UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL SANTA FE

PROYECTO FINAL DE CARRERA

Ingeniería en Sistemas de Información

“SantaPesca”
Sistema para gestionar actividades en el turismo pesquero

Alumnos:

Capdevila, Pedro
Destéfanis, Martín Ian
Zianni, Gastón

Director de proyecto:

Ing. Vrancken, Lisandro E.

Codirector de proyecto:

Ing. Medina, Pablo M.

2021

1. Introducción

La idea del presente proyecto final de carrera surgió como una oportunidad de negocio, sobre la base de percibir la existencia de muchas personas interesadas por la pesca deportiva o como ocio que no conoce, frecuenta o practica la misma, y por lo tanto no cuenta con información y opciones para realizarla. Desde esa perspectiva, se contempló como premisa que estas personas, al momento de imaginarse una excursión de pesca, no saben dónde ir, así como tampoco conocen sobre los mejores pesqueros de determinadas zonas, las temporadas de pesca, cabañas existentes en el lugar, campings, lugares de venta de carnada y/o artículos de pesca, excursiones, bajadas de lancha, guarderías, especies vedadas, reglamentaciones[1][2][3].

Por todo ello, ideamos un proyecto que brinde respuesta a estas necesidades, que acerque información como un servicio, de modo de simplificar la organización a las personas y al mismo tiempo fomentar el turismo pesquero.

De esta manera, el proyecto se enfocó entonces en la generación de un producto de software innovador denominado “Santa Pesca”, dado que en nuestro entorno no identificamos otra solución que brinde información de manera integral como la propuesta.

Este proyecto fue planteado con el propósito de lograr una aplicación que se ejecute tanto en la web como así también en un celular vía una APP, unificando toda la información del turismo pesquero y haciéndola accesible y sencilla para las personas.

Como primeros pasos del proyecto, trabajamos entonces en la fundamentación del proyecto y la definición del modelo de negocio que justifique el desarrollo de la solución, así como la definición de objetivos generales y específicos del proyecto. Estos aspectos son descriptos a continuación, como parte del primer capítulo de este informe.

1.1. Objetivo general

El objetivo general que guio el proyecto fue construir un sistema de información que permita brindar, en un espacio unificado, toda la información relacionada con la pesca, lugares recomendados y habilitados, venta de carnada, instrumentos de pesca, campings, cabañas, bajadas de lancha, red social, información turística y de reglamentación por provincia.

1.2. Objetivos específicos

Para lograr el objetivo anterior, se diseñaron los siguientes objetivos específicos:

[1] http://sentilapesca.com.ar/santa-fe-reglamento-pesca-deportiva
[2] http://www.pescaargentina.com.ar/pesca-deportiva/santa_fe/
[3] https://www.magyp.gob.ar/sitio/areas/pesca_continental/especies/

- Generar una plataforma que permita gestionar la información de una manera centralizada.
- Lograr una fuente de ingresos, mediante el uso de publicidad no invasiva en la aplicación móvil y web, de manera tal que sea rentable seguir trabajando en el espacio, permitiendo realizar actualizaciones y mejoras al producto.
- Desarrollar una arquitectura flexible que permita incorporar nuevos servicios y módulos.
- Desarrollar una arquitectura escalable para soportar futuros incrementos de tráfico y de usuarios.
- Diseñar e implementar una interfaz de usuario amigable, intuitiva y sencilla.
- Establecer normas de codificación e integración que ordenen el proceso de desarrollo.
- Diseñar y configurar un proceso de integración continua que refuerce la calidad del proceso de desarrollo del software.

1.3. Fundamentación

El origen de la idea de proyecto se basó en la detección de una gran proporción de personas que no son adeptos de la pesca como deporte pero que les gustaría realizarla y desconocen cómo, qué instrumental llevar, dónde ir, cuáles son los mejores puntos pesqueros de la zona, cuáles son las temporadas de pesca y campings. Por lo que mediante este servicio se pretende simplificar mucho la tarea y beneficiar al turismo pesquero, ya que reduce la falta de información de lugares de pesca, así como también aquellos servicios de alojamiento como campings, cabañas, shops de pesca, carnada, pescadores, bajadas de lancha, información sobre una localidad y excursiones de pesca.

La principal característica diferencial en nuestro producto es la innovación, bajo la consideración que no existe otra aplicación que brinde la misma información e integre varios sectores relacionados a la pesca en un solo lugar. Por lo tanto, la oportunidad detectada es la de fomentar el turismo en estos lugares, unificar la información para la obtención de información valiosa y relevante para una salida familiar o profesional por parte del usuario.

En la etapa inicial del proyecto se realizó un estudio preliminar de mercado (se describe con más detalle en la siguiente sección, *1.5. Modelo de Negocios*). En el mismo, se logró visualizar que en Argentina solo se encuentran aplicaciones nacionales que tienen diferentes fines como, por ejemplo, aplicaciones para fiscalizar torneos de pesca, obtener permisos de pesca, pronósticos de pesca o calendarios. Es decir, aplicaciones relacionadas pero enfocadas en distintos fines.

Como síntesis de lo expuesto, podemos decir que el problema a resolver fue la falta de información centralizada de los diferentes aspectos a tener en cuenta para poder practicar la actividad pesquera.

1.4. Modelo de Negocios

1.4.1 Introducción

A fines de poder lograr una mejor justificación en el desarrollo de la solución, el proyecto involucró acciones para lograr un modelo de negocios preliminar. Para esto, se trabajó bajo la estrategia Business Model Canvas (BMC), también conocido como “lienzo de modelo de negocio”, o simplemente “Canvas”.

Este modelo propone una plantilla (lienzo) de gestión estratégica para el desarrollo de nuevos modelos de negocio o para documentar modelos existentes. Facilita la definición y creación de modelos de negocio innovadores a través de la consideración de 4 grandes áreas:

- Clientes (Segmento de clientes, Canales y Relaciones con los clientes),
- Oferta (Oferta de Valor),
- Infraestructura (Actividades clave, Recursos clave y Red de socios), y
- Finanzas (Estructura de costos, Fuentes de ingresos).

A continuación se describe la aplicación del modelo canvas, explicando los distintos puntos y consolidando en el lienzo.

1.4.2. Etapas del modelo

Segmentos de clientes

En el corto plazo, apuntamos a usuarios objetivo localizados en la provincia de Santa Fe, con una edad superior a los 14 años.

Según una estimación realizada por el equipo, calculamos que de un grupo de 10 personas a 8 les gusta ir a pescar, pero no son pescadores recurrentes, es decir, ocasionalmente les gusta practicar la pesca pero no tienen mucha información sobre los posibles lugares en donde realizarla.

La idea inicial es comenzar en Santa Fe, ya que el equipo emprendedor se encuentra localizado allí. Además, en una primera instancia se necesita dar a conocer la aplicación, por lo que habría que salir a recorrer los principales puntos pesqueros de la zona. Luego, en una segunda instancia y cuando todo marche correctamente en Santa Fe, se podría analizar la posibilidad de ampliarse al resto del país.

En base a lo comentado, definimos dos tipos de destinatarios:

- Clientes: quienes nos generarán ingresos mediante la publicidad dentro de la aplicación. Además, en un futuro, cuando se tenga un número considerable de usuarios en la plataforma, se implementará un sistema premium para las publicaciones de negocios en donde podrán visualizarse primero las publicaciones pagas.
- Usuarios: aquellos consumidores finales de la aplicación.

Los principales clientes a los que se desea capturar el interés son: cabañas, casas de pesca, vendedores de carnada, vendedores de lancha, camping, clubes náuticos y guarderías de lancha.

La aplicación está destinada a todos aquellos usuarios que desean disfrutar de la naturaleza, quieran alquilar una cabaña, hacer excursiones de pesca, comprar carnada y obtener información variada sobre pesca (como dónde comprar elementos de pesca, que se pesca, etc.).

Propuesta de valor

Se ofrece una aplicación, tanto web como una app para celulares, donde se logra unificar toda la información sobre el turismo pesquero y brinda un espacio con tecnología de fácil acceso.

Hemos detectado que una importante cantidad de personas no se considera “fanática” o a fin a la pesca como deporte pero que les interesa y no saben cómo hacerlo, qué instrumental llevar, dónde ir, cuáles son los mejores pesqueros de la zona, cuáles son las temporadas de pesca, los campings que hay en la zona, si hay alguna excursión en el lugar, etc. Es un deporte en donde, si no se suele practicar con frecuencia ni se conocen lugares para hacerlo, es engorroso tener que buscar toda esta información únicamente para experimentar un día o un fin de semana de pesca. Por lo que, mediante este servicio, se simplifica mucho la tarea y beneficia el turismo pesquero.

Canales de distribución y comunicaciones

Nuestro producto llega a nuestros clientes de manera personal, recorriendo los diversos lugares pesqueros importantes de la provincia, haciéndonos conocer; para ponerles en conocimiento la idea propuesta.

Además, se va a llegar a los usuarios a través de campañas publicitarias en redes sociales y afiches/folletos en lugares estratégicos.

Relación con el cliente

Se va a tratar de lograr una fidelidad con nuestros clientes, brindándoles soporte y capacitación. Somos conscientes de que si nuestro producto tiene éxito no le va a impedir a nadie hacernos competencia, por lo que uno de nuestros principales objetivos va a ser tener un trato personal y generar una relación de confianza con ellos.

Reseñas y calificaciones para que los usuarios cuenten con un espacio donde puedan dar su opinión.

Fuentes de ingreso

Nuestra principal fuente de ingresos va a ser la publicidad dentro de la aplicación web y móvil. Estos ingresos, que se obtendrán por medio de los clientes, se van a determinar por un precio fijo base más una tasa variable que va a depender de las descargas de la aplicación móvil y visitas a la página web.

La publicidad a la que va a priorizar que se introduzca en nuestro espacio web y móvil, es la relacionada con el turismo pesquero:

- Cabañas
- Vendedores de lanchas
- Talleres náuticos
- Camping
- Ferreterías
- Shops de pesca
- Canales televisivos de pesca
- Guarderías de lancha

En una primera instancia, hasta conseguir clientes que quieran introducir publicidad personalizada, se va a utilizar *AdMob* por Google.

También se va a contar con una sección donde la gente va a poder donar dinero a través de Mercado Pago, criptomonedas, PayPal o cafecito app (https://cafecito.app/).

Recursos clave

El recurso más importante que conforma la estructura de nuestro modelo es la aplicación móvil y web, dentro de la cual se necesitan otros recursos para su vital funcionamiento:

- Desarrolladores encargados de diseñar e implementar el sistema, el mismo equipo emprendedor se encargará de esta tarea.
- Marketing: publicidad en redes sociales, Google, folletos, *stickers*.
- Recurso físico: servidor donde se ejecute la aplicación y que almacene la información.

Actividades clave

- Desarrollo de la plataforma y soporte.
- Reseñas de los servicios utilizados.
- Difusión en redes sociales.
- Recorrer los distintos lugares pesqueros para dar a conocer el sistema y la forma en que van a publicar sus servicios en la aplicación y en la web.

Socios claves

- Santa Fe turismo
- Estado
- Clubes náuticos
- Cabañas
- Proveedores de carnada, excursiones y elementos varios de pesca
- Inversionistas

Estructura de costos

- Infraestructura tecnológica: *GKE Autopilot, Cloud DNS and Storage from Google Cloud*.

De acuerdo a nuestra experiencia profesional estimamos que estos son los recursos que necesitamos para correr una instancia de “SantaPesca” en la nube de Google.
El costo total es de **180 USD** mensuales.

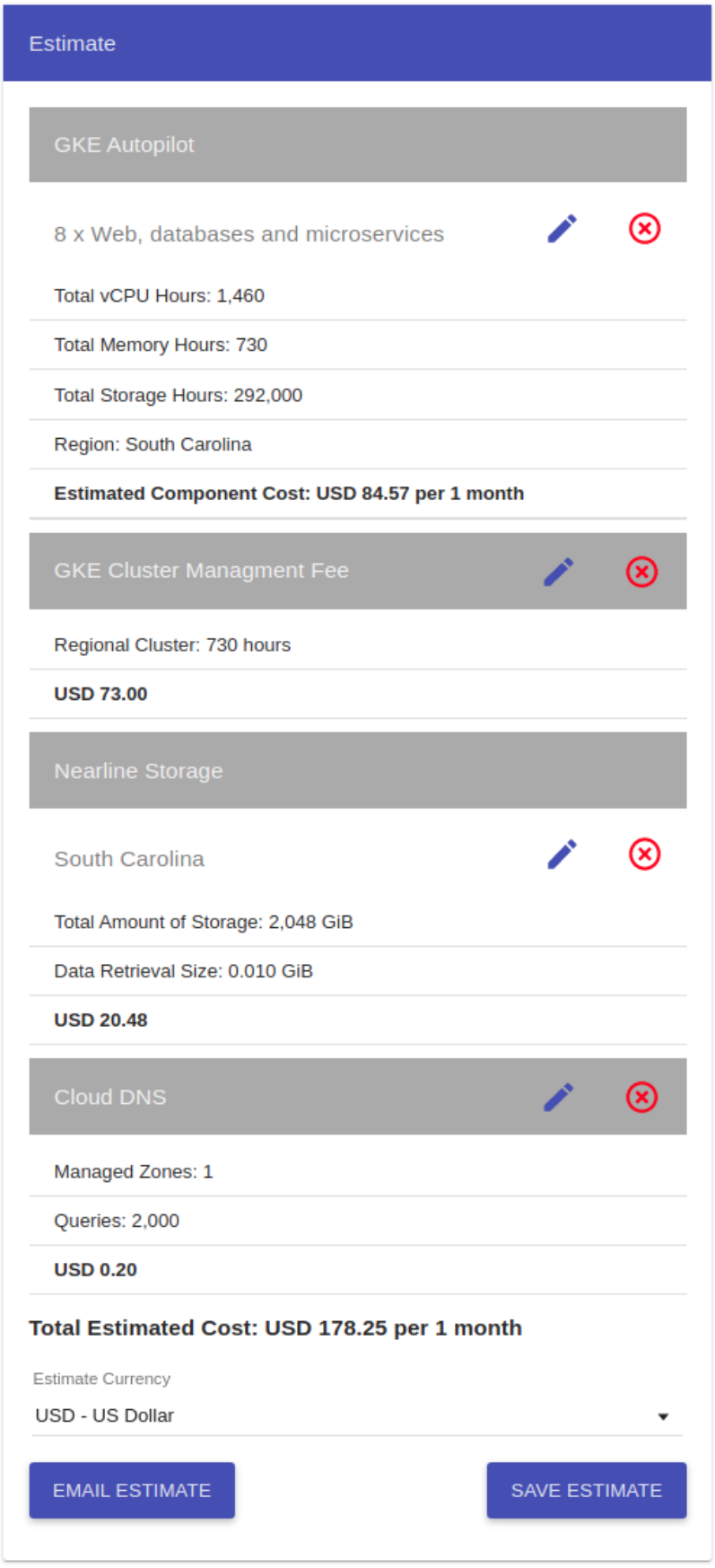


Figura 11.1 - Costo estimado en recursos de Google

- Recursos para la movilidad (nafta y transporte): Los autos que se usarían para movilidad son los propios de cada integrante. En una primera instancia se va a empezar por recorrer la provincia de Santa Fe (desde Rosario hasta Reconquista). Estimamos que se va a necesitar 1 tanque mensual (40 litros de nafta, **$4000** al día de la fecha).
- Marketing publicitario:
  - *Google Ads*: Se utilizó como zona la provincia de Santa Fe.

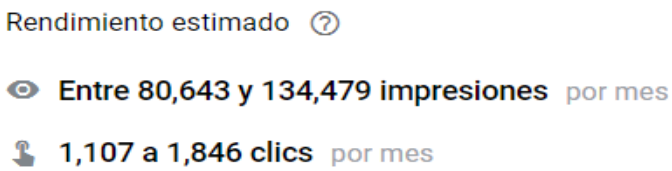


Figura 11.2. - Costo en marketing publicitario

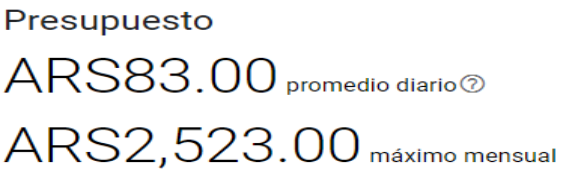


Figura 11.3. - Presupuesto

  - Publicidad en redes sociales. No hay una calculadora para estimar la publicidad a través de estos servicios por lo que recae en nosotros estimar el dinero que vamos a desear destinar a estos recursos.
- Recursos Humanos: El costo de RRHH es el tiempo invertido en nuestra aplicación. Por el momento no se desea contratar a profesionales extras.

Resumiendo, los costos mensuales son los siguientes:

Rubro de gastos	**Total Proyecto**
Recursos Humanos	$ -
Infraestructura tecnológica	$ 36000
Publicidad	$ 2500
Movilidad	$ 4000
TOTAL (mensual)	$ 42500

Estructura de ganancias estimadas

- Se piensa implementar anuncios publicitarios mediante banners dentro de la aplicación utilizando *AdMob* de Google.
- Por otro lado, estaremos abiertos a la posibilidad de que alguien desee realizar una donación brindando la funcionalidad dentro de la aplicación.
- Se piensa buscar un inversionista para hacer frente a los costos iniciales y que pueda sacar rédito. Como puede ser una empresa dedicada a la actividad pesquera y que le interese generar un espacio en donde pueda ofrecer sus servicios.

- También se ofrecerán funcionalidades que, para acceder a ellas, se deberá realizar un pago mensual.

1.5.3. Lienzo

<table>
<tr>
<td rowspan="2">SOCIOS CLAVE

* Santa Fe turismo
* Estado
* Clubes náuticos
* Cabañas
* Proveedores de carnada, excursiones y elementos varios de pesca
* Inversionistas</td>
<td>ACTIVIDADES CLAVE

* Desarrollo de la plataforma y soporte.
* Reseñas de los servicios utilizados.
* Difusión en redes sociales.
* Recorrer los distintos lugares pesqueros para dar a conocer el sistema y la forma en que van a publicar sus servicios en la aplicación y en la web.</td>
<td rowspan="2">PROPUESTA DE VALOR

Se ofrece una aplicación, tanto web como una app para celulares, donde se logra unificar toda la información sobre el turismo pesquero y brinda un espacio con tecnología de fácil acceso.</td>
<td>RELACION CLIENTE

* Fidelidad mediante el soporte y capacitación.

* Contará además con un espacio para reseñas y calificaciones.</td>
<td rowspan="2">CLIENTES

* Cabañas
* Casas de pesca
* Vendedores de carnada
* Vendedores de lancha
* Camping
* Clubes náuticos
* Guarderías de lancha</td>
</tr>
<tr>
<td>RECURSOS CLAVE

* Aplicación móvil y web
* Desarrolladores
* Marketing: publicidad en redes sociales, Google, folletos, stickers.
* Recurso físico: servidor donde se ejecute la aplicación y que almacene la información.</td>
<td>CANALES
* Nuestro producto llega a nuestros clientes de manera personal, recorriendo los diversos lugares pesqueros importantes de la provincia, haciéndonos conocer; para ponerles en conocimiento la idea propuesta.

* Además, se va a llegar a los usuarios a través de campañas publicitarias en redes sociales y afiches/folletos en lugares estratégicos</td>
</tr>
<tr>
<td colspan="3">Estructura de COSTES

* Servidores e infraestructura tecnológica: 180 USD mensuales
* Recursos para la movilidad (nafta y transporte): $4000
* Marketing y publicidad: $2500 mensuales
* No se planea contratar personas en la etapa inicial</td>
<td colspan="2">Fuentes de INGRESO

* Publicidad en la aplicación
* Donaciones</td>
</tr>
</table>

2. Metodología

Para llevar adelante el desarrollo del software como parte de este Proyecto Final de Carrera, decidimos utilizar las prácticas que más se ajustaban a nuestras necesidades tomando como base las metodologías Scrum y XP. En esta sección, se describen las mismas y cuáles aspectos de cada una se eligieron para desarrollo del proyecto.

2.1. Descripción

2.1.1. SCRUM

Scrum es un método para trabajar en equipo a partir de iteraciones o *Sprints*, se trata de una metodología ágil, y su objetivo es controlar y planificar proyectos con un gran volumen de cambios de última hora, en donde la incertidumbre es elevada.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto y los requisitos son cambiantes o poco definidos. También, dónde la innovación, competitividad, flexibilidad y la productividad son fundamentales.

El desarrollo se realiza de forma iterativa e incremental. Cada iteración tiene una duración preestablecida, obteniendo como resultado una versión del software con nuevas prestaciones listas para ser usadas. En cada nuevo *Sprint*, se va ajustando la funcionalidad ya construida y se añaden nuevas prestaciones priorizando siempre aquellas que aporten mayor valor de negocio.

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente prioriza los objetivos balanceando el valor que le aportan respecto a su coste y quedan repartidos en iteraciones y entregas.

A continuación, definimos algunos conceptos de Scrum:

- ***Product Backlog*:** es un listado de todas las historias que se pretenden hacer durante el desarrollo de un proyecto, en un lenguaje no técnico y priorizados por beneficio y coste.
- ***Sprint*:** Iteración de duración prefijada durante la cual el equipo trabaja para convertir las historias del *Product Backlog* a las que se ha comprometido, en una nueva versión del software totalmente operativo.
- ***Sprint Backlog*:** Lista de las tareas necesarias para llevar a cabo las historias del *sprint*.

Ventajas[4]

- **Gestión de las expectativas del cliente:**
  El cliente establece expectativas indicando el valor que le aporta cada requisito y cuando espera que esté completado. También da *feedback* desde el inicio del proyecto iteración a iteración, hacia su meta ahorrando esfuerzo y tiempo al evitar hipótesis.

- **Resultados anticipados:**
  Cada etapa del proceso arroja una serie de resultados. No es necesario, por tanto, que el cliente espere hasta el final para ver el resultado. Por lo que puede recuperar su inversión (y/o autofinanciarse) y sacar al mercado un producto antes que su competidor, hacer frente a urgencias o nuevas peticiones de clientes, etc.

- **Flexibilidad y adaptación a los contextos:**
  El desglose del trabajo favorece a una mayor flexibilidad a los cambios. Las necesidades del cliente se analizan e integran a las tareas en menos tiempo.

- **Gestión sistemática de riesgos:**
  Del mismo modo, los problemas que aparecen durante los procesos de gestión que pueden afectar a un proyecto son gestionados en el mismo momento de su aparición. Esto es posible debido a que la intervención de los equipos de trabajo puede ser inmediata.

- **Calidad:**
  El método de trabajo y la revisión continua produce una mayor calidad del software.

- **Alineamiento:**
  Cada persona sabe que es lo que tiene que hacer y no es necesario estar reorganizando una y otra vez las tareas.

Desventajas[5]

- Demasiadas reuniones para poco avance, a veces es muy tedioso y estresante reunirse demasiadas veces por el mismo tema, algunos van perdiendo el interés en el proyecto.

- Si una persona renuncia o hay algún cambio es complicado reemplazar ese rol ya que es la persona que se lleva el conocimiento específico y afecta a todo el proyecto.

- Si una tarea no está bien definida, la estimación de costes y el tiempo de proyecto no serán exactos, por lo que ésta podrá extenderse casi indefinidamente.

[4] https://blog.wearedrew.co/ventajas-y-desventajas-de-la-metodologia-scrum

[5] https://es.scribd.com/document/389922333/scrum-1-docx

- Cuando no se cuenta con el nivel de compromiso necesario por parte de los miembros del equipo, será complicado llevar el proyecto hasta su finalización exitosa.

- Si los miembros del equipo no cuentan con la experiencia suficiente es posible que el proyecto no se pueda completar a tiempo.

- Si una tarea no está bien definida, la estimación de costes y el tiempo de proyecto no serán exactos, por lo que ésta podrá extenderse casi indefinidamente.

2.1.2. XP

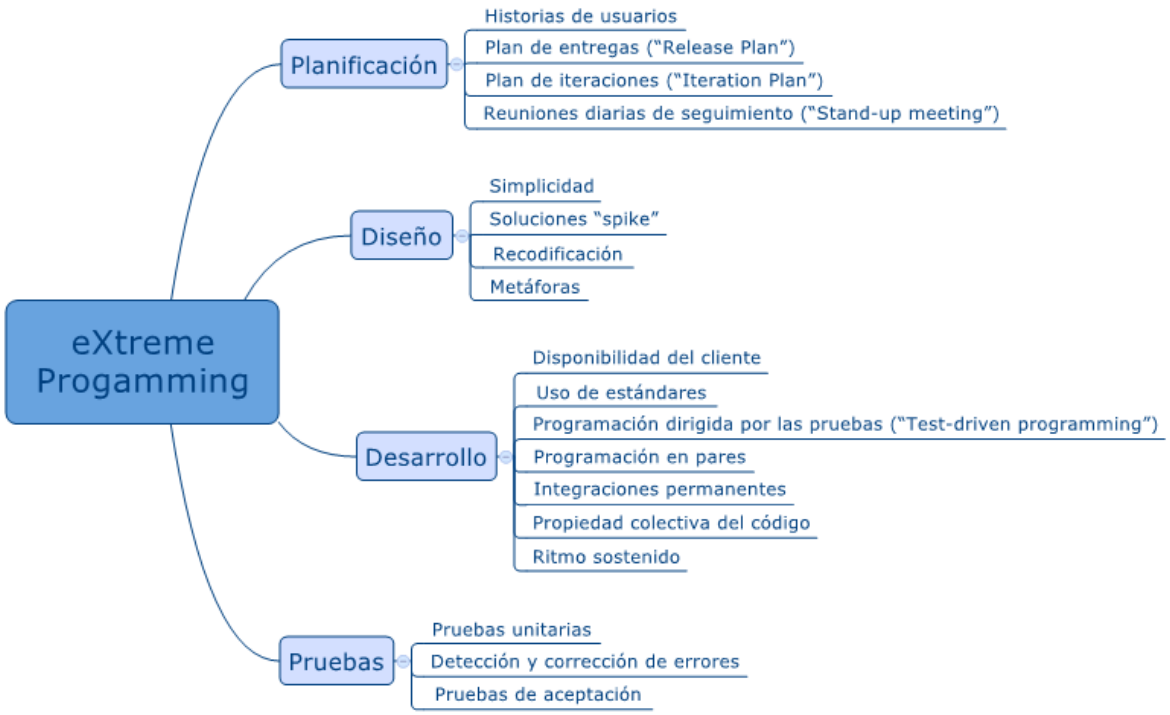


Figura 2.1. Etapas de la metodología XP

FASE 1: PLANIFICACIÓN

Según la identificación de las historias de usuario, se priorizan y se descomponen en tareas pequeñas. La planificación es revisada aproximadamente cada dos semanas de iteración.

FASE 2: DISEÑO

En este paso se intentará trabajar con un código sencillo, haciendo lo mínimo imprescindible para que funcione. Se obtendrá el prototipo. Además, para el diseño del software orientado a objetos, se crearán tarjetas CRC (Clase-Responsabilidad-Colaboración).

FASE 3: CODIFICACIÓN

La programación aquí se hace en parejas.

FASE 4: PRUEBAS

Se deben realizar pruebas automáticas continuamente. Al tratarse normalmente de proyectos a corto plazo, este testeo automatizado y constante es clave. Además, el propio cliente puede hacer pruebas, proponer nuevas pruebas e ir validando las tareas.

FASE 5: LANZAMIENTO

Si hemos llegado a este punto, significa que hemos probado todas las historias de usuario o tareas con éxito, ajustándose a los requerimientos del cliente. Tenemos un software útil y podemos incorporarlo en el producto.

Ventajas

- Se adapta al desarrollo de sistemas pequeños y grandes.
- Optimiza el tiempo de desarrollo
- Permite realizar el desarrollo del sistema en parejas para complementar los conocimientos.
- El código es sencillo y entendible, además de la poca documentación a elaborar para el desarrollo del sistema.

Desventajas

- Es recomendable emplearlo solo en proyectos a corto plazo.
- No se tiene la definición del costo y el tiempo de desarrollo.
- El sistema va creciendo después de cada entrega al cliente y nadie puede decir que el cliente no querrá una función más. Se necesita de la presencia constante del usuario, lo cual en la realidad es muy difícil de lograr.

2.2. ¿Por qué las elegimos?

Elegimos trabajar con estas metodologías por distintos motivos. En primer lugar, porque conocíamos y hemos trabajado con ambas ya sea en el ámbito laboral o en lo académico. Otros puntos a destacar en la elección son:

- La opción de contar con *sprints* con duraciones mayores a dos semanas (SCRUM).
- La posibilidad de realizar programación en parejas (XP).
- Poder realizar cambios sobre un *sprint* en particular (XP).

[6]El mayor beneficio de usar Scrum y XP juntos es que Scrum es un marco ágil y XP es una metodología de ingeniería ágil. Para utilizar las prácticas de gestión de Scrum y las prácticas de ingeniería de XP, estas dos metodologías se utilizan juntas en Scrum / XP híbrido. La ingeniería de XP se basa en el desarrollo basado en pruebas, el enfoque en pruebas automatizadas, programación de pares, diseño simple, refactorización, etc. Scrum gestiona el trabajo a realizar a través de su equipo, eventos y artefactos. La combinación de estos aspectos de ambas metodologías genera mejores resultados.

[6] https://masterofproject.com/blog/6362/scrum-xp-hybrid

2.3. Adaptación de las metodologías

Scrum y XP pueden complementarse. En Scrum el *Product Owner* decide “qué hay que hacer”, en cambio, en XP el cliente decide. En nuestro caso, al tener un producto propio y un equipo conformado por tres personas, dividimos los roles en *front-end* y *back-end*. No se asignó un líder o *scrum master* dentro del grupo, simplemente dividimos las tareas de desarrollo en base a los roles mencionados y realizamos en conjunto las relacionadas con el análisis y el diseño. Incorporamos prácticas XP referidas al desarrollo del Sistema, como las reuniones diarias dentro de *back-end* y el trabajo en pareja en ciertas ocasiones. Además de reuniones semanales entre *back-end* y *front-end* para sincronizarnos.

Prácticas utilizadas de ambas metodologías:

- [Scrum] El proyecto fue organizado en iteraciones o *sprints* que nos permitieron separar y poder entender al mismo como un conjunto de proyectos de menor tamaño. Esto nos facilitó la administración de las distintas funcionalidades que conforman al producto y pudimos así ir teniendo una solución cada vez más completa y que fue asemejándose gradualmente al objetivo final.
- [Scrum] Se elaboró y utilizó el *Product Backlog* (a través de GitHub) del proyecto para tener una mejor noción de cuales eran todas las funcionalidades y características requeridas por el equipo. Todo lo que suponía un trabajo que debía realizarse tenía que estar reflejado en el mismo.
- [Scrum] Se realizaron reuniones tanto de planificación, como así también de revisión y retrospectiva en cada uno de los *Sprints*.
- [XP] En algunas historias se tuvo que agrandar su alcance debido a la subestimación que se le dio en el plan del proyecto.
- [XP] Programación en pareja en el *back-end*.
- [XP] Integración continúa. Al trabajar con una arquitectura de microservicios (se describe con más detalle en la siguiente sección, *3. Arquitectura*), nos permitió un desarrollo independiente de cada servicio propuesto con lenguajes diferentes. Debido a esto, se implementó una integración continua con Travis-CI (explicado en el apartado de Herramientas) en un comienzo, que luego se continuó a través de GitHub Actions.
- [XP] Pruebas unitarias, encargadas de verificar el código y diseñada por los programadores, y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida diseñadas por el cliente final.

3. Arquitectura

En este apartado, se describe la arquitectura elegida para diseñar el sistema así como también las arquitecturas de las bases de datos y un detalle más preciso de cada una de las partes que componen técnicamente el producto.

3.1. Introducción

Se decidió utilizar una arquitectura basada en una colección de microservicios[7], en donde se utiliza el protocolo HTTP (*Hypertext Transfer Protocol*) para realizar la comunicación entre dichos componentes. Este estilo de estructura cuenta con las siguientes características:

- El sistema se divide en servicios donde cada uno abarca una capacidad del negocio finita. De esta forma, cada servicio tiene la responsabilidad de realizar las funcionalidades del negocio que le corresponde y no más.
- Los microservicios son pequeños, independientes y poco acoplados. Un único equipo pequeño de desarrolladores puede escribir y mantener un servicio.
- Los servicios se pueden implementar de forma independiente. Un equipo puede actualizar un servicio existente sin necesidad de reconstruir y volver a implementar la aplicación completa.
- Los servicios son responsables de conservar sus propios datos o estado externo.
- Los servicios se comunican entre sí mediante *APIs* bien definidas. Los detalles internos de implementación de cada servicio están ocultos de otros servicios.
- Los servicios no necesitan compartir la misma pila de tecnología o bibliotecas.
- Si un servicio falla, no se ve afectada el resto de la aplicación.
- Si el sistema presenta una carga elevada en alguna funcionalidad, el servicio encargado de la misma es replicable para poder atender un mayor número de peticiones.
- Cada servicio puede ser actualizado o reemplazado de forma independiente al resto.

[7] https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices

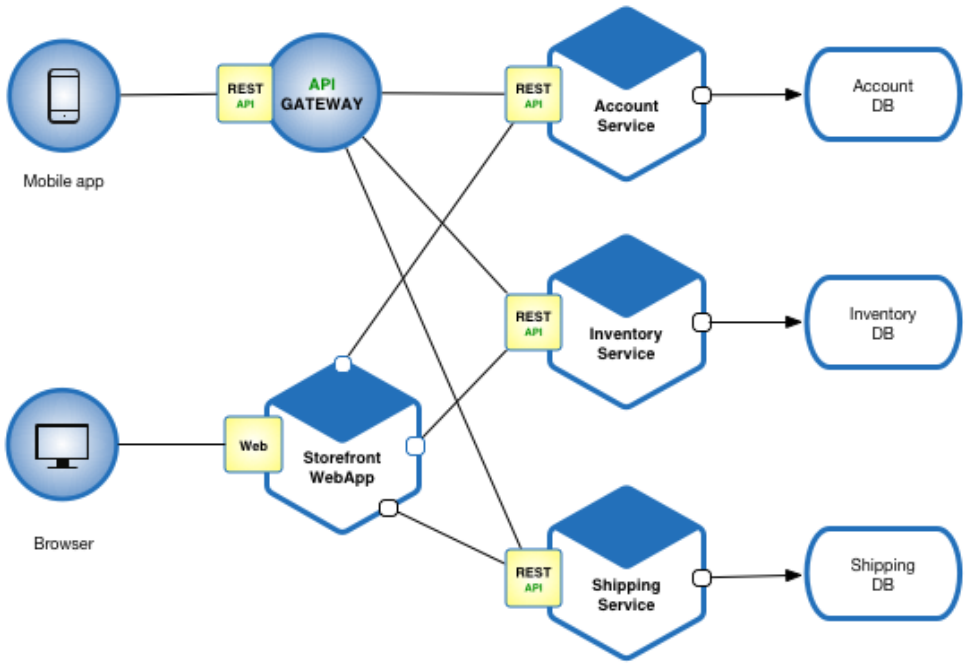


Figura 3.1. Ejemplo de arquitectura de microservicios

3.2. Diagrama de arquitectura

En esta sección, se detalla la arquitectura de la plataforma, donde se muestra el punto de entrada a la misma y cómo se comunican los diferentes servicios que la componen cuando el usuario utiliza las distintas funcionalidades.

Los usuarios interactúan con el sistema a través del servicio de *front-end* y es éste quién se comunicará con los servicios de *back-end*.

Dicha comunicación se realizará a través de un g*ateway*, quien actúa como intermediario entre los servicios *back-end* y el *front-end*.

Los servicios de *back-end*, cuando es necesario, también se comunican entre sí para poder obtener cierta información administrada por otro de los servicios. Estos componentes realizan diferentes operaciones sobre las distintas bases de datos del ecosistema (CRUD) y realizan subidas y descargas de imágenes que se almacenan en *Google Cloud Storage*.

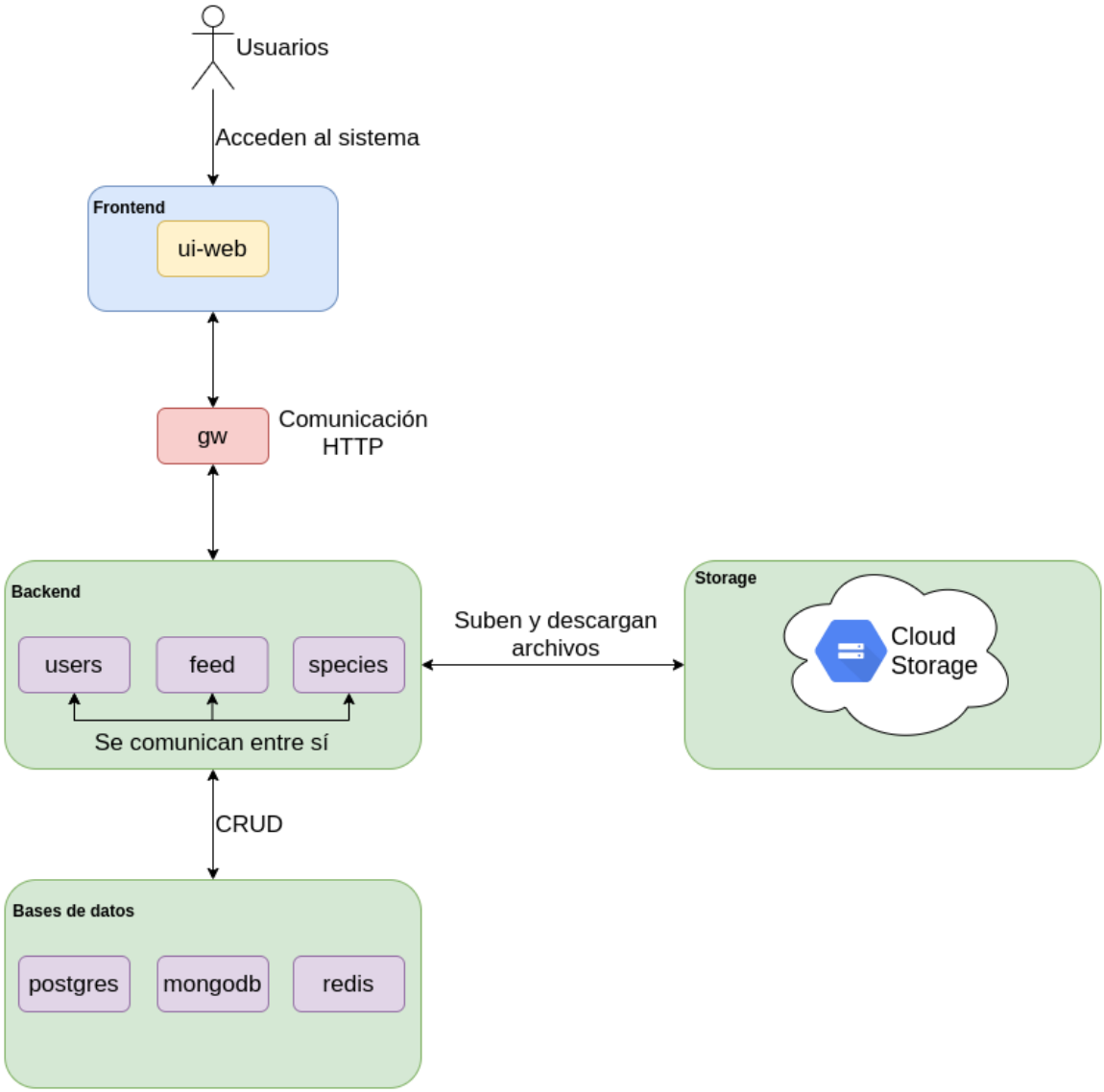


Figura 3.2. Diagrama de arquitectura

3.3. Diagramas de bases de datos

A continuación, se muestra con mayor detalle cómo es la arquitectura de las bases de datos utilizadas en la solución.

Para la base de datos relacional, se generó un diagrama de entidad-relación que muestra las tablas con sus atributos, tipos, relaciones así como las claves primarias y claves foráneas.

En el caso de la base de datos no relacional, se realizó un diagrama de colecciones donde se detalla cada colección con sus respectivos atributos, tipos y claves primarias.

3.3.1. Diagrama de Entidad-Relación

Figura 3.4. Diagrama de entidad-relación

3.3.2. Diagrama de colecciones

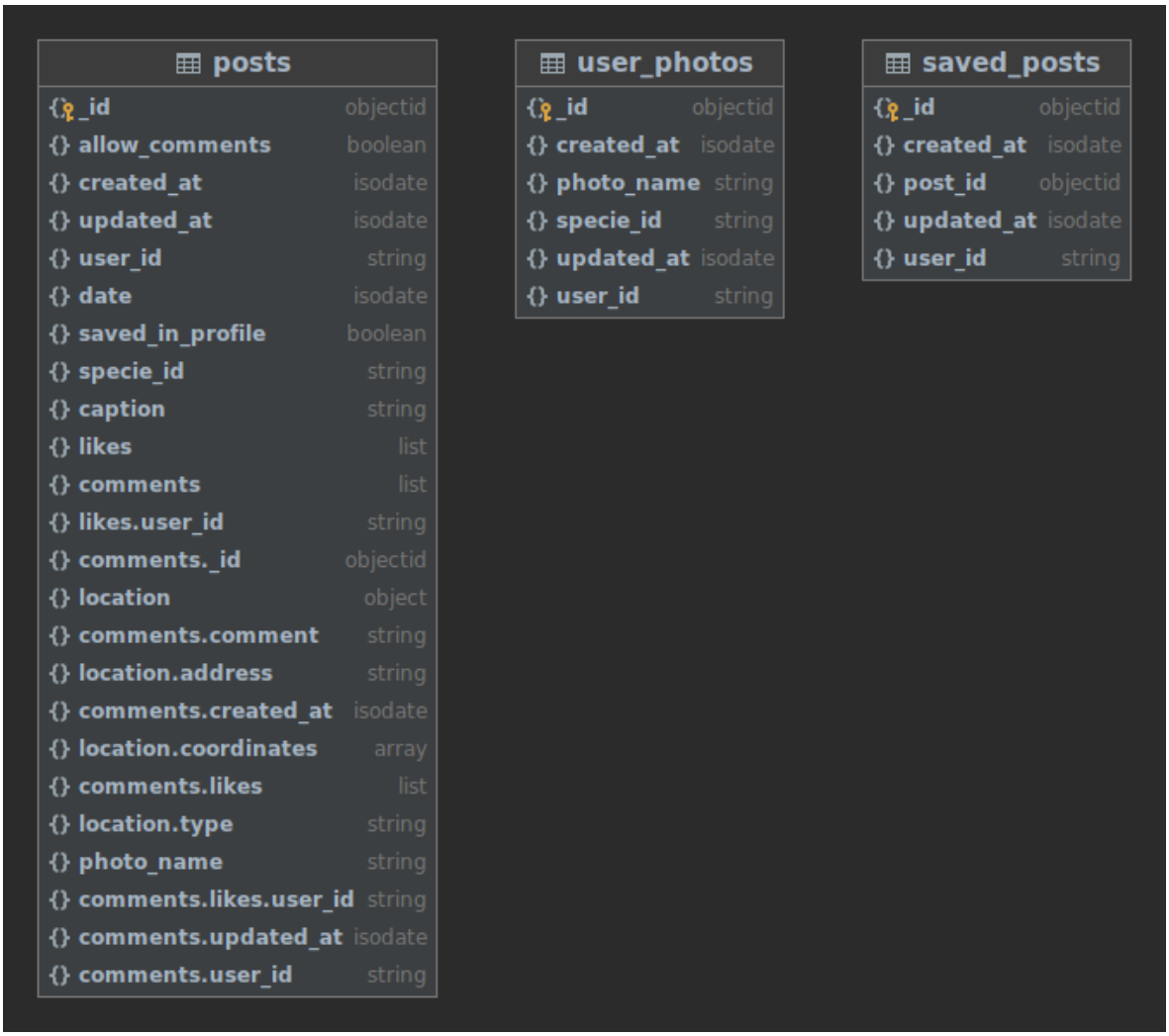


Figura 3.5. Diagrama de colecciones

3.4. Descripción de los servicios

Como se indicó anteriormente, el sistema está compuesto por un conjunto de microservicios. Dichos componentes, se pueden catalogar dentro de 3 grupos:

- *Front-end: ui-web.*
- *Back-end: baselib, feed, gw (gateway), species, users.*
- *Bases de Datos: mongo, postgres, redis.*

Dentro del grupo *front-end* encontraremos los servicios encargados de la parte visible del sistema. Esta área no trata directamente con bases de datos, servidores y todas las aplicaciones de *back-end* complejas, que explicaremos a continuación, pero aborda la usabilidad, los efectos visuales y la velocidad de carga, entre otros detalles.

En contrapartida, los servicios incluidos en el *back-end*, son aquellos con los que el usuario no tiene contacto directo, pero que contienen toda la lógica del negocio y quienes implementan las funcionalidades del sistema.

Por último, encontraremos las *Bases de Datos*, quienes se encargan de almacenar los diferentes datos requeridos para el correcto funcionamiento de la aplicación.

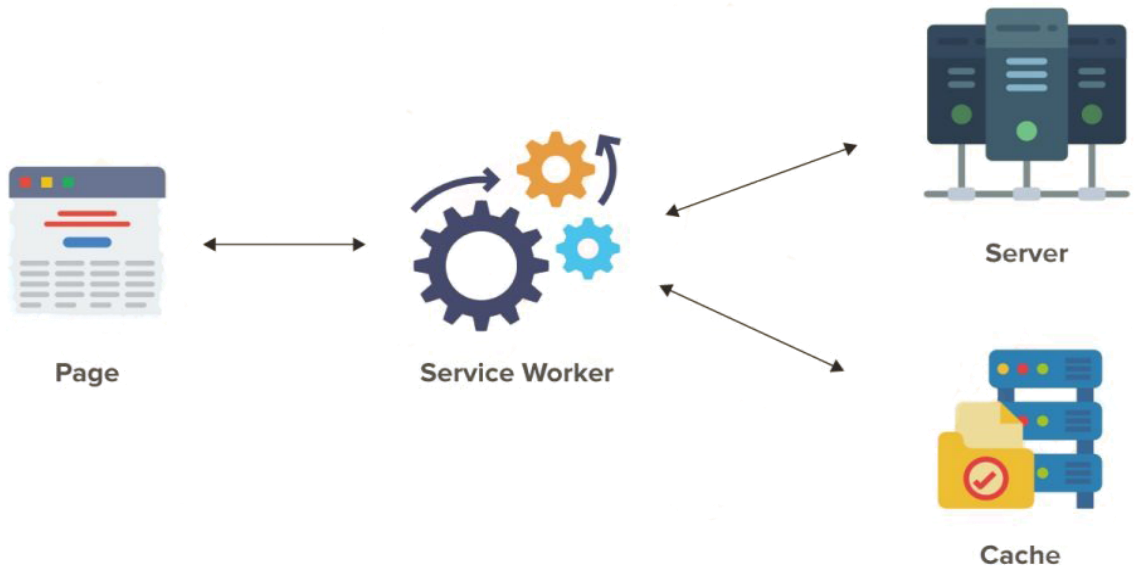
3.4.1. UI-Web

El servicio ***ui-web*** es el encargado de mostrar la interfaz gráfica y a través de la cual el usuario tendrá interacción con el sistema.

Es una PWA (*Progressive Web Application*) que ofrece, principalmente, las funcionalidades de visualización de los tipos de especies, las publicaciones de otros usuarios, el listado de negocios disponibles en el área seleccionada, así como también un perfil por cada usuario registrado. Permite la visualización de información e interacción entre usuarios pudiendo éstos no tener que depender de la conectividad a internet dada su naturaleza como una PWA. Es decir, se logra tener la posibilidad de ofrecer servicios aun cuando el usuario se encuentra sin conexión. Esto es de suma importancia ya que no suele haber conectividad en los lugares pesqueros.

Cabe aclarar que este servicio funcionará como una PWA siempre y cuando el usuario quiera utilizar la aplicación desde un navegador, es decir, como si fuera una página web convencional. Para aquellos usuarios que decidan instalar la aplicación en sus dispositivos móviles, este servicio actuará como una aplicación híbrida.

Figura 4.1. Arquitectura de una PWA



Funcionamiento de una PWA

Una PWA se diferencia de una página web por tres cosas: el registro de un *service worker*, la incorporación de un manifiesto y la utilización obligatoria de HTTPS.

El actor principal de este esquema (Figura 4.1) es el *service worker* ya que es quien nos va a permitir, por cada pedido de información, manejar el caché del navegador. Como se puede observar, el *service worker* va a actuar como intermediario entre nuestra página web y el servidor con el objetivo de, primero, actualizar los datos almacenados en el caché mediante una estrategia de caché y, en segundo lugar, ante la eventual pérdida de conexión va a poder asumir la responsabilidad de retornar al cliente (página web) los datos almacenados en el cache sin que éste perciba que su pedido de información no fue exitoso debido a la falta de conexión.

Al navegar hacia una PWA, ésta va a tratar registrar su *service worker*. En caso de que el registro sea exitoso, el *service worker* va a almacenar en caché la mínima cantidad de archivos necesarios para que la PWA pueda usarse. Esto se conoce como *precache*. Es decir, que el usuario en su primer contacto con una PWA logra tener en caché los archivos necesarios para poder navegar dentro de la aplicación independientemente de su conexión con internet. Y, por otro lado, tal como se comentó anteriormente, el *service worker* registrado va a almacenar y actualizar todos los pedidos de información que el usuario solicite para luego ser capaz de seguir brindando esa información aún si el usuario no se encuentra conectado a Internet.

Podemos visualizar al *service worker* como un proceso en segundo plano que va a vivir en el contexto del navegador en el cual se encuentra el usuario. No es un proceso cuya ejecución sea continua sino que su ciclo de vida es bastante corto. Una vez registrado, el navegador va a ser el encargado de despertarlo y dormirlo de acuerdo a cómo el flujo de solicitudes ocurra. Por ejemplo, si el usuario decide solicitar la información de una especie en particular es el mismo navegador quién va a despertar al *service worker* para que maneje esa transacción y luego dormirlo hasta la próxima solicitud.

Una PWA es una página web con el agregado de la utilización de un *service worker* y es por esto que el acceso a la misma es independiente del dispositivo que se esté utilizando. Las *PWAs* ofrecen un conjunto de funcionalidades adicionales a la independencia de la conectividad. Estas funcionalidades quedan por fuera del alcance de esta descripción dado que no se utilizan en el proyecto pero sí cabe resaltar que, actualmente, las *PWAs* poseen algunos problemas de compatibilidad con el mundo IOS en algunas de estas funcionalidades. Esto no genera inconvenientes dado que sólo estaríamos utilizando el *service worker* para el almacenamiento y actualización del caché.

Funcionamiento de una aplicación híbrida

Desde la perspectiva del usuario final, las aplicaciones híbridas no presentan diferencias frente al conjunto del resto de aplicaciones nativas. Es decir, el usuario va a poder descargar la aplicación desde un *play store* y utilizarla tal y como usa el resto de sus aplicaciones.

La gran diferencia se presenta en cómo esta aplicación está construida. Una aplicación híbrida es una página web a la cual se le proporciona una capa o carcasa nativa.

Ésta capa logra camuflar el contenido web y, por lo tanto, a vistas del sistema operativo también se trata de una aplicación como cualquier otra que se encuentre instalada. La gran ventaja de este enfoque de desarrollo es que se logra obtener, con un mismo proceso de programación, dos productos destinados a plataformas distintas. Si la aplicación necesita comunicarse con el sistema operativo para, por ejemplo, almacenar en el dispositivo algún tipo de información lo hará a través de esta carcasa nativa que se le proporcionó. De cualquier otra forma la comunicación entre el sistema operativo y la aplicación no sería posible debido a que el primero sólo entiende el lenguaje en el cual fue programado y no el lenguaje web.

3.4.2. Baselib

La librería base, o ***baselib***, no es un servicio en sí sino que se trata de un módulo desarrollado por nosotros, que contiene funciones y definiciones que son genéricas a todos los microservicios que conforman la aplicación.

La ventaja que nos trae tener este módulo es la reutilización de dichas funciones, lo que evita la repetición de código en cada uno de los servicios.

3.4.3. Feed

El servicio ***feed*** implementa toda la lógica de las publicaciones del *feed*, valga la redundancia, del sistema. Sus funciones son:

- Crear, obtener, actualizar y eliminar publicaciones.
- Crear, obtener y eliminar comentarios en las publicaciones.
- Agregar o quitar un *like* a una publicación o comentario.
- Guardar o quitar de elementos guardados una publicación.

3.4.4. KrakenD[8] - API Gateway

KrakenD es un *API Gateway* de código abierto de alta performance.

Se encarga de llevar a cabo la comunicación de nuestros servicios sin la necesidad de tener que programar uno particular, simplemente con especificar un archivo de configuración con los diferentes puntos de entrada a cada servicio. Principalmente se encarga de redirigir los *requests* HTTP que envía el usuario a través del servicio ***ui-web*** hacia el servicio encargado de realizar la tarea necesaria para brindarle una respuesta al usuario.

3.4.5. Species

Este servicio tiene como funcionalidad la administración de las especies existentes en la aplicación. Su tarea es obtener una o varias especies de peces cargadas en el sistema.

[8] https://www.krakend.io/docs

3.4.6. Users

El servicio ***users*** es el encargado de proveer las funcionalidades relacionadas al alta, baja, modificación de las cuentas de usuario, al manejo de las sesiones de los usuarios en la aplicación, a la carga o eliminación de capturas de un usuario y a los negocios que administra un usuario. Permite:

- Registrarse.
- Iniciar sesión ya sea mediante un mail registrado, Facebook o Google.
- Iniciar sesión como invitado.
- Recuperar contraseña.
- Cerrar sesión.
- Subir una captura al perfil.
- Crear un negocio.

3.4.7. MongoDB

La función del servicio es administrar los datos relacionados a las fotos de los usuarios, a las publicaciones del *feed*, así como también las publicaciones guardadas de los usuarios.

Se eligió esta base de datos no relacional debido a que presenta una mejor performance principalmente en la lectura de los datos en comparación a ***postgres***. Con esta premisa, al ser las publicaciones o fotos elementos con una tasa de consultas mucho mayor al resto de los objetos de la app, consideramos conveniente incluir esta base de datos para la persistencia de publicaciones y fotos.

3.4.8. Postgres

El servicio ***postgres*** es el encargado de la persistencia general de los datos de los microservicios:

- Usuarios: datos del usuario, roles, elementos guardados, *reviews* realizadas.
- Negocios: datos del negocio, fotos, ubicación, contacto, *reviews*, tipo de negocio.
- Especies: datos de las diferentes especies.

Se optó por utilizar una base de datos relacional para atender la persistencia de estos datos ya que los mismos estaban relacionados entre sí y es necesario que sean consistentes.

3.4.9. Redis

Redis tiene como tarea la gestión de las sesiones de usuarios *logueados* en la aplicación.

Este servicio es ideal para el manejo de persistencia de las sesiones ya que todas las operaciones son realizadas en memoria por lo que la lectura y escritura son realmente rápidas, es por esto que se optó por el uso de ***redis*** para esta tarea.

4. Herramientas y tecnologías

En esta sección, se describen las herramientas y tecnologías utilizadas a lo largo del desarrollo del proyecto.

4.1. Generales

4.1.1. Google Cloud

Google Cloud es una suite que ofrece numerosos servicios basados en la nube implementados por Google. Es utilizada para crear soluciones a través de la tecnología almacenada en la nube y permite, por ejemplo, destacar la rapidez y la escalabilidad de su infraestructura en las aplicaciones del buscador.

Consiste en un conjunto de recursos físicos, como computadoras y unidades de disco duro, y recursos virtuales, como máquinas virtuales (VM), que se encuentran en los centros de datos de Google en todo el mundo.

En esta plataforma, creamos un proyecto de Google Cloud para poder hacer uso de los recursos que ofrece. Lógicamente, cada recurso es pago pero contamos con un crédito inicial de 300 USD para poder acceder a los mismos.

Durante el transcurso del proyecto, únicamente se hizo uso del recurso denominado Cloud Storage. Este recurso nos permite almacenar objetos de forma optimizada, escalable, durable y segura. Además nos permite tener acceso a los objetos almacenados en cualquier lugar de la aplicación donde sea requerido. En Cloud Storage, almacenamos las imágenes pertenecientes a:

- Publicaciones en la red social.
- Capturas de un usuario.
- Imágenes de negocios.

Se optó por utilizar este servicio en la nube para el almacenamiento de las imágenes debido a que se comporta como un sistema de archivos, que es una forma más eficiente de realizar tareas de lectura/escritura de este tipo de objetos que en una base de datos relacional. Además, brinda las ventajas mencionadas con anterioridad al ser un servicio basado en la nube y accesible desde cualquier lugar.

4.1.2. YAML

YAML es un lenguaje de serialización de datos diseñado para ser amigable con los humanos y funciona bien con lenguajes de programación modernos para tareas cotidianas comunes.

Existen innumerables tipos de estructuras de datos, pero todas pueden representarse adecuadamente con tres primitivas básicas: mapeos (hashes / diccionarios), secuencias (matrices / listas) y escalares (cadenas / números). YAML

aprovecha estas primitivas y agrega un sistema de mecanografía simple y un mecanismo de alias para formar un lenguaje completo para serializar cualquier estructura de datos nativa.

En este proyecto se utiliza este lenguaje principalmente para implementar archivos de configuración como lo son *docker-compose* para definir todos los servicios que componen el sistema y el *workflow* de *GitHub* para implementar la integración continua de los diferentes proyectos.

4.1.3. Docker[9] y docker-compose[10]

El sistema está basado en su totalidad sobre una arquitectura de microservicios. Para cada uno de dichos servicios, se crea una imagen Docker a partir del código fuente, que luego es ejecutada dentro de un contenedor Docker. Cada uno de los diferentes contenedores son definidos y configurados en *docker-compose*, que permite iniciar toda la plataforma con un único comando.

Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos. Docker permite que "contenedores" independientes se ejecuten dentro de una sola instancia de Linux, evitando la sobrecarga de iniciar y mantener máquinas virtuales.

Un contenedor es una unidad de software estándar (un proceso) que empaqueta el código y todas sus dependencias para que la aplicación se ejecute de manera rápida y confiable de un entorno informático a otro. Una imagen de contenedor de Docker es un paquete de software ligero, independiente y ejecutable que incluye todo lo necesario para ejecutar una aplicación: código, tiempo de ejecución, herramientas del sistema, bibliotecas del sistema y configuraciones. Esta imagen se convierte en un contenedor cuando se ejecuta en el motor Docker. Los contenedores aíslan el software de su entorno y aseguran que funcione de manera uniforme a pesar de las diferencias, por ejemplo, entre el desarrollo y la puesta en escena.

Los contenedores y las máquinas virtuales tienen beneficios similares de aislamiento y asignación de recursos, pero funcionan de manera diferente porque los contenedores virtualizan el sistema operativo en lugar del hardware. Un contenedor se ejecuta de forma nativa en Linux y comparte el núcleo de la máquina host con otros contenedores. Ejecuta un proceso discreto, no ocupa más memoria que cualquier otro ejecutable, lo que lo hace liviano (figura 1.5). Los contenedores son más portátiles y eficientes. Por otro lado, una máquina virtual (VM) ejecuta un sistema operativo "invitado" completo con acceso virtual a los recursos del host a través de un hipervisor (figura 1.6). En general, las máquinas virtuales incurren en una sobrecarga más allá de lo que consume la lógica de su aplicación.

[9] https://docs.docker.com/
[10] https://docs.docker.com/compose/

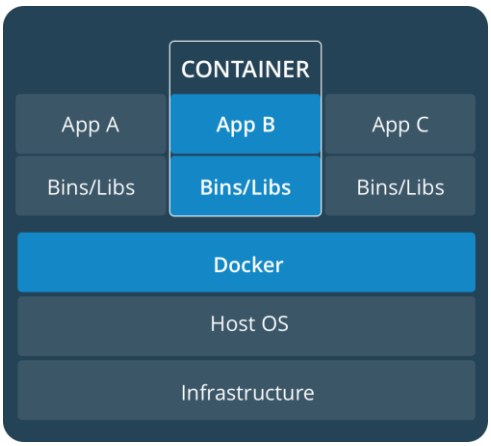


Figura 4.1. Arquitectura de un contenedor

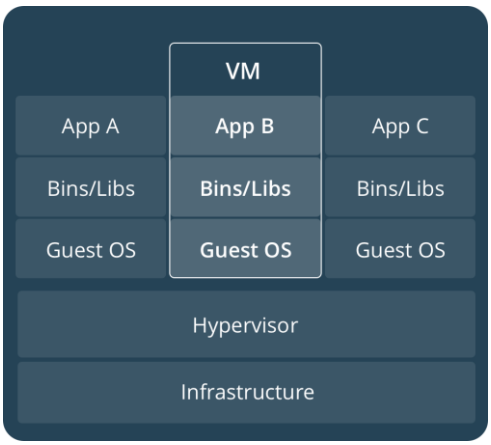


Figura 4.2. Arquitectura de una VM

Docker-compose es una herramienta para definir y ejecutar aplicaciones que se conforman de múltiples contenedores Docker. Para esto, se utiliza un archivo YAML para configurar los servicios de su aplicación. Luego, con un solo comando, crea e inicia todos los servicios desde su configuración.

4.1.4. DataGrip

DataGrip es un entorno de base de datos multimotor. También es desarrollado por *JetBrains* y contamos con una cuenta educativa gracias a la Facultad.

El IDE nos permite tener acceso a diferentes motores de bases de datos en un solo lugar, lo que nos brinda la posibilidad de implementar y depurar todo lo relacionado con la persistencia del sistema en un solo lugar.

4.1.5. Postman

Postman es una plataforma de colaboración para el desarrollo de *APIs*.

En nuestro proyecto, hacemos uso de esta herramienta para realizar consultas y operaciones a los distintos *endpoints* de nuestra aplicación. Podemos enviar *requests* HTTP con diferentes parámetros y datos para realizar pruebas del sistema.

4.1.6. Google Meet

Meet es una aplicación de videollamadas, la utilizamos diariamente durante el desarrollo de la aplicación para la comunicación con el equipo.

4.2. Back-end

4.2.1. GO

El lenguaje de programación de código abierto *Go*[11], también conocido como *Golang*, fue el que se utilizó en su totalidad para desarrollar los diferentes servicios de la plataforma. *Go* es un lenguaje diseñado por Google cuyas principales características son:

- Sintaxis similar a la del lenguaje C.
- Tipado estático: la comprobación de tipificación se realiza durante la compilación, y no durante la ejecución.
- Visualmente es fácil de entender.
- Cuenta con un recolector de basura propio.
- A pesar de ser semejante a un lenguaje funcional, admite el paradigma orientado a objetos.
- Es un lenguaje compilado y concurrente.
- Fácil declaración de variables.
- No utiliza el manejo de excepciones: un error grave en la aplicación implica la caída de la misma.

4.2.2. GoLand

GoLand[12], creado por *JetBrains*, es un IDE multiplataforma especialmente integrado para desarrolladores de *Go*. Contamos con una licencia educativa, provista por la Facultad, que nos permite acceder a este entorno y a sus poderosas herramientas.

Entre las diversas funcionalidades que el IDE ofrece, encontramos:

- Cuenta con sistemas de detección de errores y sugerencias para soluciones, refactorizaciones, finalización de código inteligente, detección de código muerto y consejos de documentación que nos ayudan a crear código de forma rápida, eficiente y fiable.
- Permite desplazarnos a lo largo del código, ya sea propio o externo, de forma rápida para explorarlo y entenderlo.
- Es compatible con GitHub, lo que nos permitirá estar siempre sincronizados a la hora del control de versiones de los servicios y del sistema en general.
- Nos da la posibilidad de instalar un gran número de extensiones para mejorar nuestra tarea de desarrollo.

[11] https://golang.org/
[12] https://www.jetbrains.com/es-es/go/

4.3. Front-end

4.3.1. Quasar

Quasar es un *framework* basado en *Vue.js*[13] que permite al desarrollador web poder programar y compilar su aplicación apuntando a distintos entornos. La principal ventaja que presenta este *framework* es que abstrae en cierta forma al desarrollador de toda la configuración necesaria para poder hacer la conversión correspondiente. Por otro lado, ofrece integraciones con distintas librerías y herramientas que logran enriquecer el proceso de desarrollo. Por ejemplo, presenta integración con herramientas de *testing* como *Cypress* o *Jest*, como así también, *Workbox* para el manejo del caché y de todo el contenido offline.

4.3.2. Ionic Capacitor

Motor híbrido desarrollado por *Ionic* y que, a su vez, presenta integración con *Quasar*. Capacitor ofrece una capa nativa que termina envolviendo todo el código web. De esta forma, se logra obtener una aplicación web pero que aparenta ser una aplicación móvil debido a la capa nativa otorgada por *Capacitor*. También, proporciona un conjunto de *APIs* que permiten comunicar la aplicación web con funcionalidades del dispositivo móvil. Dentro de las que utilizamos se encuentran principalmente las *APIs* para acceder a la cámara y la ubicación del dispositivo.

4.4. Persistencia

4.4.1. PostgreSQL

PostgreSQL es un potente sistema de base de datos relacional de código abierto. Cuenta con numerosas características destinadas a ayudar a los desarrolladores a crear aplicaciones, a los administradores a proteger la integridad de los datos y a crear entornos tolerantes a fallas, y a ayudarlo a administrar sus datos sin importar cuán grande o pequeño sea el conjunto de datos.

El uso que se le da en este proyecto es la persistencia permanente de los datos referidos a diversos componentes que conforman la aplicación: usuarios, especies de pescados, negocios, cabañas, etc. y de los que se requiere tener alta consistencia y robustez.

4.4.2. Redis

Redis es una base de datos de tipo clave-valor en memoria de código abierto, que se utiliza como agente de base de datos, caché y mensaje. Ofrece tiempos de respuesta inferiores al milisegundo lo que la hace extremadamente eficiente para realizar numerosas operaciones de consulta o escritura.

[13] https://router.vuejs.org/guide

El uso principal que se le da a ***redis*** es el guardado de los tokens de las sesiones de los usuarios. Los motivos por los cuales se optó por elegir esta tecnología para el manejo de sesiones son:

- Alta Performance: Al ser una base de datos en memoria, las operaciones tradicionales (GET, SET, DELETE) se realizan de forma rápida. Esto nos permite realizar la gestión de las sesiones sin generar impacto en el funcionamiento de la aplicación.
- *Whitelist*: Usamos ***redis*** como *whitelist*, es decir, guardamos en la base de datos de este aquellas sesiones que están "activas" y, por lo tanto, tienen acceso a los *endpoints* protegidos del entorno. Todo aquel que no tenga una sesión activa en ***redis*** no está permitido a acceder a dichos *endpoints*.
- Fácil y rápida implementación.

4.4.3. MongoDB

MongoDB[14] es una base de datos distribuida, basada en documentos, lo que significa que almacena datos en forma de documentos tipo JSON. El modelo de documentos de MongoDB resulta muy fácil de aprender y usar, y nos proporciona todas las funcionalidades que necesitamos para satisfacer los requisitos más complejos a cualquier escala. Además, cuenta con un driver oficial para el lenguaje utilizado, *Go*.

Esta base de datos se utiliza para el almacenamiento de datos que son consultados en un número mayor de ocasiones que los que se encuentran en ***postgres***. Estos datos son los pertenecientes a: información sobre las fotos, información sobre las publicaciones y mensajería.

Se optó por utilizar este sistema basado en documentos ya que tiene una mejor performance a la hora de realizar grandes cantidades de consultas a los datos y, al ser en formato JSON, nos resulta más manejable a nosotros como desarrolladores.

4.5. Testing

Go viene con soporte integrado para las pruebas unitarias, el cual facilita las pruebas sobre la marcha. Específicamente, usando convenciones de nomenclatura, con el paquete de *testing* de *Go* y el comando *go test*, se pueden escribir y ejecutar pruebas rápidamente. Para las bases de datos, se usaron diversas librerías para *mockear* las mismas.

A continuación, se dejan las referencias a las mismas:

- Para ***postgres*** se usó ***go-sqlmock***[15]: esta librería permite simular el driver de cualquier proveedor de bases de datos SQL sin necesidad de una

[14] https://docs.mongodb.com/
[15] https://github.com/DATA-DOG/go-sqlmock

conexión real a la base de datos. Con ella se puede *mockear* las llamadas a ***postgres*** retornando las filas que se requieran.

- Para ***redis*** se usó ***miniredis***[16] y ***redismock***[17]: La primera implementa parte del servidor de ***redis*** y, la segunda, permite *mockear* las diferentes respuestas a las consultas de ***redis***.
- Para las *assertions* se utilizó ***testify***[18].

4.6. Control de Versiones

4.6.1. Git

Git es un sistema de control de versiones distribuido gratuito y de código abierto diseñado para manejar todo, desde proyectos pequeños hasta muy grandes, con velocidad y eficiencia. Permite el mantenimiento de las diferentes versiones de los servicios que conforman la plataforma y la coordinación del trabajo entre los integrantes del equipo de desarrollo.

4.7. Gestión de Proyecto

4.7.1. GitHub

GitHub es un servicio de alojamiento de repositorios Git y agrega más características. Proporciona control de acceso y varias funciones de colaboración, como wikis y herramientas básicas de administración de tareas para cada proyecto.

En GitHub contamos con repositorios que contienen el código fuente de cada servicio implementado. Además de aprovechar el potencial de Git en el control de versiones, cuenta con diferentes funcionalidades de las que hacemos uso para llevar adelante la gestión del proyecto:

- Podemos crear *issues* en cada uno de los proyectos de los diferentes servicios con las diferentes funcionalidades a implementar o problemas conocidos a corregir, lo que permite tener un registro de todo el trabajo realizado y no olvidarnos de las tareas que quedan por hacer.
- Nos permite definir *boards* donde se listan todos los *issues* abiertos por hacer, en los que se está trabajando y aquellos que ya fueron resueltos.

4.7.2. Docker Hub

Docker Hub es un servicio proporcionado por Docker para buscar y compartir imágenes Docker con el equipo. Proporciona las siguientes características principales:

- Repositorios: permite subir y descargar imágenes Docker (*push* y *pull*).

[16] https://github.com/alicebob/miniredis
[17] https://github.com/go-redis/redismock
[18] https://github.com/stretchr/testify

- Equipos y organizaciones: podemos administrar el acceso a repositorios privados de imágenes Docker.
- Imágenes oficiales: permite descargar y usar imágenes Docker de alta calidad proporcionadas por Docker.
- *Builds*: crear automáticamente imágenes Docker desde GitHub y subirlas a Docker Hub.

4.7.3. Travis-CI / GitHub Actions

Travis-CI es un sistema de Integración Continua, gratuita para proyectos *Open Source* y de pago para proyectos privados. Se integra sin problemas con GitHub y automáticamente ejecuta el pipeline definido en cada *push* o *pull requests*.

Con Travis-CI, definimos una Integración Continua en cada repositorio de GitHub en donde, cada vez que se realiza un *merge* a la rama máster del repositorio, se lanza un pipeline que crea la imagen Docker del servicio en cuestión y la misma es subida a nuestro repositorio de imágenes en Docker Hub, todo de forma automática.

A mediados del desarrollo del proyecto, nos encontramos con problemas al utilizar esta plataforma para realizar la integración continua de nuestros servicios por lo que optamos por dejar de utilizarla y llevar las tareas que se realizaban aquí a GitHub *(GitHub Actions*).

4.8. Sistema Operativo

4.8.1. Ubuntu

Ubuntu es un sistema operativo de software libre y código abierto. Es una distribución de Linux basada en Debian.

Se utilizó este sistema operativo para el desarrollo *back-end* de la aplicación. Se optó por utilizar Ubuntu debido a que es el sistema operativo que utilizamos en nuestras tareas laborales y presenta numerosas ventajas a la hora de programar como ser muy personalizable y contar con un administrador de paquetes robusto y fácil de utilizar.

4.8.2. Windows

Se utilizó para desarrollar el *front-end*. No hay una justificación en particular por la elección más que era el sistema operativo utilizado por el encargado de desarrollar la interfaz gráfica.

4.9. Fundamentación

4.9.1. Back-end

Las herramientas mencionadas y explicadas fueron seleccionadas principalmente porque consideramos que eran las que mejor se ajustaban al desarrollo e

implementación de un sistema basado en microservicios. Para el desarrollo del *back-end*, se optó por utilizar *Go* debido a que se ajusta realmente bien a las características del proyecto, es ligero, rápido, permite utilizar concurrencia de una forma relativamente simple y tiene un gran soporte para desarrollo de *APIs*. Estas características hacen que este lenguaje sobresalga por otros como Java o Node, más aún cuando se desarrolla cada servicio para correr en un contenedor Docker.

En su mayoría, eran herramientas que ya conocíamos ya sea por la facultad, como el caso de Postgres, GitHub, Git, o bien por el ámbito laboral, sea Go, Docker, docker-compose, Docker Hub, Redis, Google Meets. En este último punto, cabe aclarar que otro factor en la elección de estas herramientas y tecnologías es que consideramos que nos iba a servir utilizarlas para poder ganar experiencia en las mismas de cara a nuestro conocimiento laboral.

El resto, surgieron en base a investigación realizada por el equipo y son las que se optaron por ser las que se consideraron más óptimas y que mejor se ajustaban a las exigencias del desarrollo del proyecto.

4.9.2. Front-end

En el caso del *front-end*, se optó por *Quasar* como *framework* de desarrollo dado que, en primera instancia, está basado en Vue.js que es un *framework* conocido por el equipo debido a cuestiones laborales. Por otro lado, ofrece la posibilidad de compilar el código web en distintos modos (Híbrido, PWA, SPA, entre otros) sin tener que realizar una gran configuración ya que es el mismo *framework* quien toma la responsabilidad de hacer la conversión. Es decir que, teniendo en cuenta la naturaleza de nuestro proyecto, podríamos programar y generar tanto una web como una aplicación híbrida al mismo tiempo. Y, por último, el mismo *framework* ofrece un listado de componentes ya preparados para funcionar en cualquiera de los entornos mencionados. En nuestro caso, nos interesaba que los componentes sean funcionales tanto en un entorno web como así también dentro de un entorno híbrido para luego poder utilizar el sistema como si fuera una aplicación más de un dispositivo móvil.

5. Desarrollo de la solución

En esta sección se explican aspectos relacionados a la ejecución del proyecto: requerimientos del sistema, historias de usuarios, estimaciones, separación del desarrollo de las historias en iteraciones. Como se comentó en la sección *2.* Metodologías, para poder llevar a cabo el desarrollo del proyecto se utilizaron diferentes prácticas de las metodologías Scrum y XP.

5.1. Consideraciones

Para realizar las estimaciones de los tiempos que conllevarían cada una de las etapas se tomaron las siguientes unidades de medida:

- Día de trabajo: 2 horas destinadas al proyecto.
- Semana de trabajo: 5 días a la semana, lo que equivale a 10 horas semanales.

Otra consideración es que durante el transcurso del desarrollo de la solución no se utilizaron las conocidas *Tasks* de Scrum para llevar un registro ordenado de las tareas que se fueron realizando. En su lugar, se decidió por utilizar *issues* y *pull requests* en GitHub donde, por cada historia de usuario, se abrían diferentes *issues* con las tareas que debían ser realizadas para completar dicha historia y, a su vez, para llevar un control del código fuente del sistema, se realizaban *pull requests* una vez terminado el *issue* (la tarea) para añadirla al resto del código fuente de la aplicación.

5.2. Desarrollo

5.2.1. Análisis

Al no contar con un cliente, se realizaron numerosas reuniones con los integrantes del equipo partiendo de la idea base del desarrollo de la aplicación en cuestión.

En dichas reuniones se fueron ideando y detallando los diferentes módulos del sistema y, a partir de estos, se fueron identificando los primeros requerimientos funcionales y no funcionales y se especificaron las diferentes prioridades que tenían cada uno de dichos requerimientos.

Una vez definido esto, se comenzaron a diseñar las historias de usuario, estimando la importancia y la complejidad de cada una. Para cada historia, se estimó el esfuerzo y los recursos requeridos para su implementación y se listaron en base a su prioridad.

El tiempo requerido para el análisis del sistema fue de 24 horas netas de trabajo que, tomando como referencia lo definido en el apartado Consideraciones, nos da un plazo de 12 días de trabajo, es decir, 2 semanas y 2 días de trabajo.

Esta labor se fue realizando gradualmente debido a temas de cursado, exámenes y trabajo. Se comenzó esta etapa el 16 de agosto de 2019 y se culminó el 30 de enero de 2020. Cabe aclarar que el plan de proyecto se entregó en la última fecha nombrada debido a correcciones que se tuvo que realizar en el mismo.

5.2.2. Iteraciones

Una vez culminada la etapa de análisis, comenzamos a realizar el desarrollo de la aplicación teniendo en cuenta los lineamientos plasmados en la planificación del proyecto.

En cada iteración, se fueron implementando las diferentes historias que correspondían a cada una en base a su prioridad y, una vez desarrollada la historia en cuestión, se realizaban pruebas tanto de la tarea particular como de la integración de la misma con el resto del sistema.

A su vez, en la duración real de cada historia se contemplaron aquellas tareas de mejoras o correcciones de errores que fueron surgiendo en el momento. Estas tareas se indicaron en los correspondientes diagramas de Gantt bajo el concepto de “Mejoras - corrección de errores”.

A continuación, se detalla el proceso de desarrollo a lo largo de las diferentes iteraciones:

Iteración 0

En esta iteración nos encargamos de todos los preparativos previos a comenzar el desarrollo de la solución, en el que encontramos tareas como:

- Dejar listos los entornos de desarrollo.
- Trabajar en el *product backlog*.
- Estudiar la arquitectura de microservicios.
- Realizar los mockups.
- Capacitarnos en las tecnologías a utilizar.

Iteración 1

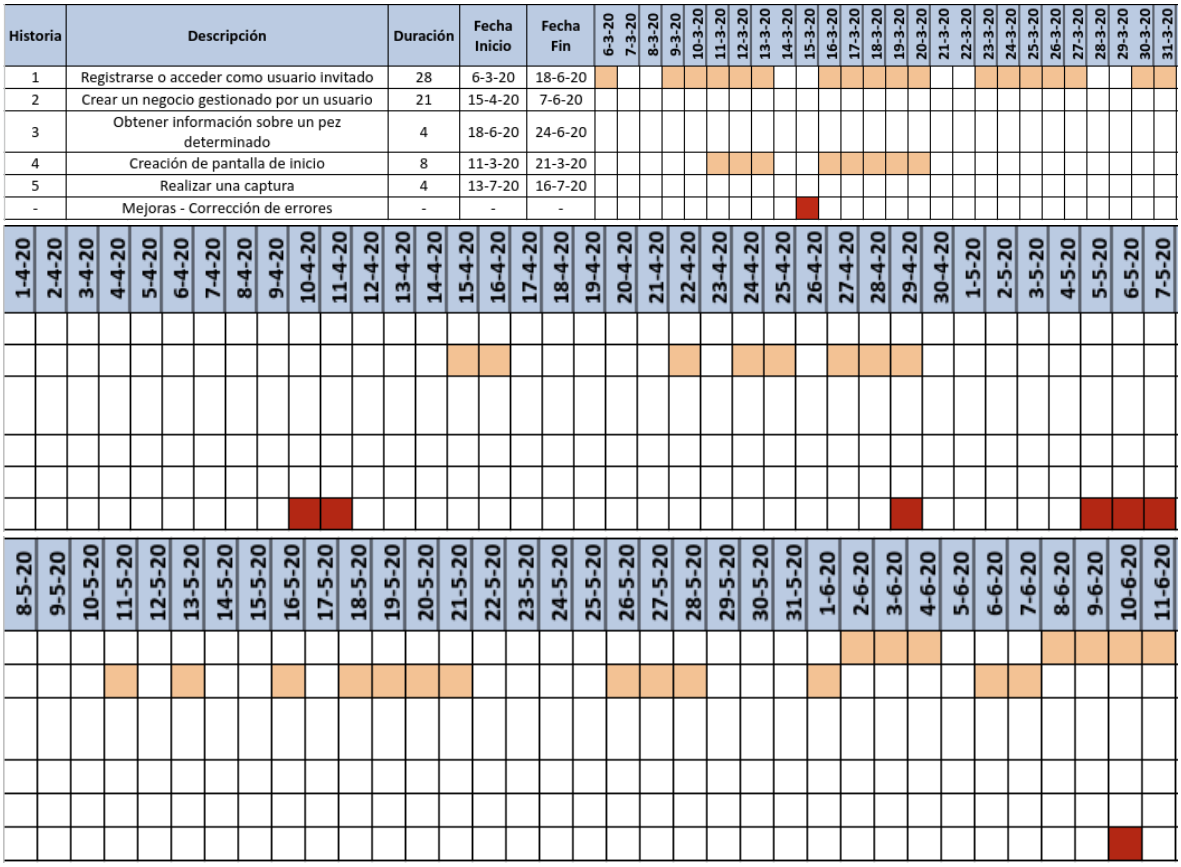
Las historias de usuario que se decidieron implementar en la primera iteración son:

- 1 - Registrarse o acceder como “usuario invitado”.
- 4 - Crear un negocio gestionado por un usuario.
- 8 - Obtener información sobre un pez determinado.
- 16 - Creación pantalla de inicio.
- 22 - Realizar una captura.

Detalle de las historias mencionadas:

Historia	Prioridad	Esfuerzo estimado (*Story Points*)	Esfuerzo estimado (Horas)	Esfuerzo real (*Story Points*)	Esfuerzo real (Horas)
Registrarse o acceder como usuario invitado	ALTA	2	8	6	24
Crear un negocio gestionado por un usuario	ALTA	2	8	5	20
Obtener información sobre un pez determinado	ALTA	1	4	1	4
Creación pantalla de inicio	ALTA	1	4	2	8
Realizar una captura	ALTA	3	12	1	4
TOTAL		9	36	15	60

Duración real de la iteración:



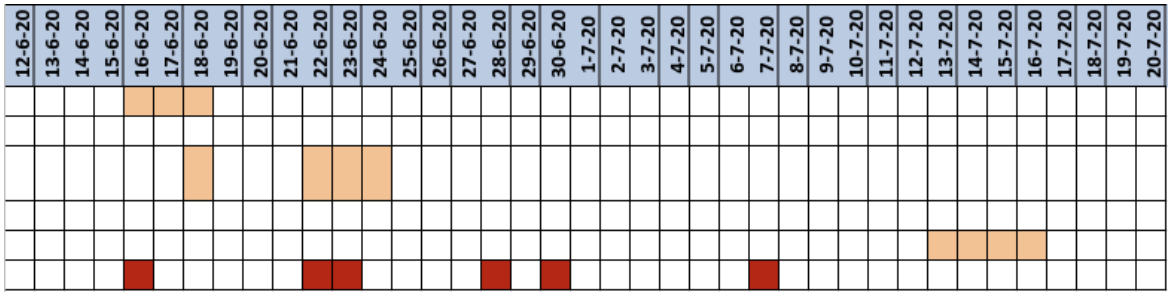


Figura 5.1. Diagrama de Gantt de la iteración 1

Se detalla a continuación cada una de dichas historias explicando cómo fue su desarrollo y si nos vimos frente a problemáticas y cómo se llegó a su solución.

Registrarse o acceder como usuario invitado

Durante el transcurso de esta historia, al ser la primera que se comenzó a desarrollar, nos encontramos frente a numerosos desafíos y nuevos conceptos que nos hicieron demorar mucho más de lo planificado.

Desde el lado del *back-end*, se comenzó a implementar el *API* de usuarios, puntualmente en el registro de usuarios y en el acceso de estos a la plataforma. En el camino de esta implementación, se tuvo que realizar mucha investigación y capacitación sobre diferentes tecnologías y herramientas que ayudarían a crear un *API* robusta y segura:

- Se tuvo que configurar las conexiones a las bases de datos correspondientes: ***postgres*** para la persistencia de nuevos usuarios y consultas de validación y ***Redis*** para el almacenamiento de las sesiones de los usuarios *logueados*.
- Se tuvo que estudiar el uso de JWT (JSON Web Token), que nos permitió definir una sesión de usuario y las llamadas seguras y validadas a los diferentes *endpoints* del *API*.
- Al ser un servicio independiente del resto, como se menciona en la definición de la arquitectura utilizada para este proyecto, se tuvo que configurar todo lo requerido para la generación de la imagen Docker que permitiría correr el servicio con *docker-compose*.
- Se debió implementar el servicio que denominamos *gw* (*gateway*) que permitía la comunicación entre los servicios del *back-end* y *front-end*.
- Se estudió el uso de las *APIs* de Google y Facebook para permitir el acceso a la aplicación mediante esas plataformas.

Todos estos problemas aquí mencionados, fueron surgiendo a medida que se iban implementando los diferentes *endpoints* HTTP que permitirían la creación y el inicio de sesión de los usuarios.

Por parte del *front-end* no hubo mayores inconvenientes más que comenzar a familiarizarse con el *framework* y las funcionalidades que ofrece.

Crear un negocio gestionado por un usuario

En esta historia, también tuvimos que enfrentar cuestiones de implementaciones desconocidas y que debían ser utilizadas para el desarrollo de la misma, esto nos retrasó aún más en la finalización del *sprint*.

Al igual que en la primera historia de esta iteración, se debieron implementar los *endpoints* necesarios para permitir la creación de negocios pertenecientes a un usuario y que cumplan con todas las reglas de negocio y requerimientos definidos en la etapa de análisis.

Lo que nos demoró aquí es el manejo de fotos pertenecientes a un negocio, su almacenamiento y su gestión:

- Se tuvo que definir e implementar el uso de archivos temporales que se almacenan inicialmente dentro del sistema de archivos propio del servicio.

Se configuró el uso del almacenamiento en la nube con *GCloud*. Además, se tuvo que investigar cómo implementar la persistencia de las fotos en la nube en el código.

Obtener información sobre un pez determinado

En este caso, los plazos requeridos para el desarrollo de la historia fueron acordes a lo planificado. Esto se debe a que la forma de implementar el servicio y sus *endpoints* eran similares a lo realizado con el *API* de usuarios, por lo que simplemente hubo que adaptar los *endpoints* de la forma adecuada.

Cabe aclarar que el *API* de especies se trata de una interfaz sencilla, que es utilizada prácticamente para realizar operaciones de consultas sobre especies de peces y su información.

Creación pantalla de inicio

Esta historia no presentó inconvenientes dado que consistía en la creación de recursos visuales para presentar o describir el objetivo de la aplicación y que funcionalidades ofrecía.

Por cuestiones de tiempos y priorización de objetivos se decidió no contemplar el diseño responsive de las imágenes. Sumado a esto, se debe aclarar que esta pantalla de inicio estará visible para todos aquellos usuarios que se encuentren utilizando la aplicación desde un navegador independientemente del dispositivo.

Realizar una captura

Esta historia, la cual estimamos que iba a ser la más compleja de todas las que integran el *sprint*, resultó ser sencilla de implementar desde el lado del *back-end* ya que pudimos aprovechar la reutilización del código generado en la historia *Crear un negocio gestionado por un usuario* para la gestión y la persistencia de las imágenes cargadas por los usuarios, lo que inicialmente era lo más complejo de realizar.

No sucedió lo mismo en el proceso de desarrollo del *front-end* ya que por primera vez se tuvo contacto con una funcionalidad que no debía depender de la conectividad del usuario. Es aquí donde se empezó a trabajar con el modo PWA para poder almacenar cada acción que se realizaba sin conexión. Por otro lado, también se debió investigar en cómo sincronizar todo lo que se había hecho al estar sin conexión una vez que el usuario recuperaba la conexión mediante una señal WI-FI o a los datos móviles. La idea era tratar de sincronizar los datos siempre que se recuperaba la conexión y en caso de no ser posible se le informaba al usuario que había elementos que se crearon y que faltan persistir.

Conclusiones de la iteración

Al culminar con la iteración, vimos que estábamos realmente atrasados con lo planificado debido a las cuestiones anteriormente mencionadas. Cabe aclarar que durante el transcurso de las distintas historias, se fueron realizando ajustes, refactorizaciones y mejoras en el código que a futuro nos permitirían tener un mejor aprovechamiento del mismo para el resto de las historias lo que también tuvo impacto en el tiempo requerido para desarrollar el *sprint*.

Para la iteración 2 ya estábamos al tanto de las diversas cuestiones que harían que el tiempo no sea cien por ciento acorde a lo planificado, aunque también sabíamos que el grueso de la implementación estaba realizada y que el resto de las historias serían similares a las ya hechas.

Iteración 2

Las historias de usuario que se decidieron implementar en la segunda iteración son:

- 9 - El usuario quiere realizar una publicación en el *feed* de una localidad.
- 15 - El usuario desea realizar la búsqueda de usuarios, servicios o locaciones.
- 18 - El usuario desea cambiar su ubicación actual.
- 19 - Teniendo una ubicación el usuario quiere saber el camino que debe seguir para llegar a ésta.

Detalle de las historias mencionadas:

Historia	Prioridad	Esfuerzo estimado (Story Points)	Esfuerzo estimado (Horas)	Esfuerzo real (Story Points)	Esfuerzo real (Horas)
Realizar una publicación en el *feed*	ALTA	3	12		
Realizar la búsqueda de usuarios, servicios o locaciones	ALTA	2	8		
Cambiar ubicación actual	ALTA	1	4		
Teniendo una ubicación el usuario quiere saber el camino que debe seguir para llegar a ésta	ALTA	3	12		
TOTAL		9	36		

En esta iteración hubo un retraso importante que no se ve en el cuadro debido a que no lo consideramos como una historia. Hubo un tiempo de inactividad en el desarrollo del servicio *front-end* (***ui-web***) debido a tareas personales y profesionales. Aprovechando el tiempo, desde el *back-end* se lo utilizó para refactorizar el código en mejores capas, haciendo un código más fácil de mantener. Además, se hizo un curso relacionado a pruebas unitarias, de integración y funcionales en *Golang* y orientado a adquirir también los conocimientos necesarios para incluir algunos *tests* en los distintos microservicios.

Se comenta a continuación cada una de dichas historias, explicando cómo fue su desarrollo, problemáticas abordadas y soluciones encontradas.

El usuario quiere realizar una publicación en el feed de una localidad

El primer punto por destacar en esta historia fue que la misma contempla tanto el alta de una publicación como la obtención de la misma para poder mostrarla al momento de crearse. Hubiera sido conveniente separar esta historia en partes más

pequeñas para poder darle una mejor estimación y un mejor tratamiento al requerimiento.

Se presentaron demoras en el desarrollo de la interfaz dado que esta funcionalidad, por definición en los requerimientos, no debía depender de la conectividad del usuario. Si bien en la historia “Realizar una captura” se trabajó con el almacenamiento de capturas realizadas sin conectividad, en esta historia se debía contemplar el hecho de que una publicación podía, adicionalmente, ser almacenada en el perfil. Con lo cual, esto implicaba un manejo de entidades en distintas pantallas, que como bien se comenta en la historia “Realizar una captura” luego debían ser sincronizadas cuando el usuario recupera la conexión.

Esta historia nos ha llevado mucho tiempo debido a que hubo mucha investigación e idas y vueltas. Se descubrió el uso de los *GeoJson* y paginado en MongoDB, como indexar en base al mismo y cómo realizar búsquedas óptimas con una latitud y longitud.

La parte que más lógica y tiempo de pensamiento/discusión requirió fue la obtención del *feed* de una localidad en base a un radio.

El primer punto fue decidir cuál iba a ser el radio máximo en el cual un usuario va a poder configurar. Esto tiene un impacto significativo cuando hay muchas publicaciones y usuarios simultáneos ya que impacta directamente en la base de datos. Por lo que hemos decidido en una primera instancia un radio de 20km, el cual nos pareció el más adecuado por el tamaño de localidades y ríos.

El segundo punto fue lo que se le mostraría al usuario por defecto. Primeramente se pensó que por defecto muestre las fotos más cercanas en un radio de 20km a la ubicación actual del usuario. Posicionándonos en la vista del usuario nos dimos cuenta de que esto es "incorrecto" dado que el principal problema se encuentra en que la mayoría de los usuarios viven en el interior y no en zonas pesqueras. Por lo que, se obtendría una pantalla vacía de publicaciones. Consideramos que es clave que el usuario tenga información en todo momento para consumir (en este caso fotos) y jamás una pantalla blanca.

En base al análisis previo y lo que no queremos que nunca ocurra, se concluyó en que por defecto el usuario reciba publicaciones de lo que pasa en el momento actual ordenado por fecha y sin importar la ubicación. De esta manera, el usuario tendrá la posibilidad de descubrir nuevos lugares pesqueros.

Por el contrario, si hubiésemos optado por la opción en la que el usuario solamente ve la localidad que selecciona previamente, el usuario para poder descubrir un lugar tendría que marcar en el mapa dicho lugar arbitrariamente y poner un radio. Dificultando así la posibilidad de descubrir lugares. El esfuerzo que llevaría descubrir una localización de esta forma hace que la aplicación pierda mucho sentido y potencial. Si a la persona le cuesta encontrar lo que quiere, que en el fondo es ir a pescar, nos encontramos que la aplicación carece de sentido. Es decir, el usuario no debería tener

que pensar cada vez que navega por la aplicación. Debe contar siempre con información para consumir.

Si se descubre un lugar pesquero, el usuario podría marcar esa localidad en el mapa con un radio y ver una lista de publicaciones mucho más concentrada sobre la zona. A continuación se dirigiría a la sección de cabañas y realizaría la reserva de alguna atracción. Ese debería ser el circuito ganador de SantaPesca.

A raíz de esto, se tuvo un nuevo problema a nivel de interfaz que era resolver cómo presentarle la información al usuario dependiendo si había seleccionado un filtro o no. Por otro lado, también se debía analizar la forma en que se actualizaban las publicaciones pensando en la mejor experiencia de usuario dependiendo de la plataforma en que se encuentre. Claramente la información debe ser enviada a medida que es solicitada, en nuestro caso, esto se traduce en que a medida que el usuario baja se empieza a cargar la información restante. De esta forma, no se hace una sobrecarga de información que puede no ser de interés para el usuario.

Desde el *back-end* esto fue muy simple, debido a que MongoDB ya tiene ordenado por fecha (id) por lo que simplemente se devuelven las N fotos por página que nos pida el *front-end* respetando el formato en ambas peticiones (default y punto en el mapa).

Un refinamiento con una visualización mejorada que quedará para más adelante es el de un mejor trabajo sobre la ubicación. Es decir, devolverle al usuario las fotos más cercanas de su ubicación. Esto es muy interesante, ya que si el usuario se entera que hay pique a 50 km de su casa, no es lo mismo que enterarse que está saliendo en tierra del fuego a 2000km de su casa.

El usuario desea realizar la búsqueda de usuarios, servicios o locaciones

El trabajo realizado para poder resolver esta historia también nos llevó un esfuerzo mayor al estimado. Esto se debe principalmente a que también se tuvo que realizar una tarea de investigación previa para poder realizar la búsqueda de forma eficiente.

Primero se comenzó con la búsqueda de servicios (negocios, bajadas de lancha, carnada, etc.) donde se tuvo que realizar un estudio sobre cómo realizar una búsqueda en ***postgres*** aplicando paginación para que esta búsqueda sea más eficiente. A esto hay que agregarle las cuestiones lógicas sobre la búsqueda: cómo ordenar los resultados, como aplicar los filtros, si los filtros son excluyentes unos de otros o no, etc.

Una vez definidas estas cuestiones, la implementación de esta historia no fue complicada aunque hubo que realizar algunas correcciones/mejoras a medida que se iba probando.

Luego de que la búsqueda de servicios funcionara correctamente, se agregó la búsqueda de usuarios mediante el nombre. Esta parte de la historia se realizó de forma rápida ya que se contaba con el conocimiento previo sobre cómo realizar búsqueda con paginación en ***postgres*** y que además es una búsqueda simple ya que cuenta con un solo criterio de búsqueda.

Conclusiones de la iteración

Como se comenta en la introducción de esta iteración acerca del tiempo en reposo que hubo en el desarrollo de un servicio, se aprovechó el estudio e investigación en cómo hacer *tests* en *Golang*. Algo que parecía sencillo termina no siéndolo. Nos hemos dado cuenta de que mucho código no lo teníamos bien estructurado y adaptado para poder hacer pruebas.

A raíz de lo mencionado se desprendieron muchas tareas de refactorización de código, estudio de *frameworks* que nos faciliten dichas pruebas y buscando la forma de hacer *mocks* de las bases de datos para que los *tests* no dependan de una aplicación encendida y se hagan efectivamente como deben hacerse (en un *stage* o paso intermedio a un *deploy* y *offline*).

Los *tests* son una parte muy importante en el desarrollo de software y debe ser tenido en cuenta en etapas temprana de desarrollo, esto en el ambiente profesional a veces se desprecia por querer tener un sistema productivo lo antes posible y se termina pagando el precio más adelante.

En la facultad se ve muy por encima de forma práctica en una materia optativa y es algo que incluso nosotros queríamos dejar para el final. Por suerte, debido a un imprevisto, nos ha tocado hacer pruebas a mitad del desarrollo aproximadamente y gracias a esto hemos aprendido bastante y se nos han clarificado muchas dudas. Si esto no hubiese ocurrido hubiésemos seguido programando de una forma “no correcta” hasta la finalización de la tesis para luego en el final encontrarnos con este problema, algo que se hubiera traducido en tiempo ya que era más código para refactorizar.

Iteración 3

Las historias de usuario que se decidieron implementar en la tercera iteración son:

- 7 - El usuario desea emitir una reseña a un servicio determinado.
- 11 - El usuario quiere eliminar una publicación propia en el *feed* y perfil.
- 12 - Un usuario desea interactuar con otros mediante el envío de mensajes privados.
- 13 - Un usuario quiere agregar comentarios en una publicación.
- 14 - El usuario desea eliminar comentarios propios en una publicación.

Historia	Prioridad	Esfuerzo estimado (Story Points)	Esfuerzo estimado (Horas)	Esfuerzo real (Story Points)	Esfuerzo real (Horas)
Un usuario emite una reseña sobre un servicio determinado	MEDIA	2	8		
Usuario debe poder eliminar una publicación del ***feed*** y perfil	MEDIA	1	4		
Implementar un chat	-	-	-	-	-
Un usuario comenta una publicación	BAJA	1	4		
Un usuario elimina un comentario propio en una publicación	BAJA	1	4		
TOTAL					

Se detalla a continuación cada una de las historias de este *sprint* explicando cómo fue su implementación y si nos vimos frente a problemáticas y cómo se llegó a su solución.

El usuario desea emitir una reseña a un servicio determinado

Historia resuelta en tiempo y forma. No presentó inconvenientes ni retrasos.

El usuario quiere eliminar una publicación propia del feed y perfil

La actual historia logró desarrollarse de forma rápida a nivel *back-end* aunque incluyó algunos desafíos y correcciones, los mismos se detallan a continuación:

- El principal desafío fue poner en dos *go-routines* (no es lo mismo que un hilo en java, pero se lo puede considerar de forma similar) el eliminado de la foto en el *bucket* de Google y el registro en la base de datos (mongodb). Esto es para que el borrado sea con paralelismo y no secuencialmente.
- Recordemos que en mongo se almacena un registro de un “Post” o “Perfil” el cual contiene información del nombre de la foto (entre otras cosas) y el id del

usuario el cual creó el mismo, luego con esos dos datos el microservicio ***feed*** es capaz de armar la URL del *bucket* en la que se encuentra la foto. Con esta información podemos darle la foto al *front-end* mediante una URL y eliminar la misma para el caso actual.

- Había un bug en donde las fotos las almacenábamos con el mismo nombre en la que el usuario las subía, esto rompía si un usuario subía dos fotos iguales debido a que a nivel base de datos quedaban dos registros (lo correcto) pero a nivel *bucket* quedaba una única foto debido a que se llamaban iguales y Google la sobrescribe porque entiende que son iguales.
- Para mitigar el problema lo que se hizo fue renombrar todas las fotos nuevas con un "*timestamp*" que incluye milisegundos. Por lo que hace casi imposible que queden dos registros iguales.
- Además se tuvo que editar el método encargado de dar de alta un nuevo post o foto en el perfil para que retorne el "id" de la nueva foto creada. De esta manera si el usuario elimina una foto, el servicio *front-end* pueda sacar un *request* al *endpoint* de eliminar con el id del post correspondiente.

En cuestión de la interfaz, la única dificultad presentada era que se debía analizar si correspondía eliminar una publicación de la base de datos o del almacenamiento local dependiendo si el elemento se había creado con conectividad a internet o no. Fuera de eso, no se presentaron mayores inconvenientes.

Implementar un chat

Esta historia se decidió quitarla de la implementación final porque contaba con una complejidad alta (había que aprender tecnologías no utilizadas como *websocket* para poder crear la comunicación) y requería demasiado tiempo.

Un usuario quiere agregar comentarios en una publicación del feed

No presentó ningún inconveniente. Se resolvió de la forma más eficiente, la cual es agregar un comentario nuevo a la lista de comentarios de un post. El mismo contiene además del comentario en sí, información del autor del mismo.

El usuario desea eliminar comentarios propios en una publicación del feed

Historia sencilla y terminada en poco tiempo.

Iteración 4

Las historias de usuario que se decidieron implementar en la tercera iteración son:

- 2 - El usuario registrado desea modificar datos propios en su perfil.
- 5 - El usuario desea realizar la modificación de datos pertenecientes a la cuenta de un negocio gestionado por él.
- 10 - El usuario desea modificar datos o configuraciones de su publicación en la red social.
- 20 - El usuario desea indicar que una publicación, mensaje o comentario le ha gustado.
- 23 - El usuario desea remover de su perfil una “captura” realizada.
- 24 - El usuario podrá visualizar las notificaciones que recibe.
- 26 - El usuario desea compartir con otra persona una publicación, ubicación, negocio o producto.

Historia	Prioridad	Esfuerzo estimado (Story Points)	Esfuerzo estimado (Horas)	Esfuerzo real (Story Points)	Esfuerzo real (Horas)
El usuario registrado desea modificar datos propios en su perfil.	MEDIA	1	4		
El usuario desea realizar la modificación de datos pertenecientes a la cuenta de un negocio gestionado por él.	MEDIA	1	4		
El usuario desea modificar datos o configuraciones de su publicación en la red social.	BAJA	1	4		
El usuario desea indicar que una publicación, mensaje o comentario le ha gustado.	BAJA	1	4		
El usuario desea remover de su perfil una “captura” realizada.	BAJA	2	8		
El usuario podrá visualizar las notificaciones que recibe.	-	-	-	-	-
El usuario desea compartir con otra	MEDIA	1	4		

persona una publicación, ubicación, negocio o producto.					
TOTAL					

El usuario registrado desea modificar datos propios en su perfil.

La modificación de datos de perfil incluyó tareas que no habíamos pensado en el plan. La principal tarea fue la posibilidad de que un usuario pueda establecer una foto de perfil. La misma involucró un nuevo manejo de fotos y junto con la lógica de un establecimiento de una imagen *default*.

Se decidió establecer una foto por defecto como perfil para todos los nuevos usuarios que crean una cuenta en SantaPesca. Luego, si él mismo decide actualizarla con una imagen personal, lo podrá hacer. Por el contrario, si quiere eliminar una foto de perfil se aplicará la *default* nuevamente.

El usuario desea realizar la modificación de datos pertenecientes a la cuenta de un negocio gestionado por él.

Lo que nos ha ocurrido fue que el “alta de negocios” fue una de las primeras historias del proyecto y también una de las más complejas. Esto es así, ya que involucra un manejo de fotos con un *upload* de forma paralela, compresiones y la integración de un *bucket* de Google donde se suben allí. Además, eran las primeras líneas de código sobre el lenguaje de programación *Golang* el cual ninguno conocía de antemano.

Una vez desarrollado el alta de negocio y de acuerdo a nuestro plan de proyecto se tuvo que continuar con otras historias. La modificación y la baja de un negocio eran tareas de esta y la última iteración (cuatro y cinco). Debido a la poca experiencia sobre un lenguaje y la complejidad de la historia en el alta, hemos cometido varios errores de diseño y de programación que tuvimos que mitigar en esta historia.

Concluimos que lo que nos ha ocurrido tiene algo positivo y negativo. Como punto a favor tenemos que con los meses de experiencia sobre un lenguaje y conocimiento de la tecnología se hizo una mejora sustancial en el código de alta y permitió resolver la modificación y baja de una forma más sencilla. Como punto en contra tenemos que nos hemos demorado más de lo esperado en esta historia debido a que, en primer lugar, no contábamos con un análisis lo suficientemente claro de los requerimientos a realizar y, en segundo lugar, no se realizó un estudio acerca de las tareas que requería hacer tanto esta historia como el alta o la eliminación.

El usuario desea modificar datos o configuraciones de su publicación en la red social.

Desde el *back-end*, la realización de esta historia llevó un poco más de tiempo de lo planificado ya que, además de realizar la correspondiente funcionalidad que le permita al usuario modificar su publicación, se agregaron otras funcionalidades que eran complementarias a ésta como, por ejemplo, la obtención de todas las publicaciones del usuario para que le sea más fácil encontrar la publicación a editar.

Este es otro claro ejemplo de que no contábamos con los requerimientos muy definidos antes de comenzar.

El usuario desea indicar que una publicación, mensaje o comentario le ha gustado.

No hubo inconvenientes que se destaquen en la resolución de esta historia. A nivel de *front-end*, fue incorporado un ícono que representa el "me gusta" en las distintas entidades y de esa forma por cada accionar se realizaba la llamada pertinente al *back-end*. Esto es, informarle al servicio ***feed*** (en el caso de que se tratase de una publicación) si una publicación era del agrado del usuario o no. Lo mismo sucedía con los comentarios. Hubo que incorporar una lógica para determinar si ese comentario o publicación ya había sido marcado como de interés para el usuario pero no representó una dificultad para el equipo.

A nivel de *back-end* era simplemente cambiar un campo en el registro correspondiente.

El usuario desea remover de su perfil una "captura" realizada

Esta historia consistía simplemente en el agregado de un botón que permita la eliminación y realizar la comunicación pertinente con el *back-end*.

Desde el *back-end*, esta historia llevó muy poco tiempo ya que se reutilizó el código generado para realizar la Historia 11 que contaba con una lógica similar.

El usuario podrá visualizar las notificaciones que recibe.

Esta historia se eliminó para reducir la complejidad y el tiempo de desarrollo del sistema.

El usuario desea compartir con otra persona una publicación, ubicación, negocio o producto.

A nivel *back-end*, se realizaron nuevos *endpoints* para poder obtener todos los datos de la publicación o el negocio que el usuario desea compartir con otra persona. También se mejoraron los *endpoints* para obtener los comentarios de las publicaciones o las *reviews* de los negocios admitiendo paginación lo que hace más eficiente la consulta. Estas tareas no necesitaron mucho esfuerzo del equipo ya que la lógica era similar a las de otras historias.

Iteración 5

Las historias de usuario que se decidieron implementar en la tercera iteración son:

- Historia 3: Un usuario no quiere seguir siendo parte de la aplicación.
- Historia 6: El usuario desea eliminar la cuenta del negocio administrado por él.
- Historia 17: El usuario desea guardar una publicación, ubicación o una cabaña.
- Historia 21: El usuario requiere de ayuda para poder comprender cómo debe utilizar el sistema.
- Historia 25: El usuario desea realizar una donación monetaria.

Historia	Prioridad	Esfuerzo estimado (Story Points)	Esfuerzo estimado (Horas)	Esfuerzo real (Story Points)	Esfuerzo real (Horas)
Un usuario no quiere seguir siendo parte de la aplicación	BAJA	2	8		
El usuario desea eliminar la cuenta del negocio administrado por él	BAJA	2	8		
El usuario desea guardar una publicación, ubicación o una cabaña	BAJA	2	8		
El usuario requiere de ayuda para poder comprender cómo debe utilizar el sistema	-	-	-	-	-
El usuario desea realizar una donación monetaria	BAJA	2	5		
TOTAL					

Un usuario no quiere seguir siendo parte de la aplicación

Historia sin complejidad pero larga. La baja de un usuario involucra el borrado de muchos datos tanto en las bases de datos como en el *bucket*, sumado a que deben hacerse muchas validaciones para asegurar la consistencia. Además, involucra la comunicación entre los servicios ***feed*** y ***users***.

El borrado se comienza invocando a un *endpoint* en ***users*** en el cual se disparan cinco rutinas para acelerar el proceso de borrado mediante paralelismo. La primera rutina dispara un *request* al servicio ***feed*** donde se encarga de eliminar toda la data del

usuario que involucra al servicio (publicaciones con fotos, comentarios y likes). La segunda borra la foto de perfil, en caso de que tenga una diferente a la default. La tercera borra todos los negocios junto con sus fotos. La cuarta borra todas las publicaciones del perfil de un usuario almacenadas en mongo. La quinta y la última borra todas las fotos de las publicaciones del perfil de un usuario. Luego de terminadas las rutinas y sin errores, se procede con el borrado de los datos restantes en ***postgres*** y la eliminación de la sesión en ***redis***.

El usuario desea eliminar la cuenta del negocio administrado por él

Historia simple, sin ninguna sorpresa ni necesidad de modificar código. Gracias al *Framework*/*ORM* utilizado para ***postgres*** (*Gorm*[19]) fue muy simple hacer la baja de un negocio. Además, se reutilizó el código empleado en métodos previos para la eliminación de las fotos de un negocio alocadas en el *bucket* de Google.

El usuario desea guardar una publicación, ubicación o una cabaña

A nivel de interfaz no hubo grandes inconvenientes más que realizar una integración total entre las entidades. Es decir, se incorporó una pantalla en donde se visualizan todos los elementos que el usuario decidió guardar. A su vez, todos estos elementos podrían estar presentes en alguna otra pantalla con lo cual cualquier modificación en cada una de estas entidades debía replicarse en el resto. Esto ocurrió dado que se tomó la decisión de mantener un almacenamiento temporal de aquellos datos consultados mientras la aplicación está siendo utilizada, con el fin de minimizar las consultas realizadas al *back-end*. También, esto favoreció al manejo de información cuando el usuario se encuentra sin conexión a internet.

Por el lado del *back-end*, esta historia no presentó complejidad pero sí requirió de tiempo ya que se tuvo que realizar el ABM completo de los elementos guardados. Además, cabe destacar que las ubicaciones o los negocios guardados se gestionan en ***postgres*** mientras que las publicaciones en MongoDB, lo cual hizo necesaria la creación del ABM para ambas bases de datos.

El usuario requiere de ayuda para poder comprender cómo debe utilizar el sistema

Se decidió eliminar esta historia dado que no aportaba algo sustancial para el usuario pero sí consumía tiempo para el equipo. La idea inicial era proporcionar un tutorial sobre las principales funcionalidades del sistema. Por ejemplo, cómo realizar una publicación de una especie capturada y, a su vez, almacenarla en el perfil del usuario. Esto se pensaba realizar mediante la ilustración de imágenes y capturas de la aplicación enfocándose en que sean de atractivo visual.

Como se describió anteriormente era una funcionalidad que se quería incorporar como complemento y no como algo crítico para el sistema. Sumado a esto, el tiempo del equipo era acotado y creemos que la remoción no presentaba un inconveniente para el sistema en su conjunto.

[19] https://gorm.io/index.html

El usuario desea realizar una donación monetaria

Es una historia en donde se trabajó únicamente en el *front-end*. No presentó mayores inconvenientes. Se utilizó el servicio PayPal, en donde se configuró una cuenta y se obtuvo un enlace al cual los usuarios son redirigidos si desean realizar una donación monetaria.

5.3. Automatización de pruebas

En esta sección realizaremos una descripción de los diferentes casos de pruebas realizados, se muestran a continuación los *tests* más representativos de los distintos servicios para no hacer extenso y repetitivo este apartado. El resto de los *tests* quedarán en el código fuente.

5.3.1. Pruebas Unitarias

Microservicio species

Tests unitarios realizados para el **microservicio species**.

Título de la prueba:	Test del controlador que devuelve especies existentes	**Autor de la prueba:**	Martín Destéfanis
Descripción:	El *test* se encarga de hacer un *mock* de ***postgres***, insertar especies, recibir un *request* y responder un json siguiendo el modelo requerido.		
Resultado esperado:	En caso de éxito se espera un 200 con un json que contiene una lista de especies con la estructura siguiente: [{"id":"cccc44be-4b08-459e-a217-f937d0a42ced","name":"Boga","photo":"https://storage.googleapis.com/species_images_santapesca/boga.jpg","canBeFished":true}]. En caso de falla ya sea porque no puede llegar a la base de datos o no existan especies devuelve un 500 y un json como el siguiente: {"message":"INTERNAL_SERVER_ERROR"}		

```go
func TestControllerSpecies_GetAllSpecies(t *testing.T) {
    //mock db postgres
    db, mock, err := sqlmock.New() // mock sql.DB
    if err != nil {...}
    gdb, err := gorm.Open( dialect: "postgres", db) // open gorm db
    if err != nil {...}
    defer db.Close()
    defer gdb.Close()

    //before we actually execute our api function, we need to expect required DB actions
    rows := sqlmock.NewRows([]string{...}).
        AddRow( values...: "cccc44be-4b08-459e-a217-f937d0a42ced", "Boga", "https://storage.googleapis.com/species_images_santapesca/boga.jpg", true).
        AddRow( values...: "54554068-2a2a-45ec-a408-17d47d0285a0", "Dorado", "https://storage.googleapis.com/species_images_santapesca/dorado.png", true).
        AddRow( values...: "3d2eb596-a2ff-4873-a37b-d37bccd15dbb", "Amarillo", "https://storage.googleapis.com/species_images_santapesca/amarillo.png", true).
        AddRow( values...: "4fd89903-f9d7-48bd-bb57-af1bc274e496", "Armado Comun", "https://storage.googleapis.com/species_images_santapesca/armado%20comun.png", true).
        AddRow( values...: "bd7383ca-c0ea-4bb5-b0ec-8ca07a0457a9", "Armado Chancho", "https://storage.googleapis.com/species_images_santapesca/armado%20chancho.jpeg", true).
        AddRow( values...: "d1bc979f-ad03-42ab-a1a7-dcc7848e752f", "Manguruyu", "https://storage.googleapis.com/species_images_santapesca/manguruyu.jpg", false).
        AddRow( values...: "479c0e8e-af5f-4246-9fbb-465e4cf8230e", "Pacú", "https://storage.googleapis.com/species_images_santapesca/pacu.jpg", false)

    mock.ExpectQuery(regexp.QuoteMeta( s: `SELECT id, name, photo, can_be_fished FROM "species"`)).WillReturnRows(rows)

    req, err := http.NewRequest( method: "GET", url: "/species", body: nil)
    if err != nil {...}
    rr := httptest.NewRecorder()
    handler := ControllerSpecies.GetAllSpecies(gdb)
    handler.ServeHTTP(rr, req)

    assert.EqualValues(t, http.StatusOK, rr.Code)

    typesExpected := []*models.ResponseSpecie{...}

    b, _ := json.Marshal(typesExpected)

    assert.JSONEqf(t, string(b), string(rr.Body.Bytes()), msg: "error message %s", args...: "formatted")
}
```

Código de la prueba sin error

Figura 12.1. Código del *test* sin error

```go
func TestControllerSpecies_GetAllSpeciesError(t *testing.T) {
    //mock db postgres
    db, mock, err := sqlmock.New() // mock sql.DB
    if err != nil {...}
    gdb, err := gorm.Open( dialect: "postgres", db) // open gorm db
    if err != nil {...}
    defer db.Close()
    defer gdb.Close()

    //before we actually execute our api function, we need to expect required DB actions
    mock.ExpectQuery(regexp.QuoteMeta( s: `SELECT id, name, photo, can_be_fished FROM "species"`)).WillReturnError(errors.New( text: "testing error"))

    req, err := http.NewRequest( method: "GET", url: "/species", body: nil)
    if err != nil {
        t.Fatal(err)
    }
    rr := httptest.NewRecorder()
    handler := ControllerSpecies.GetAllSpecies(gdb)
    handler.ServeHTTP(rr, req)

    assert.EqualValues(t, http.StatusInternalServerError, rr.Code)

    messageExpected := blModels.Error{...}

    b, _ := json.Marshal(messageExpected)

    assert.JSONEqf(t, string(rr.Body.Bytes()), string(b), msg: "error message %s", args...: "formatted")
}
```

Código de la prueba con error

Figura 12.2. Código del *test* con error

Título de la prueba:	Test del controlador que chequea si una especie existe en la base de datos	**Autor de la prueba:**	Pedro Capedevila
Descripción:	El *test* se encarga de hacer un *mock* de ***postgres***, insertar una especie moqueada, simular la *query* y el *request*.		
Resultado esperado:	En caso de éxito se espera un 200. En caso de falla ya sea porque no puede llegar a la base de datos o no exista la especie a validar devuelve los códigos 500 y 404 respectivamente con un json: {"message":"INTERNAL_SERVER_ERROR"}		

Código de la prueba sin error

```go
func TestControllerSpecies_CheckSpecieExist(t *testing.T) {
    //mock db postgres
    db, mock, err := sqlmock.New() // mock sql.DB
    if err != nil {
        t.Fatalf( format: "an error '%s' was not expected when opening a stub database connection", err)
    }
    gdb, err := gorm.Open( dialect: "postgres", db) // open gorm db
    if err != nil {
        t.Fatalf( format: "an error '%s' was not expected when opening a stub database connection", err)
    }
    defer db.Close()
    defer gdb.Close()

    //before we actually execute our api function, we need to expect required DB actions
    rows := sqlmock.NewRows([]string{"name"}).
        AddRow( values...: "Boga")

    mock.ExpectQuery(regexp.QuoteMeta( s: `SELECT name FROM "species"  WHERE (id::text = $1`)).WillReturnRows(rows)

    req, err := http.NewRequest( method: "HEAD",  url: "/exist/cccc44be-4b08-459e-a217-f937d0a42ced",  body: nil)
    if err != nil {
        t.Fatal(err)
    }
    rr := httptest.NewRecorder()
    handler := ControllerSpecies.CheckSpecieExist(gdb)
    handler.ServeHTTP(rr, req)

    assert.EqualValues(t, http.StatusOK, rr.Code)
}
```

Figura 12.3. Código del *test* sin error

Código de la prueba con error

```go
func TestControllerSpecies_CheckSpecieExistErrorNotExists(t *testing.T) {
    //mock db postgres
    db, mock, err := sqlmock.New() // mock sql.DB
    if err != nil {
        t.Fatalf( format: "an error '%s' was not expected when opening a stub database connection", err)
    }
    gdb, err := gorm.Open( dialect: "postgres", db) // open gorm db
    if err != nil {
        t.Fatalf( format: "an error '%s' was not expected when opening a stub database connection", err)
    }
    defer db.Close()
    defer gdb.Close()

    //before we actually execute our api function, we need to expect required DB actions
    mock.ExpectQuery(regexp.QuoteMeta( s: `SELECT name FROM "species"  WHERE (id::text = $1`)).WillReturnError(gorm.ErrRecordNotFound)

    req, err := http.NewRequest( method: "HEAD",  url: "/exist/cccc44be-4b08-459e-a217-f937d0a42ced",  body: nil)
    if err != nil {
        t.Fatal(err)
    }
    rr := httptest.NewRecorder()
    handler := ControllerSpecies.CheckSpecieExist(gdb)
    handler.ServeHTTP(rr, req)

    assert.EqualValues(t, http.StatusNotFound, rr.Code)

    messageExpected := blModels.Error{
        Message: "INTERNAL_SERVER_ERROR",
    }

    b, _ := json.Marshal(messageExpected)

    assert.JSONEqf(t, string(rr.Body.Bytes()), string(b),  msg: "error message %s",  args...: "formatted")
}
```

Figura 12.4. Código del *test* con error

Microservicio users

Tests unitarios realizados para el **microservicio *users***.

Título de la prueba:	Test del controlador que retorna todos los tipos de negocios	**Autor de la prueba:**	Martín Destéfanis
Descripción:	El *test* se encarga de hacer un *mock* de ***postgres***, simular un *request* al controlador, retornar los valores *mockeados* cuando se realiza la consulta en la base de datos y retornar como response del *request* los datos.		
Resultado esperado:	En caso de éxito se espera un 200 y los tipos de negocios en formato JSON. En caso de falla devuelve el código HTTP 500 con el siguiente mensaje: {"message":"INTERNAL_SERVER_ERROR"}		

```go
func TestControllerShops_GetAllTypes(t *testing.T) {
    //mock db postgres
    db, mock, err := sqlmock.New() // mock sql.DB
    if err != nil {...}
    gdb, err := gorm.Open( dialect: "postgres", db) // open gorm db
    if err != nil {...}
    defer db.Close()
    defer gdb.Close()

    //before we actually execute our api function, we need to expect required DB actions
    rows := sqlmock.NewRows([]string{...}).
        AddRow( values...: 1, "Cabaña", pq.StringArray{...}).
        AddRow( values...: 2, "Negocio", pq.StringArray{...}).
        AddRow( values...: 3, "Guarderia", pq.StringArray{...}).
        AddRow( values...: 4, "Bajada de lancha", pq.StringArray{...})

    mock.ExpectQuery(regexp.QuoteMeta( s: `SELECT * FROM "types"`)).WillReturnRows(rows)

    req, err := http.NewRequest( method: "GET",  url: "/shop/types",  body: nil)
    if err != nil {...}
    rr := httptest.NewRecorder()
    handler := ControllerShops.GetAllTypes(gdb)
    handler.ServeHTTP(rr, req)

    assert.EqualValues(t, http.StatusOK, rr.Code)

    typesExpected := []*models.Type{...}

    b, _ := json.Marshal(typesExpected)

    assert.JSONEqf(t, string(rr.Body.Bytes()), string(b),  msg: "error message %s",  args...: "formatted")
}
```

Código de la prueba sin error

Figura 12.5. Código del *test* sin error

Código de la prueba con error

```go
func TestControllerShops_GetAllTypesError(t *testing.T) {
    //mock db postgres
    db, mock, err := sqlmock.New() // mock sql.DB
    if err != nil {...}
    gdb, err := gorm.Open( dialect: "postgres", db) // open gorm db
    if err != nil {...}
    defer db.Close()
    defer gdb.Close()

    //before we actually execute our api function, we need to expect required DB actions
    mock.ExpectQuery(regexp.QuoteMeta( s: `SELECT * FROM "types"`)).WillReturnError(errors.New( text: "testing error"))

    req, err := http.NewRequest( method: "GET",  url: "/shop/types",  body: nil)
    if err != nil {...}
    rr := httptest.NewRecorder()
    handler := ControllerShops.GetAllTypes(gdb)
    handler.ServeHTTP(rr, req)

    assert.EqualValues(t, http.StatusInternalServerError, rr.Code)

    messageExpected := blModels.Error{...}

    b, _ := json.Marshal(messageExpected)

    assert.JSONEqf(t, string(rr.Body.Bytes()), string(b),  msg: "error message %s",  args...: "formatted")
}
```

Figura 12.6. Código del *test* con error

Título de la prueba:	Test del controlador que cierra la sesión del usuario	**Autor de la prueba:**	Pedro Capedevila
Descripción:	El *test* se encarga de hacer un *mock* de ***redis***, simular un *request* al controlador, simular la eliminación de la sesión del usuario de ***redis*** y retornar como response el mensaje “Session deleted”.		
Resultado esperado:	En caso de éxito se espera un 200 y el mensaje “Session deleted”. En caso de falla devuelve el código HTTP 500 con el siguiente mensaje: {"message":"INTERNAL_SERVER_ERROR"}		

Código de la prueba sin error

```go
func TestControllerUsers_Logout(t *testing.T) {
    token := "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImRldmVsb3BlciIsImp0aSI6IjJmNDJlOTAwLTk5YzgtNDU3MC05MD

    //mock db redis
    mr, err := miniredis.Run()
    if err != nil {
        log.Fatalf("an error '#{err}' was not expected when opening a stub database connection")
    }
    redisClient := redis.NewClient(&redis.Options{
        Addr: mr.Addr(),
    })
    redisMock := redismock.NewNiceMock(redisClient)

    //mock DeleteToken
    redisMock.On( methodName: "Del", context.Background(), []string{token}).Return(redis.NewIntResult( val: 0,  err: nil))

    req, err := http.NewRequest( method: "POST",  url: "/logout",  body: nil)
    if err != nil {
        t.Fatal(err)
    }
    req.Header.Add( key: "Authorization", "Bearer "+token)

    rr := httptest.NewRecorder()
    handler := ControllerUsers.Logout(redisMock)
    handler.ServeHTTP(rr, req)

    assert.EqualValues(t, http.StatusOK, rr.Code)

    b, _ := json.Marshal( v: "Session deleted")

    assert.JSONEqf(t, string(b), string(rr.Body.Bytes()),  msg: "error message %s",  args...: "formatted")
}
```

Figura 12.7. Código del *test* sin error

Código de la prueba con error

```go
func TestControllerUsers_LogoutError(t *testing.T) {
    token := "eyJhb6ciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImRldmVsb3BlciIsImp0aSI6IjJmNDJlOTAwLTk5YzgtNDU3MC05MDMzLWUxMDExODllMjNkZCJ9.gB

    //mock db redis
    mr, err := miniredis.Run()
    if err != nil {
        log.Fatalf("an error '#{err}' was not expected when opening a stub database connection")
    }
    redisClient := redis.NewClient(&redis.Options{
        Addr: mr.Addr(),
    })
    redisMock := redismock.NewNiceMock(redisClient)

    //mock DeleteToken
    redisMock.On( methodName: "Del", context.Background(), []string{token}).Return(redis.NewIntResult( val: 0, errors.New( text: "testing error")))

    req, err := http.NewRequest( method: "POST", url: "/logout", body: nil)
    if err != nil {
        t.Fatal(err)
    }
    req.Header.Add( key: "Authorization", "Bearer "+token)

    rr := httptest.NewRecorder()
    handler := ControllerUsers.Logout(redisMock)
    handler.ServeHTTP(rr, req)

    assert.EqualValues(t, http.StatusInternalServerError, rr.Code)

    messageExpected := blModels.Error{
        Message: "INTERNAL_SERVER_ERROR",
    }

    b, _ := json.Marshal(messageExpected)

    assert.JSONEqf(t, string(rr.Body.Bytes()), string(b), msg: "error message %s", args...: "formatted")
}
```

Figura 12.8. Código del *test* con error

5.3.2. Pruebas Funcionales

Microservicio users

Tests funcionales realizados para el **microservicio *users***.

Título de la prueba:	Test del controlador que retorna todos los tipos de negocios	**Autor de la prueba:**	Pedro Capdevila
Descripción:	El *test* se encarga de iniciar la aplicación pero en vez de utilizar las bases de datos reales lo hace mediante *mocks* de ***postgres*** y ***redis***. Luego el proceso es similar a los *test* unitarios pero aquí no se simula un *request* al controlador sino que se realiza un *request* real al *endpoint* de la aplicación correspondiente, se retornan los valores *mockeados* cuando se realiza la consulta en la base de datos y se devuelve como response del *request* los datos.		
Resultado esperado:	En caso de éxito se espera un 200 y los tipos de negocios en formato JSON. En caso de falla devuelve el código HTTP 500 con el siguiente mensaje: {"message":"INTERNAL_SERVER_ERROR"}		

Para este tipo de *test* se cuenta con dos partes. La primera es la encargada de inicializar la aplicación con las conexiones a las bases de datos moqueadas:

```go
func TestMain(m *testing.M) {
    var db *sql.DB
    a := app.App{}

    //mock db postgres
    db, postgresMock, _ = sqlmock.New() // mock sql.DB
    gdb, _ := gorm.Open( dialect: "postgres", db) // open gorm db
    a.DB = gdb
    defer db.Close()
    defer gdb.Close()

    //mock db redis
    mr, err := miniredis.Run()
    if err != nil {
        log.Fatalf( format: "an error '%s' was not expected when opening a stub database connection", err)
    }
    redisClient := redis.NewClient(&redis.Options{
        Addr: mr.Addr(),
    })
    redisMock = redismock.NewNiceMock(redisClient)
    a.RDB = redisMock

    //inicializar router
    a.Router = mux.NewRouter().StrictSlash( value: true)
    a.MapURLs()

    //lanzar app
    go a.Run()
    os.Exit(m.Run())
}
```

Figura 12.9. Código del *test* funcional del microservicio ***users***

La segunda parte del caso de prueba consta de ejecutar el *test*:

```go
func TestGetAllTypes(t *testing.T) {
    token := "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6ImRldmVsb3BlciIsImp0aSI6IjJmNDJlOTAwLTk5YzgtNDU3MC05MDMzLWUxMDExODllMjNkZCJ9.e

    //mock GetUserActionAndSubjectByEmail
    rows := sqlmock.NewRows([]string{"actions", "subjects"}).
        AddRow(pq.StringArray{"manage"}, pq.StringArray{"all"})
    postgresMock.ExpectQuery(regexp.
        QuoteMeta( s: `SELECT actions, subjects FROM "users" JOIN roles ON users.role_id = roles.id WHERE (users.email = $1)`)).
        WillReturnRows(rows)

    //mock GetAllTypes
    rows = sqlmock.NewRows([]string{"id", "type", "services"}).
        AddRow( values...: 1, "Cabaña", pq.StringArray{"Wi-Fi", "Estacionamiento", "Desayuno", "Servicio de limpieza", "TV", "Aire acondicionado"
        AddRow( values...: 2, "Negocio", pq.StringArray{"Venta de carnada", "Instrumental"}).
        AddRow( values...: 3, "Guardería", pq.StringArray{"Bajada de lancha", "Baño", "Luz", "Asador", "Carnada"}).
        AddRow( values...: 4, "Bajada de lancha", pq.StringArray{"Baño", "Luz", "Asador", "Carnada"})
    postgresMock.ExpectQuery(regexp.QuoteMeta( s: `SELECT * FROM "types"`)).WillReturnRows(rows)

    //mock CheckIfTokenExists
    redisMock.On( methodName: "Get", context.Background(), token).Return(redis.NewStringResult( val: "Token exists", err: nil))

    httpClient := &http.Client{}
```

Figura 12.10. Código del *test* funcional del microservicio ***users***

```
    req, err := http.NewRequest( method: "GET",  url: "http://localhost:8000/shop/types",  body: nil)
    if err != nil {
        t.Fatal(err)
    }
    req.Header.Add( key: "Authorization", "Bearer "+token)

    response, err := httpClient.Do(req)
    if err != nil {
        t.Fatal(err)
    }
    defer response.Body.Close()
    body, err := ioutil.ReadAll(response.Body)

    assert.Nil(t, err)
    assert.NotNil(t, response)

    typesExpected := []*models.Type{
        { ID: 1,  Type: "Cabaña",  Services: pq.StringArray{"Wi-Fi", "Estacionamiento", "Desayuno", "Servicio de limpieza", "TV", "Aire acondicionado"
        { ID: 2,  Type: "Negocio",  Services: pq.StringArray{"Venta de carnada", "Instrumental"}},
        { ID: 3,  Type: "Guardería",  Services: pq.StringArray{"Bajada de lancha", "Baño", "Luz", "Asador", "Carnada"}},
        { ID: 4,  Type: "Bajada de lancha",  Services: pq.StringArray{"Baño", "Luz", "Asador", "Carnada"}},
    }

    b, _ := json.Marshal(typesExpected)

    assert.JSONEqf(t, string(body), string(b),  msg: "error message %s",  args...: "formatted")
}
```

Figura 12.11. Código del *test* funcional del microservicio ***users***

6. Descripción del producto

Este proyecto ofrece como resultado una aplicación para teléfonos celulares y web donde se logra unificar gran parte de la información sobre el turismo pesquero, brindando además un espacio con tecnología de fácil acceso.

En este apartado, se muestran y describen las diferentes funcionalidades de la aplicación desarrolladas tanto para la versión web como la *mobile*.

6.1. Pantalla de inicio

Descripción

Cuando el usuario ingresa a la web, se muestra una pantalla con breves imágenes acerca de la naturaleza de la aplicación y donde se le permitirá iniciar sesión.

Versión WEB

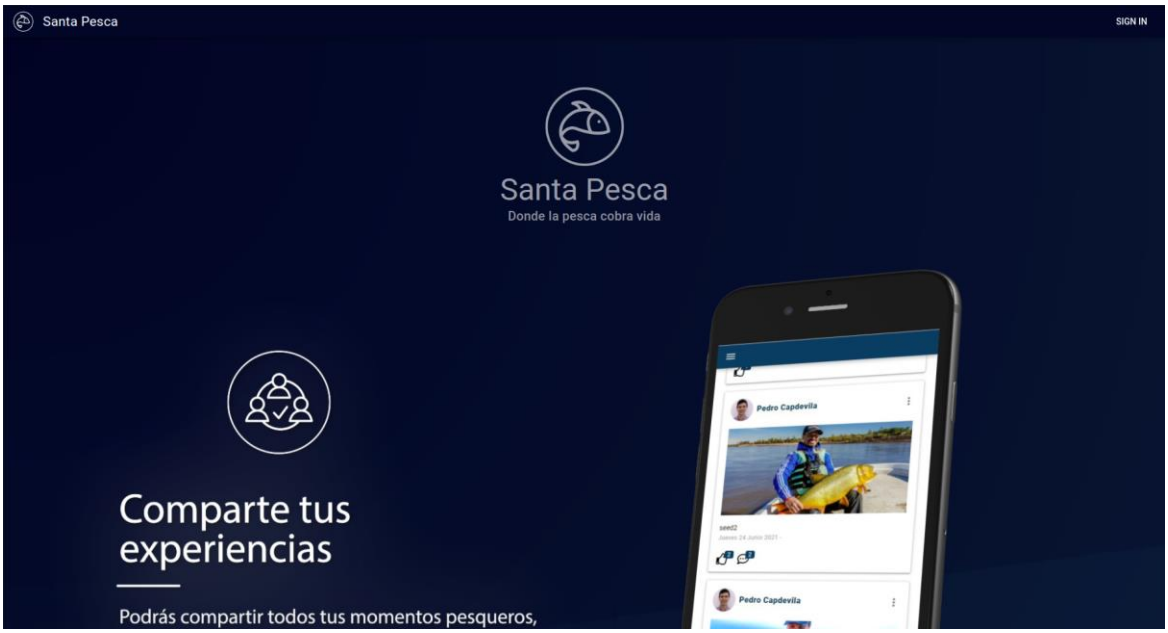


Figura 6.1.1. Pantalla de inicio - Versión WEB

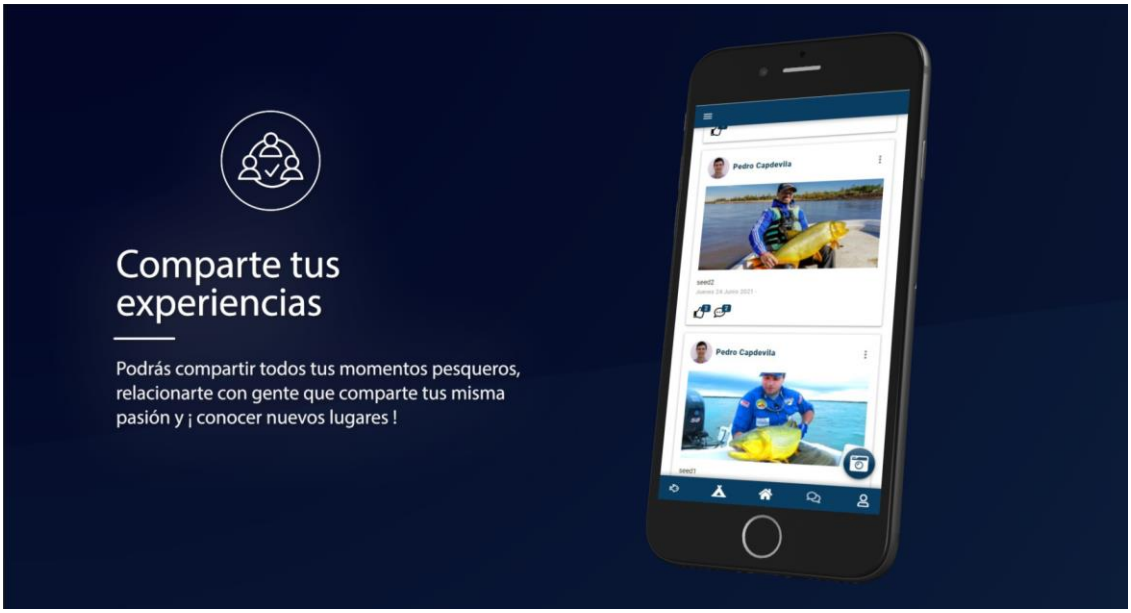


Figura 6.1.2. Pantalla de inicio - Versión WEB

Figura 6.1.3. Pantalla de inicio - Versión WEB

Versión mobile

Figura 6.2. Pantalla de inicio - Versión Mobile

6.2. Inicio de sesión

Descripción

La pantalla de inicio de sesión le permitirá al usuario realizar diferentes acciones:

1. Crear una nueva cuenta, esto lo redireccionará a la pantalla de Crear una cuenta.
2. Iniciar sesión con una cuenta del sistema, ingresando su email y contraseñas registrados.
3. Iniciar sesión con cuentas de Facebook o Google, esto lo llevará a una nueva ventana para iniciar sesión con su cuenta perteneciente a alguna de las redes sociales antes mencionadas.

4. Ingresar como invitado, si el usuario no tiene cuenta registrada puede acceder como invitado, pudiendo interactuar una versión limitada del sistema.
5. Recuperar contraseña, si el usuario olvidó su contraseña, la podrá regenerar a través de la pantalla Recuperación de contraseña.

Versión WEB

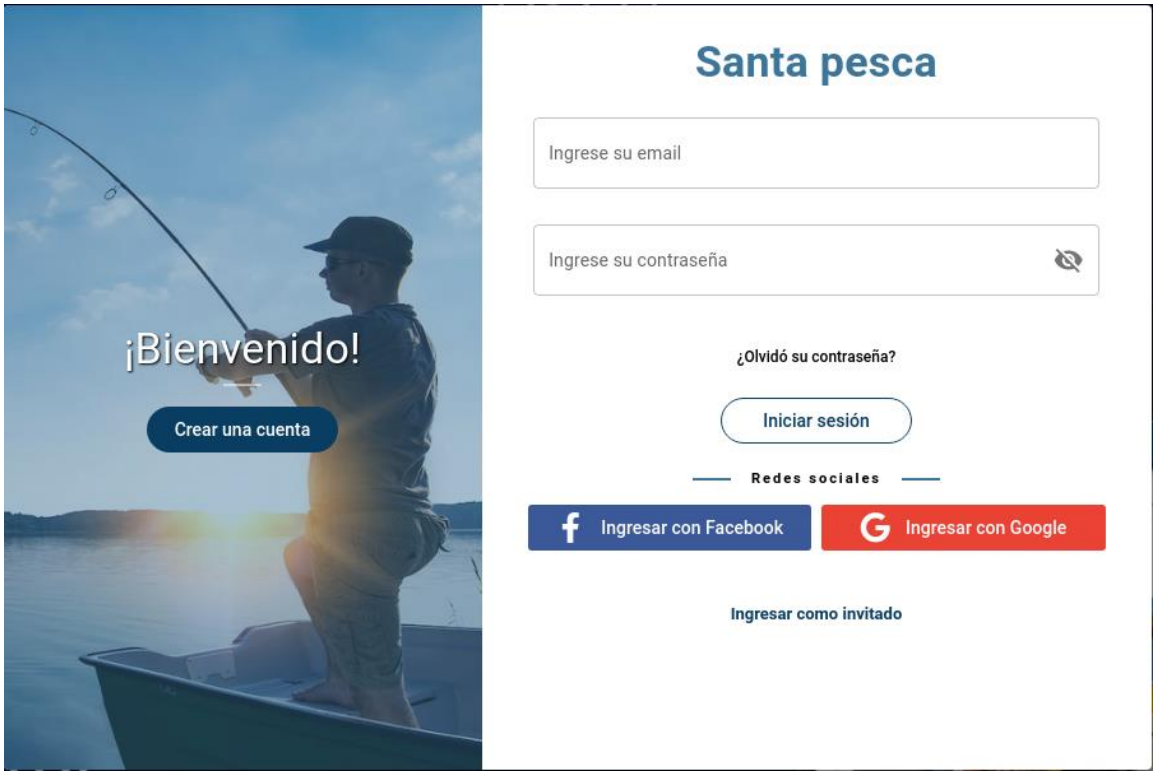


Figura 6.3. Pantalla de inicio - Versión WEB

Versión Mobile

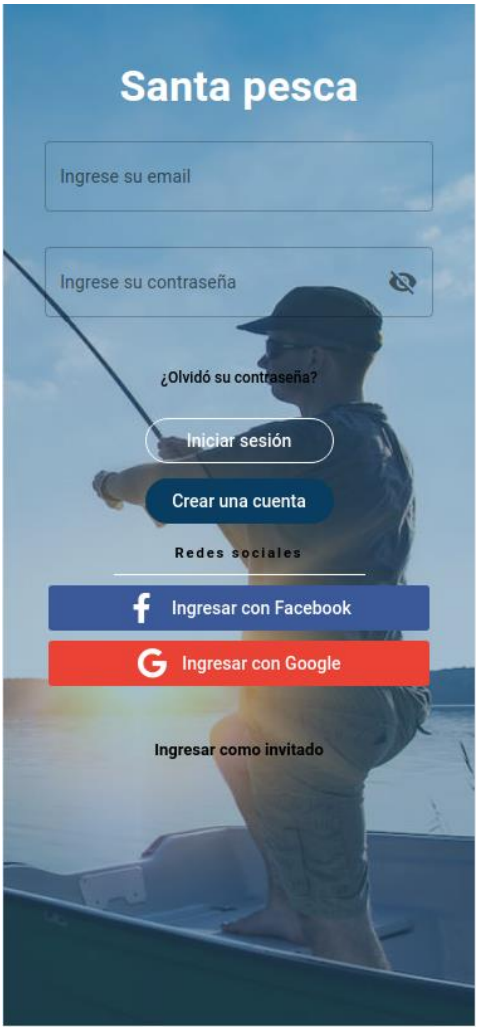


Figura 6.4. Pantalla de inicio - Versión Mobile

6.3. Crear una cuenta

Descripción

En esta pantalla, el usuario podrá dar de alta una nueva cuenta en el sistema, ingresando los datos necesarios. Cuando complete todos los campos y apriete el botón “Crear una cuenta”, se le mostrará un mensaje en la parte superior de la pantalla indicando si la cuenta se creó exitosamente o no. En caso de que se haya creado correctamente, se le redireccionará a la pantalla de inicio de sesión para que pueda ingresar al sistema con su cuenta.

Si ya posee una cuenta y entró por error a esta pantalla, podrá volver atrás mediante el botón “Iniciar sesión”.

El email es pedido al usuario para verificar que no se generen múltiples cuentas bajo el mismo email (es una *constraint* única en la base de datos relacional) y además se usa para la recuperación de dicha cuenta. Se decidió, también, que el email junto con la contraseña sean los necesarios para iniciar sesión, siempre y cuando no se use *login* con redes sociales (próxima funcionalidad detallada).

Una *feature* interesante que se podría agregar es la validación del email a través de un link. De esta manera se le daría una mejor robustez al usuario.

Versión WEB

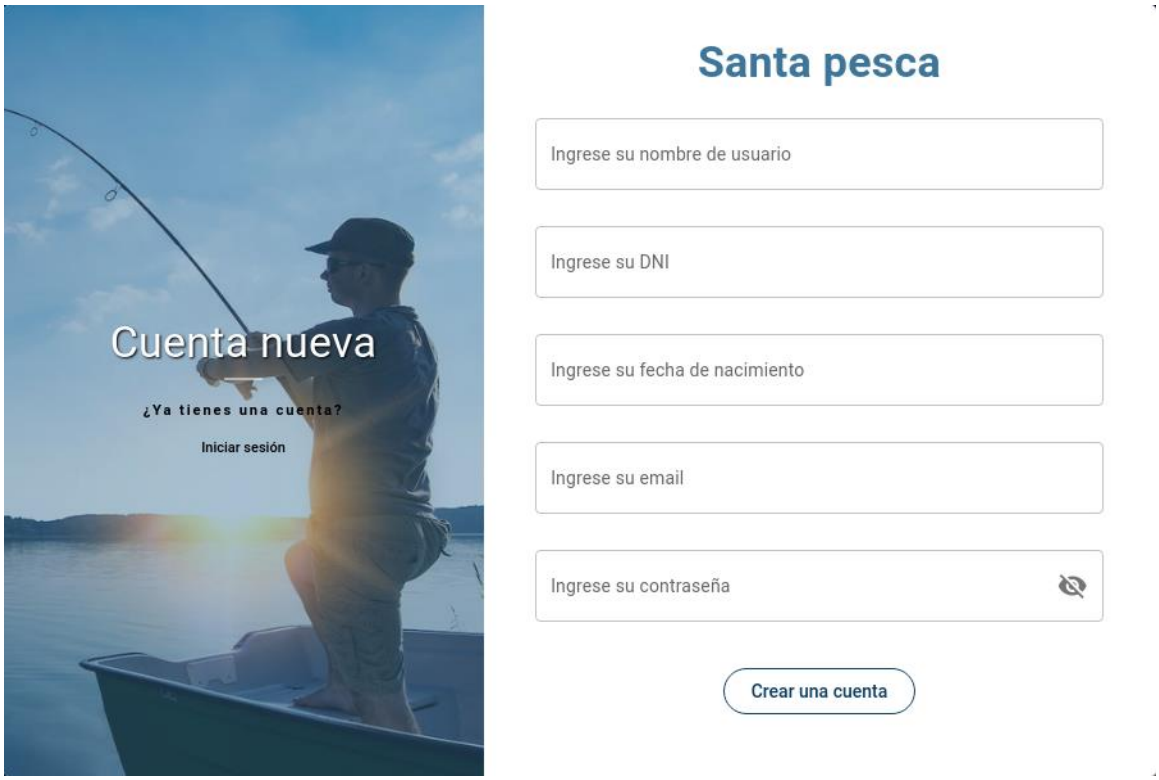


Figura 6.5. Pantalla para crear una cuenta - Versión WEB

Versión mobile

Figura 6.6. Pantalla para crear una cuenta - Versión Mobile

6.4. Iniciar sesión con cuenta de red social

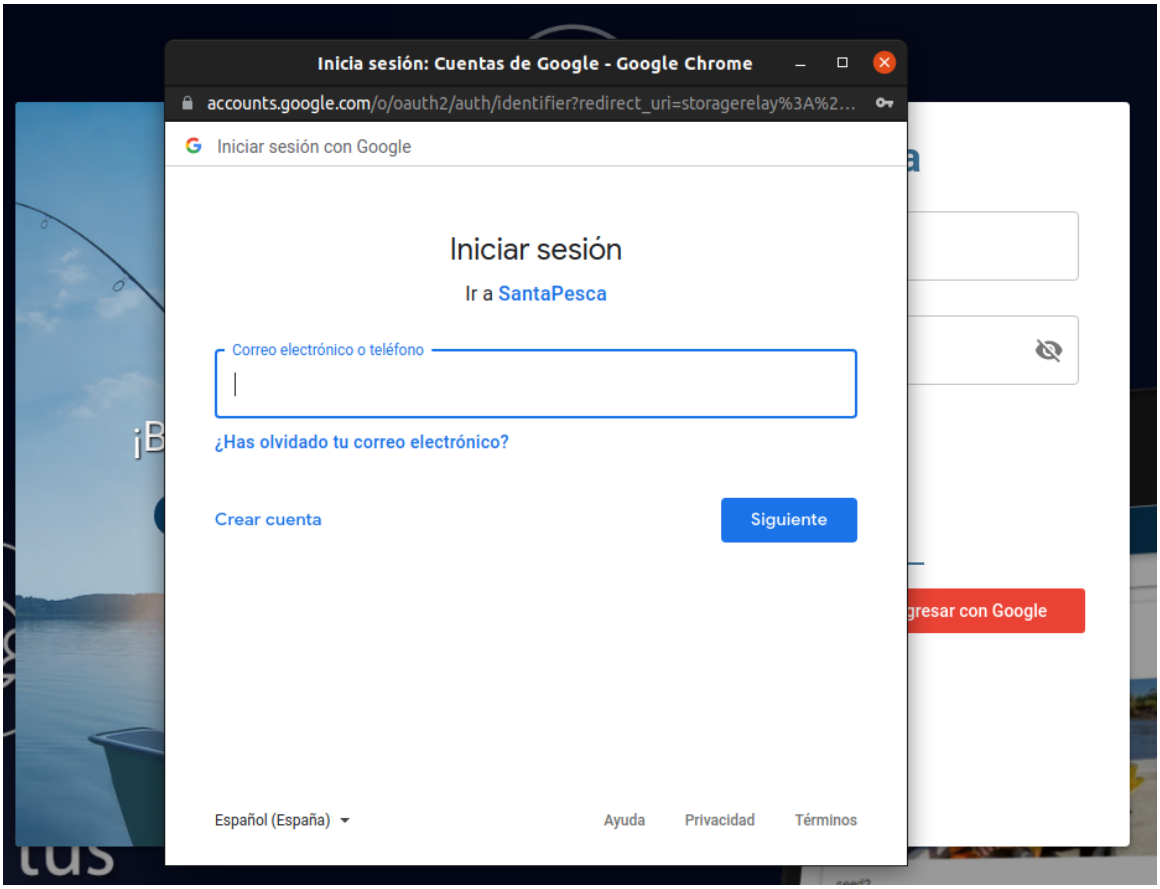


Figura 6.7. Pantalla iniciar sesión con una red social

Descripción

Si el usuario decide acceder mediante una cuenta de Facebook o Google, se le mostrará a modo de pop-up una nueva ventana que le permitirá ingresar sus correspondientes credenciales.

6.5. Recuperar contraseña

Descripción

Esta pantalla le permitirá al usuario recuperar su contraseña ingresando su email de registro. Esto le enviará un correo con una nueva *password* autogenerada al usuario para que pueda ingresar al sistema y cambiar la contraseña en su perfil.

Si no desea cambiar la contraseña, podrá volver atrás mediante el botón “Iniciar sesión”. O bien podrá directamente autenticarse con una red social, como invitado o crear una nueva cuenta.

Versión WEB

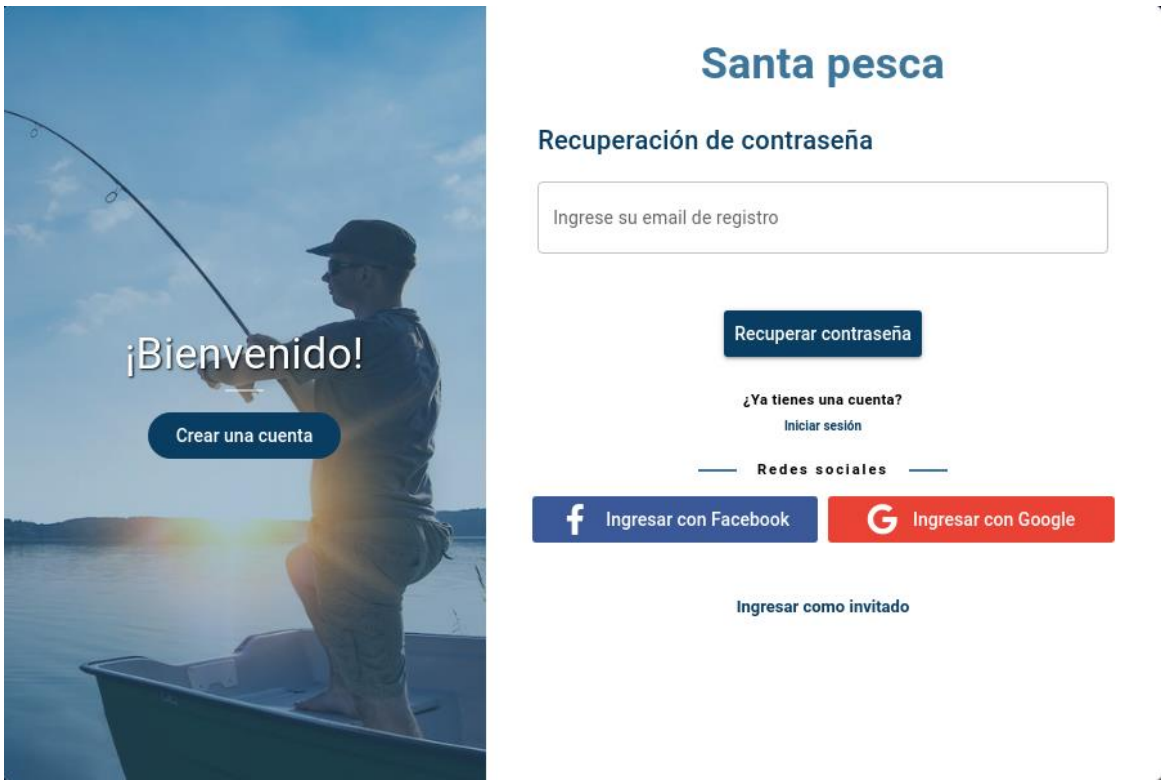


Figura 6.8. Pantalla de recuperar contraseña - Versión WEB

Versión mobile

Figura 6.9.. Pantalla de recuperar contraseña - Versión Mobile

6.6. Panel lateral

Descripción

El panel lateral estará siempre presente en la versión WEB de la aplicación y, en la versión *mobile*, se podrá desplegar haciendo *click* en el botón correspondiente (≡).

A través de este panel, el usuario podrá realizar las siguientes acciones:

- Ingresar a su perfil: Podrá hacerlo ya sea *clickeando* la imagen de perfil o el nombre. También ingresará al mismo tocando en la opción ‘Editar perfil’.
- Acceder a los elementos guardados.
- Acceder a los ajustes de la aplicación.

- Acceder a los negocios pertenecientes al usuario
- Ingresar a la pantalla donde encontrará todas las publicaciones que haya realizado.
- Cerrar sesión en el sistema.

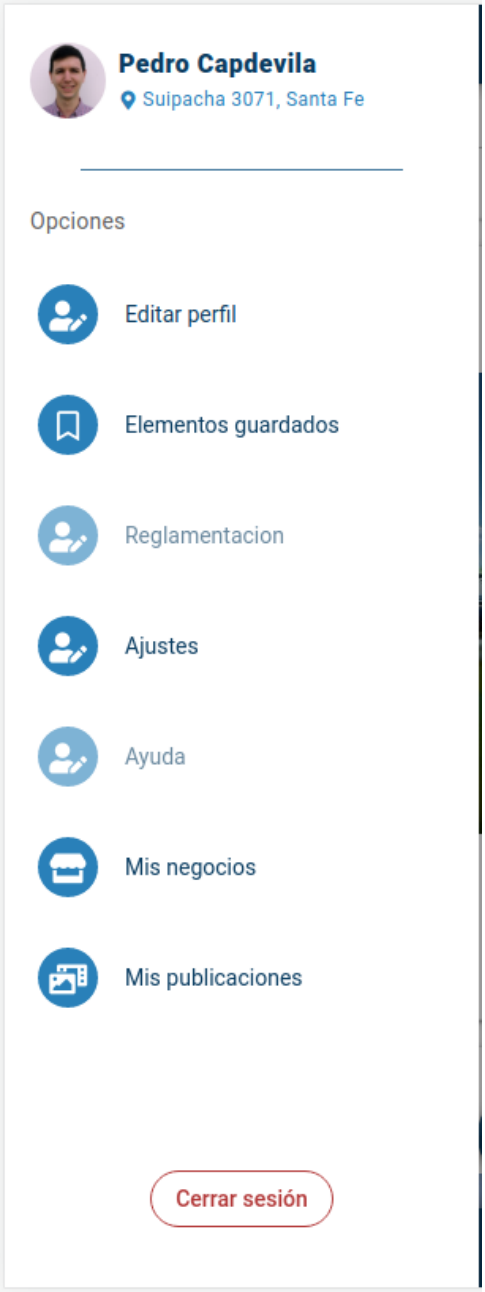


Figura 6.10.. Panel lateral

6.7. Feed

Descripción

El *feed* es la sección principal de la aplicación. En esta pantalla los usuarios podrán visualizar todas las publicaciones que hayan realizado otros usuarios

estableciendo ciertos filtros de búsqueda. Por defecto, la aplicación hará uso de la ubicación actual del usuario y tomará un radio de búsqueda de publicaciones de 1km a la redonda.

Las publicaciones del *feed* se mostrarán ordenadas por fecha de creación en caso de que el usuario no modifique los filtros por defecto y ordenadas por cercanía si el usuario configura un punto o un radio nuevo.

A continuación, se detallan todos los elementos del *feed* y la forma de interacción que el usuario tiene con estos.

Versión WEB

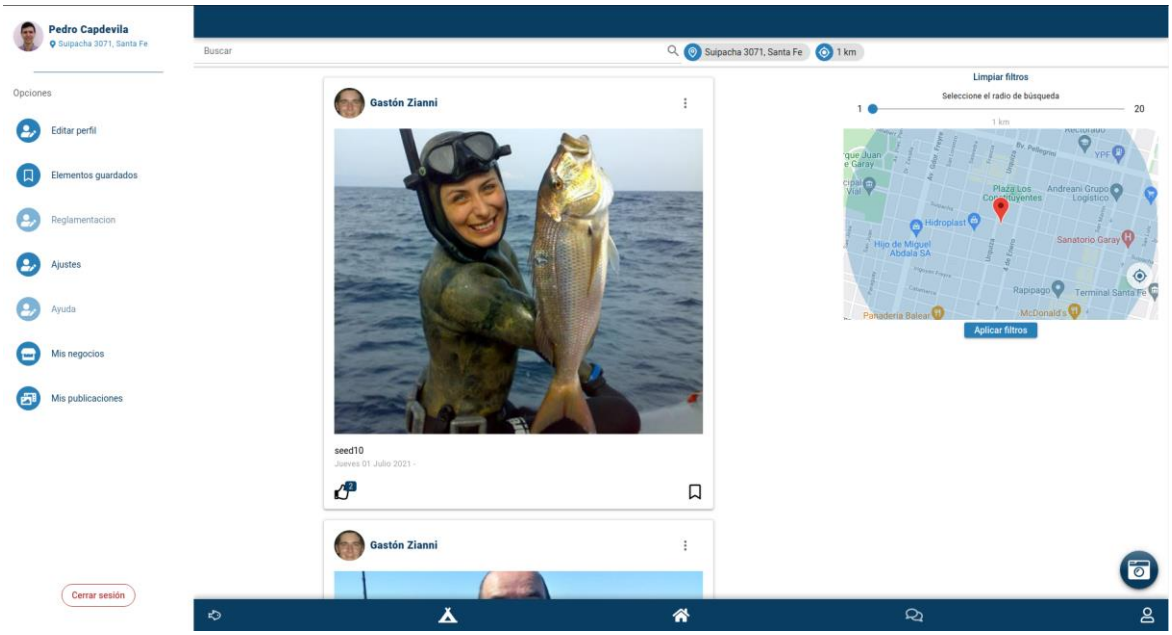


Figura 6.11.. Pantalla de del *feed* - Versión WEB

Versión Mobile

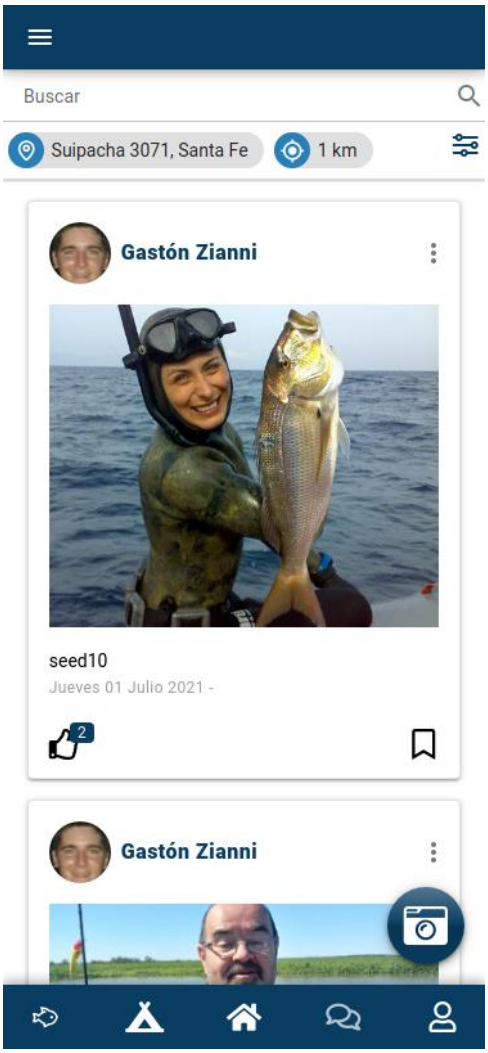


Figura 6.12.. Pantalla de del *feed* - Versión Mobile

Búsqueda de usuarios

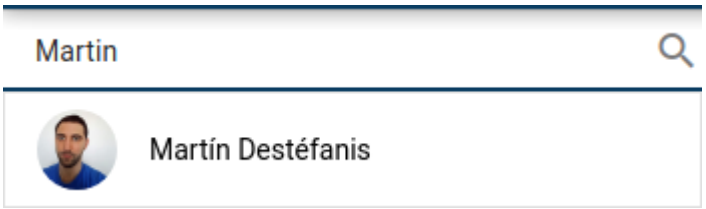


Figura 6.13.. Pantalla de búsqueda de usuarios

En la parte superior del *feed*, el usuario podrá realizar la búsqueda de otros usuarios ingresando el nombre correspondiente en el box destinado a esto. El sistema

le mostrará una lista con los resultados encontrados y, haciendo *click* en uno de ellos, podrá ir al perfil de la persona encontrada.

Filtros de búsqueda

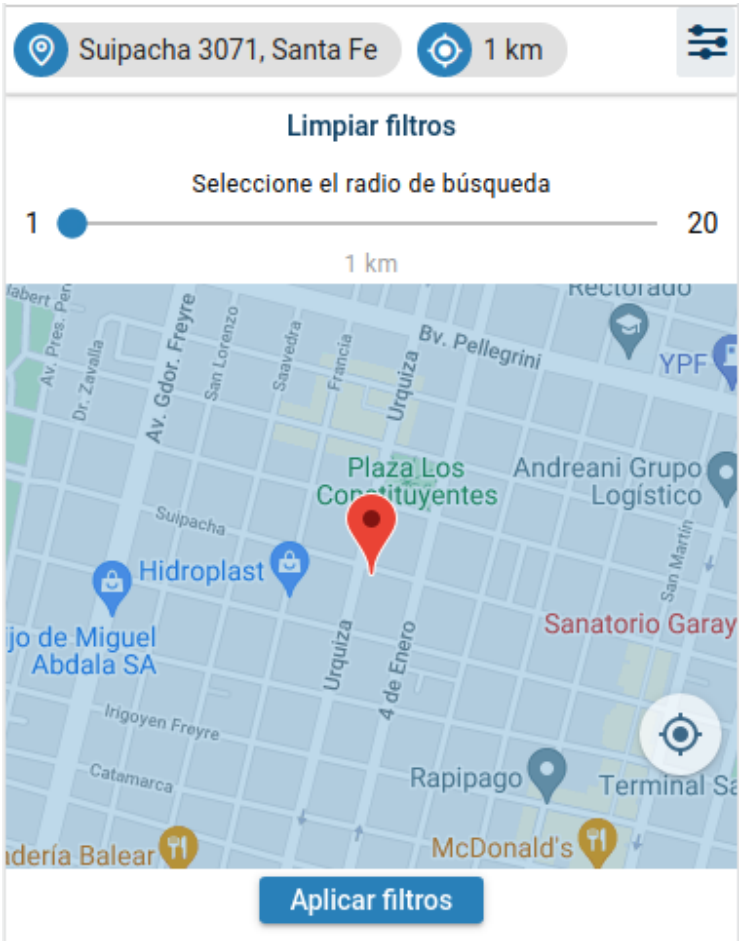


Figura 6.14.. Pantalla de filtros de búsqueda

Esta sección está siempre presente en la esquina superior derecha de la versión web y se accede a través del botón de filtros en la versión *mobile*. Aquí el usuario puede configurar un radio de búsqueda entre 1km y 20km y un punto de búsqueda haciendo uso del mapa.

Publicaciones

Figura 6.15.. Publicación

Cada publicación será visible al usuario en forma de tarjeta. En esta, se mostrará el usuario que la creó, la foto de la publicación, una descripción, la fecha de publicación, ubicación, la cantidad de *likes* y la cantidad de comentarios. A su vez, el usuario podrá interactuar con los diversos elementos de la misma.

Acceder al perfil del creador de la publicación

En cada publicación, se podrá acceder al perfil del usuario que creó la misma *clickeando* en el nombre.

Darle “like” a la publicación

El usuario podrá darle “me gusta” a la publicación *clickeando* en el ícono correspondiente. Esto lógicamente incrementará el contador de *likes* que tenga la publicación y se pintará de color indicando que el usuario *likeó* dicho posteo.

Acceder a la sección de comentarios

Al hacer *click* en el ícono de comentarios, se le mostrarán al usuario, en forma de ventana emergente, todos los comentarios que tenga la publicación:

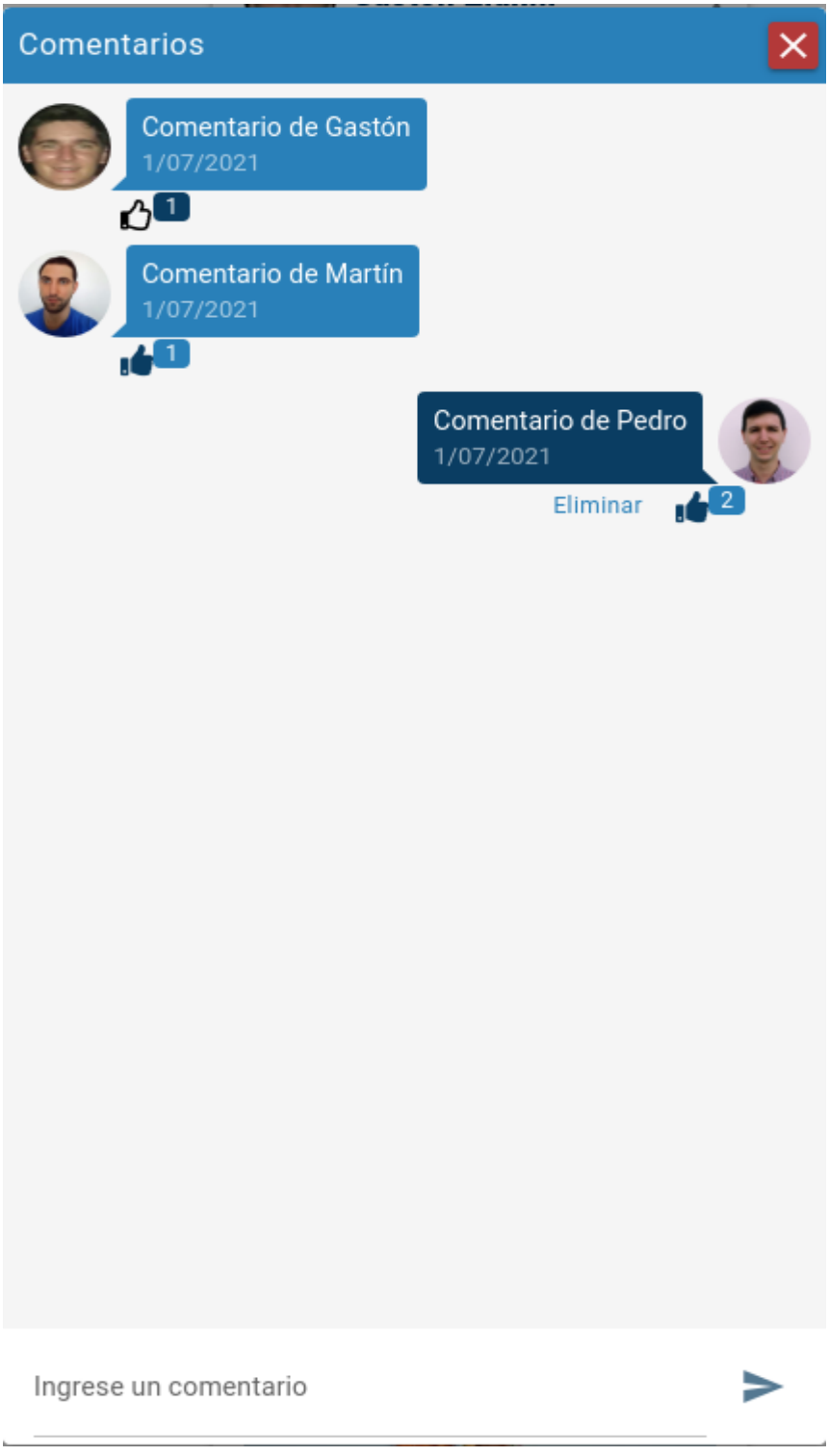


Figura 6.16.. Comentarios de una publicación

Aquí, podrá realizar un nuevo comentario, eliminar algún comentario que haya realizado, darle “me gusta” a un comentario o cerrar la ventana.

Cada comentario mostrará la fecha en la que fue publicado.

Guardar publicación

El usuario podrá guardar una publicación tocando en el ícono que se encuentra en la parte inferior derecha de la tarjeta. Esto hará que se le muestre la publicación guardada en la sección Elementos guardados.

Nueva publicación

En la versión *mobile* de la app, el usuario podrá realizar una nueva publicación sacando una foto en el momento de realizar la misma o eligiendo una del carrete de fotos de su teléfono.

Una vez que tenga la foto correspondiente a publicar, se le mostrará una pantalla donde podrá ingresar la descripción de la publicación (opcional), el lugar donde se encuentra y la especie de pez que esté en la foto (opcional). A su vez, tendrá la opción de habilitar o deshabilitar comentarios en la publicación y permitir o no que la publicación sea guardada por otros usuarios.

Cuando seleccione el botón “Guardar”, se creará la publicación y se le mostrará en el primer lugar del *feed* al usuario. A su vez, en el perfil del usuario se subirá a modo de captura la foto de la publicación.

Figura 6.17.. Pantalla de nueva publicación

6.8. Perfil

Descripción

En el perfil del usuario, éste verá en primera instancia su nombre, su foto de perfil y las capturas que haya realizado.

También podrá visualizar aquellas especies que pescó (cargadas mediante la correspondiente captura) con un contador de veces que las pudo obtener y, *clickeando* en la especie, podrá acceder a información acerca del pez.

En la versión WEB, se mostrará como información rápida del usuario su correo electrónico y su fecha de nacimiento.

Versión WEB

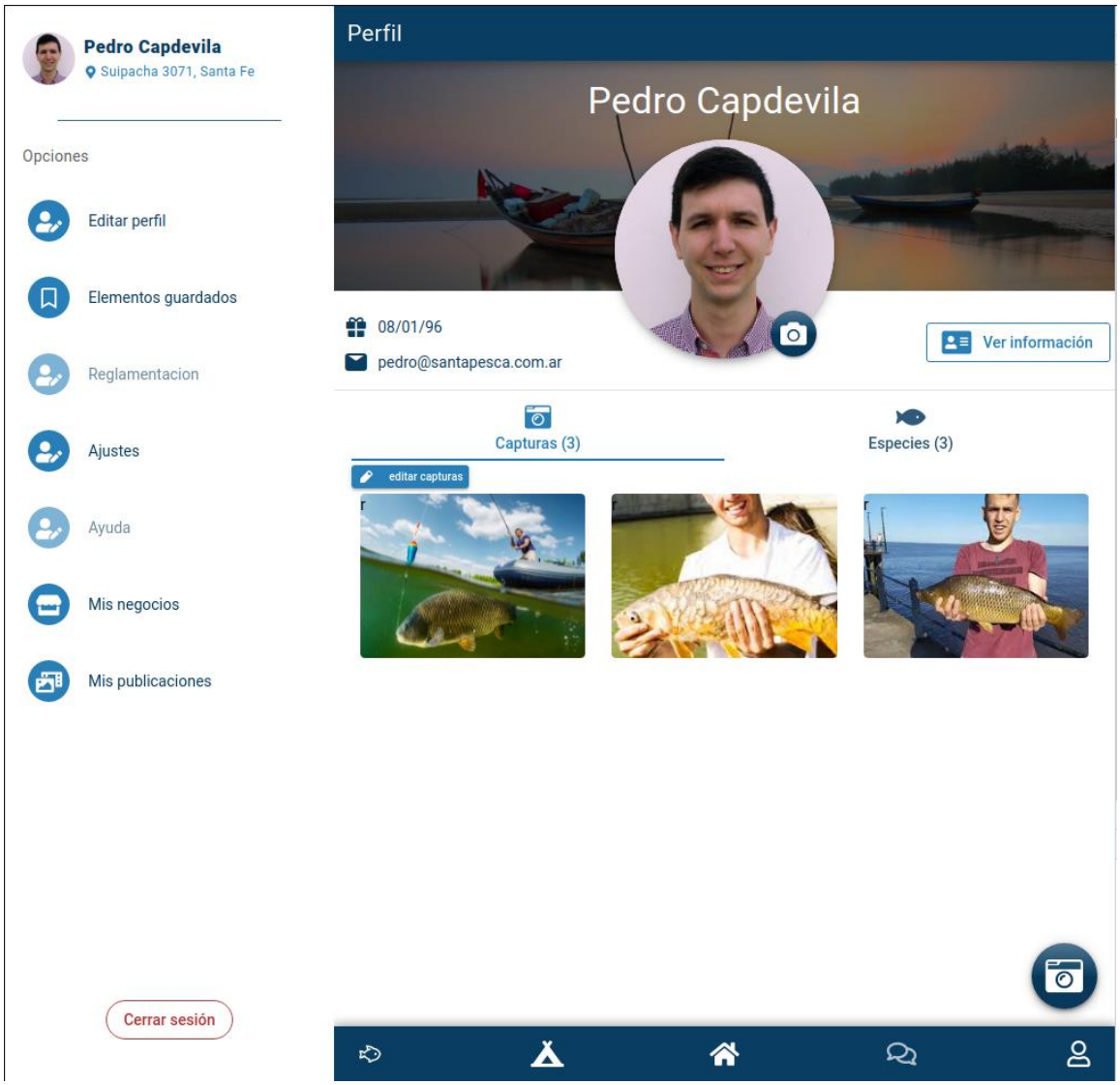


Figura 6.18.. Pantalla del perfil de un usuario - Versión WEB

Versión Mobile

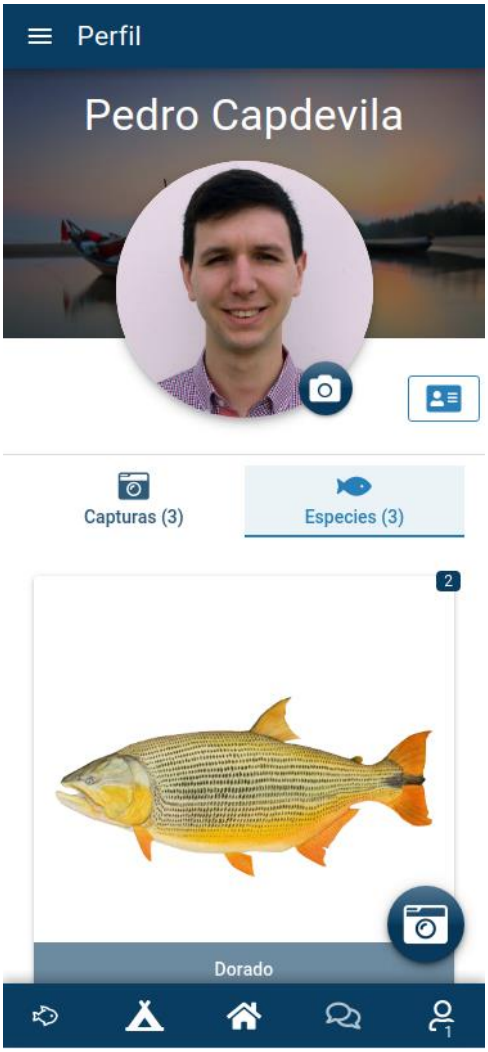


Figura 6.19.. Pantalla del perfil de un usuario - Versión Mobile

Cambiar o eliminar foto de perfil

Ingresando a través del botón con el ícono de una cámara de fotos, el usuario tendrá la opción de editar su foto de perfil, subiendo una foto nueva, o eliminarla, donde se establecerá una imagen por defecto.

Ver información completa

Al *clickear* el botón 'Ver información' el usuario podrá ver la información proporcionada al momento de registrarse en el sistema:

- Nombre
- Email
- DNI
- Fecha de nacimiento

Todos estos datos podrán ser editados en esta pantalla así como también se podrá cambiar la contraseña de la cuenta.

Eliminar capturas realizadas

Al tocar el botón '*Editar capturas'*, se mostrará sobre cada una de estas un botón que permitirá eliminarlas del perfil. Cuando elimine una de captura, aparecerá una ventana pidiendo confirmación.

Subir una nueva captura

Al hacer click en botón flotante de la esquina inferior derecha, el usuario podrá subir una nueva captura a su perfil ya sea sacando una foto en el momento con la cámara del teléfono (versión *mobile*) o bien eligiendo una foto del carrete (versiones WEB y *mobile*).

6.9. Especies

Descripción

En la pantalla de especies, el usuario podrá visualizar, a modo informativo, todas las especies y cuáles de ellas están en veda, es decir, que no pueden ser pescadas. Además, haciendo *click* en una especie, ingresará a la pantalla de 'Información de una especie' que se detalla a continuación.

Versión WEB

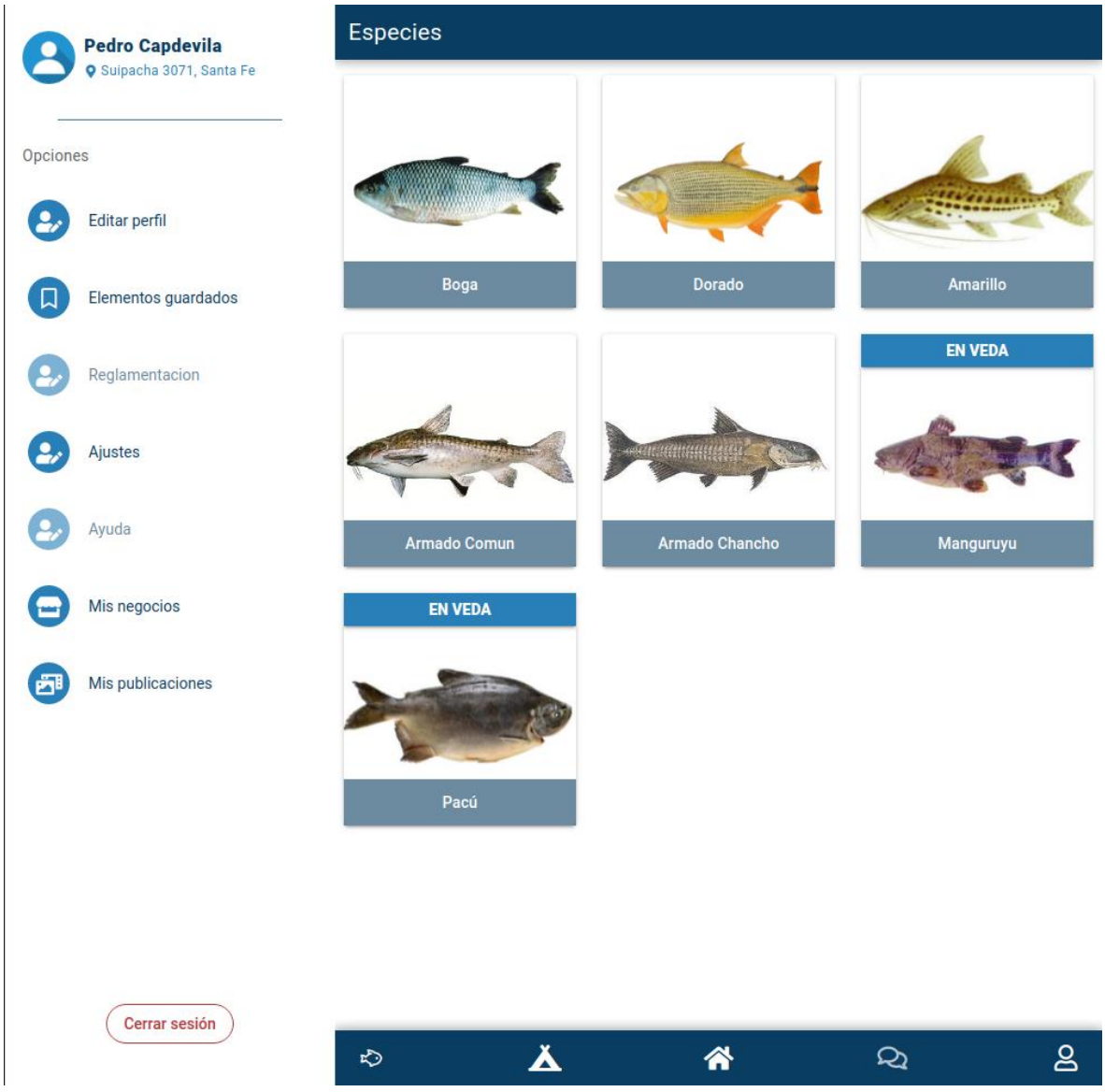


Figura 6.20.. Pantalla de especies - Versión WEB

Versión Mobile

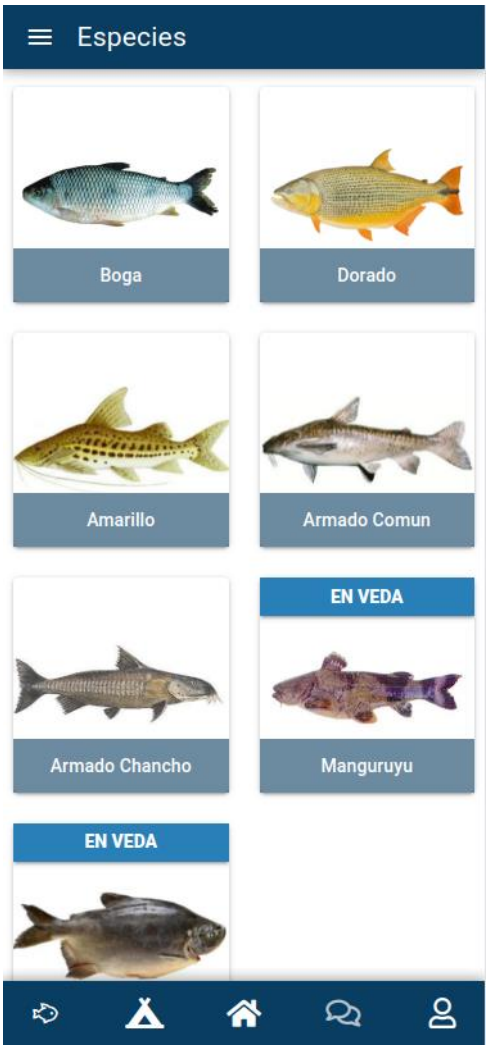


Figura 6.21.. Pantalla de especies - Versión Mobile

Información de una especie

Figura 6.22.. Pantalla información de una especie

En esta sección, se mostrará información acerca de una especie en particular:

- Nombre científico.
- Longitud promedio.
- Peso promedio.
- Época de pesca.
- Descripción de la especie.
- Modalidades de pesca.

6.10. Negocios

Descripción

Cuando el usuario ingrese a la sección de negocios, verá en primera instancia un listado de los diferentes negocios que existen en la plataforma filtrados por defecto a 1km de distancia desde la ubicación actual. También se mostrará en la parte superior cuáles son los filtros actuales aplicados.

De cada negocio, se mostrará información básica del mismo:

- Foto del negocio.
- Nombre.
- Calificación.
- Descripción.
- Contacto (teléfono, email, red social, etc.)
- Distancia desde la ubicación actual del usuario.
- Botón para guardar el negocio.

A su vez, el usuario podrá ingresar a ver más información de un negocio haciendo *click* en el botón '*Ver más'*.

El usuario también podrá cambiar los filtros de búsqueda de los negocios. Los filtros que podrá utilizar son:

- Nombre del negocio.
- Tipo de negocio. Puede ser cabaña, guardería, bajada de lancha o negocio.
- Características del negocio. Son características o servicios que el negocio debe tener, por ejemplo, se podrán buscar cabañas que cuenten con TV y aire acondicionado.
- Calificación. Se buscarán negocios que cumplan con una calificación mínima.
- Ordenar por cercanía los negocios encontrados.
- Radio de búsqueda.
- Ubicación de búsqueda.

Una vez que el usuario haya completado todos los filtros que considere, podrá aplicarlos mediante el botón 'Aplicar filtros'. A su vez, podrá también deshacer cualquier filtro aplicado con el botón 'Limpiar filtros'.

Se detallan a continuación otros elementos de esta pantalla.

Versión WEB

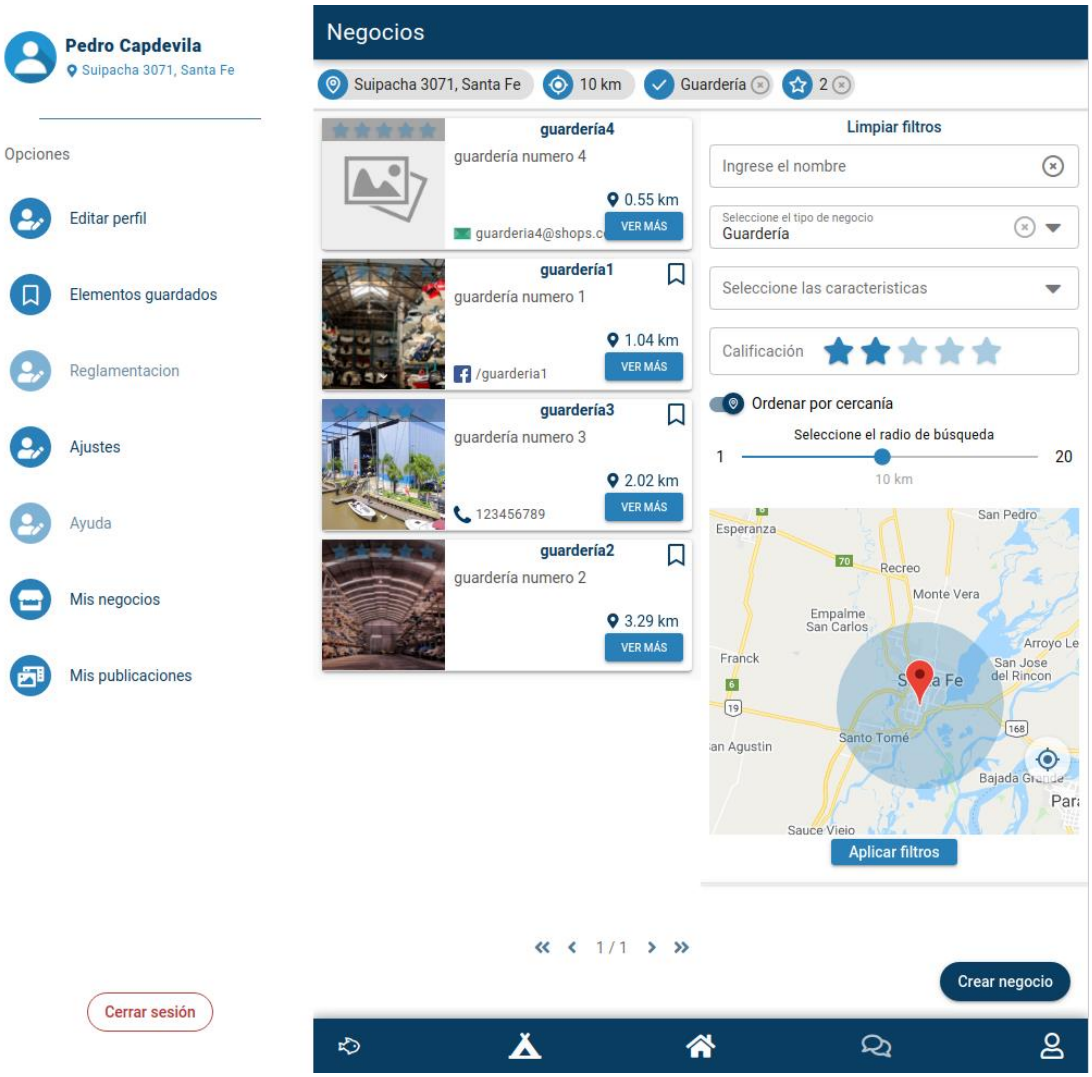


Figura 6.23.. Pantalla de negocios - Versión WEB

Versión Mobile

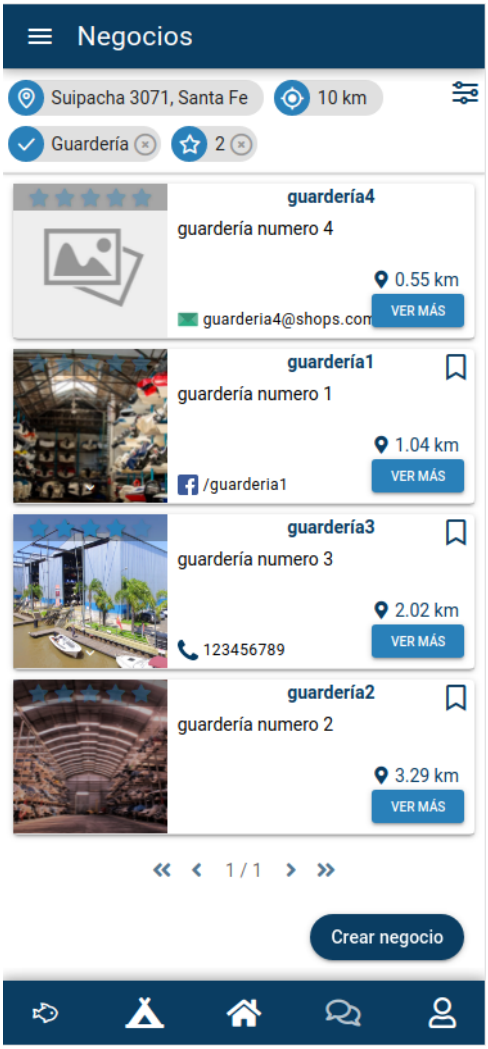


Figura 6.24.. Pantalla de negocios - Versión Mobile

Ver más información de un negocio

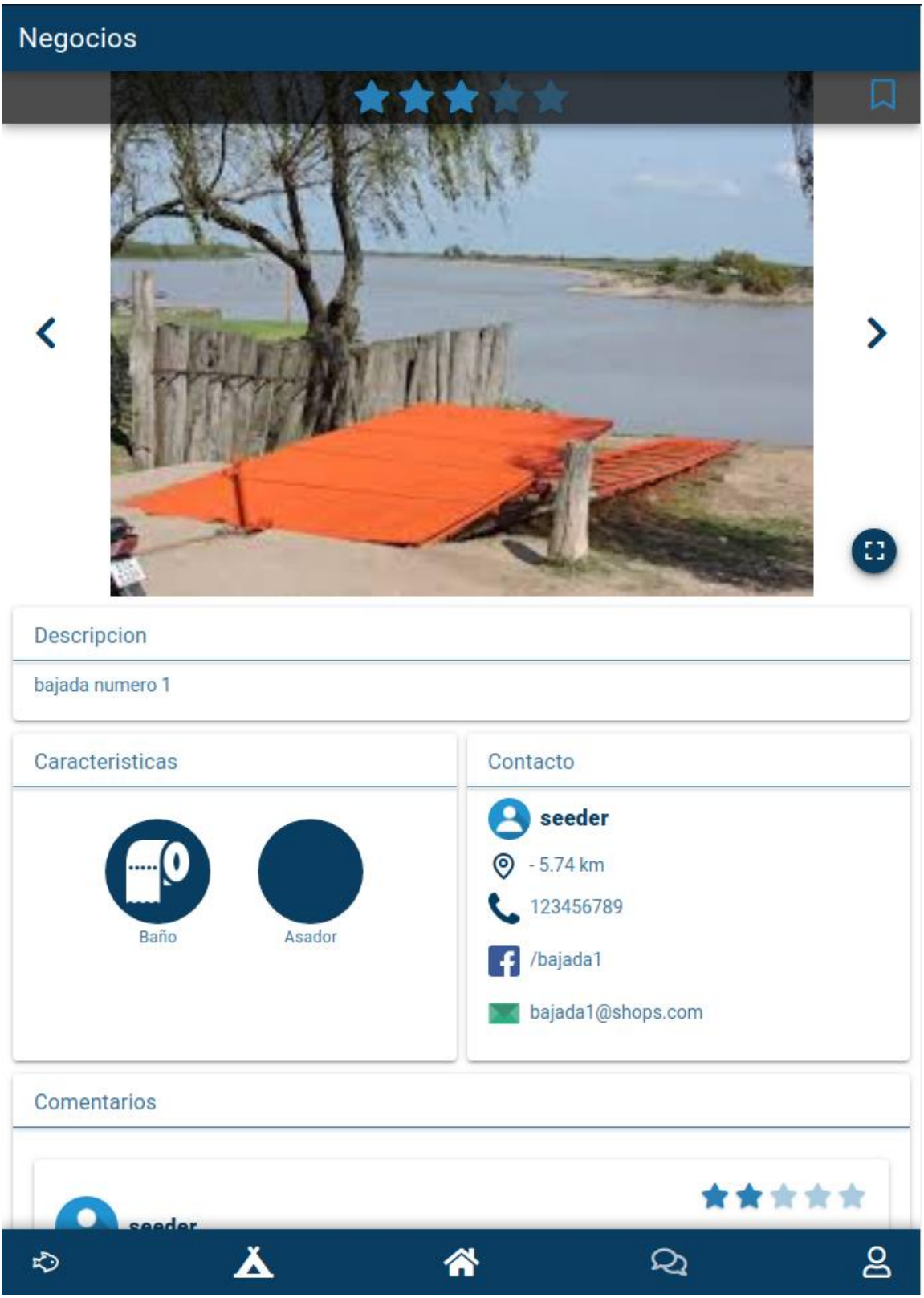


Figura 6.25.1.. Pantalla de información de un negocio

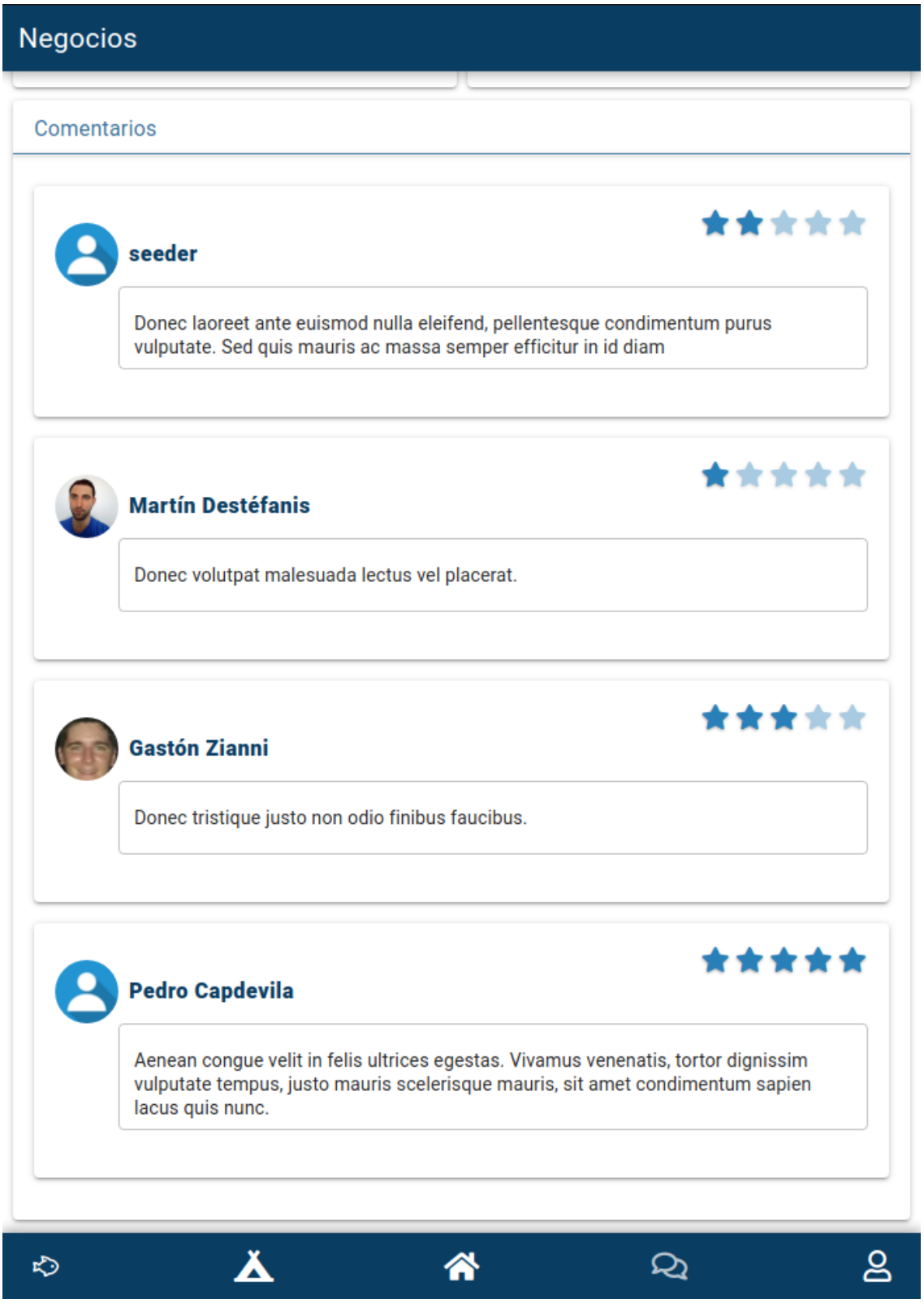


Figura 6.25.2.. Pantalla de información de un negocio

Cuando el usuario ingrese a esta pantalla, podrá ver con mayor detalle la información del negocio:

- Carrete de fotos del negocio. También, las podrá ver en pantalla completa accediendo mediante el botón correspondiente para agrandar la foto.
- Calificación promedio del negocio. Se calcula en base a las calificaciones de las reseñas.
- Botón para guardar el negocio.
- Descripción.
- Características.
- Información de contacto.
- Reseñas. En las reseñas, se mostrará un listado con los comentarios y calificaciones que los usuarios hicieron sobre el negocio. Aquí el usuario podrá dejar su propia reseña, si es que no lo ha hecho con anterioridad.

Guardar un negocio

Tanto en la pantalla principal de negocios como en la pantalla de información detallada de un negocio, el usuario podrá guardarlo en ‘Elementos guardados’.

Crear un negocio

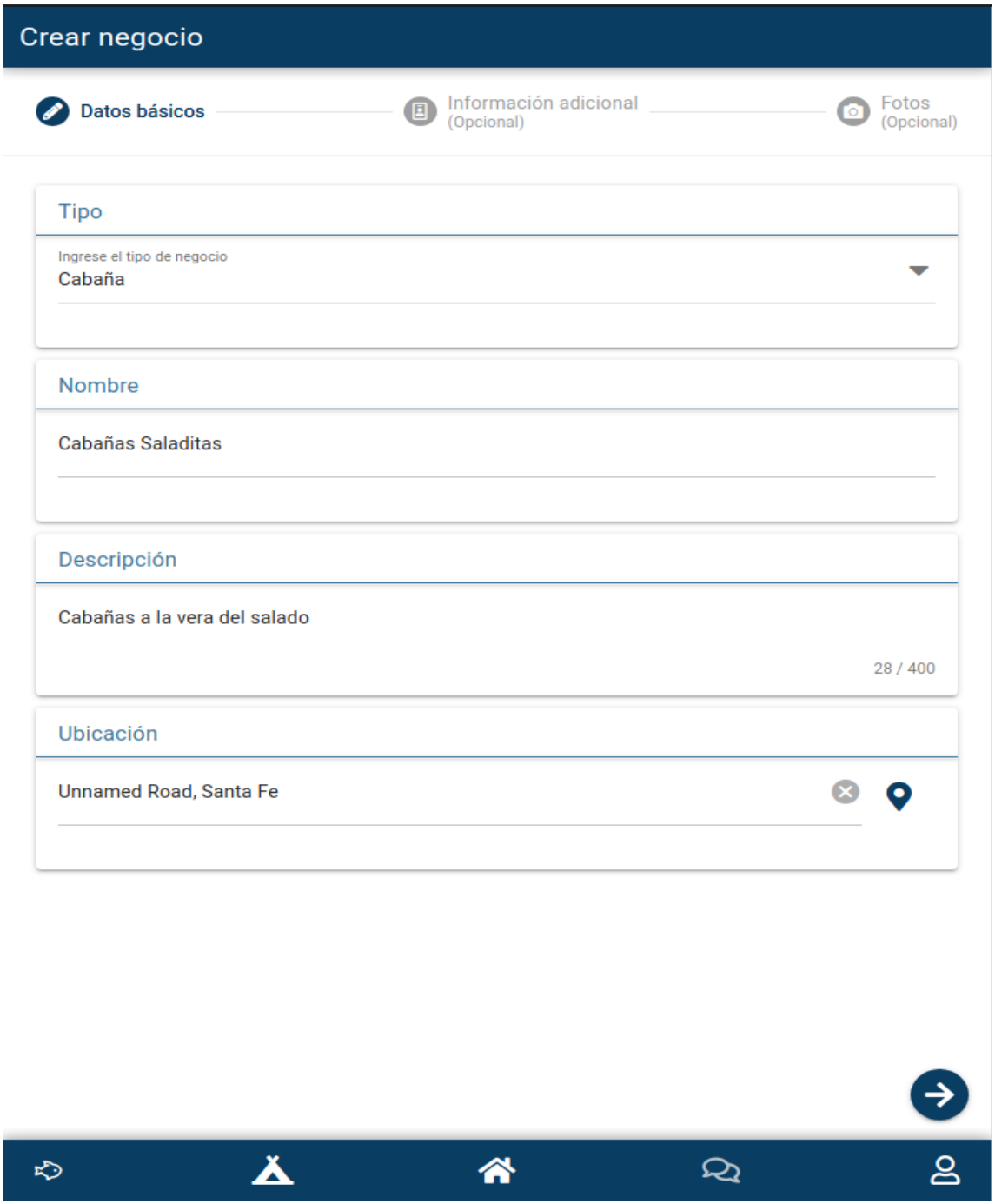


Figura 6.26.1.. Pantalla de crear un negocio

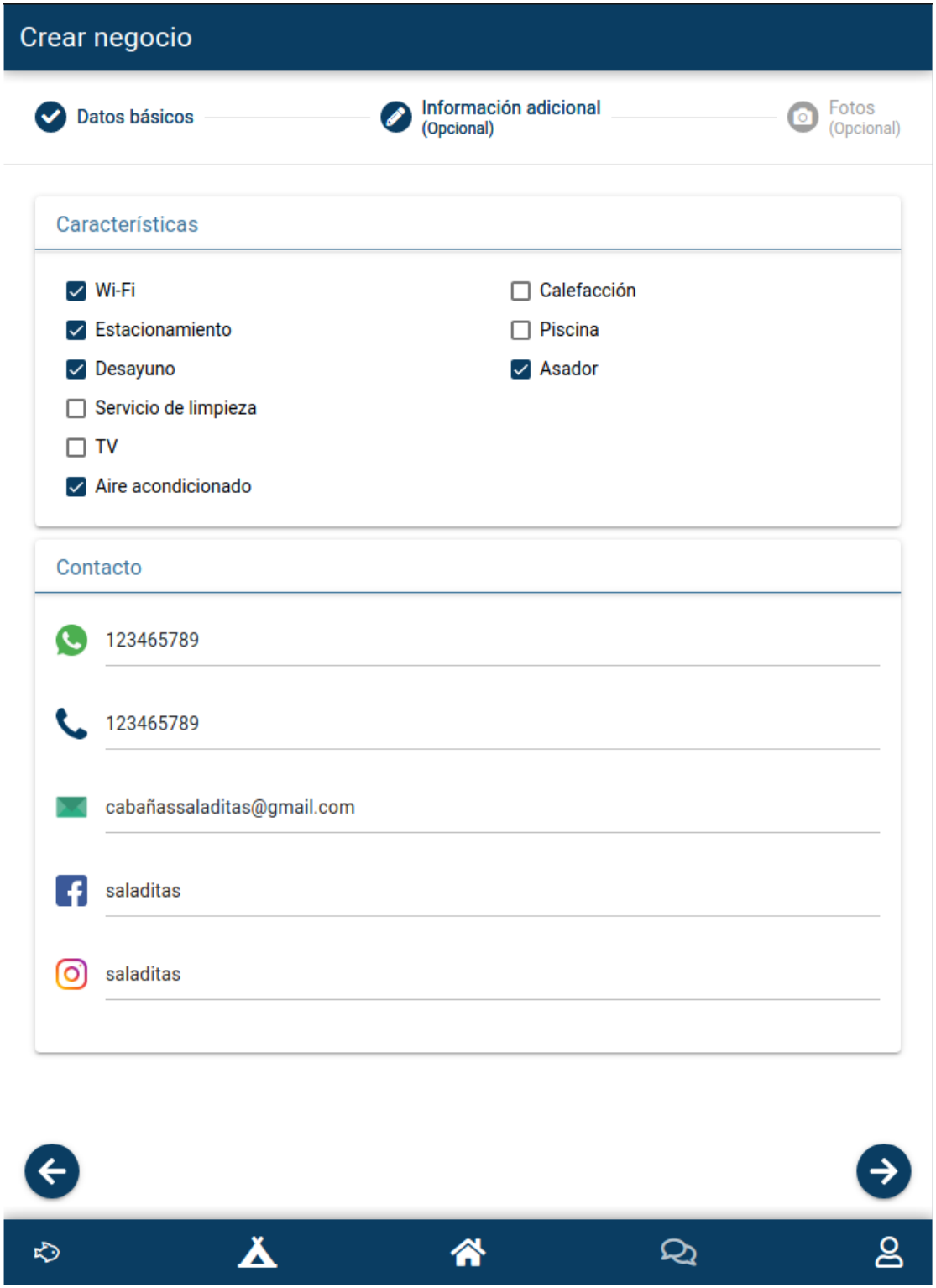


Figura 6.26.2.. Pantalla de crear un negocio

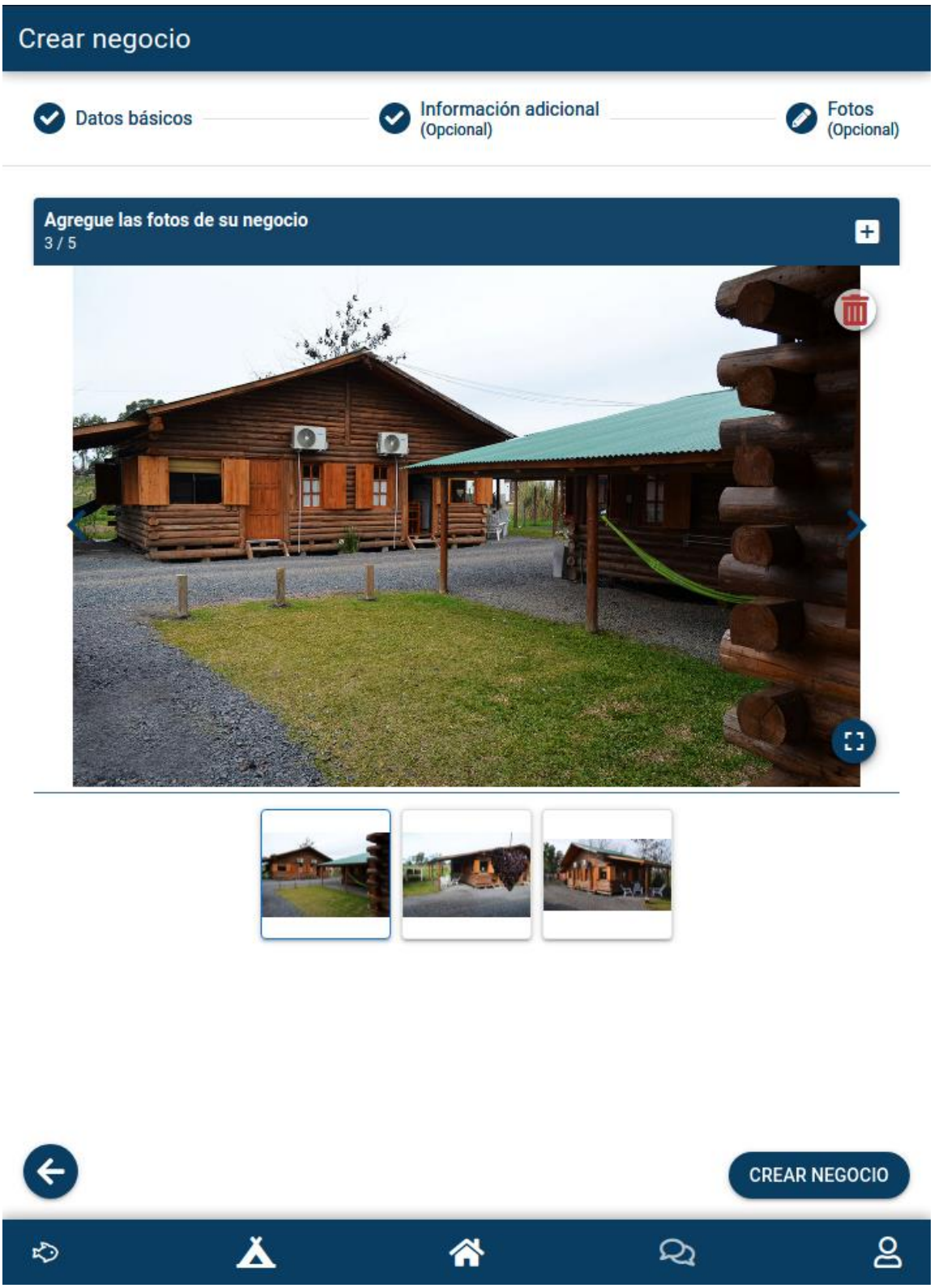


Figura 6.26.3.. Pantalla de crear un negocio

En la sección *Crear negocio* el usuario podrá cargar un negocio propio en el sistema. Para ello deberá ingresar en primera instancia, los datos básicos del negocio:

- Tipo de negocio.
- Nombre.
- Descripción.
- Ubicación.

Una vez que haya cargado todos los datos, podrá acceder a la pantalla de carga de Información adicional, tocando en el ícono de la 'flecha', donde podrá ingresar datos opcionales del negocio:

- Características.
- Contacto.

Por último, podrá agregar a su negocio hasta 5 fotos. Luego, al apoyar sobre el botón 'Crear Negocio' finalmente se cargará el negocio en el sistema y el usuario podrá visualizarlo en la sección 'Mis negocios'.

6.11. Elementos guardados

Descripción

En esta sección, el usuario podrá visualizar aquellas publicaciones, negocios y ubicaciones que haya guardado. Cada una de ella se mostrará en forma de tarjeta en su correspondiente sección. Cada tarjeta tendrá información básica del elemento guardado, con un botón para verlo en detalle y otro para quitarlo de los elementos guardados.

Versión WEB

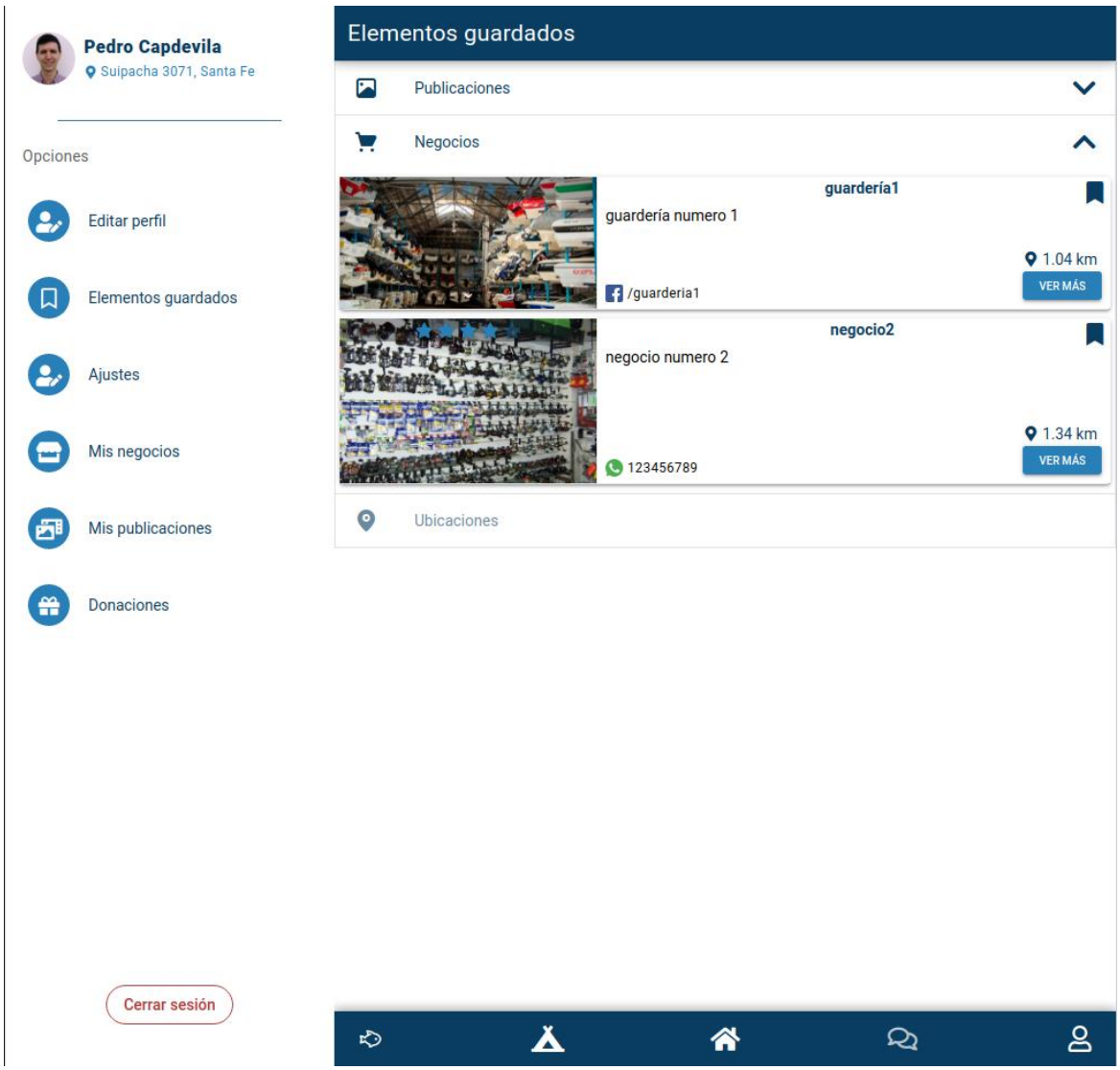


Figura 6.27. Pantalla de elementos guardados - Versión WEB

Versión Mobile

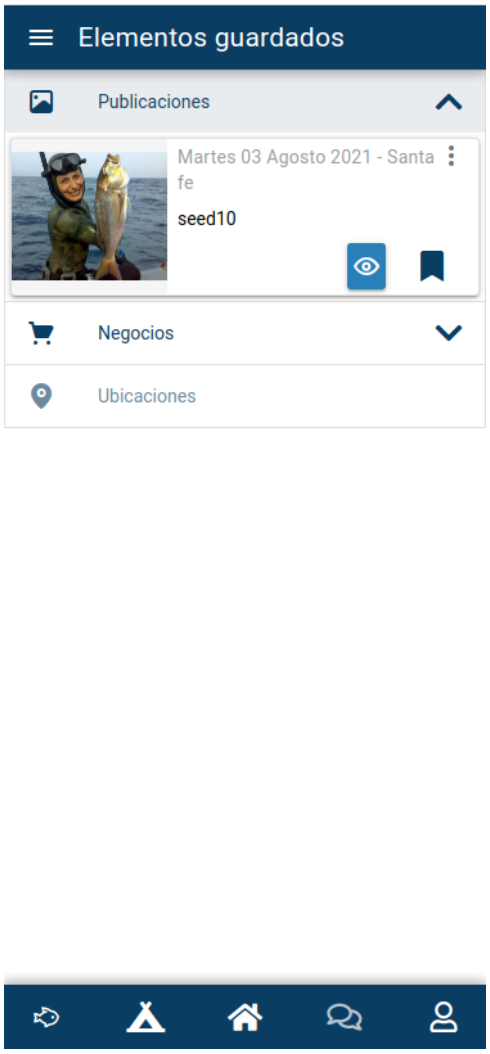


Figura 6.28. Pantalla de elementos guardados - Versión Mobile

6.12. Mis negocios

Descripción

En esta pantalla, el usuario podrá visualizar todos los negocios que tenga creados bajo su cuenta. Cada uno de sus negocios se mostrará en una tarjeta con información básica del mismo:

- Nombre.
- Descripción.
- Contacto.
- Foto.
- Rating.
- Distancia desde la ubicación actual.

Además, *clickeando* en el botón ‘Ver más’ podrá ir a la pantalla del negocio donde podrá ver la información completa del mismo.

Versión WEB

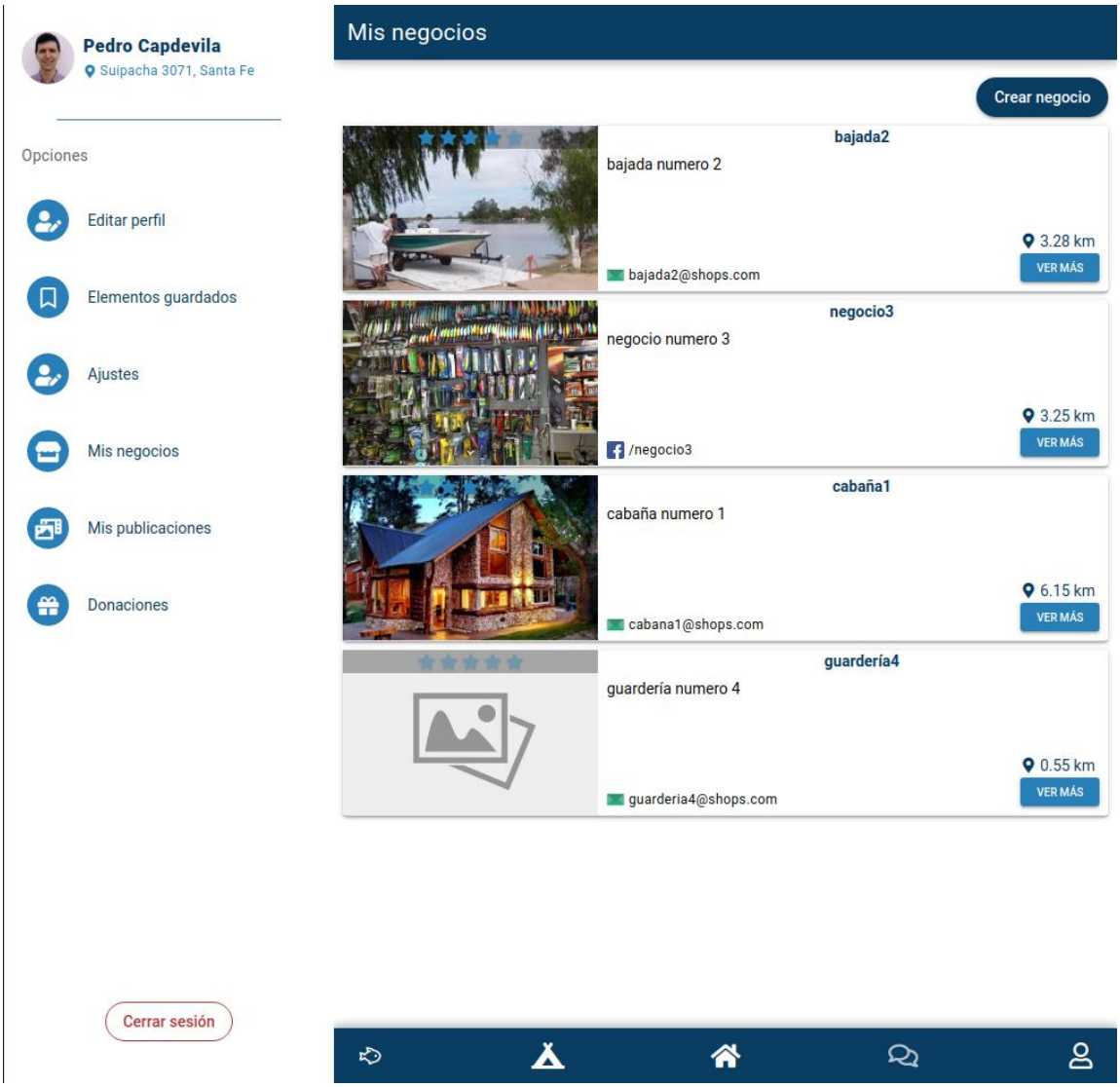


Figura 6.29. Pantalla de mis negocios - Versión WEB

Versión mobile

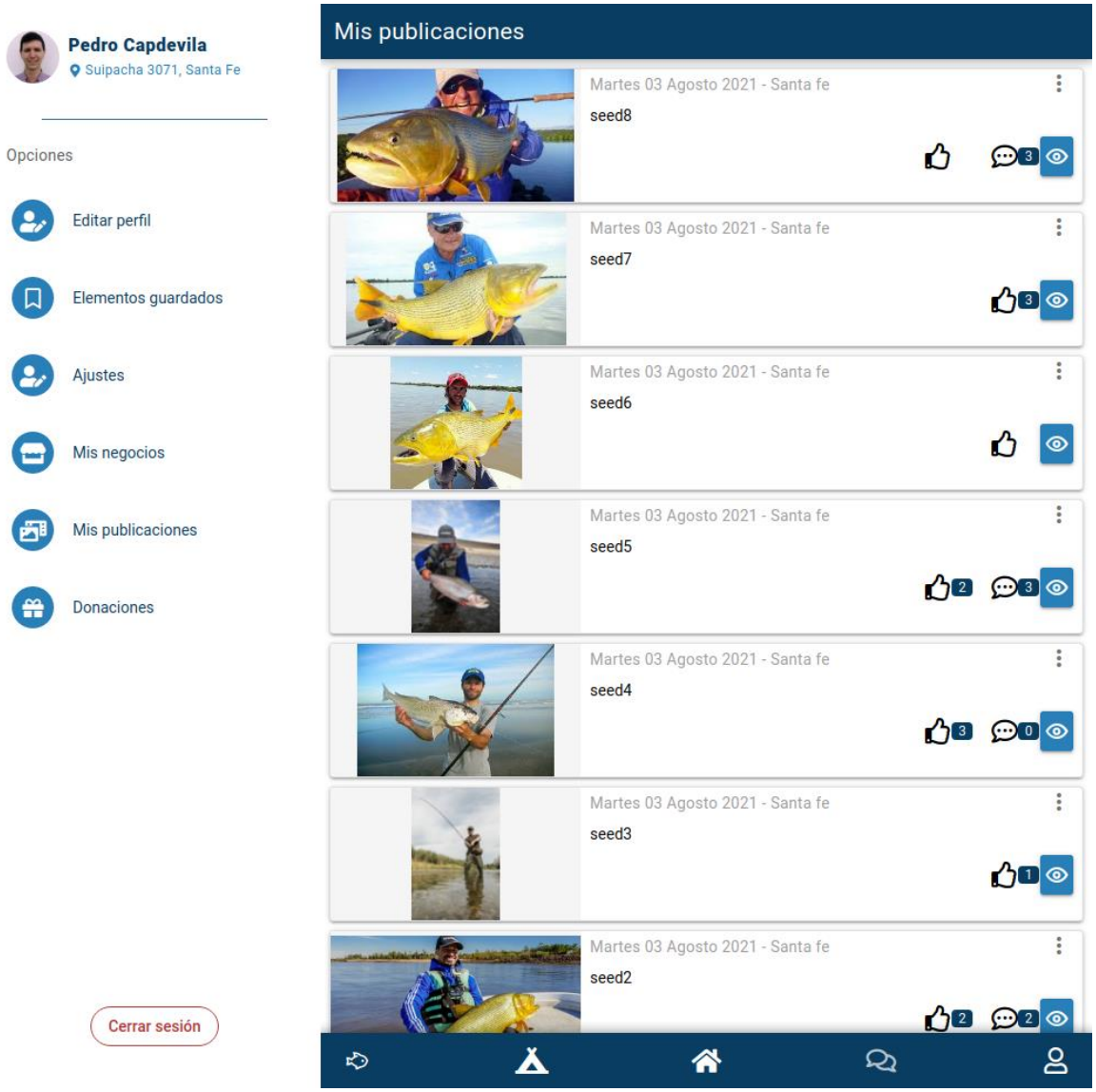
Figura 6.30. Pantalla de mis negocios - Versión Mobile

6.13. Mis publicaciones

Descripción

Al igual que con 'Mis negocios' aquí el usuario podrá visualizar todas las publicaciones que haya realizado en el *feed*. Se mostrará la foto, fecha, lugar, descripción y los botones para agregar o quitar un me gusta y para acceder a la sección de comentarios de la publicación. A su vez, tocando en el botón con el ojo, podrá ver la publicación completa en el mismo formato que aparece en el *feed*.

Por otro lado, *clickeando* en los 3 puntos de la esquina superior derecha, el usuario podrá eliminar la publicación.



Versión WEB

Figura 6.31. Pantalla de mis publicaciones - Versión WEB

Versión mobile

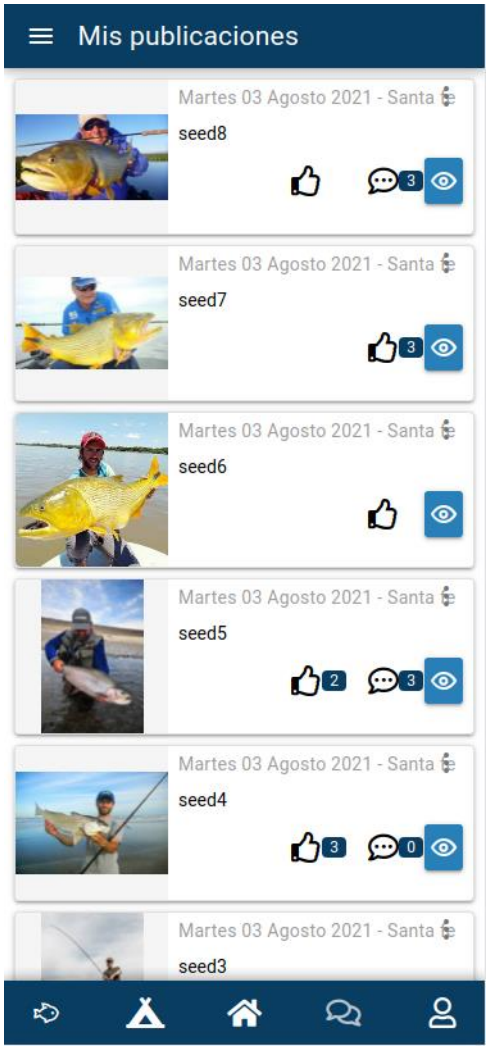


Figura 6.32. Pantalla de mis publicaciones - Versión Mobile

6.14. Donaciones

Descripción

En esta sección, el usuario encontrará información acerca de por qué es importante donar a la aplicación, las funcionalidades que requieren de mantenimiento por parte de los desarrolladores, las funciones pendientes de desarrollo que agregan valor al sistema y un link y QR a PayPal para que los usuarios puedan realizar su donación.

Versión WEB

Figura 6.33.1. Pantalla de donaciones - Versión WEB

Figura 6.33.2. Pantalla de donaciones - Versión WEB

Versión mobile

Figura 6.34.1. Pantalla de donaciones - Versión Mobile

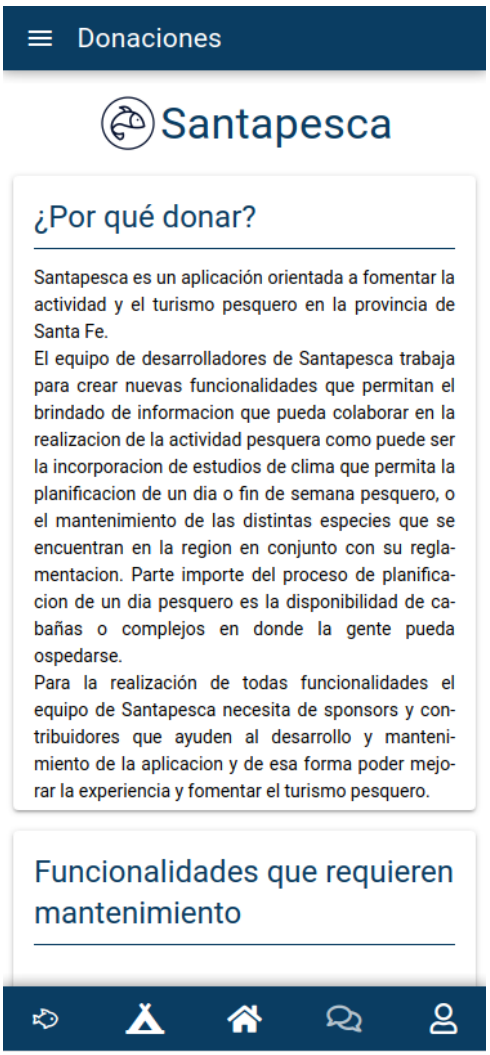


Figura 6.34.2. Pantalla de donaciones - Versión Mobile

7. Extensibilidad

A lo largo del desarrollo de las iteraciones, fueron surgiendo diferentes ideas que podrían ser tenidas en cuenta en futuros desarrollos de la aplicación:

Chat entre usuarios

La idea sería que los usuarios puedan intercambiar mensajes con otros o crear grupos de chats donde se puedan comunicar más de 2 usuarios a la vez. El objetivo principal era que los usuarios puedan intercambiar información sobre especies que se pueden pescar en algún lugar, o bien, simplemente charlar entre sí.

Notificaciones en el celular

Esta idea surgió al momento de realizar las publicaciones, donde sería de gran utilidad informarle al usuario cuando alguien interactuó con alguna de sus publicaciones, cuando le responden alguno de sus comentarios o, una vez implementado el chat, si alguien le envía un mensaje.

Manual del usuario

El requerimiento para este caso sería tener una sección dentro del sistema donde el usuario pueda ir a consultar como utilizar alguna de las funciones del mismo de forma correcta.

Usuarios premium

Como se comentaba en las etapas iniciales del proyecto, una de las formas de obtener ganancias era a través de usuarios premium que abonen un fee mensual.

Shop

Una funcionalidad atractiva que se nos ocurrió a lo largo del proceso de desarrollo que nos podría permitir generar ingresos adicionales, es la de la implementación de un mercado de pesca dentro de la aplicación. La forma en la que nos imaginamos esto es exponiendo para su venta reels, cañas, chicotes, cajas de pesca, carpas y mucha más variedad de artículos de pesca y permitir que el usuario realice su compra mediante la plataforma. Una forma rápida y escalable de llevar a cabo la idea sería mediante 'Mercado Envíos Full'[20], el cual se ocupará en almacenar el stock, preparar el stock luego de una venta y realizar el envío.

Sistema de reservas

Esta funcionalidad traería un gran valor agregado a la aplicación, ya que los usuarios no sólo podrían buscar cabañas o guarderías sino que tendrían la posibilidad de hacer una reserva directamente desde el sistema.

[20] https://www.mercadolibre.com.ar/ayuda/que-es-mercado-envios-full_5162

8. Conclusiones

En relación a lo expuesto a lo largo del informe debemos remarcar que el proyecto, desde su concepción, se pensó como un desafío para poner a prueba las habilidades adquiridas a lo largo de la carrera así como también para incorporar nuevos conocimientos a nivel de herramientas y metodologías de trabajo. Muchas de las decisiones tomadas a lo largo del proyecto fueron resultado de la experiencia y situación personal de cada uno de los integrantes, dado que comenzamos a experimentar responsabilidades laborales lo que permitió poder capacitarnos al mismo tiempo que nos desenvolvíamos profesionalmente. Esto provocó consecuencias tanto positivas como negativas. Por un lado, el incentivo no sólo era el desarrollo del proyecto final de carrera sino que también se avanzaba a la par en términos profesionales. En cambio, por otro lado, el equipo se vio afectado a cambios de intereses o nuevos objetivos que luego se traducirían en demoras e incumplimiento de los tiempos definidos en las etapas tempranas de planificación.

Un punto a analizar es que no nos regimos netamente por los lineamientos definidos por la metodología Scrum. Por ejemplo, en muchas oportunidades los tiempos estimados para cada una de las iteraciones resultaron ser poco reales lo que conllevo a que el tiempo real sea mayor. La acción tomada por el equipo fue continuar desarrollando sobre la iteración independientemente del tiempo consumido. Esto se debió haber reestimado definiendo una iteración extra que abarque aquellas historias que estaban consumiendo mayor tiempo del planificado.

Otro de los aprendizajes que nos dejó este proyecto es la valoración por la documentación y confección de pruebas a la par del desarrollo. Con el paso del tiempo, las modificaciones o resoluciones de errores sobre funcionalidades previamente realizadas requerían que volvamos a ponernos en contexto sobre la lógica y parámetros definidos que, sin lugar a duda, teniendo una documentación sobre los métodos definidos en el *back-end* o sobre las responsabilidades de componentes en el *front-end* hubiera facilitado mucho el esfuerzo de revisar código. Lo mismo ocurre con las pruebas, hubiera sido de gran ayuda poder ejecutar un conjunto de pruebas para evitar regresión lógica y, al mismo tiempo, lograr incrementar nuestro nivel de confianza a la hora de modificar funcionalidades o incluir correcciones.

El plan de negocios, realizado mediante la metodología de *Canvas*, fue algo que nos aportó mucho valor agregado debido a que nos hizo pensar en dos de las partes más importantes de un proyecto como el nuestro: ingresos y clientes. Un grave error que tal vez hubiésemos cometido, si no lo hubiésemos hecho de esta forma, es confundir clientes con usuarios, por ejemplo, pensar que una cabaña es un usuario. Otra cuestión en la que nos ayudó esta metodología es en la definición de los gastos del proyecto. Nos permitió pensar la forma de sustentar la aplicación durante los primeros meses de vida, en donde se requiere compromiso personal para dar a conocerla, es decir, costos y tiempo en recorrer los principales puntos pesqueros que consideramos potenciales para atraer clientes y usuarios.

Estamos conformes con el desempeño debido a que logramos organizarnos y encontrar una metodología de trabajo de la que todos hemos aprendido y que,

definitivamente, nos servirá para nuestra vida profesional. En cuanto a las tecnologías utilizadas, como bien se nombró anteriormente, es un valor agregado que este proyecto nos dejó ya que se presentaron situaciones particulares las cuales nos permitieron investigar a fondo sobre configuraciones y características de las distintas herramientas y *frameworks* que de otra forma en el ámbito laboral no hubiéramos logrado visualizar.

En conclusión, consideramos que hemos sido capaces de adaptarnos y encontrar las soluciones correspondientes a los diferentes problemas que se presentaron a lo largo del proyecto, logrando además obtener conocimientos y experiencia para nuestra actividad profesional.

9. Bibliografía

En este capítulo se encuentran listados algunos links, como así también libros en los cuales nos hemos basado, consultado y tomado información para el desarrollo de este proyecto.

Libros

BRACALENTI, C. “Gestión de riesgos en los proyectos de software”. Monografía. Universidad Nacional de la Plata, 2009.

CANÓS, J., LETELIER, P., PENADÉS, M. “Metodologías ágiles en el desarrollo de software”. Proceedings VIII Jornadas de Ingeniería del software y Bases de Datos, JISBD 2003. Alicante, noviembre de 2003.

PALACIO, J. “Flexibilidad con Scrum”. Edición 2008.

PRESSMAN, R. S. “Software Engineering: A Pratictioner’s Approach”. 7ta. Edición. Mc Graw-Hill. 2010.

Fuentes electrónicas

“Qué es SCRUM”, de: https://proyectosagiles.org/que-es-scrum

“Scrum (desarrollo de software)”, de: https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)

HUAMBACHANO, J. F. “¿Qué es Scrum?”. 25 de septiembre de 2017, de: https://www.scrum.org/resources/blog/que-es-scrum

GALIANA, P. ESCUELA DE NEGOCIOS DE LA INNOVACIÓN Y LOS EMPRENDEDORES. “Cómo funciona la Metodología Scrum: Qué es y cómo utilizarla”. 20 de abril de 2021, de: https://www.iebschool.com/blog/metodologia-scrum-agile-scrum

DRUMOND, C. “¿Qué es scrum?”, de: https://www.atlassian.com/es/agile/scrum

“Programación extrema”, de: https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema

BELLO, E. ESCUELA DE NEGOCIOS DE LA INNOVACIÓN Y LOS EMPRENDEDORES. “Descubre qué es el Extreme Programming y sus características”. 28 de abril de 2021, de: https://www.iebschool.com/blog/que-es-el-xp-programming-agile-scrum/

“Extreme Programming: A gentle introduction”, de: http://www.extremeprogramming.org

“Extreme Programming (XP)”, de: https://www.agilealliance.org/glossary/xp

“Arquitectura de microservicios”, de: https://es.wikipedia.org/wiki/Arquitectura_de_microservicios

JUAN, C. “Introducción a Microservicios”. 23 de agosto de 2019, de: https://jcabelloc.medium.com/introducci%C3%B3n-a-microservicios-11cf380f31a0

MATUSCHEK, J. P. “Finding Points Within a Distance of a Latitude/Longitude Using Bounding Coordinates”, de: http://janmatuschek.de/LatitudeLongitudeBoundingCoordinates

BOURGON, P. “Go for Industrial Programming”, de: https://peter.bourgon.org/go-for-industrial-programming

AYADA, B. V. “Multipart Requests in Go”. 8 de junio de 2019, de: https://ayada.dev/posts/multipart-requests-in-go

“Integrar el inicio de sesión de Google en su aplicación web”, de: https://developers.google.com/identity/sign-in/web/sign-in

“Inicio de sesión con Facebook”, de: https://developers.facebook.com/docs/facebook-login

NIELSEN, M. “Testing gorilla/mux handlers”. 13 de abril de 2019, de: https://polothy.github.io/post/2019-04-13-testing-gorrilla-mux-handlers

BLUM. S. “Why you should be using errgroup.WithContext() in your Golang server handlers”. 18 de febrero de 2020, de: https://bionic.fullstory.com/why-you-should-be-using-errgroup-withcontext-in-golang-server-handlers

KABRA, K. “Building and Testing a REST API in Go with Gorilla Mux and PostgreSQL”. 19 de marzo de 2020, de: https://semaphoreci.com/community/tutorials/building-and-testing-a-rest-api-in-go-with-gorilla-mux-and-postgresql

SILVERLOCK, M. “Testing Your (HTTP) Handlers in Go”. 23 de mayo de 2016, de: https://rollout.io/blog/testing-http-handlers-go

Cursos

LEÓN, F. “Unit, integration and functional Testing in Golang (Go)”, de: https://www.udemy.com/course/unit-integration-and-functional-testing-in-golang-go

NEAGOIE, A. “The Complete Junior to Senior Web Developer Roadmap (2022)”, de: https://www.udemy.com/course/the-complete-junior-to-senior-web-developer-roadmap

GRIDER, S. “Go: The Complete Developer's Guide (Golang)”, de: https://www.udemy.com/course/go-the-complete-developers-guide

10. Anexo 1 - Historias de usuario refinadas

HISTORIA DE USUARIO Nº1			
Número:	1	**Nombre:**	El usuario desea registrarse o utilizar la aplicación como usuario invitado
Responsable:		**Iteración asignada:**	1
Modificación historia:		**Puntos estimados:**	2
Prioridad:	ALTA	**Puntos reales:**	6
Riesgo en desarrollo:			

Descripción:
El usuario desea registrarse en el sistema creando una cuenta de usuario nueva e iniciar sesión con la misma, iniciar sesión con una cuenta de Google, iniciar sesión con una cuenta de Facebook o desea utilizar la aplicación como usuario invitado.

Especificación:
Desde la sección *Registrarse* deberá permitirse la creación de una nueva cuenta de usuario. Para dar de alta un nuevo usuario deberá especificarse:

- Nombre de usuario (text) - Campo obligatorio.
- DNI (text) - Campo obligatorio.
- Fecha de nacimiento (text) - Campo obligatorio.
- Email (text) - Campo obligatorio.
- Contraseña (text) - Campo obligatorio.

Una vez validados los datos ingresados se habrá generado un nuevo usuario. En caso de que algún dato sea erróneo, se deberá corregir.
Desde la sección *Iniciar Sesión* deberá permitirse el inicio de sesión del usuario ya sea por una cuenta generada anteriormente, por una cuenta de Google o por una cuenta de Facebook.
Para el inicio de sesión con una cuenta generada, el usuario deberá especificar:

- Email (text) - Campo obligatorio.
- Contraseña (text) - Campo obligatorio.

Para el inicio de sesión con Google o Facebook, el usuario deberá ingresar las credenciales correspondientes a su cuenta a través de las *APIs* de dichas plataformas.

Criterios de aceptación:
En el caso de la creación de una nueva cuenta, se deberá informar al usuario que la cuenta que intentó generar se creó de forma correcta, asegurándole que podrá iniciar sesión luego con dicha cuenta en la aplicación.
En el caso del inicio de sesión, el usuario deberá acceder correctamente en la aplicación y a sus funcionalidades con sus credenciales.

Observaciones:
El *Nombre de usuario* requerido en la creación de una nueva cuenta hace referencia a el Nombre y Apellido del usuario.
Para iniciar sesión con Google o Facebook, al dar *click* en el botón correspondiente, se abrirá una nueva ventana donde el usuario podrá ingresar las credenciales de la respectiva cuenta. El usuario deberá contar previamente con una cuenta en dichas plataformas.

HISTORIA DE USUARIO Nº2

Número:	2	**Nombre:**	El usuario registrado desea modificar datos propios en su perfil
Responsable:		**Iteración asignada:**	4
Modificación historia:		**Puntos estimados:**	1
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Para los usuarios activos, se ofrecerá la opción de ver y modificar sus datos.
Además, se agrega la posibilidad de que el usuario pueda poner una foto de perfil.

Especificación:
Desde la sección “Editar perfil” el usuario podrá modificar los datos solicitados en la historia número uno, es decir:

- Nombre
- Contraseña
- DNI
- Fecha de nacimiento
- Email

En el *request* del *front-end* puede venir uno o todos los campos mencionados a editar.
Se debe incorporar un nuevo *endpoint* a nivel *back-end* que permita modificar la foto de perfil.
Existirá una foto de perfil pre-cargada, genérica, la cual se devuelve a todos los usuarios como respuesta a un *login*. En caso de que el usuario quiera subir una personalizada propia, se le debe dar la posibilidad.
Para la foto de perfil se debe crear un nuevo *bucket* en el *storage* de Google donde allí se subirán las fotos de perfil de cada usuario. El mismo estará conformado con una carpeta por usuario, la cual el nombre es el id de usuario y dentro estará la foto subida por el mismo. Además, en la raíz del *bucket* estará la imagen default a todos los usuarios.
Dentro de esta historia, también se requiere, la posibilidad de que un usuario pueda eliminar su foto de perfil.

Criterios de aceptación:
La respuesta al *front-end*, sí no hubo errores es un estado 200, en caso de haya errores (ej: falló la conexión a la base de datos) es un 500.

Observaciones:
Si se solicita cambiar la contraseña, se debe encriptar la nueva previo a la actualización en ***postgres***.
En caso de que el usuario suba una nueva foto de perfil, y luego decida eliminarla, debe volver a colocarse la foto de perfil default a todos los usuarios.

HISTORIA DE USUARIO Nº3

Número:	3	**Nombre:**	Un usuario no quiere seguir siendo parte de la aplicación
Responsable:		**Iteración asignada:**	5
Modificación historia:		**Puntos estimados:**	2
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
El usuario activo puede decidir darse de baja de la aplicación por lo que se le brindará la opción de hacerlo. Eliminando de esta forma, todos los datos referidos al usuario.

Especificación:

La baja de un usuario involucra eliminar:

1. La foto de perfil en caso de tener una que no sea la default
2. Las publicaciones en el perfil del usuario
3. Negocios creados por el usuario
4. Fotos de los negocios subidas al *bucket*
5. Reseñas hechas por el usuario
6. Todas las publicaciones del *feed* del usuario
7. Fotos de las publicaciones subidas al *bucket*
8. Todos los comentarios y *likes* que realizó el usuario

Criterios de aceptación:

Observaciones:
El borrado de registros en las bases de datos (Mongo y ***postgres***) es algo rápido, pero el borrado en el *bucket* puede demorar algunos segundos y varía dependiendo el volumen de fotos que tenga un usuario almacenado. Por lo tanto, si el usuario decide darse de baja, debe devolverse un código 200 inmediatamente y el proceso se terminará en segundo plano. Esto es para no tener que

HISTORIA DE USUARIO Nº4

Número:	4	**Nombre:**	El usuario desea crear un negocio gestionado por él
Responsable:		**Iteración asignada:**	1
Modificación historia:		**Puntos estimados:**	2
Prioridad:	ALTA	**Puntos reales:**	7.5
Riesgo en desarrollo:			

Descripción:
Un usuario registrado; ya sea manualmente, vía Facebook o Google debe poder dar de alta un negocio en la plataforma. Inicialmente se pensó en 4 tipos de negocios permitidos en la aplicación: cabaña, bajada de lancha, negocio, guardería. Debe quedar en una forma sencilla la futura incorporación de nuevos tipos de negocio en caso de ser necesario (ya sea por sugerencia de un usuario o cliente).

Especificación:
Desde la sección *Crear Negocio* se le otorgará la posibilidad al usuario de publicar en la comunidad de Santa Pesca un negocio administrado por él. Se le debe solicitar al usuario la siguiente información con respecto a su negocio:

1. **Datos básicos:**
    a. Tipo (cabaña, bajada de lancha, negocio, guardería) - Campo Obligatorio.
    b. Nombre - Campo Obligatorio.
    c. Descripción - Campo Opcional.
    d. Ubicación, aquí se le debe dar la posibilidad al usuario de usar su ubicación actual, escribir una dirección validada por el *api* de Google o seleccionar un punto en el mapa. - Campo Obligatorio.
2. **Datos Opcionales:**
    a. Características. En base al tipo de negocio seleccionado por el usuario, se le dará la posibilidad de elegir características de su negocio. Ejemplo: tipo de negocio seleccionado: Cabaña. Características: *Wi-Fi*, estacionamiento, desayuno, etc. - Campo Opcional.
    b. Datos de contacto: Whatsapp, teléfono, email, Facebook o Instagram.
3. **Fotos:**
    a. Se le dará la posibilidad al usuario creador de su negocio de subir hasta **5** fotos que describan su publicación.

Criterios de aceptación:
Al momento de seleccionar una ubicación, se debe validar que esté dentro de una región permitida. Ejemplo: el usuario selecciona una localidad de Chile o Uruguay y SantaPesca todavía no tiene cobertura en ese País. Lo mismo en una ciudad/provincia de Argentina que no cuente con cobertura.
Al momento de dar el *click* final en “crear negocio” y validado los datos obligatorios se le debe mostrar al usuario un cartel o *popup* que indique que el negocio fue creado con éxito o no.

Observaciones:
Cuando el usuario abra el mapa para elegir una ubicación; y seleccione el “icono” de ubicación actual, se le abrirá un *popup* (diferente si está en la versión móvil o web, pero el resultado es el

mismo) donde le pedirá permiso para acceder a su ubicación actual.

HISTORIA DE USUARIO Nº5			
Número:	5	**Nombre:**	El usuario desea realizar la modificación de datos pertenecientes a la cuenta de un negocio gestionado por él
Responsable:		**Iteración asignada:**	4
Modificación historia:		**Puntos estimados:**	1
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			
Descripción:			

Puede que a lo largo del tiempo un negocio cambie de locación, servicios que brinda, contacto, nombre, descripción o fotos por lo que el usuario encargado podrá realizar estos cambios desde el perfil del negocio en particular.

Especificación:

En la sección “Mis negocios” se visualizarán todos los negocios pertenecientes a un usuario. En la misma se podrá seleccionar y editar cualquiera de los negocios que se visualizan en pantalla.

Los campos editables de un negocio son:

- Nombre
- Descripción
- Servicios seleccionados
- Locación (dirección, latitud y longitud)
- Ami n (WhatsApp, PhoneNumber, Facebook, Instagram y Email)
- Fotos del negocio

Criterios de aceptación:

Cuando se haya editado un negocio, debe asegurarse que no quede cacheado el negocio en el resto de los usuarios. Esto es, para que los usuarios siempre visualicen la información correcta y no una versión desactualizada sobre los mismos.

Observaciones:

Cuando un usuario elimina una foto de un negocio debe asegurarse la eliminación de la misma en el *bucket* de Google además del registro en la base de datos.

HISTORIA DE USUARIO Nº6

Número:	6	**Nombre:**	El usuario desea eliminar la cuenta del negocio administrado por él
Responsable:		**Iteración asignada:**	5
Modificación historia:		**Puntos estimados:**	2
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:

Un usuario que tiene publicado algún servicio dentro de SantaPesca, tendrá la posibilidad de eliminar el mismo.

Especificación:

Se podrá acceder a un negocio desde la sección “Mis negocios” o desde el propio *feed* de negocios hallándose mediante los filtros. Si el usuario es el dueño del negocio visualizado se le habilitará la opción de editarlo y eliminarlo. Cuando el usuario dé con la opción de “Eliminar” se procederá a realizar el borrado del mismo en ***postgres*** junto con las fotos del *bucket* si las hubiese.

Criterios de aceptación:

Observaciones:

HISTORIA DE USUARIO Nº7

Número:	7	**Nombre:**	El usuario desea emitir una reseña a un servicio determinado
Responsable:		**Iteración asignada:**	3
Modificación historia:		**Puntos estimados:**	2
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Los usuarios activos podrán realizar sus descargos y calificaciones por cada servicio. De esta forma se ofrecerá información cuantificable para aquellos usuarios pasivos o, bien, usuarios activos que deseen ver la calidad de servicio ofrecida en ese lugar.

Especificación:
En la sección de *reseñas* de un servicio el usuario podrá calificar y emitir una reseña del mismo. Los datos que se podrán ingresar son:

- Calificación de 1 a 5 estrellas (int).
- Reseña (text).

Una vez que el usuario emite una reseña no puede ni editarla ni eliminarla luego de haberla hecho.

Criterios de aceptación:
Cuando un usuario ingrese una reseña de algún servicio, esta podrá ser visible para el resto de los usuarios.

Observaciones: Un usuario no puede realizar más de una reseña sobre un mismo negocio.

HISTORIA DE USUARIO Nº8

Número:	8	**Nombre:**	El usuario quiere obtener información sobre un pez determinado
Responsable:		**Iteración asignada:**	1
Modificación historia:		**Puntos estimados:**	1
Prioridad:	ALTA	**Puntos reales:**	1
Riesgo en desarrollo:			

Descripción:
Al ser una aplicación referida a la actividad pesquera es de suma importancia el ofrecimiento de información referida al pez. El usuario logueado deberá poder visualizar datos acerca de una determinada especie de pez.

Especificación:
En la sección *Especies* se le mostrará al usuario una lista con las diferentes especies de peces existentes en el sistema y podrá seleccionar una para obtener su respectiva información.
Como dato de entrada solo será necesario que el usuario seleccione la especie de la cual desea ver los datos. Dichos datos que se mostrarán son:

- Nombre (text).
- Nombre científico (text).
- Descripción (text).
- Peso promedio (text).
- Longitud promedio (text).
- Una foto de la especie (text).
- Si puede ser pescado o está en veda (boolean).
- Modalidades de pesca (varchar(300)).
- Época en el año donde puede ser pescado (text).

Criterios de aceptación:
Al momento de seleccionar una especie determinada, el usuario podrá ver la información referida a dicha especie.

Observaciones:
Los datos de las especies deben estar previamente cargados en el sistema y se deben mantener actualizados.
Es de suma importancia que se especifique claramente si una especie está en veda.

HISTORIA DE USUARIO Nº9

Número:	9	**Nombre:**	El usuario quiere realizar una publicación en la red social
Responsable:		**Iteración asignada:**	2
Modificación historia:		**Puntos estimados:**	3
Prioridad:	ALTA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Uno de los aspectos más importantes del sistema es la red social o *feed*. Aquí el usuario podrá realizar una publicación en dicho *feed* donde se le permitirá mostrar su captura realizada, la fecha y la ubicación de la misma.
Los usuarios que vean estos *posteos* en dicha ubicación o cercana a la misma, tendrán información acerca de qué especie está saliendo en qué lugar.
Por defecto, si el usuario no selecciona ninguna ubicación, se le mostrarán *posteos* de diferentes localidades ordenados por fecha de subida. Esto nos permite brindarle al usuario información en todo momento y le dará la posibilidad de encontrar nuevos lugares para pescar que quizás no conozca o saber que en algún lugar está saliendo tal especie cuya información desconocía.

Especificación:
Cuando el usuario inicia sesión, en la pantalla principal se le mostrará la red social con las últimas publicaciones realizadas por otros usuarios de cualquier ubicación en orden de creación. Si el usuario lo desea, podrá elegir una ubicación específica y un radio de búsqueda para ver publicaciones en dicho lugar. El usuario puede volver a la pantalla de red social en todo momento.
En la misma pantalla se le dará la opción de realizar una nueva publicación, donde podrá realizar una foto e ingresar información al posteo. Los datos que el usuario puede añadir son los siguientes:

- Foto (archivo).
- Permitir o no comentarios (bool).
- Guardar o no la foto de la publicación en el perfil (bool).
- Comentario o descripción de la publicación (text).
- Fecha (date).
- Especie capturada en la publicación (string).
- Ubicación donde se realizó la publicación (Ubicación).

Una vez que el usuario ingresa los datos, podrá crear la publicación.

Criterios de aceptación:
Cuando el usuario inicie sesión o, estando *logueado*, se dirija a la pantalla inicial, podrá visualizar publicaciones realizadas por otros usuarios. Si elige una ubicación particular y un radio, podrá ver *posteos* pertenecientes a dicha ubicación buscada.
Al momento de crear una nueva publicación, el usuario podrá tomar una foto con su teléfono e ingresar todos los datos que desee. Cuando finalmente realice la publicación, podrá visualizarla en la red social.

Observaciones:
La foto es obligatoria y no puede ser seleccionada del carrete/galería del teléfono, deberá ser tomada en el momento de crear la publicación.
La fecha de creación no es ingresada por el usuario sino es la fecha actual al momento de crear la

publicación.
La ubicación es tomada a partir de la ubicación actual del usuario, tampoco es ingresada por él mismo.
Por defecto se permiten comentarios en la publicación y la foto de la misma no es guardada en el perfil del usuario.

HISTORIA DE USUARIO Nº10			
Número:	10	**Nombre:**	El usuario desea modificar datos o configuraciones de su publicación en la red social
Responsable:		**Iteración asignada:**	4
Modificación historia:		**Puntos estimados:**	1
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Un usuario activo puede decidir modificar algunos datos sobre una publicación realizada como pueden ser los permisos, la ubicación o bien cambiar el título o comentarios realizados por el autor de la misma.

Especificación:
En cualquiera de las publicaciones que el usuario haya realizado se le dará la opción de editarla. Para ello, deberá expandir el menú de opciones de la publicación *clickeando* en los tres puntos verticales del post (⋮) y luego ***clickear*** en Editar.

Criterios de aceptación:
Cuando el usuario edite una publicación, esta deberá actualizarse a todos los usuarios con los nuevos cambios aplicados.

Observaciones:
Un usuario sólo podrá editar publicaciones que él mismo haya creado. Nunca podrá modificar una publicación de un tercero.

HISTORIA DE USUARIO Nº11			
Número:	11	**Nombre:**	El usuario quiere eliminar una publicación propia en la red social
Responsable:		**Iteración asignada:**	3
Modificación historia:		**Puntos estimados:**	1
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Un usuario activo podrá realizar la eliminación de una publicación realizada. Esta acción ocasionará el borrado de todos los datos referidos a la misma.

Especificación:
En cualquiera de las publicaciones que el usuario haya realizado se le dará la opción de eliminarla. Para ello, deberá expandir el menú de opciones de la publicación *clickeando* en los tres puntos verticales del post (⋮) y luego *clickear* en Eliminar.

Criterios de aceptación:
Cuando el usuario elimine una publicación, esta deberá dejar de mostrarse a todos los usuarios ni deberá permitirse interactuar con la misma.

Observaciones:
Un usuario sólo podrá borrar publicaciones que él mismo haya creado. Nunca podrá eliminar una publicación de un tercero.

HISTORIA DE USUARIO Nº12			
Número:	12	**Nombre:**	Un usuario desea interactuar con otros mediante el envío de mensajes privados
Responsable:		**Iteración asignada:**	3
Modificación historia:		**Puntos estimados:**	4
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:

Especificación:

Criterios de aceptación:

Observaciones:

HISTORIA DE USUARIO Nº13			
Número:	13	**Nombre:**	Un usuario quiere agregar comentarios en una publicación
Responsable:		**Iteración asignada:**	3
Modificación historia:		**Puntos estimados:**	1
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Un usuario activo podrá realizar un comentario en sus publicaciones o en las de otros usuarios si estos lo permiten.

Especificación:
En la sección del *feed*, si la publicación lo permite, el usuario deberá hacer *click* en el botón de comentarios (💬) lo que expandirá los comentarios actuales y le dará la opción de redactar un nuevo comentario para dicha publicación. Los datos que el usuario podrá ingresar en un comentario son:

- Texto del comentario (text).

Una vez que el usuario ingrese su comentario, podrá publicarlo en el post.

Criterios de aceptación:
Cuando el usuario ingrese un nuevo comentario en una publicación, este deberá ser visualizado por todos los usuarios.

Observaciones:
Los comentarios serán sólo y exclusivamente de índole de texto. Quedando así no permitida la incorporación de imágenes o emoticones.

HISTORIA DE USUARIO Nº14

Número:	14	**Nombre:**	El usuario desea eliminar comentarios propios en una publicación
Responsable:		**Iteración asignada:**	3
Modificación historia:		**Puntos estimados:**	1
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Un usuario activo que ha realizado un comentario en una publicación tendrá la opción de eliminar el mensaje. Quedando de esta forma, no visible para él ni para cualquiera que interactúe con ella.

Especificación:
En la sección de comentarios de una publicación, el usuario podrá eliminar aquellos comentarios que haya escrito en dicho post. Para ello, deberá expandir el menú de opciones del comentario *clickeando* en los tres puntos verticales del mismo (⋮) y luego ***clickear*** en Eliminar.

Criterios de aceptación:
Cuando el usuario elimine un comentario, este deberá dejar de mostrarse a los usuarios.

Observaciones:
Un usuario sólo podrá eliminar comentarios que él mismo haya realizado en una publicación, no se le permitirá borrar comentarios de terceros.

HISTORIA DE USUARIO Nº15

Número:	15	**Nombre:**	El usuario desea realizar la búsqueda de usuarios, servicios o locaciones
Responsable:		**Iteración asignada:**	2
Modificación historia:		**Puntos estimados:**	2
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Un usuario podrá hacer una búsqueda sobre el perfil de otros usuarios o de servicios publicados en la plataforma, pudiendo definir una ubicación y un radio para realizar la búsqueda.

Especificación:
Cuando el usuario ingrese a la sección *Buscar* podrá encontrar resultados acerca de:

- Perfiles de otros usuarios de la aplicación.
- Servicios (negocios, bajadas de lancha, etc.).

Para realizar esto el usuario podrá ingresar como criterio de búsqueda:

- Nombre.
- Tipo de búsqueda (usuario, negocio, servicio, etc.).
- Características (si se está buscando un servicio).
- Calificación.
- Radio de búsqueda.
- Ordenar resultados por cercanía (opcional).

Además, podrá definir una ubicación específica para realizar la búsqueda de servicios y especificar un radio (en kilómetros) para poder hacer un filtrado más focalizado en la ubicación seleccionada o más general.
Los resultados de la búsqueda arrojarán los usuarios o servicios encontrados, la distancia desde la ubicación seleccionada hasta dicho lugar, la posibilidad de guardar algún resultado en *Elementos guardados* y un botón *Ver más* que permitirá acceder a los perfiles de usuarios o servicios.

Criterios de aceptación:
Cuando el usuario realice una búsqueda con los parámetros deseados, podrá visualizar correctamente los resultados de dicha búsqueda con la información correspondiente y podrá acceder a dichos resultados.

Observaciones:
El nombre es una o varias palabras que son ingresadas por el usuario. En cambio, el tipo, las características y el radio son valores predefinidos que el usuario puede seleccionar.

HISTORIA DE USUARIO Nº16

Número:	16	**Nombre:**	Creación pantalla de inicio
Responsable:		**Iteración asignada:**	1
Modificación historia:		**Puntos estimados:**	1
Prioridad:	ALTA	**Puntos reales:**	2
Riesgo en desarrollo:			

Descripción:
La pantalla de inicio de la aplicación consistirá en la visualización de las publicaciones de los otros usuarios como así también los distintos menús por los que el usuario podrá navegar y así hacer uso de las distintas funcionalidades de la aplicación.

Especificación:

Criterios de aceptación:

Observaciones:

HISTORIA DE USUARIO Nº17

Número:	17	**Nombre:**	El usuario desea guardar una publicación, ubicación o una cabaña
Responsable:		**Iteración asignada:**	5
Modificación historia:		**Puntos estimados:**	2
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Se ofrecerá la funcionalidad de que el usuario pueda guardar y gestionar aquellas publicaciones, ubicaciones o negocios/cabañas que le interesen.

Especificación:
En cada publicación, negocio o ubicación el usuario tendrá la posibilidad de guardar el elemento, haciendo *click* en el icono *bookmark*. Luego, podrá visualizar todos los elementos guardados en la sección *Elementos Guardados*.

Criterios de aceptación:
Cuando el usuario haga *click* en el icono para guardar un elemento, este deberá pintarse de negro indicando que efectivamente se guardó.
Cuando el usuario ingrese en la sección *Elementos Guardados* deberá poder visualizar los elementos que guardó.

Observaciones:

HISTORIA DE USUARIO Nº18

Número:	18	**Nombre:**	El usuario desea cambiar su ubicación actual
Responsable:		**Iteración asignada:**	2
Modificación historia:		**Puntos estimados:**	1
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Al iniciar sesión en la aplicación se establecerá, previa consulta, la ubicación del usuario. Esta se utiliza para establecer el lugar de las publicaciones o como criterio de búsqueda para aquellas en donde se desee consultar por negocios o cabañas cercanas o bien a una determinada distancia.

Especificación:

Criterios de aceptación:

Observaciones:

HISTORIA DE USUARIO Nº19

Número:	19	**Nombre:**	Teniendo una ubicación el usuario quiere saber el camino que debe seguir para llegar a ésta
Responsable:		**Iteración asignada:**	2
Modificación historia:		**Puntos estimados:**	3
Prioridad:	ALTA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Tanto las publicaciones como los negocios tendrán, dentro de sus descripciones, la ubicación correspondiente. Es por esto que el usuario podrá inspeccionarla y, a su vez, guardarla o querer visualizar el camino para llegar al destino. Si elige la segunda opción se deberá proporcionar el camino mediante los servicios de Google Maps.

Especificación:
En el caso de que el usuario esté usando el sistema desde un dispositivo móvil se abrirá Google Maps con la ubicación seleccionada.
En cambio, en el caso de que el usuario se encuentre utilizando el sistema desde una computadora se le abrirá una nueva pestaña con el sitio web de Google Maps con la ubicación seleccionada

Criterios de aceptación:
En ambos casos se debe visualizar la ubicación seleccionada.

Observaciones:

HISTORIA DE USUARIO Nº20

Número:	20	**Nombre:**	El usuario desea indicar que una publicación, mensaje o comentario le ha gustado
Responsable:		**Iteración asignada:**	4
Modificación historia:		**Puntos estimados:**	1
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Se ofrecerá la funcionalidad de darle “me gusta” a una publicación, mensaje o comentario emitido por otro usuario. Como así también la posibilidad de removerlo en caso de una equivocación o arrepentimiento.

Especificación:
En una publicación, el usuario podrá darle “me gusta” tanto a la publicación como a aquellos comentarios que tenga. Para esto, deberá *clickear* en el botón de “me gusta” el cual incrementará en 1 el contador de *likes* que tenga la publicación o comentario.
A su vez, *clickeando* nuevamente en el botón en una publicación o comentario donde haya tocado anteriormente, se deberá quitar el me gusta reduciendo el contador en 1.

Criterios de aceptación:
Cuando el usuario haga *click* en el botón de “me gusta” se deberá incrementar correctamente en 1 el contador de *likes* y el botón deberá pintarse indicándole al usuario que se realizó el *like*.
Cuando el usuario haga *click* en el botón de “me gusta” en donde ya lo realizó anteriormente, se deberá decrementar en 1 el contador de *likes* y el botón deberá despintarse indicándole al usuario que se quitó el *like*.

Observaciones:
El usuario puede agregar o quitar *likes* en cualquier comentario o publicación, ya sean propias o de terceros.
Un usuario no puede darle más de un me gusta a una publicación o comentario.

HISTORIA DE USUARIO Nº21

Número:	21	**Nombre:**	El usuario requiere de ayuda para poder comprender cómo debe utilizar el sistema
Responsable:		**Iteración asignada:**	5
Modificación historia:		**Puntos estimados:**	1
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Se debe proporcionar un tutorial en donde se puede visualizar, en primera instancia, cuáles son las funcionalidades que el sistema ofrece. En segundo lugar, cómo es que estas funciones operan. Por ejemplo, se puede presentar el caso de que el usuario quiera saber las distintas opciones que se ofrecen al querer realizar una publicación o guardar una captura en su perfil.

Especificación:
Se deben confeccionar gráficas en donde se muestra primero la pantalla en la cual la funcionalidad es accesible. Y, en segundo lugar, se debe hacer referencia en qué parte de la pantalla esta funcionalidad puede ser accionada.

Criterios de aceptación:
La ayuda proporcionada debe ser simple de entender ya que se espera que el usuario comience a utilizar la aplicación lo antes posible. También, debe ser visualmente atractiva para que el usuario decida continuar con la explicación de las distintas funcionalidades ofrecidas.
Deben presentarse gráficas y poco texto durante la explicación de cada una de las funcionalidades.

Observaciones:
Esta historia se eliminó del alcance del proyecto por falta de tiempo del equipo de desarrollo.

HISTORIA DE USUARIO Nº22

Número:	22	**Nombre:**	El usuario quiere realizar una captura
Responsable:		**Iteración asignada:**	1
Modificación historia:		**Puntos estimados:**	3
Prioridad:	ALTA	**Puntos reales:**	1
Riesgo en desarrollo:			

Descripción:
Si el usuario así lo desea, podrá subir a su perfil una foto del pescado capturado.

Especificación:
En la sección *Perfil* el usuario tendrá la opción de subir una foto, esto es posible de dos formas distintas:

1. **Utilizando la aplicación en formato web:**
    a. El usuario podrá subir una foto guardada en el dispositivo.
2. **Utilizando la aplicación en formato mobile:**
    a. El usuario podrá sacar una foto desde la cámara en el momento.
    b. El usuario podrá subir una foto guardada en el dispositivo.

Una vez que el usuario cuenta con su foto a subir, se le mostrará una ventana con la foto seleccionada y una lista desplegable donde podrá seleccionar la especie que capturó.

Criterios de aceptación:
El usuario deberá poder subir una foto ya sea desde la cámara o seleccionar una foto guardada en el dispositivo.
Una vez que el usuario suba la foto la misma se deberá visualizar en el perfil del mismo.

Observaciones:
El usuario deberá ser informado debidamente de aquellas especies que estén en veda al momento de seleccionar una para su captura.

HISTORIA DE USUARIO Nº23

Número:	23	**Nombre:**	El usuario desea remover de su perfil una "captura" realizada
Responsable:		**Iteración asignada:**	4
Modificación historia:		**Puntos estimados:**	2
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Cada "captura" realizada se guarda en el perfil del usuario. Este podrá eliminar aquellas que desee simplemente editando su perfil.

Especificación:
En el *Perfil* del usuario estarán todas las capturas que haya realizado. Para eliminar una o varias, deberá *clickear* en el botón *Editar capturas* que mostrará la opción de eliminar en cada captura que haya. Al hacer *click* en dicho botón, esta se eliminará.

Criterios de aceptación:
El usuario debe poder visualizar todas sus capturas en el perfil.
Cuando el usuario haga *click* en *Editar capturas* deberán mostrarse los botones de eliminación en cada captura.
Cuando el usuario haga *click* en el botón eliminar la captura deberá eliminarse y se quitará del perfil.

Observaciones:
El usuario solamente puede eliminar capturas de su propio perfil, nunca podrá eliminar capturas del perfil de otro usuario.

HISTORIA DE USUARIO Nº24			
Número:	24	**Nombre:**	El usuario podrá visualizar las notificaciones que recibe
Responsable:		**Iteración asignada:**	4
Modificación historia:		**Puntos estimados:**	2
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:

Especificación:

Criterios de aceptación:

Observaciones:

HISTORIA DE USUARIO Nº25			
Número:	25	**Nombre:**	El usuario desea realizar una donación monetaria
Responsable:		**Iteración asignada:**	5
Modificación historia:		**Puntos estimados:**	2
Prioridad:	BAJA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Se ofrecerá la posibilidad de emitir una donación si el usuario quiere ayudar o incentivar el desarrollo de nuevas funcionalidades del sistema.

Especificación:

Criterios de aceptación:

Observaciones:

HISTORIA DE USUARIO **Nº26**

Número:	26	**Nombre:**	El usuario desea compartir con otra persona una publicación, ubicación, negocio o producto
Responsable:		**Iteración asignada:**	4
Modificación historia:		**Puntos estimados:**	1
Prioridad:	MEDIA	**Puntos reales:**	
Riesgo en desarrollo:			

Descripción:
Puede que el usuario encuentre de interés una publicación, ubicación, negocio o producto y desee compartirla con amigos que no necesariamente estén registrados o utilicen la aplicación. Es por esto por lo que se ofrecerá la posibilidad de generar un link de lo que se desea compartir y este pueda ser transmitido por las distintas redes sociales como pueden ser Whatsapp, Instagram, Twitter o Facebook.

Especificación:
Se generará una url que se conformará con la entidad a compartir y el token que se le otorga a los usuarios no registrados. De esta forma cualquier usuario no registrado en la aplicación podrá visualizar el elemento compartido y, a su vez, podrá navegar por la aplicación con los permisos correspondientes a un usuario invitado.

Criterios de aceptación:
Un usuario no registrado no sólo podrá visualizar la entidad compartida, sino que también podrá navegar por la aplicación bajo los permisos de un *usuario invitado*

Observaciones: