I.E.S LAS SALINAS

PROYECTO FINAL FIN DEGRADO CURSO

24-25

TEAMBOX

CICLO FORMATIVO GRADO SUPERIOR

DESARROLLO DE APLICACIONES

MULTIPLATAFORMA

AUTOR: Sandra Martínez González

TUTOR: Manuel Antonio Benito Mora

Resumen

Este Trabajo de Fin de Grado presenta el desarrollo de una aplicación móvil orientada a la gestión de boxeadores y la conexión entre clubes, promotores y managers dentro del ámbito boxístico. La aplicación permite a los usuarios registrarse según su rol, crear perfiles de boxeadores, gestionar equipos y acceder a una base de datos con funciones de búsqueda y filtrado. Además, se incorpora un sistema de registro dependiente por comunidad y provincia, carga de logos y múltiples contactos por usuario. El desarrollo se ha realizado en Android Studio utilizando Jetpack Compose, y la información se gestiona mediante una API conectada a una base de datos MariaDB. El proyecto busca optimizar la visibilidad y organización de los actores clave en el boxeo amateur, habiendo cabida para el boxeo profesional.

Palabras clave: boxeo, aplicación móvil, gestión deportiva, Jetpack Compose, API

Abstract

The main objective of my final degree project is to develop a mobile application focused on the management of boxers and the connection between clubs, promoters, and managers within the boxing field. The application allows users to register according to their role, create boxer profiles, manage teams, and access a database with search and filtering functions. Additionally, it features a dependent registration system based on region and province, logo uploads, and support for multiple contact numbers per user. The development was carried out in Android Studio using Jetpack Compose, and data is managed through an API connected to a MariaDB database. The project aims to enhance the visibility and organization of key actors in amateur boxing, while also providing space for professional boxing.

Keywords: boxing, mobile application, sports management, Jetpack Compose, API

Índice

Introducción

Este proyecto nace con el objetivo de facilitar la labor de los promotores y organizadores de eventos de deportes de contacto, proporcionándoles una herramienta ágil para contactar y organizar los combates que conformarán una velada. A través de la aplicación, los usuarios podrán acceder de forma inmediata a un directorio actualizado de competidores activos, tanto a nivel local como nacional, con filtros por categoría y peso.

La aplicación está pensada para ser útil tanto en la planificación a largo plazo, permitiendo conocer el panorama deportivo en toda España, como en situaciones urgentes. Por ejemplo, durante los pesajes —que suelen realizarse el día anterior o pocas horas antes del evento— pueden surgir imprevistos como problemas personales o burocráticos que impidan a un boxeador participar. En estos casos, la herramienta permite localizar rápidamente a posibles sustitutos, optimizando así la organización y reduciendo el riesgo de combates cancelados.

Justificación

La gestión de combates y la coordinación entre clubes, promotores y boxeadores es actualmente un proceso poco optimizado, especialmente en el boxeo amateur. Muchas veces depende de contactos informales, redes sociales o grupos de mensajería, lo que genera ineficiencias, falta de transparencia y dificultades para encontrar sustitutos o planificar eventos con antelación.

El desarrollo de esta aplicación responde a la necesidad real de profesionalizar y digitalizar estos procesos. La posibilidad de contar con una base de datos actualizada y accesible desde cualquier dispositivo puede suponer una mejora significativa en la organización de veladas y en la visibilidad de los boxeadores.

Además, este proyecto representa una oportunidad personal para aplicar los conocimientos adquiridos durante el grado en un entorno que combina dos ámbitos especialmente significativos para mí: la tecnología y el deporte. La posibilidad de unir ambos mundos no solo me permite desarrollarme profesionalmente, sino que también me aporta una gran satisfacción personal, al trabajar en algo que realmente me motiva y con lo que me siento plenamente identificada. La solución propuesta busca no solo resolver un problema concreto, sino también aportar valor a la comunidad del boxeo mediante una herramienta útil, escalable y adaptada a sus necesidades reales.

Para validar la necesidad del proyecto, se ha realizado un estudio de mercado mediante una encuesta dirigida a profesionales del sector, cuyos resultados se analizan más adelante.

Objetivos

Objetivo General

Desarrollar una aplicación móvil que facilite la gestión de boxeadores y la conexión entre clubes, promotores y managers, mejorando la organización de veladas y la visibilidad de los deportistas dentro del ámbito del boxeo amateur y profesional.

Objetivos Específicos

- Permitir el registro de usuarios con distintos roles (club, promotor, manager, boxeador).
- Implementar una base de datos con perfiles de boxeadores, incluyendo filtros por peso, categoría y ubicación.
- Desarrollar una funcionalidad de búsqueda eficiente para localizar posibles contrincantes o sustitutos.

- Incorporar un sistema de registro geográfico dependiente por comunidad autónoma y provincia.
- Facilitar la gestión de equipos y la asociación de boxeadores a clubes.
- Diseñar una interfaz de usuario intuitiva y fácil adaptada a dispositivos móviles Android.

Análisis del Mercado y Oportunidad de Negocio

Estudio De Mercado

Con el objetivo de validar la necesidad real de una aplicación que facilite la gestión de boxeadores y la organización de combates, se llevó a cabo un estudio de mercado mediante una encuesta online. Esta encuesta fue distribuida entre diferentes perfiles del ámbito del boxeo amateur y profesional: promotores, entrenadores, boxeadores y organizadores de eventos.

La finalidad de este estudio era conocer de primera mano las dificultades más comunes en la organización de veladas, la forma en que actualmente se gestionan los contactos entre clubes y boxeadores, así como el interés del sector en una herramienta digital como la que se propone en este proyecto.

En total, se recopilaron **62 respuestas**, lo que permitió identificar tendencias claras y necesidades no cubiertas. A continuación, se exponen las principales preguntas, los resultados más relevantes y las conclusiones obtenidas a partir de los datos recogidos.

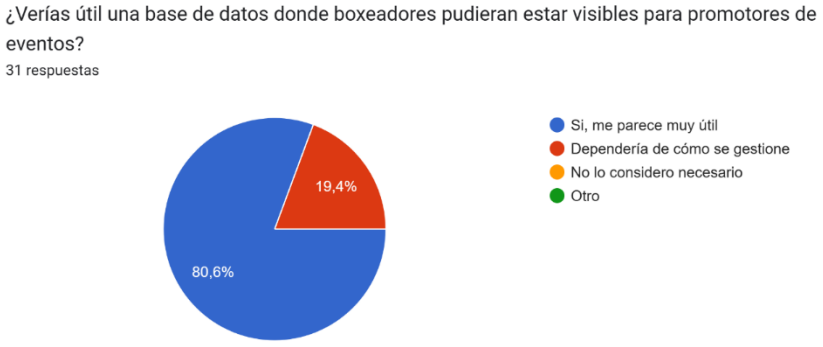


Ilustración 1.Primera pregunta de la encuesta

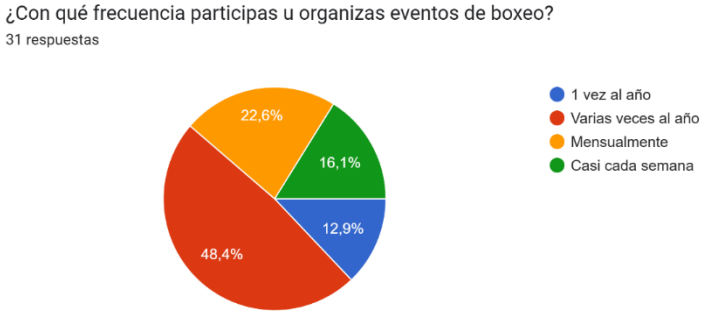


Ilustración 2. Segunda pregunta de la encuesta

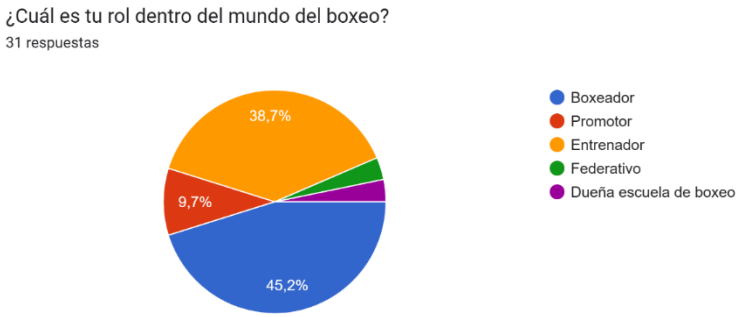


Ilustración 3. Tercera pregunta de la encuesta

¿Te gustaría aparecer o que aparecieran tus competidores en una base de datos visible para promotores?
28 respuestas

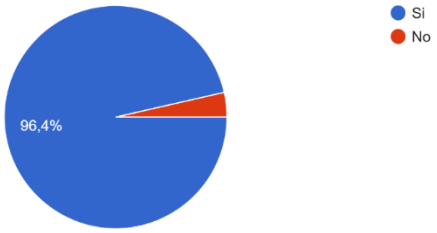


Ilustración 4. Cuarta pregunta de la encuesta

¿Alguna vez se ha caído algún combate y al intentar buscar un nuevo rival te ha resultado algo estresante?
3 respuestas

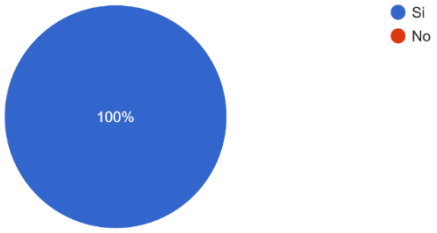


Ilustración 5. Quinta pregunta de la encuesta

¿Sería útil tener a todos los competidores activos en una sola app para poder acceder a ellos rápido y salvar el combate?
3 respuestas

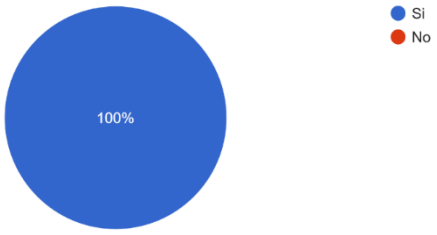


Ilustración 6. Sexta pregunta de la encuesta

Conclusión

Los resultados obtenidos a través de la encuesta realizada a personas relacionadas con el mundo del boxeo han sido altamente reveladores y positivos. La gran mayoría de los encuestados ha manifestado un alto nivel de interés y aprobación hacia la creación de una aplicación como la propuesta en este proyecto.

Este respaldo generalizado evidencia una necesidad real en el sector: la falta de herramientas digitales eficaces que faciliten la conexión entre clubes, boxeadores, promotores y managers. Además, muchos participantes destacaron que una plataforma centralizada con funciones de búsqueda, filtros por categorías y disponibilidad inmediata de competidores les resultaría extremadamente útil tanto para la planificación como para la resolución de imprevistos en eventos.

Estas respuestas refuerzan la validez de la propuesta y confirman que existe un público objetivo dispuesto a utilizar la aplicación, lo que aumenta su viabilidad y justifica plenamente su desarrollo. En definitiva, el estudio de mercado no solo apoya la iniciativa, sino que evidencia el entusiasmo con el que sería recibida por su comunidad potencial de usuarios.

Análisis Dafo

A continuación, un análisis DAFO, o también conocido como FODA de mi empresa/app

FORTALEZAS

- Aplicación especializada y enfocada en un nicho concreto: el boxeo.
- Uso de tecnologías modernas (Jetpack Compose, FastAPI, MariaDB).
- Arquitectura modular con separación clara entre frontend, backend y base de datos.
- Gestión avanzada de boxeadores, con funcionalidades específicas como favoritos y filtrado detallado.

- Interfaz intuitiva, adaptable a diferentes perfiles (club, promotor, mixto, boxeador).
- Despliegue económico en Raspberry Pi, facilitando independencia y bajo coste.
- Automatización de procesos (asignación de categoría, validación de datos).
- Fotografía en base64 para simplificar almacenamiento en base de datos.

DEBILIDADES

- Proyecto desarrollado inicialmente por una sola persona (dependencia técnica alta).
- No cuenta con una base sólida de usuarios inicial (dificultad en fase de lanzamiento).
- Requiere conocimientos técnicos para desplegar en servidores externos o Raspberry Pi.
- El uso de tecnologías modernas puede dificultar el mantenimiento a largo plazo si no se documenta bien.
- Ausencia, por el momento, de funcionalidades como mensajería, gestión de combates o inscripciones a eventos.

OPORTUNIDADES

- Creciente digitalización del deporte, especialmente en disciplinas minoritarias como el boxeo.
- Falta de herramientas específicas de gestión para clubes y promotores de boxeo: nicho poco cubierto.
- Posibilidad de escalado: integración con federaciones, rankings oficiales, o apps de entrenamiento.
- Interés de promotores y clubes por encontrar talentos a través de apps: modelo de negocio atractivo.

- Potencial para monetización: versión premium, publicidad deportiva, o suscripciones por club.

AMENAZAS

- Aparición de competidores con mayores recursos o alianzas con federaciones.
- Reticencia del público objetivo (clubes tradicionales) a adoptar nuevas tecnologías.
- Cambios en regulaciones sobre protección de datos personales (fotos, datos sensibles).
- Posibles dificultades para conseguir visibilidad sin una campaña de marketing efectiva.
- Riesgo de que la app quede obsoleta si no se actualiza regularmente o no se adapta a nuevas demandas del mercado.

Público Objetivo

El público objetivo de TeamBox está conformado por los distintos roles que participan activamente en el mundo del boxeo, sobre todo a nivel amateur en España. La App esta diseñada para satisfacer las necesidades de gestión, promoción y visualización del colectivo de boxeadores en activo, adaptándose a distintos perfiles.

- **Clubes de boxeo:** Para gestionar sus boxeadores, mejorar su visibilidad y digitalizar procesos internos.
- **Promotores:** Para buscar y filtrar boxeadores según criterios específicos, y organizar eventos o combates.
- **Boxeadores:** Para promocionarse y ser descubiertos por clubes o promotores, especialmente si no tienen club fijo.
- **Federaciones deportivas (a medio plazo):** Para facilitar la organización, control y seguimiento del talento boxístico de forma estructurada.

TEAMBOX se enfoca en ofrecer herramientas prácticas, intuitivas y adaptadas a las necesidades de cada perfil, promoviendo la digitalización del sector boxístico.

Modelo de Negocio

Propuesta de valor

TEAMBOX es una aplicación móvil que digitaliza la gestión y promoción de boxeadores amateur y semiprofesionales, conectando clubes, promotores y deportistas a través de una plataforma intuitiva y especializada. Ofrece herramientas para registrar boxeadores, filtrar por características, marcar favoritos, y facilitar el contacto entre actores del sector boxístico.

Canales de distribución

- Google Play Store (Android).
- Promoción directa en clubes deportivos y federaciones.
- Redes sociales (Instagram, Facebook, TikTok) con contenido especializado.
- Página web informativa y de contacto.
- Eventos deportivos o torneos locales.

Fuentes de ingresos

- Publicidad segmentada dentro de la app app.
- Posibles acuerdos con federaciones u organizadores de eventos.

Recursos clave

- Infraestructura tecnológica: servidor (Raspberry Pi), base de datos MariaDB, backend con FastAPI y app en Jetpack Compose.
- Conocimiento técnico del desarrollador.
- Base de datos de boxeadores y usuarios.
- Relación con entidades deportivas y primeros usuarios.

Actividades clave

- Desarrollo y mantenimiento de la aplicación móvil.
- Captación de clubes, promotores y boxeadores.
- Mejora continua de funcionalidades y validación de requisitos.
- Marketing digital y posicionamiento en redes.

Socios clave

- Clubes de boxeo que actúan como primeros usuarios.
- Promotores y entrenadores interesados en digitalizar procesos.
- Posibles acuerdos con federaciones deportivas.
- Plataformas tecnológicas (Google Play, DuckDNS, GitHub).

Estructura de costes

- Infraestructura técnica (servidor, dominio, certificados SSL si se implementan).
- Costes de desarrollo (tiempo del desarrollador).
- Mantenimiento y actualizaciones.
- Posible inversión en marketing digital.
- Costes legales y de protección de datos en fases avanzadas.

Estrategia De Marketing

La estrategia de marketing de TEAMBOX se enfoca en la captación de usuarios clave del sector boxístico, principalmente clubes y promotores, mediante una combinación de canales digitales y promoción directa en eventos deportivos.

- **Estrategia digital**

1. Redes sociales: Promoción de la app en Instagram, TikTok y Facebook con contenido visual atractivo, historias de boxeadores y demostraciones de uso.

2. Web informativa: Landing page con descripción del servicio, captación de leads y enlaces de descarga.
3. SEO y ASO: Optimización para posicionamiento en buscadores y tiendas de apps.
4. Colaboraciones: Contacto con influencers del ámbito del boxeo amateur y asociaciones deportivas.

- **Estrategia directa y alianzas**

1. Contacto con clubes de boxeo locales para pruebas piloto.
2. Presentación de la app en torneos o competiciones regionales.
3. Posibles acuerdos con federaciones para difusión y uso institucional.

- **Estrategia de retención**

1. Actualizaciones periódicas con nuevas funciones.
2. Comunicación continua con los usuarios para recibir feedback.
3. Incentivos para los primeros usuarios (clubes destacados o verificados).

Viabilidad económica

El modelo es viable gracias a los bajos costes de infraestructura y al desarrollo propio. La escalabilidad dependerá de la captación de usuarios y alianzas estratégicas. La app tiene potencial de crecimiento en el ámbito deportivo y puede adaptarse a otros deportes en el futuro.

Estado Del Arte

Actualmente, la gestión de boxeadores y la organización de veladas se realiza principalmente a través de métodos informales, como redes sociales, grupos de WhatsApp o contactos personales. Aunque existen algunas plataformas centradas en la gestión de competidores para deportes de combate, como https://boxrec.com (que se centra más en boxeo

profesional) muchas están orientadas a federaciones oficiales o requieren suscripciones costosas, lo que dificulta su adopción a nivel amateur.

No se ha identificado en el mercado una aplicación gratuita, centrada en el ámbito del boxeo amateur español, que permita registrar perfiles por rol, buscar rivales por filtros detallados y actuar con inmediatez ante bajas de última hora.

Esto evidencia la falta de una solución ágil, accesible y adaptada a las necesidades específicas del sector, especialmente en el contexto nacional.

Herramientas Y Tecnologías Utilizadas

Para el desarrollo del proyecto se han utilizado herramientas y tecnologías modernas orientadas a la creación de aplicaciones móviles multiplataforma:

- **Android Studio:** entorno de desarrollo oficial para aplicaciones Android.
- Jetpack Compose: framework de UI declarativa para construir interfaces modernas, intuitivas y reactivas.
- **Kotlin:** lenguaje principal de programación usado en el desarrollo móvil.
- **API REST:** utilizada para la comunicación entre la app y el servidor backend.
- **MariaDB:** sistema de gestión de bases de datos relacional, utilizado para almacenar la información de los usuarios y boxeadores.
- **GitHub:** sistema de control de versiones para la gestión del código fuente.

Estas tecnologías permiten una arquitectura escalable, mantenible y centrada en la experiencia de usuario, además de facilitar futuras ampliaciones o mejoras en la aplicación.

Análisis De Requisitos(REVISAR)

Requisitos Funcionales

- El sistema debe permitir el registro de usuarios diferenciando su rol (club, promotor, múltiple).
- La aplicación debe permitir la creación y edición de perfiles de boxeadores dentro del perfil Club y Múltiple.
- Los clubes deben poder gestionar un equipo de boxeadores.
- El sistema debe incluir un buscador de boxeadores con filtros por peso, categoría, ubicación, etc.
- La app debe permitir la carga de logotipos e imágenes de perfil.
- Los formularios de registro deben mostrar provincias en función de la comunidad autónoma seleccionada.
- Cada usuario debe poder introducir múltiples teléfonos de contacto.
- Los promotores deben poder acceder fácilmente a una base de datos de boxeadores en activo.

Requisitos No Funcionales

- La aplicación debe estar desarrollada para dispositivos Android.
- La interfaz debe ser intuitiva y de fácil navegación para usuarios no expertos.
- La app debe ofrecer un rendimiento fluido incluso con una base de datos extensa.
- La arquitectura debe ser escalable para futuras mejoras y ampliaciones.
- Se debe utilizar una API REST para la comunicación con el servidor.
- La base de datos debe permitir consultas rápidas y mantener la integridad de la información.

Diseño y Desarrollo

Arquitectura General De La Aplicación

La aplicación ha sido desarrollada bajo una arquitectura cliente-servidor. El cliente es una app móvil creada con Android Studio y Jetpack Compose, mientras que el servidor se basa en una API REST que comunica con una base de datos MariaDB. Esta arquitectura permite escalabilidad y separación clara entre la interfaz de usuario y la lógica de negocio.

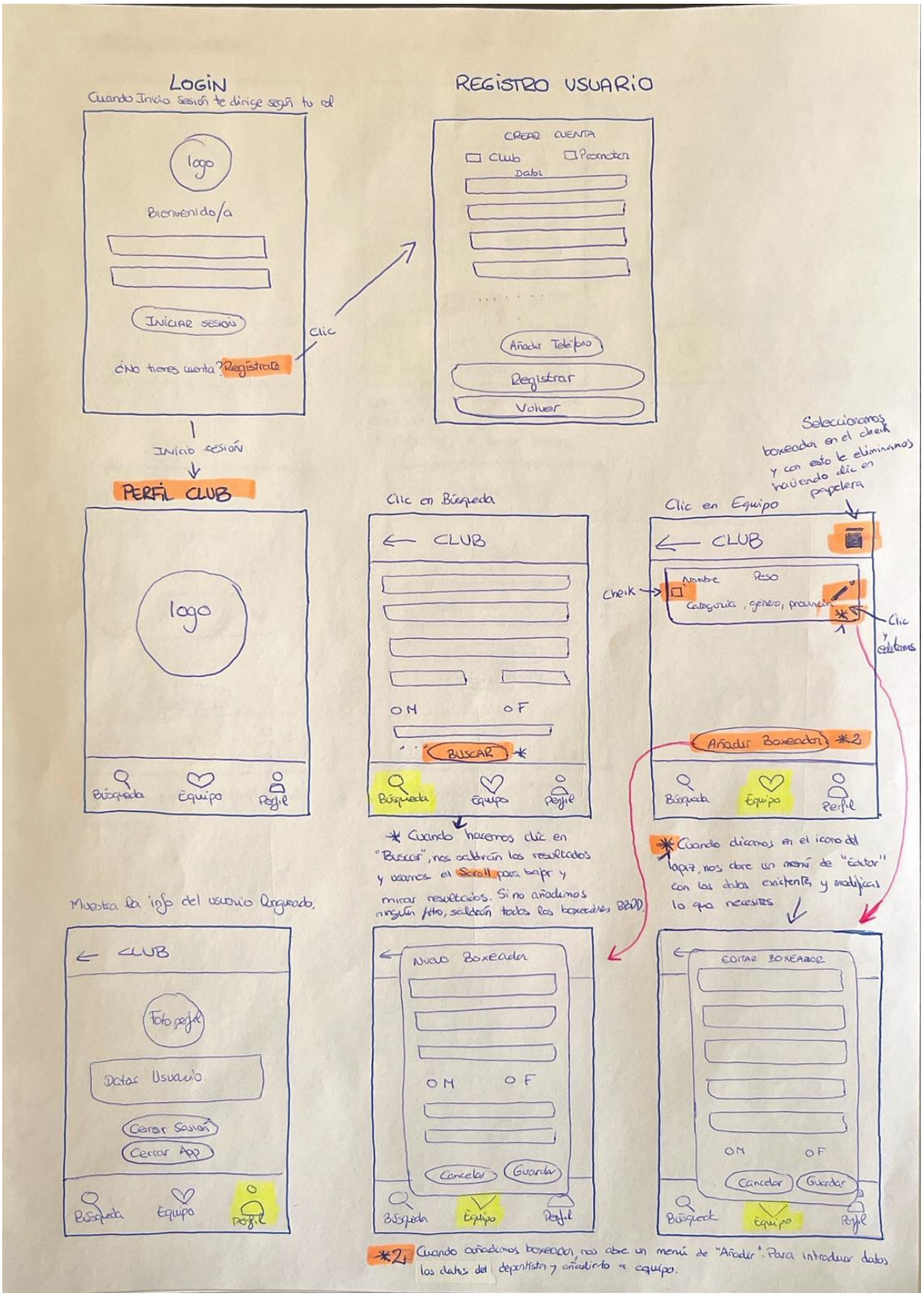
Diseño De La Interfaz De Usuario (UI)

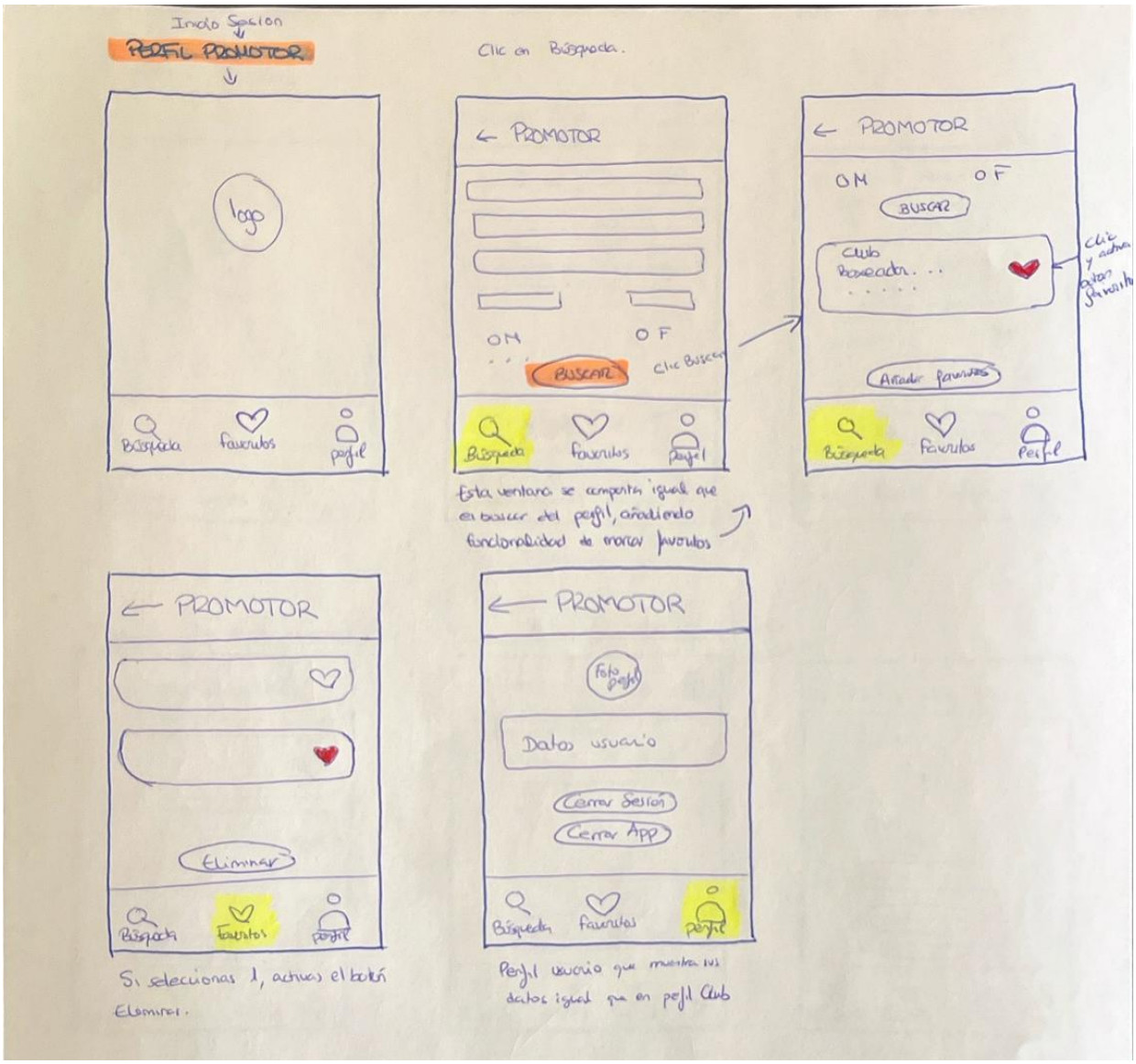
El diseño de la interfaz se ha centrado en la simplicidad y la usabilidad. Se han aplicado principios de diseño centrado en el usuario, con pantallas claras, navegación intuitiva y componentes reutilizables mediante Jetpack Compose. Las principales pantallas incluyen:

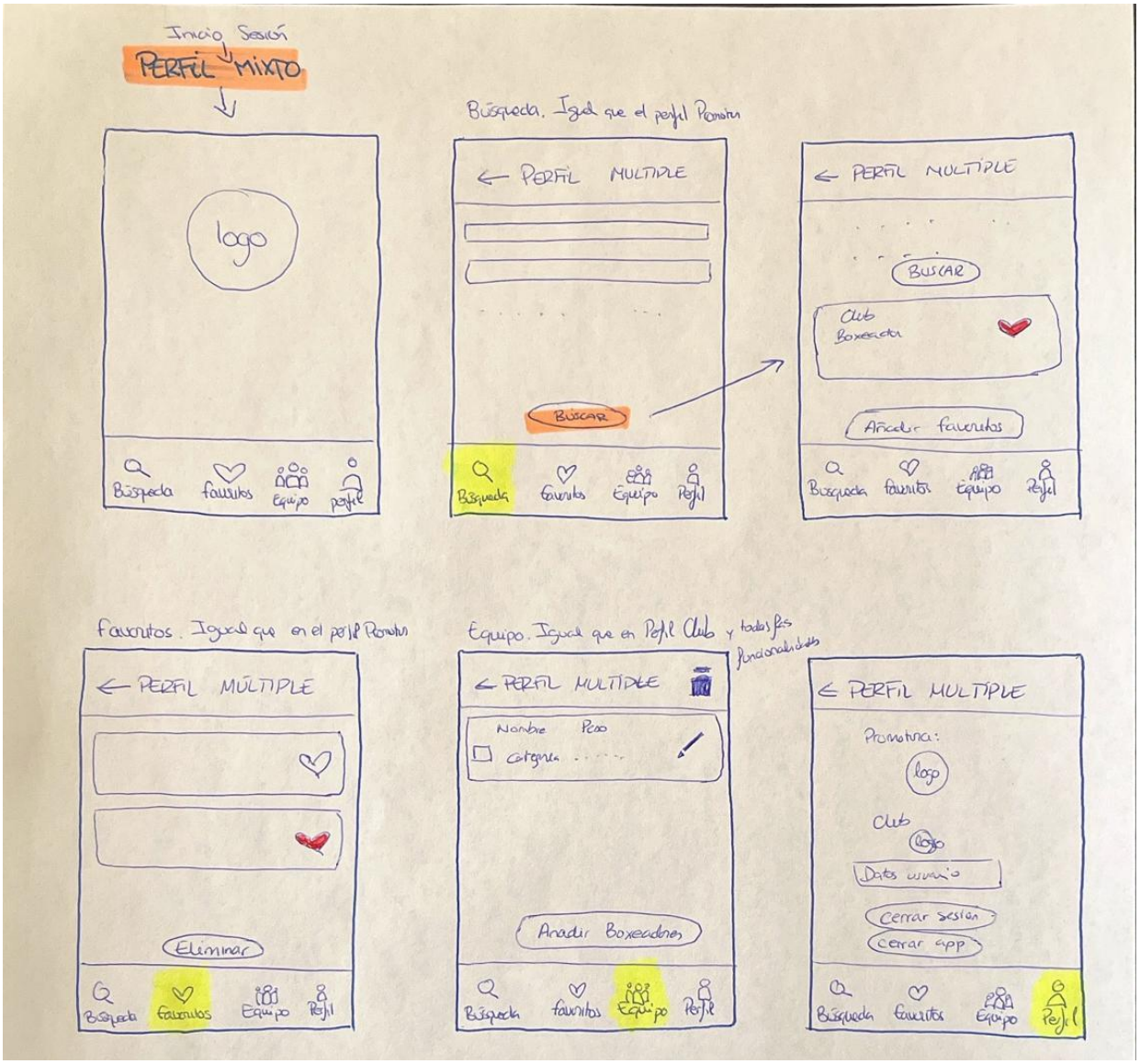
- Registro y login con distinción de roles.
- Menú inferior con acceso a: Cuenta, Equipo, y Búsqueda.
- Formulario dinámico de registro con carga de imagen, campos condicionales y selección de comunidad/provincia.
- Buscador de boxeadores con filtros avanzados.

Bocetos (Sketch)

Antes de comenzar con la implementación, se realizó un diseño preliminar de las pantallas principales de la aplicación. Estos bocetos permitieron planificar la experiencia de usuario y definir la disposición de los elementos clave en cada vista.


LOGIN
Cuando Inicio sesión te dirige según tu rol
logo
Bienvenido/a
INICIAR SESIÓN
¿No tienes cuenta? Regístrate
Clic
REGISTRO USUARIO
CREAR CUENTA
Club
Promotor
Datos
Añadir Teléfono
Registrar
Volver
INICIO SESIÓN
PERFIL CLUB
logo
Búsqueda
Equipo
Perfil
Clic en Búsqueda
CLUB
M
F
BUSCAR
* Cuando hacemos clic en "Buscar", nos saldrán los resultados y usamos el Scroll para bajar y mirar resultados. Si no añadimos ningún filtro, saldrán todos los boxeadores BBDD.
Clic en Equipo
Seleccionamos boxeador en el check y con esto le eliminamos haciendo clic en papelera
Nombre
Peso
Categoría, género, provincia
check
Clic y editamos
Añadir Boxeador *2
*1 Cuando clicamos en el icono del lapiz, nos abre un menú de "Editar" con los datos existente, y modificas lo que necesites
Muestra la info del usuario logueado.
Foto perfil
Datos Usuario
Cerrar Sesión
Cerrar App
NUEVO BOXEADOR
Cancelar
Guardar
EDITAR BOXEADOR
*2 Cuando añadimos boxeador, nos abre un menú de "Añadir". Para introducir datos los datos del deportista y añadirlo a equipo.


Inicio Sesion
PERFIL PROMOTOR
logo
Búsqueda
favoritos
perfil
Clic en Búsqueda.
← PROMOTOR
OM
OF
BUSCAR
Clic Buscar
Búsqueda
favoritos
Perfil
Esta ventana se comporta igual que en buscar del perfil, añadiendo funcionalidad de añadir favoritos
← PROMOTOR
OM
OF
BUSCAR
Club
Boxeador...
Añadir favoritos
Búsqueda
Favoritos
Perfil
Clic y activa datos favoritos
← PROMOTOR
Eliminar
Búsqueda
favoritos
perfil
Si seleccionas 1, activas el botón Eliminar.
← PROMOTOR
Foto perfil
Datos usuario
Cerrar Sesión
Cerrar App
Búsqueda
favoritos
perfil
Perfil usuario que muestra sus datos igual que en perfil Club



Gestión De Datos y Backend

La comunicación con la base de datos se realiza a través de una API RESTful. La base de datos MariaDB contiene tablas separadas para usuarios, boxeadores, favoritos y relaciones entre ellos. La API permite realizar operaciones CRUD (crear, leer, actualizar, eliminar) de forma segura y eficiente.

Funcionalidades Implementadas

App General

- Registro de usuario con múltiples roles (club, promotor, mixto).
- Inclusión de múltiples teléfonos de contacto.

Perfil Club

- Creación de perfiles de boxeadores.
- Tabla del listado del equipo.
- Búsqueda de boxeadores por filtros de peso, comunidad, provincia, etc.

Perfil Promotor

- Búsqueda de boxeadores por filtros de peso, comunidad, provincia, etc.
- Agregar boxeadores a favoritos.

Perfil Mixto

- Creación de perfiles de boxeadores.
- Tabla del listado del equipo.
- Búsqueda de boxeadores por filtros de peso, comunidad, provincia, etc.
- Agregar boxeadores a favoritos para futuros eventos.

Gestión De Errores Y Validaciones

Se han implementado validaciones tanto en la app como en el servidor para garantizar la integridad de los datos. Se controla:

- Formatos de email y teléfono.
- Campos obligatorios.
- Respuestas del servidor ante errores (por ejemplo, usuarios duplicados).

Entorno De Despliegue Y Servidor

Para evitar el uso de soluciones de terceros y tener un mayor control sobre el backend del proyecto, se optó por implementar la base de datos y la API en un servidor propio, compartido con un compañero, con quien ya teníamos experiencia en este tipo de configuraciones. Aunque esto supuso una mayor complejidad inicial en el diseño y la estructura de la base de datos, a largo plazo resultó ser una solución más flexible y ajustada a nuestras necesidades.

El servidor utilizado es una Raspberry Pi 4, un microordenador de bajo consumo energético y mínimo coste operativo. Este dispositivo resulta ideal para proyectos personales o académicos, ya que requiere únicamente una tarjeta microSD para su funcionamiento, y puede mantenerse encendido de forma continua con un consumo muy reducido.

Para la configuración del entorno se utilizaron las siguientes tecnologías:

- **SSH:** para el acceso remoto y la gestión segura del servidor.
- **Apache:** como servidor web.
- **MariaDB:** sistema gestor de bases de datos relacional utilizado para la creación y gestión de la base de datos del proyecto.
- **FastAPI:** framework de Python utilizado para construir la API REST que conecta la base de datos con la aplicación móvil.
- **Entorno virtual de Python:** creado para aislar dependencias y facilitar el despliegue de la API.

Toda la lógica del backend se basa en sentencias SQL, ejecutadas a través de FastAPI, que se encarga de recibir las peticiones desde la app, interactuar con la base de datos MariaDB y devolver los datos solicitados.

Instalación y Configuración Del Servidor

Raspberry Pi

Instalación S.O.

Para el despliegue del servidor se utilizó una Raspberry Pi 400, una solución económica, de bajo consumo energético y con la potencia suficiente para desempeñar las funciones requeridas.

El primer paso consistió en la preparación de la tarjeta microSD, en este caso de 64 GB, que garantiza el espacio necesario para el sistema operativo (aproximadamente 4 GB) y margen adicional para el resto de servicios.

Para instalar el sistema operativo se utilizó la herramienta Raspberry Pi Imager.

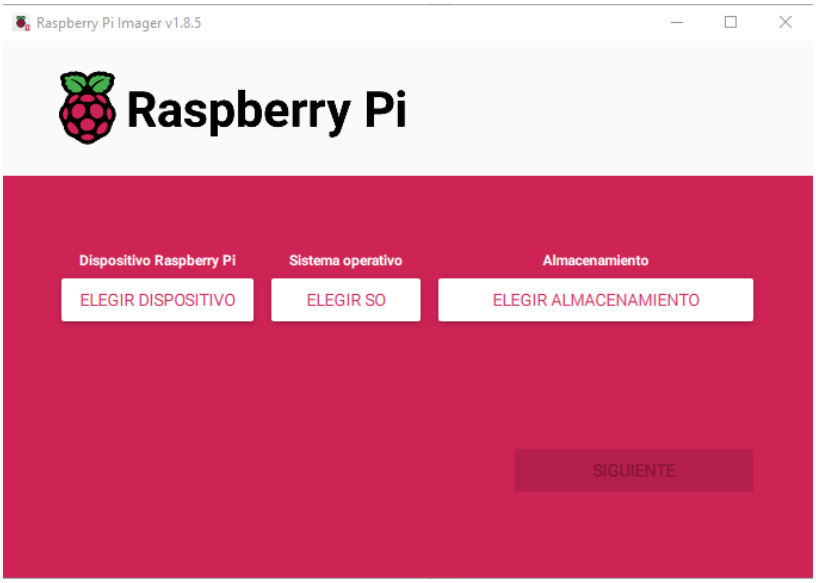


Ilustración 7. Instalador Raspberry Pi (I)

La interfaz ofrece tres pasos principales. En primer lugar, se selecciona el dispositivo, en este caso la Raspberry Pi 4, compatible con el modelo Raspberry Pi 400.

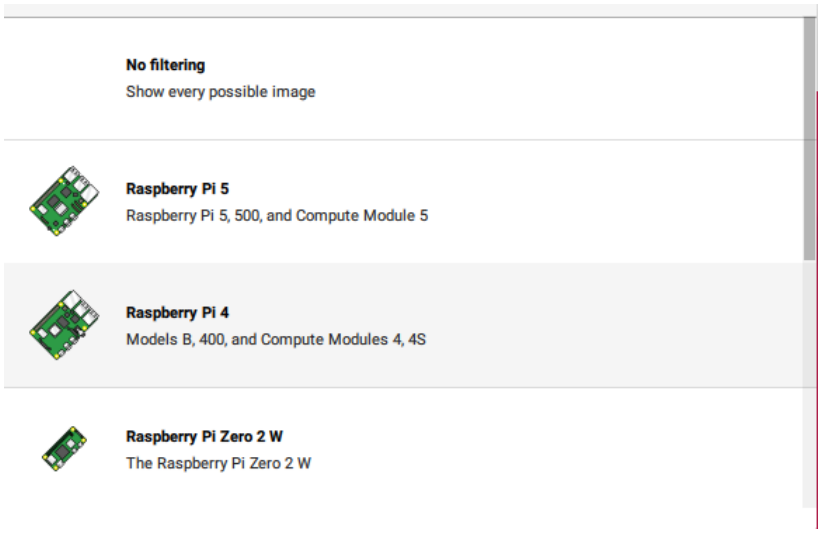


Ilustración 8. Instalador Raspberry Pi (II)

A continuación, en la sección de sistema operativo, se eligió Raspberry Pi OS Lite (64-bit), una versión sin entorno gráfico, más ligera y eficiente para su uso como servidor.

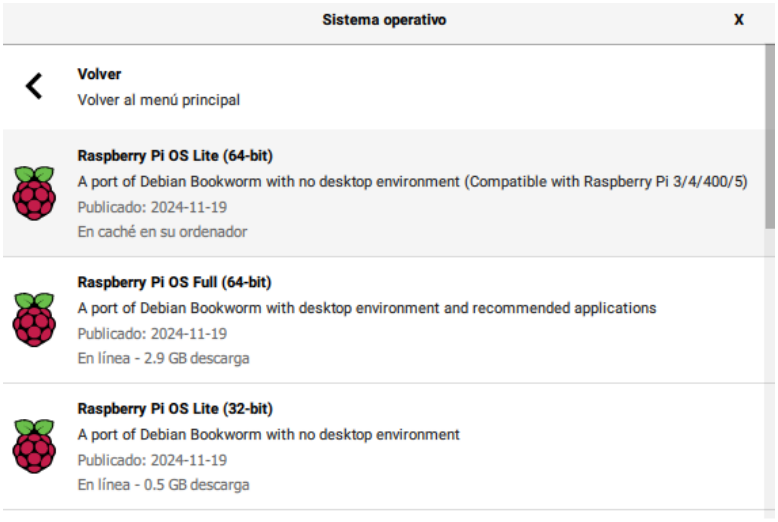


Ilustración 9. Instalador Raspberry Pi (III)

Posteriormente, se seleccionó la unidad de almacenamiento (la tarjeta SD). En los ajustes avanzados que ofrece la herramienta se configuraron varios parámetros clave: el nombre del

host, el usuario y contraseña, las credenciales de red WiFi, y la región, que inicialmente aparecía como "GB" y se ajustó a "ES" (España). También se activó el servicio SSH, esencial para la administración remota del servidor.

Ilustración 10. Instalador Raspberry Pi (IV)

Aunque las credenciales de red ya se introdujeron en el asistente, fue necesario crear manualmente el archivo `wpa_supplicant.conf`, que se añadió en el directorio raíz de la tarjeta SD. Este archivo contiene la configuración de la red WiFi para asegurar la conexión desde el primer arranque.

wpa_supplicant.conf

```
1  country=ES
2  ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
3  update_config=1
4
5  network={
6  scan_ssid=1
7  ssid="Livebox6-1461"
8  psk="contraseña"
9  }
```

Ilustración 11. Archivo Wifi SD Raspberry Pi

Una vez finalizada la configuración, se insertó la tarjeta SD en la Raspberry Pi y se encendió el dispositivo. Para comprobar su conexión a la red local, se utilizó la herramienta Advanced IP Scanner, que permite visualizar los dispositivos conectados.

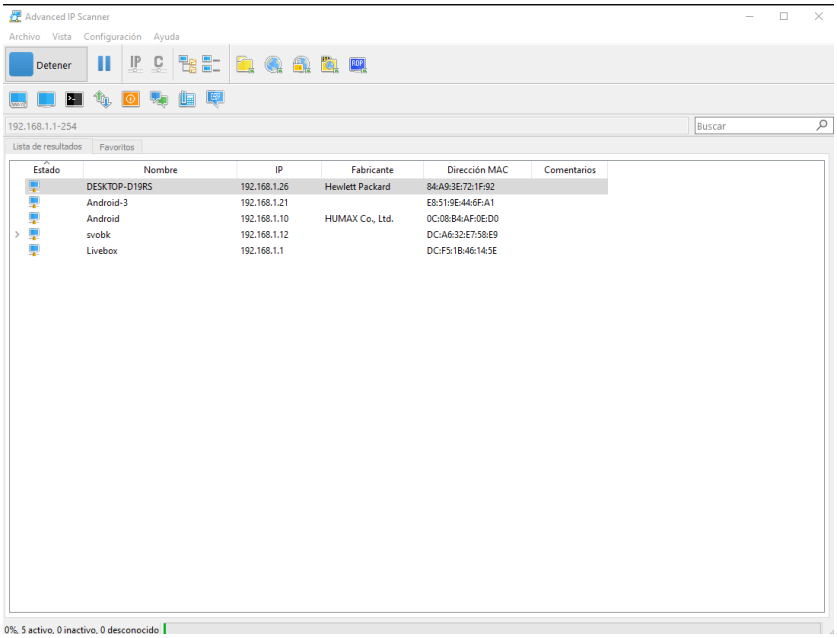


Ilustración 12. Escaneo de red

Con la dirección IP localizada, se estableció conexión con el servidor mediante PuTTY, una herramienta de acceso remoto vía SSH. Para mayor comodidad, se guardaron las credenciales de acceso y más adelante se configuró una IP estática.

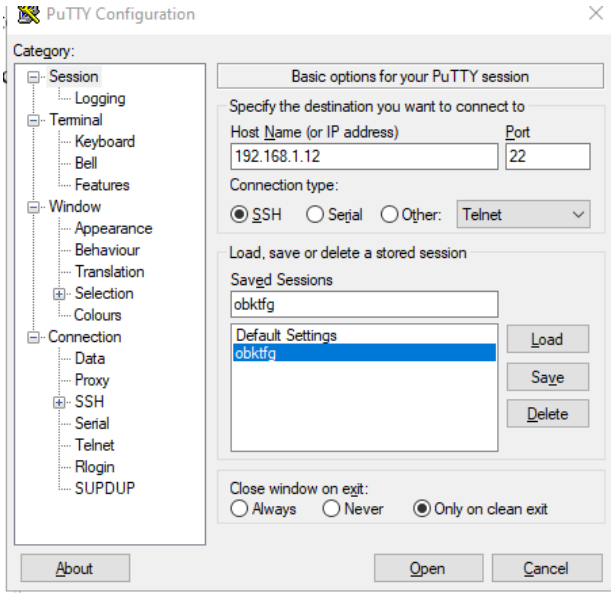


Ilustración 13. Uso de PuTTY

Instalación Y Configuración de Software Usado En El Servidor

Python Y MariaDB

Comandos necesarios para la instalación:

Python:

`sudo apt install python3`

MariaDB:

`sudo apt install mariadb-server`

Tras la instalación inicio sesión en MariaDB:

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

Tras tener el control de MariaDB desde root, se crea el usuario Sandra:

```
CREATE USER 'sandra'@'%' IDENTIFIED BY 'contraseña;
```

La forma normal de crear un usuario en localhost sería esta, que también la realizo por si me hiciera falta:

```
CREATE USER 'sandra'@'localhost' IDENTIFIED BY 'contraseña;
```

Pero yo necesito también la otra, ya que, para poder acceder desde todos los lugares, estando el servidor en otra red como es mi caso, necesito ese permiso.

Creo su base de datos:

```
CREATE DATABASE teambox;
```

Le doy permisos a Sandra (ambas opciones) para que pueda modificar lo que considere y necesite en la BBDD TeamBox:

```
GRANT ALL PRIVILEGES ON teambox.* TO 'sandra'@'%';
GRANT ALL PRIVILEGES ON teambox.* TO 'sandra'@'localhost';
```

Una vez creada la BBDD, tenemos que tener en cuenta que tablas van a ser necesarias en nuestro proyecto, cuanto mas claro lo tengas desde el principio, mejor, ya que si tienes que realizar muchos cambios puedes encontrarte con contradicciones y tener problemas a la hora de la comunicación entre la BBDD y Android Studio.

Para TeamBox, son necesarias las siguientes tablas:

- **Administradores**: siempre será necesario tener administradores para en un futuro implementar poder modificar cualquier dato desde ese tipo de cuenta

```
CREATE TABLE administradores (
    id INT(11) AUTO_INCREMENT PRIMARY KEY,
    nombre_usuario VARCHAR(50) NOT NULL,
    contrasena VARCHAR(255) NOT NULL,
    nombre_completo VARCHAR(100),
    correo_electronico VARCHAR(100),
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ultima_conexion DATETIME
);
```

- **usuarios_app**: los clasifica según su rol y determina a que tipo de cuenta accede, también guarda sus datos.

```
CREATE TABLE usuarios_app (
    id_usuario INT(11) AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100),
    apellido VARCHAR(100),
    email VARCHAR(100) NOT NULL UNIQUE,
    contrasena VARCHAR(255) NOT NULL,
    es_club TINYINT(4) DEFAULT 0,
    es_promotor TINYINT(4) DEFAULT 0,
    es_boxeador TINYINT(4) DEFAULT 0,
    nombre_club VARCHAR(100),
    logo_club MEDIUMTEXT,
    nombre_promotora VARCHAR(100),
    logo_promotora MEDIUMTEXT,
    foto_perfil MEDIUMTEXT,
    comunidad VARCHAR(100),
    provincia VARCHAR(100),
    telefono1 VARCHAR(20),
    telefono2 VARCHAR(20),
    telefono3 VARCHAR(20),
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **boxeadores**: guardará todos los datos y es la que recibe la búsqueda principal en la app.

```
CREATE TABLE boxeadores (
    Id_boxeador INT(11) AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    fecha_nacimiento DATE NOT NULL,
```

```
    dni_boxeador VARCHAR(9) NOT NULL UNIQUE,
    genero TINYINT(1) NOT NULL,
    peso DECIMAL(5,2) NOT NULL,
    categoria VARCHAR(50),
    foto_perfil TEXT,
    comunidad VARCHAR(100),
    provincia VARCHAR(100),
    club_id INT(11),
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (club_id) REFERENCES usuarios_app(id_usuario)
        ON DELETE SET NULL
        ON UPDATE CASCADE
);
```

- **favoritos**: ayuda a organizar al rol promotor los boxeadores que existen.

```
CREATE TABLE favoritos (
    id_favorito INT(11) AUTO_INCREMENT PRIMARY KEY,
    club_id INT(11) NOT NULL,
    boxeador_id INT(11) NOT NULL,
    fecha_agregado TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE (club_id, boxeador_id),
    FOREIGN KEY (club_id) REFERENCES usuarios_app(id_usuario)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (boxeador_id) REFERENCES boxeadores(Id_boxeador)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Todas las tablas tienen en común en que el ID es autoincremental y Clave primaria.

Si se quisiera añadir algún dato que nos falta en alguna tabla, podrías usar el siguiente comando, a mí me pasó cuando añadí dni_boxeador ya que al principio no lo veía necesario teniendo una clave primaria identificativa pero también necesitaba un identificador único para que no se duplicaran boxeadores.

```
ALTER TABLE boxeadores ADD COLUMN dni_boxeador VARCHAR(9) NOT NULL UNIQUE;
```

MySQL Workbench

Para poder usar MySQL Workbench desde fuera de la red, hay que abrir el puerto 3306, una vez abierto configuro mi MySQL Workbench para tener acceso visual y que no todo tenga que ser por terminal, ya que me parece mas cómodo a través de este software.

Configuro la conexión, la IP público, el puerto por el que da acceso y el usuario.

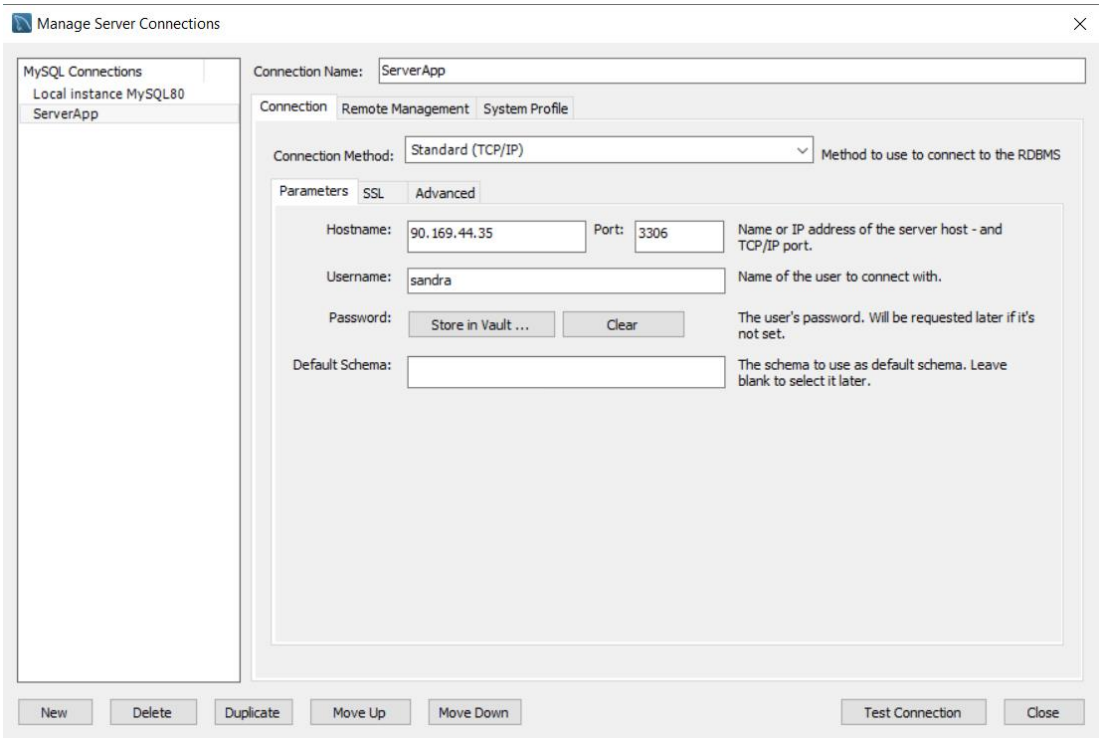


Ilustración 14. Configuración MySQL Workbench

Aquí ya tendría un acceso directo a mi BBDD

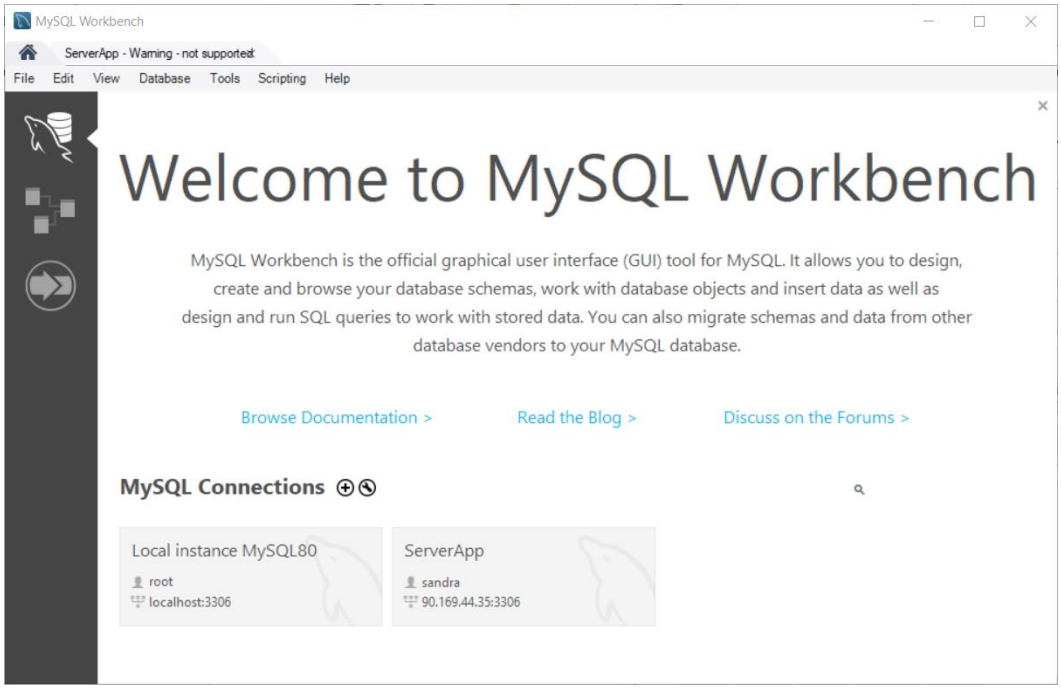


Ilustración 15.Pantalla para acceder a la BBDD

FastAPI

Instalación de FastAPI para poder crear endpoint y controlar la BBDD (añadir, borrar y editar datos) sin tener que acceder de manera directa y así evitar riesgos ya que el servidor esta abierto para poder acceder desde fuera.

Creamos un entorno virtual:

```
python -m venv .venv
```

Una vez creado tendremos la siguiente carpeta:

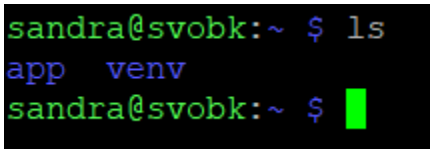


Ilustración 16. PuTTY comandos (I)

En la carpeta app es donde alojaré los .py de la API

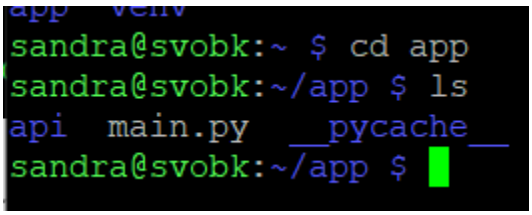


Ilustración 17. PuTTY comandos (II)

Una vez creado el entorno para activarlo necesitamos estar en el directorio raíz de venv que sería aquí:

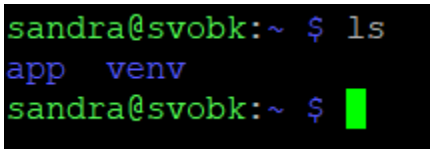


Ilustración 18. PuTTY comandos (III)

Una vez ahí, usaremos el siguiente comando para activarlo:

```
source venv/bin/activate
```

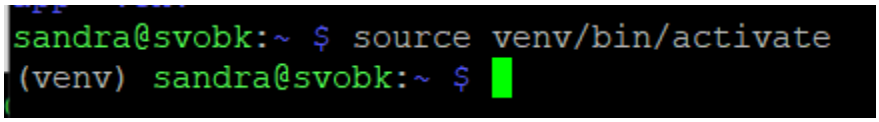


Ilustración 19PuTTY comandos (IV)

Se instala FastApi:

```
pip install fastapi
```

Para salir del entorno virtual se usara el comando:

```
deactivate
```

Desde la carpeta app, lanzaremos el servicio de FastAPI, ya que es donde tendremos un main.py programado en Python para poder crear el endpoint.

Mi estructura de carpeta es esta:

app/

├── api/

├──api_teambox.py y __pycache__/

├── main.py

En el main hay q hacer la llamada a api_teambox

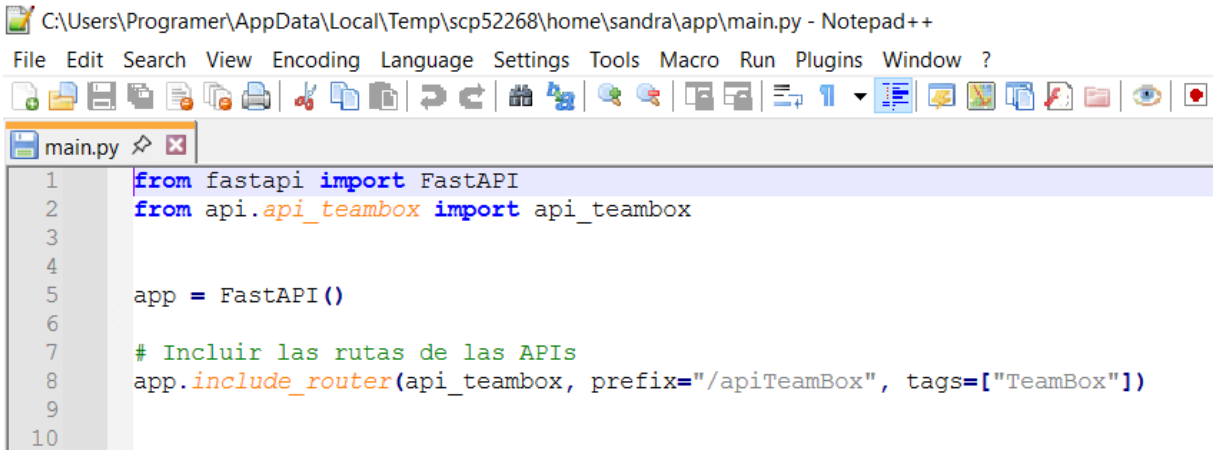


Ilustración 20. Método main del punto de acceso (endpoint)

Como se puede ver, he configurado la API con el prefijo `/apiTeamBox`. Esto significa que, para acceder a cualquiera de los métodos que haya definido (como GET, POST o DELETE), primero deberás incluir ese prefijo en la URL.

Este prefijo organiza las rutas de tu API bajo un mismo "grupo" o categoría, lo que ayuda a mantener el código más ordenado y modular.

La estructura general de las rutas será:

http://obkserver.duckdns.org:8001/apiTeamBox

C:\Users\Programer\AppData\Local\Temp\scp48603\home\sandra\app\api\api_teambox.py - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

new 1 | api_teambox.py

```python
from fastapi import FastAPI, Depends, APIRouter, HTTPException, Query
from sqlalchemy import create_engine, Column, Integer, String, DateTime, ForeignKey, Float, Boolean
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session, relationship
from sqlalchemy import Date, DateTime, TIMESTAMP
from datetime import datetime
from pydantic import BaseModel
from typing import Optional, List
from datetime import date


# Configuración de la base de datos de TeamBox
SQLALCHEMY_DATABASE_URL = "mariadb+mariadbconnector://sandra:sandra123@localhost/teambox"

# Crear el motor de la base de datos
engine = create_engine(SQLALCHEMY_DATABASE_URL)

# Crear la clase base para los modelos
Base = declarative_base()

# Crear la sesión
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)


# Crear el router de esta API
api_teambox = APIRouter()

# Dependencia para obtener la sesión de la base de datos
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Ilustración 21.Código punto de acceso (endpoint)

En la imagen se puede observar los import de FastAPI, SQLAlchemy, Pydantic y Typing, todos estos paquetes se instalan en el entorno virtual con el comando:

```
pip install fastapi sqlalchemy pydantic
```

Esta instalación es completamente necesaria para que funcione el punto de acceso (endpoint).

En este archivo también es configura la conexión a la BBDD.

En el mismo archivo, api_teambox.py, estarán los métodos GET, POST, PUT y DELETE que se encargarán de realizar los cambios en la BBDD.

A continuación, algún ejemplo de estos métodos:

POST: Utilizado para añadir datos

En este método se encarga a través de un formulario implementado en Android Studio, crear Usuarios, ya sean Club, Promotor o Mixto.

```python

# Método para crear usuarios

@api_teambox.post("/Usuarios/Crear")
def create_usuario(usuario: UsuarioCreate, db: Session = Depends(get_db)):
    # Verifica si ya existe el email
    if db.query(UsuarioApp).filter(UsuarioApp.email == usuario.email).first():
        raise HTTPException(status_code=400, detail="El email ya está registrado")

    nuevo_usuario = UsuarioApp(
        nombre=usuario.nombre,
        apellido=usuario.apellido,
        email=usuario.email,
        contrasena=usuario.contrasena,
        es_club=int(usuario.es_club),
        es_promotor=int(usuario.es_promotor),
        es_boxeador=int(usuario.es_boxeador),  # Añadido el campo para saber si es boxeador
        nombre_club=usuario.nombre_club,
        logo_club=usuario.logo_club,
        nombre_promotora=usuario.nombre_promotora,
        logo_promotora=usuario.logo_promotora,
        comunidad=usuario.comunidad,
        provincia=usuario.provincia,
        telefono1=usuario.telefonos[0] if len(usuario.telefonos) > 0 else None,
        telefono2=usuario.telefonos[1] if len(usuario.telefonos) > 1 else None,
        telefono3=usuario.telefonos[2] if len(usuario.telefonos) > 2 else None,
        foto_perfil=usuario.foto_perfil  # Añadido el campo para la foto de perfil
    )

    try:
        db.add(nuevo_usuario)
        db.commit()
        db.refresh(nuevo_usuario)  # OK en MariaDB si no usas RETURNING directamente
        return {"mensaje": "Usuario registrado correctamente", "id": nuevo_usuario.id_usuario}
    except Exception as e:
        db.rollback()
        raise HTTPException(status_code=400, detail="Error al crear el usuario: " + str(e))

```

Ilustración 22.Ejemplo POST

PUT: Utilizado para editar datos existentes

En este caso, tras registrar un boxeador, te pueden faltar datos o has podido equivocarte ala hora de introducirlos, este método hace que se pueda editar la información de la BBDD.

```python
@api_teambox.put("/Boxeadores/Editar/{id}")
def edit_boxeador(id: int, boxeador: BoxeadorCreate, db: Session = Depends(get_db)):
    db_boxeador = db.query(Boxeador).filter(Boxeador.Id_boxeador == id).first()

    if not db_boxeador:
        raise HTTPException(status_code=404, detail="Boxeador no encontrado")

    # Validar que el club existe
    club = db.query(UsuarioApp).filter(UsuarioApp.id_usuario == boxeador.club_id).first()
    if not club:
        raise HTTPException(status_code=400, detail="El club indicado no existe")

    db_boxeador.nombre = boxeador.nombre
    db_boxeador.apellido = boxeador.apellido
    db_boxeador.fecha_nacimiento = boxeador.fecha_nacimiento
    db_boxeador.dni_boxeador = boxeador.dni_boxeador
    db_boxeador.genero = boxeador.genero
    db_boxeador.peso = boxeador.peso
    db_boxeador.categoria = boxeador.categoria
    db_boxeador.comunidad = boxeador.comunidad
    db_boxeador.provincia = boxeador.provincia
    db_boxeador.club_id = boxeador.club_id
    db_boxeador.foto_perfil = boxeador.foto_perfil

    try:
        db.commit()
        db.refresh(db_boxeador)
        return {"mensaje": "Boxeador actualizado correctamente"}
    except Exception as e:
        db.rollback()
        raise HTTPException(status_code=400, detail="Error al editar el boxeador: " + str(e))
```

Ilustración 23. Ejemplo PUT

GET: Utilizado para leer datos

Este método comprueba si el DNI existe ya en la BBDD o no.

```python
# Metodo para ver si DNI existe en BBDD

@api_teambox.get("/Boxeadores/DniExiste", response_model=bool)
def dni_existe(dni_boxeador: str, db: Session = Depends(get_db)):
    try:
        existe = db.query(Boxeador).filter(Boxeador.dni_boxeador == dni_boxeador).first()
        return existe is not None
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error interno: {str(e)}")
```

Ilustración 24. Ejemplo GET

DELETE: Utilizado para eliminar datos

En este caso elimina un boxeador de la BBDD que haya sido creado previamente

```python
# Método para eliminar un boxeador

@api_teambox.delete("/Boxeadores/Eliminar/{id}")
def delete_boxeador(id: int, db: Session = Depends(get_db)):
    db_boxeador = db.query(Boxeador).filter(Boxeador.Id_boxeador == id).first()

    if not db_boxeador:
        raise HTTPException(status_code=404, detail="Boxeador no encontrado")

    try:
        db.delete(db_boxeador)
        db.commit()
        return {"mensaje": "Boxeador eliminado correctamente"}
    except Exception as e:
        db.rollback()
        raise HTTPException(status_code=400, detail="Error al eliminar el boxeador: " + str(e))


@api_teambox.get("/Boxeadores/Club/{clubId}", response_model=List[BoxeadorOut])
def obtener_boxeadores_por_club(clubId: int, db: Session = Depends(get_db)):
    boxeadores = db.query(Boxeador).filter(Boxeador.club_id == clubId).all()

    if not boxeadores:
        raise HTTPException(status_code=404, detail="No se encontraron boxeadores para este club")

    return boxeadores
```

Ilustración 25. Ejemplo DELETE

Estos son todos los métodos que tengo creados en api_teambox.py

Ilustración 26. Métodos API

Configuración Del Router

Al tener el servidor en una zona fuera de mi red local, es necesario configurar y hacer accesible desde el exterior.

Para ello se utilizó DuckDNS, ofrece un servicio gratuito, aunque corres el riesgo de que la IP externa se cambie sin avisar y pueda fallar el acceso.

Lo primero de todo es poner IP fija a la MAC de la Raspberry Pi.

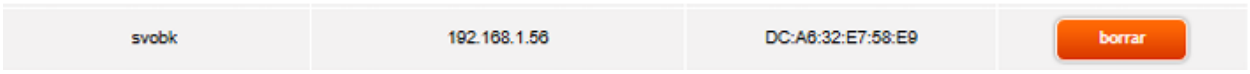


Ilustración 27. Router (I)

Configuración de puertos.

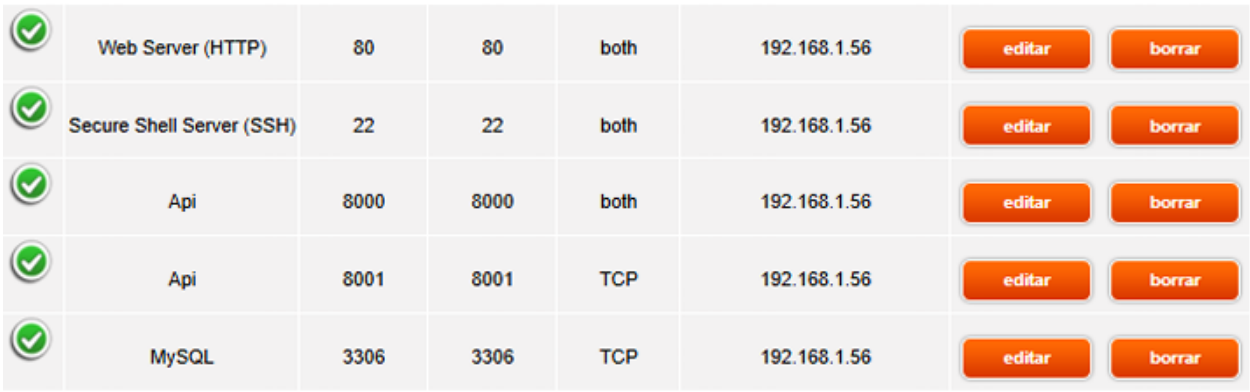


Ilustración 28. Router(II)

DuckDNS

Para usar DuckDNS, con registrarse en la página web es suficiente.

El dominio se le puso un nombre: obkserver y el dominio completo se queda asi:

Obkserver.duckdns.org

Tras realizar el registro, necesitamos averiguar cual es la IP pública , y para ello puedes usar el explorar de Google.

Ilustración 29. Consulta IP externa

Se le asigna al dominio obkserver la IP externa

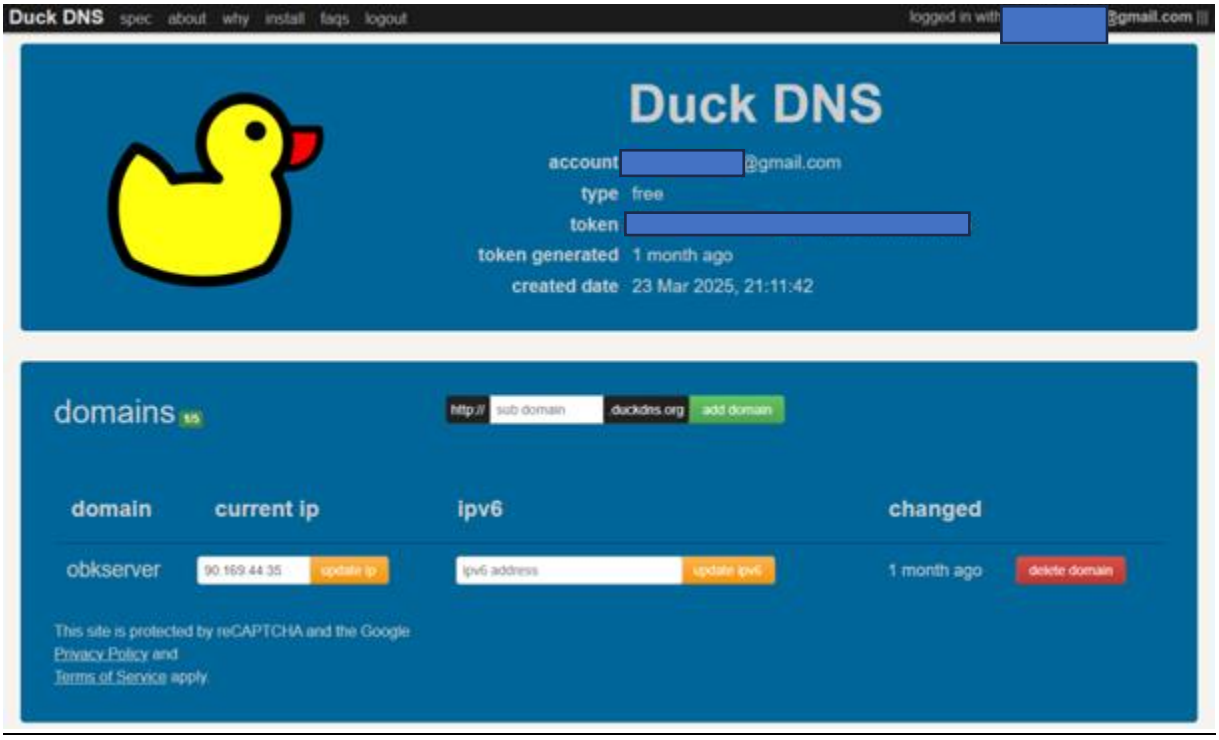


Ilustración 30. DuckDNS

APP (Android Studio)

La aplicación Android de TEAMBOX cumple la función principal de gestionar boxeadores, clubes y promotores dentro del ámbito deportivo. Según el rol del usuario que acceda (club, promotor o mixto), se habilitan unas funcionalidades específicas.

Las funcionalidades **comunes** incluyen:

- Visualizar perfiles de boxeadores.
- Realizar búsquedas con múltiples filtros.
- Visualizar detalles, datos relevantes de los boxeadores.

Las funcionalidades exclusivas del **perfil club** incluyen:

- Crear, modificar y eliminar boxeadores propios.
- Cargar fotografías de perfil (en formato Base64).
- Validar DNI/NIE y asignar automáticamente la categoría por año de nacimiento.

Las funcionalidades del **perfil promotor** incluyen:

- Buscar boxeadores mediante filtros personalizados.
- Marcar y gestionar boxeadores como favoritos.

El **perfil mixto** permite combinar las funciones anteriores.

La aplicación está diseñada para ejecutarse en dispositivos móviles Android, permitiendo su uso en terminales modernos y también en dispositivos integrados con Android, como tablets o terminales táctiles.

Estructura

La aplicación Android cuenta con múltiples pantallas, cada una con una funcionalidad específica:

- Pantalla de inicio de sesión: Permite autenticarse con rol de club, promotor o mixto.
- Pantalla de gestión de boxeadores (para clubes): Crear, modificar y eliminar boxeadores.
- Pantalla de búsqueda avanzada (para promotores y mixtos): Buscar boxeadores mediante filtros como nombre, peso, comunidad, categoría, género, etc.
- Pantalla de favoritos (para promotor): Visualizar y eliminar boxeadores marcados como favoritos.
- Pantalla de perfil de usuario: Visualiza los datos del usuario (club o promotor).

También se integran funciones esenciales para el funcionamiento:

- Conexión con el backend mediante API REST desarrollada con FastAPI.
- Manejo de datos estructurados mediante modelos sincronizados con la base de datos MariaDB.
- Validación de formularios, gestión de errores y control de imágenes en formato Base64.
- Sincronización automática de cambios realizados en el backend con la interfaz del usuario.

Código

Para permitir el acceso a de la app a la API tenemos el AndroidManifest.xml

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <!-- Permiso de internet para acceder a la API -->
    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@drawable/logob"
        android:label="AppTeamBox"
        android:roundIcon="@drawable/logob"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppTeamBox"
        android:networkSecurityConfig="@xml/network_security_config"
        tools:targetApi="31">
```

Ilustración 31. AndroidManifest.xml

Para la conexión entre BBDD y APP existen estas clases

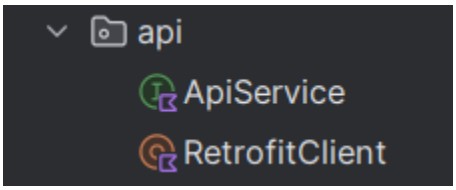


Ilustración 32. Docs Api

ApiService se ve así, es el archivo donde se ponen las llamadas a la API.

```
interface ApiService {

    // Registro de usuario
    @POST("Usuarios/Crear")
    fun registrarUsuario(@Body usuario: RegistroUsuario): Call<Unit>

    // Login
    @POST("Usuarios/Login")
    suspend fun login(@Body request: LoginRequest): Response<UsuarioResponse>

    //Sirve para buscar Usuario por email
    @GET("Usuarios/Obtener/{email}")
    suspend fun obtenerUsuarioPorEmail(@Path("email") email: String): Response<UsuarioResponse>

    //Sirve para buscar Usuario por ID
    @GET("Usuarios/ObtenerPorId/{id_usuario}")
    suspend fun obtenerUsuarioPorId(
        @Path("id_usuario") idUsuario: Int
    ): Response<UsuarioResponse>


    // **********Respecto a pantalla de Equipo**********

    // Imprime boxeadores en pantalla Equipo
    @GET("Boxeadores/Club/{clubId}")
    suspend fun getBoxeadoresPorClub(@Path("clubId") clubId: Int): List<Boxeador>

```

Ilustración 33. Código ApiService

RetrofitClient, se crea la instancia a la API y se conecta a ella.

```
package com.example.appteambox.api

import ...


object RetrofitClient {
    // URL base de la API
    private const val BASE_URL = "http://obkserver.duckdns.org:8001/apiTeamBox/"

    val apiService: ApiService by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create()) //usa Gson para convertir a JSON
            .build()
            .create(ApiService::class.java)
    }
}
```

Ilustración 34. Código RetrofitClient

En estructura del código, existen tres menús diferentes, uno para cada rol (club, promotor o mixto)

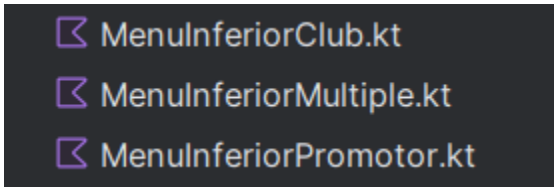


Ilustración 35. Clases Menús

He intentado separar lo máximo posible la UI de la lógica, usando ViewModel.

*RegistroUsuario no se usa, pero lo dejé ahí para futura implementación.

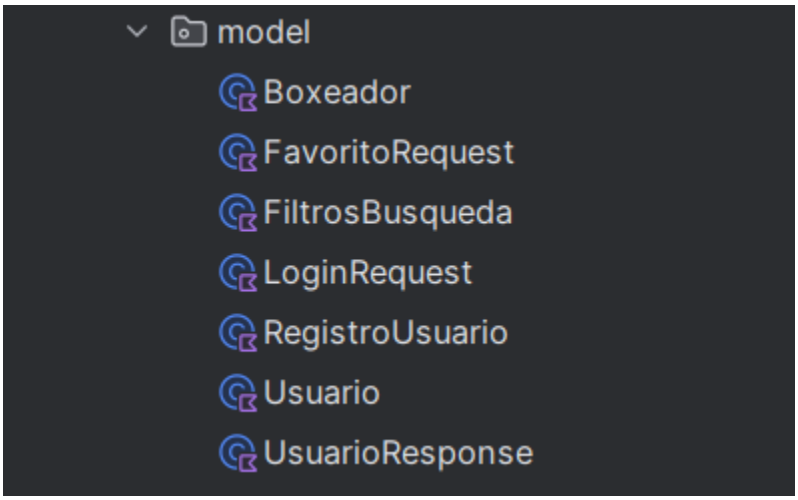


Ilustración 36. ViewModel Utilizados

Funciones mas importantes

calcularCategoria (BoxeadorViewModel)

Esta función, bajo mi punto de vista, es una de las más importante con respecto a la app.

La categoría la determina el año de nacimiento y no la edad del competidor en ese instante, por lo tanto, a veces, entre los futuros usuarios de la app existe cierta confusión de a que categoría corresponde su competidor, llegando incluso a suspenderse un encuentro (combate) por ser ambos participantes de diferentes categorías (algo que está totalmente prohibido), entonces así no tendrían duda de cual poner.

Esta función esta preparada para cuando se amplie la búsqueda en las categorías inferiores, a día de hoy puedes completar propio equipo con todas las categorías existentes, pero solo puedes realizar búsqueda de los que realmente hacen combate con contacto.

```
fun calcularCategoria(fechaNacimiento: String): String {
    // Extraer el año de nacimiento (de los primeros 4 caracteres)
    val anioNacimiento = fechaNacimiento.take( n: 4).toIntOrNull() ?: return "Desconocida"

    // Obtener el año actual
    val anioActual = LocalDate.now().year

    // Calcular la edad
    val edad = anioActual - anioNacimiento

    // Devolver la categoría según la lógica del evento SQL
    return when {
        edad > 18 -> "Élite"
        edad in 17 ≤ .. ≤ 18 -> "Joven"
        edad in 15 ≤ .. ≤ 16 -> "Junior"
        edad in 13 ≤ .. ≤ 14 -> "Cadete"
        edad in 11 ≤ .. ≤ 12 -> "Infantil"
        edad in 9 ≤ .. ≤ 10 -> "Benjamin"
        else -> "Prebenjamin"
    }
}
```

obtenerFavoritosPorClub (FavoritosViewModel)

Esta función llama a la API para obtener la lista de boxeadores favoritos.

Lo mas importante de esta función es que se realiza en hilo I/O (entrada y salida) es lo mas adecuado para cuando realizas llamadas hacia fuera, como en este caso hacia una BBDD) para que la App no deje de funcionar porque se bloquee la interfaz de usuario.

Ya que el hilo principal se encarga de pintar la interfaz, si hacemos una petición que puede ser un poco lenta, la app se puede congelar o crashear.

```
34      fun obtenerFavoritosPorClub(clubId: Int) {
35          viewModelScope.launch {
36              try {
37                  val response = withContext(Dispatchers.IO) {
38                      // Llamada en hilo de I/O para evitar bloquear la UI
39                      RetrofitClient.apiService.obtenerFavoritos(clubId)
40                  }
41                  // Se actualiza la lista observable
42                  _favoritos.value = response
43              } catch (e: Exception) {
44                  Log.e( tag: "FavoritosViewModel",  msg: "Error al obtener favoritos", e)
45              }
46          }
47      }
```

handleUserType (LoginViewModel)

Esta función es la que se encarga de determinar la experiencia del usuario que se haya registrado según su rol, dependiendo de lo que el usuario haya "elegido" accederá a un tipo de menú y de funciones de la app.

```
72      private fun handleUserType(usuario: UsuarioResponse) {
73          Log.d( tag: "Login",  msg: "Usuario recibido: es_club=${usuario.es_club}, es_promotor=${usuario.es_promotor}")
74          _navigateTo.value = when {
75              usuario.es_club && !usuario.es_promotor -> "MenuInferiorClub"
76              usuario.es_promotor && !usuario.es_club -> "MenuInferiorPromotor"
77              usuario.es_club && usuario.es_promotor -> "MenuInferiorMultiple"
78              else -> {
79                  errorMessage = "No se pudo determinar el tipo de usuario."
80                  null
81              }
82          }
83      }
```

Líneas futuras de mejora

Esta primera versión de TeamBox, es lo mínimo aceptable para que pueda ser usada, aunque me gustaría seguir desarrollándola y añadiendo mejor como, por ejemplo:

Notificaciones push: Implementar un sistema de notificaciones para informar a los clubes y promotores sobre nuevos boxeadores disponibles, cambios en sus favoritos o recordatorios de eventos.

Gestión de eventos y combates: Añadir un módulo para que los clubes puedan organizar competiciones, registrar resultados y asociar boxeadores a eventos específicos.

Roles adicionales: Ampliar el sistema de usuarios para incluir boxeadores, árbitros o médicos, permitiendo asignar funciones y accesos específicos.

Estadísticas y comparativas: Incorporar gráficos e indicadores sobre el rendimiento de los boxeadores, número de combates, progresión anual y rankings por categorías y géneros.

Soporte multiplataforma: Desarrollar una versión web o una app para iOS.

Internacionalización: Traducir la interfaz a otros idiomas (como inglés o francés) para permitir su uso en federaciones o clubes de otros países.

Estas mejoras permitirían convertir TEAMBOX en una herramienta aún más completa y profesional, adaptable a federaciones deportivas, clubes privados y promotores a nivel nacional o internacional.

Conclusiones Globales

Impacto Personal

El desarrollo de esta app ha sido una experiencia la cual me ha dado muchos momentos buenos, ya me gustaba la informática desde que tengo uso de razón pero ver que vas avanzando en un proyecto que hace dos años no hubiera sabido ni como planteármelo, ha sido bastante satisfactorio.

Impacto Profesional

Cuando fui consciente de que había que hacer un TFG, tenía claro que quería que fuera algo que pudiera luego sacarle partido, ya que eso sería una motivación extra a la hora de trabajar sobre ello y creo que lo he conseguido.

Espero a nivel profesional tener el impacto esperado, ya que es una idea que voy a llevar a cabo fuera del TFG y sacarlo al mundo REAL.

Reflexión Final

La informática es muy amplia y he tenido la suerte de cursar SMR y DAM, espero seguir durante mi carrera profesional formándome y desarrollando habilidades.

Seguir actualizándome en el campo relacionado con la informática y la tecnología.

Bibliografía

Discord

Mouredev. (n.d.). *Comunidad de programación en Discord* [Servidor de Discord]. Recuperado el 24 de mayo de 2025, de https://discord.gg/mouredev

Youtube

Mouredev. (2023). *Cómo crear una app en Jetpack Compose desde cero* [Video]. YouTube. https://youtu.be/0UjtLmAO1SA

TheCodeMate. (2023). *Jetpack Compose Firebase Authentication - Google Login* [Video]. YouTube. https://www.youtube.com/watch?v=7Mf2175h3RQ&t=3s

Mouredev. (2023). *Curso de Kotlin y Jetpack Compose* [Lista de reproducción]. YouTube. https://youtube.com/playlist?list=PLQkwcJG4YTCSpJ2NLhDTHhi6XBNfk9WiC

Raspberry PI

Piltch, A. (2023, octubre 23). *How to set up a headless Raspberry Pi, without ever attaching a monitor*. Tom's Hardware. https://www.tomshardware.com/reviews/raspberry-pi-headless-setup-how-to,6028.html

FastAPI

Tiangolo, S. (s.f.). *Tutorial - Guía del usuario* [Tutorial]. FastAPI. https://fastapi.tiangolo.com/es/tutorial/

Webs varias

Android Developers**.** (s.f.). *Camino de aprendizaje de Jetpack Compose* [Sitio web]. Android Developers. https://developer.android.com/courses/pathways/compose?hl=es

Android Developers. (s.f.). *Codelab: Conceptos básicos de Jetpack Compose* [Sitio web]. Android Developers. https://developer.android.com/codelabs/jetpack-compose-basics?hl=es-419#0

Android Developers. (s.f.). *Tutorial: Diseñar IU con Jetpack Compose* [Sitio web]. Android Developers. https://developer.android.com/develop/ui/compose/tutorial?hl=es-419

Anexos

GitHub

Mi repositorio

https://github.com/SandrMtz/APP_TeamBox