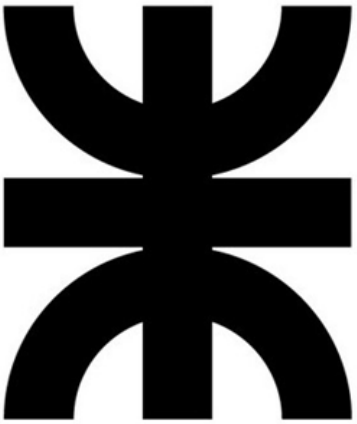
UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL SANTA FE



Informe de Proyecto Final de Carrera

Diseño e implementación de un sistema de gestión para consignación de haciendas para Ganados Remates S.A.

Carrera: Ingeniería en Sistemas de Información

Director del proyecto: Dr. Ing. Sarli, Juan Leonardo

Alumnos:

- Perussini, Martin - LU: 23815 - martin97_peru@hotmail.com
- Ravelli, Tomás - LU: 23847 - tomy_ravelli@hotmail.com

2022

Tabla de contenidos:

1 Introducción

El presente Proyecto Final de Carrera se enfocó en el diseño e implementación de un sistema de información que permita la gestión digital del proceso de generación, administración e informe de resultados de remates de hacienda, así como la impresión de los documentos generados en dicho proceso, para la firma Ganados Remates S.A.

El propósito de este Proyecto Final de Carrera es construir una solución que abarque por completo el proceso de realización de un remate, desde la gestión de información de clientes y categorías de haciendas, hasta la creación de remates y lotes de venta para los mismos. Además, hemos incluido la propia ejecución del proceso de venta durante la realización del remate, generando en el proceso los documentos requeridos, como son las boletas de compra/venta y, posteriormente, los informes de resultados, los cuales hasta el momento se realizan de forma manual.

1.1 El cliente

Ganados Remates S.A. es una firma que se dedica a la consignación de haciendas desde el año 1959 mediante la realización de remates feria y televisados a lo largo de la provincia de Santa Fe, así como ventas particulares y en el mercado de Rosario. Una persona que trabaja como consignatario de la misma se comunicó con nosotros con la idea de mejorar su actual proceso de negocio. Luego de este primer contacto, llevamos adelante una serie de entrevistas con la misma para que nos describa cómo era el proceso actual utilizado para la realización de un remate, con el objetivo de que nos ayude a entender y comprender el dominio del problema, y así ofrecer una solución acorde a sus necesidades.

1.2 Problema y fundamentación

1.2.1 Proceso de gestión actual

A partir de la consulta generada, nos enfocamos en investigar y entrevistar con mayor profundidad al consignatario de la empresa Ganados Remates S.A., con el objetivo de obtener un panorama general de cómo se llevan a cabo los remates de haciendas. Este conjunto inicial de entrevistas dieron como resultado el siguiente diagrama, en el cual mostramos cómo es la ejecución de las actividades más relevantes para la realización de un remate.

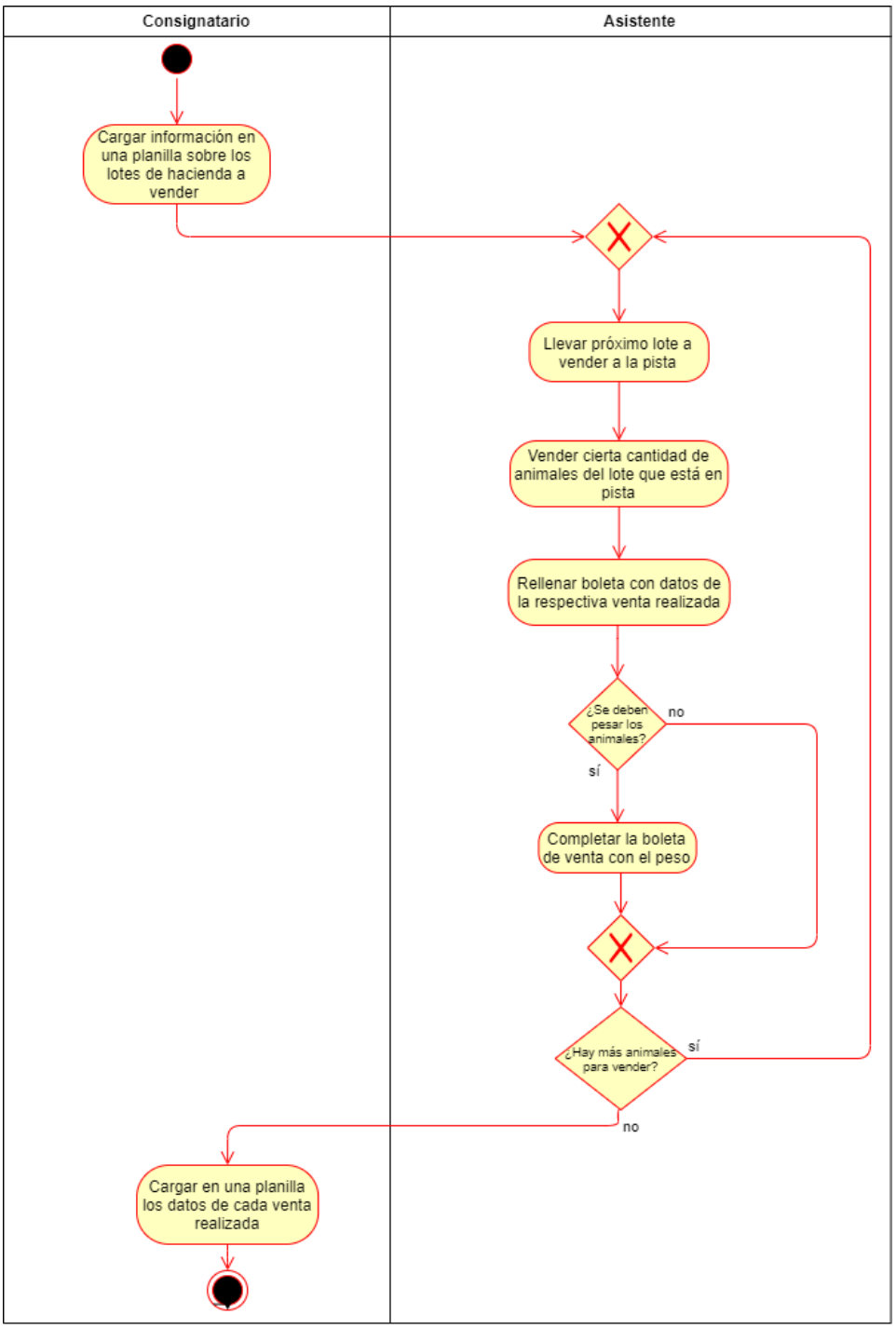


Figura 1: Proceso actual de un remate de hacienda

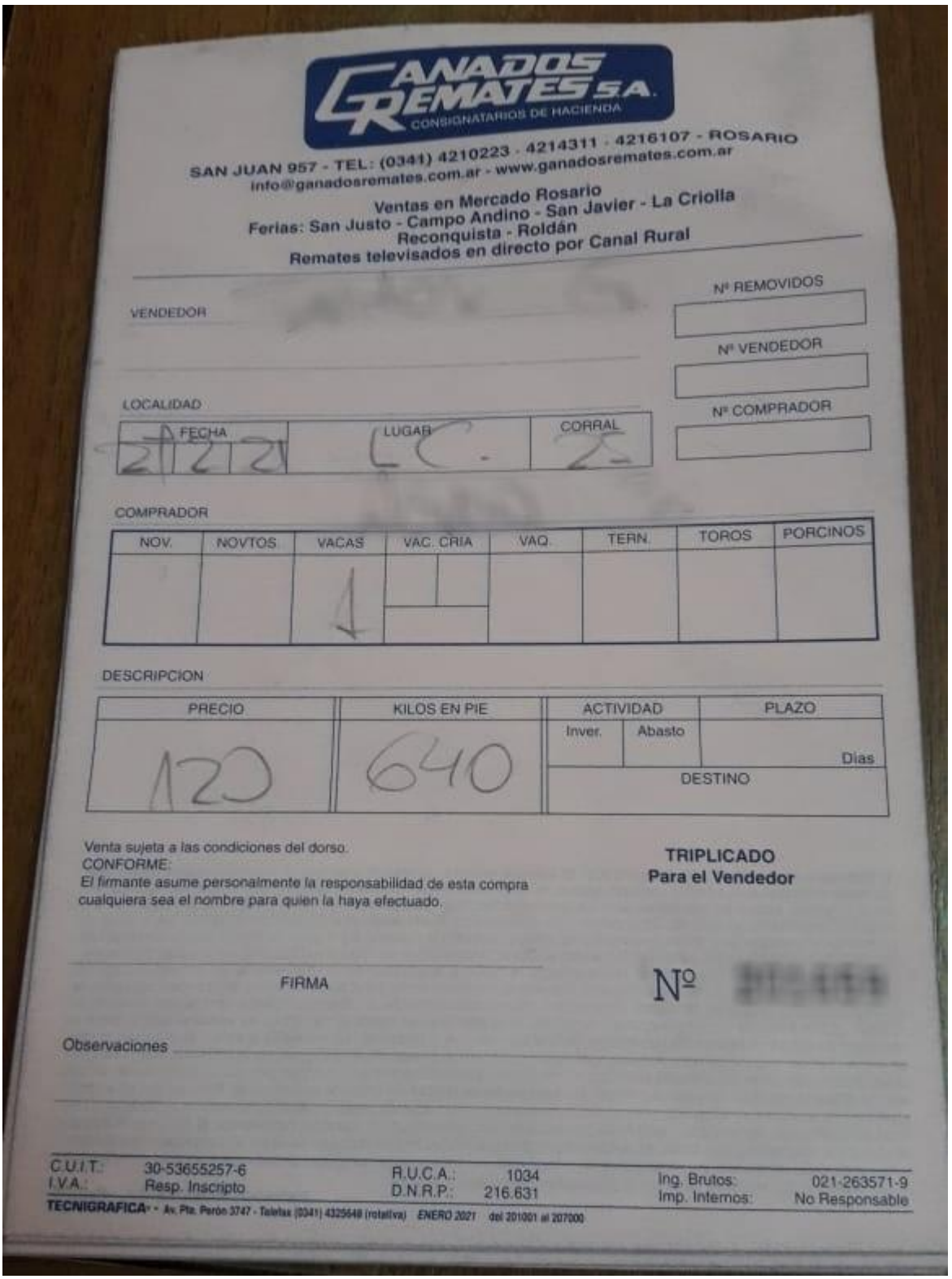
Como podemos apreciar, actualmente el trabajo antes, durante y después de un remate de haciendas se lleva a cabo de forma completamente manual y en papel. Esto incluye:

- "*Cargar información en una planilla sobre los lotes de hacienda a vender*": el consignatario, previo a la realización de un remate, carga en una planilla de papel (y de forma manual) los datos sobre los lotes de animales a vender a medida que los

clientes se comunican con el mismo. También carga la asignación de un corral para cada lote.

- "*Llevar próximo lote a vender a la pista*": una vez que comienza el remate, se lleva el próximo lote de animales que se va a vender a la pista, según el orden preestablecido, donde los compradores tienen la posibilidad de observar y comprar dicho lote.
- "*Vender cierta cantidad de animales del lote que está en pista*": un comprador decide comprar una determinada cantidad (o el total) de los animales que están en pista.
- "*Rellenar boleta con datos de la respectiva venta realizada*": la confección de las boletas de ventas durante el remate se realiza manualmente mediante el uso de un talonario especial, que es rellenado por un asistente (o responsable), donde por cada venta se generan cuatro copias de la boleta (el asistente completa los datos sólo en la primera "página" del talonario y, mediante el uso del papel especial para ese propósito con el que está hecho el talonario, la información es copiada a las 3 siguientes), donde una se entrega al vendedor, una al comprador, una para Senasa y la restante para control de la empresa. El encargado de completarlas, que suele estar al lado del martillero, introduce los datos de vendedor, fecha y corral y, una vez que se completa la venta, carga el comprador, cantidad de animales (en la categoría que corresponde) y precio final (observar Figura 2).
- "*Completar la boleta de venta con el peso*": en caso de que el lote vendido deba ser pesado, los animales pasan a la balanza y quien completó la boleta se la entrega a un segundo asistente, que es responsable de llevarla hasta el sector de la balanza, donde un tercero le carga el peso. Luego, el segundo asistente toma 2 de los cuadruplicados y se los acerca al comprador y al vendedor, quienes están en la tribuna observando el remate.

  Para este proceso se necesitan 3 personas, por el hecho de que mientras un lote de animales está siendo pesado en la balanza, en la pista va saliendo otro lote para ser vendido, por lo que la persona encargada de completar los datos, una vez que termina con una boleta, empieza con otra, además de hacer falta alguien que se encargue del transporte de las mismas de un lugar a otro.
- "*Cargar en una planilla los datos de cada venta realizada*": una vez terminado el remate, todos los datos del mismo son cargados por el consignatario en otra planilla (también en papel), a modo de resumen, donde se deja constancia de las ventas realizadas y toda la información pertinente a las mismas, partiendo de la información de las boletas.



GANADOS REMATES S.A.
CONSIGNATARIOS DE HACIENDA

SAN JUAN 957 - TEL: (0341) 4210223 - 4214311 - 4216107 - ROSARIO
info@ganadosremates.com.ar - www.ganadosremates.com.ar

Ventas en Mercado Rosario
Ferias: San Justo - Campo Andino - San Javier - La Criolla
Reconquista - Roldán
Remates televisados en directo por Canal Rural

VENDEDOR

Nº REMOVIDOS

Nº VENDEDOR

LOCALIDAD

Nº COMPRADOR

FECHA	LUGAR	CORRAL
21/2/21	LC.	2

COMPRADOR

NOV.	NOVTOS	VACAS	VAC. CRIA	VAQ.	TERN.	TOROS	PORCINOS
		1					

DESCRIPCION

PRECIO	KILOS EN PIE	ACTIVIDAD		PLAZO
120	640	Inver.	Abasto	Días
		DESTINO		

Venta sujeta a las condiciones del dorso.
CONFORME:
El firmante asume personalmente la responsabilidad de esta compra cualquiera sea el nombre para quien la haya efectuado.

TRIPLICADO
Para el Vendedor

FIRMA

Nº

Observaciones

C.U.I.T.: 30-53655257-6 R.U.C.A.: 1034 Ing. Brutos: 021-263571-9
I.V.A.: Resp. Inscripto D.N.R.P.: 216.631 Imp. Internos: No Responsable

TECNIGRAFICA - Av. Pte. Perón 3747 - Teletax (0341) 4325648 (rotativa) ENERO 2021 del 201001 al 207000

Figura 2: Boletas usadas actualmente

1.2.2 Problemas detectados

A partir de la información recabada anteriormente y de la detección de las principales actividades realizadas en el proceso de negocio, se detallan a continuación cuáles fueron los mayores problemas encontrados:

- La carga de información respecto a un remate (clientes, lotes de animales a vender, ventas, resúmenes, etc.) hoy en día es una tarea lenta y tediosa debido a que es relevada en cuadernillos de manera manual. Por otra parte, este mecanismo corre muchos riesgos de seguridad ya que, por su propia naturaleza, el soporte en papel es altamente sensible a rupturas, pérdidas de hojas o desorden de las mismas. Además, la información puede ser corrompida fácilmente (cambios en los datos, nuevos datos falsos, eliminación de datos, etc.). A medida que la cantidad de animales que participan en un remate crece, la probabilidad de estos riesgos aumenta peligrosamente.
- Durante la realización de un remate hay varios momentos durante una venta en la que existen “tiempos muertos”, es decir, momentos en los que se debe esperar a que las hojas utilizadas sean transportadas de un lugar a otro. Por ejemplo, una boleta de venta, en caso de que el lote vendido deba ser pesado, se entrega de un asistente a otro asistente, que es responsable de llevarla hasta el sector de la balanza, donde le cargan el peso. Es decir, se necesitan tres personas para llevar un papel de un lado a otro para que solo se cargue un simple número como información, además de los problemas descritos anteriormente por la utilización de papel físico como medio de información.
- A continuación, se resumen las actividades que en la actualidad utilizan papel en formato físico:
    - Los datos previos a la realización de un remate (clientes, lotes de venta, corral para los lotes, etc) se cargan en cuadernillos de papel.
    - Para cada venta de lotes se utilizan cuatro copias en papel con el mismo contenido de la boleta de compra/venta.
    - Al finalizar un remate, se carga la información de cada venta realizada en otro cuadernillo de papel.

  Si tomamos dimensión del volumen de papel que se utiliza durante la realización de un remate podemos encontrar dos problemas principales. Por un lado, los costos de adquisición de estos volúmenes de papel puede llegar a ser innecesariamente abultados. Por otro lado, y teniendo en cuenta la concientización actual sobre la protección del medio ambiente, creemos que éste punto es de suma relevancia a considerar y realizar una reingeniería del proceso de negocio que lleve a disminuir al máximo la utilización de papel.
- Por último, a medida que se van realizando los remates, la acumulación de papeles crece prácticamente de manera inmanejable para las personas y, si no se tienen criterios estrictos de almacenamiento y organización de los mismos (además que los

remates se realizan en varios lugares diferentes), encontrar información sobre remates antiguos es básicamente imposible.

1.2.3 Propuesta de mejora

Por lo expuesto anteriormente consideramos que sería de suma utilidad contar con un sistema de información que permita:

- Agilizar la carga de toda la información respectiva a un remate.
- Asegurar, integrar y agilizar el acceso a la información generada durante los remates, persistiendo la misma de manera digital. Estas tareas se tornan muy complejas actualmente debido a que toda la información se encuentra en planillas de papel.
- Lograr una mejora en la eficiencia y agilizar el proceso de venta de un lote de animales. El uso de un sistema de información ahorraría parte del tiempo perdido en transportar la boleta física de un lugar a otro y evitaría riesgos de deterioro de la misma.
- Generar digitalmente las boletas de venta e imprimirlas, solo si es necesario, ya que pueden ser entregadas a quien lo solicita de manera digital. Esto ahorraría muchos costes de papel y el proceso de reducción del uso del papel sería de gran ayuda para el cuidado del medio ambiente, por ejemplo.
- Generar automáticamente informes o resúmenes estadísticos luego de un remate, facilitando así el trabajo de los involucrados para tomas de decisiones futuras y dando además una forma de persistencia digital de todos estos datos, que de otra manera se almacenarían meramente en papel. Esto aumenta la accesibilidad y comprensión de dichos datos, además de las ventajas mencionadas sobre la reducción del uso de papel.

1.3 Población

En esta sección comentaremos aquellas personas o sectores que serán beneficiados y/o estarán involucrados en la realización de este proyecto, es decir, a quién será dirigido el artefacto resultante y por quién será desarrollado. Como se puede apreciar en lo enunciado anteriormente, podemos diferenciar dos grupos diferentes:

- Involucrados: con este término nos referimos a toda persona que está directamente relacionada a la ejecución y desarrollo del presente Proyecto Final de Carrera. En este caso, podemos destacar al equipo de desarrollo del software compuesto por los dos alumnos participantes en el proyecto y el director del mismo.

- Beneficiarios: en primera instancia, el beneficiario del sistema será la empresa Ganados Remates S.A. No obstante, debido a que las funcionalidades con las que contará el mismo son comunes en el sector al que nos estamos dirigiendo, la intención del equipo es desarrollar una aplicación con la flexibilidad suficiente para que sea adaptable a cualquier otro ente que se dedique a la consignación de haciendas.

1.4 Objetivos

A continuación, listamos aquellos puntos que debimos cumplir para poder otorgar al cliente una solución que solvente los problemas antes mencionados:

- Implementar una aplicación web que satisfaga las siguientes necesidades:
    - Generar boletas digitalmente, disponibles para impresión, con información sobre una venta durante la realización de un remate.
    - Cargar y persistir los datos generados previos a la realización de un remate:
        - Clientes (vendedores y compradores)
        - Categorías de haciendas
        - Los remates propiamente
        - Lotes de venta (para un remate en particular)
    - Cargar y persistir los datos generados durante un remate, en este caso los pertinentes a una venta realizada.
    - Generar informes de resultados sobre las ventas realizadas en un remate.
    - Generar un historial de remates, con toda la información pertinente, para que en un futuro puedan ser consultados por los usuarios del sistema.
- Implementar una aplicación web que sea portable, es decir, que esté disponible para distintos sistemas operativos desktop y móviles, así como que su visualización no se vea afectada por el tipo de navegador web que sea utilizado por el usuario.
- Realizar tests unitarios, pruebas de integración y pruebas de sistema sobre la aplicación para asegurar la calidad de la misma.

A su vez, para ejecutar los puntos anteriores de manera efectiva, previamente tuvimos que llevar a cabo las siguientes tareas:

- Investigar y recopilar información sobre el dominio para reducir las incertidumbres del problema que vamos a enfrentar. De esta manera, se va a delimitar el alcance del dominio para el presente proyecto.

- Realizar un análisis del problema para obtener como resultado los principales requerimientos funcionales, no funcionales, componentes necesarios del sistema, y definir las relaciones entre los mismos.
- Diseñar las interfaces gráficas del sistema, enfocándonos en brindar una buena experiencia de usuario, esto es, que las mismas resulten atractivas, intuitivas y adaptables para su utilización en distintos tipos de dispositivos.
- Estudiar y profundizar conocimientos sobre las tecnologías a utilizar para llevar a cabo el desarrollo del sistema propuesto, principalmente, el framework Spring (de Java), React JS y librerías de UI.
- Estudiar e investigar sobre buenas prácticas de documentación, versionado y organización de código para generar un producto de software que sea fácil de mantener para los desarrolladores.

2 Conceptos previos

En este capítulo se definirán y detallarán aquellos conceptos que creemos de suma importancia plasmar en este informe para facilitar la lectura y comprensión del lector. Cabe destacar que estas definiciones son el resultado de las distintas entrevistas e investigaciones que realizamos porque nosotros mismos necesitábamos conocer y entender el ambiente en el cual estábamos trabajando para poder brindar la mejor solución al cliente, dentro de un modelo de negocios donde nuestro conocimiento era escaso.

A continuación se describen los conceptos más relevantes.

2.1 RENSPA

El RENSPA es el Registro Nacional Sanitario de Productores Agropecuarios que abarca a todas las actividades agrícolas ganaderas y forestales y asocia al productor con la producción y el predio.[1]

2.2 Número de RENSPA

El Registro Nacional Sanitario de Productores Agropecuarios le asigna un identificador al establecimiento agropecuario, predio o lugar físico donde la explotación agropecuaria está asentada, y un subcódigo para identificar los distintos productores que coexisten en él, por lo que cada productor tiene asignado un número por cada establecimiento que posea o alquile.[1]

2.3 Senasa

El Servicio Nacional de Sanidad y Calidad Agroalimentaria es un organismo descentralizado, con autarquía económico-financiera y técnico-administrativa y dotado de personería jurídica propia, dependiente del Ministerio de Agricultura, Ganadería y Pesca, encargado de ejecutar las políticas nacionales en materia de sanidad y calidad animal y vegetal e inocuidad de los alimentos de su competencia, así como de verificar el cumplimiento de la normativa vigente en la materia.

También es de su competencia el control del tráfico federal y de las importaciones y exportaciones de los productos, subproductos y derivados de origen animal y vegetal, productos agroalimentarios, fármaco-veterinarios y agroquímicos, fertilizantes y enmiendas.
En síntesis, el Senasa es responsable de planificar, organizar y ejecutar programas y planes específicos que reglamentan la producción, orientándola hacia la obtención de alimentos inocuos para el consumo humano y animal.[2]

2.4 Número de DT-e

Documento de Tránsito Electrónico, el Senasa le asigna un número particular e irrepetible a cada productor, por cada establecimiento del que lleve animales al remate, por cada remate en particular. Ejemplo: un productor tiene un campo propio y uno alquilado, de ambos lleva animales a un remate el día 11/3/2021, ese productor va a tener 2 números de DT-e, que le van a servir solo para ese remate. Si desea llevar animales a un remate próximo, se le asignarán números distintos (que se generan en el momento).

2.5 Remate

Subasta pública donde los productores ganaderos, llevan su hacienda de diversa categoría, peso y estado para ser vendidos en subasta pública y los interesados, como frigoríficos, engordadores a campo o a través de Feedlot, carniceros, criadores, etc. luego de ver a los animales en los pequeños corrales en que son colocados, realizan su apuesta de compra del ganado que es de su interés.[3]

2.6 Consignatario

El consignatario es quien recibe en consignación la mercadería de otra persona (natural o jurídica). Esto, con el objetivo de ofrecer esos productos al público, quedándose a cambio con una parte de las ventas efectuadas.[4] En el caso que nos compete, estas "mercaderías" serían los animales.

2.7 Martillero

El Martillero Público también denominado Rematador es el profesional encargado de subastar de forma privada o pública vía judicial, cualquier clase de bienes muebles, inmuebles, semovientes y derechos, marcas, patentes y en general todo bien cuya venta no esté prohibida por la Ley o encomendadas a otras profesiones específicas.[5]

La función de los martilleros es orientar sobre los precios de la hacienda y tratar de sacar el precio más alto, usando como parámetro al mercado.

2.8 Cliente

Persona o entidad que vende y/o compra animales durante un remate.

2.9 Feria

Lugar físico donde se realizan los remates. Pueden existir variaciones según las comodidades con la que cuente cada locación, pero en términos generales cuentan con:

- **Corrales:** Recinto cercado y generalmente descubierto que sirve para guardar el ganado en espera de ser vendido.
- **Pista:** Corral de gran tamaño en el que se le presentan los animales a los clientes para que los mismos puedan observarlos de cerca.
- **Cabina del martillero:** Lugar normalmente elevado, donde se encuentra el martillero, habitualmente acompañado por el consignatario, quien lo asiste con la información relativa a los lotes que se van presentando.
- **Tribuna:** Lugar donde los clientes se ubican para presenciar el remate, con asientos que van escalando en altura para facilitar la visión hacia la pista (la cual está justo en frente).
- **Balanza:** Dispositivo usado para pesar los animales una vez vendidos (en caso de que así corresponda). Suele haber un corral justo entre la pista y la balanza donde pasan los animales una vez vendidos, para luego ingresar a la balanza propiamente. La medida del peso se toma desde una habitación contigua donde está ubicada la pantalla que arroja esta información.

2.10 Lote

Dentro del contexto que se trata en este trabajo, se define lote simplemente como un conjunto de animales. El problema con el que nos encontramos es que esta definición es demasiado abarcativa, por lo que para evitar confusiones, hemos creado sub definiciones de la misma en base a los distintos usos que tiene este término:

- **Lote de venta:** conjunto de animales de un cliente (en el rol de vendedor) que se encierran juntos en un corral y comparten un mismo número de DT-e.
- **Animales en pista:** Conjunto de animales que, como su nombre lo dice, salen todos juntos a la pista para ser vendidos. Este grupo de animales comparte la categoría (son todas vaquillas, por ejemplo). Un lote de venta puede estar conformado por uno o varios “Animales en pista” (suelen ser entre uno y dos en la mayoría de los casos).
- **Lote vendido:** Grupo de animales que fueron comprados por un cliente (en el rol de comprador). Este conjunto puede estar formado por la totalidad de un “Animales en pista” o por un subconjunto de este (por ejemplo, en el caso de que haya tres toros en pista y el cliente compre solo uno). Estos lotes, además de compartir la categoría, heredado del “Animales en pista” del que surgen, tiene un precio, un peso (en caso

de que deban ser pesados, que no siempre es el caso) y posteriormente, se les asigna un número de DT-e para que puedan viajar a su destino final.

- **Lote no vendido:** Conjunto de animales que no fueron vendidos durante el remate. Al igual que el caso anterior, puede estar formado por la totalidad de un “Animales en pista” o por un subconjunto de este. En este caso, obviamente no tienen un precio ni se los pesa.

2.11 Procedencia

Una procedencia hace referencia al establecimiento al que está relacionado un Cliente o del cual proviene un Lote de Venta. Para nuestro alcance en el sistema, una procedencia está compuesta por:

- **Referencia:** nombre identificatorio de la procedencia.
- **Número de RENSPA:** para que el Registro Nacional Sanitario de Productores Agropecuarios identifique el establecimiento.

3 Metodología

Para el desarrollo de este proyecto adoptamos un enfoque ad-hoc basado en la metodología ágil Scrum, tomando su filosofía de trabajo y valores, así como muchas de sus prácticas, desarrollando algunas de ellas tal cual lo aconseja la metodología, basándonos en el libro “Proyectos Ágiles con Scrum: Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos”[6], y adaptando otras a nuestra situación particular. Entre las adaptaciones más relevantes a nombrar está la de los roles propuestos por la metodología, los cuales, al ser el grupo de trabajo conformado por 2 personas solamente, ambos tomaríamos los papeles de Product Owner y equipo de desarrollo, siendo el rol de Scrum Master ocupado en parte por el director de proyecto, para las funciones de asesoramiento tanto sobre la metodología así como de temas técnicos o de gestión, y por nosotros mismos para las demás tareas que este rol engloba. A los elementos de la metodología (Product Backlog, Sprint Backlog e incremento funcional potencialmente entregable) los usaremos tal como se recomienda, y por último, respecto a los eventos del flujo de trabajo, también los respetaremos, con excepción de los Daily Meetings, los cuales solo se realizarán cuando surjan obstáculos para el avance del trabajo, simplemente dándose el resto de la comunicación de forma fluida y continua entre los integrantes del grupo (lo cual es posible por el hecho de ser 2).

Los detalles sobre la metodología Scrum y cómo adaptamos los roles, flujo de trabajo y elementos que la componen se describen con más detalle en el Anexo A presentado con este documento. Para mantener el orden del presente informe final, también elaboramos un anexo que corresponde a todo lo relacionado con la Gestión de Riesgos de este Proyecto, cuya denominación es Anexo C - Gestión de riesgos . Por último, el Anexo D - Ejecución del proyecto está destinado a mostrar cómo fue la evolución de este proyecto a lo largo del tiempo, junto a resultados y conclusiones parciales que se fueron obteniendo a medida que transcurrían los Sprints.

3.1 Justificación de la elección

El motivo principal por el que el equipo decidió organizarse dentro del marco de un enfoque ágil es que, si bien se realizaron varias entrevistas e investigaciones previas, existen muchas incertidumbres en el modelo de negocios que estamos trabajando. Entonces, gracias a este tipo de metodología, pudimos cerrar una primera idea general y planificamos una solución, pero la misma no fue una solución final, sino que permitimos que pueda ir evolucionando a medida que avanzamos en el proyecto y fuéramos aprendiendo más sobre el dominio del problema (o el mismo cliente pudiera presentar modificaciones de

su idea). Esto nos permitió adelantar los tiempos para comenzar con la ejecución del proyecto ya que en los primeros Sprints aprovechamos a implementar aquellas funcionalidades en las cuales teníamos mayores certezas y/o no recaían en el núcleo del modelo de negocio, por ejemplo, inicio de sesión, CRUD de entidades básicas, etc. Así, a medida que profundizamos las investigaciones y perfeccionamos las historias de usuario donde había mayores incertidumbres, ya teníamos requisitos funcionales complementarios desarrollados, lo cual nos permitió enfocarnos con mayor énfasis en la lógica de negocio esencial para nuestro sistema. Además de este motivo, nombramos otros puntos que favorecieron a la elección de este enfoque basada en Scrum:

- En la primera fase de análisis fue fundamental la participación y retroalimentación del cliente a través de la redacción de las historias de usuario debido a nuestro desconocimiento del negocio en que nos estábamos introduciendo.
- Presenta elementos para la fase de ejecución del proyecto con el cual estamos familiarizados y tenemos cierta experiencia básica.
- La entrega de pequeñas porciones de producto que den valor al cliente nos posibilitó tener retroalimentaciones constantes y que podamos comprender con mayor exactitud sus necesidades.
- Nos permitió aceptar cambios propuestos por el cliente durante el desarrollo del producto, los cuales se pudieron incorporar a la lista de tareas (backlog) y determinar su incorporación al ciclo de desarrollo sin grandes complicaciones.

3.2 Plan de monitoreo periódico del proyecto

En base a la metodología seleccionada, y como la misma aconseja, el monitoreo de avances se llevó a cabo al final de cada Sprint, donde se realizó la Sprint Review, reunión que sirve a este fin, entre otros. Al mostrar aquí los avances en el proyecto, aprovechamos también para analizar el avance del Backlog, repasando lo que se había hecho hasta el momento y mirando a futuro lo que quedaba por desarrollar.

4 Análisis y Desarrollo

4.1 Relevamiento de requerimientos

En este apartado mostraremos el proceso de relevamiento de requerimientos que tuvo lugar en un principio, previo al inicio concreto del desarrollo del sistema, y su evolución hasta llegar a los primeros “mockups” lo suficientemente detallados como para plasmar los requerimientos en historias de usuario.

En un principio, una vez surgida la idea de un sistema para la gestión general de los remates y la generación de su documentación, el siguiente paso fue tener una serie de entrevistas o charlas con un representante de la firma Ganados Remates S.A., donde surgieron las principales funcionalidades de las cuales el sistema no podría carecer. La dinámica general del proceso consistió en tener una reunión para luego generar algún tipo de “mockup” o maqueta con el fin de confirmar la correcta comprensión de las necesidades de los usuarios, para presentarlas en el siguiente encuentro, obtener una aprobación o correcciones, y así sucesivamente.

Durante y luego de la primera reunión, se procedió a plasmar las ideas iniciales a “mano alzada”, a modo de borrador, de forma de producir un artefacto de manera simple y rápida con la idea de obtener un feedback prácticamente instantáneo por parte del entrevistado, con el objetivo de ratificar lo hablado y asegurarnos de estar comprendiendo sus necesidades. A continuación se presentan estos primeros bocetos:

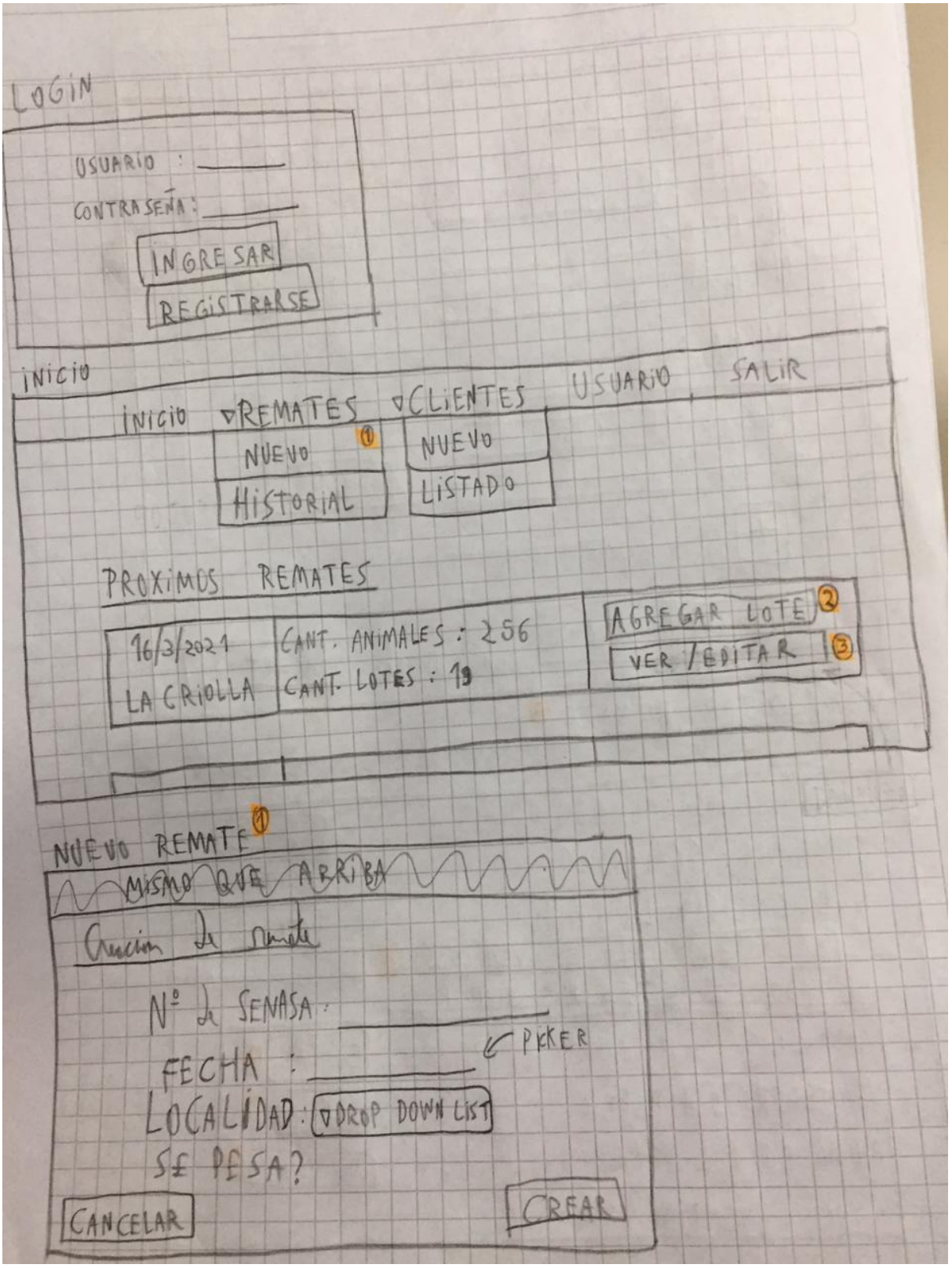


Figura 3: Bocetos en papel de LogIn, Inicio y Nuevo remate

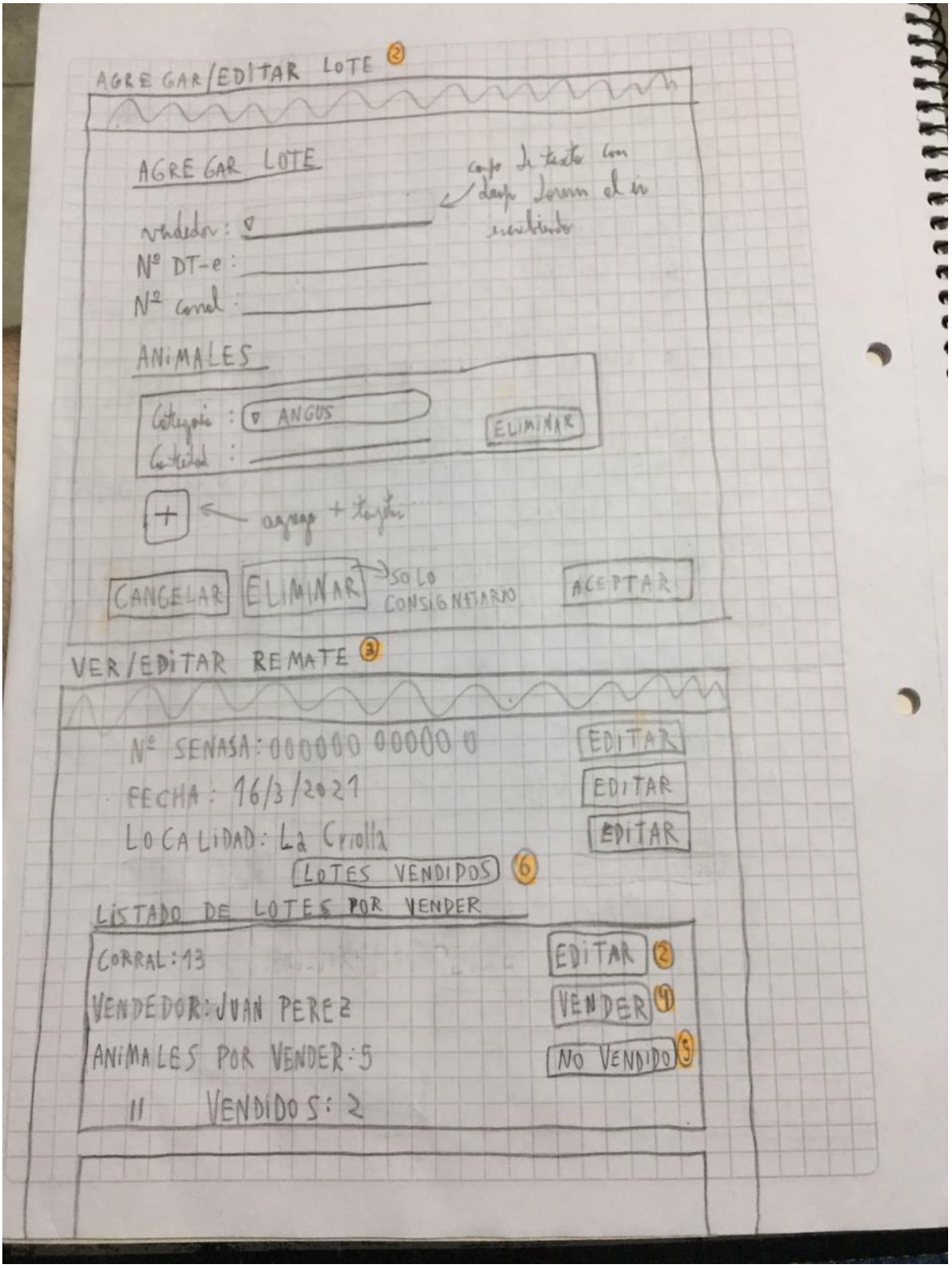


Figura 4: Bocetos en papel de Agregar/Editar lote y Ver/Editar remate

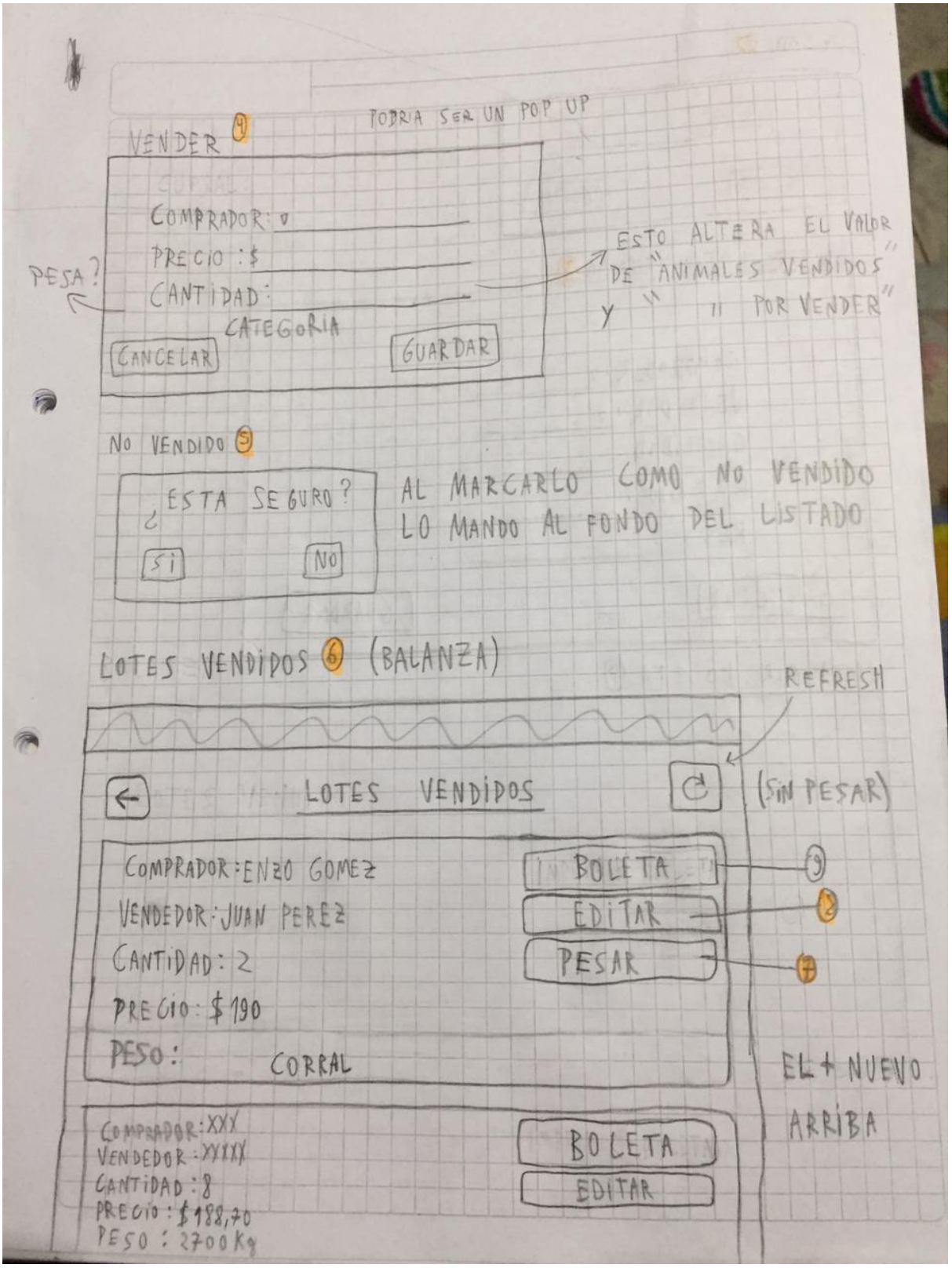


Figura 5: Bocetos en papel de Vender, No vendido y Lotes vendidos

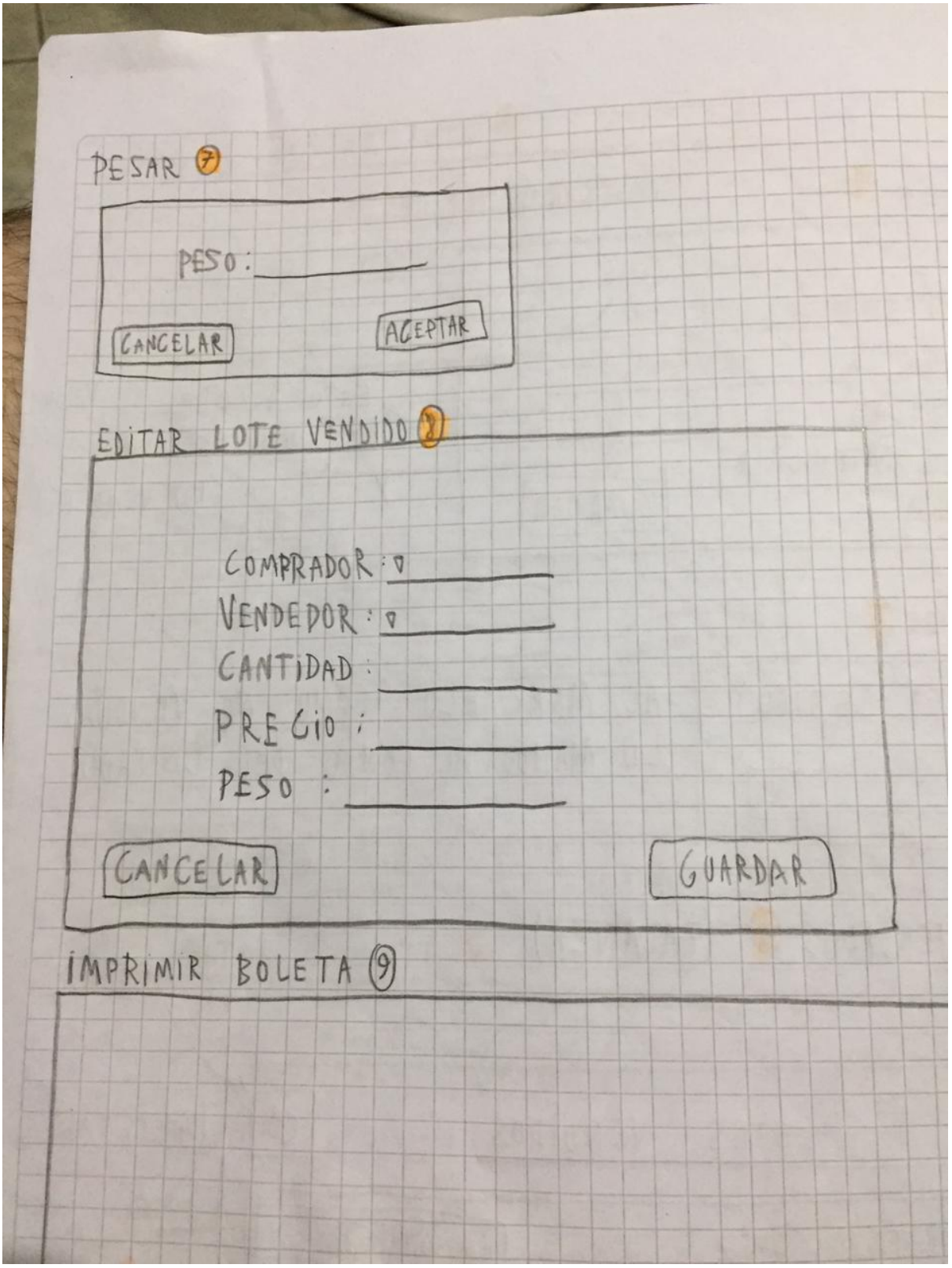


Figura 6: Bocetos en papel de Pesar y Editar lote vendido

Una vez convalidadas estas primeras muestras, nos desplazamos a la herramienta Figma, con el fin de engendrar maquetas que se aproximen con mayor exactitud a lo que luego terminaría convirtiéndose en el resultado final. Es decir, el objetivo de esta etapa fue

generar prototipos que actúen como guía a lo largo del desarrollo, además, nuevamente, de servir para revalidar con mayor precisión los requerimientos del usuario. Se presenta aquí uno de estos “mockups” a modo de ejemplo (todos ellos luego aparecerán en el Anexo E):

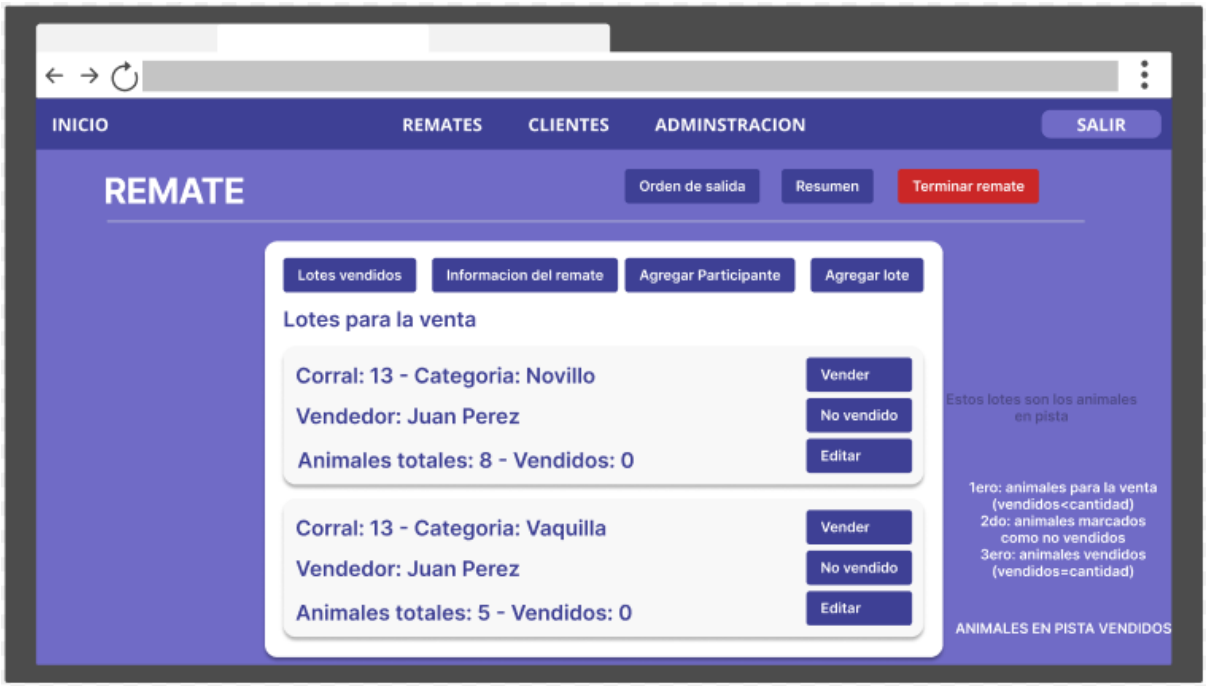


Figura 7: Mockup en Figma de Ver/Editar remate

De este apartado, también tuvimos como resultado un primer diagrama de clases del sistema, que nos sirvió internamente como orientación durante el desarrollo, el cual fue sufriendo cambios durante el mismo. A continuación presentamos el diagrama de entidades/clases definitivo:

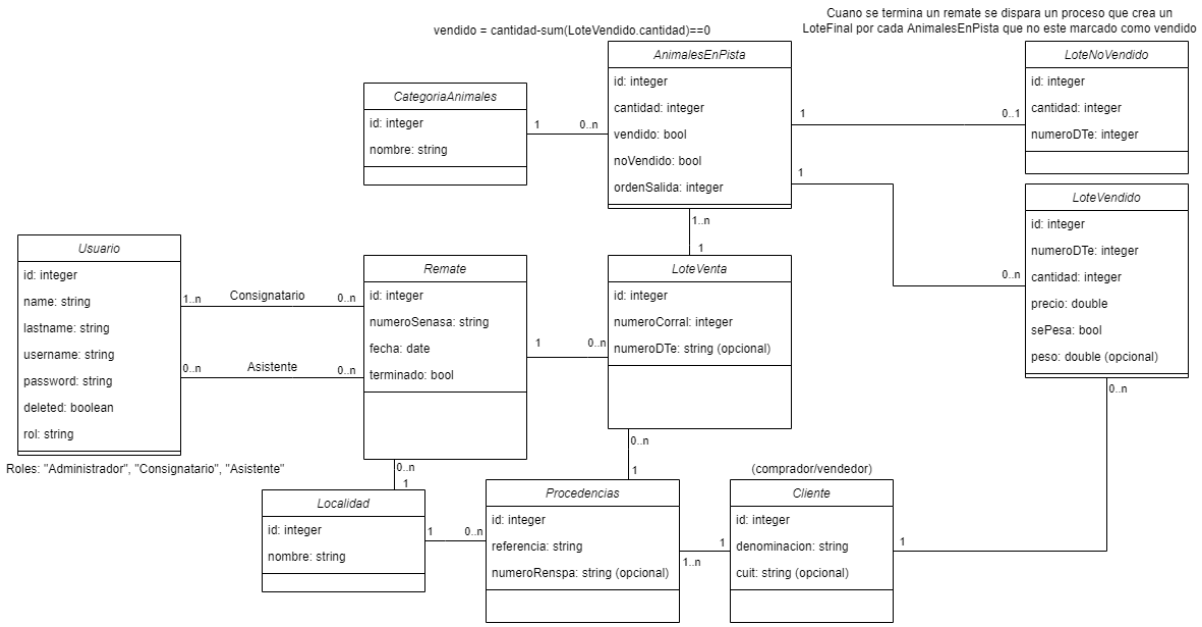


Figura 8: Diagrama de clases del sistema

Una vez despejadas nuestras dudas y con la aprobación del representante de la empresa, nos pusimos manos a la obra con el plan de ejecución del proyecto.

4.2 Historias de usuario y priorización de las mismas

El siguiente paso fue plasmar los requisitos previamente obtenidos de manera clara y precisa, para lo cual, basándonos en la metodología seleccionada, esta tarea fue realizada mediante historias de usuario. Aquí presentamos un listado de las mismas solo a manera de enunciado (su presentación formal puede encontrarse en el Anexo B).

Cabe aclarar que aquí se utiliza una jerarquía, donde tenemos 3 tipos de usuario: administrador, consignatario y asistente, donde cada uno, además de realizar sus tareas propias, puede realizar las correspondientes a su categoría inferior. Por ejemplo, cuando se habla de que “como usuario asistente” se puede hacer algo, tanto el consignatario como el administrador también pueden realizar esa actividad. Cuando simplemente se habla de usuario, sin especificar rol, se refiere justamente a cualquier usuario dentro del sistema.

4.2.1 Listado de historias de usuario

Administración de usuarios:

1. Como usuario debo poder iniciar sesión con mi usuario y password, permitiéndome entrar a ver la información dentro del sistema.
2. Como usuario debo poder ingresar a mi información personal en el sistema y poder editarla.
3. Como usuario administrador debo poder crear nuevos usuarios, con diferentes roles, para que gestionen los remates y, además, tener la posibilidad de editarlos.

Administración de entidades del sistema:

4. Como usuario quiero acceder a un listado de clientes donde, además, pueda crear/editar clientes con ciertos datos que lo identifiquen, como denominación, cuit y procedencias (como campos o estancias, por ejemplo), así como eliminarlos.
5. Como usuario consignatario quiero crear, ver, modificar y eliminar localidades para que sean asignadas a procedencias y remates.
6. Como usuario consignatario quiero crear, ver, modificar y eliminar categorías de animales para identificar los tipos de animales de un lote de venta.

Gestion de remates:

7. Como usuario consignatario quiero poder crear nuevos remates con todos los datos relacionados al mismo, así como editarlos posteriormente, o eliminarlo.

8. Como usuario consignatario, luego de crear un remate, quiero asignar asistentes que se encarguen de gestionar la información generada durante la realización del mismo.
9. Como usuario asistente debo poder cargar lotes para la venta en un remate, con información como vendedor, procedencia, número de corral, DTe, cantidad de animales de cada categoría.
10. Como usuario asistente debo poder descargar una lista en formato pdf del orden de salida de animales durante la ejecución de un remate.
11. Como usuario asistente debo poder cargar datos relativos a las ventas de animales, entre otras cosas, cantidad vendida, comprador, precio y peso del lote comprado; o marcarlos como no vendido.
12. Como usuario asistente debo poder asignarle el número de DTe a los lotes vendidos una vez terminado el remate.

Listados de remates:

13. Como usuario quiero poder ver un historial de remates ya realizados, a los cuales puedo acceder a ver más información.
14. Como usuario debo tener acceso al listado de remates próximos con los que estoy relacionado e interactuar con los mismos, así como ver información de remates creados por otros usuarios.

Reportes y boletas:

15. Como usuario asistente quiero poder acceder a información y reportes estadísticos de cada uno de los remates realizados, por ejemplo, cantidad de animales no vendidos, precio total, peso promedio, estadísticas de cada cliente, entre otros.
16. Como usuario asistente quiero tener la posibilidad de imprimir aquellos reportes estadísticos que me interesen.
17. Como usuario asistente debo poder generar, descargar e imprimir las boletas de los lotes que son vendidos.

4.2.2 Priorización de las historias de usuario

Al no tener definidas prioridades por parte del negocio, hemos decidido dar prioridad a las historias de acuerdo a las dependencias funcionales que existen en el proyecto. De esta forma, se inicia con aquellas historias que no dependen de otras para su correcto funcionamiento, y se asigna una baja prioridad (relegando para el final) a las que necesariamente dependen de otras historias.

En el siguiente diagrama se presentan las dependencias a las que arribamos luego de un estudio de los requerimientos de cada una de las historias, donde, por ejemplo, para poder desarrollar la historia 8 (asignar un asistente a un remate), antes debíamos contar con las funcionalidades de las historias 3 (crear nuevos usuarios) y 7 (crear nuevo remate).

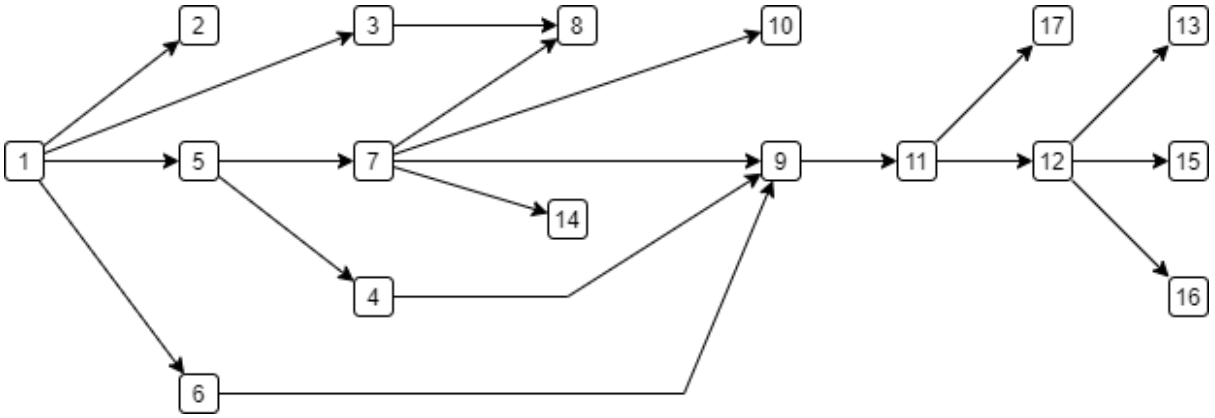


Figura 9: Diagrama de dependencias entre historias de usuario

4.3 Estimación de duración del proyecto y recursos humanos

4.3.1 Recursos humanos

La cantidad de recursos humanos involucrados en el desarrollo del proyecto fueron dos personas (ambos tesistas) con una carga horaria semanal de 30 horas de trabajo por recurso. Cabe mencionar también al Director de Proyecto, quien prestó su tiempo en función de la demanda de los antes mencionados.

4.3.2 Estimación de duración del proyecto

Basándonos en lo que propone la metodología de desarrollo seleccionada, la unidad de medida para la estimación serán **story points**, donde propusimos que un story point sea equivalente a **6 horas de trabajo** (un dia de trabajo).

Se realizarán sprints de **2 semanas** cada una (30 horas de trabajo por semana por cada integrante).

Un sprint son entonces **20 story points** (120 horas de trabajo = 30 horas semanales * 2 semanas * 2 desarrolladores), lo que entonces da un velocidad de **20 story points/Sprint.**

Con estos datos, se puede calcular que la cantidad de **sprints necesarios** es de **6**, mediante la siguiente cuenta:

116 (total de story points, explicado en el siguiente punto) / 20 (velocidad) = 5.8 ~ 6 Sprints, es decir, 12 semanas o unos 3 meses.

4.3.3 Estimación y planificación de los sprints

Para finalizar, hemos analizado cada historia de usuario y estimado la cantidad de story points que demandarían. A continuación, a modo de resumen, mostraremos el tiempo que nos tomará el desarrollo de este proyecto y cómo se organizará la implementación de las historias de usuario dentro de cada sprint, respetando las dependencias previamente enseñadas.

Planificación de Sprints			
Número de Historia	Story Points	Horas	Sprint
1	6	36	1
2	6	36	1
5	4	24	1
6	4	24	1
7	10	60	2
14	6	36	2
3	8	48	3
4	8	48	3
8	4	24	3
9	10	60	4
11	10	60	4
10	2	12	5
12	6	36	5
13	4	24	5
17	8	48	5
15	10	60	6
16	10	60	6
Suma	116	696	
Sprints	5.8		
Meses	2.9		

Sprints	SP del Sprint
Sprint 1	20
Sprint 2	16
Sprint 3	20
Sprint 4	20
Sprint 5	20
Sprint 6	20

Como se puede observar, según nuestras estimaciones iniciales, el desarrollo de este proyecto debería haber tenido una duración de 6 sprints, es decir, unos 3 meses aproximadamente. Sin embargo, adicionamos una semana al inicio del proyecto (a la cual denominamos *sprint 0*) en el cual nos dedicamos a acondicionar nuestro ambiente de trabajo, realizando actividades tales como la instalación de las herramientas necesarias, construcción del proyecto inicial, configuración del software del versionado, diseño y construcción de base de datos, entre otras tareas.

Luego de iniciado el proyecto, durante el periodo de descanso acordado por vacaciones (24/12/2021 - 17/01/2022) se desarrollaron funcionalidades extra, las cuales no fueron tenidas en cuenta a la hora de confeccionar el plan del proyecto, por lo que aquí tampoco serán tenidas en cuenta, sino que serán comentadas directamente en el Anexo D. El objetivo de desarrollar estas funcionalidades fue mejorar la experiencia de usuario en la aplicación y facilitar algunas de sus tareas relacionadas a la ejecución de un remate.

Como conclusión, la duración total de nuestro proyecto será de 6 sprints, más dicha semana para el acondicionamiento, lo cual significa que nos tomará aproximadamente 3 meses (65 días de trabajo).

4.4 Cronograma de avance del proyecto

Como acabamos de precisar en los puntos anteriores, se necesitaron 6 sprints de 2 semanas cada uno para la construcción del sistema en cuestión (más esa semana previa de preparación y algunas semanas extras al final del desarrollo que más adelante se detallarán).

Una vez demarcada la duración, se deben asignar los sprints definidos al momento de inicio concreto del proyecto. A grandes rasgos el proyecto tomaría lugar entre fines del mes de octubre de 2021 y mitad del mes de febrero de 2022.

En la siguiente tabla presentamos de forma detallada cada sprint junto a la fecha de inicio y fin de cada una de sus semanas:

Planificación por semanas de trabajo			
N°	**Desde**	**Hasta**	**Sprint**
1	25/10/2021	29/10/2021	0
2	01/11/2021	05/11/2021	1
3	08/11/2021	12/11/2021	
4	15/11/2021	19/11/2021	2
5	22/11/2021	26/11/2021	
6	29/11/2021	03/12/2021	3
7	06/12/2021	10/12/2021	

8	13/12/2021	17/12/2021	4
9	20/12/2021	24/12/2021	
10	17/01/2022	21/01/2022	5
11	24/01/2022	28/01/2022	
12	31/01/2022	04/02/2022	6
13	07/02/2022	11/02/2022	

Una vez terminada la etapa de desarrollo aquí detallada, se dio inicio a la fase de redacción de este informe final, la cual comenzó el 14/3/2022, luego de reuniones de revisión con el Director de Proyecto, donde se manifestaron ideas de mejora, principalmente relacionados a experiencia de usuario y apartados visuales del front-end, pero esto será comentado con más detalle en el Anexo D, que trata específicamente la ejecución del proyecto.

4.5 Descripción general del sistema

En este apartado mostraremos cómo es la estructura de nuestro sistema a grandes rasgos. Es decir, describiremos aquellas funcionalidades genéricas como por ejemplo, las operaciones CRUD (Create-Read-Update-Delete, o en español, Creación-Consulta-Edición-Eliminación) de las distintas entidades dentro del sistema y cómo decidimos mantener la coherencia de los componentes visuales de cada una. Mientras que, por otro lado, comentaremos aquellas funcionalidades que consideramos fundamentales para que el producto sea considerado funcional, seguro y de valor para el cliente, por ejemplo, cómo implementamos el inicio de sesión, el proceso de realización de un remate, la generación de reportes estadísticos del mismo, etc.

En la sección “Ejecución de un remate en el sistema desarrollado” se mostrará de manera detallada la solución final a la que se llegó paso a paso.

4.5.1 Log In

Por el hecho de ser una aplicación web de acceso público, se necesitó de un mecanismo de “logueo” o autenticación para que solo aquellas personas autorizadas y registradas puedan acceder al sistema. Al ingresar al sitio web, se presenta una pantalla para que aquella persona que tenga acceso utilice su nombre de usuario y contraseña con el que está registrado para ingresar.

Un punto importante que vale aclarar en este apartado es la forma en que se asegura el envío de esta información, que es crítica dentro del sistema propuesto,

fundamentalmente la contraseña. Para esto, explicaremos tanto la situación en que se da de alta un usuario (o se modifica su información), como cuando éste inicia sesión.

Entonces, desde el lado de un dispositivo cliente, previo a que la contraseña sea enviada al servidor, se aplica sobre la misma el algoritmo hash de cifrado *MD5* para evitar que viaje por Internet como texto plano y lograr cierto nivel de seguridad, asegurando confidencialidad.

Una vez que llega al servidor, existen dos posibilidades: que la contraseña tenga que almacenarse porque un nuevo usuario se dió de alta (o uno existente modificó la misma), o bien, que sea usada para validar la identidad de un usuario.

Para el primer caso, una vez que este dato es recibido y antes de ser almacenado en la base de datos, se le aplica un nuevo algoritmo hash criptográfico basado en el cifrado Blowfish[1], llamado *BCRYPT*[2].

Para el segundo caso, se debe buscar la contraseña almacenada en la base de datos según el nombre de usuario de quien intenta iniciar sesión, luego se aplica la función hash bcrypt al dato que llega en la solicitud desde el cliente y se comparan ambos resultados: si ambas cadenas de texto son iguales, entonces el inicio de sesión es exitoso, se devuelve el JWT correspondiente al cliente y el usuario ingresa al sistema. En caso contrario, se retorna un mensaje de error con el mensaje “Credenciales inválidas”.

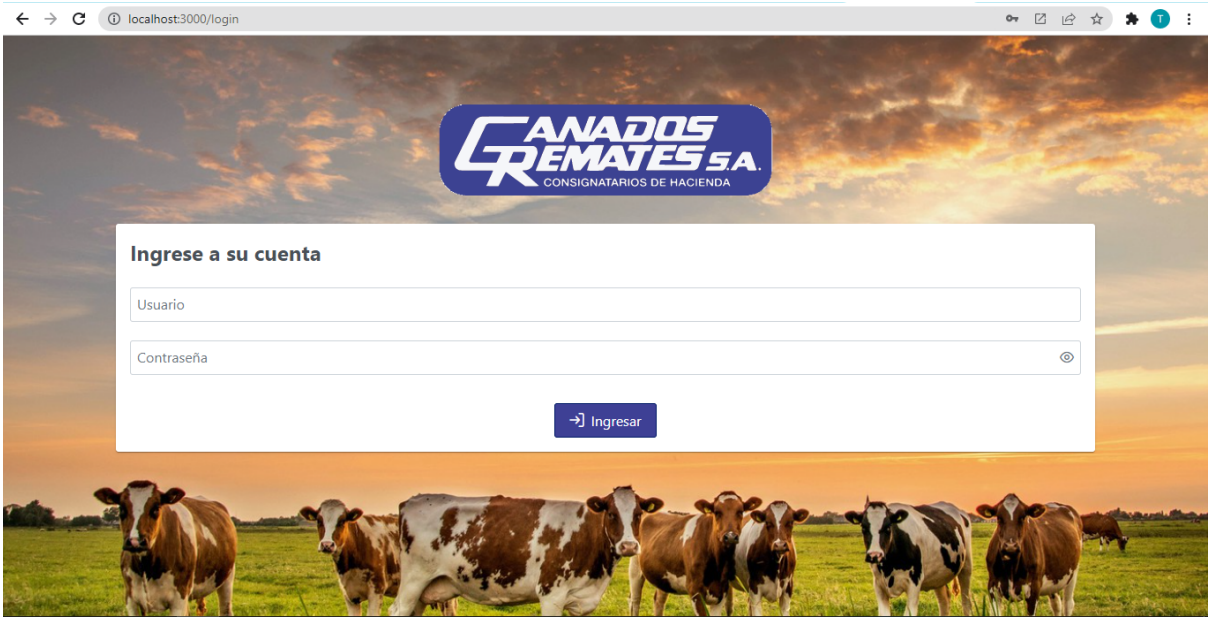


Figura 10: Pantalla de autenticación o “Log In”

[1] Es un algoritmo de cifrado de bloque de clave simétrica de bloque de 64 bits y longitud de clave variable, que se puede utilizar para cifrar cadenas de 64 bits de longitud.[7] Es uno de los primeros cifrados de bloques seguros que no están sujetos a ninguna patente y, por lo tanto, están disponibles de forma gratuita para que cualquiera los use.[8]

[2] bcrypt es una función de hash de contraseñas que, combinada con un número variable de iteraciones, aprovecha la costosa fase de configuración de claves de Blowfish para aumentar la carga de trabajo y la duración de los cálculos de hash, reduciendo aún más las amenazas de los ataques de fuerza bruta.[9]

4.5.2 Home

Para los usuarios con los roles de consignatario y asistente, la pantalla de Home muestra 2 listados de los próximos remates que serán llevados a cabo por Ganados Remates S.A., divididos en las pestañas “Mis remates” y “Remates ajenos”, según el usuario participe o no en dichos remates.

En el caso de los administradores, se les muestra directamente un solo listado con todos los remates próximos, porque se considera que “participan” en la totalidad de los remates.

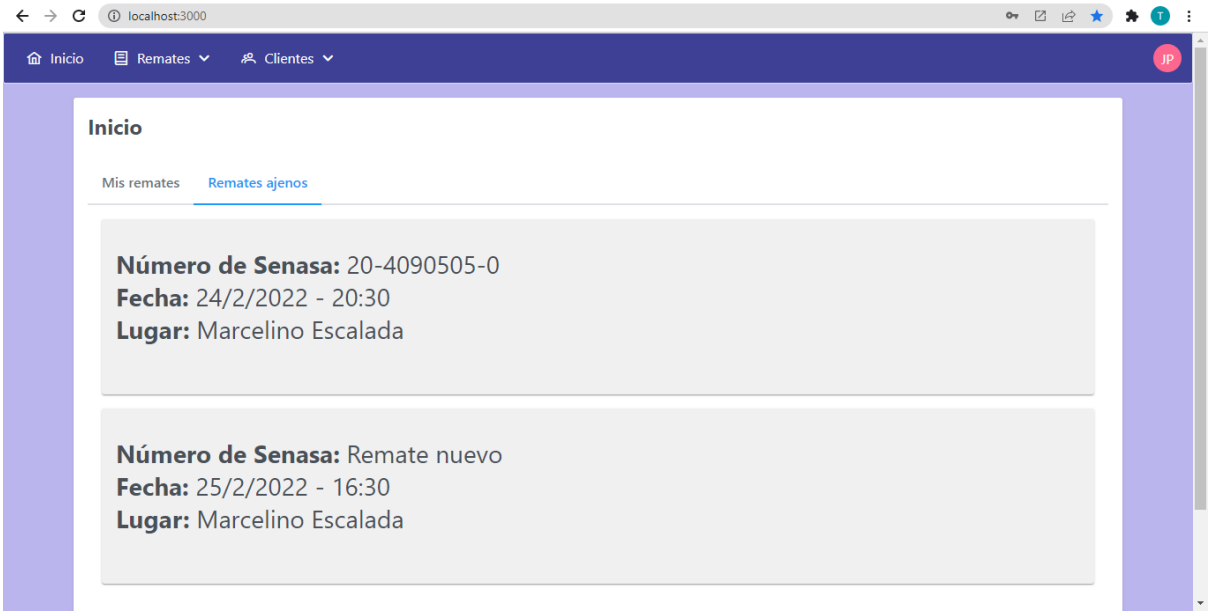


Figura 11: Pantalla de inicio o “Home”

4.5.3 Creación de entidades

Una pantalla de creación de nueva entidad dentro de la aplicación consta de un formulario para ingresar la información necesaria para la nueva entidad y luego, al menos dos botones para “Cancelar” o “Guardar” la nueva información

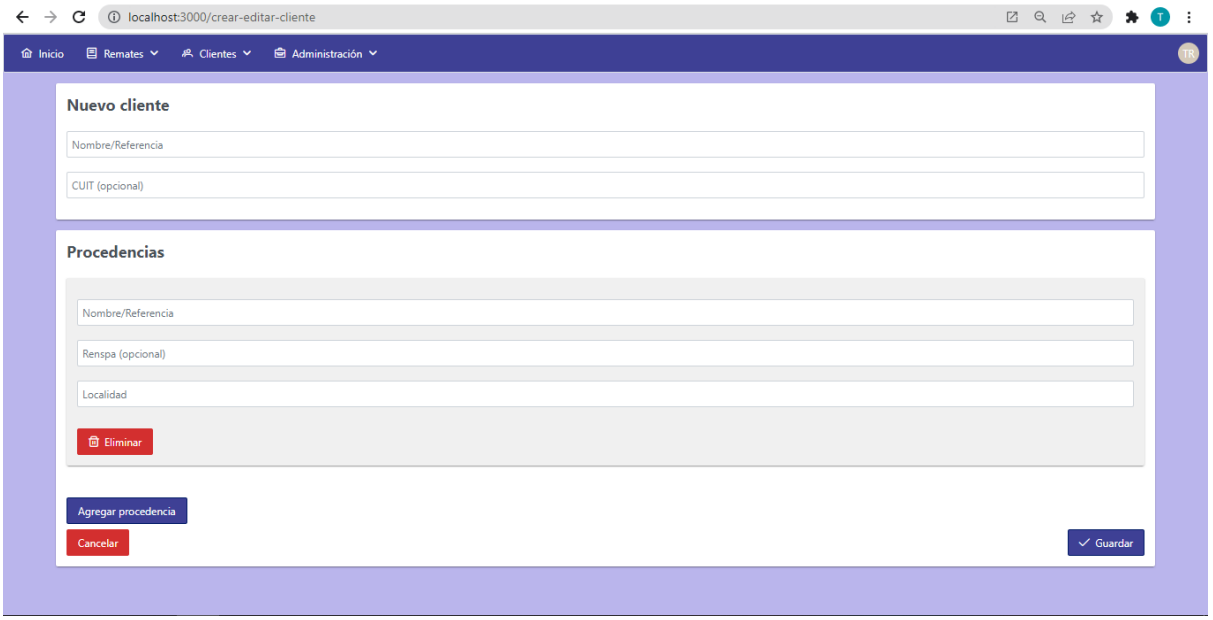


Figura 12: Pantalla CRUD de cliente (creando)

4.5.4 Modificación de entidades

Con el objetivo de reutilizar componentes ya desarrollados, en general, la pantalla de creación de entidades es la misma que la de modificación. Es decir, se utiliza el mismo formulario, con la diferencia de que, si se está editando una entidad, los campos del formulario están rellenos con la información ya existente de la misma.

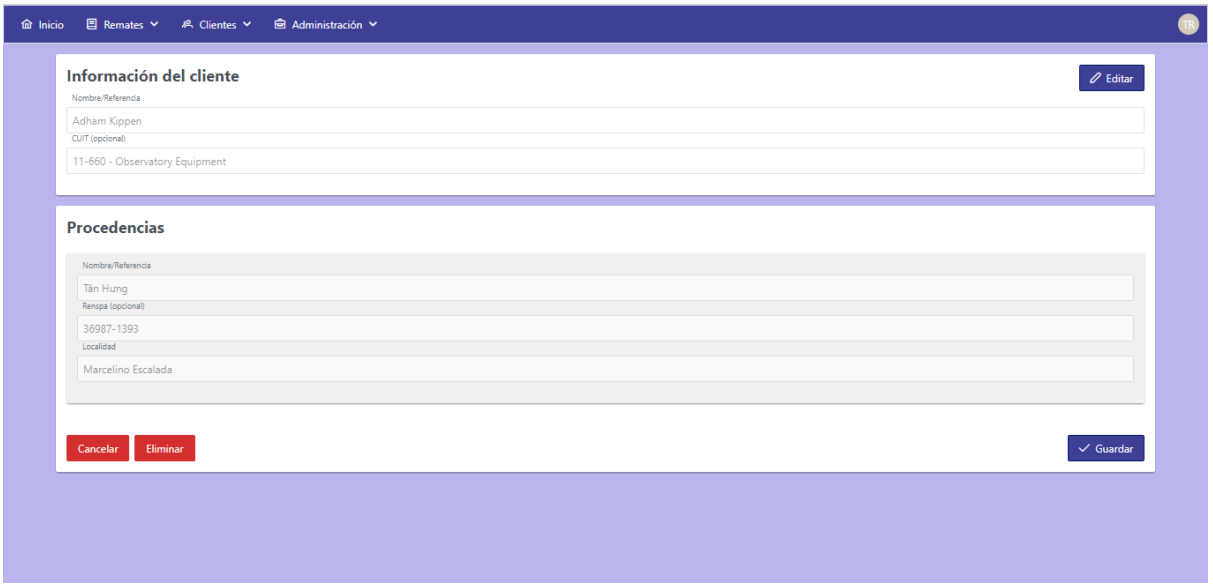


Figura 13: Pantalla CRUD de cliente (editando)

4.5.5 Consulta de entidades

Las pantallas de consultas de entidades, en general, constan de un listado en formato de “tarjetas” para cada entidad. El mismo está paginado con un número determinado de resultados por página (que es editable) y al seleccionar alguna de las entidades se accede a la pantalla para editar la misma (se muestra la información de la entidad y existe un botón para activar el “modo edición”). En caso de considerarse necesario, se agrega un campo de búsqueda por nombre de la entidad en cuestión.

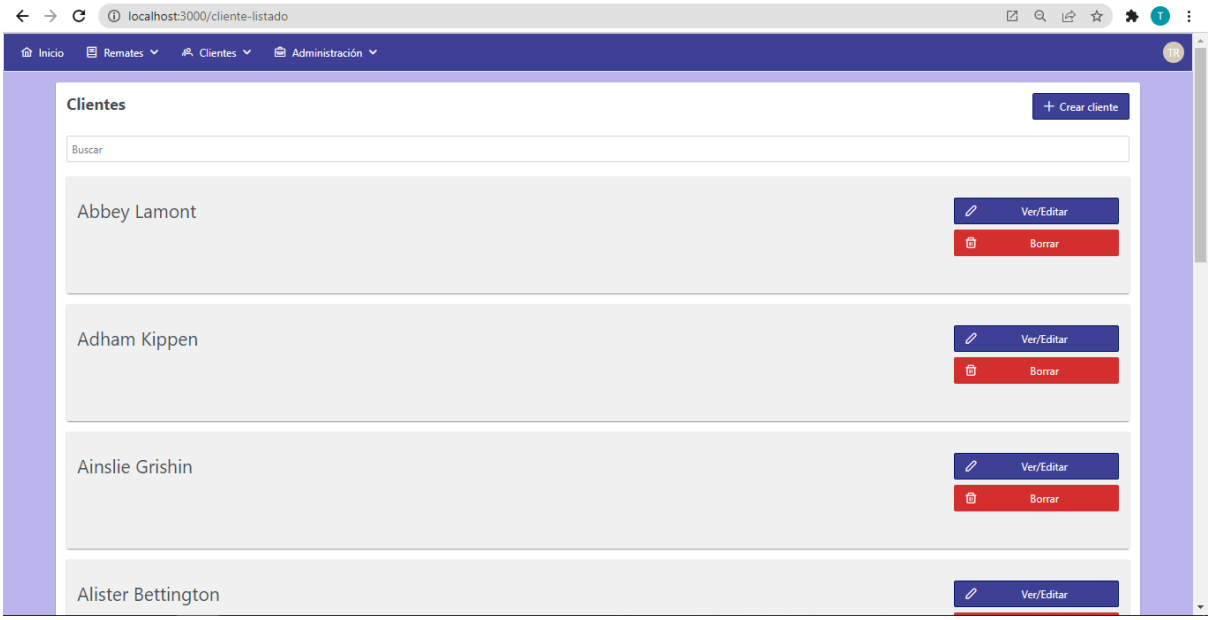


Figura 14: Pantalla listado de clientes

4.5.6 Gestión de remate

La realización de un remate puede ser considerada como la actividad fundamental dentro de nuestro sistema, por lo que merece un tratamiento más detallado.

Una vez que un remate fue creado, se puede acceder a una pantalla de gestión que está compuesta por varias pestañas o “tabs”:

1) En la primera pestaña se listan los lotes para la venta que fueron cargados como se comentó en el punto anterior. Cada tarjeta representa un conjunto de animales de la misma categoría (Animales en pista) y contiene una serie de botones para cada acción particular dentro de la venta, las cuales se detallarán en secciones posteriores.

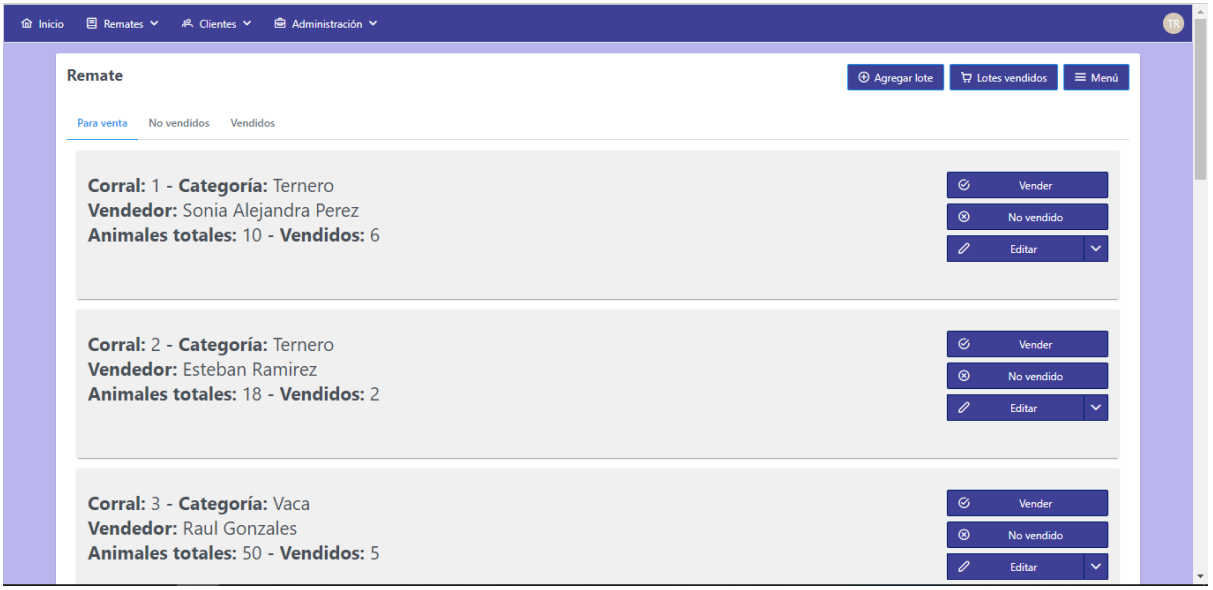


Figura 15: Pantalla de lotes para venta dentro de un remate

2) La segunda pestaña consiste en un listado de lotes provenientes de la primera pestaña que fueron marcados como “no vendidos”, ya sea porque el lote quedó sin venderse en su totalidad, o por haberse vendido parcialmente, quedando entonces animales sin ser comprados. Esto sirve para modelar la situación en que los animales salen de la pista porque nadie los compró y se sigue con el siguiente grupo de animales. El objetivo de esta sección es posibilitar la venta de estos animales luego de finalizado el remate (situación que sucede con cierta regularidad), por ello poseen un botón que dispara esa acción.

Luego, cada tarjeta dentro de este listado también presenta una serie de botones con acciones que serán descritas en secciones posteriores.

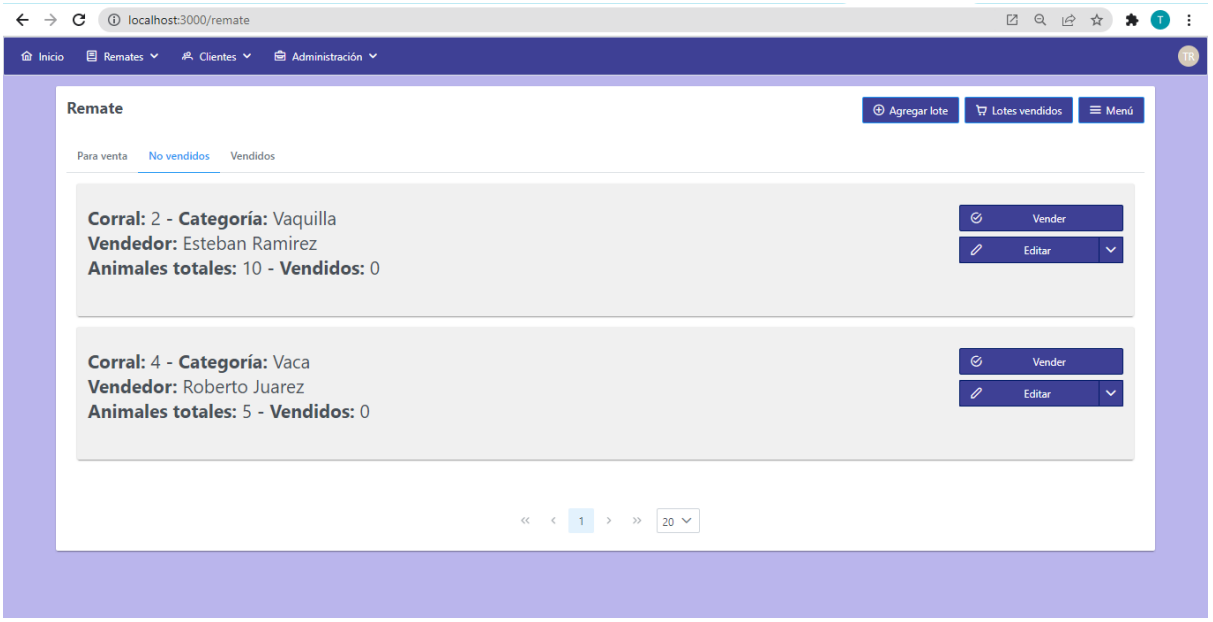


Figura 16: Pantalla de lotes para venta no vendidos dentro de un remate

3) La tercera y última pestaña consiste de una lista de aquellos lotes provenientes de la primera pestaña que fueron vendidos en su totalidad. En este caso, también se presentan varias opciones en cada tarjeta para realizar principalmente modificaciones de dicho lote.

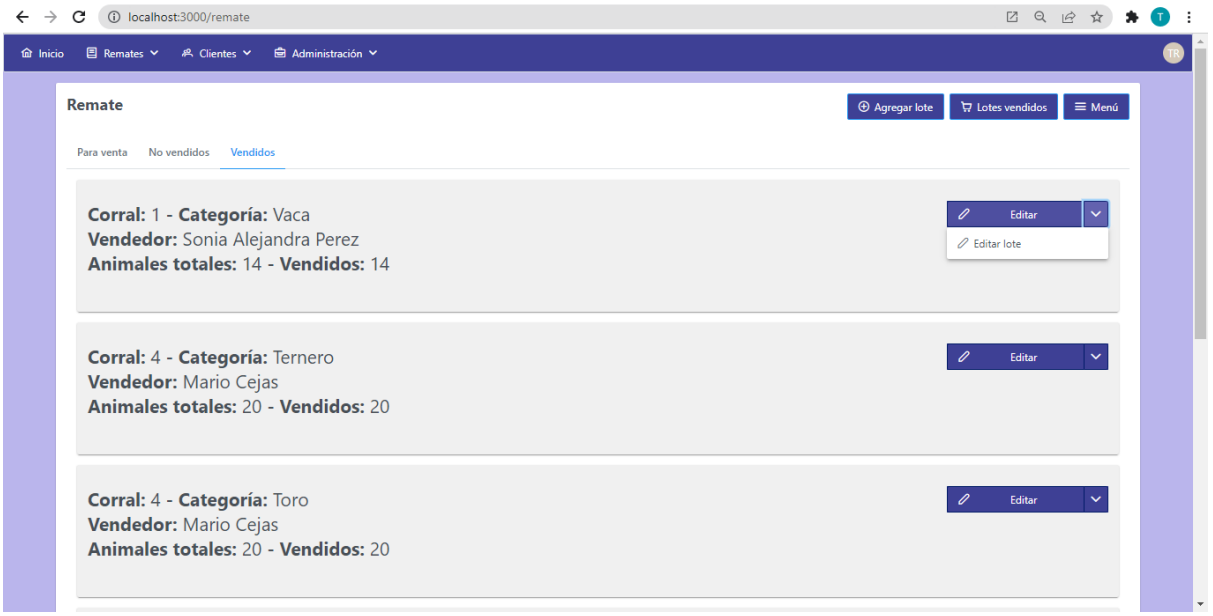


Figura 17: Pantalla de lotes para venta vendidos dentro de un remate

Por otra parte, existe una pantalla especialmente diseñada para mostrar los conjuntos de animales que van siendo vendidos (Lote Vendido). En este listado, como los casos anteriores, cada tarjeta presenta una serie de botones para realizar diferentes acciones con dicha entidad, las cuales serán explicadas con detalle en secciones posteriores.

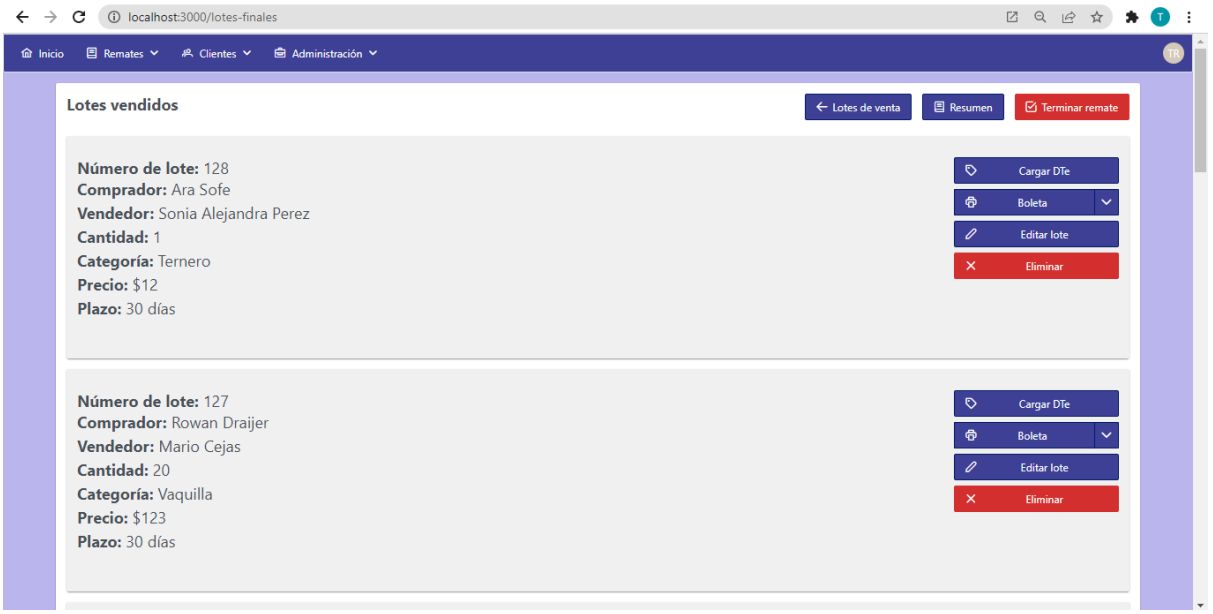


Figura 18: Pantalla de lotes vendidos dentro de un remate

4.5.7 Generación de boleta de venta

Si bien este punto podría estar incluido dentro del punto anterior, merece una breve descripción ya que fue una de las primeras solicitudes que tuvimos del cliente y de los principales problemas detectados por la utilización de grandes volúmenes de papel. En el listado de Lotes Vendidos cada tarjeta de un lote tiene botones dedicados para que se pueda generar la boleta de venta respectiva (siempre y cuando los datos del lote vendido estén completos). En la misma se detalla la información sobre la venta realizada y puede ser descargada o impresa (según el botón) para ser entregada a quienes corresponda. Posteriormente se detallará y mostrará esta funcionalidad.

4.5.8 Generación de reportes

En esta sección se presenta una pantalla con resúmenes estadísticos sobre la ejecución de un remate. Si bien en siguientes secciones ahondaremos en los detalles, estos reportes poseen la siguiente estructura:

- Información general del remate: lugar y fecha de realización, cantidad de animales ingresados, cantidad de animales vendidos, cantidad de compradores, cantidad de vendedores, dinero en movimiento, entre otros datos.
- Información por categoría de animales: se decidió que también haya un resumen por cada categoría de animales vendida, con el objetivo de ayudar a los organizadores en la toma de decisiones y que tengan un primer panorama de las tendencias para cada tipo de animales que participa en el remate. Los datos son similares a la información general, pero agrupados por categorías. Por ejemplo cantidad de animales ofrecidos, cantidad de animales vendidos, cantidad de compradores, cantidad de vendedores, dinero en movimiento, entre otros datos.

Por otro lado, para facilitar la comprensión de los usuarios, en la pantalla de resúmenes estadísticos también se puede optar por observar cierta información en formato de gráficos. Además, se puede generar un archivo en formato PDF de los reportes para que sea almacenado y/o compartido por los usuarios.

4.5.9 Diálogos de confirmación

Para aquellas acciones consideradas como de mayor criticidad (eliminar, crear, editar cierta entidad, entre otras) se muestra una ventana emergente de confirmación de dicha acción.

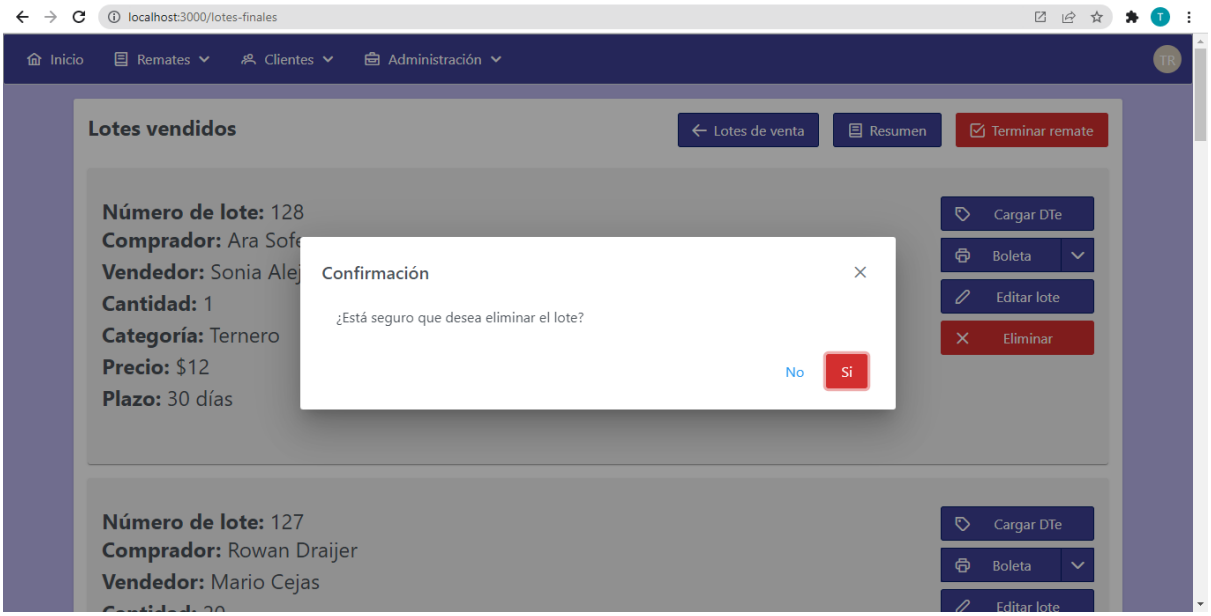


Figura 19: Diálogo de confirmación de eliminación de lote

4.6 Arquitectura utilizada

El equipo decidió organizar el desarrollo de la aplicación web en una **arquitectura de n niveles**. Esta es una arquitectura cliente-servidor en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario. En dichas arquitecturas a cada nivel se le confía una misión simple, es decir responsabilidades bien separadas, lo que permite el diseño de arquitecturas escalables (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).[10] Entre las ventajas más destacadas de este tipo de arquitecturas podemos destacar:

- Permite desacoplar la lógica reduciendo el impacto de los cambios dentro de un nivel en el resto.
- Portabilidad entre la nube y entornos locales, y entre plataformas en la nube: muy importante para las tendencias de desarrollo “cloud” actuales.
- Menor curva de aprendizaje para la mayoría de los desarrolladores: cada uno se puede enfocar en un nivel.
- Abiertas a entornos heterogéneos (Windows o Linux).

A continuación mostraremos un gráfico de la estructura general que tiene nuestra solución utilizando esta arquitectura:

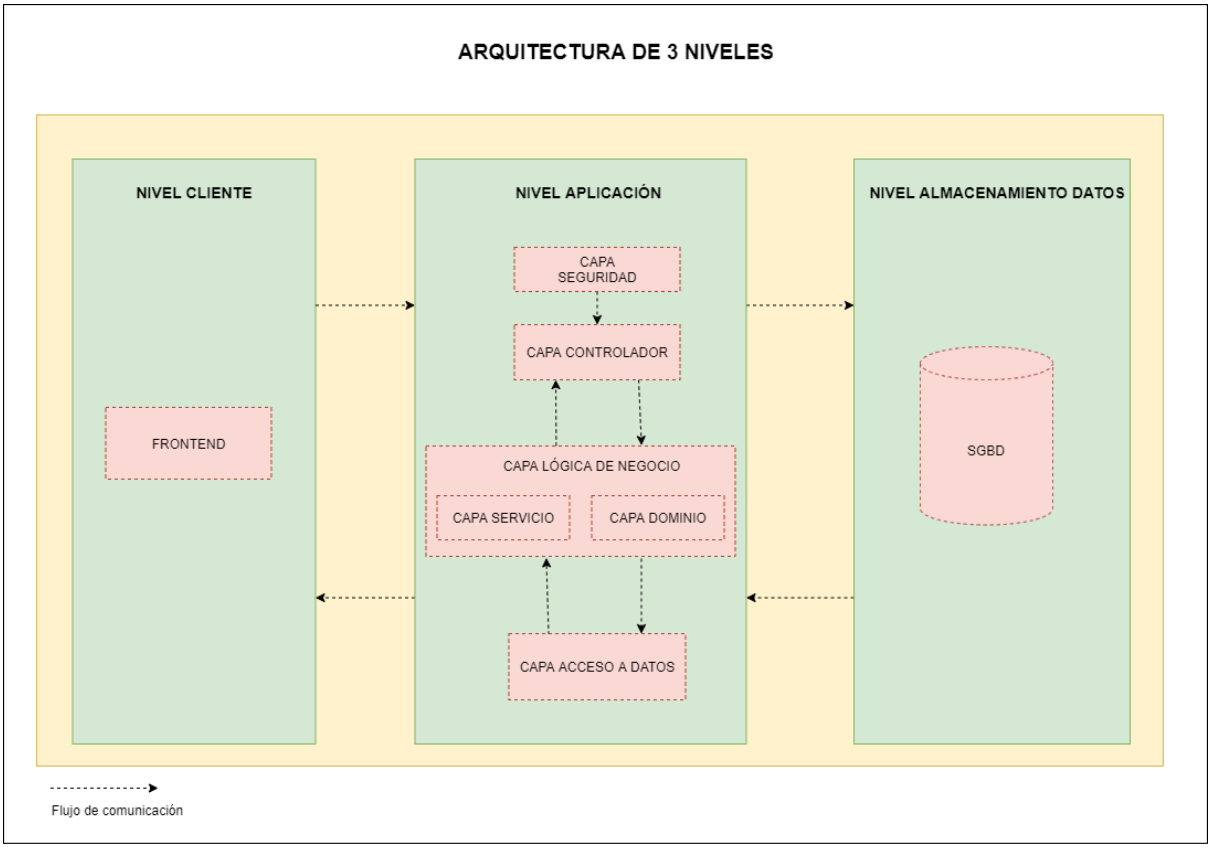


Figura 20: Diagrama de arquitectura de 3 niveles de la aplicación

Como muestra el diagrama anterior, nuestra aplicación tiene una arquitectura de 3 niveles bien definidos y que a continuación serán comentados.

4.6.1 Nivel 1 - Cliente

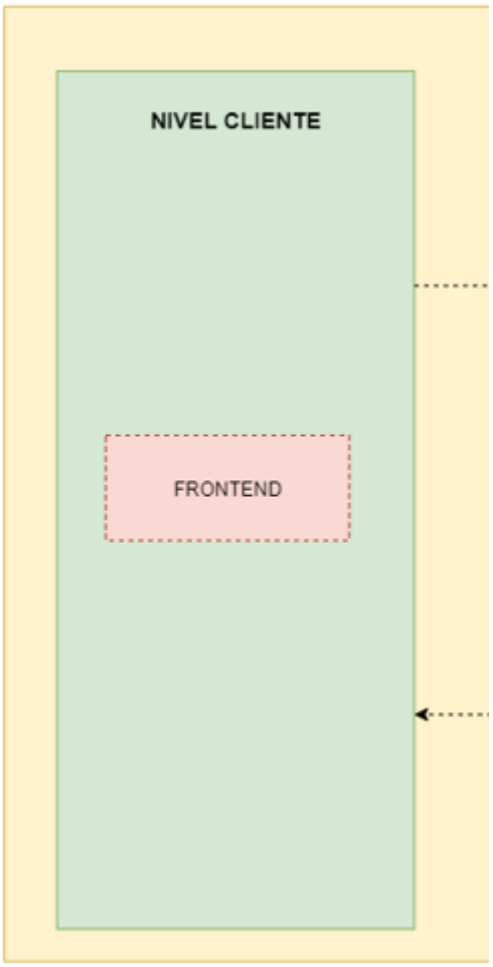


Figura 21: Nivel 1 - Cliente

Este nivel es el responsable del renderizado de la interfaz de usuario (UI - User Interface) y se encarga de la interacción directa con el mismo, es lo que denominamos Front End. Este código se ejecuta del lado de la máquina del usuario (dispositivos móviles, tablets, computadoras portátiles, computadoras de escritorio, etc.) y se comunica con los demás niveles a través de peticiones HTTP (para solicitar recursos), en general, vía Internet. Para implementar este nivel utilizamos la librería React JS, enfocándonos en el diseño de una SPA[3], y la comunicación con los demás niveles la realizamos a través de la tecnología Axios.

[3] Una Single-Page Application (SPA) es un tipo de aplicación web que ejecuta todo su contenido en una sola página. Funciona cargando el contenido HTML, CSS y JavaScript por completo al abrir la web. Al ir pasando de una sección a otra, solo necesita cargar el contenido nuevo de forma dinámica si este lo requiere, pero no hace falta cargar la página por completo. Esto mejora los tiempos de respuesta y agiliza mucho la navegación, favoreciendo así a la experiencia de usuario.[11]

4.6.2 Nivel 2 - Aplicación

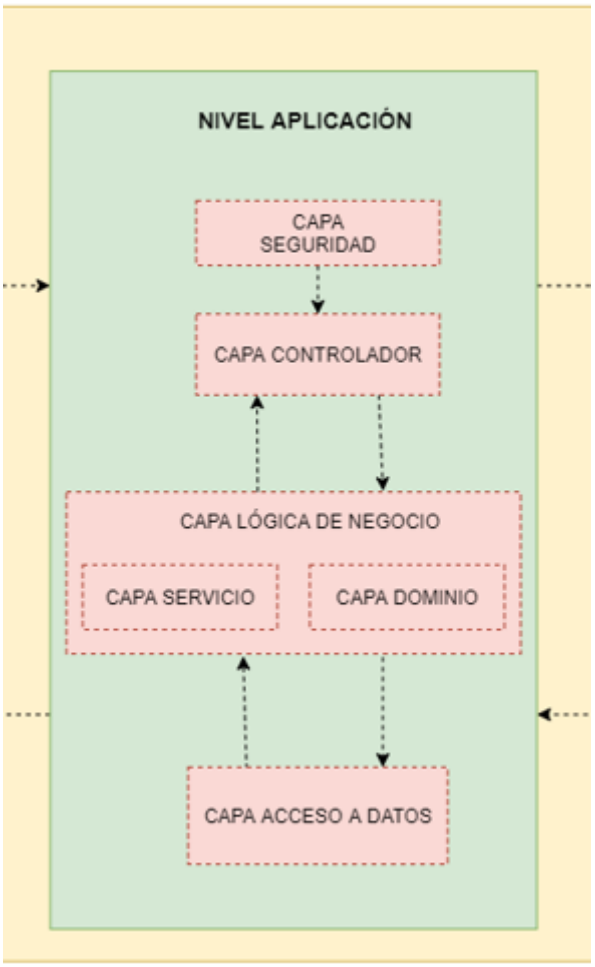


Figura 22: Nivel 2 - Aplicación

Este nivel es el responsable de un conjunto grande y complejo de tareas. Para implementarlo se utilizó código Java, más precisamente el framework Spring, y el mismo se encarga principalmente de:

- procesar entradas del usuario desde el front end,
- aplicar reglas del negocio,
- funcionalidades que garanticen la seguridad,
- y acceder a los datos almacenados.

Como se puede apreciar cada uno de los ítems anteriores presenta una complejidad muy grande en sí misma. Es por eso que, gracias a las facilidades que presenta la herramienta Spring Framework, dividimos este nivel en varias capas. Esta división presenta las mismas ventajas que las nombradas para la arquitectura de n niveles, principalmente, desacoplar la lógica y que los cambios en uno de los componentes (o capas) tengan el menor impacto posible para los demás. A continuación se describen brevemente cada una de estas capas.

4.6.2.1 Capa de seguridad

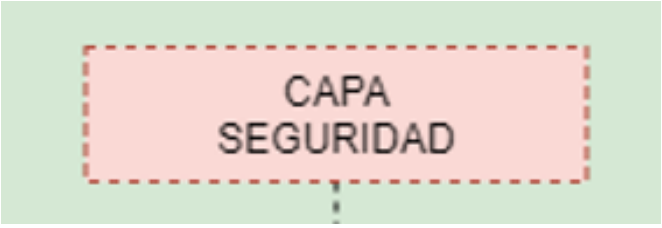


Figura 23: Capa seguridad

En esta capa nos enfocamos en 2 tareas principales:

- *Gestión de autenticación:* para la autenticación de un usuario se utilizó un mecanismo basado en JWT[4], con la información proveniente del login del sitio web (nombre de usuario y contraseña). En caso de una autenticación exitosa, se devuelve un token web JSON para mantener la información de sesión almacenada.
- *Gestión de autorización:* los recursos de la aplicación están asegurados a través del JWT. Una vez que el usuario inició sesión correctamente, cada petición de recursos que realice deberá incluir dicha información en un *header* específico. La autorización para que un usuario acceda a un recurso depende, fundamentalmente, del rol que posea el mismo y el servidor se encargará de constatar que los datos que llegan son aptos (o no) para entregar la información solicitada.

4.6.2.2 Capa de controladores

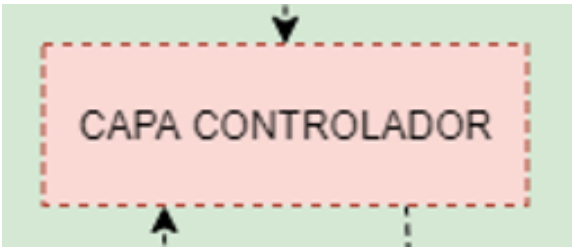


Figura 24: Capa controlador

Esta capa está compuesta por un conjunto de clases java que se encargan de recibir, validar y convertir la entrada del usuario y entregar una respuesta apropiada. Para aplicaciones basadas en servicios web, la capa API Rest reside aquí. En otras palabras, en estas clases se declaran los “endpoint” o URLs que sirven de conexión o comunicación entre el nivel 1 y el nivel 2. El nivel 1 o cliente solicita recursos a través de estos endpoints

[4] JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto JSON. Esta información se puede verificar y confiar porque está firmada digitalmente.[12]

vía HTTP y la capa de controladores del nivel 2, recibe la solicitud y devuelve la respuesta apropiada.

Ejemplo de implementación de un endpoint en nuestro proyecto:

```
@RestController
@RequestMapping("/api/sold-batch")
public class SoldBatchRest {

   @Autowired
   private SoldBatchService soldBatchService;

    @PostMapping("/{animalsOnGroundId}")
    ResponseEntity<?> createSoldBatch(@PathVariable Integer animalsOnGroundId,
@RequestBody SoldBatch newSoldBatch){
        try {
            return ResponseEntity.ok(soldBatchService.saveSoldBatch(newSoldBatch,
animalsOnGroundId));
        } catch (AnimalsOnGroundNotFound | BatchNotFoundException |
AuctionNotFoundException e) {
            logger.error(e.getMessage());
            return new ResponseEntity<>(new ErrorResponse(e.getMessage()),
HttpStatus.NOT_FOUND);
        } catch (IllegalArgumentException e){
            logger.error(e.getMessage());
            return new ResponseEntity<>(new ErrorResponse(e.getMessage()),
HttpStatus.BAD_REQUEST);
        } catch (HttpForbidenException e) {
            logger.error(e.getMessage());
            return new ResponseEntity<>(new ErrorResponse(e.getMessage()),
HttpStatus.FORBIDDEN);
        } catch (HttpUnauthorizedException e) {
            logger.error(e.getMessage());
            return new ResponseEntity<>(new ErrorResponse(e.getMessage()),
HttpStatus.UNAUTHORIZED);
        } catch (Exception e) {
            e.printStackTrace();
            logger.error(e.getMessage());
            return new ResponseEntity<>(new ErrorResponse(e.getMessage()),
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
...
}
```

4.6.2.3 Capa de lógica de negocio

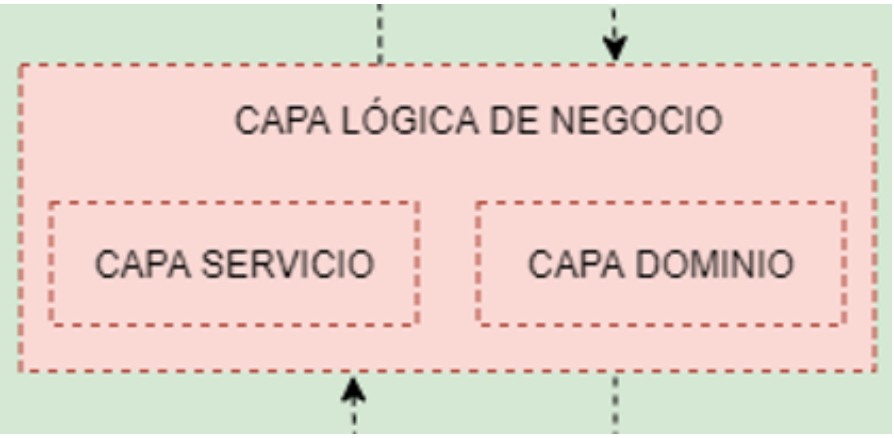


Figura 25: Capa lógica de negocio

Esta capa es fundamental dentro de la arquitectura propuesta, ya que la misma contiene:

- Todas las clases necesarias para modelar la lógica y reglas de negocios del problema, denominada **capa de servicios**. Las peticiones de la capa anterior llegan a esta capa, donde se validan, se procesan y operan modificando o devolviendo información del sistema.
- Todas las clases que modelan el **dominio** del problema, es decir, representan entidades del mundo real (como clientes, animales, remates, etc.). Estas en general, tienen una correspondencia “one to one” con la entidad respectiva en la base de datos, exceptuando aquellas entidades que tienen relaciones “many to many”, donde se genera una tabla adicional en la base de datos.

4.6.2.4 Capa de acceso a datos

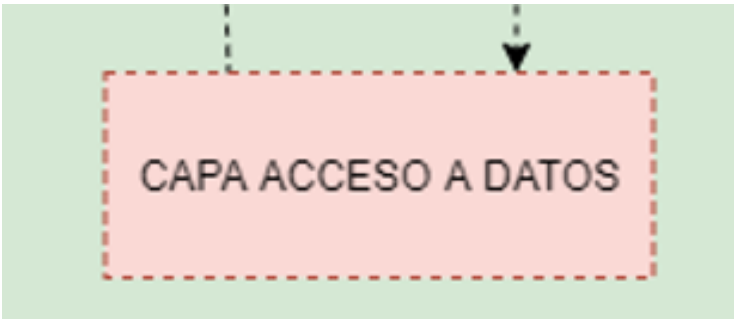


Figura 26: Capa acceso a datos

La responsabilidad de esta capa es almacenar de forma persistente los datos que se encuentran en memoria. Es decir, contiene la lógica para conectarse a un servidor de base de datos y ejecutar sentencias que garanticen la realización exitosa de las transacciones.

En nuestro caso, la implementación de esta capa se llevó a cabo bajo el patrón de diseño DAO (Data Access Object). El **patrón Data Access Object (DAO)** pretende principalmente independizar nuestra aplicación de la forma de acceder a la base de datos, o cualquier otro tipo de repositorio de datos. Para ello se centraliza el código relativo al acceso al repositorio de datos en las clases llamadas DAO. Fuera de las clases DAO no debe haber ningún tipo de código que acceda al repositorio de datos.

Ejemplo de la implementación de un objeto que gestiona el acceso a la tabla *auction* en la base de datos MySQL, mediante *Java Persistence API (JPA)*:

```
@Repository
public interface AuctionDAO extends JpaRepository<Auction, Integer> {
   Page<Auction> findByDeletedNullOrDeletedFalse(Pageable of);

   List<Auction> findByDeletedNullOrDeletedFalse();

   @Query(value = ("SELECT a FROM Auction a JOIN a.users u WHERE u.id = :userId " +
           "AND (a.finished IS NULL OR a.finished IS FALSE) " +
           "AND (a.deleted IS NULL OR a.deleted IS FALSE) " +
           "ORDER BY a.date ASC"))
   Page<Auction> findOwnById(Integer userId, Pageable of);

   @Query(value = ("SELECT DISTINCT a FROM Auction a " +
           "WHERE a NOT IN " +
           "(SELECT DISTINCT a1 FROM Auction a1 JOIN a1.users u " +
           " WHERE :userId = u.id AND (a1.finished IS NULL OR a1.finished IS FALSE) " +
           " AND (a1.deleted IS NULL OR a1.deleted IS FALSE) " +
           " AND (a1.deleted IS NULL OR a1.deleted IS FALSE)) " +
           " AND (a.finished IS NULL OR a.finished IS FALSE) " +
           " AND (a.deleted IS NULL OR a.deleted IS FALSE) " +
           "ORDER BY a.date ASC"))
   Page<Auction> findOthersById(Integer userId, Pageable of);

   @Query(value = ("SELECT a FROM Auction a WHERE " +
           "(a.finished IS NULL OR a.finished IS FALSE) " +
           "AND (a.deleted IS NULL OR a.deleted IS FALSE) " +
           "ORDER BY a.date ASC"))
   Page<Auction> findAllAdmin(Pageable of);

   Optional<Auction> findByIdAndDeletedNullOrDeletedFalse(Integer id);

   @Query(value = ("SELECT a FROM Auction a JOIN a.users u WHERE " +
           " u.id = :userId " +
           "AND a.finished IS TRUE" +
           " AND a.date BETWEEN :since AND :until " +
           "AND (a.deleted IS NULL OR a.deleted IS FALSE) " +
           "ORDER BY a.date DESC"))
   Page<Auction> findByFinishedAndBetween(Integer userId, Instant since, Instant until,
Pageable of);

   @Query(value = ("SELECT a FROM Auction a WHERE " +
           " a.finished IS TRUE " +
           " AND a.date BETWEEN :since AND :until " +
           "AND (a.deleted IS NULL OR a.deleted IS FALSE) " +
```

```
            "ORDER BY a.date DESC"))
    Page<Auction> findByFinishedAndBetween(Instant since, Instant until, Pageable of);
}
```

Como se puede apreciar, este nivel es de una complejidad considerable, sin embargo, con Java, a través de su mecanismo de paquetes, y Spring Framework, a través de su mecanismo de uso de etiquetas, se puede organizar de manera relativamente sencilla todo este código y reduce considerablemente las dificultades para su implementación.

4.6.3 Nivel 3 - Almacenamiento de datos

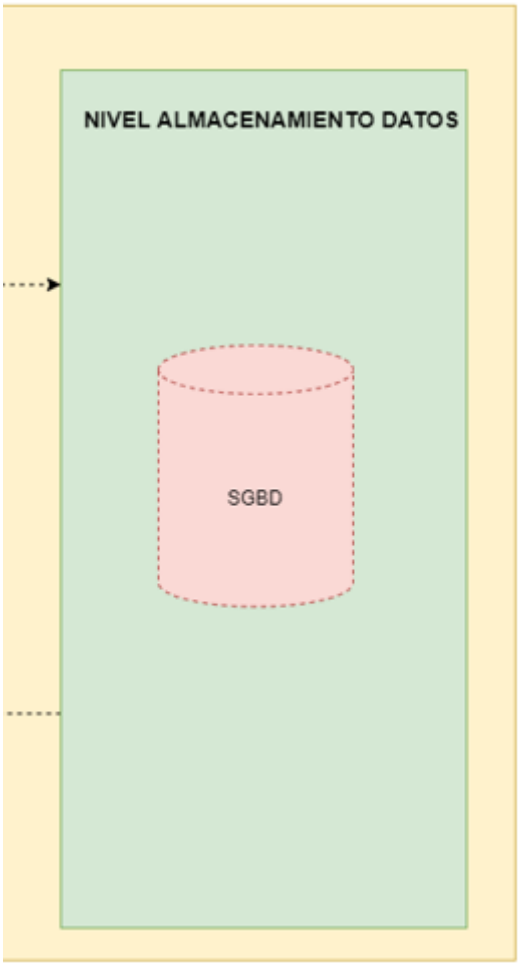


Figura 27: Nivel 3 - Almacenamiento de datos

Este último nivel de la arquitectura junto con el nivel 2 conforman lo que se denomina el Back End de la aplicación web y hace referencia a cualquier tipo de almacenamiento persistente de datos, ya sea, una base de datos relacional, una base de datos no relacional, el FileSystem, etc. En nuestro caso particular, optamos por la utilización de la base de datos relacional MySQL, cómo se describió en secciones anteriores.

4.7 Ambiente de desarrollo de software, tecnologías y plataformas

Una vez terminados estos primeros pasos, nos dispusimos a seleccionar las tecnologías, herramientas y ambientes de trabajo en los que llevar adelante el desarrollo. Los mismos son expuestos a continuación, junto con una breve justificación de su elección.

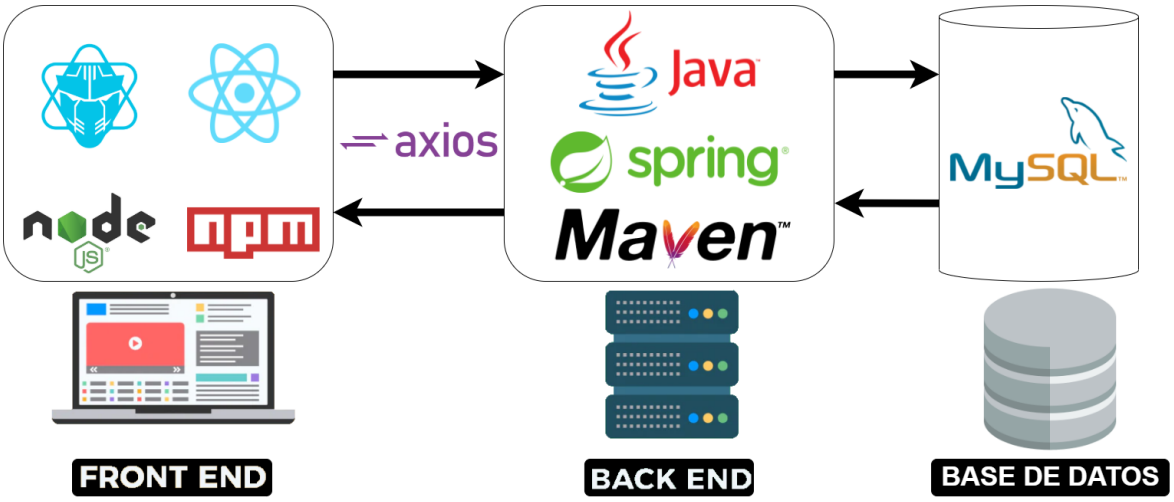


Figura 28: Diagrama de tecnologías utilizadas

4.7.1 Frontend

Para el desarrollo en este apartado hemos optado por el uso de las siguientes tecnologías:

React JS: se trata de una librería open-source para la construcción de interfaces de usuario y sus componentes. Nos hemos decantado por esta alternativa al ser la más usada a nivel comercial y por nuestra experiencia con la misma.

Node.js y npm: Node.js es un entorno de ejecución que permite ejecutar javascript fuera del navegador web, el cual es necesario para la creación, desarrollo y ejecución del proyecto de React. Por otro lado, npm (Node package manager), como su nombre lo dice, es el gestor de paquetes de Node, el cual resulta de gran utilidad a la hora de incorporar librerías al proyecto.

Axios: se trata de un cliente HTTP basado en promesas, que cumple las funciones de realizar peticiones al backend, parecido al “fetch” nativo de javascript, pero con ciertas facilidades de uso y generando un código más legible y limpio.

PrimeReact: PrimeFaces es una librería open-source de componentes para aplicaciones web, contando con adaptaciones tanto para Java como para las principales librerías web como Angular (con PrimeNG), Vue (con PrimeVue) y, la que usaremos en nuestro caso, PrimeReact, para el framework seleccionado. En nuestro caso hicimos uso de gran parte de sus componentes, como botones, campos de texto, calendarios, etc.

4.7.2 Backend

Pasando ahora al backend, las tecnologías seleccionadas para este propósito son las nombradas a continuación:

Java: Java es un lenguaje de programación de alto nivel orientado a objetos muy conocido y usado, con el cual hemos trabajado con anterioridad y contamos con cierta experiencia, motivo por el cual lo hemos elegido para ser la base del backend de este proyecto.

Maven: Junto a Java utilizaremos Maven para todo lo que es la gestión e instalación de dependencias/librerías en el backend, ya que facilita inmensamente este trabajo, debiendo los desarrolladores limitarse a escribir unas pocas líneas en un archivo pom.xml para tener todo listo para comenzar.

Spring Framework: Spring es uno de los frameworks más conocidos para Java, dando grandes facilidades en el desarrollo, como una herramienta para iniciar proyectos de forma rápida y efectiva (Spring Boot), herramientas para testing, acceso a datos con JPA Repository, facilidad para con los servicios web REST, inversión de control, entre muchas ventajas. Todo lo anterior, sumado a que también hemos trabajado previamente con este framework, son motivos más que suficientes para sumarlo a nuestro desarrollo.

MySQL: Siendo una de las bases de datos SQL más conocidas, nos hemos decidido por el uso de MySQL debido a la experiencia previa que tenemos de otros proyectos, al hecho de ser open-source, por contar con extensa documentación y facilidades en su integración con Spring y Java.

4.7.3 Entorno de desarrollo

Sistema operativo: **Windows 10**, simplemente por ser el sistema que ambos integrantes tenemos instalados y usamos normalmente.

IDEs: **Intellij IDEA** para backend y **Visual Studio Code** para frontend, ambos ampliamente usados en la industria y con los que tenemos familiaridad en su uso.

Sistema de control de versiones: Utilizamos Git para el control de versiones, donde la forma de trabajo fue crear una rama “develop”, que actuó como rama principal de cada proyecto, de la cual fuimos creando ramas auxiliares por cada una de las funcionalidades que agregamos para, al finalizar el trabajo, realizar un merge de nuevo a develop, donde posiblemente se deban resolver conflictos. Para el hosting de estos repositorios seleccionamos **Github**, al ser de uso libre y por tener experiencia en el mismo.

API Client: Con el objetivo de realizar prueba a la API, escogimos **Insomnia** como cliente para este propósito por tener versión gratuita y por haber sido usado con anterioridad por los integrantes del equipo.

4.7.4 Documentación y modelado

Gestión de incidencias y tareas/historias de usuario: **Jira** fue la herramienta seleccionada en este ámbito ya que provee las herramientas necesarias para la gestión ágil de proyectos, permitiendo crear un backlog del mismo, crear historias, task dentro de las mismas, y organizandolo todo por Sprints, a los cuales se les puede fijar su fecha de inicio y de fin. No contábamos con experiencia en el uso de este tipo de software, por lo que nos inclinamos por Jira por el solo hecho de ser uno de los más usados y por su (aparente) facilidad de uso. Además, en nuestro caso nos limitamos a crear los Sprints correspondientes con sus historias y las “tasks” que estas conllevan y, a medida que avanzaba el desarrollo, ubicar estas historias en el estado que los corresponde, siendo estos “To Do”, “In Progress”, “For Testing” y “Done”.

Modelado: utilizamos **Figma** para maquetar las interfaces de usuario, las cuales nos fueron útiles como guía luego a la hora del desarrollo frontend, y **draw.io** para diagramas en general, como diagramas de clases o entidad-relación, por nombrar algunos.

Gestión de la documentación: **Google Drive** y su suite ofimática ofrecen facilidades para trabajar en paralelo, permitiendo a los usuarios editar documentos al mismo tiempo, ver los cambios que dejaron los demás, dejar comentarios y hasta tener un seguimiento de versiones del documento, por lo que nos parece las mejor opción para trabajar sobre la documentación del este proyecto.

4.8 Seguridad en la aplicación

La Seguridad de la Información, según ISO 27001, se refiere a la **confidencialidad**, la **integridad** y la **disponibilidad** de la información y los datos importantes para la organización. Consiste en asegurar que los recursos del Sistema de Información de una empresa se utilicen de la forma que ha sido decidido y el acceso de información se encuentra contenida, así como controlar que la modificación sólo sea posible por parte de las personas autorizadas para tal fin y por supuesto, siempre dentro de los límites de la autorización.

- **Confidencialidad**: es la propiedad de prevenir que se divulgue la información a personas o sistemas no autorizados.
- **Integridad**: es la propiedad que busca proteger que se modifiquen los datos libres de forma no autorizada.
- **Disponibilidad**: es una característica, cualidad o condición de la información que se encuentra a disposición de quien tiene que acceder a ésta, bien sean personas, procesos o aplicaciones.[13]

Este apartado está orientado a explicar y mostrar qué aspectos fueron tenidos en cuenta para garantizar la seguridad de la aplicación en general, sobre todo en el acceso a los datos que la misma gestiona.

Desde un punto de vista general, la lógica general de seguridad en la plataforma se basó en el concepto de **roles** de un usuario. Es decir, un usuario puede acceder o modificar determinada información si cuenta con un rol que le permita realizar esta acción. En cuanto a éstos, un usuario solo puede contar con un único rol de los siguientes: Asistente, Consignatario o Administrador. Así, cada uno de los roles tiene mayores privilegios (los privilegios del rol anterior y algunos adicionales) en el orden que fueron nombrados anteriormente.

Entonces, al ser registrado en el sistema, a un usuario se le asigna, entre otras, la siguiente información que es relevante para el tema en cuestión: nombre de usuario, contraseña y rol. Finalmente, el mecanismo de comunicación que se utilizó para asegurar la aplicación, poder autenticar y autorizar la solicitud de recursos fue la implementación de un JSON Web Token (JWT).

4.8.1 JSON Web Token (JWT)

JSON Web Token (JWT)[14] es un estándar abierto que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto

JSON. Esta información se puede verificar y confiar porque está firmada digitalmente. Estos tokens se pueden firmar usando un secreto (con el algoritmo HMAC) o un par de claves pública/privada usando RSA o ECDSA . Los tokens firmados pueden verificar la integridad de los reclamos contenidos en ellos.

En su forma compacta, los tokens web JSON constan de tres partes separadas por puntos (.), que son:

- **Encabezamiento (x):** generalmente consta de dos partes, el tipo de token (que es JWT) y el algoritmo de firma que se utiliza (como HMAC SHA256 o RSA). Luego, este JSON está codificado en Base64Url para formar la primera parte del JWT.
- **Carga útil (y):** contiene las reclamaciones. Las reclamaciones son declaraciones sobre una entidad (normalmente, el usuario) y datos adicionales. Hay tres tipos de reclamos: registrados, públicos y privados. Luego, la carga útil se codifica en Base64Url para formar la segunda parte del token web JSON.
    - *Reclamos registrados:* se trata de un conjunto de reclamos predefinidos que no son obligatorios pero se recomiendan para proporcionar un conjunto de reclamos útiles e interoperables. Algunos de ellos son: **iss** (emisor), **exp** (tiempo de caducidad), **sub** (sujeto), **aud** (audiencia), entre otros.
    - *Reclamos públicos:* estos pueden ser definidos a voluntad por aquellos que usan JWT. Pero para evitar colisiones, deben definirse en el Registro de tokens web JSON de IANA[15] o definirse como un URI que contenga un espacio de nombres resistente a colisiones.
    - *Reclamos privados:* son los reclamos personalizados creados para compartir información entre partes que acuerdan usarlos y no son reclamos registrados ni públicos.
- **Firma (z):** se usa para verificar que el mensaje no se modificó en el camino y, en el caso de tokens firmados con una clave privada, también puede verificar que el remitente del JWT es quien dice ser. Para crear la parte de la firma, debe tomar el encabezado codificado, la carga útil codificada, un secreto, el algoritmo especificado en el encabezado y firmarlo.

  Por lo tanto, un JWT normalmente tiene el siguiente aspecto:
  x*.y*.z*

El resultado son tres cadenas de URL Base64 separadas por puntos que se pueden pasar fácilmente en entornos HTML y HTTP, mientras que son más compactas en comparación con los estándares basados en XML, como SAML. Para finalizar, mostramos el código de generación y un ejemplo de un token utilizado dentro del sistema.

```
public class MyAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
    private static final Logger logger =
LoggerFactory.getLogger(MyAuthenticationFilter.class);
    private final AuthenticationManager authenticationManager;
    private String error = "";

    @Override
    public void successfulAuthentication(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain, Authentication authentication) throws
ServletException, IOException {

       //Si la autenticación fue exitosa, armamos el JWT que revolvemos a la app
cliente.
        Principal principal = (Principal) authentication.getPrincipal();
        User usuario = new User();
        usuario.setName(principal.getName());
        usuario.setLastname(principal.getLastname());
        usuario.setUsername(principal.getUsername());
        usuario.setId(principal.getId());

        ObjectMapper objectMapper = new ObjectMapper();

        String token = Jwts.builder()
                        .signWith(SecurityConstants.JWT_SECRET,
SignatureAlgorithm.HS512)
                        .setHeaderParam("typ", SecurityConstants.TOKEN_TYPE)
                        .setIssuer(SecurityConstants.TOKEN_ISSUER)
                        .setAudience(SecurityConstants.TOKEN_AUDIENCE)
                        .setSubject(usuario.getName() + ' ' + usuario.getLastname())
                        .claim("name", usuario.getName())
                        .claim("lastname", usuario.getLastname())
                        .claim("uid", usuario.getId())
                        .claim("username", usuario.getUsername())
                        .setExpiration(new Date(System.currentTimeMillis() + 86400000))
//un dia
                        .claim("rol", authentication.getAuthorities())
                        .compact();

        LinkedHashMap<String,String> map = new LinkedHashMap<>();
        map.put("access_token", token);

        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        response.getWriter().println(objectMapper.writeValueAsString(map));

        //Seguimos con los filtros, o terminamos en el REST.
        filterChain.doFilter(request, response);
    }
}
```

Figura 29: ejemplo de implementación de JWT con datos de un usuario del sistema

4.8.2 Front end

Desde el punto de vista del cliente, el usuario ingresa a la landing page, que es una pantalla de login. En la misma, se le solicita ingresar su nombre de usuario y contraseña. En caso de que las credenciales sean válidas, se recibe un JWT, que contiene información relacionada al usuario, entre otras, su rol. Este token, es almacenado por el cliente en su local storage (almacenamiento local). Entonces, utilizando este token del lado cliente, la vista de cada una de las pantallas se renderiza mostrando o permitiendo determinadas acciones según el rol que posee el mismo.

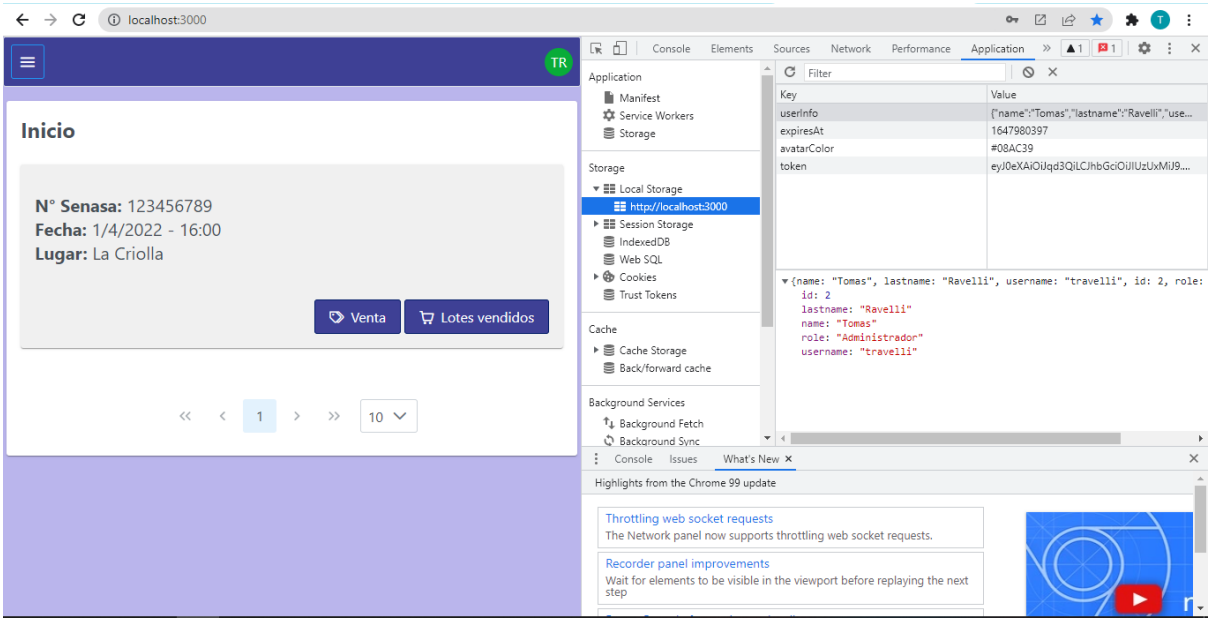


Figura 30: Almacenamiento local luego de la autenticación de un usuario

En la figura que se muestra a continuación, se compara la vista de la página principal de la aplicación según el usuario que ingresó tenga rol “Administrador” o rol “Asistente”. Se puede apreciar que, en la primera imagen, un usuario administrador tiene acceso desde el menú superior a las opciones de “Administración”, mientras que un usuario asistente no posee dicho acceso. Además, el listado de un asistente se divide en dos sub listados, según quiera ver los remates donde él mismo participa o los demas remates, mientras que para un administrador, al tener todos los privilegios, se le muestra un único listado porque tiene las facultades para acceder y/o modificar cualquier tipo de información en los mismos.

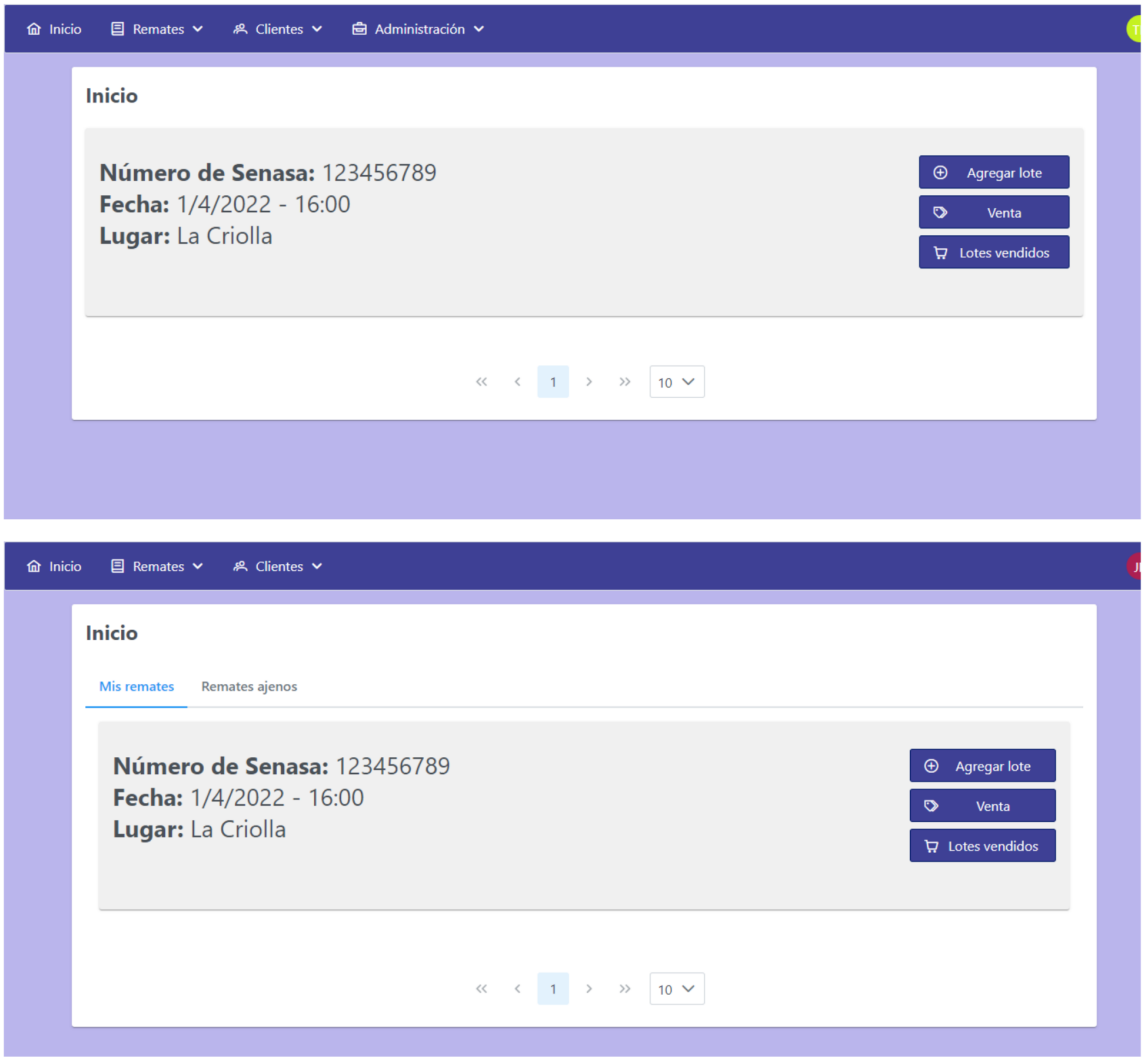


Figura 31: vista usuario “administrador” vs. vista usuario “asistente”

4.8.3 Back end

Desde el punto de vista del API Rest que se implementó para que la aplicación cliente consuma recursos, el aseguramiento de éste también recae, principalmente, en

verificar el rol del usuario que está solicitando un recurso. Entonces, para proteger los endpoints que se exponen al cliente de la aplicación, se escribieron un conjunto de reglas según:

- El recurso que se está solicitando (“path” o patrón de URL)
- La acción que se quiere realizar sobre el recurso.

Así, la primera coincidencia de una solicitud con el patrón de URL, activa la respectiva regla y se corrobora el rol del usuario para permitir o rechazar la petición. Vale aclarar que el único endpoint de acceso público es el que el usuario se autentica y obtiene sus credenciales.

La capa de seguridad del lado servidor, consiste en un proceso de 2 etapas bien definidas:

1. Obtener información de quien realiza la petición.
2. Validar que quien realiza la petición está autorizado a esto.

Para obtener información de quien realiza la petición, el usuario debe incluir un encabezado, denominado *Authorization*, en cada petición HTTP que vaya a realizar, cuyo valor es el JSON Web Token junto a un prefijo: “Bearer ”. Una vez recibida la solicitud, el servidor toma el valor de este encabezado, verifica que su estructura sea correcta, válida la firma del JWT y, en caso afirmativo, continúa al siguiente paso, si no, se retorna un error con código de respuesta HTTP 403.

```
public class MyAuthorizationFilter extends BasicAuthenticationFilter{

   private static final Logger logger = LoggerFactory.getLogger(MyAuthorizationFilter.class);
   private String error = "";
   public MyAuthorizationFilter(AuthenticationManager authenticationManager) {
       super(authenticationManager);
   }
   @Override
   protected void doFilterInternal(HttpServletRequest req,
                                   HttpServletResponse res,
                                   FilterChain chain) throws IOException, ServletException {
       String header = req.getHeader(SecurityConstants.TOKEN_HEADER);
       if(!req.getRequestURI().equals("/api/login")) {
           if(req.getRequestURI().contains("/websocket")){
               chain.doFilter(req, res);
           }
           else {
               if ((header == null || !header.startsWith(SecurityConstants.TOKEN_PREFIX))) {
                   error = "No se encontró token de acceso";
                   LinkedHashMap<String, String> map = new LinkedHashMap<>();
                   map.put("error", "Acceso denegado. " + error);
                   ObjectMapper objectMapper = new ObjectMapper();
                   error = objectMapper.writeValueAsString(map);
                   res.setContentType("application/json");
                   res.setStatus(HttpStatus.FORBIDDEN.value());
```

```
                    res.getWriter().println(error);
                    res.getWriter().flush();
//                  chain.doFilter(req, res);
                    return;
                }

                UsernamePasswordAuthenticationToken authentication = getAuthentication(req);
                if (authentication == null) {
                    LinkedHashMap<String, String> map = new LinkedHashMap<>();
                    map.put("error", "Acceso denegado. " + error);
                    ObjectMapper objectMapper = new ObjectMapper();
                    error = objectMapper.writeValueAsString(map);
                    res.setContentType("application/json");
                    res.setStatus(HttpStatus.FORBIDDEN.value());
                    res.getWriter().println(error);
                    res.getWriter().flush();
                } else {
                    SecurityContextHolder.getContext().setAuthentication(authentication);
                    chain.doFilter(req, res);
                }
            }
        }
    }

    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {
        String token = request.getHeader(SecurityConstants.TOKEN_HEADER);
        if (token != null && token.startsWith(SecurityConstants.TOKEN_PREFIX)) {
            // parse the token.
            try {
                Jws<Claims> parsedToken = Jwts.parserBuilder().
                                                setSigningKey(SecurityConstants.JWT_SECRET)
                                                .build()

.parseClaimsJws(token.replace(SecurityConstants.TOKEN_PREFIX,""));
                List<SimpleGrantedAuthority> authorities;
                if(parsedToken.getBody().get("rol")!=null){

                    authorities = ((List<?>) parsedToken.getBody().get("rol"))
                                                                .stream()
                                                                .map(authority -> new
SimpleGrantedAuthority((String) ((LinkedHashMap)authority).get("authority")))
                                                                .collect(Collectors.toList());
                    logger.debug(authorities.toString());
                    User usuario = new User();
                    //Armamos la información del usuario que luego se utilizara en SecurityConfig para
asegurar las rutas, por ejemplo, por roles.
                    if(parsedToken.getBody().get("name")!= null){
                        usuario.setName(parsedToken.getBody().get("name").toString());
                    }
                    if(parsedToken.getBody().get("lastname")!= null){
                        usuario.setLastname(parsedToken.getBody().get("lastname").toString());
                    }
                    if(parsedToken.getBody().get("username") != null){
                        usuario.setUsername(parsedToken.getBody().get("username").toString());
                    }
                    if(parsedToken.getBody().get("uid") != null){
```

```
                    usuario.setId(Integer.parseInt(parsedToken.getBody().get("uid").toString()));
                }
                logger.debug("Usuario obtenido desde JWT: " + usuario);
                return  new UsernamePasswordAuthenticationToken(usuario, null, authorities);
            }
        } catch (ExpiredJwtException exception){
            logger.warn("Request to parse expired JWT: {} failed: {}", token, exception.getMessage());
            error = "JWT expirado: " + token + " failed: " + exception.getMessage();
        } catch (UnsupportedJwtException exception){
            logger.warn("Request to parse unsupported JWT: {} failed: {}", token,
exception.getMessage());
            error = "JWT no soportado: " + token + " failed: " + exception.getMessage();
        } catch (MalformedJwtException exception){
            logger.warn("Request to parse malformed JWT: {} failed: {}", token,
exception.getMessage());
            error = "JWT mal formado: " + token + " failed: " + exception.getMessage();
        } catch (SignatureException exception){
            logger.warn("JWT with invalid signature: {} failed: {}", token, exception.getMessage());
            error = "Firma de JWT inválida: " + token + " failed: " + exception.getMessage();
        } catch (IllegalArgumentException exception){
            logger.warn("Request to parse empty or null JWT: {} failed: {}", token,
exception.getMessage());
            error = "JWT vacío o nulo: "+token+" failed: "+ exception.getMessage();
        } catch (Exception exception){
            logger.warn("Request to parse other error JWT: {} failed: {}", token,
exception.getMessage());
            error = "Error inesperado: "+token+" failed: "+ exception.getMessage();
        }
    }
    return null;
  }
}
```

En el segundo paso, se procesan las reglas de seguridad que diseñaron y escribieron los desarrolladores. Si ninguna de las reglas coincide con la petición (URL y método HTTP), se retorna un error con código de respuesta HTTP 404. Cuando la primera de las reglas coincide con la solicitud, se ejecuta la misma. Entonces, si el usuario tiene un rol cuya regla autoriza a continuar, se procede a las siguientes capas, sino, se retorna un error con código de respuesta HTTP 403.

```
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {
   private static final Logger logger = LoggerFactory.getLogger(SecurityConfig.class);

   @Override
   protected void configure( HttpSecurity http ) throws Exception {
       http.cors().and()
           .authorizeRequests()
           .antMatchers("/websocket/**").permitAll()
           .antMatchers(HttpMethod.GET,"/api/test").hasAuthority("Rol")
           .antMatchers(HttpMethod.GET,
```

```
"/api/locality/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers(HttpMethod.GET,
"/api/category/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers(HttpMethod.GET,
"/api/auction/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers(HttpMethod.GET,
"/api/client/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers( "/api/locality/**").hasAnyAuthority("Administrador","Consignatario")
            .antMatchers( "/api/category/**").hasAnyAuthority("Administrador","Consignatario")
            .antMatchers(
"/api/auction-user/history/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers( "/api/auction-user/assignment/**").hasAnyAuthority("Administrador",
"Consignatario")
            .antMatchers( "/api/auction-user/users/**").hasAnyAuthority("Administrador", "Consignatario")
            .antMatchers( "/api/auction-user/own/**").authenticated()
            .antMatchers( "/api/auction-user/others/**").authenticated()
            .antMatchers( "/api/auction-user/**").hasAnyAuthority("Administrador")

.antMatchers("/api/auction-batch/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers( "/api/auction/**").hasAnyAuthority("Administrador","Consignatario")

.antMatchers("/api/user/profile/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers("/api/user/user-list/**").hasAnyAuthority("Consignatario", "Administrador")
            .antMatchers("/api/user/**").hasAnyAuthority("Administrador")

.antMatchers("/api/sold-batch/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers("/api/pdf/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers("/api/report/**").hasAnyAuthority("Administrador","Consignatario","Asistente")
            .antMatchers(HttpMethod.POST,"/api/login").permitAll()
            .anyRequest().authenticated()
            .and()
                .addFilter(new MyAuthenticationFilter(authenticationManager()))//Filtro para los
autenticar los login.
                .addFilter(new MyAuthorizationFilter(authenticationManager()))//Filtro para autorizar las
demas peticiones
//            .csrf().csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse());
                .csrf().disable();
   }
```

4.9 Testing

La prueba de software es un elemento de un tema más amplio que usualmente se conoce como **verificación y validación (V&V)**. La verificación se refiere al conjunto de tareas que garantizan que el software implementa correctamente una función específica (¿Construimos el producto correctamente?). La validación es un conjunto diferente de tareas que aseguran que el software que se construye sigue los requerimientos del cliente (¿Construimos el producto correcto?).

Las pruebas representan el último bastión desde donde puede valorarse la calidad y, de manera más pragmática, descubrirse errores.

Para desarrollar software de computadoras, se avanza en espiral hacia adentro (contra las manecillas del reloj) a lo largo de una línea que reduce el nivel de abstracción en cada vuelta (ver figura 32). Una estrategia para probar el software también puede verse en el contexto de la espiral. La *prueba de unidad* comienza en el vértice de la espiral y se concentra en cada unidad (por ejemplo, componente, clase o un objeto de contenido de una webapp) del software como se implementó en el código fuente. La prueba avanza al moverse hacia afuera a lo largo de la espiral, hacia la *prueba de integración*, donde el enfoque se centra en el diseño y la construcción de la arquitectura del software. Al dar otra vuelta hacia afuera de la espiral, se encuentra la prueba de validación, donde los requerimientos establecidos como parte de su modelado se validan confrontándose con el software que se construyó. Finalmente, se llega a la *prueba del sistema*, donde el software y otros elementos del sistema se prueban como un todo. Para probar el software de cómputo, se avanza en espiral hacia afuera en dirección de las manecillas del reloj a lo largo de líneas que ensanchan el alcance de las pruebas con cada vuelta.[16]

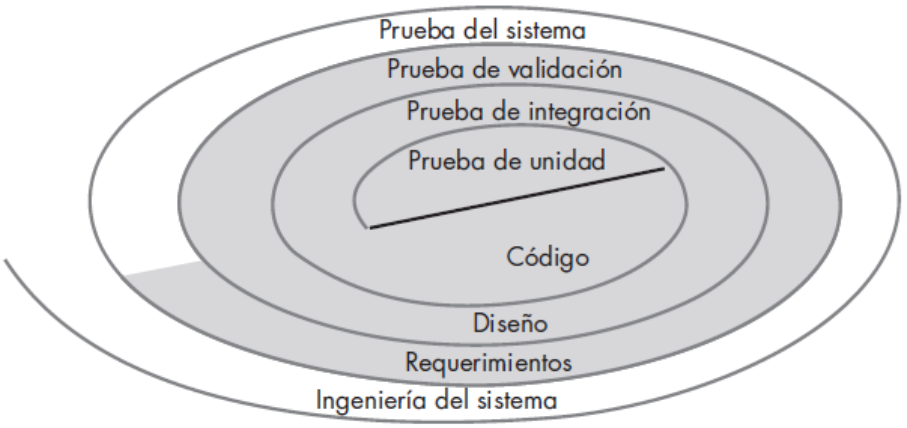


Figura 32: diseño de pruebas en espiral

A continuación se procede a describir cada una de las pruebas que se observan en el gráfico anterior y se detallará, según el caso, cómo fueron llevadas a cabo en el presente Proyecto Final de Carrera. Toda la documentación generada para este apartado será mostrada en el Anexo E.

4.9.1 Pruebas de unidad

La prueba de unidad enfoca los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente o módulo de software.[16]

En nuestro Proyecto nos enfocamos, principalmente, en ejecutar este tipo de pruebas en los métodos dentro de las clases de la *capa de servicio* en el back-end ya que estas clases eran las encargadas de gestionar toda la lógica de negocio y, por ende, la información más relevante del sistema. Las pruebas unitarias fueron diseñadas, implementadas y ejecutadas utilizando el framework de Java JUnit[5].

Básicamente, el diseño de las pruebas consistía en una tabla con tres columnas: Caso de prueba, Resultado esperado, Resultado obtenido. Las tablas eran construidas una por cada Clase de la capa de servicio que se deseaba y consideraba relevante probar (principalmente las operaciones CRUD en los primeros sprints). Luego, estos casos de pruebas eran escritos e implementados con el framework Junit para automatizar su ejecución. Se intentó, en general, mantener un patrón de escrituras de las pruebas como el que se muestra a continuación:

`accionResultadoEsperado` o `accionAnomalia`

```
@Test
   void createAuctionSuccesfully(){
       Auction auction1 = new Auction();
       auction1.setUsers(auction.getUsers());
       auction1.setDate(auction.getDate());
       auction1.setSenasaNumber(auction.getSenasaNumber());
       auction1.setLocality(auction.getLocality());
       auction1.setDeleted(auction.getDeleted());
       auction1.setFinished(auction.getFinished());
       auction1.setId(null);
       Auction auction2 = new Auction();
       auction2.setUsers(auction.getUsers());
       auction2.setDate(auction.getDate());
       auction2.setSenasaNumber(auction.getSenasaNumber());
       auction2.setLocality(auction.getLocality());
       auction2.setDeleted(auction.getDeleted());
       auction2.setFinished(auction.getFinished());
       auction2.setId(1);
       when(auctionDAO.save(any(Auction.class))).thenReturn(auction2);
//      auction1.setLocality(locality);
       AtomicReference<Auction> auctionSaved = new AtomicReference<>();
       assertDoesNotThrow(() -> auctionSaved.set(auctionService.saveAuction(auction1)));
       assertEquals(auctionSaved.get().getId(), 1);
//      assertDoesNotThrow(()->auctionService.getAuctionById(auctionSaved.getId()));
   }
```

```
@Test
void createAuctionWithInvalidDate(){
   auction.setDate(Instant.now().minus(Period.ofDays(10)));
   assertThrows(InvalidCredentialsException.class,
()->auctionService.saveAuction(auction));
}
```

[5] https://junit.org/junit5/

En estos casos, a todos los métodos o componentes auxiliares que eran necesarios para la ejecución del método a probar, por ejemplo, acceso a base de datos o llamadas desde la capa de controladores, se les simuló su comportamiento a partir de “mockups” para respetar el concepto y esencia de una prueba unitaria y no perder el foco de lo que estábamos testeando.

4.9.2 Pruebas de integración

Las *pruebas de integración* son una técnica sistemática para construir la arquitectura del software mientras se llevan a cabo pruebas para descubrir errores asociados con la interfaz. El objetivo es tomar los componentes probados de manera individual y construir una estructura de programa que se haya dictado por diseño.

En nuestro Proyecto, seguimos una estrategia de pruebas de integración de tipo **ascendente**. Esta prueba comienza con la construcción y la prueba con módulos atómicos (es decir, componentes en los niveles inferiores dentro de la estructura del programa) como se describió en el punto anterior. Puesto que los componentes se integran de abajo hacia arriba, la funcionalidad que proporcionan los componentes subordinados en determinado nivel siempre está disponible y se elimina la necesidad de representantes (stubs).[16]

Estas pruebas nos fueron de gran utilidad para testear la comunicación entre las distintas capas del back-end, sin la necesidad de simular el comportamiento de ninguno de los componentes (excepto la capa de seguridad que, a fines prácticos, se ignoró cuando la aplicación del lado del servidor se ejecutaba en modo “test”). Entonces, el diseño de las pruebas consistió en tomar la misma tabla y los mismos casos que se plantearon en las pruebas de unidad, sin embargo, ahora no se simula ningún componente y, en general, el resultado esperado era un código de respuesta HTTP. Para la implementación de estas pruebas también se utilizó el framework JUnit.

Luego, también nos interesó profundizar las pruebas de integración de la aplicación y agregar el módulo o capa de seguridad. Para esto, se ejecutaron un número importante de tests utilizando el software Insomnia, que nos permite actuar como “cliente” de la aplicación y consumir los recursos a través de los endpoints que se exponen. Con estas pruebas pudimos corroborar de forma ágil (ya que no era necesario escribir código java para implementar las pruebas), los resultados que se obtenían desde el lado servidor.

Finalmente, para este apartado, también se realizaron varias **pruebas de humo** a medida que se completaron los desarrollos de las historias de usuario. Los objetivos de estas pruebas eran, principalmente: verificar que la información se visualizaba correctamente en distintos tipos de dispositivos, verificar que los flujos de errores se

activaban consistentemente, verificar que los mensajes de error eran acordes, ingresar datos inválidos o ”de borde” y observar el comportamiento del sistema, garantizar la navegabilidad/integración entre los componentes antiguos y los que son agregados, entre otros.

Para el tradicional dilema o pregunta de “¿Hasta dónde realizamos pruebas?”, el equipo de desarrollo de este Proyecto diseñó, por cada historia de usuario, un conjunto de criterios de aceptación que, sí eran cumplidos por el sistema como un “todo”, se consideraba a esta historia de usuario como FINALIZADA. El resultado obtenido con estos criterios puede encontrarse en el Anexo E.

4.9.3 Pruebas de validación

Las pruebas de validación comienzan en la culminación de las pruebas de integración, cuando se ejercitaron componentes individuales, el software está completamente ensamblado como un paquete y los errores de interfaz se descubrieron y corrigieron. Las pruebas se enfocan en las acciones visibles para el usuario y las salidas del sistema reconocibles por el mismo.[16]

Para el contexto de este Proyecto, las pruebas de validación eran llevadas a cabo como *pruebas alfa*[6] y en un primer momento se pensaron para:

1) Cuando el equipo creía necesaria una retroalimentación por parte del cliente para asegurar que estábamos cumpliendo con lo que este esperaba. En estas ocasiones, en general, surgían pequeñas modificaciones dentro del sistema, como por ejemplo, agregar algún nuevo atributo que el cliente había olvidado, corregir la forma de visualizar algún listado o componente, etc.
2) Cuando el cliente solicitaba ver avances de su producto. Sin embargo, esta situación raramente fue materializada ya que el cliente, en general, respetaba los tiempos y con mayor frecuencia estas pruebas se activaban por el punto 1.

4.9.4 Pruebas del sistema

La prueba del sistema es una serie de diferentes pruebas cuyo objetivo principal es ejercitar por completo el sistema basado en computadora. Estas pruebas tienen como propósito ejecutar por completo el sistema en el entorno en el que deberá operar y con las bases de datos, usuarios y hardware con las que deberá trabajar normalmente. Tipos de prueba del sistema:

[6] La prueba alfa se lleva a cabo en el sitio del desarrollador por un grupo representativo de usuarios finales. El software se usa en un escenario natural con el desarrollador “mirando sobre el hombro” de los usuarios y registrando los errores y problemas de uso. Las pruebas alfa se realizan en un ambiente controlado.[16]

- Pruebas de recuperación: Fuerza al software a fallar en varias formas y verifica que la recuperación se realice de manera adecuada.
- Pruebas de seguridad: Intenta verificar que los mecanismos de protección que se construyen en un sistema en realidad lo protegerán de cualquier penetración impropia.
- Pruebas de esfuerzo: Ejecuta un sistema en forma que demanda recursos en cantidad, frecuencia o volumen anormales.
- Pruebas de rendimiento: Pone a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado.
- Pruebas de despliegue: Ejercita el software en cada entorno en el que debe operar. [16]

Debido al alcance de nuestro Proyecto Final, el sistema desarrollado se está describiendo en un **entorno de desarrollo**, por lo tanto, en el presente informe no se contempla la ejecución de este tipo de pruebas, salvo por las pruebas de seguridad mencionadas previamente, pero teniendo en cuenta que las mismas fueron llevadas a cabo mediante el uso del software Insomnia y por meras pruebas de humo hechas sobre el sistema como un todo, creemos que corresponden más bien a pruebas de integración que al presente apartado.

5 Demostración de la ejecución de un remate en el sistema desarrollado

5.1 Presentación del escenario

En este capítulo se mostrará un ejemplo de cómo funciona la aplicación web en su versión final. Con el fin de evitar que el apartado se vuelva demasiado extenso por el uso de capturas de pantallas, nos centraremos en mostrar cómo es la ejecución de un remate en particular y la gestión de sus entidades fundamentales, tratando de explicar el proceso de la forma más concisa posible. A continuación, para que el lector tenga una noción general, presentaremos los datos que serán utilizados en la demostración.

La siguiente tabla muestra quiénes serán los usuarios partícipes dentro de este remate:

USUARIO	ROL
admin admin	Administrador
Martin Perussini	Consignatario
Tomás Ravelli	Asistente (Carga datos de lotes vendidos)
Juan Perez	Asistente (El que pesa y genera boletas)

En este caso particular, el usuario con rol “Administrador” no actúa directamente con el escenario a mostrar, pero debe existir al menos uno para cualquier inconveniente que haya con las entidades del sistema ya que es el que mayor privilegios tiene en el mismo.

Siguiendo con la descripción, se muestran los clientes que actuarán como compradores y/o vendedores en el remate que será ejecutado. La primera columna muestra el nombre del cliente y, la segunda, muestra las procedencias que tienen los mismos (junto con la localidad de ésta). La procedencia será necesaria a la hora de cargar los lotes de venta del remate.

CLIENTE	PROCEDENCIA (LOCALIDAD)
Mario Cejas	La Soila (Marcelino Escalada)
Sonia Alejandra Perez	El abasto (Gdor. Crespo) El Abasto 2 (San Justo) El abasto 3 (San Javier)
Ramon Benitez	Establecimiento La Tablada (San Justo) Estancia Los Quebrachos (Mno Escalada)
Roberto Juarez	La Tabladita (San Javier)

Raul Gonzales	Los Rosales (La Criolla)

Otra información relevante a tener en cuenta en esta demostración es la carga de lotes de animales para que sean vendidos. En la tabla a continuación, mostramos cuáles lotes serán ofertados durante el remate, junto a información de vendedor, categoría y cantidad de animales.

LOTES PARA LA VENTA				
VENDEDOR	**CATEGORÍA**			
	TORO	VACA	VAQUILLONA	TERNERO
Mario Cejas	10	5	5	0
Sonia Perez	20	10	0	0
Roberto Juarez	10	10	10	5

En la última tabla se muestra un resumen de cómo quedaría el estado del remate luego de haber llevado a cabo su ejecución, principalmente, información sobre los lotes de animales que fueron vendidos. En esta tabla se presenta información relevante de cada lote vendido, como es: vendedor, comprador, categoría, cantidad, peso (si es necesario) y precio de los animales vendidos. En este punto, vale la pena realizar algunas aclaraciones:

- El peso de un lote es igual a la suma de los pesos de cada animal dentro del mismo, ya que los animales se pesan todos juntos en la balanza.
- El precio es *precio por kilogramo* si se pesa el animal o, en caso contrario, *precio unitario*.

LOTES VENDIDOS					
Vendedor	Comprador	Categoría	Cantidad	Peso	Precio
Roberto Juarez	Raul Gonzales	Vaquillona	5	750	350
Sonia Alejandra Perez	Ramon Benitez	Toro	5	-	400000
Mario Cejas	Ramon Benitez	Toro	3	-	500000
Sonia Alejandra Perez	Mario Cejas	Vaca	5	1250	300

Mario Cejas	Ramon Benitez	Vaca	5	1500	150
Mario Cejas	Sonia Alejandra Perez	Toro	7	-	400000
Mario Cejas	Sonia Alejandra Perez	Vaquillona	5	1000	250

Por último, mostraremos un resumen de la información que es más relevante para el usuario del sistema luego de haberse ejecutado un remate. Este resumen se generó manualmente por los desarrolladores para comparar, al final del capítulo, con los datos generados automáticamente por el software desarrollado, y demostrar que ambos son idénticos.

Resultados del remate simulado:

- Se vendieron 35 animales y 50 animales quedaron sin vender.
- El lote ingresado por Mario Cejas se vendió en su totalidad
- El dinero total invertido fue de $7412500
- Vendedores:
    - Sonia Perez vendió 10 animales y obtuvo $2375000.
    - Mario Cejas vendió 20 animales y obtuvo $4775000
    - Roberto Juarez venido 5 animales y obtuvo $262500
- Compradores:
    - Mario cejas compró 5 animales e invirtió $375000
    - Ramon Benitez compró 13 animales e invirtió $ 3725000
    - Raul Gonzalez compró 5 animales e invirtió $262500
    - Sonia Perez compró 12 animales e invirtió $3050000
- Por categoría:
    - Vaca:
        - Se vendieron 10 animales y no se vendieron 15 animales
        - El dinero total invertido fue $600000
        - Vendedores:
            - Sonia Perez vendió 5 animales y obtuvo $375000.
            - Mario Cejas vendió 5 animales y obtuvo $225000
            - Roberto Juarez vendio 0 animales y obtuvo $0

        - Compradores:
            - Mario Cejas compró 5 animales e invirtió $375000
            - Ramon Benitez compró 5 vacas e invirtió $225000
    - Vaquillona:
        - Se vendieron 10 animales y no se vendieron 5 animales.
        - El dinero total invertido fue de $512500
        - Vendedores:
            - Roberto Juarez vendió 5 animales y obtuvo $262500
            - Mario Cejas vendió 5 animales y obtuvo $250000
        - Compradores:
            - Raul Gonzales compró 5 animales e invirtió $262500
            - Sonia Perez compró 5 animales e invirtió $250000
    - Toro:
        - Se vendieron 15 animales y no se vendieron 25 animales.
        - El dinero total invertido fue de $6300000
        - Vendedores:
            - Sonia vendió 5 animales y obtuvo $2000000.
            - Mario cejas vendió 10 animales y obtuvo $4300000
        - Compradores:
            - Ramon Benitez compro: 8 animales e invirtió $ 3500000
            - Sonia Perez compró 7 animales e invirtió $2800000
    - Ternero:
        - Se vendieron 0 animales y no se vendieron 5 animales
        - El dinero total invertido fue $0
        - Vendedores:
            - Roberto Juarez vendió 0 animales y obtuvo $0.
        - Compradores:
            - No hay compradores

5.2 Software en tiempo de ejecución

En esta sección, plasmaremos el escenario propuesto para mostrar el funcionamiento del sistema desarrollado.

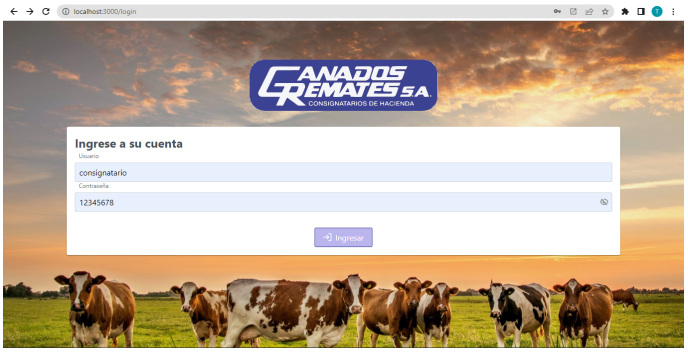


Figura 33: pantalla de inicio de sesión

Con fines meramente de demostración, se comenzará desde la pantalla de inicio de sesión, para mostrar el ingreso de un usuario con rol “Consignatario” (no confundir el *username*, que en este caso también es la misma palabra) y su vista principal.

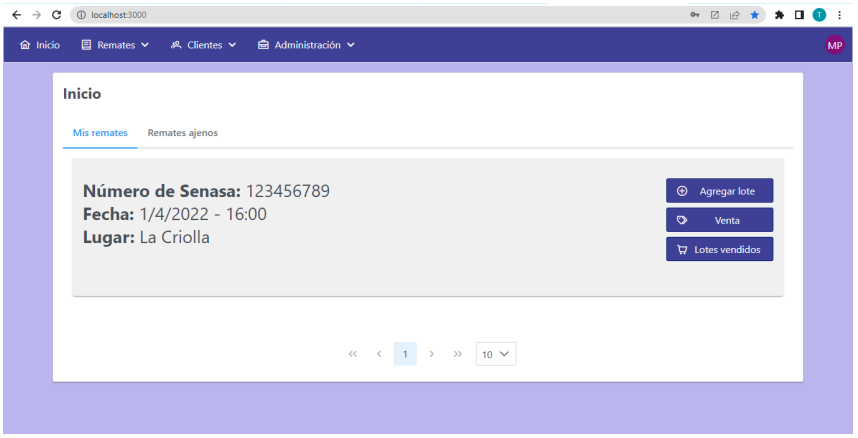


Figura 34: pantalla principal del sistema

Además, con el fin de enfocarnos en las entidades relacionadas directamente con el remate, se muestra el listado con los usuarios ya creados y sus respectivos roles.

Figura 35: listado de usuarios creados

5.2.1 Creación de categorías

Entre las primeras entidades que se necesitan para realizar la demostración y que, en general, no sufrirán grandes modificaciones a través del tiempo, están las categorías de animales. Para esto, el usuario con rol “Consignatario” accede a la sección “Categorías” desde la opción “Administración” en el menú superior y crea las categorías de animales necesarias para llevar a cabo esta muestra. En el presente capítulo, en general, se mostrará un único evento de creación, sólo a modo de ejemplo, y luego el resultado final.

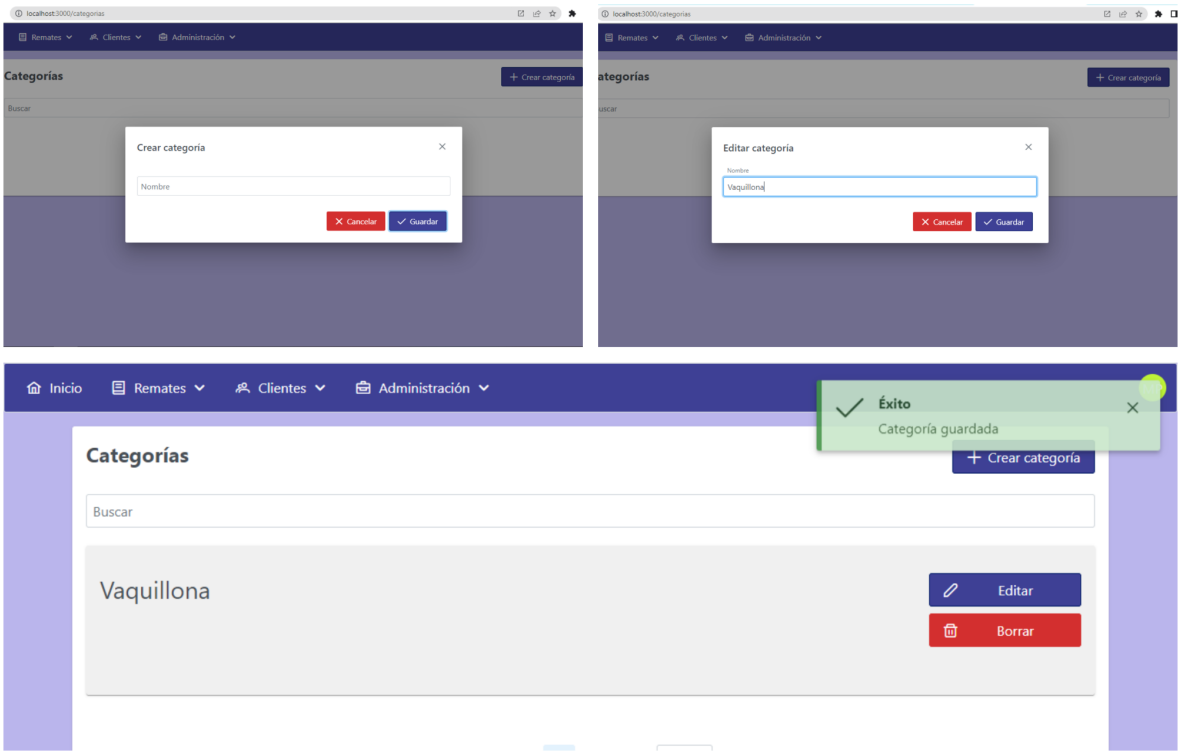


Figura 36: proceso de creación exitosa de una categoría

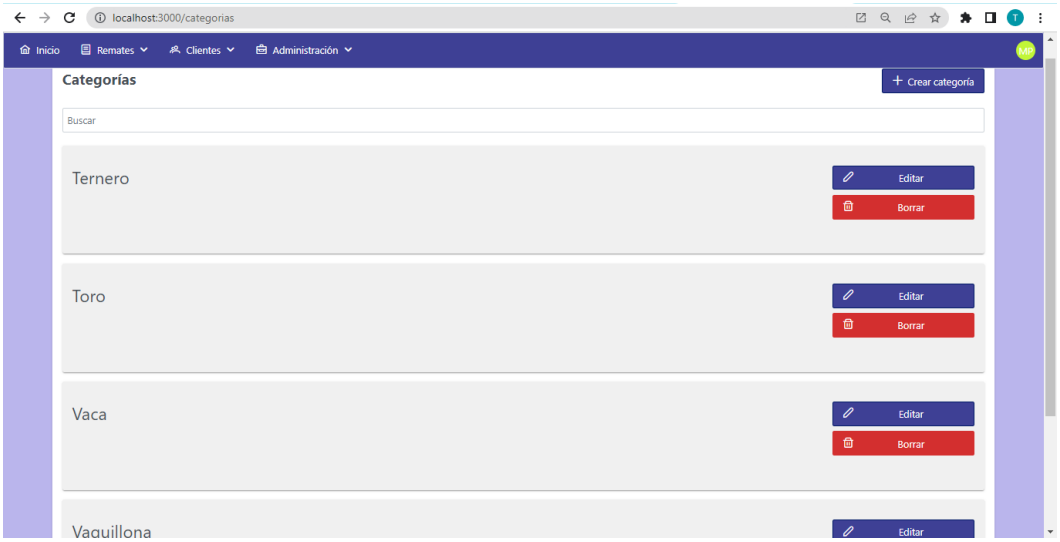


Figura 37: listado de categorías creadas

5.2.2 Creación de localidades

Las localidades son otras de las entidades necesarias desde un principio y que, luego de ser creadas, sufrirán modificaciones con escasa frecuencia. Con un procedimiento similar al descrito anteriormente, el usuario puede ingresar a gestionar las localidades desde el menú superior, en la opción “Administración”. Otra vez, solo se mostrará la creación de una sola localidad y, posteriormente, el listado final de entidades necesarias para esta demostración.

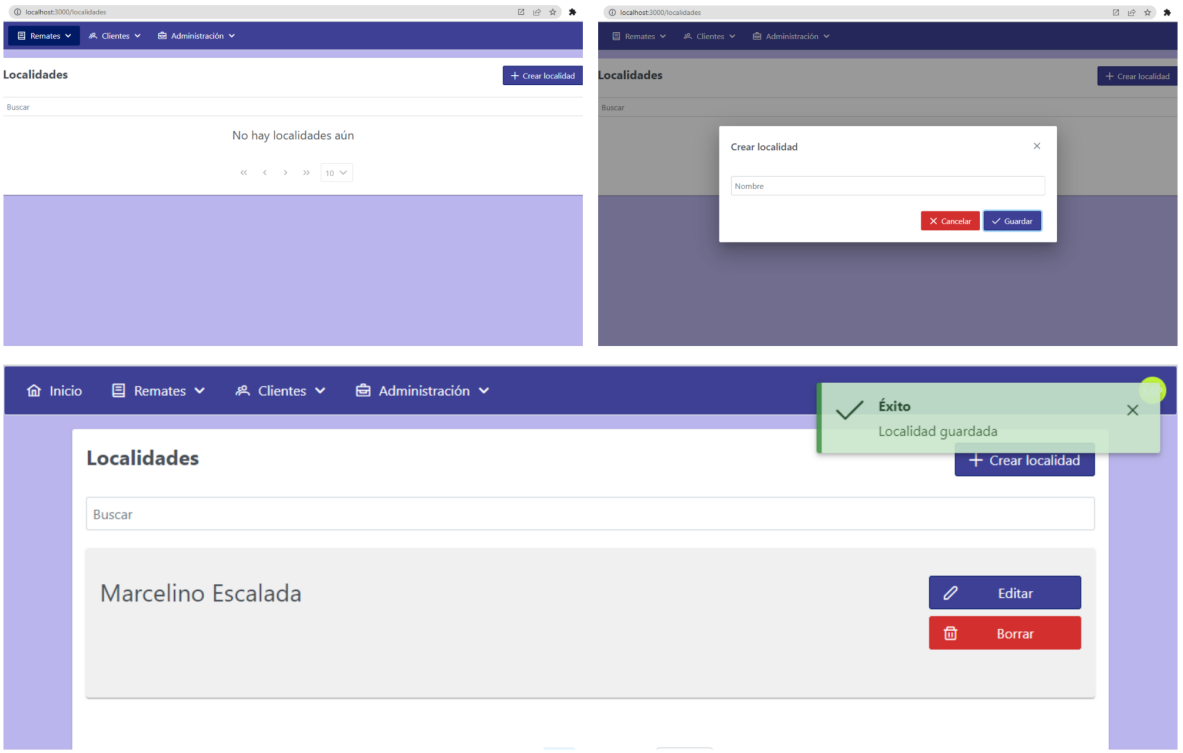


Figura 38: proceso de creación exitosa de una localidad

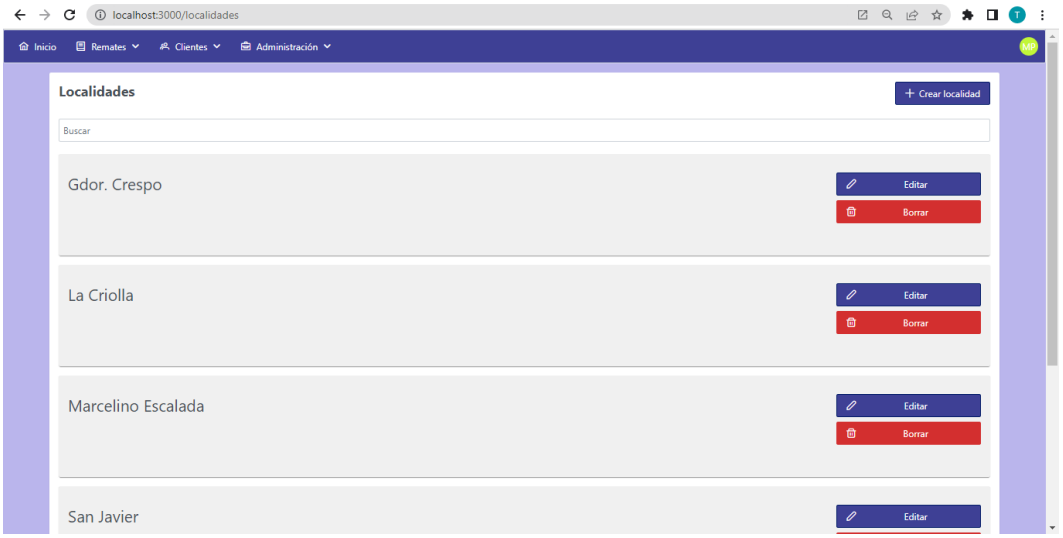


Figura 39: listado de localidades creadas

5.2.3 Creación de clientes

Los clientes son aquellas entidades que representan a compradores y vendedores dentro de nuestro sistema, según ofrezcan o compren un lote de animales. Para la creación de los clientes necesarios para este caso en particular, en la sección respectiva del menú superior, elegimos la opción “Nuevo” o desde el listado de los mismos, el usuario selecciona “Crear cliente”. Luego, el formulario es similar al de creación de las entidades anteriores.

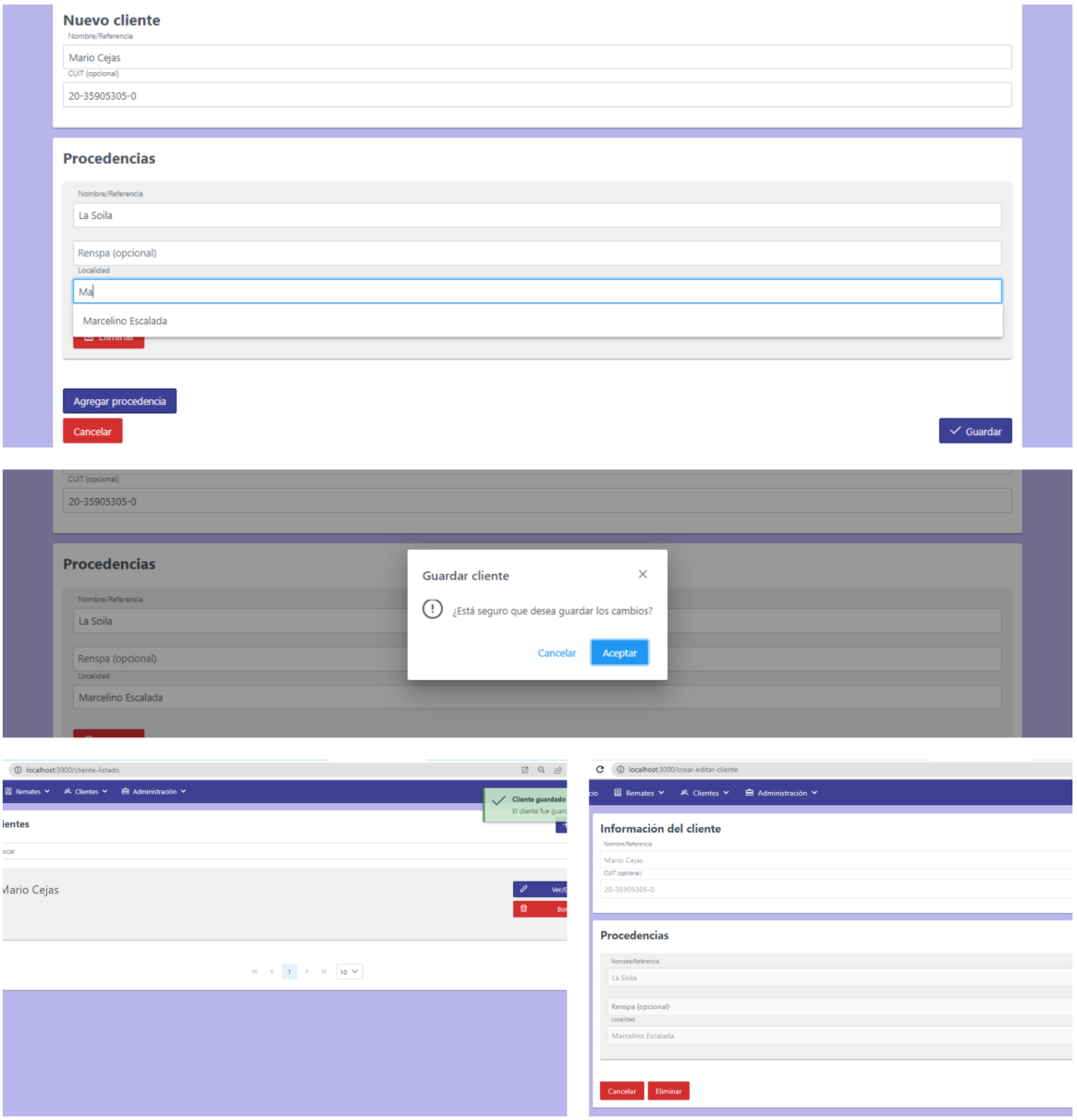


Figura 40: proceso de creación de un cliente

En este apartado cabe aclarar 2 puntos importantes:

1. Cada procedencia se creará automáticamente a partir de la creación de un cliente.
2. Para crear una procedencia debemos tener, de manera previa, la localidad necesaria creada en el sistema.

5.2.4 Crear remate

Una vez creadas las entidades previamente explicadas, se procede a crear un nuevo remate. Para esto, se nos presenta un simple formulario donde debemos completar cierta información respectiva al remate:

- Número de Senasa: campo de texto
- Fecha: campo que abre un calendario para seleccionar la fecha
- Hora: campo que despliega dos listas de números para seleccionar hora y minutos; también puede ser escritos como texto, respetando el formato
- Lugar: campo que se autocompleta con las opciones disponibles en la base de datos.

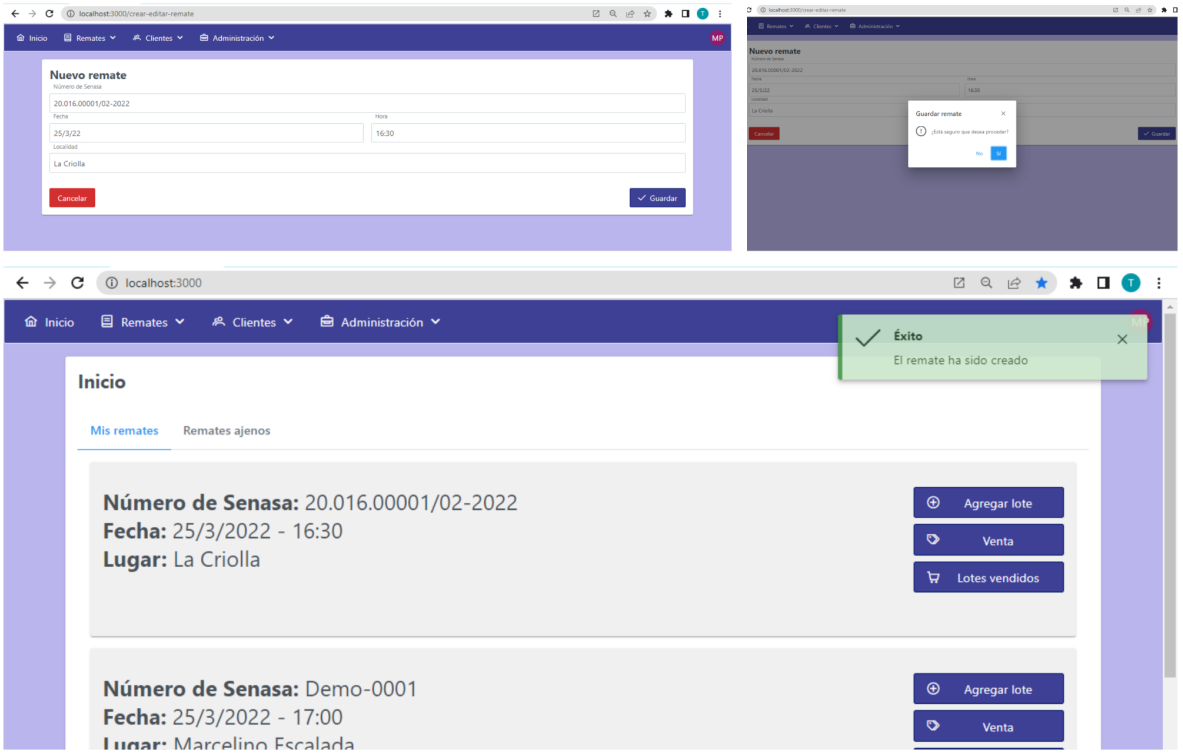


Figura 41: proceso creación de un remate

5.2.5 Asignar participantes al remate

Una vez creado el remate, deben asignarse participantes al mismo, es decir, aquellos usuarios que trabajarán para que el mismo sea llevado a cabo de manera normal y exitosa. Como se mostró en la presentación de este escenario, existirán dos participantes con rol “Asistente” y el usuario “Consignatario” que los asigna. Para agregar un participante, básicamente, se escribe en un campo de texto, mientras aparecen sugerencias de usuarios a partir del texto ingresado, hasta que se selecciona alguno.

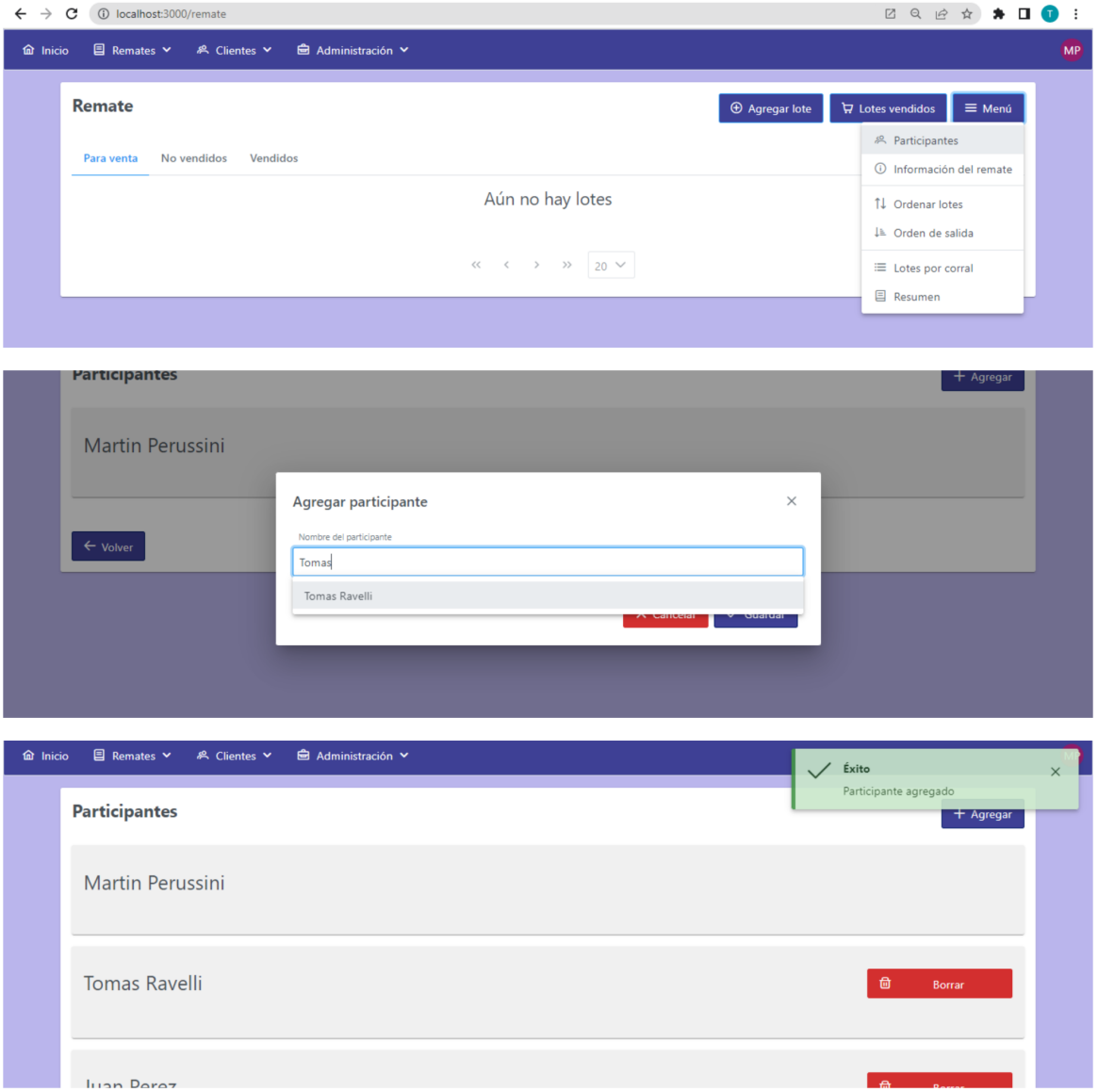


Figura 42: proceso de agregar participantes de un remate

5.2.6 Agregar lotes para la venta

El remate en cuestión ya tiene todos los elementos auxiliares para comenzar a ser cargado con animales para vender, lo cual es uno de nuestros propósitos fundamentales. Para esta tarea, en la pantalla principal del sistema existe un acceso directo.

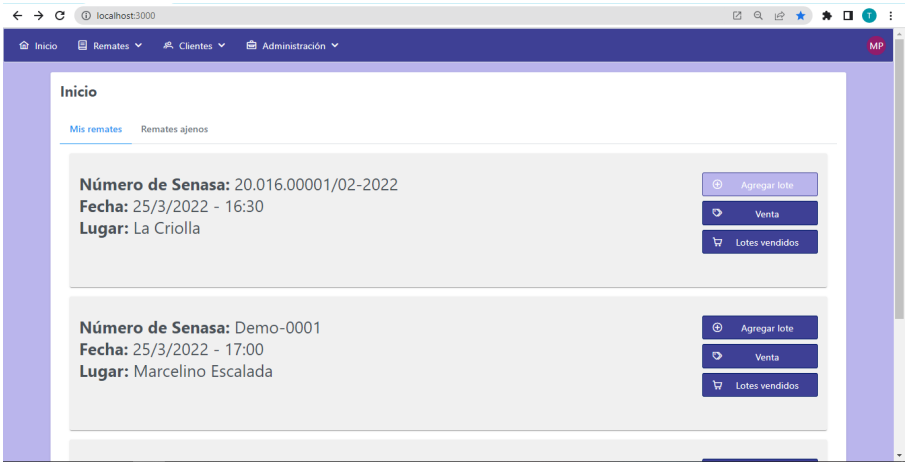


Figura 43: botón "Agregar lote"

Una vez que ingresamos a la vista correspondiente, el formulario consiste de 2 secciones claramente definidas:

1. La información del lote en general:
    a. Vendedor: campo de autocompletado, como los que se describieron anteriormente
    b. Procedencia: lista desplegable
    c. Número de corral: campo numérico
    d. Número DT-e: campo de texto opcional

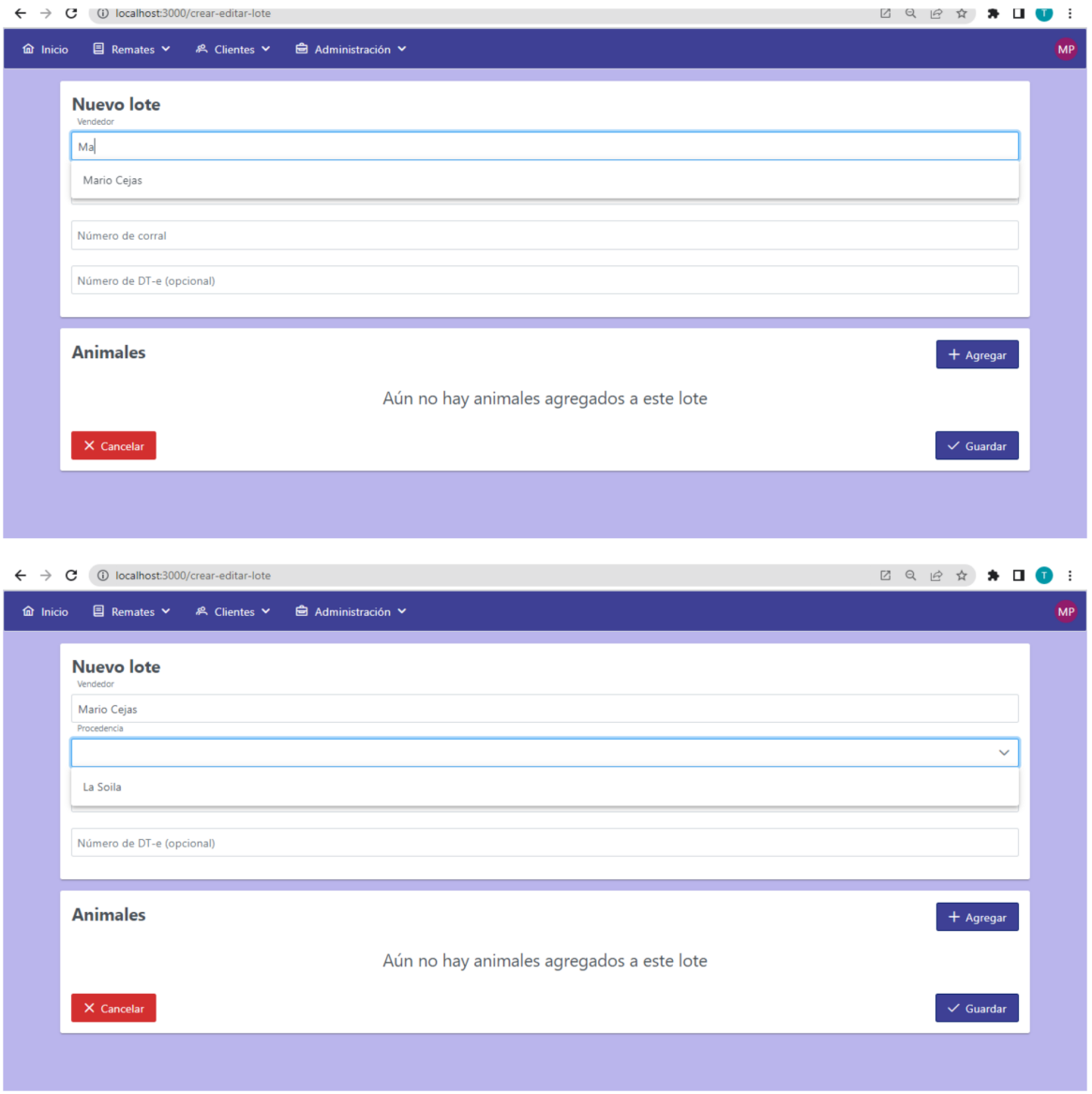


Figura 44: creación de nuevo lote para venta

2. La información de cada “Animales En Pista” que tiene este lote. Es decir, se agrupan los animales por la cantidad que serán mostrados para la venta en un mismo momento. En nuestro caso, y luego de un acuerdo con el cliente, los Animales En Pista son de la misma categoría. Entonces, por cada información que se agrega aquí, se debe seleccionar una categoría de animales y asignar la cantidad.

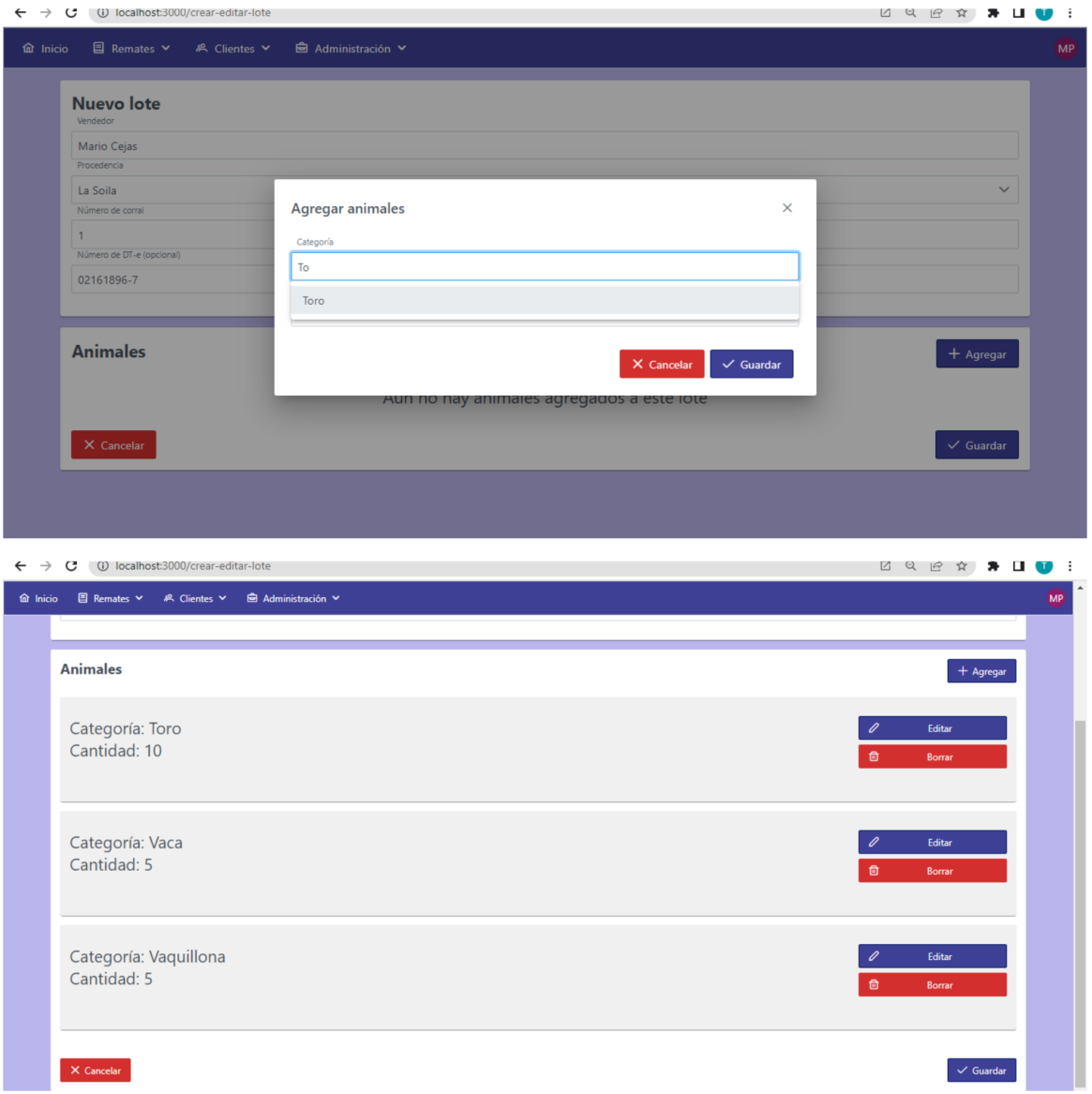


Figura 45: agregar animales al lote

Una vez creado este Lote para la Venta, en la pantalla siguiente se puede observar cómo se muestra el listado a la hora de realizar una venta. En concreto, se muestra una tarjeta por cada conjunto de animales que saldrá en el mismo momento a la pista del remate (Animales En Pista). Luego, cada una tiene las opciones respectivas para que la venta sea (o no) realizada.

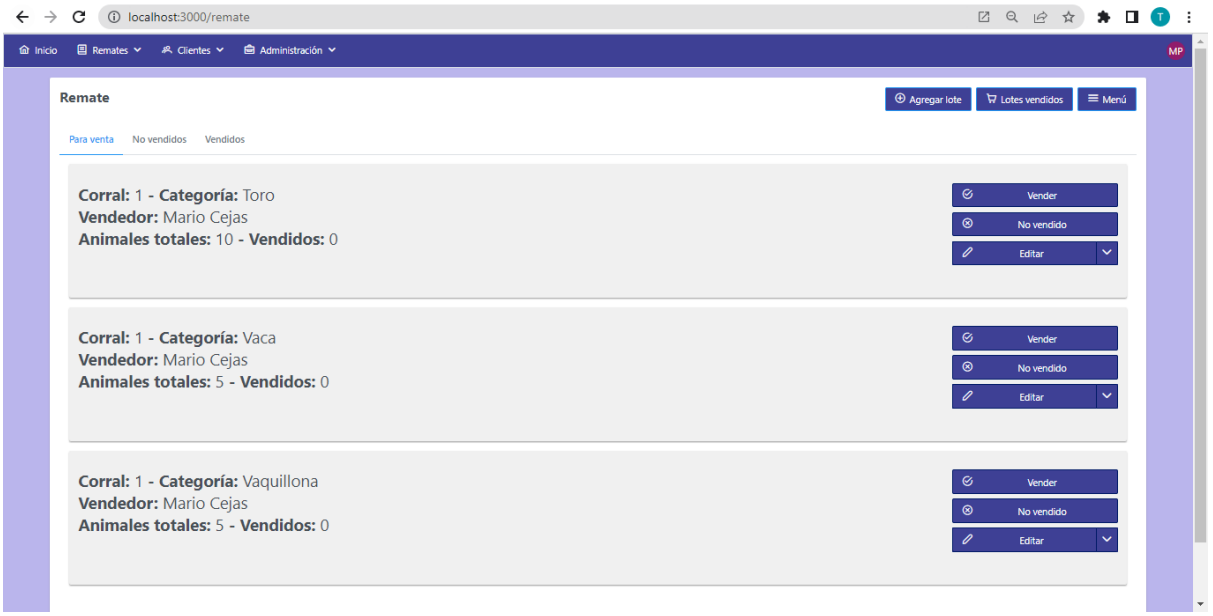


Figura 46: listado de lotes para venta

Para concluir, en las últimas pantallas de esta sección, mostramos una de las funcionalidades “extras” que se implementó dentro del proyecto, pero creíamos que era de gran utilidad para el usuario del sistema. La opción de **ordenar lotes** permite que se puedan cambiar el orden en que los animales son mostrados en el sistema mediante un mecanismo de “drag and drop”, para facilitar al usuario reordenar la manera en que cada lote de animales va a ser mostrado en la pista.

Figura 47: ordenar los lotes para su salida a pista mediante “drag and drop”

Por último, se muestra el resultado de la funcionalidad para que el usuario descargue, en formato PDF, el orden de salida de los animales a pista durante el remate.

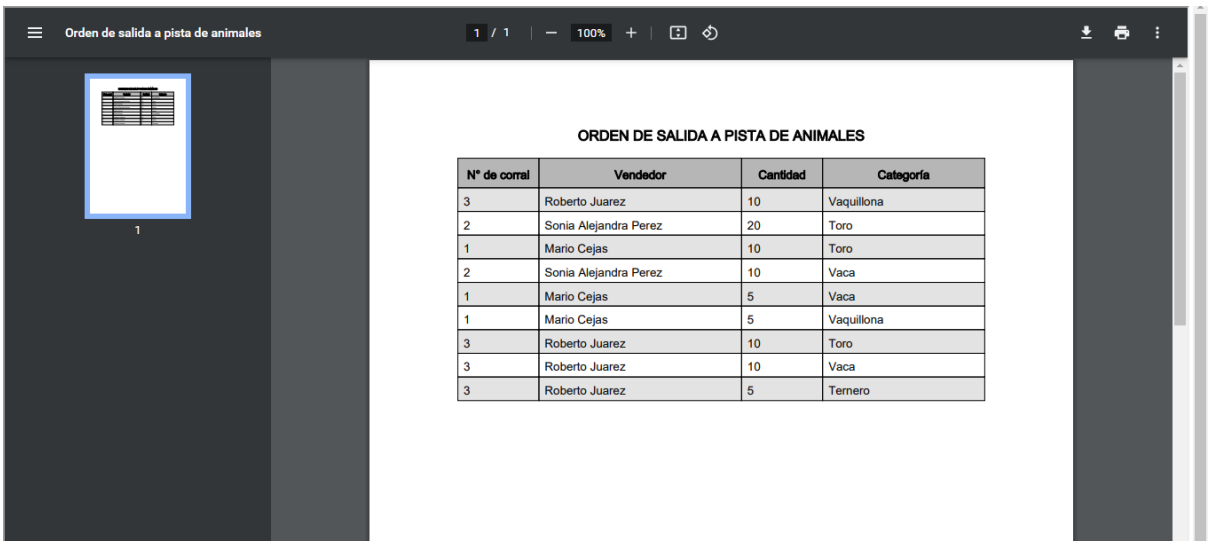


Figura 48: generación del orden de salida de los animales en formato PDF

5.2.7 Ventas de animales

Para mostrar el proceso de venta de un conjunto de animales durante el remate, ingresamos al sistema con el usuario “Asistente” llamado Tomas Ravelli.

5.2.7.1 Primer lote vendido

Al hacer click en el botón “vender” sobre una de las tarjetas en el listado de animales para venta, se abre el diálogo que se muestra a continuación. El mismo consiste de un formulario donde se completan los datos respectivos a una venta:

- Comprador: consiste en un campo que sugiere clientes (que existen en la base de datos) a medida que el usuario va completando el mismo.
- Precio: si se pesan los animales, es el precio por kilogramo; si no, es el precio por cada unidad que se vende, como en la tabla de presentación del escenario.
- Cantidad: número de animales vendidos.
- Plazo: cantidad de días que se disponen para realizar el pago de la compra.
- “Se pesa”/”No se pesa”: botón que cambia el valor de su etiqueta según se clickea para que el lote sea pesado o no.

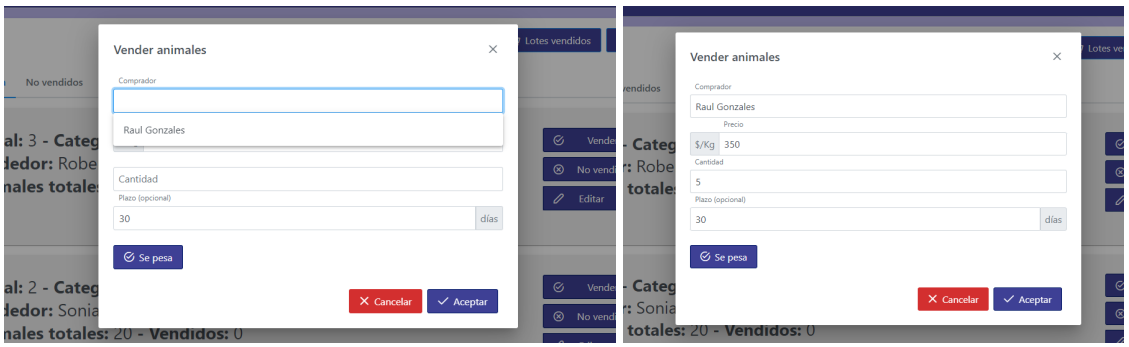


Figura 49: cargar datos de lote vendido

En la siguiente figura, se pretende mostrar que, una vez generada la venta en la pantalla de la izquierda, la pantalla de la derecha (donde está “logueado” Juan Perez) se actualiza automáticamente al ejecutarse este evento, gracias al websocket implementado para esta tarea. Esto es una gran ventaja para los usuarios que deben pesar los nuevos lotes vendidos o generar las boletas de venta correspondiente ya que, principalmente, no tienen que esperar a ser notificados por otro medio para actualizar su pantalla.

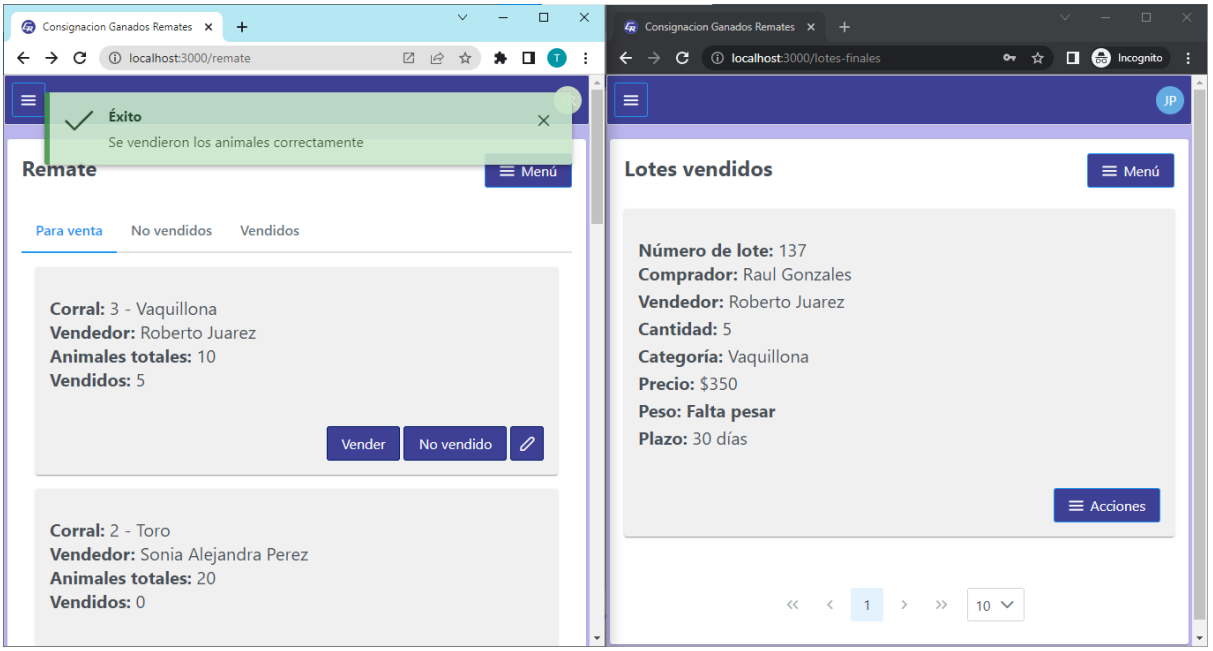


Figura 50: vender lote

Entonces, una vez que al usuario Juan Perez se le actualizó la pantalla y recibió un nuevo lote que fue vendido, debe terminar de cargar los datos, en caso de ser necesario. En el ejemplo de esta venta en particular, el usuario debe cargar el peso de los animales vendidos. Para esto, el usuario selecciona la opción correspondiente y se abre un pequeño diálogo con un campo para ingresar el valor correspondiente. A continuación se muestra dicho procedimiento.

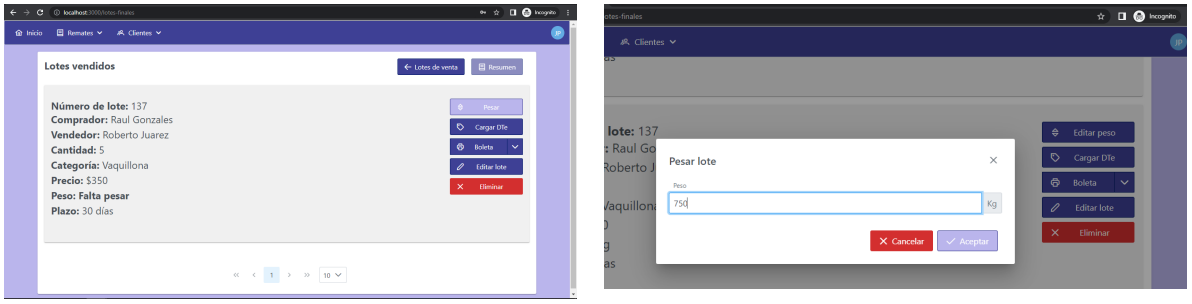


Figura 51: cargar peso a un lote vendido

5.2.7.2 Boleta de venta

Una vez que se cargaron los datos fundamentales de la venta, el usuario puede generar la boleta respectiva y descargarla o visualizarla en formato PDF, eligiendo el número de copias que necesite, como se muestra a continuación.

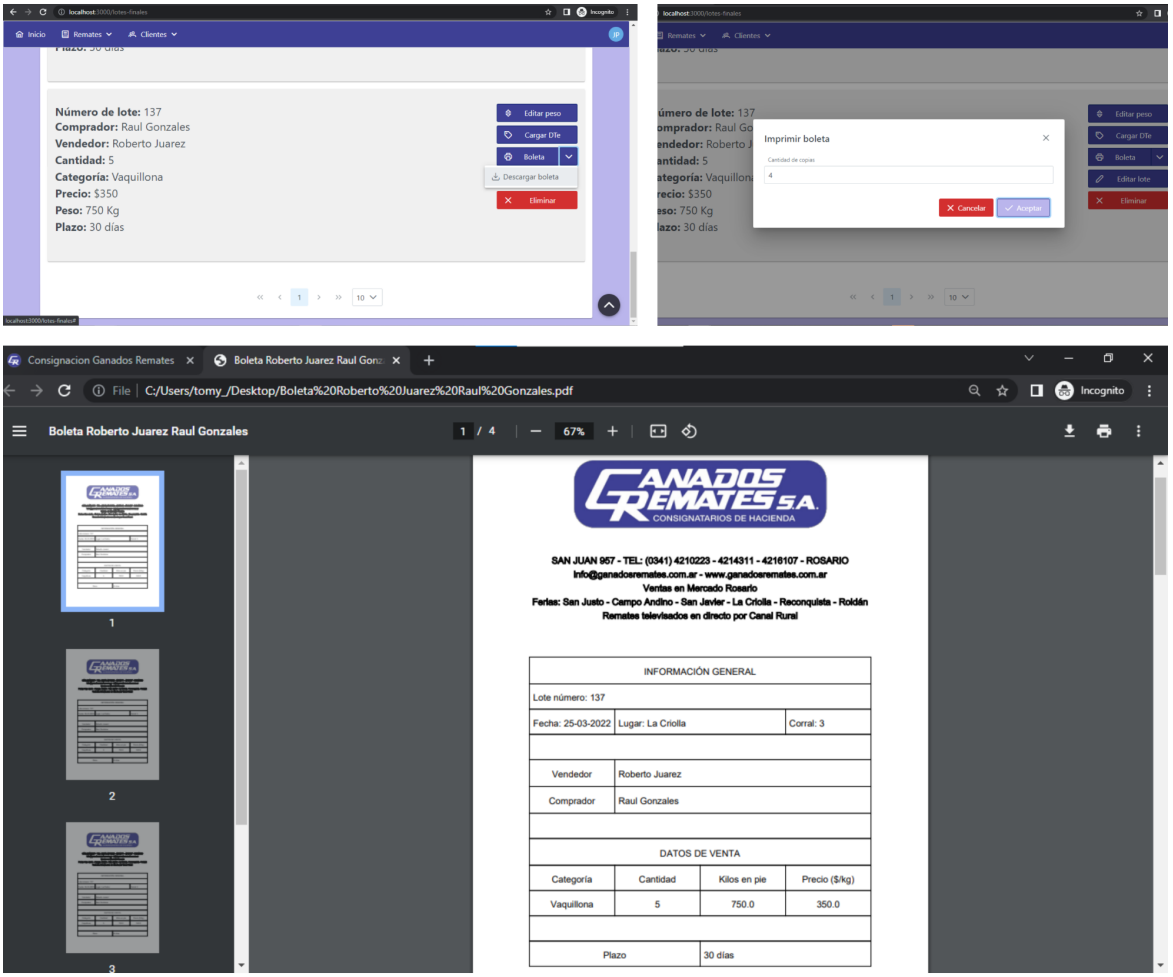


Figura 52: boleta de venta generada en formato PDF

5.2.8 Lotes vendidos en su totalidad

En esta vista, dentro de la pestaña “Vendidos” se muestran aquellos “Animales En Pista” que fueron vendidos completamente. Creamos esta vista porque, como no hay

posibilidad de que se vendan más animales de este lote, carece de sentido y se puede tornar molesto o incómodo que el mismo siga apareciendo en el listado de los lotes que todavía pueden venderse.

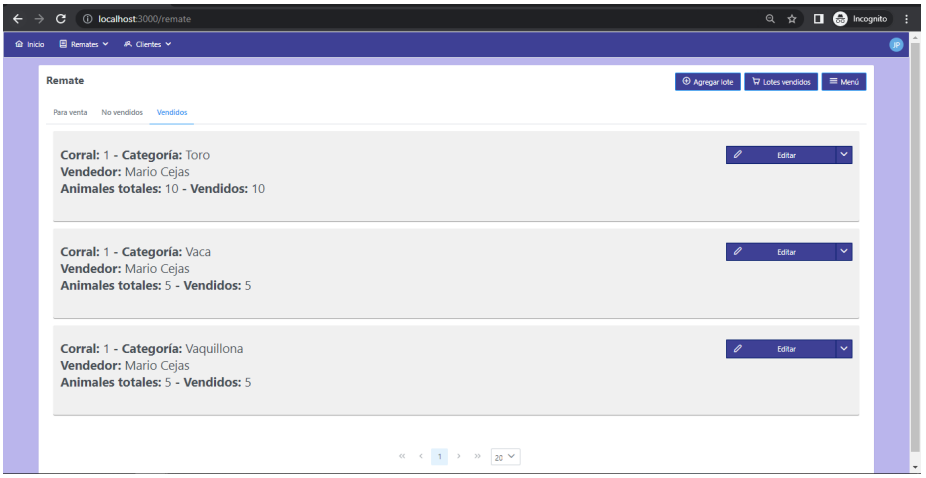


Figura 53: lista de lotes vendidos

5.2.9 Cargar DT-e a un lote vendido

Esta funcionalidad es opcional y relativa a la necesidad que tenga el usuario de agregar o no el dato de número de tránsito electrónico a un lote vendido. La decisión de que sea opcional es porque, luego de las entrevistas, concluimos que es muy poco probable que el usuario se tome el tiempo durante un remate de cargar un dato que es largo de escribir y propenso a errores. Finalmente, el usuario, una vez que termina el remate, puede cargar este dato a todos los lotes vendidos y no vendidos, cuando ingresa al listado correspondiente (es la única información que puede modificar una vez que se finaliza un remate).

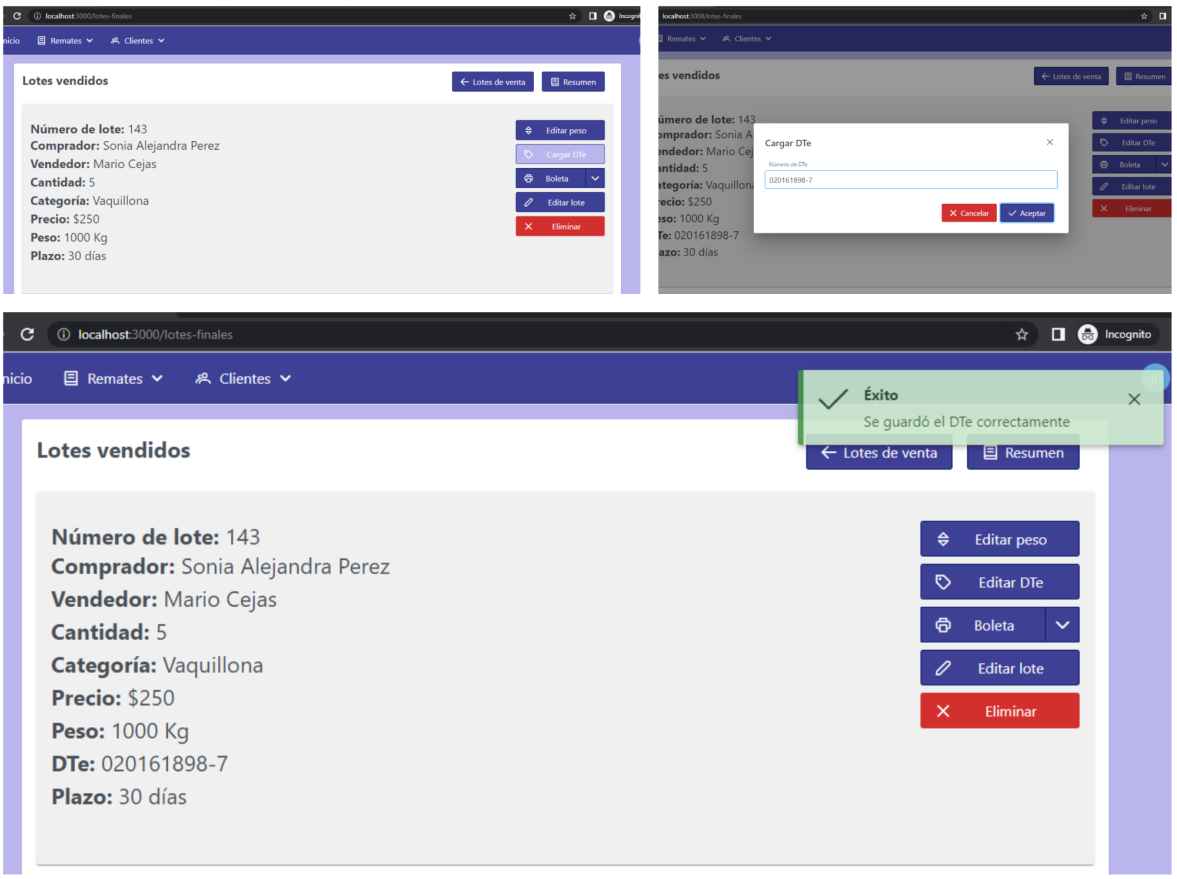


Figura 54: carga de DT-e

5.2.10 Listado de lotes vendidos

A nivel informativo y para comprensión del lector, se muestra una porción del listado completo que se generó al finalizar todas las ventas que se propusieron en la presentación de esta demostración de prueba.

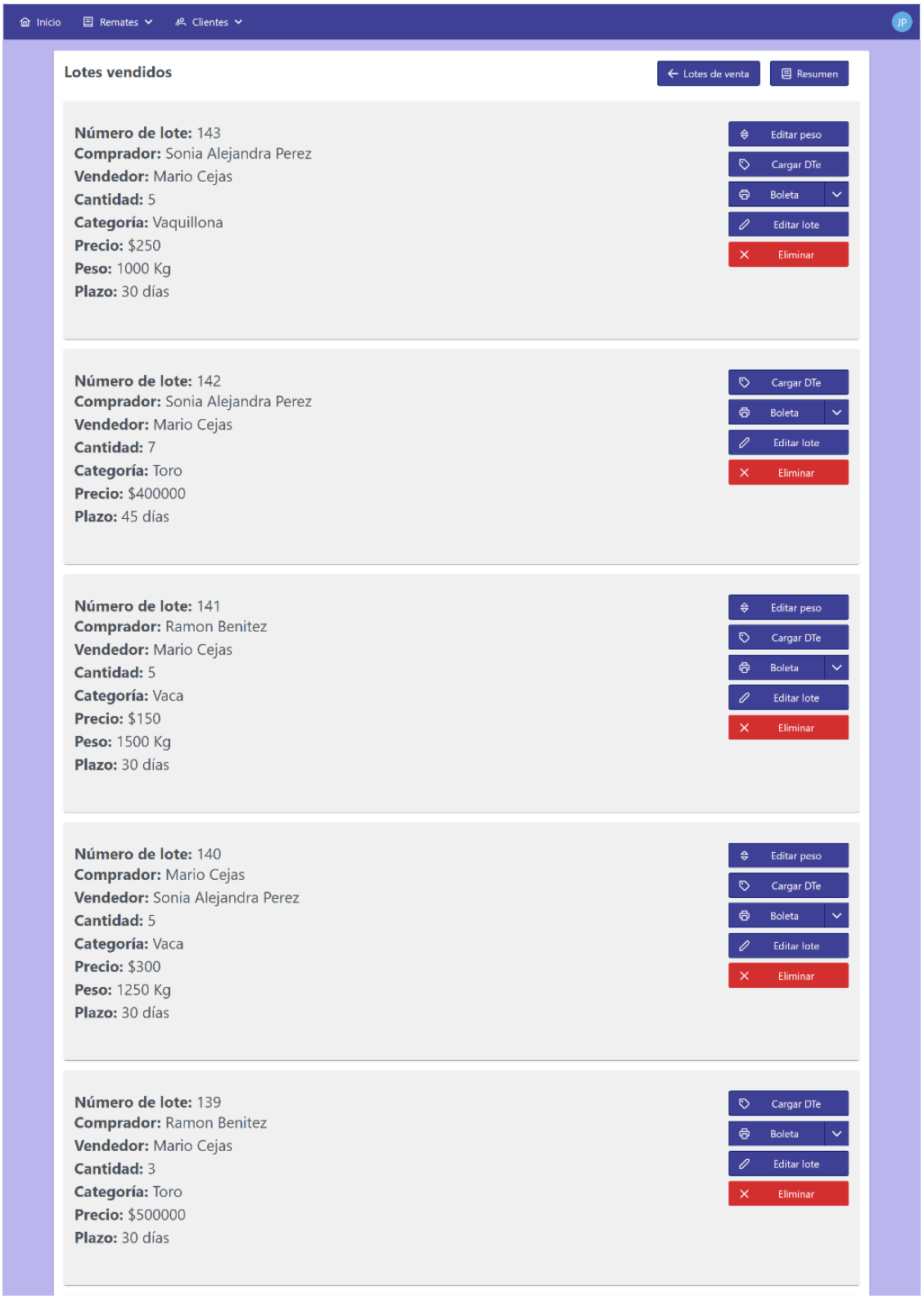


Figura 55: listado final de lotes vendidos

5.2.11 Resúmenes

La última sección de este capítulo corresponde a los reportes o resúmenes estadísticos de la información que se genera durante la realización de un remate. Como se comentó en los primeros apartados de esta tesis, dicha función fue una de los primeros requisitos que dieron origen al Proyecto. En pocas palabras, estos reportes permiten tener una organización de manera tabular y gráfica de la información que, según las entrevistas con el cliente, resultan de mayor valor para la toma de decisiones de éste. Con la forma de trabajo actual, los usuarios del sistema, debían realizar manualmente y en papel físico esta organización de la información luego de finalizar un remate. Utilizando el sistema, con el mero hecho de hacer un click en un botón, obtienen la misma información en unos pocos segundos. Más aún, la información gráfica le agrega mucho valor a esta funcionalidad y facilita mucho más la comprensión y el análisis de los datos para el usuario.

Estos resúmenes se pueden agrupar en **información general** o **información por categoría de animales** ofertados en las ventas del remate y su estructura es similar a lo que se detalló en la primera sección de este capítulo. Para concluir, se observa a continuación que los resultados generados por el software son los mismos que los calculados manualmente al inicio y, por lo tanto, se cumplió con el objetivo de esta demostración.

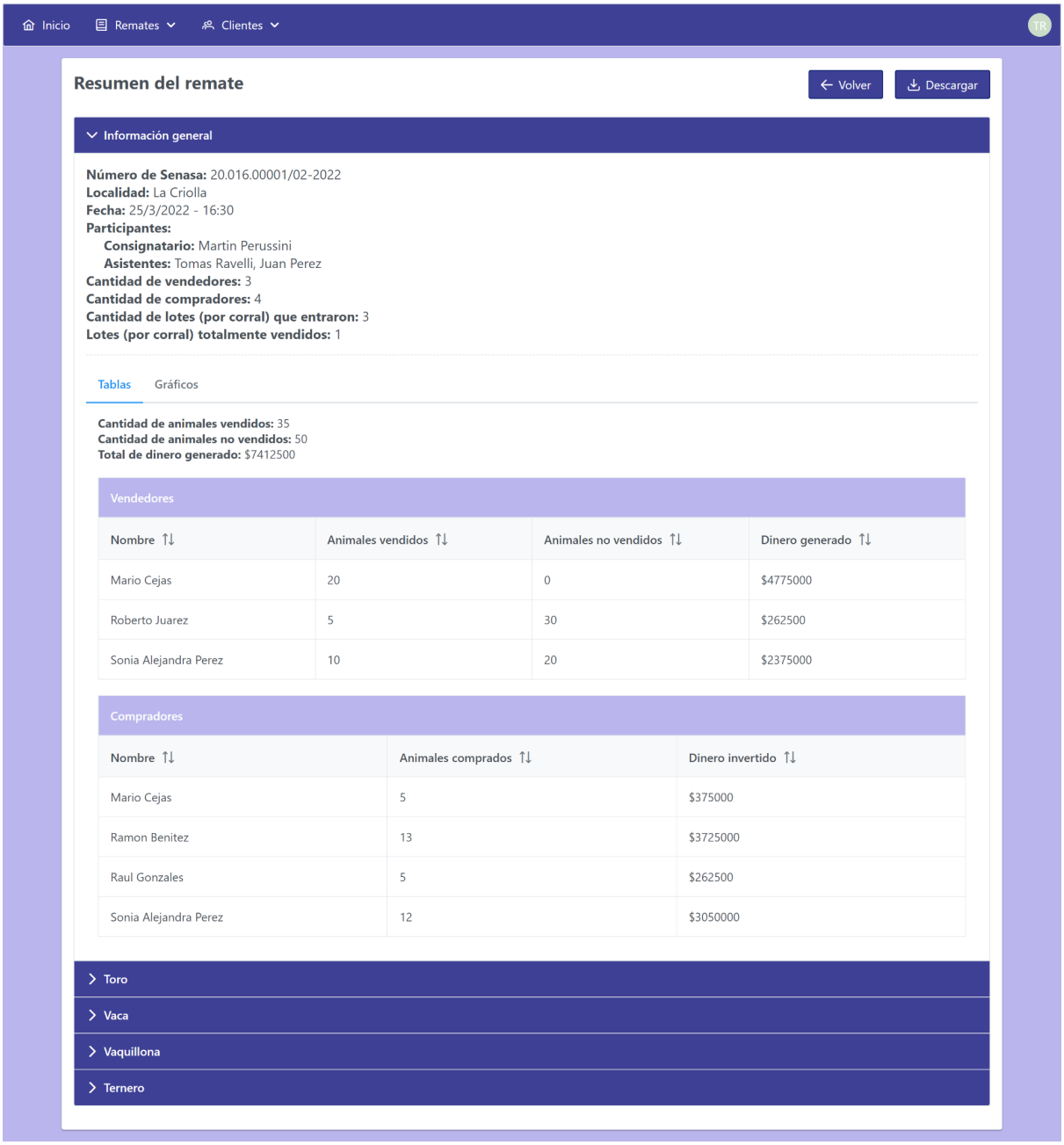


Figura 56: Información general - Reporte formato tabla

Figura 57: Información general - Gráfico de torta: Animales vendidos y no vendidos

Figura 58: Información general - Gráfico de barras: Cantidad vendida/no vendida por vendedor

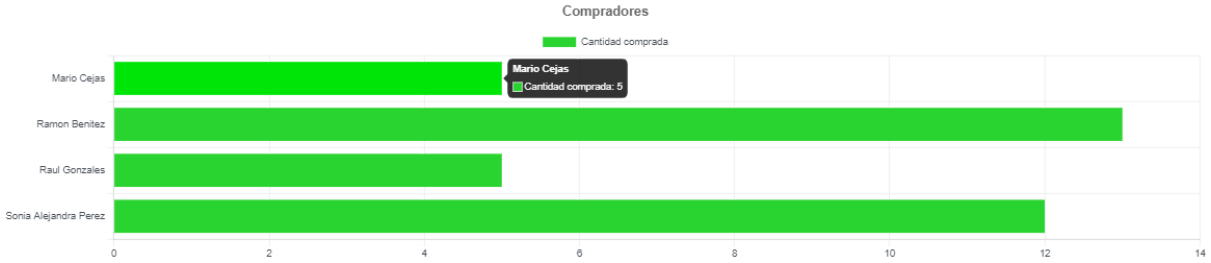


Figura 59: Información general - Gráfico de barras: Cantidad comprada por comprador

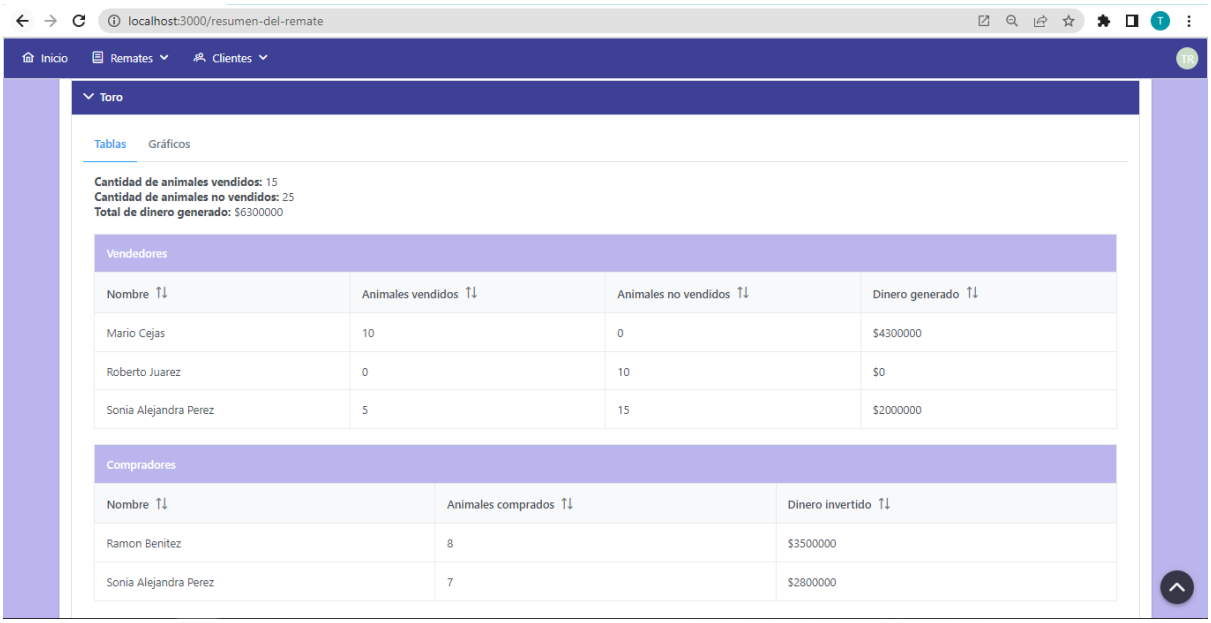


Figura 60: Información por categoría - Reporte formato tabla

Figura 61: Información por categoría - Gráfico de torta: Cantidad vendida/no vendida por vendedor

Figura 62: Información por categoría - Gráfico de barras: Cantidad vendida/no vendida por vendedor

Por último, también se agregó la posibilidad de que esta información estadística pueda ser visualizada o descargada en formato PDF en el almacenamiento local del usuario (por ejemplo, para que sea impresa posteriormente), junto con el listado detallado de todos los lotes vendidos durante el remate (similar a la lista de lotes vendidos que se muestra en el sistema).

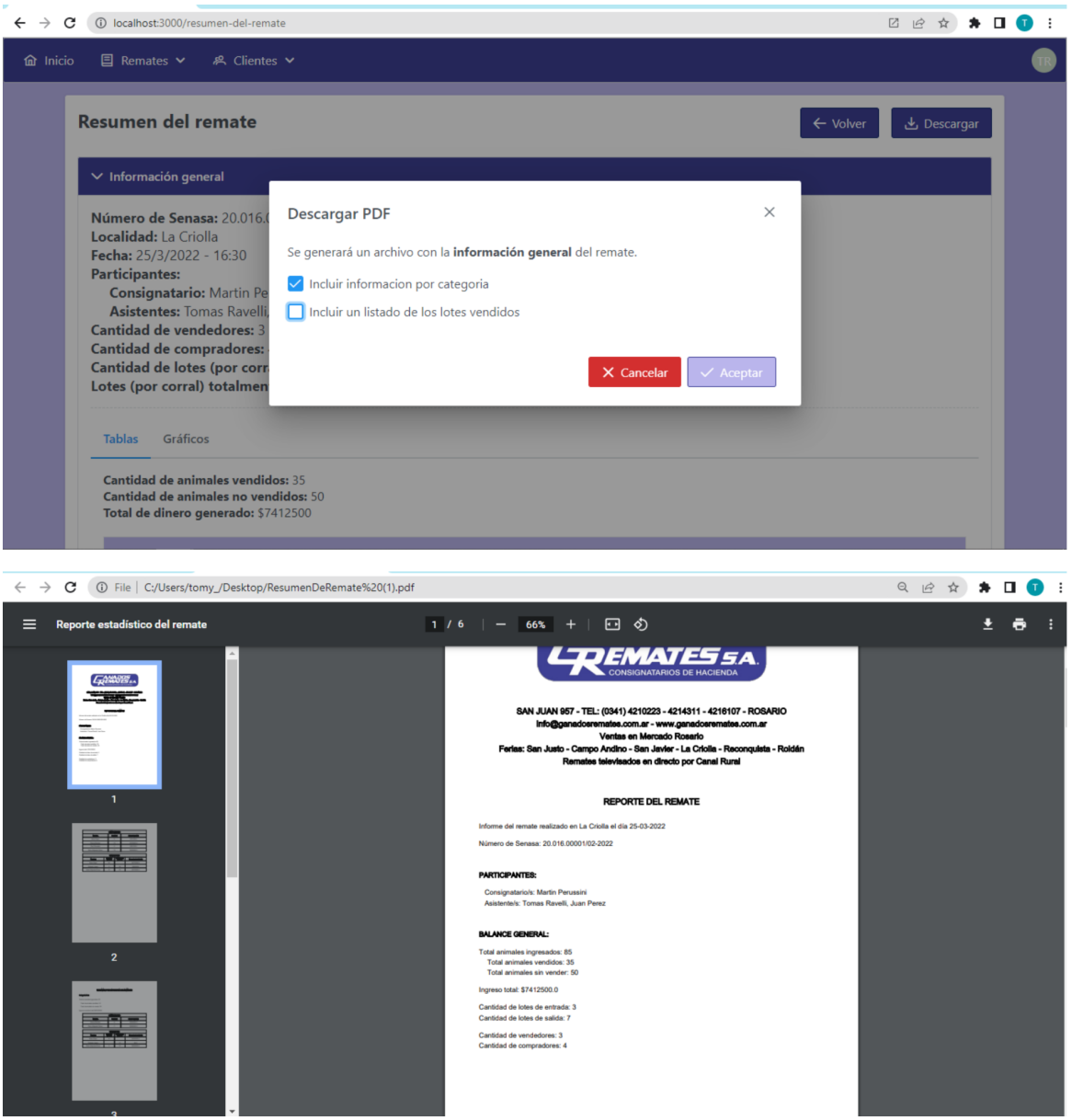


Figura 63: Generación de reportes en formato PDF

6 Conclusión

Los aportes alcanzados mediante el presente Proyecto Final de Carrera se pueden categorizar según quien los percibió, destacando aquellos a nivel de negocio, para el cliente, a nivel profesional y a nivel personal para los alumnos. Los mismos se detallan a continuación.

Respecto al cliente, como comentamos en la fundamentación, este sistema pretende facilitarle su labor a aquellas personas que se dedican al negocio de la consignación de haciendas, brindándoles una plataforma que automatiza y agiliza parte del trabajo, al permitirles obtener informes de los remates y generar digitalmente las boletas de venta, así como almacenar toda la información de forma digital para que todos lo involucrados tengan acceso a la misma de forma sencilla, rápida y segura. Esto también les permitirá ahorrar costos por el uso de papel y mejorar la toma de decisiones gracias a los reportes generados. En este caso hablamos en potencial dado que está más allá del alcance de este proyecto el despliegue efectivo del sistema.

En cuanto a los aportes a nivel profesional, hemos profundizado nuestro conocimiento sobre procesos de planeación y coordinación para el desarrollo de software, así como en el aprendizaje de las tecnologías utilizadas. También hemos ganado experiencia en el relevamiento de requerimientos, a partir de la comunicación con los futuros usuarios de este sistema, para llegar a comprender sus necesidades, tarea que es crítica en el comienzo de un nuevo proyecto de desarrollo de software.

Por último, en lo personal, además de los aportes a nivel profesional, creemos haber mejorado nuestras habilidades para trabajar en equipo a través de una efectiva comunicación, asignación de tareas e intercambios de conocimientos y experiencias a los largo de los meses en los que hemos estado trabajando. A su vez, gracias a nuestra dedicación, pudimos completar el desarrollo de la aplicación en un tiempo que no distó demasiado a lo que habíamos planificado, logrando cumplir con todos los requisitos funcionales planteados y, además, incorporando funcionalidades que le dan valor extra a nuestro producto para el cliente.

7 Bibliografía y fuentes

[1] “RENSPA”, *Argentina.gob.ar*, el 21 de agosto de 2018. https://www.argentina.gob.ar/senasa/micrositios/renspa (consultado el 26 de mayo de 2022).

[2] “¿Qué es el Senasa?”, *Argentina.gob.ar*, el 11 de junio de 2018. https://www.argentina.gob.ar/senasa/que-es (consultado el 26 de mayo de 2022).

[3] “Los remates de hacienda: la mejor manera de conocer el campo entrerriano”, *Análisis*, el 23 de agosto de 2019. https://www.analisisdigital.com.ar/opinion/2019/08/23/los-remates-de-hacienda-la-mejor-manera-de-conocer-el-campo-entrerriano (consultado el 26 de mayo de 2022).

[4] “Consignatario”, *Economipedia*. https://economipedia.com/definiciones/consignatario.html (consultado el 26 de mayo de 2022).

[5] “Martillero público”, *Wikipedia, la enciclopedia libre*. el 17 de marzo de 2022. Consultado: el 26 de mayo de 2022. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Martillero_p%C3%BAblico&oldid=142323083

[6] M. Alamio y M. Salías, *Proyectos Ágiles con Scrum: Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos*. Buenos Aires: Kleer, 2015.

[7] “Describa brevemente el algoritmo de cifrado Blowfish - programador clic”. https://programmerclick.com/article/56211277480/ (consultado el 26 de mayo de 2022).

[8] “Algoritmo Blowfish con ejemplos – Acervo Lima”. https://es.acervolima.com/algoritmo-blowfish-con-ejemplos/ (consultado el 26 de mayo de 2022).

[9] “Blowfish (cipher)”, *Wikipedia*. el 2 de noviembre de 2021. Consultado: el 26 de mayo de 2022. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Blowfish_(cipher)&oldid=1053146843

[10] “Arquitectura de n capas”. http://iutll-abdd.blogspot.com/2012/05/arquitectura-de-n-capas.html (consultado el 26 de mayo de 2022).

[11] “¿Qué son las Single-Page Application (SPA)? El desarrollo elegido por Gmail y LinkedIn”, *DIGITAL55*, el 8 de junio de 2021. https://d55.lextrendlabs.com/que-son-single-page-application-spa-desarrollo-elegido-por-gmail-linkedin/ (consultado el 26 de mayo de 2022).

[12] auth0.com, “JWT.IO - JSON Web Tokens Introduction”. http://jwt.io/ (consultado el 26 de mayo de 2022).

[13] Cátedra de Redes de Información - UTN FRSF, “Apunte Redes de Información - Unidad 6”. 2020.

[14] M. Jones, J. Bradley, y N. Sakimura, “JSON Web Token (JWT)”, Internet Engineering Task Force, Request for Comments RFC 7519, may 2015. doi: 10.17487/RFC7519.

[15] “JSON Web Token (JWT)”. https://www.iana.org/assignments/jwt/jwt.xhtml (consultado el 26 de mayo de 2022).

[16] R. S. Pressman, *Ingeniería del software: un enfoque práctico*. 2013. Consultado: el 24 de mayo de 2022. [En línea]. Disponible en: http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4272

[17] Atlassian y M. Rehkopf, “Historias de usuario | Ejemplos y plantilla”, *Atlassian*. https://www.atlassian.com/es/agile/project-management/user-stories (consultado el 26 de mayo de 2022).

[18] Emiliano Grande, “Plan de gestión del riesgo”, 21:43:07 UTC. Consultado: el 27 de mayo de 2022. [En línea]. Disponible en: https://es.slideshare.net/EmilianoGrande/plan-de-gestin-del-riesgo-77249033

ANEXO A - Modelo de desarrollo de software

Como se explicó en la sección previa, para el desarrollo de este proyecto adoptaremos un enfoque ad-hoc basado en la metodología Scrum, tomando su filosofía de trabajo y valores, así como muchas de sus prácticas, desarrollando algunas de ellas tal cual lo aconseja la metodología, apoyándonos en el libro “Proyectos Ágiles con Scrum: Flexibilidad, aprendizaje, innovación y colaboración en contextos complejos”[6], y adaptando otras a nuestra situación particular. En las próximas líneas describiremos las bases de Scrum y presentaremos nuestra propuesta de adaptación para este desarrollo.

A.1 Roles de Scrum

En un equipo de Scrum convencional intervienen 3 roles: el Scrum Master, el equipo de desarrollo y el Product Owner.

En nuestro caso, aunque lo más recomendable por la metodología es que cada persona ocupe solo un rol, al ser este un grupo integrado por 2 individuos no nos queda más opción que llevar adelante más de un rol por persona, convirtiéndose los mismos en algo más bien difuso.

El **Product Owner** es la persona responsable del éxito del proyecto desde el punto de vista de los stakeholders, realizando tareas como la recolección de requerimientos, determinar las prioridades de las características del producto y proveer feedback valioso para la evolución del producto.

En nuestro caso particular, quien tome este rol en un principio sería Martin Perussini, ya que posee el contacto más directo con el cliente dándose así una comunicación más fluida para obtener el conocimiento necesario sobre el dominio del problema y proveer feedback por parte de los stakeholders. Sin embargo, este conocimiento será compartido por ambos integrantes del grupo, por lo que las demás tareas (como la priorización de los requerimientos, ordenamiento del backlog, etc.) serían llevadas a cabo en conjunto.

El **equipo de desarrollo** está formado por todos los individuos necesarios para la construcción del producto en cuestión. Se trata de un grupo auto-organizado, el cual determina cómo se realizará el trabajo, con el objetivo final de producir un incremento funcional potencialmente entregable, o en otras palabras, transformar las funcionalidades prometidas a software funcionando. Lo recomendable es que este equipo no supere las 9

personas, además de que cada una de ellas posea los conocimientos y habilidades necesarias para el trabajo requerido, lo que implica que no hay especialistas exclusivos.

Yendo ahora a nuestro proyecto, indudablemente ambos integrantes formaremos parte del equipo de desarrollo. Pasando a los roles propios de un equipo de desarrollo (como analistas, programadores, testers/QA), está claro que al ser solo dos integrantes, ambos tomaremos parte en los distintos roles para poder llevar adelante este proyecto.

Por último, el **Scrum Master** es el coach del equipo, quien vela por la productividad del mismo y se asegura de que todos (incluido el Product Owner) entiendan y utilicen Scrum de manera correcta. Para lograr su cometido, el Scrum Master es quien se asegura de que todos asistan a los distintos eventos que la metodología propone (explicados más adelante), protege al equipo de desarrollo de distracciones externas, detecta y trata de eliminar impedimentos que surjan relacionados al proyecto o a la metodología, promueve la comunicación y cooperación en el ambiente de trabajo, trabaja también con el Product Owner para asegurar una correcta priorización de los requerimientos, entre otras cosas.

En el presente proyecto, al no contar ninguno de los integrantes de este grupo con experiencia en el uso de la metodología, proponemos tomar al director de proyecto como Scrum Master, tomando un rol no tan activo como lo dicta la metodología por el hecho de obviamente no poder destinar el tiempo necesario para tal labor, viéndose más bien su papel como aquel que responde consultas sobre las buenas prácticas de Scrum y tomando parte en las reuniones de Sprint Review principalmente. Las demás tareas que atañen a este rol, al igual que en el resto de los mismos, serán llevadas adelante por ambos miembros del grupo.

A.2 Elementos de Scrum

Una **historia de usuario** es una explicación general e informal de una función de software escrita desde la perspectiva del usuario final o cliente. El propósito de una historia de usuario es articular cómo un elemento de trabajo entregará un valor particular al cliente.[17]

La implementación de una historia de usuario se mide en una unidad relativa que permite expresar una estimación del esfuerzo total que deberá hacer el equipo, denominada **story point**. El valor de esta unidad fue establecido arbitrariamente por el equipo de desarrollo teniendo en cuenta la duración de un día de trabajo, es decir, **6 horas**.

El **Product Backlog** se trata de un listado de ítems (Product Backlog Ítems, PBIs) o características del producto a desarrollar, el cual es administrado exclusivamente por el Product Owner, siendo este quien se encarga de ordenar estos ítems por prioridad, la cual determinará el orden en que el equipo de desarrollo los toma para su realización. Se debe tener en cuenta también que este backlog se trata de un elemento “vivo”, que puede mutar a través del tiempo, a medida que vamos aprendiendo a través de las entregas iterativas y recibiendo feedback.

El **Sprint Backlog** es el conjunto de PBIs que fueron seleccionados para trabajar durante un sprint (explicado a continuación) en particular, conjuntamente con las tareas que de ellos se desprenden y que el equipo ha identificado que debe realizar para lograr un incremento funcional potencialmente entregable al final.

Un **incremento funcional potencialmente entregable** es el resultado que se obtiene al final de cada Sprint. Se dice que es un “incremento funcional” porque se trata de una funcionalidad nueva o modificada del producto que se está construyendo, y “potencialmente entregable” porque estas caracteristicas estan lo suficientemente validadas y verificadas como para ser llevadas a producción, si fuera el caso (que no siempre lo es, por eso lo de “potencialmente”).

En nuestra metodología ad-hoc respetaremos y daremos uso a todos los elementos antes mencionados.

A.3 Flujo de trabajo en Scrum

En Scrum, un **Sprint** es como se conocen a las iteraciones. Al tratarse de una metodología ágil, se basa en un enfoque incremental e iterativo, lo que quiere decir que el producto se va construyendo en base a incrementos funcionales entregados en periodos de tiempo cortos y constantes, para obtener un feedback frecuente. Se suele recomendar que los Sprints duren entre 1 y 4 semanas, siendo 2 o 3 semanas lo más habitual en la industria. Esta es una de las decisiones que se deben tomar al comienzo de un proyecto, ya que la idea es mantener esta duración a lo largo del mismo.

En nuestro caso, al carecer de experiencia en la metodología como para tener un mejor criterio, hemos decidido utilizar Sprints de **2 semanas**, tratando de apegarnos a las recomendaciones y a los estándares de la industria.

El **Sprint Planning Meeting** o reunión de planificación del Sprint se realiza al comienzo de cada Sprint con el objetivo de definir el alcance del mismo.

Suele dividirse en 2 partes con distintas finalidades:

La primera parte es estratégica, centrada en “qué” es lo que se va a hacer. En esta participan tanto el Product Owner, exponiendo todos los PBIs que podrían pasar a formar parte del Sprint, así como el equipo de desarrollo, quienes evacuan todas sus dudas e inquietudes para poder corroborar o ajustar las estimaciones. El Scrum Master también participa, encargándose de facilitar la reunión y de asegurar la presencia de cualquier stakeholder que sea requerido si se necesitará profundizar en algún detalle.

El resultado de esta reunión es un conjunto de PBIs que el equipo de desarrollo se compromete a realizar para generar un incremento funcional potencialmente entregable.

La segunda parte es táctica, girando alrededor del “cómo” será realizado el trabajo. Durante este espacio de tiempo solo están presentes el equipo de desarrollo y el Scrum Master (el Product Owner no suele participar, pero se lo podría contactar en caso de que el equipo necesite respuestas a nuevas preguntas que aparezcan). Aquí se realizará una definición inicial de un diseño de alto nivel, el cual será refinado a lo largo del Sprint, así como la identificación de las actividades que el equipo deberá llevar a cabo.

Se espera que el diseño sea emergente, que surja de las necesidades del equipo a medida que éste avance en el conocimiento del negocio. Por eso mismo no hace falta un diseño completo en este momento. Por el contrario, se buscará un acuerdo de alto nivel que irá bajando en detalle durante la ejecución del Sprint. Algo parecido sucede con las actividades, es imposible enumerar absolutamente todas al principio porque muchas de ellas surgirán a medida que se avance en la iteración.

La salida de esta actividad es un Sprint Backlog que representa el alcance del Sprint en cuestión (el cual suele colocarse en un taskboard o pizarra de actividades). Desde ese momento se da comienzo al desarrollo propiamente.

Teniendo en cuenta las adaptaciones realizadas a los roles para la realización de este proyecto, este evento Scrum será llevado adelante de la mejor forma posible, viéndose, tal vez, difusa la división entre ambas partes de dicha reunión por la “mezcla” de roles, pero tratando en la medida de lo posible de respetarlas. En conclusión, las decisiones en estas reuniones serán tomadas en conjunto entre ambos integrantes del grupo y, en caso de ser necesario, incorporaremos al director del proyecto y algún representante de la empresa cliente para evacuar dudas.

El **Daily Scrum** o Scrum diario, como su nombre lo indica, son reuniones que se realizan todos los días, las cuales no deben durar más de 15 minutos, con los objetivos de incrementar la comunicación, explicitar los compromisos asumidos y dar visibilidad a los impedimentos que surjan del trabajo que está siendo realizado. Habitualmente participan el equipo de desarrollo y el Scrum Master (quien una vez más es el encargado de facilitarla), pudiendo requerir de la presencia del Product Owner o de algún stakeholder, en caso de ser necesario. De todas maneras la idea es que la reunión sea abierta y que cualquier interesado en escuchar pueda participar en calidad de observador.

La mecánica se basa en que cada uno de los participantes toman turnos para responder las siguientes 3 preguntas:

- ¿Qué hice desde la última reunión diaria hasta ahora?
  Su finalidad es verificar el cumplimiento de los compromisos asumidos.
- ¿En qué voy a estar trabajando desde ahora hasta la próxima reunión diaria?
  El objetivo de esta pregunta es generar nuevos compromisos de cara al futuro, siendo estos asumidos ante sus compañeros.
- ¿Qué problemas o impedimentos tengo?
  Aquí se apunta a detectar y visibilizar impedimentos, los cuales no se resuelven en esta reunión, sino en posteriores, siendo responsabilidad del Scrum Master que se llegue a dicha resolución lo antes posible, organizando las reuniones que sean necesarias con las personas correctas.

Llevado al desarrollo de este proyecto, no vemos la necesidad de realizar una reunión diaria “formal” (todos los días, en un horario concreto), por el hecho de tratarse de un equipo conformado por 2 individuos, donde la comunicación se da de manera mucho más fluida. Nuestra idea sería mantenernos siempre informados sobre el avance diario de las tareas que el otro integrante está llevando adelante, principalmente mediante texto (WhatsApp) y, eventualmente, videollamadas (Discord, Google Meet, etc.).

Por otro lado, realizaremos reuniones de soporte o planificación solo en caso de que alguno encuentre algún impedimento, inconveniente u obstáculo en el desarrollo de determinada tarea para tratar de darle una pronta solución al mismo (por ejemplo, dudas técnicas de implementación o alguna solución de un inconveniente específico).

Al finalizar cada Sprint se celebra la reunión de revisión o **Sprint Review**, donde se revisa el elemento funcional potencialmente entregable desarrollado por el equipo de desarrollo. En esta reunión tanto los stakeholders como el equipo Scrum revisan el resultado del Sprint, siendo los primeros los encargados de aceptar o rechazar las funcionalidades construidas, devolviendo por supuesto el feedback correspondiente,

pudiendo también proponer cambios o nuevas funcionalidades que surjan al ver el producto funcionando, las que se traducen como nuevos PBIs en el Product Backlog. En caso de que una funcionalidad sea rechazada, su PBI correspondiente reingresa el Product Backlog con la máxima prioridad (salvo que Product Owner diga lo contrario), con el objetivo de ser tratado en el Sprint siguiente.

Esta actividad será respetada y realizada durante el desarrollo del presente proyecto. El producto funcional potencialmente entregable para este proyecto serán nuevas funcionalidades o avances de nuevas características de la aplicación web desarrollada. Es decir, cada vez que finalice un sprint, existirá un nuevo número de historias de usuario finalizadas con vistas a ser funcionales para el usuario. Con respecto a estos artefactos resultantes de cada sprint, cabe aclarar que no siempre serán destinados al cliente final, ya que algunos avances pueden llegar a ser solo mostrados al product owner, quién tendrá la facultad de constatar si éstos cumplen o no con las expectativas finales.

Luego del Sprint Review se da lugar al **Sprint Retrospective**. Su propósito es el de generar un espacio para la mejora continua, donde el equipo reflexiona sobre la forma en que se realizó el trabajo, destacando que fue lo que se hizo bien, las fortalezas, y en qué campos se podría mejorar para el siguiente Sprint, tomándolo no como debilidades, sino como oportunidades de mejora. Es vital contar con un ambiente seguro, donde todos puedan expresarse libremente, por lo cual se suele restringir la participación solo al equipo de desarrollo y al Scrum Master, pudiendo sumar al Product Owner o a los stakeholders si fuera necesario. Al final el equipo decide cuáles son las acciones de mejora a efectuar, las cuales serán evaluadas en la siguiente retrospectiva.

Esta actividad será respetada y realizada durante el desarrollo del presente proyecto. Más aún, creemos que, debido a nuestra escasa experiencia en el desarrollo de metodologías ágiles, será de gran provecho la correcta realización de esta actividad para mejorar nuestro trabajo sprint a sprint.

Por último, el **Refinamiento del Product Backlog** es una actividad constante, que se lleva a la práctica a lo largo de todo el Sprint (aunque algunos equipos prefieren concentrarse en una reunión). Su objetivo es indagar en el entendimiento de los PBIs que se encuentran fuera del Sprint actual, idealmente los que potencialmente pasarían a formar parte de los próximos 2 o 3 Sprints, y de ser posible, dividirlos en PBIs más pequeños y estimarlos, si hiciera falta. En caso de ser realizado como una reunión, la responsabilidad

de convocarla es del Product Owner, una o dos veces por Sprint, y siendo facilitadas nuevamente por el Scrum Master.

En nuestro caso optamos por llevar adelante esta actividad de forma continua, solo realizando reuniones para este fin si lo viéramos necesario.

ANEXO B - Actividades a ser realizadas

B.1 Introducción

Las actividades realizadas durante el proyecto fueron detalladas mediante la utilización de historias de usuario, las cuales se presentarán formalmente a lo largo del presente anexo. También puede accederse a un listado breve de las mismas en la sección 4.2 de este informe.

El orden de ejecución de las historias de usuario fue definido anteriormente, basándonos, como allí comentamos, en las dependencias funcionales, al no tener definidas prioridades de negocio por parte del cliente, lo que nos dio la libertad para iniciar el desarrollo por las historias que no dependian de otras para su funcionamiento.

Dentro de las historias, los roles utilizados fueron los de administrador, consignatario y asistente, los cuales cumplen con las siguientes responsabilidades:

Administrador: al cliente se le asignará un usuario con este rol para que luego cree el resto de los usuarios, con sus roles respectivos (incluyendo otros administradores), como también modificarlos. Se lo considera como un “superusuario” que tiene permitido realizar cualquier acción disponible en el sistema, incluso dentro de cualquier remate, aunque no sea creado por él, ni añadido al mismo por otro usuario (situación que de todas maneras no puede darse, justamente por el hecho de que ya tiene acceso).

Consignatario: Crea y accede a remates propios, a toda la información relacionada y asigna participantes para los mismos, pudiendo ser asistentes u otros consignatarios (adquiriendo estos últimos los mismos permisos que el creador), así como es el encargado de terminar un remate o eliminarlo en caso de que sea necesario. También tiene la posibilidad de crear/editar/eliminar localidades y categorías de animales, así como de acceder al listado de usuarios del sistemas.

Asistente: se encarga de crear clientes y de tareas específicas durante los remates donde fue asignado, como por ejemplo, agregar lotes, completar planillas, imprimir boletas, cargar datos de animales vendidos, entre otros.

Es importante aclarar que cada uno de los roles definidos anteriormente se presentan ordenados según su nivel de privilegios, de mayor a menor, lo que significa que cada rol posee los permisos de su sucesor y añade nuevas capacidades.

B.2 Refinación de historias de usuario

B.2.1 Historia de Usuario 1 - Log in

Historia de usuario	
Número: 1	**Nombre**: Log in
Usuario: consignatario, asistente, administrador.	
Modificación Historia: N/A	**Puntos Estimados:** 6
Descripción: Cuando el usuario ingresa a la página web, se mostrará una pantalla de login, donde el mismo ingresará su usuario y contraseña para poder entrar al sistema.	
Observaciones: También, ya dentro del sistema, tendrá un botón para salir.	
Criterio de aceptación: Se obtiene el token de acceso del usuario y éste logra acceder a la pantalla principal del sistema.	

B.2.2 Historia de Usuario 2 - Información del perfil

Historia de usuario	
Número: 2	**Nombre**: Información del perfil
Usuario: consignatario, asistente, administrador	
Modificación Historia: N/A	**Puntos Estimados:** 6
Descripción: El usuario podrá ingresar a una pantalla donde visualizará la información de su perfil y tenga la posibilidad de editar la misma. La información visualizada por el usuario será: nombre, usuario, contraseña y rol.	
Observaciones: Excepto el rol, todos los campos son modificables.	
Criterio de aceptación: En la pantalla se muestra la información correcta respectiva al usuario. Los datos del usuario (username y name) son modificados y guardados con éxito en la base de datos. La contraseña es modificada y guardada con éxito en la base de datos.	

B.2.3 Historia de Usuario 3 - CRUD de usuarios

Historia de usuario	
Número: 3	**Nombre**: CRUD de usuarios
Usuario: administrador.	
Modificación Historia: N/A	**Puntos Estimados:** 8
Descripción: Se presentará una pantalla a los administradores con la lista de usuarios existentes en el sistema, teniendo la posibilidad de crear, eliminar o modificar un usuario. Los datos a ingresar para un nuevo usuario son los siguientes: nombre, usuario, contraseña y rol.	
Observaciones: En principio, va a haber un primer usuario administrador, creado externamente por los desarrolladores, el cual se le va a proporcionar a la empresa cliente para que en base al mismo se puedan crear todos los demás usuarios. Una vez creado un usuario, su rol no podrá ser modificado (porque si un usuario consignatario pasa a asistente, por ejemplo, perderá la visibilidad de historial de todos sus remates).	
Criterio de aceptación: **Create/update:** verificar que los cambios se registraron correctamente en la base de datos. **Remove:** verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el usuario esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro). **Read:** mostrar correctamente el listado de usuarios.	

B.2.4 Historia de Usuario 4 - CRUD clientes

Historia de usuario	
Número: 4	**Nombre**: CRUD clientes
Usuario: administrador, asistente, consignatario.	
Modificación Historia: N/A	**Puntos Estimados:** 8
Descripción: El usuario podrá acceder a una pantalla donde se listen los clientes dentro del sistema. También tendrá la posibilidad de crear nuevos clientes o seleccionar a uno para poder acceder a su información, editarla o eliminar al mismo del sistema.	
Observaciones: Los datos que un usuario puede visualizar o editar de un cliente son: ● Denominación. ● Cuit. ● Procedencias (pudiendo agregar nuevas con su nombre, número de Renspa y localidad).	
Criterio de aceptación: **Create/update:** verificar que los cambios se registraron correctamente en la base de datos. **Remove:** verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el cliente esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro). **Read:** mostrar correctamente el listado de clientes.	

B.2.5 Historia de Usuario 5 - CRUD localidades

Historia de usuario	
Número: 5	**Nombre**: CRUD localidades
Usuario: administrador, consignatario.	
Modificación Historia: N/A	**Puntos Estimados:** 4
Descripción: El usuario podrá acceder a una pantalla donde se listen las localidades existentes dentro del sistema. En esta pantalla tendrá la posibilidad de crear nuevas localidades así como también seleccionar una localidad para poder editarla o eliminarla.	
Observaciones: Para crear la localidad solo se ingresará un dato que represente el nombre de la misma.	
Criterio de aceptación: **Create/update:** verificar que los cambios se registraron correctamente en la base de datos. **Remove:** verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que la localidad esté marcada como "deshabilitado" en la base de datos (decisión de implementación a futuro). **Read:** mostrar correctamente el listado de localidades.	

B.2.6 Historia de Usuario 6 - CRUD categoría animales

Historia de usuario	
Número: 6	**Nombre**: CRUD categoría animales
Usuario: administrador, consignatario.	
Modificación Historia: N/A	**Puntos Estimados:** 4
Descripción: El usuario podrá acceder a una pantalla donde se listen las categorías de animales existentes dentro del sistema. En esta pantalla tendrá la posibilidad de crear nuevas categorías así como también seleccionar una para poder editarla o eliminarla.	
Observaciones: Para crear la categoría solo se ingresará un dato que represente el nombre de la misma.	
Criterio de aceptación: **Create/update:** verificar que los cambios se registraron correctamente en la base de datos. **Remove:** verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que la categoría esté marcada como "deshabilitado" en la base de datos (decisión de implementación a futuro). **Read:** mostrar correctamente el listado de categoría de animales.	

B.2.7 Historia de Usuario 7 - CRUD remates

Historia de usuario	
Número: 7	**Nombre**: CRUD remates
Usuario: administrador, consignatario.	
Modificación Historia: N/A	**Puntos Estimados:** 10
Descripción: Se presenta una pantalla donde el usuario podrá crear un nuevo remate, para lo cual debe ingresar los siguientes datos: número de Senasa, fecha de remate y localidad donde se realizará. Estos datos pueden ser modificados posteriormente o, también, el remate puede ser eliminado.	
Observaciones: Un remate no podrá ser editado ni eliminado luego de que éste haya sido “finalizado”. Para finalizar un remate, existirá un botón con dicho propósito.	
Criterio de aceptación: **Create/update:** verificar que los cambios se registraron correctamente en la base de datos. **Remove:** verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el remate esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro). **Read:** mostrar correctamente el listado de remates.	

B.2.8 Historia de Usuario 8 - Asignar asistentes a un remate

Historia de usuario	
Número: 8	**Nombre**: Asignar asistentes a un remate
Usuario: administrador, consignatario.	
Modificación Historia: No se pueden agregar administradores al remate, ya que todos los usuarios con este rol tienen acceso desde que el mismo se crea.	**Puntos Estimados:** 4
Descripción: El usuario, luego de haber creado un remate, podrá ingresar a ver el mismo y tendrá un botón que le permita agregar nuevos participantes (comúnmente asistentes) para gestionar el remate. Luego de presionar el botón, se presentará una pantalla que le permita buscar usuarios por sus nombres y pueda ir agregandolos al remate.	
Observaciones: Si bien lo usual es agregar usuarios asistentes, también podría agregar otros consignatarios (dándole todos los permisos sobre el remate, pudiendo, por ejemplo, eliminarlo, aun sin ser el creador del mismo) o administradores, aunque este último caso es redundante, ya que los usuarios con este rol de por sí ya tendrían acceso al remate en cuestión (como a todos los remates en el sistema).	
Criterio de aceptación: Cuando el usuario asignado a un remate ingresa al sistema, debe tener acceso para gestionar el remate al que fue asignado.	

B.2.9 Historia de Usuario 9 - Cargar lotes de venta a un remate

Historia de usuario	
Número: 9	**Nombre**: Cargar lotes de venta a un remate
Usuario: administrador, consignatario, asistente	
Modificación Historia: N/A	**Puntos Estimados:** 10
Descripción: El usuario puede cargar lotes de venta a un remate. Para esto, debe ingresar a ver el remate y “clickear” un botón que le mostrará una nueva pantalla en la cual ingresará los siguientes datos: vendedor (cliente), procedencia, N° DT-e (documento de tránsito electrónico), N° de corral, animales (según su categoría y cantidad).	
Observaciones: Desde la pantalla del remate, el usuario puede visualizar los lotes de venta cargados y seleccionar alguno si desea editar su información o eliminarlo (si aún no fue vendido ningún animal).	
Criterio de aceptación: Cuando acceda a la información del remate, el usuario deberá poder observar el nuevo lote agregado en la lista de lotes de venta. En caso de edición de un lote de venta, esta modificación deberá verse reflejada en la lista de los lotes a vender. En caso de eliminación, deberá desaparecer el registro (junto a sus relaciones, si es necesario), de la respectiva tabla en la base de datos. O bien, que el lote esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro).	

B.2.10 Historia de Usuario 10 - Generar PDF del orden de salida de animales

Historia de usuario	
Número: 10	**Nombre**: Generar PDF del orden de salida de animales
Usuario: administrador, consignatario, asistente	
Modificación Historia: N/A	**Puntos Estimados:** 2
Descripción: El usuario tendrá acceso a un botón, dentro de la pantalla de un remate, que le permita descargar la lista de lotes de animales que se venderán en el mismo, ordenados según el orden de salida a la pista de los mismos.	
Observaciones: Cada ítem de la lista tendrá los siguientes datos: categoría de animales, cantidad de animales y corral donde se encuentra.	
Criterio de aceptación: Archivo en formato PDF descargado correctamente.	

B.2.11 Historia de Usuario 11 - Cargar datos de lotes vendidos

Historia de usuario	
Número: 11	**Nombre**: Cargar datos de lotes vendidos
Usuario: administrador, consignatario, asistente	
Modificación Historia: Los lotes no vendidos pasan a un listado aparte (ver observación).	**Puntos Estimados:** 10
Descripción: Al venderse un lote de animales, se le presenta al usuario (usualmente un asistente) una pantalla donde podrá cargar los datos relativos a la venta, lo cuales son: comprador, precio, cantidad de animales y si el lote se pesa o no. En caso de que el lote deba pesarse, un nuevo usuario (que está en la balanza) cargará esta información al lote vendido. De otra manera, se guarda el nuevo lote sin tener que ingresar información sobre el peso. En caso de no venderse un lote de animales (completo o parte del mismo), existirá un botón para marcar ese lote como no vendido.	
Observaciones: Cuando un lote sea marcado como no vendido, se mantendrá en el listado de “lotes para la venta” por si se da el caso en que se venda luego del remate, ubicandolo al final de este listado.	
Criterio de aceptación: En caso de vender un lote, el mismo deberá aparecer en el listado de lotes vendidos, con los datos correctos respectivos a la venta. En caso de marcar como “no vendido” a un lote, dicho registro deberá aparecer como tal en la base de datos.	

B.2.12 Historia de Usuario 12 - Cargar números de DTe

Historia de usuario	
Número: 12	**Nombre**: Cargar números de DTe
Usuario: administrador, consignatario, asistente	
Modificación Historia: N/A	**Puntos Estimados:** 6
Descripción: Al terminar un remate, el usuario carga los números de DTe de los lotes, tanto aquellos que fueron vendidos como de los que no.	
Observaciones: Para finalizar un remate, existirá un botón con dicho propósito. Esto hará que no se puedan seguir agregando o editando datos de ventas de animales y sólo permitirá que se puedan imprimir boletas de lotes vendidos o cargar el DT-e.	
Criterio de aceptación: Los lotes a los cuales se les cargó el número de DT-e deberán tener ese dato guardado en la base de datos, así como mostrar dicho número en el listado de todos los lotes finales que surgieron luego de la realización de un remate.	

B.2.13 Historia de Usuario 13 - Historial de remates

Historia de usuario	
Número: 13	**Nombre**: Historial de remates
Usuario: administrador, asistente, consignatario.	
Modificación Historia: N/A	**Puntos Estimados:** 4
Descripción: Se le presenta al usuario una pantalla con un historial de los remates ya realizados en los que haya participado (los administradores, a todos los remates del sistema), a los cuales puede acceder a ver la información respectiva a los mismos.	
Observaciones:	
Criterio de aceptación: Se le muestra al usuario un listado de remates, los cuales deben estar relacionados al mismo y deben estar ya finalizados.	

B.2.14 Historia de Usuario 14 - Listado de remates

Historia de usuario	
Número: 14	**Nombre**: Listado de remates
Usuario: consignatario, asistente, administrador	
Modificación Historia: Los remates propios y ajenos están en dos listados separados.	**Puntos Estimados:** 6
Descripción: Cuando el usuario se loguea, se mostrará la lista de próximos remates en los que participará, ordenados por fecha más próxima de realización. El usuario puede seleccionar uno de estos remates para poder ver su información y cargarle nueva información, como: agregar nuevos lotes de venta, vender animales (durante la realización del remate) y/o pesar animales (durante la realización del remate). A continuación, también se listarán remates próximos ajenos, mostrando información respectiva como fecha y hora, localidad y representante/consignatario (creador del remate).	
Observaciones: A los remates ajenos solo se los puede ver en el listado con la información antes mencionada, pero no acceder a los mismos.	
Criterio de aceptación: Cuando el usuario se loguea exitosamente, la pantalla debe mostrar una lista de remates en el siguiente orden: primero, remates próximos en los que participa, ordenados ascendentemente según la fecha de realización, con la posibilidad de ingresar a gestionar cada uno; segundo, los remates en los que no participa (creados por otros usuarios y donde no fue agregado como colaborador), ordenados ascendentemente por fecha de realización, sin la posibilidad de ingresar a la gestión de éstos.	

B.2.15 Historia de Usuario 15 - Reportes estadísticos de un remate

Historia de usuario	
Número: 15	**Nombre**: Reportes estadísticos de un remate
Usuario: consignatario, asistente, administrador	
Modificación Historia: N/A	**Puntos Estimados:** 10
Descripción: El usuario podrá acceder a un listado de todos los remates en los que participó (los administradores, a todos los remates del sistema) y seleccionar alguno para poder ver información y reportes estadísticos sobre el mismo.	
Observaciones: Información sobre el remate: N° Senasa, fecha de realización, lugar de realización. , listado de animales vendidos (vendedor, comprador, cantidad, peso, precio). Reportes estadísticos sobre el remate: ● General: ○ Total de animales ingresados. ○ Total de animales vendidos y sin vender. ○ Ingreso monetario total. ○ Cantidad de vendedores y compradores. ○ Lista de lotes vendidos, por comprador y cantidad de animales. ○ Lista de lotes sin vender, por vendedor y cantidad de animales. ○ Ranking de compradores, por cantidad de animales comprados en total, por cantidad de dinero pagado ○ Ranking de vendedores, por cantidad de animales ofertados, por cantidad de dinero obtenido. ● Por categoría de animales: ○ Total de animales ingresados. ○ Total de animales vendidos y sin vender. ○ Ingreso monetario total. ○ Ranking de compradores, por cantidad de animales comprados en total, por cantidad de dinero pagado	

○ Ranking de vendedores, por cantidad de animales ofertados, por cantidad de dinero obtenido.
Criterio de aceptación: todos los datos mostrados dentro de los reportes deberán ser consistentes con los datos reales que surgieron en la realización de un remate (los cuales se encuentran almacenados en la base de datos).

B.2.16 Historia de Usuario 16 - Impresión reportes estadísticos

Historia de usuario	
Número: 16	**Nombre**: Impresión reportes estadísticos
Usuario: consignatario, asistente, administrador	
Modificación Historia: Se agrega la opción de incluir también un listado de los lotes vendidos.	**Puntos Estimados:** 10
Descripción: Los reportes estadísticos serán mostrados paginados según ciertos criterios y, en cada página, el usuario tendrá la opción de descargarse la información de la misma en formato PDF.	
Observaciones: Reportes estadísticos sobre el remate: ● General (Página 1): ○ Total de animales ingresados. ○ Total de animales vendidos y sin vender. ○ Ingreso monetario total. ○ Cantidad de vendedores y compradores. ○ Lista de lotes vendidos, por comprador y cantidad de animales. ○ Lista de lotes sin vender, por vendedor y cantidad de animales. ○ Ranking de compradores, por cantidad de animales comprados en total, por cantidad de dinero pagado	

○ Ranking de vendedores, por cantidad de animales ofertados, por cantidad de dinero obtenido. ● Por categoría de animales (una página por categoría): ○ Total de animales ingresados. ○ Total de animales vendidos y sin vender. ○ Ingreso monetario total. ○ Ranking de compradores, por cantidad de animales comprados en total, por cantidad de dinero pagado ○ Ranking de vendedores, por cantidad de animales ofertados, por cantidad de dinero obtenido.
Criterio de aceptación: deberá descargarse correctamente el archivo en formato PDF de cada página que decida ser descargada.

B.2.17 Historia de Usuario 17 - Generación boletas de ventas

Historia de usuario	
Número: 17	**Nombre**: Generación boletas de ventas.
Usuario: consignatario, asistente, administrador	
Modificación Historia: N/A	**Puntos Estimados:** 8
Descripción: Luego de que un lote de animales es vendido (y pesado, en caso de que sea necesario), el usuario podrá presionar un botón que permita generar y descargar la boleta de venta correspondiente, en formato PDF, como comprobante de la realización de la misma. Dicha boleta deberá contener al menos, la siguiente información: denominación del vendedor y del comprador, fecha y lugar de realización del remate, n° de corral, cantidad de animales comprados, precio, kg (en caso de ser pesado) e información sobre el organizador del remate.	

Observaciones:

El usuario selecciona la cantidad de copias que desee imprimir, siendo lo más usual generarse por cuadruplicado, una para el vendedor, otra para el comprador, una para control de la empresa y la restante para Senasa (Servicio Nacional de Sanidad y Calidad Agroalimentaria).

Criterio de aceptación:

Descargar correctamente el archivo en formato PDF de la boleta de venta, con la información respectiva a la misma consistente y el número correcto de copias solicitadas por el usuario.

ANEXO C - Gestión de riesgos

C.1 Introducción

La Gestión de Riesgos es la identificación, evaluación y control de las incertidumbres que puedan dar lugar a retrasos en las actividades planificadas generando desvíos u otras consecuencias no deseadas.[18]

Los objetivos de la gestión de riesgos son:

- Enfocar la atención en minimizar las amenazas para que los proyectos puedan cumplir con sus objetivos.
- Proporcionar un enfoque sistemático para:
    - Identificar y evaluar los riesgos de los Proyectos.
    - Determinar acciones rentables para la reducción de los riesgos.
    - Dar seguimiento y reporte en el progreso de los riesgos al nivel adecuado.

A continuación, mostraremos la forma de clasificar los riesgos identificados según su probabilidad de ocurrencia e impacto (gravedad de las consecuencias originadas por la materialización de un riesgo). Esto permite asignar a cada uno una escala numérica, lo cual facilita la comprensión y nos podemos centrar en el impacto “real” para la priorización de las actividades destinadas a la reducción del riesgo.

C.2 Clasificación de riesgos según probabilidad e impacto

Probabilidad		
Escala	Descripción	Ponderación
Muy baja	Hasta un 10% de probabilidad de ocurrencia	0,10
Baja	Hasta un 30% de probabilidad de ocurrencia	0,30
Media	Hasta un 50% de probabilidad de ocurrencia	0,50
Alta	Hasta un 70% de probabilidad de ocurrencia	0,70
Muy alta	Hasta un 90% de probabilidad de ocurrencia	0,90

Impacto		
Escala	Descripción	Ponderación
Muy bajo	La ocurrencia del riesgo tiene una mínima influencia en los costos y tiempos.	10
Bajo	Interrupción progresiva y manejable en costos y tiempos de parte del equipo del proyecto a corto plazo.	30
Medio	Interrupción progresiva con incrementos de los tiempos del plan y los costos de corto y mediano plazo.	50
Alto	Interrupción significativa en el plan, costos, y productos de mediano y largo plazo.	70
Muy alto	Interrupción muy significativa en el plan, costos, y productos de mediano y largo plazo, pudiendo paralizar el proyecto.	90

Los riesgos identificados serán clasificados en “Bajo”, “Moderado” o “Alto” según su exposición al riesgo (es decir, la multiplicación de la probabilidad de ocurrencia de un riesgo por el impacto de la presencia del mismo), mostrándose a continuación cómo proponemos dicha clasificación.

Clasificación de riesgos	
Exposición al riesgo	Clasificación
[0; 15)	BAJO
[15; 45)	MODERADO
[45; 81]	ALTO

C.3 Clasificación de riesgos identificados

Riesgos				
N°	Motivo	Probabilidad de ocurrencia	Impacto en el proyecto	Exposición al Riesgo
1	Imposibilidad de cumplir con la cantidad de horas de trabajo semanales	Media	Muy alto	45
2	Cambios en los requisitos al principio del desarrollo del proyecto	Baja	Alto	21
3	Subestimar el esfuerzo necesario, llevando a un incumplimiento de plazos	Muy alta	Alto	63
4	Falta de experiencia con las tecnologías	Media	Medio	25
5	Un integrante del grupo se ausenta temporalmente (enfermedad, vacaciones, etc.)	Baja	Alto	21
6	Problemas relacionados al control de versiones	Media	Bajo	15
7	Pérdida de documentación generada	Muy baja	Medio	5
8	Problemas con los entornos de desarrollo	Media	Bajo	15
9	Cambios en los requisitos en medio del desarrollo del proyecto	Baja	Medio	15
10	Cambios en los requisitos al final del desarrollo del proyecto	Baja	Baja	9
11	Rechazo de una funcionalidad por parte de los stakeholders	Baja	Medio	15
12	Un integrante del grupo abandona el proyecto	Muy baja	Muy alto	9
13	Atrasos en el desarrollo de una funcionalidad	Media	Medio	25
14	Imposibilidad de concretar alguna de las reuniones en la fecha acordada	Baja	Bajo	9
15	Retrasos por problemas de hardware (rupturas, pérdidas, robos de equipos)	Baja	Alto	21
16	Problemas con la conexión a internet o cortes de electricidad	Media	Bajo	15

C.4 Plan establecido según riesgo

Hemos decidido generar planes de contingencia solo para los riesgos clasificados como “Moderado” y “Alto”. Aquellos clasificados como “Bajo” serán aceptados, ya sea porque difícilmente ocurrirán durante el desarrollo y/o porque no lo afectarán de sobremanera.

N°	Tipo de estrategia	Respuesta
1	Reducir riesgo	Diseñar el cronograma semanal de trabajo acorde a las posibilidades de cada integrante.
	Asumir riesgo	Replanificar las fechas de entrega o restar tareas del sprint en curso.
2	Asumir riesgo	Agregar los nuevos requerimientos (o las modificaciones) al backlog, alterando posiblemente las fechas de entrega.
3	Reducir riesgo	Realizar las reuniones necesarias con el director del proyecto previo a su inicio para llegar a tener una visión más realista.
	Asumir riesgo	Replanificar las fechas de entrega o reducir el alcance del proyecto, descartando funcionalidades de baja relevancia.
4	Reducir riesgo	Realizar cursos y/o capacitaciones sobre las tecnologías a utilizar previo al inicio del desarrollo del proyecto.
5	Asumir riesgo	Replanificar las fechas de entrega.
6	Reducir riesgo	Al momento de, por ejemplo, realizar un merge de ramas tener sumo cuidado y realizar consultas con el otro integrante del grupo si fuese necesario.
8	Reducir riesgo	Asegurarse entre los integrantes de usar, desde un principio, las mismas herramientas de trabajo/desarrollo y en las mismas versiones, sin modificarlas en el transcurso del proyecto.
9	Asumir riesgo	Agregar los nuevos requerimientos (o las modificaciones) al backlog, alterando posiblemente las fechas de entrega.
11	Asumir riesgo	Tomar esa funcionalidad como máxima prioridad y tratarla en el Sprint siguiente, alterando posiblemente las fechas de entrega.
13	Asumir riesgo	Continuar con esa funcionalidad en el Sprint siguiente, debiendo replanificar el Backlog del mismo y posiblemente las fechas de entrega.
15	Asumir riesgo	Replanificar las fechas de entrega o restar tareas del sprint en curso, según el tiempo perdido durante la reparación del daño.
16	Asumir riesgo	En general, esto retrasaría un período corto de tiempo al proyecto. Una opción sería recuperar dicho tiempo trabajando horas extras fuera de lo planificado.

ANEXO D - Ejecución del proyecto

D.1 Introducción

Con la intención de facilitar la lectura del presente documento, se optó por separar la ejecución del proyecto (contenida en este anexo) de la planificación del mismo, así como de los documentos generados durante éste (los cuales pueden encontrarse en el Anexo E).

Para presentar el resultado de la ejecución de la metodología de desarrollo, se definió la estructura de las subsecciones (una por Sprint) previamente, con el objetivo de estandarizar la información mostrada. La plantilla definida consta de:

- **Planificación**: presentará las fechas programadas de inicio y fin de cada una de las semanas del Sprint, así como las fechas de realización de los eventos propuestos por la metodología (Sprint Planning, Sprint Review y Sprint Retrospective) y las Historias de Usuario correspondientes a su Sprint Backlog.
- **Ejecución**: presentará el contraste entre lo planificado y lo realizado durante el desarrollo de ese Sprint, exponiendo su periodo de ejecución (fecha de inicio y fecha de fin), así como el tiempo total (medido tanto en Story Points como en horas) que requirió para llevarse a cabo, tanto a nivel Sprint como por cada Historia de Usuario. Cabe aclarar que, en la mayoría de los casos, la suma del tiempo empleado para cada Historia de Usuario no es igual al total del tiempo empleado durante el Sprint. Esto se da por el hecho de que este tiempo total, además de incluir el tiempo dedicado a cada una de las tareas programadas, también incluye tiempo invertido en actividades complementarias, como reuniones organizativas por ejemplo, por lo que este tiempo total es el tiempo “bruto” de trabajo durante ese periodo de tiempo.
- **Gestión de riesgos**: presentará los riesgos (contemplados y no contemplados) que se hayan disparado durante la ejecución del Sprint, junto al resultado de la ejecución de su plan de contingencia.
- **Actualización del Backlog**: una vez terminado el Sprint, se muestra el backlog resultante luego de quitar las Historias de Usuario concluidas.

D.2 Sprint 0

D.2.1 Planificación

En el cronograma de avance se definió que el Sprint 0 sería ejecutado entre las siguientes fechas:

Planificación por semanas de trabajo			
N°	**Desde**	**Hasta**	**Sprint**
1	25/10/2021	29/10/2021	0

Este Sprint particular fue planteado principalmente a modo organizativo, para llevar adelante tareas que debían realizarse previamente al inicio del proyecto propiamente. Presentamos a continuación un listado de las actividades realizadas.

D.2.2 Ejecución

A diferencia de los próximos Sprints, aquí no tenemos punto de comparación con lo planeado porque las labores llevadas adelante, al no tratarse de Historias de Usuario al uso (no definidas por el cliente), no fueron estimadas, por lo que nos limitamos simplemente a enumerarlas:

- Configurar entornos de desarrollo
    - Revisar versiones de Java
    - Revisar versiones de VS Code
    - Revisar versiones de IntelliJ
- Iniciar proyecto en Jira
- Configurar usuario en base de datos MySql
- Crear la base de datos inicial
- Crear proyecto Front-End (crear un nuevo proyecto utilizando npx y crear una primera versión del “menubar”)
- Crear proyecto Back-End
- Iniciar proyectos en GitHub
    - Crear repositorio para el Back-End
    - Crear repositorio para el Front-End

Todas estas tareas fueron realizadas dentro de la semana propuesta, por lo que el Sprint 1 pudo dar inicio en la fecha pactada.

D.3 Sprint 1

D.3.1 Planificación

En el cronograma de avance se definió que el Sprint 1 sería ejecutado entre las siguientes fechas:

Planificación por semanas de trabajo			
N°	**Desde**	**Hasta**	**Sprint**
2	01/11/2021	05/11/2021	1
3	08/11/2021	12/11/2021	

El Sprint Planning se llevó a cabo durante el día inicial del Sprint y tanto el Sprint Review como el Sprint Retrospective durante el primer día del siguiente Sprint (15/11/2021).

En base a la estimación y planificación de los sprints y teniendo en cuenta que se pudieron completar todas las tareas del Sprint anterior acorde a lo planificado, el Sprint Backlog para el presente Sprint quedó de la siguiente manera:

Planificación de sprints			
Número de historia	Story points	Horas	Sprint
1	6	36	1
2	6	36	1
5	4	24	1
6	4	24	1

D.3.2 Ejecución

A continuación se muestra el contraste entre lo planificado y lo que fue la ejecución de este Sprint:

	Planificado	**Realizado**
Periodo de ejecución	01/11/2021 - 12/11/2021	01/11/2021 - 12/11/2021*
Tiempo	120 horas o 20 Story Points	81.5 horas o 13.58 Story Points
Historias de usuario	1: 6 Story Points (36 horas) 2: 6 Story Points (36 horas) 5: 4 Story Points (24 horas) 6: 4 Story Points (24 horas)	1: 3.66 Story Points 2: 3.583 Story Points 5: 1.833 Story Points 6: 1.833 Story Points

*Como se comenta en el apartado de planificación, el desarrollo finalizó el 12/11/2021, pero tanto el Sprint Review como el Sprint Retrospective, durante el primer día del siguiente Sprint (15/11/2021).

D.3.3 Gestión de riesgos

Durante el desarrollo del presente Sprint se disparó un riesgo que no fue tenido en cuenta durante la fase de planificación, el cual fue, no tener en consideración tareas necesarias para el desarrollo, ya que gran parte del trabajo durante esta primera etapa fue

destinado a actividades que no fueron proyectadas a la hora de redactar la planificación: por ejemplo, desde el lado del Front-End, alrededor de un tercio del tiempo de este Sprint fue empleado en mejoras estéticas, que permanecieron como la base visual del sistema a lo largo del desarrollo, pero que no fueron previamente planteadas a la hora de planificar. Más allá de eso, este Sprint fue claramente sobreestimado, por lo que el siguiente no se vio afectado en absoluto y se pudo continuar con el trabajo de forma regular.

D.3.4 Actualización del Backlog

Habiéndose finalizado este Sprint, se procede a quitar del Backlog las historias concretadas, dando como resultado la siguiente tabla:

Número de historia	Story points	Horas	Sprint
7	10	60	2
14	6	36	2
3	8	48	3
4	8	48	3
8	4	24	3
9	10	60	4
11	10	60	4
10	2	12	5
12	6	36	5
13	4	24	5
17	8	48	5
15	10	60	6
16	10	60	6

D.4 Sprint 2

D.4.1 Planificación

En el cronograma de avance se definió que el Sprint 2 sería ejecutado entre las siguientes fechas:

Planificación por semanas de trabajo			
N°	**Desde**	**Hasta**	**Sprint**
4	15/11/2021	19/11/2021	2
5	22/11/2021	26/11/2021	

El Sprint Planning se llevó a cabo durante el día inicial del Sprint y tanto el Sprint Review como el Sprint Retrospective durante el 25/11/2021 (un día antes de finalizar el Sprint).

En base a la estimación y planificación de los sprints y teniendo en cuenta que se pudieron completar todas las tareas del Sprint anterior acorde a lo planificado, el Sprint Backlog para el presente Sprint quedó de la siguiente manera:

Planificación de sprints			
Número de historia	Story points	Horas	Sprint
7	10	60	2
14	6	36	2

D.4.2 Ejecución

A continuación se muestra el contraste entre lo planificado y lo que fue la ejecución de este Sprint:

	Planificado	**Realizado**
Periodo de ejecución	15/11/2021 - 26/11/2021	15/11/2021 - 25/11/2021
Tiempo	96 horas o 16 Story Points	61.5 horas o 10.25 Story Points
Historias de usuario	7: 10 Story Points (60 horas) 14: 6 Story Points (36 horas)	7: 5.75 Story Points 14: 3.33 Story Points

D.4.3 Gestión de riesgos

Durante la realización de este Sprint no fueron activados ninguno de los disparadores de riesgos identificados en la Gestión de riesgos.

D.4.4 Actualización del Backlog

Habiéndose finalizado este Sprint, se procede a quitar del Backlog las historias concretadas, dando como resultado la siguiente tabla:

Número de historia	Story points	Horas	Sprint
3	8	48	3
4	8	48	3
8	4	24	3
9	10	60	4
11	10	60	4
10	2	12	5
12	6	36	5

13	4	24	5
17	8	48	5
15	10	60	6
16	10	60	6

D.5 Sprint 3

D.5.1 Planificación

En el cronograma de avance se definió que el Sprint 3 sería ejecutado entre las siguientes fechas:

Planificación por semanas de trabajo			
N°	**Desde**	**Hasta**	**Sprint**
6	29/11/2021	03/12/2021	3
7	06/12/2021	10/12/2021	

El Sprint Planning se llevó a cabo durante el día inicial del Sprint y tanto el Sprint Review como el Sprint Retrospective durante el último día del mismo.

En base a la estimación y planificación de los sprints y teniendo en cuenta que se pudieron completar todas las tareas del Sprint anterior acorde a lo planificado, el Sprint Backlog para el presente Sprint quedó de la siguiente manera:

Planificación de sprints			
Número de historia	Story points	Horas	Sprint
3	8	48	3
4	8	48	3
8	4	24	3

D.5.2 Ejecución

A continuación se muestra el contraste entre lo planificado y lo que fue la ejecución de este Sprint:

	Planificado	**Realizado**
Periodo de ejecución	29/11/2021 - 10/12/2021	29/11/2021 - 10/12/2021
Tiempo	120 horas o 20 Story Points	57.5 horas o 9.58 Story Points
Historias de usuario	3: 8 Story Points (48 horas) 4: 8 Story Points (48 horas) 8: 4 Story Points (24 horas)	3: 3.33 Story Points 4: 4.166 Story Points 8: 1 Story Points

D.5.3 Gestión de riesgos

Durante la realización de este Sprint no fueron activados ninguno de los disparadores de riesgos identificados en la Gestión de riesgos.

D.5.4 Actualización del Backlog

Habiéndose finalizado este Sprint, se procede a quitar del Backlog las historias concretadas, dando como resultado la siguiente tabla:

Número de historia	Story points	Horas	Sprint
9	10	60	4
11	10	60	4
10	2	12	5
12	6	36	5
13	4	24	5
17	8	48	5
15	10	60	6
16	10	60	6

D.6 Sprint 4

D.6.1 Planificación

En el cronograma de avance se definió que el Sprint 4 sería ejecutado entre las siguientes fechas:

Planificación por semanas de trabajo			
N°	**Desde**	**Hasta**	**Sprint**
8	13/12/2021	17/12/2021	4
9	20/12/2021	24/12/2021	

El Sprint Planning se llevó a cabo durante el día inicial del Sprint, aunque en este caso, al ser dos historias complejas y fundamentales dentro del sistema, en esta reunión principalmente nos organizamos para la ejecución de la historia 9, debiendo luego realizar nuevamente una reunión el día 20/12/2021 para planear la ejecución de la historia 11. Tanto el Sprint Review como el Sprint Retrospective fueron llevados a cabo durante el día 23/12/2021.

En base a la estimación y planificación de los sprints y teniendo en cuenta que se pudieron completar todas las tareas del Sprint anterior acorde a lo planificado, el Sprint Backlog para el presente Sprint quedó de la siguiente manera:

Planificación de sprints			
Número de historia	Story points	Horas	Sprint
9	10	60	4
11	10	60	4

D.6.2 Ejecución

A continuación se muestra el contraste entre lo planificado y lo que fue la ejecución de este Sprint:

	Planificado	**Realizado**
Periodo de ejecución	13/12/2021 - 24/12/2021	13/12/2021 - 23/12/2021
Tiempo	120 horas o 20 Story Points	68 horas o 11.33 Story Points*
Historias de usuario	9: 10 Story Points (60 horas) 11: 10 Story Points (60 horas)	9: 7 Story Points 11: 10 Story Points

*Esta anomalía en la que el tiempo en “bruto” es menor a la suma del tiempo de las historias del Sprint es explicada al final del presente anexo, en la sección D.11 Resúmen de resultados

D.6.3 Gestión de riesgos

Durante la realización de este Sprint no fueron activados ninguno de los disparadores de riesgos identificados en la Gestión de riesgos.

D.6.4 Actualización del Backlog

Habiéndose finalizado este Sprint, se procede a quitar del Backlog las historias concretadas, dando como resultado la siguiente tabla:

Número de historia	Story points	Horas	Sprint
10	2	12	5
12	6	36	5
13	4	24	5
17	8	48	5
15	10	60	6
16	10	60	6

D.7 Sprint Extra

Como se comentó anteriormente en la sección de Estimación y planificación de los sprints, durante el periodo pactado de vacaciones se realizaron algunas tareas extra a las planeadas, las cuales no estaban planificadas dentro del alcance del proyecto, por lo que, al no contar con estimaciones, mostraremos sólo la ejecución de las mismas.

D.7.1 Ejecución

	Realizado
Periodo de ejecución	27/12/2021 - 05/01/2022
Tiempo	13 horas o 2.16 Story Points
Historias de usuario	Ordenar lotes de venta: 1 Story Points (6 horas) Reanudar remate: 0.583 Story Points (3.5 horas) Pantalla de listado de batches: 0.25 Story Points (1.5 horas) Diálogo de editar animales en pista en la pantalla de remate: 0.16 Story Points (1 horas)

D.8 Sprint 5

D.8.1 Planificación

En el cronograma de avance se definió que el Sprint 5 sería ejecutado entre las siguientes fechas:

Planificación por semanas de trabajo			
N°	**Desde**	**Hasta**	**Sprint**
10	17/01/2022	21/01/2022	5
11	24/01/2022	28/01/2022	

El Sprint Planning se llevó a cabo durante el día inicial del Sprint y tanto el Sprint Review como el Sprint Retrospective durante el primer día del siguiente Sprint (31/01/2021).

En base a la estimación y planificación de los sprints y teniendo en cuenta que se pudieron completar todas las tareas del Sprint anterior acorde a lo planificado, el Sprint Backlog para el presente Sprint quedó de la siguiente manera:

Planificación de sprints			
Número de historia	Story points	Horas	Sprint
10	2	12	5
12	6	36	5
13	4	24	5
17	8	48	5

D.8.2 Ejecución

A continuación se muestra el contraste entre lo planificado y lo que fue la ejecución de este Sprint:

	Planificado	**Realizado**
Periodo de ejecución	17/01/2022 - 28/01/2022	17/01/2022 - 28/01/2022*
Tiempo	120 horas o 20 Story Points	61 horas o 10.16 Story Points
Historias de usuario	10: 2 Story Points (12 horas) 12: 6 Story Points (36 horas) 13: 4 Story Points (24 horas) 17: 8 Story Points (48 horas)	10: 1.66 Story Points 12: 0.166 Story Points 13: 0.496 Story Points 17: 1.826 Story Points

Como se comenta en el apartado de planificación, el desarrollo finalizó el 28/01/2022 pero tanto el Sprint Review como el Sprint Retrospective durante el primer día del siguiente Sprint (31/01/2021).

D.8.3 Gestión de riesgos

Durante la realización de este Sprint no fueron activados ninguno de los disparadores de riesgos identificados en la Gestión de riesgos.

D.8.4 Actualización del Backlog

Habiéndose finalizado este Sprint, se procede a quitar del Backlog las historias concretadas, dando como resultado la siguiente tabla:

Número de historia	Story points	Horas	Sprint
15	10	60	6
16	10	60	6

D.9 Sprint 6

D.9.1 Planificación

En el cronograma de avance se definió que el Sprint 6 sería ejecutado entre las siguientes fechas:

Planificación por semanas de trabajo			
N°	**Desde**	**Hasta**	**Sprint**
12	31/01/2022	04/02/2022	6
13	07/02/2022	11/02/2022	

El Sprint Planning se llevó a cabo durante el día inicial del Sprint y tanto el Sprint Review como el Sprint Retrospective durante el día 19/02/2022.

Debemos notar que este Sprint duró alrededor de una semana extra a lo planificado, esto se debió a que, al ser el último Sprint, nos tomamos un tiempo extra para solucionar bugs/errores que fuimos encontrando al realizar pruebas del sistema como un todo, así como para dejar funcionando de forma correcta un Websocket que implementamos para la pantalla “Ver/Editar Remate”, con el objetivo de que el listado se actualice automáticamente al venderse animales durante un remate.

En base a la estimación y planificación de los sprints y teniendo en cuenta que se pudieron completar todas las tareas del Sprint anterior acorde a lo planificado, el Sprint Backlog para el presente Sprint quedó de la siguiente manera:

Planificación de sprints			
Número de historia	Story points	Horas	Sprint
15	10	60	6
16	10	60	6

D.9.2 Ejecución

A continuación se muestra el contraste entre lo planificado y lo que fue la ejecución de este Sprint:

	Planificado	Realizado
Periodo de ejecución	31/01/2022 - 11/02/2022	31/01/2022 - 19/02/2022
Tiempo	120 horas o 20 Story Points	70.5 horas o 11.75 Story Points
Historias de usuario	15: 10 Story Points (60 horas) 16: 10 Story Points (60 horas)	15: 7.16 Story Points 16: 2.166 Story Points

D.9.3 Gestión de riesgos

Durante la realización de este Sprint se activaron 2 de los riesgos identificados en la Gestión de riesgos:

1) Atrasos en el desarrollo de una funcionalidad:

El desarrollo de la historia de usuario correspondiente a generar reportes de los remates duró más tiempo de lo planificado debido a su complejidad y la falta de experiencia de los desarrolladores en este tipo de tareas (generación de reportes estadísticos). Tuvimos una etapa de pruebas donde encontramos muchas inconsistencias en los datos de reportes y, a medida que se planteaban nuevos escenarios, surgían otras inconsistencias. Por lo tanto, tal como se establece en el plan de contingencias, hemos seguido con esta funcionalidad. Sin embargo, como la misma pertenecía al último Sprint del proyecto, el resultado fue un atraso en la finalización de éste, en vez de replanificar las tareas de los próximos Sprints (ya que no existían próximos Sprints).

2) Falta de experiencia con las tecnologías:

Este riesgo se activó cuando decidimos implementar un Websocket que se conecte y “escuche” en la pantalla de *Ver/Editar Remate*, más precisamente, al momento de realizar una nueva venta de un lote, o al modificar/eliminar un lote vendido. Con el solo hecho de entrar a la pantalla, la misma se actualiza automáticamente con la nueva información ante los cambios descritos anteriormente. Esto facilita y agiliza de manera considerable el trabajo de los participantes de un remate, principalmente, aquellos que deben insertar nuevos datos a un lote vendido, como por ejemplo el peso de los animales, porque saben cómo y cuándo deben tratar la nueva información del sistema sin esperar avisos desde otros medios. Fue nuestra primera vez en utilizar este tipo de tecnología, por lo que las etapas de investigación y primeras pruebas de concepto tomaron un tiempo considerable. Además, al momento de implementarlo en nuestro Proyecto también tuvimos muchos problemas, principalmente en el lado del back-end, lo cual nos llevó más tiempo de lo esperado.

En este caso, debido a que la idea de esta funcionalidad surgió dentro del periodo de ejecución del proyecto, no se pudo activar el plan de contingencia planificado para capacitarnos e investigar sobre esta tecnología con anticipación. Sin embargo, utilizamos la

ventana de tiempo de “vacaciones” y, también, consideramos que valía la pena retrasar la finalización de este último Sprint en pos de agregar una funcionalidad que le daría mucho valor al sistema.

D.9.4 Actualización del Backlog

Habiéndose finalizado este Sprint, se procede a quitar del Backlog inicial las historias concretadas, dando como resultado un backlog vacío.

D.10 Arreglos y funcionalidades agregadas luego de la revisión

Una vez terminado el último Sprint, habiendo completado las tareas planeadas, nos reunimos con el Director de Proyecto para presentar los resultados obtenidos. De esta reunión surgieron sugerencias de mejora, principalmente sobre aspectos visuales y de experiencia de usuario, así como ciertos bugs/errores que no habían sido identificados con anterioridad. En esta última sección haremos un listado de estas nuevas tareas, de forma similar a lo presentado para el Sprint extra.

D.10.1 Ejecución

	Realizado
Periodo de ejecución	05/03/2022 - 10/03/2022
Tiempo	17 horas o 2.83 Story Points (aprox.)
Historias de usuario	**[bug]** Historial de remate: la fecha está en formato americano: 0.083 Story Points (0.5 horas) **[bug]** Número de lote nulo en Lotes No Vendidos: 0.083 Story Points (0.5 horas) **[mejora visual]** Quitar boton de dropdown en los autocomplete: 0.16 Story Points (1 hora) **[mejora visual]** Aplicar Media Queries para adaptar la aplicación a los distintos tamaños de pantalla: 0.66 Story Points (4 horas) **[mejora visual]** Agregar gráficos a la pantalla de resúmenes: 1.5 Story Points (9 horas) **[mejora visual]** Mostrar los botones “Resumen” y “Terminar remate” como disable si falta de pesar algún animal: 0.33 Story Points (2 horas)

D.11 Resúmen de resultados

A continuación se muestra, a modo de resumen, una tabla comparativa entre las estimaciones realizadas para cada una de las historias y su resultado obtenido:

Story	SPMartin	SPTomas	SPTotal	SPEstimado	Diferencia	Resultado
1	1	2.66	3.66	6	2.34	Sobreestimado
2	1	2.583	3.583	6	2.417	Sobreestimado
3	1	2.33	3.33	8	4.67	Sobreestimado
4	1	3.166	4.166	8	3.834	Sobreestimado
5	1	0.833	1.833	4	2.167	Sobreestimado
6	1	0.833	1.833	4	2.167	Sobreestimado
7	2	3.75	5.75	10	4.25	Sobreestimado
8	1	0	1	4	3	Sobreestimado
9	3	4	7	10	3	Sobreestimado
10	0.33	1.33	1.66	2	0.34	Sobreestimado
11	2	8	10	10	0	Correcto
12	0.166	0	0.166	6	5.834	Sobreestimado
13	0.166	0.33	0.496	4	3.504	Sobreestimado
14	2	1.33	3.33	6	2.67	Sobreestimado
15	1.5	5.66	7.16	10	2.84	Sobreestimado
16	0.166	2	2.166	10	7.834	Sobreestimado
17	0.66	1.166	1.826	8	6.174	Sobreestimado

Se procede también a presentar una tabla comparativa entre estimaciones y resultado final, ahora teniendo como objeto a los Sprints realizados:

Sprint	SP Suma	SP "en bruto"	SP Estimado	Resultado
1	10.909	13.58333333	20	Sobreestimado
2	9.08	10.25	16	Sobreestimado
3	8.496	9.583333333	20	Sobreestimado
4	17	11.33333333	20	Sobreestimado
5	4.148	10.16666667	20	Sobreestimado
6	9.326	11.75	20	Sobreestimado

"Sprint Extra"	Alrededor de 2 SP o 12 horas
"Sprint 7"	Alrededor de 2.83 SP o 17 horas

La columna “SP Suma” presenta solamente la suma del tiempo dedicado a la realización de las historias correspondiente al Sprint en cuestión, mientras que la columna “SP en bruto” muestra el total de horas de trabajo (presentado como Story Points) durante el Sprint, lo cual incluye reuniones y actividades extras, como de investigación de tecnologías a ser utilizadas o la corrección de bugs.

A modo de ejemplo, tomaremos la historia “Cargar datos de lotes vendidos”, que se planificó para el Sprint 4. Como parte de esta historia de usuario, incluimos la nueva funcionalidad del websocket, la cual se decidió incorporar luego de haberse ejecutado el Sprint 4. Entonces, por un lado, se puede notar una gran diferencia en el Sprint 5, ya que en ese momento, además de las reuniones recurrentes, dedicamos parte del tiempo a la investigación y puesta en marcha del websocket para la página de lotes vendidos, lo que conllevó alrededor de 4.66 Story Points (o 28 horas) extras, por eso la notoria diferencia entre ambas columnas. Por otro lado, también se puede observar que la cantidad de *SP suma* es superior a los *SP “en bruto”* de las historias planificadas para el Sprint 4, porque las horas dedicadas a la investigación e implementación del websocket son tenidas en cuenta en la historia de usuario “Cargar datos de lotes vendidos” y esto hace aumentar considerablemente el valor de *SP suma* aunque, en un principio, se cumplió con los criterios de aceptación correspondientes dentro del plazo estipulado.

ANEXO E - Documentación generada

E.1 Introducción

Con la intención nuevamente de facilitar la lectura de este documento, principalmente para no sobrecargar las secciones anteriores, se tomó la decisión de presentar la documentación generada a lo largo del desarrollo de cada una de las Historias de Usuario en forma de anexo. En este caso se optó por exponer dichas historias en el orden en que fueron realizadas, con la idea de mostrar la evolución del sistema a lo largo del tiempo, tanto en la lógica, exhibiendo como se fueron agregando las clases al diagrama de clases y las reglas del negocio dentro del back-end, como en lo visual.

En términos generales las historias pueden contener todos o algunos de los siguientes elementos, dependiendo esto principalmente del valor que puedan agregar al presente documento:

- **Mockups**: bosquejo realizado mediante el uso de la herramienta web Figma, donde se confeccionó una primera idea de la/s pantalla/s con las que debía contar la Historia de Usuario en cuestión. Las mismas fueron elaboradas previamente al inicio del primer Sprint, solo a modo de validación con los usuarios, por lo que cuentan con un nivel de detalle suficiente para utilizar de guía de desarrollo, pero no el suficiente (en la mayoría de los casos) como para reflejar fielmente los resultados finales (los cuales pueden observarse en la sección "Demostración de la ejecución de un remate en el sistema desarrollado").
- **Evolución del diagrama de clases**: como su nombre sugiere, se irá mostrando la evolución del diagrama de clases a medida que se iban implementando las mismas. Cabe aclarar que al igual que los mockups, este diagrama fue construido previo al inicio del trabajo de desarrollo, aunque posteriormente fue sufriendo diversos cambios a medida que iba creciendo el sistema, por detectarse inconsistencias o mejoras durante el progreso de alguna de las funcionalidades.
- **Casos de prueba**: como ya fue comentado en la sección de testing, se realizaron diversos tipos de pruebas a medida que se incorporaban funcionalidades, de las cuales fueron documentadas formalmente las prueba de unidad, realizadas con JUnit, y las pruebas de integración, realizadas mediante el software Insomnia.
- **Criterio de aceptación**: adoptamos el criterio de aceptación, el cual fue planteado desde un principio como parte de las Historias de Usuario, como "prueba de fuego" para considerar a la misma como finalizada, por lo que antes de continuar se debía dejar documentado su resultado. Es digno de mención, que especialmente en las historias pertenecientes a los primeros Sprints, las imágenes utilizadas no reflejan el

resultado final, al haber sido tomadas en el momento y luego el sistema haber sufrido cambios estéticos.

- **Código**: fragmentos de código que creemos podrían enriquecer la lectura y la comprensión de quien lee. En este apartado intentamos ser más bien acotados y limitar el código mostrado con la idea de no agobiar al lector.

E.2 Historias de Usuario

E.2.1 Historia de Usuario 1 - Log in

E.2.1.1 Mockup de la pantalla de Log in

La primera pieza de documentación generada es un mockup de la pantalla en cuestión, donde en este caso se trata simplemente de 2 campos de texto, uno para el nombre de usuario (o username) y otro para la contraseña, y un botón para ingresar a la pantalla Home.

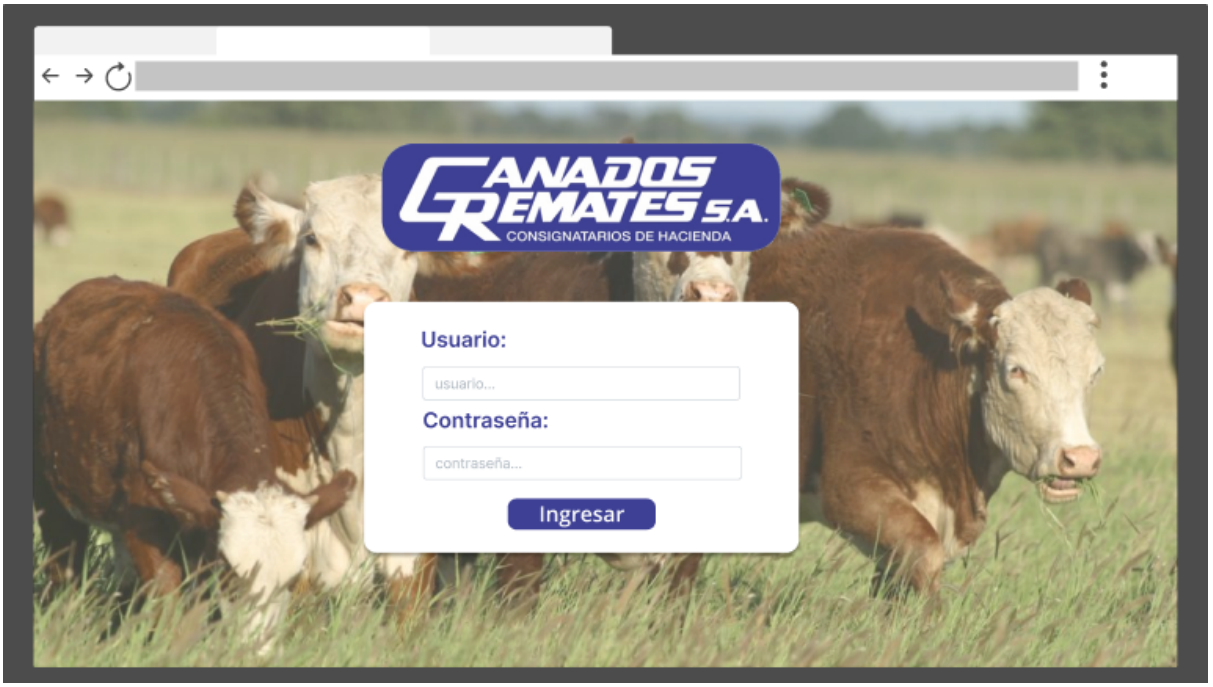
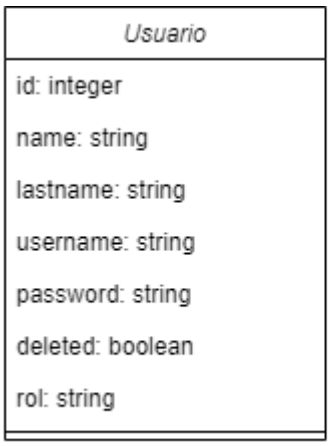


Figura 64: Mockup pantalla de Log in

E.2.1.2 Evolución del diagrama de clases

Esta historia solamente incorpora a los propios usuarios del sistema como nueva clase, por lo que el diagrama es todavía extremadamente sencillo:



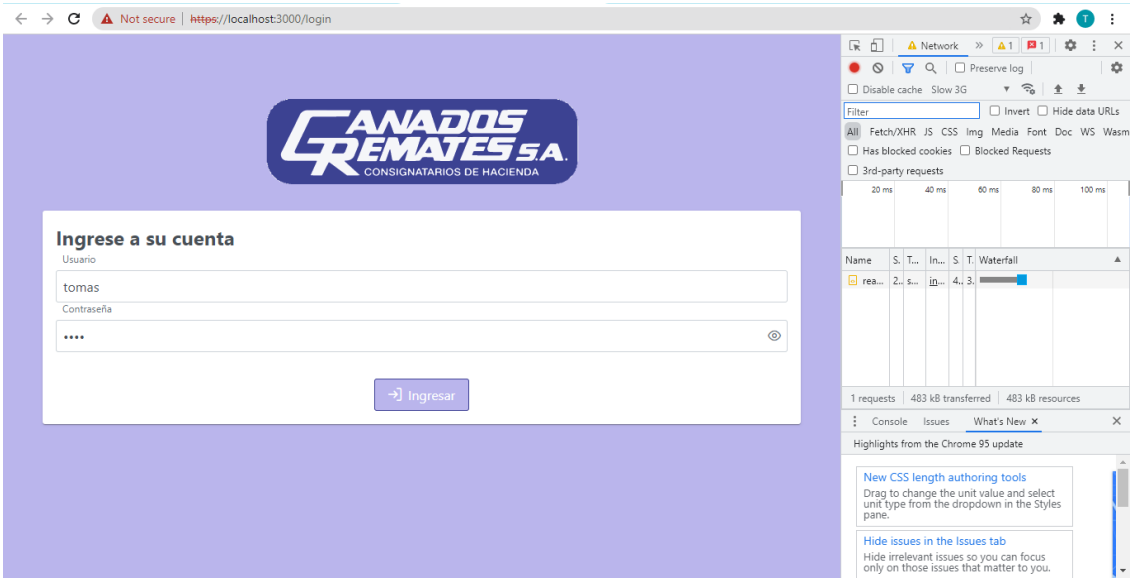
Roles: "Administrador", "Consignatario", "Asistente"

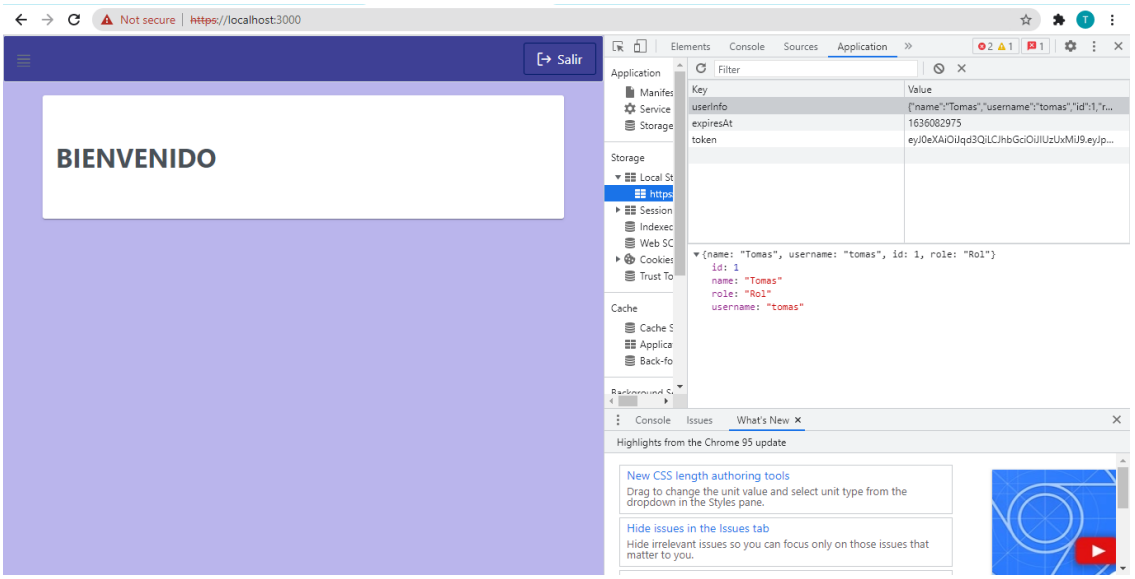
Figura 65: Diagrama de clases historia 1

E.2.1.3 Casos de prueba

CASO	RESULTADO ESPERADO
Login desde el front end con usuario existente y contraseña correcta.	HTTP 200. Ingreso al Home y jwt en localstorage.

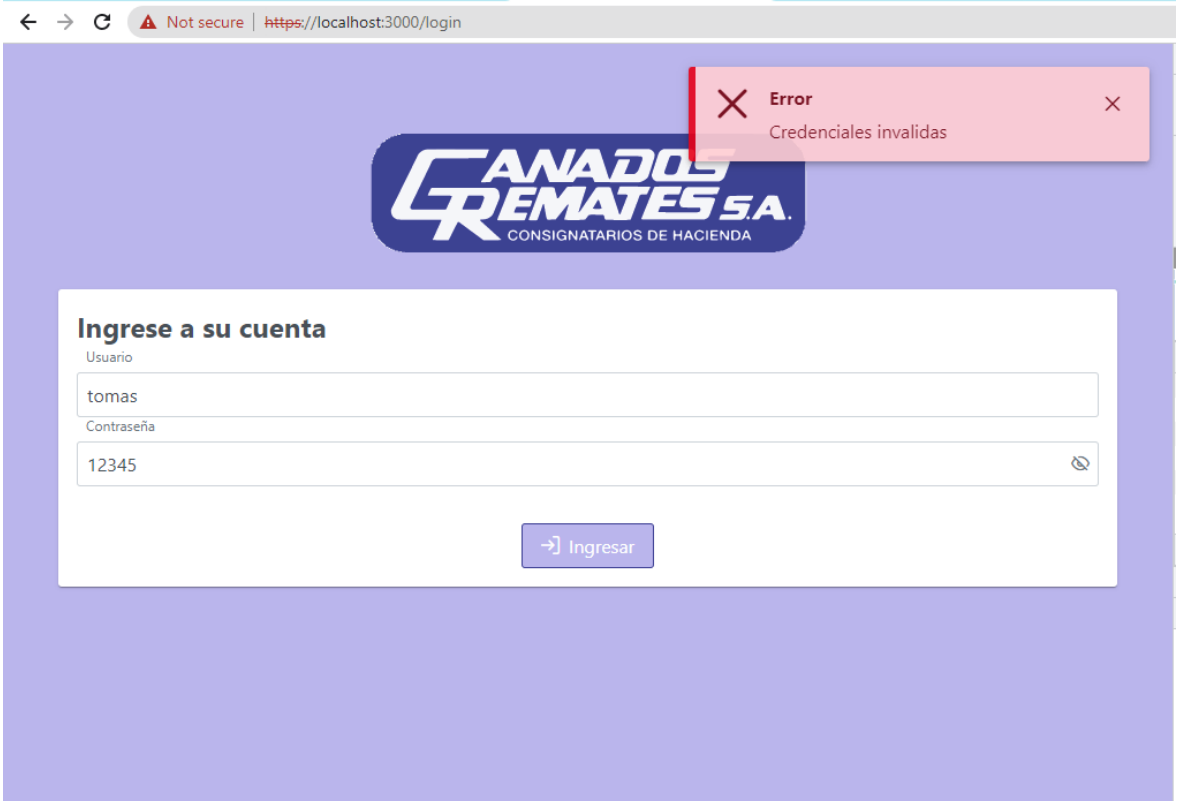
Resultado obtenido:





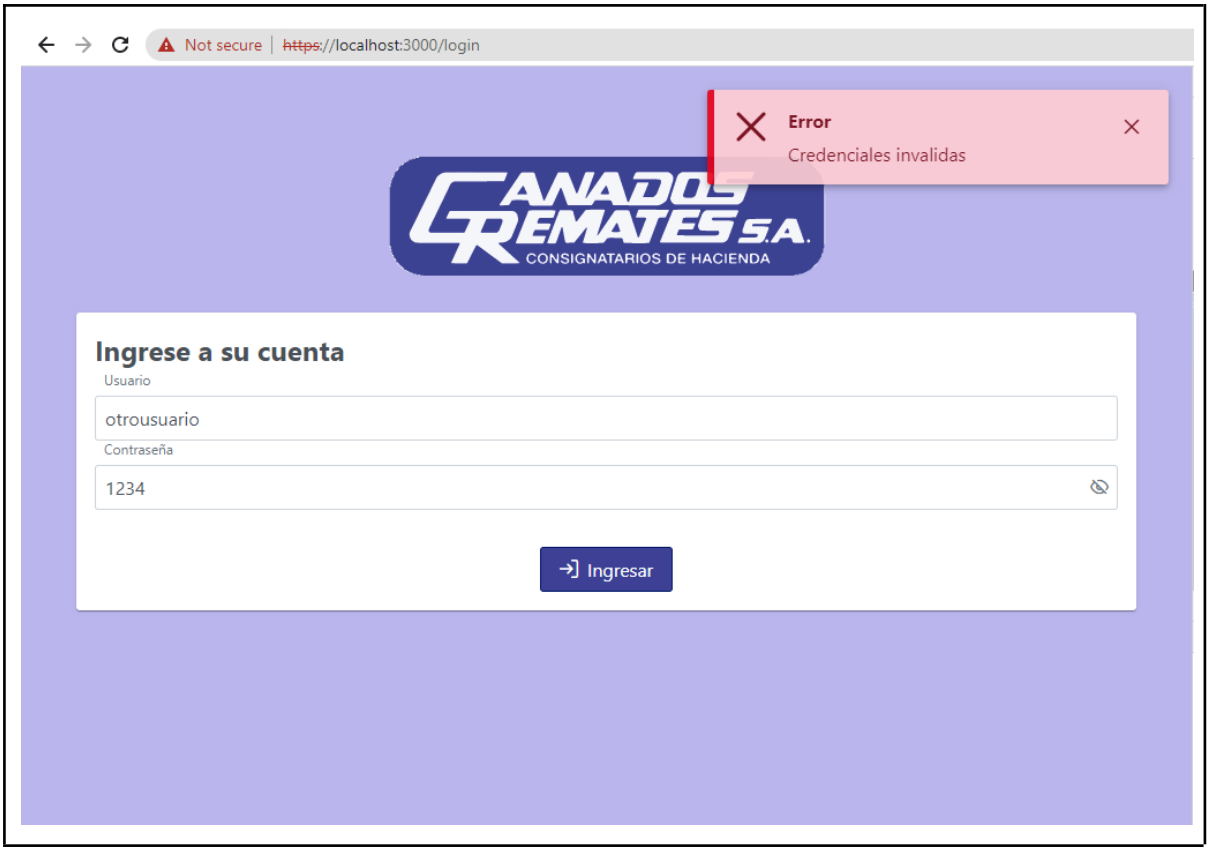
Login desde el front end con usuario existente y contraseña incorrecta.	HTTP 400. Mensaje: credenciales inválidas.

Resultado obtenido:



Login desde el frontend con usuario inexistente.	HTTP 400. Mensaje: credenciales inválidas.

Resultado obtenido:



E.2.1.4 Criterio de aceptación

En este caso el criterio de aceptación enuncia que “Se obtiene el token de acceso del usuario y éste logra acceder a la pantalla principal del sistema.”, lo cual hemos comprobado en el primer caso de prueba mostrado anteriormente.

E.2.1.5 Código

```
public class MyAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
    private static final Logger logger =
LoggerFactory.getLogger(MyAuthenticationFilter.class);
    private final AuthenticationManager authenticationManager;
    private String error = "";
    public MyAuthenticationFilter(AuthenticationManager authMgr ){
       ...
    }

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
HttpServletResponse response) {
        LinkedHashMap<String, String> map = new LinkedHashMap<>();
        response.setContentType("application/json");
        response.setCharacterEncoding("UTF-8");
        String username = null, password = null;
        //Si la petición viene en el body
        ObjectMapper mapper = new ObjectMapper();
```

```
        try {
            User user = mapper.readValue(request.getInputStream(),User.class);
            try{
                username = user.getUsername();
                password = user.getPassword();
                logger.debug(user.toString());
                if(username == null || password == null){
                    throw  new NullPointerException();
                }
            }catch (NullPointerException e){
                try {
                    response.setStatus(HttpStatus.BAD_REQUEST.value());
                    error = "Existen parámetros nulos.";
                    map.put("error", error);
                    response.getWriter().println(mapper.writeValueAsString(map));
                    response.getWriter().flush();
                } catch (IOException ex) {
                    response.setStatus(HttpStatus.INTERNAL_SERVER_ERROR.value());
                    ex.printStackTrace();
                }
            }
        } catch (IOException e) {
            response.setStatus(HttpStatus.INTERNAL_SERVER_ERROR.value());
            try {
                error = "Error al obtener parámetros desde la solicitud HTTP.";
                map.put("error", error);
                response.getWriter().println(mapper.writeValueAsString(map));
                response.getWriter().flush();
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
        UsernamePasswordAuthenticationToken authenticationToken = new
UsernamePasswordAuthenticationToken(username, password);

        try {
            //Realizamos la autenticación con MyAuthenticationProvider y capturamos
posibles excepciones.
            return authenticationManager.authenticate(authenticationToken);
        } catch (InvalidCredentialsException exception) {
            response.setStatus(HttpStatus.BAD_REQUEST.value());
            try {
                error = exception.getMessage();
                map.put("error", error);
                response.getWriter().println(mapper.writeValueAsString(map));
                response.getWriter().flush();
            } catch (IOException e) {
                response.setStatus(HttpStatus.INTERNAL_SERVER_ERROR.value());
            }

        } catch (Exception exception) {
            exception.printStackTrace();
            response.setStatus(HttpStatus.INTERNAL_SERVER_ERROR.value());
        }
        return null;
    }
```

```
    @Override
    public void successfulAuthentication(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain, Authentication authentication) throws
ServletException, IOException {

       //Si la autenticación fue exitosa, armamos el JWT que revolvemos a la app
cliente. Ya se mostró anteriormente
      ...
    }
}
```

```
public class MyAuthenticationProvider implements AuthenticationProvider {
   private static final Logger logger =
LoggerFactory.getLogger(MyAuthenticationProvider.class);

   @Autowired
   UserService userService;

   @Autowired
   private PasswordEncoder passwordEncoder;

   @Override
   public Authentication authenticate(Authentication authentication) throws
AuthenticationException {
       //Este metodo se encarga de verificar que el usario existe en la base de datos y
compara sus contraseñas.
       String username = authentication.getName();
       String password = authentication.getCredentials().toString();

       Optional<User> userOpt = userService.findByUsername(username);
       if(userOpt.isEmpty()){
           throw new InvalidCredentialsException("Las credenciales son inválidas");
       }
       User user = userOpt.get();
       if(user.isDeleted()!=null && user.isDeleted()){
           logger.error("Este usuario fue eliminado");
           throw new InvalidCredentialsException("Las credenciales son inválidas");
       }
       logger.debug(passwordEncoder.encode(password));
       if(!passwordEncoder.matches(password, user.getPassword())){
           throw new InvalidCredentialsException("Las credenciales son inválidas");
       }

       List<GrantedAuthority> rol = new ArrayList<>();
       rol.add(new SimpleGrantedAuthority(user.getRol()));

       //Este token es usado internamente por spring.
       return new UsernamePasswordAuthenticationToken(new Principal(user), null, rol);
   }
}
```

E.2.2 Historia de Usuario 2 - Información del perfil

E.2.2.1 Mockups

Aquí contamos con 2 pantallas:

- La primera se trata de la pantalla de información de perfil propiamente, la cual contiene un campo de nombre (que al implementarlo se cambió por 2 campos, uno de nombre y otro de apellido), uno para el nombre de usuario y un dropdown para el rol, además de 3 botones al final, los clásicos botones de aceptar y cancelar, y un botón para pasar a la pantalla de cambiar contraseña. Al momento de la implementación, se decidió agregar un botón para habilitar/deshabilitar la edición de los datos (salvo el rol que no es modificable).

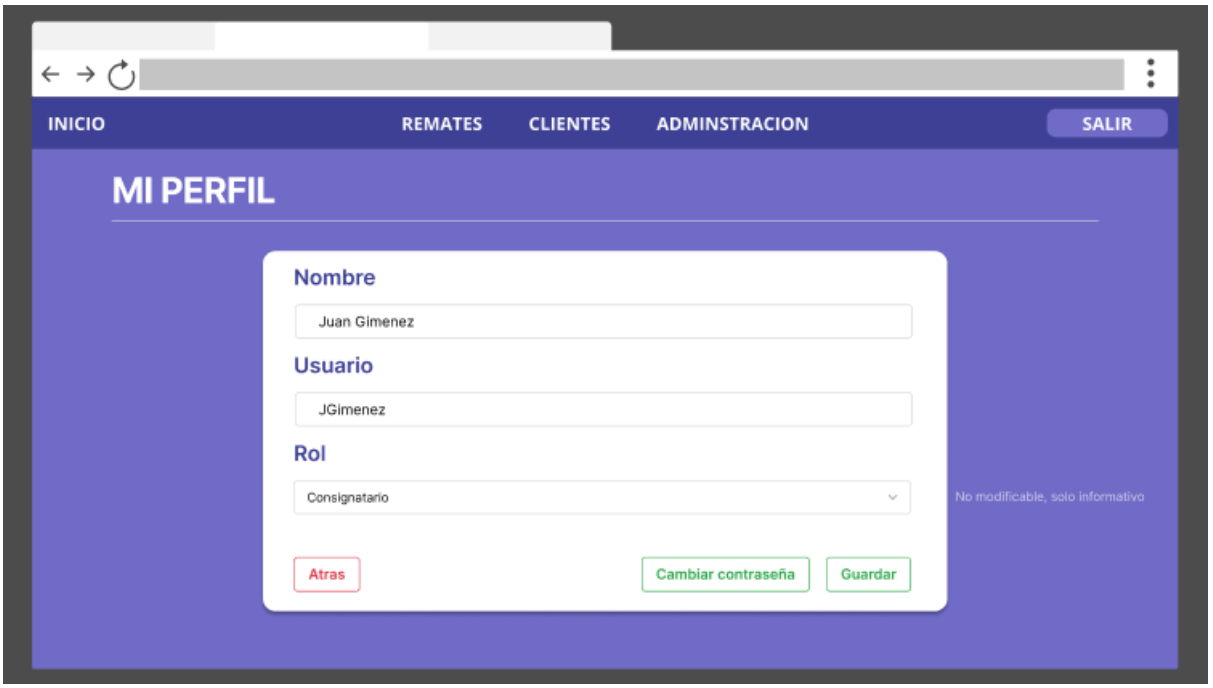


Figura 66: Mockup pantalla de Información del Perfil

- La segunda se trata justamente de la pantalla para cambiar contraseña, la cual posee, además de los botones de aceptar y cancelar, 3 campos de texto: el primero para ingresar la contraseña actual del usuario y los 2 últimos para ingresar la nueva contraseña.

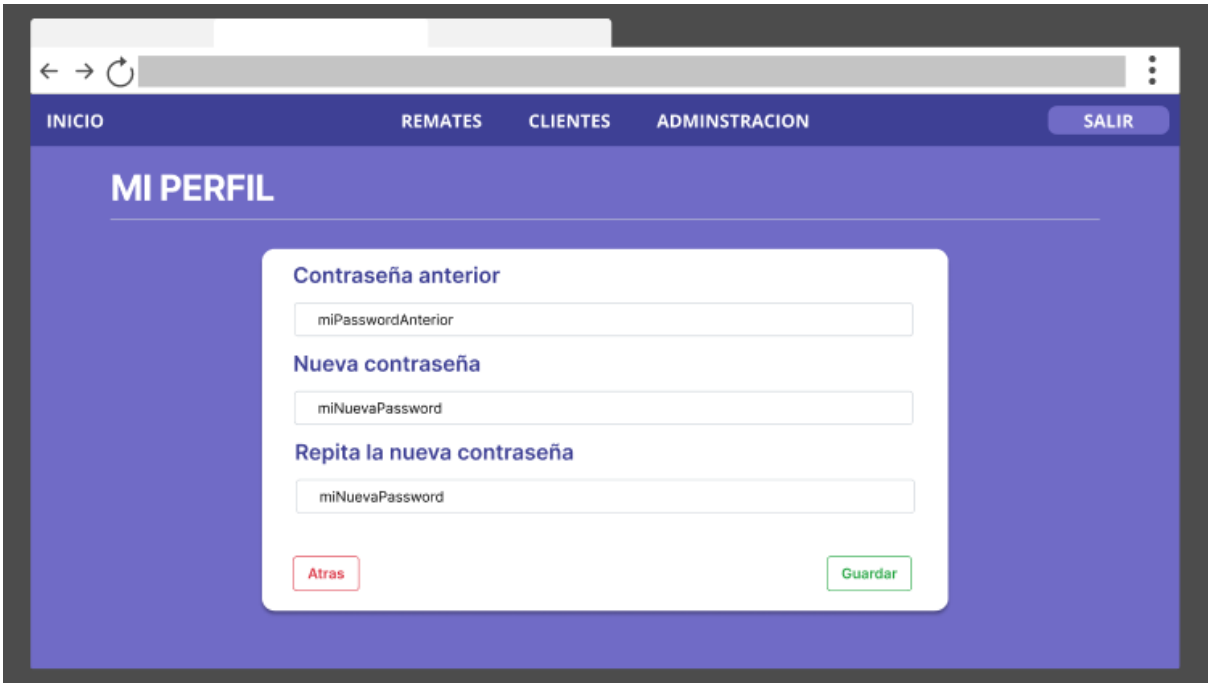


Figura 67: Mockup pantalla de Cambiar contraseña

E.2.2.2 Casos de prueba

Casos de prueba en la solicitud de datos:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Solicitar datos con un id de usuario existente.	HTTP 200. Retornar la información, en formato JSON, del usuario y mostrarla correctamente en pantalla.	Éxito.
Solicitar datos con un id de usuario inexistente.	HTTP 404	Éxito.

Modificar datos de usuario:

Tests unitarios:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Modificar los datos de un usuario, cuyo {id} existe.	Cambios reflejados en la base de datos.	Éxito.
Modificar los datos de un usuario, cuyo {id} no existe.	Exception: UserNotFoundException	Éxito.
Modificar los datos de un usuario, cuyo {id} existe, pero que los datos no coincidan con los atributos de un “user”	No hay cambios reflejados en la base de datos ni se lanza excepción.	Éxito.

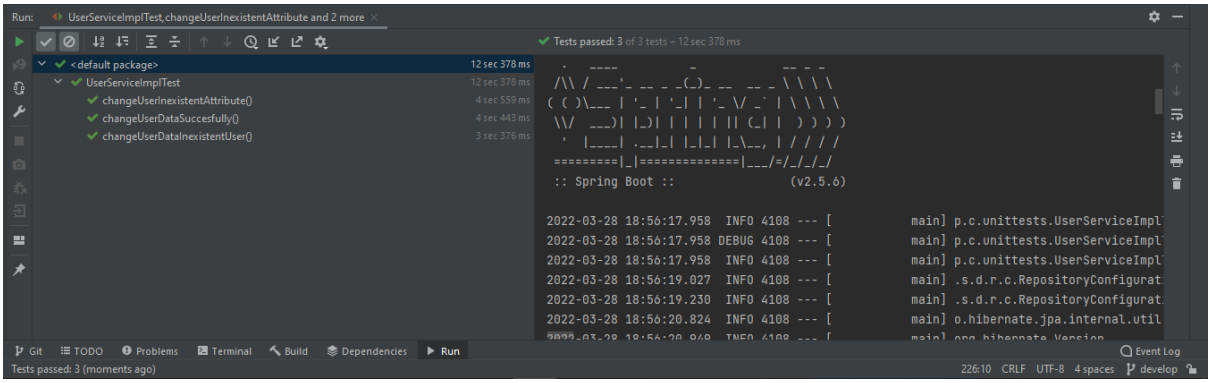


Figura 68: resultado de los tests

Tests de integración (Rest, Service, DAO):

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Modificar los datos de un usuario, cuyo {id} existe.	HTTP 200. JWT con los datos nuevos.	Éxito.
Modificar los datos de un usuario, cuyo {id} no existe.	HTTP 400 (mediante este endpoint, un usuario no puede modificar información de otro, exista o no).	Éxito.
Modificar los datos de “rol” o “id” de un usuario, cuyo {id} existe.	HTTP 400.	Éxito.
Modificar los datos de un usuario, cuyo {id} existe, pero que los datos no coincidan con los atributos de un “user”.	HTTP 200. Sin cambios en la BD.	Éxito.

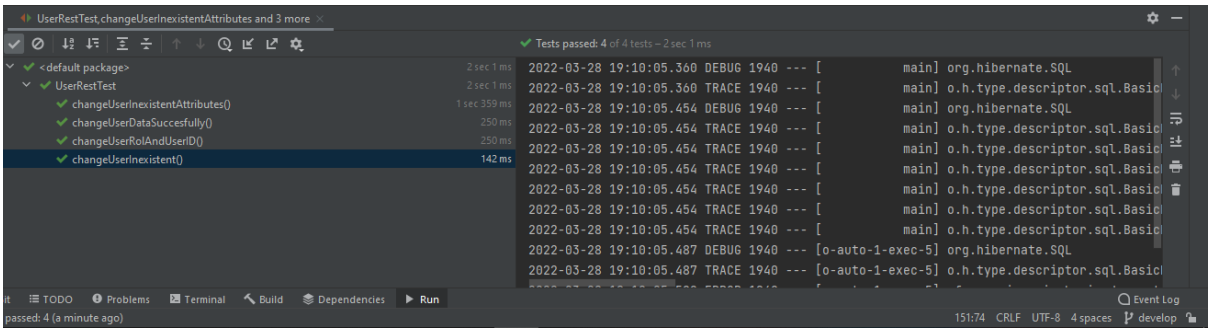


Figura 69: resultado de los tests

Cambio de contraseña:

Tests unitarios:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
oldPassword igual a la que está almacenada en la BD y newPassword diferente a la primera, no nula ni solo espacios en blanco.	Nueva contraseña del usuario, modificada con newPasword.	Éxito.
“newPassword” igual “oldPassword”	Excepción: InvalidCredentialsException	Éxito
“oldPassword” diferente a la password almacenada en la BD	Excepción: InvalidCredentialsException	Éxito
{id} de un usuario inexistente	Excepcion: UserNotFoundException	Éxito

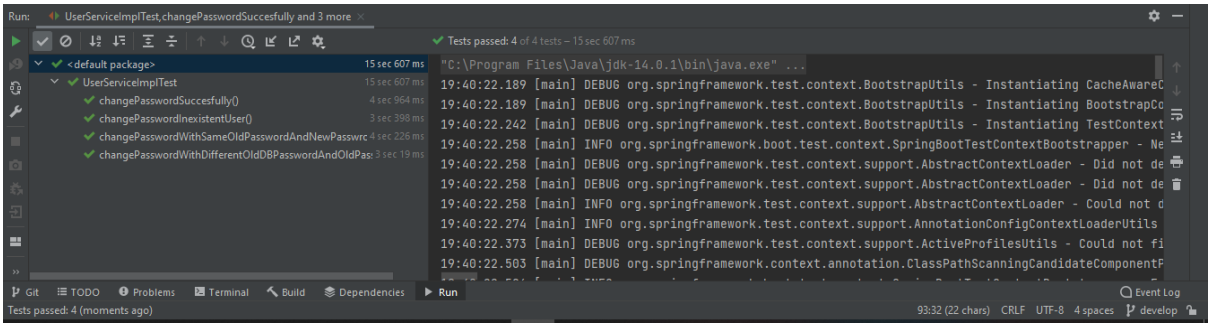


Figura 70: resultado de los tests

Casos de prueba de integración (Rest + Service + DAO):

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
oldPassword igual a la que está almacenada en la BD y newPassword diferente a la primera, no nula ni solo espacios en blanco.	HTTP 200	Éxito
oldPassword igual a la que está almacenada en la BD y newPassword igual a oldPassword.	HTTP 400	Éxito
oldPassword o newPassword nulos o solo con espacios en blanco.	HTTP 400	Éxito
{id} de un usuario inexistente	HTTP 404	Éxito

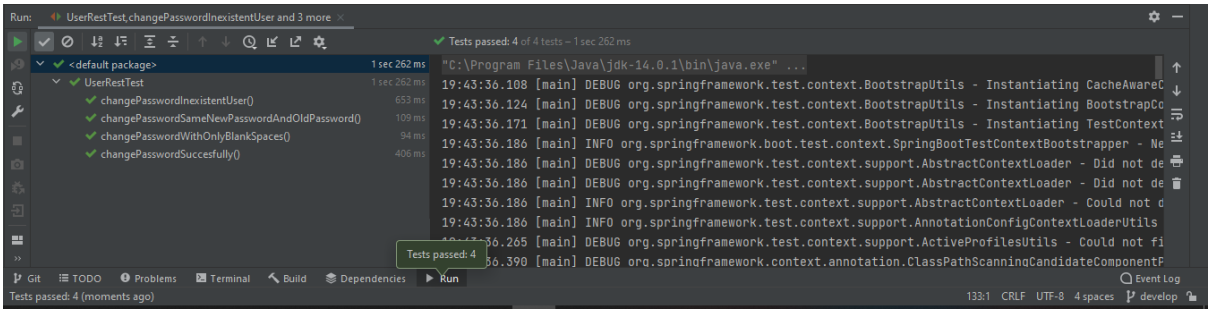


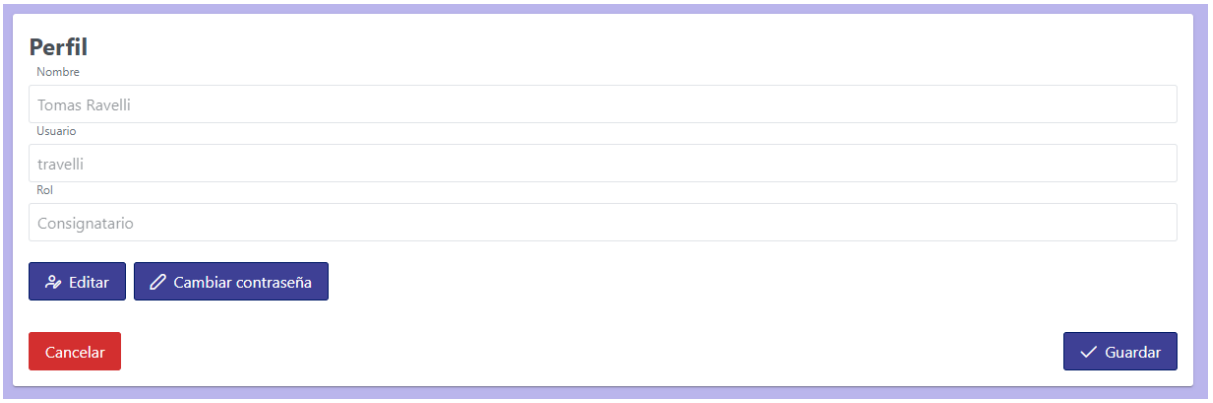
Figura 71: resultado de los tests

E.2.2.3 Criterios de aceptación

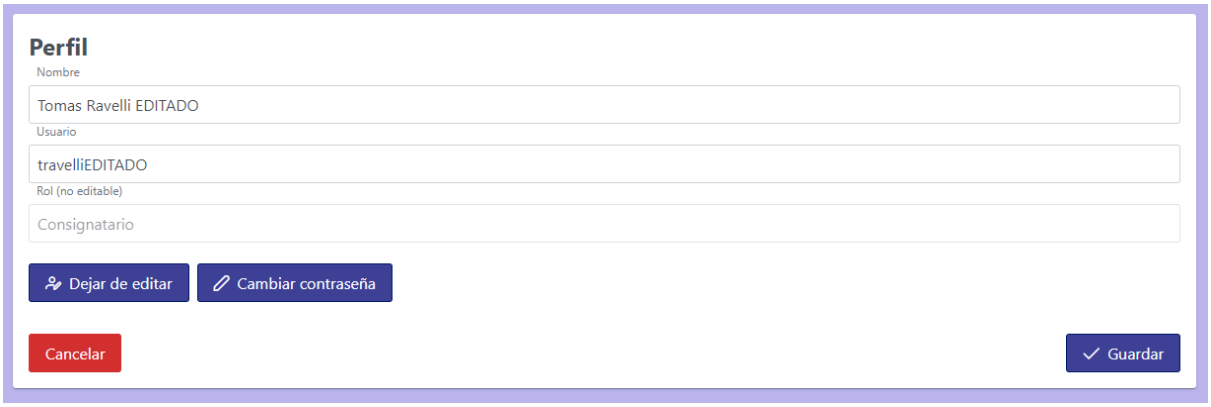
- En la pantalla se muestra la información correcta respectiva al usuario.
- Los datos del usuario (username y name) son modificados y guardados con éxito en la base de datos.
- La contraseña es modificada y guardada con éxito en la base de datos.

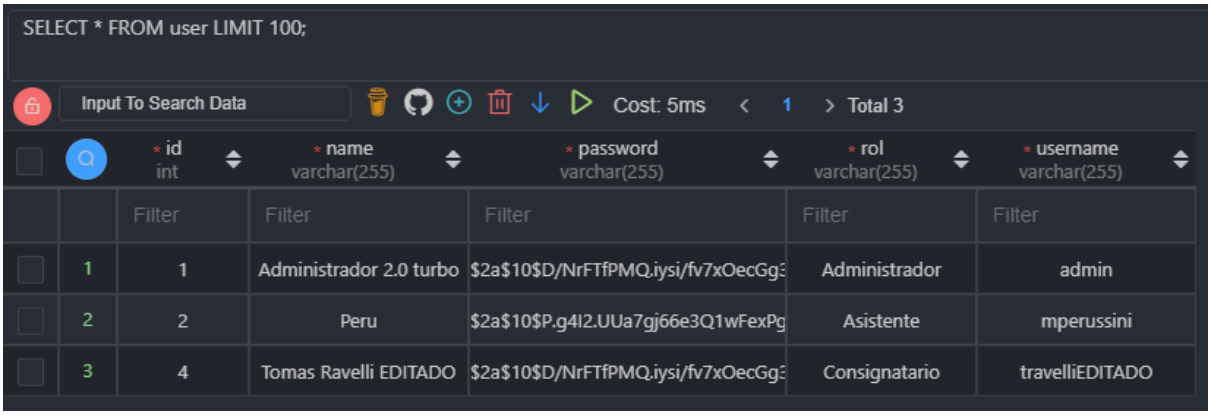
Resultados:

En la pantalla se muestra la información correcta respectiva al usuario.

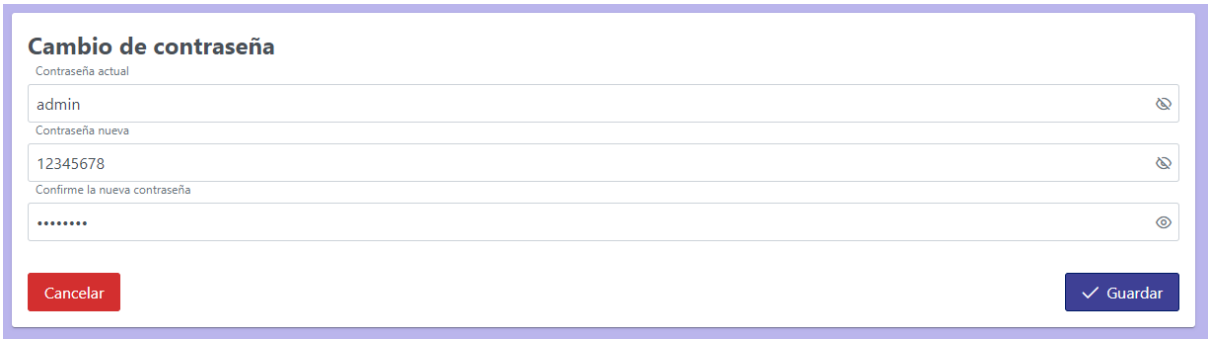


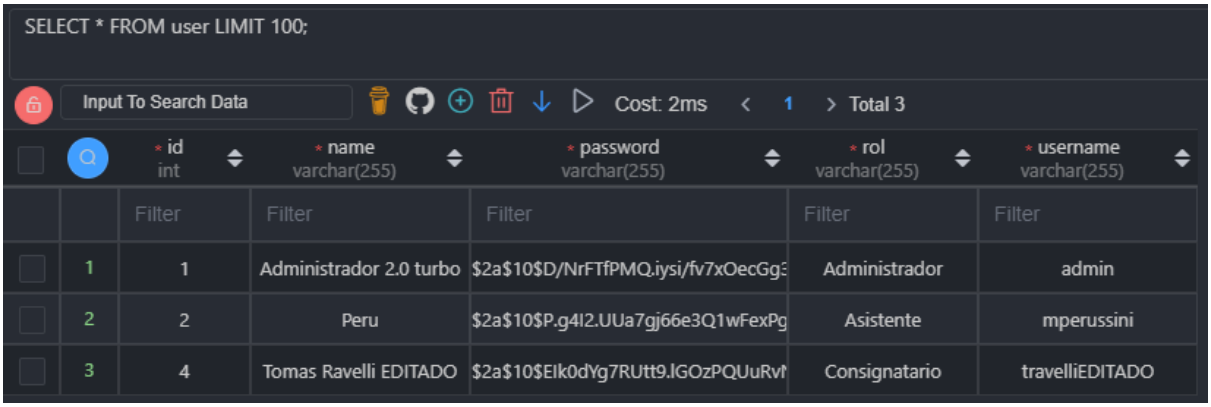
Los datos del usuario (username y name) son modificados y guardados con éxito en la base de datos.





La contraseña es modificada y guardada con éxito en la base de datos.





(Observar cómo se modificó el campo contraseña respecto a las capturas anteriores)

E.2.2.4 Código

```
@Service
public class UserServiceImpl implements UserService {
  ...
@Override
public JwtToken updateUserProfileById(Integer id, Map<Object, Object> fields) throws
DuplicateUsernameException, BadHttpRequest {
   if(!getCurrentUser().getId().equals(id)){
       throw new InvalidCredentialsException("No se puede modificar el perfil de otro
Usuario.");
   }
   Optional<User> userOpt = userDAO.findById(id);
   if(fields.containsKey("id")){
       if(!fields.get("id").equals(id)){
           throw new InvalidCredentialsException("No se puede modificar el id del
Usuario");
       }
   }
   if(userOpt.isPresent()){
       User user = userOpt.get();
       User finalUser = user;
       fields.forEach((key, value) -> {
           if(value != null) {
               Field field = ReflectionUtils.findField(User.class, (String) key);
               if(field != null) {
                   field.setAccessible(true);
                   ReflectionUtils.setField(field, finalUser, value);
               }
           }
       });
       user = saveUser(finalUser);
       String token = Jwts.builder()
               .signWith(SecurityConstants.JWT_SECRET, SignatureAlgorithm.HS512)
               .setHeaderParam("typ", SecurityConstants.TOKEN_TYPE)
               .setIssuer(SecurityConstants.TOKEN_ISSUER)
               .setAudience(SecurityConstants.TOKEN_AUDIENCE)
               .setSubject(user.getName() + ' ' + user.getLastname())
               .claim("name", user.getName())
               .claim("lastname", user.getLastname())
               .claim("uid", user.getId())
               .claim("username", user.getUsername())
               .setExpiration(new Date(System.currentTimeMillis() + 86400000)) //un dia
               .claim("rol",
SecurityContextHolder.getContext().getAuthentication()==null?
List.of("Rol"):SecurityContextHolder.getContext().getAuthentication().getAuthorities())
               .compact();

       LinkedHashMap<String,String> map = new LinkedHashMap<>();
       map.put("access_token", token);
       return new JwtToken(map.get("access_token"));
   }
   throw new UserNotFoundException("El Usuario con id: "+ id +" no existe");
}
...
}
```

```
@Service
public class UserServiceImpl implements UserService {
  ...
@Override
public void changePasswordById(Integer id, ChangePassword changePassword) throws
DuplicateUsernameException {
   if(!getCurrentUser().getId().equals(id)){
       throw new InvalidCredentialsException("No se puede modificar el perfil de otro
Usuario.");
   }
   Optional<User> userOpt = userDAO.findById(id);
   if(userOpt.isPresent()){
       User user = userOpt.get();
       String oldDBpassword = user.getPassword();
       String newPasswordEncoded =
passwordEncoder.encode(changePassword.getNewPassword());
       if(!passwordEncoder.matches(changePassword.getOldPassword(), oldDBpassword)){
           throw new InvalidCredentialsException("La contraseña antigua ingresada es
diferente a su contraseña actual.");
       }
       if(passwordEncoder.matches(changePassword.getNewPassword(), oldDBpassword)){
           throw new InvalidCredentialsException("Las contraseñas deben ser distintas");
       }
       user.setPassword(newPasswordEncoded);
       saveUser(user);
   }else{
       throw new UserNotFoundException("Usuario con id: " + id + " no encontrado.");
   }
}
...
}
```

E.2.3 Historia de Usuario 5 - CRUD localidades

E.2.3.1 Mockups

Para esta historia tenemos una pantalla, la cual lista las localidades, mostrando el nombre y ofreciendo un botón para editarla y otro para eliminarla en cada una, además de una barra de búsqueda para filtrar los resultados y un botón para crear nuevas localidades. Por otro lado, tenemos un diálogo que se abre al presionar el botón de crear localidad, con un campo de texto para introducir el nombre y los botones de aceptar y cancelar. El mismo también se abre al presionar un botón de editar localidad, mostrando en este caso el nombre actual de la misma en lugar de estar vacío.

Figura 72: Mockup pantalla de listado de localidades

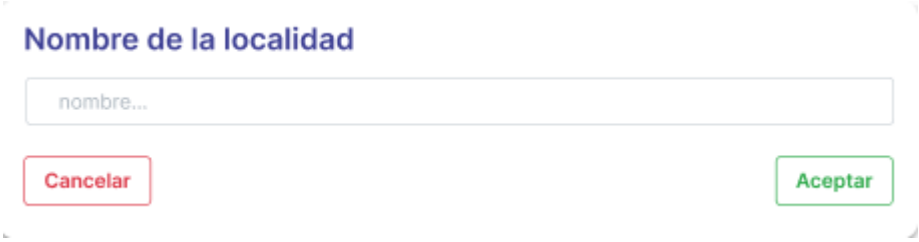


Figura 73: Mockup diálogo crear/editar localidad

E.2.3.2 Evolución del diagrama de clases

Aquí se agrega la entidad “Localidad” al sistema, aunque aún está inconexa respecto a la previamente existente:

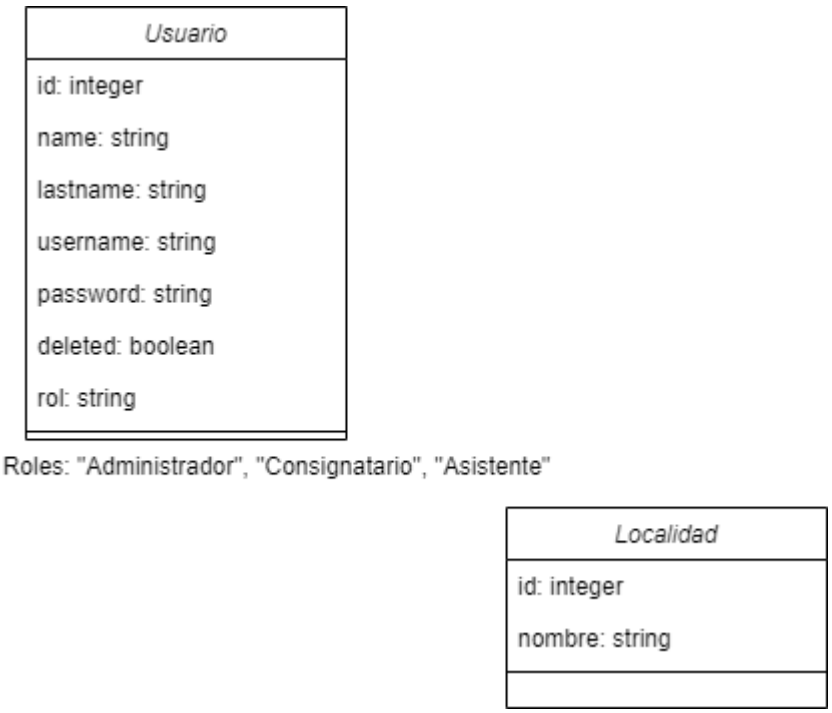


Figura 74: Diagrama de clases historia 5

E.2.3.3 Criterios de aceptación

Create/update: verificar que los cambios se registraron correctamente en la base de datos.

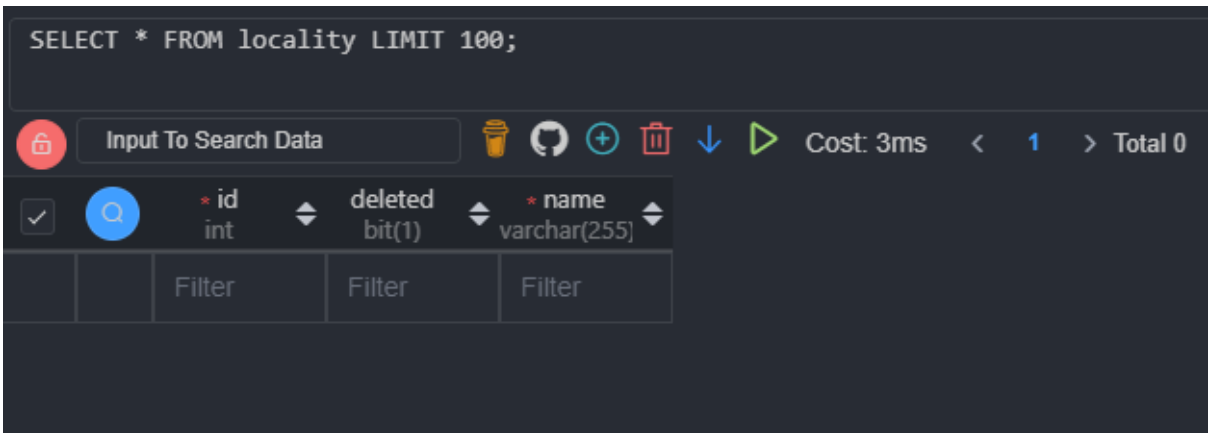
Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que la localidad esté marcada como "deshabilitado" en la base de datos ("soft delete", decisión de implementación a futuro).

Read: mostrar correctamente el listado de localidades.

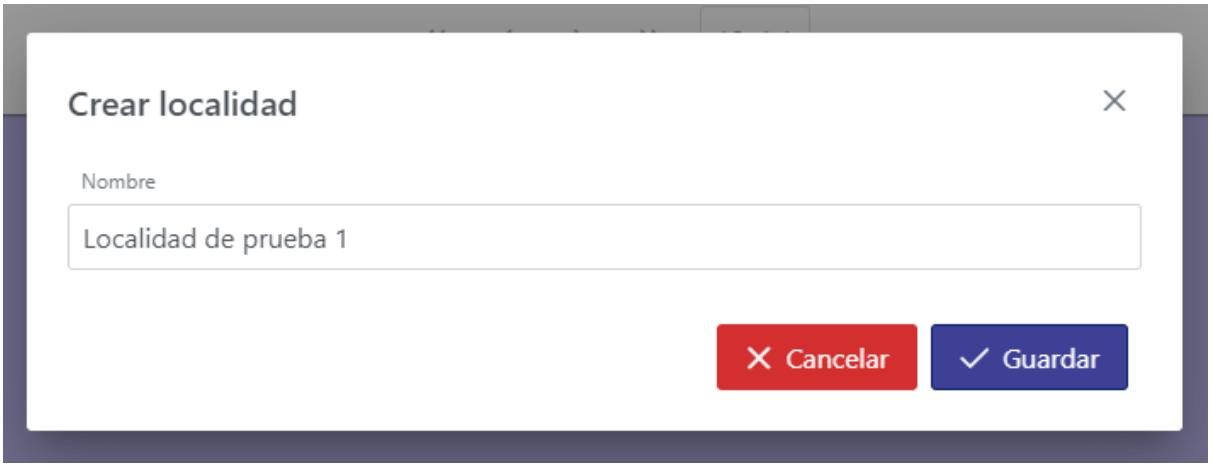
Resultados:

Create/update: verificar que los cambios se registraron correctamente en la base de datos.

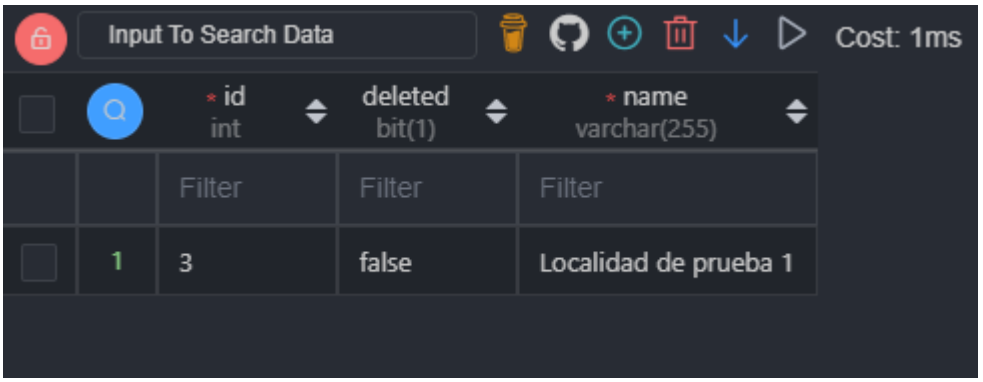
La DB arranca vacía:



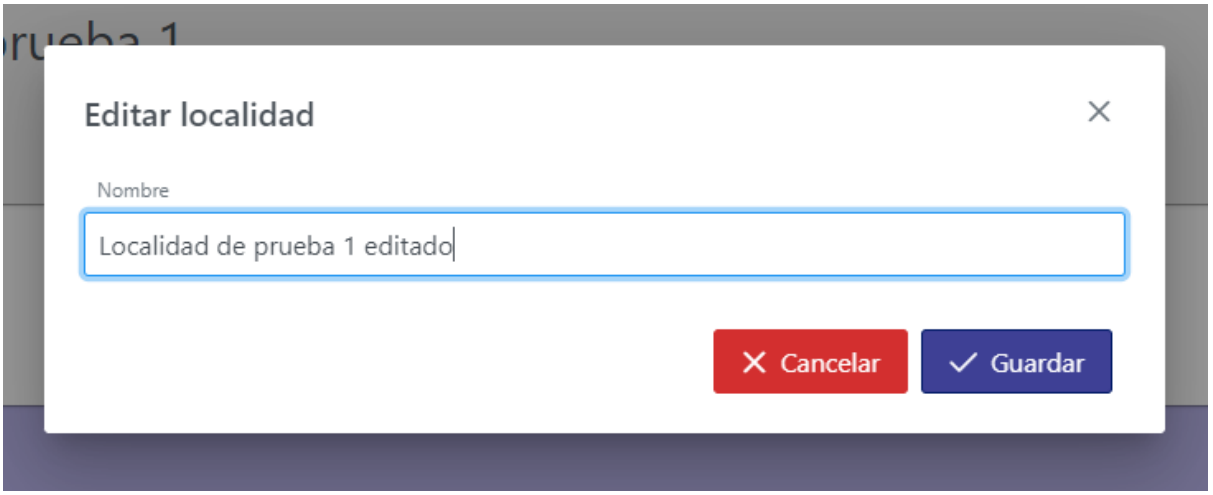
Se crea una nueva localidad:



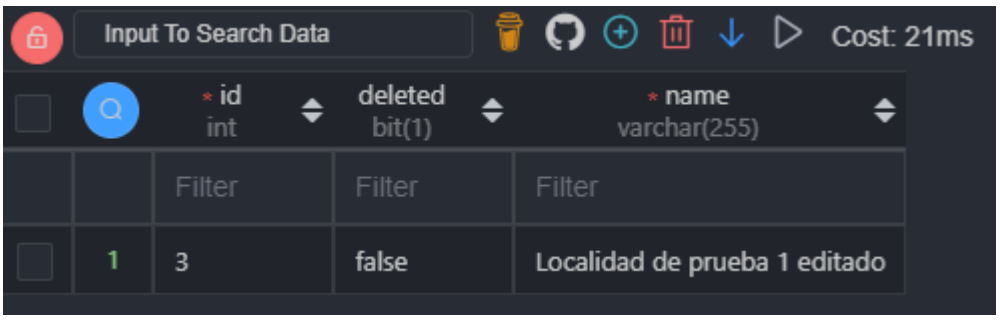
Y efectivamente ésta se guarda en la DB:



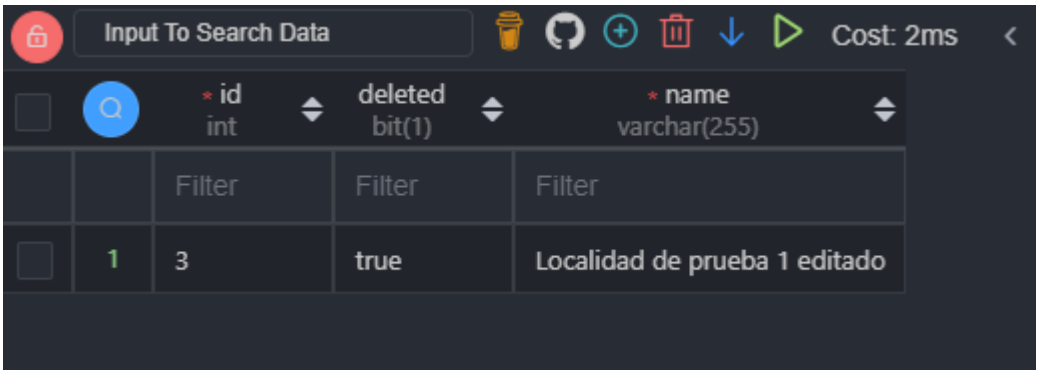
Ahora se la edita:



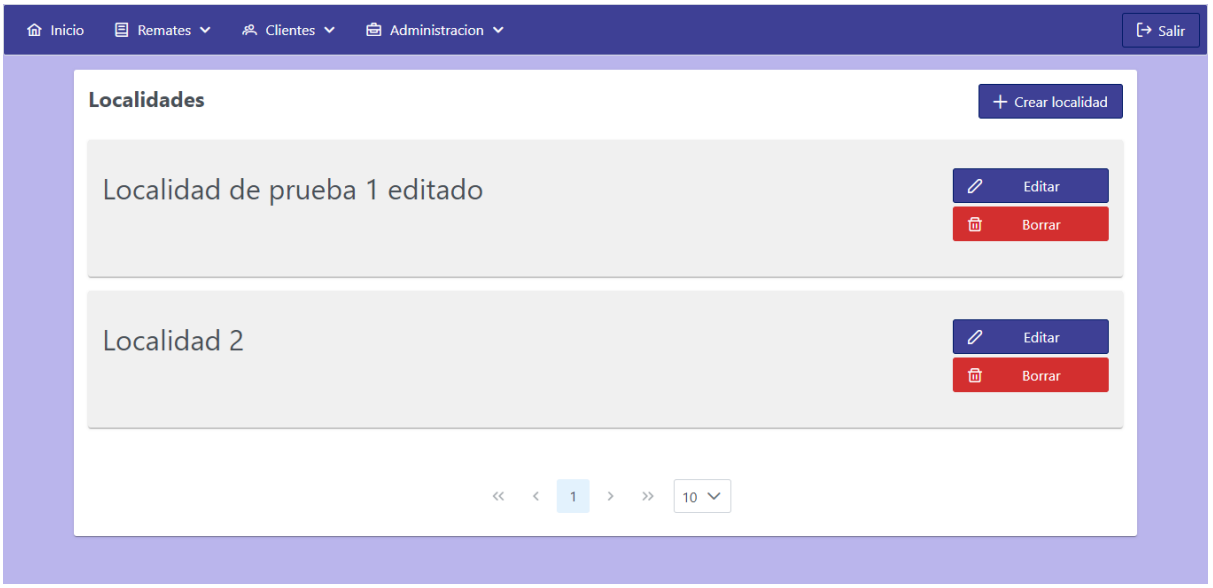
Y la misma se actualiza en la DB:



Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que la localidad esté marcada como “deshabilitado” en la base de datos (decisión de implementación a futuro).



Read: mostrar correctamente el listado de localidades.



E.2.4 Historia de Usuario 6 - CRUD categoría animales

E.2.4.1 Mockups

Los mockups de esta historia son prácticamente iguales a los de la historia anterior (más aún, a la hora de la implementación se usó el mismo componente para ambas pantallas), por lo que simplemente serán mostrados.

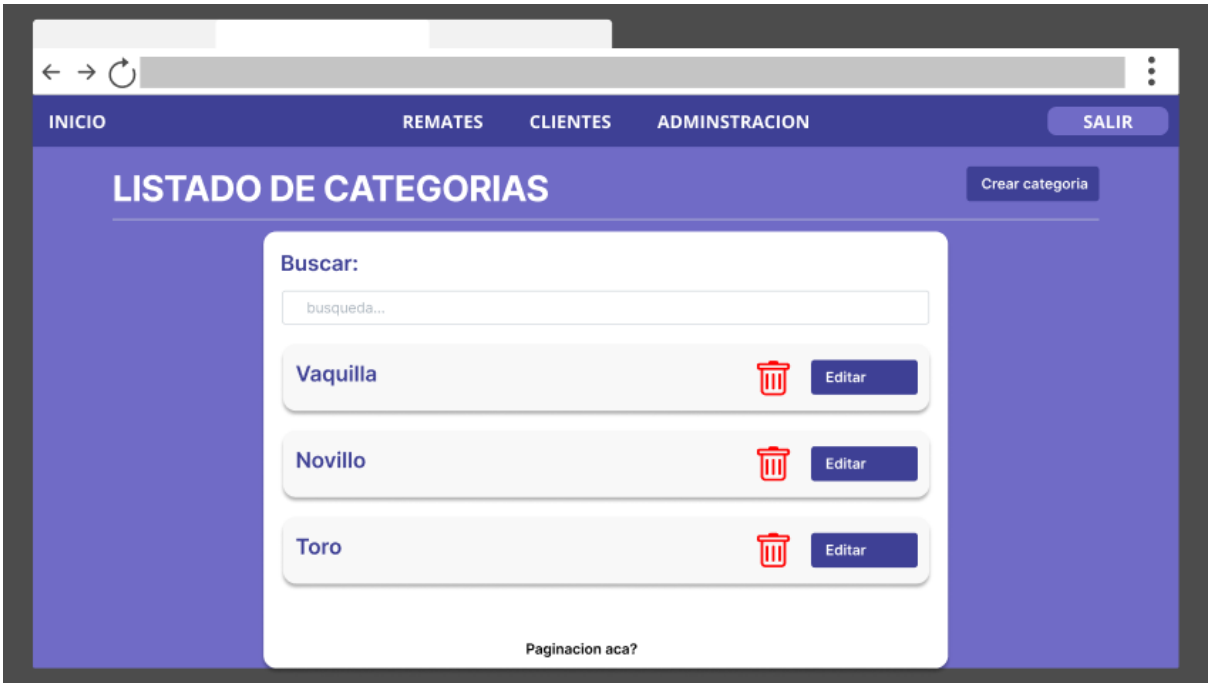


Figura 75: Mockup pantalla de listado de categorías

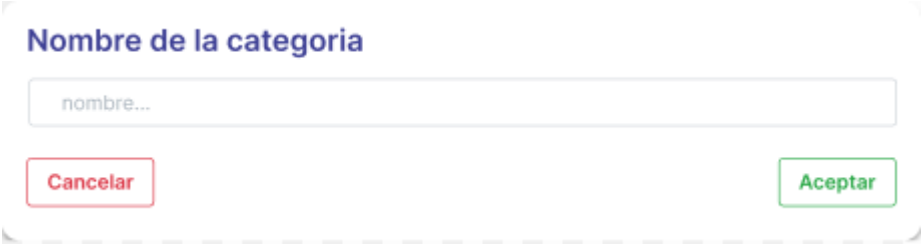


Figura 76: Mockup diálogo crear/editar categoría

E.2.4.2 Evolución del diagrama de clases

Al igual que en el caso anterior, en este también se agrega una nueva clase, pero que no se relaciona con las demás.

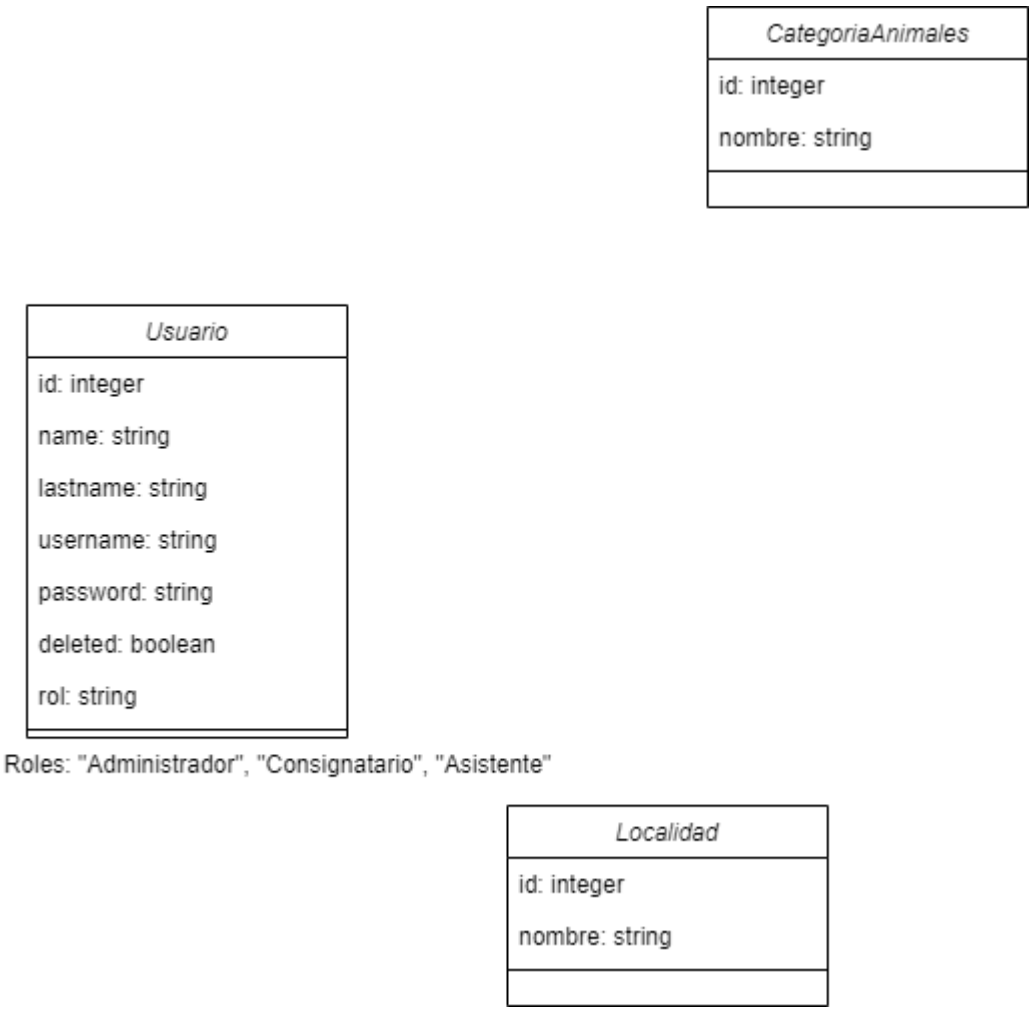


Figura 77: Diagrama de clases historia 6

E.2.4.3 Criterios de aceptación

Create/update: verificar que los cambios se registraron correctamente en la base de datos.

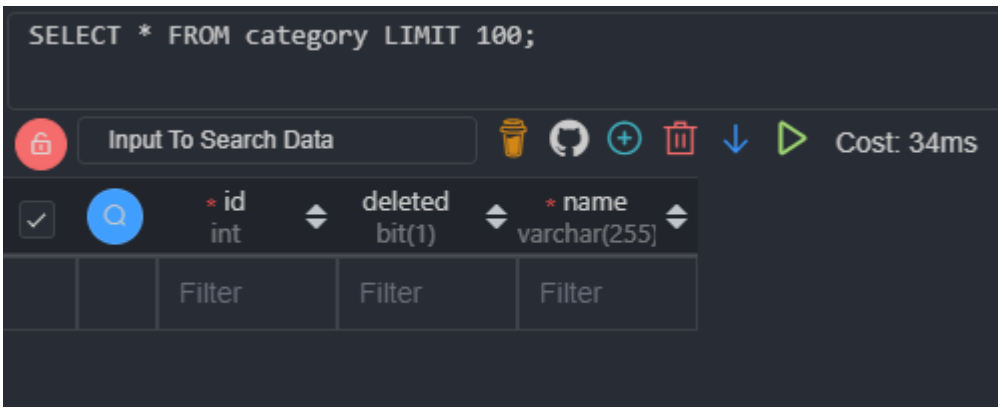
Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que la categoría esté marcada como “deshabilitado” en la base de datos (decisión de implementación a futuro).

Read: mostrar correctamente el listado de categoría de animales.

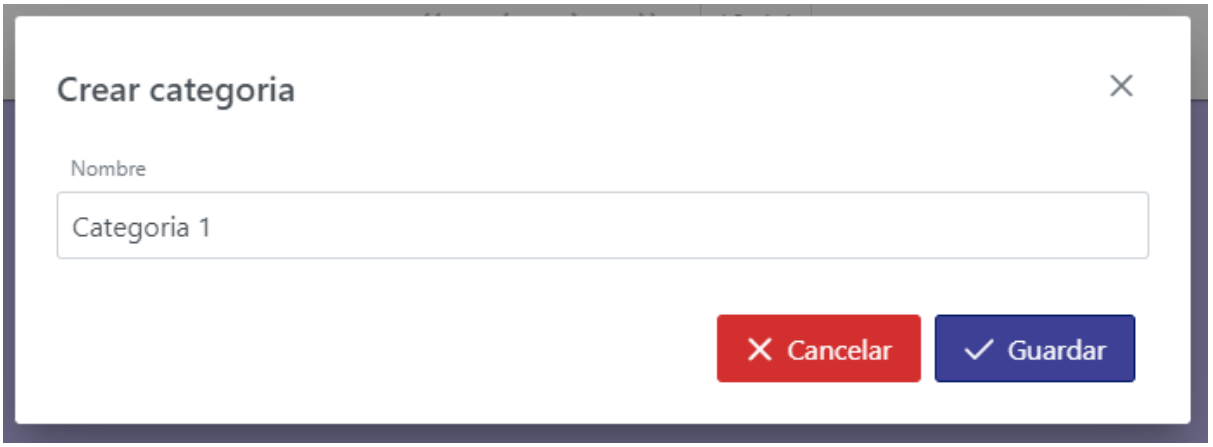
Resultados:

Create/update: verificar que los cambios se registraron correctamente en la base de datos.

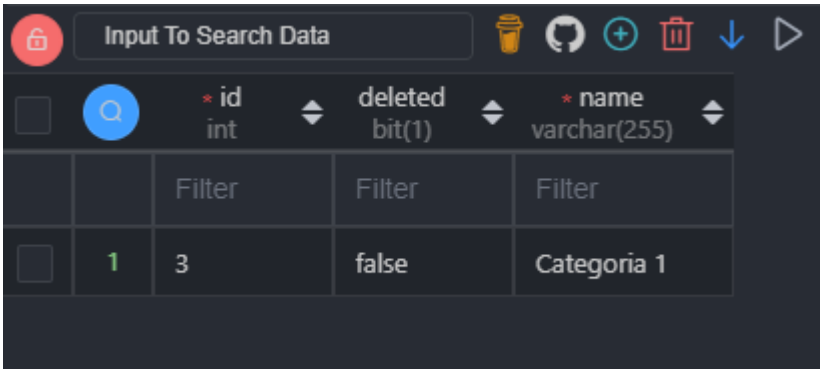
La DB arranca vacía:



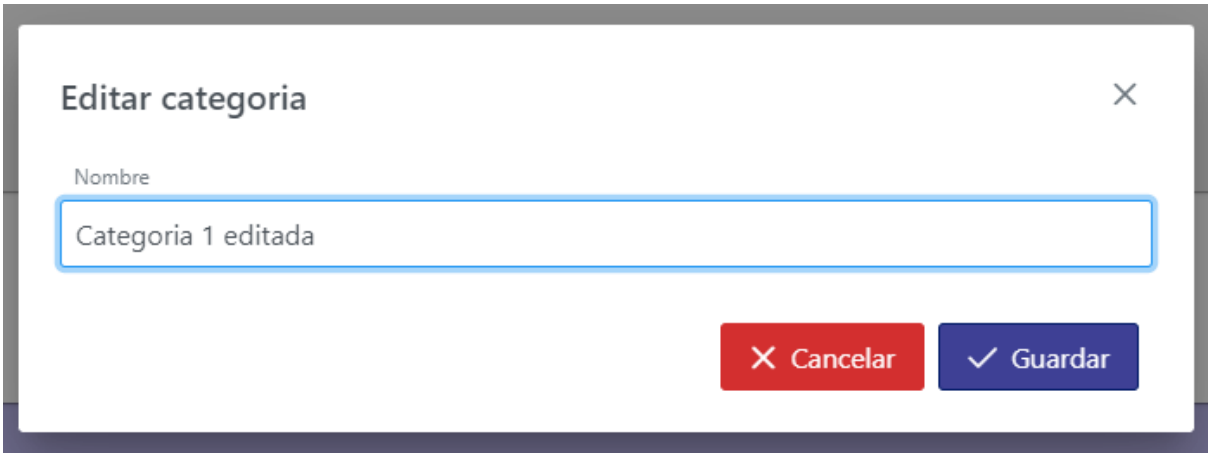
Se crea una nueva categoría:



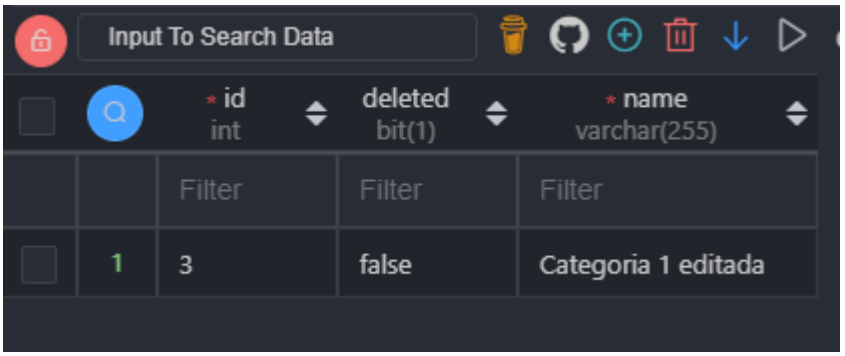
Y efectivamente ésta se guarda en la DB:



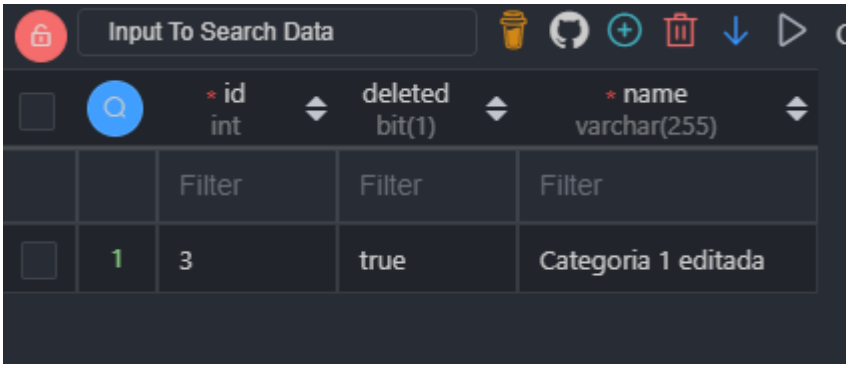
Ahora se la edita:



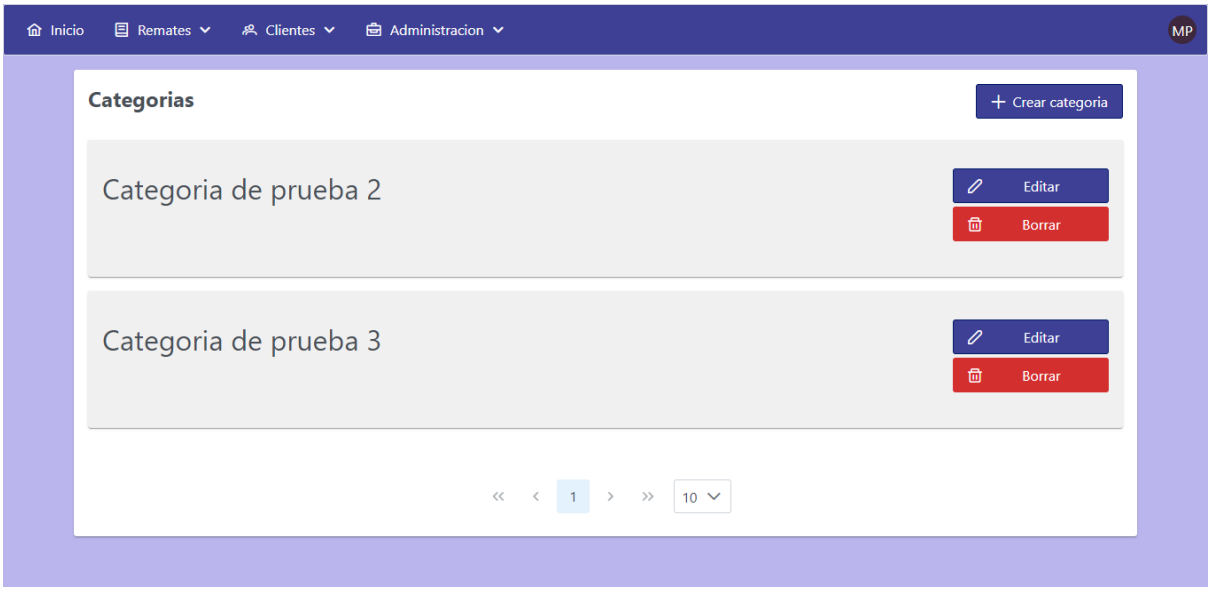
Y la misma se actualiza en la DB:



Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que la categoría esté marcada como “deshabilitado” en la base de datos (decisión de implementación a futuro).



Read: mostrar correctamente el listado de categoría de animales.



E.2.5 Historia de Usuario 7 - CRUD remates

E.2.5.1 Mockups

También en este caso se trata de una pantalla, la de crear/editar remate, pero además en este punto inició el desarrollo de la pantalla “Ver/Editar remate”, la cual luego sería continuada en los próximos Sprints a medida que se fueran desarrollando las funcionalidades que la misma contiene. Presentamos a continuación ambos bosquejos:

- **Crear/editar remate**: contiene un campo de texto para número de Senasa (solo números), input de fecha (texto o calendario en pantalla), input de hora (texto o selector en pantalla), input de localidad (autocomplete). Botones Cancelar, Aceptar y en caso de estar editando, un toggle para editar/dejar de hacerlo y un botón Eliminar.
- **Ver/Editar remate**: pantalla que actúa como “hub” de un remate. Esta pantalla involucra varias stories, en esta historia se crea el “esqueleto“ de la pantalla, debiendo luego terminarse en historias futuras: un botón del que se despliega un menú en overlay, con las opciones “Agregar lote“, “Agregar participante”, “Información del remate“ (opción que ya funciona, llevando al usuario a la pantalla de crear/editar remate), “Orden de salida“, “Resumen“, “Lotes vendidos“ y “Terminar remate“ (opción que también funciona). Un TabMenu (menú de pestañas) con las opciones "Para venta", "No vendidos" y "Vendidos", para entrar en los diferentes listados (vacíos por el momento).

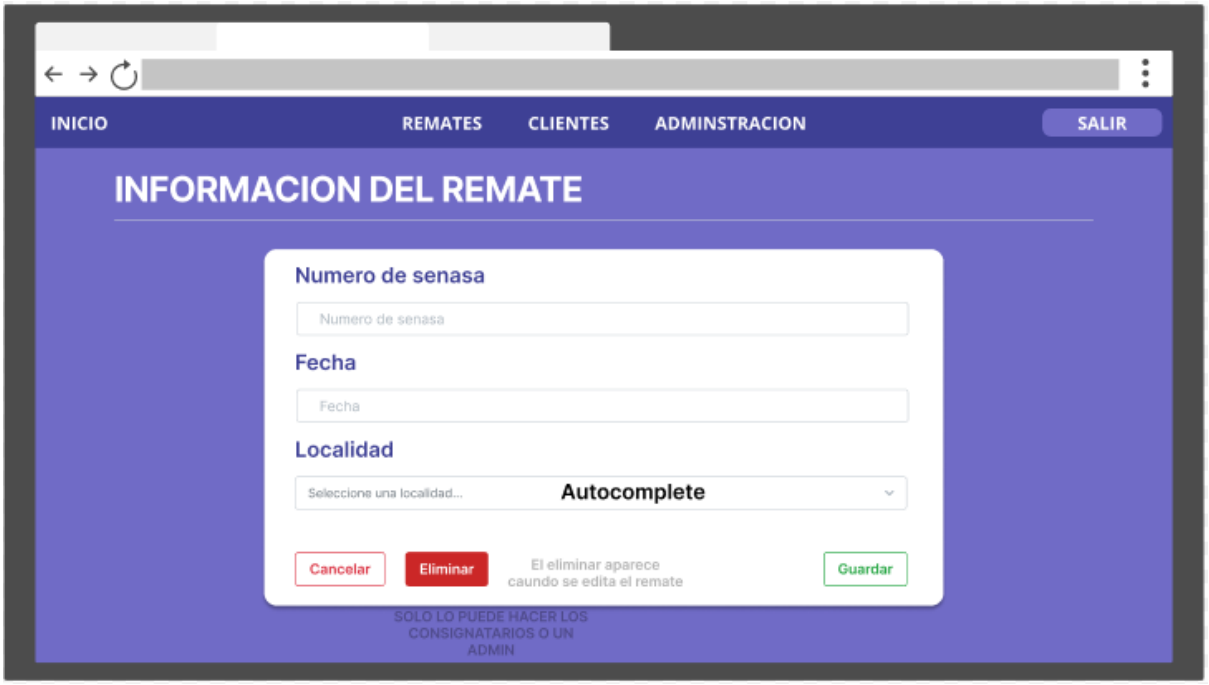


Figura 78: Mockup pantalla de crear/editar remate

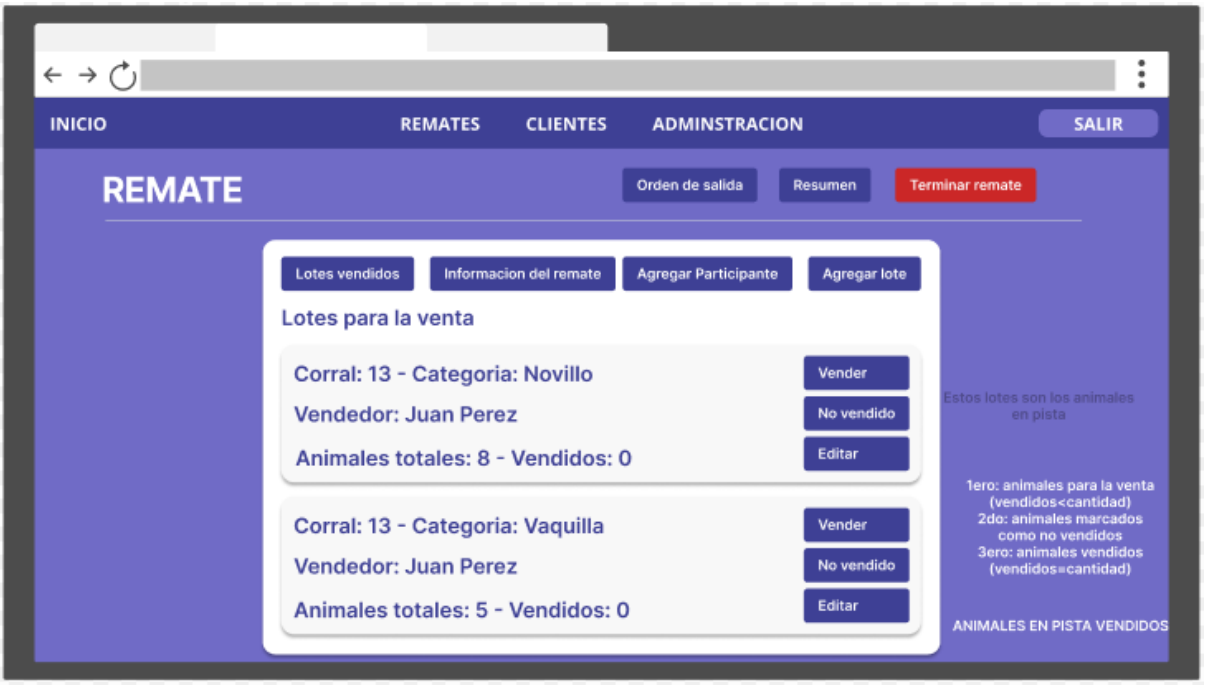


Figura 79: Mockup pantalla de ver/editar remate

E.2.5.2 Evolución del diagrama de clases

Como podría esperarse, con esta Historia de Usuario se agrega la clase “Remate”, comenzando a enlazar clases dentro del diagrama:

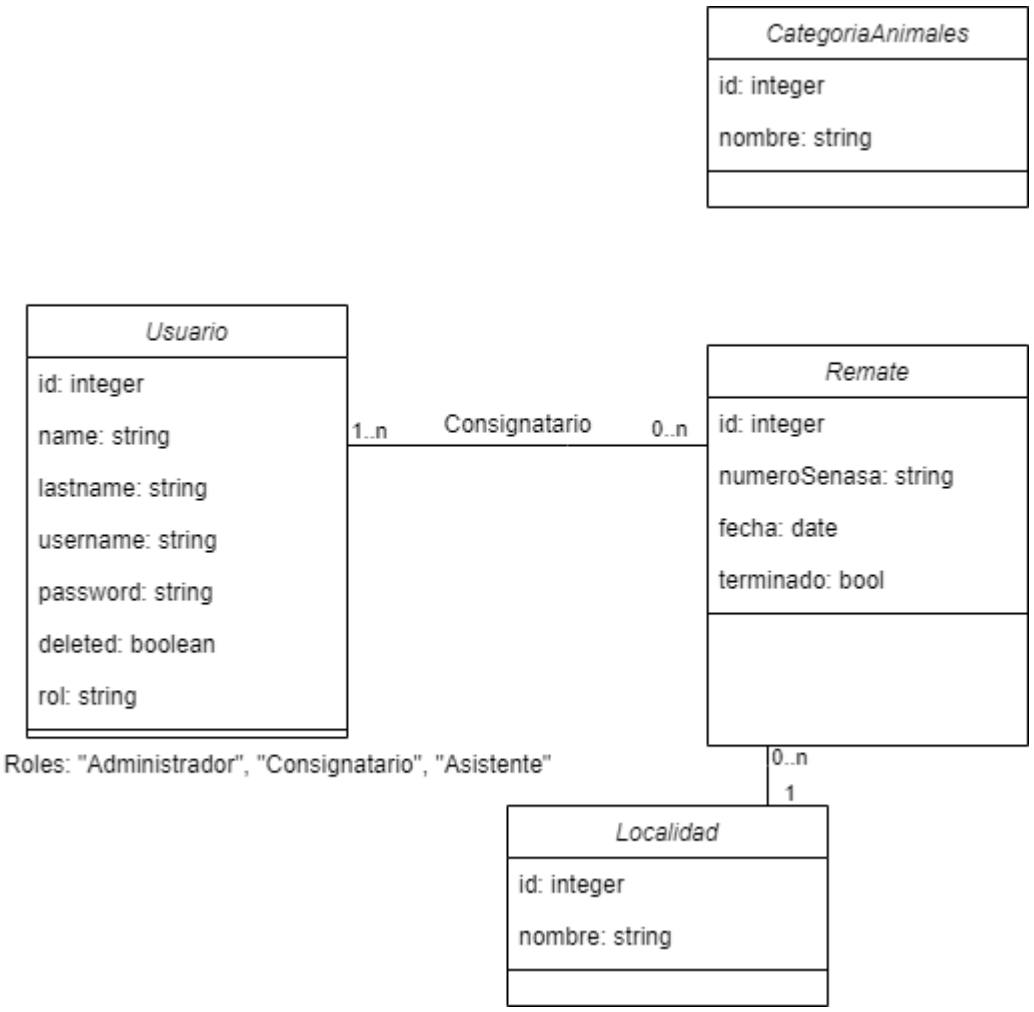


Figura 80: Diagrama de clases historia 7

E.2.5.3 Casos de prueba

Tests unitarios en AuctionServiceImpl:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Crear un remate con una localidad y una lista de usuarios.	Remate con nuevo ID.	Éxito
Crear un remate sin localidad.	N/A.	N/A. El error es gestionado por hibernate, y aquí se simula el acceso a la BD.
Crear un remate sin usuarios	N/A.	N/A. es de integración, se valida en el REST CONTROLLER.
Crear remate con fecha anterior a mañana	Exception	Éxito

Actualizar un remate con nueva localidad	Remate actualizado	Éxito
Actualizar remate con una fecha anterior a mañana	Exception	Éxito
Actualizar un remate con un usuario que no está autorizado (Rol asistente)	N/A.	
Actualizar un remate quitando todos los usuarios con rol “Consignatario” y “Administrador”.	Exception	Éxito
Actualizar remate con fecha futura correcta	Remate almacenado en la BD	Éxito
Actualizar un remate cuyo {id} no existe	Exception	Éxito
Borrar remate no finalizado	Remate actualizado con atributo “deleted” en true.	Éxito
Borrar remate finalizado	Exception	Éxito
Borrar remate cuyo {id} no existe	Exception	Éxito
Obtener los primeros diez remates (size 10, page 0)	Lista con 10 remates. N/A	Ignorar para este caso, ya que se simula el acceso a la BD

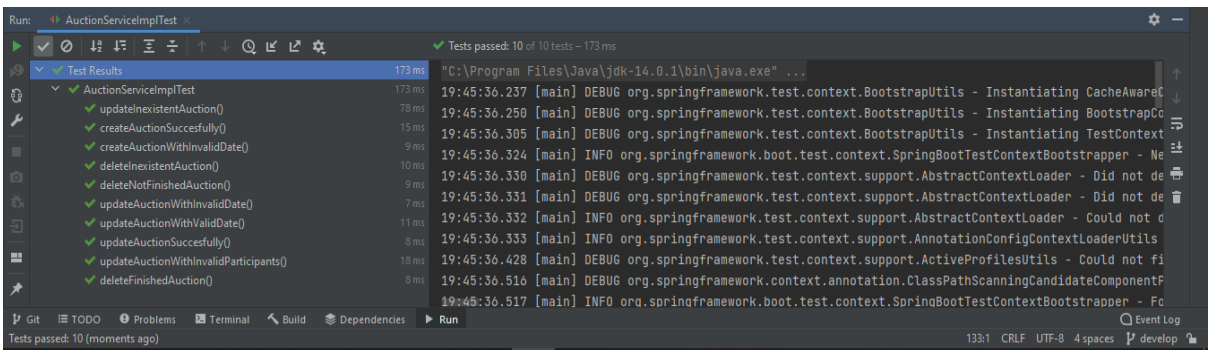


Figura 81: resultado de los tests

Tests integración en AuctionRest:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Crear un remate con una localidad y una lista de usuarios.	HTTP 200	Éxito
Crear un remate sin localidad.	HTTP 500 (porque el error es en hibernate)	Éxito
Crear un remate sin usuarios	HTTP 400	Éxito
Crear remate con fecha anterior a mañana	HTTP 400	Éxito
Actualizar un remate con nueva localidad	HTTP 200	Éxito
Actualizar remate con una fecha anterior a mañana	HTTP 400	Éxito
Actualizar un remate quitando todos los usuarios con rol “Consignatario” y “Administrador”.	HTTP 400	Éxito
Actualizar remate con fecha futura correcta	HTTP 200	Éxito
Actualizar un remate cuyo {id} no existe	HTTP 404	Éxito
Borrar remate no finalizado	HTTP 200	Éxito
Borrar remate finalizado	HTTP 403	Éxito
Borrar remate cuyo {id} no existe	HTTP 404	Éxito
Obtener los primeros 2 remates (size 2, page 0)	HTTP 200	Éxito (si hay al menos 2 remates)

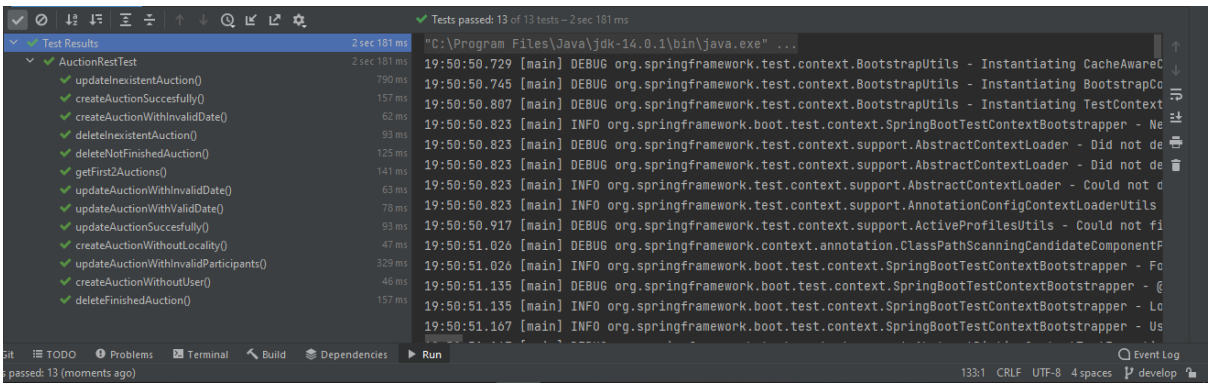


Figura 82: resultado de los tests

E.2.5.4 Criterios de aceptación

Create/update: verificar que los cambios se registraron correctamente en la base de datos.

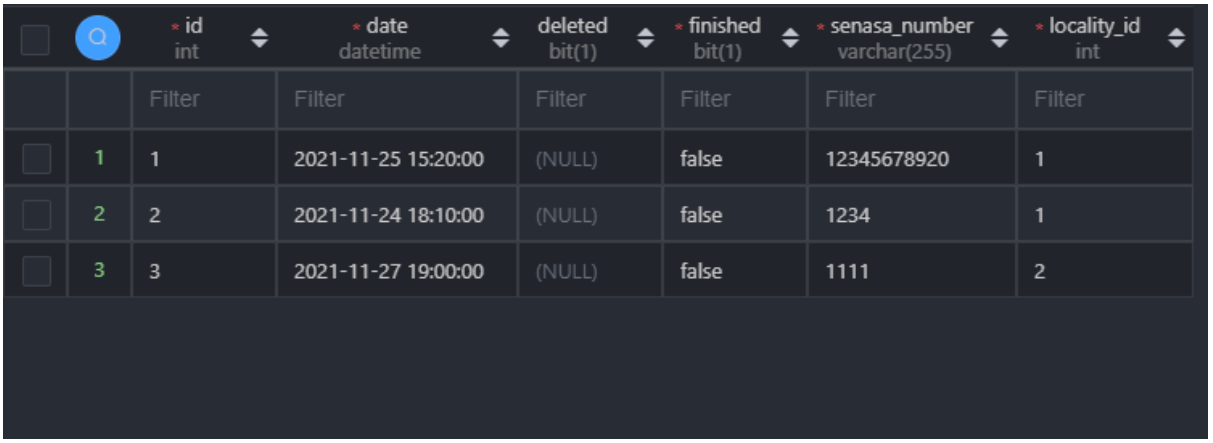
Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el remate esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro).

Read: mostrar correctamente el listado de remates.

Resultados:

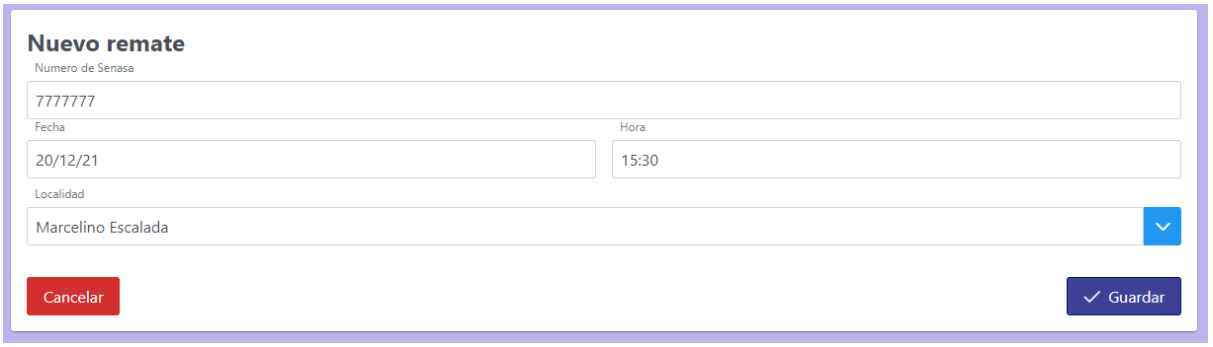
Create/update: verificar que los cambios se registraron correctamente en la base de datos.

Base de datos al inicio:



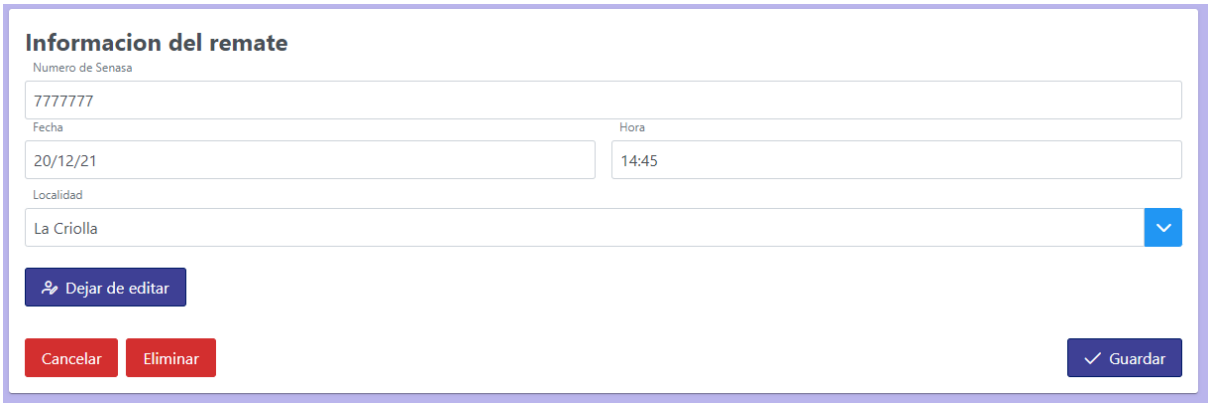
	id int	date datetime	deleted bit(1)	finished bit(1)	senasa_number varchar(255)	locality_id int
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2021-11-25 15:20:00	(NULL)	false	12345678920	1
2	2	2021-11-24 18:10:00	(NULL)	false	1234	1
3	3	2021-11-27 19:00:00	(NULL)	false	1111	2

Se crea un nuevo remate:



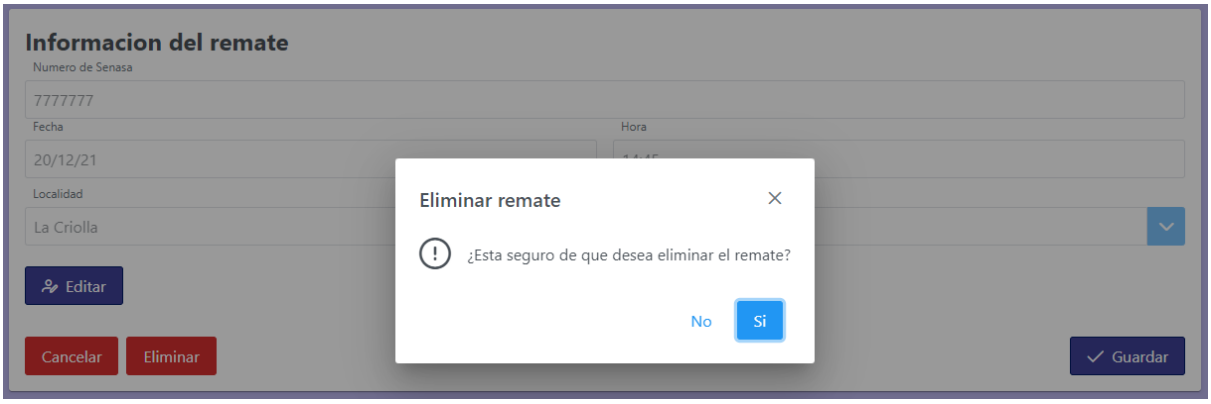
		* id int	* date datetime	deleted bit(1)	* finished bit(1)	* senasa_number varchar(255)	* locality_id int
		Filter	Filter	Filter	Filter	Filter	Filter
	1	1	2021-11-25 15:20:00	(NULL)	false	12345678920	1
	2	2	2021-11-24 18:10:00	(NULL)	false	1234	1
	3	3	2021-11-27 19:00:00	(NULL)	false	1111	2
	4	4	2021-12-20 15:30:00	(NULL)	false	7777777	2

Ahora se modifican algunos campos:



		* id int	* date datetime	deleted bit(1)	* finished bit(1)	* senasa_number varchar(255)	* locality_id int
		Filter	Filter	Filter	Filter	Filter	Filter
	1	1	2021-11-25 15:20:00	(NULL)	false	12345678920	1
	2	2	2021-11-24 18:10:00	(NULL)	false	1234	1
	3	3	2021-11-27 19:00:00	(NULL)	false	1111	2
	4	4	2021-12-20 14:45:00	(NULL)	false	7777777	1

Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el remate esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro).



		* id int	* date datetime	deleted bit(1)	* finished bit(1)	* senasa_number varchar(255)	* locality_id int
		Filter	Filter	Filter	Filter	Filter	Filter
	1	1	2021-11-25 15:20:00	(NULL)	false	12345678920	1
	2	2	2021-11-24 18:10:00	(NULL)	false	1234	1
	3	3	2021-11-27 19:00:00	(NULL)	false	1111	2
	4	4	2021-12-20 14:45:00	true	false	7777777	1

Read: mostrar correctamente el listado de remates.

E.2.6 Historia de Usuario 14 - Listado de remates

E.2.6.1 Mockup

Esta pantalla actúa como pantalla de inicio o “Home” del sistema, es la pantalla a la que accede el usuario una vez logueado.

En caso de los consignatarios y asistentes, se mostrará un TabMenu (menú de pestañas) con las opciones “Mis remates“ (remates donde el usuario participa), “Remates ajenos“ (remates donde no participa). En ambos casos se mostrará el listado con la siguiente información de cada remate: número de Senasa, fecha y hora, y lugar (localidad). Además, en el listado de remates propios cada uno tendrá 2 botones, “Agregar lote“ (aún sin funcionalidad) y “Ver“ (que lo llevará a la pantalla de ver/editar remate). En caso de los administradores será solo un listado con los botones, ya que puede acceder a cualquier remate. También se contará con paginación.

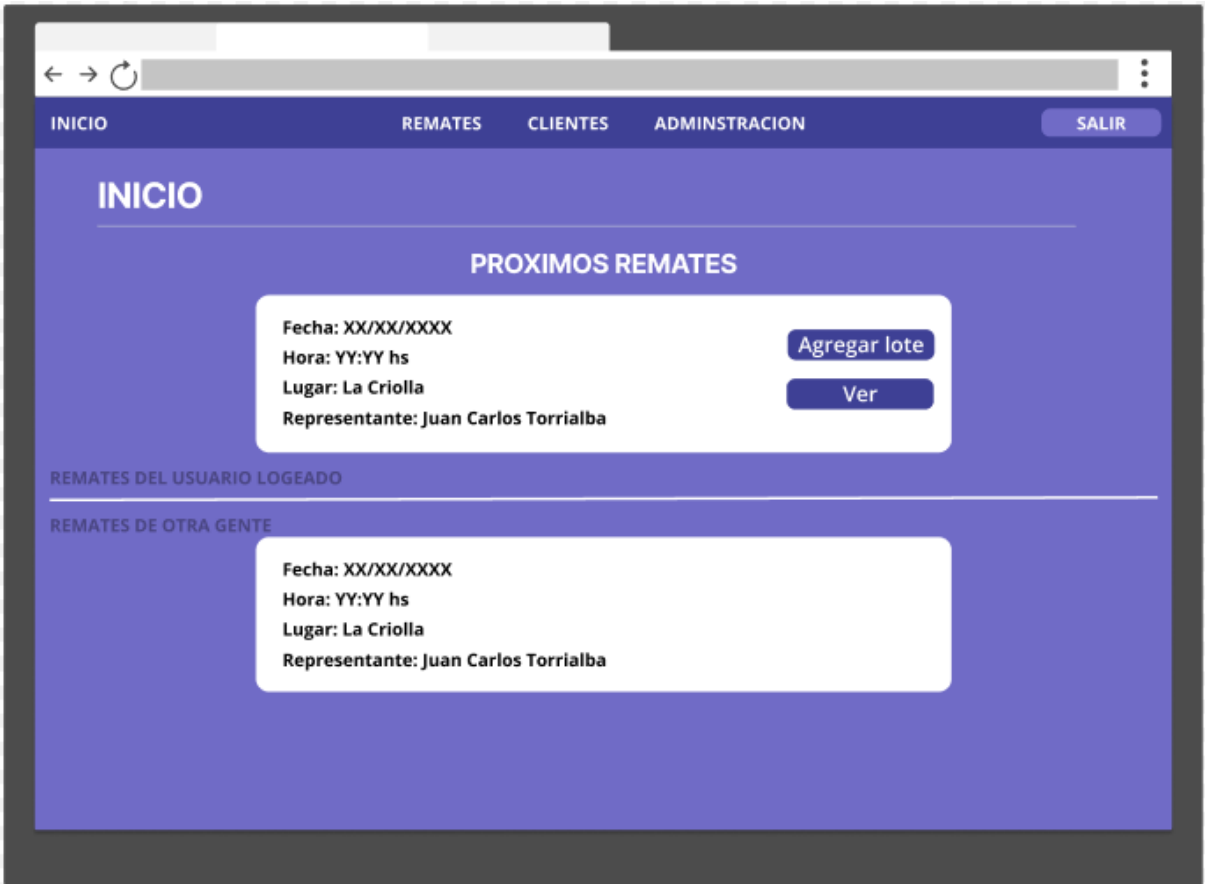


Figura 83: Mockup pantalla de listado de remates

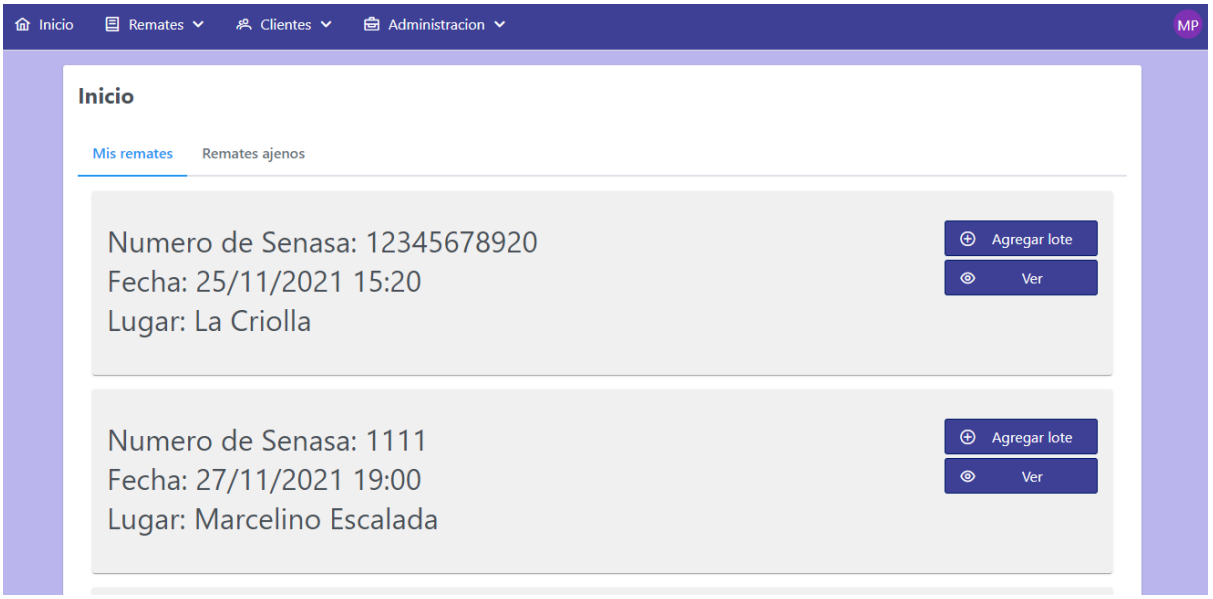
La descripción anterior no coincide completamente con el bosquejo mostrado, ya que al momento de pensarlo durante la confección del Plan de Proyecto se pensó como un solo listado dividido en dos (primero los remates en los que el usuario participa y luego los ajenos), pero al momento de implementarlo se cambió esa idea por el uso de dos pestañas, lo que generó que la imagen no refleje exactamente el resultado final.

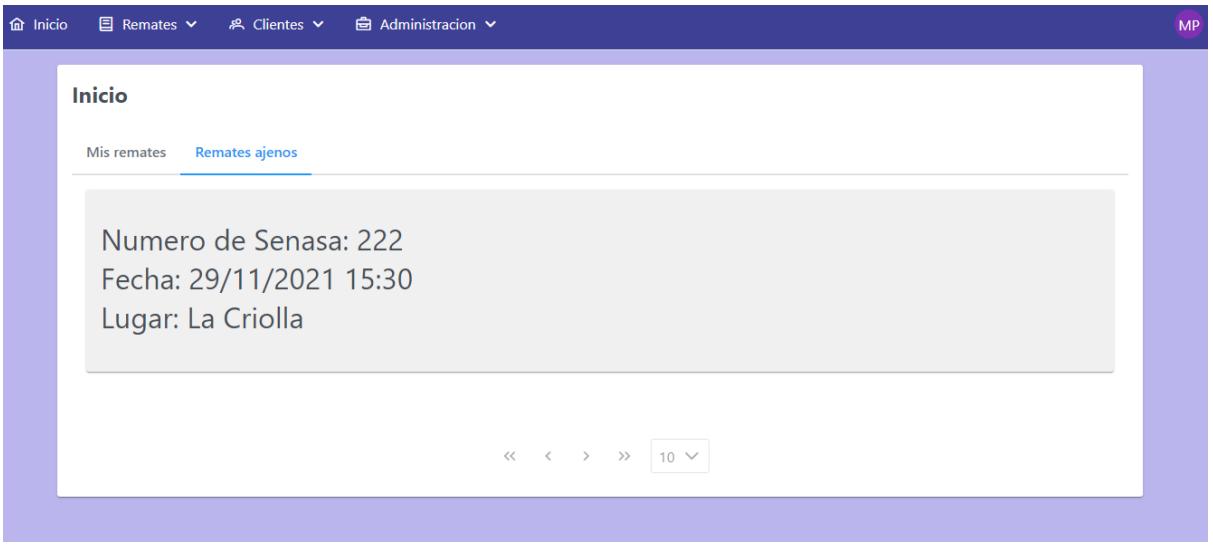
E.2.6.2 Criterios de aceptación

Cuando el usuario se loguea exitosamente, la pantalla debe mostrar una lista de remates en el siguiente orden: primero, remates próximos en los que participa, ordenados ascendentemente según la fecha de realización, con la posibilidad de ingresar a gestionar cada uno; segundo, los remates en los que no participa (creados por otros usuarios y donde no fue agregado como colaborador), ordenados ascendentemente por fecha de realización, sin la posibilidad de ingresar a la gestión de éstos.

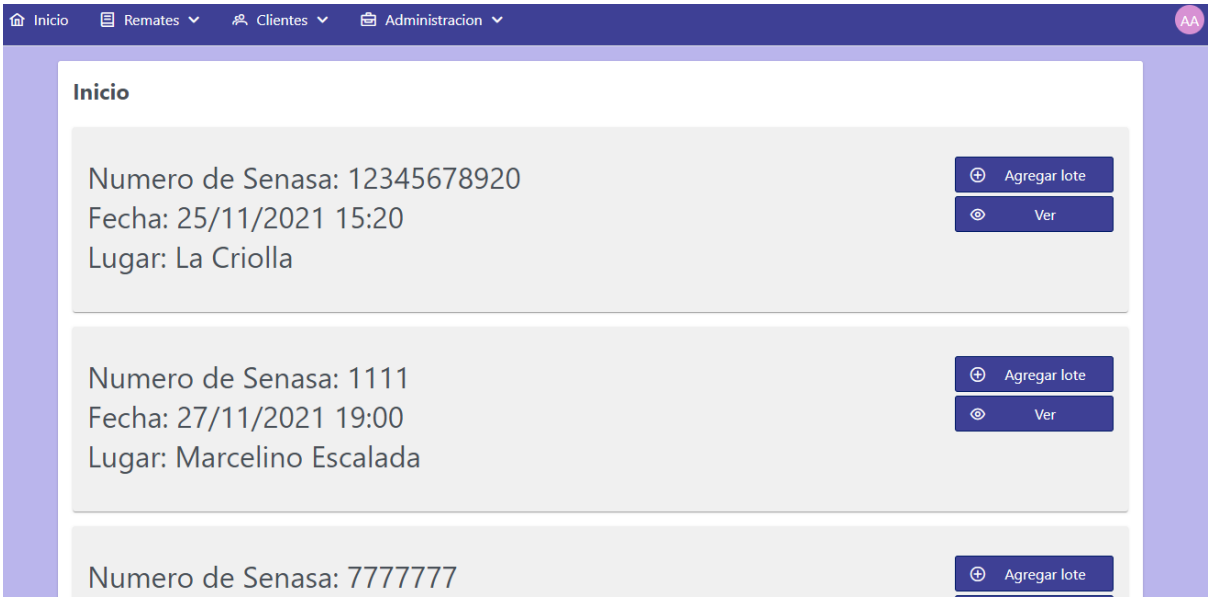
Resultados:

Encontramos una manera diferente de mostrar la información, dividiéndola en 2 listados que se pueden acceder desde 2 pestañas en la pantalla:





Y en el caso de los administradores, una sola pantalla, ya que pueden acceder a todos los remates:



E.2.7 Historia de Usuario 3 - CRUD de usuarios

E.2.7.1 Mockups

Como en anteriores CRUDs mostrados, en este caso también la Historia de Usuario cuenta con dos pantallas, siendo la primera **Crear/Editar Usuario**, la cual consta de los siguientes campos:

- Nombre (string)
- Apellido (string)
- Usuario (string, sujeto a no existir previamente en el sistema)
- Rol (Dropdown con las 3 posibilidades)
- Contraseña (string)
- Repetir contraseña
- Botones de “Cancelar”, “Guardar” y, en caso de estar editando, “Eliminar“.

La segunda es el **listado de usuarios**, una pantalla donde justamente se listarán los usuarios, mostrando sus nombres, usuario y rol (en principio), con la posibilidad de crear nuevos mediante un botón para ello, el cual redireccionará al usuario a la pantalla de Crear/Editar Usuario con los campos vacíos. Por cada uno de los usuarios listados habrá un botón “Eliminar“ (que mostrará un diálogo de confirmación) y un “Editar“, que redireccionará a la pantalla de Crear/Editar usuario, ahora mostrando los correspondientes datos. También se contará con paginación y una barra de búsqueda por nombre.

Cabe aclarar que solo los administradores podrán realizar acciones CRUD sobre usuarios, mostrándole los susodichos botones de creación/edición/borrado solo a quienes posean este rol. En caso de los consignatarios, ellos podrán acceder a esta pantalla pero

solo observar el listado, sin poder interactuar más allá de cambiar de página o realizar una búsqueda. Los asistentes ni siquiera tendrán acceso a esta funcionalidad.

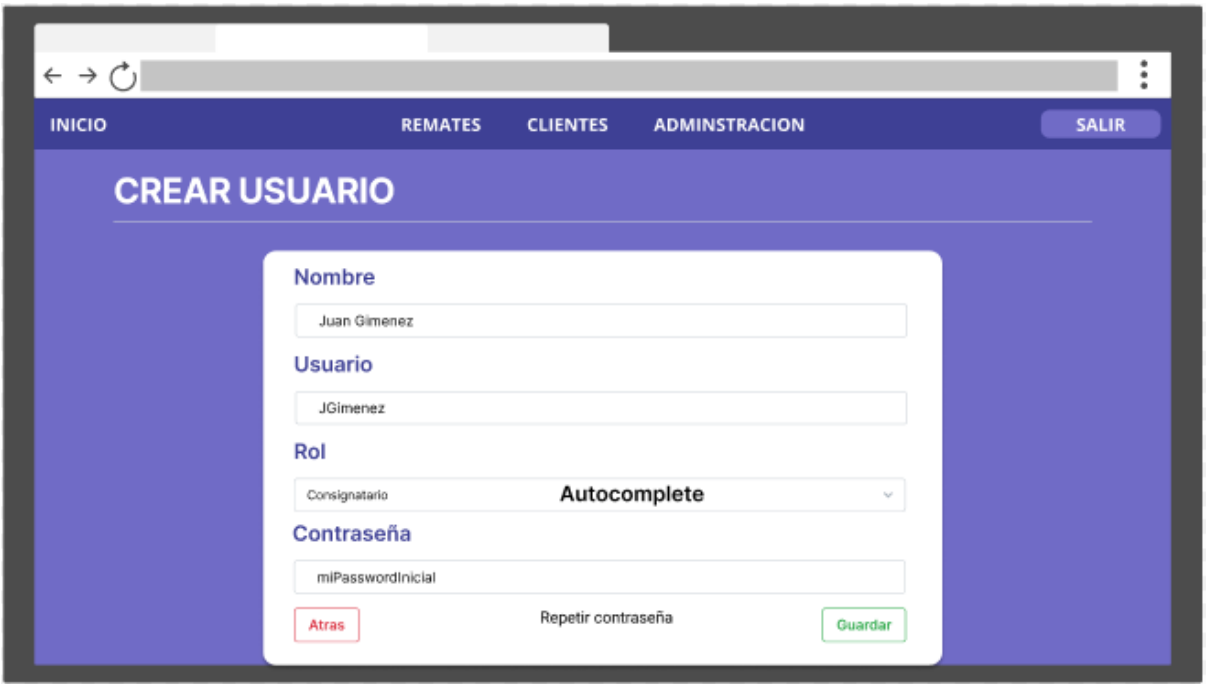


Figura 84: Mockup pantalla de crear/editar usuario

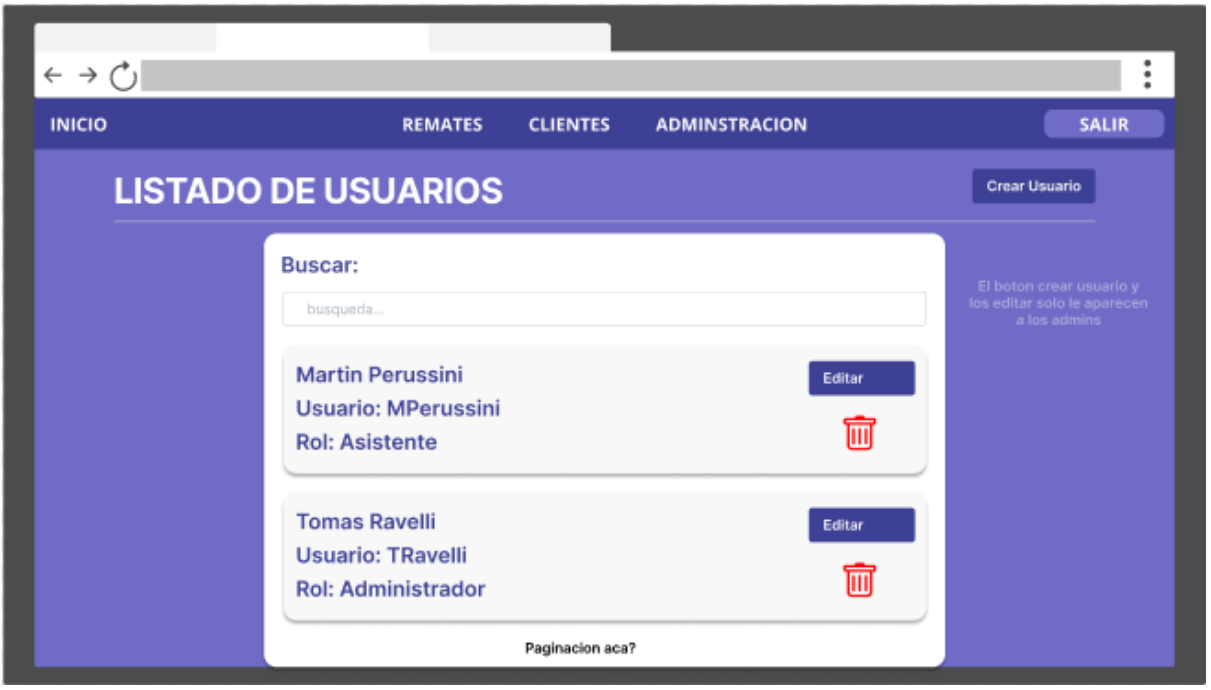


Figura 85: Mockup pantalla listado de usuarios

E.2.7.2 Casos de prueba

Tests unitarios en UserServiceImpl:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Borrar un usuario que exista	Cambios reflejados en la base de datos (deleted = true)	Éxito
Borrar un usuario que no exista	Excepción	Éxito
Crear un usuario con datos correctos	Cambios reflejados en la base de datos.	Éxito
Crear un usuario con username ya existente	Excepción	Éxito
Modificar password/username/name/lastname validos	Cambios realizados	Éxito
Modificar username con uno ya existente.	Excepción	Éxito
Modificar rol	Excepción	Éxito
Modificar usuario que no exista	Excepción	Éxito
Modificar la password con la misma que el usuario ya posee	Excepción	Éxito

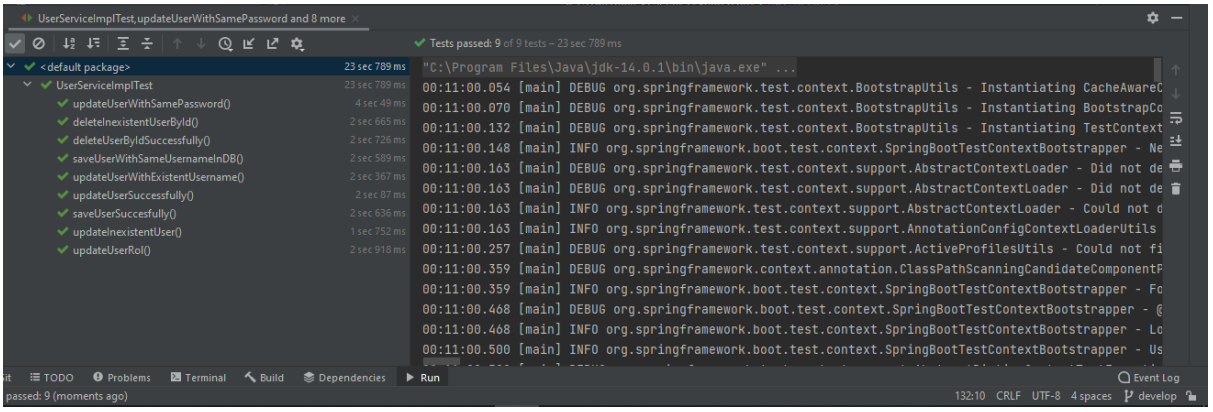


Figura 86: resultado de los tests

Tests de integración:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Borrar un usuario que exista (siendo admin)	HTTP 200	Éxito
Borrar un usuario que no exista (siendo admin)	HTTP 404	Éxito
Borrar un usuario (siendo consignatario/asistente)	HTTP 403	Esto no se puede con Junit porque la seguridad está desactivada para los tests. Probando en Insomnia, funciona correctamente.
Crear un usuario (siendo admin)	HTTP 200	Éxito
Crear un usuario (siendo consignatario/asistente)	HTTP 403	Éxito con Insomnia
Crear un usuario (siendo admin) con username ya existente	HTTP 403	Éxito
Modificar password/username/name/last name siendo admin	HTTP 200	Éxito
Modificar password/username/name/last name siendo asistente/consignatario	HTTP 403	Éxito con Insomnia
Modificar rol siendo asistente/consignatario/admin	HTTP 403	Éxito
Modificar usuario que no exista	HTTP 404	Éxito
Modificar la password con la misma que el usuario ya posee	HTTP 400	Éxito
Modificar el username con uno que ya existe en la DB	HTTP 403	Éxito

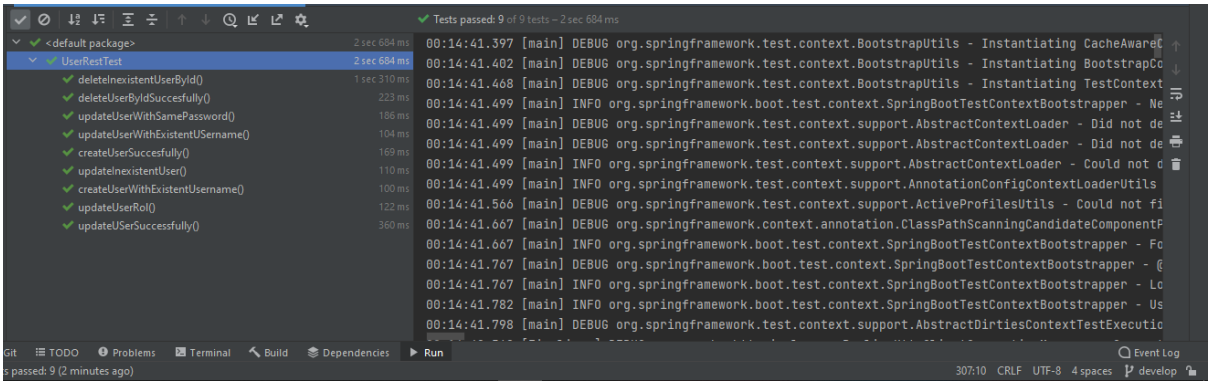


Figura 87: resultado de los tests

E.2.7.3 Criterio de aceptación

Create/update: verificar que los cambios se registraron correctamente en la base de datos.

Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el usuario esté marcado como "deshabilitado" en la base de datos (decisión de implementación a futuro).

Read: mostrar correctamente el listado de usuarios.

Resultados:

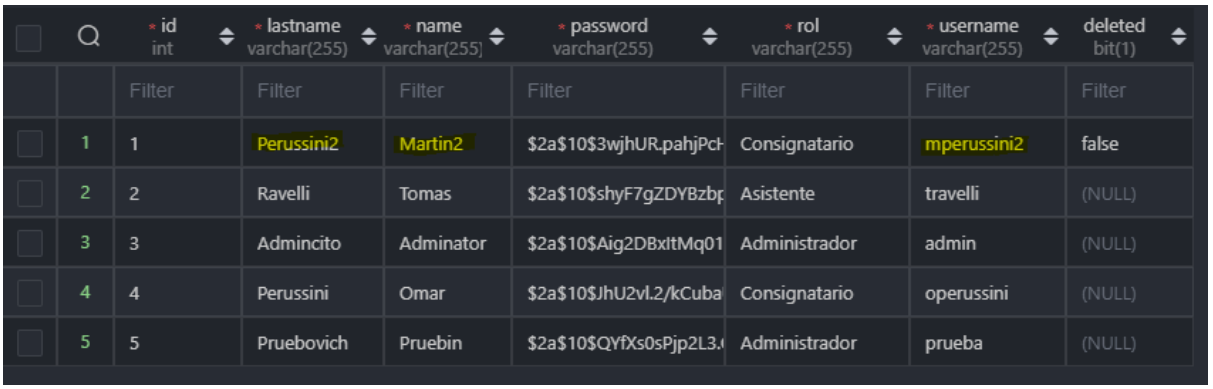
Create/update: verificar que los cambios se registraron correctamente en la base de datos.

DB al principio:

		* id int	* lastname varchar(255)	* name varchar(255)	* password varchar(255)	* rol varchar(255)	* username varchar(255)	deleted bit(1)
		Filter	Filter	Filter	Filter	Filter	Filter	Filter
	1	1	Perussini	Martin	$2a$10$s5d9yEr/kpJauH	Consignatario	mperussini	(NULL)
	2	2	Ravelli	Tomas	$2a$10$shyF7gZDYBzbp	Asistente	travelli	(NULL)
	3	3	Admincito	Adminator	$2a$10$Aig2DBxltMq01	Administrador	admin	(NULL)
	4	4	Perussini	Omar	$2a$10$JhU2vl.2/kCuba	Consignatario	operussini	(NULL)
	5	5	Pruebovich	Pruebin	$2a$10$QYfXs0sPjp2L3.(	Administrador	prueba	(NULL)

Edito uno de ellos:

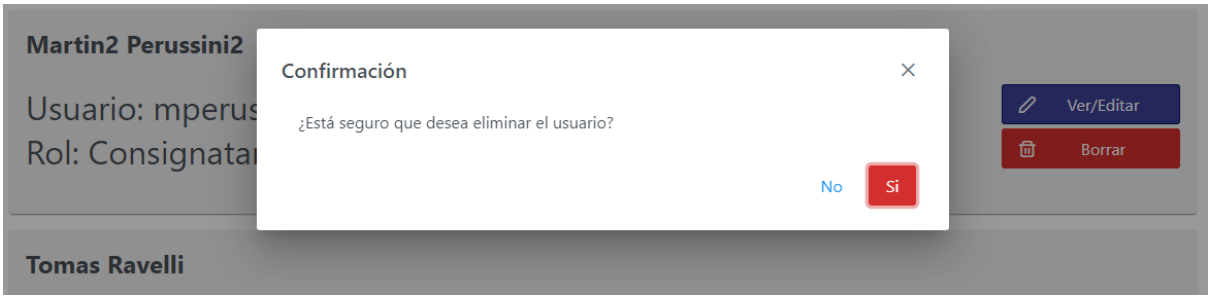
Y el mismo ha cambiado sus datos en la DB:



	id int	lastname varchar(255)	name varchar(255)	password varchar(255)	rol varchar(255)	username varchar(255)	deleted bit(1)
	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Perussini2	Martin2	$2a$10$3wjhUR.pahjPcH	Consignatario	mperussini2	false
2	2	Ravelli	Tomas	$2a$10$shyF7gZDYBzbp	Asistente	travelli	(NULL)
3	3	Admincito	Adminator	$2a$10$Aig2DBxItMq01	Administrador	admin	(NULL)
4	4	Perussini	Omar	$2a$10$JhU2vl.2/kCuba	Consignatario	operussini	(NULL)
5	5	Pruebovich	Pruebin	$2a$10$QYfXs0sPjp2L3.(	Administrador	prueba	(NULL)

Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el usuario esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro).

Se elimina el usuario:

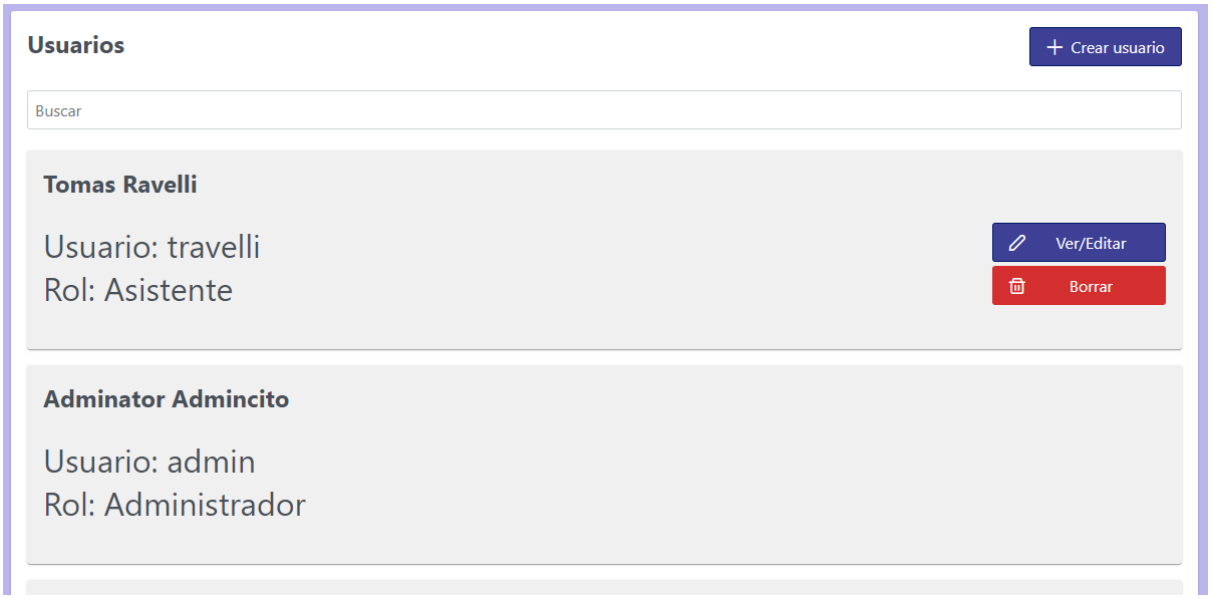


Y así figura en la DB:

		* id int	* lastname varchar(255)	* name varchar(255)	* password varchar(255)	* rol varchar(255)	* username varchar(255)	deleted bit(1)
		Filter	Filter	Filter	Filter	Filter	Filter	Filter
	1	1	Perussini2	Martin2	$2a$10$LEhpyWSfSC8q	Consignatario	mperussini2	true
	2	2	Ravelli	Tomas	$2a$10$shyF7gZDYBzbp	Asistente	travelli	(NULL)
	3	3	Admincito	Adminator	$2a$10$Aig2DBxItMq01	Administrador	admin	(NULL)
	4	4	Perussini	Omar	$2a$10$JhU2vl.2/kCuba	Consignatario	operussini	(NULL)
	5	5	Pruebovich	Pruebin	$2a$10$QYfXs0sPjp2L3.(	Administrador	prueba	(NULL)

Read: mostrar correctamente el listado de usuarios.

Como se puede observar, se muestran todos los usuarios no eliminados:



E.2.7.4 Código

```
@Service
public class UserServiceImpl implements UserService {
  ...
//CREATE
public User saveUser(User user) throws DuplicateUsernameException {
   Optional<User> u = findByUsername(user.getUsername());
   if(u.isPresent()){ //Si estamos modificando un usuario existente
       if(user.getId()!=null){
           if(user.getId().equals(u.get().getId())){
              return userDAO.save(user);
           }
           throw new DuplicateUsernameException("Ya existe un Usuario con este
username.");
       }
       throw new DuplicateUsernameException("Ya existe un Usuario con este username.");
   }
   //Si estamos creando un nuevo usuario
   user.setPassword(passwordEncoder.encode(user.getPassword()));
   return userDAO.save(user);
```

```
}
...
}
```

```
@Service
public class UserServiceImpl implements UserService {
  ...
//DELETE
@Override
public User deleteUserById(Integer id) throws DuplicateUsernameException,
UserNotFoundException {
   User u = findUserById(id);
   if(u.isDeleted() != null && u.isDeleted()){
       throw new UserNotFoundException("No existe Usuario con id: " + id);
   }
   u.setDeleted(true);
   return saveUser(u);
}
...
}
```

```
@Service
public class UserServiceImpl implements UserService {
  ...
//UPDATE
@Override
public User updateUserById(Integer id, UserDTO fields) throws
DuplicateUsernameException, InvalidCredentialsException, HttpForbidenException {
   if(fields.getId() != null){
       if(!fields.getId().equals(id)){
           throw new InvalidCredentialsException("No se puede modificar el id del
Usuario");
       }
   }
   User user = findUserById(id);
   if(fields.getRol() != null && !fields.getRol().equals(user.getRol())) {
       throw new HttpForbidenException("Los roles no pueden modificarse");
   }
   if(fields.getPassword() != null){
       String newPassword = fields.getPassword();
       if(passwordEncoder.matches(newPassword, user.getPassword())){
           throw new InvalidCredentialsException("Las contraseñas son iguales");
       }
       user.setPassword(passwordEncoder.encode(newPassword));
       fields.setPassword(null);
   }
   userMapper.updateUserFromDto(fields, user);
   return saveUser(user);
}
...
}
```

```
@Repository
public interface UserDAO extends JpaRepository<User, Integer> {
  ...
//READ
@Query("Select u from User u where u.username like %:username% and (u.deleted is null or
u.deleted = false)")
Page<User> findByUsernameAndNotDeleted(String username, Pageable of);

@Query("Select u from User u where (u.name like %:name% or u.lastname like %:name%) and
(u.deleted is null or u.deleted = false) order by u.lastname")
Page<User> getUsersNotDeletedByName(String name, Pageable of);

@Query("Select u from User u where (u.name like %:name% or u.lastname like %:name%) and
(u.deleted is null or u.deleted = false) and u.id != :userId and u.rol !=
'Administrador' order by u.lastname")
List<User> findByNameAndRolAndNotId(String name, Integer userId);
...
}
```

E.2.8 Historia de Usuario 4 - CRUD clientes

E.2.8.1 Mockups

Al tratarse también de un CRUD, aquí tenemos por un lado la pantalla donde se crea/edita la entidad en cuestión, y por otro el listado de las mismas.

Para la primera tenemos 2 partes de la pantalla, la primera para cargar los datos del cliente propiamente, el Nombre/Denominación y el número de CUIT (opcional), y luego múltiples entradas, en forma de tarjetas, para cargar las procedencias del mismo, debiéndose cargar una referencia, el número de Renspa y la localidad (un autocomplete), además de un botón “Eliminar”. Al final, un botón “Agregar“ para sumar una nueva procedencia, “Cancelar” y “Guardar”.

En la otra pantalla se listan los clientes, mostrando sus nombres, con la posibilidad de crear nuevos mediante un botón para ello, el cual redireccionará al usuario a la pantalla de Crear/Editar cliente con los campos vacíos. Por cada uno de los clientes listados habrá un botón “Eliminar“ (que mostrará un diálogo de confirmación) y un “Editar“, que redireccionará a la pantalla de Crear/Editar cliente, ahora mostrando los correspondientes datos. También se contará con paginación y una barra de búsqueda por nombre.

Figura 88: Mockup pantalla crear/editar cliente

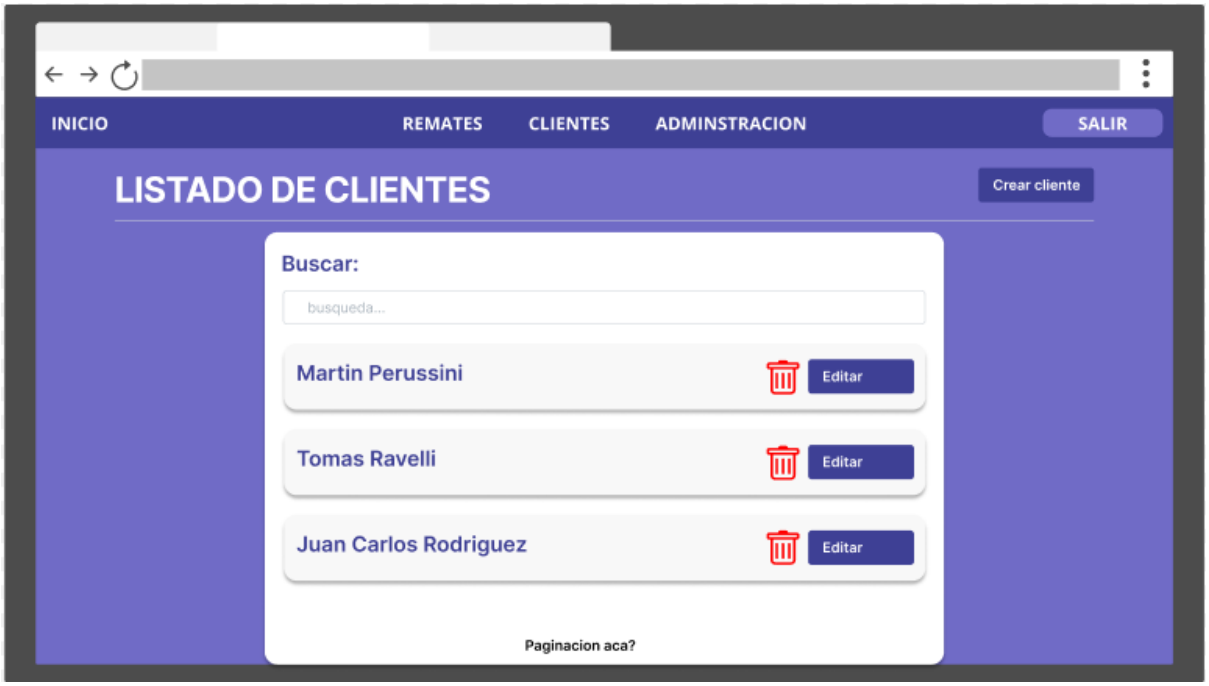


Figura 89: Mockup pantalla listado de clientes

E.2.8.2 Evolución del diagrama de clases

En este caso se agregan al diagrama dos nuevas clases: Cliente y Procedencia, junto a sus respectivas relaciones.

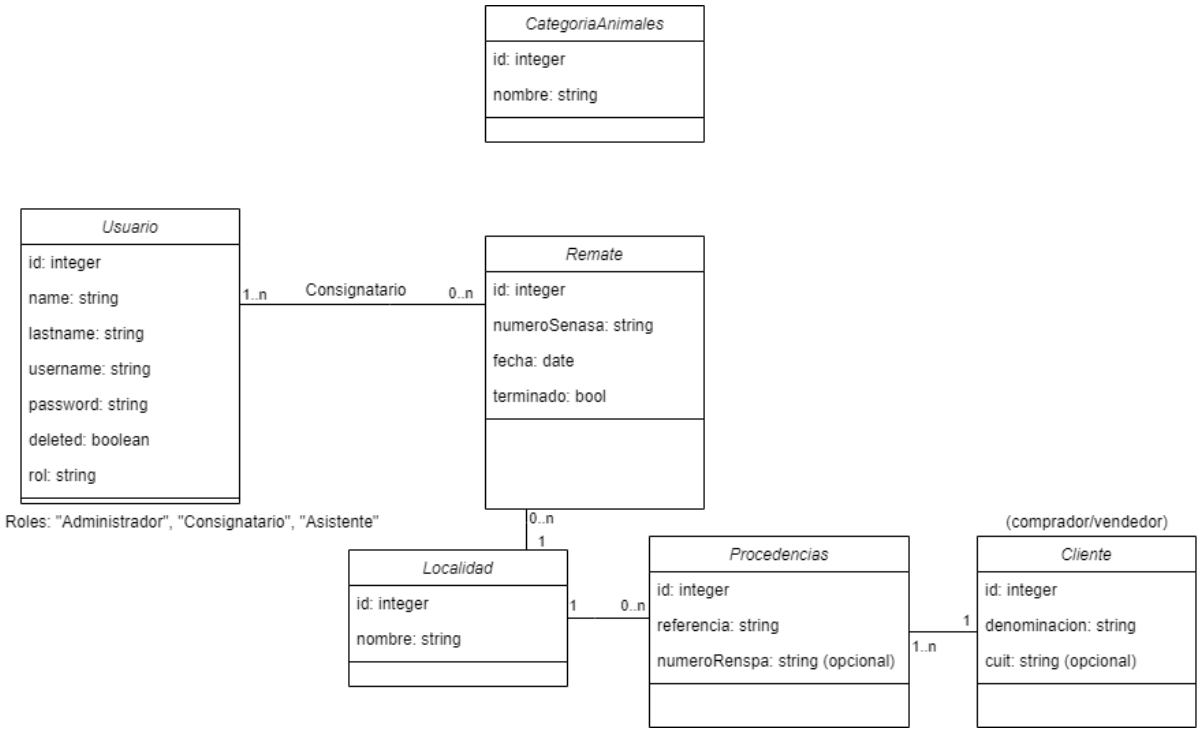


Figura 90: Diagrama de clases historia 4

E.2.8.3 Casos de prueba

Tests unitarios ClientServiceImpl:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Crear un cliente con datos correctos.	Cliente con el id nuevo retornado.	Éxito
Crear un cliente sin procedencia.	Excepción	Éxito
Crear un cliente sin denominación.	Excepción	Éxito
Eliminar un cliente existente.	atributo "deleted" igual a TRUE	Éxito
Eliminar un usuario que ya fue eliminado	Excepción	Éxito
Eliminar un usuario inexistente	Excepción	Éxito
Actualizar un cliente, cualquier datos que no sea procedencia.	Dato modificado en el cliente.	Éxito
Actualizar cliente: Eliminar todas las procedencias	Verificar que las procedencias se restablecen, en este caso lo hacemos con el número de llamadas al save() de provenanceDAO.	Éxito
Actualizar cliente: eliminar algunas procedencias	Procedencias con atributo deleted en TRUE y no se muestran en el cliente. CAMBIO: para el caso del test unitario verificamos la cantidad de veces que se llamó al método save() de ProvenanceDAO para verificar la cantidad de procedencias que se eliminaron.	Éxito
Actualizar cliente: Eliminar todas las procedencias y agregar otras.	Cliente modificado con las nuevas procedencias. Se testea el ClientMapper en este caso para verificar las nuevas procedencias.	Éxito

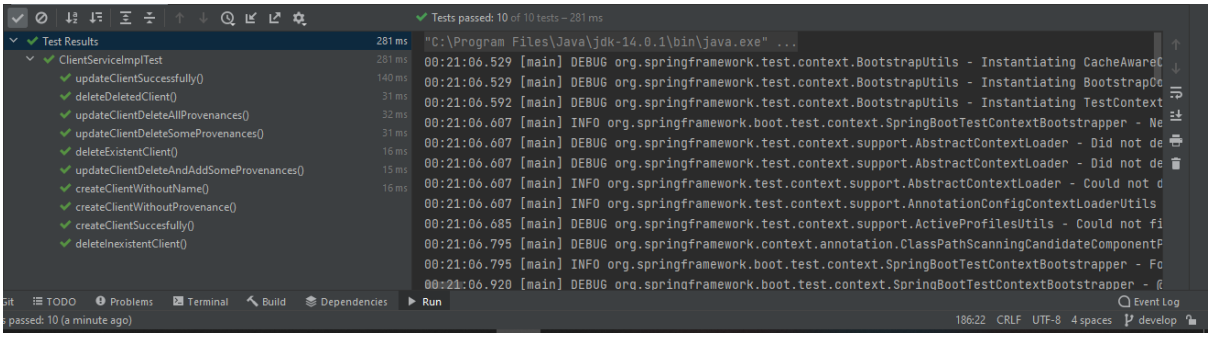


Figura 91: resultado de los tests

Tests integration ClientRest:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Crear un cliente con datos correctos.	HTTP 200. El cliente retornado tiene id.	Éxito
Crear un cliente sin procedencia.	HTTP 400	Éxito
Crear un cliente sin denominación.	HTTP 400	Éxito
Eliminar un cliente existente.	HTTP 200 cliente retornado: atributo "deleted" igual a TRUE	Éxito
Eliminar un cliente que ya fue eliminado	HTTP 404	Éxito
Eliminar un cliente inexistente	HTTP 404	Éxito
Listar usuarios, la primera página con un tamaño de 2.	HTTP 200, retornando una página de 2 cliente en el body	Éxito
Buscar client por id	usuario devuelto ya sea eliminado o no.	Éxito
Actualizar un cliente, cualquier datos que no sea procedencia.	HTTP 200 Cliente retornado con los nuevos datos.	Éxito
Actualizar cliente: Eliminar todas las procedencias	HTTP 400 Cliente retornado: mantiene las mismas procedencias	Éxito
Actualizar cliente: eliminar algunas procedencias	HTTP 200 Cliente retornado: sin las procedencias eliminadas.	Éxito

Actualizar cliente: Eliminar todas las procedencias y agregar otras.	HTTP 200 Cliente retornado: con las nuevas procedencias.	Éxito

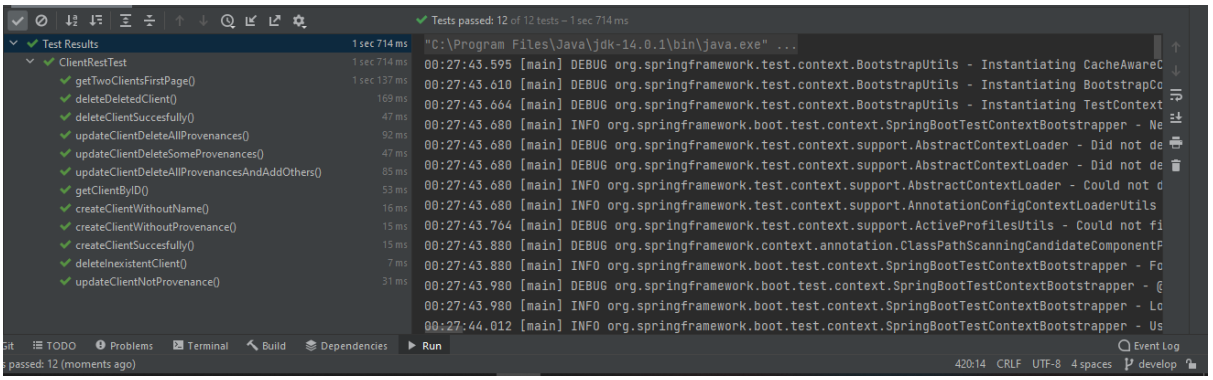


Figura 92: resultado de los tests

E.2.8.4 Criterio de aceptación

Create/update: verificar que los cambios se registraron correctamente en la base de datos.

Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el cliente esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro).

Read: mostrar correctamente el listado de clientes.

Resultados:

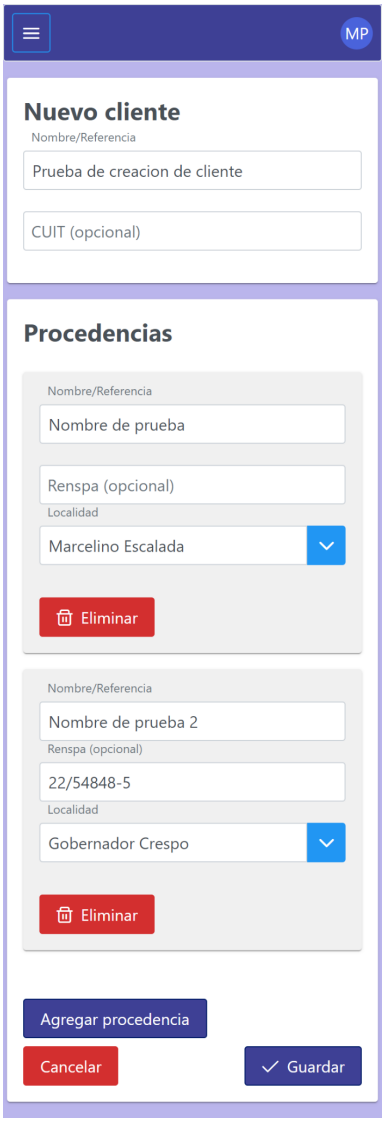
Create/update: verificar que los cambios se registraron correctamente en la base de datos.

DB al inicio (primero la de clientes, segundo la de procedencias):

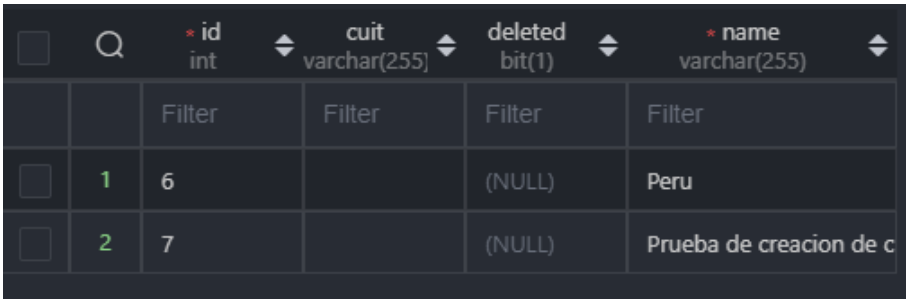
	* id int	cuit varchar(255)	deleted bit(1)	* name varchar(255)
	Filter	Filter	Filter	Filter
1	6		(NULL)	Peru

	id int	reference varchar(255)	renspa_number varchar(255)	locality_id int	deleted bit(1)	client_id int
	Filter	Filter	Filter	Filter	Filter	Filter
1	15	Campo de La Criolla		1	(NULL)	6
2	16	Campo de La Criolla2		1	true	6
3	17	Campo 2	321312	2	true	6
4	18	Campo 2	20-521515/8	3	(NULL)	6
5	19	Campo 3		3	true	6

Se crea un cliente:



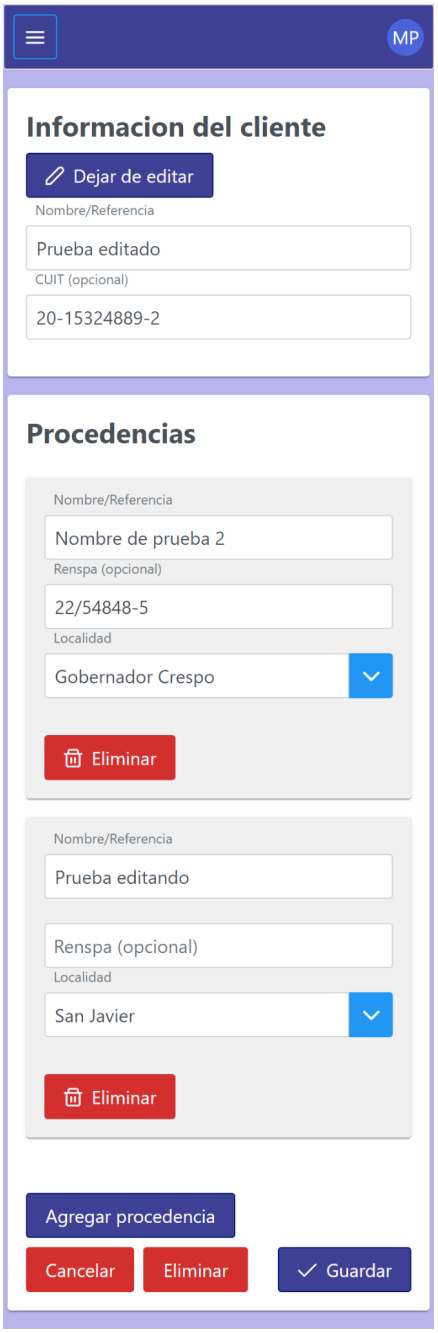
Nuevo estado de las DB:



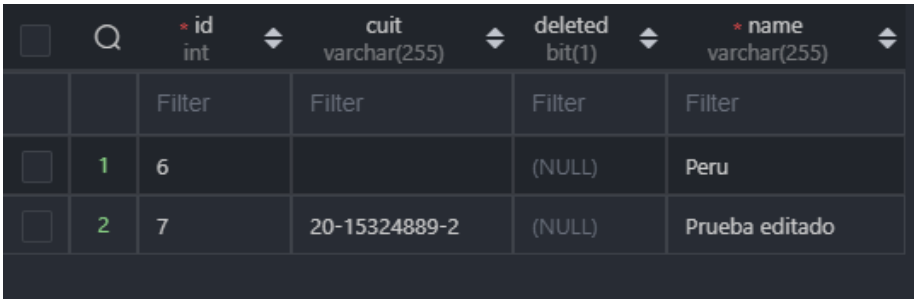
	* id int	cuit varchar(255)	deleted bit(1)	* name varchar(255)
	Filter	Filter	Filter	Filter
1	6		(NULL)	Peru
2	7		(NULL)	Prueba de creacion de c

	* id int	* reference varchar(255)	renspa_number varchar(255)	* locality_id int	deleted bit(1)	client_id int
	Filter	Filter	Filter	Filter	Filter	Filter
1	15	Campo de La Criolla		1	(NULL)	6
2	16	Campo de La Criolla2		1	true	6
3	17	Campo 2	321312	2	true	6
4	18	Campo 2	20-521515/8	3	(NULL)	6
5	19	Campo 3		3	true	6
6	20	Nombre de prueba		2	(NULL)	7
7	21	Nombre de prueba 2	22/54848-5	5	(NULL)	7

Se edita, eliminando una procedencia y agregando una nueva:



Nuevo estado de las DB:

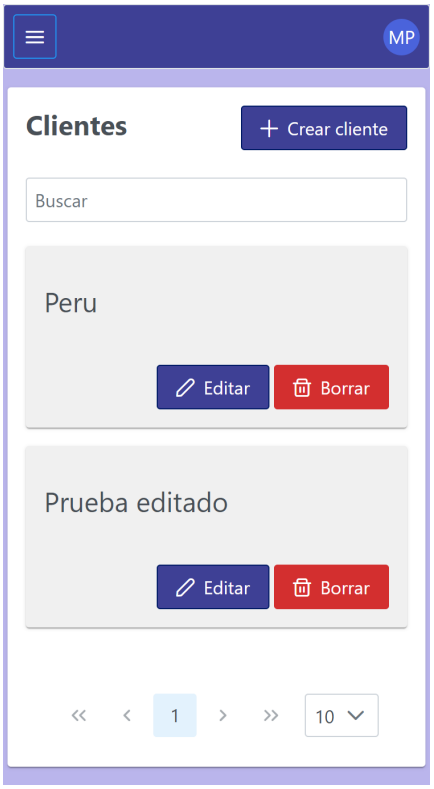


		* id int	cuit varchar(255)	deleted bit(1)	* name varchar(255)
		Filter	Filter	Filter	Filter
	1	6		(NULL)	Peru
	2	7	20-15324889-2	(NULL)	Prueba editado

		* id int	* reference varchar(255)	renspa_number varchar(255)	* locality_id int	deleted bit(1)	client_id int
		Filter	Filter	Filter	Filter	Filter	Filter
	1	15	Campo de La Criolla		1	(NULL)	6
	2	16	Campo de La Criolla2		1	true	6
	3	17	Campo 2	321312	2	true	6
	4	18	Campo 2	20-521515/8	3	(NULL)	6
	5	19	Campo 3		3	true	6
	6	20	Nombre de prueba		2	true	7
	7	21	Nombre de prueba 2	22/54848-5	5	(NULL)	7
	8	22	Prueba editando		3	(NULL)	7

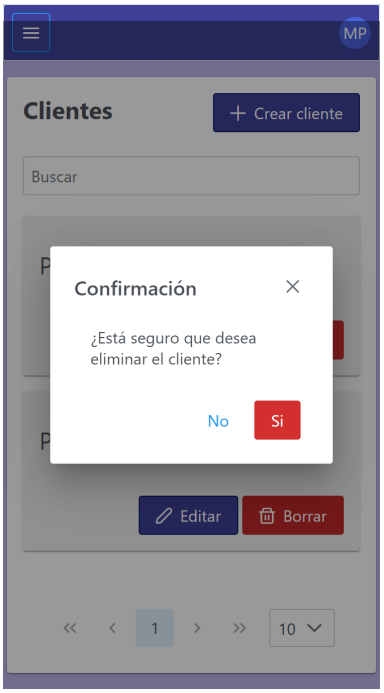
Read: mostrar correctamente el listado de clientes.

El cliente se muestra en el listado:

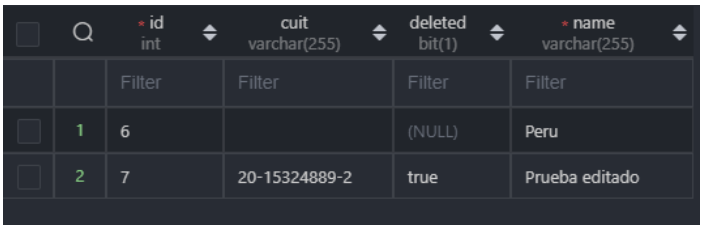


Remove: verificar que el registro y relaciones, en caso de ser necesario, se hayan eliminado de la base de datos. O bien, que el cliente esté marcado como “deshabilitado” en la base de datos (decisión de implementación a futuro).

Se elimina:

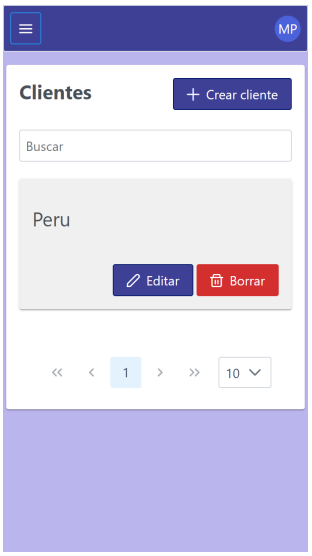


Estado de la DB:



	* id int	cuit varchar(255)	deleted bit(1)	* name varchar(255)
	Filter	Filter	Filter	Filter
1	6		(NULL)	Peru
2	7	20-15324889-2	true	Prueba editado

Tampoco aparece en el listado:



E.2.9 Historia de Usuario 8 - Asignar asistentes a un remate

E.2.9.1 Mockup

Pantalla donde aparecerán listados los usuarios previamente asignados (si hubiese alguno) y donde se podrán agregar más. Para ello se contará con un botón “Agregar“ que crea una nueva entrada, la cual consiste en un Autocomplete donde se ingresará el nombre del usuario, y un botón “Eliminar”. Al final también están los botones “Cancelar“ y “Aceptar“

En la implementación se reemplazó la acción del botón “Agregar”, que en lugar de crear una nueva tarjeta, abre un diálogo que contiene el autocomplete junto a un botón “Aceptar” y un “Cancelar”.

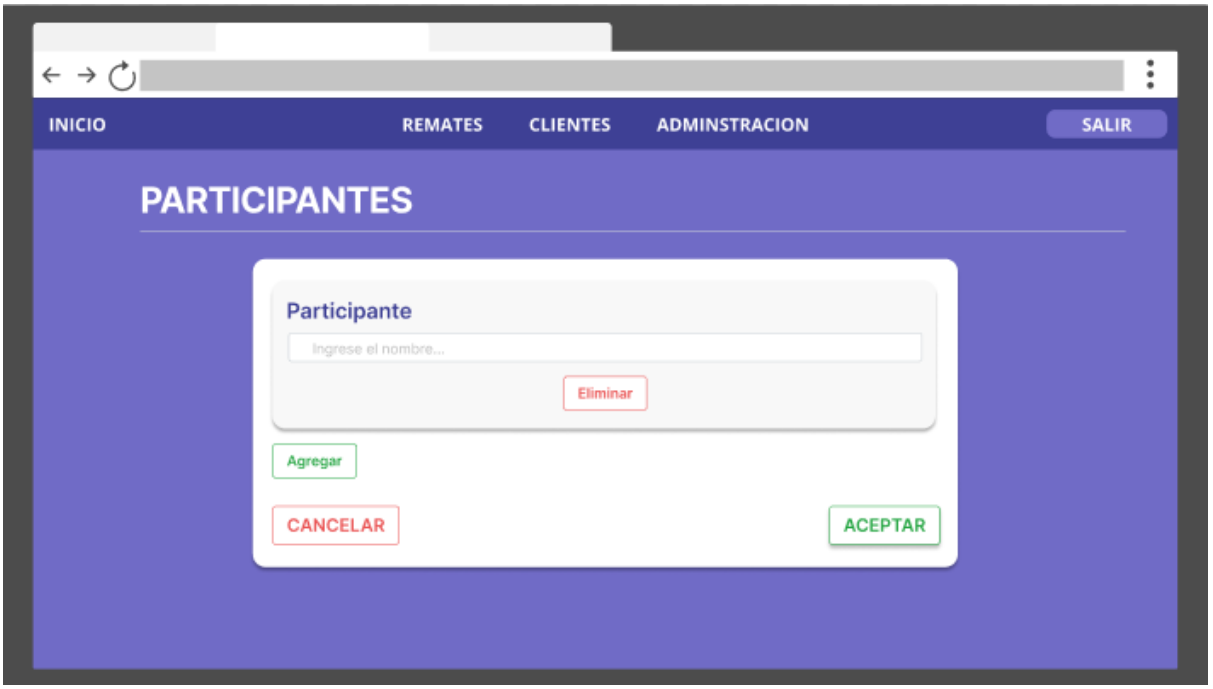


Figura 93: Mockup pantalla agregar participante

E.2.9.2 Evolución del diagrama de clases

En este caso el diagrama solo se ve modificado por el agregado de una relación extra entre las clases Usuario y Remate, como se muestra a continuación:

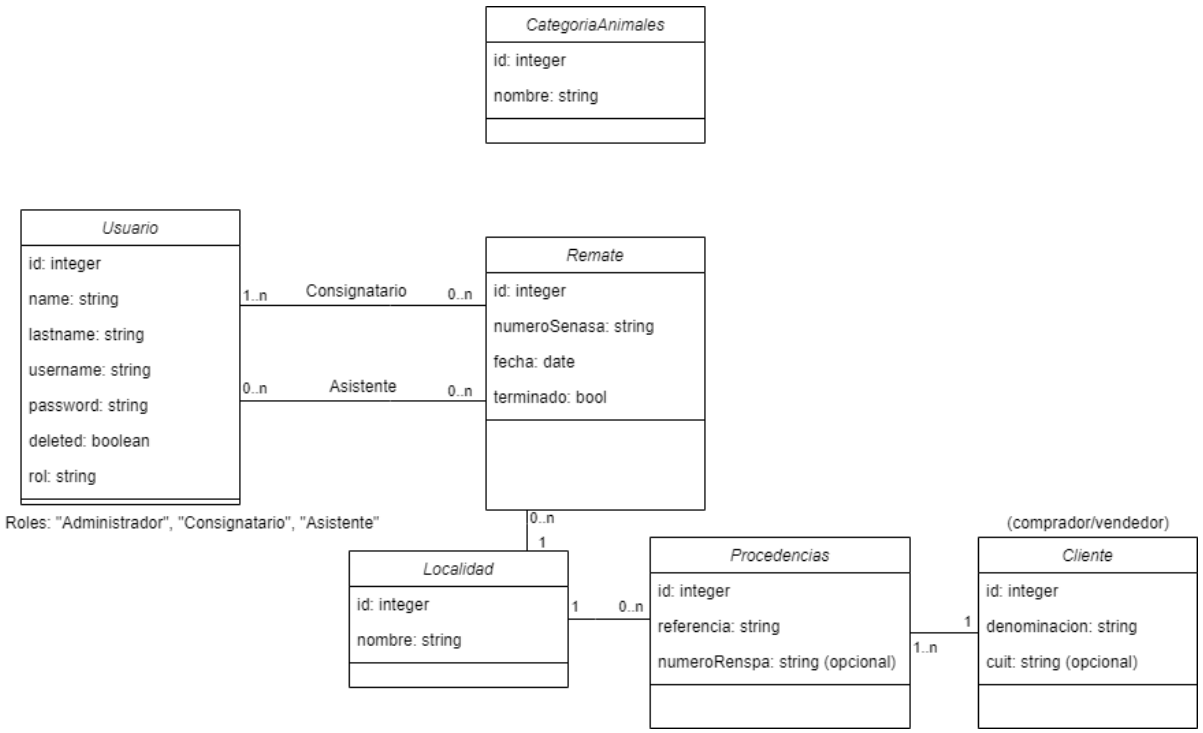


Figura 94: Diagrama de clases historia 8

E.2.9.3 Casos de prueba

Tests con insomnia:

CASO	RESULTADO ESPERADO	RESULTADO OBTENIDO
Crear un remate con un usuario administrador	HTTP 200 cuyo body contiene el remate creado con una lista vacía de participantes	Éxito
Crear un remate con un usuario consignatario	HTTP 200 cuyo body contiene el remate creado con una lista que contiene al creador como participante.	Éxito
Crear un remate con un usuario asistente	HTTP 403	Éxito
Agregar un usuario administrador al remate	HTTP 200, y el body contiene el remate pero sin modificaciones Ya que un administrador tiene acceso a cualquier remate.	Éxito
Agregar un usuario consignatario al remate	HTTP 200, y el body contiene el remate pero con la lista de participantes modificada con el nuevo participante.	Éxito

El usuario que realiza la petición se borra a sí mismo del remate	HTTP 403.	Éxito
Quitar un usuario de la lista de remates.	HTTP 200, y el body contiene el remate, cuya lista de participantes ahora no posee al usuario eliminado.	Éxito
Quitar a todos los usuarios participantes de un remate.	HTTP 200, y el body contiene el remate, cuya lista de participantes ahora está vacía.	Éxito
Quitar de un remate un usuario que no participa en el mismo.	HTTP 403	Éxito
Quitar de un remate un usuario que no existe	HTTP 404	Éxito
Agregar a un remate un usuario que no existe	HTTP 404	Éxito

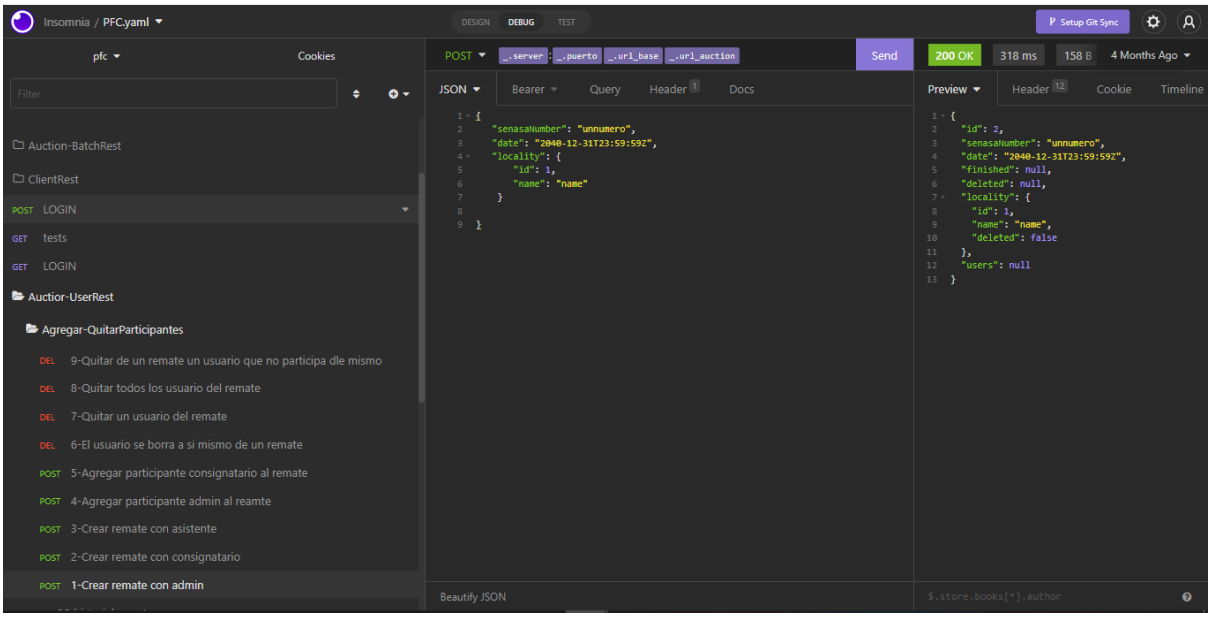


Figura 95: ejemplo de los tests

E.2.9.4 Criterio de aceptación

Cuando el usuario asignado a un remate ingresa al sistema, debe tener acceso para gestionar el remate al que fue asignado.

No vimos necesario en este caso mostrar el criterio de aceptación con imágenes como hemos hecho anteriormente por el hecho de que simplemente se exhibirían pantallas previamente mostradas sin agregar demasiado.

E.2.9.5 Código

```
@Service
public class AuctionServiceImpl implements AuctionService {
  ...
@Override
public Auction removeUserFromAuction(Integer auctionId, Integer userId) throws
AuctionNotFoundException, UserNotFoundException, HttpForbidenException,
HttpUnauthorizedException {
   Optional<Auction> auctionOpt = auctionDAO.findById(auctionId);
   if(auctionOpt.isPresent()){
       Auction auction = auctionOpt.get();
       if(auction.getDeleted() != null && auction.getDeleted()){
           throw new AuctionNotFoundException("El remate con id: " + auctionId + " no
existe.");
       }
       List<User> autionUserList = auction.getUsers();
       Optional<User> userOptional = autionUserList.stream().filter(u ->
u.getId().equals(userService.getCurrentUser().getId())).findFirst();
       if(userOptional.isEmpty() &&
!userService.getCurrentUserAuthorities().toArray()[0].toString().equals("Administrador")
){
           throw new HttpUnauthorizedException("Usted no es participante del remate que
quiere modificar");
       }
       Optional<User> userOpt = auction.getUsers().stream().filter(u ->
u.getId().equals(userId)).findFirst();
       if(userOpt.isPresent()){
           if(!userOpt.get().getId().equals(userService.getCurrentUser().getId())){
               auction.getUsers().remove(userOpt.get());
           }
...
}
```

```
@Service
public class AuctionServiceImpl implements AuctionService {
  ...
@Override
public Auction removeUserFromAuction(Integer auctionId, Integer userId) throws
AuctionNotFoundException, UserNotFoundException, HttpForbidenException,
HttpUnauthorizedException {
   Optional<Auction> auctionOpt = auctionDAO.findById(auctionId);
   if(auctionOpt.isPresent()){
       Auction auction = auctionOpt.get();
       if(auction.getDeleted() != null && auction.getDeleted()){
           throw new AuctionNotFoundException("El remate con id: " + auctionId + " no
existe.");
       }
       List<User> autionUserList = auction.getUsers();
       Optional<User> userOptional = autionUserList.stream().filter(u ->
u.getId().equals(userService.getCurrentUser().getId())).findFirst();
       if(userOptional.isEmpty() &&
!userService.getCurrentUserAuthorities().toArray()[0].toString().equals("Administrador")
){
           throw new HttpUnauthorizedException("Usted no es participante del remate que
```

```
quiere modificar");
        }
        Optional<User> userOpt = auction.getUsers().stream().filter(u ->
u.getId().equals(userId)).findFirst();
        if(userOpt.isPresent()){
            if(!userOpt.get().getId().equals(userService.getCurrentUser().getId())){
                auction.getUsers().remove(userOpt.get());
                return auctionDAO.save(auction);
            }
            throw new HttpForbidenException("No puede eliminarse a sí mismo");
        }
        throw new UserNotFoundException("El usuario con id: " + userId + " no participa
en este remate.");
    }
    throw new AuctionNotFoundException("El remate con id: " + auctionId + " no existe.");
}
...
}
```

E.2.10 Historia de Usuario 9 - Cargar lotes de venta a un remate

E.2.10.1 Mockup

Dentro de esta historia se desarrolló la pantalla de “Agregar/Editar lotes”, la cual tiene semejanzas con la pantalla de creación/edición de clientes, contando también con dos partes bien definidas, siendo la primera para cargar los datos generales del lote (ver referencia de lote de venta), como el vendedor, la procedencia de donde vienen esos animales, el número del corral donde se encierran y, opcionalmente, el número de DTe; y la segunda parte para cargar la información de cada grupo de animales en particular (ver referencia animales en pista), como la categoría de dichos animales y su cantidad.

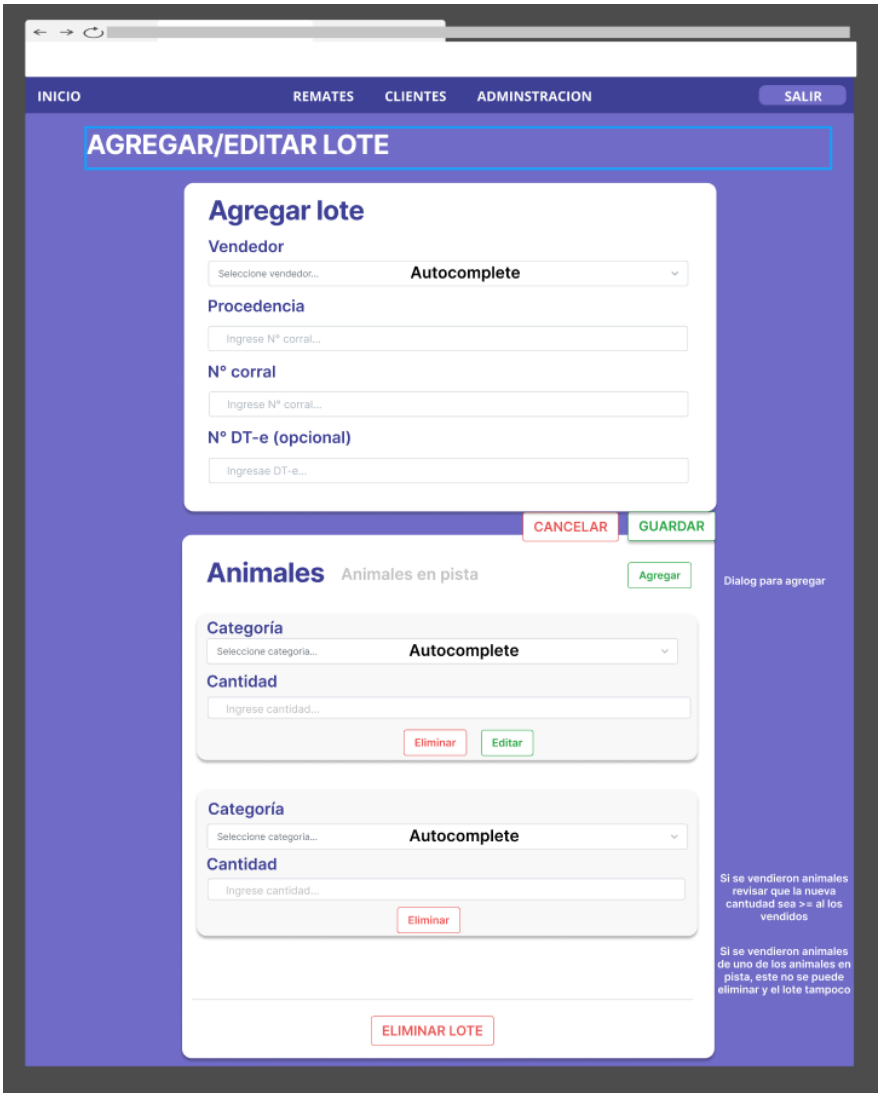


Figura 96: Mockup pantalla agregar/editar lote

También dentro de esta historia se continuó con el progreso en la pantalla “Ver/Editar remate”, a la que se le agregaron los tres listados con la información de los lotes para venta (Para venta, No vendidos y Vendidos), en la primera, con los botones “Vender“, “No vendido“ y “Editar“, en la segunda, “Vender“ y “Editar“, en la tercera solo “Editar“.

E.2.10.2 Evolución del diagrama de clases

En esta etapa se agregan dos de las clases más importantes y centrales dentro del funcionamiento del sistema: los Lotes para Venta y los Animales en Pista, los cuales además dejan a todo el diagrama interconectado.

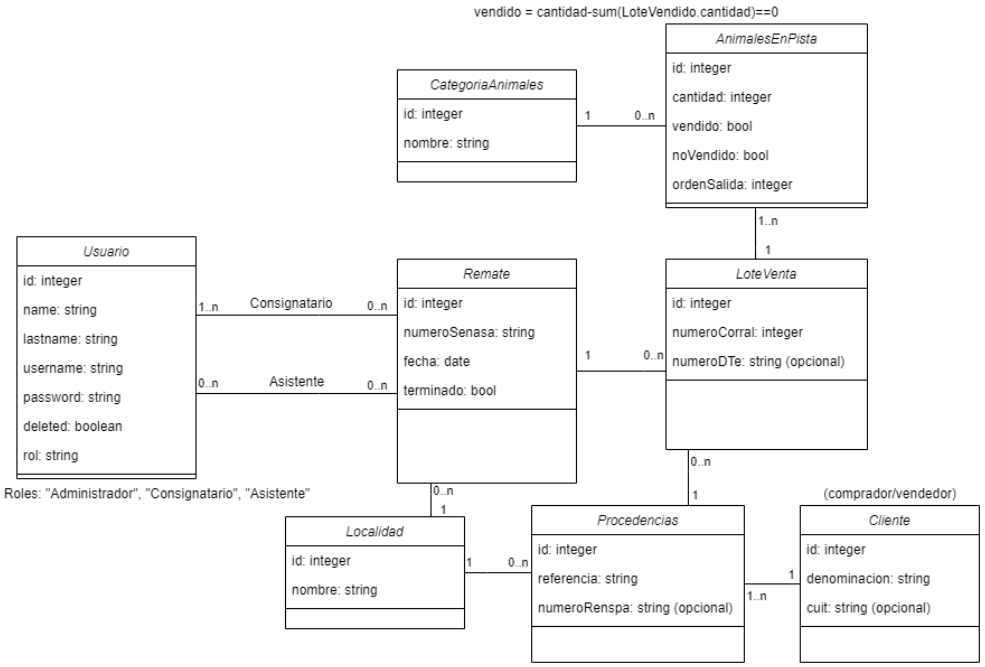


Figura 97: Diagrama de clases historia 9

E.2.10.3 Criterio de aceptación

Cuando acceda a la información del remate, el usuario deberá poder observar el nuevo lote agregado en la lista de lotes de venta.

DBs al inicio:

```
SELECT * FROM batch LIMIT 100;
```

Input To Search Data — Cost: 12601ms — 1 — Total 2

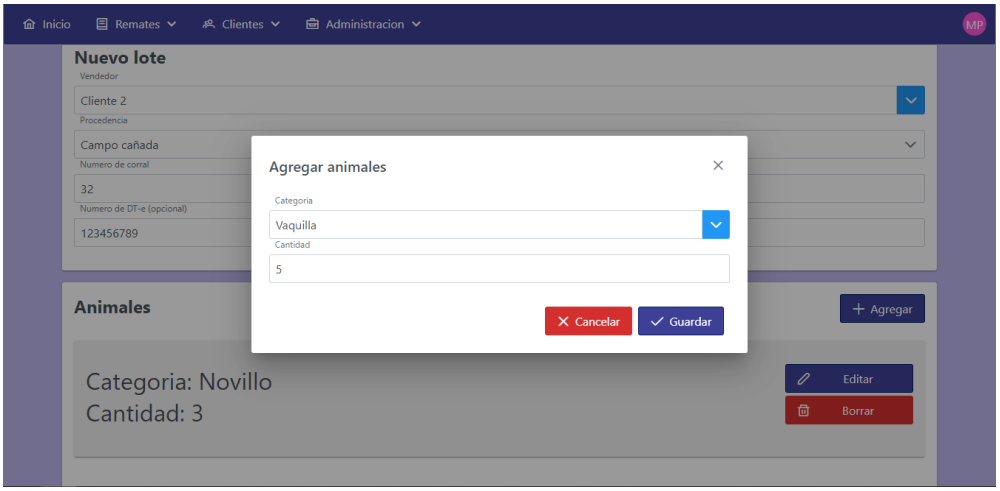
	* id int	* corral_number int	deleted bit(1)	dte_number varchar(255)	auction_id int	provenance_id int
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	false	12345	1	24
2	2	8	(NULL)	222-222	1	16

```
SELECT * FROM animals_on_ground LIMIT 100;
```

Input To Search Data — Cost: 10599ms — 1 — Total 7

	* id int	* amount int	deleted bit(1)	* not_sold bit(1)	* sold bit(1)	starting_order int	* category_id int	batch_id int
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	9	false	false	false	4	5	1
2	2	8	true	false	false	17	4	1
3	3	2	(NULL)	false	false	(NULL)	6	2
4	4	6	(NULL)	false	false	(NULL)	4	2
5	5	15	(NULL)	false	false	(NULL)	6	1
6	6	4	(NULL)	false	false	(NULL)	4	1
7	7	5	true	false	false	(NULL)	5	1

Se crea un lote con 2 animales en pista:



Resultado:


```
SELECT * FROM batch LIMIT 100;
```

Input To Search Data | Cost: 25ms | 1 | Total 3

id int	corral_number int	deleted bit(1)	dte_number varchar(255)	auction_id int	provenance_id int
Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	false	12345	1	24
2	2	8	(NULL)	222-222	1	16
3	3	32	(NULL)	123456789	1	24

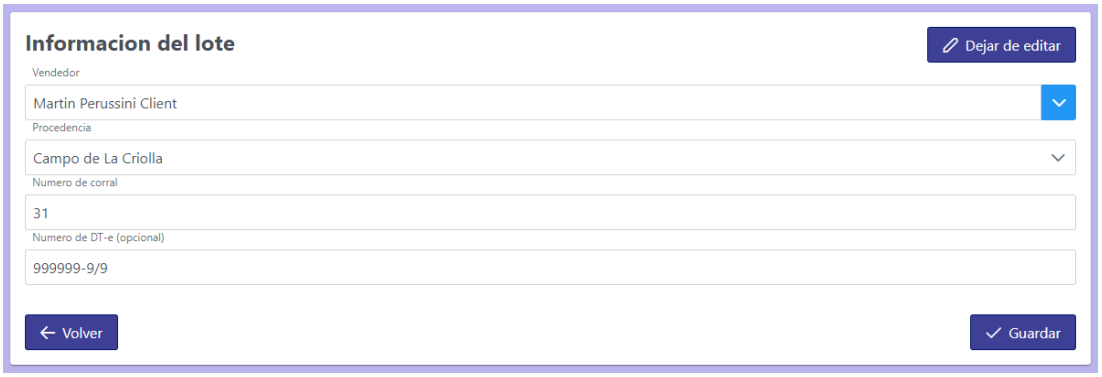
```
SELECT * FROM animals_on_ground LIMIT 100;
```

Input To Search Data — Cost: 2ms — 1 — Total 9

	id int	amount int	deleted bit(1)	not_sold bit(1)	sold bit(1)	starting_order int	category_id int	batch_id int
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	9	false	false	false	4	5	1
2	2	8	true	false	false	17	4	1
3	3	2	(NULL)	false	false	(NULL)	6	2
4	4	6	(NULL)	false	false	(NULL)	4	2
5	5	15	(NULL)	false	false	(NULL)	6	1
6	6	4	(NULL)	false	false	(NULL)	4	1
7	7	5	true	false	false	(NULL)	5	1
8	8	3	(NULL)	false	false	(NULL)	6	3
9	9	5	(NULL)	false	false	(NULL)	4	3

En caso de edición de un lote de venta, esta modificación deberá verse reflejada en la lista de los lotes a vender.

Se modifica datos del lote:

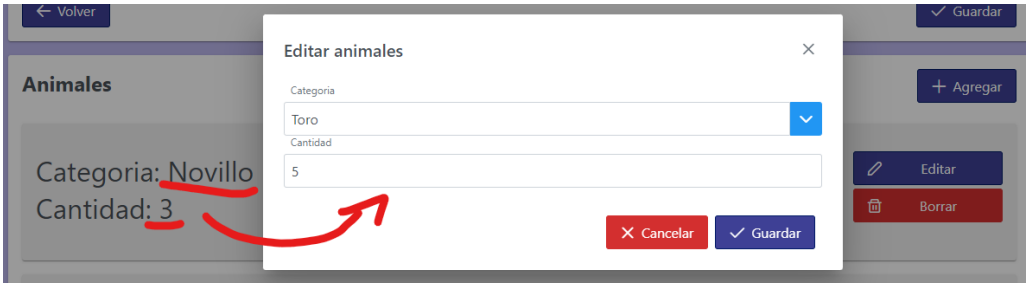


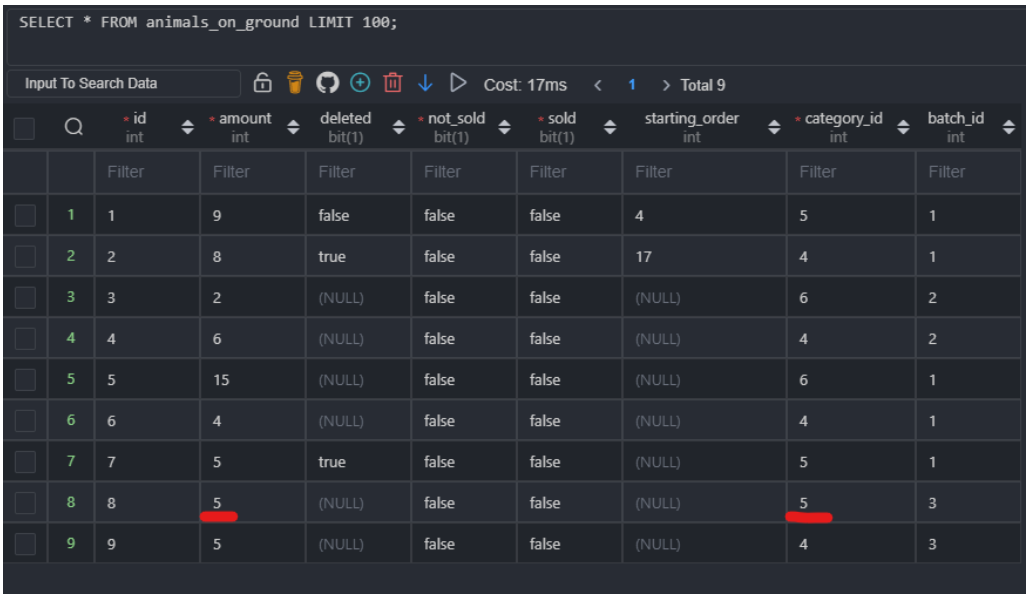

```
SELECT * FROM batch LIMIT 100;
```

Input To Search Data — Cost: 22ms — 1 — Total 3

	id int	corral_number int	deleted bit(1)	dte_number varchar(255)	auction_id int	provenance_id int
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1	false	12345	1	24
2	2	8	(NULL)	222-222	1	16
3	3	31	(NULL)	999999-9/9	1	15

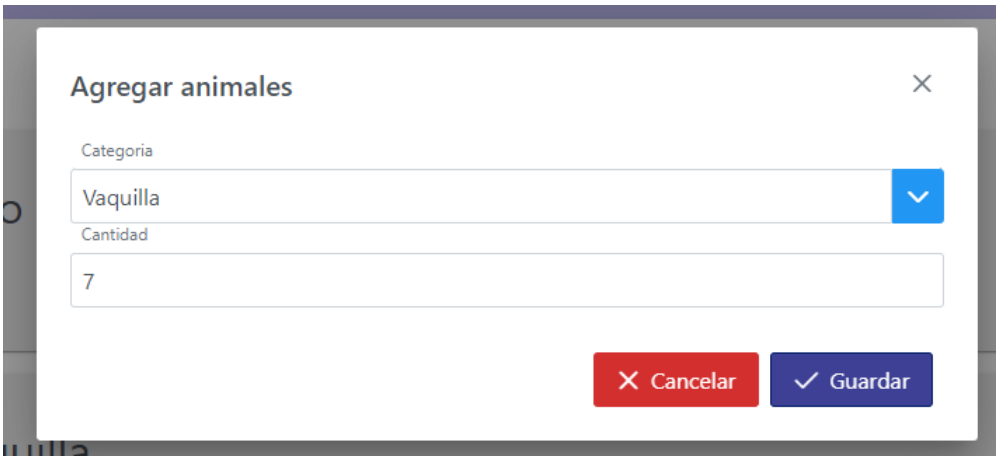
Se modifica el Animales En Pista de Novillos previamente creado:

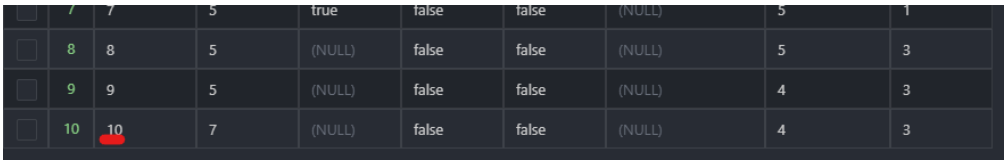




	id int	amount int	deleted bit(1)	not_sold bit(1)	sold bit(1)	starting_order int	category_id int	batch_id int
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	9	false	false	false	4	5	1
2	2	8	true	false	false	17	4	1
3	3	2	(NULL)	false	false	(NULL)	6	2
4	4	6	(NULL)	false	false	(NULL)	4	2
5	5	15	(NULL)	false	false	(NULL)	6	1
6	6	4	(NULL)	false	false	(NULL)	4	1
7	7	5	true	false	false	(NULL)	5	1
8	8	5	(NULL)	false	false	(NULL)	5	3
9	9	5	(NULL)	false	false	(NULL)	4	3

Se agrega un nuevo Animales En Pista:

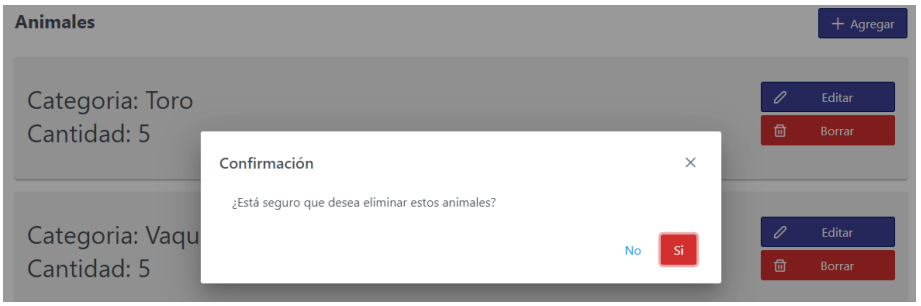


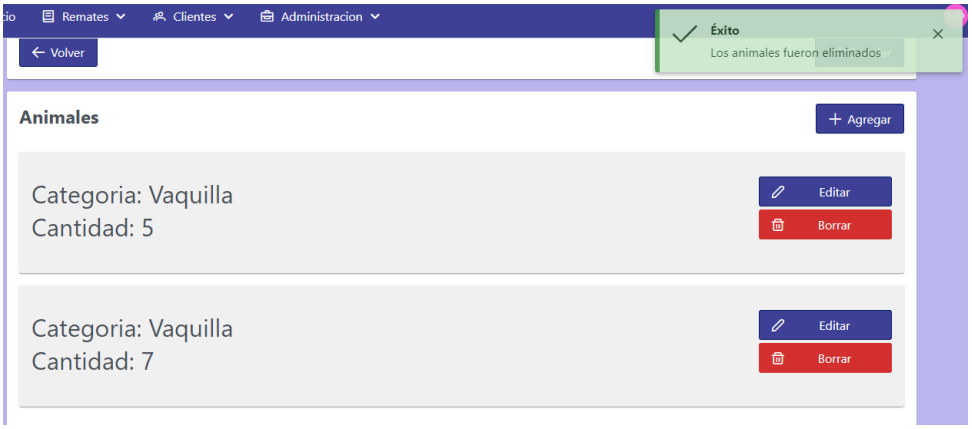


7	7	5	true	false	false	(NULL)	5	1
8	8	5	(NULL)	false	false	(NULL)	5	3
9	9	5	(NULL)	false	false	(NULL)	4	3
10	10	7	(NULL)	false	false	(NULL)	4	3

En caso de eliminación, deberá desaparecer el registro (junto a sus relaciones, si es necesario), de la respectiva tabla en la base de datos. O bien, que el lote esté marcado como "deshabilitado" en la base de datos (decisión de implementación a futuro).

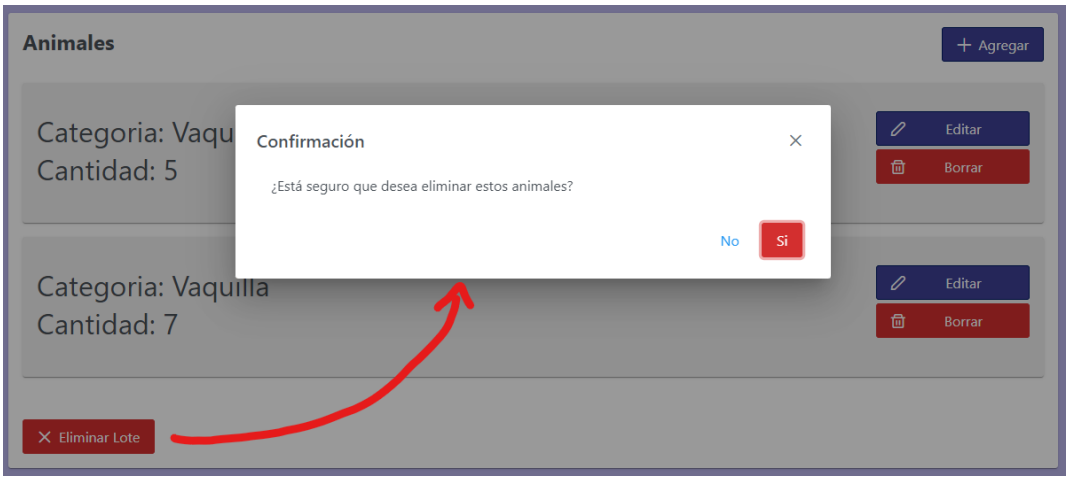
Se elimina el Animales En Pista de Toros:

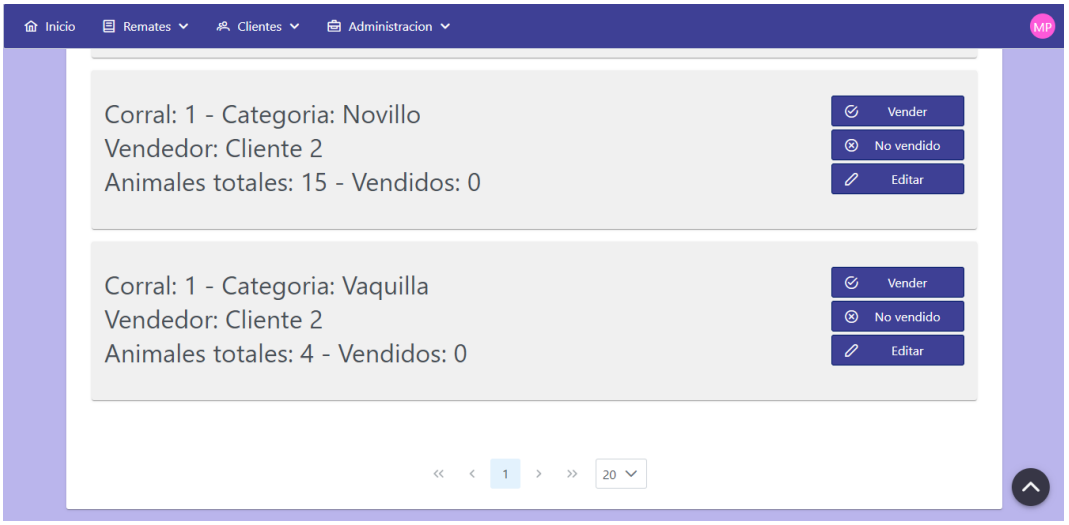


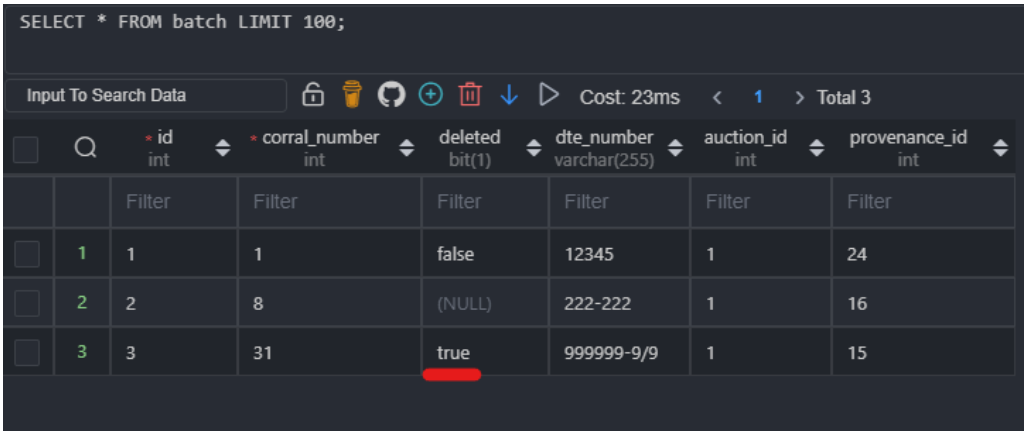


	id int	amount int	deleted bit(1)	not_sold bit(1)	sold bit(1)	starting_order int	category_id int	batch_id int
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	9	false	false	false	4	5	1
2	2	8	true	false	false	17	4	1
3	3	2	(NULL)	false	false	(NULL)	6	2
4	4	6	(NULL)	false	false	(NULL)	4	2
5	5	15	(NULL)	false	false	(NULL)	6	1
6	6	4	(NULL)	false	false	(NULL)	4	1
7	7	5	true	false	false	(NULL)	5	1
8	8	5	true	false	false	(NULL)	5	3
9	9	5	(NULL)	false	false	(NULL)	4	3
10	10	7	(NULL)	false	false	(NULL)	4	3

Y por último, se elimina el Lote:







E.2.10.4 Código

```
@Service
public class BatchServiceImpl implements BatchService {
  ...
@Override
public Batch saveBatch(Batch newBatch, Integer auctionId) throws
AuctionNotFoundException, IllegalArgumentException, HttpForbidenException {
   Auction auction = getAuctionNotDeletedById(auctionId);
   if(newBatch.getAnimalsOnGround() == null || newBatch.getAnimalsOnGround().isEmpty()){
       throw new IllegalArgumentException("El Lote debe tener al menos un conjunto de
AnimalesEnPista.");
   }
   for(AnimalsOnGround a: newBatch.getAnimalsOnGround()){
       if(a.getAmount()==null || (a.getAmount()!=null && a.getAmount()<=0)){
           throw new IllegalArgumentException("Hay un conjunto de animales con cantidad
menor a 1");
       }
   }
   newBatch.setAuction(auction);
   return batchDAO.save(newBatch);
}
...
}
```

```
@Service
public class BatchServiceImpl implements BatchService {
  ...
@Override
public Batch deleteBatchById(Integer batchId) throws HttpForbidenException,
AuctionNotFoundException, BatchNotFoundException {
   Batch batch = this.findById(batchId);
   if(batch.getDeleted() != null && batch.getDeleted()){
       throw new BatchNotFoundException("El Lote con id: "+ batchId + " no existe");
   }
   Auction auction = batch.getAuction();
   if(auction.getDeleted() != null && auction.getDeleted()){
       throw new AuctionNotFoundException("El Lote pertenece a un Remate inexistente.");
   }
   if(auction.getFinished() != null && auction.getFinished()){
```

```
        throw new HttpForbidenException("No se pueden modificar Lotes de un Remate que ya
se ha realizado");
    }
    boolean canBeDeleted = true;
    List<AnimalsOnGround> animalsOnGroundList = batch.getAnimalsOnGround();
    if(animalsOnGroundList != null && !animalsOnGroundList.isEmpty()) {
        for (AnimalsOnGround animalsOnGround : animalsOnGroundList) {
            List<SoldBatch> soldBatches =
soldBatchService.findSoldBatchesNotDeletedByAnimalsOnGroundId(animalsOnGround.getId());
            if(soldBatches != null && soldBatches.size() > 0){
                canBeDeleted = false;
            }
        }
    }
    if(!canBeDeleted){
        throw new HttpForbidenException("No puede eliminarse un Lote de animales que ya
tiene animales vendidos.");
    }
    batch.setDeleted(true);
    return batchDAO.save(batch);
}
...
}
```

```
@Service
public class BatchServiceImpl implements BatchService {
  ...
@Override
public List<AnimalsOnGround> addAnimalsOnGround(Integer batchId, AnimalsOnGround
animalsOnGround) throws BatchNotFoundException, IllegalArgumentException,
AuctionNotFoundException, HttpForbidenException {
    Batch batch = findById(batchId);
    if(batch.getDeleted() != null && batch.getDeleted()){
        throw new BatchNotFoundException("El Lote con id: " + batchId + " no existe.");
    }
    if(batch.getAuction().getDeleted() != null && batch.getAuction().getDeleted()){
        throw new AuctionNotFoundException("El Remate fue eliminado.");
    }
    if(batch.getAuction().getFinished() != null && batch.getAuction().getFinished()){
        throw new HttpForbidenException("No se pueden editar Lotes de un Remate
finalizado.");
    }
    if(animalsOnGround.getAmount() <= 0){
        throw new IllegalArgumentException("La cantidad de animales a agregar debe ser
mayor a 0");
    }
    batch.getAnimalsOnGround().add(animalsOnGround);
    logger.debug(batch.getAnimalsOnGround().toString());
    logger.debug(animalsOnGround.toString());
    Batch updatedBatch = batchDAO.save(batch);
    return updatedBatch.getAnimalsOnGround();
}
...
}
```

```
@Service
public class BatchServiceImpl implements BatchService {
  ...
@Override
public AnimalsOnGround deleteAnimalsOnGroundById(Integer animalsId) throws
AnimalsOnGroundNotFound, HttpForbidenException, AuctionNotFoundException,
BatchNotFoundException {
   Batch batchOwn = getBatchByAnimalsOnGroundId(animalsId);
   if(batchOwn == null){
       throw new AnimalsOnGroundNotFound("El conjunto de AnimalesEnPista con id " +
animalsId + " no existe");
   }
   if(batchOwn.getDeleted() != null && batchOwn.getDeleted()){
       throw new BatchNotFoundException("El Lote al que pertenece estos animales no
existe");
   }
   if(batchOwn.getAuction().getDeleted() != null && batchOwn.getAuction().getDeleted()){
       throw new AuctionNotFoundException("El Remate al que pertenecen estos animales no
existe");
   }
   if(batchOwn.getAuction().getFinished() != null &&
batchOwn.getAuction().getFinished()){
       throw new HttpForbidenException("No puede modificarse un Remate que ya se ha
realizado");
   }
   if(batchOwn.getAnimalsOnGround().size() == 1){
       throw new HttpForbidenException("El Lote no puede quedar sin animales para
vender.");
   }
   return animalsOnGroundService.deleteById(animalsId);
}
...
}
```

```
@Service
public class BatchServiceImpl implements BatchService {
  ...
@Override
public Batch updateBatchById(Integer batchId, BatchDTO batchDTO) throws
IllegalArgumentException, BatchNotFoundException, BadHttpRequest,
AuctionNotFoundException, HttpForbidenException {
   if(batchDTO.getId() != null && !batchDTO.getId().equals(batchId)){
       throw new BadHttpRequest("El id del path no coincide con el id del body del
request");
   }
   Batch batch = findById(batchId);
   if(batch.getAuction().getDeleted() != null && batch.getAuction().getDeleted()){
       throw new AuctionNotFoundException("El Lote pertenece a un remate que no
existe");
   }
   if(batch.getAuction().getFinished() != null && batch.getAuction().getFinished()){
       throw new HttpForbidenException("No se puede editar un Lote de un Remate que ya
se ha realizado.");
   }
```

```
    if(batch.getDeleted() != null && batch.getDeleted()){
        throw new BatchNotFoundException("El Lote con id: " + batchId + " no existe.");
    }
    batchMapper.updateBatchFromDto(batchDTO, batch);
    return batchDAO.save(batch);
}
...
}
```

E.2.11 Historia de Usuario 11 - Cargar datos de lotes vendidos

E.2.11.1 Mockups

En esta historia se llevaron a cabo múltiples pantallas y hubo cambios importantes a la hora de la implementación respecto a lo pensado en un principio, por lo que iniciaremos comentando los mismos.

En un principio la idea era tener dos pantallas: una de lotes vendidos y otra de lotes finales, siendo la primera presentada al usuario mientras el remate en cuestión esté en curso, y la segunda una vez que el mismo esté terminado.

Pantalla Lotes Vendidos:

Pantalla que contará con un listado de los lotes que fueron vendidos (que pueden ser distintos a los de venta, ya que de un solo lote para venta pueden salir múltiples lotes vendidos).

Cada lote se mostrará mediante una card con la siguiente información:

- Comprador
- Vendedor
- Cantidad
- Categoría
- Peso (si corresponde)
- Precio

Además, botones “Boleta“ (para obtener la boleta de venta), “Pesar“ (para acceder a dicha pantalla, solo se muestra si el lote está marcado como que debe pesarse), “Editar” (Abre la pantalla Vender, pero con los datos del lote cargados), y “Eliminar“.

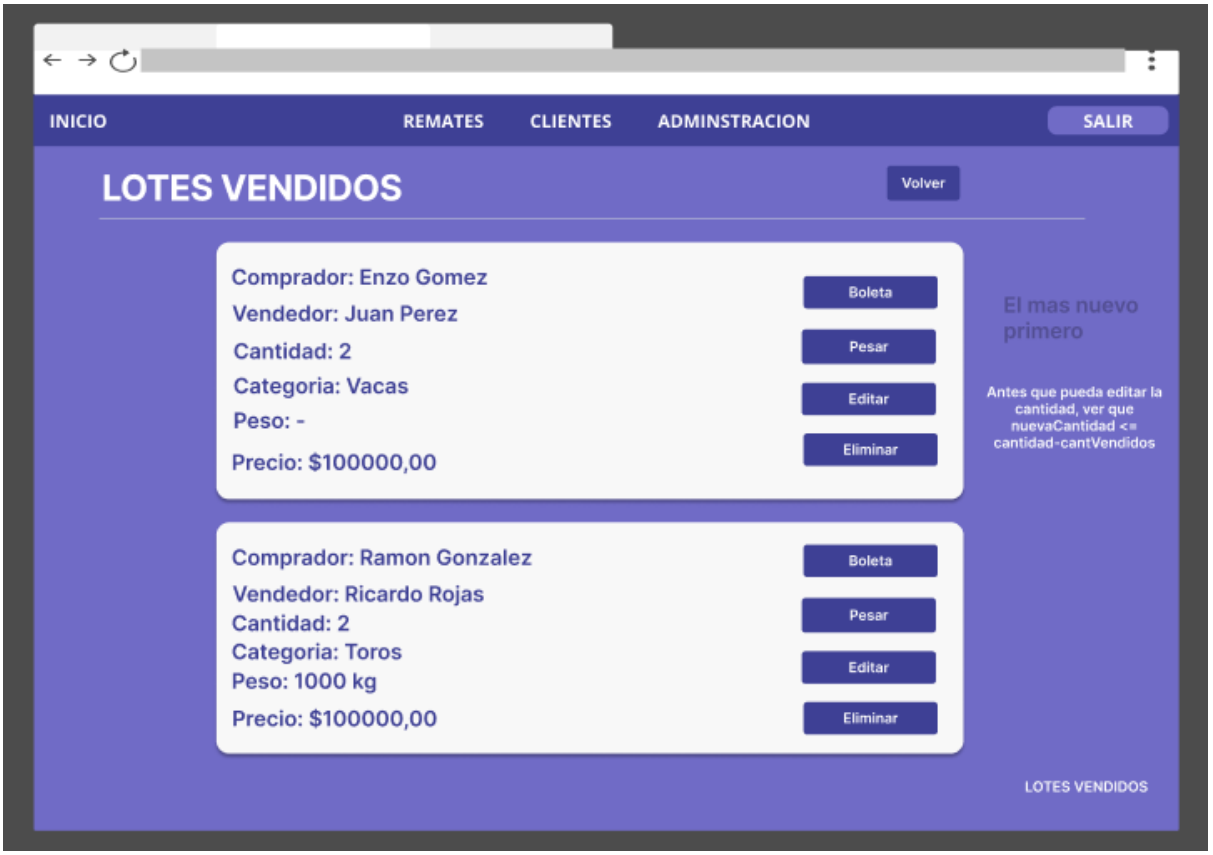


Figura 98: Mockup pantalla Lotes Vendidos (idea anterior)

Pantalla Lotes Finales:

Pantalla que contará con 2 listados de los lotes, Vendidos y No vendidos, a los que se accederá con pestañas (similar a lo que hace en la pantalla de remate), donde se mostrará para cada lote la misma información que en la pantalla de Lotes Vendidos junto al DTe, para el cual habrá un botón para cargarlo, que abrirá el diálogo de la pantalla Ingresar DTe (historia 12). También, existirá otro botón para obtener la boleta, al igual que en la pantalla antes mencionada.

Además habrá opciones para ingresar a las pantallas “Lotes vendidos“, que lleva hacia la pantalla del mismo nombre, “Lotes para venta“, que lleva a la pantalla de Ver/Editar Remate pero con las opciones para realizar cambios deshabilitadas, y “Resumen“, para la generación del mismo.

Esta pantalla se habilita una vez que el remate se finaliza desde la opción en la pantalla de Remate.

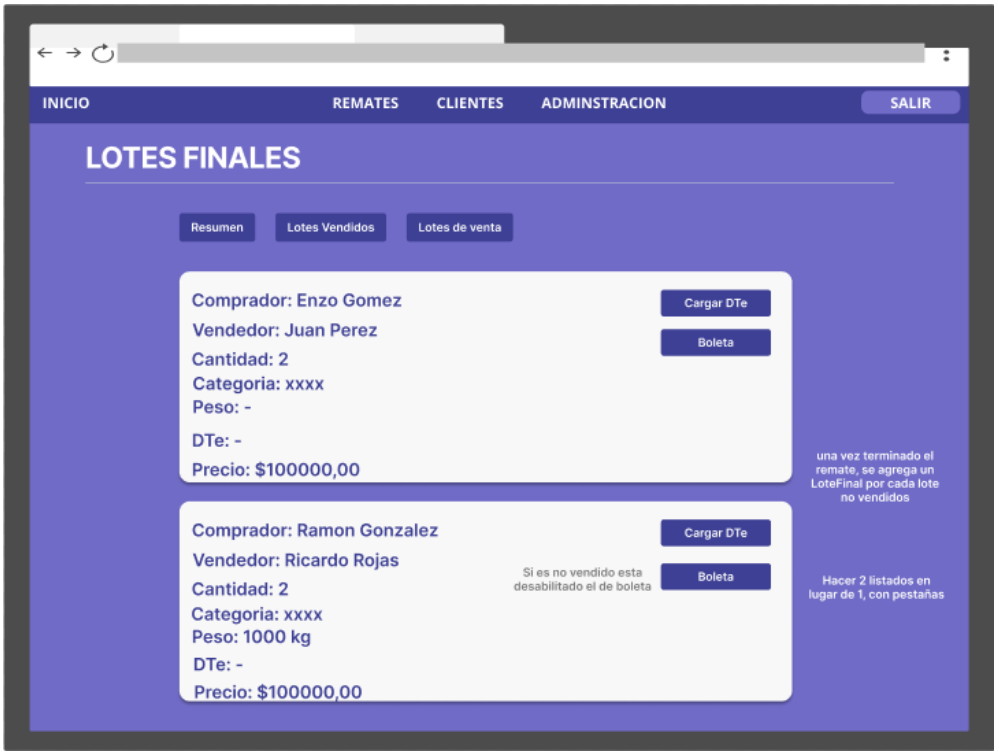


Figura 99: Mockup pantalla Lotes Finales (idea original)

Cambios en las pantallas Lotes Vendidos y Lotes Finales:

Estas pantallas pasaron a ser una sola (llamada Lotes Finales), ya que nos dimos cuenta de su parecido. En principio planteamos 2 pantallas, siendo la primera accesible mientras el remate está en curso y la segunda una vez terminado, pero al ir obteniendo mayor experiencia en el uso de React, nos pareció que podían fusionarse y simplemente mostrar o no ciertas opciones según si el remate estaba en curso o terminado.

Figura 100: Mockup pantalla Lotes Finales

Dentro de esta historia también se desarrollaron dos diálogos para la carga de datos respectivos a una venta. Por un lado, en el diálogo de vender animales, el cual es accedido mediante el botón “Vender” de un lote de venta (en la pantalla “Ver/Editar remate”), se carga la información de la venta, tal como el comprador (autocomplete), el precio, la cantidad de animales y si estos animales deben ser pesados o no (luego, en la implementación final se agregó el campo “plazo”, que no había sido contemplado en el plan del proyecto); y por otro lado el diálogo para “pesar” a los animales, el cual es accedido desde el botón correspondiente en la pantalla de Lotes Finales, el cual solo cuenta con un campo para cargar el peso arrojado por la balanza.

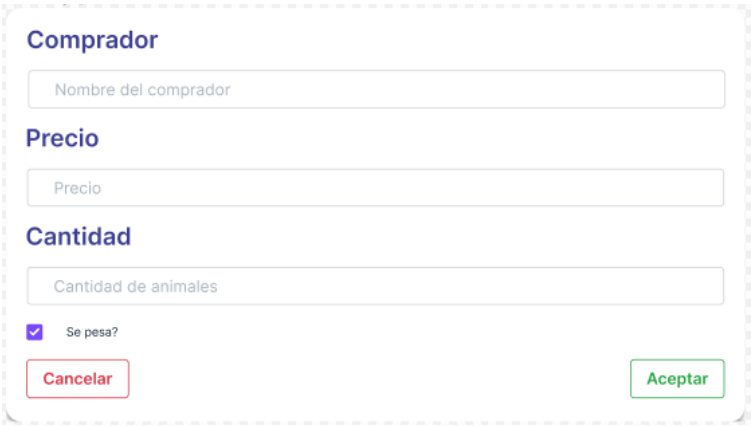


Figura 101: Mockup diálogo vender animales

Figura 102: Mockup diálogo pesar animales

E.2.11.2 Evolución del diagrama de clases

En esta historia se completa el diagrama de clases, agregando como en el caso anterior dos de las clases más importantes del sistema, en este caso los Lotes Vendidos y los Lotes No Vendidos:

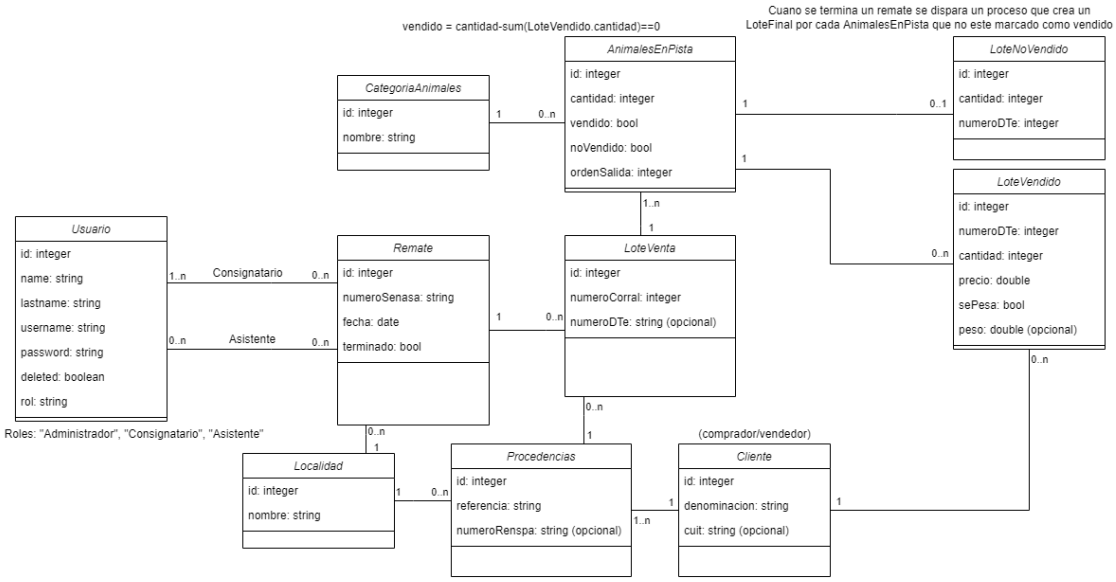


Figura 103: Diagrama de clases historia 11 (finalizado)

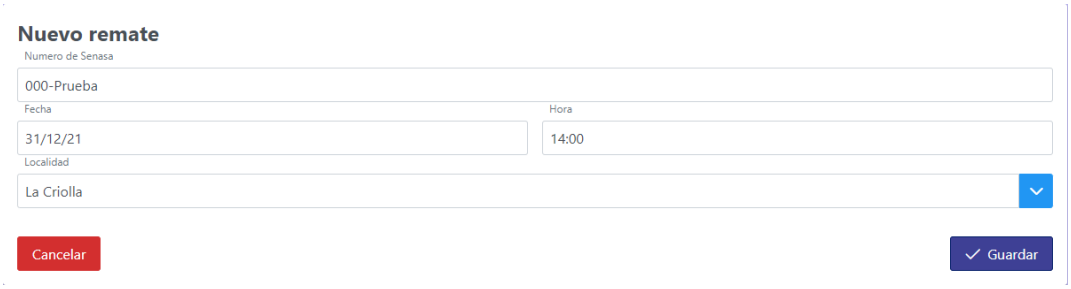
E.2.11.3 Criterio de aceptación

En caso de vender un lote, el mismo deberá aparecer en el listado de lotes vendidos, con los datos correctos respectivos a la venta.

En caso de marcar como “no vendido” a un lote, dicho registro deberá aparecer como tal en la base de datos.

Resultados:

Se crea un nuevo Remate:



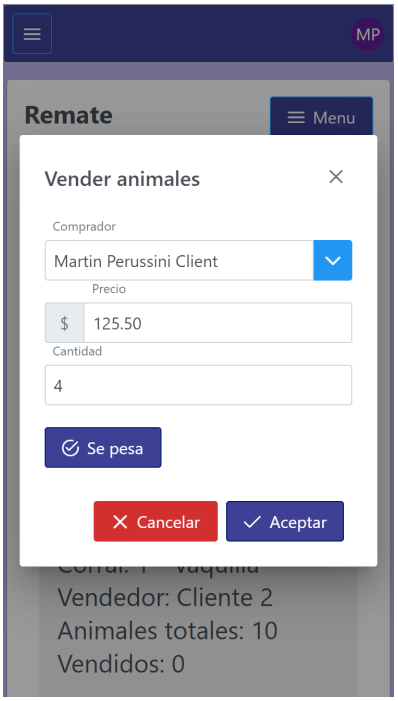
Se agrega un Lote:


MP
Nuevo lote
Vendedor
Cliente 2
Procedencia
Campito
Numero de corral
1
Numero de DT-e (opcional)
123
Animales
+ Agregar
Categoria: Novillo
Cantidad: 5
Editar
Borrar
Categoria: Vaquilla
Cantidad: 10
Editar
Borrar
Cancelar
Guardar

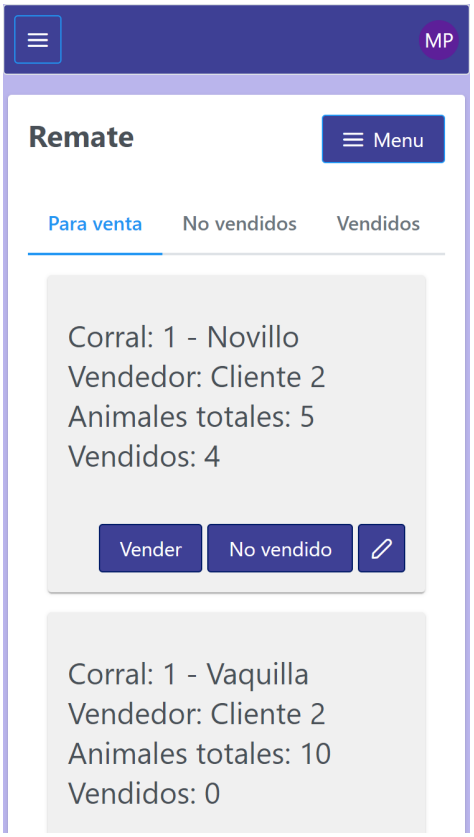
Resultado:


MP
Remate
Menu
Para venta
No vendidos
Vendidos
Corral: 1 - Novillo
Vendedor: Cliente 2
Animales totales: 5
Vendidos: 0
Vender
No vendido
Corral: 1 - Vaquilla
Vendedor: Cliente 2
Animales totales: 10
Vendidos: 0
Vender
No vendido
1
20

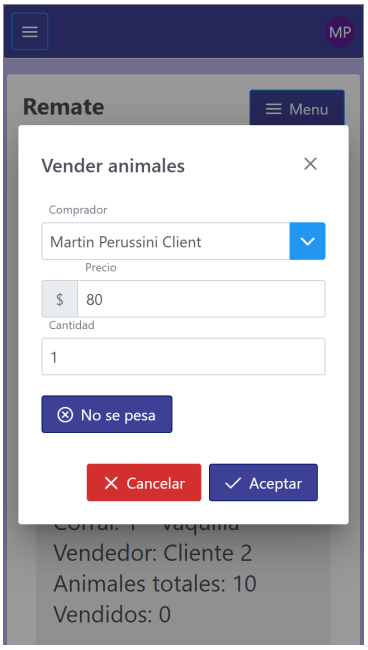
Se vende parte de los Novillos:



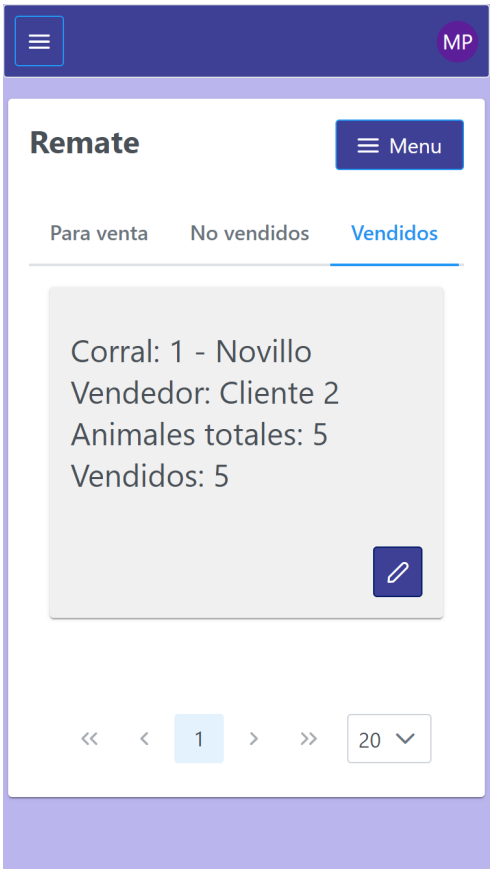
Resultado:



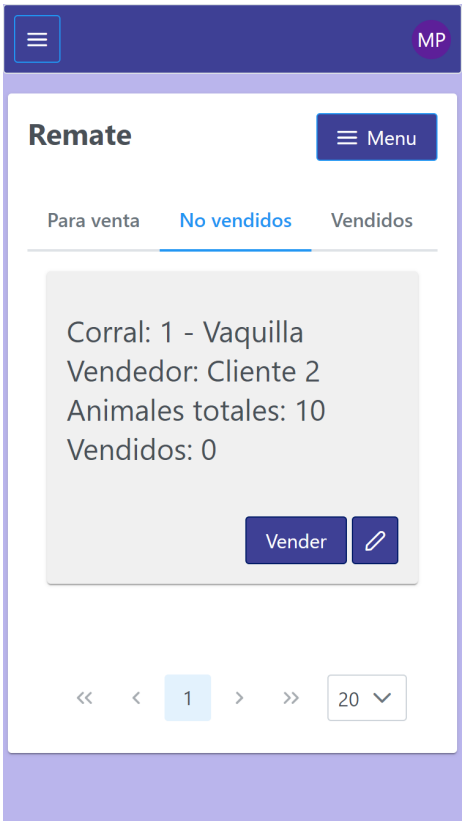
Se vende el restante:



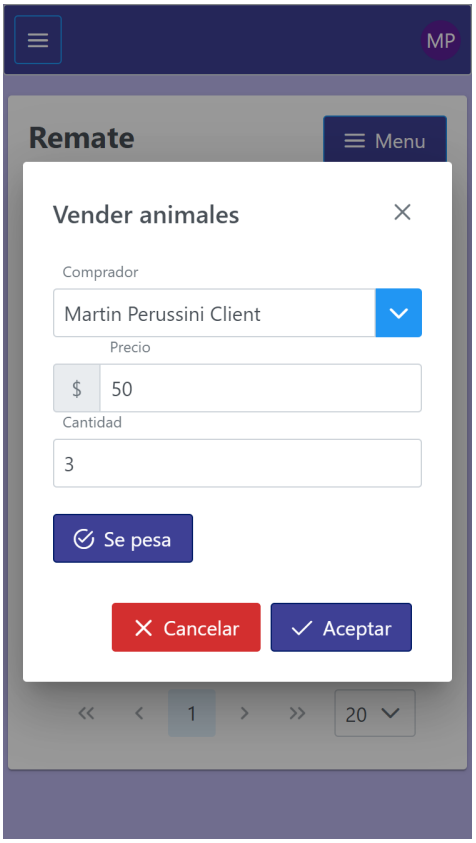
Y el lote (Animales En Pista) pasa a figurar como vendido:



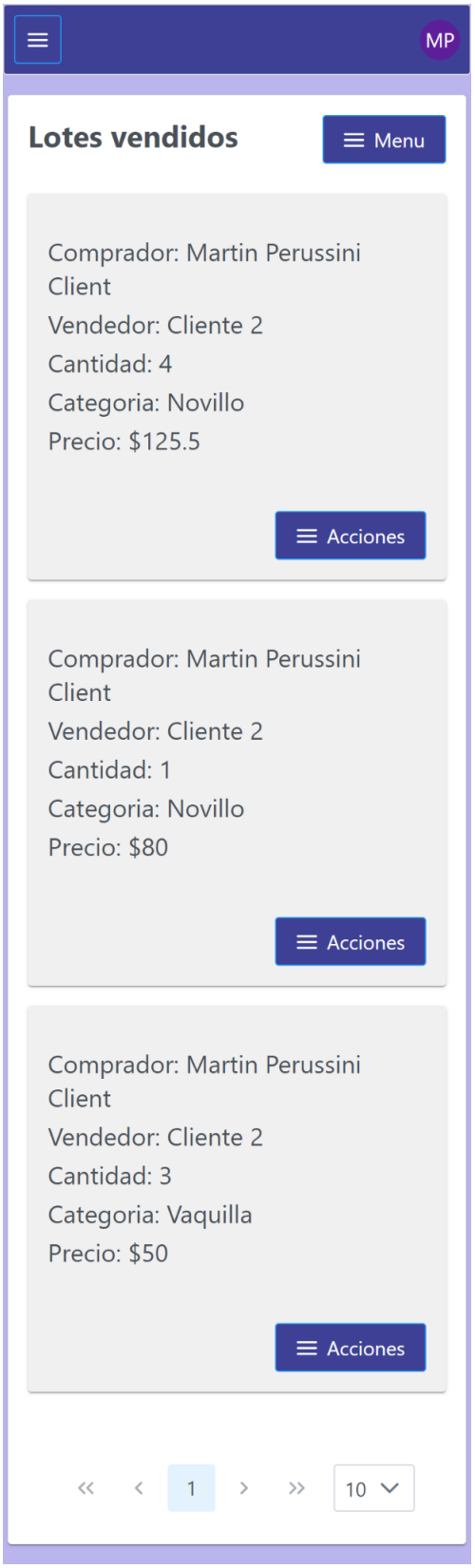
Se marcan las Vaquillas como no vendidas mediante el botón para ello y pasa al listado correspondiente:



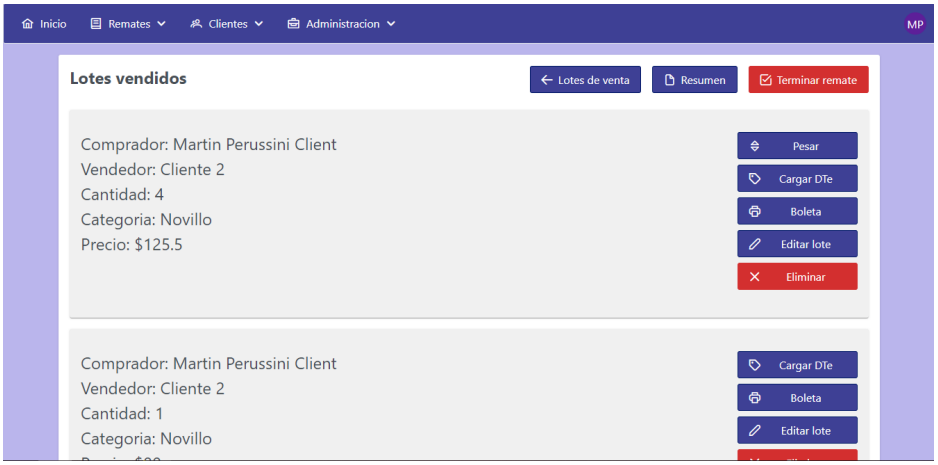
Se vende parte de esas Vaquillas (es el caso donde terminó el remate y se venden animales después):



Ahora, pasando a la pantalla de lotes finales, este fue el resultado de las operaciones realizadas:

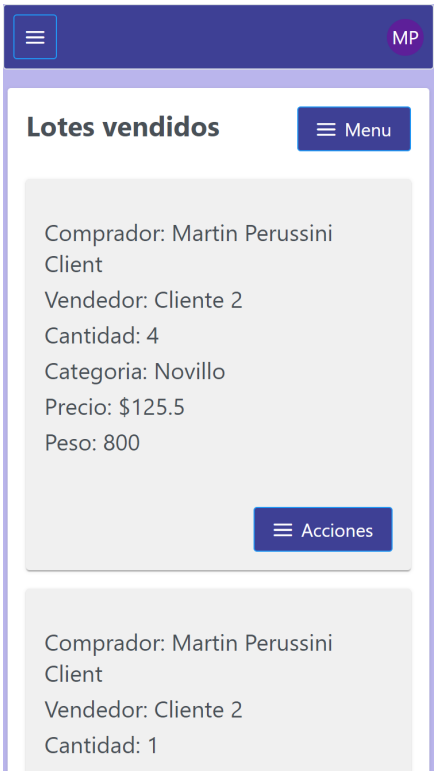


En pantalla grande:

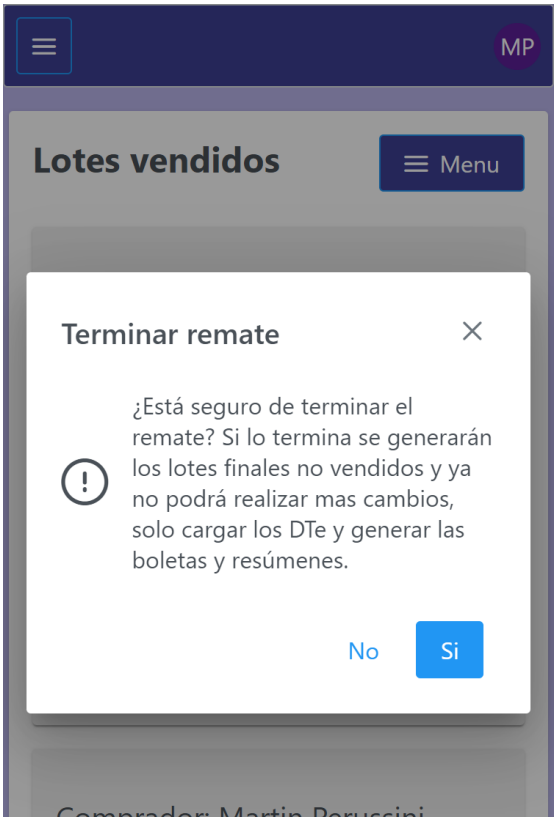


Se pesa el primer Lote:

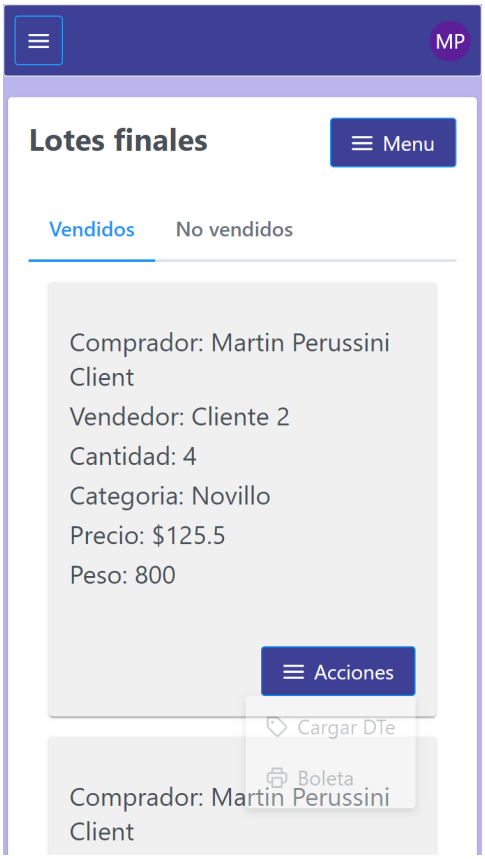
Resultado:



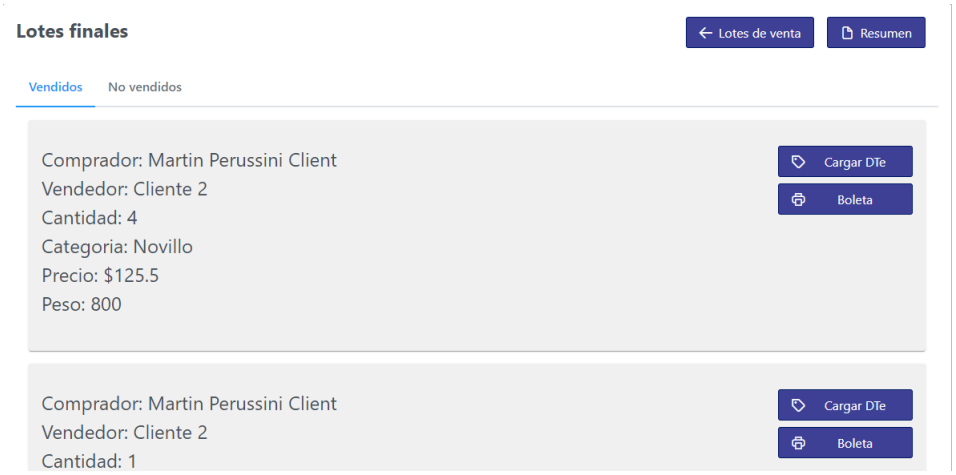
Se finaliza el remate:



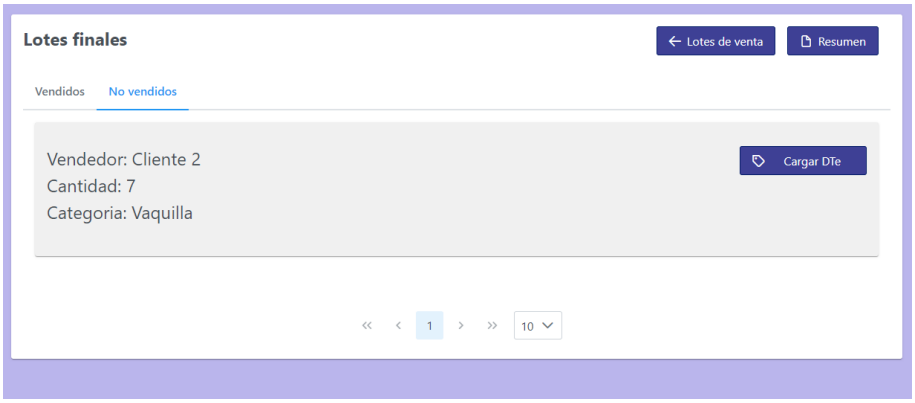
Se generaron los Lotes No Vendidos finales y ahora ya no se pueden eliminar, editar ni pesar los Vendidos:



Solo se pueden cargar los DTe e imprimir las boletas (próxima historia, ahora solo están los botones):



Y en los No Vendidos figura un Lote con las Vaquillas restantes que no se vendieron (10 - 3):



Tablas en la DB al final:

Auction (el *true* es por el atributo finished):

	11	11	2021-12-31 14:00:00	(NULL)	true	000-Prueba	1

NotSoldBatch:

		* id int	* amount int	dte_number varchar(255)	* animals_id int
		Filter	Filter	Filter	Filter
	1	11	7	(NULL)	14

SoldBatch:

		* id int	* amount int	dte_number int	* must_weigh bit(1)	* price double	weight double	animals_id int	client_id int	deleted bit(1)
		Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
	1	5	3	(NULL)	true	125.5	(NULL)	10	8	(NULL)
	2	7	1	(NULL)	true	23	(NULL)	1	6	(NULL)
	3	8	1	(NULL)	true	55	(NULL)	1	8	(NULL)
	4	9	2	(NULL)	false	55	(NULL)	3	8	(NULL)
	5	10	13	(NULL)	false	4444	(NULL)	5	6	(NULL)
	6	11	2	(NULL)	true	333	(NULL)	5	6	(NULL)
	7	12	6	(NULL)	false	12	(NULL)	11	6	(NULL)
	8	13	1	(NULL)	true	66	250	11	6	(NULL)
	9	14	1	(NULL)	true	222	(NULL)	12	6	(NULL)
	10	15	4	(NULL)	true	125.5	800	13	6	(NULL)
	11	16	1	(NULL)	false	80	(NULL)	13	6	(NULL)
	12	17	3	(NULL)	true	50	(NULL)	14	6	(NULL)

E.2.11.4 Código

```
@Service
public class SoldBatchServiceImpl implements SoldBatchService {
  ...
@Override
public SoldBatch saveSoldBatch(SoldBatch soldBatch, Integer animalsOnGroundId) throws
AnimalsOnGroundNotFound, HttpForbidenException, BatchNotFoundException,
AuctionNotFoundException, HttpUnauthorizedException {
   AnimalsOnGround animalsOnGround =
animalsOnGroundService.findByIdNotDeleted(animalsOnGroundId);
   Integer totalSold = this.getTotalSold(animalsOnGroundId);
   if(soldBatch.getAmount() == null){
       throw new IllegalArgumentException("El atributo 'cantidad' no puede ser nulo");
   }
   Batch batch = batchService.getBatchByAnimalsOnGroundId(animalsOnGroundId);
   if(batch == null || (batch.getDeleted() != null && batch.getDeleted())){
       throw new BatchNotFoundException("El LoteVendido pertenece a un Lote de Venta
inexistente");
   }
   Auction auction = batch.getAuction();
   if(auction.getDeleted() != null && auction.getDeleted()){
       throw new AuctionNotFoundException("El LoteVendido pertenece a un Remate
inexistente");
   }
   if(soldBatch.getAmount() <= 0){
       throw new HttpForbidenException("La cantidad de animales vendidos debe ser mayor
a cero.");
   }
   if(soldBatch.getAmount()+totalSold > animalsOnGround.getAmount()){
       throw new HttpForbidenException("La cantidad de animales vendidos supera a la
cantidad disponible para la venta.");
   }
   if(auction.getFinished() != null && auction.getFinished()){
       throw new HttpForbidenException("No puede editarse un Remate finalizado .");
   }

if(!userService.getCurrentUserAuthorities().toArray()[0].toString().equals("Administrado
r")) {
       boolean userBelongsToAuction = auction.getUsers().stream().anyMatch(u ->
u.getId().equals(userService.getCurrentUser().getId()));
       if (!userBelongsToAuction) {
           throw new HttpUnauthorizedException("Usted no está autorizado a editar este
Remate.");
       }
   }
   soldBatch.setAnimalsOnGround(animalsOnGround);
   SoldBatch soldBatchSaved = soldBatchDAO.save(soldBatch);
   totalSold = this.getTotalSold(animalsOnGroundId);
   if(totalSold.equals(animalsOnGround.getAmount())){
       animalsOnGround.setSold(true);
       animalsOnGroundService.save(animalsOnGround);
   }
```

```
   Optional<NotSoldBatch> notSoldBatch =
notSoldBatchService.getNotSoldBatchesByAnimalsOnGroundId(animalsOnGroundId);
   if(notSoldBatch.isPresent()){
       NotSoldBatch notSoldBatch1 = notSoldBatch.get();
       notSoldBatch1.setAmount(notSoldBatch1.getAmount()-soldBatchSaved.getAmount());
       notSoldBatchService.save(notSoldBatch1);
   }
   return soldBatchSaved;
}
...
}
```

```
@Service
public class BatchServiceImpl implements BatchService {
  ...

@Override
public AnimalsOnGround updateAnimalsOnGroundById(Integer animalsId, AnimalsOnGroundDTO
animalsOnGroundDTO) throws BadHttpRequest,IllegalArgumentException,
AnimalsOnGroundNotFound, HttpForbidenException, AuctionNotFoundException {
   return animalsOnGroundService.updateAnimalsOnGround(animalsId,animalsOnGroundDTO );
}
...
}


@Service
public class AnimalsOnGroundServiceImpl implements AnimalsOnGroundService {
  ...
@Override
public AnimalsOnGround updateAnimalsOnGround(Integer animalsId, AnimalsOnGroundDTO
animalsOnGroundDTO) throws BadHttpRequest,
IllegalArgumentException,AnimalsOnGroundNotFound, AuctionNotFoundException,
HttpForbidenException {
   if(animalsOnGroundDTO.getId() != null &&
!animalsOnGroundDTO.getId().equals(animalsId)){
       throw new BadHttpRequest("El id del path no coincide con el id del body del
request");
   }

   if(animalsOnGroundDTO.getAmount() != null && animalsOnGroundDTO.getAmount() <= 0){
       throw new IllegalArgumentException("La cantidad de animales debe ser mayor a
cero");
   }
   AnimalsOnGround animalsOnGround = getAnimalsOnGroundNotDeletedById(animalsId);
   if(animalsOnGround == null){
       throw new AnimalsOnGroundNotFound("El conjunto de animales en pista con id: " +
animalsId + " no existe");
   }
   Auction auction = batchService.getBatchByAnimalsOnGroundId(animalsId).getAuction();
   if(auction.getDeleted()!=null && auction.getDeleted()){
       throw new AuctionNotFoundException("El conjunto de Animales En Pista pertenece a
un remate inexistente.");
```

```
    }
    if(auction.getFinished()!=null && auction.getFinished()){
        throw new HttpForbidenException("No se pueden editar Animales En Pista de un
remate que ya se ha realizado");
    }

    Integer totalVendidos = soldBatchService.getTotalSold(animalsId);
    if(animalsOnGroundDTO.getAmount() != null && totalVendidos >
animalsOnGroundDTO.getAmount()){
        throw new HttpForbidenException("La cantidad de animales no puede ser menor a la
cantidad que ya se ha vendido.");
    }
    if(animalsOnGroundDTO.getAmount() != null && animalsOnGround.getAmount() <
animalsOnGroundDTO.getAmount() && animalsOnGround.getSold() != null &&
animalsOnGround.getSold()){
        animalsOnGround.setSold(false);
    }
    if(animalsOnGroundDTO.getAmount() != null &&
animalsOnGroundDTO.getAmount().equals(totalVendidos) && animalsOnGround.getSold() !=
null && !animalsOnGround.getSold()){
        animalsOnGround.setSold(true);
    }
    animalsOnGroundDTO.setSold(null);
    animalsOnGoundMapper.updateAnimalsOnGroundFromDto(animalsOnGroundDTO,
animalsOnGround);
    return animalsOnGroundDAO.save(animalsOnGround);
}
...
}
```

```
@Service
public class SoldBatchServiceImpl implements SoldBatchService {
  ...
@Override
public SoldBatch deleteById(Integer soldBatchId) throws HttpUnauthorizedException,
AnimalsOnGroundNotFound, SoldBatchNotFoundException, AuctionNotFoundException,
HttpForbidenException, BatchNotFoundException {
    SoldBatch soldBatch = findByIdNotDeleted(soldBatchId);
    AnimalsOnGround animalsOnGround =
animalsOnGroundService.findByIdNotDeleted(soldBatch.getAnimalsOnGround().getId());
    Batch batch = batchService.getBatchByAnimalsOnGroundId(animalsOnGround.getId());
    if(batch == null || (batch.getDeleted() != null && batch.getDeleted())){
        throw new BatchNotFoundException("El LoteVendido pertenece a un Lote de Venta
inexistente");
    }
    Auction auction = batch.getAuction();
    if(auction.getDeleted() != null && auction.getDeleted()){
        throw new AuctionNotFoundException("El LoteVendido pertenece a un Remate
inexistente");
    }

    if(auction.getFinished() != null && auction.getFinished()){
        throw new HttpForbidenException("No puede editarse un Remate finalizado.");
    }
```

```
if(!userService.getCurrentUserAuthorities().toArray()[0].toString().equals("Administrado
r")) {
        boolean userBelongsToAuction = auction.getUsers().stream().anyMatch(u ->
u.getId().equals(userService.getCurrentUser().getId()));
        if (!userBelongsToAuction) {
            throw new HttpUnauthorizedException("Usted no está autorizado a editar este
Remate.");
        }
    }
    animalsOnGround.setSold(false);
    animalsOnGroundService.save(animalsOnGround);
    soldBatchDAO.deleteById(soldBatchId);
    return soldBatch;
}
...
}
```

```
@Service
public class SoldBatchServiceImpl implements SoldBatchService {
  ...
@Override
public Page<SoldBatchResponseDTO> getSoldBatchesByAuctionAndPage(Integer auctionId,
Integer page, Integer limit) {
    Pageable p = PageRequest.of(page, limit);
    Page<SoldBatch> soldBatches = soldBatchDAO.findByAuctionId(auctionId, p);
    List<SoldBatchResponseDTO> responseDTOList = new ArrayList<>();
    for(SoldBatch soldBatch: soldBatches){
        SoldBatchResponseDTO soldBatchResponseDTO = new SoldBatchResponseDTO();
        soldBatchResponseDTO.setId(soldBatch.getId());
        soldBatchResponseDTO.setAmount(soldBatch.getAmount());
        soldBatchResponseDTO.setPrice(soldBatch.getPrice());
        soldBatchResponseDTO.setDteNumber(soldBatch.getDteNumber());
        soldBatchResponseDTO.setMustWeigh(soldBatch.getMustWeigh());
        soldBatchResponseDTO.setPaymentTerm(soldBatch.getPaymentTerm());
        soldBatchResponseDTO.setWeight(soldBatch.getWeight());
        soldBatchResponseDTO.setCategory(soldBatch.getAnimalsOnGround().getCategory());
        soldBatchResponseDTO.setBuyer(soldBatch.getClient());

soldBatchResponseDTO.setSeller(clientService.findByProvenanceId(batchService.getBatchByA
nimalsOnGroundId(soldBatch.getAnimalsOnGround().getId()).getProvenance().getId()));
        responseDTOList.add(soldBatchResponseDTO);
    }
    return new PageImpl<>(responseDTOList, p, soldBatches.getTotalElements());
}
...
}
```

```
@Service
public class NotSoldBatchServiceImpl implements NotSoldBatchService {
  ...
@Override
public Page<SoldBatchResponseDTO> getNotSoldBatchesByAuctionAndPage(Integer auctionId,
```

```
Integer page, Integer limit) {
   Pageable p = PageRequest.of(page, limit);
   Page<NotSoldBatch> soldBatches = notSoldBatchDAO.findByAuctionId(auctionId, p);
   List<SoldBatchResponseDTO> responseDTOList = new ArrayList<>();
   for(NotSoldBatch notSoldBatch: soldBatches){
       SoldBatchResponseDTO soldBatchResponseDTO = new SoldBatchResponseDTO();
       soldBatchResponseDTO.setId(notSoldBatch.getId());
       soldBatchResponseDTO.setAmount(notSoldBatch.getAmount());
       soldBatchResponseDTO.setDteNumber(notSoldBatch.getDteNumber());

soldBatchResponseDTO.setCategory(notSoldBatch.getAnimalsOnGround().getCategory());

soldBatchResponseDTO.setSeller(clientService.findByProvenanceId(batchService.getBatchByA
nimalsOnGroundId(notSoldBatch.getAnimalsOnGround().getId()).getProvenance().getId()));
       responseDTOList.add(soldBatchResponseDTO);
   }
   return new PageImpl<>(responseDTOList, p, soldBatches.getTotalElements());
}
...
}
```

E.2.12 Historia de Usuario 10 - Generar PDF del orden de salida de animales

E.2.12.1 Mockup

En este caso el mockup fue confeccionado en un archivo de Google Sheets, el cual mostramos a continuación:

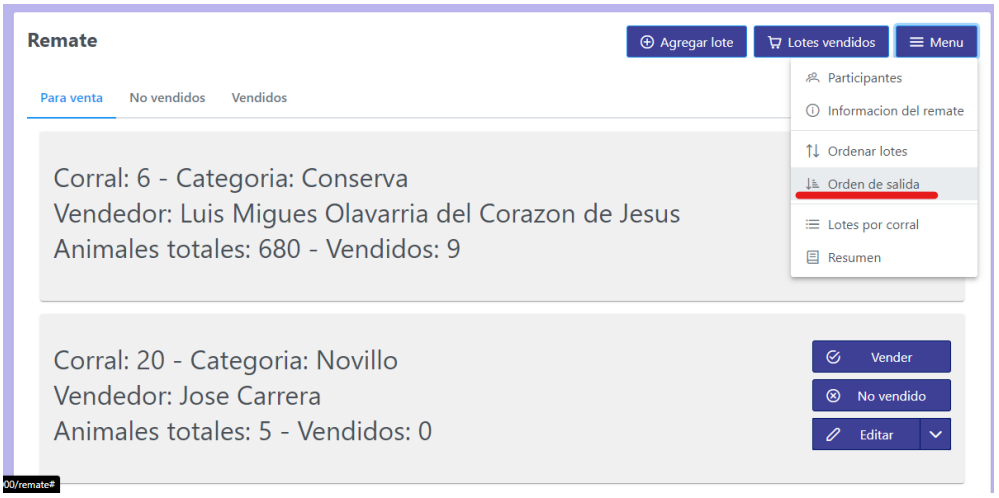
N Corral	Vendedor	Cantidad	Categoria
1	Martin Perussini	4	Vaquilla
2	Tomas Ravelli	3	Vaquilla
2	Tomas Ravelli	2	Vaquilla
3	Juan Carlos Albornoz	5	Ternero
4	Enrique Peña Nieto	2	Vaca
5	Gaston Diaz	3	Vaca
5	Gaston Diaz	3	Ternero

E.2.12.2 Criterio de aceptación

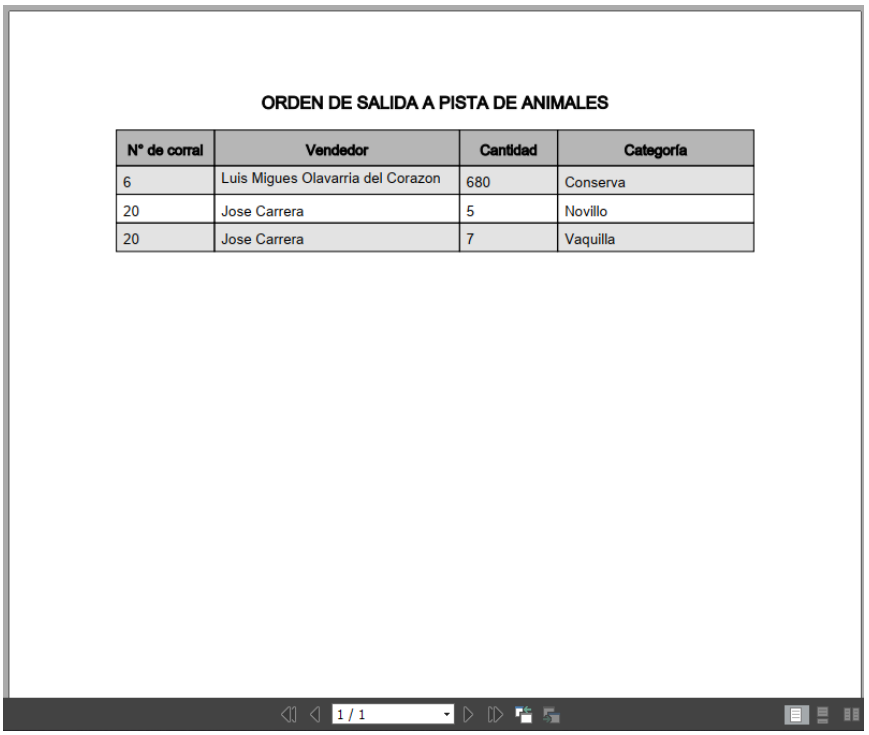
Archivo en formato PDF descargado correctamente.

Resultado:

Se presiona la opción en el menú:



PDF resultante:



ORDEN DE SALIDA A PISTA DE ANIMALES

N° de corral	Vendedor	Cantidad	Categoría
6	Luis Migues Olavarria del Corazon	680	Conserva
20	Jose Carrera	5	Novillo
20	Jose Carrera	7	Vaquilla

1 / 1

E.2.13 Historia de Usuario 12 - Cargar números de DTe

E.2.13.1 Mockup

Al presionar el botón de “Cargar DTe” se abrirá un diálogo con un campo de texto donde se permitirá cargar el dato, con un botón de Cancelar y uno de Aceptar.

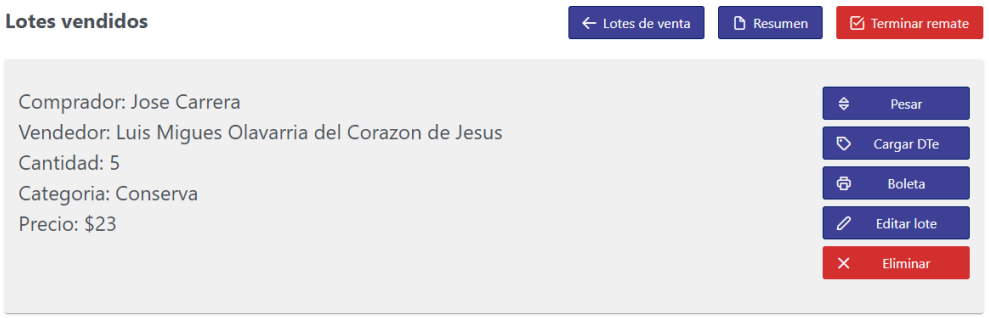
Figura 104: Mockup diálogo cargar DTe

E.2.13.2 Criterio de aceptación

Los lotes a los cuales se les cargó el número de DT-e deberán tener ese dato guardado en la base de datos, así como mostrar dicho número en el listado de todos los lotes finales que surgieron luego de la realización de un remate.

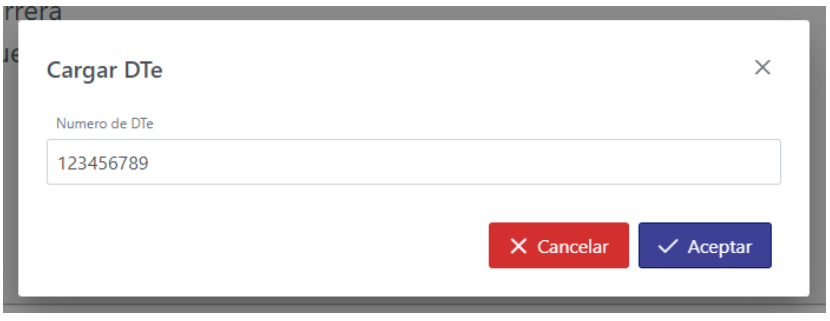
Resultado:

Al principio:

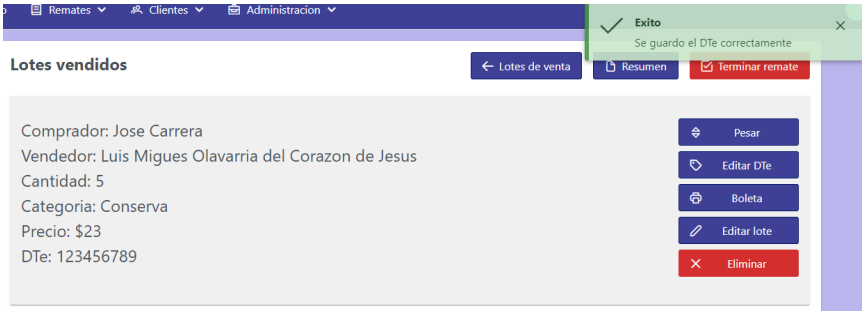


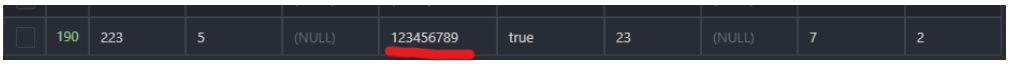
	id int	amount int	deleted bit(1)	dte_number varchar(255)	must_weigh bit(1)	price double	weight double	animals_id int	client_id int
177	206	2	(NULL)	(NULL)	true	58	(NULL)	6	1
178	207	1	(NULL)	(NULL)	true	679	(NULL)	6	2
179	208	1	(NULL)	(NULL)	true	24	(NULL)	6	3
180	209	1	(NULL)	(NULL)	true	666	(NULL)	6	1
181	210	1	(NULL)	(NULL)	true	4	(NULL)	6	3
182	211	1	(NULL)	(NULL)	true	555	(NULL)	6	3
183	212	1	(NULL)	(NULL)	true	556	(NULL)	6	1
184	213	1	(NULL)	(NULL)	true	58	(NULL)	6	2
185	214	58	(NULL)	(NULL)	true	1	(NULL)	6	2
186	215	1	(NULL)	(NULL)	true	1	55	7	3
187	217	1	(NULL)	(NULL)	true	5	(NULL)	7	1
188	218	1	(NULL)	(NULL)	true	55	(NULL)	7	2
189	219	1	(NULL)	(NULL)	true	576	(NULL)	7	3
190	223	5	(NULL)	(NULL)	true	23	(NULL)	7	2

Se carga el DTe:



Resultado:





E.2.14 Historia de Usuario 13 - Historial de remates

E.2.14.1 Mockup

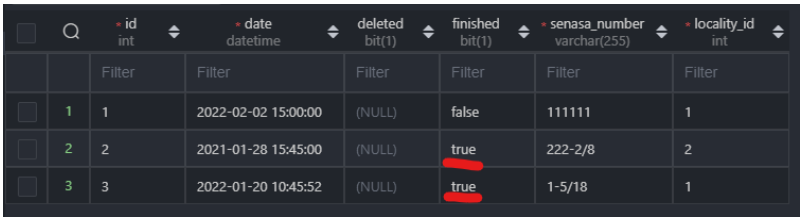
En este caso, el mockup utilizado es el mismo que se usó en la historia 14, ya que no se vió necesidad de crear uno nuevo por la similitud entre ambas pantallas.

E.2.14.2 Criterio de aceptación

Se le muestra al usuario un listado de remates, los cuales deben estar relacionados al mismo y deben estar ya finalizados.

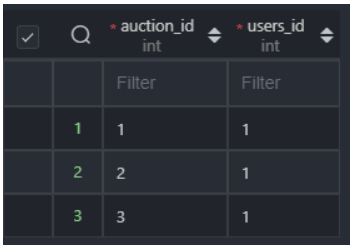
Resultado:

Listado de remates en el sistema (notar que hay 2 finalizados):



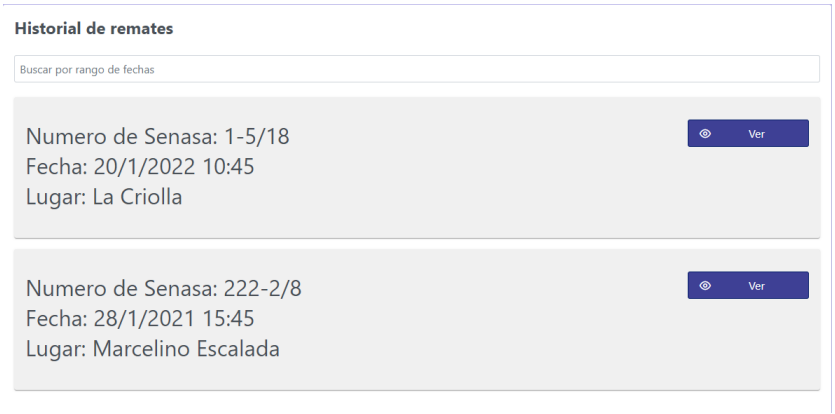
	* id int	* date datetime	deleted bit(1)	finished bit(1)	* senasa_number varchar(255)	* locality_id int
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2022-02-02 15:00:00	(NULL)	false	111111	1
2	2	2021-01-28 15:45:00	(NULL)	true	222-2/8	2
3	3	2022-01-20 10:45:52	(NULL)	true	1-5/18	1

El usuario con id 1 (mperussini) tiene relación con los 3 remates:



	* auction_id int	* users_id int
	Filter	Filter
1	1	1
2	2	1
3	3	1

Y efectivamente los mismos se listan en la pantalla del historial:



E.2.15 Historia de Usuario 17 - Generación boletas de ventas

E.2.15.1 Mockup

En este caso, al igual que en la historia 10, el mockup no fue confeccionado en Figma, sino que se utilizó un archivo de Google Docs donde, entre varios diseños, se propuso algo muy parecido al resultado final:

GANADOS REMATES S.A.
CONSIGNATARIOS DE HACIENDA

SAN JUAN 957 - TEL: (0341) 4210223 - 4214311 - 4216107 - ROSARIO
info@ganadosremates.com.ar - www.ganadosremates.com.ar
Ventas en Mercado Rosarioa
Ferias: San Justo - Campo Andino - San Javier - La Criolla - Reconquista - Roldán
Remates televisados en directo por Canal Rural

INFORMACION GENERAL		
FECHA: 24/01/2022	**LUGAR(lugar del remate): La Criolla**	**CORRAL: 14**
VENDEDOR	**TOMAS RAVELLI**	

COMPRADOR		MARTIN PERUSSINI	
DATOS DE VENTA			
CATEGORIA	CANTIDAD	KILOS EN PIE	PRECIO
VAQUILLAS	5	640 (solo si se pesa)	$120
PLAZO		30 dias	

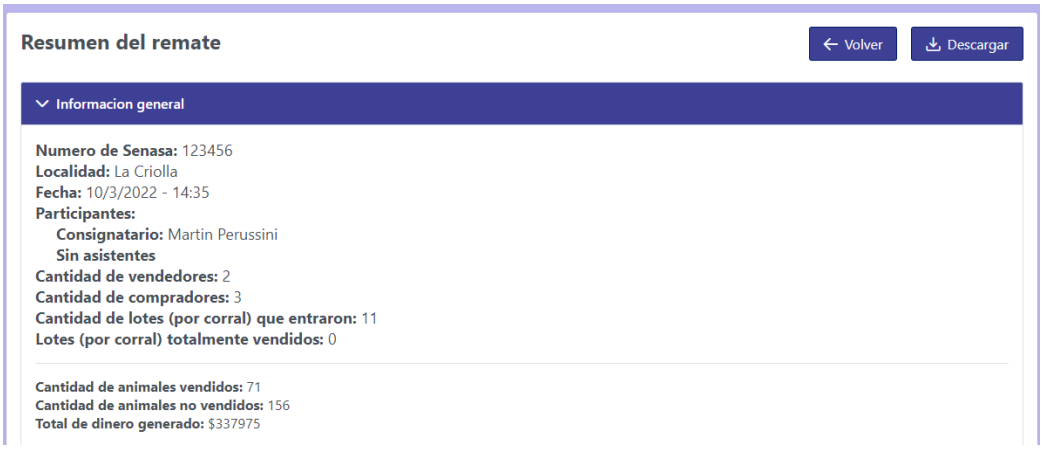
E.2.16 Historia de Usuario 15 - Reportes estadísticos de un remate

E.2.16.1 Criterio de aceptación

Todos los datos mostrados dentro de los reportes deberán ser consistentes con los datos reales que surgieron en la realización de un remate (los cuales se encuentran almacenados en la base de datos).

Resultados:

Datos del resumen (datos generales, luego se repite por categoría):



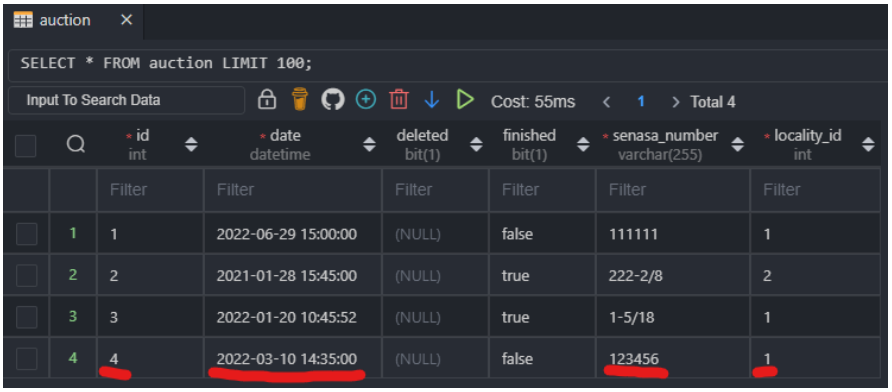
Vendedores

Nombre	Animales vendidos	Animales no vendidos	Dinero generado
Josefino Seferino Namuncurá Barrios	24	5	$77867
Jose Carrera	47	151	$260108

Compradores

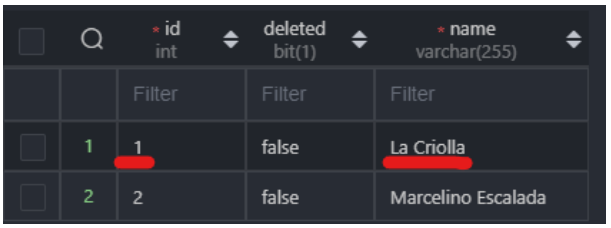
Nombre	Animales comprados	Dinero invertido
Josefino Seferino Namuncurá Barrios	44	$202886
Jose Carrera	10	$30738
Luis Migues Olavarria del Corazon de Jesus	17	$104351

Datos en la DB:

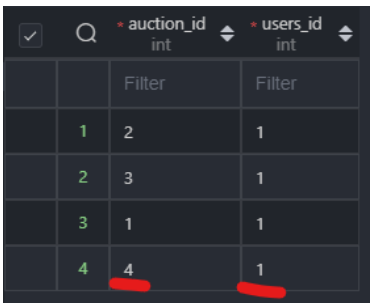


Número de Senasa, fecha y hora coinciden.

Localidad con id=1:



Participantes:



Lotes de venta por corral (11 con auctionId=4):

batch

```
SELECT * FROM batch LIMIT 100;
```

Input To Search Data | Cost: 17ms | 1 | Total 15

	* id int	* corral_number int	deleted bit(1)	dte_number varchar(255)	auction_id int	* provenance_id int
2	6	6	(NULL)	(NULL)	1	1
3	7	20	(NULL)	(NULL)	1	3
4	8	2	(NULL)	(NULL)	1	4
5	9	1	(NULL)	4124124	4	3
6	10	2	(NULL)	(NULL)	4	2
7	11	3	(NULL)	(NULL)	4	2
8	12	4	(NULL)	(NULL)	4	3
9	13	5	(NULL)	(NULL)	4	3
10	14	6	(NULL)	(NULL)	4	3
11	15	7	(NULL)	(NULL)	4	2
12	16	8	(NULL)	(NULL)	4	2
13	17	991	(NULL)	123123123	4	2
14	18	66	(NULL)	(NULL)	4	4
15	19	48	(NULL)	(NULL)	4	4

(Sus id van de 9 a 19)

AnimalsOnGround pertenecientes a esos lotes:

animals_on_ground

```
SELECT * FROM animals_on_ground LIMIT 100;
```

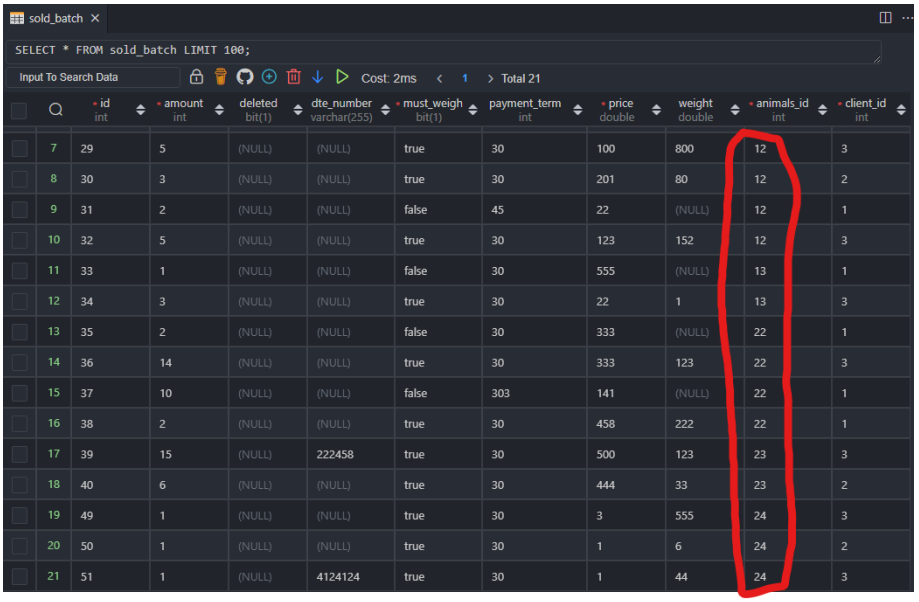
Input To Search Data | Cost: 22ms | 1 | Total 19

	* id int	* amount int	deleted bit(1)	* not_sold bit(1)	* sold bit(1)	starting_order int	* category_id int	batch_id int
6	11	5	(NULL)	false	false	-1	2	8
7	12	15	(NULL)	false	true	-1	1	9
8	13	15	(NULL)	false	false	1	3	9
9	14	15	(NULL)	false	false	3	1	10
10	15	30	(NULL)	false	false	4	1	11
11	16	6	(NULL)	false	false	5	1	12
12	17	20	(NULL)	false	false	6	4	13
13	18	15	(NULL)	false	false	7	4	14
14	19	2	(NULL)	false	false	8	4	15
15	20	1	(NULL)	false	false	9	3	15
16	21	3	(NULL)	false	false	10	3	16
17	22	76	(NULL)	false	false	2	5	17
18	23	23	(NULL)	false	false	0	4	18
19	24	6	(NULL)	false	false	-1	1	19

La suma de sus cantidades es 15+15+15+30+6+20+15+2+1+3+76+23+6=227, que es igual a 71 vendidos + 156 no vendidos.

(sus id van desde 12 a 24).

Animales vendidos:



sold_batch

```
SELECT * FROM sold_batch LIMIT 100;
```

Input To Search Data — Cost: 2ms — 1 — Total 21

	id int	amount int	deleted bit(1)	dte_number varchar(255)	must_weigh bit(1)	payment_term int	price double	weight double	animals_id int	client_id int
7	29	5	(NULL)	(NULL)	true	30	100	800	12	3
8	30	3	(NULL)	(NULL)	true	30	201	80	12	2
9	31	2	(NULL)	(NULL)	false	45	22	(NULL)	12	1
10	32	5	(NULL)	(NULL)	true	30	123	152	12	3
11	33	1	(NULL)	(NULL)	false	30	555	(NULL)	13	1
12	34	3	(NULL)	(NULL)	true	30	22	1	13	3
13	35	2	(NULL)	(NULL)	false	30	333	(NULL)	22	1
14	36	14	(NULL)	(NULL)	true	30	333	123	22	3
15	37	10	(NULL)	(NULL)	false	303	141	(NULL)	22	1
16	38	2	(NULL)	(NULL)	true	30	458	222	22	1
17	39	15	(NULL)	222458	true	30	500	123	23	3
18	40	6	(NULL)	(NULL)	true	30	444	33	23	2
19	49	1	(NULL)	(NULL)	true	30	3	555	24	3
20	50	1	(NULL)	(NULL)	true	30	1	6	24	2
21	51	1	(NULL)	4124124	true	30	1	44	24	3

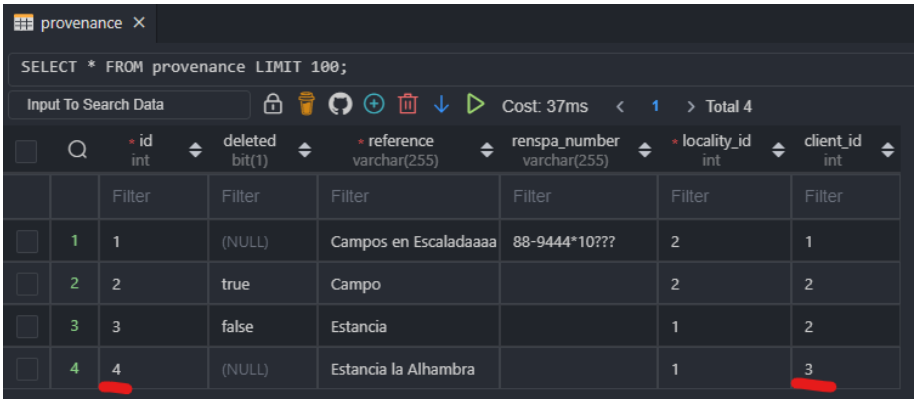
La suma de cantidades es 5+3+2+5+1+3+2+14+10+2+15+6+1+1+1=71, lo cual es consistente (de aquí también sale la cantidad no vendida: 227-71=156).

Ahora sigamos a un cliente:

Josefino Seferino Namuncurá Barrios, id=3:

Ventas:

Su procedencia (id=4):



provenance

```
SELECT * FROM provenance LIMIT 100;
```

Input To Search Data — Cost: 37ms — 1 — Total 4

	id int	deleted bit(1)	reference varchar(255)	renspa_number varchar(255)	locality_id int	client_id int
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	(NULL)	Campos en Escaladaaaa	88-9444*10???	2	1
2	2	true	Campo		2	2
3	3	false	Estancia		1	2
4	4	(NULL)	Estancia la Alhambra		1	3

Sus lotes en este remate:

batch

```
SELECT * FROM batch LIMIT 100;
```

Input To Search Data — Cost: 2ms — 1 — Total 15

	* id int	* corral_number int	deleted bit(1)	dte_number varchar(255)	auction_id int	* provenance_id int
2	6	6	(NULL)	(NULL)	1	1
3	7	20	(NULL)	(NULL)	1	3
4	8	2	(NULL)	(NULL)	1	4
5	9	1	(NULL)	4124124	4	3
6	10	2	(NULL)	(NULL)	4	2
7	11	3	(NULL)	(NULL)	4	2
8	12	4	(NULL)	(NULL)	4	3
9	13	5	(NULL)	(NULL)	4	3
10	14	6	(NULL)	(NULL)	4	3
11	15	7	(NULL)	(NULL)	4	2
12	16	8	(NULL)	(NULL)	4	2
13	17	991	(NULL)	123123123	4	2
14	18	66	(NULL)	(NULL)	4	4
15	19	48	(NULL)	(NULL)	4	4

(ids 18 y 19)

Los animalsOnGround de esos lotes:

animals_on_ground

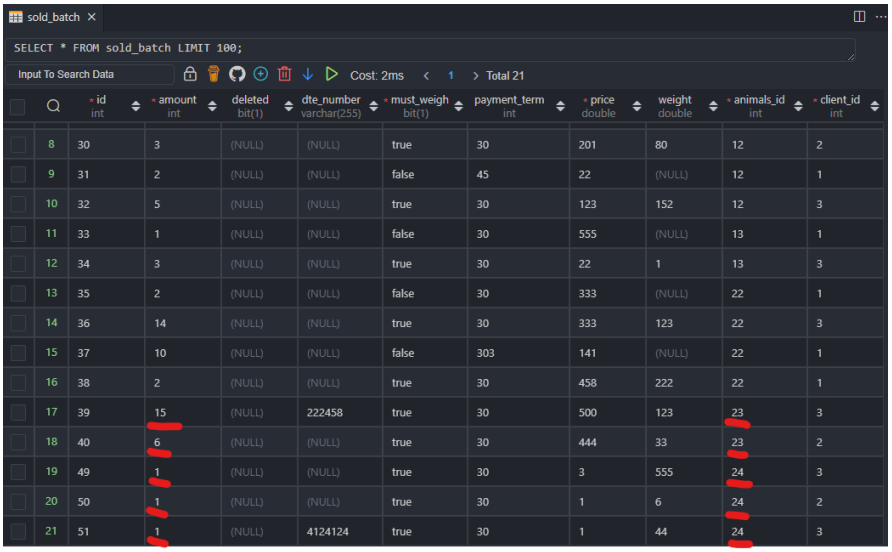
```
SELECT * FROM animals_on_ground LIMIT 100;
```

Input To Search Data — Cost: 2ms — 1 — Total 19

	* id int	* amount int	deleted bit(1)	* not_sold bit(1)	* sold bit(1)	starting_order int	* category_id int	batch_id int
6	11	5	(NULL)	false	false	-1	2	8
7	12	15	(NULL)	false	true	-1	1	9
8	13	15	(NULL)	false	false	1	3	9
9	14	15	(NULL)	false	false	3	1	10
10	15	30	(NULL)	false	false	4	1	11
11	16	6	(NULL)	false	false	5	1	12
12	17	20	(NULL)	false	false	6	4	13
13	18	15	(NULL)	false	false	7	4	14
14	19	2	(NULL)	false	false	8	4	15
15	20	1	(NULL)	false	false	9	3	15
16	21	3	(NULL)	false	false	10	3	16
17	22	76	(NULL)	false	false	2	5	17
18	23	23	(NULL)	false	false	0	4	18
19	24	6	(NULL)	false	false	-1	1	19

Un total de 23+6=29 animales para vender (ids 23 y 24).

Sus animales vendidos:



sold_batch

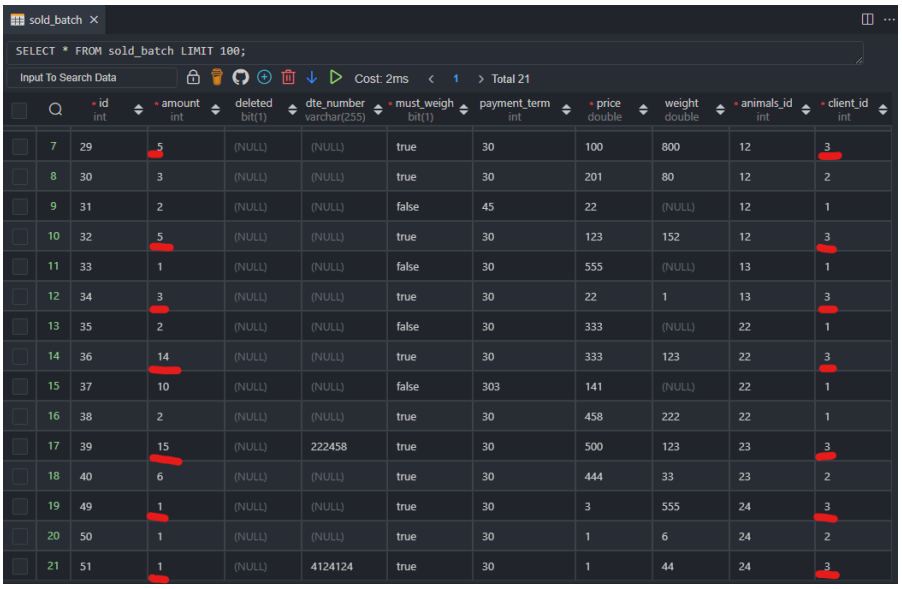
```
SELECT * FROM sold_batch LIMIT 100;
```

Input To Search Data | Cost: 2ms | 1 | Total 21

	id int	amount int	deleted bit(1)	dte_number varchar(255)	must_weigh bit(1)	payment_term int	price double	weight double	animals_id int	client_id int
8	30	3	(NULL)	(NULL)	true	30	201	80	12	2
9	31	2	(NULL)	(NULL)	false	45	22	(NULL)	12	1
10	32	5	(NULL)	(NULL)	true	30	123	152	12	3
11	33	1	(NULL)	(NULL)	false	30	555	(NULL)	13	1
12	34	3	(NULL)	(NULL)	true	30	22	1	13	3
13	35	2	(NULL)	(NULL)	false	30	333	(NULL)	22	1
14	36	14	(NULL)	(NULL)	true	30	333	123	22	3
15	37	10	(NULL)	(NULL)	false	303	141	(NULL)	22	1
16	38	2	(NULL)	(NULL)	true	30	458	222	22	1
17	39	15	(NULL)	222458	true	30	500	123	23	3
18	40	6	(NULL)	(NULL)	true	30	444	33	23	2
19	49	1	(NULL)	(NULL)	true	30	3	555	24	3
20	50	1	(NULL)	(NULL)	true	30	1	6	24	2
21	51	1	(NULL)	4124124	true	30	1	44	24	3

Un total de 15+6+1+1+1=24 vendidos (por ende, 5 sin vender).

Compras:



sold_batch

```
SELECT * FROM sold_batch LIMIT 100;
```

Input To Search Data | Cost: 2ms | 1 | Total 21

	id int	amount int	deleted bit(1)	dte_number varchar(255)	must_weigh bit(1)	payment_term int	price double	weight double	animals_id int	client_id int
7	29	5	(NULL)	(NULL)	true	30	100	800	12	3
8	30	3	(NULL)	(NULL)	true	30	201	80	12	2
9	31	2	(NULL)	(NULL)	false	45	22	(NULL)	12	1
10	32	5	(NULL)	(NULL)	true	30	123	152	12	3
11	33	1	(NULL)	(NULL)	false	30	555	(NULL)	13	1
12	34	3	(NULL)	(NULL)	true	30	22	1	13	3
13	35	2	(NULL)	(NULL)	false	30	333	(NULL)	22	1
14	36	14	(NULL)	(NULL)	true	30	333	123	22	3
15	37	10	(NULL)	(NULL)	false	303	141	(NULL)	22	1
16	38	2	(NULL)	(NULL)	true	30	458	222	22	1
17	39	15	(NULL)	222458	true	30	500	123	23	3
18	40	6	(NULL)	(NULL)	true	30	444	33	23	2
19	49	1	(NULL)	(NULL)	true	30	3	555	24	3
20	50	1	(NULL)	(NULL)	true	30	1	6	24	2
21	51	1	(NULL)	4124124	true	30	1	44	24	3

5+5+3+14+15+1+1 = 44 animales comprados.

E.2.17 Historia de Usuario 16 - Impresión reportes estadísticos

E.2.17.1 Mockup

También se trata de un archivo realizado mediante Google Docs, el cual en este caso dista mucho del resultado final en cuanto a lo estético, pero cuenta con las bases que luego serían usadas a la hora del desarrollo. El mismo se muestra a continuación:

Informe del remate realizado en [localidad] el día [fecha].
Número de Senasa: [numeroSenasa]

Participantes:

Consignatario/s: Nombre1, Nombre2...
Asistentes: Nombre3, Nombre4...

Balance general:

Total animales ingresados: XXX+YYY (cantidad de animales que fueron ofertados, derivado de los dos números que siguen)

Total animales vendidos: XXX (suma de animales vendidos)
Total animales sin vender: YYY (suma de animales sin vender)

Ingreso total: $XXXX,XX (sumatoria de precio de los animales vendidos)

Cantidad de lotes de entrada: (suma de lotes de venta)
Cantidad de lotes de salida: (suma de lotes vendidos)

Cantidad de vendedores: (suma de Cliente distintos que tienen Lotes de venta)
Cantidad de compradores: (suma de compradores distintos de cada lote)

Compradores		
Nombre	**Cantidad**	**Dinero invertido**

Vendedores			
Nombre	**Cantidad vendida**	**Cantidad no vendida**	**Dinero bruto obtenido**
		Si es cero, queda en blanco	

RESUMEN POR CATEGORÍA DE ANIMALES

Categoria: Vaca

Total de animales ingresados: XXX

Total de animales vendidos: XXX

Total de animales sin vender: XXX

Ingreso monetario total: $XXX,YY

Compradores		
Nombre	**Cantidad**	**Dinero invertido**

Vendedores			
Nombre	**Cantidad vendida**	**Cantidad no vendida**	**Dinero bruto obtenido**

Categoria: Toro

Total de animales ingresados: XXX

Total de animales vendidos: XXX

Total de animales sin vender: XXX

Ingreso monetario total: $XXX,YY

Compradores		
Nombre	**Cantidad**	**Dinero invertido**

Vendedores			
Nombre	**Cantidad vendida**	**Cantidad no vendida**	**Dinero obtenido**