5to año | Proyecto Final de Carrera

Informe de Proyecto Final de Carrera:

“Desarrollo de un framework a medida para una empresa de desarrollo de software para la ganadería de precisión”

Año	2021
Carrera	Ingeniería en Sistemas de Información
Alumno	Airaldo, César Adriel
Director	Sarli, Juan Leonardo

Tabla de contenido

1 Introducción

1.1 El solicitante

Como punto de partida para el desarrollo del presente Proyecto Final de Carrera, se emplea una solicitud formulada por una empresa que desarrolla sus propios productos de software -entre otros productos de rubros diferentes-, los cuales están presentes en el mercado desde hace casi 20 años.

La empresa se origina basada en una sociedad que tiene como integrantes a personas de la provincia de Santa Fe y de Entre Ríos, que es donde actualmente se concentra la mayor comercialización de sus productos. Su centro comercial y productivo más grande, se encuentra en las cercanías de Santa Fe Capital. La comercialización está mayoritariamente concentrada en Argentina, pero tiene presencia en los países limítrofes desde hace más de 10 años.

El contacto entre el alumno y el solicitante se origina dada una búsqueda por parte del solicitante, de un desarrollador en tecnologías del ecosistema Microsoft .Net.

El solicitante ha desarrollado sus productos sobre distintas tecnologías de Microsoft, que con el paso del tiempo han sido discontinuadas o tienen fechas de *end of life*[1] cercanas. En la actualidad tiene interés en migrar progresivamente sus aplicaciones hacia tecnologías vigentes, y continuar con las propuestas de Microsoft, haciendo hincapié en el uso de tecnologías tanto maduras como con soporte a largo plazo.

1.2 Resumen de problemas detectados

Para comprender el escenario y poder efectuar un análisis preliminar del mismo, tuvieron lugar una serie de reuniones de carácter informal, en las que el solicitante le presentó al alumno tanto las aplicaciones más importantes para la empresa, como así también su modelo de negocios.

El alumno se reunió con personal abocado a distintas actividades: programadores, personal de mesa de ayuda y soporte técnico, departamento de ventas, y usuarios internos de las aplicaciones, entre otros.

Luego de contrastar los requerimientos más importantes según la perspectiva del cliente con los elementos que imposibilitan concretarlos, y teniendo en cuenta información brindada por el área de desarrollo de la empresa, el alumno pudo caracterizar una serie de problemas a alto nivel.

Una serie de limitantes o restricciones actuales pueden ser atribuidas a las tecnologías[2] de desarrollo utilizadas por la empresa actualmente, las cuales afectan a sus productos:

[1] "End-of-life" ("EOL") es un término que se utiliza con respecto a un producto suministrado a clientes, que indica que dicho producto está al final de su vida útil (desde el punto de vista del proveedor/fabricante), y que el proveedor/fabricante detendrá el marketing, venta y/o soporte del producto.

[2] El ecosistema de tecnologías mayoritariamente utilizadas por el solicitante, está compuesto por Microsoft Visual Basic 6 para el desarrollo, Microsoft Access Database Engine como motor de base de datos, y Open DataBase Connectivity (ODBC) como conector a la base de datos.

- El código fuente se almacena en archivos binarios. Esta característica impide utilizar herramientas de versionado de código, dado que se basan en el uso de archivos de texto plano. Para resolver el problema se recurre a documentaciones externas que suelen quedar desactualizadas.
  El trabajo concurrente sobre un mismo proyecto implica la coordinación humana, cuando una persona está trabajando sobre un elemento debe anunciarle al resto del equipo dónde está efectuando cambios, una vez finalizado el trabajo, el código más reciente es copiado a una carpeta compartida por todos.
  Cada desarrollador debe periódicamente copiar el código de la carpeta compartida a su estación de trabajo para así contar con el proyecto más reciente. Suelen darse situaciones en las que, por errores en la comunicación, o por descuido, un programador realiza cambios sobre una versión de código desactualizada.
- El lenguaje de programación está orientado a eventos, con una implementación parcial y poco específica del paradigma orientado a objetos, como consecuencia coexisten códigos desarrollados en ambos paradigmas.
  Una gran desventaja del lenguaje es que no soporta el uso de interfaces, lo cual fuerza a los desarrolladores a implementar arquitecturas altamente acopladas, y limita las posibilidades de reutilización.
- No existe un mapeador objeto relacional, dado que no es posible su desarrollo sin la utilización de interfaces. Los desarrolladores se ven forzados a incorporar las consultas a la base datos dentro de su lógica de negocio.
- La arquitectura planteada por defecto mezcla el código de las vistas con la lógica de negocios y el acceso a datos. En vistas que abarcan muchas funcionalidades, las reglas de negocio en las mismas aumentan, lo cual dificulta su mantenimiento. Como resultado intrínseco, la aplicación es altamente acoplada.
- Las tecnologías de bases de datos utilizadas en el ecosistema carecen de características necesarias para proyectos a gran escala. El soporte de SQL es limitado, no soporta integridad referencial, ni el uso de transacciones.
- Los motores de bases de datos no están preparados para operar sobre grandes volúmenes de datos. Toda la base de datos se almacena en único archivo, por lo cual mientras más grande la base de datos, más lentas se tornan las operaciones ejecutadas sobre la misma. Además, estos archivos tienen vulnerabilidades bien conocidas, y hay herramientas gratuitas en el mercado para explotarlas, por lo cual ya no son útiles para almacenamiento de datos sensibles.
- Las bases de datos suelen corromperse aleatoriamente, es necesario verificarlas con frecuencia, y generan inestabilidad en las aplicaciones que las utilizan.
- Todas estas tecnologías mencionadas han alcanzado su *end of life*.

También se detectaron otras limitaciones o problemas, los cuales no son originados por el uso de ciertas tecnologías, sino por otros factores.

La empresa inició con un primer producto de software que con el tiempo adquirió la madurez, luego, para cada nuevo producto desarrollado, se tomó como punto de partida dicho producto maduro. Cuando inician un nuevo desarrollo, los programadores copian el producto maduro, eliminan las características irrelevantes, y construyen un nuevo producto sobre esta base.

Con frecuencia los programadores añaden características, mejoras o correcciones, en el núcleo de alguno de los productos de la familia de productos. Para que la actualización de un producto sea extensiva y se implemente en el resto, los desarrolladores tienen que efectuar los cambios de forma manual, uno a uno en cada uno de los proyectos.

Todos los productos ofrecen versiones en al menos dos idiomas, pero el proceso de incorporar las traducciones es complejo. Los programadores deben agregar las traducciones introduciendo cambios en el código fuente, y recompilar las aplicaciones. Por lo general, los diferentes idiomas son implementados a través de condicionales, por lo cual añadir más idiomas, implica incrementar la secuencia de condicionales.

1.3 Resumen de la solución propuesta

Luego de un análisis con mayor detalle de las soluciones de software del cliente, se detectó que, pese a que los productos del cliente comparten muchas características tales como funcionalidades, estética, tecnologías subyacentes, soporte para múltiples idiomas, estrategias de licenciamiento, estrategias de despliegue, generación de reportes, entre otras, los mismos no están nucleados de ninguna manera.

En el común de los casos, cuando se inicia un nuevo producto, se genera una réplica del producto maduro más similar en función de alguna característica, y en adelante se asume como un nuevo producto independiente en términos de desarrollo.

Un escenario que ocurre con frecuencia, es que sea necesaria una funcionalidad transversal que afecte a todos los productos, o desplegar una corrección común a los mismos, y dichas tareas se realizan de forma particular en cada producto.

Como solución a los problemas detectados, y atento a las ambiciones de crecimiento a futuro del cliente, el alumno propuso el desarrollo de un framework[3] a medida, el cual permita el desarrollo de productos homogéneos, que compartan un núcleo de características comunes asociadas a una familia de productos.

El framework sería desarrollado empleando tecnologías del ecosistema de Microsoft .Net, mediante el cual los desarrolladores de la empresa solicitante podrían desarrollar, adaptar regionalmente, y desplegar aplicaciones homogéneas.

Al momento de las primeras reuniones dadas entre el alumno y el cliente, el cliente ya contaba con un conjunto estandarizado de conceptos, tales como interfaces de usuario y código común a todos sus productos, que se replicaban de manera manual, pero que desde la perspectiva del usuario se percibían de forma homogénea.

[3] Framework (“Entorno de trabajo”, “Marco de trabajo”) es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

Tomando ventaja de dichos conceptos estandarizados, el alumno propuso normalizar una solución, y en adelante resolver tanto actuales como nuevos problemas de índole similar, bajo una misma plataforma de software.

El framework sería desarrollado empleando el modelo de desacoplamiento en capas, lo que permitirá que distintas tecnologías que estén por sobre la lógica de negocios puedan compartir las capas inferiores.

Se propuso el esquema arquitectural en capas, dado que el mismo satisface tanto el desarrollo de aplicaciones de escritorio que el solicitante quiere mejorar en el corto plazo, como el desarrollo de aplicaciones web que solicitante quiere abordar en el futuro.

Un aspecto significativo de la propuesta, es que las mejoras o correcciones que sean efectuadas al framework, impactarán en todas las aplicaciones del solicitante. Además, la generación y ejecución de pruebas se simplificará respecto al actual escenario, dado que las mismas podrán estandarizarse. Esto se debe a que las pruebas se efectuarán sobre funciones que en general sean representadas por interfaces bien conocidas que definan su comportamiento.

Teniendo en cuenta que la empresa prevé su crecimiento, la implementación de un framework con las características mencionadas también trae aparejada la facilidad de inserción de nuevos desarrolladores al equipo, dado que, para ingresar tanto a proyectos en curso como a proyectos nuevos, la curva de aprendizaje será más rápida. Los nuevos desarrolladores sólo tendrán que conocer cómo especializar componentes específicos, sin preocuparse por el funcionamiento interno de los mismos.

1.4 Fundamentación de la solución propuesta

El desarrollo del framework propuesto se fundamenta principalmente por los siguientes factores:

- Los limitantes técnicos de las tecnologías actualmente utilizadas en los productos del solicitante, impiden implementar las mejoras pretendidas sobre los mismos. Resulta necesaria la transición hacia tecnologías maduras y vigentes en el mercado actual.
- No hay en el mercado una solución que cubra las necesidades del solicitante, por lo cual es necesario desarrollarlo.
- El equipo de desarrolladores del solicitante está asociado a actividades que demandan la totalidad del tiempo de su jornada laboral, asociadas a los productos actuales. Por esta razón, se optó por delegar el desarrollo en un participante externo (el alumno).

1.5 Población

Luego de un relevamiento con el personal de la empresa solicitante, se concluyó en que la población que utilizará el nuevo producto puede ser clasificada en dos grupos generales:

- Desarrolladores de software que implementarán el framework: 4 personas.
- Usuarios del producto de software desarrollado con el framework:
    - Internos (personal propio de la empresa): 12 personas.

- Colaboradores e investigadores: aproximadamente 30 personas; en algunos casos se trata de instituciones, que vinculan a más personas.
- Licenciados: actualmente sólo en la Provincia de Santa Fe, se encuentran distribuidas aproximadamente 500 licencias, y cada una es utilizada por al menos una persona.

1.6 Objetivos generales

Desarrollar un framework que brinde funcionalidades genéricas con características específicas para la organización, que permita añadir nuevos componentes al software mediante la especialización de componentes abstractos ya definidos.

Los nuevos componentes serán homogéneos, bien definidos, y tendrán una estructura y organización predeterminada por el framework.

De esta forma, se espera que los desarrolladores centren su esfuerzo en la definición de código específico relacionado al comportamiento de los componentes, y que, como resultado, con la implementación del framework, se evite la repetición de código, disminuyan los tiempos de desarrollo, se faciliten las pruebas y se desplieguen actualizaciones sin conflictos.

1.7 Aportes que se espera realizar con este trabajo

Atento a los objetivos planteados, se esperan del presente trabajo aportes teóricos, metodológicos y prácticos, que favorezcan las actividades de las diferentes poblaciones afectadas por el desarrollo.

A continuación, se detallan los aportes esperados para las poblaciones más relevantes desde la perspectiva de este proyecto.

1.7.1 Aportes al equipo de desarrolladores que desarrollará con el framework

En la actualidad, el equipo de desarrollo de la empresa solicitante enfrenta problemas de distinta índole mencionados previamente. El framework propuesto, no solo pretende resolver los problemas conocidos en el proceso de codificación, sino que también pretende generar un cambio el proceso desarrollo que actualmente emplea el equipo. El framework será construido haciendo uso de principios y patrones de diseño, y buenas prácticas, el cual representará un cambio de mentalidad en sí mismo para los desarrolladores que lo utilicen.

1.7.2 Aportes a los usuarios finales de los sistemas que serán desarrollados con el framework

Si bien se espera que el usuario final perciba cambios tanto en los aspectos funcionales como en los no funcionales, es en los aspectos no funcionales en los que se notará una marcada diferencia. Actualmente los productos del solicitante son efectivos en la resolución de los problemas que abordan, el framework pretende brindar, además, una solución eficaz.

1.7.3 Aportes a la empresa solicitante del framework

En la actualidad, el tiempo en el proceso de desarrollo de la empresa solicitante, se ve incrementado por tareas de preparación o ajustes, como se ha mencionado previamente. El framework minimizará los tiempos de desarrollo, poniendo a disposición un escenario en el cual solo haya que abocarse a codificar la funcionalidad deseada. En consecuencia, las soluciones serán desarrolladas en menos tiempo, y supondrán una reducción en los costos de desarrollo, y un incremento en las horas disponibles de los desarrolladores para nuevas tareas.

1.7.4 Aportes al alumno

En el presente proyecto se emplea una metodología ad-hoc basada en Scrum. Tanto para la selección de la metodología, como para su adaptación, fue necesario estudiar la misma en detalle. Paralelamente, fue necesario contar con criterios que permitan justificar la elección para asegurar el éxito del proyecto, lo cual requirió la investigación y estudio de herramientas para tal fin. La elección de la metodología aportó al estudiante el conocimiento teórico de la misma, y una herramienta para seleccionar una metodología dependiendo del escenario a resolver.

Las tecnologías de desarrollo que serán utilizadas en el proyecto son conocidas por el alumno, pero en la confección del framework serán empleadas nuevas tecnologías accesorias para cumplir con solicitudes específicas del solicitante. Al finalizar el proyecto, el alumno contará con nuevos conocimientos respecto a dichas tecnologías.

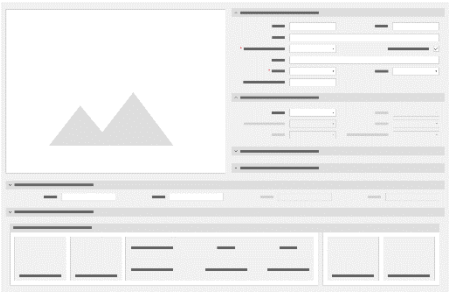
2 Conceptos previos

Previo al análisis de requerimientos, el alumno introdujo a todos los participantes e interesados en el desarrollo, una serie de conceptos con la finalidad de sentar bases comunes en el diálogo, y disminuir los posibles malos entendidos.

Todos conceptos de este capítulo fueron presentados por el alumno a los participantes de forma verbal y con la ayuda de presentaciones, con la excepción del último concepto de este capítulo, *Framework Cynefin*, el cual se introdujo con fines académicos para dar sustento teórico a la metodología de desarrollo elegida por el tesista.

2.1 Representación gráfica: "Wireframes"

Como herramienta de comunicación de ideas y puestas en común entre los involucrados en el proyecto a ser desarrollado y el alumno, y teniendo en cuenta la diversidad de áreas de conocimiento de los distintos interesados, el alumno apeló al uso de *Wireframes*.



Wireframes es un formato de diseño para representar interfaces gráficas. Este formato tiene un nivel de fidelidad media, y es comparable con el diseño de *blueprints*. Contiene mayor nivel de detalle que un sketch o esbozo, y es una instancia previa a un mock-up o prototipado.

El formato muestra el grupo principal de componentes, las proporciones entre los mismos, e información de estructura. Fue diseñado con el propósito de comunicar ideas, documentar, interactuar y reunir feedback.

La elección del formato *wireframes* no fue arbitraria. Cuando el alumno eligió dicho formato gráfico, paralelamente se discutía la metodología que sería empleada para abordar el desarrollo, siendo ampliamente probable utilizar un enfoque ágil basado en Scrum, con el que la empresa trabaja y está familiarizado.

Dadas las características de exploración de *wireframes*, que consiste en adquirir conocimiento mediante el prototipado de un concepto base, y el estudio del mismo, el alumno reconocí una finalidad y modalidad en común con Scrum. Así, se tomó provecho de dichas características en común, a las que la empresa solicitante ya estaba acostumbrada.

Un aspecto relevante de la metodología *wireframes* favorable para los rasgos de este proyecto, es la posibilidad de cambiar entre conceptos de alta y baja fidelidad. Inicialmente se puede comenzar con elementos generales de bajo nivel de detalle, confiando en que la sinergia de interacción con los *stakeholders* revelará aquellos elementos más significativos para ellos, mediante el añadido sucesivo de detalle en los mismos.

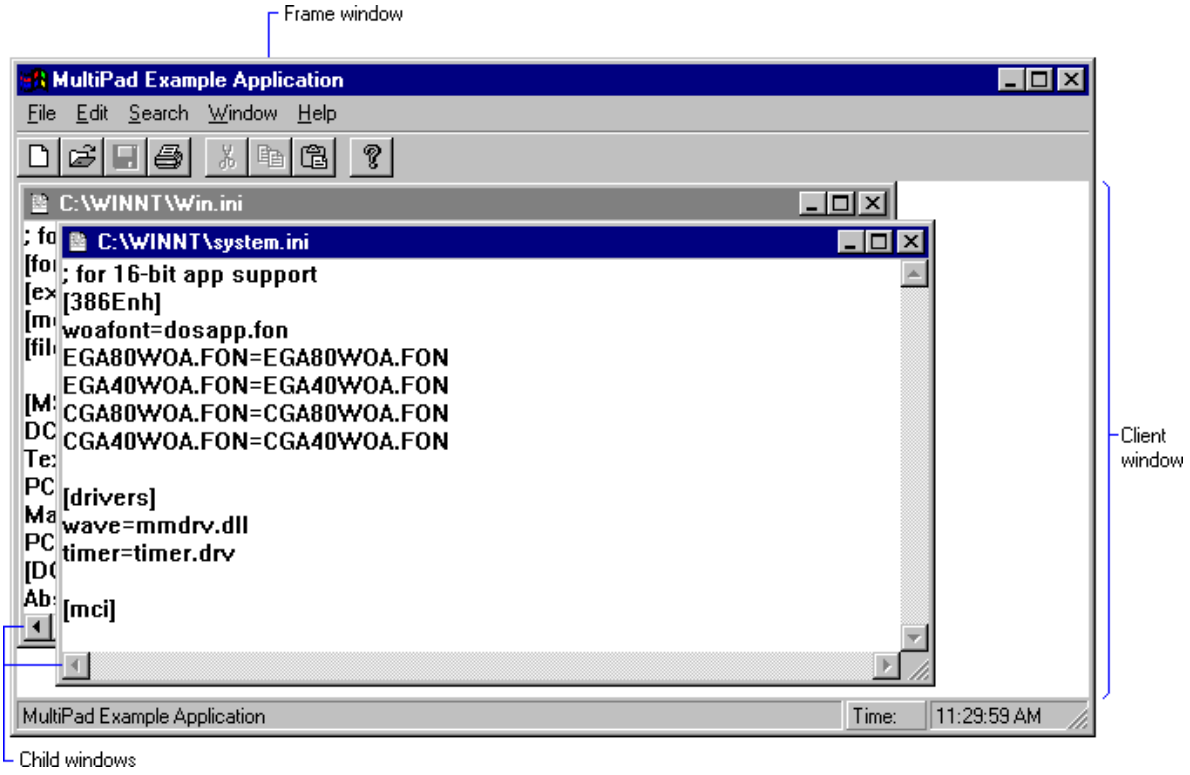
Finalmente, el formato *wireframes* tiene los siguientes propósitos intrínsecos: expresar cuál será la forma final de la aplicación, dar a entender cómo será la interacción con el usuario, explicar cómo será presentada y organizada la información, y que contenido deberá tener.

2.2 Interfaz de múltiples documentos (MDI)[4]

La *Interfaz de múltiples documentos* (MDI) es uno de los estilos de interfaces más populares en los sistemas orientados a ventanas. A continuación, se cita y traduce una introducción al mismo evitando aspectos técnicos, desde la documentación de Microsoft. Se conservan los gráficos y nombres de los elementos, tal cual la documentación oficial.

La interfaz de múltiples documentos (MDI) es una especificación que define una interfaz de usuario, que permite al usuario trabajar con más de un documento al mismo tiempo dentro de la misma interfaz.

Cada documento en una aplicación MDI se muestra en una ventana separada, pero dentro del área de la ventana principal de la aplicación. Las aplicaciones típicas de MDI incluyen aplicaciones de procesamiento de texto que permiten al usuario trabajar con múltiples documentos de texto, y aplicaciones de hojas de cálculo que permiten al usuario trabajar con múltiples gráficos y hojas de cálculo.



Una aplicación MDI tiene tres tipos de ventanas: una ventana principal *("frame window")*, un área de cliente MDI *("client window")*, y varios documentos hijos *("child window")*.

Una aplicación MDI no muestra ninguna salida en el *área del cliente* de la *ventana principal*, en su lugar muestra la *ventana del cliente MDI*.

[4] Recopilación, adaptación y traducción del alumno, basado en las siguientes referencias a la documentación oficial:
https://docs.microsoft.com/en-us/windows/win32/winmsg/about-the-multiple-document-interface

La *ventana del cliente* es hija de la *ventana principal*, sirve como espacio para crear y manipular (minimizar, maximizar, establecer el foco, etc) los *documentos hijos* o *ventanas hijas*.

Cuando el usuario abre o crea un documento, la *ventana principal* crea una *ventana hija* para el documento. El *área del cliente* es la ventana principal de todas las *ventanas hijas* de la aplicación.

2.3 Diálogos modales y no modales[5]

Los cuadros de diálogo son de dos tipos: modales y no modales. En los diálogos modales, el usuario debe cerrar el diálogo modal para que la aplicación continúe con su flujo, un ejemplo clásico de diálogo modal, es una ventana del tipo "¿Desea guardar los cambios?". Por otro lado, un cuadro de diálogo no modal permite al usuario abrir otros diálogos no modales y mantenerlos funcionando simultáneamente, cambiar el foco entre ellos, crear nuevos o eliminarlos, etc.

2.4 Framework Windows Ribbon[6]

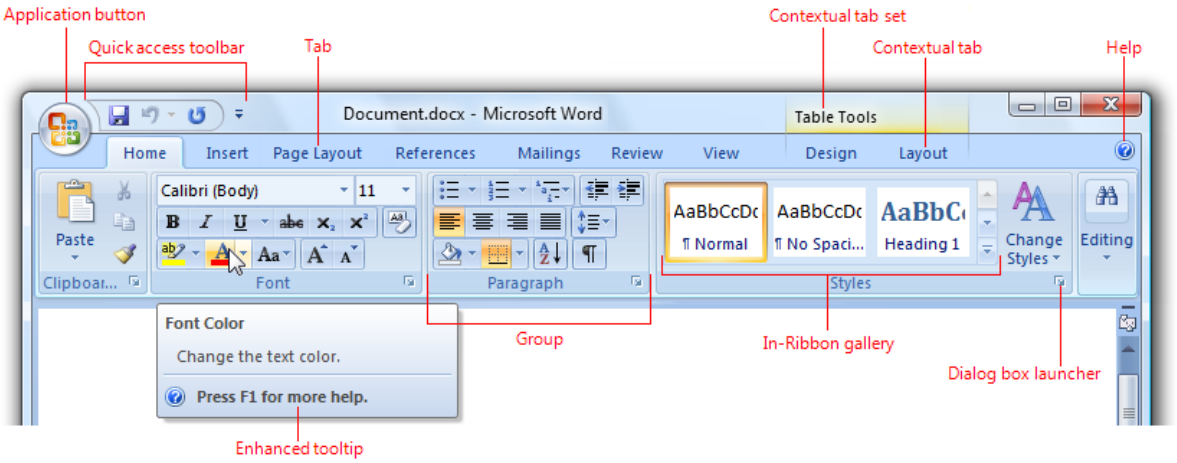
El Framework Ribbon fue diseñado por el equipo de experiencia de usuario de Microsoft. A continuación, se cita y traduce una introducción al mismo, desde la documentación de Microsoft. Se conservan los gráficos y nombres de los elementos, tal cual la documentación oficial:

Ribbon es una forma moderna de ayudar a los usuarios a encontrar, comprender y usar comandos de la aplicación de manera eficiente y directa con un número mínimo de clics, con menos necesidad de recurrir a prueba y error y sin tener que consultar la ayuda de la aplicación.

Una barra de comandos Ribbon, organiza las funciones de un programa en una serie de pestañas en la parte superior de una ventana. El uso de Ribbon aumenta la visibilidad de características y funciones, permite un aprendizaje más rápido del programa en su conjunto y hace que los usuarios se sientan más en control de su experiencia con el programa. Un menú Ribbon puede reemplazar tanto la barra de menú tradicional como las barras de herramientas.

[5] Recopilación, adaptación y traducción del alumno, basado en las siguientes referencias a la documentación oficial:
https://docs.microsoft.com/en-us/cpp/mfc/modal-and-modeless-dialog-boxes?view=vs-2019
https://docs.microsoft.com/en-us/cpp/mfc/reference/cdialog-class?view=vs-2019

[6] Recopilación, adaptación y traducción del alumno, basado en las siguientes referencias a la documentación oficial:
https://docs.microsoft.com/en-us/windows/win32/windowsribbon/windowsribbon-introduction
https://docs.microsoft.com/en-us/windows/win32/uxguide/cmd-ribbons



- Application button: el “Botón de la aplicación”, presenta un menú de comandos que implican hacer algo en o con un documento o espacio de trabajo, similar al típico menú “Archivos”.
- Quick Access Toolbar: la “Barra de herramientas de acceso rápido”, es una pequeña barra de herramientas personalizable que muestra los comandos de uso frecuente.
- Core tabs: las “Pestañas principales”, son las pestañas que siempre estarán a la vista.
- Contextual tabs: las “Pestañas contextuales”, se muestran solo cuando se selecciona un tipo de objeto en particular.
- Contextual tab set: un “Conjunto de pestañas contextuales”, es una colección de pestañas contextuales para un solo tipo de objeto. Debido a que los objetos pueden tener varios tipos de aspectos asociados, puede haber varios *conjuntos de pestañas contextuales* mostrados a la vez. Por ejemplo, una tabla tiene una imagen en su interior, tendrá dos conjuntos de pestañas, una para herramientas de la imagen, y otra para las herramientas de la tabla.
- Modal tabs: las “Pestañas modales”, son pestañas principales que se muestran con un modo temporal particular, como la vista previa de impresión.
- Galleries: las “Galerías”, son listas de comandos u opciones presentadas gráficamente. Una galería *basada en resultados* ilustra el efecto que los comandos u opciones tendrán, en lugar de sólo listar los comandos en sí. Una “In-Ribbon gallery” se muestra dentro de una cinta, a diferencia de una ventana emergente.
- Enhanced tooltips: la “Información sobre herramientas mejorada”, explica de forma concisa sus comandos asociados y proporciona los atajos de teclado. También pueden incluir gráficos y referencias a la ayuda de la aplicación. La *información sobre herramientas mejorada* reduce la necesidad de consultar ayuda extra relacionada con los comandos.
- Dialog box launchers: los “Lanzadores de ventanas de diálogo”, son botones en la parte inferior de algunos grupos, que abren ventanas de diálogo que contienen características relacionadas con el grupo.

2.5 Recursos de aplicaciones (archivos de recursos)[7]

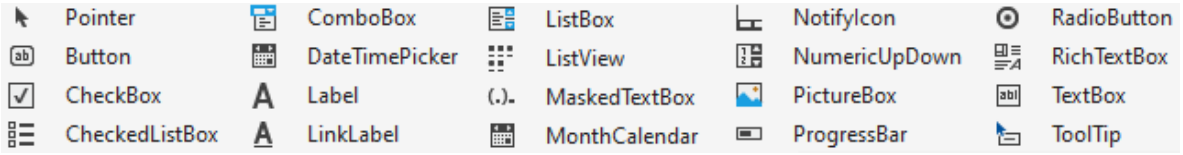
A lo largo de los años, las aplicaciones creadas con soluciones de Microsoft han estilado hacer uso de un formato de archivo para poner a disposición aquellos elementos que son parte de la aplicación, pero que pueden cambiar dependiendo de la región en la que se utilice la misma, y/o su idioma, sin tener que recompilarla.

A continuación, se cita una introducción a los "Recursos en aplicaciones", desde la documentación de Microsoft:

"Casi todas las aplicaciones de calidad de producción tienen que utilizar recursos. Un recurso es cualquier dato no ejecutable que se implemente lógicamente con una aplicación. Los recursos pueden mostrarse en una aplicación como mensajes de error o como parte de la interfaz de usuario. Los recursos pueden contener datos con varios formatos, como objetos almacenados, cadenas e imágenes. (Para poder escribir objetos almacenados en un archivo de recursos, los objetos deben ser serializables). Si los datos se almacenan en un archivo de recursos, es posible modificarlos sin volver a compilar toda la aplicación. Esto también permite almacenar los datos en una sola ubicación y elimina la necesidad de confiar en los datos codificados de forma rígida almacenados en varias ubicaciones."

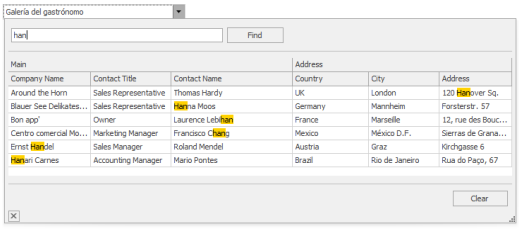
En términos simples, archivo de recursos es un archivo con extensión .resx, con una estructura similar a un archivo XML que permite almacenar recursos como texto puro o binarios en la aplicación, mediante un formato de pares clave-valor.

2.6 Controles de usuario (*"User controls"*)



Independientemente de la tecnología, los *controles de usuario* son elementos propios de las interfaces de usuario. Son componentes reutilizables, con una característica estética particular, y con una funcionalidad específica y bien definida, que pueden ser integrados dentro de unas interfaces de usuario.

Los *controles de usuario* van dese los elementos más sencillos y a los cuales estamos acostumbrados, como, por ejemplo, botones, cajas de texto, contenedores de imágenes, cajas selección (*"CheckBox"*), etc., hasta listas desplegables que muestran sus datos por columnas, y que además



[7] Recopilación, adaptación y traducción del alumno, basado en las siguientes referencias a la documentación oficial:
https://docs.microsoft.com/es-es/dotnet/framework/resources/

integran ellas la posibilidad de buscar por palabras clave, y filtrar los resultados según un criterio determinado.

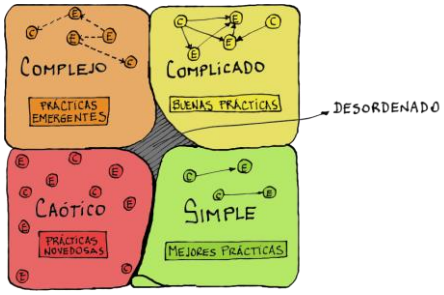
Las interfaces de usuario, están compuestas por controles de usuario. Los controles de usuario a su vez, pueden estar compuestos por otros controles de usuarios con funcionalidades específicas; un caso puntual de esta situación, es el ejemplo mencionado anteriormente, en el cual se exhibe cuánto pueden ampliarse las prestaciones otorgadas por una lista desplegable.

2.7 Framework Cynefin[8]

Al momento de enfrentar nuevos problemas, es útil poder caracterizar los mismos a través de los atributos que podemos ver de los mismos, como así también de los que no podemos ver, dado que mientras mejor los comprendamos y caractericemos, mejores decisiones podremos tomar para abordarlos.

El framework Cynefin brinda un marco para poder caracterizar un problema, e identificar las mejores técnicas para abordarlo.

En el framework se representan cinco situaciones en las que la organización puede encontrarse dada una problemática determinada. Cada una de estas situaciones predeterminadas, determinan el modo en el cual debería actuarse frente a ellas.



Cynefin compara las características de cinco situaciones o contextos con diferentes grados de complejidad: simple, complicado, complejo, caótico y desordenado (en el centro).

El concepto de complejidad es analizado mediante dos elementos, el primero se refiere a los elementos del entorno (estructura), y el segundo al comportamiento de los elementos (predictibilidad).

La *estructura* se asume *simple* cuando es fácil de comprender, y *complicada* (no complejo) cuándo no lo es. En cuanto a la *predictibilidad*, se contempla tanto el comportamiento interno del elemento como su interacción con otros, el cual va desde un comportamiento completamente predecible, a uno totalmente impredecible, pasando por situaciones "medianamente" predecibles.

La combinación de estas dos variables -la estructura, y la predictibilidad-, generan las cinco situaciones/entornos mencionados previamente, cuyas características determinarán cómo afrontarlos mejor.

A continuación, se describen brevemente las cinco situaciones mencionadas, y más adelante, se profundizará en la situación en la cual este proyecto se encuentra.

- Entornos Simples. Área de "conocimientos conocidos" *("known knowns")*. La estructura es simple, y la relación entre sus elementos es predecible. Es muy fácil identificar las causas y sus

[8] Framework Cynefin: traducción y adaptación del alumno basado en diversos autores.

efectos. La mejor manera de afrontar las situaciones simples es recurriendo al uso de buenas prácticas.

- Entornos Complicados. Área de "incógnitas conocidas" *("known unknowns")*. Pueden encontrarse múltiples respuestas correctas, y aunque la relación entre causa y efecto es clara, no todos pueden verlo. Esta situación a menudo requiere de experticia para ser resuelta.
- Entornos Complejos: Área de "incógnitas desconocidas" *("unknown unknowns")*. No se sabe si una determinada solución va a funcionar. Solo es posible examinar los resultados y adaptarse. En estos entornos se debe confiar en las soluciones emergentes; las soluciones encontradas, aplicadas anteriormente, rara vez son replicables con los mismos resultados a otros problemas similares.
- Entornos Caóticos: Área de "incognoscibles". Las relaciones entre causa y efecto son imposibles de determinar porque cambian constantemente y no existen patrones manejables. Lo más importante es solucionar el problema inmediatamente y salir de esta área, sin importar la forma técnica.
- Entornos Desordenados: No se puede asignar un área concreta, dado que no se sabe en qué situación está el escenario, y por lo tanto no se puede decidir cómo actuar. Deben enfocarse los esfuerzos en determinar un área en concreto.

3 Metodologías y criterios elegidos por el tesista

En el presente capítulo se resumen los lineamientos metodológicos con los que se guío el proceso de desarrollo del proyecto.

El modelo de desarrollo empleado, así como la gestión de riesgos, recibieron capítulos especialmente dedicados, en los *ANEXO A – Modelo de desarrollo de software seleccionado por el tesista* y *ANEXO C – Gestión de riesgos*.

Luego, los resultados de la ejecución del proyecto en lo relacionado a la gestión y ejecución de la metodología de desarrollo (planificación, resultados de ejecución de la misma, gestión de riesgos, etc.), se abordaron en detalle en el *ANEXO D – Ejecución del proyecto*.

3.1 Modelo de desarrollo: Scrum

Para guiar las actividades del proceso de desarrollo de software asociado al presente Proyecto Final de Carrera, se especificó y utilizó una metodología ad-hoc adecuada a los recursos humanos disponibles, basada en el marco de trabajo para desarrollo ágil, Scrum.

La especificación en detalle de los roles, eventos y artefactos de la metodología resultante se dispuso en el anexo *Modelo de desarrollo de software seleccionado por el tesista*.

En la presente sección se brinda un resumen de los aspectos más relevantes de la metodología, para permitir que un lector con conocimiento en la metodología de base pueda proceder con la metodología.

3.1.1 Especificación de las actividades: User Stories

Las actividades realizadas en el presente proyecto, se detallaron utilizando el formato de *User stories*. El orden de ejecución de las *User stories* se programó tomando como primer criterio de ordenamiento la *prioridad en desarrollo*, y como segundo criterio la *prioridad en el negocio*.

La *prioridad en el negocio* refleja aquellas funcionalidades que el solicitante desea conocer primero. La *prioridad en el desarrollo* refleja aquellas funcionalidades necesarias para el avance del desarrollo, tales como ciertas funcionalidades esenciales que son utilizadas por otras funcionalidades (es decir, que hay un orden técnico que no puede evitarse).

3.1.2 Estimación del esfuerzo: Story Points

Para la estimación del esfuerzo se empleó la unidad relativa *Story Point*. El peso relativo asignado a cada *Story Point* se estimó en aproximadamente 6,85 [hs], pudiendo ser cuantificado mediante el intervalo 6,5 [hs] ± 30 [min]. Luego, se tabularon las *User stories* relevadas, con su correspondiente estimación en *Story Point*.

3.1.3 Duración de los eventos de Scrum

La duración de los eventos fue asignada por el *Scrum Master*. Se empleó *la semana* como unidad constante, definidas con 5 días laborales, donde cada día laboral es de 8 horas.

Nombre del evento	Duración
Sprint	2 [semanas]
Sprint Planning	4 [horas]
Daily Scrum	15 [minutos]
Preparación de presentación para el Sprint Review	1 [horas]
Sprint Review	2 [horas]
Sprint Retrospective	2 [horas]

3.2 Modelo de desarrollo: Justificación de la elección

El cliente utiliza Scrum tanto para dar soporte a los procesos de desarrollo de software, como para dar soporte a los procesos organizacionales.

En las reuniones de toma de contacto, el líder de desarrollo de la empresa solicitante (al cual posteriormente se le asignó el rol de *Product Owner*) propuso de forma no mandatoria que, como metodología de desarrollo para su producto se utilice una metodología de desarrollo basada en Scrum, dada la familiaridad que la organización tiene con dicha metodología.

Considerando la propuesta, el alumno considero necesario evaluar si las características de Scrum eran adecuadas para las características del proyecto. Para tal fin, se utilizó el framework Cynefin, una herramienta que permite contrastar la complejidad de un escenario, contra grupos de características bien definidas, y finalmente concluir en una familia de metodologías apropiadas para abordarlo.

3.2.1 Entorno del presente proyecto según Cynefin

Dado que para la problemática planteada en el presente proyecto no se han encontrado soluciones estándar, y que no se tiene certeza de que una determinada solución vaya a funcionar, acorde a *Cynefin* este escenario es *complejo*.

En los escenarios complejos las soluciones estándares o las recomendaciones de expertos no garantizan la solución. La única manera de gestionarlos es realizar ciclos de prueba y error en los que se ensayen posibles respuestas, se evalúa su impacto y se actúa según esa información.

Ese es el ámbito de los enfoques ágiles, caracterizados por el desarrollo de una solución de manera iterativa e incremental, y basados en un enfoque empírico.

3.2.2 Conclusión

Los resultados del framework Cynefin y el criterio del líder de desarrollo de la empresa solicitante (Scrum Master dentro de la empresa) fueron concluyentes en que una adecuación de Scrum sería viable para conducir el proyecto.

3.3 Gestión de riesgo

Dada la amplitud de aspectos que son merecedores de ser desarrollados en el campo de la *Gestión de riesgos*, se dedicó un capítulo para su desarrollo.

En el anexo *Gestión de riesgos* se aborda el estudio de los riesgos de forma general, y también se aborda el riesgo desde la perspectiva de Scrum, dado que su filosofía le da cobertura a los mismos. Luego, se discuten y seleccionan las estrategias para la identificación y evaluación de riesgos, y se establecen disparadores y planes de contingencia.

Con la finalidad de brindar una noción rápida de las posturas asumidas frente a *la Gestión de riesgo* en el proyecto, en la presente sección se brinda un breve resumen de las mismas.

En busca de información respecto al rol de Scrum frente a la gestión de riesgo, el alumno encontró que en las comunidades[9] de usuarios de Scrum hay dos corrientes opuestas. Una de ellas sostiene que Scrum en si misma es una forma de controlar el riesgo, y la otra sostiene que Scrum no puede ayudar con los riesgos externos al proyecto. Por esta razón, el alumno adoptó una postura atenta a ambas perspectivas.

3.3.1 Enfoque ágil

Ken Schwaber, co-creador de Scrum dice que, dada la filosofía de Scrum, “Scrum es en sí misma una forma de control el riesgo”.

Dado que la aplicación de Scrum es ideal en entornos que tienen como factor inherente el cambio y la evolución rápida y continua, propone mantener una postura en la cual “Abrazar el cambio”, y mantener un enfoque innovador frente a los riesgos para generar oportunidades desde los mismos.

3.3.2 Enfoque estructurado

Como enfoque estructurado para la identificación de riesgos, se utilizó el *Reporte de Identificación de Riesgos basada en Taxonomía* (“Taxonomy-Based Risk Identification report”), del *Software Engineering Institute*, que cubre 13 áreas de riesgo principales, con aproximadamente 200 preguntas [SEI-93].

La taxonomía resultante que contiene los riesgos identificados puede encontrarse en la sección *Identificación de riesgos*.

3.3.3 Evaluación de riesgos

A continuación, se presenta un resumen de la estrategia implementada para evaluar los riesgos en este proyecto. La *Estrategia para la evaluación de los riesgos identificados*, la *Evaluación*

[9] Comunidades de Scrum que el alumno visitó para recolectar comentarios y experiencias de su uso: www.scrum.org, www.scrummastered.com, y www.reddit.com/r/scrum/.

de los riegos identificados, y los *Disparadores y planes de contingencia fueron abordados en secciones específicas*.

La evaluación de riesgos se realizó con respecto a *probabilidad*, *proximidad* e *impacto*. La *probabilidad* de riesgos se refiere a la probabilidad de que ocurran los riesgos, mientras que la *proximidad* se refiere a cuándo podría ocurrir el riesgo. El *impacto* se refiere al efecto probable de los riesgos en el proyecto o la organización.

Para estimar la probabilidad de un riesgo, en el presente proyecto se empleó la ecuación con la cual Boehm define el grado de exposición al riesgo [Bohem-89]. Ésta mide el impacto de un riesgo en términos del valor esperado de la pérdida, según:

$$RE\ (Exposición\ al\ riesgo) = Probabilidad\ de\ ocurrencia * Pérdida\ esperada$$

3.4 Plan de monitoreo periódico del proyecto

Los *eventos* y *artefactos* de Scrum están diseñados específicamente para permitir la transparencia crítica y la inspección, y que cada evento es una oportunidad formal de inspección y adaptación.

La *transparencia*, la *inspección* y la *adaptación* son los pilares que subyacen a la teoría del control de procesos empíricos que definen a Scrum. “Inspeccionar y adaptar” (*“Inspect and adapt”*) es uno de los eslóganes -también mantra- con los que frecuentemente se relaciona a la metodología.

La filosofía de Scrum propone, que cada adaptación efectuada se convierta en lo siguiente a inspeccionar. Cada nueva elección que se tome, será un nuevo experimento. Con ello, se aprende a hacer pequeños ajustes y a estudiar los resultados.

De este modo, el plan de monitoreo periódico de este proyecto, fue cubierto por las incumbencias de la metodología escogida para llevarlo a cabo.

4 Análisis y Desarrollo

El propósito de este capítulo, es narrar las instancias sucesivas que se atravesaron en el proyecto, desde las primeras reuniones que el alumno tuvo con el cliente en las cuales se elicitaron requerimientos y generaron modelos de alto nivel, hasta las etapas tempranas del desarrollo, en las cuales se sentaron las bases para el mismo a través de la arquitectura, seguridad, y testing.

Los apartados del presente capítulo están dispuestos en el orden cronológico en el que se sucedieron, y pretenden servir de base para el capítulo siguiente, en el cual se presentarán los componentes específicos que se han desarrollado con sus resultados en tiempo de ejecución.

Paralelamente al estudio y adecuación del *modelo de desarrollo elegido por el tesista* (dispuesto en el capítulo anterior), el alumno y el cliente acordaron en una serie de reuniones las características que debía cumplir el desarrollo de software resultante, a través de un *modelo conceptual de la aplicación*.

A la par de la confección del *modelo conceptual de la aplicación* y ya contando con una idea aproximada de cómo sería el desarrollo finalizado, el alumno presentó al personal técnico de la empresa una serie de tecnologías candidatas a ser seleccionadas para iniciar el desarrollo. Como resultado de estas reuniones, se generó una lista preliminar de *tecnologías y plataformas a ser utilizadas*, que luego se refinó y es presentada en este capítulo.

Teniendo presente las características del producto que debía desarrollarse, y las tecnologías con las que debía llevarse a cabo, se redactaron las *User Stories* (dispuestas en el *ANEXO B – Actividades a ser realizadas*).

Luego, contando con los elementos previamente mencionados, se estableció el *orden de ejecución de las User Stories*, se *estimó el esfuerzo* que demandaría su realización, se definió un *Product Backlog inicial*, y contemplando *los recursos humanos disponibles* para el proyecto, se estimo la *duración* del mismo, con lo cual se desarrolló un *cronograma de avance*.

Cómo resultado de la realización de las primeras User Stories del proyecto, se desarrolló y documento la *arquitectura de base*, la cual se dispone en este capítulo a modo de introducción de la lectura técnica.

Desde etapas tempranas en el desarrollo, se establecieron lineamientos respecto a la *seguridad* y *testing* en el proyecto, los cuales se resumen al final de este capítulo.

Toda la documentación generada en el proceso de realización de las User Stories, se dispuso en un capítulo individual, en el *ANEXO E – Documentación generada en la realización de las User Stories*.

4.1 Modelo conceptual de la aplicación

Como instancia previa a la generación de *User Stories*, y como herramienta de soporte para la ejecución de las mismas, se generó un modelo conceptual de la aplicación, que explica de forma simplificada cómo el software será percibido por el usuario final.

Este modelo, es el resultado de la iteración de debates e intercambio de ideas entre los *stakeholders* y el alumno. El modelo inicial para el debate fue desarrollado por el alumno, posterior al análisis de los productos del solicitante. Para generar el modelo inicial, se generalizaron los elementos comunes en sus productos, y se añadieron algunas propuestas de diseño.

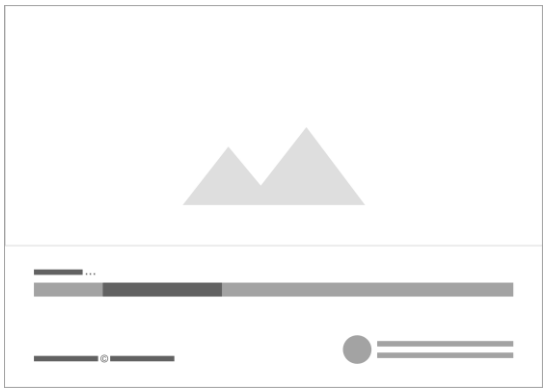
Este modelo tuvo como objetivo servir como primera herramienta de diálogo entre el desarrollador y los *stakeholders*, en términos que cualquier participante sin conocimientos técnicos pudiera comprender, y que además permitiese la incorporación de aportes de forma sencilla.

El modelo tuvo como premisas las siguientes características:

- La aplicación será desarrollada utilizando el formato "interfaz de múltiples documentos" (MDI), representando los distintos documentos en pestañas (TDI).
- Los documentos podrán lanzar diálogos modales, inhabilitando el uso de los documentos subyacentes en tanto el diálogo esté abierto.
- Para el acceso de comandos se utilizará el framework Windows Ribbon.

4.1.1 Splash screen

Al iniciar la aplicación, la primera interfaz que será mostrada será el *"Splash screen*[10]*"*. Esta interfaz no será empleada sólo con fines estéticos.



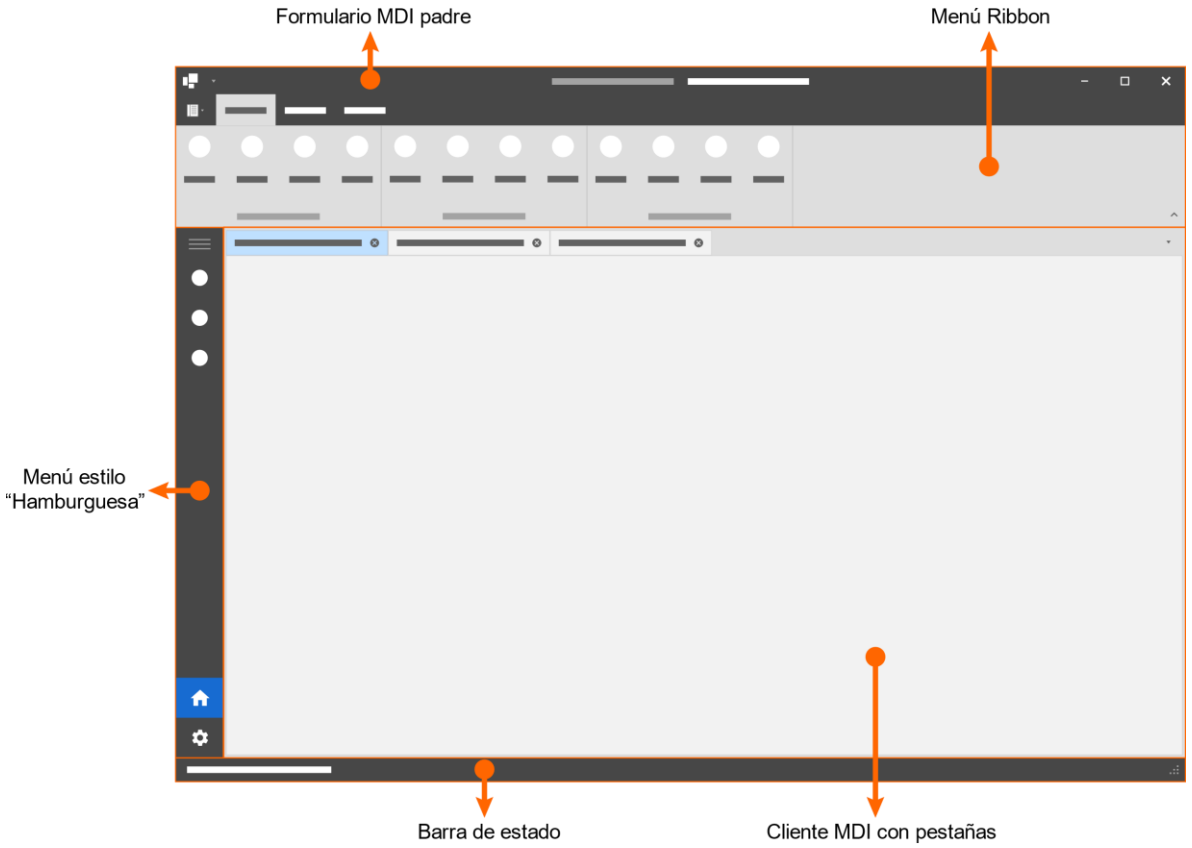
La misma permanecerá abierta mientras sean realizadas las tareas de inicio fundamentales, y las verificaciones esenciales (como, por ejemplo, la conexión a la base de datos).

Si se encuentra algún error en la etapa de inicio de la aplicación, no será mostrada la interfaz principal de la aplicación luego del *Splash screen*.

El *Splash screen* mostrará información básica de la empresa, la versión de la aplicación, una imagen y una barra de carga representando el proceso transcurrido.

[10] *"Splash screen"* está asociado comúnmente al concepto de "pantalla de bienvenida". Es un control de usuario que generalmente contiene una imagen, un logotipo, la versión actual del software, entre otros. Esta pantalla es mostrada, por lo general, mientras se inicia una aplicación.

4.1.2 Formulario principal



El acceso a las operaciones principales de la aplicación será concentrado en un formulario padre MDI. Dicho formulario padre estará asociado con una instancia activa de la aplicación, de forma tal que cerrar el formulario padre, implica detener la ejecución de la aplicación.

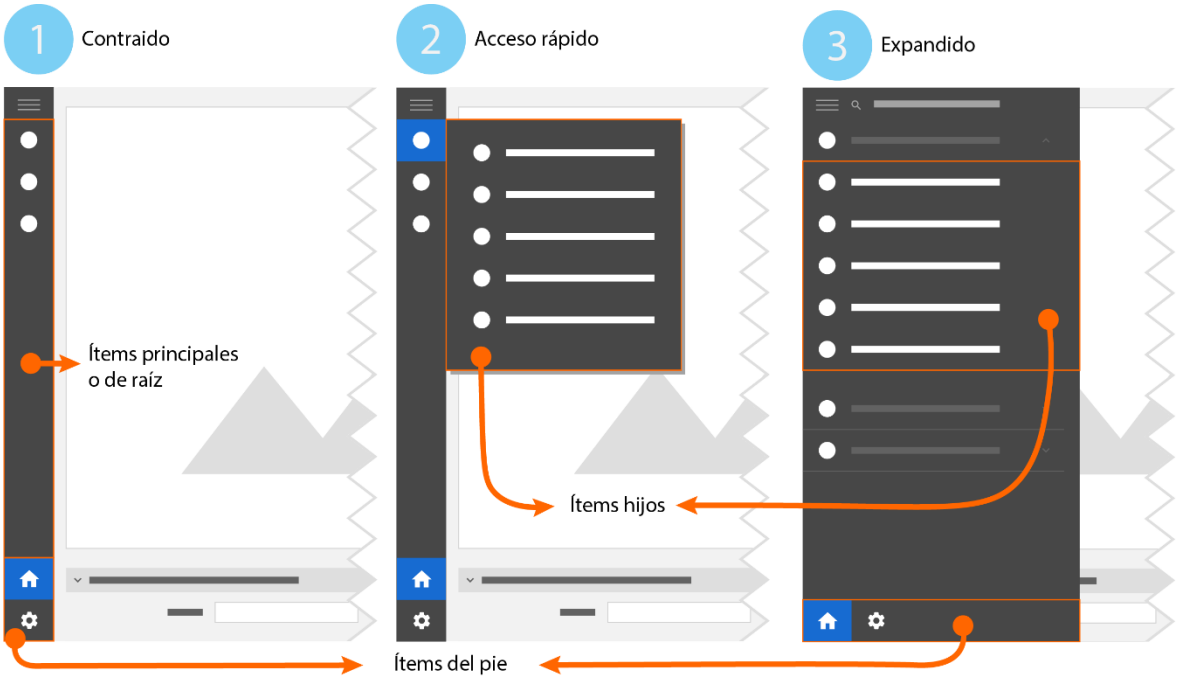
En el área superior del formulario MDI padre se ubica un menú Ribbon. El menú podrá ser contraído, quedando solo visibles las cabeceras de las pestañas de acceso. Se estipula que existan comandos del menú que estén vinculados a operaciones del formulario padre, que seguirán estando presentes en el menú cuando sean mostrados los comandos de un formulario hijo.

En el área de la izquierda se ubicará un menú vertical, estilo “hamburguesa”, que podrá ser expandido o contraído. En su área inferior se ubicarán al menos dos elementos para agrupar características: un grupo para las opciones principales de la aplicación, y otro para las opciones de configuración. Al ser expandido el menú se sobrepondrá al contenido que esté por debajo de él. Este será el principal punto de acceso a los formularios hijo, que brinden acceso a las funcionalidades del sistema.

En el centro de la aplicación se ubicará el área del cliente MDI. El cliente MDI mostrará por defecto a todas sus ventanas hijas como pestañas, pero deberá ser posible desacoplarlas, y manejarlas como ventanas individuales.

En el área inferior se ubicará una barra de estado para propósitos generales. Podrá mostrar información relevante al formulario padre, y/o a al formulario hijo que tenga el foco.

4.1.3 Menú principal



El menú de navegación principal, será un menú lateral de tipo "acordeón[11]", con estilo "hamburguesa[12]", y tendrá tres posibles estados de visualización: contraído, de acceso rápido, y expandido.

Cuando el menú esté *contraído*, si se realiza clic sobre cualquiera de los *íconos principales o de raíz*, si se trata de una funcionalidad se accederá directamente a ella, pero si se trata de un grupo de funcionalidades, entonces el menú pasará al segundo estado, de *acceso rápido*.

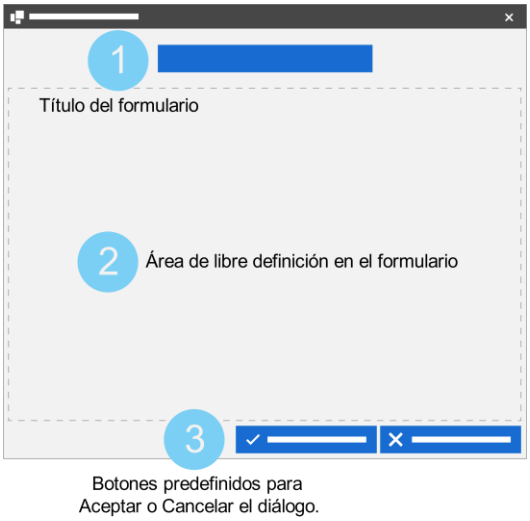
El menú en estado *acceso rápido*, permite acceder rápidamente a los *ítems hijos* pertenecientes a un grupo, sin la necesidad de expandir el menú completamente.

El menú *expandido*, permite navegar los distintos *grupos principales* o familiar de funcionalidades con sus respectivos *ítems hijos,* y, además, añade un buscador de palabras clave en área superior, para facilitar la búsqueda y acceso a funcionalidades.

En el pie del menú, habrá al menos dos elementos, un ítem que agrupará las funcionalidades principales (señalizado con una casa), y un ítem que dará acceso al *Formulario de configuración* (señalizado con un engranaje).

[11] Se conoce como menú "acordeón", al menú de navegación lateral cuyos elementos se pueden organizar en grupos.
[12] Se conoce como menú "hamburguesa", a al menú "acordeón", que proporciona el ícono hamburguesa (tres líneas verticales apiladas, ☰) que permite expandir y contraer el menú, y que tiene y tres modos de visualización: en línea ("*in line*"), superpuesto ("*overlay*") y minimalista ("*minimal*").

4.1.4 Formulario modal Aceptar-Cancelar (base)



El *Formulario modal Aceptar-Cancelar*, servirá como base para confeccionar otros formularios modales Aceptar-Cancelar, para funcionalidades específicas.

Este formulario no tendrá utilidad por sí solo, será necesario extender o heredar el mismo, y añadir los elementos gráficos correspondientes con su lógica asociada.

Al especializar el componente, será posible adecuar algunos de sus elementos gráficos mediante distintas estrategias, como se explica a continuación.

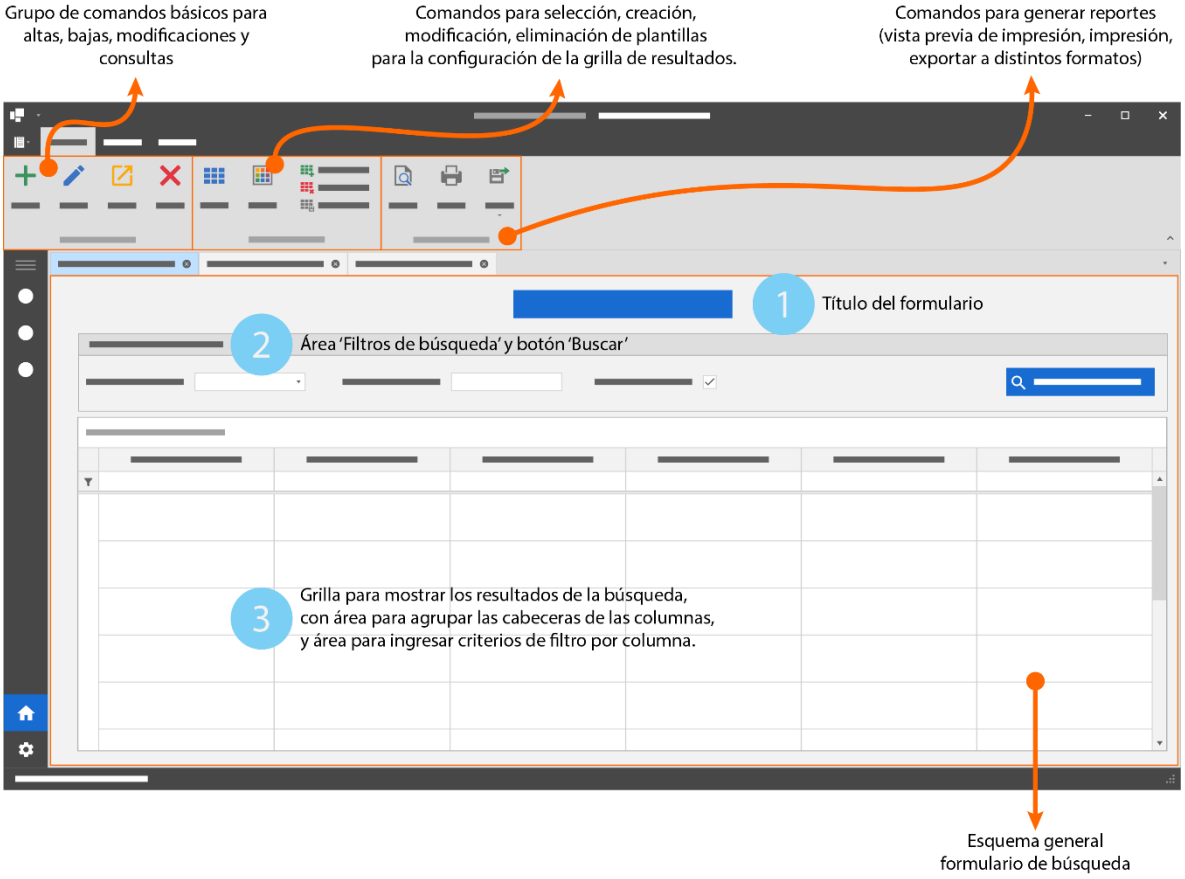
El título de formulario tendrá un estilo visual predefinido, al especializar el componente sólo podrá definir el contenido del mismo.

En el área de libre definición del formulario podrán añadirse los *controles de usuario* correspondientes a su lógica asociada. Este será la única área de este formulario donde el usuario pueda genera cambios estéticos libremente.

Los botones “Aceptar” y “Cancelar”, tendrán una lógica mínima predefinida, que el desarrollador que extienda el componente no podrá remover, pero a la cual podrá añadir lógica afín al uso específico del componente.

Este componente será ampliamente utilizado, y será empleado para definir otros componentes con funcionalidades específicas. Si este componente de base se altera, los cambios serán reflejados en todos los elementos que hereden del mismo.

4.1.5 Formulario de búsqueda (base)



El *Formulario de búsqueda*, servirá como base para confeccionar otros formularios de búsqueda para entidades específicas.

Este formulario no tendrá utilidad por sí solo, será necesario extender o heredar el mismo, y añadir los *controles de usuario* correspondientes con su lógica asociada.

Al especializar el componente, será posible adecuar algunos de sus elementos gráficos mediante distintas estrategias, como se explica a continuación.

La plantilla del formulario de búsqueda está compuesta por tres secciones, el título del formulario, un área destinada a los filtros de búsqueda y el botón buscar, y la grilla en la cual serán mostrados los resultados.

En el menú Ribbon, se dispondrán al menos tres grupos de comandos. Un grupo para operaciones de ABMC (altas, bajas, modificaciones y consultas), un grupo para la configuración de la grilla, y un grupo para operaciones de salidas y exportación de resultados.

Cada comando del grupo de comandos de comandos de ABMC podrá ser configurado para redireccionar a la interfaz correspondiente. Las modificaciones, consultas y eliminaciones sólo podrán ser realizadas a través de estos comandos, previamente seleccionando un registro de la grilla. Los formularios de altas podrán accederse desde los formularios de búsqueda, o desde el menú principal.

La grilla de cada formulario de búsqueda podrá ser configurada en término de sus aspectos estéticos y de las columnas que la componen. Los aspectos estéticos respectan al color, estilo y tamaño de texto, y colores de fondo para celdas y cabeceras de columna. Cada formulario de búsqueda tendrá asociado un determinado grupo de columnas, el usuario podrá seleccionar cuáles de esas columnas puede ser visibles, y cuáles no. La configuración de un determinado grupo de columnas con sus respectivos aspectos estéticos puede ser almacenado en una plantilla. El usuario puede utilizar distintas plantillas para un mismo formulario de búsqueda, y alternarlas acorde a su necesidad.

Los formularios de búsqueda podrán generar salidas de los resultados visibles en la grilla, a distintos medios y/o formatos. Será posible consultar una vista previa de impresión, imprimir, y exportar a distintos formatos de archivo.

Los comandos de exportación a los distintos formatos de archivo serán expuestos en un menú desplegable, dentro del mismo menú Ribbon. Al seleccionar un formato de exportación, se abrirá un diálogo para seleccionar el nombre y la ubicación del archivo resultante.

4.1.6 Formulario de búsqueda: Nueva plantilla

Desde el menú Ribbon del *Formulario de búsqueda*, en el grupo de comandos para la gestión de plantillas, se podrá acceder a un formulario para crear plantillas.

El formulario de creación de plantillas, es un formulario especializado del *Formulario modal Aceptar-Cancelar*.

Para la creación de una plantilla, solamente será requerido un nombre para la misma, y un grupo de columnas para mostrar.

En la instancia de creación de la plantilla no será posible efectuar ningún cambio relacionado a aspectos estéticos de las columnas, ni su orden.

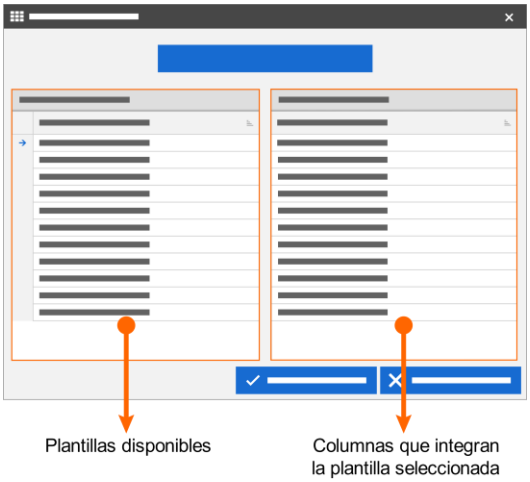
La configuración de aspectos estéticos y orden podrá ser realizada por otro formulario específico.

4.1.7 Formulario de búsqueda: Seleccionar plantilla

Desde el menú Ribbon del *Formulario de búsqueda*, en el grupo de comandos para la gestión de plantillas, se podrá acceder a un formulario para seleccionar la plantilla de configuración de la grilla.

El formulario de selección de plantilla, es un formulario especializado del *Formulario modal Aceptar-Cancelar*.

El formulario presentará a la derecha, las plantillas disponibles para el formulario de búsqueda. Al seleccionar una plantilla, a la izquierda serán listadas las columnas que la integran.



En la instancia de selección de la plantilla no será posible efectuar ningún cambio relacionado a aspectos estéticos de las columnas, ni su orden. Desde esta interfaz, tampoco será posible crear nuevas plantillas.

4.1.8 Formulario de búsqueda: Configurar plantilla

Desde el menú Ribbon del *Formulario de búsqueda*, en el grupo de comandos para la gestión de plantillas, se podrá acceder a un formulario para configurar la estética de las columnas de la grilla, y también seleccionar cuáles de ellas son visibles y cuáles no.

El formulario de configuración de plantilla, es un formulario especializado del *Formulario modal Aceptar-Cancelar*.

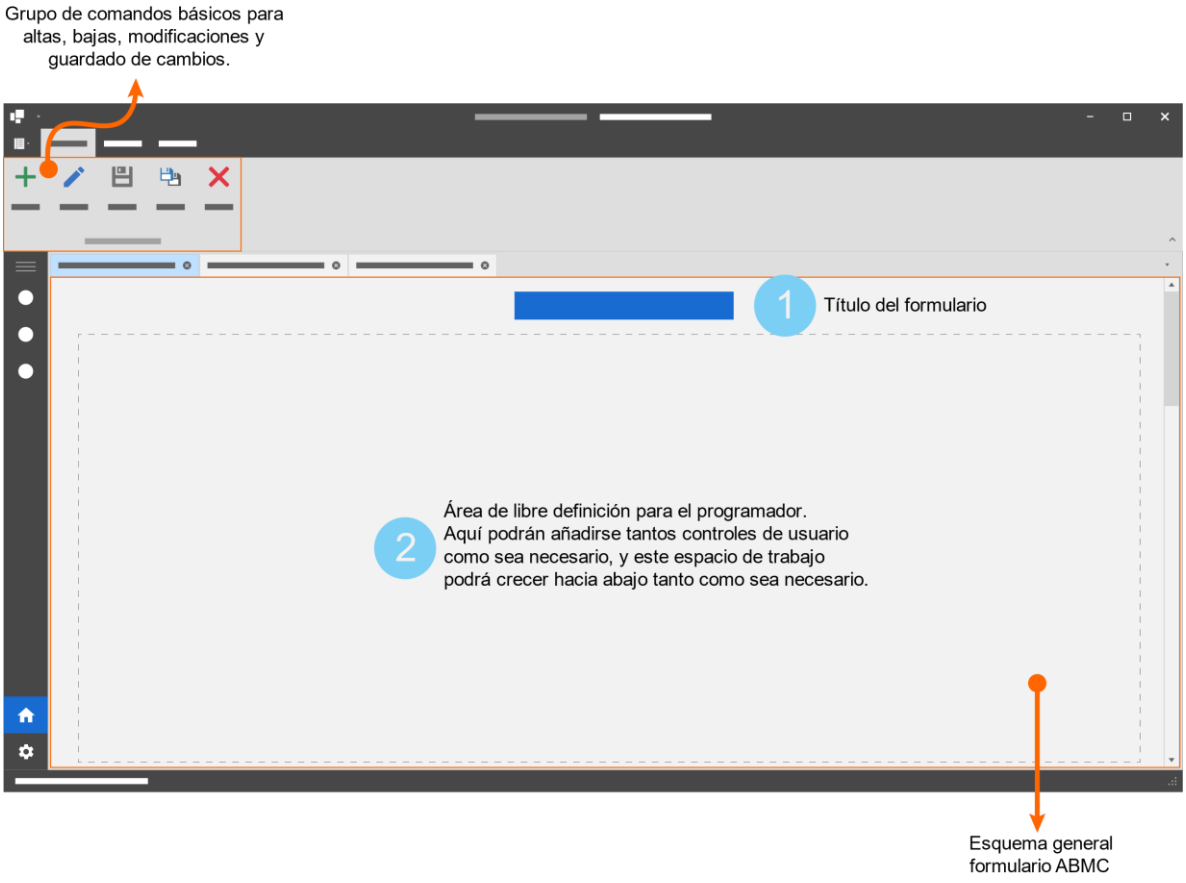
El formulario presentará en la parte superior, un filtro de palabras para las columnas. En la parte inferior, una grilla compuesta por cuatro secciones: el listado de columnas, el estado de visibilidad de las columnas, la configuración estética de las cabeceras de las columnas, y la configuración estética de las celdas de las columnas.

Las columnas serán listadas, primero descendiente por su estado de visibilidad, luego ascendientemente por su nombre, y finalmente por un índice que indicará su orden (columna invisible). Esto generará gráficamente dos grupos, primero el grupo de las columnas visibles ordenadas por su índice, y luego el grupo de las columnas no visibles ordenadas alfabéticamente (dado que tendrán índice 0).

Las filas de la grilla, correspondientes a cada columna de la plantilla, responderán a clic con el botón derecho sobre ellas. Las filas correspondientes a columnas visibles, podrán ordenarse (subir y bajar), descartar los últimos cambios realizados a las mismas, y reestablecer la configuración por defecto. Las filas correspondientes a columnas no visibles, podrán sólo descartar los últimos cambios realizados, o reestablecer la configuración por defecto.

En la instancia de selección de la plantilla no será posible crear una nueva plantilla, o eliminar la plantilla que está siendo configurada.

4.1.9 Formulario ABMC (base)



El formulario base ABMC (altas, bajas, modificaciones, y consultas), servirá como base para confeccionar otros formularios ABMC para entidades específicas.

Este formulario no tendrá utilidad por sí solo, será necesario extender o heredar el mismo, y añadir los elementos gráficos correspondientes con su lógica asociada.

Al especializar el componente, será posible adecuar algunos de sus elementos gráficos mediante distintas estrategias, como se explica a continuación.

El título de formulario tendrá un estilo visual predefinido, al especializar el componente sólo podrá definir el contenido del mismo.

El formulario base ABMC (altas, bajas, modificaciones, y consultas) está compuesto por dos secciones, el título del formulario, y un área de libre definición destinada a los controles pertinentes a la razón de ser del formulario.

En el menú Ribbon, se dispondrán al menos un grupo de comandos, en el cual se brindará acceso a las operaciones de ABMC (altas, bajas, modificaciones y consultas).

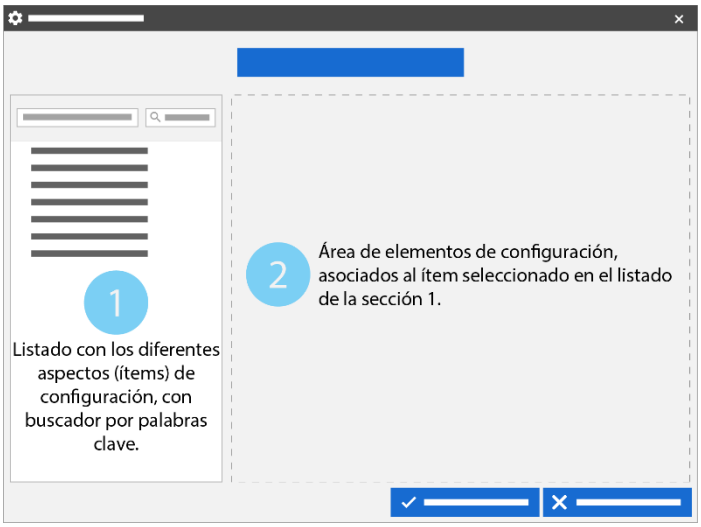
Este formulario internamente manejará tres estados bien definidos: consultar, editar y nuevo. En función del estado en curso, serán visibles diferentes comandos en el menú Ribbon:

Estado “Consultar”: cuando se seleccione un registro desde el *Formulario de Búsqueda*, y se elija consultar el mismo, se abrirá *Formulario ABMC* en modalidad de consulta, poniendo a disposición comandos para crear una nueva entidad del mismo tipo, para editar la entidad que se está consultando, o eliminarla.

Estado “Editar”: cuando se seleccione un registro desde el *Formulario de Búsqueda*, y se elija editar el mismo, se abrirá *Formulario ABMC* en modalidad de edición, poniendo a disposición comandos para crear una nueva entidad del mismo tipo, para guardar los cambios en la entidad que se está editando, o eliminarla.

Estado “Nuevo”: cuando se seleccione crear un registro desde el *Formulario de Búsqueda*, o desde el menú lateral, se abrirá un *Formulario ABMC* en modalidad de creación, poniendo a disposición sólo un comando parar crear una nueva entidad. Inmediatamente luego de la creación de la nueva entidad, el formulario pasará al estado “Editar”.

4.1.10 Formulario de configuración



Desde el *menú principal*, al pie, desde la sección de grupos principales, se podrá acceder al *Formulario de configuración*.

El *Formulario de configuración*, es un formulario especializado del *Formulario modal Aceptar-Cancelar*.

El formulario presentará a la derecha una lista con los diferentes aspectos que puedan ser configurados, y a la izquierda, un área de uso general, en la cual podrán establecerse aquellos controles de usuario que respecten al aspecto de configuración seleccionado en el menú de la izquierda.

Nombraremos como “ítem de configuración”, al conjunto comprendido por un nombre para el aspecto de configuración dado (cómo por ejemplo podría ser, “Idioma”, “Skins”, etc.), y al panel que brindará la posibilidad de configurarlo, el cual será mostrado a la derecha.

Cada *ítem de configuración* será cargado de forma dinámica. Lo cual significa que el *panel de configuración* asociado al *aspecto de configuración* seleccionado a la izquierda, no será cargado hasta que no sea utilizado.

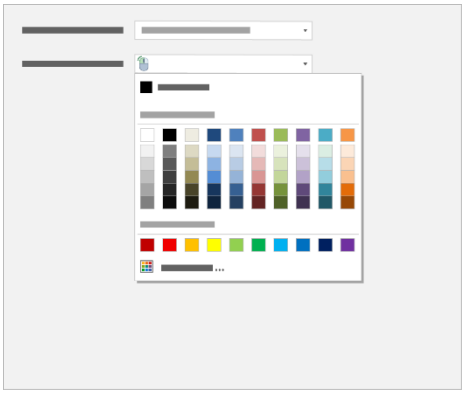
En caso de oprimir el botón *Aceptar* (el cual es una característica heredada del *Formulario modal Aceptar-Cancelar*), serán efectuadas todas las modificaciones de cada uno de los aspectos de configuración que hayan sido utilizados. Caso contrario, de oprimir el botón *Cancelar*, serán descartados todos los cambios.

El listado de características configurables de la derecha, admitirá que las características se agrupen, utilizando el esquema de árbol, y permitiendo que cada grupo se expanda o contraiga.

Un buscador en el listado de la izquierda, permitirá filtrar los aspectos de configuración por palabras clave.

El *Formulario de configuración* podrá brindar funcionalidad específica para cada aspecto de configuración, sólo a través de los paneles de configuración, pero dichos paneles no tendrán dependencia con el formulario, ni el formulario con los paneles.

4.1.11 Panel de configuración "Apariencia"

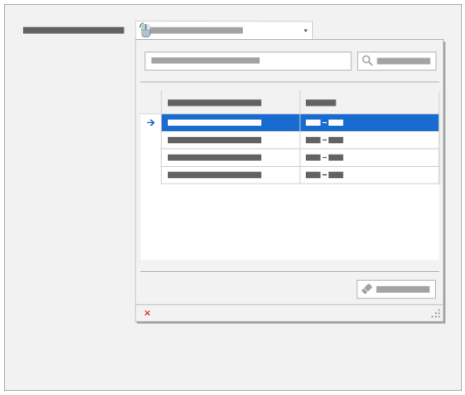


El Panel de configuración "Apariencia", será un ítem de configuración del *Formulario de configuración*.

Este panel, idealmente permitirá elegir distintos tipos de estilos gráficos, y a su vez, distintas paletas de colores para los mismos.

La flexibilidad de configuración de los estilos gráficos, y, por ende, las herramientas que puedan ponerse a disposición del usuario en este panel, dependerán de las posibilidades del paquete de controles de usuario que se escoja para implementar la aplicación.

4.1.12 Panel de configuración "Idioma"



El Panel de configuración "Idioma", será un ítem de configuración del *Formulario de configuración*.

Este panel, permitirá elegir entre los distintos tipos de idioma disponibles en la aplicación.

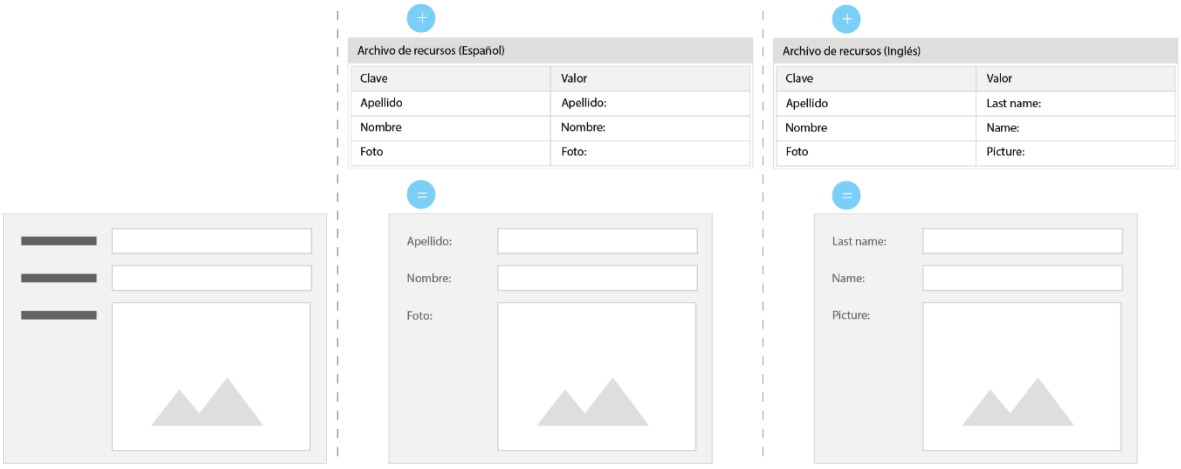
Los idiomas serán mostrados en un menú desplegable, mediante dos columnas. En la primera columna se mostrará el lenguaje y región, y en la segunda, el código del idioma.

Ejemplos lenguaje y región podrían ser "Español – España", "Inglés – Estados Unidos", y ejemplos de códigos de idioma "es-es", y "en-us", respectivamente.

4.1.13 Uso de archivos de recurso en la aplicación

Cada interfaz en la aplicación tendrá asociado un único *archivo de recursos*, en el cuál serán almacenados todos aquellos elementos de la misma, que puedan cambiar dependiendo de la región en donde la aplicación se utilizada, o del idioma que se muestre, pero que no requieran de la necesidad de recompilar la aplicación.

A continuación, se plantea un ejemplo sencillo, para una entidad del tipo "Persona", que tiene como atributos "Nombre", "Apellido", y "Foto". Se bosqueja una interfaz sencilla para la carga de sus datos, y se representa el resultado de la misma utilizando dos archivos de recursos en distintos idiomas.



4.2 Ambiente de desarrollo de software, tecnologías y plataformas a ser utilizadas

A continuación, se presenta una lista con todas las tecnologías involucradas en el desarrollo. Se seleccionaron tecnologías maduras del ecosistema de Microsoft .Net, siendo en su mayoría, tecnologías que tienen opciones de licenciamiento gratuito, o son de código abierto, dado que el alumno se propuso en esta primera etapa afrontar el desarrollo con costos mínimos.

- Las interfaces de usuario se desarrollarán empleando Microsoft Windows Forms, un marco de desarrollo de interfaces de usuario para crear aplicaciones cliente para Windows. Windows Forms es una tecnología madura que tuvo sus origines en el año 2002, y que en la actualidad ha sido rescrita en .Net Core (actual .Net 5), y dispuesta a la comunidad como código abierto.

- Como motor de base de datos se empleará SQL Server 2019. SQL Server es ideal para el cliente, dado que posee dos tipos de instancia: Express y Standard. La versión Express (gratuita) no tienen límites ni restricciones apreciables en el entorno de desarrollo, por lo cual los desarrolladores pueden emplearlas en sus estaciones de trabajo y conseguir una experiencia idéntica al uso de una instancia Standard (de pago). Incluso, el producto puede ser distribuido finalmente con la instancia Express para clientes pequeños, y escalada a instancias Standard en el momento que sea necesario.

- Se empleará Microsoft .Net Framework 4.8 para desarrollar el producto, y será codificado con C# 7.3.

- El desarrollo será efectuado empleando el IDE Microsoft Visual Studio Community 2019 (gratuito), el cual dispone de todas las herramientas necesarias para desarrollo.

- Aquellas librerías extras que sean empleadas en el proyecto, serán descargadas haciendo uso NuGet, una herramienta integrada en Visual Studio para le gestión de librerías. Esto facilita compartir el proyecto, y evita la configuración manual inicial.

- 

Se empleará el mapeador objeto relacional Microsoft Entity Framework. Se utilizará el esquema code first, el cual permite modelar las entidades de datos y sus relaciones mediante clases, y reflejar el modelo automáticamente en la base datos sin intervención del desarrollador.
El mapeador será configurado con migraciones automáticas. El uso de esta estrategia de mapeo, entre otras ventajas, permite incorporar el proyecto en distintas estaciones de trabajo de modo más sencillo, dado que no es necesario acompañar el proyecto por un Data Definition Language (DDL) y Data Manipulation Language (DML) inicial. Al ejecutar el proyecto, el mapeador verificará que tanto el modelo como los datos de la base de datos sean consistentes con la representación en código de las clases.

- 
La separación en capas se realizará utilizando distintos proyectos, dentro de una misma solución. Cada proyecto genera un ensamblado diferente (DLL). Esta estrategia, es la que permitiría en el futuro, utilizar las capas de acceso a datos y lógica de negocio en diferentes proyectos.

- La globalización se realizará empleando el uso de archivos de recurso (resx), en los cuales pueden almacenarse elementos en un formato de pares, clave-valor. Estos archivos de recurso, luego, pueden ser actualizados por fuera de la aplicación sin tener que recompilar el código. De este modo, por ejemplo, es posible entregar los archivos de recursos a personal de traducción sin conocimiento técnico, y luego incorporarlos nuevamente a la aplicación.

- Los requerimientos de interfaz del usuario serán resueltos con frameworks exclusivamente dedicados a esa finalidad. Aún no se ha resuelto la herramienta específica a ser utilizada, pero se emplearán soluciones líderes en el mercado.

- El despliegue de las aplicaciones se hará mediante Microsoft ClickOnce, que, entre otras estrategias, permite disponer de un servidor web para la descarga e instalación de la aplicación. Luego, es posible también establecer configuraciones para la actualización automática, definiendo criterios como la versión mínima instalada requerida para iniciar la aplicación, lo que posibilita lanzar actualizaciones obligatorias.

- 

Las pruebas unitarias serán desarrolladas empleando las librerías del framework xUnit, el cual tiene integración con Visual Studio, y es gratuito y de código abierto.

- 
El versionado del código se realizará mediante Git (emplearemos un servicio proporcionado por Atlassian, Bitbucket).

4.3 Orden de ejecución de las *User stories*

El proceso de relevamiento de las *User Stories* comenzó luego de que se alcanzó un *Modelo conceptual de la aplicación* maduro y que ya no requería de cambios.

Finalizado el relevamiento de las *User Stories*, las mismas se tabularon y ordenaron, tomando como primer criterio de ordenamiento la *prioridad en desarrollo*, y como segundo criterio la *prioridad en el negocio*.

El formato de representación de las *User Stories*, así como el criterio de ordenamiento de las mismas, fueron definido en detalle en apartados de la definición de la metodología.

ID	Nombre	Prioridad en el desarrollo	Prioridad en el negocio
1	Selección de herramientas esenciales del mercado	Alta	Baja
2	Desarrollo de la arquitectura inicial	Alta	Baja
3	Desarrollo del formulario principal	Alta	Baja
4	Generar lote de datos de prueba	Alta	Baja
5	Desarrollo de formulario de búsqueda (base)	Media	Alta
6	Formulario modal Aceptar-Cancelar (base)	Media	Alta
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas	Media	Alta
8	Formulario de búsqueda: Configurar plantilla	Media	Alta
9	Formulario de búsqueda: generación de salidas	Media	Alta
10	Formulario ABMC (base)	Media	Alta
11	Formulario de búsqueda (base): soporte para múltiples plantillas	Media	Alta
12	Formulario de búsqueda: Seleccionar plantilla	Media	Alta
13	Formulario de búsqueda: Nueva plantilla	Media	Alta
14	Gestor de formularios abiertos y entidades de datos en uso	Media	Baja
15	Formulario de configuración	Baja	Media
16	Formulario de configuración – “Idioma”	Baja	Media
17	Traducción de la aplicación completa al inglés	Baja	Media
18	Formulario de configuración – “Apariencia”	Baja	Media

4.4 Estimación del esfuerzo

Se definió previamente en la metodología que, para la estimación del esfuerzo se emplearía la unidad relativa *Story Point*.

Basado en la duración asignada para los eventos de la metodología, el peso relativo asignado a cada *Story Point* se estimó en aproximadamente 6,85 [hs], pudiendo ser cuantificado mediante el intervalo 6,5 [hs] ± 30 [min].

A continuación, se listan en orden las *User stories* relevadas, con su correspondiente estimación en *Story Point*, los cuales son totalizados al final:

ID	Nombre	Estimación [Story point]
1	Selección de herramientas esenciales del mercado	10
2	Desarrollo de la arquitectura inicial	4
3	Desarrollo del formulario principal	9
4	Generar lote de datos de prueba	1

5	Desarrollo de formulario de búsqueda (base)	9
6	Formulario modal Aceptar-Cancelar (base)	2
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas	8
8	Formulario de búsqueda: Configurar plantilla	7
9	Formulario de búsqueda: generación de salidas	3
10	Formulario ABMC (base)	10
11	Formulario de búsqueda (base): soporte para múltiples plantillas	10
12	Formulario de búsqueda: Seleccionar plantilla	3
13	Formulario de búsqueda: Nueva plantilla	3
14	Gestor de formularios abiertos y entidades de datos en uso	3
15	Formulario de configuración	2
16	Formulario de configuración – "Idioma"	2
17	Traducción de la aplicación completa al inglés	1
18	Formulario de configuración – "Apariencia"	2
	Total [Story point]	89

Se estimó previamente, en la *Duración de los eventos*, que un *Sprint* sería ejecutado en 2 [semanas], y en la definición de *Story Point* se estimó que, por cada *Sprint*, serían desarrollados aproximadamente 10 [Story point].

Teniendo en cuenta estos valores, es posible estimar la cantidad de *Sprints* que serán ejecutados en el proyecto:

$$Sprints\ totales = \left\lceil \frac{Story\ point\ totales}{Story\ points\ por\ Sprint} \right\rceil = \left\lceil \frac{89}{10} \right\rceil = 9\ [Sprint]$$

Contemplado que cada *Sprint* será ejecutado en 2 [semanas], las cuales pueden traducirse a 10 jornadas de trabajo, donde cada jornada laboral es de 8 [hs], es posible estimar la cantidad de horas estrictamente asociadas al proyecto laboral:

$$Hs.de\ trabajo \cong Sprints\ totales\ \times Días\ laborales\ por\ sprint\ \times Hs.de\ jorada\ laboral$$
$$\cong\ 9\ [Sprint] \times 10\ [días] \times 8\ [hs] \cong 720\ [hs]$$

Además, el alumno dedicará al menos 12 [hs] por cada semana laboral al aspecto académico:

$$Hs.aspecto\ académico = Sprints\ totales\ \times Semanas\ por\ Sprint\ \times Hs.dedic.semanal$$
$$= 9\ [Sprints]\ \times 2\ [Semanas] \times 12\ [hs] = 216\ [hs]$$

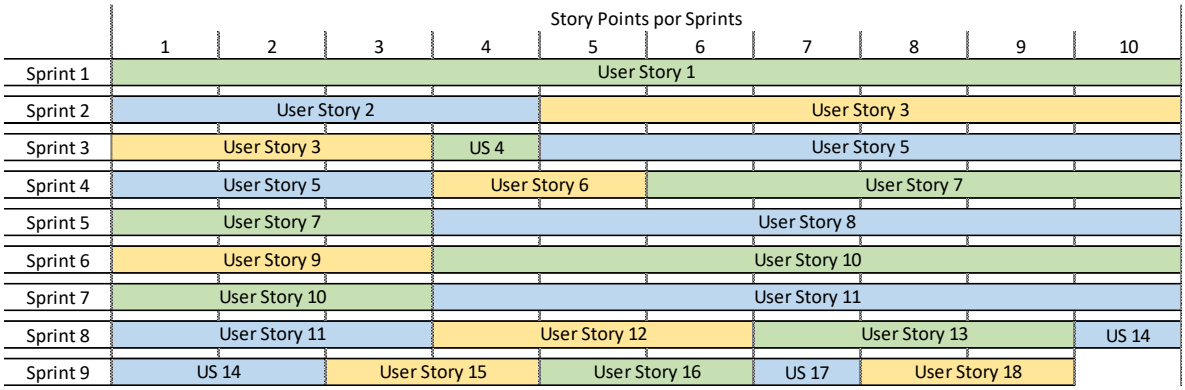
Se concluye entonces, que el esfuerzo total dado en horas que el alumno dedicará, contemplado el al aspecto estrictamente laboral, y además el académico:

$$Hs.dedicación\ total = Hs.de\ trabajo + Hs.aspecto\ académico = 720\ [hs] + 216\ [hs]$$
$$= 936\ [hs]$$

4.5 Product Backlog inicial

En la definición de la métrica *Story Point* luego de establecerse un peso relativo a dicha unidad, se estimó que, por cada *Sprint* serían desarrollados aproximadamente 10 [Story point]. Luego, en la *Estimación del esfuerzo* del proyecto, se dedujo que sería necesaria la ejecución de 9 [Sprint] para concretar el proyecto.

Si se conjugan los *Story Points* disponibles por cada *Sprint*, con los *Story Points* asignados a cada *User Story* (que pueden consultarse en la *Estimación del esfuerzo* del proyecto), y se dispone a cada *User Story* en el *Orden de ejecución* resultante basado en prioridades, se hace visible que algunas *User Stories* podrían comenzar dentro de un *Sprint*, y finalizar en el siguiente, como se representa en el siguiente gráfico:



Este análisis tuvo lugar como instancia previa a la configuración del Product Backlog, en el cual se celebraron debates y reflexiones entre el alumno y el *Scrum Master*, que el alumno encuentra merecedores de ser mencionados, dado que dieron origen al criterio asumido frente a tal situación.

Al notar la situación, en primera instancia el alumno buscó referencias asociadas en las bibliografías de los autores de la metodología, y en otros autores populares, pero no existen menciones directas a la problemática.

El alumno propone al *Scrum Master* -tal vez a modo de solución obvia- la partición de las *Users Stories* que se hallaran abarcando más de un *Sprint*, de modo de hacerlas encajar de forma perfecta dentro del *Sprint*. El *Scrum Master* rechazó el ejercicio bajo el siguiente argumento: *"Tal y como están definidas ahora, las User Stories tienen sentido para aquellos a quienes se les va a entregar funcionalidad; formar particiones más pequeñas, podría hacer que la razón de ser de las User Stories sea difícil de comprender, e incluso, que el rol destinatario [13]de una User Story cambie"*.

Ken Schwaber (cocreador de Scrum) hace referencia a situaciones en las cuales una *User Story* es interrumpida por alguna razón, mencionando que, de cualquier manera, debería haberse hecho un progreso demostrable al final del *Sprint*, y que si bien, tal vez lo desarrollado por sí solo no proporcione "valor" al cliente, podemos exhibir lo hecho, y usarlo para mostrar el progreso hacia el objetivo final

[13] El "rol destinatario", en términos de las palabras utilizadas por el *Scrum Mater*, hace referencia al "*Como un…"* en la definición de las tarjetas con las que se representan las *User Stories* en el presente proyecto.

Finalmente, como criterio generalizado para la situación, siempre que ocurra que una *User Story* abarque más de un Sprint, lo realizado respecto a la misma será mencionado en cada *Sprint Review* de cada *Sprint* que abarque, efectuando un contraste entre el objetivo de la *User Story* y lo que ya está listo, pero se hará una presentación de la finalización de la misma -y si es posible, una demo-, en el *Sprint Review* del *Sprint* en el cual se la finalice.

Teniendo en cuenta los criterios previamente mencionados, y compilando el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo*, se da origen al Product Backlog inicial[14]:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
1	Selección de herramientas esenciales del mercado	Alta	10	1
2	Desarrollo de la arquitectura inicial	Alta	4	2
3	Desarrollo del formulario principal	Alta	9	2 y 3
4	Generar lote de datos de prueba	Alta	1	3
5	Desarrollo de formulario de búsqueda (base)	Media	9	3 y 4
6	Formulario modal Aceptar-Cancelar (base)	Media	2	4
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas	Media	8	4 y 5
8	Formulario de búsqueda: Configurar plantilla	Media	7	5
9	Formulario de búsqueda: generación de salidas	Media	3	6
10	Formulario ABMC (base)	Media	10	6 y 7
11	Formulario de búsqueda (base): soporte para múltiples plantillas	Media	10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso	Media	3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

[14] Se mencionó en la definición del *Product Backlog*, que, en el presente proyecto de desarrollo de software, se enfatizaría la tarea de refinamiento y estimación de los ítems del *Product Backlog* antes de comenzar el proyecto, a los efectos de obtener un *Product Backlog* tan estático y detallado como sea posible, que posibilite cumplir con las fechas críticas del aspecto académico de esta presentación. Aun así, no se pretende modificar las características de dinamismo del Product Backlog. El mismo será adecuado siempre que la situación lo amerite, tal y como su definición lo establece.

4.6 Recursos humanos del proyecto

4.6.1 Alumno

- Rol / Función: Alumno que desarrolla el presente Proyecto Final de Carrera. Asume el rol *"Desarrollador único"* en la metodología ad-hoc desarrollada (rol sustituto a *"Development team"* de Scrum).
- Disponibilidad de tiempo:
  El alumno tiene una relación contractual con la empresa solicitante, definida por una dedicación diaria de 8 horas, 5 días a la semana, de lunes a viernes, durante el plazo que dure el proyecto. Por su parte, el alumno dedicará semanalmente al menos 12 horas a los aspectos que se relacionan estrictamente al presente Proyecto Final de Carrera.
  Totalizando, el alumno dedicará al menos 52 horas semanales al proyecto. Contemplando que serán ejecutados 9 *Sprints*, definidos con 2 semanas de duración, el alumno dedicará al menos 936 horas al Proyecto Final de Carrera completo.
  Pueden consultarse las estimaciones en detalle, en la *Estimación de esfuerzo* del proyecto.

4.6.2 Director de Proyecto Final de Carrera

- Rol / Función. Director del presente Proyecto Final de Carrera.
- Disponibilidad de tiempo: En función de la demanda de tiempo requerida por el proyecto.

4.6.3 Líder de desarrollo de la empresa solicitante

- Rol / Función. Asume el rol de *Product Owner*, y de *Scrum Master* en la metodología ad-hoc desarrollada.
- Disponibilidad de tiempo:
  Tomando como referencia las *duración de los eventos* establecidas en el marco que define la metodología ad-hoc a ser utilizada en este proyecto, y contemplando las asistencias de roles de Scrum a los distintos eventos, podemos estimar la dedicación por *Sprint* (definido con 2 semanas de duración) del líder de desarrollo de la empresa solicitante:

Nombre del evento	Duración	Participación de Scrum Master	Participación de Product Owner
Sprint Planning	4 [horas]	Si	Si
Daily Scrum	15 [minutos]	Si	No
Sprint Review	2 [horas]	Si	Si
Sprint Retrospective	2 [horas]	Si	No

  Totalizando, la dedicación cada 2 semanas, será de al menos 10 horas y 30 minutos. Contemplando que serán ejecutados 9 S*prints*, definidos con 2 semanas de duración, la dedicación será de al menos 94 horas con 30 minutos al proyecto de relación contractual.

Es importante destacar, que se menciona que esta persona "al menos" tendrá la dedicación de las horas anteriormente estimadas, dado que, es el tiempo mínimo que deberá dedicar a las ceremonias de Scrum como *Scrum Master*. Eventualmente, puede ocurrir que en alguna de las

ceremonias ambos roles merezcan una dedicación exclusiva, en cuyo caso, los tiempos de dedicación serán mayores.

4.7 Estimación de duración del proyecto

Teniendo en cuenta la *Estimación del esfuerzo* del presente proyecto, en la cual, partiendo de la estimación de *Story Points* totales demandados por el proyecto, se concluye que serán realizados 9 *Sprints*, y teniendo en cuenta que en la *Duración de los eventos* se definió cada *Sprint* con 2 [semanas] de trabajo:

$$Semanas\ de\ duración\ del\ proyecto = Sprints\ totales\ \times Semas\ por\ Sprint$$
$$= 9\ [Sprints] \times 2\ [Semana] = 18\ [semanas]$$

Se concluye que, independientemente de la fecha de inicio del proyecto, el mismos tendrá una demanda estimada en 18 [semanas].

Dependiendo de la fecha precisa de inicio del proyecto, será necesario contemplar los feriados no laborables y generar un corrimiento.

Como estimación más precisa, puede establecerse la cantidad de días laborales que demandará el proyecto:

$$Días\ laborales\ demandados\ por\ el\ proyecto$$
$$= Sprints\ totales\ \times Días\ laborales\ por\ Sprint = 9\ [Sprints] \times 10\ [Días]$$
$$= 90\ [días]$$

4.8 Cronograma de avance del proyecto

En la definición de la en la *Duración de los eventos* se definió que cada *Sprint* sería ejecutado en 2 [semanas], las cuales pueden traducirse a 10 jornadas de trabajo cada uno, donde cada jornada laboral es de 8 [hs].

Luego, se determinó en la *Estimación de la duración del proyecto*, que, independientemente de la fecha de inicio del mismo, tendría una demanda estimada en 18 [semanas], sin contemplar los feriados no laborables, o, como estimación más precisa, 90 jornadas de trabajo.

En el *Plan de Proyecto* del presente Proyecto Final de Carrera, se establecieron las siguientes fechas estimadas, relevantes para el cronograma:

Inicio del proyecto	Segunda quincena de octubre de 2020
Probable presentación escrita del proyecto	Primera quincena de marzo de 2021

Considerando las 18 [semanas] de trabajo, desde la segunda quincena de octubre de 2020, el proyecto finalizaría aproximadamente en febrero de 2021. En dicho intervalo, se encuentran los siguientes feriados no laborables:

23/11/2020	Día de la Soberanía Nacional (20/11).
07/12/2020	Feriado con Fines Turísticos. Dia no laborable.

08/12/2020	Inmaculada Concepción de María. Feriado inamovible.
25/12/2020	Navidad.
01/01/2021	Año nuevo.

Como resultado de la *Estimación del esfuerzo* se dedujo la realización de 9 *Sprints*, que según la *Duración de los eventos* se han determinado en 2 [semanas] cada uno. Contemplando los feriados en el intervalo mencionado previamente, y teniendo presente que el proyecto comenzará el **19 de octubre de 2020**, el cronograma de avance para el presente proyecto resulta:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
1	19/10/2020	23/10/2020	1
2	26/10/2020	30/10/2020	
3	02/11/2020	06/11/2020	2
4	09/11/2020	13/11/2020	
5	16/11/2020	20/11/2020	3
6	24/11/2020	30/11/2020	
7	01/12/2020	09/12/2020	4
8	10/12/2020	16/12/2020	
9	17/12/2020	23/12/2020	5
10	24/12/2020	31/12/2020	
11	04/01/2021	08/01/2021	6
12	11/01/2021	15/01/2021	
13	18/01/2021	22/01/2021	7
14	25/01/2021	29/01/2021	
15	25/01/2021	29/01/2021	8
16	01/02/2021	05/02/2021	
17	08/02/2021	12/02/2021	9
18	15/02/2021	19/02/2021	

Basado en el cronograma resultante, y teniendo en cuenta la definición de los *Eventos* de la metodología -considerando el momento en el que ocurren dentro de un *Sprint*-, se representa en la siguiente tabla los días en los que ocurrirán los mismos:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**1**	19/10/2020	30/10/2020		
	2	02/11/2020	13/11/2020		
	3	16/11/2020	30/11/2020		
	4	01/12/2020	16/12/2020		
	5	16/11/2020	30/11/2020		

	6	04/01/2021	15/01/2021
	7	18/01/2021	29/01/2021
	8	25/01/2021	05/02/2021
	9	08/02/2021	19/02/2021

4.9 Arquitectura

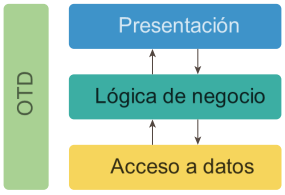
La arquitectura de base para el desarrollo se definió ente el alumno y el personal técnico de la empresa solicitante, en reuniones en las que el alumno presentó diferentes alternativas, con sus respectivas comparativas.

Se tuvieron en cuenta diferentes aspectos al momento de la selección, como la familiaridad de los desarrolladores de la empresa solicitantes con ciertos estilos arquitecturales, y las características del proyecto en curso.

Finalmente, se planteó un estilo arquitectural de alto nivel, el que luego fue implementado con las tecnologías seleccionadas para el desarrollo.

4.9.1 Propuesta independiente de la tecnología: estilo arquitectural en capas

El estilo arquitectural en capas se basa en una distribución jerárquica de los roles y responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan.



4.9.2 Implementación en .Net

La separación de responsabilidades en la arquitectura, en la visión más general de la misma, se disgregó en tres capas: “Acceso a datos”, “Lógica de negocio”, y “Presentación”. Cada capa lógica fue implementada mediante una librería de clases.

Transversalmente entre las tres capas anteriormente mencionadas, se comparte una capa lógica con entidades con finalidad de OTD (“objetos de trasferencia de datos”, habitualmente llamados DTO, del inglés *“Data transfer object”*), para comunicar las mismas entre sí. Los OTD son clases planas (sin comportamiento).

Las particularidades de cada capa (las tecnologías involucradas, la forma en que las mismas están estructuradas, etc.), serán presentadas a continuación, de forma individual para cada capa.

4.9.3 Capa “Acceso a datos”

En la capa de *acceso a datos* se configuró el mapeador objeto-relacional *Microsoft Entity Framework*, empleando el esquema *Code First* (*“Código primero”*), el cual inspecciona las clases que representan al modelo, y luego mediante un conjunto de reglas y convenciones determinar cómo

esas clases y sus relaciones describen un modelo, y cómo ese modelo debe mapearse en la base de datos.

También, se añadió *LINQ to Entities*, el cual permite ejecutar consultas escritas en C# respecto a las entidades del modelo de datos a través de *Entity Framework*, las cuales luego son transformadas a SQL de forma eficiente, y ejecutadas en el motor de base de datos.

En el mapeador objeto-relacional, se incorporó una regla que permite actualizar el modelo de datos de la base de datos, al último modelo de datos representado por las clases, siempre que la aplicación se ejecute en modo *Debug* (es decir, que la misma no esté en entorno de producción); además, para esto modo, se habilitó la *pérdida datos*, lo que implica que si alguno de los cambios efectuados en el modelo conlleva la destrucción de los datos contenidos en la base datos, los mismos serán eliminados para proceder con la actualización.

Finalmente, se desarrolló una clase que, mediante un conjunto de reglas sencillas, permite definir *semillas* que serán insertadas en la base de datos cuando la aplicación inicie en modo *Debug*. Esto permitirá a los desarrolladores contar con datos de prueba conocidos en la base de datos, independientemente que la misma sea reestablecida debido a cambios en sus tablas.

El conjunto de configuraciones mencionado, permite que el despliegue del proyecto en una nueva terminal, o para un nuevo desarrollador, se efectué rápidamente, sin la necesidad de configuraciones particulares a la base datos, y/o de importar datos de prueba a la misma, dado que tales tareas se realizarán automáticamente.

Luego, cada cambio efectuado al modelo de datos a través de cambios en las clases, será enviado al repositorio de código fuente mediante el sistema de versionado, sin la necesidad de compartir la última versión de un DDL (Data Definition Language, *"Lenguaje de definición de datos"*).

4.9.4 Capa "Lógica de negocio"

La capa de la *Lógica de negocio*, consta de la lógica que respecta a las reglas de cómo las entidades de negocio se crean, modifican, eliminan, y de cómo dichas entidades interactúan con otras entidades de negocio. También, respecta a cómo tales entidades se sirven a una capa superior, y a todas otras operaciones que tengan que ver con el flujo de trabajo de las entidades de datos.

Usa los servicios de la capa de A*cceso a datos* para efectuar todas las operaciones de persistencia, pero las reglas previas implicadas a tales operaciones no trascienden de la *Lógica de negocio*.

De forma más general, es en esta capa, en la cual se encontrarán las entidades con las responsabilidades tales, para manejar toda la operatoria del sistema, que se encuentre entre las operaciones de persistencia, y las operaciones de la *capa de presentación*.

Las distintas entidades de esta capa -clases de la *Capa de negocio*-, habitualmente llamadas "Controladores", han sido implementadas mediante clases estáticas, dado que las mismas no están asociadas con instancias particulares de objetos, sino que fueron diseñadas para manejar instancias o grupos de instancias relacionadas, de entidades de la capa de A*cceso a datos*, de forma general.

Se añadió también en esta capa, una clase estática con la finalidad de ser un *Repositorio de parámetros*. El *Repositorio de parámetros*, está compuesto por listas estáticas de entidades de datos, que serán llenadas mediante la estrategia *Lazy loading* (*"carga perezosa"*) conforme sean requeridas.

Las entidades de datos que serán consideradas "parámetros", son entidades definidas por el solicitante, para las cuales se tiene certeza absoluta que no cambiarán en tiempo de ejecución, o que, de ser actualizadas en tiempo de ejecución, no causarán un impacto en el funcionamiento de otras entidades parámetros, lo cual fuerce a recargar las listas de parámetros nuevamente.

4.9.5 Capa "Presentación"

La *capa de presentación*, reúne los aspectos del software relacionados a la interacción con el usuario final, y a cómo dicho usuario percibe el sistema. Incluye las interfaces gráficas y el comportamiento de las mismas, la lógica que determina como la información será mostrada al usuario, y también las reglas de como la información podrá ser ingresada. Esta capa se sirve de la *capa de lógica de negocio* para toda operación no relacionada con la vista, y se comunica con la misma mediante las entidades de la *capa OTD*.

La tecnología principal seleccionada para la vista es *Microsoft Windows Forms*, y sus controles de usuario fueron extendidos mediante el paquete de controles de usuario de la empresa *DevExpress*.

Windows Forms por defecto propone por una arquitectura conducida por eventos (del inglés *event-driven*). Tanto los formularios como los controles de usuario, exponen un conjunto predefinido de eventos a los cuales se les pueden asignar comportamientos. Si ocurre uno de estos eventos, y hay un código asociado, ese código se invoca.

En *Windows Forms*, las responsabilidades dentro de las vistas (tanto para formularios como para controles de usuario) se separan mediante una división lógica llamada *Code-Beside*, lo cual permite que una misma clase sea definida en múltiples archivos de código fuente, llamados *Partial Types* (*"tipos parciales"*).

Así, inicialmente, cada formulario o control de usuario está compuesto por el diseño gráfico de una vista (un lienzo en el cual se puede ubicar controles de usuario), una clase parcial para los comportamientos inherentes a la vista (como, por ejemplo, la configuración particular de objetos gráficos), y una clase parcial para la lógica de la vista (como, por ejemplo, el comportamiento detrás de los eventos que puedan ser lanzados por la vista).

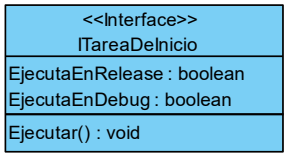
Dentro de la *capa de presentación*, se generaron divisiones lógicas en formato de carpetas (las cuales a su vez representan *namespaces* para las clases contenidas en ellas) para albergar diferentes elementos de la vista, con diferentes responsabilidades. Inicialmente, se crearon cinco carpetas: “Archivos de recursos”, “Controles de usuarios”, “Formularios”, “Interfaces” y “Tareas de inicio”.

Dentro de cada carpeta, se generarán divisiones lógicas acordes a las jerarquías que se presenten conforme el sistema sea ampliado. Podría ser que, por ejemplo, los formularios sean agrupados por el dominio al que refieren, o que los controles de usuario sean agrupados por alguna funcionalidad específica en común.

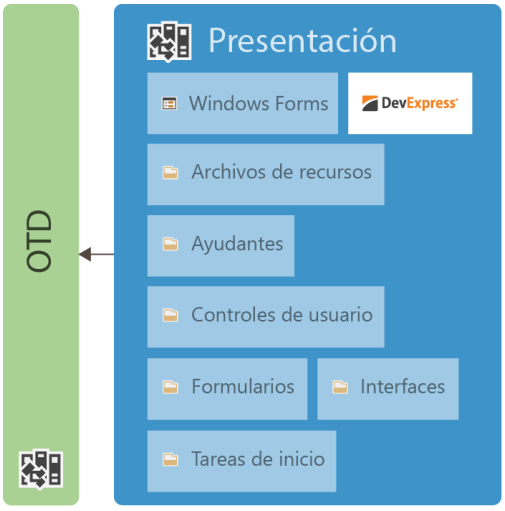
Las carpetas “Formularios” y “Controles de usuario” contienen los elementos gráficos propiamente mencionados con su lógica asociada.

La carpeta “Archivos de recursos”, contiene los archivos con las diferentes traducciones y adaptaciones geográficas de formularios y controles de usuario del sistema. A cada archivo de recurso se le asignará el mismo nombre, y ubicación relativa dentro de la carpeta de “Archivos de recurso”, de elemento al cual traducen.

“Tareas de inicio” contiene clases, que heredan de la interfaz *ITareaDeInicio*, las cuales contienen lógica específica de rutinas que deben ser ejecutadas sólo al inicio de la aplicación, dependiendo si la aplicación está siendo ejecutada en modo *Debug* (“en depuración”), o *Release* (“para liberación”).



Todas las interfaces que sean necesarias para especificar el comportamiento de clases, en la capa de presentación, serán ubicadas dentro de la carpeta “Interfaces”.



4.9.6 Vista general de la arquitectura resultante

Para facilitar la comprensión de la interacción entre los diferentes componentes a alto nivel, se presenta un diagrama de componentes, en el cual se representan las tres capas lógicas respectivas al modelo arquitectural en capas, y su relación con la capa transversal y librerías externas fundamentales.

La Figura 1 se exhibe la vista general de la arquitectura resultante.

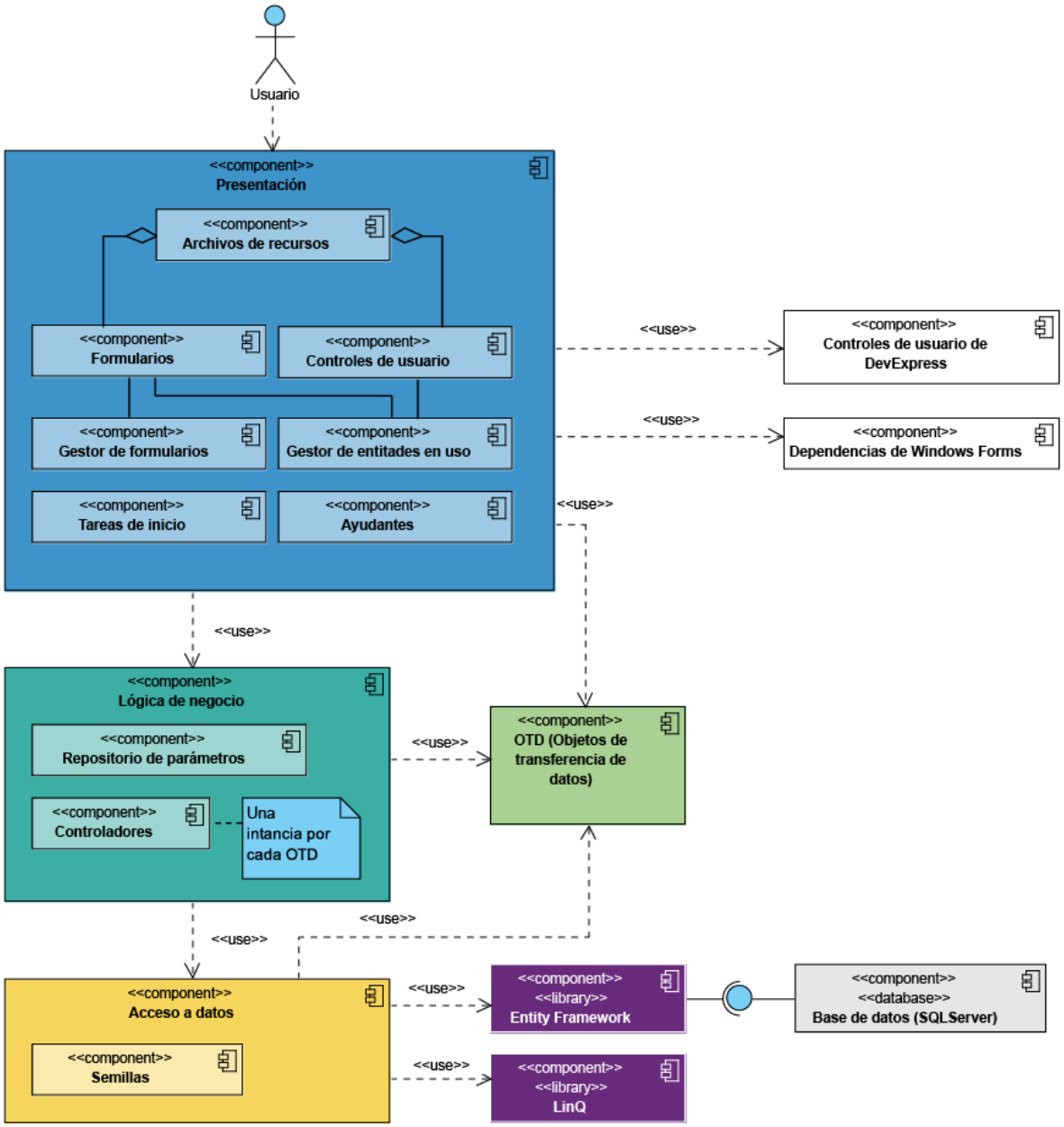


Figura 1

4.10 Seguridad

En el presente proyecto, se contempla a la seguridad en el desarrollo desde dos perspectivas diferentes: la seguridad en las herramientas de desarrollo utilizadas, pertenecientes al ecosistema de Microsoft .Net Framework, y las características de seguridad genéricas o reutilizables provistas por el framework en desarrollo por el alumno.

4.10.1 Características de seguridad en el framework en desarrollo

En las diversas aplicaciones que actualmente el cliente tiene en producción, las mismas manejan esquemas de seguridad no homogéneos.

Algunas de las aplicaciones que el alumno pudo analizar desde una perspectiva rápida utilizan esquemas de licenciamiento de activación por códigos, dentro de las cuales, algunas ofrecen la activación de características en las aplicaciones a través de esquemas de privilegios (versiones con diferentes niveles de restricciones).

Otras aplicaciones, utilizan activaciones específicas para terminales físicas, para las cuales se genera una llave única en base a características del hardware del cliente, lo que implica que, si alguno de los componentes se cambia, o si se intenta replicar el software en otro equipo, la licencia saldrá de vigencia.

El cliente también dispone de aplicaciones móviles desarrolladas por terceros con sistemas de autenticación basadas en usuario-contraseña, y con esquemas de autorización basados en roles y privilegios.

Aún no se ha definido con políticas de seguridad del tipo autenticación, autorización, y esquemas de roles y permisos asociados al framework en desarrollo.

En conversaciones relajadas con el cliente, se han hecho menciones respecto a los pasos a seguir en el futuro, luego de alcanzar versiones del framework con las características mínimas e indispensables.

Algunas de las tareas que serán realizadas en el futuro, implican el análisis de todas las soluciones del cliente, en miras de unificar la autenticación y autorización en el ecosistema del cliente, para luego poder dar soporte a dichas actividades dentro del framework. Por esta razón, estas actividades quedan fuera de los alcances del presente Proyecto Final de Carrera.

4.10.2 Características de seguridad en Microsoft .Net Framework

El presente proyecto descansa en las características de seguridad provistas en el ecosistema de tecnologías de Microsoft .Net Framework, por esta razón, se presenta una breve reseña de las mismas.

Microsoft .Net es una tecnología madura, que tuvo su primera versión estable en el año 2002, y que desde entonces recibió sucesivas mejoras incrementales de forma privada, hasta el año 2016, en cual una nueva versión del proyecto -.Net Core- fue lanzada a la comunidad como código abierto.

Por esta razón, el .Net Framework recibe un exigente mantenimiento correctivo y evolutivo de parte de sus creadores, pero también lo recibe desde una basta comunidad de usuarios. El producto de la interacción entre Microsoft y la comunidad de usuarios, consiguen que el producto resultante sea altamente fiable, y esté siempre atento a las últimas exigencias de seguridad.

.Net provee a los desarrolladores control de seguridad granular sobre sus aplicaciones, proveyendo un conjunto de herramientas fáciles de implementar en aspectos de autenticación, autorización, rutinas de criptografía, etc.

Estas herramientas eliminan muchos de los principales riesgos de seguridad que enfrentan las aplicaciones hoy en día debido a código defectuoso, y elimina la responsabilidad de tener que tomar decisiones de seguridad críticas.

Las funciones de seguridad de .Net, como lo son la seguridad de acceso al código, el proceso de verificación, el soporte de criptografía, el almacenamiento aislado, etc., trabajan en conjunto para proporcionar una plataforma de desarrollo sólida, que permite desarrollar todo tipo de aplicaciones de software, tanto del lado del cliente como del servidor.

Con .Net, Microsoft se propuso proporcionar una arquitectura para manejar el riesgo de seguridad en las aplicaciones de software, dicha arquitectura recibe el nombre de *managed code* [15](“código administrado”).

La ejecución de aplicaciones a través de *managed code*, controla de forma transparente el comportamiento del código incluso en las circunstancias más adversas, de modo que los riesgos inherentes a todo tipo de aplicaciones, tanto del lado del cliente como del servidor, se reducen considerablemente.

La conjunción de las características [16]mencionadas, resultan en desarrollos que pueden resistir ataques de seguridad conocidos hoy, y en el futuro.

4.11 Testing

El Testing[17] de Software es toda una disciplina en la ingeniería de software compuesta por procesos, métodos de trabajo y herramientas para identificar defectos en el software, con la finalidad de alcanzar la estabilidad del mismo.

Permite que lo que se está construyendo se realice de manera correcta, contrastando los resultados deseados con los resultados generados por el código, en búsqueda de discrepancias. Es una forma de detectar y prevenir posibles desviaciones del software antes de que el mismo sea operable.

El Testing es una actividad que no debería ser pensada al final de un desarrollo, sino que debería evolucionar parlamente al mismo. Por esta razón, en el presente proyecto se da cobertura

[15] Documentación oficial de *managed code*: https://docs.microsoft.com/es-es/dotnet/standard/managed-code
[16] Documentación oficial de seguridad en .Net: https://docs.microsoft.com/es-es/dotnet/standard/security/
[17] Documentación oficial de Testing en .Net: https://docs.microsoft.com/es-es/dotnet/core/testing/

a esta disciplina, brindando lineamientos de base para desarrollar y ejecutar pruebas en los componentes del framework.

Existen numerosas técnicas y estrategias para testear el software, cada una de ellas con un objetivo concreto en particular. Entre los tipos de tests más populares, se encuentran las pruebas unitarias, las pruebas de integración, y las pruebas de carga.

A grandes rasgos, las pruebas unitarias son pruebas que corroboran componentes o métodos de software individuales, también conocidos como "unidad de trabajo".

Las pruebas unitarias solo deben probar el código que esté bajo el control del desarrollador, y no deben probar aspectos relacionados a la infraestructura (funcionamiento de sistema de archivos, base de datos, redes, etc.).

Una prueba de integración, en cambio, se diferencia de una prueba unitaria en que corrobora dos o más componentes de software funcionando en conjunto, es decir, en "integración". Además, las pruebas de integración funcionan en un espectro más amplio, habitualmente incluyendo asuntos relacionados a la arquitectura.

Dadas las incumbencias de las pruebas unitarias, en el presente proyecto se hará énfasis en el uso de las mismas, brindando ejemplos de su implementación[18].

Otros tipos de pruebas, con incumbencias más abarcativas (que escapen del ámbito estricto del desarrollador), podrían ser analizadas en el futuro, conforme el framework evolucione, y los requerimientos de la organización lo demanden.

4.11.1 Patrón de escritura de pruebas unitarias

Una prueba unitaria entonces, es un procedimiento que ejercita una unidad de trabajo de software para determinar si el comportamiento del mismo es el esperado, e informa lo ocurrido al finalizar.

Dado que podría realizarse dicha actividad de contraste o de ejercicio de componentes en un sinfín de maneras -tantas como la creatividad permita-, es de suma utilidad acordar un patrón de escritura de las pruebas que permita a los desarrolladores realizarlo de forma homogénea en un equipo de trabajo.

El patrón AAA (Arrange-Act-Assert, “Organizar-Actuar-Afirmar”) se ha convertido en un estándar en la industria de software, y sugiere que se divida la prueba unitaria en las mencionadas tres secciones.

En la sección de *arrange* (“organizar”), se dispone el código necesario para configurar la prueba, es decir, las premisas de la prueba específica. En esta sección se crean todos los objetos necesarios, como es el caso, por ejemplo, de objetos que simulen una determinada situación (llamados *mocks*).

[18] Documentación oficial de *pruebas unitarias* en .Net: https://docs.microsoft.com/es-es/dotnet/core/testing/unit-testing-with-dotnet-test

Luego está la sección *act* (“actuar”), que es el punto en el que se debe realizar la invocación del método que se está probando, utilizando los objetos creados en la sección *act*.

Finalmente, en la sección *assert* (“afirmar”), se corrobora si se cumplieron las expectativas. Este es el punto de especial interés, en el que se recabará la información necesaria para determinar si la pieza de software testeada se comportó como se esperaba o no.

4.11.2 Convención de nombres de pruebas unitarias

La denominación de las pruebas es tan importante para los equipos en proyectos a largo plazo, como lo es la convención de escritura de código de pruebas unitarias definida previamente.

Definir y aplicar una convención de nombre para las pruebas, asegura que cada nombre de prueba será representativo de su finalidad, haciéndolo fácilmente comprensible para cualquier desarrollador en el equipo.

Hay muchos tipos de convenciones de nombres para pruebas, pero hay ciertos lineamientos que brindan una orientación al momento de elegir uno:

- El nombre de la prueba debe expresar un requisito específico.
- El nombre de la prueba podría incluir la entrada o el estado esperado, y el resultado esperado para esa entrada o estado.
- El nombre de la prueba debe presentarse como una declaración o un hecho preciso que exprese un flujo de trabajo y los resultados obtenidos con dicho flujo de trabajo.
- El nombre de la prueba puede incluir el nombre del método o la clase probados.

El alumno utilizará una convención de nombres para los métodos de pruebas unitarias, definido según NombreDelMétodoQueSePrueba_SituaciónQueSePrueba_ResultadoEsperado.

5 Implementación y funcionamiento en tiempo de ejecución

Tomando como punto de referencia la *arquitectura* del proyecto, a continuación, se presentarán los componentes reutilizables más relevantes el proyecto, con un ejemplo de su implementación, y su resultado en tiempo de ejecución.

La documentación del proceso de desarrollo de los compontes, así como los criterios tomados para la implementación de los mismos, y sus estados finales luego de las modificaciones iterativas, se encuentra en el anexo *Documentación generada en la realización de las User Stories*.

Para brindar al lector una presentación rápida de los componentes, y a modo de introducción de la lectura técnica, se presentan en este capítulo los componentes reutilizables, su razón de ser, su ubicación dentro de la arquitectura, y los resultados que producen los mismos en tiempo de ejecución.

5.1 Escenario de ejemplo

Para efectuar pruebas tanto visuales como funcionales en el framework en desarrollo, se planteó un modelo de datos con un escenario cotidiano que se pueda interpretar con facilidad.

El dominio elegido fue el que relaciona a *Propietarios* de mascotas, con *Mascotas*, donde los *Propietarios* tienen una relación con la *Ciudad* en la que viven, y las *Mascotas*, con su *TipoDeAnimal*.

En la Figura 2 se presenta el diagrama entidad-relación que representa el dominio de pruebas.

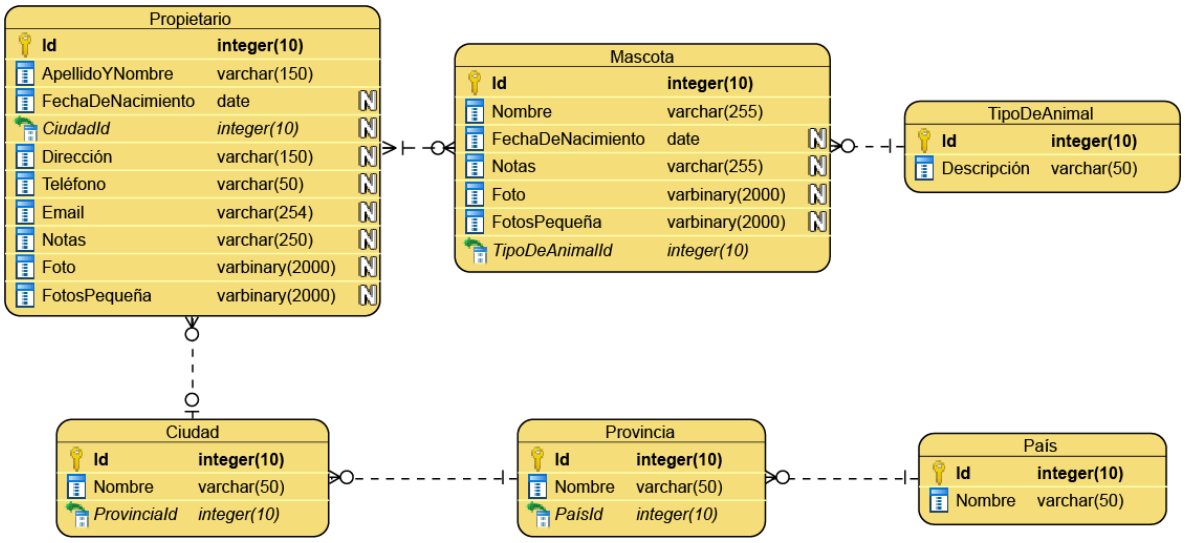


Figura 2

En la implementación, cada una de estas entidades es representada por una clase, en la capa de *Objetos de transferencia de datos*, las cuales son referenciadas por la *Capa de acceso a datos*, y posteriormente mapeadas a la base de datos por *Entity Framework*.

A modo de ejemplo, se presenta la estructura de la clase utilizadas para representar a la entidad *Mascota*:

```
public class Mascota
{
    public Mascota()
    {
        this.Propietarios = new HashSet<Propietario>();
    }

    public int Id { get; set; }

    [Required]
    [MaxLength(150)]
    public string Nombre { get; set; }

    public DateTime FechaDeNacimiento { get; set; }

    public int TipoDeAnimalId { get; set; }
    public virtual TipoDeAnimal TipoDeAnimal { get; set; }


    [MaxLength(250)]
    public string Notas { get; set; }

    public byte[] Foto { get; set; }

    public byte[] FotosPequeña { get; set; }

    public virtual ICollection<Propietario> Propietarios { get; set; }
}
```

5.1.1 Lote de datos de prueba y semillas de la base de datos

Para contar con datos iniciales, los cuales permitan hacer uso de las herramientas provistas por el framework desde la primera ejecución del sistema, se generaron lotes de datos de prueba para todas las entidades de pruebas propuestas en el escenario de ejemplo: *Ciudad*, *Provincia*, *País*, *Propietario*, *Mascota*, y Especie.

Para *Ciudad*, *Provincia* y *País*, se utilizaron todas las provincias y ciudades de la Argentina. Para las *Propietario* y *Mascota* se emplearon datos de personajes de fantasía consumidos desde un endpoint público. En el caso de *Especie*, los datos se completaron manualmente.

El código relacionado a las semillas de cada clase, fue dispuesto en la clase estática *ContenedorDeSemillas*, en la *Capa de Acceso a datos*.

La inserción de las semillas se realizó mediante el criterio *AddOrUpdate* (“*Agregar o actualizar*”), lo que implica que, si alguno de los registros aún no existe en la base de datos, será agregado, caso contrario, si un registro ya existe, pero su contenido es diferente al de la semilla, será actualizado a los valores definidos en la semilla. Es importante recordad, que la ejecución del contenedor de semillas se realiza sólo si la aplicación se ejecuta en modo *Debug*.

A modo de ejemplo, para mostrar cual es el procedimiento de definición de semillas, se cita el código de semillas de la clase *Especie*:

```
#region TipoDeAnimal

List<TipoDeAnimal> tiposDeAnimal = new List<TipoDeAnimal>();
```

```
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 1, Descripción = "es-AR:Leon|en-US:Lion" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 2, Descripción = "es-AR:Lémur|en-US:Lemur" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 3, Descripción = "es-AR:Lémur ratón|en-US:Mouse lemur"
});
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 4, Descripción = "es-AR:Lémur aye-aye|en-US:Aye-aye
lemur" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 5, Descripción = "es-AR:Hipopótamo|en-US:Hippopotamus"
});
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 6, Descripción = "es-AR:Zebra|en-US:Zebra" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 7, Descripción = "es-AR:Pingüino|en-US:Penguin" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 8, Descripción = "es-AR:Mono|en-US:Monkey" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 9, Descripción = "es-AR:Cerdo|en-US:Pig" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 10, Descripción = "es-AR:Perro|en-US:Dog" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 11, Descripción = "es-AR:Jirafa|en-US:Giraffe" });


context.TiposDeAnimal.AddOrUpdate(
    e => e.Id,
    tiposDeAnimal.ToArray()
);

#endregion // TipoDeAnimal
```

En el campo *Descripción* de la clase *TipoDeAnimal*, se provee su contenido especificando el idioma al que pertenece. De esta manera, en tiempo de ejecución, independientemente del lugar en la aplicación en el que se utilicen los datos, los mismos serán mostrados acorde a la selección de idioma en curso.

5.2 Componentes reutilizables de la capa de presentación

Dado que la finalidad de este ejemplo es proporcionar un contraste entre la cantidad de trabajo necesario para implementar una funcionalidad, y la funcionalidad resultante en tiempo de ejecución, a continuación, se proporciona un desglose de componentes con mayor especificidad de la *capa de presentación*.

Se especifica un diagrama de componentes con mayor nivel de detalle de la *capa de presentación*, orientado en particular a los componentes reutilizables. Se presta especial atención a esta capa de la arquitectura, dado que es la que mayor funcionalidad reutilizable proporciona, y la que es de mayor interés para el cliente.

En el presente capítulo, para cada funcionalidad exhibida o componente en estudio, se acompañará a los mismos con el correspondiente diagrama de componentes, señalando con color naranja el componente en cuestión, para facilitar al lector posicionarse en un lugar puntual de la arquitectura.

La Figura 3 presenta el diagrama de componentes de la capa de presentación, con mayor nivel de detalle en la interacción entre formularios base, y las especializaciones desarrolladas para el presente ejemplo.

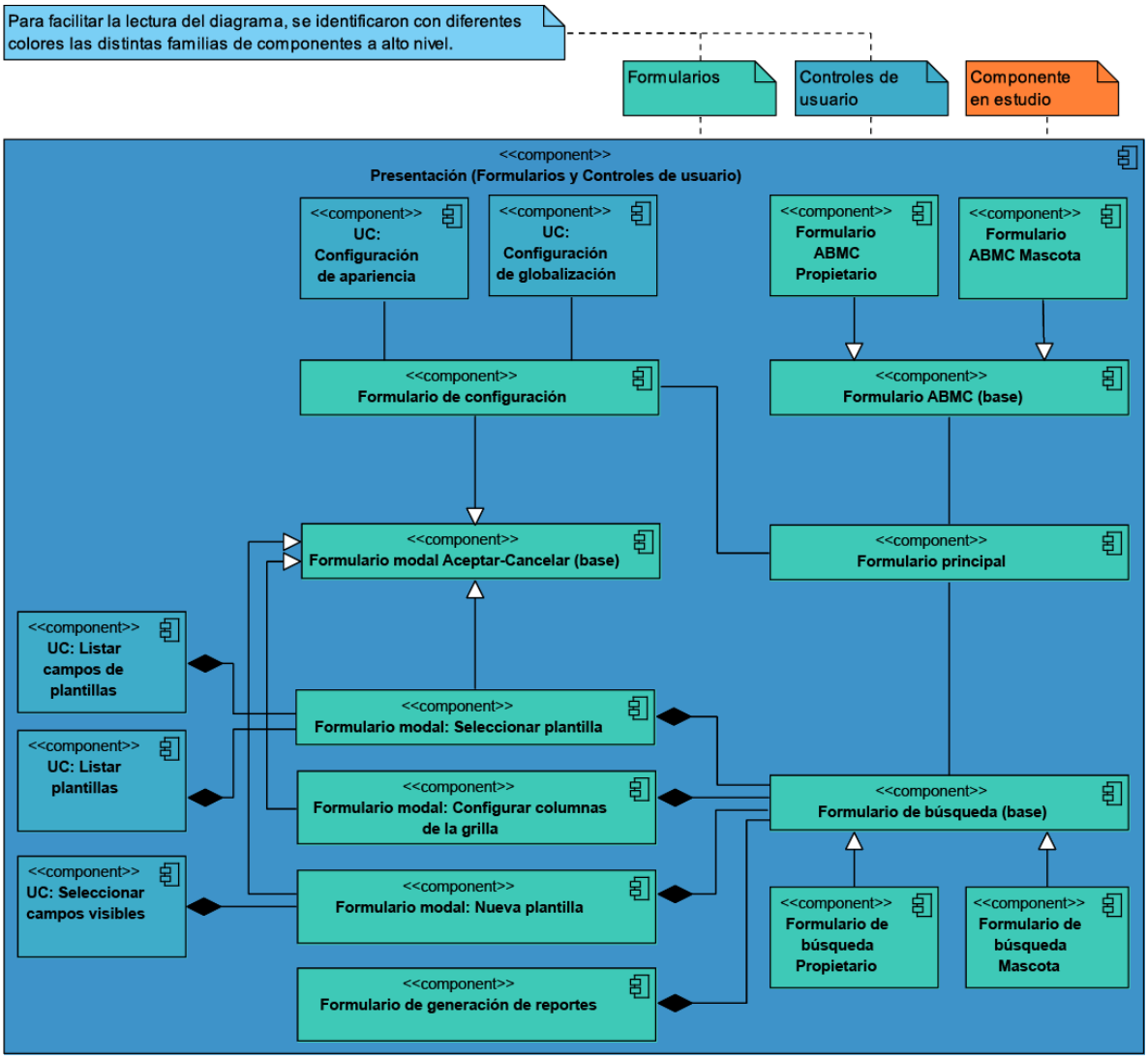


Figura 3

5.3 Formulario de Búsqueda

El cliente cuenta con numerosos formularios de búsqueda que comparten funcionalidades comunes, pero que son replicadas manualmente tras cada implementación.

Algunas funcionalidades comunes son, por ejemplo: la generación de reportes, la configuración de columnas (ocultas o visibles, orden, colores de fondo y fuente, etc.) a través de plantillas que permiten guardar las configuraciones y reutilizarlas según sea necesario, etc.

Siempre que se requiere implementar alguna nueva funcionalidad en los buscadores o corregir algún problema, esto implica no sólo actualizar código asociado a la entidad de datos que se muestra en el buscador, sino que, además, implica replicar dicha actualización al resto de los buscadores.

En contraposición, si la entidad de datos cambia (se agrega, quita, o actualiza una propiedad), esto implica actualizar código en el buscador, a raíz de los cambios generados en la entidad de datos.

Con el framework desarrollado, se consiguió abstraer completamente la lógica asociada a las funcionalidades del buscador, con la lógica correspondiente a la entidad de datos que se muestra en el buscador.

De esta manera, cada nueva funcionalidad estará inmediatamente disponible en todos los formularios de búsqueda al mismo tiempo, y, además, no implicará actualizar código asociado a las entidades que se muestran en el buscador.

Dado que las soluciones actuales del cliente están preparadas para trabajar en diferentes idiomas, y que la lógica de traducción se implementa mediante el uso de condicionales dentro del código fuente, esto añade complejidad extra al código dado.

Con el framework, *archivos de recursos* independientes al formulario permiten definir los diferentes idiomas que serán soportados, siendo estos completamente desacoplados de la lógica interna de cada formulario.

Esto posibilita que la actividad de traducción de la aplicación sea efectuada por personal específico para tal fin. Como característica adicional, el framework incorpora una herramienta que permite a los desarrolladores generar traducciones automáticas, de ser necesario.

En los escenarios de ejemplo, se pudo implementar los buscadores a través del framework con codificación muy puntual, que mayoritariamente respecta a la obtención de los datos de alguna capa de datos subyacente.

El formulario de búsqueda admite que se modifique su lienzo de trabajo, y se agreguen nuevos controles de usuario sólo en el área de los *filtros de* búsqueda para refinar los resultados, el resto de la estética del formulario no puede ser modificada.

El título del formulario se obtiene del archivo de recursos anteriormente mencionado, y las columnas del buscador se generan automáticamente mediante la definición de una clase desacoplada del formulario de búsqueda (un modelo de vista).

El *Formulario de Búsqueda (base)* se desarrolló en base a las propuestas efectuadas en el *Modelo conceptual de la aplicación*.

En la Figura 4 se exhibe el diagrama de componentes presentado en los *Componentes reutilizables de la capa de presentación*, destacando con color naranja el componente en estudio.

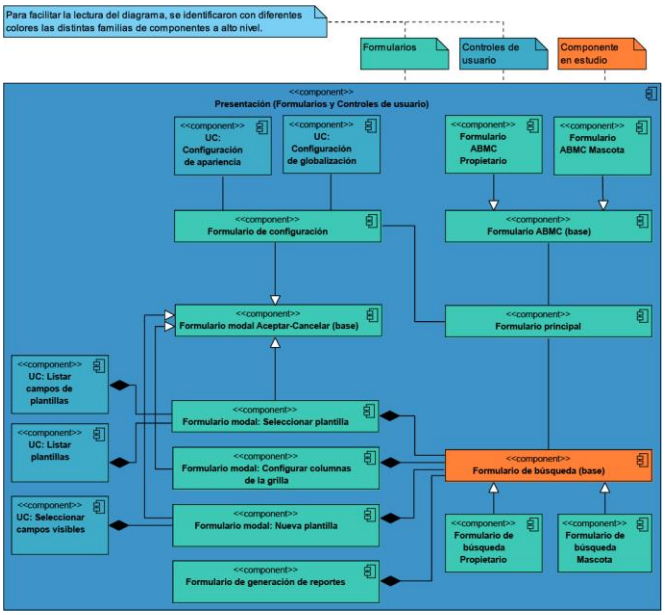


Figura 4

5.3.1 Especialización de Formularios de Búsqueda

El proceso de generación de un *Formulario de Búsqueda* consta de tres instancias fundamentales:

1) La generación de un modelo para la vista; una clase que represente las entidades que serán mostradas en la grilla del buscador (ViewModel).
2) La generación de un archivo de recursos que contenga los pares clave-valor con los elementos que deban ser traducidos.
3) Una clase que herede del *Formulario de Búsqueda* base, que implemente el método *Buscar(…)*.

En la Figura 5 se presenta un refinamiento del componente *Presentación* de la sección *Vista general de la arquitectura resultante*, en el cual se muestran las relaciones fundamentales entre los componentes que deben ser implementados acorde a las instrucciones previas. Para ejemplificar el proceso se utilizó la entidad *Mascota*.

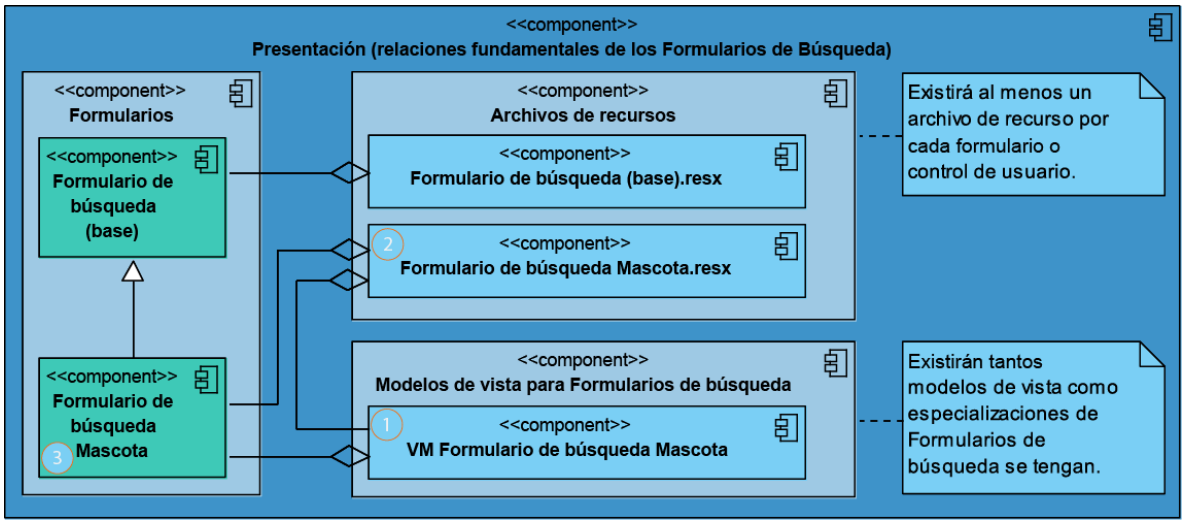


Figura 5

5.3.1.1 Modelo de vista

A continuación, en el ejemplo propuesto, se simplificó la definición de las propiedades de la clase para hacer énfasis en la lectura de los detalles importantes. Se ocultaron los campos de cada propiedad, y se colapsaron los descriptores de acceso (sentencias *get* y *set*).

Con color verde se destaca las referencias al nombre del archivo de recursos desde el cual se extraerán las cadenas para mostrar en las cabeceras de las columnas de la grilla, y en color naranja las claves:

```
public class VM_FrmBúsquedaMascota : VM_FrmBusquedaBase
{
    [Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_Nombre), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
    public string Nombre {…}

     [Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_FechaDeNacimiento), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
    public DateTime FechaDeNacimiento {…}

     [Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_TipoDeAnimal), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
    public string TipoDeAnimal {…}

     [Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_Notas), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
    public string Notas {…}

     [Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_Foto), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
    public Bitmap Foto {…}
}
```

5.3.1.2 Archivo de recursos

Para cada clave de referencia al archivo de recursos (identificadas en el código de ejemplo con color naranja) asignada en las propiedades del modelo de la vista, se le corresponde una clave en el archivo de recursos.

Una clave extra con nombre *Título*, permite definir el título que se mostrará en el formulario. La Figura 6 presenta la definición del archivo de recursos para el *Formulario de Búsqueda* de la entidad *Mascota*.

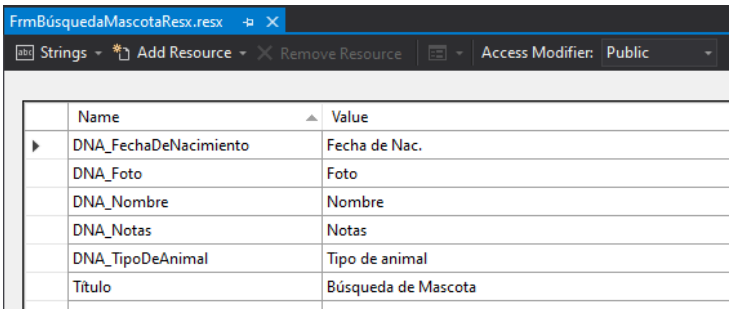


Figura 6

Las traducciones se generan replicando el archivo de recursos, y agregando al final del nombre del archivo, el nombre del idioma correspondiente.

La Figura 7 presenta el resultado de la traducción del archivo de recursos para el *Formulario de Búsqueda* de la entidad *Mascota*, en el idioma *en-US*.

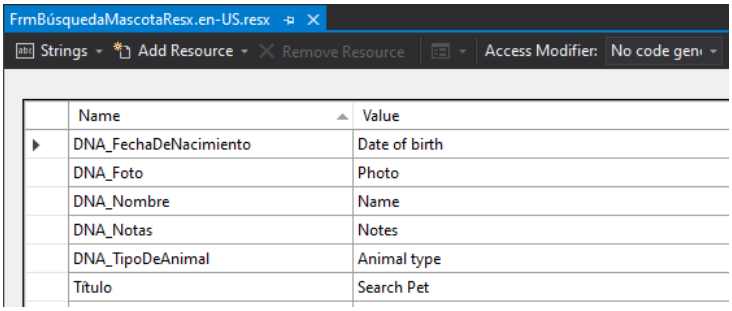


Figura 7

5.3.1.3 Especialización de la clase

En la especialización de la clase se deben referenciar el nombre del modelo de vista, el nombre del archivo de recursos, y se debe implementar el método *Buscar(…)*.

El método Buscar, obtiene los datos desde algún origen de datos (en este caso, de la capa subyacente, *Controladores*), y genera el traspaso de los datos al modelo de la vista.

Con color verde se destaca las referencias al nombre del archivo de recursos desde el cual se extraerán las cadenas para mostrar en las cabeceras de las columnas de la grilla, y en color naranja las claves:

```
public partial class FrmBúsquedaMascota : FrmBúsquedaBase
{
    public FrmBúsquedaMascota()
    {
        // Referencia de ViewModel
        base.TipoDeDatoDelVM = typeof(VM_FrmBúsquedaMascota);
        // Referencia de Archivo de Recursos
```

```
        this.resxOwn = new ComponentResourceManager(typeof(FrmBúsquedaMascotaResx));
    }

    protected override void Buscar(IBindingList listaDeEntidadesGenerica, string[] columnasACargar)
    {
        // Obtención de entidades desde capa subyacente
        List<Mascota> listEntidades = CT_Mascota.ObtenerTodas();
        // Generación de lista vacía para ViewModels
        BindingList<VM_FrmBúsquedaMascota> listVM =
(BindingList<VM_FrmBúsquedaMascota>)listaDeEntidadesGenerica;
        // Transformación de entidades de datos en ViewModels
        foreach (Mascota entidadOrigen in listEntidades)
        {
            VM_FrmBúsquedaMascota vm = new VM_FrmBúsquedaMascota();
            this.LlenarVM(vm, entidadOrigen, columnasACargar);
            listVM.Add(vm);
        }
    }
}
```

5.4 Formulario de Altas, Modificaciones y Consultas

El cliente cuenta con numerosos formularios de altas, modificaciones y consultas, que han sido codificados por diferentes desarrolladores, en diferentes momentos. No hay un proceso común o estandarizado para la codificación de los mismos, y siempre que es necesario actualizar alguno de estos formularios por alguna razón, implica el estudio particular de la implementación que deba ser actualizada.

Con el framework desarrollado, se brinda una secuencia de pasos para cada operación de persistencia (alta, modificación o consulta), en los cuales se debe programar sólo el código específico para esa instancia.

Por ejemplo, para el caso de un alta, en el momento que usuario presione el botón *Guardar*, se verificará que todos los campos necesarios estén completos, luego que los campos tengan contenido válido, luego que no haya colisión con alguna entidad que tenga los mismos datos, y finalmente se la guardará. Cada uno de estos pasos, es un método que debe ser especializado.

En formulario de base tiene la lógica que permite guiar la secuencia de pasos para cada situación de persistencia, con las validaciones pertinentes, y en caso de ser necesario, los correspondientes mensajes o salidas en pantalla para el usuario.

Dado que en los procesos de altas o modificaciones se comparten muchos de los elementos de la lógica, como por ejemplo las validaciones, varios de los métodos especializados por el desarrollador tendrán presencia en ambos procesos, y de esta manera se evita la repetición de código específico.

Un ejemplo de la situación anterior, sería, por ejemplo, el control de que todos los campos necesarios estén completos. Es muy común que, tanto en una situación de alta, como en una de modificación, la lógica mencionada no varíe.

Al igual que para los formularios de búsqueda, un archivo de recursos permite la traducción de los formularios de altas, modificaciones y consultas, de forma desacoplada a la lógica del formulario.

La implementación de este tipo de formularios, implica adecuar el lienzo o plantilla de trabajo con los controles de usuarios que sean acordes a la necesidad, implementar los métodos heredados, y generar el archivo de recursos para la traducción.

El formulario de altas, modificaciones y consultas admite que se modifique su lienzo de trabajo y se agreguen nuevos controles de usuario sólo en el área que se encuentra por debajo del título del formulario, el resto de la estética del formulario no puede ser modificada dado que es parte del formulario base.

El *Formulario de Altas, Modificaciones y Consultas (base)* se desarrolló en base a las propuestas efectuadas en el *Modelo conceptual de la aplicación*.

En la Figura 8 se exhiben los *Componentes reutilizables de la capa de presentación*, destacando con color naranja el componente en estudio.

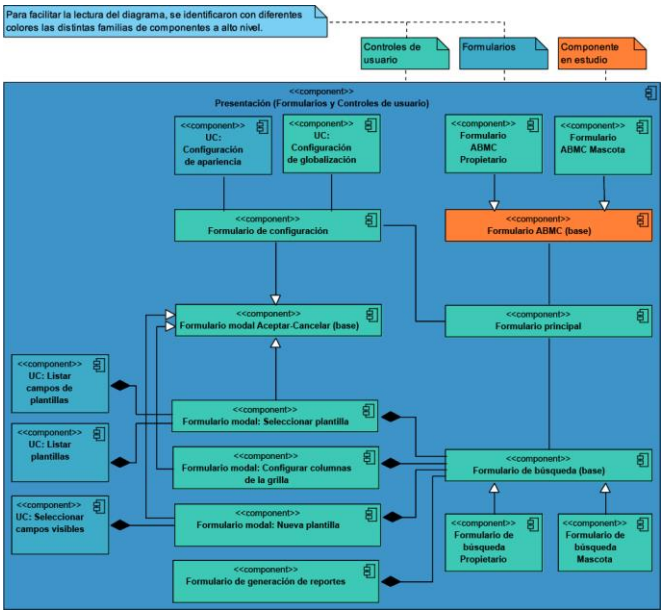


Figura 8

5.4.1 Especialización de Formularios de Altas, Modificaciones y Consultas

El proceso de generación de un *Formulario de Altas, Modificaciones y Consultas* consta de tres instancias fundamentales:

1) La generación de un archivo de recursos que contenga los pares clave-valor con los elementos que deban ser traducidos.
2) Una clase que herede del *Formulario de Altas, Modificaciones y Consultas* base, que implemente los métodos de verificación *EstanTodosLosDatosRequeridos()*, *EstanTodosLosDatosValidos()*, *HayColision()*, y *HayCambios()*, y los métodos de persistencia *PersistenciaCrearEntidad()* y *PersistenciaModificarEntidad()*.
3) Luego de especializar la clase base, automáticamente se genera la plantilla de trabajo, en la cual se deben agregar los controles de usuarios respectivos a la lógica a implementar.

En la Figura 9 se presenta un refinamiento del componente *Presentación* de la sección *Vista general de la arquitectura resultante*, en el cual se muestran las relaciones fundamentales entre los componentes que deben ser implementados acorde a las instrucciones previas. Para ejemplificar el proceso se utilizará la entidad *Mascota*.

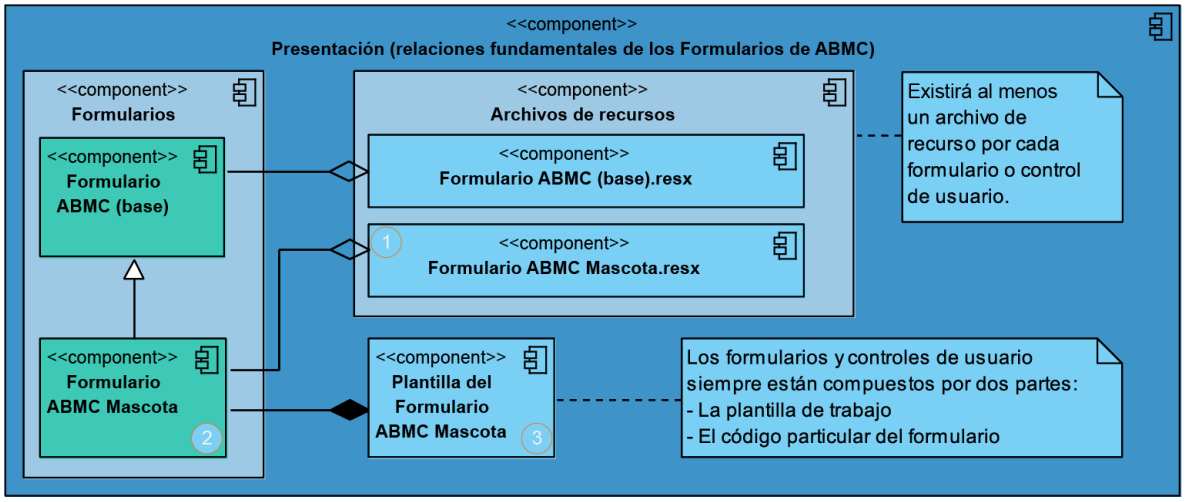


Figura 9

5.4.1.1 Archivo de recursos

Para cada situación dentro del proceso del *Formulario de Altas, Modificaciones y Consultas* en el que se desee contar con una cadena que pueda ser traducida, se puede agregar una referencia a una clave del archivo de recursos.

En el siguiente ejemplo de archivo de recursos se utiliza como prefijo 'Requerido', para identificar todas las cadenas con mensajes de error empleadas en la implementación del método *EstanTodosLosDatosRequeridos()*, el cual será especificado en el segundo elemento de la secuencia, en la *Especialización de la clase*.

Por convención, para definir el título que se mostrará en el formulario se emplearán siempre tres claves con los mismos nombres para las situaciones de Altas, Modificaciones y Consultas (TítuloEnConsulta, TítuloEnCrear, y TítuloEnModificar).

La Figura 10 presenta la definición del archivo de recursos para el *Formulario ABMC* de la entidad *Mascota*.

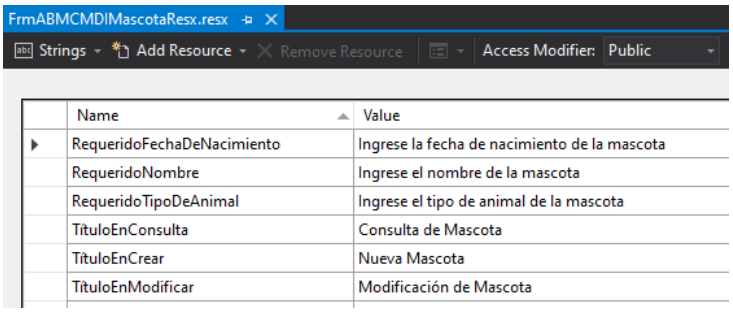


Figura 10

Las traducciones se generan replicando el archivo de recursos, agregando al final del nombre del archivo, el nombre del idioma correspondiente. En el ejemplo siguiente, *en-US*.

La Figura 11 presenta el resultado de la traducción del archivo de recursos para el *Formulario ABMC* de la entidad *Mascota*, en el idioma *en-US*.

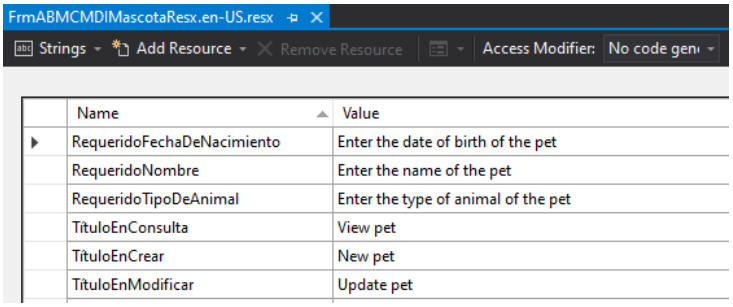


Figura 11

5.4.1.2 Especialización de la clase

La especialización de la clase se consigue heredando de la clase base (FrmABMCBase) e implementando los métodos de verificación EstanTodosLosDatosRequeridos(), EstanTodosLosDatosValidos(), HayColision(), y HayCambios(), y de los métodos de persistencia PersistenciaCrearEntidad() y PersistenciaModificarEntidad().

No es mandatoria la implementación de todos los métodos. Dado que la clase base es una clase abstracta[19], los métodos que deben ser sobrescritos tienen ya un comportamiento definido en la clase base.

En la clase base, todos los métodos de verificación carecen de lógica particular, y simplemente retornan un booleano acorde para no interrumpir el flujo de ejecución. EstanTodosLosDatosRequeridos() por defecto retorna verdadero, HayColision() por defecto retorna falso, etc.

[19] El propósito de una clase abstracta es proporcionar una definición común de una clase base que pueden compartir múltiples clases derivadas. No se puede crear una instancia de una clase abstracta. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members

De esta manera, podrían implementarse sólo los métodos necesarios para lograr el comportamiento deseado. Supongamos que es necesario un formulario que sólo permita el alta de alguna entidad y no su modificación, entonces no sería necesaria la implementación de los métodos HayCambios() y PersistenciaModificarEntidad().

Para ejemplificar el proceso de la implementación de uno de los métodos mencionados, se sobrescribirá el método *EstanTodosLosDatosRequeridos()*, que es uno de los métodos de verificación, el cual retorna verdadero o falso para detectar si todos los campos obligatorios en el formulario han sido completados.

En la entidad de datos *Mascota*, el nombre, fecha de nacimiento y tipo de animal son obligatorios. En los casos de error, se mostrarán los mensajes acordes consumiendo las cadenas del archivo de recursos.

Los desarrolladores tienen total libertad en el desarrollo de la lógica interna de los métodos, y sólo es necesario respetar la firma del método. En el siguiente ejemplo se tomará ventaja de esta libertad, añadiendo también mensajes de error junto a los controles de usuario correspondientes a las alertas. Los mensajes de error son leídos del archivo de recursos definido previamente, y mostrados en pantalla mediante un componente particular con nombre *ErrorProvider*.

Con color verde se destacan las referencias al nombre del archivo de recursos desde el cual se extraerán las cadenas para mostrar en las cabeceras de las columnas de la grilla, y en color naranja las claves:

```
protected override bool EstanTodosLosDatosRequeridos()
{
    bool blnRetorno = true;
    VMÚnico vm = this.ObtenerViewModel();

    // Nombre
    if (vm.Nombre == null)
    {
        this.dxErrorProvider.SetError(this.teNombre,
resxOwn.GetString(nameof(FrmABMCMDIMascotaResx.RequeridoNombre)));
        blnRetorno = false;
    }
    else
    {
        this.dxErrorProvider.SetError(this.teNombre, "");
    }

    // FechaDeNacimiento
    if (vm.FechaDeNacimiento == null)
    {
        this.dxErrorProvider.SetError(this.deFechaDeNacimiento,
resxOwn.GetString(nameof(FrmABMCMDIMascotaResx.RequeridoFechaDeNacimiento)));
        blnRetorno = false;
    }
    else
    {
        this.dxErrorProvider.SetError(this.deFechaDeNacimiento, "");
    }

    // TipoDeAnimalId
    if (vm.TipoDeAnimalId == null)
    {
        this.dxErrorProvider.SetError(this.slueTipoDeAnimal,
resxOwn.GetString(nameof(FrmABMCMDIMascotaResx.RequeridoTipoDeAnimal)));
        blnRetorno = false;
```

```
    }
    else
    {
        this.dxErrorProvider.SetError(this.slueTipoDeAnimal, "");
    }

    return blnRetorno;
}
```

5.4.1.3 Plantilla con controles de usuario

En la plantilla se disponen los controles de usuario en el orden conveniente, y se les asigna un nombre. El nombre de los controles de usuario permite acceder a sus propiedades en tiempo de ejecución (son objetos, cuyas instancias en tiempo de ejecución tienen el nombre asignado a los elementos en la plantilla).

En la especialización de un *Formulario de Altas, Modificaciones y Consultas* solo pueden agregarse controles de usuario en el área de trabajo (debajo del título en color verde), el resto de los componentes y funcionalidades que se observan son heredados de la clase base, y no se pueden modificar.

La Figura 12 presenta el *Formulario de Altas, Bajas, Modificaciones y Consultas* de la entidad *Mascota* en tiempo de diseño.

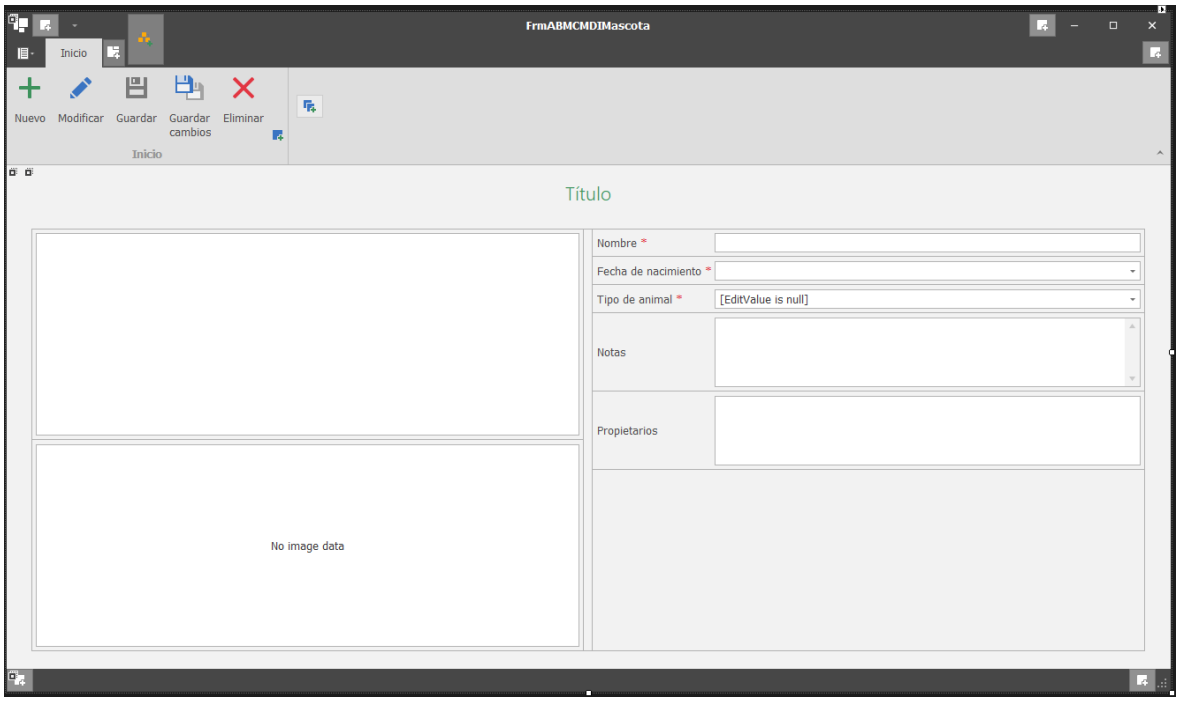


Figura 12

5.5 Formulario de configuración

El cliente cuenta con numerosos formularios de configuración, que han sido codificados por diferentes desarrolladores, empleando diferentes esquemas tanto en el aspecto visual, como en la lógica del desarrollo.

En los desarrollos del cliente no hay un proceso común o estandarizado para la codificación de los mismos, tampoco hay una plantilla gráfica que guíe el diseño, ni un punto en particular en la aplicación desde el cual pueda accederse a todos los elementos de configuración.

El *Formulario de Configuración* es un formulario modal, al cual se puede acceder desde el menú principal de la aplicación, y concentra todos los elementos de configuración de la aplicación en un único lugar.

El formulario se organiza en dos secciones verticales. A la izquierda un listado permite seleccionar la característica que se desea configurar, y a la derecha se muestran las opciones que pueden configurarse respecto a la característica seleccionada.

Se pueden agregar elementos al *Formulario de Configuración* -a los cuales llamamos "ítems de configuración"- siguiendo reglas sencillas, que a grandes rasgos consisten en generar un control de usuario que implementa una interfaz lógica, y asignarle un nombre para ser mostrado.

Los nombres para ser mostrados se utilizan en el panel de la izquierda, donde se muestran los distintos aspectos que pueden ser configurados. Los controles de usuarios se muestran a la derecha, en el área de uso general, y responden al elemento seleccionado en el panel de la izquierda.

Cada *ítem de configuración* es cargado de forma dinámica, lo que implica que el panel de configuración asociado al elemento seleccionado en el menú de la izquierda, no será cargado hasta que no sea utilizado.

El listado de la izquierda cuenta con un buscador que permite filtrar los distintos *ítems de configuración* por palabras clave, facilitando la búsqueda cuando la cantidad de *ítems de configuración* sea numerosa.

Los ítems de configuración en el panel de la izquierda pueden ser anidados mediante una estructura de padres-hijos. La funcionalidad es especialmente útil cuando, por ejemplo, se generan agrupaciones por características particulares, para un mismo sector de la aplicación.

Cada panel de configuración (mostrado a la derecha) es independiente del *Formulario de Configuración* y viceversa, por lo cual, es posible generar modificaciones en el *Formulario de Configuración* o en los paneles de configuración sin afectar al resto de los componentes.

La estética del formulario, como así también su lógica, se inspiró en los formularios de configuración habitualmente utilizados en las aplicaciones de Microsoft, como el paquete de Office, por ejemplo.

El *Formulario de configuración* es en sí mismo, es la especialización de otro elemento genérico del framework, llamado *Formulario modal Aceptar-Cancelar*.

Al igual que en el resto de los elementos genéricos del framework, la traducción de todos los elementos se efectúa mediante archivos de recursos desacoplados de los componentes.

El *Formulario de configuración* se desarrolló en base a las propuestas efectuadas en el *Modelo conceptual de la aplicación*.

En la Figura 13 se exhiben los *Componentes reutilizables de la capa de presentación*, destacando con color naranja el componente en estudio.

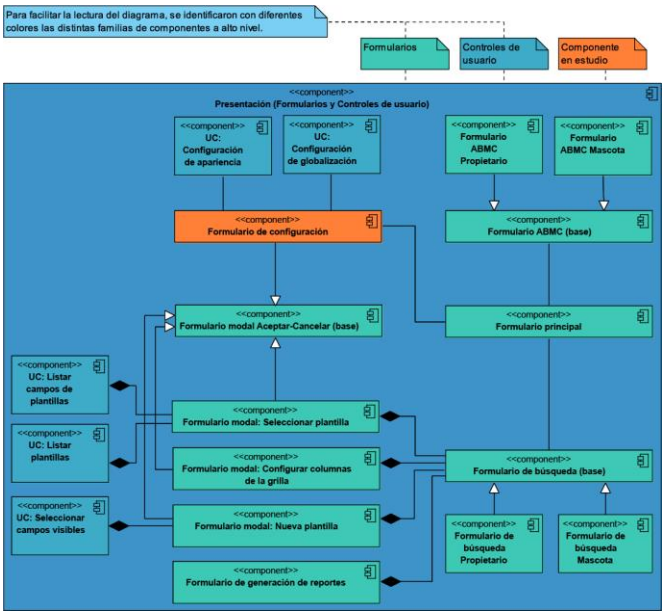


Figura 13

5.5.1 Agregado de ítems de configuración al Formulario de configuración

El proceso de agregado de *ítems de configuración* al *Formulario de configuración* consta de dos instancias fundamentales:

1) Desarrollar un control de usuario (una clase específica provista por *Windows Forms*) que herede e implemente la interfaz *IUserControlDeConfiguracion* (será el control de usuario mostrado en el panel de la derecha en el *Formulario de configuración*)
2) Referenciar el control de usuario dentro del *Formulario de configuración*, asignándole un nombre para mostrar.

En la Figura 14 se presenta un diagrama de clases en el cual se muestra la relación entre el *Formulario de configuración* y la interfaz *IUserControlDeConfiguracion*, y el *Formulario de configuración* y las especializaciones de la interfaz *IUserControlDeConfiguracion*. Para ejemplificar el proceso de especialización de la interfaz *IUserControlDeConfiguracion* se utilizó el ítem de configuración *UC Configuración de apariencia*.

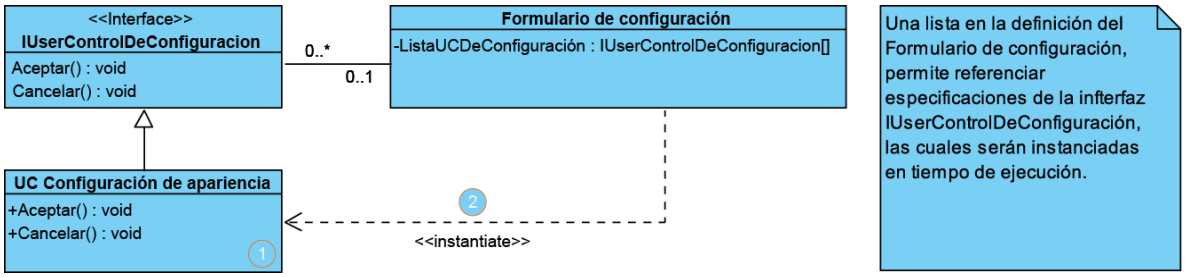


Figura 14

Visto desde una perspectiva más general, en la Figura 15 se exhibe el componente tomado como ejemplo, desde la perspectiva de los *Componentes reutilizables de la capa de presentación*, destacando el componente con color naranja.

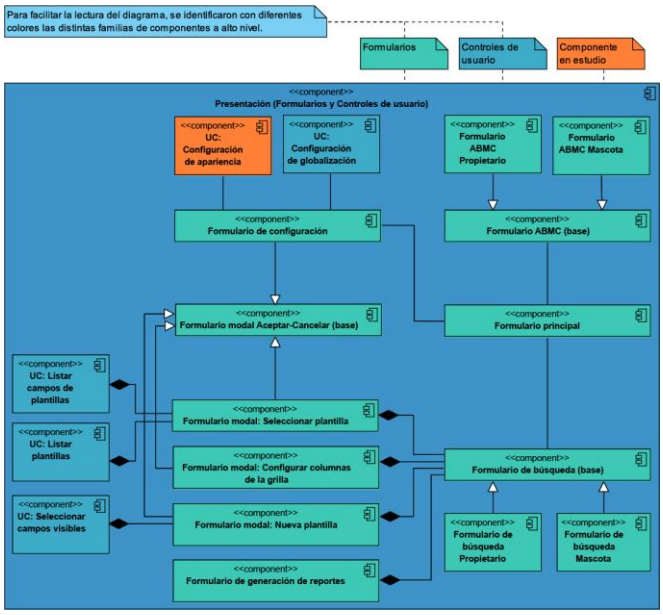


Figura 15

5.5.1.1 Desarrollo de control de usuario para configuración

Los controles de usuario para el *Formulario de configuración*, son controles de usuario que implementan la interfaz *IUserControlDeConfiguracion*.

La interfaz *IUserControlDeConfiguracion* establece un contrato en el cual deben implementarse los métodos Aceptar() y Cancelar().

El método Aceptar() ejecuta la lógica relevante para el ítem de configuración al momento de presionar botón 'Aceptar' en el *Formulario de configuración*, con el cual se aplican los cambios configurados y se cierra el formulario.

De forma análoga, el método Cancelar(), es invocado por el Formulario de configuración cuando se presiona el botón "Cancelar()".

El método Cancelar() es relevante, dado que muchos cambios se aplican inmediatamente, por ejemplo, las configuraciones gráficas que ofrecen previsualizar la configuración. De esta manera, todos los cambios aplicados si el *Formulario de configuración* se finaliza con el botón "Cancelar".

A modo de ejemplo, se presenta el código resultante en la implementación del control de usuario de Configuración de apariencia. Se presenta sólo la estructura de la clase con sus métodos, y se omite el código de las tareas específicas, dado que el objeto de este ejemplo es brindar una idea de una implementación genérica:

```
public class ConfiguraciónApariencia : XtraUserControl, IUserControlDeConfiguracion
{
    public ConfiguraciónApariencia()
    {
        …
    }

    public void Aceptar()
    {
        …
    }

    public void Cancelar()
    {
        …
    }
}
```

5.5.1.2 Referencia del ítem de configuración en el Formulario de configuración

Para agregar un control de usuario al *Formulario de configuración*, es necesario referenciar el control de usuario, asignarle un nombre para mostrar, y otorgarle una posición en el listado (panel en la izquierda del formulario).

Las referencias se realizan a través del tipo de dato del control de usuario, lo cual permite que el Formulario de configuración instancie los paneles de configuración sólo cuando los mismos van a ser utilizados.

Para referenciar un panel de configuración, se debe localizar el método AdjuntarGestoresDeConfiguración() en la definición de la clase del *Formulario de configuración*, y añadir la referencia al control de usuario.

A continuación se presenta el estado actual del método AdjuntarGestoresDeConfiguración(), en el cual se han referenciado los paneles de configuración de *Idioma* y *Apariencia*.

```
private void AdjuntarGestoresDeConfiguración()
{
    items.Add(new ItemConfiguracion(0, -1, "Apariencia", typeof(ConfiguraciónApariencia)));
    items.Add(new ItemConfiguracion(1, -1, "Idioma", typeof(ConfiguraciónGlobalización)));
}
```

Tomando como ejemplo la implementación anterior, el primer elemento en la definición de cada ítem de configuración define el orden en el listado, y actúa como identificador de cada nodo (0 para *Apariencia*, 1 para *Idioma*).

El segundo elemento en la definición estable el padre del nodo, siendo -1 la raíz del listado (sin nodo padre), y cualquier valor mayor o igual a 0 la referencia a otro nodo.

El tercer elemento define el nombre con el cual será mostrado el ítem de configuración en el menú de la izquierda, y el cuarto, es el tipo de datos del control de usuario de configuración.

5.6 Gestión de apertura y cierre de formularios

Para abrir y cerrar formularios tanto de búsqueda como de altas, modificaciones y consultas, se utiliza el *Gestor de Formularios*.

El *Gestor de Formularios* es una clase estática que tiene el rol de arbitrar diferentes situaciones que pueden darse con este tipo de formularios, y es una más de las herramientas genéricas provistas por el framework.

Por ejemplo, si ya se ha abierto un *Formulario de Búsqueda* para la entidad *Mascota*, y se intenta volver a abrir dicho formulario, nos interesa que la aplicación automáticamente nos posicione sobre el formulario abierto, en vez de crear un nuevo.

Para los casos de formularios ABMC, pueden darse una combinación de situaciones más amplia, dado que las situaciones de apertura en torno a una misma entidad pueden ser de tres tipos: *Alta*, *Modificaciones* y *Consultas*. A continuación, se resumen las posturas adoptadas para casa situación.

- Solicitud de apertura en modo Consulta:

- Si ya hay un formulario del mismo tipo en verbo "Consultar", para la misma entidad, poner el foco en él.
- Si no, si hay un formulario del mismo tipo en verbo "Modificar", para la misma entidad, poner el foco en él notificando que no se puede "Consultar" cuando se está modificando.
- Si no, crear un nuevo formulario en verbo "Consultar" y abrirlo.

- Solicitud de apertura en modo Crear:

- Si ya hay un formulario del mismo tipo en verbo "Crear", para la misma entidad, poner el foco en él.
- Si no, crear un nuevo formulario en verbo "Crear" y abrirlo.

- Solicitud de apertura en modo Modificar:

- Si hay un formulario del mismo tipo en verbo "Consultar", pasarlo a verbo "Modificar" y poner el foco en él.
- Si ya hay un formulario del mismo tipo en verbo "Modificar", poner el foco en él.
- Si no, crear un nuevo formulario en verbo "Modificar" y abrirlo.

Un aspecto favorable de esta esta estrategia es que encapsula la lógica de instanciamiento y desechado de formularios de tipo conocido (Búsqueda y ABMC), dado que será *el Gestor de Formularios* quien efectúe esas operaciones, el desarrollador simplemente las solicitará.

En la Figura 16 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio.

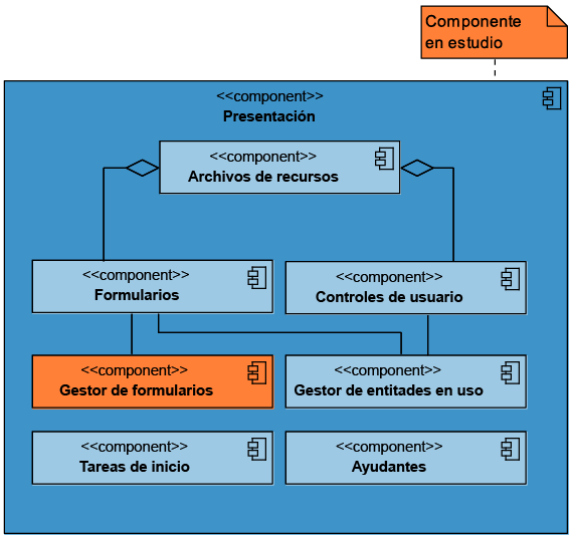


Figura 16

5.6.1 Apertura de formularios

La apertura de *formularios de búsqueda* se realiza mediante el método AbrirFrmBusqueda(…) empleando como parámetro de la función, el tipo de dato del formulario que se desea abrir.

Para el caso de los formularios ABMC el proceso es similar, se debe utilizar el método AbrirFrmABMC(…), y utilizar como parámetros el tipo de datos del formulario que se desea abrir, y además, el verbo (Consultar, Crear o Modificar) con el que será inicializado el formulario.

Para las situaciones de Consultas o Modificaciones, además es necesario proveer el Id de la entidad que se quiere Consultar o Modificar, y es opcional proveer el objeto (instancia) de dicha entidad.

Como ejemplo del uso del *Gestor de Formulario* para apreturas, se presenta el código utilizado en situaciones dadas con el escenario de ejemplo, contrastado con su resulta desde la perspectiva de un usuario.

5.6.1.1 Menú lateral

En el menú lateral se proporciona acceso a los buscadores tanto de *Propietarios* como de *Mascotas*, también se proporciona acceso a los formularios ABMC de dichas entidades para la creación rápida de nuevas entidades, por lo cual se emplea el verbo *Crear*.

En la Figura 17 se exhibe una captura de pantalla del menú lateral del sistema en ejecución. Cada uno de los botones internamente realiza una llama al *Gestor de formularios*, indicando el tipo de formulario que requiere.

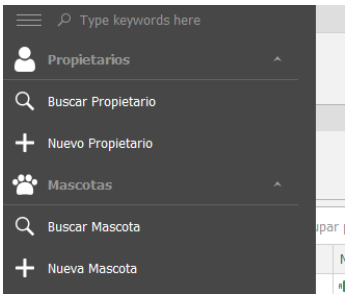


Figura 17

A continuación, se cita el código ejecutado por cada botón del menú lateral del sistema, en el orden en el que son presentados en tiempo de ejecución:

```
GestorDeFormularios.AbrirFrmBusqueda(typeof(FrmBúsquedaPropietario));

GestorDeFormularios.AbrirFrmABMC(typeof(FrmABMCPropietario), Verbo.Crear);

GestorDeFormularios.AbrirFrmBusqueda(typeof(FrmBúsquedaMascota));

GestorDeFormularios.AbrirFrmABMC(typeof(FrmABMCMascota), Verbo.Crear);
```

5.6.1.2 Comandos del menú ribbón

Cunado desde el buscador se selecciona alguna de las entidades resultantes del proceso de búsqueda, se habilitan los botones “Modificar” y “Consultar” en el menú Ribbon (encuadrados en color naranja).

El algoritmo detrás de esos comandos emplea llamadas al método AbrirFrmABMC(…), con el verbo *Modificar* o *Consultar* según corresponde, y el Id de la entidad seleccionada.

En la Figura 18 se exhibe una captura de pantalla del sistema en ejecución, con foco en el *Formulario de Búsqueda* para la entidad *Propietario*, en la cual se ha seleccionado un registro de la grilla, y como consecuencia se habilitó el uso de los botones ‘Modificar’ y ‘Consultar’ en el menú ribbon.

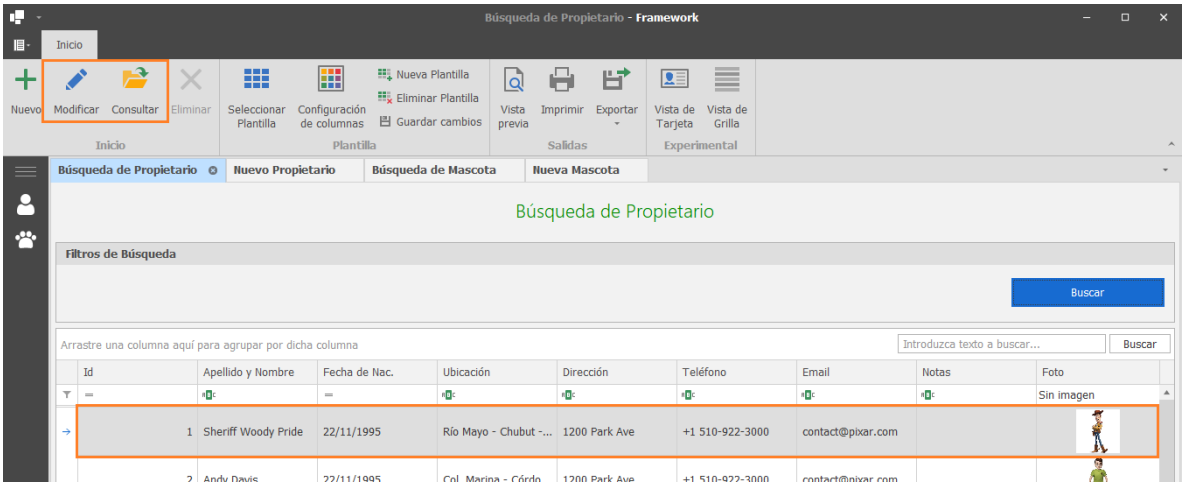


Figura 18

A continuación, se cita el código ejecutado por los botones 'Modificar' y 'Consultar' del menú ribbon:

```
GestorDeFormularios.AbrirFrmABMC(typeof(FrmABMCMDIPropietario), Verbo.Modificar,
((Propietario)entidad).Id, entidad);
```

```
GestorDeFormularios.AbrirFrmABMC(typeof(FrmABMCMDIPropietario), Verbo.Consultar,
((Propietario)entidad).Id, entidad);
```

5.6.2 Cierre de formularios

El cierre de los formularios es transparente al desarrollador, ya que la invocación al cierre está implementada en los formularios base.

Al momento del cierre, dependiendo del tipo de formulario -de búsqueda, o ABMC-, los formularios ejecutan un método del *Gestor de formularios* en el que explícitamente solicitan su propio cierre y liberación de recursos.

Los formularios de búsqueda ejecutan el método DesecharFrmBusqueda(this), DesecharFrmABMC(this). Como parámetro se emplea la palabra reserva *this*, que hace alusión a la misma instancia (objeto) que está invocando el método.

5.7 Gestor de entidades en uso

Similar al *Gestor de formularios*, el *Gestor de entidades en uso* es una clase estática que tiene el rol de registrar qué entidades de datos están siendo utilizadas en tiempo de ejecución, y por qué formulario en particular y es una más de las herramientas genéricas provistas por el framework.

Al momento de utilizar una entidad de datos, se cuenta con dos tipos de permisos (almacenados en un enumerado público en la aplicación), que brindan la posibilidad de registrar el uso de la misma en modo *Lectura*, o *Escritura*.

El *Gestor de entidades en uso* permite que una entidad de datos sea utilizada por un número indeterminado de formularios en modo *Lectura* al mismo tiempo, pero sólo un formulario puede utilizarla en modo *Escritura*.

Por ejemplo, cuando se abre un *Formulario de Búsqueda* para la entidad de datos *Mascota*, todas las instancias de la entidad de datos *Mascota* que son presentadas en la grilla como resultados de la búsqueda, están siendo utilizadas por dicho formulario en modo de *Lectura*.

Si alguna de las mascotas es seleccionada para ser modificada, entonces la entidad de datos pasa a estar en uso por una instancia del *Formulario ABMC*, en modo *Escritura*.

Para evitar que el uso de una misma entidad de datos con distintos fines lleve a una situación de inconsistencias en tiempo de ejecución, es que se registran que entidades son utilizadas, de qué modo, y por quién.

Internamente, el *Gestor de entidades en uso* mantiene una tabla en la que relaciona el tipo de datos de una entidad de datos, su Id, y una lista con los formularios que están utilizando la entidad y con qué tipo de permiso.

El *Gestor de formularios* presentado previamente se vale de los servicios del *Gestor de entidades en uso* para saber, por ejemplo, si un formulario de un determinado tipo, ya está utilizando la entidad con un determinado permiso.

El registro de una entidad en uso se efectúa mediante el método RegistrarEntidad(…), el cual como parámetros recibe el tipo de datos de la entidad de datos que se desea registrar, el tipo de permiso, el Id de la entidad, y el formulario que está efectúan la registración.

A continuación, se presenta a modo de ejemplo la llamada utilizada tras cada búsqueda de *Mascotas*, en un *Formulario de búsqueda*:

```
GestorDeEntidadesEnUso.RegistrarEntidad(
    typeof(Mascota),
    TipoDePermiso.Lectura,
    entidadOrigen.Id,
    this);
```

Los formularios de búsqueda tienen implementado en su lógica de base, que, tras cada nueva búsqueda se liberan las entidades que tenían registradas previamente de forma automática, pero la lógica puede ser implementada por un desarrollador a discreción.

Para liberar los recursos, se debe invocar el método LiberarRecursosDeFormulario(…), enviando como parámetro la instancia del formulario que efectúa la liberación.

```
GestorDeEntidadesEnUso.LiberarRecursosDeFormulario(this);
```

En la Figura 19 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio.

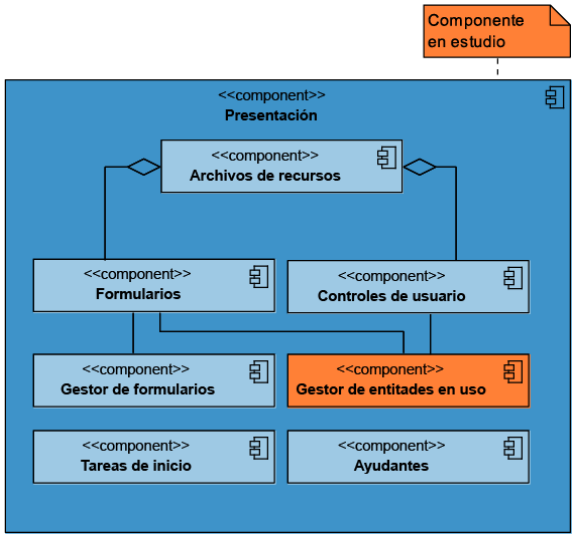


Figura 19

5.8 Herramienta para generar traducciones automáticas

Dado que el framework desarrollado utiliza archivos de recursos para desacoplar los elementos que definan la globalización (idioma y características regionales), se añadido a las herramientas que acompañan el framework un traductor automático de archivos de recursos.

La finalidad de la herramienta no es generar traducciones para ser liberadas a producción, sino contar con la posibilidad de generar traducciones rápidas, que podrían ser especialmente útiles en escenarios de prueba.

La herramienta propuesta brinda acceso centralizado a todos los archivos de recursos del sistema, y permite traducir a un gran número de idiomas utilizando los servicios de traductores online gratuitos (Google, Microsoft, Azure, etc.). La Figura 20 muestra la interfaz de selección de idioma, con todos los idiomas disponibles.

Figura 20

La herramienta[20] cumple con todas las premisas establecidas previamente para el trabajo con archivos de recursos, es de código abierto, y cuenta con una comunidad activa que le brinda soporte.

[20] Herramienta de traducción automática de *archivos de recursos*: https://github.com/dotnet/ResXResourceManager

En la Figura 21 se muestra la perspectiva de edición del archivo de los archivos de recursos generados para el *Formulario ABMC de Mascota*, a través de la herramienta de traducción automática.

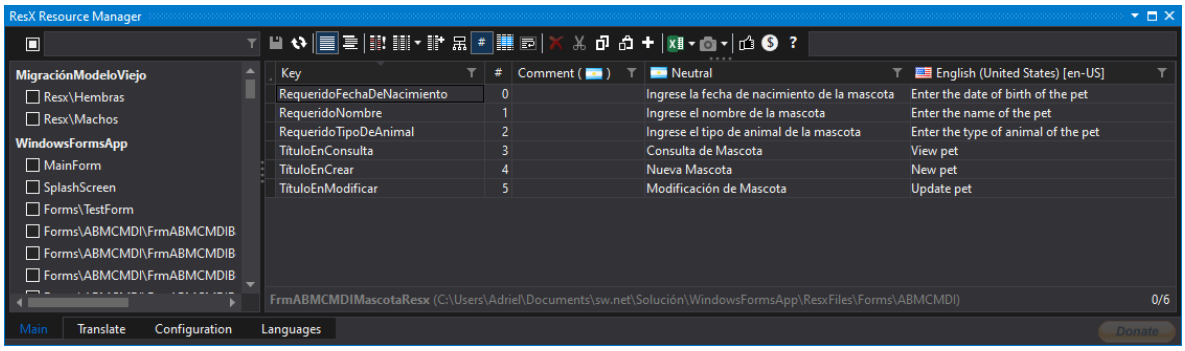


Figura 21

5.9 Pruebas unitarias

Como se mencionó en el capítulo *Desarrollo*, en el apartado de *Testing*, en el presente proyecto se efectuarían *pruebas unitarias*, dado que son estas las que tienen completo control por el desarrollador, ya que están apartadas de incumbencias de infraestructura (funcionamiento de sistema de archivos, base de datos, redes, etc.).

Continuando con la metodología planteada en este capítulo, lo que se pretende es brindar una perspectiva del uso de las herramientas, procurando el uso de ejemplos sencillos.

Para ejemplificar el proceso de generación y ejecución de un test unitario, se hará uso de un método particular de la capa de *Lógica de negocio*, el cual es invocado para insertar una nueva entidad del tipo *Mascota* en la base de datos.

En la *Arquitectura* del framework se definió que en la capa de *Lógica de negocio* se establecerían los componentes de tipo *Controladores*, en los cuales se dispondrían las clases particulares que actúen como proveedores de servicio para la capa de *Presentación*, mediando situaciones con la capa de *Acceso a datos*.

A continuación, se cita el método que será testeado. El resto de los métodos de la clase se ocultaron dado que enfocaremos las pruebas para este método en particular. En la definición se hace referencia al espacio de nombres *CapaDeNegocio*, en cual contiene la clase estática *CT_Mascota* (clase con rol de controlador), la cual contiene el método *InsertarNuevo*:

```
namespace CapaDeNegocio
{
    public static class CT_Mascota
    {
        public static bool InsertarNuevo(Mascota mascota)
        {
            // Datos requeridos
            if (string.IsNullOrEmpty(mascota.Nombre) ||
                mascota.FechaDeNacimiento == null)
            {
                return false;
            }
```

```
            // Datos válidos
            if (mascota.FechaDeNacimiento > DateTime.Now ||
                mascota.TipoDeAnimalId < 1)
            {
                return false;
            }

            // Agrego la Mascota a la DB
            using (Model model = new Model())
            {
                try
                {
                    model.Mascotas.Add(mascota);
                    model.SaveChanges();
                }
                catch (Exception)
                {
                    return false;
                }
            }

            return true;
        }
    }
}
```

Como se mencionó en el aparado de *Testing* en el capítulo anterior, y diversas formas de organizar las pruebas unitarias en un proyecto, y diferentes formas de nombrar a cada prueba unitaria en particular.

En este proyecto, las pruebas unitarias se encuentran en un ensamblado diferente, llamado *UnitTests*, y las pruebas se organizan en clases que llevan el mismo nombre que las clases para las cuales testean funcionalidad, siendo para el caso de este ejemplo *CT_Mascota* la clase testeada, por lo cual *CT_MascotaTest* es el nombre de la clase que contendrá sus métodos de prueba.

La convención de nombres para los métodos de pruebas unitarias, se definió según NombreDelMétodoQueSePrueba_SituaciónQueSePrueba_ResultadoEsperado.

Prestando atención al método de ejemplo *InsertarNuevo*, de la clase controladora *CT_Mascota*, se puede observar que hay ciertas situaciones que no permitirían que la entidad sea persistida: que la mascota no tenga nombre, que su fecha de nacimiento sea mayor a la fecha actual, y que tenga un *TipoDeAnimal* inexistente, entre otros.

Teniendo en cuenta que cuando la inserción de la entidad *Mascota* falla, el método *InsertarNuevo* retorna falso, para las tres situaciones planteadas previamente los nombres de los métodos de prueba unitaria resultarían:

- InsertarNuevo_SinNombre_RetornaFalso
- InsertarNuevo_FechaDeNacimientoFuturo_RetornaFalso
- InsertarNuevo_TipoDeAnimalInexistente_RetornaFalso

A continuación, se presenta el código resultante para las tres pruebas unitarias planteadas, siguiendo la convención de escritura de código *(arrange-act-assert)* para pruebas unitarias planteado en el aparatado de Testing. El nombre de los métodos se destaca con color verde:

```
namespace UnitTests
```

```
{
    public class CT_MascotaTest
    {
        [Fact]
        public void InsertarNuevo_SinNombre_RetornaFalso()
        {
            #region Arrange
            // Instancio la entidad de datos:
            Mascota mascota = new Mascota();

            // Configuro la entidad convenientemente:
            mascota.Nombre = null;

            // Resultado esperado en el test:
            bool resultadoEsperado = false;

            #endregion // Arrange


            #region Act

            bool resultadoObtenido = CT_Mascota.InsertarNuevo(mascota);

            #endregion // Act


            #region Assert

            Assert.Equal(resultadoEsperado, resultadoObtenido);

            #endregion // Assert
        }

        [Fact]
        public void InsertarNuevo_FechaDeNacimientoFuturo_RetornaFalso()
        {
            #region Arrange
            // Instancio la entidad de datos:
            Mascota mascota = new Mascota();

            // Configuro la entidad convenientemente:
            mascota.FechaDeNacimiento = DateTime.Now.AddDays(1);

            // Resultado esperado en el test:
            bool resultadoEsperado = false;

            #endregion // Arrange


            #region Act

            bool resultadoObtenido = CT_Mascota.InsertarNuevo(mascota);

            #endregion // Act


            #region Assert

            Assert.Equal(resultadoEsperado, resultadoObtenido);

            #endregion // Assert
        }

        [Fact]
        public void InsertarNuevo_TipoDeAnimalInexistente_RetornaFalso()
        {
            #region Arrange
            // Instancio la entidad de datos:
            Mascota mascota = new Mascota();
```

```
            // Configuro la entidad convenientemente:
            mascota.TipoDeAnimalId = 0;

            // Resultado esperado en el test:
            bool resultadoEsperado = false;

            #endregion // Arrange


            #region Act

            bool resultadoObtenido = CT_Mascota.InsertarNuevo(mascota);

            #endregion // Act


            #region Assert

            Assert.Equal(resultadoEsperado, resultadoObtenido);

            #endregion // Assert
        }
    }
}
```

Para ejecutar las pruebas unitarias y conocer sus resultados, el IDE utilizado en este proyecto -Visual Studio-, brinda un *explorador de pruebas*. La Figura 22 muestra el resultado de la ejecución de las pruebas unitarias citadas previamente, en el *explorador de pruebas*.

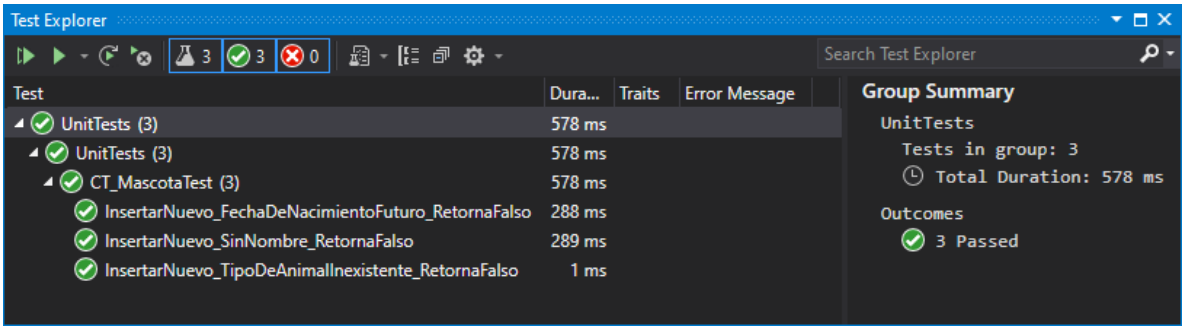


Figura 22

5.10 Pantallas de la aplicación en tiempo de ejecución

5.10.1 Formulario de búsqueda

5.10.1.1 Pantalla principal

En la Figura 23 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio. Luego, la Figura 24 exhibe una captura de pantalla del sistema en ejecución del componente en estudio.

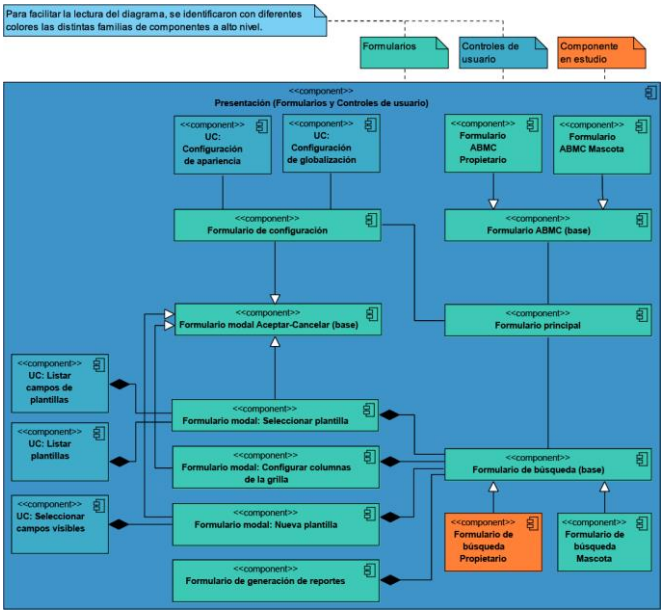


Figura 23

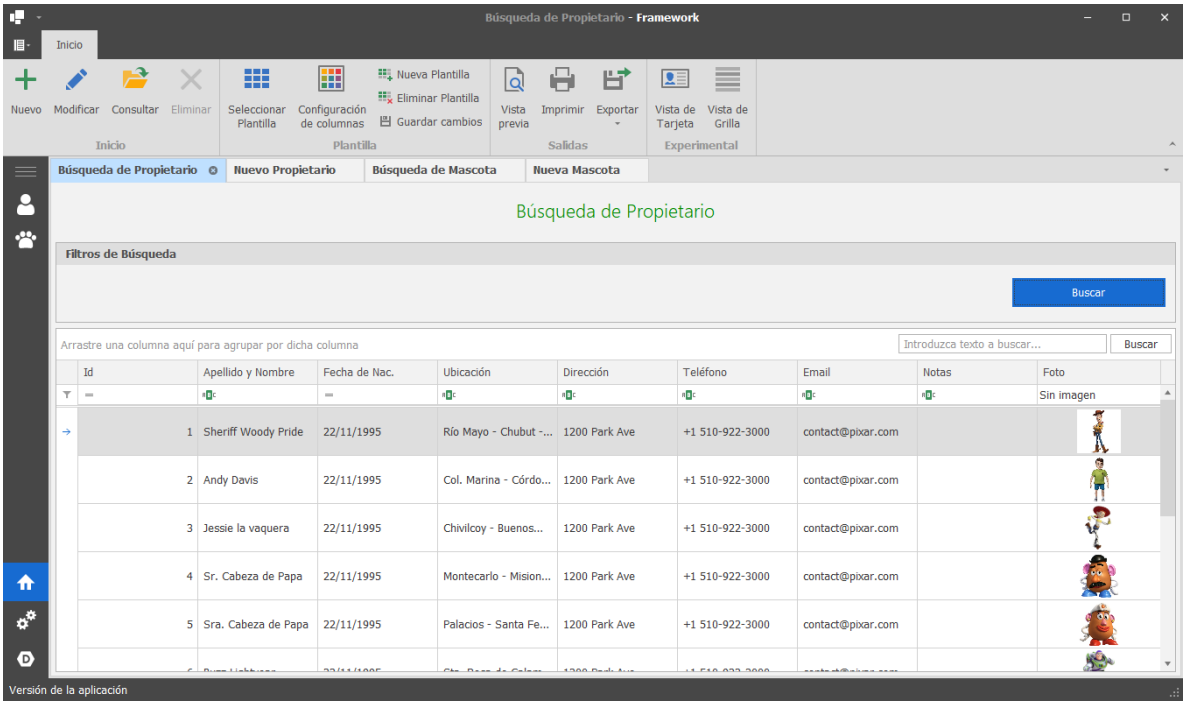


Figura 24

5.10.1.2 Nueva plantilla

En la Figura 25 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio. Luego, la Figura 26 exhibe una captura de pantalla del sistema en ejecución del componente en estudio.

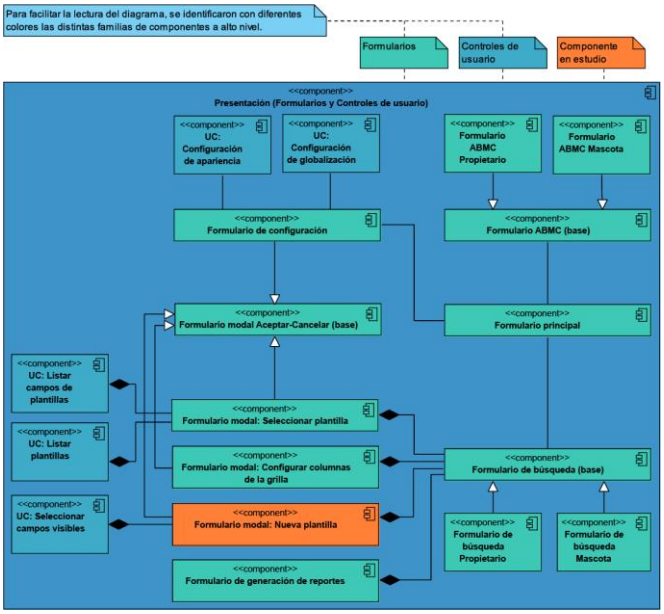


Figura 25

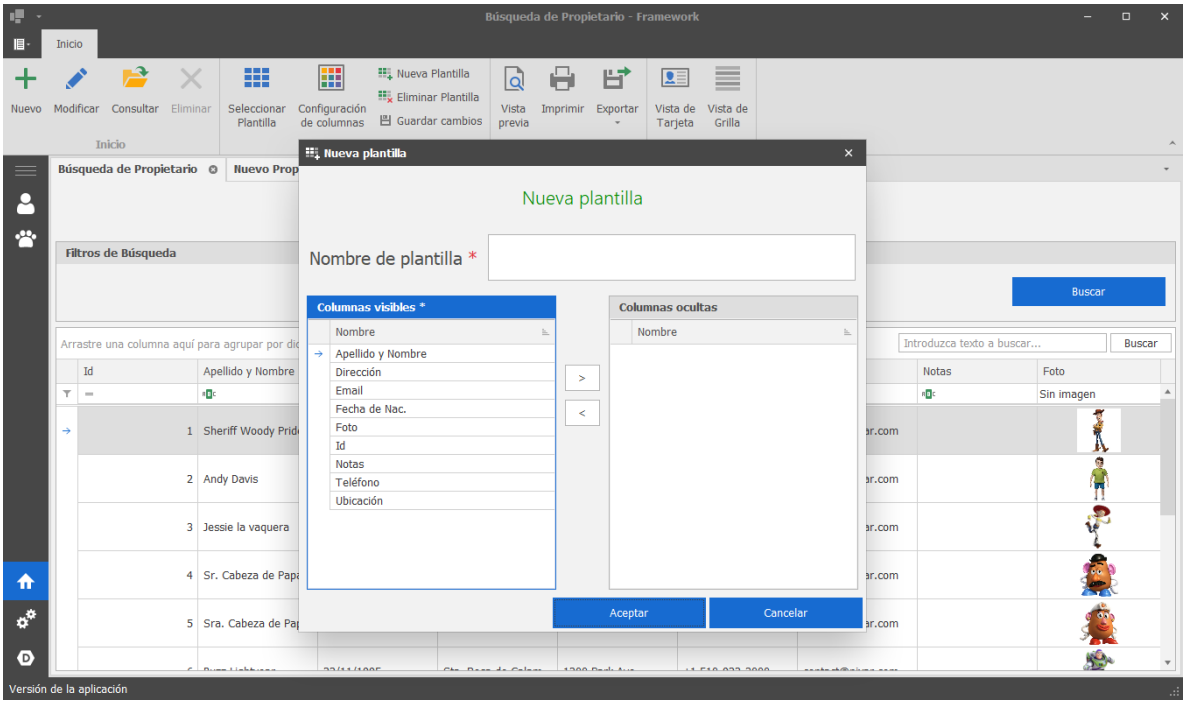


Figura 26

5.10.1.3 Selección de plantilla

En la Figura 27 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio. Luego, la Figura 28 exhibe una captura de pantalla del sistema en ejecución del componente en estudio.

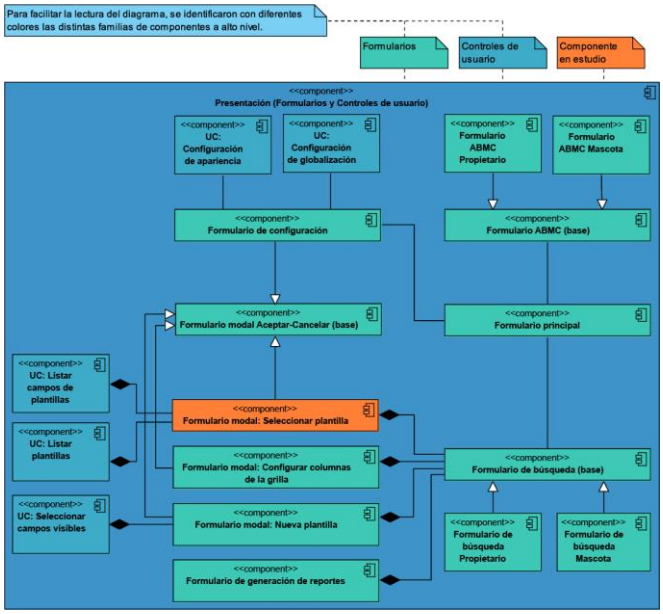


Figura 27

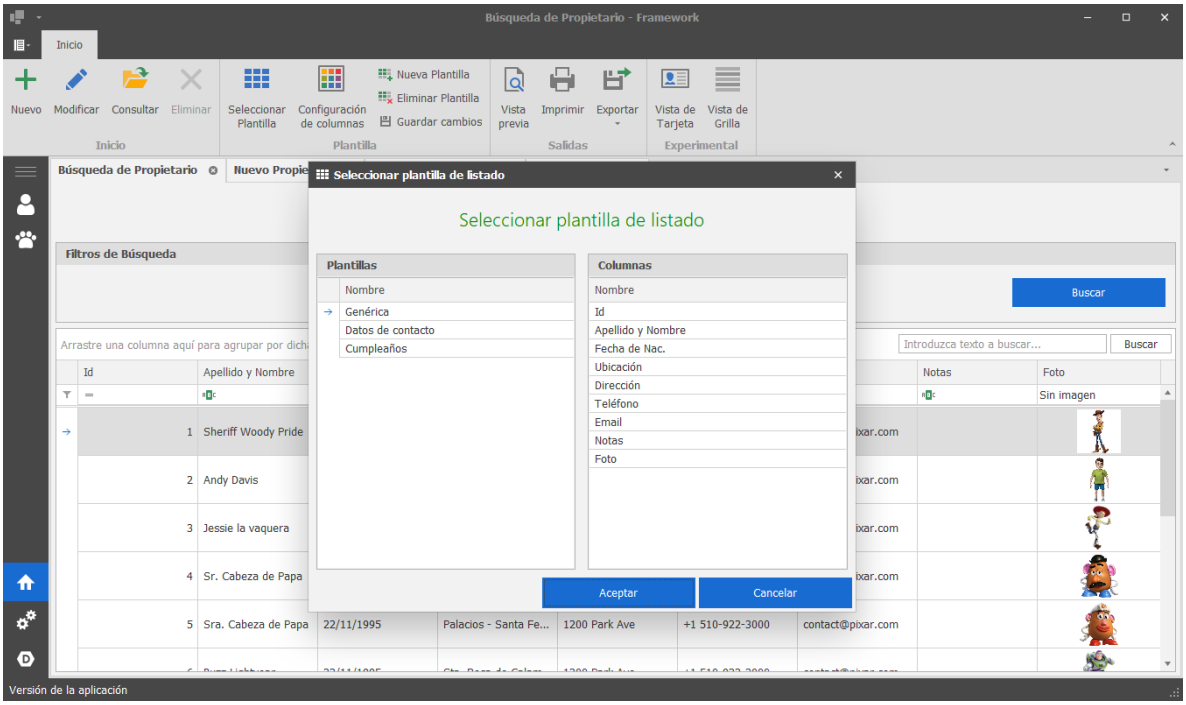


Figura 28

5.10.1.4 Configuración de columnas

En la Figura 29 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio. Luego, la Figura 30 exhibe una captura de pantalla del sistema en ejecución del componente en estudio.

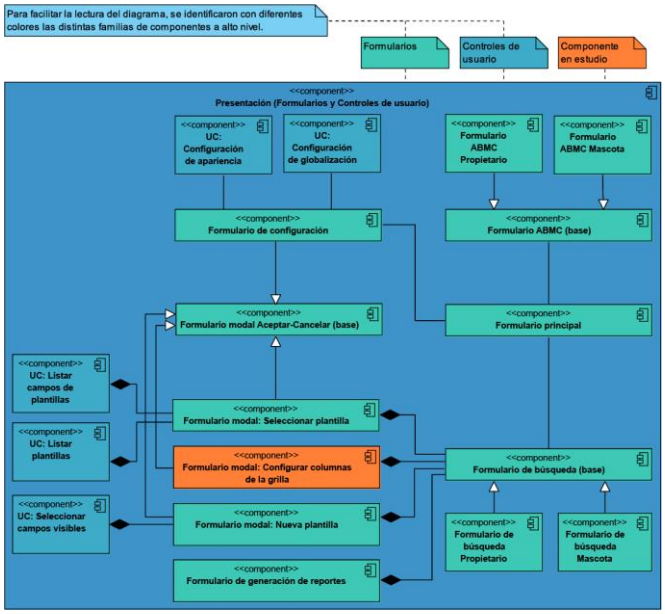


Figura 29

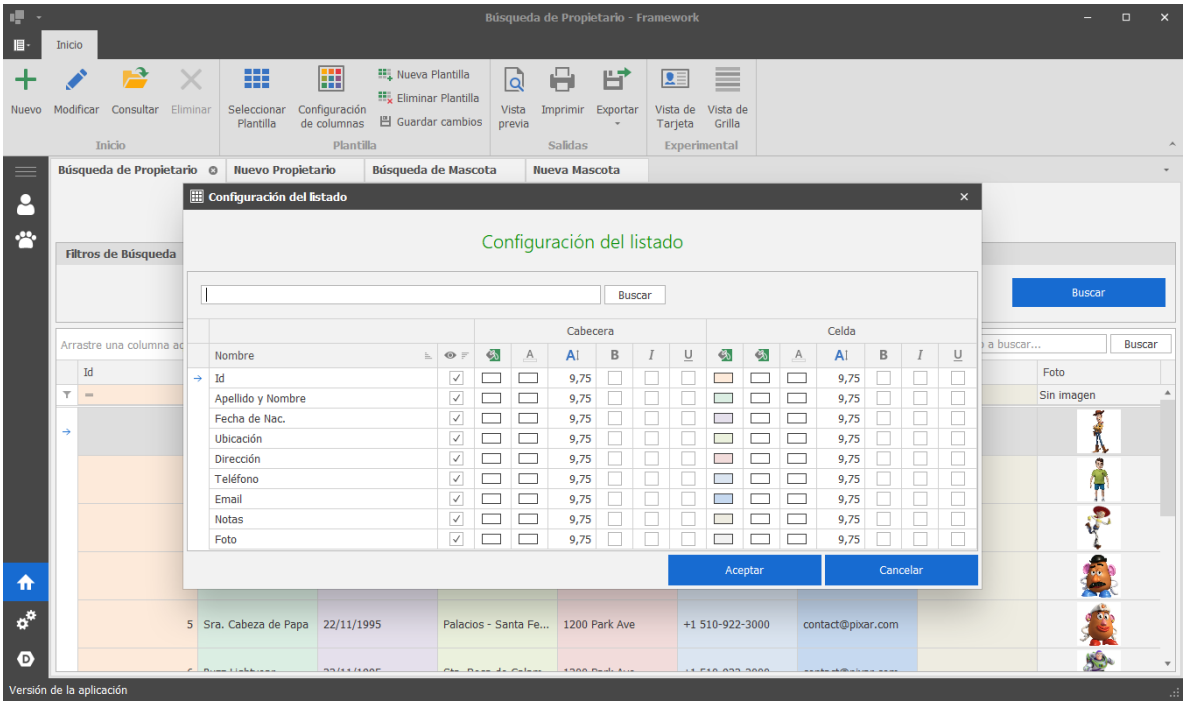


Figura 30

5.10.1.5 Generación de reportes

En la Figura 31 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio. Luego, la Figura 32 exhibe una captura de pantalla del sistema en ejecución del componente en estudio.

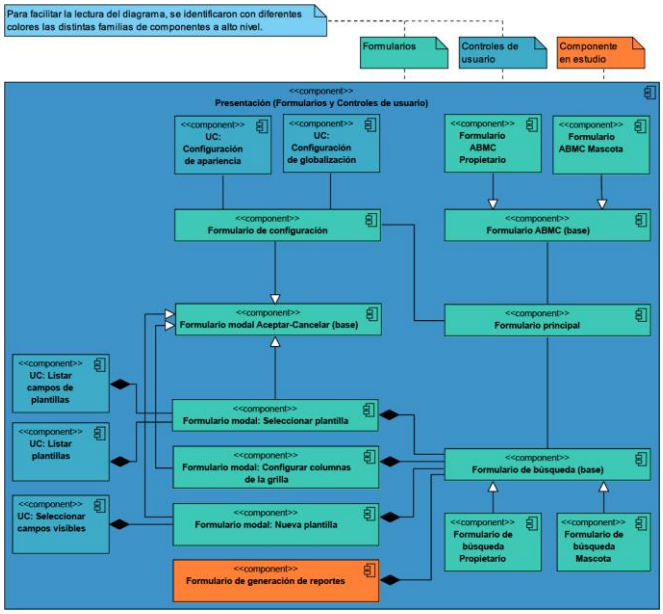


Figura 31

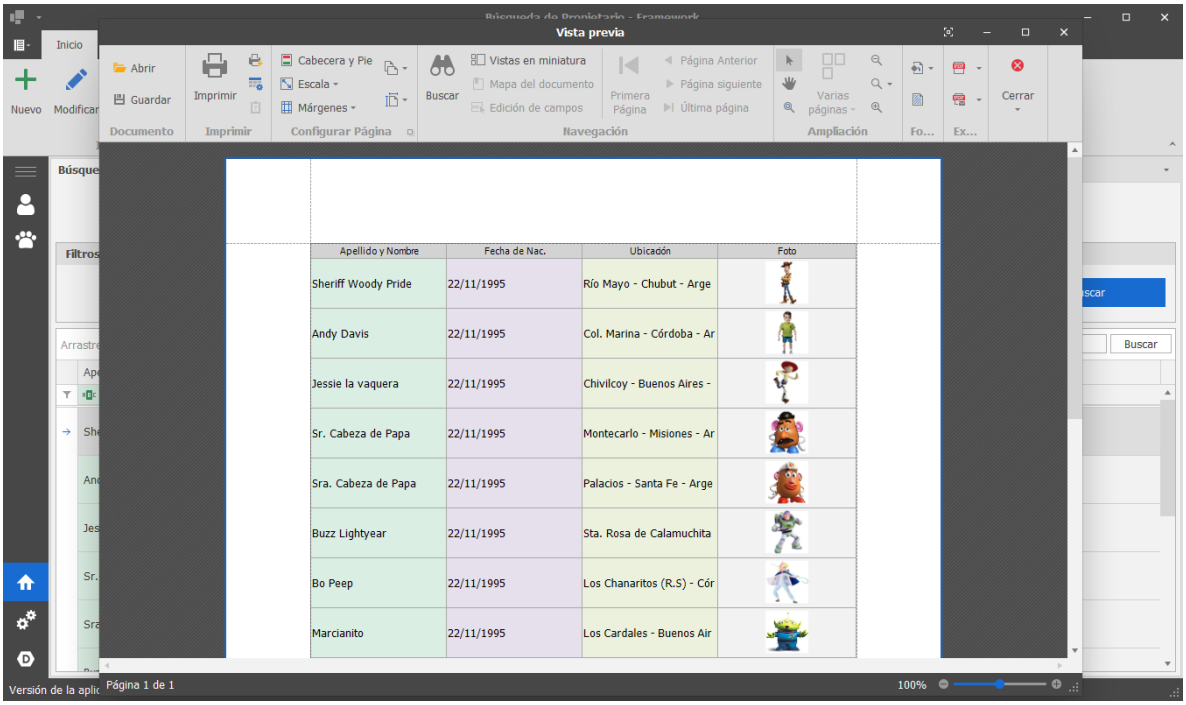


Figura 32

5.10.2 Formularios de Altas, Modificaciones y Consultas

En la Figura 33 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio. Luego, la Figura 34 exhibe una captura de pantalla del sistema en ejecución del componente en estudio.

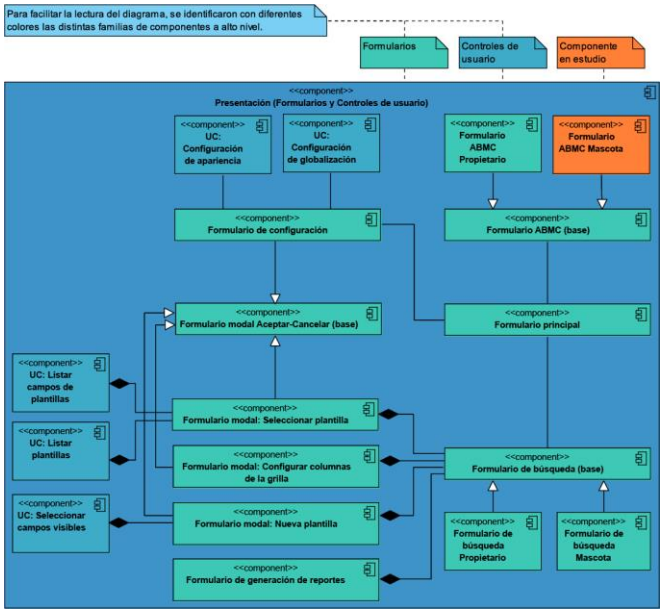


Figura 33

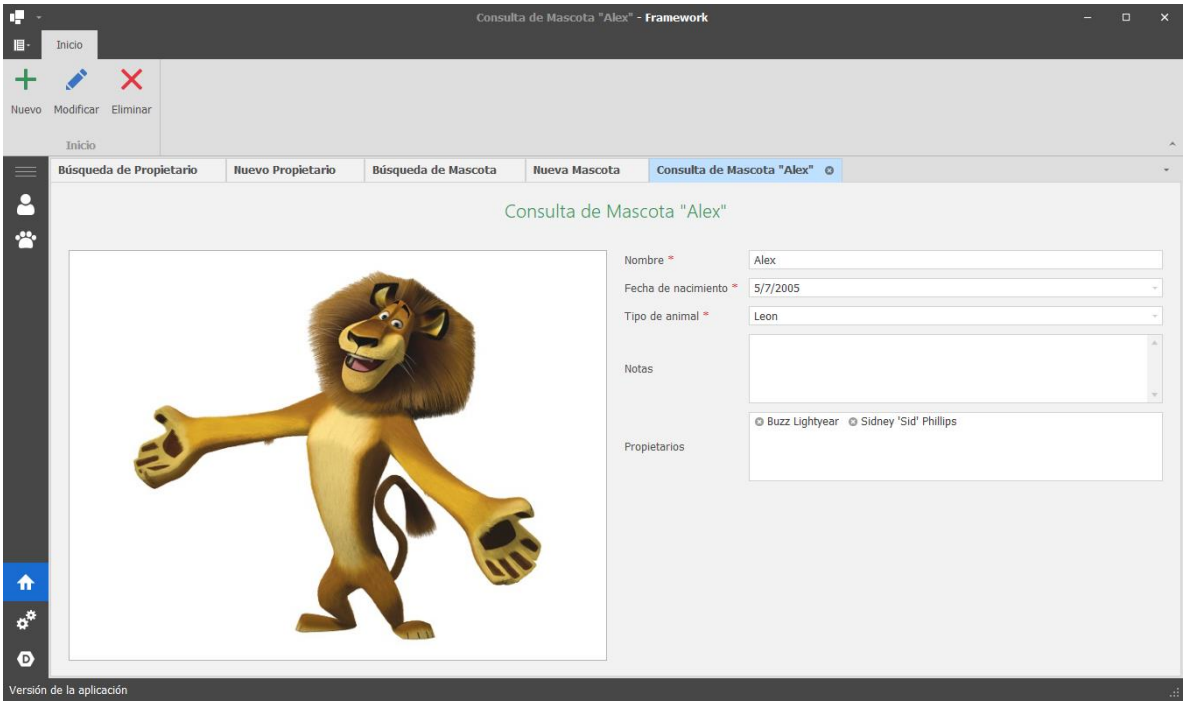


Figura 34

5.10.3 Panel de configuraciones

5.10.3.1 Apariencia

En la Figura 35 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio. Luego, la Figura 36 exhibe una captura de pantalla del sistema en ejecución del componente en estudio.

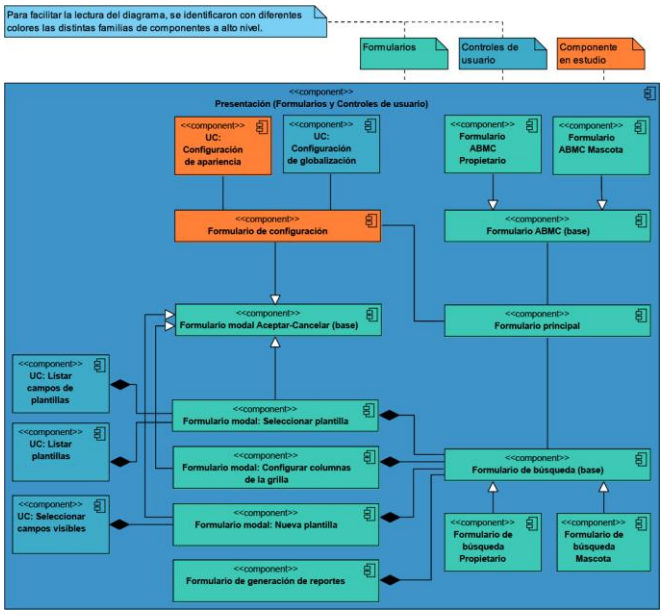


Figura 35

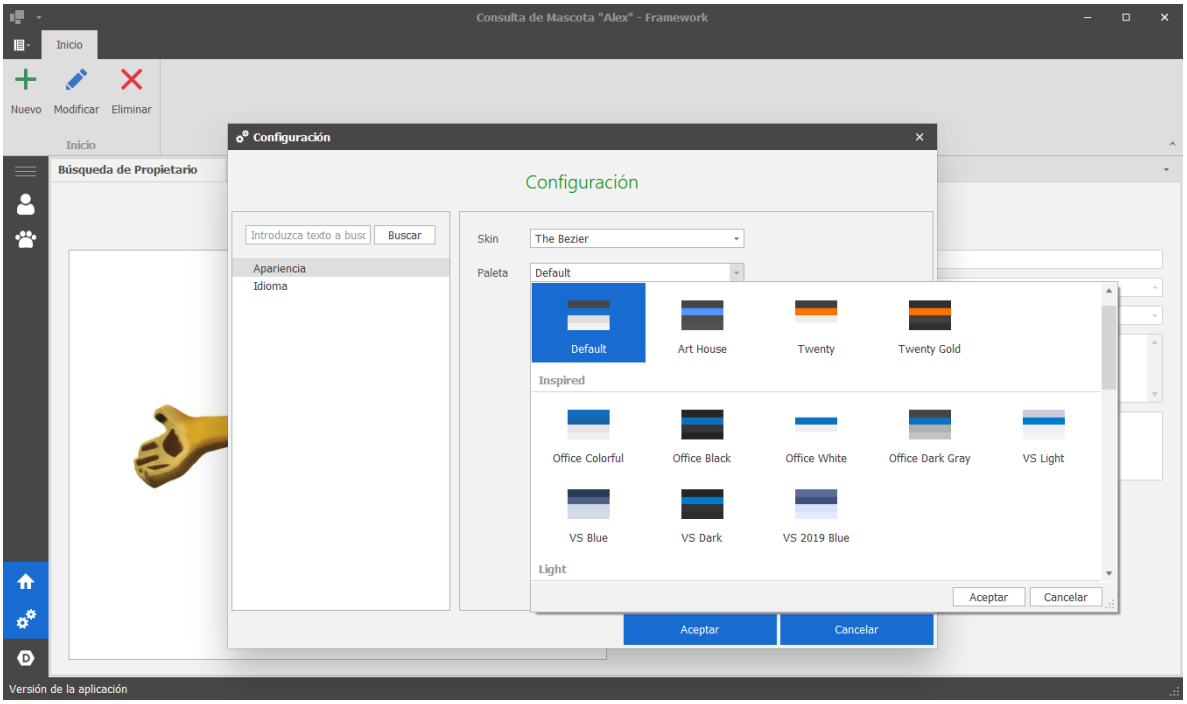


Figura 36

5.10.3.2 Selección de idioma y características regionales

En la Figura 37 se exhibe la capa de presentación desde la perspectiva de la *Vista general de la arquitectura resultante*, destacando con color naranja el componente en estudio. Luego, la Figura 38 exhibe una captura de pantalla del sistema en ejecución del componente en estudio.

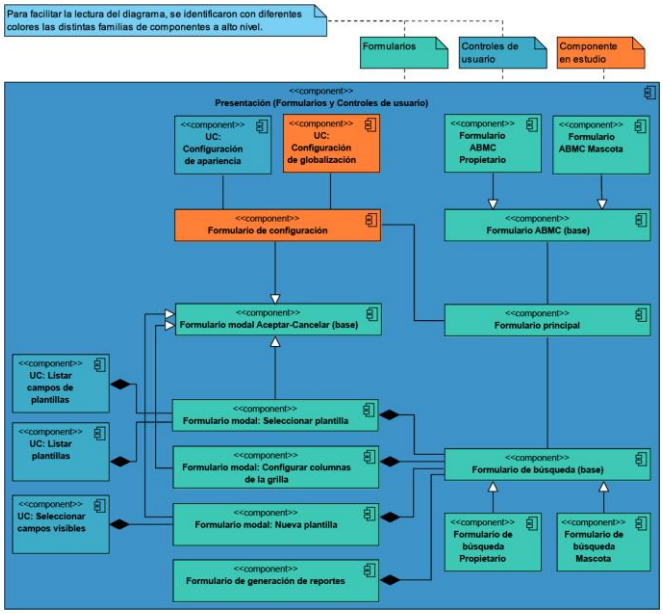


Figura 37

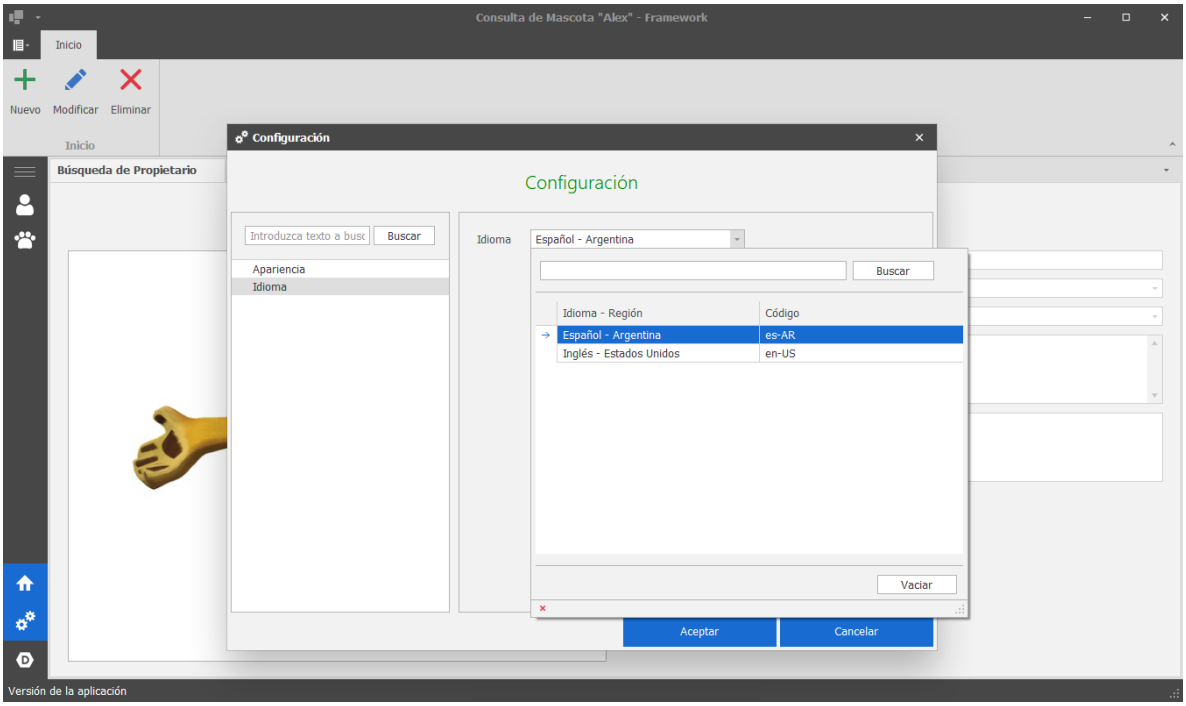


Figura 38

5.10.4 Demostración de diferentes estilos gráficos

En la Figura 39 se exhiben capturas de pantalla del sistema en ejecución, con la aplicación de diferentes estilos gráficos, seleccionados desde el *Formulario de configuración*, a través del *ítem de configuración* "Apariencia".

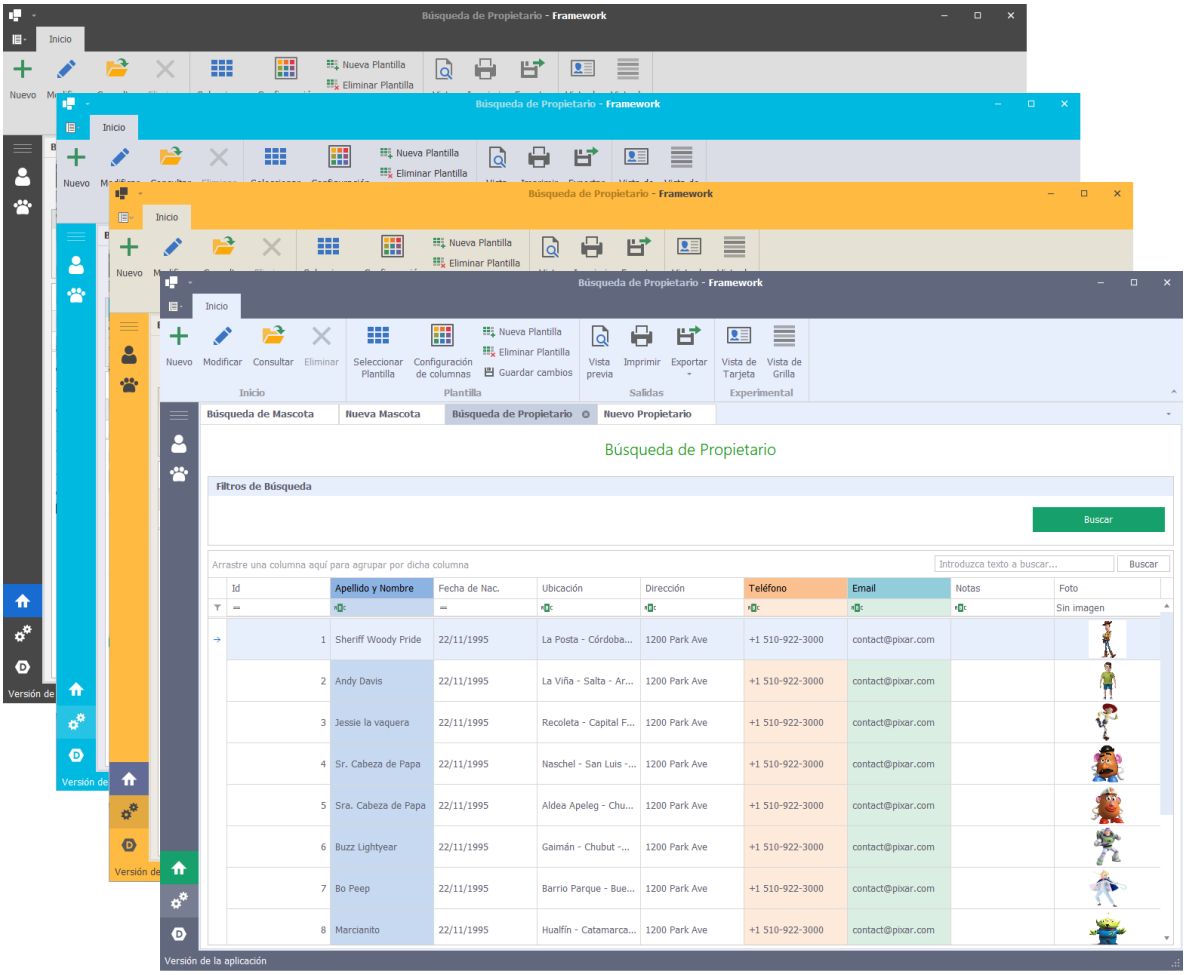


Figura 39

5.10.5 Configuración en otros idiomas

En la Figura 40 se exhiben una captura de pantalla del sistema en ejecución, con la aplicación en idioma inglés, seleccionado desde el *Formulario de configuración*, a través del *ítem de configuración* "Idioma".

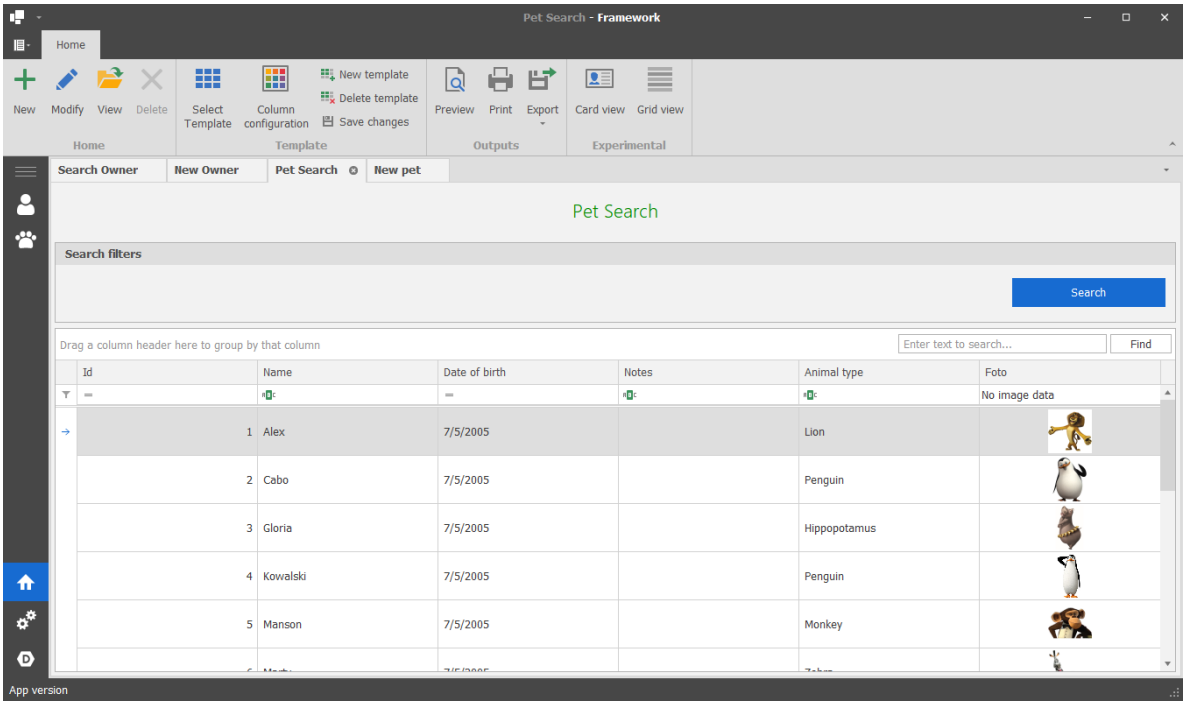


Figura 40

6 Conclusión

Como resultado de la realización del presente proyecto, se obtuvo una solución que satisface las necesidades del cliente y brinda solución a los problemas que le dieron origen al mismo.

El cliente, además, percibe una serie de aportes y mejoras en su actividad, fruto de la implementación de la herramienta desarrollada.

Desde la perspectiva tecnológica, la solución está desarrollada sobre un ecosistema de herramientas modernas, con comunidades amplias que brinda soporte de calidad y alto mantenimiento, y que en su gran mayoría están preparadas para operar en entornos distribuidos, lo cual se alinea con las proyecciones a futuro del cliente.

Por otra parte, desde la perspectiva metodológica, dado que el framework está construido con el uso de principios de desarrollo que alientan las buenas prácticas, el cliente ha incorporado criterios que podrían trascender del uso del framework como herramienta de desarrollo.

El framework y la metodología de trabajo inherente se adecúa a la naturaleza de la empresa, facilitando el trabajo en equipo, permitiendo que distintos programadores desarrollen al mismo tiempo en diferentes áreas de la aplicación.

La metodología de desarrollo seleccionada, y su adaptación al presente escenario resultó apropiada, y la gestión de riesgos acompaño el proceso conduciéndolo a los resultados proyectados.

La finalización del presente proyecto representa sólo el comienzo del ciclo de vida de la solución desarrollada. Las instancias siguientes implican implementar las soluciones actualmente en producción del cliente, sobre el framework desarrollado.

El marco metodológico con sus procedimientos y estimaciones, la gestión de riesgos tanto en su enfoque ágil como estructurado, y la dinámica generada entre el alumno y los stakeholders, sentaron las bases con las que el proyecto actualmente se sigue ejecutando.

6.1 Respecto a los riesgos activados durante la ejecución del proyecto

En el *ANEXO D - Ejecución del proyecto*, para cada Sprint se brindan el contraste entre la planificación y los resultados de su ejecución, la gestión de riesgos, y la actualización del Product Backlog al finalizar cada Sprint.

Durante la ejecución del Sprint Nº 2 se detectó un riesgo no contemplado, el cual se resolvió empleando el enfoque ágil, en una solución en conjunto entre la administración de la empresa y el alumno, y se pudo continuar con el Sprint según lo planificado, no habiéndose generado desvíos en la planificación.

Luego, a lo largo de la ejecución del proyecto, no se activaron disparadores de riesgos, ni se detectaron riesgos bajo el enfoque no estructurado. Al finalizar el proyecto, el alumno prestó especial atención a este aspecto.

Se considera que dada la metodología de elicitación utilizada (wireframes), con un relevamiento acertado de los requerimientos, cubriendo las expectativas en general de todos los

stakeholders involucrados, y además el acompañamiento de las reuniones diarias propuestas por Scrum, condujeron el proyecto desde sus orígenes de forma muy estable y apegada a la planificación.

Por otra parte, el conocimiento previo de las tecnologías involucradas en el desarrollo del proyecto por parte del alumno, mitigo la necesidad de exploración y experimentación con las mismas, llevando en la mayoría de los casos, a desarrollar siempre las propuestas planteadas, sin la necesidad de replanificar.

6.2 Aportes al alumno

Desde las instancias preliminares del proyecto, como la toma de contacto con el cliente, el proyecto en particular tuvo desafíos que el alumno debió resolver con herramientas que no había utilizado antes.

La utilización de wireframes como medio de comunicación, implicó no sólo la capacitación de su uso para todos los stakeholders, sino que también implicó para el alumno el aprendizaje de herramientas para la creación de imágenes vectoriales.

Luego, encuadrar en un marco teórico la metodología de desarrollo utilizada, con el uso de wireframes, para satisfacer las necesidades académicas del proyecto, implicó la investigación del framework Cynefin, para lo cual fue necesario el estudio en detalle de cada elemento mencionado.

Finalizando esta instancia del proyecto desarrollado, el alumno concluye que el desarrollo del framework propuesto fue más que el desarrollo de la solución propuesta, dado que implicó la capacitación de los stakeholders en diferentes aspectos -en situaciones, alejadas de los aspectos técnicos del desarrollo-, y forzó al alumno a incorporar nuevas técnicas y herramientas para hacer frente al desafío.

7 ANEXO A – Modelo de desarrollo de software seleccionado por el tesista

7.1 Introducción

Para guiar las actividades del proceso de desarrollo de software asociado a este Proyecto Final de Carrera, se especificó y empleó una metodología ad-hoc, basada en el marco de trabajo para desarrollo ágil, Scrum.

A los efectos de brindar una perspectiva completa de la ejecución de la metodología, se introducirán de forma resumida los elementos que conforman a la misma: roles, eventos y artefactos.

Se hará mención de aquellos aspectos de Scrum que deban ser adecuados al presente escenario, y finalmente serán presentados todos los desvíos de la metodología ad-hoc, con respecto a la metodología de base.

Los elementos de la metodología de base -Scrum-, son globalmente utilizados en su idioma nativo, el inglés. Por esta razón, se emplearán los nombres establecidos por sus creadores, tal cual. No así las definiciones de las mismas que se presentarán en este documento, las cuales serán en español.

A los fines de evitar confusiones, se hará distinción de las definiciones, separándolas de los comentarios relacionados a las mismas.

7.2 Roles

Scrum define su *Scrum Team* basado en tres roles fundamentales: el *Product Owner*, el *Development Team*, y el *Scrum Master*. A continuación, se definirán los mismos, se presentarán las adecuaciones en caso que existan, y se detallará que personas representarán dichos roles en el presente proyecto.

7.2.1 Product Owner y Scrum Master

Product Owner, definición

El *Product Owner* es un representante de las partes interesadas, tanto internas como externas. Es quien debe comprender correctamente las necesidades externas, para maximizar el valor del producto resultante del trabajo del *Development Team*.

La única persona responsable de administrar el *Product Backlog* es el *Product Owner*. Debe expresar claramente los elementos del *Product Backlog*, garantizando que el *Development Team* comprenda correctamente los mismos. También es su responsabilidad el orden de los ítems del *Product Backlog* para lograr los objetivos y misiones más relevantes en el tiempo apropiado.

Scrum Master, definición

El *Scrum Master* es responsable de promover a Scrum tal como se define. Los *Scrum Masters* hacen esto al ayudar a todos los involucrados a comprender la teoría, las prácticas, las reglas y los valores de Scrum.

Una de las funciones del *Scrum Master* es ayudar a que aquellos que están fuera del *Scrum Team* entiendan cuales interacciones con el *Scrum Team* son beneficiosas, y cuáles no. De esta forma, el *Scrum Master* maximiza el valor creado por el *Scrum Team*.

Como regla general para todos los eventos, el *Scrum Master* asegura que los mismos se lleven a cabo y que los asistentes comprendan su propósito. El *Scrum Master* les enseña a los participantes a mantenerlos dentro del tiempo estipulado.

Por otro lado, el *Scrum Master* brinda servicios al *Product Owner*, tales como asegurar que los objetivos, el alcance y el dominio del producto sean entendidos por todos en el *Scrum Team* lo mejor posible.

Comentarios

El rol *Product Owner* y *Scrum Master* serán representado por el líder de desarrollo de la empresa, quien tiene experiencia en la ejecución de los roles de Scrum, y, además, posee el concepto del producto deseado.

El hecho que ambos roles sean representados por una misma persona, y además en este caso, que dicha persona tenga absoluto conocimiento del producto deseado tiene numerosas ventajas tanto para la perspectiva del cliente, como para la del alumno.

Uno de los servicios que el *Scrum Master* le brinda al *Product Owner*, es asegurar que los objetivos, el alcance y el dominio del producto sean entendidos por todos en el *Scrum Team*. Tratándose en este caso de la misma persona, este aspecto será concretado óptimamente.

Dada la numerosa y diversa cantidad de *Stakeholders* (los cuales son presentados más adelante), y los escenarios/intereses variados que cada uno representa, que el *Scrum Master* vele por una comunicación eficaz evita la solicitud de requerimientos encontrados, o expresados ambiguamente, entre otras.

Scrum define que la definición y orden de los ítems del *Product Backlog* es efectuada por el *Product Owner*. En el marco de este proyecto, los ítems serán definidos por el alumno basados en la interpretación de la necesidad del solicitante, y posteriormente serán supervisados por el *Product Owner*. El refinamiento de las definiciones de los ítems y la organización de los mismos será efectuado en conjunto.

7.2.2 Desarrollador único

Development Team, definición

El *Development Team* está formado por un grupo de profesionales que realizan el trabajo de entregar un potencial incremento del producto al final de cada *Sprint*.

Los integrantes del *Development Team* están calificados para autoorganizar y administrar su propio trabajo. La sinergia resultante entre los integrantes del equipo optimiza la eficiencia y efectividad general del *Development Team*.

Comentarios

En este escenario se cuenta con solo un desarrollador, el alumno. El *Desarrollador único* reemplazará al rol de *Scrum Team* de Scrum, pero desempeñará sus funciones. Se utilizará el nombre *Desarrollador único* para referirse al rol. Finalmente, el alumno será el único responsable de la generación de incrementos.

7.2.3 Stakeholders

Definición

El término no fue acuñado por Scrum. Un *stakeholder* es cualquier individuo u organización que, de alguna manera, es impactado por o tiene intereses sobre las acciones de determinada empresa. En una *traducción libre*[21] para el español, significa “partes interesadas”.

En Scrum se asocia a los *Stakeholders* con cualquier individuo u organización que pueda tener intereses que deban ser tenidos en cuenta en el producto a desarrollar.

Comentarios

En el presente proyecto, el resto del personal de la empresa solicitante, asociado de alguna forma al producto, será enmarcado con el perfil de *Stakeholder*. Los integrantes de la sociedad, desarrolladores, usuarios internos, etc., incluyendo al líder de desarrollo (*Product Owner* y *Scrum Master* en este proyecto), serán asociados a este perfil.

7.3 Eventos

Se transitarán los eventos de Scrum según lo propone el framework, sin efectuar cambios en su flujo. Más adelante en este apartado, luego de introducir cada tipo de evento, se especificará la duración de los mismos.

Los eventos prescritos en Scrum se usan para crear regularidad y minimizar la necesidad de reuniones no definidas. Cada evento en Scrum es una oportunidad formal para inspeccionar y adaptar algo.

Estos eventos están diseñados específicamente para permitir la transparencia crítica y la inspección. Si no se incluye alguno de estos eventos, se reduce la transparencia y se pierde la oportunidad de inspeccionar y adaptar.

Todos los eventos en Scrum se basan en una técnica llamada *Timebox*, que consiste en establecer una duración máxima a cada evento. De esta manera, en lugar de ponerse a trabajar en algo hasta que esté hecho, se especifica de antemano que se dedicará un tiempo limitado a ello.

7.3.1 Sprint

Definición

[21] La traducción literal se realiza “palabra por palabra”. En la traducción libre el traductor analiza el sentido de toda la frase e intenta reflejarlo en su idioma de la manera en que lo haría un hablante nativo. Esto puede suponer modificar la estructura del texto si el traductor considera que es necesario para que suene mejor.

El *Sprint* es uno de los elementos centrales de Scrum, ya que actúa como contenedor del resto de los eventos. Con la finalización de un *Sprint* se genera un incremento que cumple con el concepto de *"Done"* (el cual será especificado luego de introducir los artefactos).

Para el caso del *Sprint*, una vez que el mismo comienza, su duración es fija y no se puede acortar o alargar. Los eventos restantes pueden finalizar antes siempre que se logre el propósito del evento, evitando el desperdicio de tiempo.

Los *Sprints* tienen duraciones constantes durante todo el esfuerzo de desarrollo. Un nuevo *Sprint* comienza inmediatamente después de la conclusión del *Sprint* anterior.

El *Sprint* contiene y consiste en el *Sprint Planning* ("Planificación de Sprint"), los *Daily Scrums* ("Scrum diarios"), el trabajo de desarrollo, el *Sprint Review* ("Revisión de Sprint") y el *Sprint Retrospective* ("Retrospectiva de Sprint").

7.3.2 Sprint Planning

Definición

El trabajo a realizar en el *Sprint* se planifica en el *Sprint Planning*. Este plan es creado por el trabajo colaborativo de todo el *Scrum Team*.

En el *Sprint Planning* se decide qué será realizado durante el *Sprint*, mediante la selección de elementos del *Product Backlog*, lo cual genera como resultado al *Sprint Backlog*. Contando con el *Sprint Backlog*, el equipo establecerá el *Sprint Goal*, que es un objetivo único que será alcanzado tras la realización de todos los ítems del *Sprint Backlog*.

Finalmente, se obtiene una perspectiva del *"¿Qué?"* mediante el *Sprint Goal*, y del *"¿Cómo?"* mediante el *Sprint Backlog*.

7.3.3 Daily Scrum

Definición

El *Daily Scrum* es un evento breve destinado al *Development Team*, que se lleva a cabo todos los días del Sprint. Basado en el último *Daily Scrum* el equipo de desarrollo planea el trabajo para el próximo día de trabajo en el *Sprint*. Esto optimiza la colaboración y el rendimiento del equipo mediante las tareas de inspección en conjunto, e incrementa la probabilidad de que el equipo cumpla con el objetivo de *Sprint*.

El *Daily Scrum* se lleva a cabo a la misma hora y lugar cada día para reducir la complejidad. El *Development Team* usa el *Daily Scrum* para inspeccionar el progreso hacia el *Sprint Goal* mediante la completitud de los ítems del *Sprint Backlog*.

El *Daily Scrum* es una reunión interna para el equipo de desarrollo. Si hay otros presentes, el *Scrum Master* se asegura que no generen interrupciones en la reunión.

Suele ser de utilidad plantearse las siguientes preguntas en esta reunión:

- ¿Qué hice ayer para ayudar al *Development Team* a alcanzar el *Sprint Goal*?
- ¿Qué haré hoy para ayudar al *Development Team* a alcanzar el *Sprint Goal*?
- ¿Veo algún impedimento que me impida a mí o al *Development Team* a cumplir con el *Sprint Goal*?

Comentarios

En el presente proyecto, este evento será llevado a cabo sólo con dos participantes, el *Desarrollador único* (alumno) y el *Scrum Master* (líder de desarrollo de la empresa solicitante).

Como se mencionó anteriormente, la persona que desempeñará el rol de *Scrum Master* y el de *Product Owner* es la misma. Además, dicha persona es un importante *Stakeholder*, ya que se trata del líder de desarrollo de la empresa solicitante, y uno de los integrantes de la sociedad de la misma.

De este modo, estas reuniones adquieren un valor extra para el alumno, dado que además las mismas formalizarán de forma indirecta, un contacto diario con la empresa, y se espera que fortalezcan el vínculo de diálogo y confianza, sin desvirtuar la finalidad específica de este evento.

7.3.4 Sprint Review

Definición

Se realiza un *Sprint Review* al final del *Sprint* para inspeccionar el incremento generado y adaptar el *Product Backlog* en caso de ser necesario. Durante el *Sprint Review*, el *Scrum Team* analiza lo realizado en el *Sprint* en conjunto con el *Product Owner*. Es un momento importante para que el *Product Owner* y los *Stakeholders* sepan lo que está ocurriendo con el producto.

Esta es una reunión de carácter informal, no se trata de una reunión de estado, y la presentación del Incremento está destinada a generar comentarios y fomentar la colaboración. Se trata de una reunión clave para la inspección y consecuente adaptación del producto.

Una porción del *Sprint Review* es dedicada a la presentación de una demo del incremento. No se utilizan presentaciones de diapositivas, y la preparación de esta presentación debe ser muy breve. No debe confundirse el carácter de esta reunión con solo la demo del producto.

Finalmente, el grupo completo colabora en la idea de que hacer a continuación, por lo que este *Sprint Review* genera un importante elemento para el comienzo del siguiente *Sprint*. El resultado es un *Product Backlog* revisado, que define los elementos probables del *Producto Backlog* para el próximo *Sprint*.

7.3.5 Sprint Retrospective

Definición

El *Sprint Retrospective* es una oportunidad para que el *Scrum Team* se inspeccione a sí mismo y cree un plan de mejoras que se implementarán durante el próximo *Sprint*.

El propósito de este evento es:

- Inspeccionar cómo fue el último *Sprint* con respecto a las personas, a las relaciones entre los participantes, los procesos y las herramientas.
- Identificar y ordenar los elementos principales que salieron bien y las posibles mejoras.
- Cree un plan para implementar mejoras en la forma en que el Equipo Scrum hace su trabajo.

Comentarios

En el marco de este proyecto, el *Sprint Retrospective* será desarrollado sólo por el *Desarrollador único*. Contando con la asistencia del *Scrum Master* en caso de ser necesaria.

7.3.6 Duración de los eventos

La duración de los eventos fue asignada por el *Scrum Master*. A continuación, se empleará *la semana* como unidad constante, definidas con 5 días laborales, donde cada día laboral es de 8 horas.

Nombre del evento	Duración
Sprint	2 [semanas]
Sprint Planning	4 [horas]
Daily Scrum	15 [minutos]
Preparación de presentación para el Sprint Review	1 [horas]
Sprint Review	2 [horas]
Sprint Retrospective	2 [horas]

Basados en la duración de cada evento, y las horas laborales semanales, podemos estimar de forma aproximada la cantidad de horas de desarrollo por *Sprint* con las que contará el Desarrollador único:

- Horas totales por *Sprint* ≈ 2 x 1 [semana] ≈ 2 x 5 [días] x 8 [horas] ≈ 80 [horas]
- Horas totales en eventos del *Sprint* ≈
  ≈ Sprint Planning + 10 x Daily Scrum + Preparación de presentación para el Sprint Review + Sprint Review + Sprint Retrospective ≈
  ≈ 4 [horas] + 10 x 15 [min] + 1 [hora] + 2 [horas] + [2 horas] ≈ 11,5 [horas]
- Cantidad de horas de desarrollo por *Sprint* ≈ Horas totales por Sprint – Horas totales en eventos del Sprint ≈ 80 [horas] – 11,5 [horas] ≈ 68,5 [horas]

7.4 Artefactos

Scrum define artefactos que representan trabajo, basado en el concepto de transparencia, y los mismos siempre proporcionan oportunidades de inspección y adaptación. Scrum define los siguientes tres artefactos: *Product Backlog*, *Sprint Backlog*, e *Increment*.

7.4.1 Product Backlog

Definición

El *Product Backlog* es una lista ordenada de todo lo que se sabe que se necesita en el producto. Es la única fuente de requerimientos -funcionales y no funcionales- para cualquier cambio que se realice en el producto. El *Product Owner* es responsable del *Product Backlog*, incluido su contenido, disponibilidad y orden.

Un *Product Backlog* nunca está completo. El desarrollo inicial del *Product Backlog* establece los requisitos inicialmente conocidos, pero evoluciona a medida que evoluciona el producto y el entorno en el que se utilizará. El *Product Backlog* es dinámico, cambia constantemente para identificar lo que el producto necesita para ser apropiado, competitivo y útil.

El *Product Backlog* enumera todas las características, funciones, requisitos, mejoras y correcciones que constituyen los cambios que se realizarán en el producto en futuras versiones. Los elementos del *Product Backlog* tienen los atributos descripción, orden, estimación y valor.

Comentarios

En el presente proyecto de desarrollo de software, se enfatizará la tarea de refinamiento y estimación de los ítems del *Product Backlog* antes de comenzar el proyecto, a los efectos de obtener un *Product Backlog* tan estático y detallado como sea posible, que posibilite cumplir con las fechas críticas del aspecto académico de esta presentación.

Aun así, no se pretende modificar las características de dinamismo del *Product Backlog*. El mismo será adecuado siempre que la situación lo amerite, tal y como su definición lo establece.

Previamente, en la definición del rol *Product Owner*, se mencionó que Scrum define que la tarea de orden y definición de los ítems del *Product Backlog* debe ser realizada por dicho rol, pero que en el presente proyecto la organización de los ítems será realizada en conjunto.

Para establecer el orden de los elementos en conjunto (entre el *Desarrollador único* y el *Product Owner*), serán tenidas en cuenta la p*rioridad en el negocio* y la *prioridad en el desarrollo* de cada *User story*.

La *prioridad en el negocio* refleja aquellas funcionalidades que el solicitante desea conocer primero. La *prioridad en el desarrollo* refleja aquellas funcionalidades necesarias para el avance del desarrollo, tales como ciertas funcionalidades esenciales que son utilizadas por otras funcionalidades (es decir, que hay un orden técnico que no puede evitarse).

Se ordenará la secuencia las *User stories* otorgando mayor peso a las prioritarias en términos de desarrollo. Podría pensarse que las *User stories* serán ordenadas tomando como primer criterio de ordenamiento la *prioridad en desarrollo*, y como segundo criterio la *prioridad en el negocio*.

Es decir, que de existir una *User story* “A” que sea de alta prioridad en el negocio, pero de baja prioridad en el desarrollo, y de existir una *User Story* “B” que sea de alta prioridad en el desarrollo, pero de baja prioridad en el negocio, será mayormente ponderada la *User Story* “B”.

7.4.2 Sprint Backlog

Definición

El *Sprint Backlog* es el conjunto de elementos del *Product Backlog* para realizar en el *Sprint* actual, más un plan para entregar el *Incremento* del producto y alcanzar el *Sprint Goal*. El *Sprint Backlog* es un pronóstico del *Development Team* sobre qué funcionalidad estará en el próximo Incremento y cuál es el trabajo necesario para entregar esa funcionalidad en un incremento. Cada *Sprint*, está ligado a sólo un *Sprint Backlog* que lo define.

7.4.3 Increment

Definición

El incremento, es la suma de todos los ítems del *Product Backlog* completados en el último *Sprint*, y del valor de todos los incrementos previos. Al final de un *Sprint*, el incremento debería cumplir con el concepto de *"Done"*. El *incremento* es un elemento inspeccionable, que soporta las prácticas empíricas al final del *Sprint*. Un incremento representa un avance hacia el objetivo del proyecto, y el *Product Owner* podría elegir cuando desplegarlo.

Comentarios

En el presente proyecto, la decisión de cuándo desplegar un incremento será tomada en conjunto entre el *Product Owner* y el *Desarrollador único*.

7.5 Transparencia de los artefactos

Scrum se basa en la transparencia. Las decisiones para optimizar el valor y controlar el riesgo se toman en función del estado percibido en los artefactos. En la medida en que los principios de "transparencia" sean implementados, estas decisiones tendrán una base sólida.

7.5.1 Definición de "Done" ("Completo")

Definición

Cuando un elemento del *Product Backlog* o un *Incremento* se describe cómo "Done" (*"Completo"*), todos deben entender lo que eso significa. Aunque esto puede variar según el *Scrum Team*, los miembros deben tener una comprensión compartida de lo que significa que el trabajo esté completo, para garantizar la transparencia. Esta es la definición de *"Done"* para el *Scrum Team* y se utiliza para evaluar cuándo el trabajo está completo en un *Incremento* de producto.

7.6 Ítems del Product Backlog

Como se ha definido previamente, Scrum especifica las características mínimas con las que debe contar un ítem del *Product Backlog*, pero deja abierto a criterio y necesidad del escenario la definición de los mismos.

En el presente proyecto, los ítems del *Product Backlog* serán especificados utilizando el enfoque ágil de *User Stories*, y las mismas serán estimadas mediante *Story Points*.

La selección de *User Stories* para representar los ítems del *Product Backlog* fue una sugerencia del *Scrum Master*, dado que es de las más comunes en la gestión y desarrollo ágil de proyectos. Además, la empresa solicitante la utiliza y está familiarizada con las misma.

7.6.1 User Story

Definición

En el desarrollo de software y la gestión ágil de productos, una *User Story* es una descripción informal en lenguaje natural de una o más características de un sistema de software.

Una *User Story* es una herramienta utilizada para capturar la descripción de un requerimiento -funcional o no funcional- del software desde la perspectiva del usuario final. La ventaja de la herramienta es, que se centra exactamente en lo que el usuario necesita/quiere sin entrar en detalles sobre cómo lograrlo.

Una *User Story* describe el tipo de usuario, lo que quiere y por qué, para finalmente dar origen a una descripción simplificada de un requerimiento.

Comentarios

En el presente proyecto, las *User Stories* serán presentadas mediante la siguiente plantilla:

ID [n]	**[Título de la *User Story*]**
Como un…	[Rol]
quiero…	[Descripción de la necesidad]
para…	[Razón de ser de la necesidad]
Observaciones:	[Cualquier información que ayude a mejorar el entendimiento de la *User Story*]

7.6.2 Story Points

Definición

Un *Story Point* es una métrica utilizada en el desarrollo y gestión ágil de productos, para estimar la dificultad de implementar una *User Story*. Se trata de una medida abstracta -un número- con el que el equipo está familiarizado y puede dimensionar. Un *Story Point* representa el "esfuerzo relativo" necesario para concretar una tarea.

Comentarios

En el apartado "*Duración de los eventos*" se ha definido que un *Sprint* tiene una duración de 2 semanas, de 10 días laborales, y se estimó que por *Sprint* se contaría con aproximadamente 68,5 horas dedicadas exclusivamente a la actividad de desarrollo.

Teniendo en cuenta lo anterior, se define un peso proporcional para los *Story Point*, entre la cantidad de horas de desarrollo disponibles por *Sprint*, y la cantidad de días laborales por *Sprint*, resultando:

- Story Point ≈ Cantidad de horas de desarrollo por *Sprint* / Días laborales por *Sprint* ≈
  ≈ 68,5 [hs] / 10 [días] ≈ 6,85 [hs]

Se pretende, de esta manera, asignar un peso relativo a cada *Story Point*, acorde a la medida de "un día de trabajo" realizado por el *Desarrollador único*.

Luego, el *esfuerzo relativo* asociado a un *User Story*, puede ser cuantificado utilizando el intervalo 6,5 [hs] ± 30 [min].

Es posible estimar la cantidad de Story Points a ser realizados por *Sprint*, efectuando el cociente entre la cantidad de horas dedicadas exclusivamente a la actividad de desarrollo en un *Sprint*, y al peso de un Story Point:

- Story Poins por *Sprint* ≈ Peso de Story Point / Cantidad de horas de desarrollo por *Sprint* ≈
  ≈ 6,85 [hs] / 68,5 [hs] ≈ 10 [Story Point]

7.7 Resumen de desvíos y particularidades respecto a Scrum

7.7.1 En Roles

- El *Development Team* cuenta con solo una persona, el *Desarrollador único*. Este rol es ejecutado por el alumno.
- El rol de *Product Owner* y *Scrum Master* es ejecutado por una misma persona, el líder de desarrollo de la empresa solicitante.
- Scrum establece que la definición y orden de los ítems del *Product Backlog* es efectuada por el *Product Owner*. En el marco de este proyecto, los ítems serán definidos por el alumno basados en la interpretación de la necesidad del solicitante, y posteriormente serán supervisados por el *Product Owner*. El refinamiento de las definiciones de los ítems y la organización de los mismos será efectuado en conjunto.

7.7.2 En Eventos

- El *Daily Scrum* será realizado sólo con dos participantes, el *Desarrollador único* y el *Scrum Master*.
- El *Sprint Retrospective* será ejecutado solo por el *Desarrollador único*, contando con la asistencia del *Scrum Master* de ser necesaria.

7.7.3 En Artefactos

- Se enfatizará la tarea de refinamiento y estimación de los ítems del *Product Backlog* antes de comenzar el proyecto, a los efectos de obtener un *Product Backlog* tan estático como sea posible, que posibilite cumplir con las fechas críticas del aspecto académico de esta presentación.
- Para establecer el orden de los ítems del *Product Backlog* en conjunto (entre el *Desarrollador único* y el *Product Owner*), serán tenidas en cuenta la *prioridad en el negocio* y la *prioridad en el desarrollo* de cada *User story*.
  Las *User stories* serán ordenadas tomando como primer criterio de ordenamiento la *prioridad en desarrollo*, y como segundo criterio *la prioridad en el negocio*.

- En el presente proyecto, la decisión de cuando desplegar un incremento será tomada en conjunto entre el *Product Owner* y el *Desarrollador único*.

7.7.4 En ítem del Product Backlog

- En el presente proyecto, las *User Stories* serán presentadas mediante la siguiente plantilla:

ID [n]	**[Título de la *User Story*]**
Como un…	[Rol]
quiero…	[Descripción de la necesidad]
para…	[Razón de ser de la necesidad]
Observaciones:	[Cualquier información que ayude a mejorar el entendimiento de la *User Story*]

- Un *Story Point* será aproximadamente 6,85 [hs], y tiene un peso relativo acorde a la medida de "un día de trabajo" realizado por el *Desarrollador único*.

8 ANEXO B – Actividades a ser realizadas

Introducción

Las actividades a ser realizadas en el presente Proyecto Final de Carrera, se detallaron utilizando el formato de *User stories*, como se ha definido en el *Modelo de desarrollo de software seleccionado por el tesista*.

El *Orden de ejecución de las User stories* se definió posteriormente, basado en los criterios de ordenación definido para los ítems del *Product Backlog*, y se documentó en

En el proceso de recolección de requerimientos empleando el formato de *User stories*, surgieron los siguientes roles:

- Desarrollador: respecta al desarrollador que hará uso del framework, es decir, aquel que agregará valor haciendo uso de las características provistas por el mismo.
- Usuario: respecta al usuario final que consumirá el producto de software finalizado, es decir, que hará uso del producto desarrollado por el rol *Desarrollador*.

ID 1 - Selección de herramientas esenciales del mercado

ID 1	Selección de herramientas esenciales del mercado
Como un…	Desarrollador
quiero…	Un paquete de controles de usuario
para…	Construir el framework
Observaciones:	En el mercado existen numerosos paquetes de controles de usuarios, que se adecúan a las exigencias de usabilidad y estética de los estándares actuales. Muchos de los paquetes de controles de usuarios, se distribuyen como soluciones homogéneas que brindan además de los controles de usuarios, el soporte de diferentes estilos gráficos en los mismos y la generación de reportes. La elección de un paquete maduro que se adecúe a las necesidades del producto, condicionará directamente la velocidad de avance del proyecto. Serán analizadas las siguientes soluciones del mercado: DevExpress https://www.devexpress.com/products/net/controls/winforms/ DotNetBar https://www.devcomponents.com/dotnetbar/ Grapecity https://www.grapecity.com/componentone/winforms-ui-controls Telerik https://www.telerik.com/products/winforms.aspx Syncfusion https://www.syncfusion.com/winforms-ui-controls El conjunto de soluciones a ser analizadas consta de los exponentes líderes del mercado. El costo de adquisición y licenciamiento del producto seleccionado será asumido por la empresa solicitante. Las soluciones preseleccionadas cumplen con los criterios económicos establecidos por el solicitante.

ID 2 - Desarrollo de la arquitectura inicial

ID 2	Desarrollo de la arquitectura inicial
Como un…	Desarrollador
quiero…	Una arquitectura inicial
para…	Soportar el desarrollo de los módulos planificados
Observaciones:	Algunas de las tareas previstas en esta etapa, y fundamentales para la evolución del proyecto son: - Crear y configurar el proyecto inicial en Microsoft Visual Studio. - Crear la Base de Datos para el proyecto (vacía) en Microsoft SQLServer. - Añadir capas (“layers”) esenciales al proyecto, que inicialmente serán: Acceso a datos, Lógica de negocio, y Presentación. - Configurar el mapeador objeto-relacional Microsoft Entity Framework en la capa de acceso a datos, utilizando el esquema “code first”. - Desarrollar Splash Screen. - Desarrollar lógica de validaciones esenciales en el inicio (acceso a la base de datos, otra instancia de la aplicación abierta, etc). - Desarrollar lógica que instancie la Base de Datos si no existe. - Desarrollar lógica que permita declarar “semillas”, para llenar la base datos con información esencial, en caso que no exista.

	- Desarrollar un repositorio de parámetros para almacenar aquellas entidades de datos que no cambien durante la ejecución de la aplicación. - Desarrollar entidad de datos que permita almacenar parámetros internos del sistema, y configuraciones del usuario. - Desarrolla lógica que permita declarar rutinas que serán ejecutadas en serie en el inicio, dependiendo si la aplicación se está ejecutando en modo debugging, o reléase. Considerar interfaz que represente a estas rutinas, y posible inyección de dependencia. - Configurar herramientas de versionado.

ID 3 - Desarrollo del formulario principal

ID 3	Desarrollo del formulario principal
Como un…	Desarrollador
quiero…	Contar con un formulario principal
para…	Brindar acceso a las funcionalidades principales
Observaciones:	Algunas de las tareas previstas en esta etapa: - Confeccionar la interfaz acorde a la definición en wireframes. - El formulario principal debe ser un cliente MDI, que soporte la apertura y gestión de los formularios hijos en pestañas. - Debe contar con un menú lateral estilo “hamburguesa” para el acceso a las funcionalidades principales. - Debe emplear un menú Ribbon con la posibilidad de mezclar los comandos de la pantalla principal (cliente MDI), con los comandos de las interfaces hijas. - Debe contar con un pie, que permita mostrar información tanto del cliente MDI, como de las interfaces hijas. - Debe soportar el uso de estilos gráficos. - Debe soportar el cambio de idioma. - Establecer el vínculo entre el Splash screen, la lógica de inicio de la aplicación, y el formulario principal.

ID 4 - Generar lote de datos de prueba

ID 4	Generar lote de datos de prueba
Como un…	Desarrollador
quiero…	Tener datos de prueba disponibles en la aplicación
para…	Probar y exhibir funcionalidad asociada a entidades de datos
Observaciones:	Algunas de las tareas previstas en esta etapa: - Crear entidades de datos de prueba para Persona, Mascota, País, Provincia, Ciudad. - Implementar estrategia que permita que las entidades de datos que serán parámetros en la aplicación, tengan un mecanismo que permita que las mismas sean localizables (que soporten traducción).

	- Generar un lote de datos para las entidades de datos Persona, Mascota, País, Provincia, Ciudad. - Utilizar lógica de semillas para la base datos, de forma que al iniciar la aplicación en modo debugging, si la base datos no tiene cargadas las entidades de prueba, las inserte. Esto permitirá efectuar pruebas, y si es necesario limpiar la base de datos, al iniciar la aplicación nuevamente, los datos serán restablecidos.

ID 5 - Desarrollo de formulario de búsqueda (base)

ID 5	**Desarrollo de formulario de búsqueda (base)**
Como un…	Desarrollador
quiero…	Especializar un formulario de búsqueda
para…	Brindar funcionalidad de búsqueda para entidades de datos específicas
Observaciones:	Algunas de las tareas previstas en esta etapa: - Confeccionar la interfaz acorde a la definición en wireframes. - Desarrollar lógica que permita crear automáticamente las columnas de la grilla del buscador, en función de los atributos de una clase; es decir, utilizar un modelo de vista, para definir el buscador, de forma tal que las columnas sean mapeadas o inferidas automáticamente desde el modelo. - Desarrollar lógica que permita asociar archivos de recursos, a las propiedades de los modelos de vista del buscador, para que además de generar automáticamente las columnas de la grilla, le asigne encabezados acordes al idioma seleccionado. - Definir arquitectura asociada a los modelos de vista y archivos de recursos. - Extender el componente *Formulario de búsqueda*, y crear formularios de búsqueda para las entidades Persona y Mascota. - Brindar acceso a los buscadores de Persona y Mascota desde el menú lateral.

ID 6 - Formulario modal Aceptar-Cancelar (base)

ID 6	**Formulario modal Aceptar-Cancelar (base)**
Como un…	Desarrollador
quiero…	Especializar un formulario modal de tipo “Aceptar-Cancelar”
para…	Desarrollar solamente lógica específica, evitando la repetición, y aumentando el reúso
Observaciones:	Algunas de las tareas previstas en esta etapa: - Confeccionar la interfaz acorde a la definición en wireframes. - Planificar y realizar cambios en arquitectura para posicionar los formularios de este tipo. - Confeccionar lógica de base.

ID 7 - Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas

ID 7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas
Como un…	Desarrollador
quiero…	Ordenar las columnas de la grilla, ocultarlas y mostrarlas, y persistir esos cambios
para…	Mostrar sólo información relevante, de forma ordenada
Observaciones:	Algunas de las tareas previstas en esta etapa: - Recuperar en runtime las propiedades de un viewmodel del buscador, para saber previamente cuáles serán las columnas que lo componen. - Desarrollar lógica para gestionar el estado de visibilidad, y el orden de las columnas de la grilla del buscador. - Desarrollar una entidad de datos que permita almacenar en la base de datos, los nombres de las propiedades de un viewmodel. - Extender la entidad de datos que almacenará las propiedades de un viewmodel, para que además pueda almacenar el estado de visibilidad y orden de una columna. - Desarrollar un algoritmo que, dado el nombre de un ViewModel, extraiga los nombres de sus propiedades, y los inserte en DB. - Desarrollar un algoritmo, que se ejecute al inicio de la aplicación cuando la aplicación inicia en modo debugging, que dada una lista de ViewModels, realice el mapeo de las mimas a DB, verificando y corrigiendo consistencias. Es decir, si un ViewModel es actualizado (se añaden o remueven propiedades), el algoritmo de mapeo deberá ser capaz de mantener la consistencia de datos con la base de datos. - Agregar algoritmo al Formulario de Búsqueda Base, que recupere la información de las columnas de la base de datos, y las aplique a la grilla. - Desarrollar formulario modal Aceptar-Cancelar que permita ordenar las columnas de la grilla, y cambiar su estado de visibilidad. - Implementar lógica de cargar de datos eficiente en la grilla, para que en la misma sólo sean cargados aquellos datos que son visibles. Si se muestra una nueva columna que estaba oculta, los datos serán cargados bajo demanda. - Desarrollar lógica en el formulario de búsqueda base, para que, al iniciar el formulario, recupere de la base de datos la configuración de la grilla y la aplique. - Desarrollar lógica en el formulario de búsqueda base, para que, al cerrar el formulario, detecte si hubo cambios en la configuración de la grilla, y la persista en la base de datos.

ID 8 - Formulario de búsqueda: Configurar plantilla

ID 8	Formulario de búsqueda: Configurar plantilla
Como un…	Desarrollador
quiero…	Configurar los aspectos gráficos de una plantilla del buscador

para…	Mostrar sólo información relevante, de forma ordenada, destacándola con colores y características tipográficas
Observaciones:	Algunas de las tareas previstas en esta etapa: - Confeccionar la interfaz acorde a la definición en wireframes. - Desarrollar lógica que permita efectuar los cambios estéticos necesarios a las columnas de la grilla. - Adaptar lógica para visualizar los cambios realizados en tiempo real. - Adecuar la entidad de datos que almacena las propiedades de los viewmodel, su orden y estado de visibilidad, para almacenar también la nueva configuración estética. - Actualizar algoritmo de mapeo automático en el inicio de la aplicación. - Actualizar algoritmo de aplicación de plantilla en el formulario de búsqueda base. - Actualizar el algoritmo de guardado de cambios en las plantillas, en el formulario de búsqueda base.

ID 9 - Formulario de búsqueda: generación de salidas

ID 9	Formulario de búsqueda: generación de salidas
Como un…	Usuario
quiero…	Previsualizar, imprimir, y exportar a diferentes formatos las resultas de búsquedas
para…	Disponer de la información en otros medios
Observaciones:	Algunas de las tareas previstas en esta etapa: - Añadir al menú Ribbon del *Formulario de búsqueda* base los comandos de las salidas estipuladas según el diseño planteado en wireframes. - Añadir lógica que permita previsualizar el contenido de la grilla, antes de imprimir o exportar. - Añadir lógica que permita imprimir un reporte con el resultado de una búsqueda. - Añadir lógica que permita exportar a distintos formatos el resultado de una búsqueda.

ID 10 - Formulario ABMC (base)

ID 10	Formulario ABMC (base)
Como un…	Desarrollador
quiero…	Especializar un formulario para altas, bajas, modificaciones y consultas
para…	Brindar funcionalidad de altas, bajas, modificaciones y consultas para entidades de datos específicas
Observaciones:	Algunas de las tareas previstas en esta etapa: - Confeccionar la interfaz acorde a la definición en wireframes. - Desarrollar lógica general que permita crear una única interfaz para soportar las operaciones de ABMC.

	- Desarrollar lógica que permita implementar validaciones de forma sencilla y ordenada, dependiendo de la operación que se está realizando. - Extender el componente Formulario ABMC, y crear formularios de ABMC para las entidades Persona y Mascota. - Anexar los accesos a los formularios ABMC desde las operaciones de los formularios de búsqueda, para las entidades Persona y Mascota.

ID 11 - Formulario de búsqueda (base): soporte para múltiples plantillas

ID 11	**Formulario de búsqueda (base): soporte para múltiples plantillas**
Como un…	Desarrollador
quiero…	Tener configuraciones múltiples para un mismo buscador
para…	Adecuar la funcionalidad del buscador rápidamente a diferentes demandas
Observaciones:	Algunas de las tareas previstas en esta etapa: - Actualizar modelo de datos, y desarrollar entidades de datos que permitan soportar el uso de múltiples plantillas en cada buscador. - Actualizar el algoritmo de mapeo automático al inicio de la aplicación. - Desarrollar un algoritmo que cree plantillas con configuraciones aleatorias, automáticamente para cada buscador de la aplicación, cuando la aplicación inicie en modo debugging. Esto permitirá contar con lotes de datos para testear las funcionalidades. - Actualizar la lógica del formulario de búsqueda base que previamente funcionaba para una plantilla única.

ID 12 - Formulario de búsqueda: Seleccionar plantilla

ID 12	**Formulario de búsqueda: Seleccionar plantilla**
Como un…	Usuario
quiero…	Seleccionar distintas plantillas
para…	Para adecuar la visualización de los resultados de búsqueda
Observaciones:	Algunas de las tareas previstas en esta etapa: - Extender el *Formulario modal Aceptar-Cancelar*. - Confeccionar la interfaz acorde a la definición en wireframes. - Desarrollar lógica que permita mostrar las plantillas disponibles para el formulario de búsqueda, dependiendo del tipo de dato del viewmodel que lo representa. - Agregar al Formulario de búsqueda base, lógica que detecte si no se ha seleccionado una plantilla por defecto, y que invite a seleccionar una. Si ya hay una plantilla por defecto seleccionada, que la utilice en la apertura del formulario. - Agregar al Formulario de búsqueda base, lógica que detecte si se han realizado cambios sobre la plantilla seleccionada por defecto, y que al cerrar el formulario consulte si se desean guardar los cambios.

	- Añadir acceso al Formulario de Selección de Plantillas en el menú Ribbon del Formulario de búsqueda.

ID 13 - Formulario de búsqueda: Nueva plantilla

ID 13	Formulario de búsqueda: Nueva plantilla
Como un…	Usuario
quiero…	Confeccionar una nueva plantilla para un Formulario de búsqueda en particular
para…	Para adecuar la visualización de los resultados de búsqueda
Observaciones:	Algunas de las tareas previstas en esta etapa: - Extender el *Formulario modal Aceptar-Cancelar*. - Confeccionar la interfaz acorde a la definición en wireframes. - Desarrollar lógica que permita confeccionar plantillas para el formulario de búsqueda, dependiendo del tipo de dato del viewmodel que lo representa. - Añadir acceso al Formulario de creación de Plantillas en el menú Ribbon del Formulario de búsqueda. - Agregar al Formulario de búsqueda base, lógica que detecte si se ha creado una nueva plantilla (es decir, que el Formulario de creación de Plantillas finalizó con el botón “Aceptar”). Si se ha creado una nueva plantilla, invitar al usuario a seleccionarla.

ID 14 - Gestor de formularios abiertos y entidades de datos en uso

ID 14	Gestor de formularios abiertos y entidades de datos en uso
Como un…	Desarrollador
quiero…	Gestionar los formularios abiertos y las entidades de datos en uso
para…	Evitar la reapertura en simultaneo, de formularios con la misma responsabilidad, y gestionar las entidades de datos que estén en uso (lectura o escritura)
Observaciones:	Algunas de las tareas previstas en esta etapa: - Desarrollar gestor de entidades de datos en uso. - Desarrolla gestor de formularios en uso. - Reemplazar en el Formulario principal, cada llamada directa a la apertura de un formulario, por llamadas mediante el Gestor de formularios.

ID 15 - Formulario de configuración

ID 15	Formulario de configuración
Como un…	Desarrollador
quiero…	Un formulario de configuración
para…	Añadir aspectos de configuración específicos, evitando la repetición
Observaciones:	Algunas de las tareas previstas en esta etapa: - Extender el *Formulario modal Aceptar-Cancelar*.

	- Confeccionar la interfaz acorde a la definición en wireframes. - Desarrollar interfaz que deberá ser implementada por todos los paneles de configuración que serán mostrados a la derecha. - Añadir lógica específica del formulario. - Añadir acceso al *Formulario de configuración*, desde el *Menú principal*. - Crear entidad de datos que permita reunir información específica de aspectos configurables por el usuario (idealmente, "ParámetrosInternos").

ID 16 - Formulario de configuración – "Idioma"

ID 16	Formulario de configuración – "Idioma"
Como un…	Usuario de la aplicación
quiero…	Cambiar el idioma de la aplicación
para…	Adecuarla a las características lingüísticas y regionales que mejor se adapten a mi perfil.
Observaciones:	Algunas de las tareas previstas en esta etapa: - Crear control de usuario, que extienda la interfaz que generaliza a los paneles de configuración, del *Formulario de configuración*. - Confeccionar la interfaz acorde a la definición en wireframes. - Desarrollar entidades de datos que permitan almacenar la información de los distintos idiomas soportados por la aplicación. - Agregar las configuraciones de idiomas soportados por la aplicación, a las semillas automáticas de la base de datos. - Añadir a la entidad que almacena los "ParámetrosInternos", la posibilidad de almacenar un idioma por defecto según la aplicación (de facto), y un idioma seleccionado por el usuario. - Desarrollar algoritmo que se ejecute al inicio de la aplicación, que funcione con el siguiente criterio: 1) Si existe un idioma seleccionado por el usuario, aplicarlo. 2) Detectar idioma del Sistema Operativo, si está disponible en la aplicación, aplicarlo. 3) Aplicar idioma por defecto en la aplicación según facto.

ID 17 - Traducción de la aplicación completa al inglés

ID 17	Traducción de la aplicación completa al inglés
Como un…	Usuario
quiero…	Utilizar la aplicación en idioma inglés
para…	Adecuarla a las características lingüísticas del usuario
Observaciones:	Algunas de las tareas previstas en esta etapa: - Implementar herramienta de traducción automática, que utilice las API de alguna herramienta pública y gratuita (Google Translator, por ejemplo). - Generar traducciones para todos los archivos de recurso. - Refinar las traducciones manualmente (en los archivos de recursos).

ID 18 - Formulario de configuración – "Apariencia"

ID 18	Formulario de configuración – "Apariencia"
Como un…	Usuario de la aplicación
quiero…	Cambiar los estilos gráficos de la aplicación
para…	Adecuar la experiencia de usuario a mi perfil
Observaciones:	Algunas de las tareas previstas en esta etapa: - Crear control de usuario, que extienda la interfaz que generaliza a los paneles de configuración, del *Formulario de configuración*. - Confeccionar la interfaz acorde a la definición en wireframes. - Desarrollar entidades de datos que permitan almacenar la información de los distintos aspectos estéticos (estilos y paletas de colores) soportados por la aplicación. - Añadir a la entidad que almacena los "ParámetrosInternos", la posibilidad de almacenar un esquema gráfico y una paleta de colores por defecto según la aplicación (de facto), y un esquema gráfico y una paleta de colores seleccionado por el usuario. - Desarrollar algoritmo que se ejecute al inicio de la aplicación, que funcione con el siguiente criterio: 1) Si existe un esquema gráfico y/o paleta de colores seleccionado por el usuario, aplicarlo. 2) Aplicar esquema gráfico y/o paleta de colores por defecto en la aplicación según facto.

9 ANEXO C – Gestión de riesgos

9.1 Introducción

El riesgo se define como un evento incierto que puede afectar los objetivos de un proyecto y puede contribuir a su éxito o fracaso.

Los riesgos con potencial de impacto positivo en el proyecto se denominan oportunidades, mientras que las amenazas son riesgos que podrían afectar negativamente a un proyecto.

La gestión del riesgo debe realizarse de forma proactiva y es un proceso iterativo que debe comenzar al inicio del proyecto y continuar durante toda la vida del proyecto.

El proceso de gestión de riesgos debe seguir algunos pasos estandarizados para garantizar que los riesgos se identifiquen, evalúen y se determine un curso de acción adecuado y se actúe en consecuencia.

Los riesgos deben ser identificados, evaluados y tratados con base principalmente en dos factores: la probabilidad de que ocurra y el impacto probable en caso de que ocurra. Los riesgos con alta probabilidad y alta calificación de impacto deben abordarse antes que aquellos con una calificación más baja.

En general, una vez que se identifica un riesgo, es importante comprender los aspectos básicos del riesgo con respecto a las posibles causas, el área de incertidumbre y los efectos potenciales si se produce el riesgo.

9.2 El riesgo en Scrum

En un entorno Scrum, los riesgos generalmente se minimizan, en gran parte debido al trabajo que se realiza en los *Sprints*, mediante el cual se produce una serie continua de entregables

en ciclos muy cortos. Los entregables se comparan con las expectativas y el propietario del producto participa activamente en el proyecto.

Como cocreador de Scrum, Ken Schwaber dice: *"Scrum es una forma de controlar el riesgo"*. Schwaber propone "abrazar" la verdadera esencia de lo ágil, que es tener un enfoque innovador para hacer frente a los riesgos.

Por otro lado, muchos "agilistas" consideran que Scrum, no puede ayudar con los riesgos externos al proyecto. Puede ayudar con los riesgos relacionados con el cambio en los requisitos, la falta de comunicación y el bajo rendimiento de los miembros del equipo. Sin embargo, los riesgos que son externos al proyecto necesitan más que Scrum para resolverse.

Por esta razón, el alumno adoptará una postura atenta a ambas perspectivas. "Abrazar" el cambio, y mantener un enfoque innovador frente a los riesgos para generar oportunidades desde los mismos, y también, proceder con un enfoque estructurado para gestionar el riesgo.

9.3 Estrategia para la identificación de riesgos

"Existen muchas estrategias para determinar peligros que abarcan en forma más o menos estructurada dicha determinación pero que persiguen el mismo objetivo: la elicitación de los riegos." [Bracalenti-2009].

El equipo Westfall propone una lista de técnicas para identificar riesgos, que incluyen entrevistas, informes, descomposición, análisis de supuestos, análisis de ruta crítica y la utilización de taxonomías de riesgo.

Luego, como ejemplo de una taxonomía de riesgos, Westfall menciona al Reporte de Identificación de Riesgos basada en Taxonomía ("Taxonomy-Based Risk Identification report"), del Software Engineering Institute, que cubre 13 áreas de riesgo principales, con aproximadamente 200 preguntas [SEI-93].

Finalmente, este último es el enfoque que utilizará el alumno como enfoque estructurado para identificar riesgos.

9.4 Identificación de riesgos

El método TBQ ("Taxonomy-Based Questionnaire"), del SEI (Software Engineering Institute), se descompone en tres clases principales:

1. Ingeniería del producto. Los aspectos técnicos del trabajo a realizar (Requerimiento, Diseño, Código y pruebas de unitarias, Integración y prueba, Especialidades de ingeniería).
2. Entorno de desarrollo. Los métodos, procedimientos y herramientas utilizadas para fabricar el producto (Proceso de desarrollo, Sistema de desarrollo, Proceso de gestión, Método de gestión, Entorno de trabajo).
3. Restricciones del programa. Los factores contractuales, organizativos y operacionales dentro de los cuales se desarrolla el software (Recursos, Contratos, Interfaz del programa).

9.4.1 Ingeniería del producto

Elemento	Atributo	Id	Riesgo
Requerimiento	Precedente	1	Los requerimientos especifican algo que el alumno no ha hecho antes
Diseño	Dificultad	2	El diseño y/o la implementación de algunos componentes pude ser difícil de lograr.
	Testeo	3	Algunos componentes abstractos que definan características específicas de comportamiento (por ejemplo, en interfaz de usuario) pueden ser difíciles de probar.
Código y pruebas unitarias	Factibilidad	4	Algunos componentes pueden ser difíciles de implementar.
	Testeo	5	Las pruebas unitarias serán desarrolladas y ejecutadas luego del diseño y codificación de algunos componentes, dadas restricciones intrínsecas a la tecnología utilizada, y al desconocimiento inicial de la arquitectura final.

9.4.2 Entorno de desarrollo

Elemento	Atributo	Id	Riesgo
Sistema de desarrollo	Familiaridad	6	Algunas herramientas en utilizadas en el proceso de desarrollo, serán nuevas para el alumno (por ejemplo, paquete de controles de usuario).

9.4.3 Restricciones del programa

Elemento	Atributo	Id	Riesgo
Contrato	Restricciones	7	El producto a desarrollar será un producto comercial. El alumno firmó un contrato de confidencialidad con la empresa. Algunos componentes podrían necesitar ser especificados con menor nivel de detalle o parcialmente especificados (en caso de interacciones específicas con escenarios del cliente).

9.5 Estrategia para la evaluación de los riesgos identificados

La evaluación del riesgo ayuda a comprender el impacto potencial de un riesgo, la probabilidad de que ocurra y cuándo el mismo podría materializarse. Se debe estimar el efecto

general sobre el valor del proyecto; si ese impacto es lo suficientemente significativo como para superar al beneficio, se debe tomar una decisión sobre la continuación del proyecto.

La evaluación de riesgos se realiza con respecto a *probabilidad*, *proximidad* e *impacto*. La probabilidad de riesgos se refiere a la probabilidad de que ocurran los riesgos, mientras que la proximidad se refiere a cuándo podría ocurrir el riesgo. El impacto se refiere al efecto probable de los riesgos en el proyecto o la organización.

Para estimar la probabilidad de un riesgo, se pueden utilizar varias técnicas. En el presente proyecto, se empleará la ecuación cuantitativa con la cual Boehm define el grado de exposición al riesgo [Bohem-89]. Ésta mide el impacto de un riesgo en términos del valor esperado de la pérdida, según:

$$RE\ (Exposición\ al\ riesgo) = Probabilidad\ de\ ocurrencia * Pérdida\ esperada$$

Asignar valores a la ecuación de *Exposición al riesgo* y organizarlos en una grilla comparativa, permitirá establecer un ranking de prioridades para la adopción de contramedidas.

Dado que el alumno no cuenta con experiencia para otorgar valores a la *Pérdida esperada* en la ecuación de *Exposición al riesgo*, se utilizará el criterio propuesto por [Bracalenti-2009], *"En caso de no poder estimarse los costos del daño a ciencia cierta se lo puede sustituir por el grado de impacto en términos cualitativos."*.

Para cuantificar la *Pérdida esperada*, se utilizará una categorización cualitativa propuesta por [Boehm-89], y el alumno otorgará un peso en tanto por uno a cada categoría:

Impacto de la pérdida	Peso
Catastrófico	1
Crítico	0.75
Marginal	0.50
Despreciable	0.25

Luego, continuando con la estrategia utilizada para la *Pérdida esperada*, el alumno confecciona una tabla para la *Probabilidad de ocurrencia*:

Probabilidad de ocurrencia	Peso
Alto	1
Medio	0.75
Bajo	0.50
Mínimo	0.25

Si se conjugan en una grilla, la *Probabilidad de ocurrencia* y la *Pérdida esperada*, ordenados por su peso, *"es posible definir una línea de corte"* [Pressman-04], de modo que el esfuerzo se concentre en los riesgos que están por sobre ésta, implica que sólo los riesgos que se encuentran por arriba de la línea recibirán mayor atención. *"Es decir que los eventos improbables y de impacto marginal o despreciable pueden ser dejados de lado"* [Pressman-04].

La nueva grilla contempla un valor más de referencia para el análisis y seguimiento del riesgo. El alumno traza la línea de corte propuesta por Pressman en el intervalo [0 ; 0,25], y añade una nomenclatura para cada intervalo generado:

		Probabilidad de ocurrencia			
		0,25	**0,5**	**0,75**	**1**
Impacto de la pérdida	**0,25**	0,0625	0,125	0,1875	0,25
	0,5	0,125	0,25	0,375	0,5
	0,75	0,1875	0,375	0,5625	0,75
	1	0,25	0,5	0,75	1

Exposición de riesgo	
Aceptable	[0 ; 0,25]
Poco riesgoso	]0,25 ; 0,5]
Riesgoso	]0,5 ; 0,75]
Muy riesgoso	]0,75 ; 1]

9.6 Evaluación de los riesgos identificados

Id	Descripción	Probabilidad	Impacto	ER
1	Los requerimientos especifican algo que el alumno no ha hecho antes.	Medio (0,75)	Crítico (0,75)	Riesgoso (0,5625)
2	El diseño y/o la implementación de algunos componentes pude ser difícil de lograr.	Baja (0,5)	Marginal (0,5)	Aceptable (0,25)
3	Algunos componentes abstractos que definan características específicas de comportamiento (por ejemplo, en interfaz de usuario) pueden ser difíciles de probar.	Baja (0,50)	Despreciable (0,25)	Aceptable (0,125)
4	Algunos componentes pueden ser difíciles de implementar.	Baja (0,5)	Marginal (0,5)	Aceptable (0,25)
5	Las pruebas unitarias serán desarrolladas y ejecutadas luego del diseño y codificación de algunos componentes, dadas restricciones intrínsecas a la tecnología utilizada, y al desconocimiento inicial de la arquitectura final.	Mínima (0,25)	Despreciable (0,25)	Aceptable (0,0625)
6	Algunas herramientas en utilizadas en el proceso de desarrollo, serán nuevas para el alumno (por ejemplo, paquete de controles de usuario).	Baja (0,5)	Marginal (0,5)	Aceptable (0,25)
7	El producto a desarrollar será un producto comercial. El alumno firmó un contrato de confidencialidad con la empresa. Algunos componentes podrían necesitar ser especificados con menor nivel de detalle o parcialmente especificados (en caso de interacciones específicas con escenarios del cliente).	Mínima (0,25)	Despreciable (0,25)	Aceptable (0,0625)

9.7 Disparadores y planes de contingencia

Se determinarán acciones para aquellos riesgos que tienen un valor de Exposición al riesgo ponderador dentro del intervalo]0,25 ; 1] (*Poco riesgoso*, *Riesgoso* y *Muy riesgoso*). Los riesgos dentro del intervalo [0 ; 0,25] (*Aceptables*) serán tolerados inicialmente, *"con la debida conciencia"*

[Bracalenti-2009], pero controlados por si adquieren mayor relevancia en el transcurso de las iteraciones, y por lo tanto elegibles para ser contemplados en el *Plan de contingencia*.

Id	Descripción	Disparadores y acciones
1	Los requerimientos especifican algo que el alumno no ha hecho antes.	Disparador: cada vez que el alumno (Desarrollador único) reconozca en los requerimientos, un desafío técnico de forma temprana.
		* Es importante destacar que Scrum en sí mismo, provee mecanismos para hacer frente a este tipo de situaciones, como, por ejemplo, profundizar el estudio en el área implicada utilizando la técnica *Timebox*, para aumentar el conocimiento, y replanificar acordemente. - Acción preventiva: gestionar el requerimiento con especial atención, dividiéndola en unidades menores -en tanto sea posible-, para aumentar el nivel de detalle en sus estimaciones. - Acción correctiva: informar al *ProductOwner* para desarrollar acciones en conjunto (también, parte de la gestión de riesgo proporcionada por Scrum).

10 ANEXO D – Ejecución del proyecto

Introducción

Con la finalidad de estandarizar la estructura del presente documento, facilitar la lectura, y brindar un hilo conductor, se optó por presentar en capítulos independientes lo relacionado a la gestión del proyecto y ejecución de la metodología de desarrollo (planificación, resultados de ejecución de la misma, gestión de riesgos, etc.) y, la documentación generada en la realización de los *Users Stories* que no esté asociada con la gestión del proyecto.

En el presente capítulo no se harán menciones a aspectos técnicos del desarrollo asociados al software, tales como documentación del software generada en el proceso, arquitectura, etc. Estos aspectos fueron dispuestos en un anexo individual, *Documentación generada en la realización de las User Stories*.

Para representar el resultado de la ejecución de la metodología de desarrollo, en términos de la gestión del proyecto, se definió previamente una estructura de representación, aplicable a cada *Sprint* que sea ejecutado, para documentar el proceso de forma estandarizada.

El esquema de plantillas definido consta de los siguientes elementos:

- Planificación: presentará las fechas programadas para la ejecución del *Sprint* acorde al *Cronograma de avance del proyecto*, las fechas programadas para los eventos del *Sprint*, el *Sprint Backlog* de acuerdo a lo programado en el *Product Backlog inicial* en caso de no haber habido cambios en la planificación, o el *Sprint Backlog* actualizado según cambios en la planificación.

- Ejecución: presentará un contraste entre lo planificado, y el resultado en la ejecución, abordando el *Periodo de ejecución*, el *Tiempo (días hábiles)*, las *User Stories (IDs)*, y el *Esfuerzo (Story Points)* entre otros. Además, se hará mención de cualquier información relevante como resultado de la ejecución del *Sprint*.
- Gestión de riesgos: presentará los riesgos que se hayan activado dentro del *Sprint* (contemplados, y no contemplados), y el resultado de la ejecución del plan adecuado; en caso de detectarse riesgos no contemplados, se incluirá también el desarrollo del plan acorde.
- Actualización de Product Backlog: finalizado el *Sprint*, serán removidas del *Product Backlog* aquellas *User Stories* concretadas.

Sprint Nº 1

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 1* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
1	19/10/2020	23/10/2020	1
2	26/10/2020	30/10/2020	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**1**	19/10/2020	30/10/2020		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo*, el *Sprint Backlog* para el *Sprint Nº 1* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
1	Selección de herramientas esenciales del mercado	Alta	10	1

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 1*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	19/10/2020 - 30/10/2020	19/10/2020 - 30/10/2020
Tiempo (días hábiles)	10	10
User Stories (IDs)	- 1: 10 [Story Point]	- 1: 10 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 1* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado. Tampoco se detectaron riesgos no contemplados, utilizando el enfoque ágil.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 1*, se procede a remover del *Product Backlog inicial* las *User Stories* concretadas, resultando:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
2	Desarrollo de la arquitectura inicial	Alta	4	2
3	Desarrollo del formulario principal	Alta	9	2 y 3
4	Generar lote de datos de prueba	Alta	1	3
5	Desarrollo de formulario de búsqueda (base)	Media	9	3 y 4
6	Formulario modal Aceptar-Cancelar (base)	Media	2	4
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas	Media	8	4 y 5
8	Formulario de búsqueda: Configurar plantilla	Media	7	5
9	Formulario de búsqueda: generación de salidas	Media	3	6
10	Formulario ABMC (base)	Media	10	6 y 7
11	Formulario de búsqueda (base): soporte para múltiples plantillas	Media	10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso	Media	3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Sprint Nº 2

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 2* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
3	02/11/2020	06/11/2020	2
4	09/11/2020	13/11/2020	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**2**	02/11/2020	13/11/2020		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo* -y además considerando que *Sprint Backlog* del *Sprint Nº 1* se concretó acorde a lo planificado lo cual no genera cambios en el *Sprint* posterior- el *Sprint Backlog* para el *Sprint Nº 2* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
2	Desarrollo de la arquitectura inicial	Alta	4	2
3	Desarrollo del formulario principal [inicio]	Alta	6 de 9	2 y 3

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 2*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	02/11/2020 - 13/11/2020	02/11/2020 - 13/11/2020
Tiempo (días hábiles)	10	10
User Stories (IDs)	- 2: 4 [Story Point] - 3: 6 de 9 [Story Point]	- 2: 4 [Story Point] - 3: 6 de 9 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 2* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado.

Surgió un riesgo no contemplado, al momento de concretar la adquisición de la licencia del paquete de controles de DevExpress, debido a restricciones cambiarias (el producto es comercializado por una empresa extranjera).

Se pudo continuar con el *Sprint* según lo planificado con una versión de pruebas, no habiéndose generado demoras en la planificación.

Antes de la finalización del *Sprint* la empresa solicitante pudo concretar la adquisición del producto.

La solución del problema fue transparente al alumno. La administración de la empresa resolvió el inconveniente.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 2*, se procede a remover del *Product Backlog actualizado del Sprint Nº 1* las *User Stories* concretadas, resultando:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
3	Desarrollo del formulario principal [fin]	Alta	3 de 9	2 y 3
4	Generar lote de datos de prueba	Alta	1	3
5	Desarrollo de formulario de búsqueda (base)	Media	9	3 y 4
6	Formulario modal Aceptar-Cancelar (base)	Media	2	4
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas	Media	8	4 y 5
8	Formulario de búsqueda: Configurar plantilla	Media	7	5
9	Formulario de búsqueda: generación de salidas	Media	3	6
10	Formulario ABMC (base)	Media	10	6 y 7
11	Formulario de búsqueda (base): soporte para múltiples plantillas	Media	10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso	Media	3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Sprint Nº 3

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 3* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
5	16/11/2020	20/11/2020	3
6	24/11/2020	30/11/2020	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**3**	16/11/2020	30/11/2020		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo* -y además considerando que *Sprint Backlog* del *Sprint Nº 2* se concretó acorde a lo planificado lo cual no genera cambios en el *Sprint* posterior- el *Sprint Backlog* para el *Sprint Nº 3* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
3	Desarrollo del formulario principal [fin]	Alta	3 de 9	2 y 3
4	Generar lote de datos de prueba	Alta	1	3
5	Desarrollo de formulario de búsqueda (base) [inicio]	Media	6 de 9	3 y 4

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 3*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	16/11/2020 - 30/11/2020	16/11/2020 - 30/11/2020
Tiempo (días hábiles)	10	10
User Stories (IDs)	- 3: 3 de 9 [Story Point] - 4: 1 [Story Point] - 5: 6 de 9 [Story Point]	- 3: 3 de 9 [Story Point] - 4: 1 [Story Point] - 5: 6 de 9 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 3* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado. Tampoco se detectaron riesgos no contemplados, utilizando el enfoque ágil.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 3*, se procede a remover del *Product Backlog actualizado del Sprint Nº 2* las *User Stories* concretadas, resultando:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
5	Desarrollo de formulario de búsqueda (base) [fin]	Media	3 de 9	3 y 4
6	Formulario modal Aceptar-Cancelar (base)	Media	2	4
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas	Media	8	4 y 5
8	Formulario de búsqueda: Configurar plantilla	Media	7	5
9	Formulario de búsqueda: generación de salidas	Media	3	6
10	Formulario ABMC (base)	Media	10	6 y 7
11	Formulario de búsqueda (base): soporte para múltiples plantillas	Media	10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso	Media	3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Sprint Nº 4

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 4* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
7	01/12/2020	09/12/2020	4
8	10/12/2020	16/12/2020	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**4**	01/12/2020	16/12/2020		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo* -y además considerando que *Sprint Backlog* del *Sprint Nº 3* se concretó acorde a lo planificado lo cual no genera cambios en el *Sprint* posterior- el *Sprint Backlog* para el *Sprint Nº 4* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
5	Desarrollo de formulario de búsqueda (base) [fin]	Media	3 de 9	3 y 4
6	Formulario modal Aceptar-Cancelar (base)	Media	2	4
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas [inicio]	Media	5 de 8	4 y 5

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 4*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	01/12/2020 - 16/12/2020	01/12/2020 - 16/12/2020
Tiempo (días hábiles)	10	10
User Stories (IDs)	- 5: 3 de 9 [Story Point] - 6: 2 [Story Point] - 7: 5 de 8 [Story Point]	- 5: 3 de 9 [Story Point] - 6: 2 [Story Point] - 7: 5 de 8 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 4* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado. Tampoco se detectaron riesgos no contemplados, utilizando el enfoque ágil.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 4*, se procede a remover del *Product Backlog actualizado del Sprint Nº 3* las *User Stories* concretadas, resultando:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas [fin]	Media	3 de 8	4 y 5
8	Formulario de búsqueda: Configurar plantilla	Media	7	5
9	Formulario de búsqueda: generación de salidas	Media	3	6
10	Formulario ABMC (base)	Media	10	6 y 7
11	Formulario de búsqueda (base): soporte para múltiples plantillas	Media	10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso	Media	3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Sprint Nº 5

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 5* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
9	17/12/2020	23/12/2020	5
10	24/12/2020	31/12/2020	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**5**	17/12/2020	31/12/2020		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo* -y además considerando que *Sprint Backlog* del *Sprint Nº 4* se concretó acorde a lo planificado lo cual no genera cambios en el *Sprint* posterior- el *Sprint Backlog* para el *Sprint Nº 5* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
7	Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas [fin]	Media	3 de 8	4 y 5
8	Formulario de búsqueda: Configurar plantilla	Media	7	5

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 5*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	17/12/2020 - 31/12/2020	17/12/2020 - 31/12/2020
Tiempo (días hábiles)	10	10
User Stories (IDs)	- 7: 3 de 9 [Story Point] - 8: 2 [Story Point]	- 7: 3 de 9 [Story Point] - 8: 2 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 5* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado. Tampoco se detectaron riesgos no contemplados, utilizando el enfoque ágil.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 5*, se procede a remover del *Product Backlog actualizado del Sprint Nº 4* las *User Stories* concretadas, resultando:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
9	Formulario de búsqueda: generación de salidas	Media	3	6
10	Formulario ABMC (base)	Media	10	6 y 7
11	Formulario de búsqueda (base): soporte para múltiples plantillas	Media	10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso	Media	3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Sprint Nº 6

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 6* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
11	04/01/2021	08/01/2021	6
12	11/01/2021	15/01/2021	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**6**	04/01/2021	15/01/2021		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo* -y además considerando que *Sprint Backlog* del *Sprint Nº 5* se concretó acorde a lo planificado lo cual no genera cambios en el *Sprint* posterior- el *Sprint Backlog* para el *Sprint Nº 6* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
9	Formulario de búsqueda: generación de salidas	Media	3	6
10	Formulario ABMC (base) [inicio]	Media	7 de 10	6 y 7

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 6*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	04/01/2021 - 15/01/2021	04/01/2021 - 15/01/2021
Tiempo (días hábiles)	10	10
User Stories (IDs)	- 9: 3 [Story Point] - 10: 7 de 10 [Story Point]	- 9: 3 [Story Point] - 10: 7 de 10 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 6* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado. Tampoco se detectaron riesgos no contemplados, utilizando el enfoque ágil.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 6*, se procede a remover del *Product Backlog actualizado del Sprint Nº 5* las *User Stories* concretadas, resultando:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
10	Formulario ABMC (base) [fin]	Media	3 de 10	6 y 7
11	Formulario de búsqueda (base): soporte para múltiples plantillas	Media	10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso	Media	3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Sprint Nº 7

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 7* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
13	18/01/2021	22/01/2021	7
14	25/01/2021	29/01/2021	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**7**	18/01/2021	29/01/2021		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo* -y además considerando que *Sprint Backlog* del *Sprint Nº 6* se concretó acorde a lo planificado lo cual no genera cambios en el *Sprint* posterior- el *Sprint Backlog* para el *Sprint Nº 7* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
10	Formulario ABMC (base) [inicio]	Media	3 de 10	6 y 7
11	Formulario de búsqueda (base): soporte para múltiples plantillas [fin]	Media	7 de 10	7 y 8

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 7*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	18/01/2021 - 29/01/2021	18/01/2021 - 29/01/2021
Tiempo (días hábiles)	10	10
User Stories (IDs)	- 10: 3 de 10 [Story Point] - 11: 7 de 10 [Story Point]	- 10: 3 de 10 [Story Point] - 11: 7 de 10 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 7* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado. Tampoco se detectaron riesgos no contemplados, utilizando el enfoque ágil.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 7*, se procede a remover del *Product Backlog actualizado del Sprint Nº 6* las *User Stories* concretadas, resultando:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
11	Formulario de búsqueda (base): soporte para múltiples plantillas [fin]	Media	3 de 10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso	Media	3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Sprint Nº 8

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 8* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
15	25/01/2021	29/01/2021	8
16	01/02/2021	05/02/2021	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**8**	25/01/2021	05/02/2021		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo* - y además considerando que *Sprint Backlog* del *Sprint Nº 7* se concretó acorde a lo planificado lo cual no genera cambios en el *Sprint* posterior - el *Sprint Backlog* para el *Sprint Nº 8* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
11	Formulario de búsqueda (base): soporte para múltiples plantillas [fin]	Media	3 de 10	7 y 8
12	Formulario de búsqueda: Seleccionar plantilla	Media	3	8
13	Formulario de búsqueda: Nueva plantilla	Media	3	8
14	Gestor de formularios abiertos y entidades de datos en uso [inicio]	Media	1 de 3	8 y 9

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 8*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	25/01/2021 - 05/02/2021	25/01/2021 - 05/02/2021
Tiempo (días hábiles)	10	10
User Stories (IDs)	- 11: 3 de 10 [Story Point] - 12: 3 [Story Point] - 13: 3 [Story Point]	- 11: 3 de 10 [Story Point] - 12: 3 [Story Point] - 13: 3 [Story Point]

	- 14: 2 de 3 [Story Point]	- 14: 2 de 3 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 8* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado. Tampoco se detectaron riesgos no contemplados, utilizando el enfoque ágil.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 8*, se procede a remover del *Product Backlog actualizado del Sprint Nº 7* las *User Stories* concretadas, resultando:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
14	Gestor de formularios abiertos y entidades de datos en uso [fin]	Media	2 de 3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Sprint Nº 9

Planificación

En el *Cronograma de avance del proyecto*, se definió que el *Sprint Nº 9* sería ejecutado acorde a la siguiente definición de fechas:

Semana de trabajo			Sprint Nº
Nº	Desde	Hasta	
17	08/02/2021	12/02/2021	9
18	15/02/2021	19/02/2021	

Al ubicar los eventos sobre el cronograma, tomando como base la definición de los *Eventos* en la metodología, los mismos resultaron programados en las siguientes fechas:

		Eventos de la metodología			
		Sprint Planning	**Preparación de presentación para el Sprint Review**	**Sprint Review**	**Sprint Retrospective**
Sprint	**9**	08/02/2021	19/02/2021		

De acuerdo al *Product Backlog inicial*, en el cual se compilaron el *Orden de ejecución de las User Stories*, con la *Estimación del esfuerzo* -y además considerando que *Sprint Backlog* del *Sprint Nº 8* se concretó acorde a lo planificado lo cual no genera cambios en el *Sprint* posterior- el *Sprint Backlog* para el *Sprint Nº 9* fue planificado según:

ID	Nombre	Prioridad en el desarrollo	Estimación [Story point]	Sprint
14	Gestor de formularios abiertos y entidades de datos en uso [fin]	Media	2 de 3	8 y 9
15	Formulario de configuración	Baja	2	9
16	Formulario de configuración – “Idioma”	Baja	2	9
17	Traducción de la aplicación completa al inglés	Baja	1	9
18	Formulario de configuración – “Apariencia”	Baja	2	9

Ejecución

A continuación, se contrasta la planificación del *Sprint Nº 9*, con el resultado de su ejecución:

	Planificado	Realizado
Periodo de ejecución	08/02/2021 - 19/02/2021	25/01/2021 - 05/02/2021
Tiempo (días hábiles)	9	9
User Stories (IDs)	- 14: 2 de 3 [Story Point] - 15: 2 [Story Point] - 16: 2 [Story Point]	- 14: 2 de 3 [Story Point] - 15: 2 [Story Point] - 16: 2 [Story Point]

	- 17: 1 [Story Point] - 18: 2 [Story Point]	- 17: 1 [Story Point] - 18: 2 [Story Point]
Esfuerzo (Story Points)	10	10

La ejecución resultó como fue planificada, en el periodo estipulado, concretando las *User Stories* dispuestas, y resultando vacío el *Sprint Backlog* asociado vacío. Finalmente, no se encontraron desvíos en la planificación.

Gestión de Riesgos

Durante la ejecución del *Sprint Nº 9* no se activaron ninguno de los disparadores de los riesgos identificados en la *Identificación de riesgos*, del *Plan de gestión de riesgos*, mediante el enfoque estructurado. Tampoco se detectaron riesgos no contemplados, utilizando el enfoque ágil.

Actualización de Product Backlog

Habiéndose finalizado el *Sprint Nº 9*, se procede a remover del *Product Backlog actualizado del Sprint Nº 8* las *User Stories* concretadas, resultando vacío el Product Backlog.

11 ANEXO E – Documentación generada en la realización de las User Stories

Introducción

Con la finalidad de estandarizar la estructura del presente documento, facilitar la lectura, y brindar un hilo conductor, se optó por presentar en capítulos independientes lo relacionado a la gestión del proyecto y ejecución de la metodología de desarrollo propuesta por el tesista (planificación, resultados de ejecución de la misma, gestión de riesgos, etc.) y, la documentación generada en la realización de los *Users Stories* que no esté asociada con la gestión del proyecto.

El presente capítulo será organizado a través de la identificación de los *User Stories*, en el orden cronológico en el que se sucedan.

La documentación contenida para cada *User Story* puede ser de diversos tipos, dependiendo de la finalidad del *User Story*. Por ejemplo, de tratarse de avances en la arquitectura del software, podrían encontrarse diagramas estructurales que la representen, de tratarse de la implementación de una lógica concreta, encontrar entonces diagramas de comportamiento o pseudocódigo que la

expliquen, pero también podría disponerse información particular sobre el uso de alguna herramienta interna específica (una librería, un complemento, etc.), o la razón de ser de un determinado criterio adoptado.

Podría entenderse a la documentación de las *User Stories* como bitácoras con información relevante tanto para el alumno (*Desarrollador único*), como para Stakeholders que tienen visión sobre el proyecto. En el futuro, la información allí contenida, podría formar parte de un documento de arquitectura, de un manual de procedimientos para los desarrolladores, etc.

User Story ID 1 - Selección de herramientas esenciales del mercado

Introducción

La selección de un paquete de controles de usuario para desarrollar la aplicación, tiene lugar como primer *User Story* del proyecto. Esto no es trivial, dado que cada paquete de controles de usuario adopta una forma de trabajo particular respecto a los componentes que lo integran, y no hay en el mercado una estandarización de los mismos.

Esto significa, que no es posible reemplazar un paquete de controles de usuario por otro cuando el desarrollo está avanzado, al menos no de forma sencilla. El acoplamiento entre los componentes de un paquete de controles de usuario y la aplicación desarrollada en base a ellos es alta.

Por otro lado, no existe en el mercado un conjunto de interfaces que estandaricen su comportamiento, lo que permitiría el intercambio entre distintas soluciones. Y si bien, la tecnología de base (*Windows Forms*) brinda componentes esenciales para el desarrollo, los mismos son simplistas en términos de funcionalidades que proveen, interfaz gráfica, y flexibilidad al momento de ser configurados.

Se remarca entonces, la dependencia entre el éxito, la continuidad y evolución del proyecto, y la elección realizada en esta etapa.

Criterios de selección

En la planificación del *User Story* se propusieron para el análisis, cinco paquetes de controles de usuarios. Estos paquetes no fueron seleccionados al azar, se trata de soluciones líderes del mercado, las cuales incluso, han recibido menciones positivas por parte de Microsoft[22].

Para poder efectuar una comparación crítica y cuantificable entre los distintos paquetes de controles de usuario, de antemano se habían propuesto los siguientes elementos de análisis:

- Disponibilidad de controles de usuario que de antemano serán necesarios, acorde a las definiciones del *Modelo conceptual de la aplicación*:
  - Soporte de estilos gráficos (requerido).
  - Herramienta para la creación y modificación de estilos gráficos (deseable).
  - Soporte de archivos de recursos (para la globalización) en los controles de usuario (requerido).
  - Paquetes de idioma prefabricados para los controles de usuario (deseable).
  - Menú Ribbon (requerido).
  - Splash Screen (deseable).
  - Menú acordeón estilo hamburguesa (requerido).

[22] La tecnología con la cual el presente proyecto será realiza, está recibiendo actualizaciones importantes en este último tiempo. Microsoft ha decidido trabajar colaborativamente con las empresas líderes del mercado que fabrican componentes anexos para sus tecnologías. En un artículo reciente, en el cual se mencionan los aspectos evolutivos de la tecnología con la cual será desarrollado el presente proyecto, Microsoft menciona cuáles empresas están trabajando con ellos: https://devblogs.microsoft.com/dotnet/windows-forms-designer-for-net-core-released/

  - Formulario MDI padre que permita mostrar los formularios hijos como pestañas, y además desacoplarlos para trabajar en ellos como formularios individuales (requerido).
  - Grilla que permita agrupar las cabeceras de las columnas, filtrar datos por columnas, y generar reportes con sus datos (requerido).
  - Lista desplegable que permita seleccionar colores desde una paleta (requerido).
  - Lista desplegable que permita incorporar un buscador en ella (requerido).
  - Lista desplegable que permita mostrar sus ítems en formato de columnas (requerido).
  - Herramienta de generación de reportes (deseable)
- Soporte a .Net Core: el soporte actual, o actualización a futuro hacia .Net Core es fundamental para garantizar la evolución tecnológica del proyecto.
- Calidad de la documentación y recursos para el aprendizaje: será considerado el nivel de detalle de las explicaciones, si hay o no ejemplos disponibles, la claridad en los ejemplos, y la facilidad de uso de los manuales.
- Foro, comunidad, u otro tipo de soporte oficial: de existir, se evaluará la velocidad de respuesta (realizando una pregunta siempre al mismo horario, para cada paquete), y si las respuestas son de personal de la empresa, o de integrantes externos de la comunidad.
- Galería con presentación de cada control de usuario del paquete: este tipo de presentaciones del producto son útiles para conocer rápidamente cuáles son los controles de usuarios disponibles, con ejemplos de su implementación en un escenario simplificado, y con ejemplos de código asociado; de existir, será contemplado como un accesorio importante.
- Conocimiento por parte del *Desarrollador único* (el alumno): el conocimiento previo de las soluciones, y el contar con un concepto previo de las mismas, representa una ventaja al acortar la curva de aprendizaje de las mismas. Si bien, no se considerará este aspecto como un motivo de descarte, puede ser útil al momento de generar, por ejemplo, un "desempate", frente a soluciones ganadoras que hayan alcanzado una misma valoración.

El precio de las soluciones no formó parte del criterio de selección. Todas las soluciones que formaron parte del listado en la *User Story* tenían costos evaluados previamente, y que estaban dentro de los valores aceptables por la empresa solicitante.

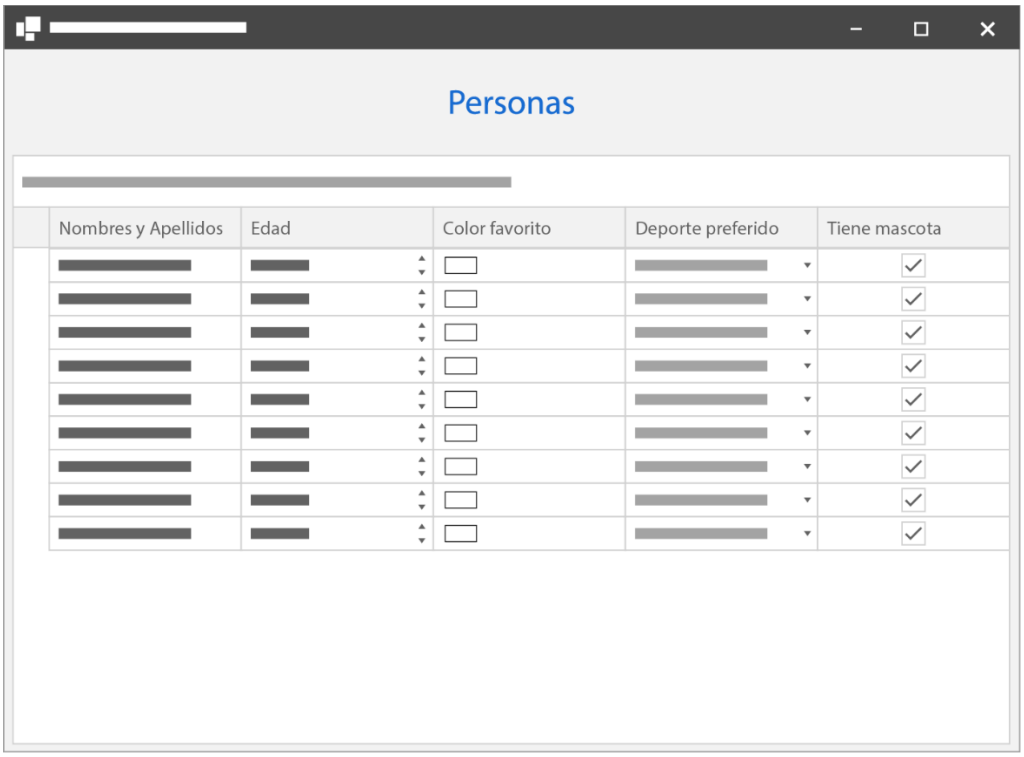
Además de los elementos de análisis mencionados previamente, con cada paquete de controles de usuario, se realizó una aplicación sencilla —a la cuál en adelante llamaremos *"aplicación de contacto"*—, que consistía en realizar altas, bajas, modificaciones y eliminaciones a través de la grilla.

La propuesta tenía como objetivo establecer un escenario mínimo en el cual probar funcionalidades con iguales características, y bajo las mismas exigencias, lo cual permitiría no solo testear las capacidades de las mismas, sino que también brindar una idea del tiempo requerido para su realización, con cada paquete de controles de usuario.

Las preguntas del cuestionario inicial serían siempre respondidas al comienzo del análisis de cada paquete de controles de usuario, la *aplicación de contacto* sería desarrollada luego, empleando el tiempo restante para el paquete de controles de usuario en análisis. Al finalizar la aplicación de contacto, la información resultante sería añadida al cuestionario inicial, bajo una ponderación personal con el nombre "Experiencia general en el uso".

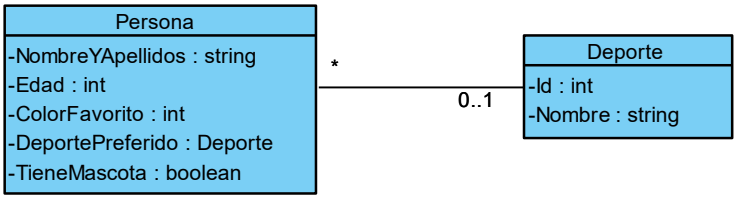
La *aplicación de contacto*, se planteó en torno a la grilla, dado que, de todos los controles de usuario requeridos, era posiblemente el que recibirá mayor demanda, y del cual dependerían fuertemente las funcionalidades otorgadas por el sistema en sus interfaces de búsqueda de entidades, y generación de reportes. Podríamos decir, que se asumía de antemano, que cuanto mayores las prestaciones la grilla, mayores serían las funcionalidades otorgadas al usuario final.

Para la realización de la *aplicación de contacto*, se propuso un wireframe a modo de maqueta rápida. Se trató de un formulario sencillo, con una grilla que representaba a una entidad *Persona*, con algunas características sencillas, que permitiesen utilizar dentro de la grilla distintos tipos de editores de datos, y la posibilidad de generar agrupaciones por columna.



La grilla estaría vinculada con una estructura de datos en memoria, utilizando un enlazado del tipo “two-way-binding”, dado que este se trataba de un aspecto especialmente requerido en la estrategia que el alumno tenía en mente utilizar.

Cada propiedad de la entidad *Persona*, la cual se correspondería a un tipo de dato específico, sería representada en la grilla utilizando un editor de datos particular (por ejemplo, un valor de tipo si/no con un checkbox, una edad con un numeric-up-down, etc).



Emplear diferentes editores de datos en una grilla, tiene especial utilidad para conocer cuan compleja es su implementación, pero, por otro lado, brinda una idea temprana de cómo funciona el editor de forma independiente (es decir, sin estar embebido en una celda), dado que, por lo general, se utilizan editores clásicos que implementan una interfaz acorde, para poder funcionar dentro de una celda.

A continuación, se presenta la grilla en la cual se recabaron los datos de cada uno de los paquetes de controles de usuario.

	Soprote de estilos gráficos		Herramienta para desarrollar estilos		Soporte de archivos de recurso		Paquete de idiomas		Menú Ribbon		Splash Screen		Menú hamburguesa		Soporte MDI con pestañas		La grilla cumple los requisitos		Lista con selector de colores		La lista desplegable cumple los requisitos		Herramienta de generación de reportes		Documentación y recursos		Soporte oficial		Galería de presentación y ejemplos		Total	Solución conocida por el alumno	Soporte o actualización prevista a .Net Core
DotNetBar	Si	5	No	0	No	0	No	0	Si	5	No	0	Si	3	Si	5	Si	6	Si	8	Si	7	No	0	Si	3	Si	3	Si	3	3	Si	No
Grapecity	Si	7	No	0	No	0	Si	7	Si	7	No	0	Si	7	Si	5	Si	10	Si	9	Si	8	Si	7	Si	7	Si	6	Si	6	6	Si	Si
DevExpress	Si	10	Si	8	Si	10	Si	10	Si	8	Si	8	Si	8	Si	8	Si	9	Si	8	Si	10	Si	7	Si	8	Si	8	Si	8	9	Si	Si
Telerik	Si	10	Si	6	Si	10	Si	10	Si	8	Si	10	Si	7	Si	8	Si	9	Si	7	Si	10	Si	7	Si	8	Si	7	Si	8	8	Si	Si
Syncfusion	Si	8	Si	7	Si	10	Si	7	Si	7	No	0	Si	6	Si	7	Si	9	Si	8	Si	8	Si	5	Si	7	Si	7	Si	7	7	No	Si

Motivo de descarte de la solución por incumplimiento de requerimiento

Luego del avanzar en el análisis que se había propuesto, las soluciones candidatas con puntuaciones muy similares, resultaron los paquetes de controles de usuario de DevExpress y Telerik, los cuales además tienen el mismo costo. Finalmente, se seleccionó la solución de DevExpress.

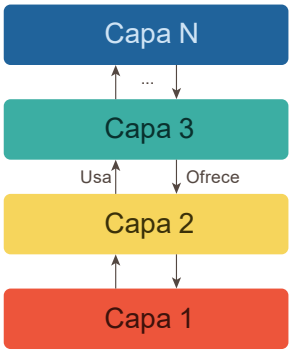
User Story ID 2 - Desarrollo de la arquitectura inicial

Visión general de la arquitectura

Propuesta independiente de la tecnología: estilo arquitectural en capas

El estilo arquitectural en capas se basa en una distribución jerárquica de los roles y responsabilidades para proporcionar una división efectiva de los problemas a resolver. Los roles

indican el tipo y la forma de la interacción con otras capas y las responsabilidades la funcionalidad que implementan.

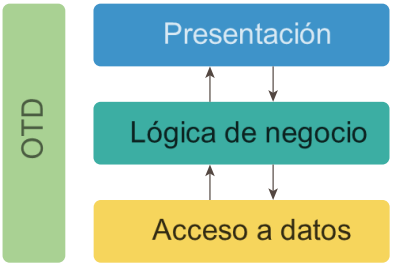


Implementación en .Net

La separación de responsabilidades en la arquitectura, en la visión más general de la misma, se disgregó en tres capas: "Acceso a datos", "Lógica de negocio", y "Presentación". Cada capa lógica fue implementada mediante una librería de clases.

Transversalmente entre las tres capas anteriormente mencionadas, se comparte una capa lógica con entidades con finalidad de OTD ("objetos de trasferencia de datos", habitualmente llamados DTO, del inglés *"Data transfer object"*), para comunicar las mismas entre sí. Los OTD son clases planas (sin comportamiento).

Las particularidades de cada capa (las tecnologías involucradas, la forma en que las mismas están estructuradas, etc.), serán presentadas a continuación, de forma individual para cada capa.



Capa "Acceso a datos"

En la capa de *acceso a datos* se configuró el mapeador objeto-relacional *Microsoft Entity Framework*, empleando el esquema *Code First* (*"Código primero"*), el cual inspecciona las clases que representan al modelo, y luego mediante un conjunto de reglas y convenciones determinar cómo esas clases y sus relaciones describen un modelo, y cómo ese modelo debe mapearse en la base de datos.

También, se añadió *LINQ to Entities*, el cual permite ejecutar consultas escritas en C# respecto a las entidades del modelo de datos a través de *Entity Framework*, las cuales luego son transformadas a SQL de forma eficiente, y ejecutadas en el motor de base de datos.

En el mapeador objeto-relacional, se incorporó una regla que permite actualizar el modelo de datos de la base de datos, al último modelo de datos representado por las clases, siempre que la

aplicación se ejecute en modo *Debug* (es decir, que la misma no esté en entorno de producción); además, para esto modo, se habilitó la *pérdida datos*, lo que implica que si alguno de los cambios efectuados en el modelo conlleva la destrucción de los datos contenidos en la base datos, los mismos serán eliminados para proceder con la actualización.

Finalmente, se desarrolló una clase que, mediante un conjunto de reglas sencillas, permite definir *semillas* que serán insertadas en la base de datos cuando la aplicación inicie en modo *Debug*. Esto permitirá a los desarrolladores contar con datos de prueba conocidos en la base de datos, independientemente que la misma sea reestablecida debido a cambios en sus tablas.

El conjunto de configuraciones mencionado, permite que el despliegue del proyecto en una nueva terminal, o para un nuevo desarrollador, se efectué rápidamente, sin la necesidad de configuraciones particulares a la base datos, y/o de importar datos de prueba a la misma, dado que tales tareas se realizarán automáticamente.

Luego, cada cambio efectuado al modelo de datos a través de cambios en las clases, será enviado al repositorio de código fuente mediante el sistema de versionado, sin la necesidad de compartir la última versión de un DDL (Data Definition Language, *"Lenguaje de definición de datos"*).

Capa "Lógica de negocio"

La capa de la *Lógica de negocio*, consta de la lógica que respecta a las reglas de cómo las entidades de negocio se crean, modifican, eliminan, y de cómo dichas entidades interactúan con otras entidades de negocio. También, respecta a cómo tales entidades se sirven a una capa superior, y a todas otras operaciones que tengan que ver con el flujo de trabajo de las entidades de datos.

Usa los servicios de la capa de A*cceso a datos* para efectuar todas las operaciones de persistencia, pero las reglas previas implicadas a tales operaciones no trascienden de la *Lógica de negocio*.

De forma más general, es en esta capa, en la cual se encontrarán las entidades con las responsabilidades tales, para manejar toda la operatoria del sistema, que se encuentre entre las operaciones de persistencia, y las operaciones de la *capa de presentación*.

Las distintas entidades de esta capa -clases de la *Capa de negocio*-, habitualmente llamadas "Controladores", han sido implementadas mediante clases estáticas, dado que las mismas no están asociadas con instancias particulares de objetos, sino que fueron diseñadas para manejar instancias o grupos de instancias relacionadas, de entidades de la capa de A*cceso a datos*, de forma general.

Se añadió también en esta capa, una clase estática con la finalidad de ser un *Repositorio de parámetros*. El *Repositorio de parámetros*, está compuesto por listas estáticas de entidades de

datos, que serán llenadas mediante la estrategia *Lazy loading* (*"carga perezosa"*) conforme sean requeridas.

Las entidades de datos que serán consideradas "parámetros", son entidades definidas por el solicitante, para las cuales se tiene certeza absoluta que no cambiarán en tiempo de ejecución, o que, de ser actualizadas en tiempo de ejecución, no causarán un impacto en el funcionamiento de otras entidades parámetros, lo cual fuerce a recargar las listas de parámetros nuevamente.

Capa "Presentación"

La *capa de presentación*, reúne los aspectos del software relacionados a la interacción con el usuario final, y a cómo dicho usuario percibe el sistema. Incluye las interfaces gráficas y el comportamiento de las mismas, la lógica que determina como la información será mostrada al usuario, y también las reglas de como la información podrá ser ingresada. Esta capa se sirve de la *capa de lógica de negocio* para toda operación no relacionada con la vista, y se comunica con la misma mediante las entidades de la *capa OTD*.

La tecnología principal seleccionada para la vista es *Microsoft Windows Forms*, y sus controles de usuario fueron extendidos mediante el paquete de controles de usuario de la empresa *DevExpress*.

Windows Forms por defecto propone por una arquitectura conducida por eventos (del inglés *event-driven*). Tanto los formularios como los controles de usuario, exponen un conjunto predefinido de eventos a los cuales se les pueden asignar comportamientos. Si ocurre uno de estos eventos, y hay un código asociado, ese código se invoca.

En *Windows Forms*, las responsabilidades dentro de las vistas (tanto para formularios como para controles de usuario) se separan mediante una división lógica llamada *Code-Beside*, lo cual permite que una misma clase sea definida en múltiples archivos de código fuente, llamados *Partial Types* (*"tipos parciales"*).

Así, inicialmente, cada formulario o control de usuario está compuesto por el diseño gráfico de una vista (un lienzo en el cual se puede ubicar controles de usuario), una clase parcial para los comportamientos inherentes a la vista (como, por ejemplo, la configuración particular de objetos gráficos), y una clase parcial para la lógica de la vista (como, por ejemplo, el comportamiento detrás de los eventos que puedan ser lanzados por la vista).

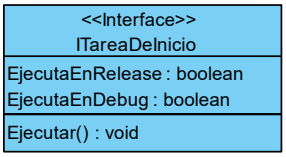
Dentro de la *capa de presentación*, se generaron divisiones lógicas en formato de carpetas (las cuales a su vez representan *namespaces* para las clases contenidas en ellas) para albergar diferentes elementos de la vista, con diferentes responsabilidades. Inicialmente, se crearon cinco carpetas: "Archivos de recursos", "Controles de usuarios", "Formularios", "Interfaces" y "Tareas de inicio".

Dentro de cada carpeta, se generarán divisiones lógicas acordes a las jerarquías que se presenten conforme el sistema sea ampliado. Podría ser que, por ejemplo, los formularios sean agrupados por el dominio al que refieren, o que los controles de usuario sean agrupados por alguna funcionalidad específica en común.

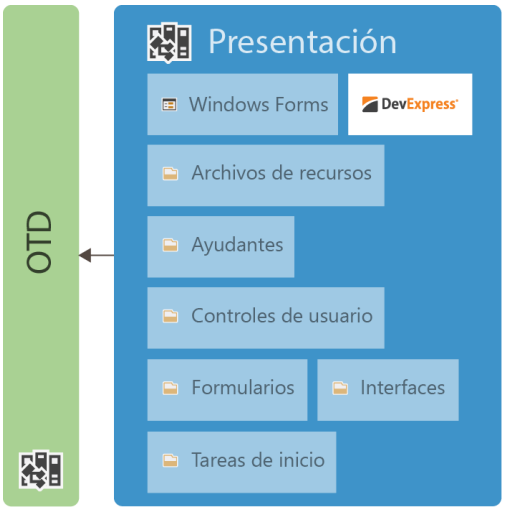
Las carpetas "Formularios" y "Controles de usuario" contienen los elementos gráficos propiamente mencionados con su lógica asociada.

La carpeta "Archivos de recursos", contiene los archivos .resx con las diferentes traducciones y adaptaciones geográficas de formularios y controles de usuario del sistema. A cada archivo de recurso se le asignará el mismo nombre, y ubicación relativa dentro de la carpeta de "Archivos de recurso", de elemento al cual traducen.

"Tareas de inicio" contiene clases, que heredan de la interfaz *ITareaDeInicio*, las cuales contienen lógica específica de rutinas que deben ser ejecutadas sólo al inicio de la aplicación, dependiendo si la aplicación está siendo ejecutada en modo *Debug*, o *Release*.



Todas las interfaces que sean necesarias para especificar el comportamiento de clases, en la capa de presentación, serán ubicadas dentro de la carpeta "Interfaces".



Comentarios respecto a decisiones en la arquitectura final

- Patrón de diseño "Repositorio" para abstraer la lógica de acceso a datos provista por el Mapeador Objeto-Relacional:
  Se encuesto al Product Owner sobre la inquietud a futuro, de cambiar el mapeador objeto-relacional (Entity Framework) por otro, a futuro. En caso de una respuesta afirmativa, entonces se propondría el uso de del patrón de diseño "Repositorio". También se consultó sobre el uso de dicho patrón de diseño, dado que es ampliamente utilizado en la industria.
  Uno de los elementos más importantes a favor de confiar en Entity Framework de forma perpetua para el proyecto, es que es open-source desde su versión 6, desde el año 2017, y tiene amplio soporte por parte de la comunidad.

Por otro lado, *Entity Framework* opera empleando el patrón de diseño ("Repositorio"), mediante componentes que encapsulan la lógica necesaria para acceder a las fuentes de datos. Más específicamente, en Entity Framework, un *DbContext* representa la UoW (Unit of Work, "Unidad de trabajo"), y cada DbSet representan los distintos repositorios.
Una preocupación importante frente a no implementar interfaces que describan los repositorios, es el testing. Para hacer frente a esta dificultar, *Entity Framework* permite generar *Mocking Contexts*, que respondan al comportamiento deseado, con datos en memoria.

- OTD (objetos de transferencias de datos) específicos entre cada capa lógica:
  Con la finalidad de contar con entidades específicamente adecuadas al dominio en el cual serán utilizadas, es que se propuso al Product Owner contar con diferentes grupos de entidades, para cada capa lógica utilizada.
  El Product Owner comentó que, en sus desarrollos de la actualidad, en la gran mayoría de los casos, la definición de las entidades "del mundo real" que son plasmadas en la base de datos mediante tablas y relaciones, atraviesan los diferentes procesos intermedios, hasta llegar a la vista, casi sin alteración alguna.
  Por esta razón, es que se utilizarán las entidades de la capa de acceso a datos, como objetos de transferencia de datos entre las tres capas lógicas actuales.
  Entity Framework, de cierto modo, facilita esta práctica, dado que la definición de las clases que luego serán mapeadas, según Entity Framework, resulta en clases planas (sin comportamiento).
  Por otro lado, al tratarse de una aplicación stand-alone, en la cual, todas las capas serán ejecutadas sobre un mismo entorno -al menos de forma inicial-, no hay preocupación respecto a la comunicación entre las capas, cuando las mismas se encuentran separadas físicamente.

Punto de inicio de la aplicación

Las aplicaciones en *Windows Forms* (como muchas otras aplicaciones del ecosistema de .Net, en particular, las aplicaciones de consola) tienen como punto de inicio la clase *Program.cs*, que es en donde se encuentra el método *Main*, que es la primera invocación que será realiza al iniciar la aplicación.

La clase *Program.cs* se encuentra en la raíz del proyecto, y por convención, se emplea esta clase para establecer los elementos que respecten a las validaciones y configuraciones esenciales, que deban ocurrir al inicio de la aplicación.

La clase por defecto está preparada para invocar -si así se desea- una pantalla de *Splash screen*, al tiempo que se ejecutan de fondo tareas particulares del inicio. Una vez finalizadas todas las tareas definidas en la clase *Program.cs*, se procede con la apertura de un formulario principal, que permanecerá abierto durante todo el ciclo de vida del proceso en ejecución.

Tareas de inicio

Tanto la lista de instancias de clases que heredan de *ITareaDeInicio*, como el algoritmo que discierne si las mismas deben ser ejecutados o no, dependiendo si el entorno se está ejecutando en *Debug* o *Release*, se encuentran dentro de la clase *Program.cs*.

A continuación, se presenta un extracto del código que ejecuta las *Tareas de inicio*:

```
// Lista de Tareas de inicio:
List<ITareaDeInicio> tareasDeInicio = new List<ITareaDeInicio>();
tareasDeInicio.Add(new CrearBaseDeDatosSiNoExiste()); // Solo Debug
tareasDeInicio.Add(new LlenarBaseDeDatosConSemillasSiEstáVacía()); // Solo Debug
tareasDeInicio.Add(new ProbarConexiónABaseDeDatos()); // Debug y Release
// Ejecución de Tareas de Inicio:
foreach (ITareaDeInicio tareaDeInicio in tareasDeInicio)
{
    if (this.isDebug == true && tareaDeInicio.EjecutaEnDebug == true)
    {
        tareaDeInicio.Ejecutar();
    }
    else if (this.isDebug == false && tareaDeInicio.EjecutaEnRelease == true)
    {
        tareaDeInicio.Ejecutar();
    }
}
```

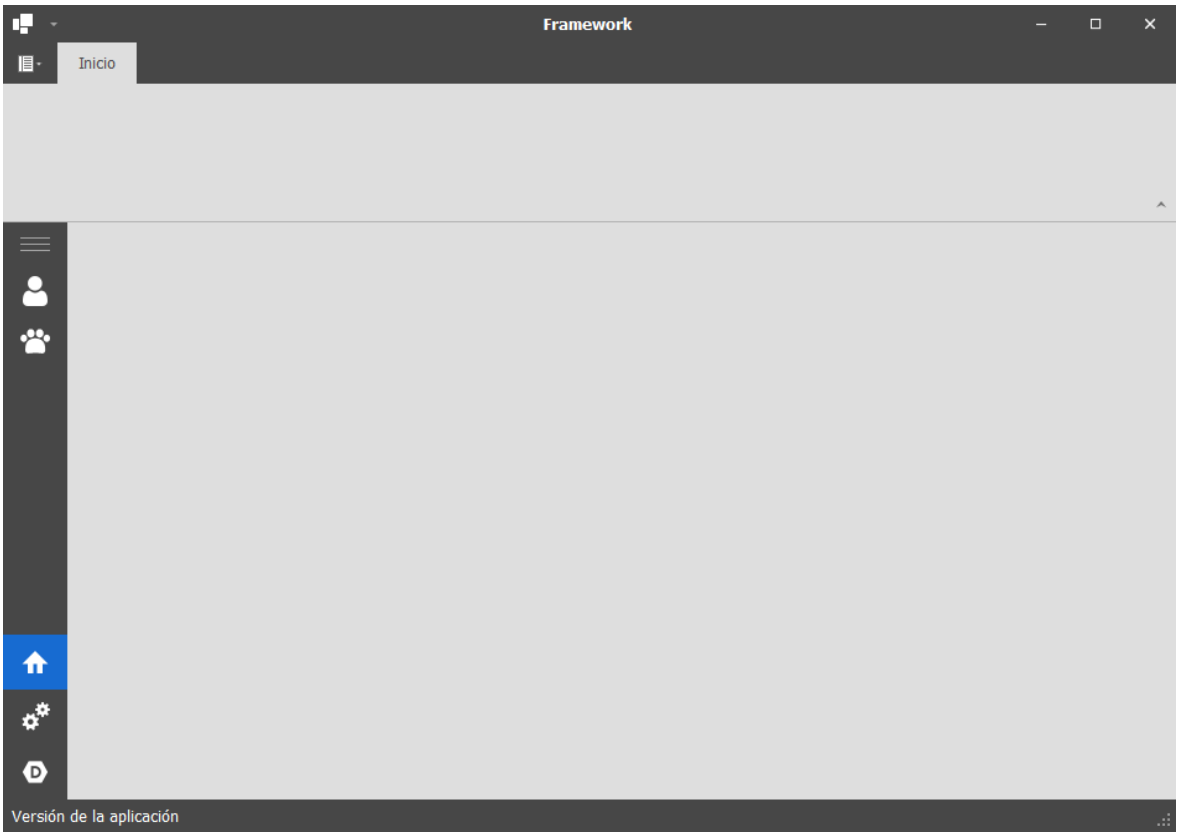
User Story ID 3 - Desarrollo del formulario principal

Se desarrolló el formulario principal, cumpliendo con todos los requerimientos estéticos y funcionales especificados en el modelo conceptual de la aplicación (*Formulario principal*, *y* *Menú principal*), que luego formaron parte del presente *User story*.

Al formulario principal se le asignó el nombre "*FormularioPrincipal.cs*", y se ubicó en la raíz del proyecto, en la misma jerarquía que la clase *Program.cs*.

La lógica interna del formulario se preparó para acoplar y desacoplar formularios MDI hijos, de forma tal que, al acoplarlos, el contenido de los menús *Ribbon* y de las barras de estado se combine. Esta funcionalidad será transparente a los programadores que desarrollen sobre el framework.

Se añadieron ítems principales o de raíz, sólo para testear funcionalidades. Los ítems del pie se configuraron según lo definido en los requerimientos, y, además, se añadió un tercer ítem, para contener elementos asociados exclusivamente al modo *Debug*, el cual no será visible en *Release*.

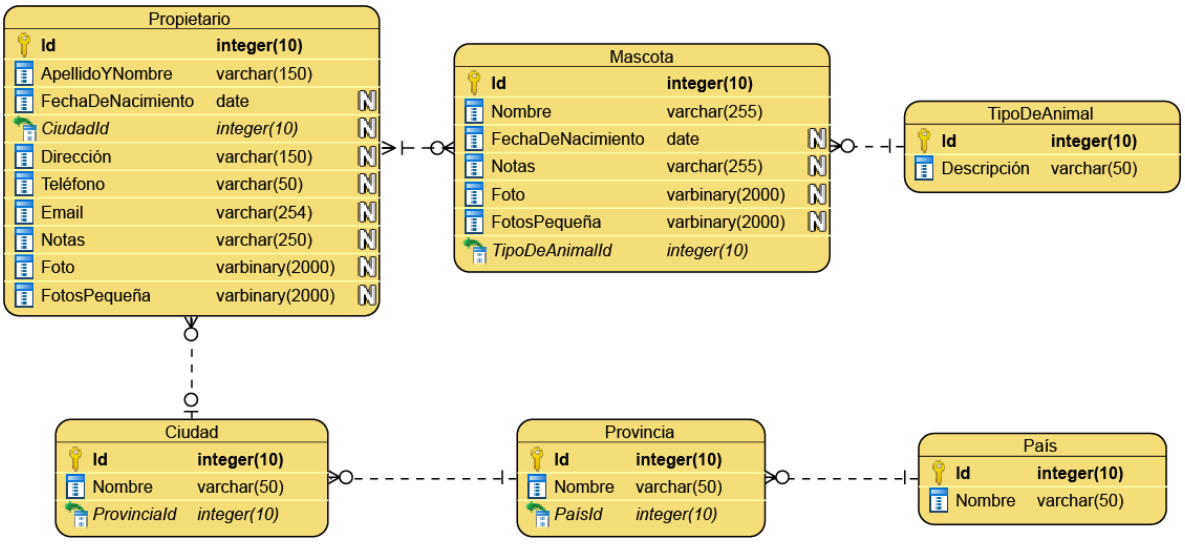


User Story ID 4 - Generar lote de datos de prueba

Modelo de datos con entidades de prueba

Para efectuar pruebas tanto visuales como funcionales en el framework en desarrollo, se planteó un modelo de datos trivial que sea fácilmente interpretado.

El dominio elegido fue el que relaciona a *Propietarios* de mascotas, con *Mascotas*, donde los *Propietarios* tienen una relación con la *Ciudad* en la que viven, y las *Mascotas*, con su *TipoDeAnimal*.



Estrategia de traducción de entidades de datos que son *Parámetros* en el sistema

Como estrategia para traducir diferentes elementos del sistema (Formularios, Controles de usuarios, etc.), y permitir que la adición de nuevos idiomas, o modificación de idiomas existentes, se realice de forma sencilla y sin la necesidad de manipular los elementos que están siendo traducidos, se utilizaron archivos de recurso como solución ortodoxa.

Un limitante de la estrategia propuesta, es que los archivos de recurso son siempre utilizados por otro elemento del sistema (Formularios, Controles de usuarios, etc.) que pueda vincular alguno de sus elementos (como, por ejemplo, un texto particular) a una clave del archivo de recursos, y luego ser compilado, por lo cual se genera una relación fuerte entre los dos extremos.

En el presente escenario, en el cual se propone una herramienta extensible, surge la necesidad de permitir la traducción de texto contenido en las propiedades de distintas entidades de datos. No es posible vincular las instancias de entidades de datos, las cuales son creadas en tiempo de ejecución, con archivos de recursos estáticos, creados en tiempo de diseño.

Para tal fin, se propuso una solución a media, la cual consiste en añadir una etiqueta única a las propiedades de cada clase que soportarán ser traducidas, para que cada programador que trabaje con ellas sea consciente de ello.

La etiqueta no tiene un fin dinámico, por si solo el lenguaje o el entorno de desarrollo no asumirá una postura frente a esto, es decir que el funcionamiento de la solución dependerá de que los programadores sean conscientes de su uso e implementación.

A continuación, como ejemplo del uso de la etiqueta "*Localizable*", se presenta la clase *TipoDeAnimal*, empleada en el *Modelo de datos con entidades de prueba*:

```
public class TipoDeAnimal
{
    public int Id { get; set; }

    [Required]
    /// <summary>
```

```
    /// ! Localizable
    /// </summary>
    public string Descripción { get; set; }
}
```

Como convención, para definir el lenguaje del contenido de todas aquellas propiedades que tengan la etiqueta "*Localizable*", se optó por utilizar las etiquetas de lenguaje definidas por Microsoft, conocidas como LCID[23] ("Language Code Identifier"). Tal cómo Microsoft menciona, utilizar LCID asegura mantener la consistencia entre el manejo de lenguajes de la aplicación, y el sistema operativo huésped.

La etiqueta consiste en una subetiqueta principal que identifica el idioma (por ejemplo, "es") y una subetiqueta que especifica la variedad nacional (por ejemplo, "AR"), separadas por un guion medio.

Inicialmente, el alumno pensó en utilizar las etiquetas de lenguaje definidas por la IETF[24] (las cuales a su vez fueron desarrolladas incorporando la norma ISO 639-1[25]), dado que son ampliamente utilizadas en estándares de Internet, como HTTP, HTML, etc. Dado que se están utilizando herramientas y tecnologías de Microsoft, finalmente se optó por LCID.

Una cadena que soporte ser traducida, y esté almacenada dentro de una propiedad de una clase que sea *Localizable*, deberá entonces cumplir con el siguiente formato: iniciar con la etiqueta de lenguaje, continuar con el carácter *dos-puntos*, y luego el texto correspondiente. Para múltiples idiomas, se repite el formato, separando las cadenas a través del carácter *pipe*.

Por ejemplo, para una instancia de la clase *TipoDeAnimal*, que representa a un *Perro*, su contenido localizable para el idioma español de España, e inglés de Estados Unidos, resultaría:

```
"es-AR:Perro|en-US:Dog"
```

Traducción de propiedades de entidades de datos que son localizables

Para efectuar la traducción de las propiedades de aquellas clases que sean localizables, se desarrolló un algoritmo, el cual debe ser invocado por los desarrolladores de forma discrecional cuando se trate de una clase que tiene una propiedad *Localizable*.

Se generó una carpeta con nombre "*Ayudantes*" (del inglés "*Helpers*"), en la raíz del proyecto de la capa de presentación. Dentro de dicha carpeta, serán almacenadas las clases que contengan métodos que permitan resolver problemas genéricos.

Dentro de la carpeta *Ayudantes*, se generó la clase estática *AyudanteDeIdioma*, con el método estático *Localizar*.

El método *Localizar*, recibe como parámetro una cadena que responde al formato de cadenas localizables definido previamente (dos caracteres para identificar el idioma, guion medio, dos caracteres para definir la región, carácter dos puntos, cadena en lenguaje apropiado, y carácter

[23] Códigos de identificación de lenguaje de Microsoft (LCID): https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-lcid/
[24] Etiquetas para identificación de lenguajes de la IETF: https://tools.ietf.org/html/bcp47#section-2.2.1
[25] Etiquetas para identificación de lenguajes de la ISO: https://www.iso.org/iso-639-language-codes.html

pipe para separar distintos idiomas alternativos), y retorna la porción de la cadena que corresponde al lenguaje actual de la aplicación, o el lenguaje por defecto (es-AR) en caso de no encontrarse.

A continuación, a modo de referencia, se cita el código que permite localizar propiedades según el idioma actual de la aplicación:

```
internal static string Localizar(string propiedadLocalizable)
{
    // Obtiene etiqueta de lenguaje actual en la aplicación
    string etiquetaDeLenguajeActual = CultureInfo.CurrentCulture.ToString();

    // Almacena la cadena resultante, localizada (con su adecuada traducción)
    string localizado = "";

    // Obtiene diferentes idiomas separadamente
    string[] localizaciones = propiedadLocalizable.Split('|');

    // Busca dentro de los diferentes idiomas, si existe el idioma actual de la aplicación
    foreach (string localización in localizaciones)
    {
        if (localización.Contains(etiquetaDeLenguajeActual))
        {
            localizado = localización.Substring(6).Trim(); // 5 caracteres de etiqueta, más :
            break;
        }
    }

    // Si no encontró cadena con la etiqueta de lenguaje actual, busca por es-AR (por defecto)
    if (string.IsNullOrEmpty(localizado) == true)
    {
        foreach (string localización in localizaciones)
        {
            if (localización.Contains("es-AR"))
            {
                localizado = localización.Substring(6).Trim();
                break;
            }
        }
    }

    // Si no encontró cadena con el idioma indicado, ni con el por defecto, utiliza la cadena tal
cual es
    if (string.IsNullOrEmpty(localizado) == true && localizaciones.Length == 1)
    {
        localizado = propiedadLocalizable.Trim();
    }

    return localizado;
}
```

Así, tomando como ejemplo de entrada, la cadena de ejemplo propuesta previamente para una instancia de la clase *Especie*, que represente al objeto *Perro*, "es-AR:Perro|en-US:Dog", y suponiendo que la aplicación está configurada con lenguaje español de España (es-ES), si se aplicara el método Localizar("es-AR:Perro|en-US:Dog"), el resultado sería “Perro”.

Lote de datos de prueba y semillas de la base de datos

Se generaron lotes de datos de prueba para todas las entidades de pruebas propuestas: *Ciudad*, *Provincia*, *País*, *Propietario*, *Mascota*, y Especie. El código relacionado a las semillas de cada clase, fue dispuesto en la clase estática *ContenedoDeSemillas*, en la *Capa de Acceso a datos*.

Para *Ciudad*, *Provincia* y *País*, se utilizaron todas las provincias y ciudades de la Argentina. Para las *Propietario* y *Mascota* se emplearon datos de personajes de fantasía consumidos desde un endpoint público. En el caso de *Especie*, los datos se completaron manualmente.

La inserción de las semillas se realizará mediante el criterio *AddOrUpdate* ("*Agregar o actualizar*"), lo que implica que, si alguno de los registros aún no existe en la base de datos, será agregado, caso contrario, si un registro ya existe, pero su contenido es diferente al de la semilla, será actualizado a los valores definidos en la semilla.

A modo de ejemplo, para mostrar cual es el procedimiento de definición de semillas, se cita el código de semillas de la clase *Especie*:

```
#region TipoDeAnimal

List<TipoDeAnimal> tiposDeAnimal = new List<TipoDeAnimal>();

tiposDeAnimal.Add(new TipoDeAnimal() { Id = 1, Descripción = "es-AR:Leon|en-US:Lion" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 2, Descripción = "es-AR:Lémur|en-US:Lemur" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 3, Descripción = "es-AR:Lémur ratón|en-US:Mouse lemur"
});
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 4, Descripción = "es-AR:Lémur aye-aye|en-US:Aye-aye
lemur" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 5, Descripción = "es-AR:Hipopótamo|en-US:Hippopotamus"
});
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 6, Descripción = "es-AR:Zebra|en-US:Zebra" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 7, Descripción = "es-AR:Pingüino|en-US:Penguin" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 8, Descripción = "es-AR:Mono|en-US:Monkey" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 9, Descripción = "es-AR:Cerdo|en-US:Pig" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 10, Descripción = "es-AR:Perro|en-US:Dog" });
tiposDeAnimal.Add(new TipoDeAnimal() { Id = 11, Descripción = "es-AR:Jirafa|en-US:Giraffe" });


context.TiposDeAnimal.AddOrUpdate(
    e => e.Id,
    tiposDeAnimal.ToArray()
);

#endregion // TipoDeAnimal
```

User Story ID 5 - Desarrollo de formulario de búsqueda (base)

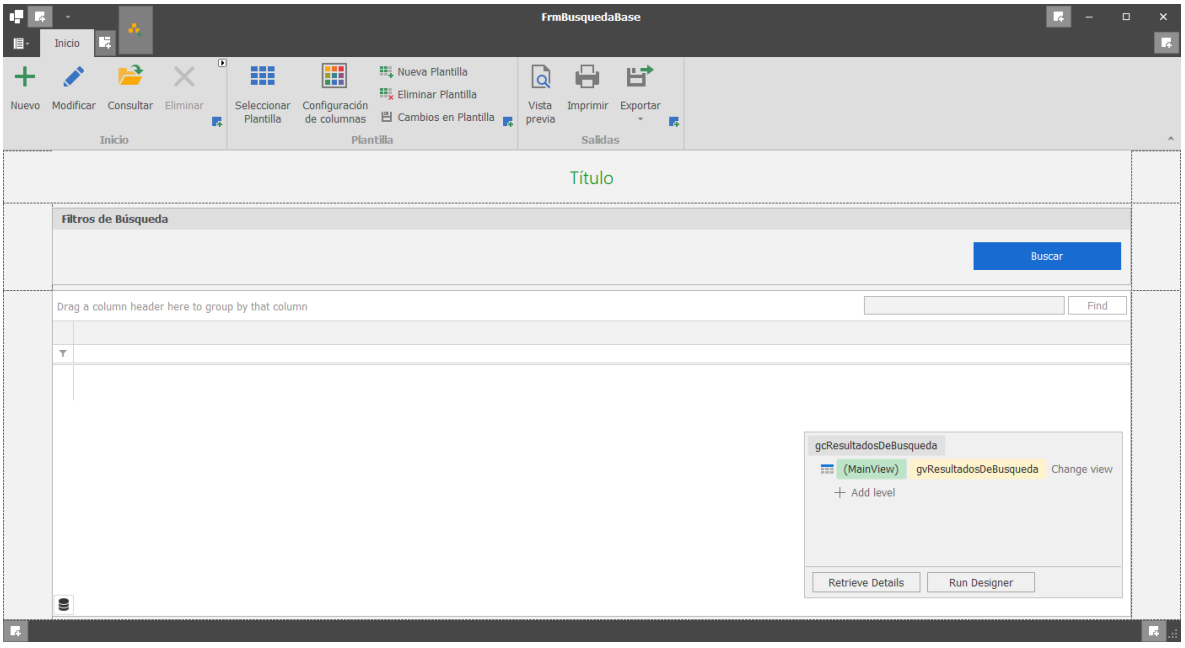
Interfaz de Formulario de Búsqueda (base)

Se desarrolló el Formulario de Búsqueda base, acorde a las especificaciones dadas en wireframes. Además, se hizo la plantilla responsiva a los cambios de tamaño de pantalla, de forma tal que el largo del lienzo siempre se adecuará a la altura en la resolución ocupando todo el espacio posible, y el ancho completará sus laterales con espacios uniformemente distribuidos (simulando un típico comportamiento web).

Los botones de la barra superior, así como los grupos, son elementos aún sin comportamiento. El comportamiento de los mismos será añadido posteriormente.

Todos los elementos gráficos del formulario (menú Ribbon, grilla, etc.) excepto la zona de filtros de búsqueda (entre la etiqueta del título y la grilla) es privada, esto implica que, al ser especializado el formulario de base, la especialización no podrá alterar dichos elementos.

En lo que respecta a elementos gráficos, las especializaciones sólo tendrán la libertad de añadir elementos gráficos en área de filtros de búsqueda, tal como se planteó en la especificación.



Conceptos base: ViewModel, DataBinding, y DataAnnotation

Con el objetivo de generar un buscador que reúna todas las características requeridas, que sea extensible con facilidad, y permita a los desarrolladores especializarlo rápidamente, se emplearán ciertos conceptos, los cuales serán introducidos y vinculados a la solución, a continuación.

ViewModels

Anteriormente se mencionó el uso de modelos en distintas capas de la aplicación, para describir las entidades de dominio en cada una de ellas. Un *ViewModel* es un modelo específicamente empleado para describir algún elemento de la capa de presentación (de la vista); puede tratarse de un formulario o de un control de usuario.

Al igual que en situaciones previas, los modelos serán representados mediante clases. Las propiedades de dichas clases, con sus respectivos tipos de datos, representarán a un elemento particular de la vista.

De este modo, una instancia de una clase, reflejará el contenido de un elemento particular de la vista, o viceversa.

DataBinding

DataBinding[26] ("*El enlace de datos*") es el proceso que establece una conexión entre un elemento de la interfaz de usuario, y el origen de datos desde el cual extrae los datos que muestra.

Cuando se establece un enlace de datos, en el momento que un origen de datos cambia su valor, los elementos que están vinculados a él reflejan los cambios automáticamente.

La vinculación de datos también puede significar que, si cambia una representación externa (mediante interfaz de usuario) de los datos en un elemento, los datos subyacentes en el origen de datos se pueden actualizar automáticamente para reflejar el cambio.

Cuando los datos se enlazan desde un origen de datos a un elemento de la interfaz de usuario, se lo llama "*one-way-binding*" ("*enlace en una dirección*"), cuando los datos se enlazan de forma bidireccional, se lo llama "*two-way-binding*" ("*enlace en las dos direcciones*").

Para soportar el enlace de datos en una o ambas direcciones, el origen de datos debe notificar sus cambios cuando ocurren, de forma tal que los destinos vinculados reflejen automáticamente dichos cambios dinámicos. Para que una clase provea las notificaciones correctas cuando una de sus propiedades cambia, debe implementar la interfaz *INotifyPropertyChanged*[27] [28].

DataAnnotation

Los *DataAnnotation*[29] ("*Anotaciones de datos*") son atributos que se aplican a la clase y/o a miembros de la clase, que especifican diferentes aspectos relevantes para los consumidores de la clase y/o miembros.

Por ejemplo, si la clase está siendo enlazada con algún elemento de la interfaz de usuario, se podrían especificar reglas de validación para su ingreso (por ejemplo, mediante una expresión regular), o especificar cómo se muestran los datos en la interfaz (por ejemplo, en el caso de una fecha, utilizar diferentes formatos dependiendo de la ubicación geográfica), etc.

Anteriormente en este documento, se vio el uso de *DataAnnotations* en la definición de entidades de *Entity Framework*, cuando se utiliza el esquema *Code First*. En este caso, las anotaciones permiten especificar relaciones entre clases, atributos de los campos (como si son requeridos, su longitud, etc.), es incluso especificar que una determinada clase no debe ser mapeada.

[26] Documentación oficial de DataBinding en Windows Forms: https://docs.microsoft.com/en-us/dotnet/desktop/winforms/windows-forms-data-binding?view=netframeworkdesktop-4.8
[27] Documentación oficial de INotifyPropertyChanged: https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.inotifypropertychanged?view=net-5.0
[28] Documentación oficial sobre implementación de INotifyPropertyChanged: https://docs.microsoft.com/en-us/dotnet/desktop/wpf/data/how-to-implement-property-change-notification?view=netframeworkdesktop-4.8
[29] Documentación oficial de DataAnnotation: https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations?view=net-5.0

Uso de ViewModels y GridControl

Se empleó un *ViewModel* que describe como se ve y se comporta un *GridControl* (grilla de *DevExpress*), dado que para cada propiedad del *ViewModel* se le corresponde una columna, y para cada tipo de dato de cada propiedad, se le corresponde un editor de datos particular. Así, para una columna propiedad de tipo de dato *Date*, por ejemplo, se le asignará en la grilla un selector de tipo calendario, de forma automática.

De este modo, cada columna de la grilla, tiene como nombre en sus encabezados el nombre de la propiedad que representan. Si, por ejemplo, una propiedad de la clase tiene por nombre "ApellidoYNombre", el encabezado de la columna será exactamente el mismo.

Resulta necesario entonces, poder especificar un *nombre para mostrar* en la interfaz de usuario, cuando se hace referencia a una determinada propiedad de un *ViewModel*. Para tal fin se empleará el *DataAnnotation Name*, de la clase de atributos *DisplayAtribute*[30], que permite especificar cadenas localizables para tipos y miembros de clases.

Para asignar el *nombre para mostrar* "Apellido y nombre", a la propiedad "ApellidoYNombre", haciendo uso del *DataAnnotation* "*Name*", resultaría:

```
[Display(Name = "Apellido y nombre")]
public string ApellidoYNombre { get; set; }
```

Ahora, es necesario que el *nombre para mostrar* pueda declararse en diferentes idiomas, dado que la aplicación que se está generando es globalizable. Además, es necesario almacenar los diferentes *nombres para mostrar* fuera de la clase, para evitar generar cambios en la misma cuando se trata de cuestiones meramente estéticas.

Separar los aspectos visuales de la clase de su declaración, favorece el *Principio de Responsabilidad Única*[31], en el cual su creador ha dicho "*Una clase debe tener solo una razón para cambiar*".

El *DataAnnotation DisplayAtribute*, también permite referenciar un archivo de recursos mediante la etiqueta *ResourceType*. Se accederá al contenido del archivo de recursos, utilizando como clave el contenido de la etiqueta *Name*.

Es posible dar el nombre tanto del archivo de recursos en *ResourceType*, como de la etiqueta *Name* mediante cadenas. Si se hace de esa manera, el compilador no podrá verificar que los orígenes existen. Para que la aplicación sea de tipado estático, y asegurar que los orígenes existen (archivo de recursos, y claves dentro del mismo), se utilizarán referencias a los mismos para obtener sus nombres.

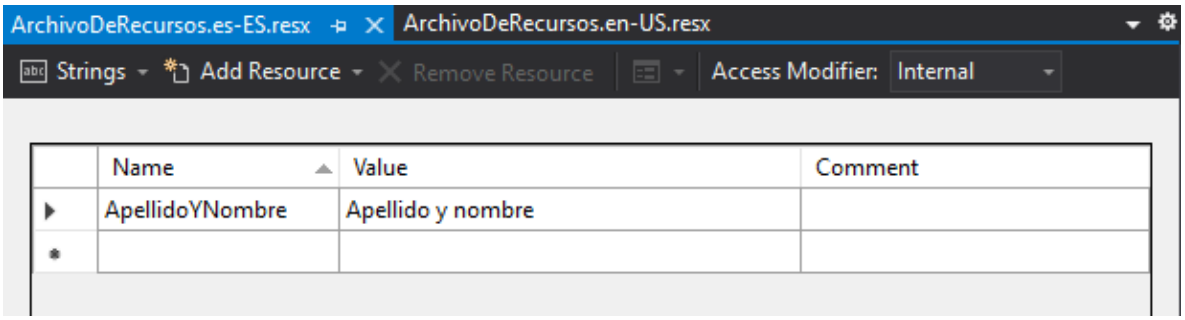
La estrategia completa, haciendo uso de *DataAnnotation*, con referencia a un archivo de recursos, resulta:

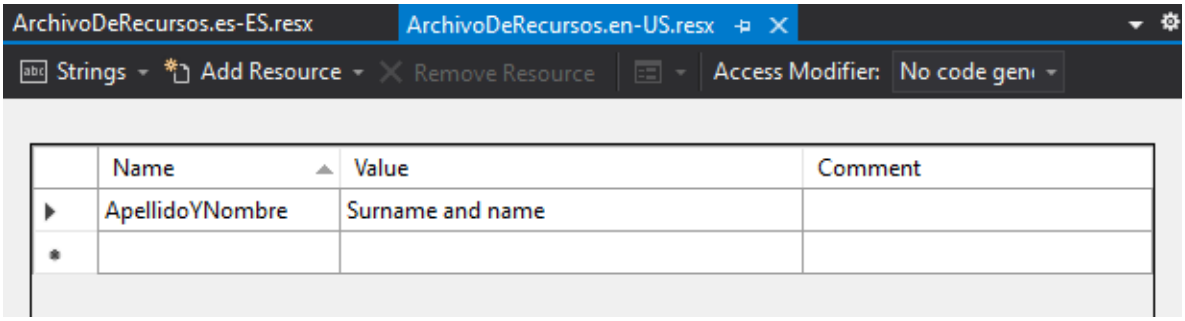
```
[Display(Name = nameof(ArchivoDeRecursos.ApellidoYNombre), ResourceType =
typeof(ArchivoDeRecursos))]
```

[30] Documentación oficial de DisplayAtribute: https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations.displayattribute?view=netframework-4.7
[31] Principio de Responsabilidad única, de Robert C. Martin: https://blog.cleancoder.com/uncle-bob/2014/05/08/SingleReponsibilityPrinciple.html

```
public string ApellidoYNombre { get; set; }
```

Y el archivo de recursos con nombre *ArchioDeRecursos*, que cuenta con una clave *ApellidoYNombre*, en sus versiones “es-ES”, y “en-US”, resulta:





Luego de la aplicación de estas estrategias en conjunto, conseguimos que la definición de la grilla se realice automáticamente conforme el estado del *ViewModel* que tiene asociado, y que el encabezado de las columnas responda al contenido de un archivo de recursos, posibilitando que la grilla soporte múltiples idiomas, dejando cada responsabilidad por separado, con tipado estático para asegurar que no se den errores de tipo en tiempo de ejecución.

Lógica de Formulario de Búsqueda (base)

La lógica fundamental del *Formulario de Búsqueda (base)* está compuesta por tres aspectos que son ejecutados en forma inmediata dado que son invocados en el constructor de la clase base, los cuales serán explicados a continuación: estados internos del formulario -que no dependen de la especialización-, y configuración del título e inicialización de la grilla con su respectivo *ViewModel* -que dependen de datos añadidos en la especialización-.

Estados del formulario

El comportamiento del Formulario de Búsqueda (base) se describe a través de seis estados, mediante los cuales se cubren las diferentes situaciones en las cuales el mismo puede encontrarse dependiendo de factores como: una búsqueda con o sin resultados, un ítem del listado seleccionado, etc.

Los posibles estados del formulario se describieron a través de un enumerado[32], de la siguiente manera:

```
public enum EstadoBusqueda
{
    Oculto,
    Inicial,
    Buscando,
    SinResultados,
    ConResultados,
    ItemSeleccionado
};
```

Cada uno de los estados describe un estado general para un conjunto de elementos del formulario: cursor del mouse, botones nuevo, modificar y consultar del menú ribbon, grilla, y caja de filtros de búsqueda. Dependiendo de los estados, se habilitan o deshabilitan.

Por ejemplo, cuando el formulario está en estado *Buscando*, los todos los botones, la grilla y la caja de filtros se deshabilitan, y el cursor del mouse es de tipo *espera*:

```
System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.WaitCursor;

// Botones
this.bbiNuevo.Enabled = false;
this.bbiModificar.Enabled = false;
this.bbiConsultar.Enabled = false;

// Grilla
this.gcResultadosDeBusqueda.Enabled = false;

// Filtros de búsqueda
this.gcFiltrosDeBusqueda.Enabled = false;
```

El resto de los estados siguen la misma estrategia. Es probable que la lógica dentro de cada uno de los estados crezca, dado que, por ejemplo, podría añadirse información a las barras de estado inferiores sobre la actividad en curso.

Título del formulario

El título del formulario se configurará en diferentes lugares al mismo tiempo (etiqueta principal del formulario, título de la pestaña, etc.). Las tareas de colocar el contenido del título en diferentes lugares pertinentes, es tarea del formulario base, pero el contenido del título será establecido en los formularios que especialicen al formulario base.

Para esto, en el formulario base se ha declarado un método virtual[33] (lo que permite que sea sobrescrito en la especialización):

```
protected virtual string ConfigurarTituloDeFormulario() { }
```

Podría pensarse que una propiedad en la base, que sea visible por la especialización resuelve el problema de un modo más directo. Se optó por esta estrategia para permitirle al programador

[32] Documentación oficial de Enum ("*enumerados*"): https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/enum
[33] Documentación oficial de la palabra reservada *virtual*: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual

que añada cierta lógica dentro del método especializado, como acceder a un archivo de recursos, o incluso, obtener la información desde alguna capa inferior.

Inicialización de la grilla con datos de su respectivo ViewModel

Para que la grilla añada columnas que representen las propiedades de un determinado *ViewModel*, es necesario asignar al *DataSource* ("*origen de datos*") de la grilla, el listado con entidades que serán mostrados en la grilla.

En el formulario base no se cuenta con el tipo de dato del *ViewModel*, el mismo es cargado en el formulario especializado, en una propiedad del formulario base, de la siguiente manera:

```
base.TipoDeDatoDelVM = typeof(UnViewModel);
```

El *DataSource* de la grilla utilizará una lista con el tipo de datos del *ViewModel*, pero, además, debe ser una lista que soporte *DataBinding*, dado que utilizaremos *ViewModels* en que estarán preparados para trabajar con *DataBinding*.

El framework .Net proporciona una colección genérica que brinda soporte para *DataBinding*, llamada *BindingList*[34].

BindingList genera listas genéricas, por lo cual es necesario proporcionarle el tipo de datos de la lista que se desea generar, en este caso, el tipo de datos de un *ViewModel*. En el formulario base, la lista que se generará, es del tipo de datos del *ViewModel* cargado en una propiedad del formulario base por parte de un formulario especializado, como se mostró anteriormente.

Generamos el tipo de datos de una *BindingList*, del tipo de datos de un *ViewModel* particular, de la siguiente manera:

```
Type listaDeTipoGenerico = typeof(BindingList<>).MakeGenericType(tipoDeDatoDelVM);
```

Generar el tipo de datos genérico anterior es realizado por el método *MakeGenericType*[35], que es aplicable a tipos de datos de listas.

Ahora, es necesario generar una lista, con el tipo de datos de la lista genérica que se creo anteriormente, para permitir que las entidades sean añadidas a la misma:

```
IBindingList listaEntidades = (IBindingList)Activator.CreateInstance(listaDeTipoGenerico);
```

Es posible crear instancias de un tipo de dato específico, mediante el método *CreateInstance* [36] de la clase *Activator*[37]. La clase *Activator*, en general contiene métodos que permiten crear tipos de objetos, mientras que su método *CreatInstance* permite crear una instancia de un tipo de dato específico, empleando el constructor que mejor se adapta a los parámetros dados.

---

[34] Documentación oficial de BindingList: https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.bindinglist-1
[35] Documentación oficial de MakeGenericTypo: https://docs.microsoft.com/en-us/dotnet/api/system.type.makegenerictype
[36] Documentación oficial de CreateInstance: https://docs.microsoft.com/en-us/dotnet/api/system.activator.createinstance
[37] Documentación oficial de Activator: https://docs.microsoft.com/en-us/dotnet/api/system.activator

Contando ya con la lista de entidades del tipo de datos *BindingList<UnViewModel>*, sólo resta asignarla al *DataSource* de la grilla:

```
grilla.DataSource = listaEntidades;
```

Dado que la lista está enlazada a la grilla mediante *BindingList*, conforme se añadan entidades a la misma, serán mostradas de inmediato en la grilla.

A modo de resumen, lo que se ha conseguido es, que con sólo el suministro del tipo de dato de un *ViewModel* preparado para la grilla, generar una lista de tipo *BindingList* inicializada con el tipo de datos del *ViewModel*, con la cual se inicializa el *DataSource* de la grilla, lo cual genera la inicialización de la grilla con las columnas y sus respectivas cabeceras según se definió en el *ViewModel*.

Lógica requerida en la especialización

Asociación de un ViewModel para la grilla de búsqueda

Es fundamental para que el formulario base inicialice la grilla, proveer el tipo de datos del *ViewModel* que será utilizado, como se mostró anteriormente. Esto debe realizarse en el constructor de la especialización:

```
base.TipoDeDatoDelVM = typeof(UnViewModel);
```

De esta manera, la lógica desarrollada en el formulario base inicializará la grilla de búsqueda, y una lista con *DataBinding*, preparada para recibir entidades y mostrarlas en la grilla.

Buscar, agregar filas a la grilla

Para agregar entidades (filas) a la grilla, en el formulario especializado, se debe sobrescribir el método *Buscar* del formulario base, el cual tras cada invocación recibirá por parámetro la lista de entidades que debe ser actualizada (que es en efecto, la lista asociada a la grilla mediante su *DataSource*).

```
protected override void Buscar(IBindingList listaDeEntidadesGenerica)
{
    // Lista de entidades que serán mapeadas a su correspondiente ViewModel:
    // (Las cuales serán recuperadas de la capa de Lógica de Negocios, por
    // ejemplo)
    List<Propietario> listaPropietarios;

    // Casteo de la lista genérica recibia por parámetro, al tipo de lista
    // correspondiente según el ViewModel:
    BindingList<VM_Propietario> listaVMs = (BindingList<VM_Propietario>)listaDeEntidadesGenerica;

    // Lógica de mapeo de la lista de entidades, a los ViewModel
    // Por ejemplo:
    List<VM_Propietario> entidadesMapeadas = this.Mapear(listaPropietarios);
    listaVMs.AddRange(entidadesMapeadas);
}
```

La operatoria consiste en obtener las entidades que deban ser mostradas en los resultados de búsqueda, y en mapear dichas entidades en su correspondiente *ViewModel*, es decir, generar un traspaso -y si se requiere, adaptar la información- a *ViewModels*. Luego, la lista de ViewModel, es añadida a la lista recibida por parámetro, y los datos serán inmediatamente visualizados en la grilla, sin efectuar ninguna otra operación.

Internamente, el formulario de base, generará un cambio de estado a *Buscando* mientras se efectúa la búsqueda, y al finalizar la búsqueda cambiará al estado *ConResultados* o *SinResultados*, según corresponda.

Organización del código fuente

El código dentro de un mismo archivo, será organizado empleado la palabra reservada *region*[38], la cual permite especificar un bloque de código que puede ser expandido o contraído. Además, los bloques pueden anidarse, lo que permite facilitar la organización de bloques de código extenso.

Tanto para el Formulario de Búsqueda (base), como para el resto de las implementaciones, se mantendrá un esquema no estricto, en el cual se utilizarán las siguientes categorías, con el siguiente orden:

Region	Descripción
Enumerados	Enum asociados a la clase.
Campos	Campos de las propiedades.
Propiedades	Propiedades de la clase, sin sus campos.
Constructores	Constructores de la clase, tanto públicos como privados.
Métodos públicos	Métodos públicos de la clase (etiqueta public).
Métodos de la clase	Métodos privados de la clase (etiequeta private).
Métodos para sobrescribir en las especializaciones	Métodos sobrescribibles de la clase (etiquetas protected virtual)
Eventos	Para Formularios y Controles de usuario, métodos que respondan a Eventos

A su vez, tanto para Formularios, como para Controles de usuario, en los casos de Métodos de la clase y Métodos para sobrescribir en las especializaciones, se emplearán las siguientes subdivisiones:

Region	Descripción
Métodos de Inicialización y Carga	Todos aquellos métodos que tengan funcionalidades específicamente destinadas a configurar aspectos, o realizar tareas que sólo se realicen en la apertura.
Métodos de acción	Todos aquellos métodos que estén directamente relacionados a dar respuesta a acciones específicas del usuario sobre algún componente (click en un botón, atajo de teclado, etc). No se debe confundir con métodos accesorios que ayuden a concretar una operación mayor.
Métodos de comportamiento	Métodos que permitan resolver cuestiones específicas, y tareas accesorias dentro de la clase, para concretar alguna tarea mayor.

Ejemplo para la región *Constructores*, de una clase con nombre "*ClaseDeEjemplo*":

[38] Documentación oficial de Region: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/preprocessor-directives/preprocessor-region

```
#region Constructores

public ClaseDeEjemplo()
{
    //…
}

#endregion // Constructores
```

Ejemplo de especialización de Formulario de Búsqueda (base)

Para clarificar el proceso de especialización de un *Formulario de Búsqueda (base)*, y a modo de ejemplo, se mostrarán en orden todos los componentes que integran la especialización, utilizando la entidad de datos de ejemplo *Mascota*.

Resumiendo, el esquema con el cual se relacionan los componentes, el proceso para especializar un *Formulario de Búsqueda (base)* implica:

1. Definir el *ViewModel* que representará los datos en la grilla del buscador.
2. Definir archivo de recursos para el *Formulario de Búsqueda*.
   2.1. Definir el título del *Formulario de Búsqueda*.
   2.2. Definir las claves para los campos del *ViewModel*.
3. Especializar la clase del *Formulario de Búsqueda (base)*.
   3.1. Indicar cual es el *ViewModel* de la grilla que será utilizado para representar los datos (el definido en 1.).
   3.2. Referencia el archivo de recursos, y la clave correspondiente para el título.
   3.3. Especializar los métodos de búsqueda (que consiguen los datos que deben ser mostrados de alguna capa subyacente, como los *Controladores*) que generan el traspaso de los datos de la capa subyacente, al *ViewModel*. En este punto, ya se tiene con una versión funcional del buscador.
4. Referencias claves del archivo de recursos, en los campos del *ViewModel* mediante *DataAnnotations*.
5. Traducir archivo de recursos.

Definición de ViewModel

Los datos que serán representador por el *ViewModel* para la entidad de datos *Mascota*, son en algunos casos una copia literal de los datos originales, y en otros casos implica algún procesamiento. A modo de referencia se cita la definición de la entidad de datos *Mascota*:

```
public class Mascota : IEntidadConFoto
{
    public Mascota()
    {
        this.Propietarios = new HashSet<Propietario>();
    }

    public int Id { get; set; }

    [Required]
    [MaxLength(150)]
    public string Nombre { get; set; }

    public DateTime FechaDeNacimiento { get; set; }
```

```
    public int TipoDeAnimalId { get; set; }
    public virtual TipoDeAnimal TipoDeAnimal { get; set; }


    [MaxLength(250)]
    public string Notas { get; set; }

    public byte[] Foto { get; set; }

    public byte[] FotosPequeña { get; set; }

    public virtual ICollection<Propietario> Propietarios { get; set; }
}
```

El *ViewModel* debe heredar de la clase *VM_FrmBusquedaBase*. Por convención su nombre inicia con '*VM_*' y continúa con el nombre de la entidad que lo consumirá, resultando '*VM_FrmBúsquedaMascota*', y se almacena en *ViewModels > FrmBúsqueda*. El *ViewModel* para la entidad de datos *Mascota* resulta:

```
public class VM_FrmBúsquedaMascota : VM_FrmBusquedaBase
{
    public VM_FrmBúsquedaMascota() { }

    private int id;
    [Display(AutoGenerateField = true)]
    public int Id
    {
        get { return id; }
        set
        {
            if (id != value)
            {
                id = value;
                OnPropertyChanged();
            }
        }
    }

    private string nombre;
    public string Nombre
    {
        get { return nombre; }
        set
        {
            if (nombre != value)
            {
                nombre = value;
                OnPropertyChanged();
            }
        }
    }

    private DateTime fechaDeNacimiento;
    public DateTime FechaDeNacimiento
    {
        get { return fechaDeNacimiento; }
        set
        {
            if (fechaDeNacimiento != value)
            {
                fechaDeNacimiento = value;
                OnPropertyChanged();
            }
        }
    }

    private string tipoDeAnimal;
```

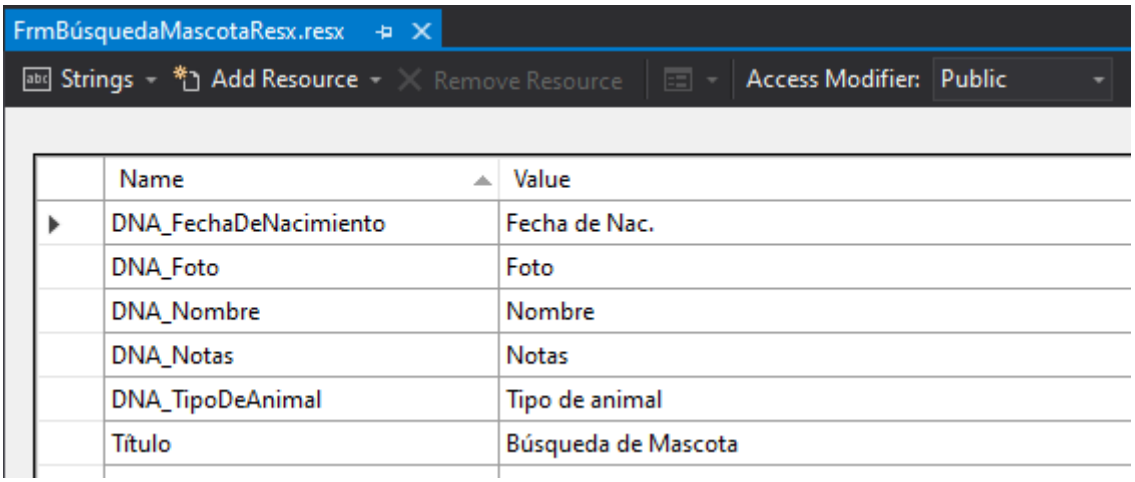
```
    public string TipoDeAnimal
    {
        get { return tipoDeAnimal; }
        set
        {
            if (tipoDeAnimal != value)
            {
                tipoDeAnimal = value;
                OnPropertyChanged();
            }
        }
    }

    private string notas;
    public string Notas
    {
        get { return notas; }
        set
        {
            if (notas != value)
            {
                notas = value;
                OnPropertyChanged();
            }
        }
    }

    private Bitmap foto;
    public Bitmap Foto
    {
        get { return foto; }
        set
        {
            if (foto != value)
            {
                foto = value;
                OnPropertyChanged();
            }
        }
    }
}
```

Notas como en la mayoría de los casos, los tipos de datos de la entidad origen (la entidad de datos *Mascota*) y el *ViewModel*, son los mismos. Pero en el caso del *TipoDeAnimal*, la entidad en el *ViewModel* es de tipo *String*, y la entidad de origen del tipo *TipoDeAnimal*. Esto es porque en tiempo de ejecución se hará una transformación conveniente de los datos para mostrarlos en pantalla.

Definición de archivo de recursos

Se debe definir una clave para el título del *Formulario de Búsqueda*, y una clave para cada propiedad del *ViewModel* que se quiera traducir (no es necesario traducir todas, de ser conveniente, pueden mostrar el nombre de la propiedad de forma literal).

Por convención su nombre inicia con el nombre de la entidad que lo consumirá, y finaliza con '*Resx*', resultando '*FrmBúsquedaMascotaResx*', y se almacena en *ArchivosDeRecurso > FrmBúsqueda*. El archivo de recursos para el *Formulario de Búsqueda* de la entidad de datos *Mascota* resulta:



Especialización del Formulario de Búsqueda

Para mantener a la vista sólo los elementos importantes, se eliminan los comentarios y las regiones de separación de código.

```
public partial class FrmBúsquedaMascota : FrmBúsquedaBase
{
    public FrmBúsquedaMascota()
    {
        // Referencia de ViewModel
        base.TipoDeDatoDelVM = typeof(VM_FrmBúsquedaMascota);
        // Referencia de Archivo de Recursos
        this.resxOwn = new ComponentResourceManager(typeof(FrmBúsquedaMascotaResx));
    }

    protected override void Buscar(IBindingList listaDeEntidadesGenerica, string[] columnasACargar)
    {
        // Obtención de entidades desde capa subyacente
        List<Mascota> listEntidades = CT_Mascota.ObtenerTodas();
        // Generación de lista vacía para ViewModels
        BindingList<VM_FrmBúsquedaMascota> listVM =
(BindingList<VM_FrmBúsquedaMascota>)listaDeEntidadesGenerica;
        // Transformación de entidades de datos en ViewModels
        foreach (Mascota entidadOrigen in listEntidades)
        {
            VM_FrmBúsquedaMascota vm = new VM_FrmBúsquedaMascota();
            this.LlenarVM(vm, entidadOrigen, columnasACargar);
            listVM.Add(vm);
        }
    }

    // Traspaso de datos, desde entidad de origen, a ViewModel
    private void LlenarVM(VM_FrmBúsquedaMascota vm, Mascota entidadOrigen, string[] columnasACargar)
    {
        // Propiedades de VM_Herencia
        if (columnasACargar.Contains(nameof(VM_FrmBúsquedaMascota.Id)))
        {
            vm.Id = entidadOrigen.Id;
        }
        if (columnasACargar.Contains(nameof(VM_FrmBúsquedaMascota.Nombre)))
        {
            vm.Nombre = entidadOrigen.Nombre;
        }
        if (columnasACargar.Contains(nameof(VM_FrmBúsquedaMascota.FechaDeNacimiento)))
        {
            vm.FechaDeNacimiento = entidadOrigen.FechaDeNacimiento;
        }
        if (columnasACargar.Contains(nameof(VM_FrmBúsquedaMascota.TipoDeAnimal)))
        {
```

```
            vm.TipoDeAnimal = LanguageHelpers.Localizar(entidadOrigen.TipoDeAnimal.Descripción);
        }
        if (columnasACargar.Contains(nameof(VM_FrmBúsquedaMascota.Notas)))
        {
            vm.Notas = entidadOrigen.Notas;
        }
        if (columnasACargar.Contains(nameof(VM_FrmBúsquedaMascota.Foto)))
        {
            if (entidadOrigen.FotosPequeña != null)
            {
                vm.Foto = (Bitmap)((new ImageConverter()).ConvertFrom(entidadOrigen.FotosPequeña));
            }
        }

        // Propiedades de VM_Base
        if (vm.EntidadOrigen == null)
        {
            vm.EntidadOrigen = entidadOrigen;
        }
    }

    // Configuración de título del formulario
    protected override void ConfigurarTituloDeFormulario()
    {
        return resxOwn.GetString(nameof(FrmBúsquedaMascotaResx.Título));
    }
}
```

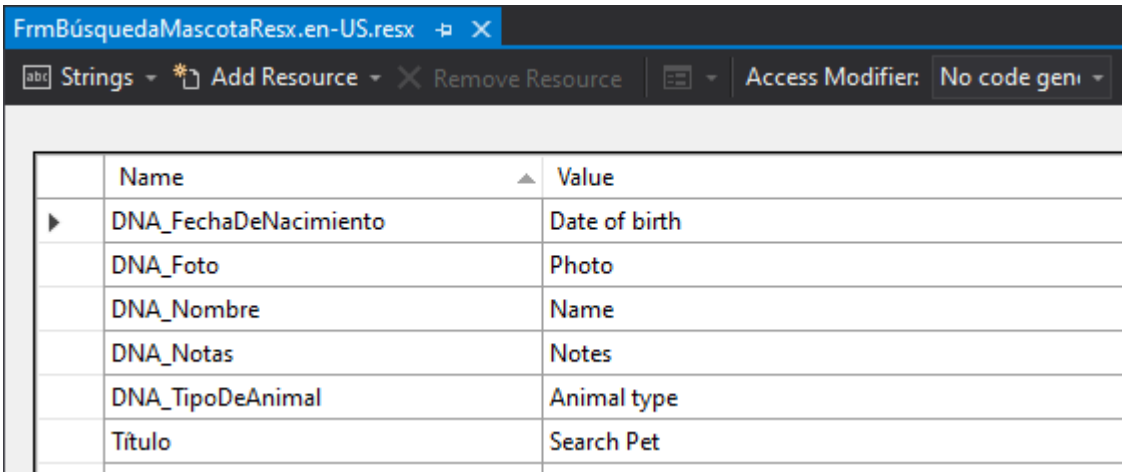
Vincular archivo de recursos con ViewModel

Al *ViewModel* definido previamente, se le adjuntan las referencias a las claves del archivo de recursos que debe utilizar para localizar sus propiedades.

```
private string nombre;
[Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_Nombre), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
public string Nombre {…}

private DateTime fechaDeNacimiento;
[Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_FechaDeNacimiento), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
public DateTime FechaDeNacimiento {…}

private string tipoDeAnimal;
[Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_TipoDeAnimal), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
public string TipoDeAnimal {…}

private string notas;
[Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_Notas), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
public string Notas {…}

private Bitmap foto;
[Display(Name = nameof(FrmBúsquedaMascotaResx.DNA_Foto), ResourceType =
typeof(FrmBúsquedaMascotaResx), AutoGenerateField = true)]
public Bitmap Foto {…}
```

Traducción de archivo de recursos

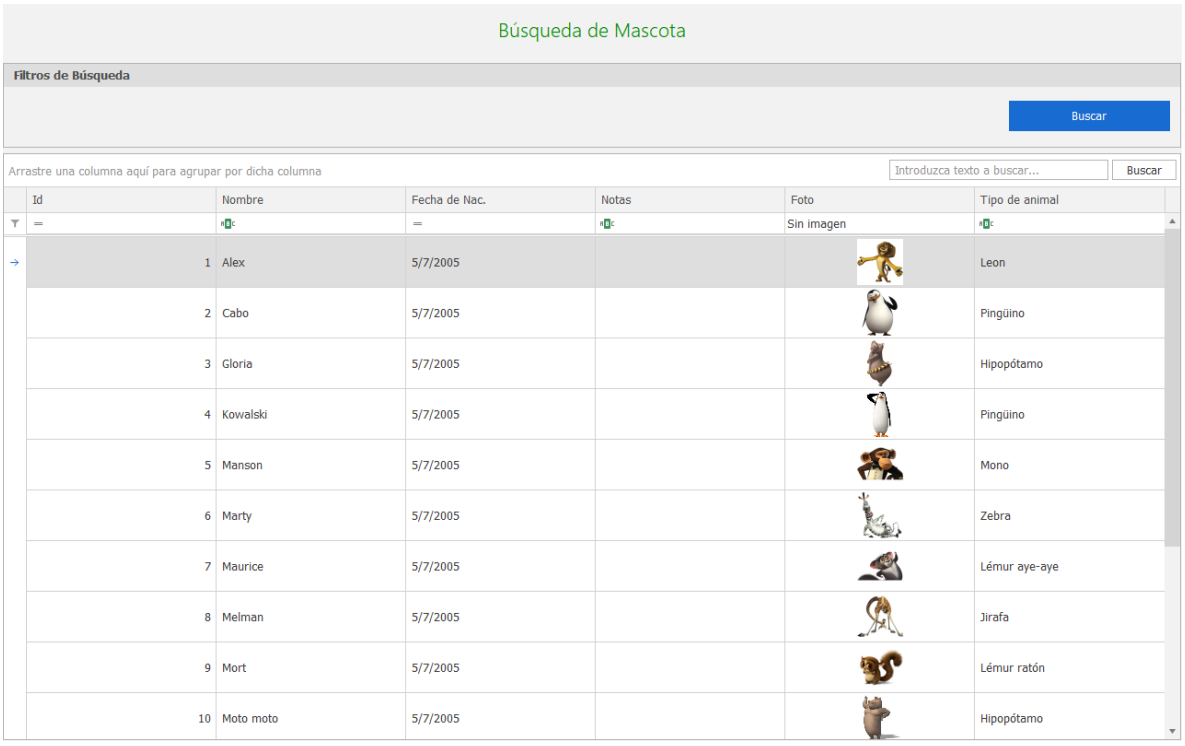
La traducción del archivo de recursos implica replicar la estructura del archivo original, agregando al final del nombre del archivo, la clave de localización del idioma correspondiente (en este caso, ‘es-US’.



Name	Value
DNA_FechaDeNacimiento	Date of birth
DNA_Foto	Photo
DNA_Nombre	Name
DNA_Notas	Notes
DNA_TipoDeAnimal	Animal type
Titulo	Search Pet

Ejecución

Con la definición de un *ViewModel* y un archivo de recursos, y añadiendo unas cincuenta líneas de código fáciles de comprender y replicar en el formulario especializado, se cuenta con un *Formulario de Búsqueda* que goza de todas las funcionalidades especificadas en la base, y que es sencillamente traducible a cualquier lenguaje.



User Story ID 6 - Formulario modal Aceptar-Cancelar (base)

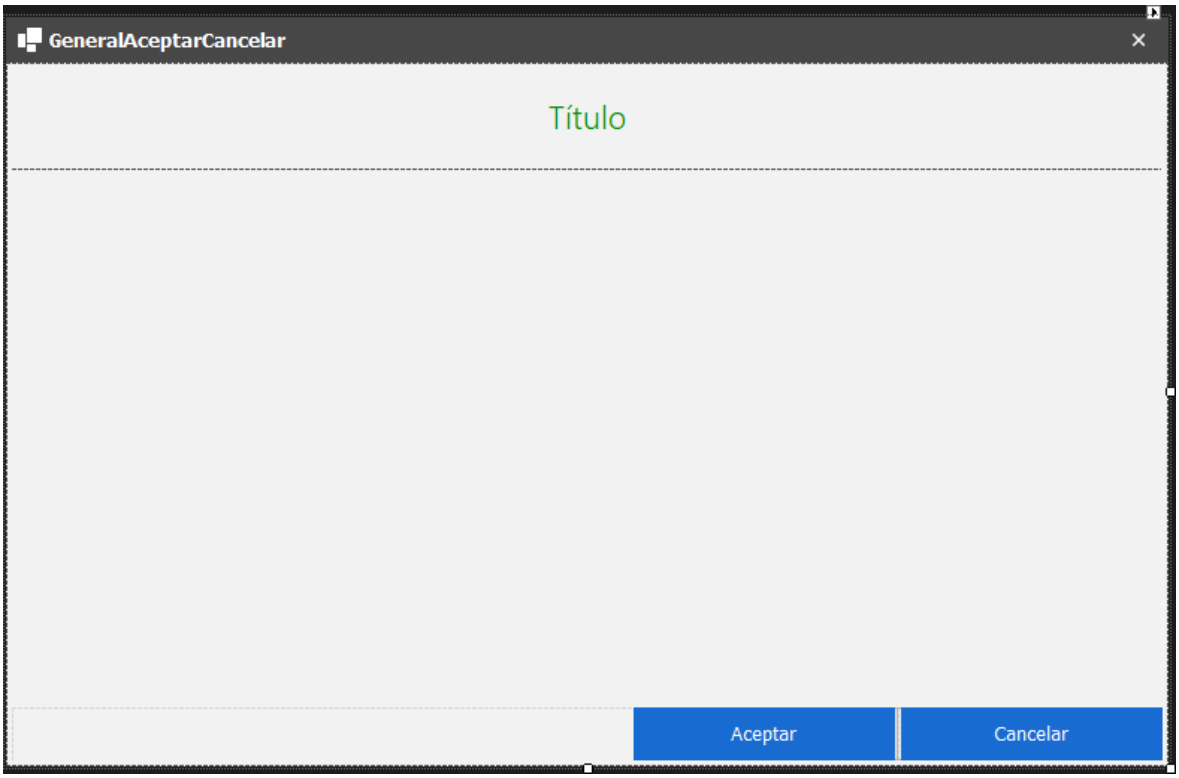
Interfaz de Formulario modal Aceptar-Cancelar (base)

Se desarrolló el Formulario modal Aceptar-Cancelar (base), acorde a las especificaciones dadas en wireframes. Además, se hizo la plantilla responsiva a los cambios de tamaño de pantalla,

de forma tal que el largo del lienzo siempre se adecuará a las dimensiones que el desarrollador le asigne.

Todos los elementos gráficos del formulario (etiqueta de título, botones Aceptar y Cancelar, etc) excepto la zona central del lienzo (entre la etiqueta del título y la barra con los botones inferiores) es privada, esto implica que, al ser especializado el formulario de base, la especialización no podrá alterar dichos elementos.

En lo que respecta a elementos gráficos, las especializaciones sólo tendrán la libertad de añadir elementos gráficos en el centro del formulario, tal como se planteó en la especificación.



Lógica del Formulario modal Aceptar-Cancelar (base)

La lógica fundamental del *Formulario modal Aceptar-Cancelar (base)* está compuesta por ciertos aspectos, los cuales serán explicados a continuación: configuración del título, y lógica de los botones Aceptar y Cancelar; todos ellos dependen de datos añadidos en la especialización.

Título del formulario

El título del formulario se configurará en diferentes lugares al mismo tiempo (etiqueta principal del formulario, título de la pestaña, etc.). Las tareas de colocar el contenido del título en diferentes lugares pertinentes, es tarea del formulario base, pero el contenido del título será establecido en los formularios que especialicen al formulario base.

Siguiendo la lógica empleada en el Formulario de Búsqueda (base), en el formulario base se ha declarado un método virtual (lo que permite que sea sobrescrito en la especialización):

```
protected virtual string ConfigurarTituloDeFormulario() { }
```

De esta manera, el programador tiene la libertad de añadir cierta lógica dentro del método especializado, como, por ejemplo, acceder a un archivo de recursos, obtener la información desde alguna capa inferior, etc.

Botón Aceptar

La lógica del botón *Aceptar*, la cual es representada por el método que responde a su evento click *sbAceptar_Click*, se base en resultado de la ejecución de un método especializado por la clase que hereda al *Formulario modal Aceptar-Cancelar (base)*. Si el resultado de dicha operación es verdadero, la lógica en el formulario base es ejecutada.

```
private void sbAceptar_Click(object sender, EventArgs e)
{
    if (this.Aceptar() == true)
    {
        // Desecho el formulario
        this.Dispose();
    }
}
```

Botón Cancelar

La lógica del botón *Aceptar*, la cual es representada por el método que responde a su evento click *sbCancelar_Click*, se base en resultado de la ejecución de un método especializado por la clase que hereda al *Formulario modal Aceptar-Cancelar (base)*. Si el resultado de dicha operación es verdadero, la lógica en el formulario base es ejecutada.

```
private void sbCancelar_Click(object sender, EventArgs e)
{
    if (this.Cancelar() == true)
    {
        // Desecho el formulario
        this.Dispose();
    }
}
```

User Story ID 7 - Formulario de búsqueda (base): ordenar, ocultar y mostrar columnas

Estrategia para persistir la configuración de las columnas de la grilla

Como se explicó previamente en el *Uso de ViewModel y GridControl*, se emplea una estrategia para configurar la grilla a través de *ViewModels*, lo cual permite interactuar de forma genérica y estandarizada con la misma (todos los programadores lo harán del mismo modo), sin tener incluso conocimiento sobre el funcionamiento interno de la grilla.

Se podría decir entonces, que el sentido de la información, es desde un *ViewModel* particular hacia una determinada grilla, la cual representa las configuraciones del mismo (tipos de datos de sus propiedades, nombre de las columnas dependiendo del valor de un archivo de recursos, etc).

Sin embargo, la grilla soporta la posibilidad de configurar ciertos aspectos gráficos, como el orden de sus columnas, y el estado de visibilidad de las mismas, por lo cual surge la necesidad de

persistir de alguna manera, la información que se genera esta vez, desde la grilla con respecto a las propiedades del *ViewModel*.

Siendo capaz de persistir la configuración de cada columna, relacionada a cada propiedad de cada *ViewModel*, es posible recuperar la configuración para cada columna, y aplicarla a la grilla luego de que se le vinculó un *ViewModel* particular.

De forma general, la estrategia utilizada es la siguiente:

1) Una entidad de datos permite persistir la información de un *ViewModel*, con cada una de sus propiedades (columnas en la grilla), y, además, atributos tales como su estado de visibilidad, y su orden.
2) Un algoritmo al inicio de la aplicación, cuando la aplicación inicia en modo *debug*, mapea cada *ViewModel* de formulario de búsqueda a la base datos, con información genérica (columnas siempre visibles, y orden equivalente al orden de las propiedades en la clase).
3) En la apertura del formulario de búsqueda, se obtienen desde la base de datos todas las propiedades del *ViewModel* asociado, y la configuración de cada propiedad (columna) es aplicada a la grilla.
4) En el cierre del formulario, si hubo cambios en la configuración de las columnas (orden o estado de visibilidad), los mismos se persisten en la base de datos.

Extracción de propiedades de clases en tiempo de ejecución

Para poder persistir cualquier información relacionada a las propiedades una clase (concretamente, a las propiedades de un *tipo de dato*), es necesario contar con el nombre del tipo de dato, y el nombre que recibe la propiedad.

En .Net, es posible obtener esa información en tiempo de ejecución, utilizando *Reflection*[39], que este compuesto por diversas herramientas que permiten recuperar información sobre los ensamblados cargados, y los tipos de datos definidos en ellos, como clases, interfaces, etc.

Un ejemplo del uso de *Reflection* que permite comprender su potencial, es el autocompletado en el editor de código fuente, cuando al escribir el nombre de una clase, inmediatamente sugiere el uso de los métodos de la misma; la estrategia que utilizamos, es en efecto, la misma.

El nombre de una clase, simplemente puede ser recuperado desde el atributo *Name* [40]del *tipo de datos*. La información de las propiedades de una clase es recuperada con el método *GetProperties*[41] en el tipo de datos, el cual retorna el tipo de datos *PropertyInfo*[42].

[39] Documentación oficial de Reflection: https://docs.microsoft.com/en-us/dotnet/framework/reflection-and-codedom/reflection
[40] Documentación oficial de Name, en Reflection: https://docs.microsoft.com/en-us/dotnet/api/system.reflection.memberinfo.name
[41] Documentación oficial de GetProperties, en Reflection: https://docs.microsoft.com/en-us/dotnet/api/system.type.getproperties
[42] Documentación oficial de PropertyInfo, en Reflection: https://docs.microsoft.com/en-us/dotnet/api/system.reflection.propertyinfo

PropertyInfo expone los atributos de una propiedad y proporciona acceso a los metadatos de la misma. Puntualmente, de todos los atributos de la propiedad es necesario el nombre, el cual se obtiene mediante la propiedad *Name*.

Obtención programatical de nombre de clase:

```
// Obtengo el objeto Type (tipo de datos), desde el nombre de la clase
Type tipoDeDatosVM_Propietario = typeof(VM_Propietario);
// Obtengo el nombre de la clase (string) en tiempo de ejecución
string nombreDeLaClase = tipoDeDatosVM_Propietario.Name;
```

Obtención programatical de propiedades de una clase:

```
// Obtengo metadatos de las propiedades de la clase
PropertyInfo[] propertyInfos =
tipoDeDatosVM_Propietario.GetProperties(System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.Instance | System.Reflection.BindingFlags.DeclaredOnly);
// De todos los metadados, conservo sólo el nombre de las propiedades
string[] nombresDePropiedades = propertyInfos.Select(item => item.Name).ToArray();
```

Atributos de visibilidad y orden en las columnas de la grilla

La grilla empleada, *GridControl* de *DevExpress*, permite gestionar los atributos de visibilidad y orden de sus columnas, independientemente de que las columnas hayan sido autogeneradas mediante *DataBinding*, o generadas manualmente.

La forma de acceder a las propiedades de una columna, es a través de su nombre interno (no confundir con el texto que se muestra en el encabezado de la columna). Cuando se utiliza *DataBinding*, este campo representa el nombre de la propiedad del *ViewModel* que se quiere vincular con la columna.

La grilla, tiene un atributo que retorna una colección con todas sus columnas, y es obtener una columna particular de dicha colección, empleando su nombre (el nombre de la propiedad de la clase, cuando se emplea *DataBinding*).

La colección de columnas se accede mediante el método *Columns*[43], en la grilla, lo cual retorna un *GridColumCollection*[44]. El acceso a una columna particular, dentro de la colección de columnas, retorna un elemento de tipo *GridColumn*[45], que es la columna en la cual se realizarán las operaciones de visibilidad y orden.

La forma de en la cual se identifican las columnas, dentro de la colección de columnas, es a través del atributo FieldName[46].

[43] Documentación oficial de Columns, para GridControl, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.Views.Base.ColumnView.Columns
[44] Documentación oficial de GridColumnCollection, para GridControl, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.Columns.GridColumnCollection
[45] Documentación oficial de GridColumn, para GridControl, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.Columns.GridColumn
[46] Documentación oficial de FieldName, para GridControl, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.Columns.GridColumn.FieldName

El estado de visibilidad se modifica a través de la propiedad *Visible*[47], de tipo booleano. El orden de las columnas puede alterarse mediante la propiedad *VisibleIndex*, de tipo entera. Las columnas ocultas (*Visible = false*), recibirán un *VisibleIndex = -1*.

```
// Acceso a la columna correspondiente a la propiedad ApellidoYNombre
GridColumn columna = grilla.Columns["NombreYApellido"];

// Ocultar columna
columna.Visible = false;

// Mostrar columna
columna.Visible = true;

// Establecer columna en 2do lugar (base 0)
columna.VisibleIndex = 1;
```

Entidad de datos que representa propiedades de ViewModel

Para persistir los atributos visuales de la grilla, relacionados a las propiedades de los *ViewModel*, o cualquier otra información que sea merecedora de persistencia en el futuro, se desarrolló una clase que es manejada por el mapeador objeto-relacional, y que tiene como clave primaria el nombre de la clase, y el nombre de la propiedad.

La entidad de datos, con sus atributos fundamentales, sumados los atributos relacionados a los aspectos visuales de la grilla resulta:

```
public class CampoDeVM
{
    [Key, Column(Order = 0)]
    public string NombreDeVM { get; set; }

    [Key, Column(Order = 1)]
    public string NombreDeCampoDeVM { get; set; }

    #region Atributos de la grilla

    public bool Visible { get; set; }

    public int? VisibleIndex { get; set; }

    #endregion // Atributos de la grilla
}
```

Mapeo automático de campos de ViewModel, a base de datos

Dada una lista de *ViewModels*, que representan columnas en la grilla del *Formulario de Búsqueda*, es necesario persistir de forma automática los pares *NombreDeVM-NombreDeCampoDeVM*, para evitar la tarea manual.

Además, es necesario, que, tras cada añadido o eliminación de propiedades a los *ViewModels*, o añadido o eliminación de *ViewModels*, los mismos mantengan consistencia con lo persistido.

[47] Documentación oficial de Visible, para GridControl, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.Columns.GridColumn.Visible

Por esta razón, el algoritmo de mapeo heredará de la interfaz para las *Tareas de Inicio*, se será sólo utilizado cuando el entorno se ejecute en modo *Debug*.

```
internal static void Ejecutar()
{
    List<Type> typesViewModelFrmBusqueda = new List<Type>();

    // Lista de VM (tipos de dato) de FrmBusqueda a consolidar con DB
    typesViewModelFrmBusqueda.Add(typeof(VM_Propietario));
    typesViewModelFrmBusqueda.Add(typeof(VM_Mascota));

    // Obtengo VM (nombres) en tiempo de ejecución
    string[] nombresVMEnRuntime = typesViewModelFrmBusqueda.Select(item => item.Name).ToArray();

    // Obtengo VM actualmente almacenados en DB
    string[] nombresVMEnDB = CT_CampoViewModelFrmBusqueda.GroupBy(item =>
item.NombreDeVM).Select(group => group.Key).ToArray();

    // Remover aquellos VM que estén en DB, y que no estén en el listado actual
    string[] nombresVMARemoverDeDB = nombresVMEnDB.Where(p1 => !nombresVMEnRuntime.Any(p2 => p2 ==
p1)).ToArray();
    foreach (string nombreVMARemoverDeDB in nombresVMARemoverDeDB)
    {
        CampoViewModelFrmBusqueda[] camposViewModelFrmBusquedaARemover =
CT_CampoViewModelFrmBusqueda.CamposViewModelFrmBusqueda.Where(c => c.NombreDeVM ==
nombreVMARemoverDeDB).ToArray();

        .RemoveRange(camposViewModelFrmBusquedaARemover);
    }

    foreach (Type typeViewModelFrmBusqueda in typesViewModelFrmBusqueda)
    {
        // Obtengo nombre de campos del ViewModel en Runtime
        PropertyInfo[] propertyInfos =
typeViewModelFrmBusqueda.GetProperties(System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.Instance | System.Reflection.BindingFlags.DeclaredOnly);
        string[] nombresCampoEnRuntime = propertyInfos.Select(item => item.Name).ToArray();

        // Obtengo nombres de campos del ViewModel almacenados en DB
        string[] nombresCamposEnDB = CT_CampoViewModelFrmBusqueda.CamposViewModelFrmBusqueda.Where(c
=> c.NombreDeVM == typeViewModelFrmBusqueda.Name).Select(c => c.NombreDeCampo).ToArray();

        // Remover aquellos Campos que estén en DB, y que no estén en el listado actual
        string[] camposARemoverDeDB = nombresCamposEnDB.Where(p1 => !nombresCampoEnRuntime.Any(p2 =>
p2 == p1)).ToArray();
        foreach (string camposRemoverDeDB in camposARemoverDeDB)
        {
            CampoViewModelFrmBusqueda[] camposViewModelFrmBusquedaARemover =
CT_CampoViewModelFrmBusqueda.CamposViewModelFrmBusqueda.Where(c => c.NombreDeVM ==
typeViewModelFrmBusqueda.Name && c.NombreDeCampo == camposRemoverDeDB).ToArray();
            CT_CampoViewModelFrmBusqueda.RemoveRange(camposViewModelFrmBusquedaARemover);
        }

        // Agrego aquellos Campos que no estén en DB, y que estén en el listado actual
        string[] camposAAgregarADeDB = nombresCampoEnRuntime.Where(p1 => !nombresCamposEnDB.Any(p2
=> p2 == p1)).ToArray();
        int visibleIndex = 0;
        foreach (string campoAAgregarADeDB in camposAAgregarADeDB)
        {
            CampoViewModelFrmBusqueda campoViewModelFrmBusqueda = new CampoViewModelFrmBusqueda();
            campoViewModelFrmBusqueda.NombreDeVM = typeViewModelFrmBusqueda.Name;
            campoViewModelFrmBusqueda.NombreDeCampo = campoAAgregarADeDB;
            campoViewModelFrmBusqueda.Visible = true;

            CT_CampoViewModelFrmBusqueda.Add(campoViewModelFrmBusqueda);
        }
    }
}
```

Aplicación de configuración de columnas automática en Formulario de Búsqueda (base)

Al realizar la apertura de un *Formulario de Búsqueda*, si existen configuraciones de columnas guardas relacionadas al mismo, dichas configuraciones se aplican automáticamente en la apertura del formulario.

El *Formulario de Búsqueda (base)* incorpora un algoritmo, el cual es llamado en su constructor. El algoritmo consiste en recuperar desde la base de datos las configuraciones de las columnas guardas, en función del tipo de datos del *ViewModel* asociado, cuyo nombre se utiliza como clave primaria.

El algoritmo consiste en recorrer las columnas de la grilla, y aplicar en ellas las configuraciones correspondientes según lo persistido.

```
private void AplicarConfiguracionAColumnas()
{
    // Obtendo información de las columnas desde DB
    CampoViewModelFrmBusqueda[]  camposViewModelFrmBusqueda = CT_CamposViewModelFrmBusqueda.Where(c
=> c.NombreDeVM == tipoDeDatoDelVM.Name).ToArray();

    // Detengo pintado en la grilla (mientras se actualiza su contenido)
    this.gcResultadosDeBusqueda.BeginUpdate();

    foreach (CampoViewModelFrmBusqueda campoViewModelFrmBusqueda in camposViewModelFrmBusqueda)
    {
        if (campoViewModelFrmBusqueda.Visible == true)
        {
            // Por defecto las columnas son visibles

            // Establezco el orden de las columnas
            if (campoViewModelFrmBusqueda.VisibleIndex != null)
            {

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].VisibleIndex =
(int)campoViewModelFrmBusqueda.VisibleIndex;
            }
        }
        else
        {
            this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].Visible =
false;
        }
    }

    // Reanudo el pintado en la grilla (mientras se actualiza su contenido)
    this.gcResultadosDeBusqueda.EndUpdate();
}
```

Guardado de configuración de columnas en Formulario de Búsqueda (base)

Al cerrar un *Formulario de Búsqueda*, se dispara un evento en el cual se realiza la comprobación de cambios en la configuración de columnas, contrastando la información persistida con la actual en tiempo de ejecución.

Si hay cambios en la configuración de las columnas, se pregunta al usuario si desea almacenar esos cambios, en cuyo caso se actualiza la información persistida. De esta manera, en la próxima apertura del formulario, el usuario verá la grilla tal y como la configuró en su último uso.

```
private bool GuardarConfiguraciónDeColumnas()
{
    bool blnRetorno = true;

    // Obtengo los GridColums actualmente utilizados en la grilla
    GridColumn[] gridColumns = this.gvResultadosDeBusqueda.Columns.ToArray();

    /* Para GridColumn, busco su CampoViewModelFrmBusqueda asociado y actualizo todas las
     * propiedades de CampoViewModelFrmBusqueda con la información de GridColumn. */
    foreach (GridColumn gridColumn in gridColumns)
    {
        CampoViewModelFrmBusqueda campoViewModelFrmBusqueda =
this.camposViewModelFrmBusqueda.First(c => c.NombreDeCampo == gridColumn.FieldName);

        campoViewModelFrmBusqueda.Visible = gridColumn.Visible;
        campoViewModelFrmBusqueda.VisibleIndex = gridColumn.VisibleIndex;
    }

    // Persisto las actualizaciones
    try
    {
        CT_CamposViewModelFrmBusqueda.UpdateRange(this.camposViewModelFrmBusqueda);
    }
    catch (Exception)
    {
        blnRetorno = false;
    }

    return blnRetorno;
}
```

Búsqueda eficiente

Inicialmente, la estrategia propuesta permitía efectuar el llenado de los *ViewModels* completos (todas sus propiedades) sin importar si luego serían mostrados o no en la grilla.

A modo de ejemplo, pensar en un *ViewModel* con diez propiedades, de las cuales luego se muestran sólo tres columnas. Siete de estas propiedades involucraron un procesamiento de fondo, que vas desde la recuperación de datos desde la capa de persistencia, hasta la adaptación de los mismos para ser acordemente mostrados en una celda de la grilla, para luego simplemente permanecer ocultos.

Para permitir cargar sólo la información necesaria, es decir, la información de aquellas columnas que están siendo visibles, a la estrategia original se le añade la posibilidad de saber cuáles son las columnas actualmente visibles.

```
protected override void Buscar(IBindingList listaDeEntidadesGenerica , string[] columnasACargar)
{
    // Lista de entidades que serán mapeadas a su correspondiente ViewModel:
    // (Las cuales serán recuperadas de la capa de Lógica de Negocios, por
    // ejemplo)
    List<Propietario> listaPropietarios;

    // Casteo de la lista genérica recibida por parámetro, al tipo de lista
    // correspondiente según el ViewModel:
    BindingList<VM_Propietario> listaVMs = (BindingList<VM_Propietario>)listaDeEntidadesGenerica;

    // Lógica de mapeo de la lista de entidades, a los ViewModel
    // Por ejemplo:
    List<VM_Propietario> entidadesMapeadas = this.Mapear(listaPropietarios, columnasACargar);
    listaVMs.AddRange(entidadesMapeadas);
}
```

En el ejemplo anterior, la diferencia con la primera estrategia planteada radica en el método *Mapear*, en el cual se genera el traspaso y adecuación de la información desde los orígenes de datos, al *ViewModel* empleado en la grilla. En el método *Mapear*, simplemente se debe consultar qué columnas deben mapearse, e ignorar el resto.

El método *Buscar* será ejecutado por única vez, tras cada vez que se presione el botón "Buscar". Si el usuario hace visible una columna que estaba oculta, dicha columna no tiene aún datos asociados en el *ViewModel*. Para manejar estas situaciones, los formularios que especializan el *Formulario de Búsqueda (base)* tiene que sobrescribir el método *Actualizar*.

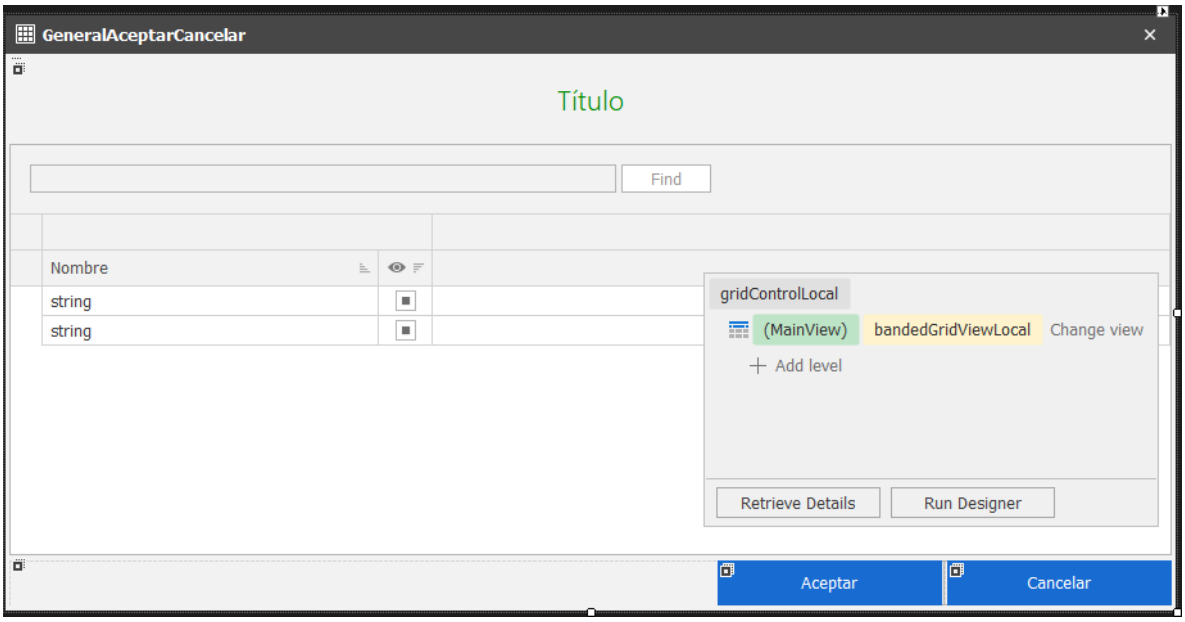
El método *Actualizar* tiene una lógica similar al método *Buscar*, pero existen dos diferencias fundamentales. Mientras el método *Buscar* recibe un arreglo con columnas a cargar, el método *Actualizar* recibe sólo un elemento, acorde a la columna que debe ser actualizada. Por otro lado, en el método *Buscar*, son creadas las instancias de *ViewModels*, mientras que, en *Actualizar*, se actualizan las mismas.

Dado que se opera con *DataBinding*, en el momento en que los datos de alguna instancia de *ViewModel* son actualizados, la actualización se refleja en la grilla.

```
protected override void Actualizar(IBindingList listaDeEntidadesGenerica , string columnaACargar)
{
    // Lista de entidades que serán mapeadas a su correspondiente ViewModel:
    // (Las cuales serán recuperadas de la capa de Lógica de Negocios, por
    // ejemplo)
    List<Propietario> listaPropietarios;

    // Casteo de la lista genérica recibida por parámetro, al tipo de lista
    // correspondiente según el ViewModel:
    BindingList<VM_Propietario> listaVMs = (BindingList<VM_Propietario>)listaDeEntidadesGenerica;

    foreach (VM_Propietario vm in listaVMs)
    {
        // Ejemplo para la propiedad ApellidoYNombre de VM_Propietario
        if (columnaACargar == (VM_Propietario.ApellidoYNombre)))
        {
            // Llenat propiedad con datos del origen de datos, por ej:
            vm.ApellidoYNombre = listaPropietarios.First(p => p.Id == vm.Id).ApellidoYNombre;
        }
    }
}
```

Formulario modal Aceptar-Cancelar: Configurar grilla

Siguiendo parte de las especificaciones funcionales del *Formulario de búsqueda: Configurar plantilla*, se desarrolló el *Formulario modal Aceptar-Cancelar: Configurar grilla*, dado que se trata de una versión preliminar, que sólo se aboca al orden y visibilidad de las columnas, y que no da tratamiento a plantillas (será realizado en incrementos posteriores).



Como estrategia para gestionar la grilla que permite configurar el orden y visibilidad de las columnas, se empleó el uso de un *ViewModel* con *DataBinding*, de la misma manera que se está realizando en los *Formularios de Búsqueda*, sólo que, para este caso, se trata de sólo un *ViewModel* de uso interno del formulario.

El *ViewModel* consta de un constructor que recibe como parámetro un *GridColumn*, correspondiente a una columna de la grilla del *Formulario de Búsqueda* que se está configurando.

En el *ViewModel* se cargan las tres propiedades fundamentales que se están gestionando en cada columna: *FieldName*, *Visible*, y *VisibleIndex*. El contenido de cada propiedad se extrae del *GridColumn* que se recibe por parámetro en el constructor, y para cada propiedad, a su vez, se guarda una copia del valor inicial, lo que permitirá deshacer los cambios efectuados.

Además, el *ViewModel* implementa una propiedad pública con nombre *SubirColumna* que permite desplazar una columna. El análogo, para conseguir bajar una columna, se efectúa subiendo la columna anterior.

```
class VM : INotifyPropertyChanged
{
    public VM(GridColumn gridColumn)
    {
        // Valores visibles en el VM
        this.fieldName = gridColumn.FieldName;
        this.visible = gridColumn.Visible;
        this.visibleIndex = gridColumn.VisibleIndex;

        // Valores de backup para el VM (ocultos)
        this.backupVisibleIndex = gridColumn.VisibleIndex;

        // GridColumn que se está vinculando con el VM
        this.gridColumn = gridColumn;
    }

    private GridColumn gridColumn;
    public GridColumn GridColumn
    {
        get { return gridColumn; }
```

```
    }

    private string fieldName;
    public string FieldName
    {
        get { return fieldName; }
    }

    private bool visible;
    public bool Visible
    {
        get { return visible; }
        set
        {
            if (visible != value)
            {
                visible = value;

                OnPropertyChanged();

                if (value == true)
                {
                    gridColumn.VisibleIndex = this.gridColumn.View.VisibleColumns.Count;
                }
                else
                {
                    gridColumn.Visible = false;
                }
            }
        }
    }

    private int visibleIndex;
    public int VisibleIndex
    {
        get { return visibleIndex; }
        set
        {
            if (visibleIndex != value)
            {
                visibleIndex = value;

                OnPropertyChanged();
            }
        }
    }


    public event PropertyChangedEventHandler PropertyChanged;
    protected void OnPropertyChanged([CallerMemberName] string propertyName = "")
    {
        if (PropertyChanged != null)
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }

    // Campos para valores de backup
    private int backupVisibleIndex;
    public int BackupVisibleIndex { get { return backupVisibleIndex; } }

    #region Metodos Públicos

    public void SubirColumna()
    {
        this.gridColumn.VisibleIndex = this.VisibleIndex - 1;
    }

    #endregion // Metodos Públicos
}
```

Dado que todos los elementos de la secuencia emplean *DataBinding*, cada cambio efectuado en la presente interfaz, serán inmediatamente visualizados en la grilla del *Formulario de Búsqueda* que se está configurando.

El botón “Aceptar” no tiene comportamiento, dado que cada cambio efectuado se refleja inmediatamente en las columnas. En cambio, el botón “Cancelar” se vale de las copias de los valores iniciales para deshacer los cambios.

Al hacer click con el botón derecho sobre cada fila, se despliega un popup que permite subir o bajar las filas, considerando si no se trata de situaciones de frontera, en las cuales una fila ya está en la primera o última posición.

```
private void bandedGridViewLocal_PopupMenuShowing(object sender,
DevExpress.XtraGrid.Views.Grid.PopupMenuShowingEventArgs e)
{
    // Si se hizo click derecho en una fila:
    if (e.HitInfo.InRow)
    {
        // Obtengo VM asociado a la fila en la que se hizo click derecho
        VM vm = (VM)bandedGridViewLocal.GetFocusedRow();

        if (vm != null)
        {
            // Habilito/Deshabilito botón "Subir columna"
            if (vm.VisibleIndex > 0)
            {
                this.bbiSubirColumna.Enabled = true;
            }
            else
            {
                this.bbiSubirColumna.Enabled = false;
            }

            // Habilito/Deshabilito botón "Bajar columna"
            int cantidadDeColumnasVisibles = this.vms.Where(i => i.VisibleIndex != -1).Count();
            if (vm.VisibleIndex > -1 && vm.VisibleIndex < cantidadDeColumnasVisibles - 1)
            {
                this.bbiBajarColumna.Enabled = true;
            }
            else
            {
                this.bbiBajarColumna.Enabled = false;
            }
        }

        // Obtengo el punto en el que se hizo click, y muestro el popup
        System.Drawing.Point p2 = Control.MousePosition;
        this.popupMenu1.ShowPopup(p2);
    }
}
```

User Story ID 8 - Formulario de búsqueda: Configurar plantilla

Atributos estéticos en las columnas de la grilla

Anteriormente se trabajó en los atributos de visibilidad y orden en las columnas de la grilla, por los cuales surge la necesidad de persistir información asociada a los *ViewModels* de los *Formularios de Búsqueda*.

Por esta razón se desarrolló una entidad de datos que permite persistir información anexa a las propiedades de un *ViewModel* (columnas de la grilla), se desarrolló un algoritmo de mapeo al inicio de la aplicación, y se implementaron algoritmos que aplican las configuraciones persistidas a las grillas de los formularios, y guardan sus cambios en caso de existir.

La grilla empleada permite efectuar ciertas configuraciones gráficas a sus columnas, brindando atributos para configurar la estética de las columnas y de las cabeceras, por separado.

Tanto las cabeceras como las celdas, admiten la configuración de su color de fondo y fuente, y tipo de fuente (negrita, cursiva, subrayada, y regular).

Las configuraciones de apariencia de las cabeceras, se encuentras dentro de la propiedad *AppearanceHeader*[48], y las propiedades de apariencia de las celdas, dentro de *AppearanceCell*[49].

Dentro de las propiedades de apariencia, pueden aplicarse configuraciones para uno o dos colores de fondo (lo cual proporciona un degradé), con *BackColor*[50] y *BackColor2*[51], y colores de fuente con *ForeColor*[52].

El estilo de fuente se configura mediante la propiedad *Font*[53], la cual es del tipo de datos *Font*[54] de .Net. Para configurar una fuente, se debe emplear el enumerado *FontStyle*. *FontStyle* está decorado con la etiqueta *Flags*[55], atributo que permite una combinación bit a bit de los valores de sus miembros.

Actualización de entidad de datos que representa propiedades de ViewModel

* Es importante destacar que se actualizó el nombre de la entidad de datos *CampoDeVM* a *CampoViewModelFrmBusqueda*.

Para soportar las nuevas configuraciones gráficas de las columnas de la grilla, la entidad de datos es actualizada con las propiedades que representan las configuraciones gráficas de cabeceras y celdas, de la siguiente manera:

---

[48] Documentación oficial de AppearenceHeader, en GridControl: https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.Columns.GridColumn.AppearanceHeader

[49] Documentación oficial de AppearenceCell, en GridControl: https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.Columns.GridColumn.AppearanceCell

[50] Documentación oficial de BackColor, en GridControl: https://docs.devexpress.com/WindowsForms/DevExpress.Utils.AppearanceObject.BackColor

[51] Documentación oficial de BackColor, en GridControl: https://docs.devexpress.com/WindowsForms/DevExpress.Utils.AppearanceObject.BackColor2

[52] Documentación oficial de ForeColor, en GridControl: https://docs.devexpress.com/WindowsForms/DevExpress.Utils.AppearanceObject.ForeColor

[53] Documentación oficial de Font, en GridControl: https://docs.devexpress.com/WindowsForms/DevExpress.Utils.AppearanceObject.Font

[54] Documentación oficial de Font, en .Net: https://docs.microsoft.com/en-us/dotnet/api/system.drawing.font

[55] Documentación oficial de FlagsAttribute, en .Net: https://docs.microsoft.com/en-us/dotnet/api/system.flagsattribute

```
public class CampoViewModelFrmBusqueda
{
    [Key, Column(Order = 0)]
    [MaxLength(100)]
    public string NombreDeVM { get; set; }

    [Key, Column(Order = 1)]
    [MaxLength(100)]
    public string NombreDeCampo { get; set; }

    public bool Visible { get; set; }

    public int? VisibleIndex { get; set; }

    // Propiedades de la cabecera

    public int? HeaderBackColor { get; set; }

    public int? HeaderForeColor { get; set; }

    public float? HeaderFontSize { get; set; }

    public bool HeaderFontBold { get; set; }

    public bool HeaderFontItalic { get; set; }

    public bool HeaderFontUnderline { get; set; }

    // Propiedades de las celdas

    public int? CellBackColor { get; set; }

    public int? CellBackColor2 { get; set; }

    public int? CellForeColor { get; set; }

    public float? CellFontSize { get; set; }

    public bool CellFontBold { get; set; }

    public bool CellFontItalic { get; set; }

    public bool CellFontUnderline { get; set; }
}
```

Actualización de algoritmo de mapeo automático de campos de ViewModel, a base de datos

Para soportar las nuevas configuraciones gráficas de las columnas de la grilla, al algoritmo inicial se le añaden las propiedades de las cabeceras y celdas de la grilla.

```
internal static void Ejecutar()
{
    List<Type> typesViewModelFrmBusqueda = new List<Type>();

    // Lista de VM (tipos de dato) de FrmBusqueda a consolidar con DB
    typesViewModelFrmBusqueda.Add(typeof(VM_Propietario));
    typesViewModelFrmBusqueda.Add(typeof(VM_Mascota));

    // Obtengo VM (nombres) en tiempo de ejecución
    string[] nombresVMEnRuntime = typesViewModelFrmBusqueda.Select(item => item.Name).ToArray();

    // Obtengo VM actualmente almacenados en DB
    string[] nombresVMEnDB = CT_CampoViewModelFrmBusqueda.GroupBy(item =>
item.NombreDeVM).Select(group => group.Key).ToArray();

    // Remover aquellos VM que estén en DB, y que no estén en el listado actual
```

```
    string[] nombresVMARemoverDeDB = nombresVMEnDB.Where(p1 => !nombresVMEnRuntime.Any(p2 => p2 ==
p1)).ToArray();
    foreach (string nombreVMARemoverDeDB in nombresVMARemoverDeDB)
    {
        CampoViewModelFrmBusqueda[] camposViewModelFrmBusquedaARemover =
CT_CampoViewModelFrmBusqueda.CamposViewModelFrmBusqueda.Where(c => c.NombreDeVM ==
nombreVMARemoverDeDB).ToArray();

        .RemoveRange(camposViewModelFrmBusquedaARemover);
    }

    foreach (Type typeViewModelFrmBusqueda in typesViewModelFrmBusqueda)
    {
        // Obtengo nombre de campos del ViewModel en Runtime
        PropertyInfo[] propertyInfos =
typeViewModelFrmBusqueda.GetProperties(System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.Instance | System.Reflection.BindingFlags.DeclaredOnly);
        string[] nombresCampoEnRuntime = propertyInfos.Select(item => item.Name).ToArray();

        // Obtengo nombres de campos del ViewModel almacenados en DB
        string[] nombresCamposEnDB = CT_CampoViewModelFrmBusqueda.CamposViewModelFrmBusqueda.Where(c
=> c.NombreDeVM == typeViewModelFrmBusqueda.Name).Select(c => c.NombreDeCampo).ToArray();

        // Remover aquellos Campos que estén en DB, y que no estén en el listado actual
        string[] camposARemoverDeDB = nombresCamposEnDB.Where(p1 => !nombresCampoEnRuntime.Any(p2 =>
p2 == p1)).ToArray();
        foreach (string camposRemoverDeDB in camposARemoverDeDB)
        {
            CampoViewModelFrmBusqueda[] camposViewModelFrmBusquedaARemover =
CT_CampoViewModelFrmBusqueda.CamposViewModelFrmBusqueda.Where(c => c.NombreDeVM ==
typeViewModelFrmBusqueda.Name && c.NombreDeCampo == camposRemoverDeDB).ToArray();
            CT_CampoViewModelFrmBusqueda.RemoveRange(camposViewModelFrmBusquedaARemover);
        }

        // Agrego aquellos Campos que no estén en DB, y que estén en el listado actual
        string[] camposAAgregarADeDB = nombresCampoEnRuntime.Where(p1 => !nombresCamposEnDB.Any(p2
=> p2 == p1)).ToArray();
        int visibleIndex = 0;
        foreach (string campoAAgregarADeDB in camposAAgregarADeDB)
        {
            CampoViewModelFrmBusqueda campoViewModelFrmBusqueda = new CampoViewModelFrmBusqueda();
            campoViewModelFrmBusqueda.NombreDeVM = typeViewModelFrmBusqueda.Name;
            campoViewModelFrmBusqueda.NombreDeCampo = campoAAgregarADeDB;
            campoViewModelFrmBusqueda.Visible = true;
            campoViewModelFrmBusqueda.HeaderFontBold = false;
            campoViewModelFrmBusqueda.HeaderFontItalic = false;
            campoViewModelFrmBusqueda.HeaderFontUnderline = false;
            campoViewModelFrmBusqueda.CellFontBold = false;
            campoViewModelFrmBusqueda.CellFontItalic = false;
            campoViewModelFrmBusqueda.CellFontUnderline = false;

            CT_CampoViewModelFrmBusqueda.Add(campoViewModelFrmBusqueda);
        }
    }
}
```

Actualización a aplicación de configuración de columnas automática en Formulario de Búsqueda (base)

Para soportar las nuevas configuraciones gráficas de las columnas de la grilla, al algoritmo inicial se le añaden los valores por defecto para las propiedades de las cabeceras y celdas de la grilla, al momento de su mapeo.

```
private void AplicarConfiguracionAColumnas()
{
```

```
    // Obtendo información de las columnas desde DB
    CampoViewModelFrmBusqueda[]  camposViewModelFrmBusqueda = CT_CamposViewModelFrmBusqueda.Where(c
=> c.NombreDeVM == tipoDeDatoDelVM.Name).ToArray();

    // Detengo pintado en la grilla (mientras se actualiza su contenido)
    this.gcResultadosDeBusqueda.BeginUpdate();

    foreach (CampoViewModelFrmBusqueda campoViewModelFrmBusqueda in this.camposViewModelFrmBusqueda)
    {
        if (campoViewModelFrmBusqueda.Visible == true)
        {
            // Por defecto las columnas son visibles

            // Establezco el orden de las columnas
            if (campoViewModelFrmBusqueda.VisibleIndex != null)
            {

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].VisibleIndex =
(int)campoViewModelFrmBusqueda.VisibleIndex;
            }
        }
        else
        {
            this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].Visible =
false;
        }

        // Establezco color de fondo de las cabeceras
        if (campoViewModelFrmBusqueda.HeaderBackColor != null)
        {
            Color appearanceHeaderBackColor =
Color.FromArgb((int)campoViewModelFrmBusqueda.HeaderBackColor);

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceHeader.BackCo
lor = Color.FromArgb(appearanceHeaderBackColor.A, appearanceHeaderBackColor.R,
appearanceHeaderBackColor.G, appearanceHeaderBackColor.B);
        }

        // Establezco color de texto de las cabeceras
        if (campoViewModelFrmBusqueda.HeaderForeColor != null)
        {
            Color appearanceHeaderForeColor =
Color.FromArgb((int)campoViewModelFrmBusqueda.HeaderForeColor);

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceHeader.ForeCo
lor = Color.FromArgb(appearanceHeaderForeColor.A, appearanceHeaderForeColor.R,
appearanceHeaderForeColor.G, appearanceHeaderForeColor.B);
        }

        // Establezco configuraciones de fuente de las cabeceras
        if (campoViewModelFrmBusqueda.HeaderFontSize != null ||
            campoViewModelFrmBusqueda.HeaderFontBold != false ||
            campoViewModelFrmBusqueda.HeaderFontItalic != false ||
            campoViewModelFrmBusqueda.HeaderFontUnderline != false)
        {
            float fontSize;
            FontStyle fontStyle;

            // Obtengo fuente asignada por el sistema a la columna
            Font font =
this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceHeader.GetFon
t();

            if (campoViewModelFrmBusqueda.HeaderFontSize != null)
            {
                fontSize = (float)campoViewModelFrmBusqueda.HeaderFontSize;
            }
            else
            {
```

```
                fontSize = font.SizeInPoints;
            }

            fontStyle = FontStyle.Regular;
            if (campoViewModelFrmBusqueda.HeaderFontBold != false)
            {
                fontStyle |= FontStyle.Bold;
            }
            if (campoViewModelFrmBusqueda.HeaderFontItalic != false)
            {
                fontStyle |= FontStyle.Italic;
            }
            if (campoViewModelFrmBusqueda.HeaderFontUnderline != false)
            {
                fontStyle |= FontStyle.Underline;
            }

            // Actualizo fuente original con los parámetros preferidos por el usuario
            font = new System.Drawing.Font(font.FontFamily, fontSize,
((System.Drawing.FontStyle)fontStyle), System.Drawing.GraphicsUnit.Point, ((byte)(0)));

            // Asigno fuente actualizada a la columna

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceHeader.Font =
font;
        }

        // Establezco color de fondo de las celdas
        if (campoViewModelFrmBusqueda.CellBackColor != null)
        {
            Color appearanceCellBackColor =
Color.FromArgb((int)campoViewModelFrmBusqueda.CellBackColor);

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceCell.BackColo
r = Color.FromArgb(appearanceCellBackColor.A, appearanceCellBackColor.R, appearanceCellBackColor.G,
appearanceCellBackColor.B);
        }

        // Establezco color de fondo de las celdas (segundo color de fondo)
        if (campoViewModelFrmBusqueda.CellBackColor2 != null)
        {
            Color appearanceCellBackColor2 =
Color.FromArgb((int)campoViewModelFrmBusqueda.CellBackColor2);

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceCell.BackColo
r2 = Color.FromArgb(appearanceCellBackColor2.A, appearanceCellBackColor2.R,
appearanceCellBackColor2.G, appearanceCellBackColor2.B);
        }

        // Establezco color de texto de las celdas
        if (campoViewModelFrmBusqueda.CellForeColor != null)
        {
            Color cellHeaderForeColor =
Color.FromArgb((int)campoViewModelFrmBusqueda.CellForeColor);

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceCell.ForeColo
r = Color.FromArgb(cellHeaderForeColor.A, cellHeaderForeColor.R, cellHeaderForeColor.G,
cellHeaderForeColor.B);
        }

        // Establezco configuraciones de fuente de las celdas
        if (campoViewModelFrmBusqueda.CellFontSize != null ||
            campoViewModelFrmBusqueda.CellFontBold != false ||
            campoViewModelFrmBusqueda.CellFontItalic != false ||
            campoViewModelFrmBusqueda.CellFontUnderline != false)
        {
            float fontSize;
            FontStyle fontStyle;
```

```
            // Obtengo fuente asignada por el sistema a la columna
            Font font =
this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceCell.GetFont(
);

            if (campoViewModelFrmBusqueda.CellFontSize != null)
            {
                fontSize = (float)campoViewModelFrmBusqueda.CellFontSize;
            }
            else
            {
                fontSize = font.SizeInPoints;
            }

            fontStyle = FontStyle.Regular;
            if (campoViewModelFrmBusqueda.CellFontBold != false)
            {
                fontStyle |= FontStyle.Bold;
            }
            if (campoViewModelFrmBusqueda.CellFontItalic != false)
            {
                fontStyle |= FontStyle.Italic;
            }
            if (campoViewModelFrmBusqueda.CellFontUnderline != false)
            {
                fontStyle |= FontStyle.Underline;
            }

            // Actualizo fuente original con los parámetros preferidos por el usuario
            font = new System.Drawing.Font(font.FontFamily, fontSize,
((System.Drawing.FontStyle)fontStyle), System.Drawing.GraphicsUnit.Point, ((byte)(0)));

            // Asigno fuente actualizada a la columna

this.gvResultadosDeBusqueda.Columns[campoViewModelFrmBusqueda.NombreDeCampo].AppearanceCell.Font =
font;
        }
    }

    // Reanudo el pintado en la grilla (mientras se actualiza su contenido)
    this.gcResultadosDeBusqueda.EndUpdate();
}
```

Actualización a guardado de configuración de columnas en Formulario de Búsqueda (base)

Para soportar las nuevas configuraciones gráficas de las columnas de la grilla, al algoritmo inicial se le añaden las propiedades de las cabeceras y celdas de la grilla.

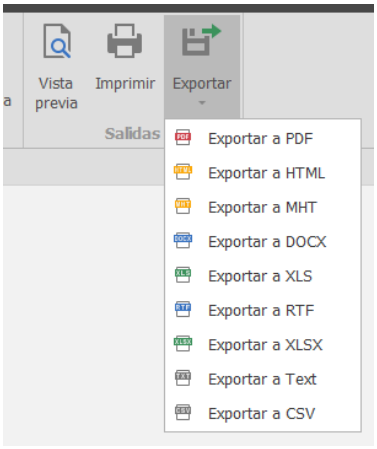
```
private bool GuardarConfiguraciónDeColumnas()
{
    bool blnRetorno = true;

    // Obtengo los GridColums actualmente utilizados en la grilla
    GridColumn[] gridColumns = this.gvResultadosDeBusqueda.Columns.ToArray();

    /* Para GridColumn, busco su CampoViewModelFrmBusqueda asociado y actualizo todas las
     * propiedades de CampoViewModelFrmBusqueda con la información de GridColumn. */
    foreach (GridColumn gridColumn in gridColumns)
    {
        CampoViewModelFrmBusqueda campoViewModelFrmBusqueda =
this.camposViewModelFrmBusqueda.First(c => c.NombreDeCampo == gridColumn.FieldName);

        campoViewModelFrmBusqueda.Visible = gridColumn.Visible;
        campoViewModelFrmBusqueda.VisibleIndex = gridColumn.VisibleIndex;

        if (gridColumn.AppearanceHeader.BackColor.IsEmpty == true)
```

```
        {
            campoViewModelFrmBusqueda.HeaderBackColor = null;
        }
        else
        {
            campoViewModelFrmBusqueda.HeaderBackColor =
gridColumn.AppearanceHeader.BackColor.ToArgb();
        }

        if (gridColumn.AppearanceHeader.ForeColor.IsEmpty == true)
        {
            campoViewModelFrmBusqueda.HeaderForeColor = null;
        }
        else
        {
            campoViewModelFrmBusqueda.HeaderForeColor =
gridColumn.AppearanceHeader.ForeColor.ToArgb();
        }

        campoViewModelFrmBusqueda.HeaderFontSize = gridColumn.AppearanceHeader.Font.Size;
        campoViewModelFrmBusqueda.HeaderFontBold = gridColumn.AppearanceHeader.Font.Bold;
        campoViewModelFrmBusqueda.HeaderFontItalic = gridColumn.AppearanceHeader.Font.Italic;
        campoViewModelFrmBusqueda.HeaderFontUnderline = gridColumn.AppearanceHeader.Font.Underline;

        if (gridColumn.AppearanceCell.BackColor.IsEmpty == true)
        {
            campoViewModelFrmBusqueda.CellBackColor = null;
        }
        else
        {
            campoViewModelFrmBusqueda.CellBackColor = gridColumn.AppearanceCell.BackColor.ToArgb();
        }

        if (gridColumn.AppearanceCell.BackColor2.IsEmpty == true)
        {
            campoViewModelFrmBusqueda.CellBackColor2 = null;
        }
        else
        {
            campoViewModelFrmBusqueda.CellBackColor2 =
gridColumn.AppearanceCell.BackColor2.ToArgb();
        }

        if (gridColumn.AppearanceCell.ForeColor.IsEmpty == true)
        {
            campoViewModelFrmBusqueda.CellForeColor = null;
        }
        else
        {
            campoViewModelFrmBusqueda.CellForeColor = gridColumn.AppearanceCell.ForeColor.ToArgb();
        }

        campoViewModelFrmBusqueda.CellFontSize = gridColumn.AppearanceCell.Font.Size;
        campoViewModelFrmBusqueda.CellFontBold = gridColumn.AppearanceCell.Font.Bold;
        campoViewModelFrmBusqueda.CellFontItalic = gridColumn.AppearanceCell.Font.Italic;
        campoViewModelFrmBusqueda.CellFontUnderline = gridColumn.AppearanceCell.Font.Underline;
    }

    // Persisto las actualizaciones
    try
    {
        CT_CamposViewModelFrmBusqueda.UpdateRange(this.camposViewModelFrmBusqueda);
    }
    catch (Exception)
    {
        blnRetorno = false;
    }

    return blnRetorno;
```

```
}
```

User Story ID 9 - Formulario de búsqueda: generación de salidas

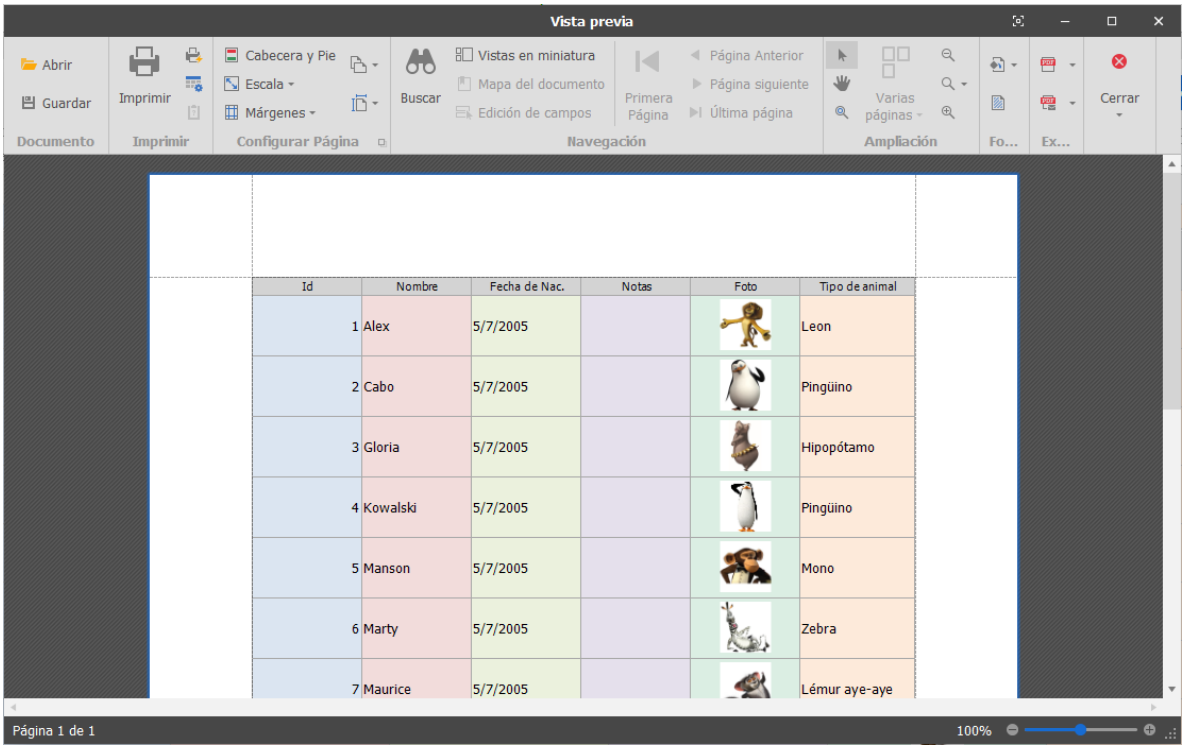
Acceso a generaciones de salidas para Formulario de Búsqueda (base)



Se agregaron las opciones de exportación al menú Ribbon del *Formulario de Búsqueda (base)*, de acuerdo a las especificaciones funcionales, y el comportamiento definido en wireframes.

Los formatos de exportación están sujetos a los formatos soportados por la herramienta de generación de reportes de *DevExpress*.

Vista previa de impresión



Id	Nombre	Fecha de Nac.	Notas	Foto	Tipo de animal
1	Alex	5/7/2005			Leon
2	Cabo	5/7/2005			Pingüino
3	Gloria	5/7/2005			Hipopótamo
4	Kowalski	5/7/2005			Pingüino
5	Manson	5/7/2005			Mono
6	Marty	5/7/2005			Zebra
7	Maurice	5/7/2005			Lémur aye-aye

El paquete de controles de usuario de DevExpress, cuenta con una serie de controles de usuario que facilitan la generación de vistas de impresión.

El control *PrintControl* [56]es utilizado para generar una vista previa de impresión, en el control *PrintPreview* [57] con menú ribbon.

Dado un GridControl, la propiedad *IsPrintingAvailable* [58] permite conocer si está o no disponible una versión imprimible de la grilla. De mismo modo, el método *ShowRibbonPrintPreview()* [59]sobre la grilla, permite generar el visor de vista previa de impresión, con menú Ribbon.

```
// Verifico si el GridControl puede ser previsualizado.
if (!gcResultadosDeBusqueda.IsPrintingAvailable)
{
    XtraMessageBox.SmartTextWrap = true;
    XtraMessageBox.Show(
        resxOwn.GetString(nameof(FrmBúsquedaBaseResx.LibreríaNoEncontrada)),
        this.Text,
        MessageBoxButtons.OK,
        MessageBoxIcon.Error,
        DevExpress.Utils.DefaultBoolean.True
        );

    return;
}

// Abro la ventana de vista previa
this.gcResultadosDeBusqueda.ShowRibbonPrintPreview();
```

Impresión

Del mismo modo que con la generación de vistas previas, para generar una impresión primero es necesario conocer si está disponible o no una versión imprimible de la gilla, con la propiedad *IsPrintingAvailable*. Luego, es posible efectuar la impresión con el método *Print()*[60], en la grilla.

```
// Verifico si el GridControl puede ser previsualizado.
if (!gcResultadosDeBusqueda.IsPrintingAvailable)
{
    XtraMessageBox.SmartTextWrap = true;
    XtraMessageBox.Show(
        resxOwn.GetString(nameof(FrmBúsquedaBaseResx.LibreríaNoEncontrada)),
        this.Text,
        MessageBoxButtons.OK,
        MessageBoxIcon.Error,
        DevExpress.Utils.DefaultBoolean.True
        );

    return;
```

[56] Documentación oficial de PrintControl, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraPrinting.Control.PrintControl
[57] Documentación oficial de PrintPreview, en DevExpress:
https://docs.devexpress.com/XtraReports/5186/winforms-reporting/print-preview/gui/print-preview-with-a-ribbon-toolbar
[58] Documentación oficial de IsPrintingAvailable, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.GridControl.IsPrintingAvailable
[59] Documentación oficial de ShowRibbonPrintPreview, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.GridControl.ShowRibbonPrintPreview
[60] Documentación oficial de PrintControl, en DevExpress:
https://docs.devexpress.com/WindowsForms/DevExpress.XtraGrid.GridControl.Print

```
}

// Envío documento al spool de impresion
gcResultadosDeBusqueda.Print();
```

Exportar a diferentes formatos

Del mismo modo que la grilla soporta los métodos Print() y PrintDialog() utilizados previamente, soporta la generación de salidas en doce diferentes formatos, de los cuales sólo se utilizarán nueve.

```
private void ExportarA(string extensión, string descripciónDeExtensión)
{
    string nombreDeArchivo = this.GetFileName(string.Format("*.{0}", extensión),
descripciónDeExtensión);

    if (!String.IsNullOrEmpty(nombreDeArchivo))
    {
        try
        {
            this.ExportarAFormatosBienConocidos(nombreDeArchivo, extensión);
            this.OpenExportedFile(nombreDeArchivo);
        }
        catch (Exception e)
        {
            this.ShowExportErrorMessage(e);
        }
    }
}
```

```
private void ExportarAFormatosBienConocidos(String nombreDeArchivo, string extensión)
{
    if (this.gcResultadosDeBusqueda == null)
        return;

    // Establezco cursores de espera
    Cursor currentCursor = Cursor.Current;
    Cursor.Current = Cursors.WaitCursor;

    if (extensión == "rtf") this.gcResultadosDeBusqueda.ExportToRtf(nombreDeArchivo);
    if (extensión == "csv") this.gcResultadosDeBusqueda.ExportToCsv(nombreDeArchivo);
    if (extensión == "docx") this.gcResultadosDeBusqueda.ExportToDocx(nombreDeArchivo);
    if (extensión == "pdf") this.gcResultadosDeBusqueda.ExportToPdf(nombreDeArchivo);
    if (extensión == "mht") this.gcResultadosDeBusqueda.ExportToMht(nombreDeArchivo);
    if (extensión == "html") this.gcResultadosDeBusqueda.ExportToHtml(nombreDeArchivo);
    if (extensión == "txt") this.gcResultadosDeBusqueda.ExportToText(nombreDeArchivo);
    if (extensión == "xls") ExportToXlsInternal(nombreDeArchivo);
    if (extensión == "xlsx") ExportToXlsxInternal(nombreDeArchivo);

    // Aplico cursor normal
    Cursor.Current = currentCursor;
}
```

User Story ID 10 - Formulario ABMC (base)

Formulario ABMC (base)

Estados internos

En el *Formulario de Búsqueda (base),* los posibles estados internos del mismo se representaron mediante enumerados. En el Formulario ABMC (base) se empleará la misma estrategia.

Tres estados describen los posibles usos del formulario, donde cada estado responde a un verbo: Consultar, Crear o Modificar.

Siendo que se utilizará un mismo formulario para representar cada una de estas operaciones, la lógica interna se adecuará dependiendo del verbo con el cual se inicialice.

```
#region Enumerados

/// <summary>
/// Posibles estados del formulario.
/// </summary>
public enum Verbo
{
    Consultar,
    Crear,
    Modificar
};

#endregion // Enumerados
```

Adecuación del formulario según el verbo

El formulario tiene un método público con el cual se le indica a la instancia, cuál es el verbo que será utilizado, y con qué entidad de datos (en caso de altas o modificaciones).

Luego del instanciamiento, el formulario permanecerá oculto (en estado no visible), y no será visible hasta tanto no se ejecute el método *PrepararFormulario(…)*, en el cual se adecuará el formulario al verbo correspondiente.

Las secuencias que se seguirán para adecuar el formulario, tienen comportamientos encapsulados en los siguientes métodos, que serán introducidos de forma breve para comprender de modo general, el resultado de cada ejecución.

Método	Propósito
CargarEntidadEnPropiedad(entidad)	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Se utiliza en consultas y modificaciones. Recibe como parámetro la entidad de datos principal que será utilizada en el formulario (por ejemplo, para un *Formulario ABMC* para *Mascota*, la entidad de datos principal sería *Mascota*).

	Dado que la entidad se recibe con el tipo de datos *object*[61], es necesario castearla al tipo de datos correspondiente (el cual es conocido por el formulario apropiado).
CargarEntidadEnControles()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Se utiliza en consultas y modificaciones. Carga los atributos de la entidad de datos principal en los controles de usuario del formulario (por ejemplo, se cargaría el nombre de la mascota en una caja de texto, se seleccionaría su sexo en una lista desplegable, etc.).
EstablecerControlesEnSoloLectura()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Se utiliza sólo en consultas. Establece en solo lectura a todos aquellos controles que reflejen información de la entidad que está siendo exhibida mediante el formulario. Esto limita todo tipo de interacción con los controles que permitan la carga o modificación de datos
PrepararFormularioParaConsulta()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Se utiliza sólo en consultas. Sirve para refinar el formulario en una situación particular de consulta. Hay circunstancias en las cuales se utilizan controles diferentes para un mismo atributo, en función del verbo que se realiza (alta, modificación o consulta). Por ejemplo, la foto de una mascota se da de alta y/o modifica con un control de usuario, pero se muestra con otro. Ambos controles están dentro del formulario, pero nunca ambos están visibles al mismo tiempo.
CambiarEstado(Verbo)	Cambia el valor del estado del formulario a un verbo en particular (Consultar, Crear, Modificar). El cambio de estado puede implicar el ocultamiento o muestreo de controles, el estado de lectura o solo lectura de los controles, el cambio de del texto del título del formulario, invocar algún método que sea pertinente al nuevo estado, etc.
CargarNuevaEntidadEnPropiedad()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Se utiliza sólo en altas. Es el método análogo a CargarEntidadEnControles(), pero para situaciones de alta.

[61] Documentación oficial de object: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/reference-types

	Crea una nueva entidad de tipo de dato acorde, y la almacena en una propiedad local para posteriormente cargar en ella todos los datos cargados en los controles de usuario del formulario, y finalmente persistirla.
PrepararFormularioParaNuevo()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Se utiliza sólo en altas. Sirve para refinar el formulario en una situación particular de altas. Hay circunstancias en las cuales se utilizan controles diferentes para un mismo atributo, en función del verbo que se realiza (alta, modificación o consulta). Por ejemplo, la foto de una mascota se da de alta y/o modifica con un control de usuario, pero se muestra con otro. Ambos controles están dentro del formulario, pero nunca ambos están visibles al mismo tiempo.
PrepararFormularioParaEdición()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Se utiliza sólo en modificaciones. Sirve para refinar el formulario en una situación particular de modificaciones. Hay circunstancias en las cuales se utilizan controles diferentes para un mismo atributo, en función del verbo que se realiza (alta, modificación o consulta). Por ejemplo, la foto de una mascota se da de alta y/o modifica con un control de usuario, pero se muestra con otro. Ambos controles están dentro del formulario, pero nunca ambos están visibles al mismo tiempo.

```
public void PrepararFormulario(Verbo estadoABMC, object entidad = null)
{
    switch (estadoABMC)
    {
        case Verbo.Consultar:
            this.CargarEntidadEnPropiedad(entidad);
            this.CargarEntidadEnControles();
            this.EstablecerControlesEnSoloLectura();
            this.PrepararFormularioParaConsulta();
            this.CambiarEstado(Verbo.Consultar);
            break;

        case Verbo.Crear:
            this.CargarNuevaEntidadEnPropiedad();
            this.PrepararFormularioParaNuevo();
            this.CambiarEstado(Verbo.Crear);
            break;

        case Verbo.Modificar:
            this.CargarEntidadEnPropiedad(entidad);
            this.CargarEntidadEnControles();
            this.PrepararFormularioParaEdición();
            this.CambiarEstado(Verbo.Modificar);
            break;
```

```
        default:
            // TODO: Manejar error.
            break;
    }

    this.ConfigurarTituloDeFormulario();
}
```

Cambio de estados

Como se explicó previamente, en los métodos empleados en la *Adecuación del formulario según el verbo*, el método *CambiarEstado(…)*, cambia el valor del estado del formulario a un verbo en particular (Consultar, Crear, Modificar).

El cambio de estado puede implicar el ocultamiento o muestreo de controles, el estado de lectura o solo lectura de los controles, el cambio de del texto del título del formulario, invocar algún método que sea pertinente al nuevo estado, etc.

En el siguiente caso, se ocultan o muestran los botones para operaciones de alta, baja, o modificación, dependiendo de la operación que se esté realizando:

```
private void CambiarEstado(Verbo nuevoEstado)
{
    this.verboActualDelFormulario = nuevoEstado;

    switch (nuevoEstado)
    {
        case Verbo.Consultar:
            this.bbiAbrirNuevo.Visibility = DevExpress.XtraBars.BarItemVisibility.Always;
            this.bbiAbrirModificar.Visibility = DevExpress.XtraBars.BarItemVisibility.Always;
            this.bbiGuardarNuevo.Visibility = DevExpress.XtraBars.BarItemVisibility.Never;
            this.bbiGuardarCambios.Visibility = DevExpress.XtraBars.BarItemVisibility.Never;
            this.bbiEliminar.Visibility = DevExpress.XtraBars.BarItemVisibility.Always;

            break;

        case Verbo.Crear:
            this.bbiAbrirNuevo.Visibility = DevExpress.XtraBars.BarItemVisibility.Never;
            this.bbiAbrirModificar.Visibility = DevExpress.XtraBars.BarItemVisibility.Never;
            this.bbiGuardarNuevo.Visibility = DevExpress.XtraBars.BarItemVisibility.Always;
            this.bbiGuardarCambios.Visibility = DevExpress.XtraBars.BarItemVisibility.Never;
            this.bbiEliminar.Visibility = DevExpress.XtraBars.BarItemVisibility.Never;

            break;

        case Verbo.Modificar:
            this.bbiAbrirNuevo.Visibility = DevExpress.XtraBars.BarItemVisibility.Always;
            this.bbiAbrirModificar.Visibility = DevExpress.XtraBars.BarItemVisibility.Never;
            this.bbiGuardarNuevo.Visibility = DevExpress.XtraBars.BarItemVisibility.Never;
            this.bbiGuardarCambios.Visibility = DevExpress.XtraBars.BarItemVisibility.Always;
            this.bbiEliminar.Visibility = DevExpress.XtraBars.BarItemVisibility.Always;

            break;

        default:
            break;
    }
}
```

Lógica de botón Crear y Modificar

Al presionar botón Crear o Modificar se ponen en marcha una serie de consultas que determinan el curso de la operación. La lógica de base (alojada en el formulario base) requiere de información del formulario especializado para la entidad que se está tratando.

Hay cuatro métodos virtuales que podrían ser especializados, que responden a las precondiciones necesarias para proceder con las operaciones de persistencia (alta o modificación, según corresponda):

Método	Propósito
EstanTodosLosDatosRequeridos()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Verifica si están todos los datos requeridos para proceder con un alta o modificación. En caso de encontrar faltantes, muestra un mensaje de error en el control de usuario que corresponda.
EstanTodosLosDatosValidos()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Verifica si están todos los datos válidos (dentro de los parámetros establecidos) para proceder con un alta o modificación. En caso de encontrar un caso de incumplimiento, muestra un mensaje de error en el control de usuario que corresponda.
HayColision()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Verifica si hay colisión con otros datos previamente persistidos.
HayCambios()	Este método es virtual, y debe ser implementado en el formulario de búsqueda que especialice al formulario de búsqueda base. Verifica si se realizaron modificaciones sobre los datos originales de la entidad de datos. En el caso de altas, se asume como modificación a cualquier campo que tenga un valor distinto al estado inicial del formulario. En el caso de modificaciones, a cualquier campo que tenga un valor distinto a la entidad en edición.

La lógica empleada al accionar el botón Guardar o Modificar, en cualquiera de los casos sigue un proceso ordenado, en el cual si supera tres instancias: (1) que estén todos los datos requeridos, (2) que estén todos los datos válidos, y (3) que no haya colisión, entonces guarda o actualiza la entidad, y consulta al usuario si desea cerrar el formulario.

```
private void LogicaBotonCrearYModificarEntidad()
{
    if (this.EstanTodosLosDatosRequeridos() && this.EstanTodosLosDatosValidos() &&
!this.HayColision())
    {
        if (this.HayCambios() == true)
        {
            switch (this.verboActualDelFormulario)
            {
                case Verbo.Crear:
```

```
                    if (this.PersistenciaCrearEntidad() == false)
                    {
                        XtraMessageBox.SmartTextWrap = true;
                        XtraMessageBox.Show(
                            resxDialog.GetString(nameof(Dialog.ErrorAlIntentarGuardar)),
                            this.Text,
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Error,
                            DevExpress.Utils.DefaultBoolean.True
                            );
                        return;
                    }
                    break;

                case Verbo.Modificar:
                    if (this.PersistenciaModificarEntidad() == false)
                    {
                        XtraMessageBox.SmartTextWrap = true;
                        XtraMessageBox.Show(
                            resxDialog.GetString(nameof(Dialog.ErrorAlIntentarModificar)),
                            this.Text,
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Error,
                            DevExpress.Utils.DefaultBoolean.True
                            );
                        return;
                    }
                    break;
            }

            // Hago el cambio de estado desde Nuevo a Edición
            if (this.verboActualDelFormulario == Verbo.Crear)
            {
                this.PrepararFormularioParaEdición();
                this.CambiarEstado(Verbo.Modificar);
                this.ConfigurarTituloDeFormulario();
            }
            else
            {
                /* Puede que al guardar los cambios, estando en verbo Modificar, cambien
                * los elementos que inciden en la generación del título, por esto se lo
                * configura nuevamente tras cada "Guardar cambios". */
                this.ConfigurarTituloDeFormulario();
            }
        }
    }
    else
    {
        XtraMessageBox.SmartTextWrap = true;
        XtraMessageBox.Show(
            resxDialog.GetString(nameof(Dialog.FaltaRequeridoOValidoOHayColision)),
            this.Text,
            MessageBoxButtons.OK,
            MessageBoxIcon.Exclamation,
            DevExpress.Utils.DefaultBoolean.True
            );
    }
}
```

Lógica de botón Eliminar

La lógica de eliminación se ejecuta solo en situaciones de modificación o consulta. Previamente se consulta al usuario si desea continuar con la operación, y luego de ejecutar la operación de persistencia, se cierra el formulario.

```
private void LogicaBotonEliminar()
{
    bool puedeCerrarFormulario = false;
```

```
    if (this.verboActualDelFormulario == Verbo.Modificar || this.verboActualDelFormulario ==
Verbo.Consultar)
    {
        XtraMessageBox.SmartTextWrap = true;
        DialogResult dialogResult = XtraMessageBox.Show(
            resxDialog.GetString(nameof(Dialog.PreguntaDeseaEliminar)),
            this.Text,
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question,
            DevExpress.Utils.DefaultBoolean.True
            );

        if (dialogResult == DialogResult.Yes)
        {
            if (this.PersistenciaEliminarEntidad() == false)
            {
                XtraMessageBox.SmartTextWrap = true;
                XtraMessageBox.Show(
                    resxDialog.GetString(nameof(Dialog.ErrorAlIntentarEliminar)),
                    this.Text,
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error,
                    DevExpress.Utils.DefaultBoolean.True
                    );
            }
            else
            {
                puedeCerrarFormulario = true;
            }
        }
    }

    if (puedeCerrarFormulario == true)
    {
        this.CerrarFormulario();
    }
}
```

Lógica de botón Cerrar

La lógica del botón Cerrar se ejecuta tanto al presionar el botón Cerrar del menú ribbon, al presionar la X de la barra de título del formulario, o de cualquier botón que invoque el método.

En primera instancia verifica que no haya cambios sin guardar en el formulario, de haberlos, pide confirmación al usuario para proceder con el cierre. Si no hay cambios, consulta al usuario si desea proceder con el cierre.

Para la detección de cambios se utiliza el método *HayCambios()*, tal como en la *Lógica de botón Crear y Modificar*.

```
public void LogicaBotonCerrar()
{
    bool puedeCerrarFormulario = true;

    // Si el estado del formulario es Nuevo o Edición, busco si hubo cambios, y pregunto como
proceder
    if (VerboActualDelFormulario == Verbo.Crear || VerboActualDelFormulario == Verbo.Modificar)
    {
        if (this.HayCambios())
        {
            XtraMessageBox.SmartTextWrap = true;
            DialogResult dialogResult = XtraMessageBox.Show(
                resxDialog.GetString(nameof(Dialog.PreguntaDeseaDesartarCambios)),
                this.Text,
                MessageBoxButtons.YesNo,
```

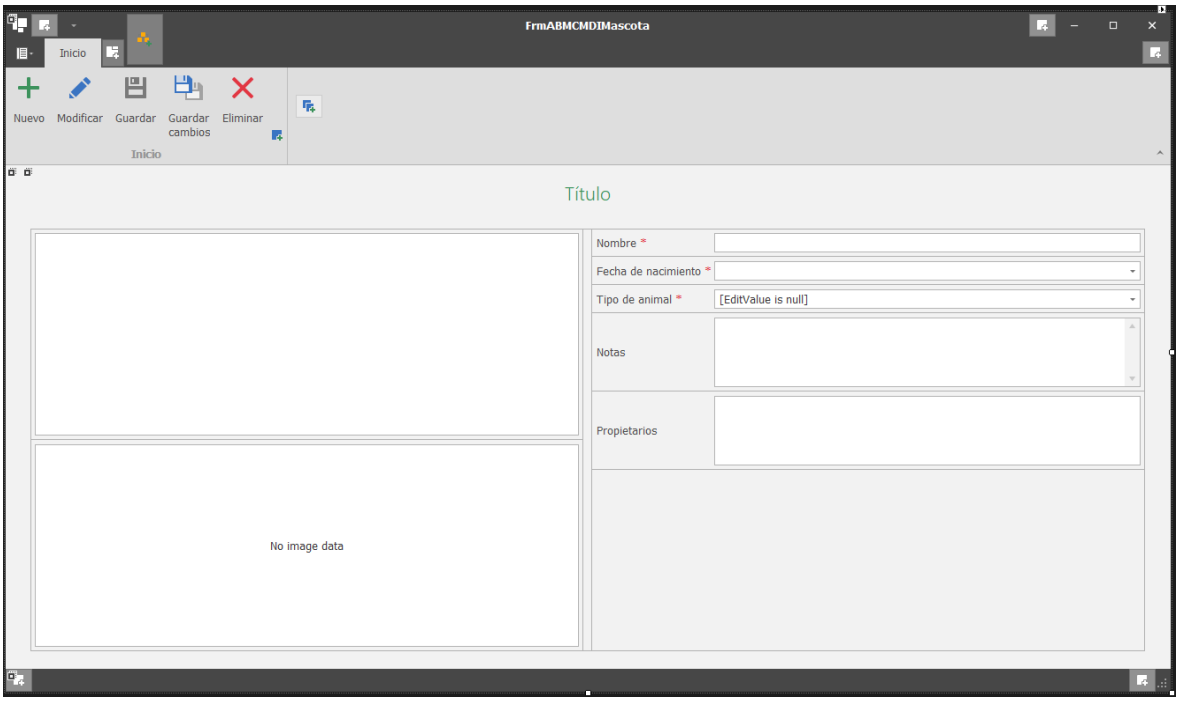
```
                MessageBoxIcon.Question,
                DevExpress.Utils.DefaultBoolean.True
                );

            if (dialogResult == DialogResult.No)
            {
                puedeCerrarFormulario = false;
            }
        }
    }

    if (puedeCerrarFormulario == true)
    {
        this.CerrarFormulario();
    }
}
```

Ejemplo de especialización de Formulario ABMC (base)

Para clarificar el proceso de especialización de un *Formulario ABMC (base)*, y a modo de ejemplo, se mostrarán en orden todos los componentes que integran la especialización, utilizando la entidad de datos de ejemplo *Mascota*.

Layout para Formulario ABMC de Mascota

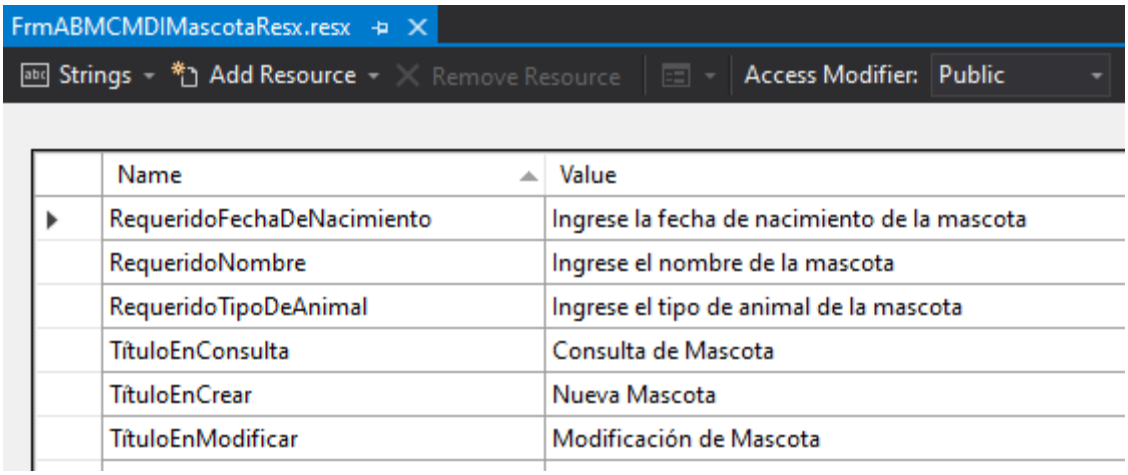
Luego de especializar el formulario, debe confeccionarse el layout acorde a las necesidades de la entidad de datos que se representa, para luego proceder con la codificación de la lógica subyacente.



Definición de archivo de recursos

Se deben definir claves para el título del formulario, en cada una de las posibles situaciones (alta, modificación y consulta), y tantas claves sean necesarias para los procesos de verificación (datos válidos, datos requeridos, colisiones, etc).

Por convención su nombre inicia con el nombre de la entidad que lo consumirá, y finaliza con '*Resx*', resultando '*FrmABMCMascotaResx*, y se almacena en *ArchivosDeRecurso* > *FrmABMC*. El archivo de recursos para el *Formlario ABMC* de la entidad de datos *Mascota* resulta:



Lógica especializada

Posterior a la definición del layout, debe definirse la lógica subyacente, en la cual se sobrescriben los métodos de la clase padre.

```
protected override void EstablecerControlesEnSoloLectura()
{
    this.teNombre.ReadOnly = true;
    this.deFechaDeNacimiento.ReadOnly = true;
    this.slueTipoDeAnimal.ReadOnly = true;
    this.meNotas.ReadOnly = true;
    this.tokenEditPropietario.ReadOnly = true;
}
```

```
protected override void PrepararFormularioParaConsulta()
{
    // En consulta el editor de imágenes se oculta
    this.lciPictureEdit.Visibility = DevExpress.XtraLayout.Utils.LayoutVisibility.Never;
}
```

```
protected override void PrepararFormularioParaNuevo()
{
    // En nuevo y edición el visor de imágenes se oculta
    this.lciImagenSlider.Visibility = DevExpress.XtraLayout.Utils.LayoutVisibility.Never;
}
```

```
protected override bool EstanTodosLosDatosRequeridos()
{
    bool blnRetorno = true;
    VMÚnico vm = this.ObtenerViewModel();

    // Nombre
    if (vm.Nombre == null)
    {
        this.dxErrorProvider.SetError(this.teNombre,
resxOwn.GetString(nameof(FrmABMCMDIMascotaResx.RequeridoNombre)));
        blnRetorno = false;
```

```
    }
    else
    {
        this.dxErrorProvider.SetError(this.teNombre, "");
    }

    // FechaDeNacimiento
    if (vm.FechaDeNacimiento == null)
    {
        this.dxErrorProvider.SetError(this.deFechaDeNacimiento,
resxOwn.GetString(nameof(FrmABMCMDIMascotaResx.RequeridoFechaDeNacimiento)));
        blnRetorno = false;
    }
    else
    {
        this.dxErrorProvider.SetError(this.deFechaDeNacimiento, "");
    }

    // TipoDeAnimalId
    if (vm.TipoDeAnimalId == null)
    {
        this.dxErrorProvider.SetError(this.slueTipoDeAnimal,
resxOwn.GetString(nameof(FrmABMCMDIMascotaResx.RequeridoTipoDeAnimal)));
        blnRetorno = false;
    }
    else
    {
        this.dxErrorProvider.SetError(this.slueTipoDeAnimal, "");
    }

    return blnRetorno;
}
```

```
protected override bool HayCambios()
{
    bool blnRetorno = false;
    VMÚnico vm = this.ObtenerViewModel();

    switch (VerboActualDelFormulario)
    {
        case Verbo.Crear:
            if (vm.Nombre != null ||
                vm.FechaDeNacimiento != null ||
                vm.TipoDeAnimalId != null ||
                vm.Notas != null ||
                vm.Foto != null ||
                vm.PropietariosId.Count() > 0
                )
            {
                blnRetorno = true;
            }

            break;

        case Verbo.Modificar:
            // Busco cambios en Propietarios
            IEnumerable<int> propietariosIdAnterior = this.entidadPrincipal.Propietarios.Select(p =>
p.Id);
            IEnumerable<int> propietariosIdActual = this.tokenEditPropietario.PropietariosId;
            bool cambiosEnPropietarios = !(new
HashSet<int>(propietariosIdAnterior)).SetEquals(propietariosIdActual);

            if (this.entidadPrincipal.Nombre != vm.Nombre ||
                this.entidadPrincipal.FechaDeNacimiento != vm.FechaDeNacimiento ||
                this.entidadPrincipal.TipoDeAnimalId != vm.TipoDeAnimalId ||
                this.entidadPrincipal.Notas != vm.Notas ||
                this.HayCambiosEnImágenes() ||
```

```
                cambiosEnPropietarios
                )
            {
                blnRetorno = true;
            }
            break;
    }

    return blnRetorno;
}
```

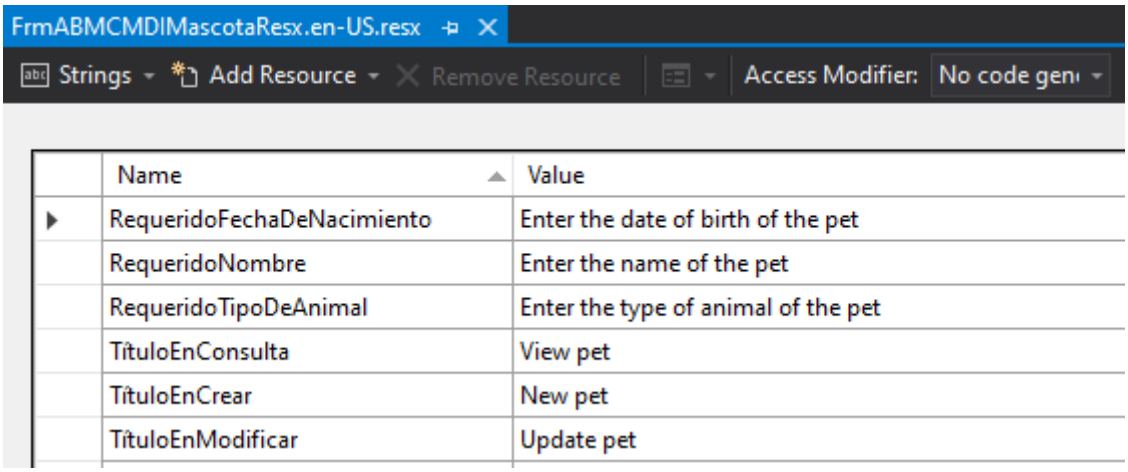
```
protected override bool PersistenciaCrearEntidad()
{
    bool retorno = true;

    this.TransferirDatosDeViewModelAEntidad();

    CT_Mascota.Insertar(this.entidadPrincipal);

    return retorno;
}
```

```
protected override bool PersistenciaModificarEntidad()
{
    bool retorno = true;

    this.TransferirDatosDeViewModelAEntidad();

    CT_Mascota.Actualizar(this.entidadPrincipal);

    return retorno;
}
```

```
protected override string ConfigurarTituloDeFormulario()
{
    return string.Format("{0} \"{1}\"",
resxOwn.GetString(nameof(FrmABMCMDIMascotaResx.TítuloEnConsulta)), this.entidadPrincipal.Nombre);
            break;
}
```

Traducción de archivo de recursos

La traducción del archivo de recursos implica replicar la estructura del archivo original, agregando al final del nombre del archivo, la clave de localización del idioma correspondiente (en este caso, ‘es-US’.



Ejecución

Con la definición de un archivo de recursos, y añadiendo unas cien líneas de código fáciles de comprender y replicar en el formulario especializado, se cuenta con un *Formulario ABMC* que goza de todas las funcionalidades especificadas en la base, y que es sencillamente traducible a cualquier lenguaje.

User Story ID 11 - Formulario de búsqueda (base): soporte para múltiples plantillas

Entidades de dato para soporte de Plantillas de configuración en la grilla de búsqueda

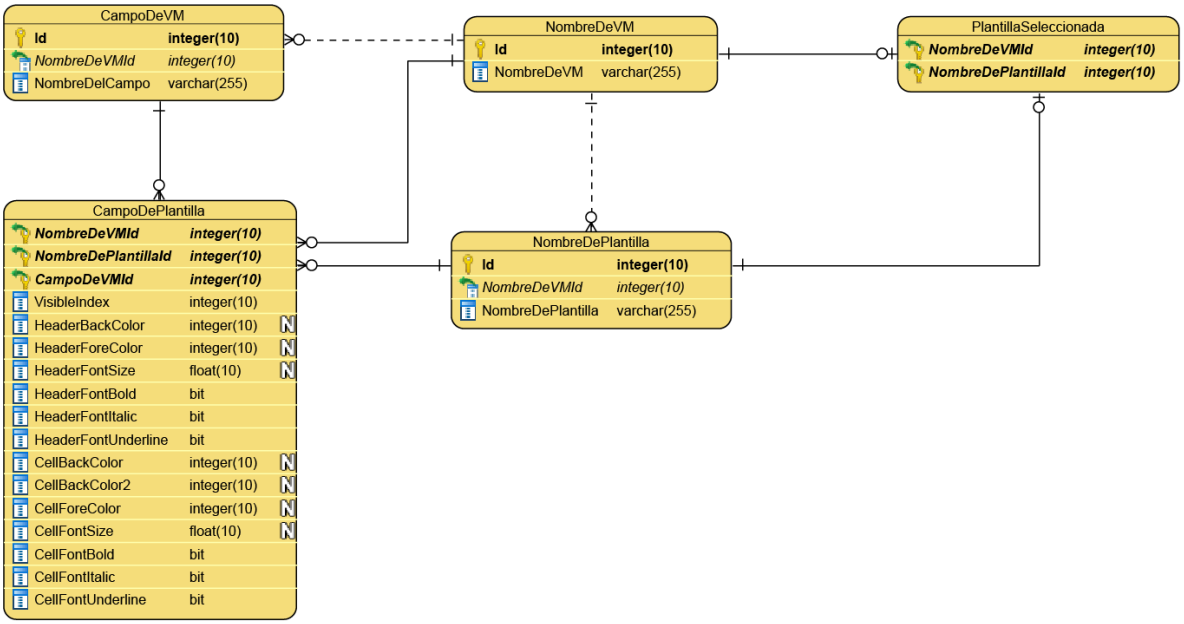
Hasta este punto, se contaba con un esquema que permitía reflejar el estado de una grilla de un *Formulario de Búsqueda* en una entidad de datos, y viceversa, comenzando por el mapeo de las propiedades de su *ViewModel* asociado, y finalizando con el añadido de atributos gráficos de las columnas de la grilla; es decir, había una relación unívoca entre una grilla, y una configuración para ella.

Se redefinió el modelo de datos que soporta la persistencia de configuraciones de la grilla, para que una misma grilla pueda tener muchas configuraciones diferentes, con un nombre que las identifique. A cada grupo de configuraciones, las nombraremos *Plantillas de configuración*.

En el modelo que soportaba sólo una configuración por grilla, cada entidad de datos se identificaba por el nombre de la clase *ViewModel* ("*NombreDeVM*"), y el nombre del campo al que se hacía referencia ("*NombreDeCampo*"), las cuales, en conjunto, conformaban la clave primaria de la entidad de datos.

El nuevo modelo de datos define una plantilla mediante cuatro entidades relacionadas, *NombreDeVM* y *CampoDeVM* (que anteriormente, en conjunto, eran la clave primaria de la entidad de datos que persistía la información de la grilla), y *NombreDePlantilla* y *CampoDePlantilla* (que representan a una plantilla, con las respectivas configuraciones para cada columna).

Una quinta entidades datos, que relaciona el *NombreDeVM* con *NombreDePlantilla*, permite establecer cuál es la plantilla activa para un determinado formulario de búsqueda.



- La entidad *NombreDeVM* no tiene dependencias con ninguna otra entidad, solamente representa a un *ViewModel* de *Formulario de Búsqueda*.
- La entidad *CampoDeVM*, representa a un campo de *ViewModel* de Formulario de Búsqueda, por lo cual tiene un vínculo fuerte con *NombreDeVM*.
- La entidad *NombreDePlantilla*, representa a una plantilla específica, asociada a un *NombreDeVM* particular. Para un mismo *ViewModel* de *Formulario de Búsqueda*, podrán existir múltiples plantillas.
- La entidad *CampoDePlantilla*, representa las configuraciones gráficas de un *CampoDeVM*, que pertenece a un *NombreDeVM*, para un *NombreDePlantilla* particular, por lo cual, tiene relaciones fuertes con dichas tres entidades.

- La entidad PlantillaSelccionada, representa la plantilla que está seleccionada para un determinado Formulario de Búsqueda, y tiene una relación fuerte con *NombreDeVM* y *NombreDePlantilla*.

Actualización de algoritmo de mapeo automático de campos de ViewModel, a base de datos

Para soportar el nuevo modelo de datos que permite el uso de múltiples plantillas de configuración, la última versión del algoritmo ha sufrido importantes actualizaciones, por lo cual se lo presentará por secciones de lógica relacionada.

Estrategia de persistencia

En las versiones anteriores del algoritmo, sólo era necesario verificar la consistencia a través de una única entidad de datos, mediante la cual se persistía el estado de una grilla, y su relación con su correspondiente *ViewModel* y sus campos.

En la versión actual, se emplean cuatro entidades de datos, por lo cual las tareas de verificación de consistencia se complejizan.

Para mantener el orden en la secuencia del algoritmo, y facilitar su legibilidad, se crearon inicialmente listas de entidades, que están directamente relacionadas con la aplicación que tendrán en la capa de persistencia.

Las listas pertenecen a uno de los siguientes tres grupos:

- Listas para almacenar entidades de mapeo: son listas que contienen las entidades de mapeo tal y como se encuentran en la base de datos, al momento de ejecutar el algoritmo.
- Listas para almacenar entidades de mapeo que serán insertadas en la base de datos: son las listas que contienen las entidades mapeo que serán persistidas al momento de finalizar el algoritmo.
- Listas para almacenar entidades de mapeo que serán borradas: son las listas que contendrán aquellas entidades, que, en el proceso de verificación de consistencia, por alguna razón, sean merecedoras de ser eliminadas.

```
// Listas para almacenar entidades de mapeo
List<FrmBúsqueda_NombreDeVM> frmBúsqueda_NombresDeVM_EnDB;
List<FrmBúsqueda_CampoDeVM> frmBúsqueda_CamposDeVM_EnDB;
List<FrmBúsqueda_NombreDePlantilla> frmBúsqueda_NombresDePlantilla_EnDB;
List<FrmBúsqueda_CampoDePlantilla> frmBúsqueda_CamposDePlantilla_EnDB;
List<FrmBúsqueda_PlantillaSeleccionada> frmBúsqueda_PlantillasSeleccionada_EnDB;

// Listas para almacenar entidades de mapeo que serán insertadas en DB
List<FrmBúsqueda_NombreDeVM> frmBúsqueda_NombresDeVM_APersistir = new
List<FrmBúsqueda_NombreDeVM>();
List<FrmBúsqueda_CampoDeVM> frmBúsqueda_CamposDeVM_APersistir = new List<FrmBúsqueda_CampoDeVM>();

// Listas para almacenar entidades de mapeo que serán borradas
List<FrmBúsqueda_NombreDeVM> frmBúsqueda_NombresDeVM_ABorrar = new List<FrmBúsqueda_NombreDeVM>();
List<FrmBúsqueda_CampoDeVM> frmBúsqueda_CamposDeVM_ABorrar = new List<FrmBúsqueda_CampoDeVM>();
List<FrmBúsqueda_NombreDePlantilla> frmBúsqueda_NombresDePlantilla_ABorrar = new
List<FrmBúsqueda_NombreDePlantilla>();
List<FrmBúsqueda_CampoDePlantilla> frmBúsqueda_CamposDePlantilla_ABorrar = new
List<FrmBúsqueda_CampoDePlantilla>();
List<FrmBúsqueda_PlantillaSeleccionada>     frmBúsqueda_PlantillasSeleccionada_ABorrar     =     new
List<FrmBúsqueda_PlantillaSeleccionada>();
```

Clasificación de ViewModels según su circunstancia actual

En primera instancia, se clasifican los *ViewModels* según su situación, por la cual se le dará un tratamiento diferente a cada grupo. Existen tres posibles situaciones: nuevos *ViewModels* que deban ser persistidos, *ViewModels* que ya no estén en uso y deban ser eliminados, y *ViewModels* que sigan existiendo en tiempo de ejecución, y ya estén persistidos.

```
// Obtengo VM en tiempo de ejecución (los que el desarrollador declaró)
string[] nombresVMEnRuntime = typesViewModelFrmBusqueda.Select(item => item.Name).ToArray();

// Obtengo VM almacenados en DB
string[] nombresVMEnDB = frmBúsqueda_NombresDeVM_EnDB.Select(item => item.NombreDeVM).ToArray();

// Obtengo intersecciones de VM en Runtime y DB
string[] nombresVMNuevos = nombresVMEnRuntime.Where(p1 => nombresVMEnDB.All(p2 => p2 !=
p1)).ToArray();
string[] nombresVMARemoverDeDB = nombresVMEnDB.Where(p1 => !nombresVMEnRuntime.Any(p2 => p2 ==
p1)).ToArray();
string[] nombresVMYaConsolidados = nombresVMEnRuntime.Intersect(nombresVMEnDB).ToArray();
```

Inserción de nuevos ViewModels en la base de datos

Se trata de la circunstancia más similar a la del algoritmo anterior. Consiste en mapear un nombre de *ViewModel*, con todos sus campos. En este caso no hace falta corroborar consistencia de plantillas, dado que no existe alguna.

```
// Obtengo VM en tiempo de ejecución (los que el desarrollador declaró)
string[] nombresVMEnRuntime = typesViewModelFrmBusqueda.Select(item => item.Name).ToArray();

// Obtengo VM almacenados en DB
string[] nombresVMEnDB = frmBúsqueda_NombresDeVM_EnDB.Select(item => item.NombreDeVM).ToArray();

// Obtengo intersecciones de VM en Runtime y DB
string[] nombresVMNuevos = nombresVMEnRuntime.Where(p1 => nombresVMEnDB.All(p2 => p2 !=
p1)).ToArray();
string[] nombresVMARemoverDeDB = nombresVMEnDB.Where(p1 => !nombresVMEnRuntime.Any(p2 => p2 ==
p1)).ToArray();
string[] nombresVMYaConsolidados = nombresVMEnRuntime.Intersect(nombresVMEnDB).ToArray();
```

Remoción de *ViewModels* inexistentes, y sus entidades relacionadas

En esta situación, simplemente añaden a las listas de entidades de ser eliminadas, todos los *ViewModels* que ya no se encuentren en tiempo de ejecución, y todas sus entidades relacionadas (campos, plantillas, y campos de plantilla).

```
foreach (string nombreVMARemoverDeDB in nombresVMARemoverDeDB)
{
    FrmBúsqueda_NombreDeVM FrmBúsqueda_NombreDeVM = frmBúsqueda_NombresDeVM_EnDB.First(i =>
i.NombreDeVM == nombreVMARemoverDeDB);

    frmBúsqueda_NombresDeVM_ABorrar.Add(FrmBúsqueda_NombreDeVM);
    frmBúsqueda_CamposDeVM_ABorrar.AddRange(frmBúsqueda_CamposDeVM_EnDB.Where(i => i.NombreDeVMId ==
FrmBúsqueda_NombreDeVM.Id));
    frmBúsqueda_NombresDePlantilla_ABorrar.AddRange(frmBúsqueda_NombresDePlantilla_EnDB.Where(i =>
i.NombreDeVMId == FrmBúsqueda_NombreDeVM.Id));
    frmBúsqueda_CamposDePlantilla_ABorrar.AddRange(frmBúsqueda_CamposDePlantilla_EnDB.Where(i =>
i.NombreDeVMId == FrmBúsqueda_NombreDeVM.Id));

frmBúsqueda_PlantillasSeleccionada_ABorrar.AddRange(frmBúsqueda_PlantillasSeleccionada_EnDB.Where(i
=> i.NombreDeVMId == FrmBúsqueda_NombreDeVM.Id));
}
```

Actualización de campos de ViewModel, para ViewModels que ya están presentes en la base de datos

En este caso, se trata de *ViewModels* existentes en tiempo de ejecución, y presentes en la base de datos, pero hay cambios en sus campos (es necesario eliminar algunos, o agregar nuevos).

Los campos que deban ser eliminados, pueden estar vinculados a algún campo de plantilla. De ser el caso, la política que se asume es la de remover el campo de plantilla, pero mantener la plantilla. Esto implica que la plantilla ya no será la misma dado que tendrá menos campos, pero es importante tener presente que estas situaciones sólo ocurrirán en modo debug, por lo cual nunca afectará a un usuario final.

```
Type[] typesViewModelFrmBusquedaConsolidados = typesViewModelFrmBusqueda.Where(t =>
nombresVMYaConsolidados.Contains(t.Name)).ToArray();
foreach (Type typesViewModelFrmBusquedaConsolidado in typesViewModelFrmBusquedaConsolidados)
{
    string[] nombresCamposEnRuntime;
    string[] nombresCamposDeDB;

    // Obtengo NombreDeVM actual en el bucle
    FrmBúsqueda_NombreDeVM FrmBúsqueda_NombreDeVM = frmBúsqueda_NombresDeVM_EnDB.First(i =>
i.NombreDeVM == typesViewModelFrmBusquedaConsolidado.Name);

    // Obtengo nombres de campos de la clase en runtime
    PropertyInfo[] propertyInfos =
typesViewModelFrmBusquedaConsolidado.GetProperties(System.Reflection.BindingFlags.Public |
System.Reflection.BindingFlags.Instance | System.Reflection.BindingFlags.DeclaredOnly);
    nombresCamposEnRuntime = propertyInfos.Select(item => item.Name).ToArray();

    // Obtengo nombres de campos de la clase en DB
    nombresCamposDeDB = frmBúsqueda_CamposDeVM_EnDB.Where(c => c.NombreDeVMId ==
FrmBúsqueda_NombreDeVM.Id).Select(c => c.NombreDelCampo).ToArray();

    // Obtengo intersecciones de VM en Runtime y DB
    string[] camposARemoverDeDB = nombresCamposDeDB.Where(p1 => nombresCamposEnRuntime.All(p2 => p2
!= p1)).ToArray();
    string[] camposAAgregarADeDB = nombresCamposEnRuntime.Where(p1 => nombresCamposDeDB.All(p2 => p2
!= p1)).ToArray();

    // Elimino los campos de VM, y sus campos de plantilla asociados (No se eliminan las plantillas)
    foreach (string campoARemoverDeDB in camposARemoverDeDB)
    {
        FrmBúsqueda_CampoDeVM frmBúsqueda_CampoDeVM = frmBúsqueda_CamposDeVM_EnDB.First(i =>
i.NombreDelCampo == campoARemoverDeDB);

        frmBúsqueda_CamposDeVM_ABorrar.Add(frmBúsqueda_CampoDeVM);
        frmBúsqueda_CamposDePlantilla_ABorrar.AddRange(frmBúsqueda_CamposDePlantilla_EnDB.Where(i =>
i.CampoDeVMId == frmBúsqueda_CampoDeVM.Id));
    }

    // Agrego los campos del VM
    foreach (string campoAAgregarADeDB in camposAAgregarADeDB)
    {
        FrmBúsqueda_CampoDeVM campoViewModelFrmBusqueda = new FrmBúsqueda_CampoDeVM();
        campoViewModelFrmBusqueda.NombreDelCampo = campoAAgregarADeDB;
        campoViewModelFrmBusqueda.NombreDeVMId = FrmBúsqueda_NombreDeVM.Id;
        frmBúsqueda_CamposDeVM_APersistir.Add(campoViewModelFrmBusqueda);
    }
}
```

Persistencia

Luego del procesamiento diferencial para cada circunstancia en la que se podría encontrar un ViewModel, se ejecutan las distintas operaciones de persistencia para cada lista de entidades.

Las operaciones de persistencia, son ejecutadas con un tipo especial de operación desarrollada específicamente para operaciones por lote ("*Bulk*").

```
// Listas para almacenar entidades de mapeo que serán insertadas en DB
CT_FrmBúsqueda_NombresDeVM.BulkInsert(frmBúsqueda_NombresDeVM_APersistir);


// Listas para almacenar entidades de mapeo que serán borradas
CT_FrmBúsqueda_PlantillasSeleccionada.BulkDelete(frmBúsqueda_PlantillasSeleccionada_ABorrar);
CT_FrmBúsqueda_CamposDePlantilla.BulkDelete(frmBúsqueda_CamposDePlantilla_ABorrar);
CT_FrmBúsqueda_NombresDePlantilla.BulkDelete(frmBúsqueda_NombresDePlantilla_ABorrar);
CT_FrmBúsqueda_CamposDeVM.BulkDelete(frmBúsqueda_CamposDeVM_ABorrar);
CT_FrmBúsqueda_NombresDeVM.BulkDelete(frmBúsqueda_NombresDeVM_ABorrar);

foreach (FrmBúsqueda_CampoDeVM campoDeVM in frmBúsqueda_CamposDeVM_APersistir)
{
    if (campoDeVM.NombreDeVMId == 0)
    {
        campoDeVM.NombreDeVMId = campoDeVM.NombreDeVM.Id;
        campoDeVM.NombreDeVM = null;
    }
}

CT_FrmBúsqueda_CamposDeVM.BulkInsert(frmBúsqueda_CamposDeVM_APersistir);
```

Rutina de inicio: Creación de plantillas aleatorias para pruebas

Para contar con un conjunto de plantillas que permitan testear los aspectos funcionales en el futuro, se desarrolló un algoritmo que genera una cantidad aleatoria de plantillas (arbitrariamente, entre tres y siete) para cada *ViewModel*, con una cantidad aleatoria de columnas (arbitrariamente, entre una y la cantidad máxima de campos posibles).

Esta rutina se ejecutará sólo en modo debug, y es una funcionalidad que nunca alcanzará al usuario final.

Se emplea la misma estrategia de organización del código fuente que se utilizó en *la Actualización de algoritmo de mapeo automático de campos de ViewModel, a base de datos*, se utilizan listas con las entidades que al finalizar el algoritmo se aplican a las diferentes operaciones de persistencia, según corresponda.

```
// Listas para almacenar entindades de mapeo
List<FrmBúsqueda_NombreDeVM> frmBúsqueda_NombresDeVM_EnDB;
List<FrmBúsqueda_CampoDeVM> frmBúsqueda_CamposDeVM_EnDB;
List<FrmBúsqueda_NombreDePlantilla> frmBúsqueda_NombresDePlantilla_EnDB;

// Listas para almacenar entidades de mapeo que serán insertadas en DB
List<FrmBúsqueda_NombreDePlantilla> frmBúsqueda_NombresDePlantilla_APersistir = new
List<FrmBúsqueda_NombreDePlantilla>();
List<FrmBúsqueda_CampoDePlantilla> frmBúsqueda_CamposDePlantilla_APersistir = new
List<FrmBúsqueda_CampoDePlantilla>();

// Obtengo todas las entidades de mapeo desde la DB
frmBúsqueda_NombresDeVM_EnDB = CT_FrmBúsqueda_NombresDeVM.ToList();
frmBúsqueda_CamposDeVM_EnDB = CT_FrmBúsqueda_CamposDeVM.ToList();
frmBúsqueda_NombresDePlantilla_EnDB = CT_FrmBúsqueda_NombresDePlantilla.ToList();

Random random = new Random((int)DateTime.Now.Ticks);

// Para cada NombreDeVM sin plantilla, creo una plantilla con sus campos asociados
foreach (FrmBúsqueda_NombreDeVM nombreDeVM in frmBúsqueda_NombresDeVM_EnDB)
{
```

```
    int cantidadDePlantillas = random.Next(3, 7);

    for (int i = 1; i <= cantidadDePlantillas; i++)
    {
        // Creo plantilla genérica
        FrmBúsqueda_NombreDePlantilla nombreDePlantilla = new FrmBúsqueda_NombreDePlantilla();
        nombreDePlantilla.NombreDePlantilla = string.Format("Genérica_{0}", i);
        nombreDePlantilla.NombreDeVMId = nombreDeVM.Id;
        frmBúsqueda_NombresDePlantilla_APersistir.Add(nombreDePlantilla);

        // Obtengo campos del VM en curso en el bucle, y los reordeno aleatoriamente
        FrmBúsqueda_CampoDeVM[] camposDeVM = frmBúsqueda_CamposDeVM_EnDB.Where(c => c.NombreDeVMId
== nombreDeVM.Id).OrderBy(x => random.Next()).ToArray();
        // Conservo una cantidad aleatoria de campos (para que cada Plantilla tenga distinta candiad
de elementos)
        camposDeVM = camposDeVM.Take(random.Next(1, camposDeVM.Length)).ToArray();


        int visibleIndex = 0;
        // Creo campos de plantilla
        foreach (FrmBúsqueda_CampoDeVM campoDeVM in camposDeVM)
        {
            FrmBúsqueda_CampoDePlantilla campoDePlantilla = new FrmBúsqueda_CampoDePlantilla();
            campoDePlantilla.NombreDeVMId = nombreDeVM.Id;
            campoDePlantilla.CampoDeVMId = campoDeVM.Id;
            campoDePlantilla.NombreDePlantilla = nombreDePlantilla;

            campoDePlantilla.VisibleIndex = visibleIndex;
            visibleIndex++;
            campoDePlantilla.HeaderFontBold = false;
            campoDePlantilla.HeaderFontItalic = false;
            campoDePlantilla.HeaderFontUnderline = false;
            campoDePlantilla.CellFontBold = false;
            campoDePlantilla.CellFontItalic = false;
            campoDePlantilla.CellFontUnderline = false;

            frmBúsqueda_CamposDePlantilla_APersistir.Add(campoDePlantilla);
        }
    }
}

CT_FrmBúsqueda_NombresDePlantilla.BulkInsert(frmBúsqueda_NombresDePlantilla_APersistir);

foreach (FrmBúsqueda_CampoDePlantilla camposDePlantilla in frmBúsqueda_CamposDePlantilla_APersistir)
{
    camposDePlantilla.NombreDePlantillaId = camposDePlantilla.NombreDePlantilla.Id;
    camposDePlantilla.NombreDePlantilla = null;
}

CT_FrmBúsqueda_CamposDePlantilla.BulkInsert(frmBúsqueda_CamposDePlantilla_APersistir);
```

Actualización a aplicación de configuración de columnas automática en Formulario de Búsqueda (base)

Respecto a la versión anterior, que operaba con el modelo de datos previo basado en una única entidad de datos, el cambio fundamental en esta nueva versión, es saber si hay o no seleccionada una plantilla de configuración para el *Formulario de Búsqueda*.

Si hay una plantilla seleccionada, entonces se recuperar sus campos, y las configuraciones gráficas se aplican del mismo modo en que se hizo en la versión previa. De no existir una plantilla previamente seleccionada, se ocultan todos los campos de la grilla.

Además, se agregó una mejora estética, para que en el proceso de actualización de la grilla se muestre un indicador de tarea en curso.

```
private void AplicarConfiguracionAColumnas()
{
    // Recupero PlantillaSeleccionada por defecto de Db
    FrmBúsqueda_PlantillaSeleccionada plantillaSeleccionada =
CT_FrmBúsqueda_PlantillasSeleccionada.ObtenerPorNombreViewModel(NombreDeVM);

    FrmBúsqueda_NombreDePlantilla nombreDePlantillaSeleccionada = (plantillaSeleccionada == null) ?
null : plantillaSeleccionada.NombreDePlantilla;

    this.ShowProgressPanel();

    GridColumn[] gridColumns = this.gvResultadosDeBusqueda.Columns.ToArray();

    //* Si no hay plantilla seleccionada, oculto todas las columnas
    if (this.nombreDePlantillaSeleccionada == null)
    {
        this.gcResultadosDeBusqueda.BeginUpdate();

        // Obtengo los GridColums actualmente utilizados en la grilla

        foreach (GridColumn gridColumn in gridColumns)
        {
            gridColumn.Visible = false;
        }

        this.gcResultadosDeBusqueda.EndUpdate();
    }
    else
    {
        this.camposDePlantilla =
CT_FrmBúsqueda_CamposDePlantilla.ObtenerPorIdDePlantilla(nombreDePlantillaSeleccionada.Id);

        // Hago consistentes el estado de "Visible" entre los GridColumns y los CamposDePlantilla
        foreach (GridColumn gridColumn in gridColumns)
        {
            FrmBúsqueda_CampoDePlantilla campoDePlantilla = this.camposDePlantilla.FirstOrDefault(c
=> c.CampoDeVM.NombreDelCampo == gridColumn.FieldName);

            gridColumn.Visible = (campoDePlantilla != null);
        }

        /* Atención, si se detiene el renderizado en el GridControl, y luego se ocultan/muestran
columnas,
        * dichas columnas no responderán al evento ColumnPositionChanged que se utiliza para
actualizar
        * el contenido de las columnas aún vacías.
        * En resumen, BeginUpdate debe estar luego de las operaciones de mostar/ocultar columnas. */

        this.gcResultadosDeBusqueda.BeginUpdate();

        // Aplico el resto de las configuraciones restantes (Orden y estética)
        foreach (FrmBúsqueda_CampoDePlantilla campoDePlantilla in this.camposDePlantilla)
        {
            // Establezco el orden de las columnas

this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].VisibleIndex =
(int)campoDePlantilla.VisibleIndex;

            // Establezco color de fondo de las cabeceras
            if (campoDePlantilla.HeaderBackColor != null)
            {
                Color appearanceHeaderBackColor =
Color.FromArgb((int)campoDePlantilla.HeaderBackColor);

this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceHeader.Back
```

```
Color = Color.FromArgb(appearanceHeaderBackColor.A, appearanceHeaderBackColor.R,
appearanceHeaderBackColor.G, appearanceHeaderBackColor.B);
            }

            // Establezco color de texto de las cabeceras
            if (campoDePlantilla.HeaderForeColor != null)
            {
                Color appearanceHeaderForeColor =
Color.FromArgb((int)campoDePlantilla.HeaderForeColor);

this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceHeader.Fore
Color = Color.FromArgb(appearanceHeaderForeColor.A, appearanceHeaderForeColor.R,
appearanceHeaderForeColor.G, appearanceHeaderForeColor.B);
            }

            // Establezco configuraciones de fuente de las cabeceras
            if (campoDePlantilla.HeaderFontSize != null ||
                campoDePlantilla.HeaderFontBold != false ||
                campoDePlantilla.HeaderFontItalic != false ||
                campoDePlantilla.HeaderFontUnderline != false)
            {
                float fontSize;
                FontStyle fontStyle;

                // Obtengo fuente asignada por el sistema a la columna
                Font font =
this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceHeader.GetF
ont();

                if (campoDePlantilla.HeaderFontSize != null)
                {
                    fontSize = (float)campoDePlantilla.HeaderFontSize;
                }
                else
                {
                    fontSize = font.SizeInPoints;
                }

                fontStyle = FontStyle.Regular;
                if (campoDePlantilla.HeaderFontBold != false)
                {
                    fontStyle |= FontStyle.Bold;
                }
                if (campoDePlantilla.HeaderFontItalic != false)
                {
                    fontStyle |= FontStyle.Italic;
                }
                if (campoDePlantilla.HeaderFontUnderline != false)
                {
                    fontStyle |= FontStyle.Underline;
                }

                // Actualizo fuente original con los parámetros preferidos por el usuario
                font = new System.Drawing.Font(font.FontFamily, fontSize,
((System.Drawing.FontStyle)fontStyle), System.Drawing.GraphicsUnit.Point, ((byte)(0)));

                // Asigno fuente actualizada a la columna

this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceHeader.Font
= font;
            }

            // Establezco color de fondo de las celdas
            if (campoDePlantilla.CellBackColor != null)
            {
                Color appearanceCellBackColor = Color.FromArgb((int)campoDePlantilla.CellBackColor);

this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceCell.BackCo
```

```
lor = Color.FromArgb(appearanceCellBackColor.A, appearanceCellBackColor.R,
appearanceCellBackColor.G, appearanceCellBackColor.B);
            }

            // Establezco color de fondo de las celdas (segundo color de fondo)
            if (campoDePlantilla.CellBackColor2 != null)
            {
                Color appearanceCellBackColor2 =
Color.FromArgb((int)campoDePlantilla.CellBackColor2);

this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceCell.BackCo
lor2 = Color.FromArgb(appearanceCellBackColor2.A, appearanceCellBackColor2.R,
appearanceCellBackColor2.G, appearanceCellBackColor2.B);
            }

            // Establezco color de texto de las celdas
            if (campoDePlantilla.CellForeColor != null)
            {
                Color cellHeaderForeColor = Color.FromArgb((int)campoDePlantilla.CellForeColor);

this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceCell.ForeCo
lor = Color.FromArgb(cellHeaderForeColor.A, cellHeaderForeColor.R, cellHeaderForeColor.G,
cellHeaderForeColor.B);
            }

            // Establezco configuraciones de fuente de las celdas
            if (campoDePlantilla.CellFontSize != null ||
                campoDePlantilla.CellFontBold != false ||
                campoDePlantilla.CellFontItalic != false ||
                campoDePlantilla.CellFontUnderline != false)
            {
                float fontSize;
                FontStyle fontStyle;

                // Obtengo fuente asignada por el sistema a la columna
                Font font =
this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceCell.GetFon
t();

                if (campoDePlantilla.CellFontSize != null)
                {
                    fontSize = (float)campoDePlantilla.CellFontSize;
                }
                else
                {
                    fontSize = font.SizeInPoints;
                }

                fontStyle = FontStyle.Regular;
                if (campoDePlantilla.CellFontBold != false)
                {
                    fontStyle |= FontStyle.Bold;
                }
                if (campoDePlantilla.CellFontItalic != false)
                {
                    fontStyle |= FontStyle.Italic;
                }
                if (campoDePlantilla.CellFontUnderline != false)
                {
                    fontStyle |= FontStyle.Underline;
                }

                // Actualizo fuente original con los parámetros preferidos por el usuario
                font = new System.Drawing.Font(font.FontFamily, fontSize,
((System.Drawing.FontStyle)fontStyle), System.Drawing.GraphicsUnit.Point, ((byte)(0)));

                // Asigno fuente actualizada a la columna
```

```
this.gvResultadosDeBusqueda.Columns[campoDePlantilla.CampoDeVM.NombreDelCampo].AppearanceCell.Font =
font;
            }
        }

        this.gcResultadosDeBusqueda.EndUpdate();
    }

    this.CloseProgressPanel();
}
```

Actualización a guardado de configuración de columnas en Formulario de Búsqueda (base)

Respecto a la versión anterior, que operaba con el modelo de datos previo basado en una única entidad de datos, el cambio fundamental en esta nueva versión, es saber si hay o no seleccionada una plantilla de configuración para el *Formulario de Búsqueda*, y al momento del cierre del formulario persistir los cambios.

De esta manera, en conjunto con la actualización a la aplicación de configuración de columnas automática, el usuario final percibirá en cada apertura del formulario, que el estado reflejado es "tal cual lo había dejado configurado".

Detección de cambios en la configuración de columnas, con respecto a lo persistido

Previa a la ejecución del algoritmo que verifica la consistencia entre el estado de la plantilla en tiempo de ejecución, y la plantilla almacenada en base de datos, se ejecuta un algoritmo que detecta si hay algún cambio.

```
private bool HayCambiosEnConfiguraciónDeColumnas()
{
    // Obtengo los GridColums actualmente utilizados en la grilla
    GridColumn[] gridColumns = this.gvResultadosDeBusqueda.Columns.ToArray();

    // Obtengo PlantillaSeleccionada de Db para realizar la comparación
    FrmBúsqueda_PlantillaSeleccionada plantillaSeleccionadaEnDb =
CT_FrmBúsqueda_PlantillasSeleccionada.ObtenerPorVM(NombreDeVM);

    // Niego la condición siguiente:
    // Son iguales si ambos son nulos, o si ambos son no nulos y refieren al mismo Id de
NombreDePlantilla
    if (!((this.nombreDePlantillaSeleccionada == null && plantillaSeleccionadaEnDb == null) ||
        (this.nombreDePlantillaSeleccionada != null && plantillaSeleccionadaEnDb != null &&
this.nombreDePlantillaSeleccionada.Id == plantillaSeleccionadaEnDb.NombreDePlantillaId)))
    {
        return true;
    }

    /* Si llego a este punto es porque el formulario no tiene plantilla seleccionada (es nula),
        * o porque la plantilla seleccionada en runtime es la misma que se registró en DB, en
        * cuyo caso hay que verificar que no haya cambios estéticos. */
    if (this.nombreDePlantillaSeleccionada != null)
    {
        // Obtengo los CampoDePlantilla de Db para realizar la comparación con los de Runtime
        FrmBúsqueda_CampoDePlantilla[] camposDePlantilla =
CT_FrmBúsqueda_CamposDePlantilla.ObterPorPlantillaId(this.nombreDePlantillaSeleccionada.Id);

        /* Estrategia: Para cada GridColumn buscamos su entidad CampoDePlantilla
            * asociada (almacenadas en this.camposViewModelFrmBusqueda) y comparamos todas
            * sus propiedades, si alguna de las propiedades es diferente se retorna true
            * de inmediato, sino, al finalizar el algoritmo se retorna false. */
        foreach (GridColumn gridColumn in gridColumns)
```

```
        {
            FrmBúsqueda_CampoDePlantilla campoDePlantillaAsociado =
this.camposDePlantilla.FirstOrDefault(c => c.CampoDeVM.NombreDelCampo == gridColumn.FieldName);

            /* Recordar que los GridColumn que son no-visibles no almacenan un CampoDePlantilla en
Db.
                * Si se encuentra un GridColumn visible sin CampoDePlantilla en Db, o un GridColumn
no-visible
                * con un CampoDePlantilla en Db, implica que ha habido cambios en esa columna. */
            if ((gridColumn.Visible == false && campoDePlantillaAsociado != null) ||
                (gridColumn.Visible == true && campoDePlantillaAsociado == null))
            {
                return true;
            }

            /* Finalmente, si el GridColumn es visible, es necesario verificar si ha habido cambios
de
                * configuración estética en el mismo. */
            if (gridColumn.Visible == true &&

                (gridColumn.VisibleIndex != campoDePlantillaAsociado.VisibleIndex ||

                !((gridColumn.AppearanceHeader.BackColor.IsEmpty == true &&
campoDePlantillaAsociado.HeaderBackColor == null) ||
                (gridColumn.AppearanceHeader.BackColor.ToArgb() ==
campoDePlantillaAsociado.HeaderBackColor)) ||

                !((gridColumn.AppearanceHeader.ForeColor.IsEmpty == true &&
campoDePlantillaAsociado.HeaderForeColor == null) ||
                (gridColumn.AppearanceHeader.ForeColor.ToArgb() ==
campoDePlantillaAsociado.HeaderForeColor)) ||

                gridColumn.AppearanceHeader.Font.Size != campoDePlantillaAsociado.HeaderFontSize ||
                gridColumn.AppearanceHeader.Font.Bold != campoDePlantillaAsociado.HeaderFontBold ||
                gridColumn.AppearanceHeader.Font.Italic != campoDePlantillaAsociado.HeaderFontItalic
||
                gridColumn.AppearanceHeader.Font.Underline !=
campoDePlantillaAsociado.HeaderFontUnderline ||

                !((gridColumn.AppearanceCell.BackColor.IsEmpty == true &&
campoDePlantillaAsociado.CellBackColor == null) ||
                (gridColumn.AppearanceCell.BackColor.ToArgb() ==
campoDePlantillaAsociado.CellBackColor)) ||

                !((gridColumn.AppearanceCell.BackColor2.IsEmpty == true &&
campoDePlantillaAsociado.CellBackColor2 == null) ||
                (gridColumn.AppearanceCell.BackColor2.ToArgb() ==
campoDePlantillaAsociado.CellBackColor2)) ||

                !((gridColumn.AppearanceCell.ForeColor.IsEmpty == true &&
campoDePlantillaAsociado.CellForeColor == null) ||
                (gridColumn.AppearanceCell.ForeColor.ToArgb() ==
campoDePlantillaAsociado.CellForeColor)) ||

                gridColumn.AppearanceCell.Font.Size != campoDePlantillaAsociado.CellFontSize ||
                gridColumn.AppearanceCell.Font.Bold != campoDePlantillaAsociado.CellFontBold ||
                gridColumn.AppearanceCell.Font.Italic != campoDePlantillaAsociado.CellFontItalic ||
                gridColumn.AppearanceCell.Font.Underline !=
campoDePlantillaAsociado.CellFontUnderline
                ))
            {
                return true;
            }
        }
    }

    return false;
}
```

Actualización al algoritmo de base

Es fundamental recorrer este algoritmo, teniendo presente que el mismo es ejecutado luego de la ejecución de *HayCambiosEnConfiguraciónDeColumnas()*, esto significa que nos basamos en el hecho que algún cambio ha ocurrido. La política de guardado de configuración es eliminar todas las configuraciones previas, y guardar las nuevas, no se actualiza. Por esta razón es, que para cualquiera sea la plantilla que se haya seleccionado, se eliminarán todos sus campos, y se persistirán nuevos. Lo mismo ocurre para la entidad *PlantillaSelccionada*, se elimina la anterior, y se persiste una nueva (incluso si apuntan a las mismas referencias).

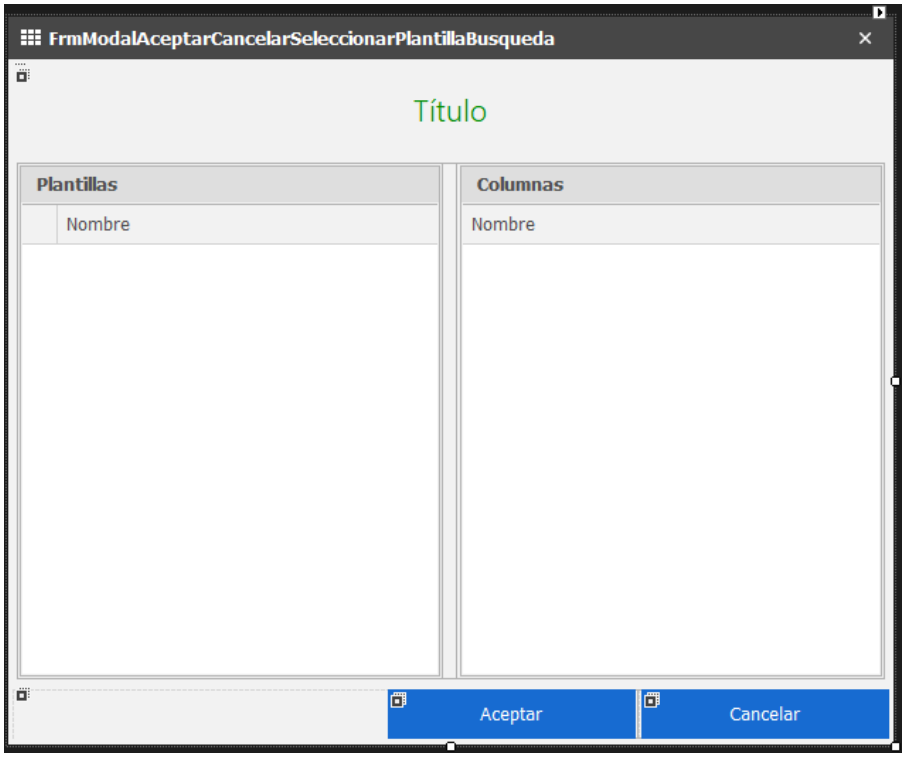
```
private bool GuardarConfiguraciónDeColumnas()
{
    bool blnRetorno = true;

    // Lista para almacenar los nuevos CamposDePlantilla a persistir
    List<FrmBúsqueda_CampoDePlantilla> camposDePlantillaAPersistir = new
List<FrmBúsqueda_CampoDePlantilla>();

    // Nueva PlantillaSeleccionada a persistir
    FrmBúsqueda_PlantillaSeleccionada plantillaSeleccionadaAPersistir = new
FrmBúsqueda_PlantillaSeleccionada();
    plantillaSeleccionadaAPersistir.NombreDeVMId = this.nombreDePlantillaSeleccionada.NombreDeVMId;
    plantillaSeleccionadaAPersistir.NombreDePlantillaId = this.nombreDePlantillaSeleccionada.Id;

    // Obtengo los CamposDeVM (que se corresponden con cada GridColumn)
    FrmBúsqueda_CampoDeVM[] camposDeVm =
CT_FrmBúsqueda_CamposDeVM.ObtenerPorNombreDeVMId(this.nombreDePlantillaSeleccionada.NombreDeVMId);

    // Obtengo los GridColums actualmente utilizados en la grilla
    GridColumn[] gridColumns = this.gvResultadosDeBusqueda.Columns.Where(g => g.Visible ==
true).ToArray();

    foreach (GridColumn gridColumn in gridColumns)
    {
        // Creo CampoDePlantilla vacío, y lo agrego a la lista APersistir
        FrmBúsqueda_CampoDePlantilla campoDePlantilla = new FrmBúsqueda_CampoDePlantilla();
        camposDePlantillaAPersistir.Add(campoDePlantilla);

        // Asigno el CampoDeVM correspondiente
        FrmBúsqueda_CampoDeVM campoDeVm = camposDeVm.First(c => c.NombreDelCampo ==
gridColumn.FieldName);
        campoDePlantilla.CampoDeVMId = campoDeVm.Id;

        // Asigno el resto de las claves primarias
        campoDePlantilla.NombreDePlantillaId = this.nombreDePlantillaSeleccionada.Id;
        campoDePlantilla.NombreDeVMId = this.nombreDePlantillaSeleccionada.NombreDeVMId;

        // Asisgno las propiedades del GridColumn al CampoDePlantilla:
        campoDePlantilla.VisibleIndex = gridColumn.VisibleIndex;

        if (gridColumn.AppearanceHeader.BackColor.IsEmpty == true)
        {
            campoDePlantilla.HeaderBackColor = null;
        }
        else
        {
            campoDePlantilla.HeaderBackColor = gridColumn.AppearanceHeader.BackColor.ToArgb();
        }

        if (gridColumn.AppearanceHeader.ForeColor.IsEmpty == true)
        {
            campoDePlantilla.HeaderForeColor = null;
        }
        else
        {
            campoDePlantilla.HeaderForeColor = gridColumn.AppearanceHeader.ForeColor.ToArgb();
```

```
        }

        campoDePlantilla.HeaderFontSize = gridColumn.AppearanceHeader.Font.Size;
        campoDePlantilla.HeaderFontBold = gridColumn.AppearanceHeader.Font.Bold;
        campoDePlantilla.HeaderFontItalic = gridColumn.AppearanceHeader.Font.Italic;
        campoDePlantilla.HeaderFontUnderline = gridColumn.AppearanceHeader.Font.Underline;

        if (gridColumn.AppearanceCell.BackColor.IsEmpty == true)
        {
            campoDePlantilla.CellBackColor = null;
        }
        else
        {
            campoDePlantilla.CellBackColor = gridColumn.AppearanceCell.BackColor.ToArgb();
        }

        if (gridColumn.AppearanceCell.BackColor2.IsEmpty == true)
        {
            campoDePlantilla.CellBackColor2 = null;
        }
        else
        {
            campoDePlantilla.CellBackColor2 = gridColumn.AppearanceCell.BackColor2.ToArgb();
        }

        if (gridColumn.AppearanceCell.ForeColor.IsEmpty == true)
        {
            campoDePlantilla.CellForeColor = null;
        }
        else
        {
            campoDePlantilla.CellForeColor = gridColumn.AppearanceCell.ForeColor.ToArgb();
        }

        campoDePlantilla.CellFontSize = gridColumn.AppearanceCell.Font.Size;
        campoDePlantilla.CellFontBold = gridColumn.AppearanceCell.Font.Bold;
        campoDePlantilla.CellFontItalic = gridColumn.AppearanceCell.Font.Italic;
        campoDePlantilla.CellFontUnderline = gridColumn.AppearanceCell.Font.Underline;
    }

    // Persisto las actualizaciones
    // Elimino todos los campos de plantilla, y la información de plantilla seleccionada
    FrmBúsqueda_CampoDePlantilla[] camposDePlantillaAEliminar =
CT_FrmBúsqueda_CamposDePlantilla.ObtenerPorNombreDePlantillaId(this.nombreDePlantillaSeleccionada.Id
);

    FrmBúsqueda_PlantillaSeleccionada[] plantillaSeleccionadaAEliminar =
CT_FrmBúsqueda_PlantillasSeleccionada.ObtenerPorNombreDeVMId(this.nombreDePlantillaSeleccionada.Nomb
reDeVMId);

    CT_FrmBúsqueda_CamposDePlantilla.RemoveRange(camposDePlantillaAEliminar);
    CT_FrmBúsqueda_PlantillasSeleccionada.RemoveRange(plantillaSeleccionadaAEliminar);

    // Inserto los nuevos campos de plantilla, y la informaicón de la plantilla seleccionada
    CT_FrmBúsqueda_CamposDePlantilla.AddRange(camposDePlantillaAPersistir);
    CT_FrmBúsqueda_PlantillasSeleccionada.Add(plantillaSeleccionadaAPersistir);

    /* Actualizo los CamposDePlantilla locales, acorde a los que fueron insertados en DB
    * (Recordar que se utilizan para corroborar si hubo cambios, y si no se actualizan
    * en este momento, prevervarán el estado previo. */
    this.camposDePlantilla =
model.FrmBúsqueda_CamposDePlantilla.ObtenerPorNombreDePlantillaId(nombreDePlantillaSeleccionada.Id);
    return blnRetorno;
}
```

User Story ID 12 - Formulario de búsqueda: Seleccionar plantilla

Formulario modal Aceptar-Cancelar: Seleccionar plantilla

Se desarrolló el *Formulario de búsqueda: Selecionar plantilla*, acorde a las especificaciones funcionales, y el comportamiento definido en wireframes.



El formulario recibe por parámetro en su constructor, el tipo de datos de un *ViewModel* asociado a un *Formulario de Búsqueda*, para el cual se construirá una plantilla, y el nombre de la plantilla actualmente seleccionada, en caso de haber una.

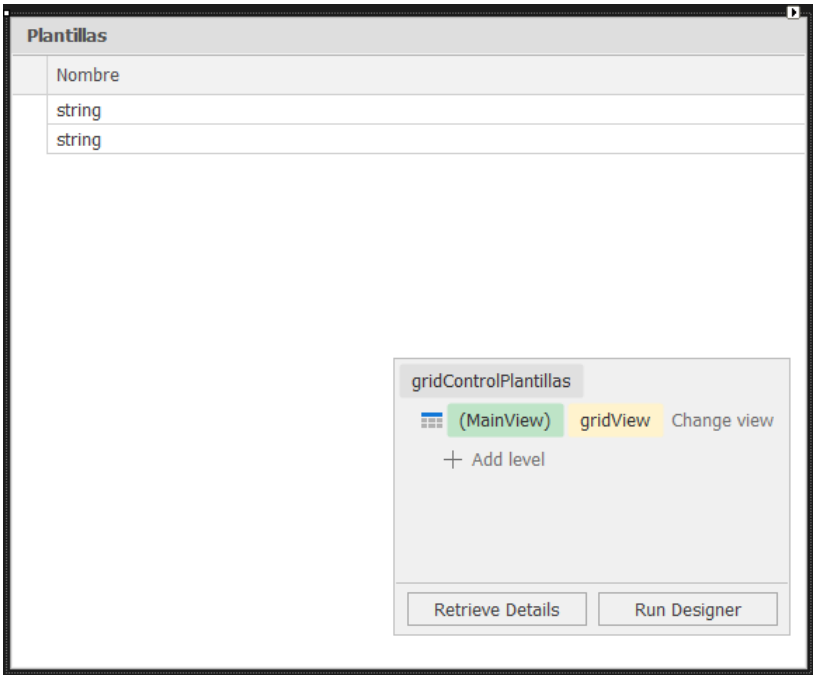
En la columna de la izquierda se muestran todas las plantillas disponibles para el *ViewModel*. Siempre que se seleccione una plantilla, a la derecha se listarán todas las columnas que conforman la misma, a la derecha.

Por parámetro se recibe la plantilla seleccionada actualmente, porque, de existir alguna selección, en el listado se establece como primera selección.

Hay dos acciones importantes en el formulario que serán explicadas a continuación, que respectan a eventos que ocurren sobre el listado de plantillas: efectuar un click (o el equivalente a seleccionar un elemento del listado), o doble click (o el equivalente a confirmar que se desea utilizar una plantilla específica).

Tanto el *Listado de plantillas* como el *Listado de columnas* son controles de usuarios independientes. El *Listado de plantillas* permite que se le adjunten eventos para utilizar acciones del exterior (métodos definidos en otra clase, fuera de la clase del control de usuario) del componente cuando los eventos ocurren.

Control de usuario: Listar plantillas de Formulario de Búsqueda



El control de usuario tiene definidos dos EventHandlers[62], que permiten adjuntar acciones definidas por fuera del control de usuario, y que serán ejecutadas al dispararse determinados eventos.

El primer *EventHandler* responde al nombre de *PlantillaSeleccionada*. Este evento se dispara al efectuar click sobre una fila (o cualquier otra acción que implique selección, como utilizar las flechas arriba/abajo del teclado). Inmediatamente luego de la acción, se invoca el método adjunto al *EventHanlder*, y se le envía la entidad de datos de la plantilla seleccionada.

Esta situación es particularmente útil, cuando el comportamiento que se pretende en el *Formulario modal Aceptar-Cancelar: Seleccionar plantilla*, que al hacer click sobre una plantilla, se listen sus columnas en el listado de la derecha.

Es el formulario que aloja a ambos controles de usuario, el que efectúa el rol del "observador", y atento a los disparos de eventos del control de usuario de la izquierda, invoca acciones en el control de usuario de la derecha.

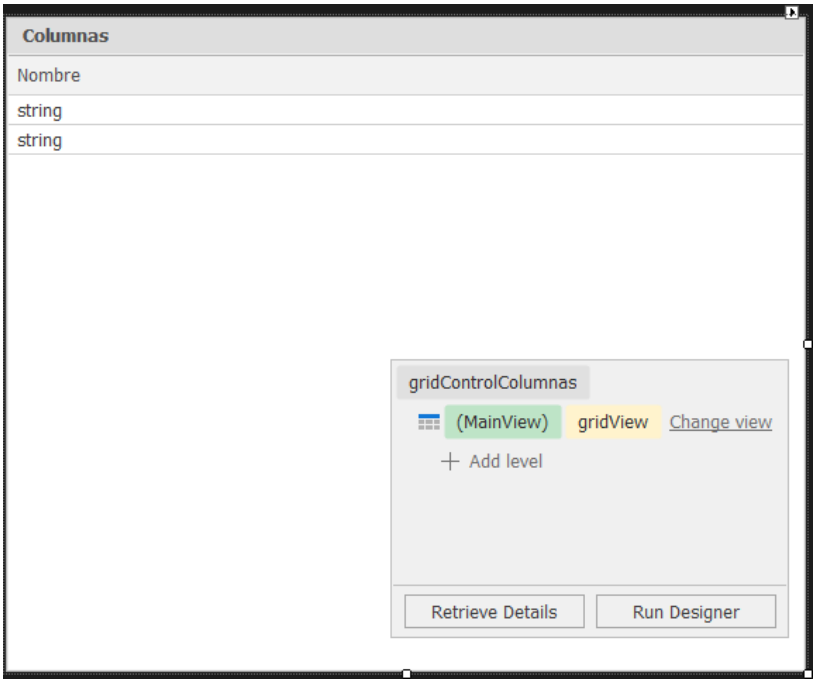
A continuación, se presenta el desarrollo de los *EventHandlers* del control de usuario:

```
#region Eventos

private void gridView_DoubleClick(object sender, EventArgs e)
{
    /* 1. Obtiene la entidad seleccionada en FocusedRowChanged.
        * 2. La setea en el getter NombreDePlantillaSeleccionada.
        * 3. Invoca al event handler OnDobleClicEnPlantilla.
        * */
```

[62] Documentación oficial de EventHandler: https://docs.microsoft.com/en-us/dotnet/api/system.eventhandler

```
    FrmBúsqueda_NombreDePlantilla entidadSeleccionada =
(FrmBúsqueda_NombreDePlantilla)gridView.GetFocusedRow();

    // Seteo la variable para obtención externa (por get)
    this.nombreDePlantillaSeleccionada = entidadSeleccionada;

    // Llamada al event handler
    if (entidadSeleccionada != null)
    {
        PlantillaSeleccionadaEventArgs args = new PlantillaSeleccionadaEventArgs();
        args.NombreDePlantilla = entidadSeleccionada;

        OnDobleClicEnPlantilla(args);
    }
}

private void gridView_FocusedRowChanged(object sender,
DevExpress.XtraGrid.Views.Base.FocusedRowChangedEventArgs e)
{
    /* 1. Obtiene la entidad seleccionada en FocusedRowChanged.
        * 2. La setea en el getter NombreDePlantillaSeleccionada.
        * 3. Invoca al event handler OnPlantillaSeleccionada.
        * */

    FrmBúsqueda_NombreDePlantilla entidadSeleccionada =
(FrmBúsqueda_NombreDePlantilla)gridView.GetFocusedRow();

    // Seteo la variable para obtención externa (por get)
    this.nombreDePlantillaSeleccionada = entidadSeleccionada;

    // Llamada al event handler
    if (entidadSeleccionada != null)
    {
        PlantillaSeleccionadaEventArgs args = new PlantillaSeleccionadaEventArgs();
        args.NombreDePlantilla = entidadSeleccionada;

        OnPlantillaSeleccionada(args);
    }
}

#endregion // Eventos

#region Métodos protegidos
protected virtual void OnDobleClicEnPlantilla(PlantillaSeleccionadaEventArgs e)
{
    DobleClicEnPlantilla?.Invoke(this, e);
}

protected virtual void OnPlantillaSeleccionada(PlantillaSeleccionadaEventArgs e)
{
    PlantillaSeleccionada?.Invoke(this, e);
}

#endregion // Métodos protegidos
```

Control de usuario: Listar columnas de plantilla de Formulario de Búsqueda



Dado que la lógica mediante la cual se listan los campos de un *ViewModel* tiene cierta complejidad, y que la lógica de selección de una plantilla podría separarse con facilidad de la lógica que implica la visualización de las columnas de una plantilla, se desarrolló un control de usuario que encapsula dichas características, para favorecer la separación de responsabilidades.

Hay una lógica extra en la visualización de las columnas de una plantilla (o bien, los campos de un *ViewModel*), inherente del esquema de globalización elegido que consiste en el uso de *DataAnnotations* y archivos de recursos, que añade lógica particular que no comparte el dominio con la selección de una plantilla, y que fácilmente puede ser separada.

Recordar que cada propiedad de un *ViewModel* tiene asociado un archivo de recursos, y una clave, con los cuales se obtiene, dependiendo del idioma en ejecución, el nombre para mostrar de una propiedad, que posteriormente se refleja en la cabecera de una columna.

Como se mostró previamente, la grilla de *DevExpress* está preparada para interpretar las decoraciones de *DisplayAttribute*, en el namespace *DataAnnotations*, por lo cual la lógica de traducción del nombre de una propiedad se realiza de forma transparente al desarrollador.

Para listar los campos de un *ViewModel*, es necesario desarrollar una lógica que permita *localizar* su nombre asociado, valiéndose de la información en su *DisplayAttribute*, tal y como lo hace la grilla de *DevExpress*. No existe dentro de las librerías de .Net una herramienta que permita la traducción del nombre de un campo de forma directa, por lo cual se ha desarrollado una herramienta a medida.

Localización de campos de ViewModel

Dado el nombre de un *ViewModel*, y el nombre de uno de sus campos, es necesario retornar su nombre para mostrar localizado, basado en el archivo de recursos que tenga asignado.

Anteriormente se utilizó una estrategia en la cual se obtenían los metadatos de una propiedad de una clase, la cual tomaba como elemento de partida, el tipo de datos de la clase. Por esta razón, en primera instancia se generará el tipo de datos de una clase, basado en su nombre.

Es posible generar el tipo de datos de una clase en tiempo de ejecución, conociendo su nombre de clase y el ensamblado al que pertenece, mediante el método *GetType*[63], de *Type*[64].

Contando con el tipo de dato de la entidad, es posible recuperar el *MemberInfo* de la propiedad, tal y como se realizó previamente en *Extracción de propiedades de clases en tiempo de ejecución*.

Nuevamente haciendo uso del namespace *Reflection*, se hará uso del método *GetCustomAttributes*[65] de la clase *CustomAttributeData*[66], para obtener los atributos con los que se decoraron las propiedades del *ViewModel*.

Cómo se especificó en *Uso de ViewModels y GridControl*, los atributos con los que se definieron el archivo de recursos y la clave para obtener la cadena localizada, son *Name* y *ResourceType*. Se obtienen el contenido de ambos atributos mediante casteo de tipos, a los tipos adecuados.

Finalmente, se instancia el archivo de recursos obtenido por *ResourceType*, y se busca la clave por l atributo *Name*, y se retorna su contenido.

```
private string ObtenerNombreDelCampoLocalizado(string nombreDelVMString, string nombreDelCampo)
{
    /* Como sé que el VM que estoy buscando está en el mismo ensamblado que VM_FrmBusquedaBase,
    * obtengo el ensamblado mediante dicha clase, y entonces obtengo el Type del VM buscado. */
    string nameSpacePlusClassName = string.Format("{0}.{1}", typeof(VM_FrmBusquedaBase).Namespace,
nombreDelVMString);
    Type vmType = Type.GetType(nameSpacePlusClassName);

    /* Mediante el Type del VM, obtengo la propidad deseada por su nombre, y luego, buesco el
        * DisplayNameAttribute, que es lo que queremos mostrar en el listado (el nombre traducido).
*/
    MemberInfo property = vmType.GetProperty(nombreDelCampo);

    var customAttributes = CustomAttributeData.GetCustomAttributes(property).FirstOrDefault();

    if (customAttributes == null)
    {
        return nombreDelCampo;
    }

    var customAttributeName = customAttributes.NamedArguments.FirstOrDefault(n => n.MemberName ==
"Name");
    var name = (customAttributeName != null) ? (string)customAttributeName.TypedValue.Value : null;

    var customAttributeResourceType = customAttributes.NamedArguments.FirstOrDefault(n =>
n.MemberName == "ResourceType");
    var resourceType = (customAttributeResourceType != null) ?
(Type)customAttributeResourceType.TypedValue.Value : null;
```

[63] Documentación oficial de GetType: https://docs.microsoft.com/en-us/dotnet/api/system.type.gettype
[64] Documentación oficial de Type: https://docs.microsoft.com/en-us/dotnet/api/system.type
[65] Documentación oficial de GetCustomAttributes: https://docs.microsoft.com/en-us/dotnet/api/system.reflection.customattributedata.getcustomattributes
[66] Documentación oficial de CustomAttributeData: https://docs.microsoft.com/en-us/dotnet/api/system.reflection.customattributedata

```
    if (name != null && resourceType != null)
    {
        var resx = new ComponentResourceManager(resourceType);
        return resx.GetString(name);
    }
    else
    {
        return nombreDelCampo;
    }
}
```

Formulario modal Aceptar-Cancelar: Seleccionar plantilla como observador

Se mencionó en *Control de usuario: Listar plantillas de Formulario de Búsqueda* que el *Formulario modal Aceptar-Cancelar: Seleccionar plantilla* actúa como "observador" en los eventos definidos en Control de usuario: Listar plantillas de Formulario de Búsqueda.

El Formulario modal Aceptar-Cancelar: Seleccionar plantilla adjunta la lógica particular que debe ser ejecutada en cada evento, de la siguiente manera:

```
this.ucListarPlantillasFrmBúsqueda.DobleClicEnPlantilla += new
System.EventHandler(this.ucListarPlantillasFrmBúsqueda_DobleClicEnPlantilla);
this.ucListarPlantillasFrmBúsqueda.PlantillaSeleccionada += new
System.EventHandler(this.ucListarPlantillasFrmBúsqueda_PlantillaSeleccionada);
```

Donde la lógica adjunta, correspondiente a cada evento, es la siguiente:

```
#region Eventos

private void ucListarPlantillasFrmBúsqueda_DobleClicEnPlantilla(object sender, EventArgs e)
{
    // Obtengo la plantilla que acaba de ser seleccionada
    PlantillaSeleccionadaEventArgs eventArgs = (PlantillaSeleccionadaEventArgs)e;

    // Muestro los campos de la plantilla
    this.ucListarCamposDePlantillasFrmBúsqueda.NombreDePlantilla = eventArgs.NombreDePlantilla;

    // Simulo click en botón Aceptar
    this.ClickEnBotónAceptar();
}

private void ucListarPlantillasFrmBúsqueda_PlantillaSeleccionada(object sender, EventArgs e)
{
    // Obtengo la plantilla que acaba de ser seleccionada
    PlantillaSeleccionadaEventArgs eventArgs = (PlantillaSeleccionadaEventArgs)e;

    // Muestro los campos de la plantilla
    this.ucListarCamposDePlantillasFrmBúsqueda.NombreDePlantilla = eventArgs.NombreDePlantilla;
}

#endregion // Eventos
```

En ambos casos, se luego de detectar una selección de plantilla en el lado derecho, se actualiza el lado izquierdo con las columnas correspondientes. Y, adicionalmente, cuando se efectúa doble click, además se simula un click en el botón "Aceptar", lo que selecciona la plantilla y la aplica en el *Formulario de Búsqueda*.

Lógica de detección de plantilla seleccionada en la apertura del Formulario de Búsqueda (base)

En la apertura de un Formulario de Búsqueda, inmediatamente luego de su apertura, se obtiene la plantilla seleccionada por defecto para el formulario, basado en el tipo de dato del *ViewModel* que lo representa.

Si existe una plantilla seleccionada en la base datos, se aplica mediante el método *AplicarConfiguracionAColumnas()*, desarrollado previamente, pero el cual se ejecutaba de forma automática.

Si aún no se ha seleccionado una plantilla por defecto para el formulario, entonces se pregunta al usuario si quiere seleccionar una, y de ser así, se ejecuta el método *SeleccionarPlantillaDeBusqueda()*, el cual en esencia abre el *Formulario modal Aceptar-Cancelar: Seleccionar plantilla*, y queda en espera de la plantilla que seleccione el usuario.

```
private void CargarPlantillaDeColumnasSeleccionada()
{
    // Recupero PlantillaSeleccionada por defecto de Db
    FrmBúsqueda_PlantillaSeleccionada plantillaSeleccionada =
CT_FrmBúsqueda_PlantillasSeleccionada.ObtenerPorNombreDeVM(tipoDeDatoDelVM.Name);

    FrmBúsqueda_NombreDePlantilla nombreDePlantillaSeleccionada = (plantillaSeleccionada == null) ?
null : plantillaSeleccionada.NombreDePlantilla;

    // Applico la nueva configuración de columnas (maneja también valores nulos)
    this.AplicarConfiguracionAColumnas(nombreDePlantillaSeleccionada);

    // Si tipo de dato aún no cuenta con una planitlla por defecto, muestro mensaje
    if (plantillaSeleccionada == null)
    {
        XtraMessageBox.SmartTextWrap = true;
        DialogResult dialogResult = XtraMessageBox.Show(
            resxOwn.GetString(nameof(FrmBúsquedaBaseResx.PreguntaDeseaSeleccionarUnaPlantilla)),
            this.Text,
            MessageBoxButtons.YesNo,
            MessageBoxIcon.Question,
            DevExpress.Utils.DefaultBoolean.True
            );

        if (dialogResult == DialogResult.Yes)
        {
            this.SeleccionarPlantillaDeBusqueda();
        }
    }
}
```

Lógica de detección de cambios en la plantilla seleccionada en el cierre del Formulario de Búsqueda (base)

En el evento de cierre de un *Formulario de Búsqueda*, se agregó una lógica que se basa en la respuesta del método *HayCambiosEnConfiguraciónDeColumnas()*, mediante el cual, si hay cambios se consulta al usuario si desea conservar los mismos.

Si el usuario opta por conservar los cambios, entonces se emplea el método *GuardarConfiguraciónDeColumnas()*.

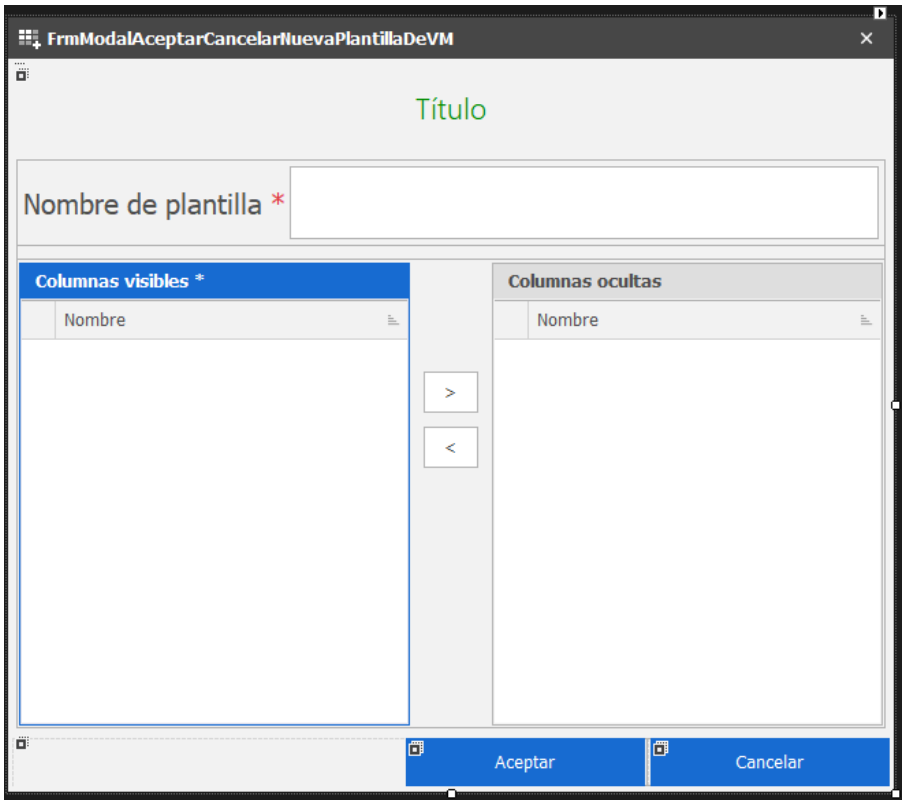
```
if (this.HayCambiosEnConfiguraciónDeColumnas())
```

```
{
    XtraMessageBox.SmartTextWrap = true;
    DialogResult dialogResult = XtraMessageBox.Show(
        resxOwn.GetString(nameof(FrmBúsquedaBaseResx.PreguntaDeseaGuardarCambiosEnPlantilla)),
        this.Text,
        MessageBoxButtons.YesNo,
        MessageBoxIcon.Question,
        DevExpress.Utils.DefaultBoolean.True
        );

    if (dialogResult == DialogResult.Yes)
    {
        if (this.GuardarConfiguraciónDeColumnas() == false)
        {
            XtraMessageBox.SmartTextWrap = true;
            XtraMessageBox.Show(
                resxDialog.GetString(nameof(Dialog.ErrorAlIntentarGuardar)),
                this.Text,
                MessageBoxButtons.OK,
                MessageBoxIcon.Error,
                DevExpress.Utils.DefaultBoolean.True
                );

            puedeCerrarFormulario = false;
        }
    }
}
```

User Story ID 13 - Formulario de búsqueda: Nueva plantilla

Formulario modal Aceptar-Cancelar: Nueva plantilla

Se desarrolló el *Formulario de búsqueda: Nueva plantilla*, acorde a las especificaciones funcionales, y el comportamiento definido en wireframes.

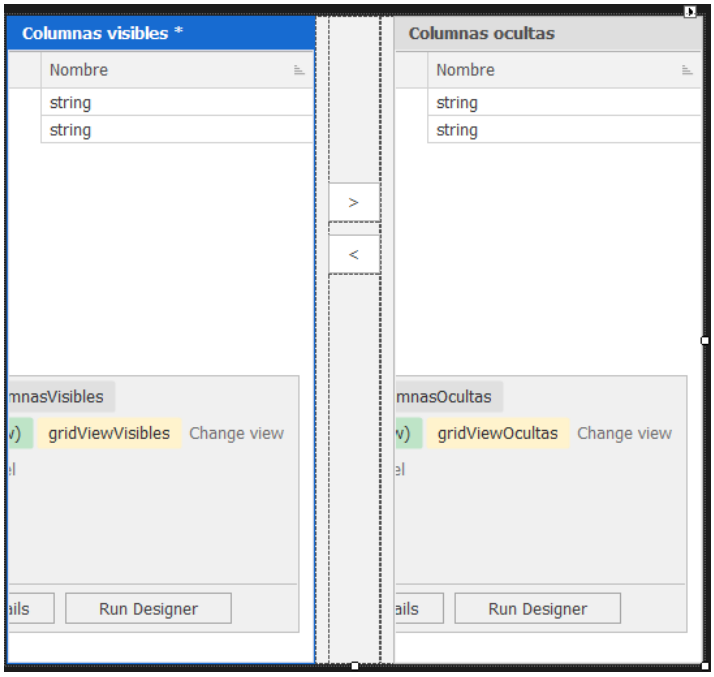


El formulario recibe por parámetro en su constructor, el tipo de datos de un *ViewModel* asociado a un *Formulario de Búsqueda*, para el cual se construirá una plantilla.

Inicialmente, en la columna de la izquierda se muestran todas las columnas disponibles (campos del *ViewModel*). Las columnas que permanezcan en esta área, serán las que luego formen parte de la plantilla.

Al momento de guardar la plantilla, se valida que ya no exista una plantilla con el mismo nombre, para el *ViewModel*, y que se haya seleccionado al menos una columna para mostrar.

Control de usuario: seleccionar campos visibles de ViewModel



Dado que la lógica mediante la cual se listan los campos de un *ViewModel* y se alterna su estado entre visibles y ocultos tiene cierta complejidad, y que la lógica de creación de una plantilla podría separarse con facilidad de la lógica de configuración de columnas, se desarrolló un control de usuario que encapsula la configuración de columnas, para favorecer la separación de responsabilidades.

Este control de usuario utiliza la misma estrategia que se desarrolló para *Control de usuario: Listar columnas de plantilla de Formulario de Búsqueda*, y permite localizar los campos de un ViewModel.

Lógica de selección de plantilla recientemente creada en el Formulario de Búsqueda (base)

Luego del cierre de un *Formulario modal Aceptar-Cancelar: Nueva plantilla*, el *Formulario de Búsqueda (base)* detecta si se ha generado una nueva plantilla, o se ha cerrado el formulario sin dar de alta una nueva plantilla.

En caso de haberse creado una nueva plantilla, invita a su selección.

```
private void SeleccionarPlantillaDeBusqueda()
{
    FrmModalAceptarCancelarSeleccionarPlantillaBusqueda frm = new
FrmModalAceptarCancelarSeleccionarPlantillaBusqueda(this.tipoDeDatoDelVM,
this.nombreDePlantillaSeleccionada);
    GestorDeFormularios.AbrirComoDialogoModal(frm, this);

    // Recupero respuesta del diálogo modal
    FrmBúsqueda_NombreDePlantilla nuevaPlantillaSeleccionada =
(FrmBúsqueda_NombreDePlantilla)GestorDeFormularios.RespuestaDeDialogoModal;

    // Si la selección fue null, entonces no se opta por no generar cambios
    if (nuevaPlantillaSeleccionada != null)
    {
```

```
        // Applico la nueva configuración de columnas (maneja también valores nulos)
        this.AplicarConfiguracionAColumnas(nuevaPlantillaSeleccionada);
    }
}
```

User Story ID 14 - Gestor de formularios abiertos y entidades de datos en uso

El *Gestor de Formularios* tiene el rol de arbitrar diferentes situaciones que pueden darse al emplear formularios de Búsqueda o de ABMC.

Por ejemplo, si ya se ha abierto un Formulario de Búsqueda para la entidad Mascota, y se intenta volver a abrir dicho formulario, nos interesa que la aplicación automáticamente nos posicione sobre el formulario abierto, en vez de crear un nuevo.

Para los casos de formularios ABMC, pueden darse una combinación de situaciones más amplia, dado que las situaciones de apertura en torno a una misma entidad pueden ser de tres tipos: Alta, Modificaciones y Consultas.

Un aspecto favorable de esta esta estrategia es que encapsula la lógica de instanciamiento y desechado de formularios de tipo conocido (Búsqueda y ABMC), dado que será *el Gestor de Formularios* quien efectúe esas operaciones, el desarrollador simplemente los solicitará.

El *Gestor de Formularios* es una clase estática, que se ubicó en la raíz de la solución. A continuación, se especificará el código resultante, y su rol de resolución en cada caso.

Gestión de formularios ABMC

Para tener presentes los formularios que se han abierto en la aplicación, se dispone una lista privada sólo para *Formularios ABMC*:

```
rivate static List<IFrmABMC> FormulariosABMCAbiertos = new List<IFrmABMC>();
```

En los *Estados internos* se definió que un *Formulario ABMC* puede ser utilizado en torno a tres diferentes verbos (Consultar, Crear, y Modificar). Cuando el usuario solicite la apertura de un formulario con un determinado verbo, se verificará si el formulario ya está abierto, y con cuál de esos verbos, de ese modo se actuará en consecuencia.

Solicitud de apertura en modo Consulta

Se procede con el siguiente criterio:

- Si ya hay un formulario del mismo tipo en verbo "Consultar", para la misma entidad, poner el foco en él.
- Si no, si hay un formulario del mismo tipo en verbo "Modificar", para la misma entidad, poner el foco en él notificando que no se puede "Consultar" cuando se está modificando.
- Si no, crear un nuevo formulario en verbo "Consultar" y abrirlo.

```
FrmABMCMDIBase frmAsociadoALaEntidad = (FrmABMCMDIBase)FormulariosABMCAbiertos.FirstOrDefault(f =>
f.GetType() == tipoDeDatoFrmABMC && f.IdEntidadPrincipal == (int)idDeEntidadPrincipal);

if (frmAsociadoALaEntidad != null) // Está en Consulta o Modificación
{
    switch (frmAsociadoALaEntidad.VerboActualDelFormulario)
    {
```

```
        case Verbo.Consultar:
            PonerFocoEnMDIChild(frmAsociadoALaEntidad);
            break;

        case Verbo.Modificar:
            // TODO: Refinar mensaje
            MessageBox.Show("El formulario está en modificación, no se puede consultar");
            PonerFocoEnMDIChild(frmAsociadoALaEntidad);
            break;
    }
}
else // No está en Consulta ni Modificación
{
    // Preparación de fomulario
    FrmABMCMDIBase frmVerboConsultar = (FrmABMCMDIBase)Activator.CreateInstance(tipoDeDatoFrmABMC);
    frmVerboConsultar.PrepararFormulario(Verbo.Consultar, entidad, parámetros);

    // Registro de formulario
    FormulariosABMCAbiertos.Add(frmVerboConsultar);

    // Apertura
    GestorDeFormularios.AbrirComoMDIChild(frmVerboConsultar);
}
```

Solicitud de apertura en modo Crear

Se procede con el siguiente criterio:

- Si ya hay un formulario del mismo tipo en verbo "Crear", para la misma entidad, poner el foco en él.
- Si no, crear un nuevo formulario en verbo "Crear" y abrirlo.

```
FrmABMCMDIBase frmVerboCrear = (FrmABMCMDIBase)FormulariosABMCAbiertos.FirstOrDefault(f =>
f.GetType() == tipoDeDatoFrmABMC && f.VerboActualDelFormulario == Verbo.Crear);

if (frmVerboCrear != null)
{
    PonerFocoEnMDIChild(frmVerboCrear);
}
else
{
    // Preparación de fomulario
    frmVerboCrear = (FrmABMCMDIBase)Activator.CreateInstance(tipoDeDatoFrmABMC);
    frmVerboCrear.PrepararFormulario(Verbo.Crear, entidad = null, parámetros);

    // Registro de formulario
    FormulariosABMCAbiertos.Add(frmVerboCrear);

    // Apertura
    GestorDeFormularios.AbrirComoMDIChild(frmVerboCrear);
}
```

Solicitud de apertura en modo Modificar

Se procede con el siguiente criterio:

- Si hay un formulario del mismo tipo en verbo "Consultar", pasarlo a verbo "Modificar" y poner el foco en él.
- Si ya hay un formulario del mismo tipo en verbo "Modificar", poner el foco en él.
- Si no, crear un nuevo formulario en verbo "Modificar" y abrirlo.

```
FrmABMCMDIBase frmAsociadoALaEntidad = (FrmABMCMDIBase)FormulariosABMCAbiertos.FirstOrDefault(f =>
f.GetType() == tipoDeDatoFrmABMC && f.IdEntidadPrincipal == (int)idDeEntidadPrincipal);

if (frmAsociadoALaEntidad != null) // Está en Consulta o Modificación
```

```
{
    switch (frmAsociadoALaEntidad.VerboActualDelFormulario)
    {
        case Verbo.Consultar:
            // Cerrar el formulario en verbo Consultar
            DesecharFrmABMC(frmAsociadoALaEntidad);

            // Preparación de fomulario
            FrmABMCMDIBase frmVerboModificar =
(FrmABMCMDIBase)Activator.CreateInstance(tipoDeDatoFrmABMC);
            frmVerboModificar.PrepararFormulario(Verbo.Modificar, entidad, parámetros);

            // Registro de formulario
            FormulariosABMCAbiertos.Add(frmVerboModificar);

            // Apertura
            GestorDeFormularios.AbrirComoMDIChild(frmVerboModificar);
            break;

        case Verbo.Modificar:
            PonerFocoEnMDIChild(frmAsociadoALaEntidad);
            break;
    }
}
else // No está en Consulta ni Modificación
{
    // Preparación de fomulario
    FrmABMCMDIBase frmVerboConsultar = (FrmABMCMDIBase)Activator.CreateInstance(tipoDeDatoFrmABMC);
    frmVerboConsultar.PrepararFormulario(Verbo.Modificar, entidad, parámetros);

    // Registro de formulario
    FormulariosABMCAbiertos.Add(frmVerboConsultar);

    // Apertura
    GestorDeFormularios.AbrirComoMDIChild(frmVerboConsultar);
}
```

Desechar formularios ABMC

Encuentra el formulario a ser desechado, y aplica *Dispose()* sobre el sin hacer ningún tipo de investigación respecto a su estado.

Este método debe ser invocado cuando finalmente se tiene certeza que el formulario debe ser eliminado.

Si el formulario pasado por parámetros no es encontrado, el método no toma ninguna acción en particular.

```
internal static void DesecharFrmABMC(XtraForm frmABMC)
{
    for (int i = 0; i < FormulariosABMCAbiertos.Count; i++)
    {
        if (Object.ReferenceEquals(FormulariosABMCAbiertos[i], frmABMC))
        {
            // Desecho el formulario
            FormulariosABMCAbiertos[i].Dispose();
            // Remuevo el Vinculo FormularioEntidad del listado
            FormulariosABMCAbiertos.RemoveAt(i);
            return;
        }
    }
}
```

Apertura de formularios de Búsqueda

Del mismo modo que con los formularios ABMC, se almacenan los *Formularios de Búsqueda* abiertos en una lista, para posteriormente verificar su existencia, o efectuar su desechado. Si el formulario ya existe, entonces se pone el foco en él, de lo contrario se crea.

```
FrmBúsquedaBase frmBusqueda = GestorDeFormularios.BuscarFrmBusquedaAbierto(tipoDeDatoFrmBusqueda);

if (frmBusqueda == null)
{
    // Preparación de fomulario
    frmBusqueda = (FrmBúsquedaBase)Activator.CreateInstance(tipoDeDatoFrmBusqueda);
    frmBusqueda.PrepararFormulario(tipoDeBusqueda);

    // Registro de formulario
    FormulariosDeBusquedaAbiertos.Add(frmBusqueda);

    // Apertura
    GestorDeFormularios.AbrirComoMDIChild(frmBusqueda);
}
else
{
    GestorDeFormularios.PonerFocoEnMDIChild(frmBusqueda);
}
```

Desechar formularios de Búsqueda

Encuentra el formulario a ser desechado, y aplica *Dispose()* sobre el sin hacer ningún tipo de investigación respecto a su estado.

Este método debe ser invocado cuando finalmente se tiene certeza que el formulario debe ser eliminado.

Si el formulario pasado por parámetros no es encontrado, el método no toma ninguna acción en particular.

```
internal static void DesecharFrmBusqueda(XtraForm frmBusqueda)
{
    for (int i = 0; i < FormulariosDeBusquedaAbiertos.Count; i++)
    {
        if (Object.ReferenceEquals(FormulariosDeBusquedaAbiertos[i], frmBusqueda))
        {
            // Desecho el formulario
            FormulariosDeBusquedaAbiertos[i].Dispose();
            // Remuevo el Vinculo FormularioEntidad del listado
            FormulariosDeBusquedaAbiertos.RemoveAt(i);
            return;
        }
    }
}
```

Uso del Gestor de Formularios

En el *Formulario Principal*, tras cada acceso a cada formulario deseado, el formulario principal era el responsable de instanciarlos, haciendo uso del *Gestor de Formulario*, cada invocación dentro de los eventos click de los botones, resulta de la siguiente manera:

```
private void aceBuscarPropietario_Click(object sender, EventArgs e)
{
    GestorDeFormularios.AbrirFrmBusqueda(typeof(FrmBúsquedaPropietario));
}

private void aceBuscarMascota_Click(object sender, EventArgs e)
{
    GestorDeFormularios.AbrirFrmBusqueda(typeof(FrmBúsquedaMascota));
```

```
}

private void aceNuevoPropietario_Click(object sender, EventArgs e)
{
    GestorDeFormularios.AbrirFrmABMC(typeof(FrmABMCMDIPropietario), Verbo.Crear);
}

private void aceNuevaMascota_Click(object sender, EventArgs e)
{
    GestorDeFormularios.AbrirFrmABMC(typeof(FrmABMCMDIMascota), Verbo.Crear);
}
```

Gestión de entidades en uso

La gestión de entidades en uso tiene como propósito registrar qué entidades de dato están siendo utilizadas, y en qué parte de la aplicación, en tiempo de ejecución.

Por ejemplo, cuando se abre un *Formulario de Búsqueda* para la entidad de datos *Mascota*, todas las instancias de la entidad de datos *Mascota* que son presentadas en la grilla como resultados de la búsqueda, están siendo utilizadas por dicho formulario en modo de "Consulta".

Si alguna de las mascotas es seleccionada para ser modificada, entonces la entidad de datos pasa a estar en uso por el *Formulario ABMC*, en modo "Escritura".

Para evitar que el uso de una misma entidad de datos, con distintos fines lleve a una situación de inconsistencias en tiempo de ejecución, es que se registran que entidades son utilizadas, de qué modo, y por quién.

Los tipos de permiso se registran por el siguiente enumerado, acorde a la descripción previa:

```
internal enum TipoDePermiso
{
    Lectura,
    Escritura
}
```

El tipo de entidad (referido por el tipo de datos), el tipo de permiso (Lectura o Escritura), y el formulario que la está utilizando, se registran con la siguiente clase:

```
internal class EntidadEnUso
{
    public EntidadEnUso(Type tipoDeEntidad, TipoDePermiso tipoDePermiso, int id, XtraForm
formulario)
    {
        TipoDeEntidad = tipoDeEntidad;
        TipoDePermiso = tipoDePermiso;
        Id = id;
        FormulariosYCantidadDeSolicitudes = new List<Formulario_CantidadDeSolicitudes>() { new
Formulario_CantidadDeSolicitudes(formulario) };
    }

    public Type TipoDeEntidad { get; set; }
    public TipoDePermiso TipoDePermiso { get; set; }
    public int Id { get; set; }
    public List<Formulario_CantidadDeSolicitudes> FormulariosYCantidadDeSolicitudes { get; set; }
}
```

En la clase se dispone un método que permite registra las entidades en uso desde cualquier parte de la aplicación:

```
internal static bool RegistrarEntidad(Type tipoDeEntidad, TipoDePermiso tipoDePermiso, int id,
XtraForm formulario)
{
    EntidadEnUso entidadEnUso = BuscarEntidad(tipoDeEntidad, id);

    if (entidadEnUso != null)
    {
        if (entidadEnUso.TipoDePermiso == TipoDePermiso.Escritura)
        {
            return false;
        }
        else
        {
            // Verifico si el formulario ya está asociado a la Entidad
            Formulario_CantidadDeSolicitudes formulario_CantidadDeSolicitudes =
entidadEnUso.FormulariosYCantidadDeSolicitudes.Where(fc => object.ReferenceEquals(fc.Formulario,
formulario)).FirstOrDefault();

            if (formulario_CantidadDeSolicitudes != null)
            {
                // Incremento la cantidad de solicitudes del formulario, para la Entidad utilizada
                formulario_CantidadDeSolicitudes.CantidadDeSolicitudes =
formulario_CantidadDeSolicitudes.CantidadDeSolicitudes + 1;
            }
            else
            {
                // Asocio el formulario a la entidad
                entidadEnUso.FormulariosYCantidadDeSolicitudes.Add(new
Formulario_CantidadDeSolicitudes(formulario));
            }

            return true;
        }
    }
    else
    {
        // Registro nueva Entidad en Uso
        entidadesEnUso.Add(new EntidadEnUso(tipoDeEntidad, tipoDePermiso, id, formulario));
        return true;
    }
}
```

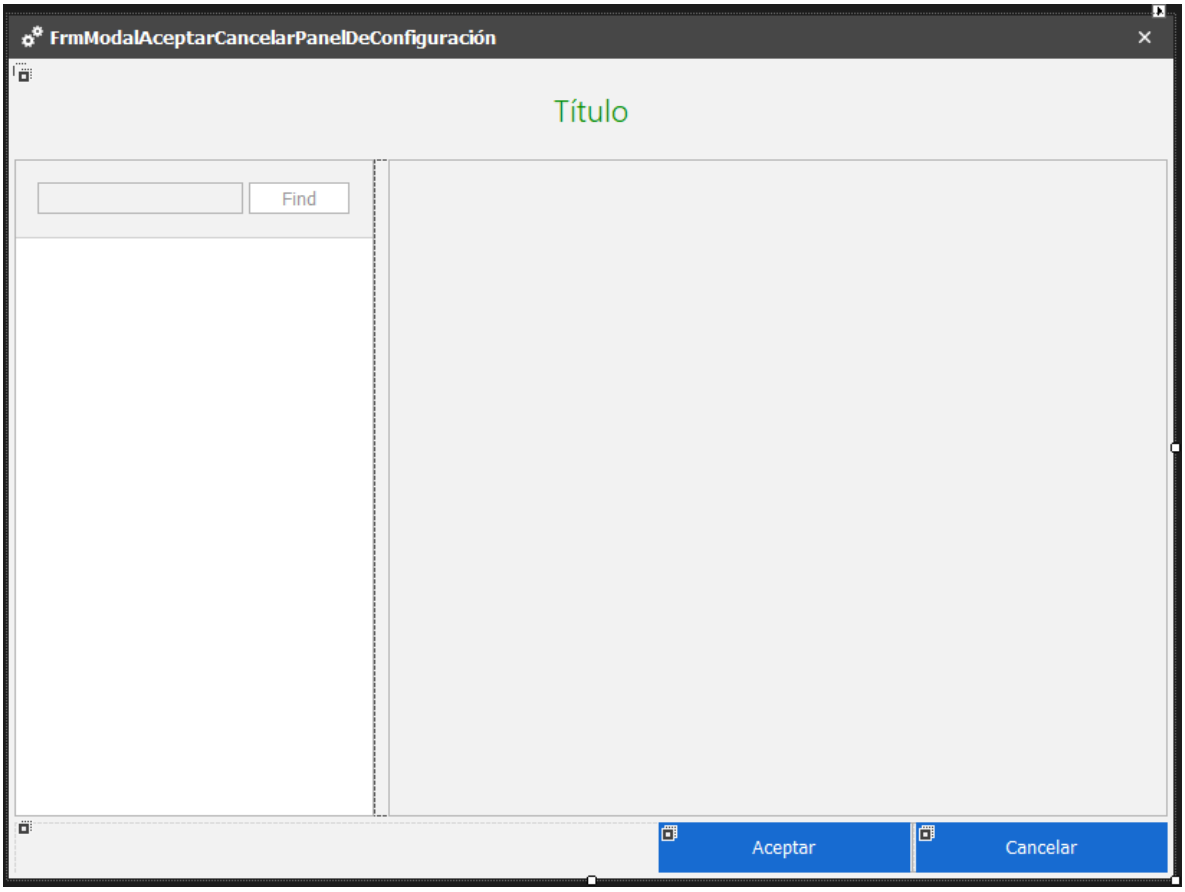
Como así también liberar recursos:

```
internal static void LiberarEntidad(Type tipoDeEntidad, int id, XtraForm formulario)
{
    EntidadEnUso entidadEnUso = BuscarEntidad(tipoDeEntidad, id);

    if (entidadEnUso != null)
    {
        // Verifico si el formulario ya está asociado a la Entidad
        Formulario_CantidadDeSolicitudes formulario_CantidadDeSolicitudes =
entidadEnUso.FormulariosYCantidadDeSolicitudes.Where(fc => object.ReferenceEquals(fc.Formulario,
formulario)).FirstOrDefault();

        if (formulario_CantidadDeSolicitudes != null)
        {
            // Decremento la cantidad de solicitudes del formulario, para la Entidad utilizada
            formulario_CantidadDeSolicitudes.CantidadDeSolicitudes =
formulario_CantidadDeSolicitudes.CantidadDeSolicitudes - 1;

            if (formulario_CantidadDeSolicitudes.CantidadDeSolicitudes == 0)
            {

entidadEnUso.FormulariosYCantidadDeSolicitudes.Remove(formulario_CantidadDeSolicitudes);
            }

            if (entidadEnUso.FormulariosYCantidadDeSolicitudes.Count == 0)
            {
```

```
                entidadesEnUso.Remove(entidadEnUso);
            }
        }
    }
}
```

User Story ID 15 - Formulario de configuración

Se desarrolló el *Formulario de configuración*, acorde a las especificaciones funcionales, y el comportamiento definido en wireframes.



Agregar ítems de configuración al Panel

Internamente el formulario maneja una lista de *ítems de configuración*, a la cual deben agregarse todos los ítems que faciliten configuraciones.

El listado puede ser accedido desde el constructor de la clase, y permite que los elementos tengan un identificador (que debe ser único), y que además especifiquen el identificador de un padre, en el caso de ser necesaria una representación de tipo árbol.

```
items.Add(new ItemConfiguracion(0, -1, "Grupo 1", typeof(ControlDeUsuario1)));
items.Add(new ItemConfiguracion(1, -1, "Grupo 2", typeof(ControlDeUsuario2)));
items.Add(new ItemConfiguracion(2, 0, "Hijo Grupo 1", typeof(ControlDeUsuario3)));
items.Add(new ItemConfiguracion(3, 2, "Nieto Grupo 1", typeof(ControlDeUsuario4)));
items.Add(new ItemConfiguracion(4, -1, "Grupo 3", typeof(ControlDeUsuario5)));
```

Definición de un Ítem de Configuración

Como se mostró previamente, la forma de agregar un control de usuario al Panel de Configuración, es mediante la definición de un *ItemConfiguracion*. Esta definición es transparente al desarrollador, dado que el mismo simplemente debe respetar su definición.

La clase *ItemConfiguración* se define mediante cuatro elementos externos (que son los vistos previamente en la carga de elementos al listado): un identificador, el identificador de un padre, una descripción (que será mostrada en el panel de la derecha), y un control de usuario que implemente la interfaz *UserControlDeConfiguracion*.

Luego, internamente la clase almacena información del control de usuario de configuración referido, para saber si el mismo ha sido instanciado o no. La información de instanciamiento es útil al momento de aplicar las configuraciones, dado que, de todos los ítems de configuración del panel, sólo se ejecutan las acciones de aquellos controles que han sido utilizados.

```
public class ItemConfiguracion
{
    public ItemConfiguracion(int Id, int PadreId, string Descripción, Type TipoDelUserControl)
    {
        this.Id = Id;
        this.PadreId = PadreId;
        this.Descripción = Descripción;
        this.TipoDelUserControl = TipoDelUserControl;

        this.UserControlInstanciado = false;
    }

    public int Id { get; set; }

    public int PadreId { get; set; }

    public string Descripción { get; set; }

    public bool UserControlInstanciado { get; set; }

    public Type TipoDelUserControl { get; set; }

    private UserControlDeConfiguracion intanciaDeUserControl;
    public UserControlDeConfiguracion InstanciaDelUserControl
    {
        // Los usercontrol se instanciarán sólo cuando sean solicitados
        get
        {
            if (this.UserControlInstanciado == false)
            {
                this.intanciaDeUserControl =
(UserControlDeConfiguracion)Activator.CreateInstance(TipoDelUserControl);
                this.UserControlInstanciado = true;
            }

            return intanciaDeUserControl;
        }
    }
}
```

Controles de usuario para configuración

Los controles de usuario para el panel de configuraciones, deben heredar de una interfaz sencilla, que define cual será la acción en caso de presionar el botón Aceptar, o Cancelar en el panel.

Muchas configuraciones pueden ser efectuadas dentro del panel de configuraciones, en diferentes controles de usuario, pero las mismas no serán aplicadas hasta que no se presione el botón Aceptar, es decir, se ejecutan todas las acciones de configuración de forma secuencial.

```
public interface UserControlDeConfiguracion
{
    /// <summary>
    /// Ejecuta la lógica necesaria al momento de presionar botón Aceptar;
    /// inmediatamente después, se invoca la lógica de cierre de forma
    /// automática.
    /// </summary>
    void Aceptar();

    /// <summary>
    /// Ejecuta la lógica necesaria al momento de presionar botón Cancelar;
    /// inmediatamente después, se invoca la lógica de cierre de forma
    /// automática.
    /// </summary>
    void Cancelar();
}
```

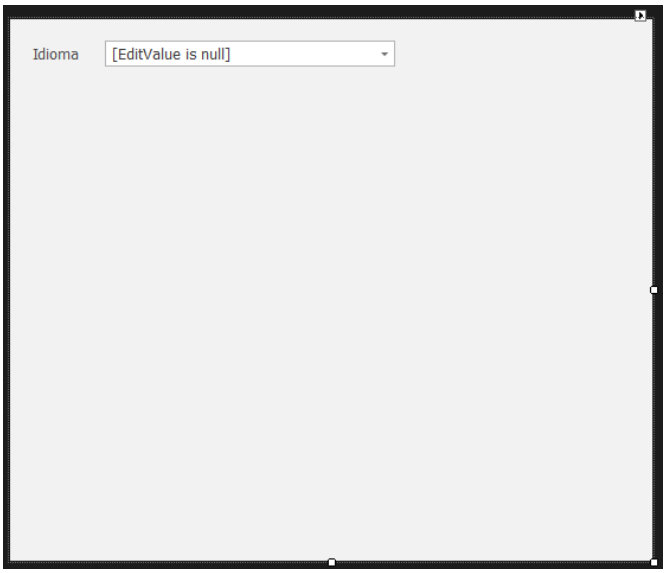
Ejecución del botón Aceptar o Cancelar

En ambos casos, al presionar el botón Aceptar o Cancelar, se recorren los controles de usuario que han sido instanciados, y se ejecuta los métodos Aceptar() o Cancelar() según corresponda, lo cual es posible dado que implementan la interfaz *UserControlDeConfiguracion*.

```
/// <summary>
/// Ejecuta la lógica necesaria al momento de presionar botón Aceptar;
/// inmediatamente después, se invoca la lógica de cierre de forma
/// automática.
/// </summary>
protected override bool Aceptar()
{
    bool puedeCerrarFormulario = true;

    List<UserControlDeConfiguracion> userControlInstanciados = this.items?.Where(i =>
i.UserControlInstanciado == true).Select(i => i.InstanciaDelUserControl).ToList();
    foreach (UserControlDeConfiguracion userControlDeConfiguracion in userControlInstanciados)
    {
        userControlDeConfiguracion.Aceptar();
    }

    return puedeCerrarFormulario;
}

/// <summary>
/// Ejecuta la lógica necesaria al momento de presionar botón Cancelar;
/// inmediatamente después, se invoca la lógica de cierre de forma
/// automática.
/// </summary>
protected override bool Cancelar()
{
    bool puedeCerrarFormulario = true;

    List<UserControlDeConfiguracion> userControlInstanciados = this.items?.Where(i =>
i.UserControlInstanciado == true).Select(i => i.InstanciaDelUserControl).ToList();
    foreach (UserControlDeConfiguracion userControlDeConfiguracion in userControlInstanciados)
    {
        userControlDeConfiguracion.Cancelar();
    }

    return puedeCerrarFormulario;
}
```

User Story ID 16 - Formulario de configuración – "Idioma"

Se desarrolló el control de usuario para la configuración de idioma, acorde a las especificaciones funcionales, y el comportamiento definido en wireframes.



La lógica del control de usuario busca en los parámetros de la aplicación (explicado a continuación) si el usuario ha seleccionado un idioma, y lo selecciona automáticamente en el listado. Si el usuario no ha seleccionado ningún idioma, entonces se muestra el idioma por defecto en la aplicación.

Los cambios de idioma se reflejan luego del reinicio de la aplicación.

Entidad de datos para almacenar parámetros

Se desarrolló una entidad de datos para almacenar parámetros de índole general en la aplicación. Sólo habrá una instancia de dicha entidad de datos.

Como primeros parámetros de la entidad, se registró el idioma por defecto en la aplicación (configurado por los desarrolladores), y el idioma seleccionado por el usuario.

```
public class ParámetrosInternos
{
    public int Id { get; set; }

    public int? CultureSeleccionadoPorElUsuarioId { get; set; }
    /// </summary>
    /// Almacena el CultureInfo seleccionado manualmente por el usuario desde UI.
    /// </summary>
    public virtual CultureDisponible CultureSeleccionadoPorElUsuario { get; set; }

    public int CulturePorDefectoId { get; set; }
    /// <summary>
    /// Almacena el CultureInfo que el sistema asumirá por defecto, en caso
    /// que el usuario no haya seleccionado uno, y que el idioma del sistema
    /// operativo no esté disponible en la aplicación.
    /// </summary>
    public virtual CultureDisponible CulturePorDefecto { get; set; }
}
```

Entidad de datos para almacenar idiomas disponibles en la aplicación

Los idiomas disponibles en la aplicación se registraron mediante la siguiente entidad de datos localizable:

```
public class CultureDisponible
{
    public int Id { get; set; }

    public string CultureName { get; set; }

    [Required]
    /// <summary>
    /// ! Localizable
    /// </summary>
    public string Descripción { get; set; }
}
```

Y se agregaron a las semillas los dos idiomas que estarán en uso actualmente en la aplicación (Inglés y Español:

```
#region CultureInfoDisponible

List<CultureDisponible> cultureInfoDisponible = new List<CultureDisponible>();

cultureInfoDisponible.Add(new CultureDisponible() { Id = 1, CultureName = "es-AR", Descripción =
"es-AR:Español - Argentina|en-US:Spanish - Argentina" });
cultureInfoDisponible.Add(new CultureDisponible() { Id = 2, CultureName = "en-US", Descripción =
"es-AR:Inglés - Estados Unidos|en-US:English - United States" });

context.CultureDisponibles.AddOrUpdate(
    r => r.Id,
    cultureInfoDisponible.ToArray()
);

#endregion // CultureInfoDisponible
```

Rutina al inicio para la configuración de idioma

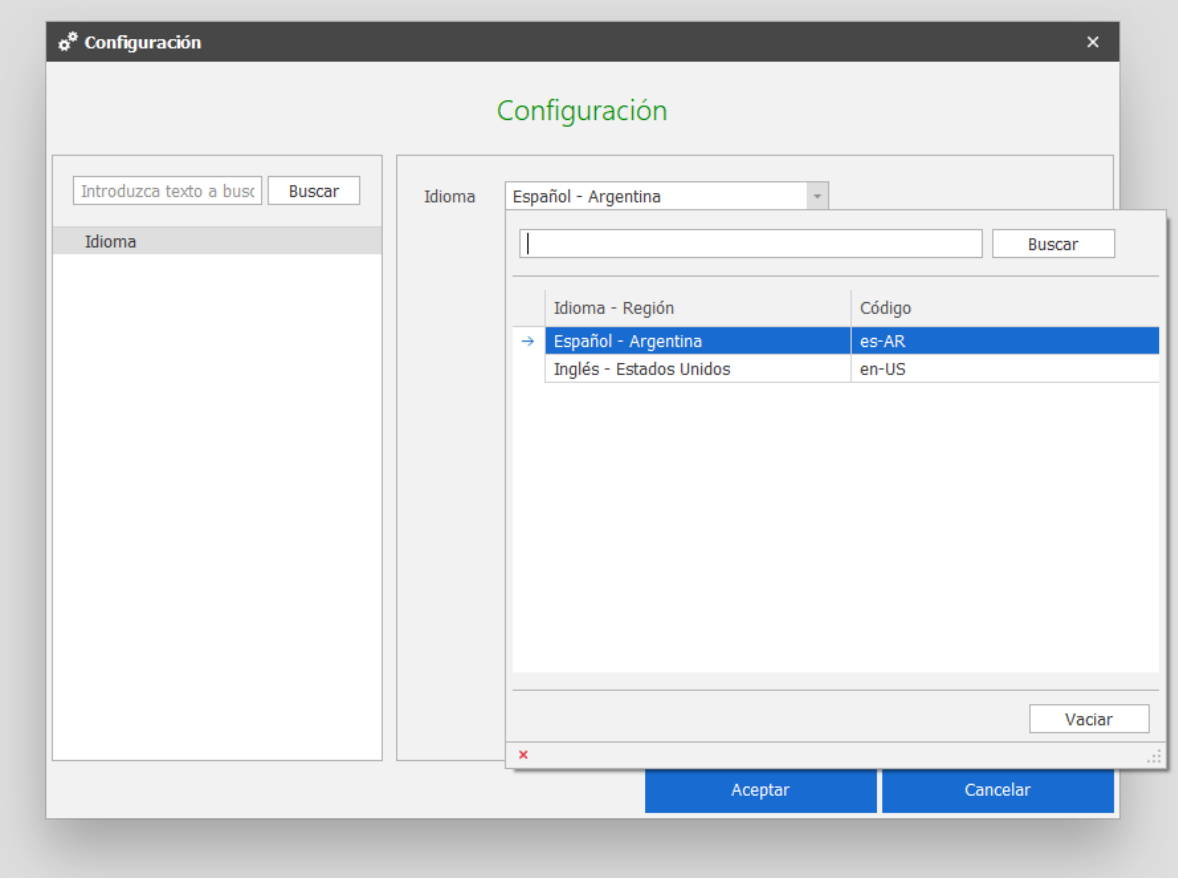
Se añadió tarea al inicio para establecer la configuración de idioma, que funciona con el siguiente criterio:

- Si hay idioma seleccionado por el usuario, lo aplica.
- Si no, si el idioma del sistema operativo de huésped está disponible en la aplicación, lo aplica.
- Si no, aplica el idioma por defecto en la aplicación según facto.

```
// Obtengo variables necesarias del RepositorioDeParámetro
CultureDisponible cullture SeleccionadoPorElUsuario =
RepositorioDeParametros.ParámetrosInternos.CultureSeleccionadoPorElUsuario;
CultureDisponible cullturePorDefecto = RepositorioDeParametros.ParámetrosInternos.CulturePorDefecto;
List<CultureDisponible> cultureDisponibles = RepositorioDeParametros.CultureDisponibles;

if (cullture SeleccionadoPorElUsuario != null)
{
    // Si hay un CultureInfo seleccionado explícitamente por el usuario, lo seteamos
    Thread.CurrentThread.CurrentCulture =
CultureInfo.GetCultureInfo(cullture SeleccionadoPorElUsuario.CultureName);
    Thread.CurrentThread.CurrentUICulture =
CultureInfo.GetCultureInfo(cullture SeleccionadoPorElUsuario.CultureName);
}
else
```

```
{
    // Recupero el Culture del sistema operativo
    CultureInfo osCulure = CultureInfo.InstalledUICulture;

    // Busco si el Culture del OS está en los disponibles por la aplicación
    CultureDisponible cultureDisponible = cultureDisponibles.FirstOrDefault(c => c.CultureName ==
osCulure.Name);

    if (cultureDisponible == null)
    {
        // Si el Culture del OS no está presente en la aplicación, establezco el por defecto de
facto
        Thread.CurrentThread.CurrentCulture =
CultureInfo.GetCultureInfo(cullturePorDefecto.CultureName);
        Thread.CurrentThread.CurrentUICulture =
CultureInfo.GetCultureInfo(cullturePorDefecto.CultureName);
    }
    else
    {
        // Si el Culture del OS si está presente en la aplicación, lo establezco
        Thread.CurrentThread.CurrentCulture =
CultureInfo.GetCultureInfo(cultureDisponible.CultureName);
        Thread.CurrentThread.CurrentUICulture =
CultureInfo.GetCultureInfo(cultureDisponible.CultureName);
    }
}
```

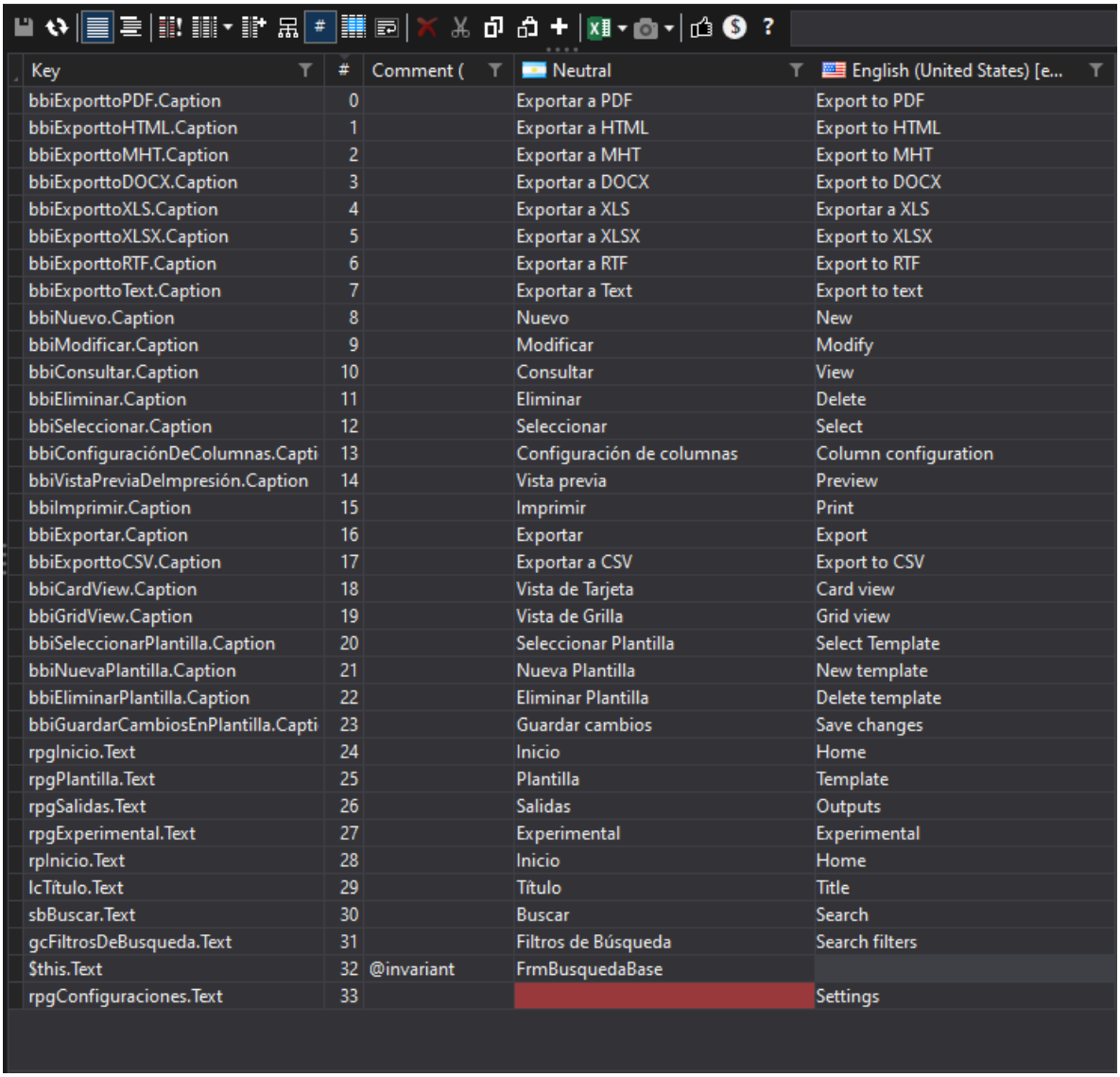
Resultados en tiempo de ejecución



User Story ID 17 - Traducción de la aplicación completa al inglés

Para efectuar la traducción de archivos de recurso, se implementó el plugin *ResX Resource Manager*[67], el cual permite efectuar traducciones con los traductores más populares, configurando las llaves de las APIs correspondientes.

Como ejemplo, se presenta la traducción automática del archivo de recursos del *Formulario de Búsqueda (base)*:



Key	#	Comment (	Neutral	English (United States) [e...
bbiExporttoPDF.Caption	0		Exportar a PDF	Export to PDF
bbiExporttoHTML.Caption	1		Exportar a HTML	Export to HTML
bbiExporttoMHT.Caption	2		Exportar a MHT	Export to MHT
bbiExporttoDOCX.Caption	3		Exportar a DOCX	Export to DOCX
bbiExporttoXLS.Caption	4		Exportar a XLS	Exportar a XLS
bbiExporttoXLSX.Caption	5		Exportar a XLSX	Export to XLSX
bbiExporttoRTF.Caption	6		Exportar a RTF	Export to RTF
bbiExporttoText.Caption	7		Exportar a Text	Export to text
bbiNuevo.Caption	8		Nuevo	New
bbiModificar.Caption	9		Modificar	Modify
bbiConsultar.Caption	10		Consultar	View
bbiEliminar.Caption	11		Eliminar	Delete
bbiSeleccionar.Caption	12		Seleccionar	Select
bbiConfiguraciónDeColumnas.Capti	13		Configuración de columnas	Column configuration
bbiVistaPreviaDelmpresión.Caption	14		Vista previa	Preview
bbiImprimir.Caption	15		Imprimir	Print
bbiExportar.Caption	16		Exportar	Export
bbiExporttoCSV.Caption	17		Exportar a CSV	Export to CSV
bbiCardView.Caption	18		Vista de Tarjeta	Card view
bbiGridView.Caption	19		Vista de Grilla	Grid view
bbiSeleccionarPlantilla.Caption	20		Seleccionar Plantilla	Select Template
bbiNuevaPlantilla.Caption	21		Nueva Plantilla	New template
bbiEliminarPlantilla.Caption	22		Eliminar Plantilla	Delete template
bbiGuardarCambiosEnPlantilla.Capti	23		Guardar cambios	Save changes
rpgInicio.Text	24		Inicio	Home
rpgPlantilla.Text	25		Plantilla	Template
rpgSalidas.Text	26		Salidas	Outputs
rpgExperimental.Text	27		Experimental	Experimental
rpInicio.Text	28		Inicio	Home
lcTítulo.Text	29		Título	Title
sbBuscar.Text	30		Buscar	Search
gcFiltrosDeBusqueda.Text	31		Filtros de Búsqueda	Search filters
$this.Text	32	@invariant	FrmBusquedaBase	
rpgConfiguraciones.Text	33			Settings

Vista de los idiomas soportados por el sistema:

[67] Repositorio del proyecto ResX Resource Manager: https://github.com/dotnet/ResXResourceManager

User Story ID 18 - Formulario de configuración – "Apariencia"

Se desarrolló el control de usuario para la configuración de idioma, acorde a las especificaciones funcionales, y el comportamiento definido en wireframes.

La lógica del control de usuario recupera la configuración aplicada, para seleccionarlo en listado automáticamente.

Los cambios se visualizan inmediatamente luego de aplicarlos, pero si se cancela la operación, se vuelve al estado previo.

Entidad de datos para almacenar parámetros

Se actualiza la entidad de datos *ParámetrosInternos* para almacenar el estilo y paleta seleccionado por el usuario, y el por defecto en la aplicación.

```
public class ParámetrosInternos
{
    public int Id { get; set; }

    public int? CultureSeleccionadoPorElUsuarioId { get; set; }
    /// <summary>
    /// Almacena el CultureInfo seleccionado manualmente por el usuario desde UI.
    /// </summary>
    public virtual CultureDisponible CultureSeleccionadoPorElUsuario { get; set; }

    public int CulturePorDefectoId { get; set; }
    /// <summary>
    /// Almacena el CultureInfo que el sistema asumirá por defecto, en caso
    /// que el usuario no haya seleccionado uno, y que el idioma del sistema
    /// operativo no esté disponible en la aplicación.
    /// </summary>
    public virtual CultureDisponible CulturePorDefecto { get; set; }

    /// <summary>
    /// Almacena el nombre del skin que el sistema asumirá por defecto, en caso
    /// que el usuario no haya seleccionado uno.
    /// </summary>
    public string EstiloPorDefecto { get; set; }
    /// <summary>
    /// Almacena el nombre de la paleta de colores que el sistema asumirá por
    /// defecto, en caso que el usuario no haya seleccionado uno.
    /// </summary>
    public string PaletaPorDefecto { get; set; }

    /// <summary>
    /// Almacena el nombre del skin seleccionado manualmente por el usuario desde UI.
    /// </summary>
    public string EstiloSeleccionadoPorElUsuario { get; set; }
    /// <summary>
    /// Almacena el nombre de la paleta de colores seleccionada manualmente por
    /// el usuario desde UI.
    /// </summary>
    public string PaletaSeleccionadaPorElUsuario { get; set; }
}
```

Rutina al inicio para la configuración de idioma

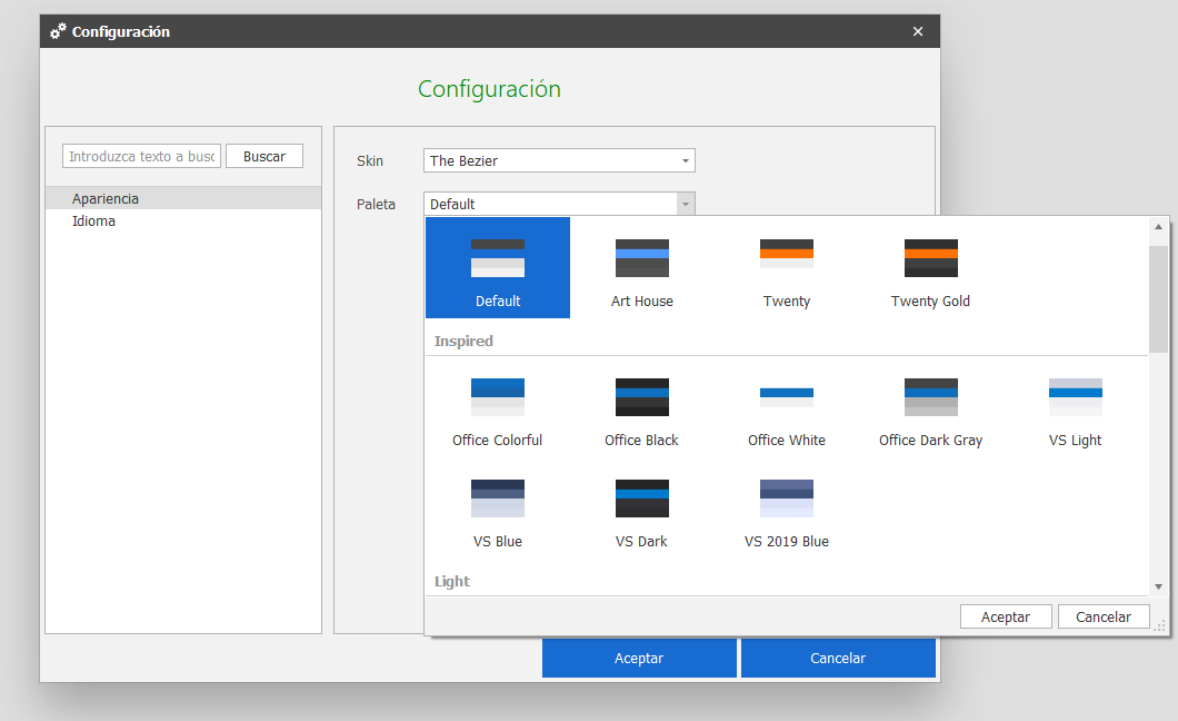
Se añadió tarea al inicio para establecer la configuración gráfica, que funciona con el siguiente criterio:

- Si hay una configuración realizada por el usuario, la aplica.
- Si no, aplica la configuración por defecto según facto.

```
// Obtengo variables necesarias del RepositorioDeParámetro
```

```
string estiloSeleccionadoPorElUsuario =
RepositorioDeParametros.ParámetrosInternos.EstiloSeleccionadoPorElUsuario;
string paletaSeleccionadaPorElUsuario =
RepositorioDeParametros.ParámetrosInternos.PaletaSeleccionadaPorElUsuario;
string estiloPorDefecto = RepositorioDeParametros.ParámetrosInternos.EstiloPorDefecto;
string paletaPorDefecto = RepositorioDeParametros.ParámetrosInternos.PaletaPorDefecto;

if (string.IsNullOrEmpty(estiloSeleccionadoPorElUsuario) == false)
{
    // Si hay un Estilo seleccionado explícitamente por el usuario, lo seteamos
    UserLookAndFeel.Default.SetSkinStyle(estiloSeleccionadoPorElUsuario,
paletaSeleccionadaPorElUsuario);
}
else
{
    // Si no, seteamos el estilo por defecto de facto
    UserLookAndFeel.Default.SetSkinStyle(estiloPorDefecto, paletaPorDefecto);
}
```

Resultados en tiempo de ejecución



12 Bibliografía

1. **Sommerville, Ian.** *Ingeniería del Software.* 9na. : Pearson. ISBN-13: 9780137035151.

2. **Pressman, Roger S.** *Ingeniería de Software.* 7ma. : Mcgraw-hill. ISBN-13: 9786071503145.

3. **SCRUMstudy.** *A guide to the SCRUM Body of knowledge (SBOK Guide)*. 3ra. : SCRUMstudy. ISBN: 978-0-9899252-0-4.

4. **Sutherland, Jeff.** *Scrum handbook.*

5. **UXPin.** *The guide to wireframing for Designers, PMs, Engineers and Anyone who touches product.*

6. **Greg Brougham.** *The Cynefin mini-book, an introduction to Complexity and the Cynefin Framework.*

7. **Bracalenti, Claudio** *Gestión de riesgos en los proyectos de software.*

8. **Software Engineering Institute (SEI).** *Taxonomy-Based Risk Identification.*

9. **Project Management Institute (PMI).** *A guide to the project management body of knowledge (PMBOK guide).* 7ma. : Project Management Institute. ISBN: 978-62825-184-5.

10. **The Westfall Team.** *Software risk management.*

11. **Boehm, Barry W.** *Software Risk Management: Principles and Practices.* : Defense Advanced Research Projects Agency.