



# **Universidad Nacional de San Juan**

**Facultad de Ciencias Exactas Físicas y Naturales**

**Departamento de Informática**

**Licenciatura en Ciencias de la Computación**

**Trabajo Final**

## **Sistema multiplataforma para la optimización de reservas deportivas basado en Microservicios, BaaS, Cloud Servers y Contenedores**

**Autor: Martín Pellicer Forcada**

**Director: Mg. Lic. Juan Antonio Aranda Romera**

**San Juan**

**2024**



*Dedicado :*



## **AGRADECIMIENTOS**

A xxx, asesora de Tesis, por xx.

A xxxx

A xxxx

A la Directora y al Vicedirector del Departamento de Informática de la Facultad de Ciencias Exactas, Físicas y Naturales de la Universidad Nacional de San Juan, Prof. Evangelina Sanz, Lic. Manuel Ortega.

# Índice

<b>1.1</b>	<b>Introducción</b>	<b>9</b>
<b>1.2</b>	<b>Planteamiento del problema y justificación</b>	<b>10</b>
<b>1.3</b>	<b>Solución Propuesta</b>	<b>13</b>
<b>1.4</b>	<b>Objetivos</b>	<b>15</b>
<b>1.4.1</b>	<b>Objetivo General</b>	<b>15</b>
<b>1.4.2</b>	<b>Objetivos Específicos</b>	<b>15</b>
<b>1.5</b>	<b>Estructura del documento</b>	<b>16</b>
<b>2.1</b>	<b>Concepto de Turno/Reserva</b>	<b>19</b>
<b>2.2</b>	<b>Sistema de Gestión de Turnos y Reservas</b>	<b>20</b>
<b>2.2.1</b>	<b>Objetivos</b>	<b>21</b>
<b>2.2.2</b>	<b>Funciones</b>	<b>21</b>
<b>2.2.3</b>	<b>Beneficios</b>	<b>22</b>
<b>2.3</b>	<b>Multiplataforma</b>	<b>23</b>
<b>2.3.1</b>	<b>Sistemas Multiplataforma</b>	<b>24</b>
<b>2.3.2</b>	<b>Frameworks Multiplataforma</b>	<b>25</b>
<b>2.3.3</b>	<b>Flutter</b>	<b>26</b>
<b>2.4</b>	<b>Arquitectura de Microservicios</b>	<b>27</b>
<b>2.5</b>	<b>Bases de Datos y Sistemas de Gestión</b>	<b>31</b>
<b>2.5.1</b>	<b>Modelos de Bases de Datos</b>	<b>31</b>
<b>2.6</b>	<b>BaaS</b>	<b>32</b>
<b>2.6.1</b>	<b>Firebase</b>	<b>34</b>
<b>2.7</b>	<b>Cloud Servers</b>	<b>35</b>
<b>2.7.1</b>	<b>Ventajas frente a Servidores Tradicionales</b>	<b>36</b>
<b>2.7.2</b>	<b>DonWeb</b>	<b>38</b>
<b>2.8</b>	<b>Contenedores</b>	<b>38</b>
<b>2.8.1</b>	<b>Docker</b>	<b>40</b>
<b>3.1</b>	<b>Orientación General: Solución para el rubro deportivo</b>	<b>43</b>
<b>3.2</b>	<b>Caso de Estudio: Análisis de requerimientos para una organización deportiva particular</b>	<b>44</b>

3.2.1 Módulo de Administración de Reservas .....	51
3.2.2 Módulo Reservas .....	63
3.2.3 Módulo de Autenticación .....	71
3.3 Análisis y elección de tecnologías .....	76
3.3.1 Frontend: Desarrollo de aplicaciones móviles nativas y multiplataforma .....	79
3.3.2 Backend: Arquitecturas y servicios .....	85
3.3.3 Bases de Datos: Tipos y modelado .....	93
3.3.4 Servidores en la nube: Infraestructura y contenedores .....	101
4.1 Estructura de las aplicaciones .....	106
4.2 Desarrollo a través de dependencias .....	109
4.3 Implementación del BaaS .....	115
4.4 Contratación de un Cloud Server .....	119
4.5 Despliegue de contenedores en la nube .....	125
4.6 Interfaces del software en funcionamiento .....	131
5.1 Propuestas de trabajos futuros .....	138

# **Capítulo 1 - Introducción**



## 1.1 Introducción

La presente tesis se desarrolla para la obtención del título de Licenciado en Ciencias de la Computación, de la Facultad de CEFyN de la Universidad Nacional de San Juan. La investigación se aborda desde una perspectiva de enfoque mixto que implica la aplicación de un conjunto de procesos sistemáticos, empíricos y críticos de investigación. Incluye la recolección y el análisis de datos cuantitativos y cualitativos, así como la integración y discusión conjunta de estos datos. El propósito principal es realizar inferencias producto de toda la información recabada, conocidas como "meta inferencias", con el fin de lograr un mayor entendimiento del fenómeno bajo estudio (Sampieri, 2014).

El enfoque aplicado se toma desde un paradigma pragmático y social, según Capurro R.(2007) *“reconoce que existe una relación entre el estudio de los campos cognitivos y las comunidades discursivas”* (págs. 11-29). Es decir, que estas ciencias toman un giro pragmático, siempre responderá a la pregunta: ¿Para qué sirve?. Así, se

distancia de las ciencias naturales, en donde el “descubrir lo que hay en la naturaleza” es independiente al uso que se le puede dar a ese descubrimiento.

El producto principal de este trabajo es el diseño e implementación de un software que permite la digitalización y optimización de reservas deportivas. Para hacer esto posible, el sistema se apoya en un desarrollo multiplataforma implementado en base a una arquitectura de microservicios y haciendo uso de una arquitectura de servicios en la nube como lo es BaaS Backend como Servicio (del inglés Backend as Service) y el despliegue del sistema en contenedores ubicados en un servidor en la nube. Además de posibilitar el desarrollo con una inversión mínima de capital mínima, esta solución permite crear un sistema operable con eficacia y eficiencia. Asimismo, se justifica la elección de la arquitectura y tecnologías considerando ventajas y desventajas de su uso. Si bien en el caso de estudio, se intenta resolver un problema concreto, esta combinación de tecnologías es aplicable para cualquier situación, siempre teniendo en cuenta que se debe evaluar con exhaustividad cada requerimiento y entorno en donde se desea aplicar, para determinar, si este conjunto de herramientas, tecnologías y lenguajes son los adecuados.

## **1.2 Planteamiento del problema y justificación**

Para comprender el problema, es oportuno analizar su dominio, que en particular se refiere al grado de informatización en organizaciones del sector de prácticas deportivas. Basado en la investigación, la digitalización y automatización de procesos en empresas e instituciones deportivas es escasa, lo que resulta en una gestión ineficiente tanto para las empresas como para los usuarios. Esta falta de informatización dificulta la

práctica de deportes y la reserva deficiente de canchas, generando así gastos y pérdidas de tiempo innecesarios.

Argentina es un país con una alta cantidad de practicantes. Según una encuesta realizada a 1150 argentinos en 2022 por Kantar Global Monitor, en la práctica, casi la mitad de la población se ejercita al menos una vez por semana. 45% de los argentinos hace algún tipo de ejercitación o va al gimnasio, mientras que el 48% practica algún deporte. También existen informes realizados por el Ministerio de Turismo y Deporte, en el año 2009 a 34.372 individuos y en el 2021 a 5.000, en donde se constató en ambos casos, que más de la mitad de la población de menores de edad, práctico algún deporte en los 30 días previos a la realización dicha encuesta, por su parte, los mayores de edad, casi la mitad también afirmó haberlo hecho. Y no solo se trata de los individuos que practican deportes, si hablamos de establecimientos, clubes o complejos deportivos el RENACED (Relevamiento de clubes y entidades deportivas), en el año 2023, realizó por primera vez en la historia un informe que identificó 11.870 clubes y entidades deportivas en todo el país, además, estimó en 4.928.574 la cantidad de participantes activos en estos establecimientos.

Si bien los estudios mencionados no abordan específicamente el problema de la falta de digitalización en los clubes deportivos y sus efectos en la gestión de reservas, es plausible inferir que la escasez de herramientas digitales eficientes en este ámbito probablemente impacte negativamente en la experiencia de miles de deportistas y la gestión de los establecimientos deportivos en Argentina.

Particularmente en San Juan, la práctica de fútbol 5 y pádel, este último arraigado como deporte tradicional en la provincia, ha experimentado una popularidad notable. Según GEOSanJuan (2022), un directorio digital público y gratuito de ubicaciones en la provincia, existen actualmente más de 230 complejos deportivos, entre los cuales el fútbol

y el pádel acaparan la mayor cantidad de complejos respectivamente. A pesar de la considerable cantidad de instalaciones, en proporción al tamaño de San Juan, encontrar turnos disponibles para reservar una cancha un día antes, e incluso el mismo día, se vuelve una tarea desafiante, especialmente en horarios de gran demanda. Según la secretaría de deportes de la provincia de San Juan, un estudio realizado en el 2021 detalla que la provincia tiene el índice de actividad física más alto de Argentina.

San Juan, es un claro ejemplo que representa la necesidad de una herramienta que permita optimizar el proceso de gestión de turnos y reservas. Mayoritariamente, los complejos deportivos, emplean Whatsapp, la aplicación de mensajería instantánea, junto con llamadas telefónicas tradicionales para gestionar todo tipo de turnos, reservas y horarios. Esta situación genera, en ocasiones, superposiciones de horarios, lo que puede resultar en que un grupo de personas que ha reservado una cancha para practicar un deporte llegue al establecimiento sólo para encontrarse con que su turno no fue debidamente registrado. Además, la tendencia a realizar reservas a través de Whatsapp no siempre es bien recibida por los clientes, la gran mayoría prefiere utilizar la aplicación para fines más personales.

También, muchos establecimientos aún dependen de anotaciones manuales en papel, lo que desencadena una serie de problemáticas como la pérdida de información y la falta de claridad en los registros. Algunos clubes recurren a herramientas como Excel para la gestión, esto se podría mejorar significativamente si se adopta un software a medida especialmente diseñado para administrar los turnos en el ámbito deportivo. Otro aspecto problemático es la deficiencia en la organización. La falta de rigurosidad en el registro de los turnos, combinada con la dispersión de esta información en distintos lugares, representa un desafío crucial para asegurar una gestión coherente y efectiva.

---

## 1.3 Solución Propuesta

Para resolver el problema planteado, se desarrolla un software de gestión destinado a facilitar las reservas y diseñado para optimizar el proceso de programación de turnos. Al automatizar la toma y gestión de reservas, este software ahorra tiempo y mejora la eficiencia operativa. Ofrece a los usuarios la visualización de turnos disponibles y la posibilidad de realizar reservas instantáneas y en tiempo real, garantizando notificaciones inmediatas para mantener a los usuarios informados acerca de los eventos programados. Además, permite a las empresas un seguimiento detallado de sus reservas y clientes, facilitando una gestión más efectiva del negocio, brindando herramientas estadísticas para realizar análisis comerciales adecuados y mejorar la rentabilidad. Su integración con diversos procesos y medidas de seguridad contribuye significativamente a reducir la falta de asistencia a turnos registrados, optimizando así la experiencia del cliente.

El sistema se desarrolla de forma multiplataforma, es decir se utiliza una herramienta, en este caso un framework, llamado Flutter, que permite a través de un sólo código fuente, compilar y producir aplicaciones para múltiples entornos, en este contexto es de especial interés la plataforma Web y Móvil. Esto debido a que las personas encargadas en las entidades deportivas cuentan con un celular o una computadora con acceso a internet, lo mismo sucede con los usuarios que desean practicar un deporte.

Como se mencionó antes se hace uso de BaaS con el fin de ahorrar costos y tiempos de desarrollo. Esto permite integrar al sistema un servicio que será de utilidad para gestionar usuarios, contemplar la autenticación y todos los aspectos de seguridad que conlleva un desarrollo de inicio de sesión y registro en un sistema, esto con el fin de no hacer lo que se conoce popularmente como “reinventar la rueda”, es decir ¿Para qué desarrollar un módulo completamente desde cero que el sistema requiere si ya se ha

realizado miles de veces en el pasado? El BaaS utilizado en esta ocasión es Firebase, que no provee únicamente un servicio, en este caso la autenticación (Firebase Auth), si no que provee muchos otros de utilidad, particularmente es de interés también Firebase Crashlytics, una solución que permite obtener métricas para rastrear fallas en dispositivos en tiempo real, Firebase Messaging, lo que permite notificaciones push para enviar ciertos eventos o mensajes a los usuarios que se desee y Firebase Hosting para alojar la página web en un servidor.

La arquitectura, la cual es fundamental para poder desarrollar un sistema sólido, mantenible y escalable, es de microservicios. Toda la lógica relativa a código de interfaces se encuentra en el cliente, en este caso la Web y los Dispositivos Móviles, pero la lógica de negocio, relativa al rubro deportivo, se encuentra en una aplicación completamente distinta e independiente, un microservicio. Éste es quien se comunica con Firebase (BaaS) y con los clientes mencionados.

Para poder contar con una aplicación funcionando el tiempo que se la requiera, es necesario poder alojar la misma. Para el caso de la Web ya se mencionó que se usa Firebase Hosting, para los Dispositivos Móviles, esto no es necesario porque cada usuario la instala en su dispositivo a través de tiendas oficiales como Play Store en dispositivos Android y App Store en dispositivos iOS. Sin embargo, para el caso de la aplicación o microservicio que contiene la lógica de negocio y la base de datos con la cual se comunica, hace falta un lugar donde puedan estar ejecutándose constantemente y de esta manera, recibir y enviar peticiones. Esto se logra haciendo uso de contenedores y servidores en la nube. Los contenedores permiten aislar la aplicación de forma que pueda “empaquetarse” y así poder ser desplegada fácilmente en un servidor. Los Cloud Servers o servidores en la nube son necesarios a fin de no alquilar o comprar un servidor propio el cual implica conocimientos técnicos de hardware y valores de coste muy altos.

## 1.4 Objetivos

### 1.4.1 Objetivo General

- Implementar un sistema multiplataforma basado en Microservicios, BaaS, Cloud Servers y Contenedores para la optimización de reservas deportivas.

### 1.4.2 Objetivos Específicos

- Realizar un análisis detallado del dominio de gestión de reservas y turnos en complejos polideportivos.
- Investigar los procesos actuales de gestión de reservas y turnos en instituciones deportivas, identificando necesidades y falencias específicas.
- Recopilar y analizar datos sobre las prácticas y herramientas utilizadas en la gestión de reservas deportivas.
- Evaluar diferentes opciones de bases de datos (SQL, NoSQL) para determinar la más adecuada en función de los requerimientos del sistema.
- Seleccionar y utilizar frameworks orientados al desarrollo multiplataforma.
- Diseñar la arquitectura del sistema, definir los componentes y módulos utilizando microservicios.
- Analizar las tecnologías adecuadas para BaaS, Cloud Servers y contenedores.
- Desarrollar y desplegar los microservicios necesarios para las funciones clave del sistema.
- Configurar y desplegar el sistema en servidores en la nube utilizando contenedores.

- Realizar pruebas de validación y testeo del sistema en diferentes plataformas.
- Documentar el diseño, desarrollo e implementación del sistema para futuras referencias y mantenimiento.

## 1.5 Estructura del documento

El presente trabajo final se ha estructurado en cinco capítulos. El primero es una introducción sobre la temática, el problema y la forma de abordarlo. Finalmente se menciona el objetivo general y otros específicos.

En el Capítulo 2 – Marco Teórico, se describen conceptos fundamentales para la comprensión y desarrollo del proyecto. Se realiza una recopilación teniendo en cuenta los términos específicos del dominio del problema, propios de la investigación en sí, a saber: turnos, reservas y sistemas basados en turnos y reservas. También se abordan conceptos más técnicos como lo son: multiplataforma, arquitectura de microservicios, bases de datos, sistemas de gestión de bases de datos, BaaS, Cloud servers y contenedores. Estos son fundamentales porque forman la base del sistema de software desarrollado.

El Capítulo 3 – Análisis y Diseño, describe la situación actual del problema especificando en particular el caso de estudio de un club de tenis. Para ello, se lleva a cabo una metodología incremental y un modelo de prototipo. Esto consiste en determinar las falencias o problemas del club y las posibles soluciones que implican simplificación de procesos y automatización. A partir de este análisis se procede a elaborar requerimientos para posteriormente realizar prototipos de interfaces que permiten aclarar y perfeccionar dichos requerimientos. Finalmente, con los prototipos validados y refinados, se procede al análisis, comparación y selección de arquitecturas, servicios y tecnologías adecuadas para la implementación del sistema.



En el Capítulo 4 - Implementación, se aborda el desarrollo del sistema mediante la implementación de un microservicio encargado de la lógica de negocio y una aplicación multiplataforma (iOS, Android y Web), la cual se comunica directamente con el microservicio. También se describe la elección de un BaaS y un proveedor Cloud Server para, luego, desplegar el sistema en la nube utilizando contenedores, los cuales permiten simplificar el proceso de despliegue. Para finalizar con el capítulo, se muestran algunas capturas de pantalla del sistema en funcionamiento.

En el Capítulo 5 - Conclusiones, se enuncian las conclusiones del proyecto, como así también posibles líneas futuras de acción a seguir.

Finalmente, el informe incluye una sección de referencias bibliográficas en donde se citan las fuentes y autores en los que se basa este trabajo.

# **Capítulo 2 - Marco Teórico**

En este capítulo, se presenta el marco teórico que sustenta este proyecto. El propósito del marco teórico es proporcionar un fundamento conceptual que permita entender y analizar el fenómeno en estudio de manera profunda y crítica. Se proporciona información detallada acerca de los aspectos más cruciales que abarcan desde el área de negocios hasta áreas más técnica relativa a la informática que permiten el desarrollo del sistema.

## 2.1 Conceptos específicos

En el contexto de la práctica de deportes, “turno” y “reserva” son términos que se utilizan para describir la acción de asegurar el uso programado de una cancha deportiva en un momento específico. Es decir, es la asignación de un período de tiempo específico durante el cual un usuario o grupo de usuarios tiene acceso garantizado a una cancha para realizar una actividad deportiva, como jugar al fútbol, tenis, pádel, entre otros. Aunque a menudo se utilizan de manera intercambiable, pueden tener leves diferencias en su aplicación:

**Turno:** Según la Real Academia Española es el “Orden según el cual se suceden varias personas en el desempeño de cualquier actividad o función”. En el ámbito deportivo un turno se refiere a un bloque de tiempo específico en el que una persona o grupo tiene el derecho o la programación para usar una cancha deportiva. Los turnos suelen ser de duración fija y se asignan a intervalos regulares, como horas o media hora. Pueden ser asignados de manera regular sin necesidad de una solicitud previa. Por ejemplo, una cancha deportiva puede tener horarios específicos para

diferentes actividades deportivas, y los turnos se asignan automáticamente a los usuarios que llegan a la cancha durante esos horarios.

**Reserva:** La Real Academia Española lo define como “Guarda o custodia que se hace de algo, o prevención de ello para que sirva a su tiempo”. Si lo extrapolamos al ámbito deportivo, una reserva es la solicitud anticipada de un turno en una cancha en particular en una fecha y hora específicas. Cuando se confirma una reserva, se bloquea este turno para el usuario que lo solicitó. Las reservas implican una acción proactiva por parte de los usuarios. Quienes deseen asegurarse de tener un turno en una cancha en particular en un momento específico deben realizar una reserva con anticipación.

Los turnos y sus correspondientes reservas son elementos claves que el sistema debe manejar para poder lograr su cometido y ser lo más óptimo posible para sus usuarios. La reserva de turnos es necesaria para garantizar la disponibilidad de canchas deportivas y evitar conflictos en el uso de las instalaciones. También tiene gran importancia en diversos campos de aplicación como en empresas de entretenimiento, instituciones educativas, organizaciones de salud, empresas de servicios, etc. Por lo que la gestión de turnos y reservas es una práctica clave en diversos campos y sectores, para permitir una programación eficiente.

## **2.2 Sistema de Gestión de Turnos y Reservas**

Un Sistema de Gestión de Turnos y Reservas es una solución tecnológica diseñada para administrar de manera eficiente y ordenada la programación y asignación de turnos. Son herramientas que permiten a empresas, organizaciones e instituciones gestionar y

optimizar la asignación de citas, horarios, espacios y otros recursos de acuerdo con las necesidades de los usuarios.

### 2.2.1 Objetivos

El propósito principal de un Sistema de Gestión de Turnos y Reservas es mejorar la organización, la eficiencia y la satisfacción de los usuarios al facilitar la programación y el acceso a servicios y recursos. Según Appvizer (2022) algunos de los objetivos son:

- **Optimización de la utilización de recursos:** Garantiza que los recursos estén disponibles y se utilicen de manera eficiente, minimizando los tiempos muertos y maximizando la capacidad de servicio.
- **Mejora de la experiencia del cliente:** Permite a los usuarios reservar citas o servicios según sus preferencias, evitando esperas innecesarias y proporcionando una experiencia más personalizada.
- **Gestión de la demanda:** Ayuda a equilibrar la oferta y la demanda, permitiendo una distribución justa de los recursos en función de las solicitudes y prioridades de los usuarios.

### 2.2.2 Funciones

Este tipo de sistemas también suelen incluir una variedad de funciones clave, que suelen variar de sistema en sistema, pero dentro de las más comunes se encuentran:

- **Reservas en línea:** Permite a los usuarios realizar reservas a través de plataformas web o aplicaciones móviles.

- **Calendarios y programación:** Facilita la visualización y la gestión de horarios disponibles y reservados.
- **Recordatorios y notificaciones:** Envía recordatorios automáticos a los usuarios para reducir las cancelaciones y las ausencias.
- **Administración de recursos:** Ayuda a programar y asignar recursos, como habitaciones, equipos, personal, etc.
- **Gestión de tarifas y pagos:** Permite la fijación de precios, el procesamiento de pagos y la facturación asociada a las reservas.
- **Análisis de datos:** Incluyen una serie de gráficos y recopilación de datos que permiten tomar decisiones de negocios de valor.

### 2.2.3 Beneficios

Los Sistemas de Gestión de Turnos y Reservas se utilizan en diversos contextos, como en empresas de servicios, instalaciones deportivas, instituciones de salud, escuelas y muchas otras organizaciones. En general, cualquier institución que requiera atención personalizada a sus clientes, precisa de uno. Su utilización conlleva beneficios significativos:

- **Mayor eficiencia:** Simplifica la programación y la asignación de recursos, lo que ahorra tiempo y recursos administrativos.
- **Mejora de la experiencia del cliente:** Facilita a los usuarios la reserva de servicios y recursos según sus preferencias y disponibilidad.
- **Reducción de errores:** Minimiza los errores humanos en la asignación de turnos y reservas, lo que conduce a una gestión más precisa, sin solapamiento de turnos.

- **Optimización de la capacidad:** Permite a las organizaciones maximizar el uso de sus activos y recursos, lo que a menudo se traduce en un aumento de la rentabilidad.
- **Reducción de inasistencias:** La mayoría de sistemas suelen incluir medidas de prevención que permiten evitar o reducir la cantidad de inasistencias por parte de los clientes que lleva a grandes pérdidas de la institución que brinda el servicio.

## 2.3 Multiplataforma

En el dominio del Software, multiplataforma se refiere a, como su nombre lo indica, muchas plataformas, entendiéndose a “plataforma” como un entorno de ejecución. Es decir aquel contexto en el cual una aplicación vive. Algunos ejemplos de estas plataformas o entornos de ejecución son:

- **Windows:** Sistema operativo para computadoras, propiedad de Microsoft. Es donde se ejecutan la mayoría de aplicaciones para ordenadores convencionales.
- **MacOS:** Sistema operativo específico de las computadoras Mac, desarrollado por Apple.
- **Linux:** Un sistema operativo para computadoras que se caracteriza por ser de código abierto y gratuito. El cual suele ser más común en el mundo de los programadores debido a que ofrece mucha más flexibilidad que sus pares.
- **Android:** También se trata de un sistema operativo de código abierto y gratuito, pero desarrollado para dispositivos móviles como celulares y tabletas.

- **iOS:** Sistema operativo para dispositivos móviles, también es propiedad de Apple y al igual que MacOS con las computadoras Mac, únicamente es utilizado en sus productos particulares como los iPhones y iPads.

- **Web:** Este es el único caso de esta lista que no se trata de un sistema operativo, sino de un conjunto de tecnologías y herramientas que permiten desarrollar o acceder a una aplicación web.

### 2.3.1 Sistemas Multiplataforma

Teniendo en cuenta el término descrito, cuando se habla de Sistemas Multiplataforma se hace referencia a una aplicación que puede ser ejecutada en más de una plataforma. Una aplicación multiplataforma idealmente puede ser ejecutada en todas las plataformas listadas. Aunque este no es el caso de la mayoría, ya que, lograr soportar una gran cantidad de entornos, consecuentemente implica mayor complejidad, requiriendo de conocimientos de diversos lenguajes de programación y herramientas. Por ejemplo, el lenguaje de programación utilizado en Android y las herramientas necesarias para desarrollar en dicha plataforma, son completamente distintas a las de iOS, por lo que si se quiere desarrollar una aplicación que funcione en dispositivos móviles, de diversas marcas como Samsung, Motorola, LG, Apple, se necesita de conocimientos de dos lenguajes de programación diferentes y de diversas herramientas para su desarrollo. Es por esto que una aplicación multiplataforma no suele cubrir todas las plataformas mencionadas anteriormente, ni mucho menos otras plataformas no mencionadas, utilizadas en menor medida.

Generalmente un sistema o aplicación multiplataforma suele buscar soportar el entorno Web y de dispositivos móviles, Android e iOS. Esto es debido a su amplia portabilidad, se puede pensar en por ejemplo, un aplicación de mensajería instantánea.



Este tipo de aplicaciones suelen soportar estos entornos mencionados, y con esto, pueden usarse desde casi cualquier dispositivo. Si cuenta con una computadora e internet, puede acceder a la Web, y por lo tanto a la aplicación de mensajería instantánea. Si cuenta con un celular, también. Y eso es precisamente lo que se busca cuando se habla de “multiplataforma”, se busca portabilidad, en pocas palabras, abarcar un gran número de dispositivos en que la aplicación pueda funcionar correctamente, de modo que la aplicación pueda utilizarse desde la mayor cantidad de dispositivos posibles.

### **2.3.2 Frameworks Multiplataforma**

El diccionario de Cambridge lo define como “Estructura de soporte alrededor de la cual se puede construir algo”. En informática, no es más que una aplicación o herramienta que facilita el desarrollo de nuevas aplicaciones. Esto lo logra brindando componentes que los programadores pueden utilizar para desarrollar aplicaciones partiendo desde esos componentes y no desde cero. Es como construir una casa, el Framework le brinda los ladrillos y materiales para que uno pueda construir la casa más rápidamente porque fabricar estos materiales desde cero sería muchísimo más costoso en cuanto a tiempo. Una forma de lograr que un sistema sea ejecutado en varias plataformas de manera sencilla es por medio de un Framework Multiplataforma. Esto se debe a que el Framework permite con un sólo código fuente, desarrollar una aplicación para múltiples entornos o plataformas distintas como lo son la Web o los sistemas operativos móviles como Android e iOS. Si no se contara con un único código fuente, entonces se debería tener un código fuente para cada entorno, y como se mencionó, se requeriría de distintos lenguajes de programación, distintas herramientas y conocimientos para poder construir una aplicación que funciones en más

dispositivos. Por lo que la ventaja de un Framework Multiplataforma es evidente, no solo reduce tiempos, porque si uno quiere programar una aplicación disponible en dos entornos puede hacerlo con una sola aplicación en vez de dos (lo que reduce la carga de trabajo a la mitad) si no que los costos de desarrollo y mantenimiento se reducen enormemente también. Para dar un ejemplo, si una empresa, recién se lanza al mercado, cuenta con un capital reducido, y requiere una aplicación disponible en Android e iOS para que sus usuarios puedan comenzar a darle un uso, puede contratar a un equipo de desarrolladores que domine cierto Framework Multiplataforma, en vez de contratar un equipo de desarrolladores Android y otro equipo de desarrolladores iOS. Lógicamente la empresa podría contratar a un equipo que domine ambas tecnologías, pero ese equipo, no solo es más difícil de contratar debido a que requieren altos conocimientos, si no que no va a ser tan eficiente como uno que adopte un Framework Multiplataforma, el cual solo se centra en un código.

A los marcos de desarrollo o herramientas que se crearon y se utilizan específicamente para desarrollar en un único sistema operativo se los conoce como Nativo. Por ejemplo, Apple creó Swift, un lenguaje de programación específicamente para desarrollar apps en iOS, dicha herramienta, en este caso un lenguaje, se lo puede mencionar como Nativo. Por otro lado, las herramientas como cualquier Framework Multiplataforma, se suelen llamar Híbridos.

### **2.3.3 Flutter**

Flutter es un framework framework de código abierto desarrollado por Google, se utiliza “para construir aplicaciones hermosas, compiladas nativamente y multiplataforma desde un único código base”, según flutter.dev, sitio oficial del framework. Flutter compila el código tanto a código de máquina ARM o Intel como a JavaScript, lo que

permite un rendimiento rápido en cualquier dispositivo. Esto significa que puede ejecutarse de manera eficiente en una amplia gama de plataformas, incluyendo dispositivos móviles, computadoras de escritorio y la web. La capacidad de compilar a código de máquina nativo contribuye a un rendimiento óptimo en términos de velocidad y eficiencia de recursos, mientras que la compilación a JavaScript permite ejecutar aplicaciones Flutter en navegadores web sin necesidad de plugins adicionales. Esto hace que Flutter sea una opción versátil para el desarrollo de aplicaciones multiplataforma.

## 2.4 Arquitectura de Microservicios

En el contexto de la arquitectura de software moderna, los microservicios han surgido como una alternativa al enfoque monolítico tradicional. Según el sitio oficial de AWS, una arquitectura monolítica es un modelo de desarrollo de software tradicional que utiliza un código base para realizar varias funciones empresariales. Todos los componentes de software de un sistema monolítico son interdependientes debido a los mecanismos de intercambio de datos dentro del sistema. Modificar la arquitectura monolítica es restrictivo y lleva mucho tiempo, ya que los pequeños cambios afectan a grandes áreas del código base.

Por otro lado, según Newman (2015), los microservicios son "un enfoque de diseño que organiza una aplicación como una colección de servicios poco vinculados, implementables de forma independiente, altamente mantenidos y comprobables centrados en funciones comerciales".

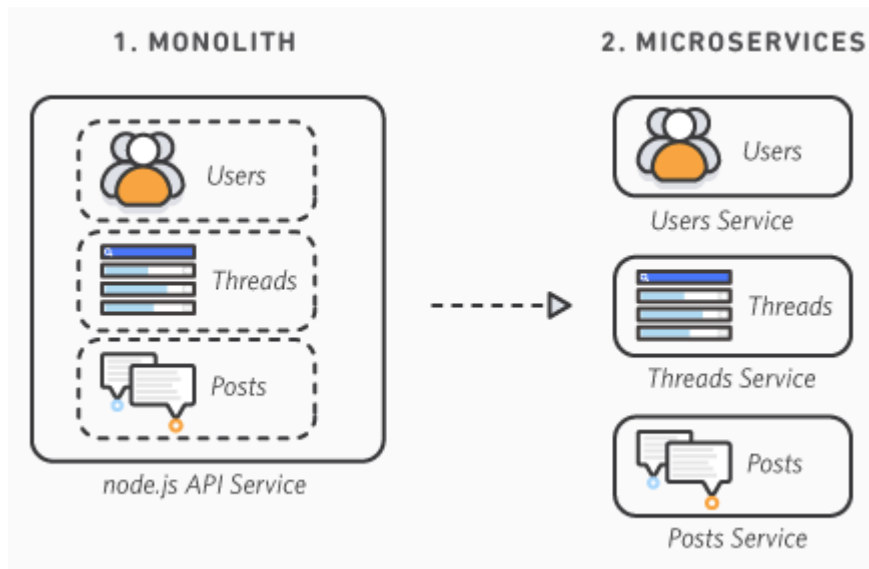


Figura 1 :Arquitecturas Monolítica y de Microservicios<sup>1</sup>

La definición anterior y el gráfico de la Figura 2 resalta la naturaleza modular y distribuida de los microservicios, lo que permite una mayor escalabilidad y flexibilidad en el desarrollo de aplicaciones, esto es una de las principales ventajas frente a las aplicaciones monolíticas. Newman, en su libro *Building Microservices* destaca siete beneficios clave de los microservicios sobre una aplicación monolítica.

- **Heterogeneidad tecnológica:** en una aplicación monolítica, todos los componentes se desarrollan con la misma tecnología. Esto no permite utilizar la tecnología óptima para un componente determinado en función de su funcionalidad prevista. Con el enfoque de microservicios, cada servicio puede elegir la mejor tecnología que se adapte a sus necesidades.
- **Resiliencia:** en una aplicación monolítica, si falla, todos sus componentes también fallan. Podemos ejecutar estas aplicaciones en varias máquinas para reducir la posibilidad de fallas, pero con los microservicios se puede

<sup>1</sup> Composición de una arquitectura monolítica versus una arquitectura de microservicios. Fuente: “¿Cuál es la diferencia entre la arquitectura monolítica y la de microservicios?”, Amazon Web Services.

- construir sistemas que manejen la falla total de los servicios y degraden la funcionalidad en consecuencia.
- **Escalado:** una aplicación monolítica lleva todos sus servicios en una sola unidad: esa es la aplicación monolítica en sí. Si desea escalar la aplicación, entonces necesita escalarla toda; no puede escalar cada componente individual por separado en función de su uso. Con los microservicios, se puede simplemente escalar aquellos servicios que necesitan escalabilidad, lo que nos permite ejecutar otras partes del sistema en hardware más pequeño y menos potente.
  - **Facilidad de implementación:** en aplicaciones monolíticas, incluso en un sistema que tenga las mejores pruebas automatizadas, un cambio de una línea introducido en un solo componente requeriría ejecutar todo el conjunto de pruebas. También requeriría lanzar el producto completo, por lo tanto, una nueva implementación. (Si cree que la aplicación de parches solucionaría esto, no es del todo cierto, ya que los parches se lanzan sólo para solucionar un problema, la introducción de nuevas funciones requeriría el lanzamiento de la aplicación monolítica completa). Con los microservicios se puede realizar un cambio en un único servicio y desplegarlo independientemente del resto del sistema.
  - **Alineación organizacional:** la idea de un “equipo de dos pizzas” fue acuñada por Jeff Bezo, el fundador de Amazon, y resalta la importancia de tener equipos más pequeños (un equipo debe ser lo suficientemente pequeño como para que todo el equipo pueda alimentarse con dos pizzas). La naturaleza de los microservicios permite alinear mejor una arquitectura con una organización. A diferencia de las aplicaciones monolíticas, el

dominio y el alcance de un microservicio determinado es pequeño y, por lo tanto, puede ser propiedad de un equipo más pequeño.

- **Componibilidad:** la reutilización de la funcionalidad es una promesa clave hecha en la arquitectura orientada a servicios. Los microservicios se pueden consumir de diferentes maneras para diferentes propósitos.
- **Optimización para la reemplazabilidad:** la mayoría de las aplicaciones heredadas son difíciles de reemplazar y, por la misma razón, se encuentran en muchas empresas creando barreras para la innovación. Estas aplicaciones monolíticas son difíciles de reemplazar, ya que nadie sabe qué pasaría incluso con la introducción de un pequeño cambio. Los microservicios eliminan este cuello de botella. Un microservicio puede ser reemplazado completamente por una nueva implementación sin demasiadas complicaciones.

Con el surgimiento de las aplicaciones basadas en arquitecturas de microservicios, se popularizaron aún más dos términos en el mundo de la programación, el Backend y Frontend. El acuñamiento se debe a que, generalmente la lógica de negocio de los sistemas se separa en microservicios, que es lo que se conoce como Backend, mientras que las aplicaciones con interfaces que interactúan con esos microservicios se conocen como Frontend. Esto significa que, por ejemplo, un desarrollador Frontend, es una persona que puede cambiar la forma o el color de un botón en una interfaz. Por el contrario, Backend hace referencia las aplicaciones que se ejecutan en un trasfondo y que el usuario no ve comúnmente. Esto incluye toda la lógica referente a el negocio, los datos, donde se almacenan, cómo se almacenan, cómo se recuperan y cómo se manipulan. Es común, y como se explicó en uno de sus beneficios, que las organizaciones tengan equipos específicamente dedicados al Backend y equipos dedicados al Frontend. Aunque

también pueden haber personas que manejen ambos, de hecho es el caso de este proyecto en el cual se necesitan habilidades dentro del área del Backend y dentro del área del Frontend para hacer posible el desarrollo del sistema, el dominio de ambos se conoce como Full-Stack.

## **2.5 Bases de Datos y Sistemas de Gestión**

"Un sistema gestor de bases de datos (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos, normalmente denominada Base de Datos, contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto práctica como eficiente. Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos para la manipulación de la información." (Abraham Silberschatz, 2002, p. 1).

Las bases de datos juegan un papel fundamental en el almacenamiento, organización y gestión de grandes cantidades de información en diversos ámbitos, desde empresas hasta aplicaciones web y sistemas de información.

### **2.5.1 Modelos de Bases de Datos**

Existen varios modelos de bases de datos que se utilizan para representar la estructura y la relación entre los datos. Entre los más comunes se encuentran el modelo relacional y el modelo no relacional (también conocido como NoSQL). Cada modelo

tiene sus propias características y ventajas, y la elección del modelo adecuado depende de los requisitos específicos de cada aplicación.

**Modelo Relacional:** En este modelo, la información se organiza en tablas que están interrelacionadas mediante llaves primarias y llaves foráneas. Cada tabla representa una entidad y cada fila en la tabla representa una instancia de esa entidad. Las consultas se realizan mediante un lenguaje estructurado de consultas (SQL), que permite realizar operaciones como selección, inserción, actualización y eliminación de datos de manera eficiente.

**Modelo No Relacional (NoSQL):** El modelo NoSQL, también conocido como no relacional, es una categoría de bases de datos que no sigue el modelo relacional tradicional. Estas bases de datos se utilizan principalmente para gestionar grandes volúmenes de datos no estructurados o semiestructurados. A diferencia de las bases de datos relacionales, en las bases de datos NoSQL no se requiere un esquema fijo y predefinido, lo que las hace más flexibles y escalables. Ejemplos de bases de datos NoSQL incluyen MongoDB, Cassandra y Redis.

## 2.6 BaaS

BaaS del inglés Backend as a Service, es como su nombre lo indica, un Backend como Servicio. Que un sistema esté basado, implementado o integrado en un BaaS se refiere, a muy grandes rasgos, a una solución que hace uso de los servicios de otro sistema. Generalmente cuando se desea desarrollar una aplicación, muchas veces requiere de, por ejemplo, usuarios y autenticación. Como esto no es una tarea fácil de desarrollar, y sobre todo que consume mucho tiempo, se puede usar un sistema que proporcione ese servicio, ahorrando un gran cantidad de tiempo, dependiendo de los



---

requisitos del mismo. Los usuarios y la autenticación es solo un ejemplo, existen múltiples funcionalidades que se repiten de sistema en sistema y que podrían ser reutilizadas, y es precisamente eso lo que ofrece un BaaS. Según Roman Zomko (2023), CEO y Cofundador de Impressit, “es un enfoque de computación en la nube que ofrece un backend para aplicaciones web y móviles. En pocas palabras, este método proporciona una API y un instrumento para que varios lenguajes informáticos se integren con su Backend. Además, BaaS existe para proporcionar algunos servicios cruciales, como almacenamiento, notificaciones push, análisis, etc. De esta forma, no es necesario crear el Backend desde cero, si no que puede aprovechar los SDK, las API y otras herramientas existentes para obtener un Backend que pueda escalarse. En el mejor de los casos, cuando se tiene un Backend proporcionado por una plataforma BaaS, no hay necesidad de dedicar demasiado tiempo a mantener el Backend, en cambio, se puede concentrar en mejorar la experiencia del usuario”.

Lógicamente como se cuenta con ciertos beneficios, hay consecuencias: el precio y la flexibilidad. Al ser un servicio que se consume, lógicamente viene arraigado a un precio que el consumidor debe pagar. El buen punto de esta contrapartida es que muchos BaaS ofrecen planes gratuitos hasta que se llega a cierto límite en donde se debe comenzar a pagar. Pero estos límites suelen ser lo suficientemente altos para acompañar a quien desarrolla el software en sus niveles iniciales y no tenga que realizar ningún tipo de gasto hasta que el mismo comience a tener éxito. Por ejemplo, un BaaS puede ofrecer un sistema de usuarios y autenticación en donde se comienza a pagar cuando se alcanzan los 10.000 usuarios. Esto le da un buen marco a quien consume el servicio para asegurarse de pagar solo si el sistema prospera, es un punto muy importante. En cuanto a la flexibilidad, puede ser un punto negativo si se quiere desarrollar una aplicación que tenga

funcionalidades muy específicas que el propio BaaS no puede ofrecer o no permite extender.

Algunos ejemplos de BaaS que se encuentran actualmente en el mercado, aptos para ser usado para cualquiera que se registre son:

- Firebase
- Heroku
- Strapi
- Supabase
- AWS Amplify
- Appwrite
- Nhost

### **2.6.1 Firebase**

Para este proyecto se escogió Firebase, el cual según el sitio oficial, ofrece un basta cantidad de servicios: Bases de datos en tiempo real, configuración remota, extensiones, chequeo del estado de la app, funciones en la nube, autenticación, mensajes en la nube, hosting, almacenamiento en la nube y Machine Learning.

El sistema que se desarrolla requiere de una base de datos con usuarios, un servidor que permita autenticarlos y registrarlos. También necesita un servicio que le permita enviar notificaciones a dispositivos móviles o a la Web. Es fundamental que el sistema cuenta con una herramienta que le permita controlar las fallas que ocurren de manera aislada en los dispositivos móviles y en la web. Por último, precisa de un servidor que le permita a la Web poder alojarse.

Todos estos requerimientos nombrados, son fundamentales para garantizar una aplicación que sea beneficiosa para sus usuarios. Pero también son requerimientos que son complejos e involucran mucho conocimiento y tecnologías. Desarrollar todos ellos podría llevar años si se trata de un desarrollador o incluso de un equipo de desarrolladores. Afortunadamente, Firebase proporciona todos estos servicios al alcance de los programadores, por lo que solo es cuestión de integrarlos y hacer uso de ellos. Lógicamente la integración no es fácil, e involucra muchos conocimientos técnicos, pero implementarlos desde cero, involucraría aún más tiempo y conocimientos.

## **2.7 Cloud Servers**

El término Cloud Servers se puede traducir al español como “Servidores en la Nube”, pero el término en español es muy poco usado, de hecho los proveedores a menudo venden sus servicios refiriéndose a ellos como “Cloud Servers” y no “Servidores en la Nubes” debido a que el uso de la primera ha sido estandarizado. Según IBM un Cloud Server es una poderosa infraestructura física o virtual que entrega aplicaciones, procesa información o proporciona almacenamiento de datos. Algunos servidores en la nube se crean utilizando software de virtualización que divide un único servidor físico en varios servidores virtuales. Los proveedores de servicios en la nube utilizan un modelo de infraestructura como servicio (IaaS) para poner servidores virtuales o un servidor físico a disposición de los clientes.

### 2.7.1 Ventajas frente a Servidores Tradicionales

La idea de los Cloud Servers es proporcionar una alternativa a los servidores tradicionales. Si un sistema requiere del procesamiento de datos, conexión a una base de datos, envío de emails, integraciones, entre otras acciones, dicha aplicación requiere de un servidor. Los servidores tradicionales son capaces de almacenar y correr constantemente estos sistemas que procesan datos. Están conformados por estructuras de hardware que son muy costosas así como también con software especialmente diseñado para ellos. En contrapartida a los servidores tradicionales, los Cloud Servers son fáciles de contratar y requieren casi nulo conocimiento de hardware. Existen varias razones por las cuales los Cloud Server han ido en creciente aumento los últimos años, algunas de sus ventajas frente a los servidores tradicionales son:

**Flexibilidad y escalabilidad:** Se puede acceder a recursos adicionales cuando es necesario. Esto es particularmente valioso para los clientes que tienen picos en el recurso requerido en ciertas épocas del año, o aquellos cuyo recurso es difícil de predecir.

Si se está intentando vender un producto que consta de un software, es muy posible que a edades tempranas el mismo cuente con una baja cantidad de usuarios. Pero a medida que el negocio tiene éxito, los usuarios comienzan a incrementarse, esto a su vez significa aumentos en la red, el tráfico y los recursos que se utilizan dentro de un sistema, por lo que si no se escala con el hardware necesario, la aplicación entera puede terminar colapsando. Un proveedor de un Cloud Server es capaz de escalar estos recursos en tan solo minutos.

**Rentabilidad:** Como se dijo, contar con un servidor tradicional es muy costoso, por todo el conocimiento técnico que requiere y por los precios del

hardware. Además no solo son los costos iniciales, si no que también se requieren costos adicionales para mantenimiento del hardware, por si algún componente deja de funcionar o necesita una mejora. Las licencias de software que ayudan a administrar un servidor también son costosas y, de hecho, muchas involucran suscripciones mensuales. Todos estos costos pueden ser neutralizados si reemplazamos un servidor tradicional con un Cloud Server. El mismo requiere un pago mensual a un proveedor, que suele ser relativamente económico si se tiene en cuenta todos los gastos mencionados.

**Confiableidad:** Los servidores en la nube son mucho más confiables que los servidores tradicionales. Un proveedor de un Cloud Server cuenta con un espacio especializado para el despliegue de servidores, por lo que se cuenta con múltiples medidas para evitar el sobrecalentamiento de los componentes, para reducir las probabilidades de cortes de luz o de posibles incendios. La mayoría de proveedores cuentan con sistemas de copias de respaldo así como también con múltiples nodos para que en caso de que alguno deje de funcionar, haya una réplica de inmediato que pueda reemplazarlo sin afectar al cliente.

**Seguridad:** Los proveedores de Cloud Servers cuentan con todo tipo de seguridad para que los datos siempre estén a salvo. Un ejemplo muy común es el Firewall que suelen ofrecer. El mismo puede filtrar llamadas según la IP, evitando que ingresen peticiones no deseadas al servidor.

Algunas claras desventajas son por ejemplo, el hecho de que los datos estén en manos de un tercero, en este caso un proveedor, siempre es prudente evitarlo. La sola dependencia de un tercero es una desventaja. Por ejemplo, ¿Qué sucede si el proveedor de Cloud Server entra en bancarrota? Muy probablemente se perderían los datos y las

aplicaciones dejarían de funcionar. De todas formas estas son desventajas que se pueden evitar haciendo un buen análisis y elección de proveedor. Por otro lado, las ventajas antes mencionadas conllevan a dos características imprescindibles: reducir costos y tiempos.

### **2.7.2 DonWeb**

Para este proyecto en particular, se escogió DonWeb, plataforma que ofrece múltiples servicios como: dominios Web y de correo, Web hosting, Windows hosting, Wordpress hosting, email marketing, email transaccional, certificados SSL, Servidores Dedicados y Cloud Servers. Algunas de las características que brindan para los Cloud Servers, el cual es el servicio contratado en este proyecto son el despliegue rápido, inclusión de SSL, inclusión de dos IP públicas, una IPv4 y otra IPv6, backups semanales, snapshots, estadísticas útiles, licencias de software, Firewall y consola para el acceso mediante la web.

## **2.8 Contenedores**

Antiguamente cuando se quería ejecutar una aplicación web, se compraba un servidor, se instalaba Linux, se configuraba una pila LAMP y se ejecutaba la aplicación. Si la aplicación se hacía popular, se debía poner un buen equilibrio de carga configurando un segundo servidor para garantizar que la aplicación no fallara debido a demasiado tráfico. Sin embargo, los tiempos cambiaron y, en lugar de centrarse en servidores únicos, Internet se basa en conjuntos de servidores redundantes e interdependientes en un sistema comúnmente llamado "la nube". El concepto de servidor pudo eliminarse de las limitaciones del hardware y, en su lugar, convertirse en una pieza de software. Estos

servidores basados en software se denominan contenedores y son una combinación híbrida del sistema operativo Linux en el que se ejecutan y un entorno de ejecución hiper localizado (el contenido del contenedor).

La tecnología de contenedores se puede considerar como tres categorías diferentes:

**Builder:** una herramienta o serie de herramientas utilizadas para construir un contenedor.

**Motor:** una aplicación utilizada para ejecutar un contenedor.

**Orquestación:** tecnología utilizada para gestionar muchos contenedores.

Los contenedores a menudo entregan tanto una aplicación como una configuración, lo que significa que un administrador de sistemas no tiene que dedicar tanto tiempo a ejecutar una aplicación en un contenedor en comparación con cuando una aplicación se instala desde una fuente tradicional. Dockerhub y Quay.io son repositorios que ofrecen imágenes para su uso en motores de contenedores.

Sin embargo, el mayor atractivo de los contenedores es su capacidad de "morir" y reaparecer cuando el equilibrio de carga lo exige. Ya sea que la desaparición de un contenedor se deba a una falla o simplemente porque ya no es necesario porque el tráfico del servidor es bajo, los contenedores son "baratos" para comenzar y están diseñados para aparecer y desaparecer sin problemas. Debido a que los contenedores están destinados a ser efímeros y generar nuevas instancias con la frecuencia necesaria, se espera que un humano no realice su monitoreo y administración en tiempo real, sino que esté automatizado.

## 2.8.1 Docker

Docker es un framework para crear, ejecutar y administrar contenedores en servidores y la nube. El término "docker" puede referirse a las herramientas (los comandos y un demonio) o al formato de archivo Dockerfile.

En este proyecto, se utilizan los contenedores a través de docker, con el objetivo de poder construir una imagen del Backend, la cual contiene y procesa la lógica de negocio, y de la Base de Datos, la cual almacena los datos que procesa dicho Backend. De forma que sea fácil de parar, iniciar, reiniciar o eliminar dichos contenedores en el Cloud Server. En este caso Docker y sus contenedores se utilizan como una herramienta para administrar fácilmente las aplicaciones para que todo el sistema completo funcione correctamente y esté disponible para los usuarios finales en el momento que se lo requiera.



# Capítulo 3 - Análisis y Diseño

En este capítulo se aborda el análisis realizado para poder posteriormente contemplar el diseño y luego la implementación del sistema. El mismo se orienta a cualquier organización, empresa, entidad, o negocio del rubro deportivo que requiera de la gestión de alquiler de canchas para su uso. Cumplen con este requisito esencialmente complejos deportivos y clubes.

Estas organizaciones buscan constantemente formas innovadoras de mejorar la gestión de sus recursos y servicios. En respuesta a esta necesidad, surge la propuesta de un sistema multiplataforma basado en microservicios, BaaS (Backend as a Service), Cloud Servers y Contenedores, diseñado para optimizar las reservas deportivas y mejorar la eficiencia operativa. Este enfoque no solo representa una solución integral para una organización deportiva específica, sino que también es adaptable y escalable para satisfacer las necesidades de una amplia gama de organizaciones en el ámbito deportivo.

Para poder comenzar con la implementación, el análisis y diseño se lleva a cabo siguiendo una metodología incremental y de prototipo. Incremental, porque se va construyendo el producto final de manera progresiva, en donde el software se empieza a usar antes de que esté completo y de prototipo porque se basa en un prototipo de software que se construye rápidamente para que los usuarios puedan probarlo y aportar feedback. Así, se puede arreglar lo que está mal e incluir otros requerimientos que puedan surgir. Es un modelo iterativo que se basa en el método de prueba y error para comprender las especificidades del producto.

## 3.1 Orientación General: Solución para el rubro deportivo

El desarrollo de un sistema de optimización de reservas deportivas basado en tecnologías emergentes como microservicios, BaaS, Cloud Servers y Contenedores se presenta como una solución universal para abordar los desafíos comunes que enfrentan las organizaciones deportivas en la gestión de sus instalaciones y actividades. Este enfoque ofrece flexibilidad, escalabilidad y eficiencia, lo que permite a las organizaciones deportivas adaptarse rápidamente a las demandas cambiantes del mercado y optimizar sus recursos de manera efectiva.

Es importante tener en cuenta que un sistema desarrollado a medida ofrece soluciones más específicas, y por ende, más eficientes y valiosas. Pero una solución a medida, es como su nombre lo indica, una solución aplicable a un único caso, como una organización y por consiguiente, más costosa de adquirir. En contrapartida, un sistema más genérico, es aplicable a un gran número de casos de uso y más económico.

En el momento de diseñar un sistema se requiere un buen dominio en el problema, en este caso en el área deportiva, específicamente en la reserva de canchas. El buen dominio del mismo y de todo su entorno permite reducir costos y tiempo en el desarrollo. Es de extrema importancia garantizar los límites necesarios. Por ejemplo, si se desarrolla un sistema demasiado genérico, el mismo aumenta en complejidad y por consiguiente en tiempos y costos. A grandes rasgos, un factor determinante de la complejidad está dado por la flexibilidad. Si se quiere desarrollar un sistema de reservas por ejemplo, para cualquier rubro, es decir, muy flexible, el número de casos de usos, los requisitos y las

necesidades son incontables. Esto no solo trae problemas y complejidad de diseño, sino que también aumenta la cantidad de posibles errores, de información que se maneja, entre otros.

Para poder lograr tener un dominio del problema se realiza un caso de estudio específico en un complejo deportivo que se enfrenta en el día a día al problema de las reservas, la organización y la administración de las mismas. Para así tener una visión más amplia, detectar requerimientos, casos de uso y problemas y a modo organizativo, se realizan diagramas de clases y de arquitecturas. Teniendo en cuenta que como en un principio no se domina del todo el problema, estos están sujetos a modificaciones permanentes para lograr así una versión final y representativa.

## **3.2 Caso de Estudio: Análisis de requerimientos para una organización deportiva particular**

A modo de acotar el problema y partir de una solución básica para poder así comenzar a escalar a una más compleja, se toma un complejo deportivo como caso de estudio. El mismo cuenta con las siguientes características principales:

- El único deporte con el que cuenta es el tenis.
- Tiene 8 canchas disponibles para alquiler.
- Todas las canchas son de polvo de ladrillo.
- Todas las canchas a excepción de 2 de ellas cuentan con luz artificial.
- Las reservas son atendidas ya sea por Whatsapp o por llamadas.

- Las reservas se registran en una computadora a través del programa Excel.
- Los empleados del establecimiento se encargan del mantenimiento del mismo así como también de la administración de reservas y pagos.
- Los horarios de apertura y cierre del establecimiento varían constantemente de acuerdo a la demanda y el clima de la ciudad.
- Los pagos se realizan siempre en el lugar, ya sea con efectivo o a través de la plataforma Mercado Pago.
- El precio del alquiler varía según la hora y la cancha.
- Se ofrecen servicios complementarios como torneos y clases de tenis
- El establecimiento cuenta con estacionamiento y vestuarios.
- Al final de cada mes se realizan tareas financieras y contables.

Se observa y analiza detenidamente el modo de uso del software Excel que se utiliza en la institución con el fin de entender la mejor solución al problema y en donde predomine la automatización para una gestión eficiente.

Hora	Cancha 1	Cancha 2	Cancha 3	Cancha 4	Cancha 5	Cancha 6	Cancha 7	Cancha 8
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								

Figura 2: Planilla Diaria<sup>2</sup>

<sup>2</sup> Hoja vacía que se crea al comienzo de cada día. Gráfico tomado del software Excel, utilizado por la institución bajo estudio para organizar los turnos.

Hora	Cancha 1	Cancha 2	Cancha 3	Cancha 4	Cancha 5	Cancha 6	Cancha 7	Cancha 8
9								
10								
11								
12		Ramirez						Escuela
13								
14				Torres				
15								
16						Cáceres		
17								
18								
19								
20								
21			Prieto					
22		Munizaga						
23								

Figura 3: Planilla de Reservas<sup>3</sup>

En la *Figura 2* puede observarse cómo la institución crea una hoja de cálculo al comienzo del día utilizando un software de Planilla de Cálculo. La hoja de cálculo no es más que una tabla, en donde las horas representan las filas y las canchas, las columnas. De este modo se puede ver claramente que el club cuenta con 8 canchas cuyo horario de apertura es a las 9 horas de la mañana y cierre a las 23 horas de la noche.

En la *Figura 3* se observa la misma hoja del día, pero ya completada con las actividades que ocurrieron en el transcurso del día. El rellenado, consiste básicamente en colorear una celda y colocar el apellido de la persona que reserva. Esto lógicamente haciéndolo en la columna que corresponda, según la cancha en la que se va a jugar, y extendiéndose en las filas dependiendo de la cantidad de horas a jugar. Se puede notar que la reserva de la cancha 8, no es un reserva común, es decir de un cliente, si no que ese horario, de 12 hs a 23hs está reservado y dedicado para actividades de escuela de tenis que ocurren en la institución.

El proceso de creación de la hoja se realiza todos los días, copiando la hoja del día anterior, pegándole en una nueva hoja y borrando las reservas existentes a modo de tener una tabla vacía lista para ser utilizada.

<sup>3</sup> Gráfico tomado del software de Planilla de Cálculo utilizado por la institución bajo estudio para organizar los turnos.

Si se centra específicamente en esta funcionalidad o módulo, la de administrar reservas, la cual es el punto de inflexión principal para tratar el problema, es posible crear un software que automatice el flujo de trabajo utilizado, y con posibilidad de muchas mejoras. Se puede crear un algoritmo que cree una representación de las tablas mostradas en las figuras 2 y 3 simplemente guardando cierta información como la cantidad de canchas y el horario de apertura y cierre. Pero, porque se busca un software más genérico, que solucione no solo el problema de este club en particular, sino también el de cualquier otro que se encuentre en el rubro deportivo y requiera de reservas, se tienen en cuenta otros parámetros adicionales y algunos casos de uso específicos. Además, se puede crear una tabla por cada deporte perteneciente a un club, así el complejo deportivo puede contar con múltiples planillas una para cada deporte según las canchas de las cuales disponga.

Desde ahora en adelante, en este trabajo, se denominará “Módulo de Administración de Reservas” al módulo que soluciona el problema descrito en el cual un club tiene que llevar un seguimiento a las reservas que suceden en distintas canchas con el fin de evitar solapamientos y tener una mejor organización.

El Módulo de Administración de Reservas tiene varios puntos claves que se pueden desarrollar con más detalle para obtener una buena automatización y optimización del control de las reservas.

Si bien el club tomado como caso de estudio, utiliza su tabla con horas, es mejor tener filas cada media hora, debido a que aumenta la flexibilidad para poder registrar una reserva. Lo mismo sucede con el número de canchas. Si bien el club siempre contó con 8 canchas, existe la posibilidad de agregar nuevas. O incluso los clubes deben poder eliminar canchas con libertad.

El horario de apertura y cierre para considerar la creación de una tabla de reservas, no es relativo al club o complejo deportivo, sino que es relativo a cada cancha, esto es

debido a que no todas las canchas tienen los mismos horarios. Por ejemplo, un club puede que tenga una cancha con luz, la cual abre de 9 horas a 23 horas y puede que tenga una cancha sin luz, como ocurre en este establecimiento, la cual abre de 9hs a 19hs. Tomando todas las canchas de un establecimiento, el horario de apertura del mismo es el de aquella cancha con el horario más temprano, y el horario de cierre del club es el de aquella cancha con el horario más tardío.

Hay clubes en donde los horarios de las canchas no son de corrido. Por ejemplo, provincias como San Juan, donde el calor en el horario de la siesta es muy alto, muchos complejos deciden cerrar sus puertas durante unas horas.

Otro detalle a tener en cuenta es que existen distintos tipos de reserva. No todas las reservas son de uso cotidiano por parte de los clientes, hay reservas propias del club, como por ejemplo reservas de horarios para la escuelita, en este caso de tenis, para realizar un torneo o incluso para realizar un evento, como un cumpleaños.

En cuanto a los precios, las reservas suelen tener el precio de la cancha, el cual varía según las condiciones de la misma. Una cancha que cuente con un material más caro, tiene un precio más elevado. Sin embargo, existen casos excepcionales en que el precio es propio de la reserva y no de la cancha en la cual se encuentra. Por ejemplo, en una reserva destinada a un evento de cumpleaños, el precio puede ser más alto de lo habitual.

También es importante notar que hay reservas que se repiten cada cierto tiempo, pueden repetirse todos los días, todas las semanas, todos los fines de semana, etcétera. Es el caso de un “turno fijo”, nombre con el cual se conoce en el club a las reservas de un grupo de jugadores que, por ejemplo, tienen un turno o reserva fija todos los lunes.

La repetición de estas reservas, es finita, normalmente se realiza durante un mes, y luego se renueva.



De las características listadas y del análisis realizado, se obtiene una nueva lista de requisitos funcionales y no funcionales. Estos requerimientos, se definen de forma genérica, es decir teniendo en cuenta, múltiples deportes y clubes. El club bajo estudio simplemente tiene como objetivo aclarar los requerimientos, facilitando la comprensión del dominio y problema.

### **Requisitos Funcionales:**

- Selección de deportes: El sistema debe permitir tanto a clientes como a clubes seleccionar y cambiar de deporte. Es decir el sistema no se limita a un único deporte. Un cliente puede realizar reservas de tenis y pádel, y un club puede gestionar sus reservas de tenis y padel, por ejemplo.
- Administración de horarios: El sistema debe permitir al personal del establecimiento crear, editar y eliminar canchas por cada deporte. En donde, por cada cancha, sea posible gestionar eficientemente la disponibilidad horaria.
- Creación y gestión de reservas: El sistema le debe permitir al complejo deportivo, crear, editar y eliminar reservas para cada cancha de un deporte seleccionado.
- Automatización de reservas: En caso de que el club requiere de una reserva repetitiva, como un turno fijo, el sistema debe poder generar una reserva reiteradamente durante cierta cantidad de tiempo según los parámetros de repetición elegidos, como “Todos los días”, “De Lunes a Viernes”, “Todas las Semanas”.
- Generación de Reportes Financieros y Contables: El sistema debe generar reportes financieros, por deporte, los cuales se puedan filtrar diaria o

mensualmente. Los reportes incluyen los ingresos de acuerdo a las reservas.

- Registro de clientes: El sistema debe permitir el registro de clientes de clubes, es decir, personas dispuestas a reservar en un club para posteriormente practicar un deporte.
- Inicio de sesión: El sistema debe permitir el inicio de sesión e ingreso de los clientes y complejos deportivos.
- Reservas en Línea. El sistema debe permitir a los clientes realizar reservas de canchas de un deporte en línea de manera eficiente y a través de un proceso simple e intuitivo. Las mismas, deben poder confirmarse por parte del complejo deportivo.
- Historial y preferencias: El sistema debe facilitar el seguimiento de las preferencias y el historial de reservas de los clientes.

**Requisitos No Funcionales:**

- Portabilidad: El sistema debe ser portable y funcionar en múltiples plataformas, incluyendo iOS, Android y Web.
- Seguridad: El sistema debe garantizar la seguridad de los datos personales y financieros de los clientes mediante la implementación de medidas de seguridad adecuadas.
- Usabilidad: El sistema debe tener una interfaz de usuario intuitiva y fácil de usar tanto para los clientes como para el personal de los establecimientos deportivos.
- Escalabilidad: El sistema debe ser diseñado siguiendo buenas prácticas para facilitar la escalabilidad en caso de requerir de miles transacciones diarias.

- Disponibilidad: El sistema debe estar disponible las 24 horas del día, los 7 días de la semana.
- Rendimiento: El sistema debe estar optimizado para garantizar tiempos de respuesta rápidos y eficientes, incluso durante períodos de alta demanda.

### **3.2.1 Módulo de Administración de Reservas**

Partiendo del problema planteado, el análisis detallado realizado, el listado de requerimientos y del método actual utilizado por el club para lograr la administración de reservas, se realiza el siguiente listado de funcionalidades más específico con las cuales el Módulo de Administración de Reservas debe contar para que un club logre la administración de reservas de forma que se automatice la mayor cantidad de procesos que el club realiza. Un club o complejo deportivo, a través del sistema, debe poder:

- Crear, eliminar o editar canchas. Cuando una cancha es creada, consecuentemente, una columna es agregada en la tabla. Cuando es eliminada, la tabla cuenta con una columna menos.
- Al momento de crear o editar una cancha, debe guardar cierta información relativa a la cancha primordial para la gestión: Nombre, Horario, Horario discontinuo, Precio por hora, Tipo de superficie.
  - Nombre: Es el nombre de la cancha, en este caso, el club coloca los nombres con numeraciones, por ejemplo “Cancha 2”. Campo obligatorio.
  - Horario: Es un rango en donde se debe elegir una hora de apertura y una de cierre, por ejemplo “9:00 hs a 21:00 hs”. Campo obligatorio.

- Horario discontinuo: Es un rango en donde se debe elegir una hora inicial y una hora final, en la cual la cancha está cerrada. Por ejemplo “15:30 hs a 17:00 hs”. Campo opcional.
- Precio por hora: Es el precio del valor por hora que tiene la cancha, este es el precio por defecto que toman las reservas que se crean en la cancha. Campo opcional.
- Tipo de superficie: Es una descripción del tipo de superficie que tiene la cancha. Campo opcional.
- Crear, eliminar o editar reservas. Cuando una reserva es creada, consecuentemente, una celda es rellenada en la tabla. Cuando es eliminada, la tabla vuelve a mostrar un lugar vacío en el lugar donde se encontraba la reserva.
- Al momento de crear o editar una reserva, debe guardar cierta información relativa a la reserva para la gestión: Fecha, Horario, Tipo, Cancha, Repetición, Nombre, Precio por hora, Precio de luz por hora.
  - Fecha: Es la fecha de la reserva, lógicamente se debe guardar, día, mes y año. Campo obligatorio.
  - Horario: Es un rango en donde se debe elegir una hora de inicio de la reserva y un horario de fin, por ejemplo “10:00 hs a 11:30 hs”. Campo obligatorio.
  - Tipo: Es una opción, la cual debe elegirse entre varias: Alquiler, Torneo, Escuela, Evento, Bloqueo y Cancelado.
    - Alquiler: Este es el tipo de reserva más común o regular, dedicada a los clientes del club, quienes reservan para la práctica de un deporte.

- 
- Torneo: Este tipo de reserva está destinado a representar un torneo ya sea organizado por el club o no.
  - Escuela: Es un tipo de reserva útil para instituciones que ofrecen escuelas con profesores dedicados a la enseñanza de un deporte.
  - Evento: Tipo de reserva para eventos, como un cumpleaños o una celebración del club.
  - Bloqueo: Este no es un tipo de reserva, si no que se utiliza para manifestar que un horario está bloqueado, puede ser por ejemplo por refacciones en la cancha en el horario deseado.
  - Cancelado: Este tipo de reserva se utiliza para cuando el club no quiere eliminar el espacio de reserva de la tabla, si no que pretende dejar retratado que hubo una reserva cancelada. Una reserva puede ser de tipo Alquiler, y el club posteriormente puede editarla y cambiarla a Cancelado en caso de que sea necesario.
  - Cancha: Es una opción, la cual debe elegirse de entre las canchas que el club ha creado. Campo obligatorio.
  - Repetición: Es una opción, la cual debe elegirse entre varias: No se repite, Todas las semanas, De lunes a viernes, Todos los días. Campo obligatorio.
    - No se repite: La reserva se crea únicamente en la fecha seleccionada.
    - Todas las semanas: La reserva se repite todos los días del día de la semana seleccionada hasta finalizar el mes. Por ejemplo, Todos los lunes.
    - De lunes a viernes: La reserva se repite desde la fecha seleccionada hasta finalizar el mes, obviando los fines de semana.

- Todos los días. La reserva se repite desde la fecha seleccionada hasta finalizar el mes.
- Nombre: Es el nombre de la reserva, por ejemplo “Juan Perez”. Campo obligatorio
- Precio por hora. Es el precio de la reserva por hora. Si la reserva se elige en una cancha que tiene un precio por defecto, entonces se toma el precio de la cancha, sujeto a modificación del usuario. Sino, el usuario puede ingresar o no un precio relativo a la hora. Campo opcional.
- Precio de luz por hora. Es el precio de la reserva de luz por hora. Campo opcional.
- Ver cualquier fecha de la tabla, y las reservas guardadas en ella.
- Cambiar de deporte y ver las reservas del deporte seleccionado.

El Módulo de Administración de Reservas es entonces, la pieza de software que le permite a un complejo deportivo contar con una planilla o tabla la cual sirve para administrar reservas de diversos tipos. La tabla se crea o modifica conforme el club añade canchas con horarios, siendo las columnas de la tabla, las canchas y siendo las filas, las horas definidas por esas canchas, con un incremento de media hora.

A continuación se muestran prototipo de la interfaz de usuario a modo de tener un mejor panorama y entendimiento de la funcionalidad del Módulo de Administración de Reservas

① Tenis								
② ◀ 10/01/2023 ▶								
Hora	Cancha 1	Cancha 2	Cancha 3	Cancha 4	Cancha 5	Cancha 6	Cancha 7	
8:00			⑧ Torres					
8:30								
9:00								
9:30								
10:00								
10:30	Sánchez	Perez						
11:00								
11:30						Alcaraz	Evento Lopez	
12:00								
12:30	Almirón							
13:00								
13:30	Zapata							
14:00								
14:30		Escuela	Escuela		Suarez			
15:00								
15:30								
16:00							Bloqueo por refacciones	
16:30								
17:00								
17:30					Ibraz		⑦ +	
18:00								
18:30								

Figura 4: Prototipo de Interfaz del Módulo Administración de Reservas<sup>4</sup>

En la Figura 4 se pueden apreciar anotaciones englobadas. El elemento 1, hace referencia al deporte seleccionado por el club o complejo deportivo. Haciendo click sobre el mismo, se despliega una lista de opciones de los deportes que han sido asociados a dicho club. Si selecciona un deporte de la lista entonces la planilla o tabla cambia a la de dicha selección. De este modo un club puede contar con diversas tablas o planillas para cada deporte que requiera.

El elemento 2, hace referencia a la fecha en la cual se encuentra el club. Haciendo click sobre la flecha ubicada a la izquierda, retrocede un día en el calendario. Haciendo click sobre la flecha ubicada a la derecha, avanza un día. De esta forma se puede ir moviendo rápidamente en los días adyacentes para ver fácilmente las distintas reservas que un club tiene en fechas próximas o anteriores.

<sup>4</sup> Fuente: elaboración propia.



2.1 Seleccionar fecha

Enero 2023

1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27
28	29	30	31					

Cancelar Aceptar

Figura 5: Prototipo de la interfaz de selección de fecha

Si el club desea visualizar una fecha en específico, puede hacerlo rápidamente presionando sobre la fecha, lo cual abre el prototipo de la figura 5. El mismo no es más que una representación de un componente que emula un calendario y el cual permite seleccionar cualquier fecha. De este modo, el usuario puede cambiar de fecha de dos formas, a través de un acceso rápido, es decir las flechas, o abriendo un calendario donde el usuario posee mayor flexibilidad a la hora de elegir.

El elemento 3 de la Figura 4, es un botón de menú, el cual se encuentra posicionado en la esquina superior derecha. Al presionarlo abre un listado de opciones, éstas pueden ser acciones específicas o configuraciones.

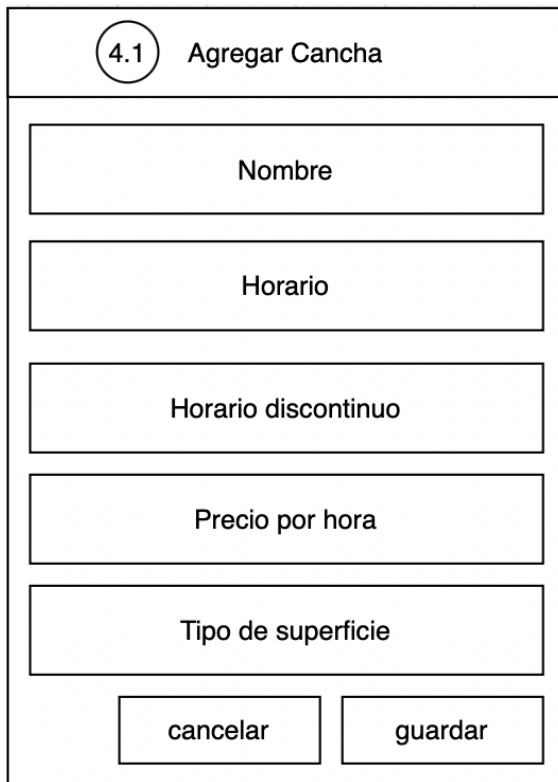


① Tenis						④ Agregar Cancha
② ◀ 10/01/2023 ▶						
Hora	Cancha 1	Cancha 2	Cancha 3	Cancha 4	Cancha 5	⑥ Eliminar Cancha
8:00			⑧ Torres			
8:30						
9:00						
9:30						
10:00						
10:30	Sánchez	Perez				
11:00						
11:30						
12:00						
12:30	Almirón					
13:00						
13:30	Zapata					
14:00						
14:30		Escuela	Escuela		Suarez	
15:00						
15:30						
16:00					Bloqueo por refacciones	
16:30						
17:00						
17:30					Ibraz	
18:00						
18:30						

Figura 6: Prototipo de la interfaz del menú de opciones

La Figura 6 muestra un panel el cual se despliega al presionar el elemento 3 anteriormente mencionado. Se pueden observar 3 opciones, Agregar Cancha, Editar Cancha y Eliminar Cancha. Por el momento, se hace especial énfasis en lo justo y necesario para solucionar el problema de administración de reservas en clubes, por lo que se obvian otras posibles opciones.

Al presionar el elemento 4, se abre un formulario con la información necesaria a guardar. El formulario cuenta con la información definida previamente.



4.1 Agregar Cancha

Nombre

Horario

Horario discontinuo

Precio por hora

Tipo de superficie

cancelar guardar

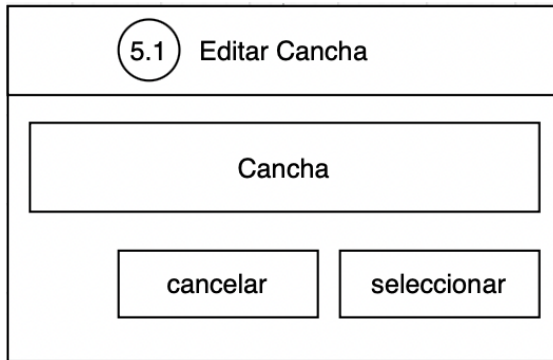
Figura 7: Prototipo de la interfaz del formulario de creación de una cancha

La Figura 7 retrata de forma básica la creación de una cancha por parte de los usuarios o clubes. A modo de ejemplo, si se vuelve a la Figura 4, se puede notar que la “Cancha 4”, es una cancha en cuyo formulario se ingresa lo siguiente:

- Nombre: Cancha 4
- Horario: de 8 hs a 18:30 hs
- Horario discontinuo: de 11:30 hs a 14:00 hs

Se obvian los campos Precio por hora y Tipo de superficie ya que, por el momento, no tienen una representación gráfica en la figura mostrada.

Volviendo a la Figura 6 los prototipos de interfaces para seleccionar, editar y eliminar una cancha al presionar los elementos 5 y 6 respectivamente son los siguientes.



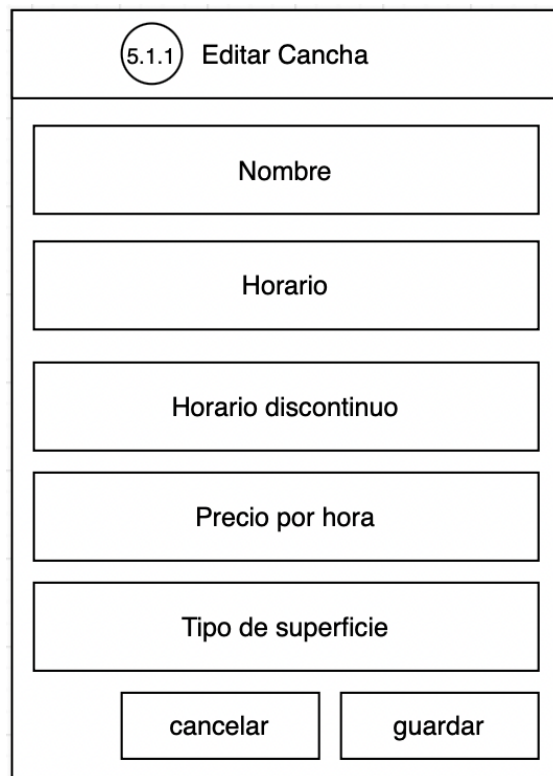
5.1 Editar Cancha

Cancha

cancelar seleccionar

Detailed description: This is a wireframe for a '5.1 Editar Cancha' form. It features a title bar with the text '5.1 Editar Cancha'. Below the title bar is a large rectangular input field labeled 'Cancha'. At the bottom of the form, there are two buttons: 'cancelar' on the left and 'seleccionar' on the right.

Figura 8: Prototipo de la interfaz del formulario de selección de cancha



5.1.1 Editar Cancha

Nombre

Horario

Horario discontinuo

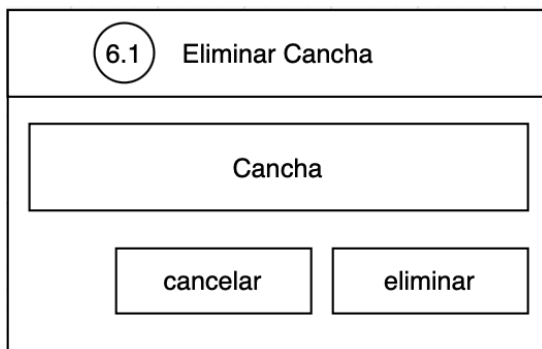
Precio por hora

Tipo de superficie

cancelar guardar

Detailed description: This is a wireframe for a '5.1.1 Editar Cancha' form. It features a title bar with the text '5.1.1 Editar Cancha'. Below the title bar are five stacked rectangular input fields, each with a label: 'Nombre', 'Horario', 'Horario discontinuo', 'Precio por hora', and 'Tipo de superficie'. At the bottom of the form, there are two buttons: 'cancelar' on the left and 'guardar' on the right.

Figura 9: Prototipo de la interfaz del formulario de edición de una cancha



6.1 Eliminar Cancha

Cancha

cancelar eliminar

Detailed description: This is a wireframe for a '6.1 Eliminar Cancha' form. It features a title bar with the text '6.1 Eliminar Cancha'. Below the title bar is a large rectangular input field labeled 'Cancha'. At the bottom of the form, there are two buttons: 'cancelar' on the left and 'eliminar' on the right.

Figura 10: Prototipo de la interfaz del formulario de eliminación de una cancha

Continuando con la Figura 4, el elemento 7 permite la creación de reservas por parte del club. Al presionar el elemento, se despliega el formulario de creación el cual contiene los campos analizados y definidos.

7.1 Añadir Reserva

Horario

Tipo

Cancha

Repetición

Nombre

Precio por hora

Precio de luz por hora

cancelar guardar

Figura 11: Prototipo de la interfaz del formulario de creación de una cancha

La Figura 11 retrata de forma básica la creación de una reserva por parte de los usuarios o clubes. A modo de ejemplo, si se vuelve a la figura 4, se puede notar que la reserva “Torres”, es una reserva en cuyo formulario se ingresa lo siguiente:

- Horario: de 8 hs a 9:30 hs
- Tipo: Alquiler
- Cancha: Cancha 4
- Nombre: Torres

Se obvian los campos Repetición, Precio por hora y Precio de luz por hora ya que no tienen una representación gráfica evidente en la figura.

Continuando con la figura 4, el elemento 8 es una representación de las reservas. Nótese que las reservas de distintos tipos tienen diferentes colores para que sea más fácil identificarlas. Presionando sobre el elemento 8, o cualquier reserva, dicho elemento es solo un ejemplo, se puede acceder a un menú con distintas acciones relativas a la reserva presionada.

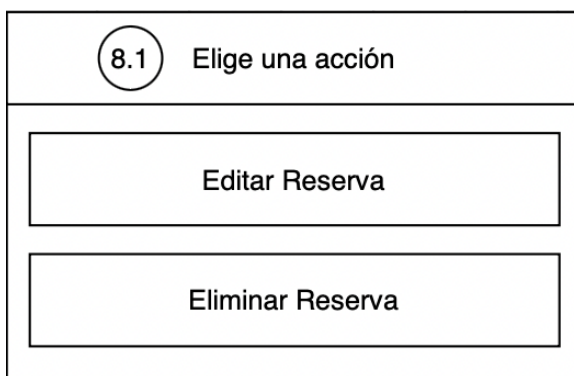
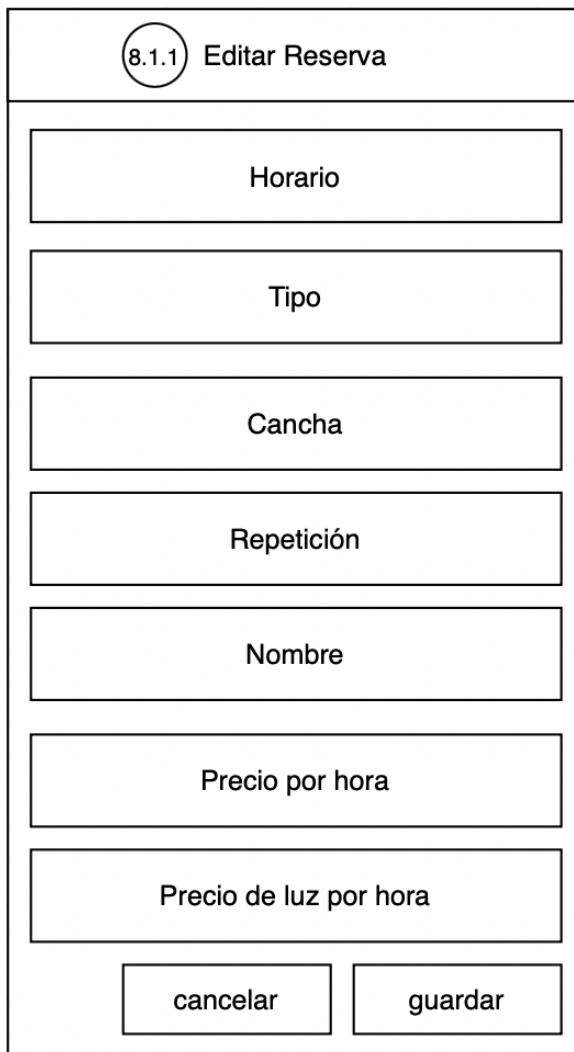


Figura 12: Prototipo de la interfaz de selección de una acción sobre una reserva



8.1.1 Editar Reserva

Horario

Tipo

Cancha

Repetición

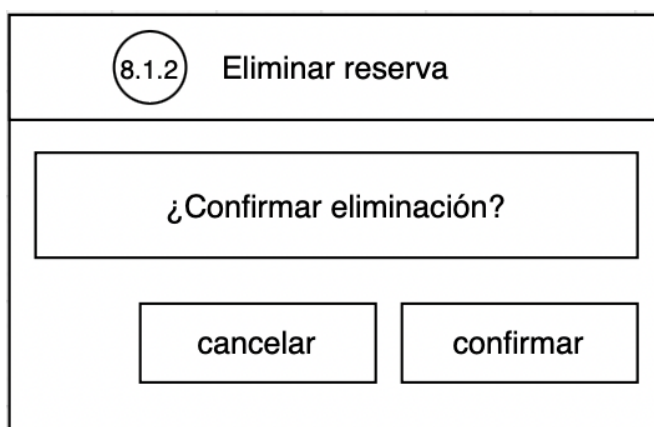
Nombre

Precio por hora

Precio de luz por hora

cancelar guardar

Figura 13: Prototipo de la interfaz del formulario de edición de reserva



8.1.2 Eliminar reserva

¿Confirmar eliminación?

cancelar confirmar

Figura 14: Prototipo de la interfaz del formulario de baja de reserva

### 3.2.2 Módulo Reservas

El Módulo de Reservas no debe confundirse con el Módulo de Administración de Reservas. Si bien las necesidades básicas de un club pueden estar cubiertas con un módulo que permite crear canchas y reservas, falta un punto muy importante, los usuarios de los clubes. Mientras que el Módulo de Administración de Reservas está enfocado en los complejos deportivos, el Módulo de Reservas se centra en los usuarios de los mismos.

Uno de los problemas más comunes a los que se enfrenta el club bajo estudio en el día a día es al hecho de que, si bien el registro se lleva a cabo en una hoja de Excel, las vías para tomar o atender reservas son varias:

- La aplicación de mensajería instantánea, Whatsapp.
- La red social Instagram.
- Llamadas telefónicas.

El solo hecho de tener tres distintas aproximaciones para atender el registro de reservas es un punto negativo. Si todo estuviese centralizado en un solo canal, se evitarían errores o malentendidos. No son pocas las situaciones en las que se han superpuesto reservas debido a que el registro de una se tomó por Whatsapp y otra por Teléfono y al no contar con una computadora en el momento, no se registró uno de los turnos en la planilla, produciendo como resultado que uno de los usuarios asista al club sin ninguna reserva registrada por culpa de una mala organización o administración.

El caso se torna aún más negativo cuando se habla del hecho de que las aplicaciones utilizadas no están específicamente diseñadas para registrar reservas o atender inquietudes de los usuarios. El personal encargado del club puede obviar un mensaje en el chat ya sea de Instagram o Whatsapp simplemente porque no lo vio. También ocurren casos en los que los usuarios envían reiterados mensajes a los clubes,

a modo de spam y de captar la atención del club, el cual no contesta ya sea porque se encuentra ocupado o porque no está en línea.

Otras situación que presenta dificultades a las que se enfrenta el club bajo estudio es la queja por parte de los usuarios cuando no reciben una confirmación. El actual flujo para reservas en el complejo deportivo es el siguiente:

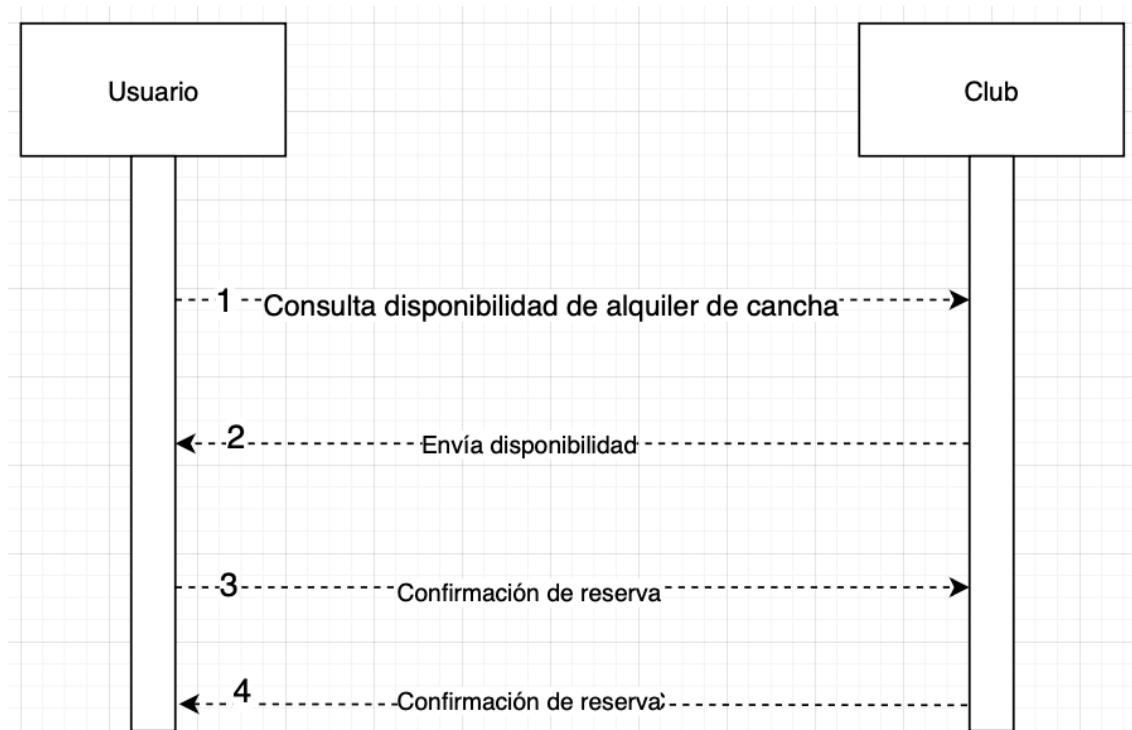


Figura 15. Diagrama de tiempo de reservas por parte de un usuario y un club. Imagen del autor.

En la figura 15, se pueden observar 4 pasos necesarios para concretar una reserva de una cancha deportiva. Se puede ver con claridad que hay dos confirmaciones, una por parte del club y otra por parte del usuario. El usuario confirma que quiere dicha reserva por la cual está consultando disponibilidad, mientras que el club confirma que registró la reserva que el usuario desea. Esto en sí no es un problema, pero si se torna una complicación como por ejemplo, en Whatsapp, el club al tener mucha demanda, comienza a perder noción de la organización de los chats, y lo más común es que termine olvidando el paso 4, por lo que el usuario no recibe una confirmación por parte



del club, llevando a dos escenarios, el primero, que el usuario se presente en el club y no haya un registro de la reserva y el segundo, que el usuario nunca acuda al club, cuando el mismo había registrado la reserva pero nunca se la confirmó.

Se puede diseñar el Módulo de Reservas, de modo que se acople con el Módulo de Administración de Reservas y que cubra estos problemas con un solución en donde se utilice una única vía o canal de registro de reservas. Anteriormente se definió el Módulo de Administración de Reservas como una planilla o tabla que el club modifica según sus necesidades y en la cual el club es capaz de agregar nuevas reservas. Pero a esto se le puede añadir una funcionalidad que resolvería la gran mayoría de problemas identificados, esto es, la capacidad del Módulo de Administración de Reservas de recibir reservas en línea provenientes del Módulo de Reservas. En otras palabras, un cliente de un club puede ver los horarios disponibles para la búsqueda de reserva que realice según sus filtros, como por ejemplo, deporte, hora y fecha. La disponibilidad se muestra según los horarios de canchas que definió el club y según los espacios o celdas vacías con los que cuenta. Cuando encuentra una disponibilidad que cubre sus necesidades, realiza la reserva. El club no tiene que confirmar nada, porque la reserva, eventualmente, llegará a su planilla y cubrirá el espacio según los filtros que definió el usuario. Las celdas ocupadas en la planilla del club no aparecerán en la búsqueda del usuario. De esta forma se elimina la interacción del usuario con el club, ya no hay confirmación necesaria porque todo se encuentra automatizado por el sistema. El diagrama de la figura 15 queda completamente simplificado a dos pasos, el primero la consulta de disponibilidad, y el segundo la concretación de la reserva.

Partiendo del análisis anterior, nuevamente se realiza el un listado de funcionalidades con las cuales el Módulo de Reservas debe contar para que un cliente

de un complejo deportivo logre la búsqueda y confirmación de una reserva. Un cliente de un club o complejo deportivo, a través del sistema, debe poder:

- Ver disponibilidad de reservas a través de una búsqueda.
- Filtrar la búsqueda de acuerdo a diversos parámetros como provincia, club, fecha y hora, etc.
- Concretar la reserva. En caso de que la búsqueda arroje resultados y el usuario esté conforme, el mismo debe poder confirmar o concretar dicha reserva.
- Ver las reservas que todavía no han pasado en el tiempo, es decir, visualizar las próximas reservas.
- Ver las reservas que ya pasaron en el tiempo, esto es, un historial.

Al igual que con el módulo anterior, como ya se tienen identificado bien los problemas y requisitos, se procede a realizar ciertos prototipos que ayuden a la planificación y al posterior desarrollo del software. Teniendo bien definidos los problemas, los requisitos y los prototipos, se puede realizar el diseño arquitectónico de la implementación. Es bueno recordar que los prototipos y requisitos no cubren todos los casos de uso, de hecho, sólo cubren lo justo y necesario para resolver el problema denotado y automatizar la mayor parte del proceso. De este modo se puede desarrollar un sistema que no sea altamente complejo y por lo tanto se logre su construcción en una cantidad de días razonable, para posteriormente poder ser utilizado, recopilar opiniones de usuarios, corregir problemas y comenzar a añadir funcionalidades de forma incremental.

① Tenis				
② Todos los Clubes				
③ Hoy a las 19:00 hs				
Imagen				④
Club Del Bono				
⑤	19:30	20:00	20:30	21:00 21:30
Cancha 1		Cancha 2 ⑥		
Imagen				
Aristides				
19:30	20:00	20:30		
Cancha A		Cancha B		Cancha C
⑦ Buscar			⑧ Reservas	

Figura 16. Prototipo de la interfaz principal del Módulo de Reservas. Imagen del autor.

En la figura 16 se puede observar un prototipo de la interfaz principal para clientes de clubes, es decir, personas que desean practicar un deporte y requieren del alquiler de una cancha, y por lo tanto de una reserva. Si se observa la parte superior de la figura 16 hay diversos filtros presentes. Lógicamente, al tratarse de un prototipo, esto no es un resultado final y es solo un boceto para comprender el problema, por lo que se pueden agregar innumerables filtros en la búsqueda de acuerdo a las necesidades de los usuarios. El globo con el número 1 hace referencia al filtro de deporte, el usuario en este caso se encuentra en la búsqueda de reserva de Tenis,

presionando, puede acceder a una lista de deportes con los que cuenta la aplicación y cambiarlo. En el globo número 2, no hay un club seleccionado, por lo que en este caso se muestran las reservas relativas a todos los clubes de Tenis con los que la aplicación cuenta en su base de datos. Presionando sobre el mismo, puede elegir entre un listado para filtrar por un único club en caso de que sea de preferencia del usuario. El globo número 3 simula la fecha elegida por el usuario, se puede mostrar un componente como el de la figura 6 para que el usuario seleccione primero una fecha, luego un horario y finalmente se aplique el filtro de búsqueda..

El elemento que contiene el globo número 4, es la representación de la interfaz para que el cliente del complejo deportivo pueda seleccionar una reserva de forma intuitiva. En este caso se busca una reserva para la fecha actual a las 19:00 hs, para una mejor experiencia, se muestra el horario elegido y otros horarios disponibles del club como en el caso del “Club Del Bono” el cual tiene disponibilidad a las 19:30, 20:00, 20:30, 21:00 y 21:30. El usuario puede presionar las distintas horas, representado por el globo 5, para ver las canchas con disponibilidad con el fin de ser reservadas. Por ejemplo, para el “Club del Bono” la hora seleccionada es la primera, las 19:30 y se muestra que hay disponibilidad en la “Cancha 1” y “Cancha 2”. Para el caso del club “Arístides” la hora seleccionada son las 20:30 y hay disponibilidad en “Cancha A”, “Cancha B” y “Cancha C”. Presionando sobre la cancha, como lo denota

el globo 5, permite abrir una nueva interfaz en donde se despliegan los detalles y se

6.1

Imagen

Club Del Bono

Deporte: Tenis

Cancha: Cancha 2

Superficie: Polvo de ladrillo

Fecha: 05/04/2023

Hora: 19:30

Duración: 60 min

Reservar

puede concretar la reserva.

Figura 17. Prototipo de la interfaz para concretar una reserva. Imagen del autor.

Como se nota en la figura 17, se muestran los detalles de la reserva presionada para un club. Al presionar sobre “Reservar” la reserva se confirma y es en ese momento cuando sucede la integración con el Módulo de Administración de Reservas, a modo que la reserva llegue a la planilla o tabla del club y rellene el espacio correspondiente, simplificando todo el proceso descrito.

Volviendo a la figura 16, la interfaz mostrada allí, es la correspondiente a la pestaña de “Buscar”, representado por el globo 7. Presionando la pestaña adyacente “Reservas”, el globo número 8, la interfaz cambia completamente. La figura siguiente representa la interfaz.

Proximas Reservas	Historial
⑨ Reserva confirmada el 05/04/2023	
Buscar	Reservas

Figura 18. Prototipo de la interfaz de reservas. Imagen del autor.

En la figura 18 se notan dos nuevas solapas cuando el usuario se encuentra sobre “Reservas”, estas son, “Próximias Reservas” e “Historial”. Ambas interfaces son iguales simplemente se utilizan a fin de organizar reservas suponiendo que el usuario

tiene un gran listado, en “Historial” aparecen reservas que ya pasaron y en “Próximas Reservas” las que están siendo concretadas en el momento o que están por suceder, es decir, que están pendientes.

Otra diferencia entre estas pestañas es que cuando una reserva cuya fecha todavía no ha pasado, es decir entra en el área de “Próximas Reservas”, entonces el usuario tiene la posibilidad de cancelar la misma. Presionando sobre ella, en el caso de la figura 18, presionando sobre el elemento con el globo número 9 se abre una nueva interfaz.

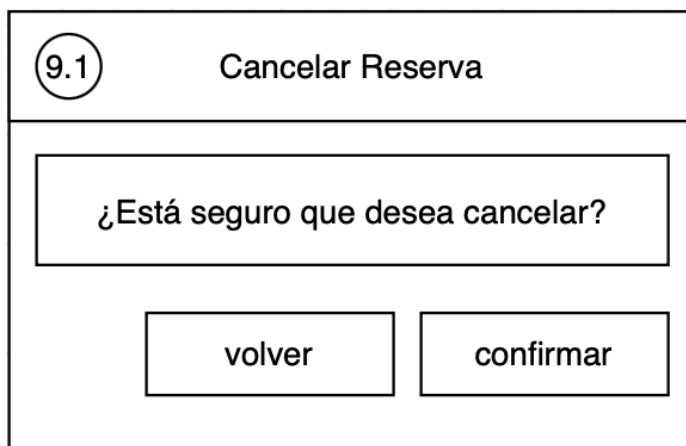


Figura 19. Prototipo de la interfaz de cancelación de una reserva. Imagen del autor.

### 3.2.3 Módulo de Autenticación

Para el caso del sistema definido en este proyecto, lo que se necesita es simplemente una forma de registrar clientes y de autenticar clientes y clubes. No es necesario tener una interfaz o diseñar el registro de clubes porque no es algo que sucede en gran escala, o al menos no por el momento, y además este es un proceso el cual es mejor que se realice manualmente por un humano para validar que efectivamente se trata de un club real, a modo de filtro, para que no cualquiera pueda registrar un complejo deportivo fácilmente en la aplicación y evitar posibles fraudes.

Para el club bajo estudio, se pueden tomar los datos necesarios del club y almacenarlos directamente en la base de datos, sin contar con una interfaz.

Para que un club sea registrado en la base de datos, teniendo en cuenta las funcionalidades de los dos módulos anteriores, los siguientes datos son fundamentales:

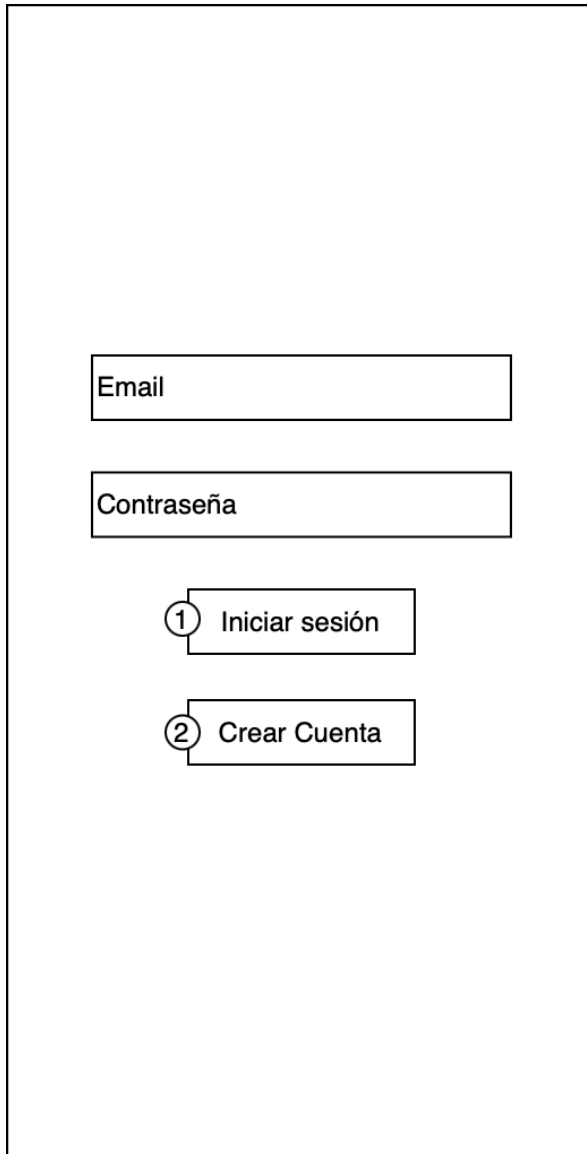
- Nombre del club.
- Provincia donde reside.
- Imágenes del establecimiento. Se utilizan como portadas con fines estéticos cuando los usuarios realizan búsquedas de reservas.
- Deportes con los que cuenta el establecimiento.
- Número de teléfono.

Para el caso de las personas interesadas en realizar reservas, si se debe contar con una interfaz desde un principio y se pueden realizar validaciones de email o teléfono. Los datos mínimos requeridos para complementar el Módulo de Reservas son:

- Nombre.
- Apellido.
- Teléfono.
- Email.

A continuación se muestran prototipos básicos con los que el sistema debe contar para capturar el registro de quienes desean realizar reservas y el inicio de sesión tanto de estas personas como de los clubes.



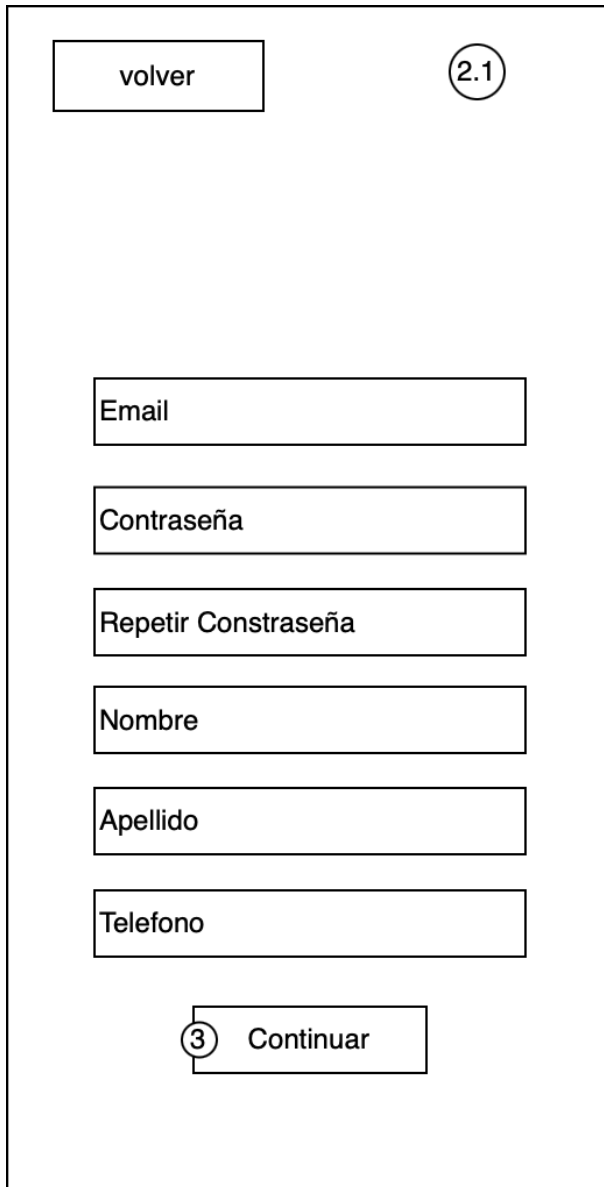


El prototipo de la interfaz de inicio de sesión se muestra dentro de un recuadro rectangular. En el interior, hay un campo de texto etiquetado como "Email" y otro etiquetado como "Contraseña". Debajo de estos campos, hay dos botones. El primer botón está etiquetado con un número "1" dentro de un círculo y el texto "Iniciar sesión". El segundo botón está etiquetado con un número "2" dentro de un círculo y el texto "Crear Cuenta".

Figura 20. Prototipo de la interfaz de Inicio de sesión. Imagen del autor.

En la figura 20 se muestra una interfaz para contemplar el inicio de sesión por parte de ambos usuarios, los clubes y las personas que desean practicar un deporte en un club. Este debe ser lo más sencillo posible, es por eso que únicamente ingresar un email y contraseña debe ser suficiente. Al presionar el elemento número 1, “Iniciar Sesión” la aplicación internamente, de acuerdo a la información asociada a ese usuario, debe ser capaz de reenviar al mismo ya sea al Módulo de Reservas o al Módulo de Administración de Reservas.

Cuando el usuario presiona sobre el globo número 2, la aplicación cambia de interfaz mostrando lo necesario para la creación de una cuenta de un usuario que desea practicar un deporte y concretar una reserva.



El prototipo de la interfaz del registro de usuarios se muestra dentro de un recuadro rectangular. En la parte superior izquierda hay un botón rectangular con el texto "volver". En la parte superior derecha hay un globo de anotación circular con el número "2.1". Debajo de estos elementos se encuentran seis campos de entrada de texto, cada uno con un label a la izquierda: "Email", "Contraseña", "Repetir Contraseña", "Nombre", "Apellido" y "Telefono". En la parte inferior del formulario hay un botón rectangular con el texto "Continuar" y un globo de anotación circular con el número "3" a su izquierda.

Figura 21. Prototipo de la interfaz del registro de usuarios. Imagen del autor.

Nuevamente, la figura 21 hace referencia a la creación de un usuario que quiere realizar una reserva. Como se dijo antes, la creación de un club es un proceso manual.

A continuación se muestra la interfaz para culminar con el proceso de registro, en donde los usuarios deben verificar el email que ingresaron previamente.

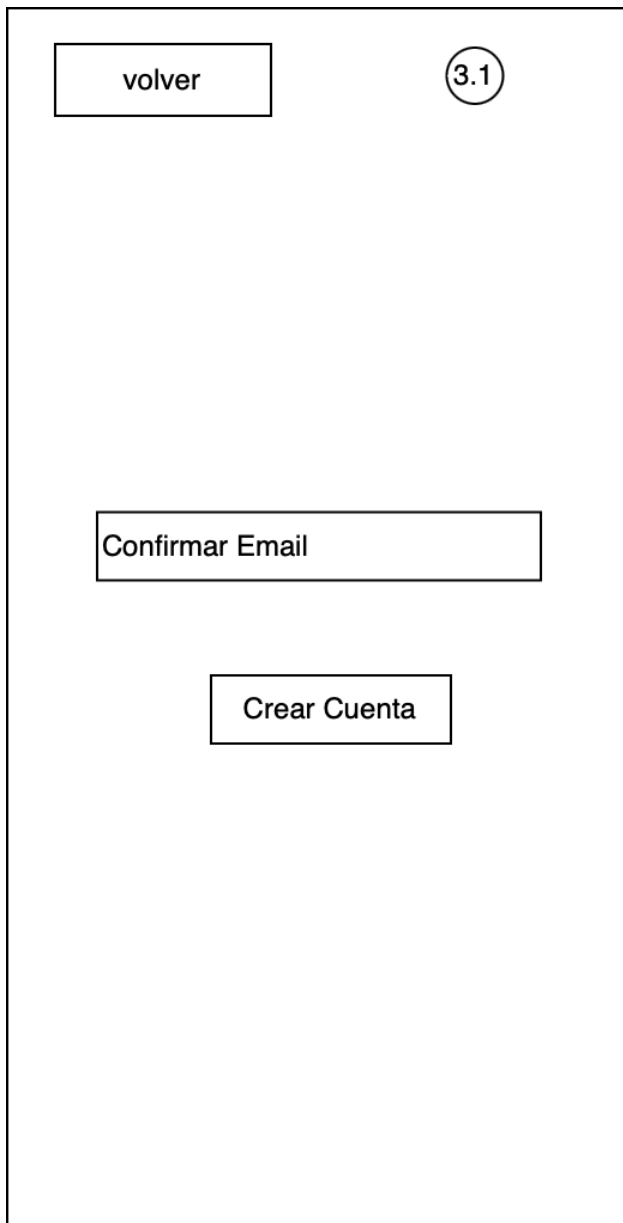


Figura 22. Prototipo de la interfaz de confirmación de email. Imagen del autor.

La confirmación del email, mostrada en la figura 22, se basa en un proceso interno en el cual el sistema envía un correo con un código al email ingresado en el paso anterior por el usuario. Así, el usuario debe ingresar dicho código en la interfaz para que el sistema lo valide y le permita continuar al Módulo de Reservas. La validación es necesaria para verificar que el correo ingresado es real.

### 3.3 Análisis y elección de tecnologías

Se ha observado y analizado las falencias y requisitos necesarios para solventar defectos en un complejo deportivo bajo estudio. Luego se ha procedido a diseñar prototipos muy básicos de las interfaces del sistema con el fin de reforzar los requerimientos y abordar mejor el problema. A continuación se procede a analizar, comparar y elegir la tecnología más adecuada para llevar a cabo el desarrollo del sistema de software.

Teniendo en cuenta los dos módulos principales de la aplicación, el Módulo de Administración de Reservas y el Módulo de Reservas, se analiza por separado los dispositivos utilizados por los usuarios objetivos de cada módulo.

**Módulo de Administración de Reservas:** Este módulo es manejado por personal del club o complejo deportivo. En el caso de estudio se observó el manejo de la administración de reservas tanto por el personal como por el dueño del mismo. Si bien la administración se hace desde un punto fijo, esto es, el mostrador de la cantina del club, es evidente que la mejor opción es tener una administración portable. Se presentan casos en los que los clientes solicitan la disponibilidad horaria de alguna cancha por medio de Whatsapp y el personal del club al no contar con el software en sus dispositivos móviles, tienen que hacer esperar al cliente hasta que el personal se encuentre en el establecimiento para chequear el horario. Esto sin contar que en muchos casos el personal se olvida, y nunca termina enviando el itinerario solicitado. También hay casos en donde el operario no se encuentra en el establecimiento, entonces procede a comunicarse con aquél que sí se encuentra para consultar por la disponibilidad horaria, haciendo de intermediario con el cliente y resultando en un proceso más largo y engorroso. Otro punto importante de por qué la portabilidad es realmente necesaria, es el hecho de que muchas veces en el

club se encuentra un único operario, el mismo realiza variadas tareas de mantenimiento de canchas o del establecimiento en general. También se encarga de llevar productos comestibles o bebidas a clientes que terminaron su turno y se encuentran descansando. Por lo tanto, es lógico tener un sistema que pueda ejecutarse en una computadora de escritorio o notebook, pero también que pueda ejecutarse en un dispositivo móvil para una comodidad óptima y un acceso al itinerario todas las horas del día.

**Módulo de Reservas:** Como se mencionó, este módulo es específico para los clientes del club. Como los mismos realizan reservas por distintos medios como Whatsapp, llamadas telefónicas entre otros, lo importante es canalizar toda la comunicación en un mismo lugar para resolver los problemas planteados. Si bien la mejor opción es contar con este módulo en cualquier dispositivo que se lo requiera, existen varias razones por las cuales tiene sentido centrarse únicamente en dispositivos móviles.

**Experiencia de usuario optimizada para dispositivos móviles:** Las aplicaciones móviles pueden ofrecer una experiencia de usuario más fluida y adaptada a las pantallas táctiles de los dispositivos móviles. Esto es importante para aplicaciones que implican interacciones rápidas o en movimiento, como realizar reservas.

**Acceso a características del dispositivo:** Las aplicaciones móviles pueden acceder a características específicas del dispositivo, como la ubicación GPS o las notificaciones push, que pueden mejorar la funcionalidad de la aplicación y proporcionar una experiencia más personalizada.

**Necesidades de aplicaciones en una computadora de escritorio:** Para el módulo anterior si tiene sentido desarrollar una versión web porque el club puede contar con una computadora de uso general, facilitando la administración del mismo y donde puede, en una pantalla o monitor más grande, ver más cómodamente la planilla. Pero en este caso, no hay necesidad, si se tiene en cuenta que casi cualquier persona que realiza una reserva

cuenta con un dispositivo móvil. Además no se requiere de alguna interfaz, como el caso de la planilla, donde la experiencia del usuario mejora con una pantalla más grande.

El siguiente gráfico ayuda a analizar y comprender que no es necesario el desarrollo de este módulo en plataformas que permitan la ejecución del sistema en una computadora ya sea de escritorio o no, debido a que la gran mayoría de individuos cuenta con un dispositivo móvil.

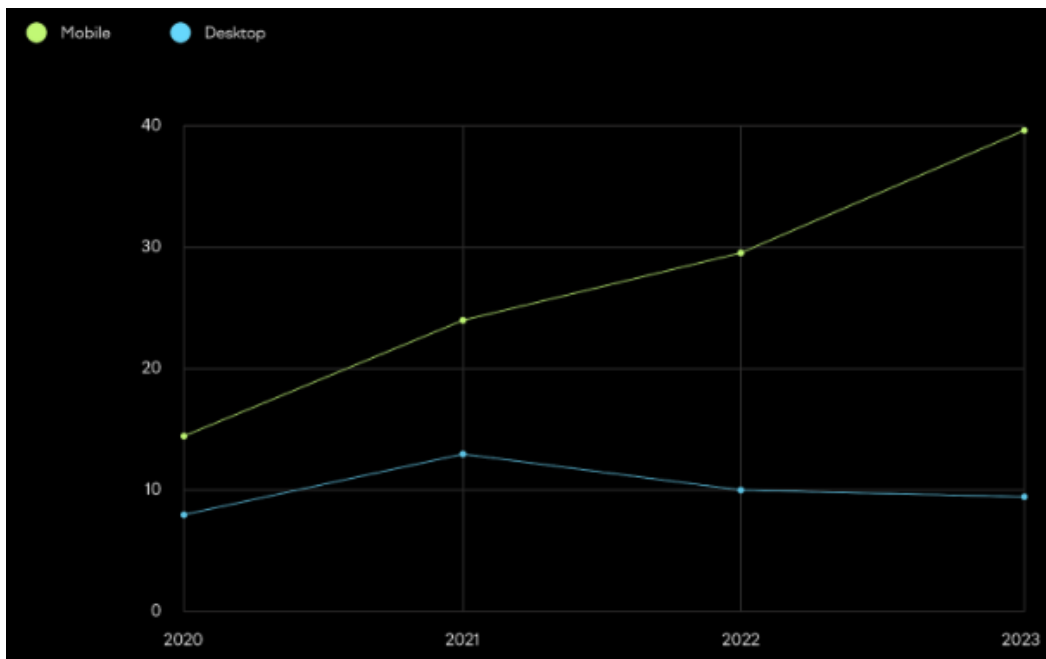


Figura 23. Uso Web histórico de Computadoras Vs. Móviles (Visitas totales en mil millones). Imagen de Semrush.

De acuerdo con Semrush, prestigioso sitio de análisis de datos, de la figura 23 se pueden obtener algunos datos importantes: En el 2020 las visitas desde dispositivos móviles fueron un 81% más altas que desde computadoras de escritorio. En el 2021 registró el mismo número. Para el 2022 hubo 203% más de visitas en dispositivos móviles en comparación con computadoras de escritorio. Y para el 2024 este número llegó a 313% más.

Si bien estos son datos del tráfico Web, sirven para evidenciar el creciente uso de dispositivos móviles. Teniendo en cuenta que los clientes del complejo deportivo bajo

estudio realizan las reservas por Whatsapp o por llamadas telefónicas, el desarrollo de este módulo en dicha plataforma es suficiente. De todas formas, en caso de que esta afirmación no se cumpla, al desarrollar la aplicación con un Framework Multiplataforma, la extensión del Módulo de Reservas desde dispositivos móviles hacia la Web no trae mayores complicaciones en caso de que sea necesario.

### **3.3.1 Frontend: Desarrollo de aplicaciones móviles nativas y multiplataforma**

Se parte desde la elección de la tecnología del Frontend, estos es el conjunto de tecnologías, lenguajes y herramientas que permiten el desarrollo de interfaces con las cuales los usuarios interactúan. Ya se analizó la necesidad de contar con una aplicación móvil y una aplicación que funcione también en ordenadores para el caso del Módulo de Administración de Reservas.

Por el lado de los entornos móviles, existen dos tecnologías de vanguardia para cada plataforma, Swift y Kotlin. Estas se distinguen sobre el resto, porque son lo que comúnmente se denominan “nativas” es decir son lenguajes creados específicamente para desarrollar aplicaciones en su respectiva plataforma. Pero existe otro tipo de tecnologías, las multiplataformas o también conocidas como híbridas. La siguiente lista enumera algunas de las más conocidas según Statista (2022):

- React Native
- Flutter
- Cordova
- Ionic
- Xamarin

Como su nombre indica, las aplicaciones nativas están ligadas a la plataforma para la que están diseñadas. Como tal, si se escribe aplicaciones para Android, solo funcionarán en ese sistema operativo. Y si eventualmente se decide ampliar y apuntar a dispositivos Apple, el equipo de desarrollo tiene que escribir un nuevo código desde cero. Si uno, por ejemplo, se encuentra seguro de que el público objetivo está dedicado a Apple, se puede reducir significativamente los gastos y el tiempo de comercialización con una aplicación nativa en Swift para Apple.

Por otro lado, el entorno multiplataforma permite escribir código de tal manera que la aplicación que se desarrolle se vea y sienta similar en dispositivos iOS y Android. Es una solución muy buena para audiencias generales o mercados donde ambos sistemas operativos están representados por igual.

Las tecnologías nativas se descartan desde un principio por el hecho de que se sabe que el público objetivo puede contar con dispositivos Android o iOS. Elegir tecnologías nativas significa la elección tanto de Swift para poder desarrollar en iOS como de Kotlin para poder desarrollar en Android. Lo que consecuentemente lleva a tener dos aplicaciones distintas, esto es aumentar los costos de desarrollo y por lo tanto, aumentar el tiempo de desarrollo. Tener dos aplicaciones a las cuales desarrollar nuevas funcionalidades y corregir errores es una desventaja muy grande para este tipo de proyectos donde se busca agilidad y portabilidad.

En cuanto a ventajas de las tecnologías nativas que destacan sobre el resto son dos: el rendimiento que éstas logran y que llevan las últimas novedades al día. El rendimiento no es en sí un problema, con los dispositivos móviles de hoy en día, al contar con un hardware tan potente, esa diferencia de rendimiento entre una aplicación



nativa y una híbrida es casi imperceptible. La siguiente es una prueba de performance profunda entre Flutter, React Native y Nativo, realizada por Inverita (2020).

Android	FPS	CPU %	Max Memory Mb	Battery mAh
Native (Android)	60	2.4	58	49.7 mAh
RN	58	11.7	139	79.01 mAh
Flutter	60	5.4	114	65.28 mAh

Figura 24. Prueba de benchmark en un dispositivo Android. Imagen de Inveritas.

iOS iPhone 6s	FPS	CPU %	GPU %	Memory Mb
Native (iOS)	60	12.72	21.24	154
RN	59	113.13	19.56	220
Flutter	60	33.3	10.75	159

Figura 25 Prueba de benchmark en un dispositivo iOS. Imagen de Inveritas.

La figura 24 corresponde a las pruebas en Android.

- Todas han mostrado aproximadamente la misma cantidad de FPS.
- Android Nativo utiliza la mitad de memoria en comparación con Flutter y React Native.
- React Native requiere el uso más significativo de la CPU. La razón es el uso de JS Bridge entre JS y código nativo que incita al desperdicio de recursos en serialización y deserialización.

- En cuanto al uso de la batería, Android Nativo tiene el mejor resultado. React Native está rezagado tanto detrás de Android como de Flutter. La ejecución de animaciones continuas consume más energía de la batería en React Native.

La figura 25 corresponde a las pruebas en iOS.

- FPS. Los resultados de React Native son peores que los de Flutter y Swift. La razón es la incapacidad de utilizar la compilación de IoT en iOS;
- Memoria. Flutter casi coincide con el nativo en el consumo de memoria pero sigue siendo más pesado en la CPU. React Native queda muy rezagado detrás de Flutter y del nativo en esta prueba.
- Diferencia entre Flutter y Swift. Flutter está utilizando activamente la CPU mientras que iOS Nativo está utilizando activamente la GPU. La reconciliación en Flutter aumenta la carga en la CPU.

Tener en cuenta que esta es una prueba de un caso de uso de una aplicación sencilla en donde se realiza la misma aplicación en cada lenguaje. Si se aumenta la complejidad, las animaciones y las tareas pesadas, la diferencia de rendimiento de las aplicaciones nativas será mayor. Sin embargo, este es un test ejecutado en dispositivos que al día de hoy se encuentran casi obsoletos, por ejemplo el iPhone 6S. Si tenemos en cuenta el avance del hardware en los últimos años y de las tecnologías híbridas, esta diferencia es todavía más insignificante.

Volviendo a las ventajas, la segunda mencionada está relacionada al hecho de que las tecnologías nativas poseen las últimas novedades de hardware de los fabricantes. Por ejemplo si un dispositivo Apple introduce un nuevo sensor, las tecnologías nativas van a ser las primeras en ser capaces de implementar desarrollo para dicho sensor, caso contrario a lo que sucede en las tecnologías multiplataformas,

dicha implementación tarda más en llegar, primero deben estar disponibles en las nativas para posteriormente poder implementarse en las híbridas. Pero este no es un punto de mucho interés en el sistema que se desarrolla.

Como se descarta el uso de lenguajes y herramientas nativas, la siguiente fase consiste en elegir la tecnología multiplataforma más adecuada al proyecto. En el ámbito del desarrollo de software, los Frameworks Multiplataforma han ganado popularidad como herramientas que permiten a los desarrolladores crear aplicaciones que funcionan en múltiples sistemas operativos y dispositivos. De hecho, se puede observar un crecimiento en el uso de algunos Frameworks Multiplataforma, sobre todo en Flutter. El siguiente gráfico nos muestra el crecimiento del uso de Frameworks Móviles Multiplataforma.

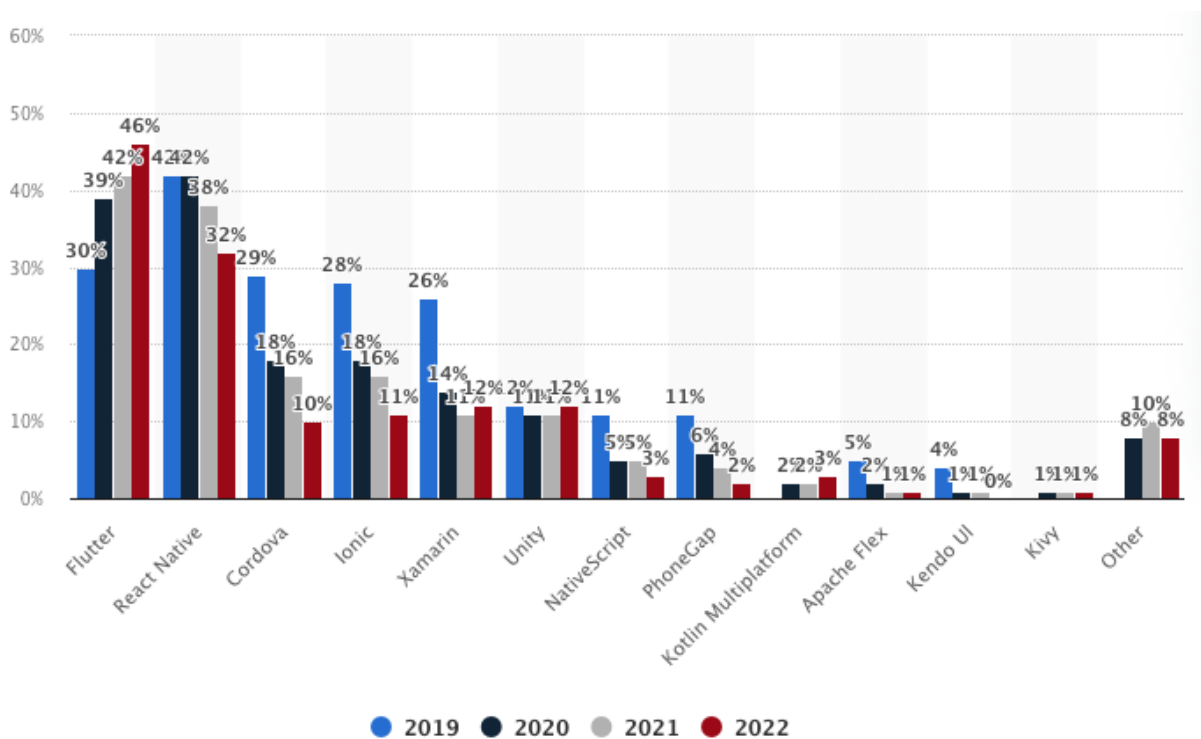


Figura 26. Gráfico de barras que muestra el uso de Frameworks Multiplataforma móviles en los años 2019, 2020, 2021 y 2022. Fuente: “Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022”, Statista.

Con casi 30.000 encuestados, Flutter es el marco móvil multiplataforma más popular utilizado por los desarrolladores globales, según esta encuesta realizada en 2022. El 46 por ciento de los desarrolladores de software utilizaron Flutter. En general, aproximadamente un tercio de los desarrolladores de dispositivos móviles utilizan tecnologías o marcos multiplataforma; el resto de desarrolladores de dispositivos móviles utilizan herramientas Nativas. También se puede observar un gran aumento de porcentaje de uso de Flutter con respecto a años anteriores, en comparación con React Native, el crecimiento de éste último fue mucho menor, de hecho, no hubo crecimiento, si no que perdió usuarios, que posiblemente Flutter haya absorbido.

Se vio que el rendimiento de una aplicación desarrollada con el marco multiplataforma React Native tiene un rendimiento peor a Flutter en términos generales. También se vio como Flutter ha ganado popularidad en los últimos años. Estas son las dos principales razones por las cuales se escoge Flutter como entorno de desarrollo Frontend en este proyecto. Se pueden tomar otras opciones como Cordova, Xamarin o Ionic, pero debido a su baja popularidad, encontrar documentación al respecto, o usar plugins que permitan acelerar el desarrollo es mucho más difícil que escogiendo uno más popular.

Con el fin de reducir tiempos y costos, el Framework Multiplataforma elegido en este trabajo para llevar a cabo el desarrollo de las interfaces multiplataformas del sistema es Flutter. Cuenta con otras ventajas además de ser multiplataforma, como por ejemplo, ser de código abierto. Esto permite que cualquiera que desee pueda ver cómo está desarrollado Flutter por dentro, es decir cualquiera puede ver su código si lo desea. Además se puede contribuir al mismo, si uno encuentra un error en Flutter, o quiere añadir una funcionalidad que éste no posee, entonces puede hacerlo sin ningún tipo de

coste alguno. Otro aspecto fundamental es que ya se encuentran aplicaciones desarrolladas con Flutter por grandes corporaciones que confían y utilizan dicha tecnologías en muchos de sus productos, como por ejemplo Alibaba o BMW (Flutter showcase 2021).

### **3.3.2 Backend: Arquitecturas y servicios**

Para llegar a una arquitectura robusta del sistema en desarrollo, se elige la arquitectura de microservicios. Para ello, se compara a través de diferencias, ventajas y desventajas entre las arquitecturas monolíticas, SOA y de microservicios.

Aunque los monolitos tienen una mala reputación, existen ventajas en mantener una única aplicación con un único código fuente. Hay principios de desarrollo fundamentales y bien conocidos que frecuentemente se usan en aplicaciones tradicionales que se vuelven deficientes cuando se utiliza un sistema distribuido y requieren un enfoque diferente.

- El flujo empresarial es visible: Con un único código fuente, se puede inspeccionar rápidamente el flujo de extremo a extremo y encontrar fácilmente cualquier característica que se desee, ya que toda la funcionalidad está en un solo repositorio.
- Sin sobrecarga de red y dependencias externas limitadas: Todos los diferentes módulos se llaman entre sí directamente dentro de la aplicación. No hay llamadas remotas a través de la red a través de APIs externas o brokers de eventos. Esta característica puede disfrutar de un impulso de rendimiento (aunque limitado por los límites de escalabilidad) ya que no hay sobrecarga de red. Tampoco se necesita lidiar con la versión de APIs o eventos y la compatibilidad hacia atrás. Si hay necesidad de realizar un

cambio radical, se puede lograr en una sola versión. En una arquitectura, por ejemplo, de microservicios, se necesita soportar temporalmente dos versiones de una API, para permitir que los consumidores de éstas se adapten al nuevo.

- Monitoreo y solución de problemas: El monitoreo de un monolito es sencillo ya que es solo una aplicación y los registros son triviales de obtener. Solucionar un problema también es más fácil ya que el alcance se reduce a una sola aplicación. Por el contrario en otro tipo de arquitecturas, hay varias instancias de docenas o cientos de microservicios. Esto requiere una estrategia completamente diferente para monitorear e identificar incidentes. Existe la necesidad de una infraestructura preexistente para gestionar todas las señales y métricas de esos microservicios.
- Estrategia de implementación más sencilla: Dado que solo hay una aplicación, el canal de implementación solo necesita tener en cuenta las necesidades de esa aplicación. Una arquitectura de microservicios tiene que admitir la implementación de varios servicios diferentes, posiblemente escritos en varios otros lenguajes y con varias dependencias diferentes. El sobrecosto de construir ese tipo de canal puede ser menor o mayor dependiendo de la diversidad del entorno. La topología de implementación suele ser mucho más simple y más económica con un monolito que con una arquitectura de microservicios.

Los monolitos pueden ser una solución adecuada en varios contextos, sin embargo, las empresas, organizaciones o entidades suelen enfrentar mayores dificultades al alcanzar una gran escala, tanto en datos, uso y desarrolladores. Cuando

se tira de uno de estos factores porque lo necesita o ha tenido éxito, los límites del monolito tiran en la dirección opuesta. Como un nudo, cuanto más se tira, más aprieta.

Con el tiempo suficiente, puede detener por completo el negocio.

- Acoplamiento y Falta de Límites: El principal problema con los monolitos es que, con el tiempo, se vuelven insoportablemente complejos y acoplados. Dado que toda la funcionalidad está dentro de una sola aplicación, es relativamente fácil comprometer los límites de cada dominio. Con el tiempo, se desvanecen y se entrelazan, siendo difíciles de leer. Un solo cambio puede afectar varias partes del sistema y tener impactos impredecibles, por ejemplo, cambiar la lógica de suscripción afecta el inicio de sesión, algo que en realidad, no debería tener ninguna relación.
- Autonomía del Equipo: A medida que el equipo de desarrollo crece, se vuelve cada vez más difícil trabajar en una sola aplicación. Incluso con módulos claramente definidos, el desarrollo aún requiere comunicación y alineación, y la implementación de la aplicación necesita coordinación entre los diferentes equipos. El sobrecosto de la comunicación, la garantía de calidad y la coordinación aumenta con el número de desarrolladores que trabajan en la aplicación. Con frecuencia se habla de escalar los recursos de la aplicación, pero las aplicaciones monolíticas a menudo limitan la escalabilidad del equipo.
- Ciclo de Lanzamiento: El ciclo de lanzamiento de una única aplicación monolítica suele ser más largo que el de un microservicio. Validar toda la aplicación a la vez requiere una suite de pruebas gigantesca que puede

llevar mucho tiempo ejecutar (suponiendo que no haya validación manual).

- Escalabilidad: La mayoría de las veces, los monolitos también alimentan una base de datos monolítica, que es muy difícil de escalar. Cuando los datos comienzan a tener un volumen muy alto, se vuelve problemático, especialmente con consultas que requieren muchos joins. La escalabilidad vertical (aumentar el número de recursos, por ejemplo, memoria, CPU, etc.) siempre es una opción tanto para la base de datos como para la aplicación, pero se vuelve costosa muy rápido. Si un negocio crece a un nivel que necesita distribución geográfica para proporcionar baja latencia a cada ubicación, una base de datos monolítica podría limitar su capacidad para lograrlo. Un enfoque de microservicios brilla en este aspecto particular, ya que cada microservicio posee su propia base de datos, allanando el camino para una distribución más escalable de los datos. Los microservicios son muy fáciles de escalar, ya que manejan eventos de una cola y se benefician en gran medida del desacoplamiento asociado. Cuando se escala una aplicación monolítica, se escala toda la aplicación. La mayoría de las veces, solo un módulo o parte de los módulos necesita escalar. Pero como todos los módulos están vinculados entre sí, la única opción es escalar todo. Con el enfoque de microservicios, es posible escalar solo la parte de la arquitectura que necesita escalabilidad y dejar las partes restantes con recursos mínimos, optimizando costos y recursos.
- Confiabilidad: El riesgo de implementar un monolito es que cualquier cambio puede hacer que toda la aplicación falle. Incluso un cambio en una parte menos crítica de la aplicación puede producir un problema (como



una fuga de memoria o un uso inesperado de la CPU) que puede afectar a toda la aplicación.

SOA típicamente intenta construir funcionalidades empresariales utilizando componentes reutilizables que se comunican a través de un medio desacoplado como una red o un bus de servicios. La mentalidad detrás de los servicios de aplicación es hacer que la funcionalidad sea reutilizable, y esta es una preocupación de diseño central en este tipo de arquitectura, en contraposición a los microservicios donde compartir funcionalidad está limitado (incluso evitado). Mientras que SOA se centra en funcionalidades abstractas y reutilizables, los microservicios se centran en organizar sus componentes en torno a dominios. Además, en SOA el bus de servicios tiende a volverse cada vez más grande. La lógica empresarial tiende a agregarse al bus en lugar de a los servicios. El bus también tiene varias responsabilidades además de la orquestación. El bus suele ser el componente que enruta y sabe cómo comunicarse con cada servicio y a menudo tiene lógica para adaptar y transformar las solicitudes. Tiende a crecer y convertirse en un monolito por sí solo, aunque toda la arquitectura está distribuida, muchas veces tiene lo peor de ambos mundos.

Una arquitectura de microservicios comprende varios servicios con propósitos específicos que interactúan entre sí para llevar a cabo un proceso o flujo de trabajo dado. Los servicios están agrupados según un contexto delimitado y en el ámbito de un dominio determinado. Están desacoplados e interactúan entre sí utilizando un intermediario de mensajes o solicitudes HTTP. Cada servicio es responsable de una sola acción y está modelado en torno a un concepto de dominio. También es importante tener en cuenta que cada servicio tiene una base de datos en lugar de un almacenamiento compartido y exponen interfaces para que otros servicios interactúen con ellos y exponiendo solo los datos relevantes. A diferencia de SOA, donde la

reutilización es una prioridad significativa, los microservicios evitan el compartir. En cambio, se centran en ese dominio y contexto delimitado. Este tipo de arquitectura está altamente desacoplado ya que hay un límite físico entre los componentes. Tienen que comunicarse a través de la red; esto impone límites naturales a cada servicio. La organización de contextos delimitados específicos en el ámbito de un dominio dado permite que el dominio cambie sin afectar a otros componentes no relacionados.

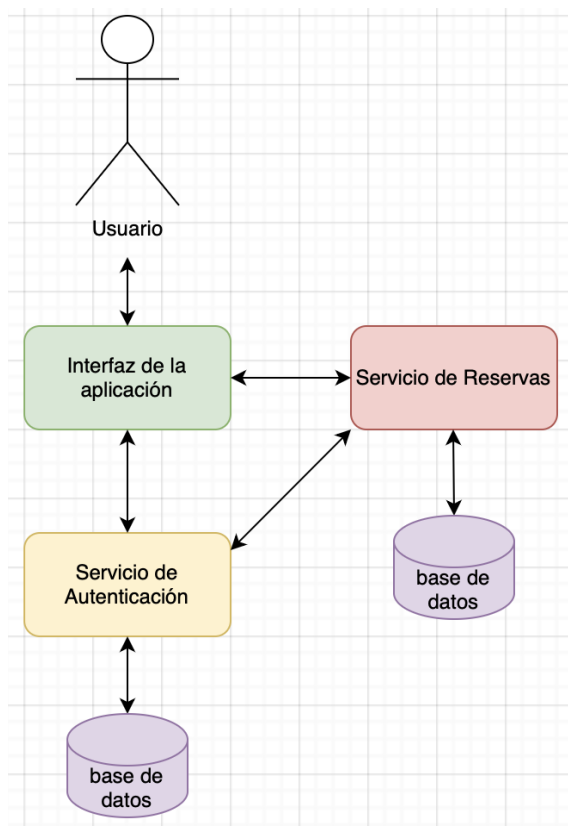


Figura 27. Arquitectura de microservicios del sistema. Imagen del autor.

En la figura 27 se observa el diseño de la arquitectura de microservicios para el presente sistema. Se puede ver que existen dos microservicios los cuales se comunican directamente con la interfaz de la aplicación o Frontend. El flujo de los datos es el siguiente:

- Un usuario inicia sesión, lo que genera una llamada al servicio de autenticación que verifica la existencia del usuario y genera un token para ese usuario.
- Una vez que la sesión está iniciada, el usuario procede a buscar una reserva. Esto desencadena una llamada al servicio de reservas.
- El servicio de reservas verifica el token asociado a la solicitud del usuario comunicándose con el servicio de autenticación.
- Cuando el servicio de reservas obtiene una respuesta exitosa procede a buscar las reservas y devolver una respuesta a la interfaz de la aplicación.

Este tipo de esquema es lo suficientemente simple como para aplicar la arquitectura desde un principio sin ningún tipo de complicaciones. Al contar con más microservicios, surgen más problemas de despliegue y de complejidad, ya que cada servicio implica un nuevo repositorio de código. El hecho de usar una arquitectura de microservicios en este proyecto quizá no es la opción más simple, pero indudablemente deja las puertas abiertas para la escalabilidad en caso de tener éxito y que los usuarios comiencen a crecer exponencialmente. También es plausible decir que agregar nuevas funcionalidades en el futuro será más fácil sin afectar el funcionamiento de otros componentes. Por ejemplo si en un futuro se quiere agregar un servicio de pagos, para señalar reservas antes de concretarlas, se puede crear un servicio independiente específicamente con ese objetivo y que se comunique con el servicio de reservas.

Existe un gran detalle a tener en cuenta, y es que el servicio de autenticación es Firebase. Es decir se utiliza un BasS (Backend as Service o Backend como servicio en español) para la implementación de la autenticación, de este modo no se desarrolla nada relacionado con la autenticación, simplemente se implementa el servicio que

Firestore ofrece. Esto reduce los tiempos y costos de desarrollo enormemente porque se evitan muchos puntos repetitivos que la mayoría de aplicaciones requiere. El desarrollador no necesita preocuparse por la selección del lenguaje de programación que debe dominar para desarrollar el servicio, tampoco por la elección de la base de datos, y su diseño, el despliegue o el mantenimiento de ambos componentes.

La elección de Firestore por sobre otras alternativas viene estrictamente ligada a la decisión de Flutter como Framework multiplataforma. Ambos son desarrollados y promocionados por Google, lo que permite que la integración sea posible en tan solo unos minutos y tengan compatibilidad asegurada. Con compatibilidad se hace referencia a la integración de un lenguaje con otro y a la documentación para poder integrar Firestore a Flutter. Además un punto importante de Firestore por sobre otras alternativas es su forma de pago, se puede contar con hasta 50.000 usuarios activos por mes con el plan gratuito, haciendo posible no afrontar ningún costo para este proyecto.

Por el lado del servicio de reservas se elige Golang también conocido como GO, lenguaje de programación que cuenta con cierta popularidad por sus ventajas en la computación en la nube ya que mantiene un aspecto de lenguaje de alto nivel, pero con características de un lenguaje de bajo nivel, como por ejemplo el uso de punteros o el manejo de la concurrencia de una manera muy sencilla. Golang es también desarrollado por Google y de código abierto como Flutter, lo que permite nuevamente una fácil integración con Firestore y con Flutter respectivamente.

### 3.3.3 Bases de Datos: Tipos y modelado

A la hora de tomar una decisión sobre cuál lenguaje de bases de datos utilizar es necesario primero destacar los distintos tipos o modelos de bases de datos. Según "Database Systems: The Complete Book" estos son algunos de los tipos de bases de datos utilizados:

- Bases de datos relacionales: Este tipo de base de datos organiza los datos en tablas con filas y columnas. Utiliza relaciones entre las tablas para representar la información y soporta consultas complejas a través del lenguaje SQL.
- Bases de datos NoSQL: Este tipo de base de datos se caracteriza por su capacidad para manejar grandes volúmenes de datos no estructurados o semiestructurados. NoSQL se refiere a "Not Only SQL" y puede incluir bases de datos de documentos, bases de datos de grafos, bases de datos clave-valor, entre otros.
- Bases de datos de grafos: Estas bases de datos están diseñadas para almacenar y consultar datos que tienen relaciones complejas entre ellos. Utilizan estructuras de grafo para representar y almacenar los datos y son especialmente útiles en aplicaciones como redes sociales y análisis de datos y redes.
- Bases de datos distribuidas: Estas bases de datos distribuyen los datos a través de múltiples servidores o nodos, lo que permite un mejor rendimiento y escalabilidad. Son útiles en entornos donde se necesita manejar grandes volúmenes de datos o se requiere alta disponibilidad.

Se comienza descartando las bases de datos de grafos porque evidentemente el sistema no tiene como foco principal el análisis de datos ni tampoco se trata de una red social. Las bases de datos de grafos se caracterizan también por tener mayor relevancia en la relación que la propia clase u objeto, y este no es el caso.

Las bases de datos distribuidas también se descartan debido a su complejidad. Por otro lado, manejar inmensos volúmenes de datos no es materia de importancia en este desarrollo.

Las bases de datos NoSQL, también conocidas como bases de datos no relacionales, son especialmente útiles en situaciones donde las aplicaciones realizan un alto volumen de operaciones de lectura en comparación con las operaciones de escritura. Esto se debe a su notable eficiencia en estas circunstancias. A diferencia de las bases de datos relacionales, las NoSQL no requieren de la operación JOIN, que es característica de las bases de datos relacionales y puede consumir significativos recursos computacionales. Esta ausencia de JOIN en las operaciones de lectura las hace considerablemente más eficientes.

En el software en desarrollo no predominan las lecturas de manera considerable con respecto a las escrituras. La creación, eliminación y actualización de canchas y reservas es una operación común en el día a día de las entidades deportivas por lo que se desestima esta última ventaja. Se elige, entonces, las bases de datos relacionales por descarte y por los siguientes factores imprescindibles

1. **Integridad de los datos:** Las bases de datos relacionales están diseñadas para mantener la integridad de los datos mediante el cumplimiento de restricciones de clave primaria, clave foránea y otras reglas definidas. Esto garantiza que los datos

sean precisos y coherentes, lo que es esencial cuando las operaciones de escritura son frecuentes.

2. **Transacciones ACID:** Las bases de datos relacionales ofrecen garantías de transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), esto asegura que las operaciones de escritura sean confiables incluso en entornos concurrentes o en caso de fallos del sistema, es crucial para mantener la coherencia de los datos en aplicaciones donde las escrituras son frecuentes.
3. **Consultas complejas:** Las bases de datos relacionales ofrecen un lenguaje de consulta estructurado (SQL) que es potente y versátil para realizar consultas y generar informes.

Una vez definido el tipo de bases de datos a utilizar, se realiza el modelado de la base de datos. El mismo consiste en un diagrama que muestra los componentes que almacenan datos, los tipos de datos y cómo se relacionan estos componentes entre ellos. Se parte desde el análisis hecho en el caso de estudio para así poder definir las estructuras y relaciones teniendo en cuenta las necesidades del complejo deportivo bajo estudio. Es un primer diseño del mismo por lo que está sujeto a modificaciones en un futuro en caso de haber errores, mejoras o nuevos requerimientos.

`clubs`	
<b>PK</b>	<u>`uid` varchar(50) NOT NULL</u>
	`name` varchar(25) NOT NULL
	`address` varchar(50) NOT NULL
	`image` varchar(300) DEFAULT NULL
	`phone` varchar(20) NOT NULL
	`whatsapp` varchar(20) NOT NULL

<b>`accounts`</b>	
<b>PK</b>	<b><u>`uid` varchar(50) NOT NULL</u></b>
	<b>`name` varchar(25) NOT NULL</b>
	<b>`surname` varchar(25) NOT NULL</b>
	<b>`date_of_birth` datetime DEFAULT NULL</b>
	<b>`gender` varchar(15) DEFAULT NULL</b>
	<b>`phone` varchar(16) NOT NULL</b>
	<b>`email` varchar(50) NOT NULL</b>
	<b>`image` varchar(300) DEFAULT NULL</b>

Figura 28. Modelado de las entidades “accounts” y “clubs”. Donde “accounts” son los clientes de clubes, quienes realizan reservas y “clubs” los clubes mismos quienes aceptan, rechazan o generan reservas. Imagen del autor



<b>`reserves`</b>	
<b>PK</b>	<b><u>`id` int NOT NULL</u></b>
FK	`club_uid` varchar(50) NOT NULL `type` varchar(15) NOT NULL `date_time_of_reserve` datetime DEFAULT NULL `state` varchar(15) NOT NULL
FK	`sport_id` int NOT NULL `field_id` int NOT NULL `name` varchar(25) DEFAULT NULL
FK	`account_uid` varchar(50) DEFAULT NULL `replicate` varchar(10) DEFAULT NULL `price` float DEFAULT NULL `light_price` float DEFAULT NULL `start_time` datetime NOT NULL DEFAULT '2023-01-01 00:00:00' `end_time` datetime NOT NULL DEFAULT '2023-01-01 00:00:00' `replicate_id` int DEFAULT NULL
<b>`fields`</b>	
<b>PK</b>	<b><u>`id` int NOT NULL</u></b>
FK	`name` varchar(30) NOT NULL `club_uid` varchar(50) NOT NULL `sport_id` int NOT NULL `price` float DEFAULT NULL `start_schedule_time` datetime NOT NULL DEFAULT '2023-01-01 00:00:00' `end_schedule_time` datetime NOT NULL DEFAULT '2023-01-01 00:00:00' `start_break_time` datetime DEFAULT NULL `end_break_time` datetime DEFAULT NULL
FK	`surface_id` int NOT NULL

Figura 29. Modelado de las entidades “reserves” y “fields”. Donde “reserves” son las reservas que realizan los clubes o clientes y “fields” son las canchas que ponen a disposición los clubes. Imagen del autor.

<b>`surfaces`</b>		<b>`sports`</b>	
<b>PK</b>	<u>`id` int NOT NULL</u>	<b>PK</b>	<u>`id` int NOT NULL</u>
	<u>`name` varchar(30) NOT NULL</u>		<u>`name` varchar(25) NOT NULL</u>
<b>`surfaces_sports`</b>		<b>`clubs_sports`</b>	
<b>PK FK</b>	<u>`surface_id` int NOT NULL</u>	<b>PK FK</b>	<u>`club_uid` varchar(50) NOT NULL</u>
<b>PK FK</b>	<u>`sport_id` int NOT NULL</u>	<b>PK FK</b>	<u>`sport_id` int NOT NULL</u>

Figura 30. Modelado de las entidades “surfaces”, “sports”, “surfaces\_sports” y “clubs\_sports”. Dónde “sports” son los deportes que abarca el sistema, como por ejemplo Tenis, Fútbol, Pádel. “surfaces” son los tipos de superficies que pueden tener los distintos deportes, es por eso que existe la entidad “surfaces\_sports”. Y “clubs\_sports” representa los distintos deportes que pueden detener distintos clubes. Imagen del autor.

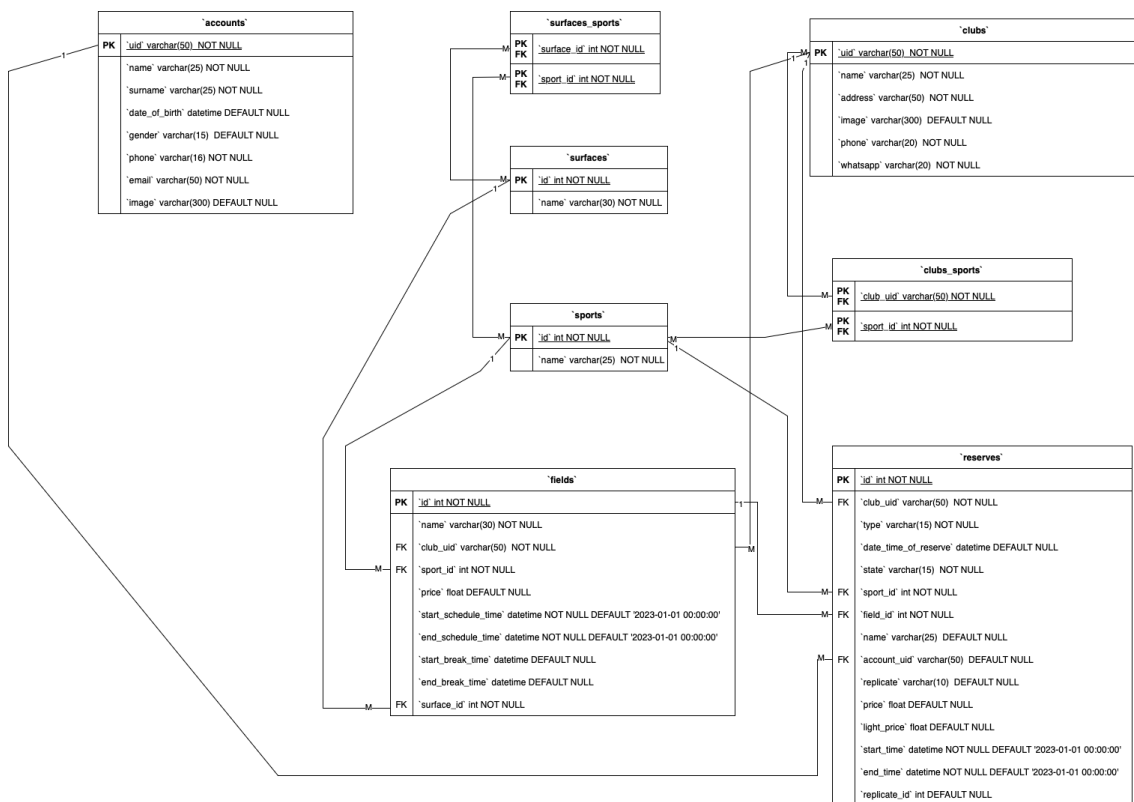


Figura 31 Modelo completo de la base de datos. Imagen del autor.

El modelado de la base de datos se basa teniendo en cuenta el problema planteado en este trabajo, los requerimientos y prototipos definidos. Se puede expandir de

innumerables formas y de hecho puede que haya que hacer correcciones durante el desarrollo del sistema. Según Garcia-Molina, Ullman y Widom (2009), el modelado de

bases de datos proporciona una representación clara y estructurada de cómo se organizan los datos dentro del sistema. Proporcionan:

**Claridad en la estructura de los datos:** El modelado de bases de datos proporciona una representación clara y estructurada de cómo se organizan los datos dentro del sistema, facilitando su comprensión y gestión.

**Facilita la comunicación:** Al tener un modelo visual de la base de datos, los desarrolladores, diseñadores y partes interesadas pueden comunicarse de manera más efectiva sobre los requisitos y la estructura del sistema.

**Identificación de requisitos y restricciones:** El proceso de modelado de bases de datos permite a los equipos identificar y documentar los requisitos del sistema, así como las restricciones que deben cumplirse, lo que ayuda a evitar malentendidos y errores durante el desarrollo.

**Reducción de redundancias y anomalías:** Al analizar y diseñar la base de datos de manera cuidadosa, es posible identificar y eliminar redundancias y anomalías en los datos, lo que conduce a un sistema más eficiente y confiable.

**Soporte para cambios futuros:** Un modelo de base de datos bien diseñado proporciona una base sólida que puede adaptarse y escalar para satisfacer las necesidades cambiantes del negocio y los requisitos del sistema a lo largo del tiempo.

**Mejora el mantenimiento del sistema:** Al tener una comprensión clara de la estructura de la base de datos, el mantenimiento y la modificación del sistema se vuelven más fáciles y menos propensos a errores.

### **3.3.4 Servidores en la nube: Infraestructura y contenedores**

Anteriormente se definió el uso de una arquitectura de microservicios. Uno de los servicios, el de autenticación, no es materia de preocupación en cuanto a el hardware y software necesario para ejecutarlo porque se encuentra en manos de un proveedor, Firebase, quien se encarga de su desarrollo, despliegue y ejecución por lo que solo hay que preocuparse sobre cómo utilizar dicho servicio. En contrapartida el servicio dedicado al manejo de las reservas es un servicio propio, por lo que precisa de una infraestructura de hardware y software para que se ejecute continuamente.

Existen dos opciones bien definidas para elegir una infraestructura en la cual se pueda ejecutar un software, servidores en la nube y la infraestructura on-premise. La diferencia entre on-premise y el software en la nube radica en la ubicación. El software on-premise se instala y se ejecuta en la infraestructura de hardware propia de una empresa, y se hospeda localmente, mientras que el software en la nube se almacena y administra en los servidores del proveedor y se accede a través de un navegador web u otra interfaz.

En este proyecto se adopta el software en la nube por ahorro de tiempo y costos. Uno de los mayores problemas de la infraestructura on-premise es que requiere hardware para ser instalado, esto lleva a la necesidad de tener conocimientos técnicos fuera del área del software y donde absorber dichos conocimientos puede llevar mucho tiempo. También se debe realizar mantenimiento tanto del hardware utilizado como del software. En cuanto a costos, no solo las piezas de hardware son caras tanto de adquirir como de mantener, si no también las licencias de software no son particularmente económicas. Todos estos problemas de conocimientos y costos son

resueltos por los proveedores de servidores en la nube debido a que ellos mismos son los que se encargan de resolver estos problemas a cambio de una suscripción pagada mensual o anualmente dependiendo del proveedor y su plan de pago. Generalmente es muy asequible porque se puede comenzar contratando un servidor en la nube de muy bajos recursos para realizar pruebas y tener un sistema funcionando y conforme se lo requiera, se pueden ir escalando recursos en tan solo unos minutos. Existen muchos proveedores que incluso ofrecen planes completamente gratuitos hasta que se alcanza cierto grado de complejidad en donde se comienza a abonar.

Una vez elegida el tipo de infraestructura es necesario resolver un segundo problema, y este es, cómo ejecutar el servicio de reservas en dicha infraestructura puesta a disposición por un proveedor. Esa lógica de negocio, que refiere a todo el procesamiento y almacenamiento de los datos debe ser alojado, por lo tanto, se necesita ejecutar la aplicación en sí que contiene el código fuente del servicio y la propia base de datos de la cual depende. Son dos componentes que pueden ser alojados y ejecutados de acuerdo a dos alternativas. La primera es almacenar ambos en un mismo servidor en la nube y la segunda es almacenar cada uno en un servidor en la nube por separado. Esto se resuelve relativamente fácil con el uso de contenedores. Según Poulton, N. (2018) “Un contenedor es una entidad ligera, portátil y autocontenida que encapsula todo lo necesario para ejecutar una pieza de software, incluidos el código, las bibliotecas, las herramientas del sistema y las configuraciones.” En pocas palabras, encapsulando el servicio de reservas en un contenedor y la base de datos en otro, podemos ejecutar de manera muy sencilla ambos componentes de forma portable, tanto en un ordenador local, como en un servidor en la nube, o en dos servidores o donde uno lo requiera, la principal ventaja de los contenedores, es esa, la portabilidad que ofrecen.

DonWeb se elige como proveedor de servidores en la nube y hay dos razones de la elección por sobre otros con mayor popularidad como Google Cloud o AWS. La primera razón es que DonWeb permite pagos en moneda local, por lo que puede ser significativamente más económico que otros proveedores. La segunda razón y no menos importante es que DonWeb es una de las empresas de Web hosting y dominio más grande de latinoamérica, y sus orígenes se encuentran en Rosario, Argentina, por lo que la empresa cuenta con servidores en dicho país a diferencia del resto. Tener un servidor en la nube que se encuentre en la Argentina puede llevar a tiempos de respuesta mucho más eficientes que otros proveedores como [Google](#) y [AWS](#) donde los servidores más cercanos se encuentran en Brasil. Aunque los tiempos de respuesta o latencia dependen de varios factores, uno de los principales es la localización geográfica de los servidores. Los tiempos de carga del sistema pueden disminuir considerablemente gracias a este punto a favor de DonWeb. El sitio [meter.net](#) permite fácilmente correr pruebas de latencia en pocos segundos. De un servidor que se encuentra en Buenos Aires se puede obtener, por ejemplo, aproximadamente entre 10 ms a 30 ms de latencia, mientras que de uno radicado en Brasil puede llegar a ser entre 50 ms a 100 ms. Como se mencionó, esto depende de muchos factores como por ejemplo el tráfico y congestión de la red o la capacidad de procesamiento del servidor que da la respuesta, pero es un muy buen indicio para ver las diferencias de latencias que hay geográficamente.

23 ms	AR: Buenos Aires	33 ms	CL: Vina del Mar	103 ms	EC: Quito	179 ms	MX: Mexico City
123 ms	US: Miami	124 ms	US: Miami	178 ms	MX: Queretaro	159 ms	US: Farmville
152 ms	US: Dallas	159 ms	US: Ashburn	155 ms	US: Washington	206 ms	US: Los Angeles
162 ms	US: New York	155 ms	US: New York	387 ms	ZA: Johannesburg	174 ms	CA: Toronto
170 ms	CA: Toronto	161 ms	US: Chicago	176 ms	US: Chicago	159 ms	CA: Montreal
177 ms	CA: Montreal	186 ms	US: Los Angeles	255 ms	ES: Seville	253 ms	ES: Madrid
250 ms	ES: Madrid	205 ms	US: Seattle	220 ms	CA: Vancouver	237 ms	IE: Ireland
232 ms	FR: Paris	229 ms	FR: Paris	286 ms	IT: Palermo	270 ms	IT: Palermo
239 ms	FR: Paris	256 ms	IM: Ballasalla	230 ms	GB: London	— ms	GB: London
245 ms	GB: London	231 ms	GB: London	235 ms	GB: London Docklands	95 ms	BR: Fortaleza
230 ms	GB: Manchester	222 ms	FR: Gravelines	415 ms	AU: Melbourne	244 ms	IT: Milan
270 ms	IS: Hafnarfjordur	266 ms	IT: Arezzo #1	259 ms	IT: Arezzo #2	257 ms	IT: Bergamo
240 ms	BE: Brussels	236 ms	CH: Zurich	246 ms	CH: Zurich	251 ms	CH: Zurich
367 ms	AU: Sydney	345 ms	AU: Sydney	414 ms	AU: Sydney	333 ms	AU: Sydney
246 ms	US: Honolulu	244 ms	NL: Amsterdam	229 ms	NL: Amsterdam	250 ms	NL: Amsterdam
242 ms	DE: Frankfurt	240 ms	DE: Frankfurt	250 ms	DE: Frankfurt	243 ms	DE: Frankfurt
260 ms	SI: Ljubljana	260 ms	HR: Zagreb	266 ms	AT: Graz	271 ms	CZ: Prague
248 ms	CZ: Prague	255 ms	CZ: Prague	251 ms	AT: Vienna	266 ms	MK: Skopje
255 ms	CZ: Havlickuv Brod	257 ms	GR: Thessaloniki	262 ms	SK: Bratislava	251 ms	CZ: Brno
239 ms	CZ: Brno	266 ms	RS: Belgrade	255 ms	RS: Belgrade	253 ms	CZ: Rtyne v Podkrkonosi
252 ms	HU: Budapest	267 ms	HU: Budapest	271 ms	CZ: Zlin	254 ms	DK: Copenhagen
269 ms	PL: Stargard	255 ms	CZ: Ostrava	269 ms	PL: Katowice	260 ms	NO: Oslo
268 ms	RO: Bucharest	249 ms	PL: Warsaw	260 ms	PL: Warsaw	265 ms	PL: Warsaw
269 ms	PL: Warsaw	274 ms	TR: Istanbul	286 ms	CY: Limassol	284 ms	CY: Limassol
285 ms	IL: Tel Aviv	292 ms	IL: Tel Aviv	292 ms	IL: Tel Aviv	271 ms	SE: Stockholm
269 ms	SE: Stockholm	256 ms	SE: Stockholm	281 ms	MD: Chisinau	55 ms	BR: Rio de Janeiro
279 ms	LT: Vilnius	284 ms	LV: Riga	264 ms	EE: Tallinn	269 ms	EE: Tallinn
328 ms	SA: Riyadh	285 ms	RU: St. Petersburg	282 ms	RU: Moscow	— ms	AZ: Nakhchivan
360 ms	AE: Dubai	99 ms	CO: Bogota	101 ms	CO: Bogota	413 ms	TL: Dili
388 ms	IN: Mangalore	59 ms	PE: Lima	348 ms	IN: Mumbai	387 ms	IN: Bangalore
371 ms	IN: Hyderabad	29 ms	CL: Santiago	454 ms	SG: Singapore	381 ms	SG: Singapore
409 ms	SG: Singapore	427 ms	IN: Delhi	384 ms	BD: Dhaka	414 ms	BD: DHAKA
424 ms	BD: Dhaka	406 ms	BD: CUMILLA	298 ms	JP: Tokyo	307 ms	JP: Tokyo
317 ms	JP: Osaka	337 ms	HK: Kwai Chung	362 ms	HK: Hong Kong	344 ms	KR: Seoul
343 ms	KR: Seoul						

Figura 32 Tiempos de respuesta de diferentes servidores ubicados en distintos países del mundo. Imagen tomada de meter.net.

Observando la figura 32, se realiza una prueba en el sitio meter.net donde los sitios con mejor respuesta son coloreados en verde, los intermedios en naranja y los de peor, en rojo. Se puede ver que la respuesta de un servidor ubicado en Buenos Aires, es de 24 ms mientras que de uno radicado en Río de Janeiro es de 55 ms. Si bien ambos tiempos son muy aceptables, el de un servidor radicado en Buenos Aires permite obtener el doble de rápido la respuesta en comparación con uno ubicado en Río de Janeiro.



# Capítulo 4 - Implementación

A continuación, se detalla el proceso de implementación del sistema de software, el cual se llevó de manera iterativa e incremental de la mano del club utilizado como caso de estudio. Esto permite empezar con la entrega de una aplicación mínima que cumpla su función base y a partir de la retroalimentación que se obtiene por parte del club, añadir correcciones, mejoras y nuevas funcionalidades conforme es necesario.

## 4.1 Estructura de las aplicaciones

La estructura que se desarrolla conforme se crean carpetas y archivos dentro de una aplicación, a la hora de escribir código, es fundamental para garantizar buenas prácticas permitiendo escalar fácilmente y tener un código más legible. Para contar con un software robusto, la aplicación Frontend sigue los patrones de diseño MVC (modelo, vista, controlador). Este patrón es independiente de la plataforma en que se lo utiliza, y es por esto mismo que puede variar levemente de acuerdo a la tecnología utilizada.

Comenzando por el Frontend, es decir con la aplicación multiplataforma desarrollada en Flutter.

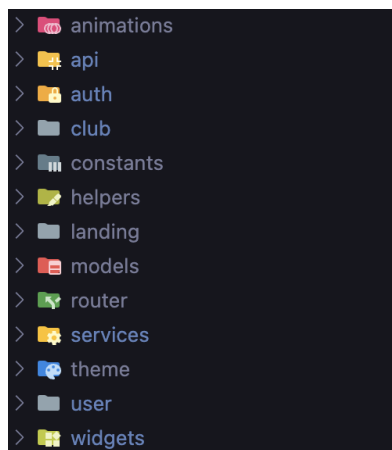


Figura 33 Estructura de proyecto en Flutter. Imagen del autor.

En la figura 33 se observa el patrón MVC con leves modificaciones adaptadas especialmente para el entorno de desarrollo en Flutter. A continuación se describe el contenido del código que se guarda dentro de cada carpeta:

- /animations: Animaciones visuales.
- /api: Llamadas API REST, mediante estas llamadas HTTP es como se comunica la aplicación.
- /auth: Es el módulo de autenticación, contiene tanto las vistas de inicio de sesión, registro, recuperar contraseña, como los controladores
- /club: Es el módulo para clubes, contiene vistas y controladores para administrar reservas y todo lo referido al club.
- /constants: Constantes que se utilizan a lo largo de toda la aplicación.
- /helpers: Funciones reutilizables útiles.
- /models: Los modelos referidos al negocio.
- /router: Las rutas de la aplicación para poder navegar en distintas pantallas o interfaces.
- /services: Servicios que utiliza como dependencia la aplicación. En este caso Firebase (BaaS) es uno.
- /theme: Tema de la aplicación, diseños comunes, fuentes de texto.
- /user: Es el módulo para usuarios. Contiene vistas y controladores para las reservas y todo lo referido al usuario.
- /widgets: Interfaces visuales reutilizables en distintas pantallas de la aplicación.

Siguiendo por el Backend, es decir con la aplicación del servidor con la lógica de negocio que se comunica directamente con la Base de Datos, la cual se desarrolla

en Golang. Se sigue una arquitectura de Service Layer con un enfoque DAO para el acceso a datos. Esto ayuda a comunicar la estructura general de la aplicación y cómo se organizan las responsabilidades entre diferentes componentes. Proporciona una separación clara de responsabilidades, donde los servicios manejan la lógica de negocio y la coordinación entre diferentes partes de la aplicación, mientras que los managers se encargan de la manipulación directa con la base de datos.

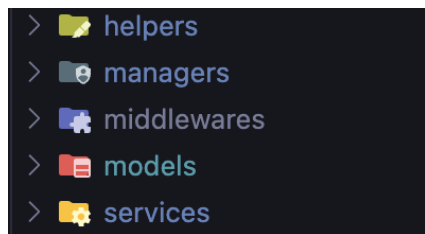


Figura 33 Estructura de proyecto en Golang. Imagen del autor.

En la figura 33 se observa la estructura y organización del proyecto en Golang. A continuación se describe el contenido del código que se guarda dentro de cada carpeta:

- /services: Los servicios actúan como intermediarios entre los controladores o los endpoints de la aplicación y la lógica de negocio subyacente. En este caso, los servicios serían responsables de orquestar las operaciones de la aplicación, manejar la lógica de negocio y traducir las peticiones del exterior (como peticiones HTTP) en llamadas a los managers o DAOs correspondientes.
- /managers: Pueden considerarse una implementación de DAO. Los DAOs se encargan de interactuar directamente con la capa de persistencia de datos (como una base de datos) para realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar).

- `/models`: Es donde se definen las estructuras de datos fundamentales utilizadas en la aplicación. Estas estructuras pueden representar objetos del mundo real, como reservas, canchas, provincias, etc.
- `/helpers`: En la carpeta helpers, típicamente se encuentran funciones de utilidad o herramientas que son útiles en varias partes de la aplicación pero que no están relacionadas con la lógica de negocio principal. Estas funciones pueden incluir operaciones comunes, manipulación de cadenas, conversión de tipos, etc.

## 4.2 Desarrollo a través de dependencias

La mayoría de frameworks modernos permiten utilizar componentes en forma de dependencias para un desarrollo más rápido. Es el caso de Flutter, al tratarse de una solución de código abierto, muchos de estos componentes son desarrollados por la comunidad y puesto a disposición para todos los desarrolladores. No se desarrolla partiendo desde cero, si no que se hace uso de componentes de otros desarrolladores, si así se lo desea. La comunidad, al ser tan grande casi siempre tiene un componente para lo que se necesita, y en caso de no tenerlo, uno puede desarrollar uno propio y si así lo desea compartirlo con la comunidad. El sitio [pub.dev](https://pub.dev) es donde se alojan todos estos componentes, llamados “packages”.

Por ejemplo, si se requiere hacer un gráfico de barras, probablemente realizar esta tarea lleve varios días y mucho escritura de código. Pero uno puede dirigirse al sitio y en cuestión de minutos utilizar un componente que permita realizar este tipo de gráficos. Los “packages” son agregados como dependencias del proyecto y son una práctica muy común.

Hay que tener en cuenta que una de las desventajas es que estos paquetes pueden quedar obsoletos si el programador que lo desarrolló no le dió soporte a lo largo del tiempo. Supongamos que uno utiliza la versión de Flutter 1.0.0 y se encuentra desarrollando una aplicación que utiliza varios de estos paquetes, entonces si hay una nueva versión del framework Flutter (ya sea con mejoras, nuevas funcionalidades o corrección de errores) por ejemplo la versión 2.0.0, y uno actualiza a la versión más reciente, tiene que asegurarse que los componentes o paquetes utilizados en el proyecto están actualizados a dicha versión, de lo contrario la aplicación puede dejar de funcionar o incluso no compilar.

Es por esto que uno no debe añadir paquetes sin analizar, si evidentemente es necesario y sin determinar si el paquete fue desarrollado por un programador o entidad de confianza que lo mantenga a lo largo del tiempo. Existe un equipo designado por Google que se encarga de verificar que estos paquetes sean en esencia útiles, y también marcan aquellos que son de excelente calidad, por lo que la elección de estas dependencias para un proyecto es una tarea relativamente sencilla. La parte más compleja se encuentra en hacer uso de estos paquetes, muchas veces requieren configuraciones adicionales en cada plataforma. Por ejemplo si se añade un paquete que facilita manipular fotos, en iOS este paquete requiere de una configuración de cierta forma, en Android de otra y en la Web de otra distinta, esto debido a que requiere interactuar con el hardware respectivo de cada plataforma.

A continuación se listan las dependencias más importantes utilizadas en este proyecto y el uso que se les da a cada una de ellas, estas dependencias aparecen listadas con sus respectivas versiones en un archivo llamado “pubspec.yaml”, el cual es automáticamente creado por Flutter al iniciar un nuevo proyecto.

**cached\_network\_image:** Permite cargar automáticamente imágenes a través de un enlace https y guardarlas en la memoria caché para no desperdiciar recursos. Lo único que debe hacer el programador, es realizar una llamada al paquete pasando como parámetro el enlace https donde se encuentra la imagen y el paquete automáticamente toma la imagen de la memoria caché y si no la encuentra entonces del servidor remoto.

**cloudinary:** Este paquete hace uso de funciones que permiten comunicarse con Cloudinary, un proveedor de almacenamiento de archivos en la nube. Cloudinary no fue tenido en cuenta durante la planeación del desarrollo del presente proyecto pero fue estrictamente necesario para almacenar imágenes como las portadas de los clubes, ya que de lo contrario se hubiese necesitado una infraestructura completa con un servidor de archivos propios.

**data\_table\_2:** Este, como su nombre lo indica, es una extensión al componente para crear tablas que viene por defecto en Flutter. Es necesario para aplicaciones donde se requieran tablas de datos más complejas porque Flutter ofrece un componente muy básico. Una de las ventajas que ofrece son los encabezados fijos. Es decir los encabezados de una tabla permanecen en el lugar que se encuentran cuando uno se mueve haciendo scroll a través de la tabla, esto fue necesario para que se puedan crear reservas sin perder de vista en qué canchas se está colocando la reserva. Se puede ver una demostración del creador del componente en este [sitio](#). Para este caso particular, me encontré con un limitante en el componente, por lo que hice un fork del repositorio del creador (es decir cloné su código) hice el cambio que requería, y finalmente abrí un Pull Request (Petición para unir mi código al repositorio del creador), el cual fue aprobado por el autor del paquete. En otras palabras añadí mi cambio a su repositorio, para que todas las personas que usen el paquete puedan contar con el cambio que yo hice. El link al Pull request puede verse en [Github](#).

**dio:** Se utiliza para crear un cliente HTTP, si bien Flutter cuenta con un cliente HTTP propio, este paquete es muy popular en la comunidad porque permite una customización mucho más compleja. Por ejemplo se pueden añadir “interceptors”, estas son funciones que se ejecutan en cada petición HTTP. Es de mucha utilidad si, por ejemplo, queremos verificar en cada petición si un token de autenticación está vencido antes de hacer una llamada a un servidor.

**firebase\_auth:** Facilita funciones de autenticación a través de Firebase. Esto quiere decir que el backend como servicio Firebase, no sólo ahorra la creación de todo un microservicio de autenticación como backend, si no que también facilita la conexión al mismo poniendo a disposición este paquete en el Frontend.

**firebase\_crashlytics:** Permite enviar reporte de errores a una consola de Firebase, permitiendo ver exactamente en qué línea de código ocurrió el error, en qué fecha, hora, dispositivo, sistema operativo, y demás información.

**firebase\_messaging:** Otro paquete que ofrece funcionalidades de Firebase, en este caso se utiliza para recibir notificaciones tanto en iOS, Android como en la Web.

**flutter\_bloc:** Se trata de un manejador de estados que sigue el patrón BLoC (Business Logic Component). Flutter ofrece una forma de mantener los estados, por ejemplo al pasar de una pantalla a otra, mantener los valores de las variables. Pero este paquete es muy popular en la comunidad porque actúa como intermediario entre las vistas (Widgets en Flutter) y las fuentes de datos remotas o locales. La siguiente imagen muestra cómo funciona BLoC permitiendo separar toda la lógica de negocio en una nueva capa, manteniendo en los Widgets únicamente código referido a la interfaz.





Figura 35: Esquema de funcionamiento del manejador de estados BLoC. Imagen de bloclibrary.dev

Se puede observar que los widgets, UI, vistas o interfaces son los encargados de enviar eventos, los cuales son recibidos y procesados por bloc, él mismo hace peticiones a data, que puede ser por ejemplo el servidor de reservas y también se encarga de emitir estados a los widgets. De esta forma las vistas solo emiten eventos, como por ejemplo “ObtenerReservasEvento” el cual es recibido por bloc, bloc emite un estado a los widget “CargandoEstado”, lo cual permite mostrar alguna interfaz de carga, a su vez bloc envía una petición HTTP al servidor, espera una respuesta, la procesa y emite un estado “ReservasObtenidasEstado” y los widget pueden entonces mostrar las reservas. Esto no es más que la separación de la lógica de negocio en una nueva capa, por lo que puede considerarse a bloc como el controlador del patrón MVC en Flutter.

**hive\_flutter:** Hive es una base de datos local. Se utiliza para guardar cualquier objeto que no sea necesario guardarlo en la base de datos remota. Por ejemplo si un usuario ha elegido Tenis como deporte, al cerrar y abrir la aplicación se pierde el estado y por lo tanto no tiene un deporte seleccionado, esto puede guardarse a través de Hive localmente para que cuando inicie la aplicación se carguen desde el almacenamiento local las preferencias del usuario.

Por el lado del servidor, en Golang también existen estas dependencias pero estas se conocen como “modules”. Estas si bien son de gran utilidad y se suelen utilizar

varias por proyecto, comúnmente no se usan en gran cantidad como sí sucede en Flutter por el hecho de que necesita abstraerse de las multiplataformas, haciendo que haya una gran cantidad de paquetes que a través de la abstracción, hacen de intermediario entre el código en Flutter y el código Nativo como lo es en iOS, Android, Web, Windows y MacOS. A continuación se listan las dependencias más importantes utilizadas y el uso que se les da a cada una de ellas, estas dependencias aparecen listadas con sus respectivas versiones en un archivo llamado “go.mod”, el cual es automáticamente creado por GO al iniciar un nuevo proyecto.

**firebase:** Facilita funciones de autenticación a través de Firebase. Este módulo sirve para comunicar el microservicio de reservas con el de Firebase, facilitando el proceso de llamadas y peticiones.

**go-sql-driver:** Se utiliza para conectarse de manera relativamente sencilla a una bases de datos, en este caso se hace uso del driver para la conexión con MYSQL.

**echo:** Este módulo pone a disposición todas las funciones necesarias para crear una API REST utilizando el lenguaje GO. A través de este módulo se simplifica la creación de endpoints, middlewares, la serialización y deserialización de respuestas JSON.

**gorilla:** Permite hacer uso del protocolo Websocket. Se usa para poder realizar reservas en tiempo real sin que el complejo deportivo tenga que actualizar manualmente la aplicación con el fin de recibir nuevas reservas.

**gorm:** Es lo que comúnmente se conoce como un ORM, es decir es una librería que facilita la comunicación con la Base de Datos, haciendo uso del propio lenguaje, en este caso GO. De esta forma se reduce el código SQL para hacer consultas a la Base de Datos y se utiliza en mayor proporción Golang. La desventaja que suele tener cualquier ORM es la falta de flexibilidad, llega un punto que hacer una consulta muy

compleja, no es posible a través de la librería y por consecuente hay que hacerlo en SQL plano. Los ORM permiten desarrollar utilizando el mismo lenguaje que se está utilizando para el servidor, abstrayendo el lenguaje utilizado en la Base de Datos, facilitando el proceso de consultas a la misma.

## 4.3 Implementación del BaaS

Firebase, al igual que muchos servicios similares, provee una consola o “dashboard” en donde se puede administrar los servicios que se utilizan. Como se mencionó anteriormente, los servicios de Firebase que se utilizan en el presente proyecto son “Web Hosting”, “Authentication”, “Crashlytics” y “Messaging”.

**Web Hosting:** Ofrece un servidor en donde alojar una página web con certificado SSL de manera completamente gratuita. Para poder hacer uso de este servicio, se requiere del CLI (Command Line Interface) de Firebase. Esto no es más que un programa que se instala en una computadora que a diferencia de programas convencionales, se hace uso de él a través de comandos. Una vez descargado, se debe compilar el proyecto en la versión web para poder ser alojado en el hosting. En Flutter, para poder compilar un proyecto para la plataforma Web, se debe ejecutar el comando “flutter build web”, esto genera una serie de carpetas y archivos que contienen el código Web compilado. Luego a través del CLI de Firebase, estando en el directorio raíz del proyecto de Flutter, ejecutando el comando “firebase deploy” se muestra una serie de pasos para desplegar la web de forma automática.

**Authentication:** Como su nombre lo indica, se refiere al servicio de autenticación. Dentro del mismo se ofrecen múltiples funcionalidades, pero en este proyecto se hizo uso del registro, inicio de sesión, recuperación de contraseña y de la

renovación y verificación de sesiones. El servicio de autenticación tiene su propia Base de Datos la cual es administrada a través de la consola de Firebase, en la misma es entonces posible visualizar los usuarios creados, así como también eliminarlos. A su vez el equipo de Firebase se encarga de desarrollar librerías que permitan la integración con la autenticación de Firebase de manera sencilla. Como se mostró, se usa una dependencia en Flutter que facilita las funciones que Firebase ofrece. El siguiente código permite crear un usuario en Firebase haciendo uso de la librería en Flutter.

```
await firebase.createUserWithEmailAndPassword(  
  email: event.email,  
  password: event.password,  
);
```

Figura 36: Registro de un usuario en Firebase. Imagen del autor.

Si la respuesta de la llamada del código mostrado en la Figura 36 es exitosa, entonces en la consola de Firebase se puede visualizar un nuevo usuario, como lo muestra la siguiente figura.



The screenshot shows the 'Authentication' section of the Firebase console. The 'Usuarios' tab is selected, displaying a table of users. The table has columns for 'Identificador', 'Proveedores', 'Fecha de creación', 'Fecha de acceso', and 'UID de usuario'. There are six rows of user data, each with a redacted email address and a unique UID.

Identificador	Proveedores	Fecha de creación	Fecha de acceso	UID de usuario
[redacted]@gmail.com	✉	18 abr 2023	18 abr 2023	WbxyDjHv1jyp0b8loVVli52i2
[redacted]@gmail.com	✉	19 jun 2023	19 jun 2023	2W6d[redacted]aKrwTXZ...
[redacted]@gmail.c...	✉	22 jun 2023	22 jun 2023	vg[redacted]3diw9vM3u...
[redacted]@gmail...	✉	28 mar 2023	1 jul 2023	hbK3nmoorgaaxr5uAtKEx2...
[redacted]@icloud.com	✉	1 jul 2023	1 jul 2023	Xfw[redacted]OLRLScug4...
[redacted]@gm...	✉	31 may 2023	18 jul 2023	bb3[redacted]1925juFW...

Figura 37: Consola de Firebase del servicio de "Authentication". Imagen del autor.

Por otro lado, para poder validar una sesión en el servidor, en este caso, el microservicio de reservas, se utiliza nuevamente una dependencia de Firebase que permite validar el token asociado a cada petición HTTP. De esta forma el flujo es de la siguiente forma: Se llama a la función de inicio de sesión provista por Firebase en Flutter, si el inicio de sesión es exitoso, la función devuelve un token. Este token es guardado en la aplicación de Flutter y es incluido en cada petición HTTP que se realiza al servidor de reservas. El servidor de reservas utiliza un middleware que verifica en cada petición que el token sea válido . La siguiente figura muestra el código utilizado en el servidor GO para validar una sesión a través de la dependencia de Firebase.

```
bearerToken := strings.Replace(c.Request().Header.Get("Authorization"), "Bearer ", "", -1)

if len(bearerToken) == 0 {
    return echo.NewHTTPError(http.StatusUnauthorized, "Unauthorized")
}

_, err = client.VerifyIDToken(firebase.FirebaseAppInstance.Context, bearerToken)
if err != nil {
    return echo.NewHTTPError(http.StatusUnauthorized, "Unauthorized")
}

return next(c)
```

Figura 38: Verificación del token en el servidor de reservas. Imagen del autor.

**Crashlytics:** Este servicio de Firebase se utiliza para reportar errores de manera remota. Me encontré en situaciones en las que el club bajo estudio tenía problemas para abrir la aplicación y no había manera de determinar cuál era el problema, ya que en mis dispositivos móviles el error no se reproducía. Con la dependencia de Crashlytics en Firebase, cada vez que sucede un error, se reporta de forma automática a la consola de Firebase, mostrando información muy útil para detectar y corregir errores. La siguiente figura muestra un error reportado en la consola de Firebase perteneciente a la plataforma iOS.



Resumen del evento 1.3.2 (1) iOS+ iOS 17.3.1 iPhone 14 Pro 29 feb 2024, 1:02:44 p.m.

Seguimiento de pila Claves Registros y rutas de navegación Datos Lanzamientos

TXT

**Fatal Exception: FlutterError**  
Null check operator used on a null value. Error thrown .

0 tabs.dart - Línea 1911  
\_TabBarViewState.\_handleTabControllerAnimationTick + 1911

1 listener\_helpers.dart - Línea 161  
AnimationLocalListenersMixin.notifyListeners + 161

Figura 39: Consola de Crashlytics en Firebase. Imagen del autor.

En la figura 39 se puede ver que esta es una falla que ocurrió en un dispositivo móvil iPhone 14 Pro corriendo el sistema operativo iOS 17.3.1 el 29 de febrero del 2024 a las 1:02:44 pm en la versión de la aplicación 1.3.2.

**Messaging:** Este servicio de Firebase se usa para enviar notificaciones push a diversos dispositivos. En el caso de este proyecto se utiliza para enviar notificaciones al club como por ejemplo “Tienes una nueva reserva” o al usuario como por ejemplo “El club ha cancelado tu reserva”. Para poder hacer uso de este servicio, se instala la dependencia correspondiente en Flutter que permite obtener un token o código asociado al dispositivo. La siguiente figura muestra la función en Flutter que permite obtener dicho token.

```
static Future<String?> getToken() async {  
  try {  
    final String? fcmToken = await messaging.getToken(vapidKey: kIsWeb ? ConfigReader.getWebFCMToken() : null);  
    return fcmToken;  
  } catch (error) {  
    if (!kIsWeb) {  
      FirebaseCrashlytics.instance.recordError(error, null, reason: 'failed to get notification token!');  
    }  
    return null;  
  }  
}
```

Figura 40: Obtención del token para enviar notificaciones. Imagen del autor.

Luego de que el token es obtenido, el mismo debe ser almacenado en la Base de Datos asociándolo al usuario que ha iniciado sesión. De este modo, cuando por

ejemplo se realiza una reserva, se busca el token de notificación asociado al club al cual se le realiza la misma y se envía una notificación. La siguiente figura muestra el código de la función de firebase en el microservicio de reservas que envía la notificación haciendo uso de la dependencia de Firebase en GO.

```
_, err := firebase.FirebaseAppInstance.MessagingClient.Send(context.Background(), message)
if err != nil {
    fmt.Printf("Failed to send FCM message: %v", err)
}
```

Figura 41: Envío de notificación desde el servidor de reservas. Imagen del autor.

En la figura 41 se puede observar que la función recibe como parámetro “message” el cual contiene el token de la notificación anteriormente obtenido, para poder saber quien es el destinatario, y el mensaje en sí.

## 4.4 Contratación de un Cloud Server

Una vez que se tiene una versión compilada del servidor de reservas, es necesario alojarlo en un servidor en la nube para poder acceder remotamente a él, ya sea para que la aplicación móvil se comunice con él o para realizar tareas de administración. DonWeb no cuenta con un plan gratuito, pero el plan básico es totalmente accesible y de mucha utilidad para realizar pruebas o incluso poner en producción la aplicación hasta que la misma obtenga usuarios y requiere escalar.

Planes y precios de Servidor Cloud

## Precio predecible cada mes

	CPU	MEMORIA	STORAGE *	TRANSFERENCIA		Con tu pago anual 19% OFF
	1 vCPU	1 GB	10 GB	1 TB	\$1.057 <sup>19</sup> /mes	AGREGAR AL CARRO
	2 vCPU	2 GB	20 GB	2 TB	\$2.081 <sup>99</sup> /mes	AGREGAR AL CARRO
	2 vCPU	4 GB	30 GB	2 TB	\$2.895 <sup>22</sup> /mes	AGREGAR AL CARRO
	4 vCPU	4 GB	40 GB	3 TB	\$3.913 <sup>40</sup> /mes	AGREGAR AL CARRO
	8 vCPU	8 GB	50 GB	5 TB	\$7.272 <sup>08</sup> /mes	AGREGAR AL CARRO

**MÁS VENDIDO** →

**Elige la mejor combinación para tu proyecto**  
hasta **16 vCPU's**, **64 GB RAM** y **480 GB** almacenamiento SSD

[CONFIGURA EL TUYO](#)

Figura 42: Precios para un Cloud Server según recursos. Imagen del autor.

En esta ocasión se toma la primera opción con el fin de realizar pruebas. La opción pone a disposición una máquina virtual de 1 vCPU, memoria RAM de 1 GB, almacenamiento de 10 GB y transferencia mensual de hasta 1 TB. Una vez contratado el servicio se puede acceder a un menú en donde se cuenta con información del Cloud Server.

**ESTADO** ●

Nombre descriptivo

**Encendido** desde el 12/08/2023, 4:48:26 pm.

Nodo: nova001

<> Consola VNC
🔌 Apagar
🔄 Reiniciar

**RECURSOS**

<p><b>Procesamiento</b></p> <p>2 vCPU's / 2 GB</p>	<p><b>Almacenamiento</b></p> <p>Contratado: 10 GB</p> <p>Sistema Operativo &amp; otros: 4 GB</p> <p>Utilizado: <b>12.4 GB</b> de 14 GB SSD</p> <div style="background-color: #007bff; width: 100%; height: 10px; margin-top: 5px;"></div>	<p><b>Transferencia</b></p> <p>Utilizado: <b>2.68 GB</b> de 1000 GB</p> <p>Tu consumo mensual de transferencia se reiniciará en: 1 días.</p> <div style="background-color: #007bff; width: 100%; height: 10px; margin-top: 5px;"></div>
--	---	---

Escalar recursos



Figura 43: Menú principal de DonWeb. Imagen del autor

En este menú se puede observar algunas métricas, como el almacenamiento o la transferencia en tiempo real así como también se puede acceder al menú para escalar recursos. La siguiente figura muestra la pestaña abierta una vez que se hace click sobre el botón verde “Escalar recursos”. Uno puede seleccionar los recursos que quiera escalar, y en aproximadamente menos de 5 minutos tendrá los cambios aplicados y funcionando, esto gracias a cómo los Cloud Servers están gestionados a través de máquinas virtuales en un servidor físico mucho más grande y de mayor potencia.

Selecciona recursos luego haz clic en Continuar.

The image shows a resource scaling interface with four sliders:

- vCPU:** Slider with values 1, 2, 4, 8, 12, 16. The slider is positioned at 2.
- RAM (GB):** Slider with values 1, 2, 4, 6, 8, 10, 12, 14, 16, 24, 32, 48, 64. The slider is positioned at 2.
- Almacenamiento:** Slider with values 10 and 480. The slider is positioned at 10. Below it, the text reads: "Recuerda que el cambio de almacenamiento no se puede volver atrás."
- Transferencia (TB):** Slider with values 1, 2, 3, 4, 5, 6, 8, 10, 15. The slider is positioned at 1.

At the bottom right, there are two buttons: "Cancelar" (blue text) and "Continuar" (green button).

Figura 43: Menú principal de DonWeb. Imagen del autor.

También existe la posibilidad de visualizar métricas en tiempo real, muy útiles para entender el tráfico del servidor y determinar si es necesario escalar.

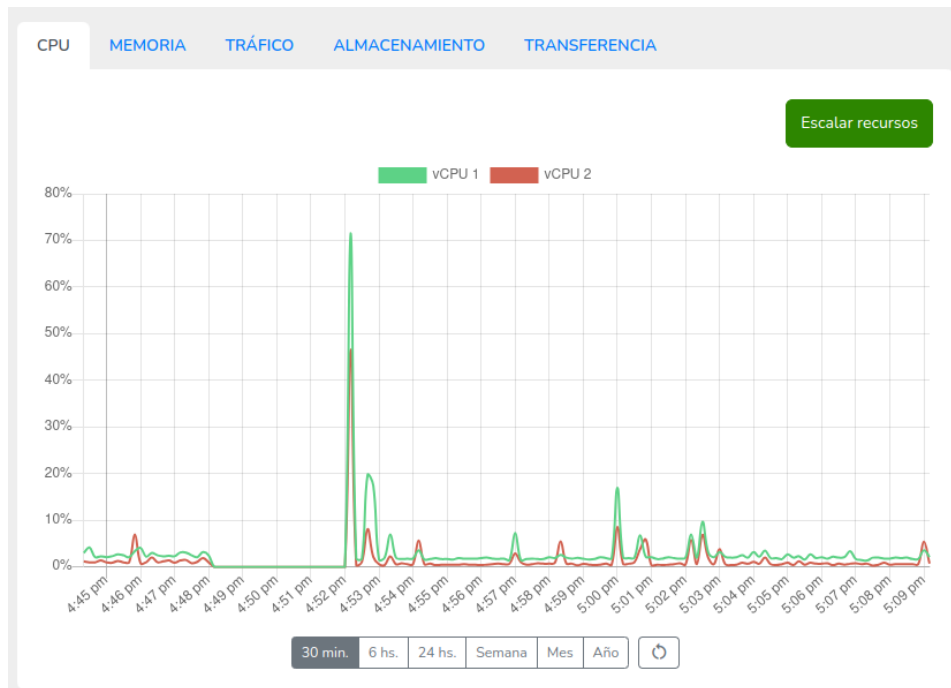


Figura 44: Uso del CPU en el Cloud Server. Imagen del autor

La figura 44 muestra el uso del CPU en los últimos 30 minutos, pero también es posible visualizar cada 6 hs, día, semana, mes y año. Si bien en la figura 44 se muestra la CPU, la interfaz permite cambiar de solapa y observar la memoria RAM, el tráfico, almacenamiento y la transferencia.

Otra funcionalidad importante ofrecida por el proveedor es la posibilidad de crear, instalar y eliminar Snapshots.

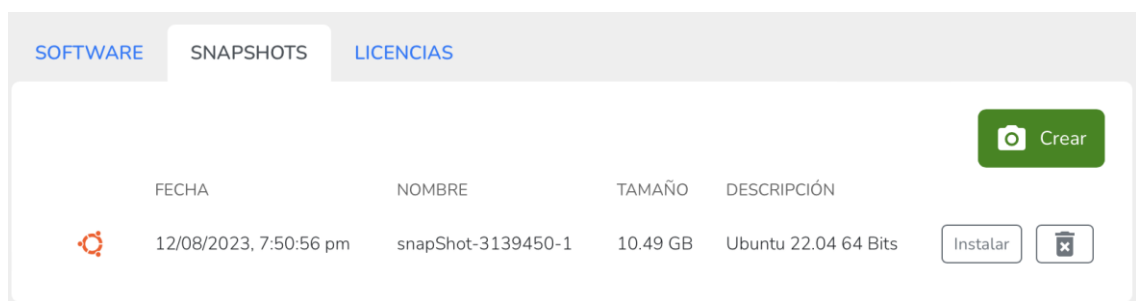


Figura 45: Solapa de Snapshots. Imagen del autor

Un Snapshot es como una foto de la máquina virtual del Cloud Server el cual permite guardar el estado en el que se encuentra al momento de tomar la foto. De esta

manera es muy fácil recuperar un estado previo. Se utiliza como medida de seguridad en caso de requerir de un estado anterior al del que se encuentra actualmente en uso.

El proveedor proporciona los datos suficientes para poder acceder de manera remota al Cloud Server. La siguiente figura muestra datos proporcionados por DonWeb que permiten, a través de una conexión SSH, acceder al mismo. Según Ylönen (1995), "Secure Shell (SSH) es un protocolo de red que permite a los usuarios comunicarse de manera segura con un dispositivo remoto a través de una conexión encriptada. SSH proporciona una alternativa segura a los protocolos de comunicación no cifrados, como Telnet, al cifrar tanto los datos transmitidos como las credenciales de autenticación, lo que garantiza la confidencialidad e integridad de la comunicación."

The screenshot displays a cloud server management interface with two main sections: 'SOFTWARE' and 'SSH'.

**SOFTWARE Section:**

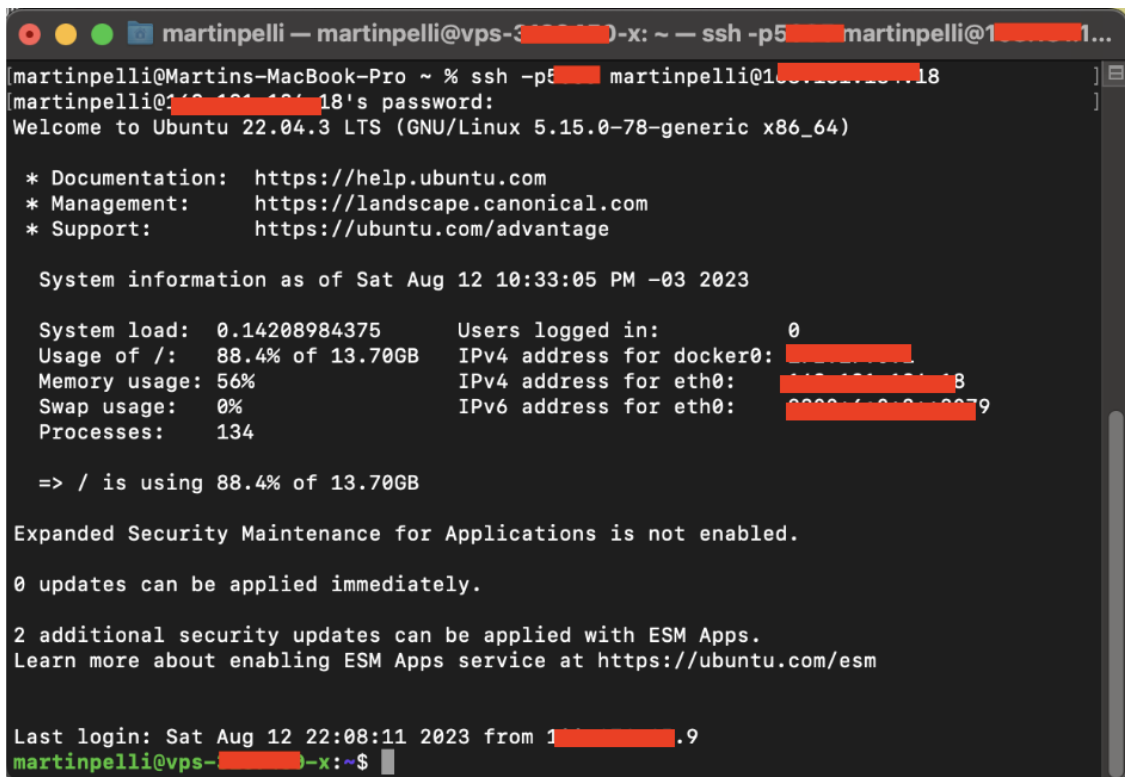
- Navigation tabs: SOFTWARE (selected), SNAPSHOTS, LICENCIAS.
- Sistema operativo:**
  - Plataforma: Linux
  - Sistema operativo: Ubuntu 22.04
  - Versión de SO: 22.04
  - Arquitectura: 64 bits
- Aplicación o Stack:**
  - Aplicación: Instalacion Minima
  - Imagen: Ubuntu2204-64-min
  - [Conocer primeros pasos](#)
- Buttons: Recrear, Vaciar.

**SSH Section:**

- Tab: SSH.
- Datos de acceso:**
  - Hostname: vps-...-x.dattaweb.com
  - IP: ...
  - Usuario: root
  - Contraseña: [masked]
  - Puerto: 22
- Text: Si deseas conectarte remotamente a través de SSH, sigue [estas instrucciones](#).
- Button: Acceso VNC.

Figura 46: Información de software y conexión SSH del Cloud Server. Imagen del autor

La figura 46 muestra el software que se encuentra ejecutado la máquina virtual y otros datos menores. En la solapa de SSH se presentan los datos necesarios para poder acceder de forma remota y comenzar a utilizar la máquina virtual a través de una consola. SSH no viene por defecto en Windows, pero está disponible como opción para instalar. En sistemas operativos basados en Unix, como Mac OS y la mayoría de las distribuciones de Linux, SSH suele estar preinstalado como parte del sistema operativo. Sin embargo, en Windows, los usuarios pueden instalar un cliente SSH de terceros o utilizar la característica opcional "OpenSSH Client" que se puede habilitar a través de la Configuración de Windows. En este caso particular se cuenta con una versión de Mac OS y no es necesario instalar nada.



```
martinpelli — martinpelli@vps-3[REDACTED]-x: ~ -- ssh -p5[REDACTED]martinpelli@1[REDACTED]1...
[martinpelli@Martins-MacBook-Pro ~ % ssh -p[REDACTED] martinpelli@1[REDACTED]18
martinpelli@1[REDACTED]18's password:
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 5.15.0-78-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sat Aug 12 10:33:05 PM -03 2023

System load:  0.14208984375      Users logged in:      0
Usage of /:   88.4% of 13.70GB   IPv4 address for docker0: [REDACTED]
Memory usage: 56%              IPv4 address for eth0: [REDACTED]8
Swap usage:   0%                IPv6 address for eth0: [REDACTED]9
Processes:   134

=> / is using 88.4% of 13.70GB

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

2 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Sat Aug 12 22:08:11 2023 from 1[REDACTED].9
martinpelli@vps-[REDACTED]-x:~$
```

Figura 47: Conexión SSH al Cloud Server. Imagen del autor

En la figura 47 se puede observar la conexión establecida al Cloud server ejecutando el comando SSH y pasando como argumentos los datos proporcionados por el proveedor, el número de puerto y la IP. En la figura se muestra el nombre de usuario

“martinpelli” porque fue añadido como usuario para evitar el uso del usuario root lo cual es una buena práctica de seguridad. Pero inicialmente es necesario ingresar con el usuario root y la contraseña proporcionada por el proveedor.

Una vez establecida la conexión ya se puede navegar a través de la línea de comandos como si fuese un ordenador convencional con sistema operativo Linux, de hecho puede ser idéntico a uno si se instala un software que proporcione una interfaz gráfica, pero en este caso se trabaja a través de la línea de comandos para evitar licencias de software.

## 4.5 Despliegue de contenedores en la nube

Para poder ejecutar en un contenedor, tanto la Base de Datos como el microservicio de reservas, es necesaria la instalación de Docker en el Cloud Server. La instalación se realiza siguiendo la documentación oficial de [Docker](#) para Ubuntu. La misma consta de ejecutar una serie de comandos, para instalar el repositorio y luego para instalar el paquete de Docker en sí mismo. Una vez instalado, se puede comenzar a ejecutar el comando “docker” seguido de la tarea que se quiere realizar y los parámetros necesarios.

Teniendo en cuenta que los contenedores de Docker se crean a partir de imágenes, se comienza con la creación de la Base de Datos:

```
docker run --name=clickbox-db -p3307:3306 -v mysql-volume:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=SomePassword -d mysql/mysql-server:latest
```

En donde:

**docker run:** Este es el comando principal de Docker para crear y ejecutar un contenedor a partir de una imagen.

**--name=clickbox-db:** Esta opción establece el nombre del contenedor como "clickbox-db". Es útil para hacer referencia al contenedor de manera más fácil y legible.

**-p3307:3306:** Esta opción mapea el puerto 3306 del contenedor al puerto 3307 del host (es decir, del Cloud Server). Esto significa que el servicio de base de datos MYSQL dentro del contenedor estará disponible en el puerto 3307 del host.

**-v mysql-volume:/var/lib/mysql:** Esta opción crea un volumen nombrado "mysql-volume" y lo monta en el directorio "/var/lib/mysql" dentro del contenedor. Los datos persistentes de MYSQL se almacenarán en este volumen, lo que permite que los datos persistan incluso si el contenedor se elimina.

**-e MYSQL\_ROOT\_PASSWORD=SomePassword:** Esta opción establece la variable de entorno MYSQL\_ROOT\_PASSWORD dentro del contenedor con el valor "SomePassword". Esto configura la contraseña de root para acceder a la base de datos MYSQL dentro del contenedor como "SomePassword".

**-d:** Esta opción ejecuta el contenedor en segundo plano, es decir, en modo "detached". Esto significa que el contenedor se ejecutará en segundo plano y no bloqueará la terminal desde la que se lanzó el comando.

**mysql/mysql-server:latest:** Especifica la imagen de Docker a partir de la cual se creará el contenedor. En este caso, se utilizará la imagen mysql/mysql-server con la

etiqueta latest, la cual es la última versión disponible en ese momento en el repositorio Docker Hub.

Una vez creado el contenedor con la Base de Datos, se puede acceder al contenedor con el siguiente comando:

```
docker exec -it clickbox-db bash
```

Luego, dentro del contenedor ya se puede manipular la Base de Datos accediendo a ella a través de MYSQL

```
mysql -u root -p
```

De esta manera, se está accediendo a MYSQL con el usuario root creado con Docker por lo que se debe introducir la contraseña anteriormente creada, en este caso “SomePassword”. Una vez autenticado se puede comenzar a realizar consultas SQL, por ejemplo crear un usuario, una base de datos, una tabla, etc. Pero realizar todos estos pasos solo para manipular la Base de Datos es muy complejo y engorroso, por eso se instala PhpMyAdmin para interactuar directamente con una interfaz desde una página web sin necesidad de acceder al Cloud Server y al contenedor. El siguiente comando corre un contenedor llamado phpmyadmin y lo vincula al contenedor de la base de datos:

```
docker run --name phpmyadmin -v phpmyadmin-volume:/etc/phpmyadmin/config.user.inc.php --link clickbox-db:db -p 82:80 -d phpmyadmin/phpmyadmin
```

De esta forma escribiendo la dirección IP del Cloud Server seguido del puerto 82, se puede acceder a la interfaz.

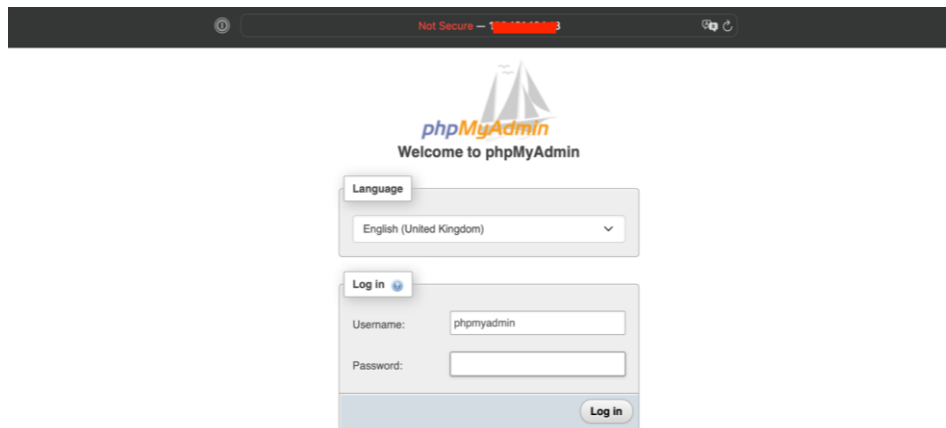


Figura 49: Acceso a PhpMyAdmin alojado en contenedor de Cloud Server. Imagen del autor.

Solo resta clonar el repositorio donde se encuentra el código del microservicio de reservas en el Cloud Server y a partir de dicho código crear una imagen para poder finalmente correr un contenedor del servicio.

Una vez que se cuenta con el repositorio del servicio en el Cloud Server, el cual se puede clonar a través de una herramienta de versionamiento como GIT, se debe crear un archivo Dockerfile el cual le indica cómo crear un imagen a Docker para luego ser montada.

```
# syntax=docker/dockerfile:1

FROM golang:1.19-alpine

WORKDIR /go/golang_wiplay_api

COPY go.mod ./
COPY go.sum ./
RUN go mod download

COPY . .

RUN go build -o /golang_wiplay_api src/main.go

EXPOSE 4747

CMD [ "/golang_wiplay_api" ]
```

Figura 50: Contenido del archivo Dockerfile creado en la raíz del repositorio del servicio Imagen del autor.



La figura 50 muestra un archivo Dockerfile que define los pasos para construir una imagen de Docker que ejecutará el microservicio de reservas.

**# syntax=docker/dockerfile:1:** Esta línea especifica la versión de la sintaxis del Dockerfile que se utiliza. En este caso, se usa la versión 1 de la sintaxis.

**FROM golang:1.19-alpine:** Indica la imagen base para construir la imagen de Docker. Esto significa que se utiliza una imagen de Alpine Linux que ya tiene GO 1.19 instalado como base para la aplicación.

**WORKDIR /go/golang\_wiplay\_api:** Establece el directorio de trabajo dentro del contenedor en /go/golang\_wiplay\_api. Todas las instrucciones posteriores se ejecutarán en este directorio.

**COPY go.mod ./ y COPY go.sum ./:** Copian los archivos go.mod y go.sum desde el directorio local del contexto de construcción al directorio de trabajo /go/golang\_wiplay\_api dentro del contenedor. Estos archivos son necesarios para administrar las dependencias de la aplicación GO.

**RUN go mod download:** Ejecuta el comando go mod download dentro del contenedor. El cual descarga todas las dependencias especificadas en el archivo go.mod y las guarda en el caché del módulo GO.

**COPY . .:** Copia todos los archivos y directorios del contexto de construcción al directorio de trabajo dentro del contenedor. Esto incluye el código fuente de la aplicación GO.

**RUN go build -o /golang\_wiplay\_api src/main.go:** Compila el código fuente de la aplicación GO en un ejecutable llamado `golang_wiplay_api`. El archivo fuente principal `main.go` se encuentra en el directorio `/src` dentro del directorio de trabajo `/go/golang_wiplay_api`.

**EXPOSE 4747:** Expone el puerto 4747 del contenedor. Esto significa que la aplicación GO que se ejecuta dentro del contenedor estará escuchando en el puerto 4747.

**CMD [ "/golang\_wiplay\_api" ]:** Especifica el comando predeterminado que se ejecutará cuando se inicie un contenedor basado en esta imagen. En este caso, se ejecutará `golang_wiplay_api` que se compiló anteriormente.

Para resumir, el archivo `Dockerfile` construye una imagen de Docker que contiene una API escrita en GO. La imagen se basa en Alpine Linux con Go 1.19 instalado. Se copian los archivos de configuración y código fuente de la aplicación, se descargan las dependencias GO, se compila la aplicación y se expone un puerto para que la aplicación sea accesible. Cuando se ejecute un contenedor basado en esta imagen, se ejecutará la aplicación GO.

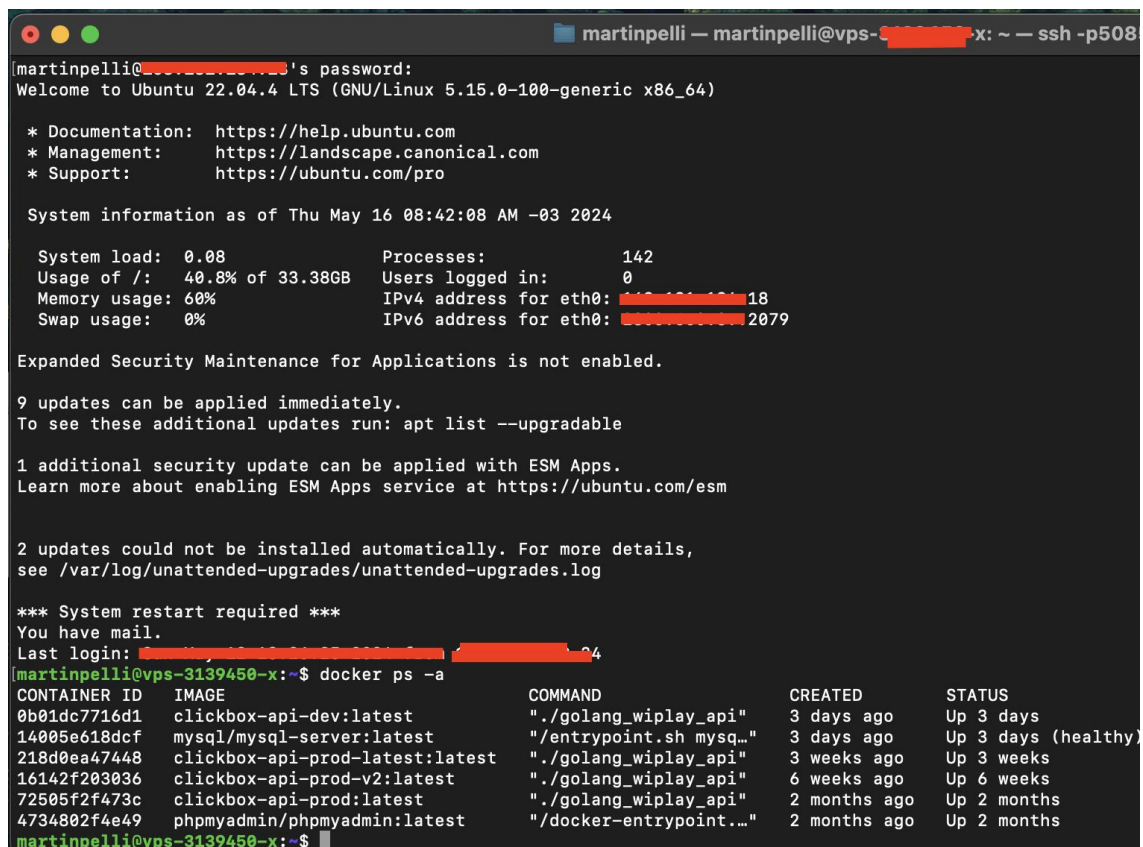
De esta forma ejecutando el comando `docker build` en el directorio donde se encuentra el archivo `Dockerfile` antes creado, se logra crear la imagen.

**`docker build --tag clickbox-api-dev .`**

Una vez creado, se corre el contenedor con dicha imagen

**`docker run -d -p 4747:4747 --name clickbox-api-dev clickbox-api-dev`**

Finalmente, la siguiente figura muestra una lista de los contenedores corriendo en el Cloud Server, los cuales pueden ser accedidos remotamente desde cualquier parte del mundo, siempre y cuando se cuente con la autenticación necesaria. De todas maneras, el Firewall del Cloud Server se configuró para que solo pueda accederse desde una IP específica.



```

martinpelli@vps-3139450-x: ~ -- ssh -p508
martinpelli@vps-3139450-x's password:
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-100-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Thu May 16 08:42:08 AM -03 2024

System load:  0.08          Processes:            142
Usage of /:   40.8% of 33.38GB Users logged in:     0
Memory usage: 60%          IPv4 address for eth0: [REDACTED]18
Swap usage:  0%           IPv6 address for eth0: [REDACTED]2079

Expanded Security Maintenance for Applications is not enabled.

9 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

1 additional security update can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

2 updates could not be installed automatically. For more details,
see /var/log/unattended-upgrades/unattended-upgrades.log

*** System restart required ***
You have mail.
Last login: [REDACTED] 24
martinpelli@vps-3139450-x:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS
0b01dc7716d1   clickbox-api-dev:latest             "./golang_wiplay_api"   3 days ago    Up 3 days
14005e618dcf   mysql/mysql-server:latest           "/entrypoint.sh mysql..." 3 days ago    Up 3 days (healthy)
218d0ea47448   clickbox-api-prod-latest:latest     "./golang_wiplay_api"   3 weeks ago   Up 3 weeks
16142f203036   clickbox-api-prod-v2:latest         "./golang_wiplay_api"   6 weeks ago   Up 6 weeks
72505f2f473c   clickbox-api-prod:latest            "./golang_wiplay_api"   2 months ago  Up 2 months
4734802f4e49   phpmyadmin/phpmyadmin:latest       "/docker-entrypoint..." 2 months ago  Up 2 months
martinpelli@vps-3139450-x:~$

```

Figura 51: Cloud Server ejecutando contenedores. Imagen del autor.

## 4.6 Interfaces del software en funcionamiento

Esta sección está destinada a mostrar las capturas de pantalla más relevantes del sistema desarrollado ya en funcionamiento. Cabe destacar que las interfaces son numerosas, es por esto que se procede a mostrar únicamente una selección de las más

importantes. Un aspecto que contribuye a tener un mayor número de interfaces es el hecho de que el sistema es multiplataforma y si bien, la mayoría de interfaces son comunes en la Web y en los dispositivos móviles, hay muchas otras que están adaptadas a cada entorno para una mejor experiencia de usuario.

Otro aspecto importante a tener en cuenta es que muchas de estas interfaces presentan funcionalidades adicionales no descritas en este trabajo. Esto se debe a que el mismo se desarrolló conforme a cubrir las necesidades básicas de un club y resolver los problemas planteados. Pero una vez finalizado el propósito de este proyecto, el software, al encontrarse operativo, fue creciendo conforme a corrección de errores y necesidades del club bajo estudio, por lo que se cuenta con nuevas funcionalidades, un módulo adicional de torneos, nuevos clubes que lo utilizan y cientos de usuarios con la aplicación instalada en sus dispositivos, los cuales la usan diariamente.

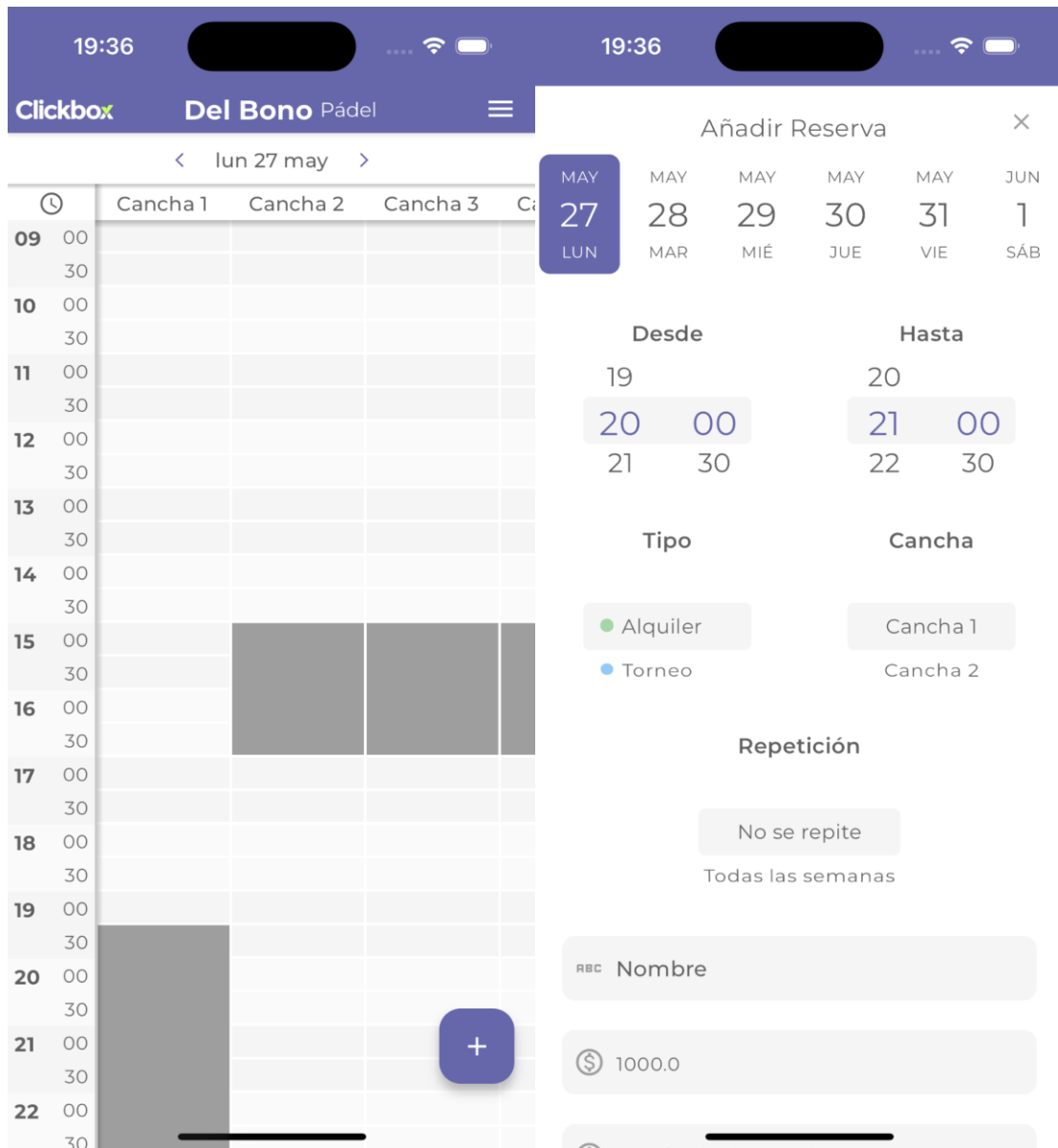


Figura 52. Interfaz de planilla y creación de reservas. Imagen del autor.

La figura 52 muestra la planilla de un club de prueba creado denominado Del Bono. El mismo tiene seleccionada la planilla de pádel el día lunes 27 de mayo. Nuevamente, las celdas grises muestran los horarios no disponibles porque se han marcado como cierre en cada cancha. En esta ocasión no hay reservas presentes. La interfaz de creación de la reserva se abre al presionar el botón de “+” o al tocar una celda en la planilla, en donde se pueden seleccionar los distintos parámetros.

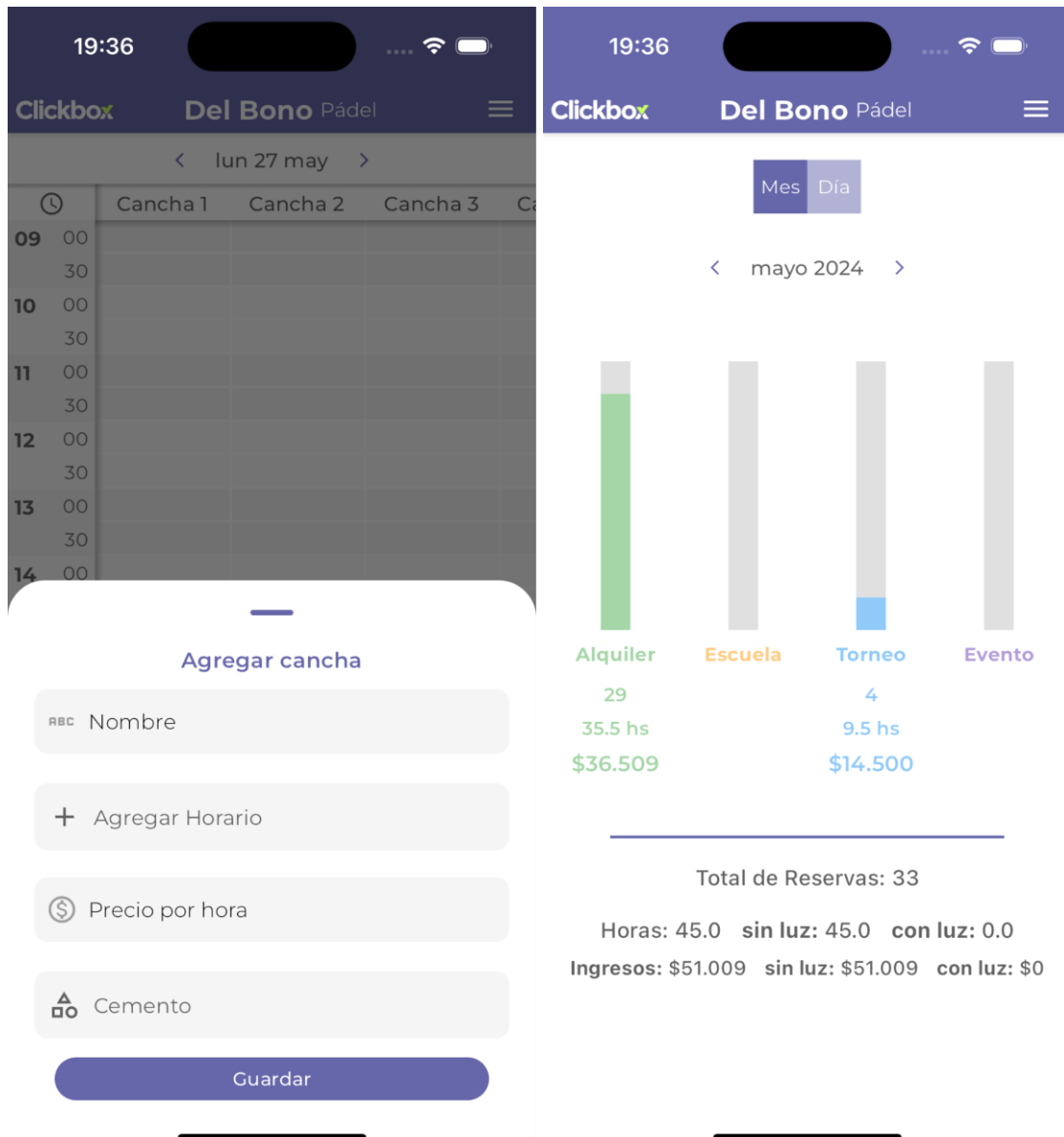


Figura 53. Interfaz de creación de canchas y de estadísticas. Imagen del autor.

La figura 53 muestra la interfaz para agregar una nueva cancha, en el deporte seleccionado, en este caso, pádel. La interfaz de la derecha refiere a las estadísticas para el deporte seleccionado, las cuales se pueden filtrar por mes o día.

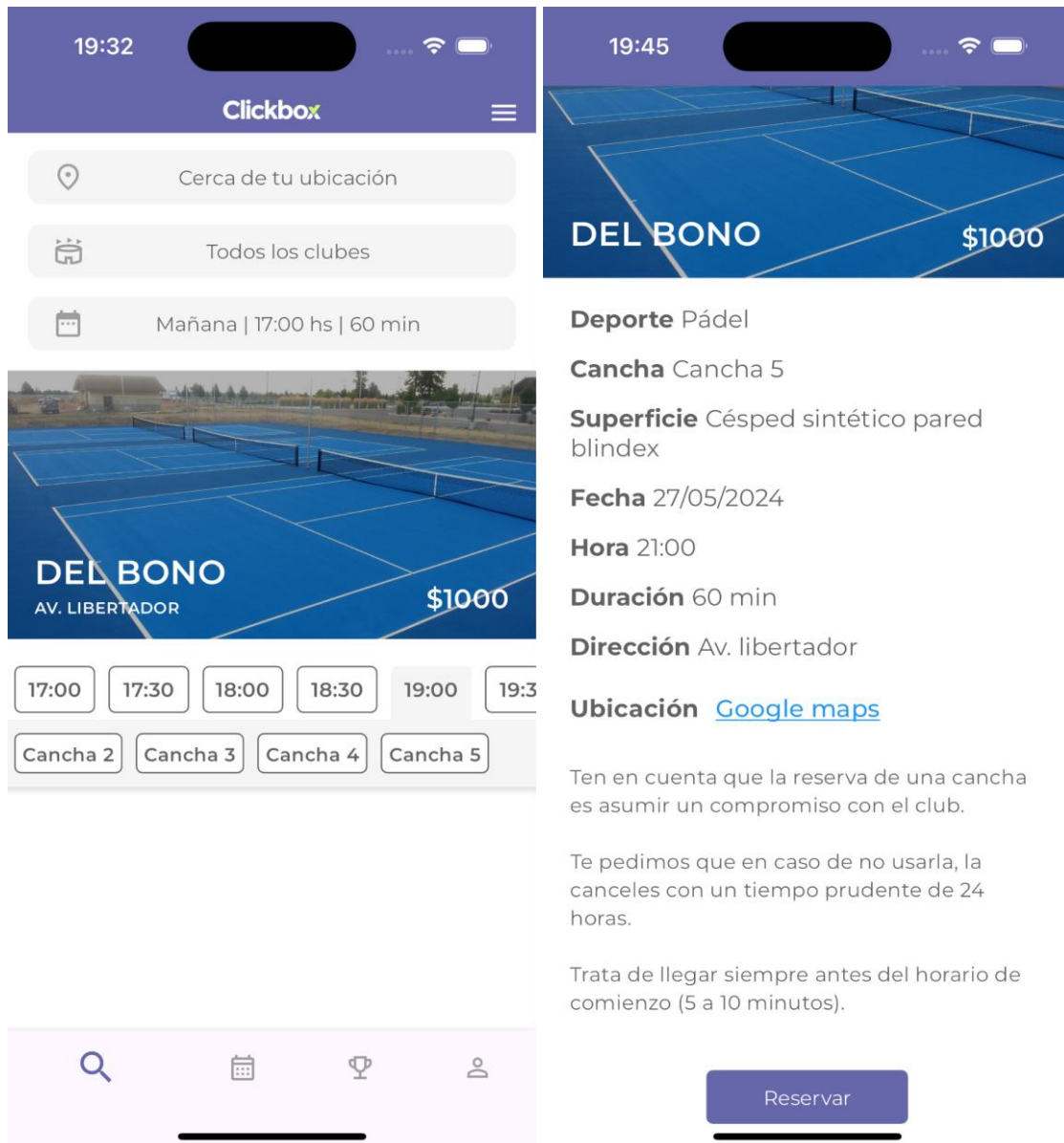


Figura 54. Interfaz de resultados de búsqueda y concretación de reservas por parte de clientes de clubes. Imagen del autor.

La figura 54 muestra la interfaz que despliega los resultados de búsquedas de reservas para un deporte seleccionado. En esta ocasión se están buscando reservas cerca de la ubicación del usuario (funcionalidad agregada post trabajo) en todos los clubes disponibles en el sistema para las 17hs y con duración de 60 minutos.

# **Capítulo 5 - Conclusiones**



En el presente trabajo se planteó como objetivo general desarrollar e implementar un sistema multiplataforma basado en arquitectura Microservicios, BaaS, Cloud Servers y contenedores para la automatización y optimización de reservas deportivas. Para cumplir con este objetivo, se tomó como caso de estudio un complejo deportivo real con el fin de analizar en profundidad los procesos de reservas y las variables involucradas; e identificar las principales dificultades, como así también las posibilidades de automatización de tareas y subprocesos, y la manera de optimizarlos mediante la implementación de sistemas.

Se determinaron varios problemas, como por ejemplo, pasos adicionales innecesarios, uso de herramientas ofimáticas con licencias y no específicas para el proceso de reservas, uso de registros manuales, múltiples canales de administración y toma de reservas, falta de digitalización y procesos repetitivos en el día a día. Se procedió al diseño de tres módulos distintos pero que en conjunto resuelven los problemas mencionados, especialmente por el hecho de evitar pasos innecesarios en el flujo y por la posibilidad de un complejo deportivo de crear reservas (las cuales pueden ser creadas reiteradas veces de manera automática) y recibir reservas por parte de usuarios que desean practicar un deporte en dicho club.

Cumplido el primer objetivo se continuó con un proceso de comparación, análisis y selección de arquitecturas, servicios, herramientas y tecnología adecuada para la elaboración del proyecto. Se utilizó una arquitectura de microservicios teniendo en mente la escalabilidad, Flutter como tecnología Frontend facilitando enormemente el desarrollo multiplataforma, Golang como tecnología Backend para la creación de un microservicio, Firebase como BaaS para facilitar el proceso de autenticación, DonWeb como proveedor Cloud Server y finalmente, Docker para el uso de contenedores y despliegue en la nube.

Una vez realizada la elección, se procedió al desarrollo e implementación apuntando a un ambiente multiplataforma. Esto incluyó tanto la programación de código para el Frontend como para el Backend.

Finalmente se realizó el despliegue del Backend (microservicio de reservas) en un contenedor en la nube para su posterior acceso o uso remoto.

## 5.1 Propuestas de trabajos futuros

Durante el desarrollo de este trabajo se vieron distintos inconvenientes que fueron resueltos con el desarrollo e implementación del sistema de software propuesto. Principalmente simplificando el flujo al realizar una reserva. Sin embargo, existen diversas funcionalidades a implementar que permiten mejorar aún más el proceso de gestión y administración de reservas.

Por ejemplo, un módulo a desarrollar en este sistema puede consistir en otorgar distintos permisos a usuarios de un club para que los mismos tengan diferentes niveles de acceso. Un club, así, podría designar a sus operarios para que únicamente tengan permiso de visualizar la planilla, otros operarios que puedan visualizar y editar u otros que puedan acceder a todas las funcionalidades. Esto enfrenta problemas complejos de actualización a tratar a través de websockets. Por ejemplo si un operario crea una reserva en determinado deporte, cancha y hora, esto se debería ver inmediatamente reflejado en todas las planillas de los operarios que tengan iniciada su sesión en el club, evitando así conflictos. Adicionalmente podría ocasionar conflictos que algún usuario cambie el horario de una cancha, esto se debería refrescar automáticamente en todas las cuentas de los operarios del club.

Otra propuesta es la de resolver un punto crítico de las reservas: los pagos. Se puede realizar alguna integración con una plataforma de pagos, como Apple Pay,

Google Pay, Mercado Pago, entre otras. Pero esto introduce problemáticas nuevas a resolver. Por ejemplo ¿Se debería pagar solo una seña de la reserva o la reserva completa? Se podría optar por ambas posibilidades y que eso lo determine el club. Existen algunas complicaciones mayores, como ¿Qué sucede si el usuario paga y el club tiene que cancelar la reserva, por razones específicas como lluvias intensas? Se debería, entonces, implementar alguna lógica de devolución de dinero que son difíciles de implementar si se tiene en cuenta que la mayoría de plataformas de pagos cobran comisiones, que no dan la posibilidad de ser reembolsadas en muchos casos.

Finalmente una propuesta útil, es la de integrar un chat dentro del sistema, ya sea integrar alguno ya existente o crear uno desde cero. El chat sería de especial utilidad para comunicaciones urgentes y para mantener una única vía de comunicación sin hacer uso de otras aplicaciones, centralizando todo en un único lugar.

# Bibliografía

## Referencias Bibliográficas

[1] Amazon Web Services. (n.d.). ¿Cuál es la diferencia entre la arquitectura monolítica y la de microservicios? Recuperado de [https://aws.amazon.com/about-aws/global-infrastructure/regions\\_az/](https://aws.amazon.com/about-aws/global-infrastructure/regions_az/)

[2] Appvizer. (2022). *Sistemas de reservas online*. Recuperado de <https://www.appvizer.es/revista/relacion-cliente/sistema-reservas/sistema-de-reservas-online-gratuito-de-pago>

[3] Bloc Library. (n.d.). Recuperado de <https://bloclibrary.dev/>

[4] Capurro, R. (2007). Epistemología y ciencia de la información. *Revista Venezolana de Información, Tecnología y Conocimiento*, 11-29.

[5] DonWeb. (2021). *Configuración inicial de un servidor con Ubuntu 20.04*. Recuperado de <https://guias.donweb.com/configuracion-inicial-de-un-servidor-con-ubuntu-20-04/>

[6] DonWeb. (2021). *Cómo instalar un stack LEMP (Linux, Nginx, MySQL y PHP) en Ubuntu 20.04*. Recuperado de <https://guias.donweb.com/como-instalar-un-stack-lemp-linux-nginx-mysql-y-php-en-ubuntu-20-04/>

[7] DonWeb. (2021). *Instalar Docker y crear un contenedor en Ubuntu 20.04*. Recuperado de <https://guias.donweb.com/instalar-docker-y-crear-un-contenedor-en-ubuntu-20-04/>

- [8] Firebase. (n.d.). *Firestore*. Recuperado de <https://firebase.google.com>
- [9] Flutter. (n.d.). *Sitio oficial del framework Flutter*. Recuperado de <https://flutter.dev>
- [10] Flutter Showcase. (2021). *Flutter Showcase*. Recuperado de <https://flutter.dev/showcase>
- [11] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2002). *Fundamentos de bases de datos* (4ta ed.). McGraw-Hill.
- [12] Framework. (n.d.). Cambridge <https://dictionary.cambridge.org/es/diccionario/ingles/framework>
- [13] García-Molina, H., Ullman, J. D., & Widom, J. (2009). *Database Systems: The Complete Book*. Pearson Education.
- [14] Hernández Sampieri, R. (2014). *Metodología de la investigación* (6ta ed.). México: Editorial.
- [15] IBM. (n.d.). What is a Cloud Server? Recuperado de <https://www.ibm.com/cloud/learn/cloud-server>
- [16] Inveritas. (2020). *Flutter vs React Native vs Native: Deep Performance Comparison*. Recuperado de <https://inveritasoft.com/blog/flutter-vs-react-native-vs-native-deep-performance-comparison>

[17] Kantar Global Monitor. (2022). *Encuesta sobre la práctica de deportes*. Recuperado de <https://dossiernet.com/articulo/los-argentinos-y-el-deporte-un-estudio-de-kantar-que-muestra-la-importancia-de-la-actividad-fisica-en-la-vida-de-los-argentinos/30678>

[18] Meter.net. (n.d.). Recuperado de <https://www.meter.net/tools/world-ping-test/>

[19] Ministerio de Turismo y Deporte. (2009 y 2021). *Encuesta sobre la práctica de deportes y actividad física en la Argentina*.

[20] Ministerio de Turismo y Deporte. (2023). *Relevamiento RENACED*.

[21] Newman, S. (2015). *Building Microservices*. O'Reilly Media.

[22] Opensource. (n.d.). What is Docker? Recuperado de <https://opensource.com/resources/what-docker>

[23] Poulton, N. (2018). *Docker Deep Dive*.

[24] Real Academia Española. (n.d.). *Reserva*.

[25] Real Academia Española. (n.d.). *Turno*.

[26] Rocha, H. F. O. (2022). *Practical Event-Driven Microservices Architecture*.

[27] Saplin, M. (n.d.). *Data Table 2*. Recuperado de [https://maxim-saplin.github.io/data\\_table\\_2/#/datatable2](https://maxim-saplin.github.io/data_table_2/#/datatable2)

[28] Secretaría de Deportes de San Juan. (2021). *San Juan tiene el mejor índice de actividad física del país*.

[29] Semrush. (n.d.). *Estadísticas de uso histórico de dispositivos móviles frente a computadoras de escritorio*. Recuperado de <https://www.semrush.com/blog/mobile-vs-desktop-usage/>

[30] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2002). *Fundamentos de bases de datos* (4ta ed.).

[31] Sojay, L. (2023). *Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022*. Statista.

[32] Ylönen, T. (1995). Secure Shell (SSH). IETF Network Working Group, RFC 4251.

[33] Zomko, R. (2023). *Brief Guide To BaaS (Backend as a Service)*. Impressit.