

BeSport24Training

Alumno: David Peñalver Navarro
Curso: 2º DAW
Promoción: 2023/2025



CC BY-NC 4.0

Trabajo bajo licencia:

Atribución-No Comercial-Compartir Igual CC BY-NC-SA

Usted puede compartir, adaptar y construir sobre este trabajo, siempre y cuando se acredite adecuadamente y no se utilice con fines comerciales y cualquier trabajo derivado se acredite bajo la misma licencia.



CC BY-NC 4.0

Gracias a todas las personas que me han acompañado en este viaje de más de 29 años.

En especial, a mi familia por darme soporte y ayuda siempre que lo he necesitado. A mis amigos por esos ratos de confesiones, de risas y de momentos mágicos. A mi pareja por convertirse en un pilar fundamental en mi vida.

A mis profesores por todo lo que me han enseñado en estos dos años de aventura. A Flit2Go por todos estos aprendizajes y nuevos conocimientos.

Índice

<i>Introducción</i>	9
<i>Temporalización</i>	11
<i>Herramientas y tecnologías</i>	17
Herramientas	17
Macintosh	17
iPhone	17
PC Linux	18
IntelliJ IDEA	18
Visual Studio Code	19
Xcode.....	19
Simulador de Xcode	20
GitHub.....	21
Canva	22
Draw.io.....	22
Warp	22
Konsole.....	23
Tecnologías	23
Git	23
MariaDB.....	24
Java	24

Spring	24
Maven.....	25
JPA	25
Lombok.....	26
MapStruct	26
CORS	26
Postman.....	27
Swift.....	27
SwiftUI	27
SFSymbols	28
<i>Implementación.</i>	29
Análisis.....	29
Requerimientos iniciales.....	29
Actores del sistema	30
Diagrama de casos de uso.....	31
Descripción de cada caso de uso.....	32
Diseño	46
Logo.....	46
Diagrama entidad-relación	48
Base de datos	50
Back End	53

Paquete persistence.....	74
Front End	88
Manual de usuario.....	119
Sección 1: Login	119
Sección 2: Perfil.....	121
Sección 3: Ejercicios	124
Sección 4: Sesiones	129
Sección 5: Entrenamientos.....	132
Propuestas de mejora	134
Valoración personal.....	137
Puntos a destacar del proyecto	140
One More Thing.....	142
Referencias	145

Introducción

BeSport24 fue un proyecto de prácticamente cinco años de duración que combinó los entrenamientos presenciales con los entrenamientos online y la divulgación científica en plataformas como Instagram y YouTube. En dicha era de mi vida, el alcance del proyecto fue masivo, llegando a cientos de miles de visualizaciones y ayudando personalmente a decenas de personas a conseguir sus ansiados objetivos. Dichos objetivos fueron muy variados: desde futbolistas semiprofesionales que aspiraban a un puesto en Estados Unidos a pacientes oncológicos.



Figura 1. Visualizaciones en uno de los Reels de Instagram.

Tal y como dice la primera palabra de esta introducción, dicho proyecto se llamó BeSport24, ya que reflejaba la filosofía de que un estilo de vida saludable se logra a lo largo del día y no solamente en una hora de entrenamiento, por lo que se buscaba ese enfoque holístico. Asimismo, todo ello se llevaba a la práctica en base a la evidencia científica más actualizada, pudiéndose encontrar en la cuenta de Instagram centenares de vídeo-resúmenes de dichos *papers* o artículos científicos de cara a que la ciencia fuera más accesibles a la población general.

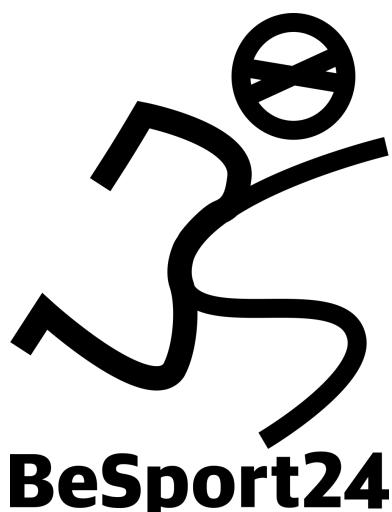


Figura 2. Logo de BeSport24.

Sin embargo, siempre había *algo* que se echaba en falta en ese proyecto: una aplicación de seguimiento de los entrenamientos y de valoración de cada uno de los pacientes. De esta forma, el seguimiento del entrenamiento sería eficiente y eficaz tanto para mis deportistas presenciales como para aquellos que entrenaban en México, Uruguay, Italia o Estados Unidos.

Tuve la ocasión de probar diferentes alternativas: un Excel de elaboración propia con macros, documentos PDF con las programaciones o aplicaciones web como Trainiks. Pero nada de eso me convencía. Necesitaba algo mucho más completo e intuitivo para el cliente y, al mismo tiempo, que me ahorrara faena en mis interminables jornadas laborales que duraban no de sol a sol, sino desde antes de la salida hasta la puesta de nuestro astro rey.

Es por ello que nace este proyecto como idea de TFG: la aplicación con la que tanto soñé para poder planificar, valorar y evaluar todo el proceso de entrenamiento.

En cuanto al nombre de esta aplicación, combina el nombre del proyecto de cara a una mayor visibilidad junto a la palabra entrenamiento en inglés (“training”) debido a su enfoque internacional. Asimismo, el nombre del proyecto está en negrita para diferenciarlo de “Training”, con el “4” en negrita y cursiva jugando como nexo de unión debido a su fonética similar a “for” (para, en inglés).

Temporalización

De cara a la monitorización del avance del proyecto, se ha empleado una plantilla del diagrama de Gantt desarrollada por Vertex42. Dicha plantilla facilita la elaboración del tiempo de cada proyecto, conocer el progreso y las fechas concretas.

A continuación, se presenta el diagrama completo que, posteriormente, será desgranado en cada apartado para poder profundizar en mayor medida en la evolución del proyecto:

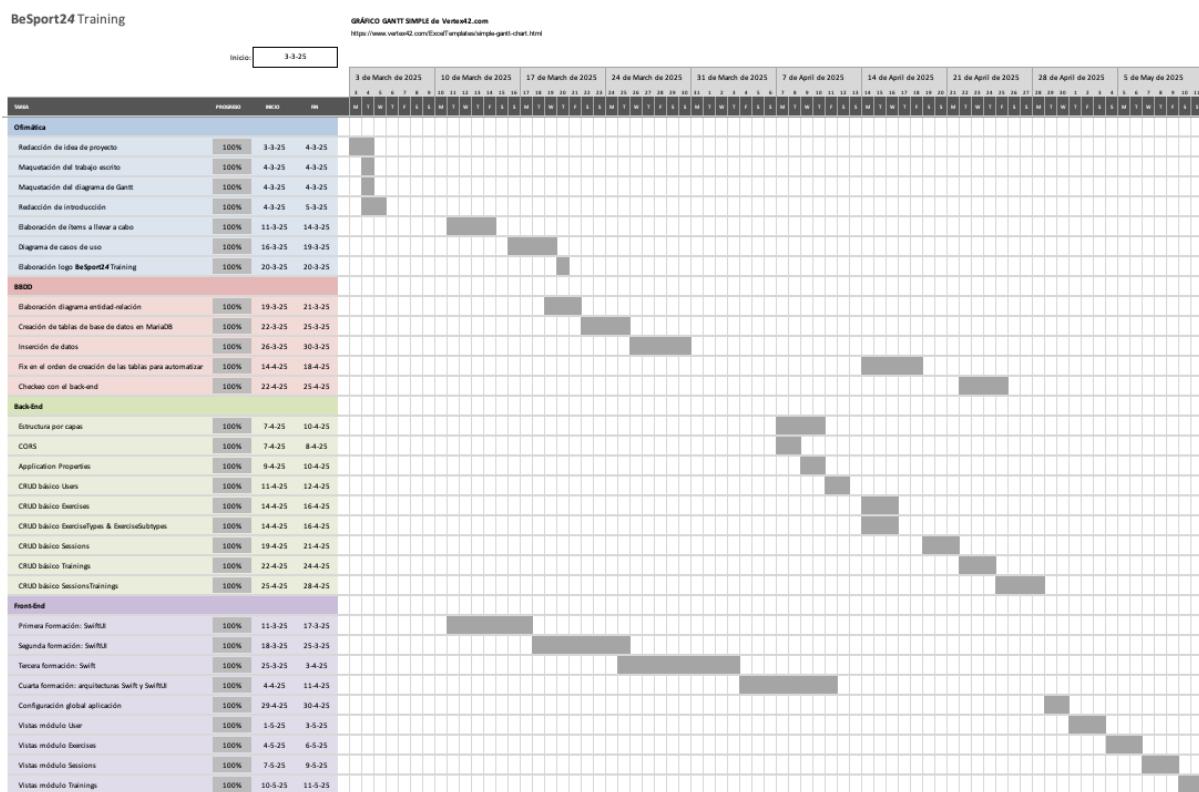


Figura 3. Diagrama de Gantt de **BeSport24Training**.

En la primera fase, la de ofimática, tenemos las siguientes tareas:

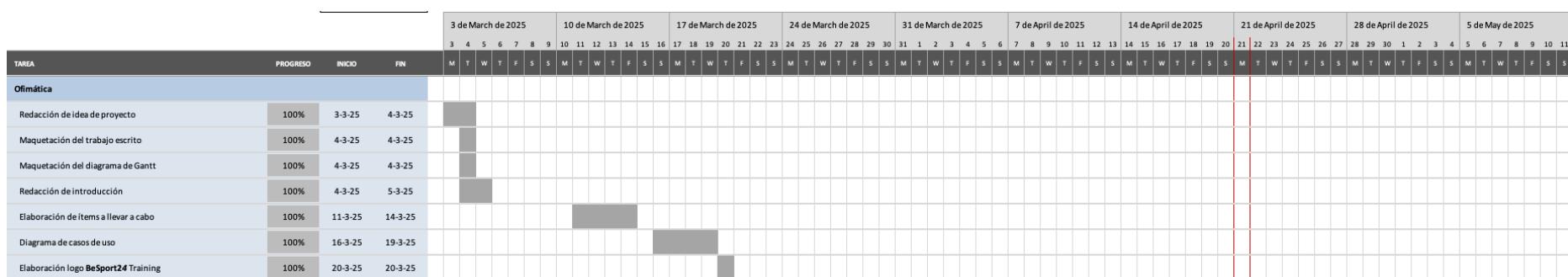


Figura 4. Diagrama de Gantt: Ofimática.

Tal y como se puede apreciar, el primer paso fue el de la elaboración de la idea del proyecto a realizar y la de la maquetación del trabajo escrito, de cara a ver qué tiempo tenía disponible, qué opciones se podían implantar, qué era necesario hacer y qué calendario debía estructurarme y cómo organizar el trabajo escrito de cara a plasmar todo aquello que habría desarrollado desde el inicio hasta la entrega.

Una vez maquetado cómo sería el trabajo y viendo qué tenía que hacer y, aproximadamente, cuándo, llegaba el turno de realizar el diagrama de Gantt de cara a organizar todo el trabajo de manera clara, precisa y concisa.

En cuanto a la maquetación, esta no es estricta, sino que cada semana se va analizando la viabilidad y ajustando.

Comenzando ya con la parte visible de este proyecto, entre el cuatro y cinco de marzo se realizó la redacción de la introducción.

Tal y como se puede observar, en los días siguientes se produjo una pausa, debida al viaje a Colonia para desconectar unos días del periodo final de exámenes, el comienzo de esta tarea y el trabajo por las tardes en el centro de entrenamiento personal. Posteriormente, se redactaron los ítems a realizar en cada fase del proyecto, se realizó el diagrama de casos de uso y se diseñó el logo de la aplicación, el cual se puede ver en la portada del trabajo y en el apartado de diseño, al igual que el diagrama también comentado en este párrafo.

En la segunda fase, la de Base de Datos (BBDD), se realizaron las siguientes tareas:

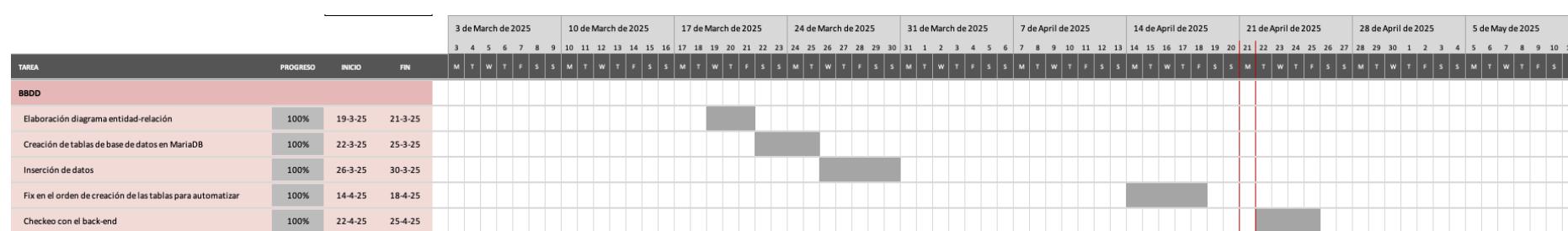


Figura 5. Diagrama de Gantt: BBDD.

En este apartado, el primer paso consistió en la elaboración del diagrama entidad-relación de cara a cumplimentar los casos del diagrama de casos de uso satisfactoriamente. El diagrama en sí no fue excesivamente laborioso, si bien se alargó varios días de cara a asegurarme de que reunía todos los requisitos necesarios, sobre todo en el ámbito de poder plasmar correctamente las necesidades de cada sesión: el número de series de cada ejercicio, las repeticiones, el tipo del ejercicio o el subtipo para poder filtrarlos y registrarlos correctamente en cada entrenamiento.

Tras ello, se procedió a realizar la base de datos propiamente dicha. En primer lugar, se crearon las tablas basándome en el mapa entidad-relación para, posteriormente, insertar los diferentes registros.

BeSport24Training

Finalmente en este apartado, hubo un periodo de varios días en el cual tuve que modificar el orden de creación de las tablas para que todo el proceso de creación de la base de datos e inserción de los datos iniciales se pudiera hacer simplemente copiando y pegando el script en la terminal. Asimismo, también se tuvo que comprobar algunos nombres de los campos de las tablas para que la nomenclatura en *Back-End* y BBDD fuera coherente de cara a evitar fallos en el futuro por si deseó escalar la aplicación con las propuestas de mejora que se mencionan más adelante en su correspondiente apartado.

En la tercera fase, la de *Back-End*, se llevaron a cabo las siguientes tareas:

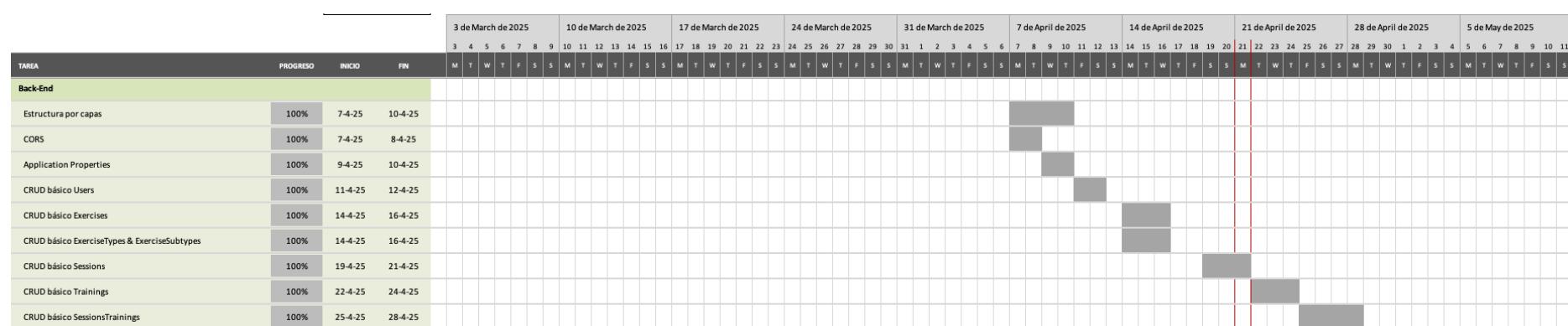


Figura 6. Diagrama de Gantt: Back-End.

En el Back-End, el primer paso fue crear la estructura que iba a conformar la arquitectura del proyecto, siendo la arquitectura por capas vista en el módulo de *Desarrollo de Aplicaciones Web*: Servidor de César Guijarro. A su vez, se hizo la implementación del CORS para poder conectar el *back* con el *front* más adelante y la del *Application Properties* de cara a tener toda la estructura preparada para hacer las implementaciones.

Tras ello, llegaba el turno de hacer los CRUDS de los diferentes *endpoints* de la aplicación. El orden que se siguió fue hacer primeramente las entidades independientes para, tras ello, realizar aquellas que necesitaban mapear otras entidades. De ahí que la de los usuarios fuera la primera y que la de Sesiones-Ejercicios fuera la última, ya que accede a la de Sesiones y la de Ejercicios.

En la última fase, la de Front-End, disponemos de las siguientes tareas:

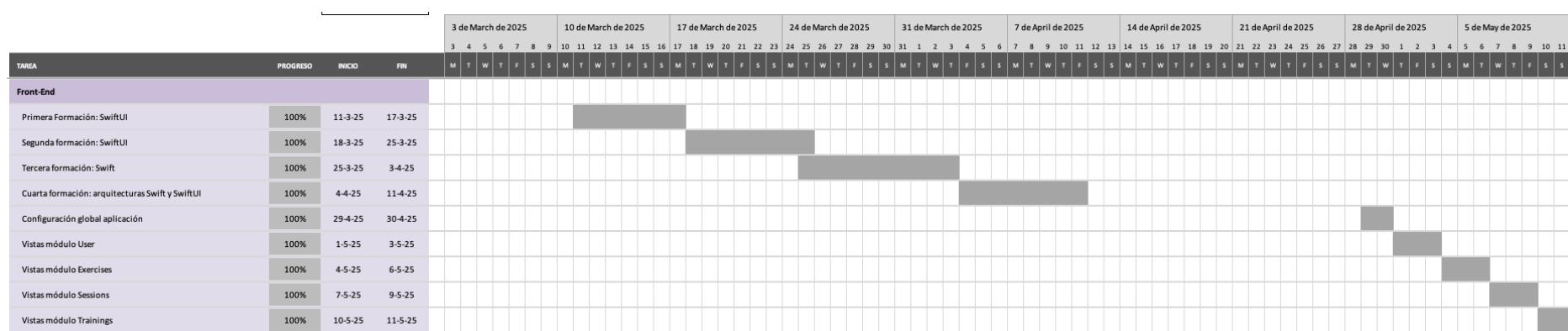


Figura 7. Diagrama de Gantt: Front-End.

Finalmente, se realizó el front-end de la aplicación. Debido a las tecnologías vistas durante el periodo de prácticas y, al mismo tiempo, a la naturaleza de las sesiones de entrenamiento, escogí realizarlo con el lenguaje de programación nativo de Apple y con su correspondiente *framework*: Swift y SwiftUI, respectivamente.

Por ello, las primeras fases de este periodo fueron las de aprendizaje tanto de las peculiaridades del lenguaje de programación como de su correspondiente framework. En este periodo de aprendizaje, destaco las diferencias en la sintaxis del lenguaje respecto a lo que estaba acostumbrado a hacer, que era Java y JavaScript. Asimismo, al ser un lenguaje orientado al dispositivo móvil

(específicamente el iPhone), SwiftUI cuenta con muchas peculiaridades que no había visto previamente. Por estos motivos, esta fase se prolongó tanto en el tiempo.

Tras aprender todo lo necesario con estas tecnologías, llegaba el momento de la arquitectura a utilizar. Aprovechando que fue una de las tareas que tuve que llevar a la empresa, tomé los apuntes pertinentes para, también, poder escoger la arquitectura más adecuada en este proyecto final. Se trató de un proceso de aprendizaje guiado, por lo que las conclusiones a las que llegué tras las pautas iniciales son propias.

Finalmente, la parte restante del proyecto fue empleada para realizar el propio *frontend* de la aplicación. En esta parte, se crearon los modelos de negocio, los DTO de los datos que se recibían del *backend* y la capa que se encargaba de ello y, finalmente, se procedió a crear las vistas (*View*) y los archivos que se encargaban del negocio de las vistas (*ViewModel*).

Herramientas y tecnologías

De cara a poder hacer este posible este proyecto y cumplir con la temporalización previa, han sido necesarias numerosas herramientas y tecnologías que se describen a continuación.

Herramientas

El listado de herramientas empleadas es el siguiente:

Macintosh



Macintosh o Mac es la línea de ordenadores personales diseñada por el gigante americano Apple.

El equipo Macintosh se ha utilizado como entorno principal de desarrollo para la parte del proyecto orientada a iOS, dado que el sistema operativo macOS es requisito para ejecutar herramientas como Xcode y el simulador de iPhone.

Para ser más concretos, el equipo empleado ha sido un Mac Mini con procesador M1 de 256GB de almacenamiento y 8GB de memoria RAM unificada, el cual emplea la arquitectura ARM en su CPU, permitiéndole una gran eficiencia a la par que potencia por watio.

El sistema operativo sobre el que ha corrido esta computadora es macOS Sequoia 15.4.0.

iPhone



El iPhone ha sido el dispositivo móvil utilizado para realizar pruebas reales de la aplicación, permitiendo validar su funcionamiento fuera del entorno simulado.

Para ser más concretos, se ha empleado un iPhone 11 Pro con 256GB de almacenamiento interno.

PC Linux



Un ordenador portátil con sistema operativo Linux se ha utilizado para el desarrollo del backend en Java. Asimismo, ha sido el equipo empleado como servidor a la hora de realizar las pruebas en un entorno real.

El equipo concreto es un Lenovo Ideapad Gaming 3 Gen 6 con CPU AMD Ryzen de la gama 5000H, 16GB de memoria RAM DDR5, 512GB de almacenamiento interno y GPU Nvidia GeForce RTX 3050.

El sistema operativo sobre el que corre es KDE Plasma 6.

IntelliJ IDEA



IntelliJ IDEA es un entorno de desarrollo integrado (IDE) desarrollado por JetBrains, ampliamente reconocido por su potencia, velocidad y herramientas avanzadas de asistencia al programador.

Una de las características más destacadas de IntelliJ IDEA es su sistema de autocompletado inteligente, el análisis estático del código y la detección temprana de errores, lo que ha permitido un desarrollo más ágil y con menor margen de fallos.

En el contexto de este proyecto, se ha utilizado principalmente para el desarrollo del backend en Java. Se ha empleado la versión IntelliJ Idea Ultimate gracias a la cuenta de estudiante de CIPFP Mislata.

Visual Studio Code



Visual Studio Code, desarrollado por Microsoft, es un editor de código fuente ligero, multiplataforma y de código abierto, que se ha convertido en una de las herramientas más populares entre desarrolladores de todo el mundo debido a que su instalación y uso son gratuitos.

Una de las grandes ventajas de VS Code es su sistema de extensiones, que permite ampliar sus funcionalidades de manera sencilla. Gracias a ello, se ha empleado para la elaboración de la base de datos y de las inserciones iniciales en la misma.

Xcode



Xcode es el entorno de desarrollo integrado (IDE) oficial de Apple para macOS, utilizado para desarrollar aplicaciones para iOS, macOS, watchOS y tvOS. En el contexto de este proyecto, Xcode ha sido fundamental para construir y probar la aplicación destinada a dispositivos iOS, ya que proporciona todas las herramientas necesarias para compilar, depurar y lanzar la aplicación.

Uno de los componentes más valiosos de Xcode ha sido su diseñador visual, especialmente útil para trabajar con SwiftUI, el framework de diseño de interfaces moderno de Apple. Gracias a su vista previa en tiempo real (*canvas*), ha sido posible ajustar de forma visual cada pantalla de la aplicación, comprobando cómo se adaptaba a distintos tamaños de dispositivo sin necesidad de compilar constantemente el proyecto.

Simulador de Xcode



El simulador de iOS es una herramienta clave para probar la aplicación sin necesidad de disponer físicamente de un iPhone o iPad en todo momento. Esto ha sido especialmente útil para validar comportamientos, realizar pruebas de diseño y verificar la navegación entre pantallas.

Una de las ventajas que tiene es que permite simular cualquier dispositivo iOS/iPad OS, ya sea iPhone o iPad, para poder comprobar que el aspecto visual y la funcionalidad es la adecuada en toda la gama de productos Apple en los que puede funcionar la aplicación sin la necesidad de emplear un dispositivo físico.



Figura 8. Ejemplo de visualización del simulador de un iPhone 16 Pro.

BeSport24Training

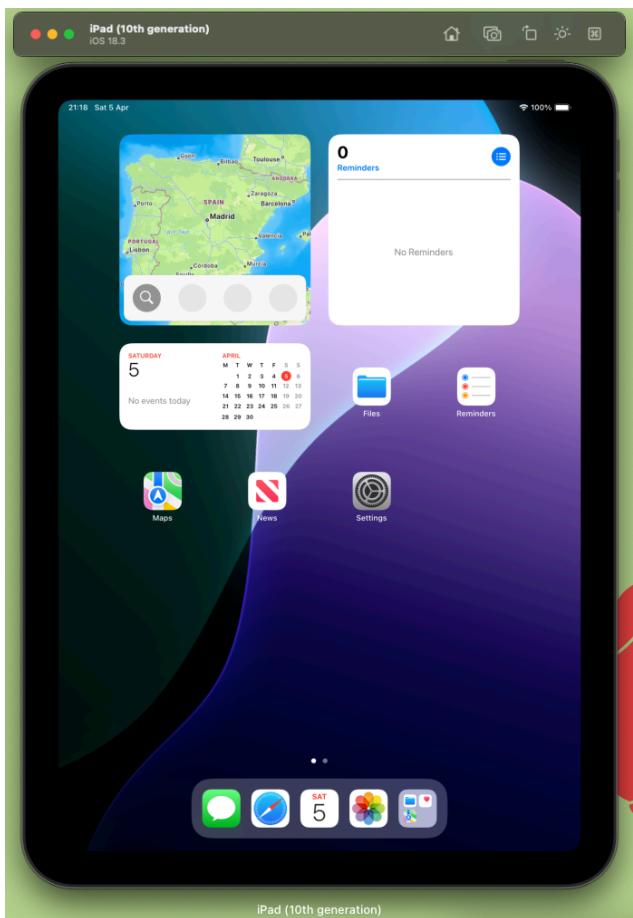


Figura 8. Ejemplo de visualización del simulador de un iPad 10^a generación.

GitHub



GitHub es una plataforma de desarrollo colaborativo basada en la web que utiliza el sistema de control de versiones Git. Ha sido una herramienta central en este proyecto, ya que ha permitido llevar un control detallado y organizado del código fuente y la documentación del mismo.

Gracias a su sistema de repositorios y ramas, ha sido posible dividir el trabajo en funcionalidades concretas sin afectar el desarrollo general, lo que ha favorecido un flujo de trabajo limpio y estructurado.

BeSport24Training

Por último, GitHub también ha servido como plataforma para alojar de forma segura el proyecto, permitiendo el acceso desde cualquier dispositivo con conexión a Internet, ya que recordemos que dos equipos distintos se han empleado para llevar a cabo esta tarea.

Canva



Canva es una plataforma de diseño gráfico en línea que permite crear contenido visual de manera rápida, intuitiva y profesional, sin necesidad de conocimientos avanzados en diseño. Su interfaz basada en el sistema de arrastrar y soltar, junto con una amplia biblioteca de plantillas, iconos, imágenes y tipografías, la convierte en una herramienta ideal para crear elementos gráficos que complementen un proyecto, como presentaciones, carteles, logotipos o infografías.

En el presente proyecto, Canva se ha empleado para la realización del logo y materiales gráficos de este proyecto tales como la portada.

Draw.io



Draw.io es una herramienta en línea gratuita y de código abierto que permite crear diagramas de manera sencilla y visual.

En este proyecto ha resultado útil para diseñar el diagrama de casos de uso y el diagrama de entidad-relación.

Warp



Warp es una terminal moderna para macOS que permite autocompletado inteligente, historial navegable, bloques de comandos interactivos y compatibilidad con atajos modernos.

Durante el desarrollo de este proyecto, Warp ha sido una herramienta muy útil para interactuar con los sistemas de control de versiones en macOS.

Konsole



Konsole es el emulador de terminal predeterminado del entorno de escritorio KDE Plasma en sistemas Linux.

Ofrece funcionalidades como pestañas múltiples, la cual ha sido ampliamente usada de cara a abrir en una pestaña la base de datos, en otra el servidor local, en otra el explorador de archivos Ranger y en otra situarme en el proyecto Java para ejecutar los comandos de Maven.

Tecnologías

Por lo que respecta a las tecnologías, se han empleado las que se muestran a continuación:

Git



Git es un sistema de control de versiones de código abierto desarrollado por Linus Torvald que permite gestionar el código fuente de manera eficiente.

Permite realizar seguimientos de cambios, crear ramas para nuevas funcionalidades y fusionar diferentes versiones de código sin temor a perder información. En el

proyecto, Git se ha utilizado para gestionar el repositorio de código en sus distintas versiones: base de datos, backend, frontend y documentación.

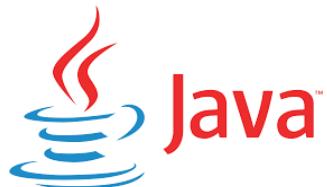
MariaDB



MariaDB es un sistema de gestión de bases de datos relacional de código abierto, desarrollado como un fork de MySQL. Actualmente, mantiene las mismas funcionales y características que este último, el cual está mantenido por Oracle.

En el contexto de este proyecto, MariaDB ha sido utilizada como la base de datos para almacenar la información de los usuarios, sesiones de entrenamiento y ejercicios.

Java



Java es un lenguaje de programación orientado a objetos que se utiliza ampliamente en el desarrollo de aplicaciones empresariales, móviles y de servidor. Es conocido por su portabilidad, ya que las aplicaciones escritas en Java pueden ejecutarse en cualquier dispositivo que tenga instalada una Máquina Virtual Java (JVM).

En este proyecto, Java ha sido empleado para desarrollar la parte del backend, donde se gestionan las solicitudes, la lógica de negocio y la conexión con la base de datos mediante MariaDB.

Spring



Spring Boot es un framework de Java que simplifica el proceso de creación de aplicaciones empresariales. Su principal objetivo es aumentar la productividad de los desarrolladores al proporcionar configuraciones predeterminadas sensatas y un conjunto de herramientas que permiten comenzar rápidamente cualquier proyecto.

Ofrece herramientas como inyección de dependencias, control de transacciones y seguridad, entre otras.

En este proyecto, se ha utilizado Spring Boot para agilizar el desarrollo del backend, simplificando la configuración y el despliegue, lo que ha permitido centrarse en la lógica de negocio y la integración con otras tecnologías de forma eficiente.

Maven



Maven es una herramienta de gestión y construcción de proyectos para aplicaciones Java, que facilita la automatización de tareas como la compilación, el empaquetado y la ejecución de pruebas.

En el proyecto, se ha empleado para hacer operaciones como el Clean y el Install para poder probar el código.

JPA



JPA es una especificación de Java para el mapeo de objetos Java a bases de datos relacionales.

JPA se ha usado en este proyecto para gestionar la persistencia de los objetos en la base de datos, simplificando las operaciones CRUD y optimizando la interacción entre el código de la aplicación y MariaDB.

Lombok



Lombok es una biblioteca Java que elimina el código repetitivo mediante anotaciones, lo que mejora la legibilidad, acelera el flujo de trabajo y aumenta la productividad.

Al utilizar Lombok en este proyecto, se ha simplificado el proceso de creación de clases Java al eliminar la necesidad de escribir constructores, *getters*, *setters*, *toString* y *equals*.

MapStruct



MapStruct es una herramienta de mapeo de objetos Java que facilita la conversión entre objetos de diferentes clases, generando automáticamente el código necesario para las conversiones de tipo.

En este proyecto, MapStruct se ha utilizado para transformar los objetos de JPA al modelo de negocio y del modelo de negocio al modelo de la capa de presentación.

CORS

CORS (**Cross-Origin Resource Sharing**) es un mecanismo que permite a los recursos restringidos en una página web ser solicitados desde otro dominio. Este proceso es importante cuando una aplicación frontend interactúa con un backend alojado en un servidor diferente. En este proyecto, se ha implementado CORS para permitir que la aplicación iOS (frontend) realice peticiones seguras a la API backend (Java), garantizando que las solicitudes entre dominios sean correctamente gestionadas sin comprometer la seguridad de la aplicación.

Postman



Postman es una herramienta de desarrollo que facilita la prueba de los diferentes métodos de una API REST. Gracias a Postman, se ha podido comprobar que la API devolvía aquello que se le solicitaba en cada método (GET) y que se podían realizar los cambios pertinentes (POST, PUT y DELETE).

Swift



Swift es un lenguaje de programación creado por Apple para el desarrollo de aplicaciones en sus plataformas, como iOS, macOS, watchOS y tvOS. Swift es conocido por su velocidad y seguridad, ofreciendo características avanzadas como la inferencia de tipos y la programación funcional.

En este proyecto, Swift se ha utilizado para desarrollar el frontend de la aplicación iOS, permitiendo crear una interfaz de usuario atractiva y eficiente para gestionar sesiones de entrenamiento y monitorear el progreso de los usuarios.

SwiftUI



SwiftUI es un framework declarativo de interfaz de usuario para desarrollar aplicaciones en plataformas Apple. Facilita la creación de interfaces de usuario complejas mediante el uso de una sintaxis simple y concisa.

En este proyecto, SwiftUI ha sido la herramienta principal para desarrollar la interfaz de usuario, permitiendo una integración fluida con el backend, mientras que su capacidad de previsualización instantánea ha agilizado el proceso de diseño y desarrollo de las pantallas.

SFSymbols



SFSymbols es un conjunto de iconos vectoriales diseñados por Apple que se integran nativamente en las aplicaciones de iOS. Estos iconos están optimizados para adaptarse a cualquier tamaño de pantalla y mantener una apariencia coherente en todas las aplicaciones.

En este proyecto, se ha utilizado SFSymbols de cara a enriquecer la interfaz de usuario de la aplicación, incorporando iconos que mejoran la usabilidad y la estética de la aplicación sin tener que crear iconos personalizados.

Asimismo, al ser iconos desarrollados por la propia Apple, son coherentes con el diseño del resto del sistema operativo.

Implementación

Análisis

En este apartado se especifican las diferentes partes que conforman la base de lo que será la propia aplicación. Se explicarán, por tanto, los requerimientos iniciales de la aplicación, los actores del sistema, el diagrama de casos de uso, la descripción de cada caso y el diagrama de actividades de cada caso de uso.

Requerimientos iniciales

Los requisitos de los que consta la aplicación son los siguientes:

Usuario

En la pantalla inicial, se dan dos opciones:

- Login de cada usuario de la aplicación.
- Registro de cada usuario de la aplicación.

Asimismo, una vez se ha iniciado sesión, el usuario debe ser capaz de modificar sus datos y de poder borrar su cuenta si así lo desea.

Ejercicios

El usuario de la aplicación tendrá un listado general de ejercicios por defecto, los cuales puede visualizar detalladamente, modificar o incluso eliminar si así lo desea.

Asimismo, puede crear sus propios ejercicios (y también eliminarlos posteriormente) rellenando los campos necesarios.

Sesiones

También será capaz de, al igual que con los ejercicios, visualizar sesiones predefinidas, modificarlos y eliminarlos; al mismo tiempo que podrá realizar esos mismos pasos con las sesiones que cree.

En dichas sesiones se podrán añadir o eliminar ejercicios.

Entrenamientos

Finalmente, está la opción de entrenamientos, el cual incluye la sesión realizada con la fecha y el tiempo que le ha llevado hacerla para poder tener un seguimiento del proceso de entrenamiento.

Podrá realizar las mismas acciones que con las sesiones. Por defecto, habrá entrenamientos simulados por defecto para que pueda ver el funcionamiento, los cuales se recomienda borrar una vez entienda el funcionamiento de este apartado.

Actores del sistema

Los actores del sistema son los siguientes:

Usuario no identificado

Es el usuario que no ha iniciado sesión en la aplicación. Solamente podrá iniciarla o registrarse en la aplicación.

Usuario identificado

Es el usuario que ha iniciado sesión en la aplicación.

Tiene acceso a la visualización, edición, creación y borrado de los ejercicios, las sesiones y los entrenamientos que se han explicado en el apartado anterior.

Diagrama de casos de uso



Figura 9. Diagrama de casos de uso.

Descripción de cada caso de uso

CASO-1-001	Registrar usuario
Actor	Usuario no identificado.
Descripción	El usuario se registra en la aplicación llenando el formulario.
Precondiciones	El usuario no debe existir en la base de datos. El correo electrónico no puede estar registrado.
Secuencia normal	<ol style="list-style-type: none">1. Clica la opción de registrarse.2. Rellena los datos.3. Envía el formulario.4. El sistema valida los datos introducidos.5. El sistema guarda al usuario.6. Se vuelve a la pantalla principal para que inicie sesión.
Postcondiciones	Se registra y puede iniciar sesión.
Excepciones	<ul style="list-style-type: none">- Formato no correcto.- Usuario ya existente.

Tabla 1. Caso de uso 1.

CASO-1-002	Inicio de sesión
Actor	Usuario no identificado.
Descripción	El usuario inicia sesión rellenando los campos de correo electrónico y contraseña.
Precondiciones	El usuario no está identificado.
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario escribe su correo electrónico. 2. El usuario escribe su contraseña. 3. El sistema valida los datos introducidos por el usuario. 4. El sistema identifica al usuario. 5. El sistema redirige al usuario a la página principal.
Postcondiciones	El usuario está identificado y puede acceder a todos los recursos de la aplicación.
Excepciones	Error en el correo electrónico y/o contraseña introducidos.

Tabla 2. Caso de uso 2.

CASO-2-001	Ver listado de entrenamientos
Actor	Usuario identificado.
Descripción	El usuario puede visualizar un listado con todas las sesiones de entrenamiento que ha completado. Es la pantalla principal.
Precondiciones	Iniciar sesión.
Secuencia normal	<ol style="list-style-type: none"> 1. Tras iniciarse sesión. 2. Es la pantalla principal de la aplicación. 3. También puede acceder pulsando en el <i>Tab</i> correspondiente del menú inferior.
Postcondiciones	Puede crear, ver o borrar una sesión de entrenamiento.
Excepciones	No hay sesiones de entrenamiento disponibles. Sale un listado vacío. Puede crear una nueva sesión de entrenamiento.

Tabla 3. Caso de uso 3.

CASO-2-002	Ver detalles del entrenamiento
Actor	Usuario identificado.
Descripción	El usuario puede ver todos los campos relacionados con el entrenamiento.
Precondiciones	Acceder desde el listado de entrenamientos.
Secuencia normal	<ol style="list-style-type: none"> 1. El usuario pulsa sobre el entrenamiento deseado. 2. Se carga el entrenamiento con los detalles.
Postcondiciones	Volver al listado de entrenamientos. Modificar el entrenamiento. Borrar el entrenamiento.
Excepciones	No se puede acceder a los detalles de una sesión que no existe.

Tabla 4. Caso de uso 4.

CASO-2-003	Eliminar entrenamiento
Actor	Usuario identificado.
Descripción	Borra del registro ese entrenamiento.
Precondiciones	Estar en los detalles del entrenamiento o en el listado de entrenamientos.
Secuencia normal	<p>Opción A:</p> <ol style="list-style-type: none"> 1. El usuario pulsa sobre un entrenamiento del listado. 2. Desciende al final de la sesión de entrenamiento. 3. Le da a borrar. 4. Confirma el borrado. <p>Opción B:</p> <ol style="list-style-type: none"> 1. El usuario está en el listado de entrenamientos. 2. Desliza en una sesión hacia la izquierda. 3. Pulsa la opción de borrar. 4. Confirma el borrado.
Postcondiciones	Se carga el listado de sesiones sin ya ese entrenamiento.
Excepciones	Tiene que existir la sesión.

Tabla 5. Caso de uso 5.

CASO-2-004	Modificar entrenamiento
Actor	Usuario identificado.
Descripción	El usuario puede modificar los valores que ya existían en el entrenamiento.
Precondiciones	Estar en los detalles del mismo.
Secuencia normal	<ol style="list-style-type: none"> 1. Acceder al detalle del entrenamiento. 2. Ir a la parte inferior 3. Clicar en modificar. 4. Modificar los campos deseados. 5. Guardar los cambios.
Postcondiciones	Se accede a los detalles de dicho entrenamiento con los valores ya modificados.
Excepciones	Tiene que existir la sesión.

Tabla 6. Caso de uso 6.

CASO-2-001	Ver listado de ejercicios
Actor	Usuario identificado.
Descripción	El usuario ve el listado de todos los ejercicios. Puede filtrar en el buscador por nombre o tipo.
Precondiciones	<ol style="list-style-type: none"> 1. Haber iniciado sesión. 2. Pulsar en su <i>Tab</i> correspondiente.
Secuencia normal	Tras pulsar en su <i>Tab</i> correspondiente, sale el listado automáticamente.
Postcondiciones	Puede crear, ver o borrar ejercicios.
Excepciones	No haber iniciado sesión.

Tabla 7. Caso de uso 7.

CASO-2-002	Ver detalles del ejercicio
Actor	Usuario identificado.
Descripción	Se pueden visualizar todas las características pertinentes a un ejercicio.
Precondiciones	Estar en el listado de ejercicios.
Secuencia normal	<ol style="list-style-type: none"> 1. Estar en el listado de ejercicios. 2. Pulsar sobre un ejercicio concreto.
Postcondiciones	Volver al listado, modificar el ejercicio o borrar el ejercicio.
Excepciones	No haber iniciado sesión.

Tabla 8. Caso de uso 8.

CASO-2-003	Borrar ejercicio
Actor	Usuario identificado.
Descripción	Borra del registro ese ejercicio.
Precondiciones	Estar en los detalles del ejercicio o en el listado de ejercicios.
Secuencia normal	<p>Opción A:</p> <ol style="list-style-type: none"> 1. El usuario pulsa sobre un ejercicio del listado. 2. Desciende al final del ejercicio. 3. Le da a borrar. 4. Confirma el borrado. <p>Opción B:</p> <ol style="list-style-type: none"> 1. El usuario está en el listado de ejercicios. 2. Desliza en un ejercicio hacia la izquierda. 3. Pulsa la opción de borrar. 4. Confirma el borrado.
Postcondiciones	Se carga el listado de ejercicios sin ya ese ejercicio.
Excepciones	Tiene que existir el ejercicio.

Tabla 9. Caso de uso 9.

CASO-2-004	Modificar ejercicio
Actor	Usuario identificado.
Descripción	El usuario puede modificar los valores que ya existían en el ejercicio.
Precondiciones	Estar en los detalles del mismo.
Secuencia normal	<ol style="list-style-type: none"> 1. Acceder al detalle del ejercicio. 2. Ir a la parte inferior 3. Clicar en modificar. 4. Modificar los campos deseados. 5. Guardar los cambios.
Postcondiciones	Se accede a los detalles de dicho ejercicio con los valores ya modificados.
Excepciones	Tiene que existir el ejercicio.

Tabla 10. Caso de uso 10.

CASO-3-001	Ver listado de sesiones
Actor	Usuario identificado.
Descripción	El usuario ve el listado de todas las sesiones. Puede filtrar en el buscador por nombre o tipo.
Precondiciones	<ol style="list-style-type: none">1. Haber iniciado sesión.2. Pulsar en su <i>Tab</i> correspondiente.
Secuencia normal	Tras pulsar en su <i>Tab</i> correspondiente, sale el listado automáticamente.
Postcondiciones	Puede crear, ver o borrar sesiones.
Excepciones	No haber iniciado sesión.

Tabla 11. Caso de uso 11.

CASO-3-002	Ver detalles de la sesión
Actor	Usuario identificado.
Descripción	Se pueden visualizar todas las características pertinentes a una sesión.
Precondiciones	Estar en el listado de sesiones.
Secuencia normal	<ol style="list-style-type: none"> 1. Estar en el listado de sesiones. 2. Pulsar sobre un ejercicio concreto.
Postcondiciones	Volver al listado, modificar el ejercicio o borrar la sesión.
Excepciones	No haber iniciado sesión.

Tabla 12. Caso de uso 12.

CASO-2-003	Borrar sesión
Actor	Usuario identificado.
Descripción	Borra del registro esa sesión.
Precondiciones	Estar en los detalles de la sesión o en el listado de sesiones.
Secuencia normal	<p>Opción A:</p> <ol style="list-style-type: none"> 1. El usuario pulsa sobre una sesión del listado. 2. Desciende al final de la sesión. 3. Le da a borrar. 4. Confirma el borrado. <p>Opción B:</p> <ol style="list-style-type: none"> 1. El usuario está en el listado de sesiones. 2. Desliza en una sesión hacia la izquierda. 3. Pulsa la opción de borrar. 4. Confirma el borrado.
Postcondiciones	Se carga el listado de sesiones sin ya esa sesión.
Excepciones	Tiene que existir la sesión.

Tabla 13. Caso de uso 13.

CASO-3-004	Modificar sesión
Actor	Usuario identificado.
Descripción	El usuario puede modificar los valores que ya existían en la sesión.
Precondiciones	Estar en los detalles de la misma.
Secuencia normal	<ol style="list-style-type: none"> 1. Acceder al detalle de la sesión. 2. Ir a la parte inferior 3. Clicar en modificar. 4. Modificar los campos deseados. 5. Guardar los cambios.
Postcondiciones	Se accede a los detalles de dicha sesión con los valores ya modificados.
Excepciones	Tiene que existir la sesión.

Tabla 14. Caso de uso 14.

Diseño

Respecto a lo que es el diseño de la aplicación, lo separaré en los siguientes apartados: logo, diagrama entidad-relación, base de datos, backend y frontend.

Logo

En primer lugar, lo que vemos inicialmente cuando buscamos una aplicación en la App Store es el logo. Asimismo, cuando vamos a abrirla, también es el logo. Por tanto, la idea es lograr un logo atractivo a la par que coherente con el logo de BeSport24 (Figura 2).

Fruto de ello, el logo es el siguiente:

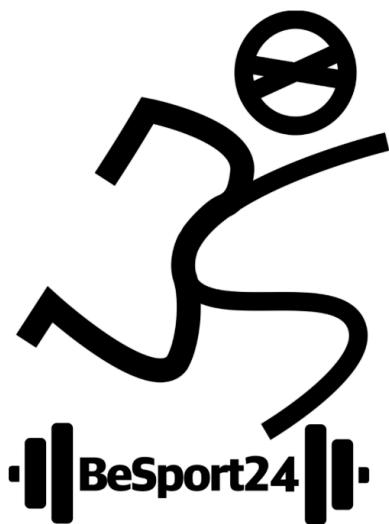


Figura 10. Logo de BeSport24Training.

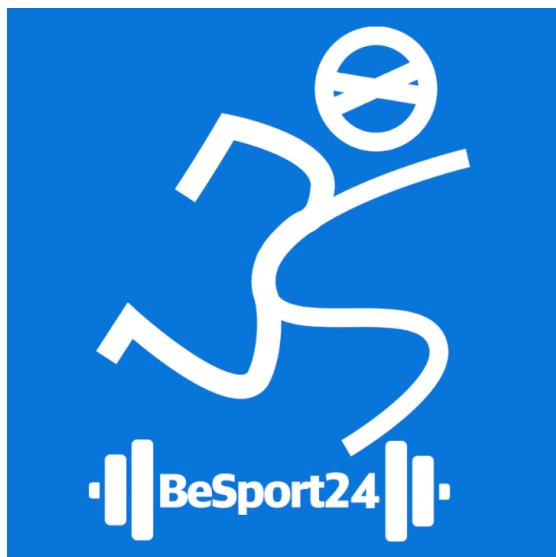


Figura 11. Logo de BeSport24Training invertido con el fondo del color primario de la aplicación.

Tal y como podemos observar, se han diseñado dos vertientes del mismo logo. El primero de ellos es el diseño principal que se usará dentro de la aplicación, mientras que el segundo será el empleado como ícono de la aplicación y también el que aparecerá en la *Launch Screen*, es decir, el nombre que le da Apple a la pantalla que se muestra al abrirse una aplicación.

La idea del logo es mantener la estructura del logo original, pero dándole un aspecto más moderno y atractivo.

Fruto de ello, mantiene la figura principal, la cual se compone de una B y una S. La B muestra un diseño más recto, simbolizando que este proyecto siempre se ha basado en la evidencia científica más actual sobre los beneficios del ejercicio en la salud de las personas. Por el contrario, la S se muestra más libre, ya que representa el arte de reunir todos los conocimientos de la ciencia para poder llevarlos al mundo real y que logra que se avance en las aplicaciones prácticas de los planes de entrenamiento.

La cabeza parece mostrar una cinta de pelo y el propio pelo, pero a su vez es la unión de un 2 y de un 4, formando el 24, el cual simboliza que un estilo de vida saludable se mantiene con las acciones que realizamos a lo largo de cada día y no solamente con la hora de entrenamiento que hagamos dos o tres días a la semana.

Finalmente, la modificación que se ha añadido es la de reducir el nombre de BeSport24 para dar cabida a unas mancuernas en la cual el nombre del proyecto es la barra. Con ello, se simboliza que el mejor enfoque de un entrenamiento físico es aquel que engloba la parte de fuerza y la parte cardiovascular.

Diagrama entidad-relación

En este aspecto, se ha buscado simplificar en la medida de lo posible el diagrama entidad-relación para poder hacerlo práctico a la hora de realizar un TFG, si bien la idea en el futuro es poder ampliarlo partiendo de esta base.

Aquí podemos apreciar las propiedades de las que se conforma cada entidad, la clave primaria de cada una de ellas y las cardinalidades que posee. Asimismo, se muestra qué propiedades son opcionales y también la relación de las sesiones con los ejercicios con sus propias propiedades, la cual formará una tabla independiente en la base de datos.

La lógica que persigue la base de datos es que en la aplicación existan usuarios, los cuales pueden tener sesiones y entrenamientos. Las sesiones son prototipos de aquello que realizar, mientras que los entrenamientos son los que ya se han realizado. Por ello, un entrenamiento está compuesto de la sesión, mientras que una sesión puede conformar más de un entrenamiento. Pongamos un ejemplo para entenderlo mejor: podemos tener una sesión que consista en 5' de carrera continua, 30' corriendo en primer umbral y 5' de vuelta a la calma. Esta sesión, si la hacemos todos los lunes de un mes, conformará 4 entrenamientos en los cuales se usa esa misma sesión.

Estas sesiones, evidentemente, tendrán ejercicios, que pueden ser como los explicados en el ejemplo anterior. Ahora bien, esos ejercicios pueden ser de distintas modalidades, por lo que los ejercicios están formados por tipos y subtipos de ejercicios.

El tipo de ejercicios es si es un ejercicio de fuerza, un ejercicio cardiovascular o un ejercicio de movilidad; mientras que el subtipo es la región a la que afecta. De nuevo, pondremos ejemplos para entenderlo mucho mejor: una prensa de piernas es un ejercicio, valga la redundancia, de piernas (subtipo) y también de fuerza (tipo); mientras que la bicicleta (tipo) también involucra a las piernas (subtipo).

BeSport24Training

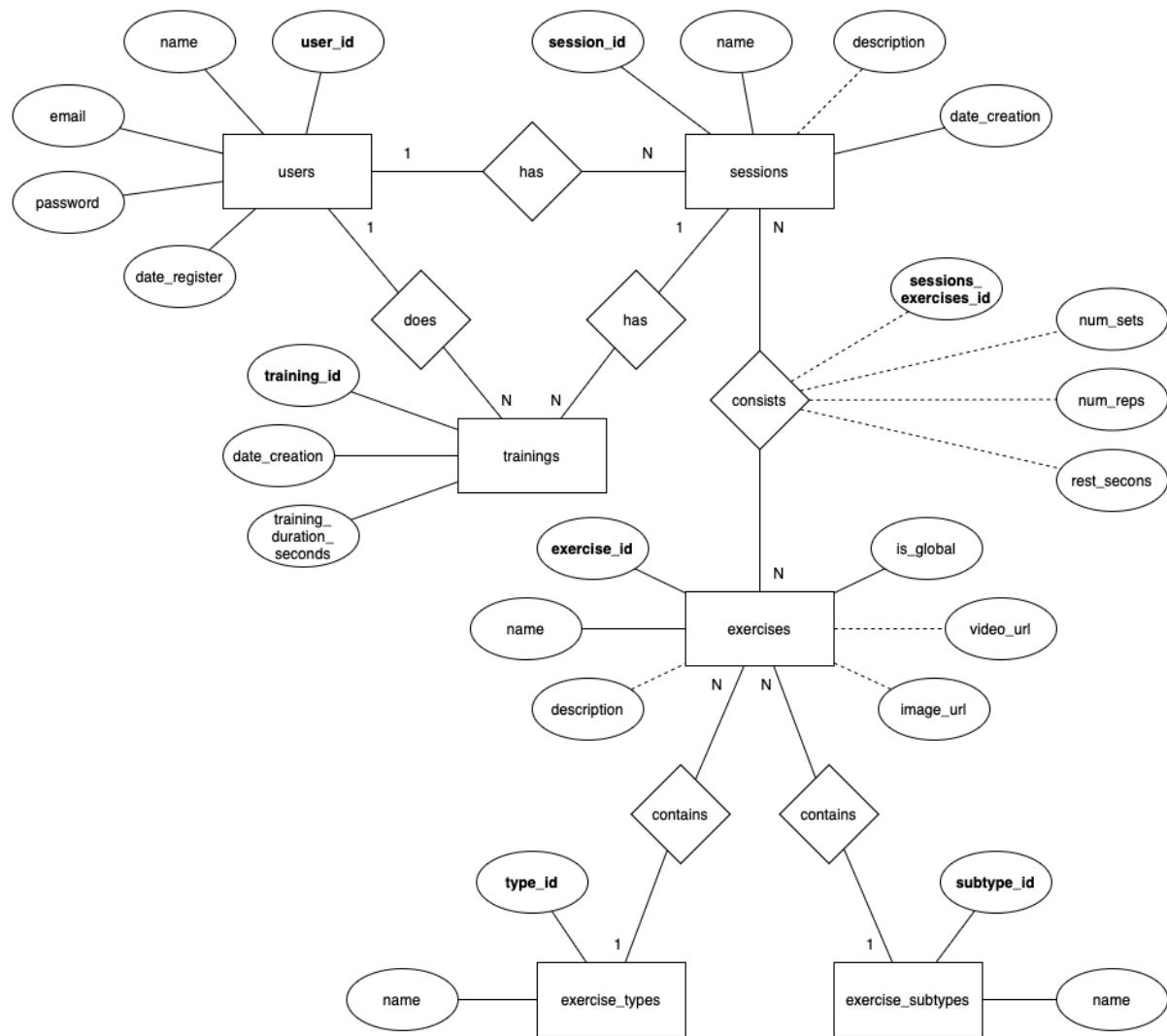


Figura 12. Diagrama entidad-relación.

Base de datos

Una vez realizado y revisado el diagrama de entidad-relación, es el turno de realizar la base de datos y sus respectivas tablas

En la siguiente figura tenemos las tablas que se derivan del diagrama:

Tables_in_besport24
exercise_subtypes
exercise_types
exercises
sessions
sessions_exercises
trainings
users

Figura 13. Tablas que conforman la base de datos.

A continuación, se muestran, por orden alfabético, para seguir con la estructura de la Figura 13 la descripción de cada una de las tablas usando el comando `desc nombre_de_la_tabla` de MariaDB.

Field	Type	Null	Key	Default	Extra
subtype_id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(100)	NO	UNI	NULL	

Figura 14. Descripción de las tablas `exercise_subtypes` y `exercise_types`.

En esta tabla simplemente se introducen el código y el nombre del tipo o del subtipo, dependiendo de la tabla.

Field	Type	Null	Key	Default	Extra
exercise_id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(100)	NO		NULL	
type_id	int(11)	NO	MUL	NULL	
subtype_id	int(11)	YES	MUL	NULL	
is_global	tinyint(1)	YES		1	
description	longtext	YES		NULL	
video_url	varchar(255)	YES		NULL	
image_url	varchar(255)	YES		NULL	

Figura 15. Descripción de la tabla exercises.

Aquí podemos observar la tabla de los ejercicios, con su clave foránea para las tablas de los tipos y subtipos vistas anteriormente.

Field	Type	Null	Key	Default	Extra
session_id	int(11)	NO	PRI	NULL	auto_increment
user_id	int(11)	NO	MUL	NULL	
name	varchar(100)	NO		NULL	
description	text	YES		NULL	
date_creation	timestamp	NO		NULL	

Figura 16. Descripción de la tabla sessions.

En esta figura vemos la estructura de la tabla de las sesiones, las cuales pertenecen a un usuario concreto de cara a que cada usuario de la aplicación pueda tener sus sesiones correspondientes.

Field	Type	Null	Key	Default	Extra
sessions_exercises_id	int(11)	NO	PRI	NULL	auto_increment
session_id	int(11)	NO	MUL	NULL	
exercise_id	int(11)	NO	MUL	NULL	
num_sets	int(11)	YES		NULL	
num_reps	int(11)	YES		NULL	
rest_seconds	int(11)	YES		NULL	

Figura 17. Descripción de la tabla sessions_exercises.

Esta tabla corresponde a la tabla que nace de la unión de las tablas de sesiones y ejercicios. En ella, además, se incluyen los atributos propios de esta tabla para reflejar con mayor claridad qué queremos hacer en cada sesión con cada uno de los ejercicios que la conforman.

Field	Type	Null	Key	Default	Extra
training_id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(100)	NO		NULL	
user_id	int(11)	NO	MUL	NULL	
session_id	int(11)	NO	MUL	NULL	
date_creation	timestamp	NO		NULL	
training_duration_seconds	int(11)	YES		NULL	

Figura 18. Descripción de la tabla trainings.

En esta tabla tenemos la tabla de entrenamientos, con su relación con los usuarios para que, al igual que en las sesiones, los entrenamientos sean exclusivos de cada uno de los usuarios de la aplicación. Además, también tiene la relación con la sesión de la cual se compone dicho entrenamiento, junto a sus campos propios.

Field	Type	Null	Key	Default	Extra
user_id	int(11)	NO	PRI	NULL	auto_increment
name	varchar(100)	NO		NULL	
email	varchar(255)	NO	UNI	NULL	
password	varchar(255)	NO		NULL	
date_register	timestamp	NO		NULL	

Figura 19. Descripción de la tabla users.

Por último por orden alfabético tenemos la tabla de usuarios, con los campos necesarios. Esta es una de las tablas que, en el futuro y como propuesta de mejora, se ampliará de cara a ser más completa. Por ejemplo, se introducirían parámetros como la altura, el peso, el IMC, el porcentaje de grasa corporal, el porcentaje de masa muscular, las actividades deportivas que realiza el usuario, los registros de mejores marcas en ejercicios de fuerza y los mejores registros en pruebas como 5K, 10K media maratón o maratón.

Back End

El siguiente apartado en la línea cronológica ha sido la realización del backend de la aplicación. En el mismo, se ha seguido la arquitectura por capas explicada por el profesor César Guijarro. Muestra de ello es el *scaffolding* que se muestra en la Figura 20, el cual resultará familiar a cualquier compañero que lea este TFG.

BeSport24Training

```
└─ bs24 [demo] ~/github_projects/DAW-2/TFG/backend/bs24
    ├─ .idea
    ├─ .mvn
    └─ src
        ├─ main
        │   └─ java
        │       ├─ com.bs24.demo
        │       │   ├─ common
        │       │   │   └─ annotation
        │       │   │       @ DomainService
        │       │   │       @ DomainTransactional
        │       │   │       @ DomainUseCase
        │       │   └─ config
        │       │       © PropertiesConfig
        │       └─ cors
        │           © CorsConfig
        └─ error
            © ApiExceptionHandler
            © ErrorMessage
        └─ exception
            ⚡ ResourceAlreadyExistsException
            ⚡ ResourceNotFoundException
            ⚡ ValidationException
        └─ locale
            © CustomLocaleChangeInterceptor
            © LanguageUtils
            © LocaleConfig
        └─ controller
            ├─ common
            │   © PaginatedResponse
            └─ user
                └─ webmodel
                    Ⓜ .gitkeep
                    Ⓜ ExerciseCollection
                    Ⓜ ExerciseCollectionMapper
                    Ⓜ ExerciseDetail
                    Ⓜ ExerciseDetailMapper
                    Ⓜ ExerciseSubtypeCollection
```

Figura 20. Scaffolding del Back-End.

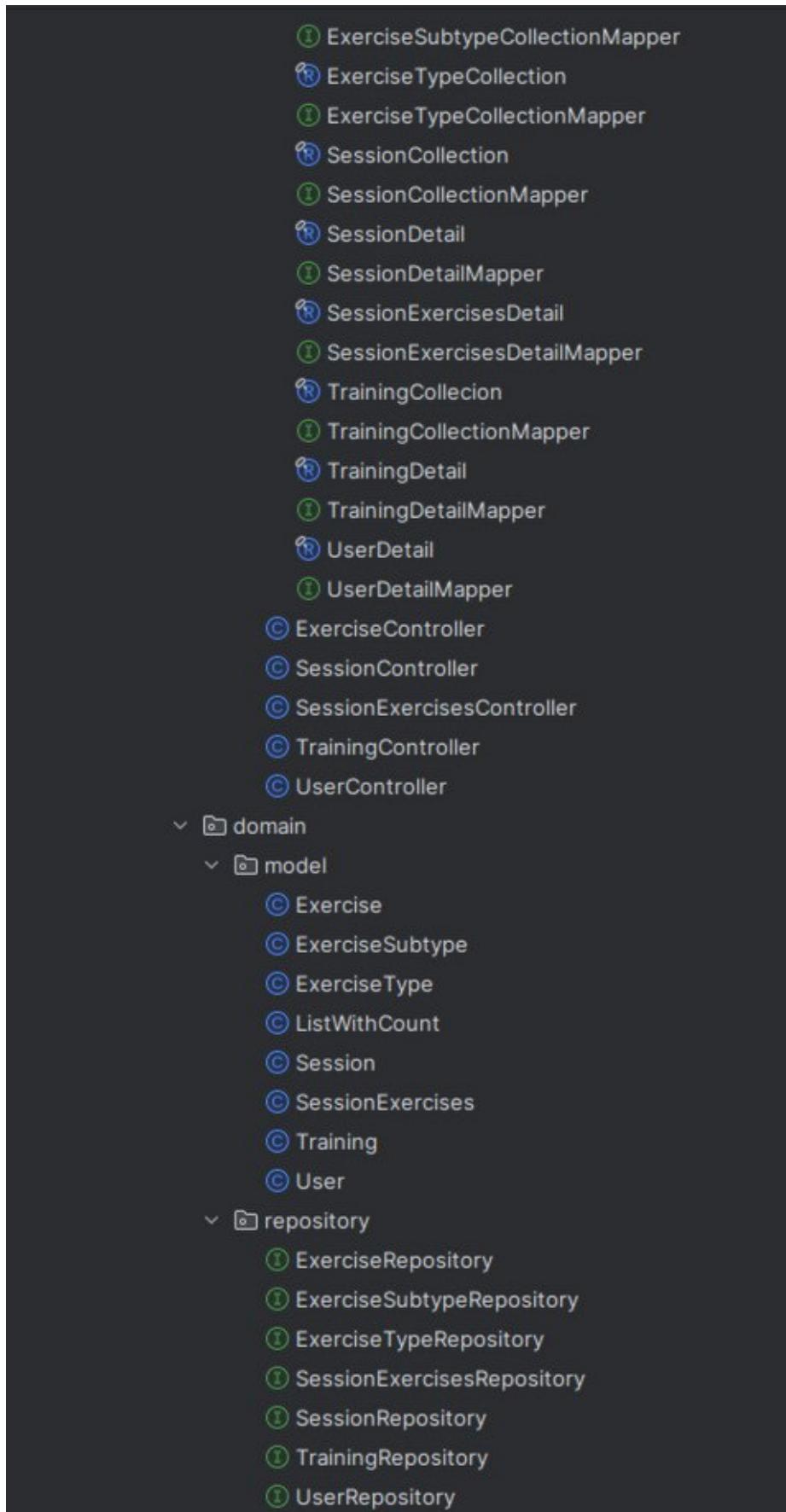


Figura 20. Scaffolding del Back-End (continuación).

BeSport24Training

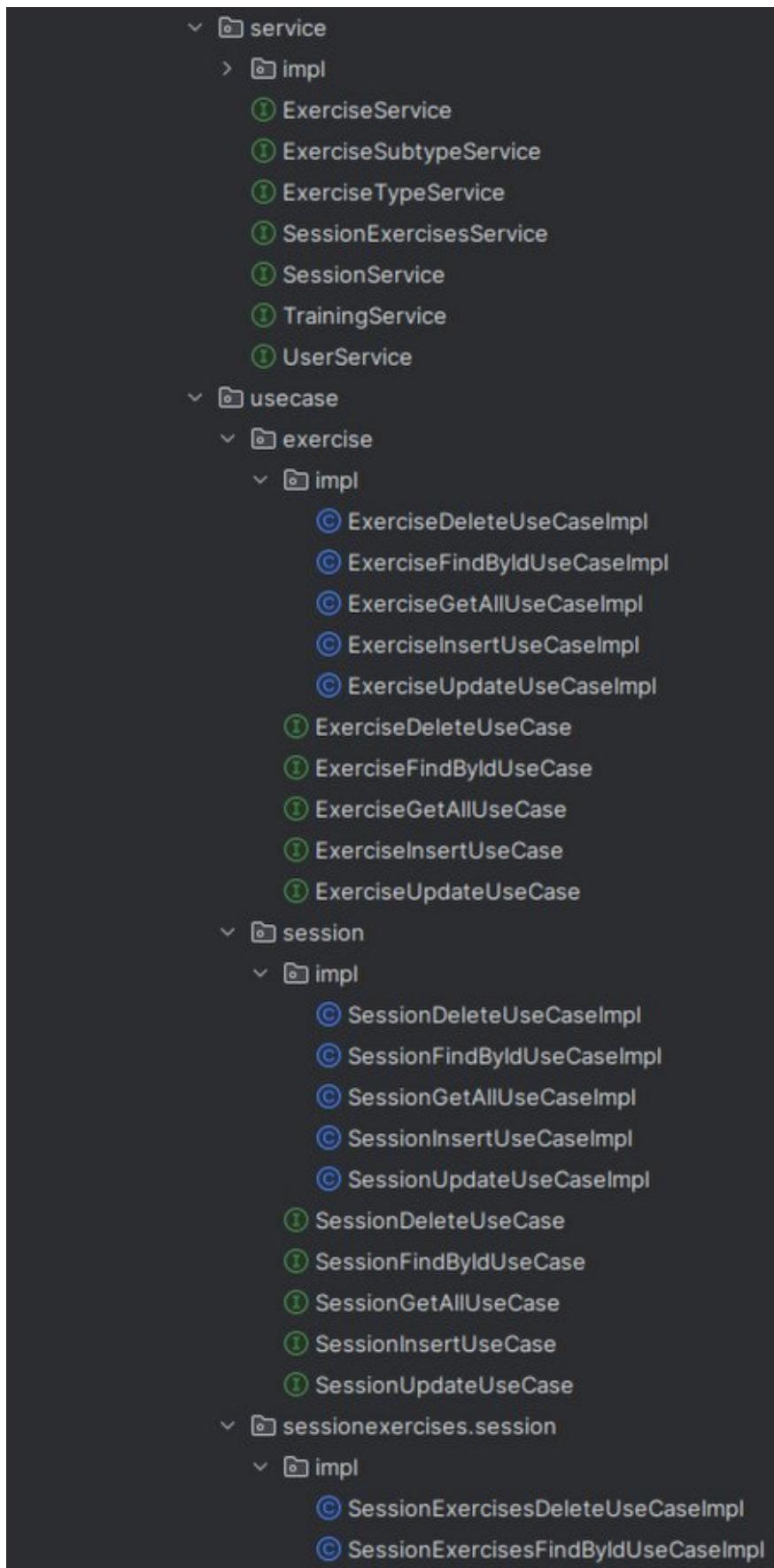


Figura 20. Scaffolding del Back-End (continuación).

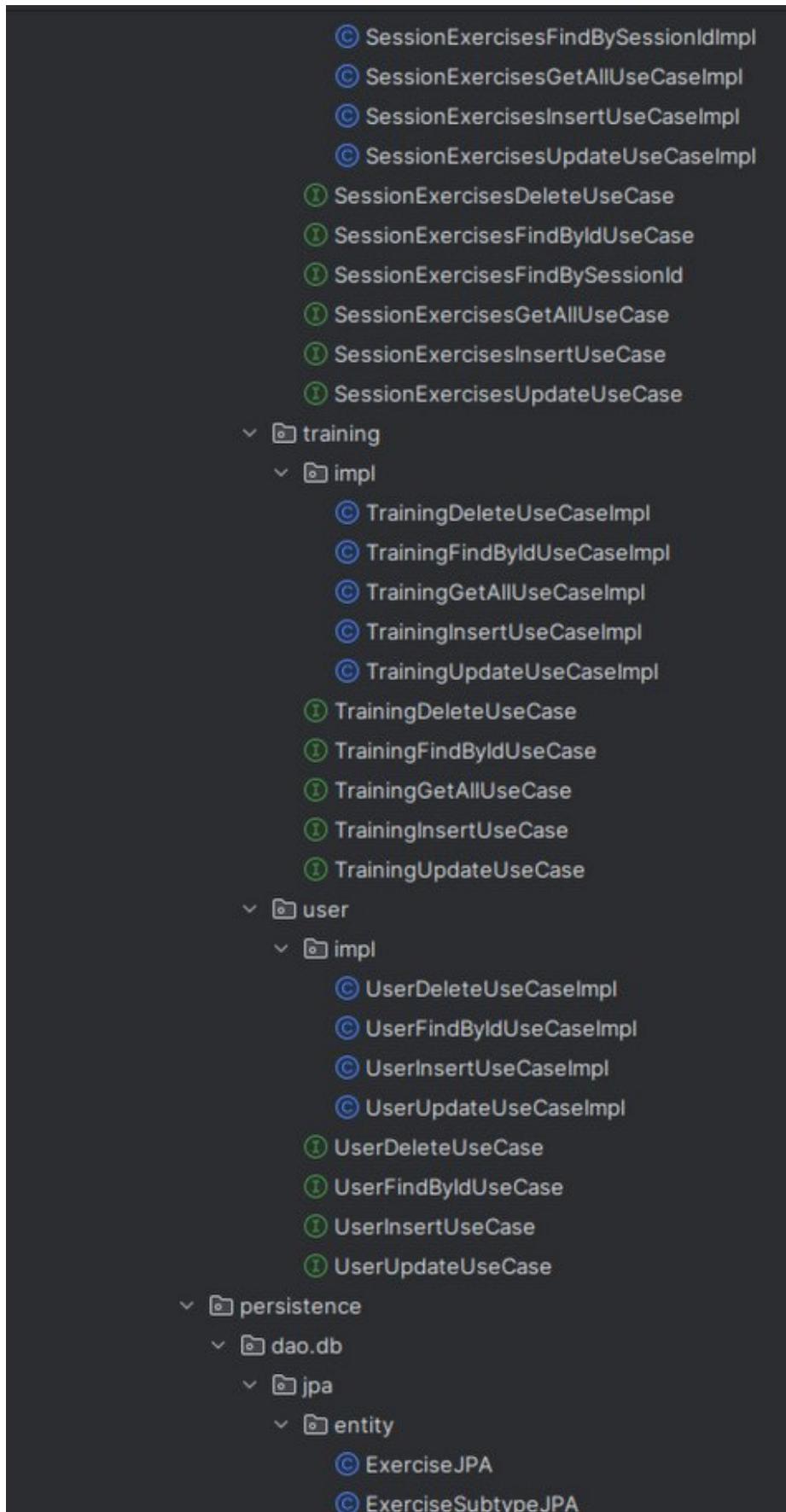


Figura 20. Scaffolding del Back-End (continuación).

```
    Ⓜ ExerciseTypeJPA
    Ⓜ SessionExercisesJPA
    Ⓜ SessionJPA
    Ⓜ SessionSaveJPA
    Ⓜ TrainingJPA
    Ⓜ UserJPA
    ↴ Ⓛ mapper
        ⓘ ExerciseJPAMapper
        ⓘ ExerciseSubtypeJPAMapper
        ⓘ ExerciseTypeJPAMapper
        ⓘ SessionExercisesJPAMapper
        ⓘ SessionJPAMapper
        ⓘ TrainingJPAMapper
        ⓘ UserJPAMapper
    ↴ Ⓛ repository
        ⓘ ExerciseJPARespository
        ⓘ ExerciseSubtypeJPARespository
        ⓘ ExerciseTypeJPARespository
        ⓘ SessionExercisesJPARespository
        ⓘ SessionJPARespository
        ⓘ TrainingJPARespository
        ⓘ UserJPARespository
    Ⓜ ExerciseDaoDbImpl
    Ⓜ ExerciseSubtypeDaoDbImpl
    Ⓜ ExerciseTypeDaoDbImpl
    Ⓜ SessionDaoDbImpl
    Ⓜ SessionExercisesDaoDbImpl
    Ⓜ TrainingDaoDbImpl
    Ⓜ UserDaoDbImpl
    ⓘ ExerciseDaoDb
    ⓘ ExerciseSubtypeDaoDb
    ⓘ ExerciseTypeDaoDb
    ⓘ GenericDaoDb
    ⓘ SessionDaoDb
    ⓘ SessionExercisesDaoDb
    ⓘ TrainingDaoDb
    ⓘ UserDaoDb
    ↴ Ⓛ repository.imp
```

Figura 20. Scaffolding del Back-End (continuación).

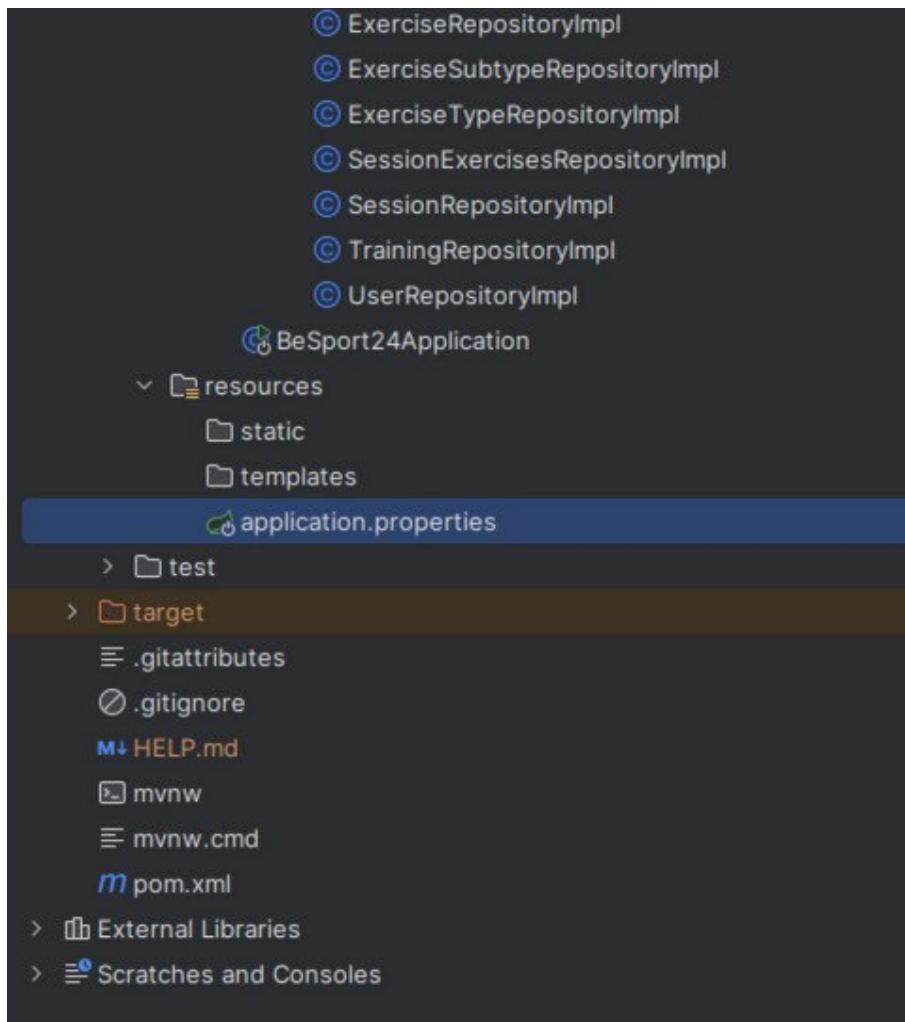


Figura 20. Scaffolding del Back-End (continuación).

Pasando a visualizar aspectos importantes de lo que se ha realizado en este apartado, podemos destacar las siguientes funcionalidades:

En el Application Properties se establecen las variables globales como la URL base para, según el entorno de ejecución, emplear una u otra URL. También destaca que JPA no borre ni cree la base de datos de cero y que se utilice la base de datos antes explicada que se encuentra en la URL citada en la segunda línea.

```
1  spring.application.name=BeSport24
2  spring.datasource.url=jdbc:mariadb://localhost:3306/besport24
3  spring.datasource.username=root
4  spring.datasource.password=root
5
6  app.language.default=es
7  app.base.url=http://localhost:8080
8  app.pageSize.default=20
9  app.api.path=/api
10
11 # JPA
12 spring.jpa.hibernate.ddl-auto=none
13 spring.jpa.show-sql=true
```

Figura 21. Application Properties.

Paquete common

Continuando con las carpetas en el orden en el que se encuentran en el *scaffolding*, la primera de ellas es la carpeta *common*, la cual aglutina funcionalidades que se emplean en diferentes partes de la aplicación.

Entre ellas tenemos el paquete de las anotaciones, las cuales se usan para aislar la capa de negocio de nuestra aplicación de Spring para que sea más independiente y, si es necesario cambiarlo, sea más rápido y sencillo realizarlo, pues solo habría que modificar estas anotaciones y no ir yendo a cada clase. Como ejemplo, se pone la anotación de @Service de Spring que usamos como @DomainService:

```
1  package com.bs24.demo.common.annotation;
2
3  import org.springframework.stereotype.Service;
4
5  import java.lang.annotation.ElementType;
6  import java.lang.annotation.Retention;
7  import java.lang.annotation.RetentionPolicy;
8  import java.lang.annotation.Target;
9
10 @Target(ElementType.TYPE)
11 @Retention(RetentionPolicy.RUNTIME)
12 @Service
13 public @interface DomainService {
14 }
```

Figura 22. Ejemplo de anotación.

El segundo paquete está por defecto ya que es uno de los que añadió César en su modelo por capas, por lo que se implementó en sus inicios por si fuera necesario.

Posteriormente tenemos el de CORS para poder usar la aplicación desde el frontend:

```
1 package com.bs24.demo.common.cors;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.web.cors.CorsConfiguration;
6 import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
7 import org.springframework.web.filter.CorsFilter;
8
9 @Configuration
10 public class CorsConfig {
11
12     @Bean
13     public CorsFilter corsFilter() {
14         UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
15         CorsConfiguration config = new CorsConfiguration();
16         config.addAllowedOrigin("http://localhost:4200");
17         config.addAllowedMethod("*");
18         config.addAllowedHeader("*");
19         config.setAllowCredentials(true);
20         source.registerCorsConfiguration("/**", config);
21         return new CorsFilter(source);
22     }
23 }
```

Figura 23. Fichero de CORS.

Como la aplicación se va a usar mediante una VPN y un servidor alojados en una Raspberry Pi, se ha dejado el puerto usado por defecto por el front para que pueda acceder.

También tenemos el apartado de errores, el cual se ha dejado tal y como nos lo explicó César, es decir, con el manejador y con los distintos errores que pueden saltar.

Similar es la situación con las excepciones, en las cuales se ha partido de las facilitadas en clase:

```
1 package com.bs24.demo.common.exception;  
2  
3 public class ResourceAlreadyExistsException extends RuntimeException {  
4     private static final String DESCRIPTION = "Resource already exists";  
5  
6     public ResourceAlreadyExistsException(String message) {  
7         super(DESCRIPTION + ". " + message);  
8     }  
9 }
```

Figura 24. Ejemplo de excepción utilizada.

Finalmente en este paquete común tenemos el de *locale*, el cual se mantiene tal y como lo realizó César. Será un paquete que en este momento no se emplee ya que solamente se usa el idioma inglés, aunque sí será de utilidad en el futuro cuando se amplíen los idiomas empleados en la aplicación.

Paquete controller

El siguiente paquete es el de *controller* o presentación, el cual se encarga de transformar los datos a como los queremos mostrar o, en este caso, enviar como JSON al front.

Dentro de este paquete tenemos el *PaginatedResponse* de cara a paginar los resultados, relevante cuando el número de elementos crezca conforme los ejercicios sean mayores o, por ejemplo, se lleven meses o años de entrenamientos registrados:

BeSport24Training

```
1  package com.bs24.demo.controller.common;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Data;
5
6  import java.util.List;
7
8  @Data
9 @AllArgsConstructor
10 public class PaginatedResponse<T> {
11
12     private List<T> data;
13     private long total;
14     private int currentPage;
15     private int pageSize;
16     private String next;
17     private String previous;
18
19     public PaginatedResponse(List<T> data, long total, int currentPage, int pageSize, String baseUrl) {
20         this.data = data;
21         this.total = total;
22         this.currentPage = currentPage;
23         this.pageSize = pageSize;
24         this.next = createNextLink(baseUrl);
25         this.previous = createPreviousLink(baseUrl);
26     }
27
28     private String createNextLink(String baseUrl) {
29         if(currentPage * pageSize < total) {
30             return baseUrl + "?page=" + (currentPage + 1) + "&size=" + pageSize;
31         }
32         return null;
33     }
34
35     private String createPreviousLink(String baseUrl) {
36         if(currentPage > 1) {
37             return baseUrl + "?page=" + (currentPage - 1) + "&size=" + pageSize;
38         }
39         return null;
40     }
41 }
```

Figura 25. PaginatedResponse.

Tras ello, tenemos el paquete *user* con los webmodelos y los distintos controladores.

En los WebModel tenemos el cómo vamos a transformar los datos del modelo de dominio a los que queremos enviar en formato JSON al front. Por ello, se han realizado diferentes modelos según sea un listado o un detalle, contando con su correspondiente mapeador en cada caso.

A modo de ejemplo, se muestra el de las sesiones:

En primer lugar, tenemos el modelo de la lista de sesiones, en la cual solamente queremos enviar el identificador y el nombre. En este caso, el identificador es

BeSport24Training

importante ya que facilita en gran medida la navegación en SwiftUI mediante el uso de los *NavigationStack* que veremos más adelante.

```
1 package com.bs24.demo.controller.user.webmodel;
2
3 public record SessionCollection(
4     int sessionId,
5     String name
6 ) {
7 }
```

Figura 26. Ejemplo de webmodel de listado.

Y su mapeador:

```
1 package com.bs24.demo.controller.user.webmodel;
2
3 import com.bs24.demo.domain.model.Session;
4 import org.mapstruct.Mapper;
5
6 @Mapper(uses = {UserDetailMapper.class})
7 public interface SessionCollectionMapper {
8
9     SessionCollectionMapper INSTANCE = org.mapstruct.factory.Mappers.getMapper(SessionCollectionMapper.class);
10
11    SessionCollection toSessionCollection(Session session);
12
13    Session toSession(SessionCollection sessionCollection);
14
15 }
```

Figura 27. Ejemplo de mapeador del listado del webmodel.

En cuanto al modelo de los detalles, muestra todo aquello que queremos que el usuario pueda ver cuando accede a una sesión y también qué usuario es para poder enviar aquellos que le pertenecen:

```
1 package com.bs24.demo.controller.user.webmodel;
2
3 public record SessionDetail(
4     int sessionId,
5     String name,
6     String description,
7     String dateCreation,
8     UserDetail user
9 ) {
10 }
```

Figura 28. WebModel del detalle de la sesión.

Y su correspondiente mapeador:

```
1 package com.bs24.demo.controller.user.webmodel;
2
3 import com.bs24.demo.domain.model.Session;
4 import org.mapstruct.Mapper;
5
6 @Mapper(uses = {UserDetailMapper.class})
7 public interface SessionDetailMapper {
8
9     SessionDetailMapper INSTANCE = org.mapstruct.factory.Mappers.getMapper(SessionDetailMapper.class);
10
11    SessionDetail toSessionDetail(Session session);
12
13    Session toSession(SessionDetail sessionDetail);
14
15 }
```

Figura 29. Mapeador del detalle del WebModel.

El resto de los mapeadores seguirían una estructura similar, por lo que se pueden revisar si así se desea en mi repositorio de GitHub.

A continuación, tenemos ya los propios controladores.

A modo de ejemplo, adjunto el controlador de los ejercicios, el cual hace uso del CRUD completo para mostrar el listado de ejercicios, los detalles del ejercicio, la adición de un nuevo ejercicio, la modificación de uno existente o el borrado de un ejercicio.

En general, todos los controladores hacen uso de estos métodos básicos; aunque el controlador de los usuarios no posee el del listado de usuarios, ya que el front necesita los detalles un usuario concreto y no hace uso de ningún listado, por lo que no es necesario elaborar este método; sí que posee el resto de métodos básicos.

BeSport24Training

```
20  public class ExerciseController {
21
22      public static final String URL = "/api/exercises";
23      private final String defaultPageSize = PropertiesConfig.getSetting("app.pageSize.default");
24
25      private final ExerciseGetAllUseCase exercise GetAllUseCase;
26      private final ExerciseFindByIdUseCase exerciseFindByIdUseCase;
27      private final ExerciseInsertUseCase exerciseInsertUseCase;
28      private final ExerciseUpdateUseCase exerciseUpdateUseCase;
29      private final ExerciseDeleteUseCase exerciseDeleteUseCase;
30
31      @GetMapping
32      public ResponseEntity<PaginatedResponse<ExerciseCollection>> getAll(
33          @RequestParam(defaultValue = "1") int page,
34          @RequestParam(required = false) Integer size) {
35
36          int pageSize = (size != null) ? size : Integer.parseInt(defaultPageSize);
37          String baseUrl = PropertiesConfig.getSetting("app.base.url") + URL;
38          ListWithCount<Exercise> exerciseListWithCount = exercise GetAllUseCase.execute(page - 1, pageSize);
39          PaginatedResponse<ExerciseCollection> response = new PaginatedResponse<>(
40              exerciseListWithCount
41                  .getList()
42                  .stream()
43                  .map(ExerciseCollectionMapper.INSTANCE::toExerciseCollection)
44                  .toList(),
45                  exerciseListWithCount.getCount(), page, pageSize, baseUrl);
46          return new ResponseEntity<>(response, HttpStatus.OK);
47      }
48
49      @GetMapping("/{id}")
50      public ResponseEntity<ExerciseDetail> findById(@PathVariable int id) {
51          ExerciseDetail exerciseDetail = ExerciseDetailMapper.INSTANCE.toExerciseDetail(exerciseFindByIdUseCase.execute(id));
52          return new ResponseEntity<>(exerciseDetail, HttpStatus.OK);
53      }
54
55      @PostMapping
56      public ResponseEntity<Void> insert(@RequestBody Exercise exercise) {
57          exerciseInsertUseCase.execute(exercise);
58          return new ResponseEntity<>(HttpStatus.CREATED);
59      }
60
61      @PutMapping("/{id}")
62      public ResponseEntity<Void> update(@PathVariable int id, @RequestBody Exercise exercise) {
63          exercise.setExerciseId(id);
64          exerciseUpdateUseCase.execute(exercise);
65          return new ResponseEntity<>(HttpStatus.OK);
66      }
67
68      @DeleteMapping("/{id}")
69      public ResponseEntity<Void> delete(@PathVariable int id) {
70          exerciseDeleteUseCase.execute(id);
71          return new ResponseEntity<>(HttpStatus.NO_CONTENT);
72      }
73  }
```

Figura 30. Controlador de los ejercicios.

Como aspecto a destacar dentro de lo que son los controladores, tenemos el método del controlador de Sesiones-Ejercicios, que es el que muestra los ejercicios de las sesiones. Por ello, es necesario un método que muestre el listado de ejercicios en función del identificador de la sesión:

```
59     @GetMapping("/session/{sessionId}")
60     public ResponseEntity<PaginatedResponse<SessionExercisesDetail>> getAllBySessionId(
61         @PathVariable int sessionId,
62         @RequestParam(defaultValue = "1") int page,
63         @RequestParam(required = false) Integer size) {
64
65     int pageSize = (size != null) ? size : Integer.parseInt(defaultPageSize);
66     String baseUrl = PropertiesConfig.getSetting("app.base.url") + URL;
67     ListWithCount<SessionExercises> sessionExercisesListWithCount = sessionExercisesFindBySessionId.execute(sessionId, page - 1, pageSize);
68     PaginatedResponse<SessionExercisesDetail> response = new PaginatedResponse<>(
69         sessionExercisesListWithCount
70             .getList()
71             .stream()
72             .map(SessionExercisesDetailMapper.INSTANCE::toSessionExercisesDetail)
73             .toList(),
74         sessionExercisesListWithCount.getCount(), page, pageSize, baseUrl);
75     return new ResponseEntity<>(response, HttpStatus.OK);
76 }
```

Figura 31. Método del controlador de Sesiones-Ejercicios para sacar el listado de ejercicios según el identificador de la sesión.

Con ello, podemos pasar al siguiente paquete, el del dominio.

Paquete domain

En este paquete encontramos el corazón de la aplicación, el núcleo. Fruto de ello, la única “injerencia” externa que se presenta es la de Lombok, el cual ya es, según el profesor César Guijarro, casi un estándar. Asimismo, si hubiera algún cambio, sería fácilmente remediable ya que tan solo se ha empleado para el constructor vacío, el constructor con todos los argumentos, los getters y los setters.

Otro aspecto importante es que se ha extraído la presencia de Spring con las anotaciones personalizadas que se han visto en el paquete común (Figura 22).

Hablando ya del contenido de este paquete, engloba los modelos, los servicios, los casos de uso y las interfaces de los repositorios (claves para la inversión de control).

Respecto a los modelos, estos son los siguientes:

BeSport24Training

```
1 package com.bs24.demo.domain.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class Exercise {
11
12     private int exerciseId;
13     private String name;
14     private String description;
15     private String imageURL;
16     private String videoURL;
17     private Boolean isGlobal;
18     private ExerciseType exerciseType;
19     private ExerciseSubtype exerciseSubtype;
20 }
```

Figura 32. Modelo de Ejercicio.

```
1 package com.bs24.demo.domain.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class ExerciseSubtype {
11
12     private int exerciseTypeId;
13     private String name;
14 }
```

Figura 33. Modelo del Subtipo de Ejercicio.

BeSport24Training

```
1 package com.bs24.demo.domain.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class ExerciseType {
11
12     private int exerciseTypeId;
13     private String name;
14 }
```

Figura 34. Modelo del Tipo de Ejercicio.

```
1 package com.bs24.demo.domain.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 import java.util.List;
8
9 @Data
10 @AllArgsConstructor
11 @NoArgsConstructor
12 public class ListWithCount<T> {
13
14     private List<T> list;
15     private long count;
16 }
```

Figura 35. Modelo del listado para los métodos getAll.

```
1 package com.bs24.demo.domain.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class Session {
11
12     private int sessionId;
13     private String name;
14     private String description;
15     private String dateCreation;
16     private User user;
17 }
```

Figura 36. Modelo de la Sesión.

```
1 package com.bs24.demo.domain.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 import java.util.List;
8
9 @Data
10 @AllArgsConstructor
11 @NoArgsConstructor
12 public class SessionExercises {
13
14     private int sessionExercisesId;
15     private int numSets;
16     private int numReps;
17     private int restSeconds;
18     private Session session;
19     private Exercise exercise;
20 }
```

Figura 37. Modelo del Sesión-Ejercicio.

BeSport24Training

```
1 package com.bs24.demo.domain.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 import java.util.List;
8
9 @Data
10 @AllArgsConstructor
11 @NoArgsConstructor
12 public class Training {
13
14     private int trainingId;
15     private String name;
16     private int trainingDurationSeconds;
17     private String dateCreation;
18     private User user;
19     private Session session;
20 }
```

Figura 38. Modelo del Entrenamiento.

```
1 package com.bs24.demo.domain.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6
7 @Data
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class User {
11
12     private int userId;
13     private String name;
14     private String email;
15     private String password;
16     private String dateRegister;
17 }
```

Figura 39. Modelo del Usuario.

En cuanto a los servicios, cada modelo de negocio cuenta con su respectivo servicio. Estos servicios están formados por una interfaz y por su correspondiente implementación, siguiendo las guías de código limpio de libros como Clean Code de Robert C. Martin, al igual que su nomenclatura, la cual concuerda con la del profesor

BeSport24Training

César Guijarro. A continuación, se muestra un ejemplo de interfaz de servicio con su correspondiente implementación:

```
1 package com.bs24.demo.domain.service;
2
3 import com.bs24.demo.domain.model.Exercise;
4 import com.bs24.demo.domain.model.ListWithCount;
5
6 import java.util.Optional;
7
8 ▼ public interface ExerciseService {
9     ListWithCount<Exercise> getAll(int page, int pageSize);
10
11     Optional<Exercise> findById(int exerciseId);
12
13     void save(Exercise exercise);
14
15     void delete(int id);
16 }
```

Figura 40. Interfaz del servicio de ejercicios.

```
1 package com.bs24.demo.domain.service.impl;
2
3 import com.bs24.demo.common.annotation.DomainService;
4 import com.bs24.demo.domain.model.Exercise;
5 import com.bs24.demo.domain.model.ListWithCount;
6 import com.bs24.demo.domain.repository.ExerciseRepository;
7 import com.bs24.demo.domain.service.ExerciseService;
8 import lombok.RequiredArgsConstructor;
9
10 import java.util.Optional;
11
12 @DomainService
13 @RequiredArgsConstructor
14 ▼ public class ExerciseServiceImpl implements ExerciseService {
15
16     private final ExerciseRepository exerciseRepository;
17
18     @Override
19     public ListWithCount<Exercise> getAll(int page, int pageSize) {
20         return exerciseRepository.getAll(page, pageSize);
21     }
22
23     @Override
24     public Optional<Exercise> findById(int exerciseId) {
25         return exerciseRepository.findById(exerciseId);
26     }
27
28     @Override
29     public void save(Exercise exercise) {
30         exerciseRepository.save(exercise);
31     }
32
33     @Override
34     public void delete(int id) {
35         exerciseRepository.delete(id);
36     }
37 }
```

Figura 41. Implementación del servicio de ejercicios.

Estos servicios son empleados por los casos de uso de cara a obtener los resultados deseados de los diferentes métodos (GET, POST, PUT o DELETE). A continuación, se muestra el caso de uso de insertar un ejercicio, el cual acaba llamando al servicio save.

```
1 package com.bs24.demo.domain.usecase.exercise;
2
3 import com.bs24.demo.controller.user.webmodel.ExerciseDetail;
4 import com.bs24.demo.domain.model.Exercise;
5
6 public interface ExerciseInsertUseCase {
7     void execute(Exercise exercise);
8 }
```

Figura 42. Interfaz del caso de uso de insertar un ejercicio.

```
1 package com.bs24.demo.domain.usecase.exercise.impl;
2
3 import com.bs24.demo.common.annotation.DomainTransactional;
4 import com.bs24.demo.common.annotation.DomainUseCase;
5 import com.bs24.demo.common.exception.ResourceAlreadyExistsException;
6 import com.bs24.demo.controller.user.webmodel.ExerciseDetail;
7 import com.bs24.demo.domain.model.Exercise;
8 import com.bs24.demo.domain.service.ExerciseService;
9 import com.bs24.demo.domain.service.ExerciseSubtypeService;
10 import com.bs24.demo.domain.service.ExerciseTypeService;
11 import com.bs24.demo.domain.usecase.exercise.ExerciseInsertUseCase;
12 import lombok.RequiredArgsConstructor;
13
14 @DomainUseCase
15 @DomainTransactional
16 @RequiredArgsConstructor
17 public class ExerciseInsertUseCaseImpl implements ExerciseInsertUseCase {
18
19     private final ExerciseService exerciseService;
20     private final ExerciseTypeService exerciseTypeService;
21     private final ExerciseSubtypeService exerciseSubtypeService;
22
23     @Override
24     public void execute(Exercise exercise) {
25         if (exerciseService.findById(exercise.getExerciseId()).isPresent()) {
26             throw new ResourceAlreadyExistsException("Exercise with ID " + exercise.getExerciseId() + " already exists");
27         }
28
29         exercise.setExerciseType(
30             exerciseTypeService.findById(exercise.getExerciseType().getExerciseTypeId())
31                 .orElseThrow(() -> new ResourceAlreadyExistsException("Exercise type not found")));
32     };
33
34     exercise.setExerciseSubtype(
35         exerciseSubtypeService.findById(exercise.getExerciseSubtype().getExerciseTypeId())
36             .orElseThrow(() -> new ResourceAlreadyExistsException("Exercise type not found")));
37     };
38
39     exerciseService.save(exercise);
40 }
41 }
```

Figura 43. Implementación del caso de uso de insertar un ejercicio.

Con ello, logramos modularizar en la mayor medida posible la aplicación para que sea más fácilmente mantenible y legible. Asimismo, en el caso de uso podemos hacer las distintas comprobaciones, por lo que, si se añaden o modifican nuevas comprobaciones, será más sencillo ir al propio caso de uso que yendo a los distintos servicios al ser clases más escuetas.

Siguiendo con el orden que seguiría el proceso desde que llega la petición desde el *frontend*, tendríamos que el controlador llama al caso de uso, que llama al servicio. Estos servicios llamarían a los repositorios. En este caso, no encontramos al mismo nivel la interfaz y la implementación, sino que la interfaz se encuentra dentro del dominio y la implementación en el repositorio. Esto se realiza de este modo para lograr la inversión de control y, de este modo, que el servicio no dependa del repositorio, sino que sea el núcleo central de la aplicación.

Así pues, siguiendo con el ejemplo del apartado de los ejercicios de la aplicación, la interfaz del repositorio de ejercicios sería la siguiente:

```
1 package com.bs24.demo.domain.repository;
2
3 import com.bs24.demo.domain.model.Exercise;
4 import com.bs24.demo.domain.model.ListWithCount;
5
6 import java.util.Optional;
7
8 public interface ExerciseRepository {
9     ListWithCount<Exercise> getAll(int page, int pageSize);
10
11     Optional<Exercise> findById(int exerciseId);
12
13     void save(Exercise exercise);
14
15     void delete(int id);
16 }
```

Figura 44. Interfaz del repositorio de ejercicios.

Paquete **persistence**

La interfaz, consecuentemente, se encuentra ya en el paquete de persistencia, conjuntamente a los DAOs y al repositorio de JPA que se encarga de hacer la llamada final que, en este caso, es a la base de datos. Por el camino, encontramos las propias entidades de JPA y los mapeadores entre estas entidades y las de nuestro modelo de

dominio. Así pues, nuestra primera parada en este paquete es la implementación del repositorio, la cual es la siguiente:

```
1 package com.bs24.demo.persistence.repository.imp;
2
3 import com.bs24.demo.domain.Exercise;
4 import com.bs24.demo.domain.model.ListWithCount;
5 import com.bs24.demo.domain.repository.ExerciseRepository;
6 import com.bs24.demo.persistence.dao.db.ExerciseDaoDb;
7 import lombok.RequiredArgsConstructor;
8 import org.springframework.stereotype.Repository;
9
10 import java.util.Optional;
11
12 @Repository
13 @RequiredArgsConstructor
14 public class ExerciseRepositoryImpl implements ExerciseRepository {
15
16     private final ExerciseDaoDb exerciseDaoDb;
17
18     @Override
19     public ListWithCount<Exercise> getAll(int page, int pageSize) {
20         return exerciseDaoDb.getAll(page, pageSize);
21     }
22
23     @Override
24     public Optional<Exercise> findById(int exerciseId) {
25         return exerciseDaoDb.findById(exerciseId);
26     }
27
28     @Override
29     public void save(Exercise exercise) {
30         exerciseDaoDb.save(exercise);
31     }
32
33     @Override
34     public void delete(int id) {
35         exerciseDaoDb.delete(id);
36     }
}
```

Figura 45. Implementación del repositorio de ejercicios.

Estas implementaciones, tal y como podemos observar, llaman a la interfaz de los DAO, en este caso, evidentemente, a la del DAO de ejercicios. De nuevo, como se llama a la interfaz, tendremos la interfaz que es empleada por su implementación:

```
1 package com.bs24.demo.persistence.dao.db;
2
3 import com.bs24.demo.domain.model.Exercise;
4
5 public interface ExerciseDaoDb extends GenericDaoDb<Exercise> {
6 }
```

Figura 46. Interfaz del DAO de ejercicios.

BeSport24Training

```
1 package com.bs24.demo.persistence.dao.db.jpa;
2
3 import com.bs24.demo.domain.Exercise;
4 import com.bs24.demo.domain.ListWithCount;
5 import com.bs24.demo.persistence.dao.db.ExerciseDaoDb;
6 import com.bs24.demo.persistence.dao.db.jpa.entity.ExerciseJPA;
7 import com.bs24.demo.persistence.dao.db.jpa.mapper.ExerciseJPAMapper;
8 import com.bs24.demo.persistence.dao.db.jpa.repository.ExerciseJPARespository;
9 import lombok.RequiredArgsConstructor;
10 import org.springframework.data.domain.Page;
11 import org.springframework.data.domain.PageRequest;
12 import org.springframework.data.domain.Pageable;
13 import org.springframework.stereotype.Repository;
14
15 import java.util.List;
16 import java.util.Optional;
17
18 @Repository
19 @RequiredArgsConstructor
20 public class ExerciseDaoDbImpl implements ExerciseDaoDb {
21
22     private final ExerciseJPARespository exerciseJPARespository;
23
24     @Override
25     public List<Exercise> getAll() {
26         return List.of();
27     }
28
29     @Override
30     public ListWithCount<Exercise> getAll(int page, int size) {
31         Pageable pageable = PageRequest.of(page, size);
32         Page<ExerciseJPA> exerciseJPAPage = exerciseJPARespository.findAll(pageable);
33         return new ListWithCount<>(
34             exerciseJPAPage.stream()
35                 .map(ExerciseJPAMapper.INSTANCE::toExercise)
36                 .toList(),
37             exerciseJPAPage.getTotalElements()
38         );
39     }
40
41     @Override
42     public Optional<Exercise> findById(long id) {
43         return exerciseJPARespository.findById(id)
44             .map(ExerciseJPAMapper.INSTANCE::toExercise);
45     }
46
47     @Override
48     public long insert(Exercise exercise) {
49         return 0;
50     }
51
52     @Override
53     public void update(Exercise exercise) {
54     }
55
56     @Override
57     public void delete(long id) {
58         exerciseJPARespository.deleteById(id);
59     }
60
61     @Override
62     public long count() {
63         return 0;
64     }
65
66     @Override
67     public Exercise save(Exercise exercise) {
68         ExerciseJPA exerciseJPA = ExerciseJPAMapper.INSTANCE.toExerciseJPA(exercise);
69         return ExerciseJPAMapper.INSTANCE.toExercise(exerciseJPARespository.save(exerciseJPA));
70     }
71 }
72 }
```

Figura 47. Implementación del DAO de ejercicios.

Puede llamarnos la atención que la interfaz esté vacía y la implementación contenga numerosos métodos. Esto se debe a que la interfaz hereda de una interfaz genérica

que no amplía ningún método, por lo que se heredan todos estos métodos que se deben implementar en la Figura 47:

```
1 package com.bs24.demo.persistence.dao.db;
2
3 import com.bs24.demo.domain.model.ListWithCount;
4
5 import java.util.List;
6 import java.util.Optional;
7
8 public interface GenericDaoDb<T> {
9
10     List<T> getAll();
11     ListWithCount<T> getAll(int page, int size);
12     Optional<T> findById(long id);
13     long insert(T t);
14     void update(T t);
15     void delete(long id);
16     long count();
17     T save(T t);
18
19 }
```

Figura 48. Interfaz genérica del DAO.

Así pues, si analizamos detenidamente la Figura 47 y la Figura 48, encontraremos que se implementan todos los métodos decretados por este “contrato” que se establece al implementar la interfaz vacía que hereda los métodos de la interfaz genérica.

Echando de nuevo un vistazo a la Figura 47, podemos constatar que se llama a la interfaz JPA, pero que por medio se realizan diversos pasos: el uso de mapeadores y de entidades JPA.

La entidad de JPA de los ejercicios es la siguiente:

BeSport24Training

```
1 package com.bs24.demo.persistence.dao.db.jpa.entity;
2
3 import jakarta.persistence.*;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7
8 @Entity
9 @Table(name = "exercises")
10 @Data
11@AllArgsConstructor
12@NoArgsConstructor
13 public class ExerciseJPA {
14
15     @Id
16     @GeneratedValue(strategy = GenerationType.IDENTITY)
17     @Column(name = "exercise_id")
18     private int exerciseId;
19     private String name;
20     private String description;
21     @Column(name = "image_url")
22     private String imageURL;
23     @Column(name = "video_url")
24     private String videoURL;
25     @Column(name = "is_global")
26     private Boolean isGlobal;
27     @ManyToOne(fetch = FetchType.LAZY)
28     @JoinColumn(name = "type_id")
29     private ExerciseTypeJPA exerciseType;
30     @ManyToOne(fetch = FetchType.LAZY)
31     @JoinColumn(name = "subtype_id")
32     private ExerciseSubtypeJPA exerciseSubtype;
33 }
```

Figura 49. Entidad JPA de ejercicios.

Por tanto, necesitará un mapeador, el cual se automatiza gracias a MapStruct de nuevo, al igual que sucedió en los controladores:

```
1 package com.bs24.demo.persistence.dao.db.jpa.mapper;
2
3 import com.bs24.demo.domain.model.Exercise;
4 import com.bs24.demo.persistence.dao.db.jpa.entity.ExerciseJPA;
5 import org.mapstruct.Mapper;
6
7 @Mapper
8 public interface ExerciseJPAMapper {
9
10     ExerciseJPAMapper INSTANCE = org.mapstruct.factory.Mappers.getMapper(ExerciseJPAMapper.class);
11
12     Exercise toExercise(ExerciseJPA exerciseJPA);
13     ExerciseJPA toExerciseJPA(Exercise exercise);
14 }
```

Figura 50. Mapeador de la entidad JPA a la entidad de dominio y viceversa.

Todo ello nos permite, con poco código, aislar nuestro modelo de negocio de la fuente de datos y que, con MapStruct, sea rápidamente mapeada para poder convertir los datos tanto para recibirlos de la fuente de datos como para insertarlos o modificarlos.

Así pues, la implementación del DAO, tras hacer los mapeos pertinentes, puede efectuar el método de insertar el ejercicio, en el cual llama al repositorio de JPA de los ejercicios, el cual es el siguiente:

```
1 package com.bs24.demo.persistence.dao.db.jpa.repository;
2
3 import com.bs24.demo.persistence.dao.db.jpa.entity.ExerciseJPA;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface ExerciseJPRepository extends JpaRepository<ExerciseJPA, Long> {
7 }
```

Figura 51. Interfaz JPA de ejercicios.

Esta interfaz, tal y como podemos ver, está vacía ya que lleva implícitos los métodos básicos del CRUD, por lo que no es necesario escribir el código, pero sí que hará la inserción del ejercicio en la base de datos.

Por tanto, acabamos de ver el proceso que vendría a hacerse de manera detallada. No obstante, al recorrer y explicar cada paquete es posible que perdamos la noción del proceso completo, por lo que, mediante otro ejemplo, se detallará de nuevo el proceso, en este caso empezando en otro controlador y siguiendo un proceso más complejo (haremos el ejemplo sin entrar en los diferentes modelos ni mapeadores para ir más al grano):

En primer lugar, tenemos el controlador, en el cual se realizan diferentes procesos. En nuestro caso, nos interesa el de encontrar los ejercicios según la sesión de entrenamiento (método de la línea 60, Figura 53). Ello es necesario para, en el front, mostrar los detalles de la sesión y, a su vez, el compendio de todos los ejercicios que la muestran.

BeSport24Training

```

1 package com.bs24.demo.controller.user;
2
3 import com.bs24.demo.common.config.PropertiesConfig;
4 import com.bs24.demo.controller.common.PaginatedResponse;
5 import com.bs24.demo.controller.user.webmodel.*;
6 import com.bs24.demo.domain.model.Exercise;
7 import com.bs24.demo.domain.model.ListWithCount;
8 import com.bs24.demo.domain.model.Session;
9 import com.bs24.demo.domain.model.SessionExercises;
10 import com.bs24.demo.domain.usecase.session.SessionFindByIdUseCase;
11 import com.bs24.demo.domain.usecase.session.SessionGetAllUseCase;
12 import com.bs24.demo.domain.usecase.session.SessionInsertUseCase;
13 import com.bs24.demo.domain.usecase.sessionexercises.session.*;
14 import com.bs24.demo.domain.usecase.sessionexercises.session.impl.SessionExercisesGetAllUseCaseImpl;
15 import lombok.RequiredArgsConstructor;
16 import org.springframework.http.HttpStatus;
17 import org.springframework.http.ResponseEntity;
18 import org.springframework.web.bind.annotation.*;
19
20 @RestController
21 @RequiredArgsConstructor
22 @RequestMapping("/{app.api.path}/sessionsexercises")
23 public class SessionExercisesController {
24
25     public static final String URL = "/api/sessionsexercises";
26     private final String defaultPageSize = PropertiesConfig.getSetting("app.pageSize.default");
27
28     private final SessionExercisesGetAllUseCase sessionExercisesGetAllUseCase;
29     private final SessionExercisesFindByIdUseCase sessionExercisesFindByIdUseCase;
30     private final SessionExercisesFindBySessionId sessionExercisesFindBySessionId;
31     private final SessionExercisesInsertUseCase sessionExercisesInsertUseCase;
32     private final SessionExercisesUpdateUseCase sessionExercisesUpdateUseCase;
33     private final SessionExercisesDeleteUseCase sessionExercisesDeleteUseCase;
34
35     @GetMapping
36     public ResponseEntity<PaginatedResponse<SessionExercisesDetail>> getAll(
37         @RequestParam(defaultValue = "1") int page,
38         @RequestParam(required = false) Integer size) {
39
40         int pageSize = (size != null) ? size : Integer.parseInt(defaultPageSize);
41         String baseUrl = PropertiesConfig.getSetting("app.base.url") + URL;
42         ListWithCount<SessionExercises> sessionExercisesListWithCount = sessionExercisesGetAllUseCase.execute(page - 1, pageSize);
43         PaginatedResponse<SessionExercisesDetail> response = new PaginatedResponse<>(
44             sessionExercisesListWithCount
45                 .getList()
46                 .stream()
47                 .map(SessionExercisesDetailMapper.INSTANCE::toSessionExercisesDetail)
48                 .toList(),
49             sessionExercisesListWithCount.getCount(), page, pageSize, baseUrl);
50
51         return new ResponseEntity<>(response, HttpStatus.OK);
52     }
53
54     @GetMapping("/{id}")
55     public ResponseEntity<SessionExercisesDetail> findById(@PathVariable int id) {
56         SessionExercisesDetail sessionExercisesDetail = SessionExercisesDetailMapper.INSTANCE.toSessionExercisesDetail(sessionExercisesFindByIdUseCase.execute(id));
57         return new ResponseEntity<>(sessionExercisesDetail, HttpStatus.OK);
58     }
59
60     @GetMapping("/session/{sessionId}")
61     public ResponseEntity<PaginatedResponse<SessionExercisesDetail>> getAllBySessionId(
62         @PathVariable int sessionId,
63         @RequestParam(defaultValue = "1") int page,
64         @RequestParam(required = false) Integer size) {
65
66         int pageSize = (size != null) ? size : Integer.parseInt(defaultPageSize);
67         String baseUrl = PropertiesConfig.getSetting("app.base.url") + URL;
68         ListWithCount<SessionExercises> sessionExercisesListWithCount = sessionExercisesFindBySessionId.execute(sessionId, page - 1, pageSize);
69         PaginatedResponse<SessionExercisesDetail> response = new PaginatedResponse<>(
70             sessionExercisesListWithCount
71                 .getList()
72                 .stream()
73                 .map(SessionExercisesDetailMapper.INSTANCE::toSessionExercisesDetail)
74                 .toList(),
75             sessionExercisesListWithCount.getCount(), page, pageSize, baseUrl);
76
77         return new ResponseEntity<>(response, HttpStatus.OK);
78     }
79
80     @PostMapping
81     public ResponseEntity<Void> insert(@RequestBody SessionExercises sessionExercises) {
82         sessionExercisesInsertUseCase.execute(sessionExercises);
83         return new ResponseEntity<>(HttpStatus.CREATED);
84     }
85
86     @PutMapping("/{id}")
87     public ResponseEntity<Void> update(@PathVariable int id, @RequestBody SessionExercises sessionExercises) {
88         sessionExercises.setSessionExercisesId(id);
89         sessionExercisesUpdateUseCase.execute(sessionExercises);
90         return new ResponseEntity<>(HttpStatus.OK);
91     }
92
93     @DeleteMapping("/{id}")
94     public ResponseEntity<Void> delete(@PathVariable int id) {
95         sessionExercisesDeleteUseCase.execute(id);
96         return new ResponseEntity<>(HttpStatus.NO_CONTENT);
97     }

```

Figura 52. Controlador de Sesiones-Ejercicios.

```

59     @GetMapping("/session/{sessionId}")
60     public ResponseEntity<PaginatedResponse<SessionExercisesDetail>> getAllBySessionId(
61         @PathVariable int sessionId,
62         @RequestParam(defaultValue = "1") int page,
63         @RequestParam(required = false) Integer size) {
64
65         int pageSize = (size != null) ? size : Integer.parseInt(defaultPageSize);
66         String baseUrl = PropertiesConfig.getSetting("app.base.url") + URL;
67         ListWithCount<SessionExercises> sessionExercisesListWithCount = sessionExercisesFindByIdUseCase.execute(sessionId, page - 1, pageSize);
68         PaginatedResponse<SessionExercisesDetail> response = new PaginatedResponse<>(
69             sessionExercisesListWithCount
70                 .getList()
71                 .stream()
72                 .map(SessionExercisesDetailMapper.INSTANCE::toSessionExercisesDetail)
73                 .toList(),
74             sessionExercisesListWithCount.getCount(), page, pageSize, baseUrl);
75
76         return new ResponseEntity<>(response, HttpStatus.OK);
77     }

```

Figura 53. Método concreto del controlador que empleamos.

Llama a la interfaz del caso de uso:

```
1 package com.bs24.demo.domain.usecase.sessionexercises.session;
2
3 import com.bs24.demo.domain.model.ListWithCount;
4 import com.bs24.demo.domain.model.SessionExercises;
5
6 public interface SessionExercisesFindBySessionId {
7     ListWithCount<SessionExercises> execute(int sessionId, int page, int pageSize);
8 }
```

Figura 54. Interfaz del caso de uso.

Que llama, a su vez, a la implementación:

```
1 package com.bs24.demo.domain.usecase.sessionexercises.session.impl;
2
3 import com.bs24.demo.common.annotation.DomainTransactional;
4 import com.bs24.demo.common.annotation.DomainUseCase;
5 import com.bs24.demo.domain.model.ListWithCount;
6 import com.bs24.demo.domain.model.SessionExercises;
7 import com.bs24.demo.domain.service.SessionExercisesService;
8 import com.bs24.demo.domain.usecase.sessionexercises.session.SessionExercisesFindBySessionId;
9 import lombok.RequiredArgsConstructor;
10
11 @DomainUseCase
12 @DomainTransactional
13 @RequiredArgsConstructor
14 public class SessionExercisesFindBySessionIdImpl implements SessionExercisesFindBySessionId {
15
16     private final SessionExercisesService sessionExercisesService;
17
18     @Override
19     public ListWithCount<SessionExercises> execute(int sessionId, int page, int pageSize) {
20         return sessionExercisesService.getAllBySessionId(sessionId, page, pageSize);
21     }
22 }
```

Figura 55. Implementación del caso de uso.

Esta implementación llama a la interfaz del servicio propio (Figura 56), que es implementada (Figura 57):

```
1 package com.bs24.demo.domain.service;
2
3 import com.bs24.demo.domain.model.ListWithCount;
4 import com.bs24.demo.domain.model.SessionExercises;
5
6 import java.util.Optional;
7
8 ▼ public interface SessionExercisesService {
9     ListWithCount<SessionExercises> getAll(int page, int pageSize);
10
11     Optional<SessionExercises> findById(int sessionId);
12
13     void save(SessionExercises sessionExercises);
14
15     ListWithCount<SessionExercises> getAllBySessionId(int sessionId, int page, int pageSize);
16
17     void delete(int id);
18 }
```

Figura 56. Interfaz del servicio de Sesiones-Ejercicios.

BeSport24Training

```
1  package com.bs24.demo.domain.service.impl;
2
3  import com.bs24.demo.common.annotation.DomainService;
4  import com.bs24.demo.domain.model.ListWithCount;
5  import com.bs24.demo.domain.model.SessionExercises;
6  import com.bs24.demo.domain.repository.SessionExercisesRepository;
7  import com.bs24.demo.domain.service.SessionExercisesService;
8  import lombok.RequiredArgsConstructor;
9
10 import java.util.Optional;
11
12 @DomainService
13 @RequiredArgsConstructor
14 public class SessionExercisesServiceImpl implements SessionExercisesService {
15
16     private final SessionExercisesRepository sessionExercisesRepository;
17
18     @Override
19     public ListWithCount<SessionExercises> getAll(int page, int pageSize) {
20         return sessionExercisesRepository.getAll(page, pageSize);
21     }
22
23     @Override
24     public Optional<SessionExercises> findById(int sessionId) {
25         return sessionExercisesRepository.findById(sessionId);
26     }
27
28     @Override
29     public void save(SessionExercises sessionExercises) {
30         sessionExercisesRepository.save(sessionExercises);
31     }
32
33     @Override
34     public ListWithCount<SessionExercises> getAllBySessionId(int sessionId, int page, int pageSize) {
35         return sessionExercisesRepository.getAllBySessionId(sessionId, page, pageSize);
36     }
37
38     @Override
39     public void delete(int id) {
40         sessionExercisesRepository.delete(id);
41     }
42 }
```

Figura 57. Implementación del servicio de Sesiones-Ejercicios.

Concretamente, es el método de la línea 34. Aquí, siguiendo el proceso habitual, llama a la interfaz del repositorio correspondiente (dentro de la capa de dominio, Figura 58), que es implementado por su repositorio (en la capa de persistencia para realizar la inversión de control, Figura 59):

```
1 package com.bs24.demo.domain.repository;
2
3 import com.bs24.demo.domain.model.ListWithCount;
4 import com.bs24.demo.domain.model.SessionExercises;
5
6 import java.util.Optional;
7
8 public interface SessionExercisesRepository {
9     ListWithCount<SessionExercises> getAll(int page, int pageSize);
10    Optional<SessionExercises> findById(int sessionId);
11    void save(SessionExercises sessionExercises);
12    ListWithCount<SessionExercises> getAllBySessionId(int sessionId, int page, int pageSize);
13    void delete(int id);
14 }
15
16
17
18 }
```

Figura 58. Interfaz del repositorio de Sesiones-Ejercicios.

BeSport24Training

```
1  package com.bs24.demo.persistence.repository.impl;
2
3  import com.bs24.demo.domain.model.ListWithCount;
4  import com.bs24.demo.domain.model.SessionExercises;
5  import com.bs24.demo.domain.repository.SessionExercisesRepository;
6  import com.bs24.demo.persistence.dao.db.SessionExercisesDaoDb;
7  import lombok.RequiredArgsConstructor;
8  import org.springframework.stereotype.Repository;
9
10 import java.util.Optional;
11
12 @Repository
13 @RequiredArgsConstructor
14 public class SessionExercisesRepositoryImpl implements SessionExercisesRepository {
15
16     private final SessionExercisesDaoDb sessionExercisesDaoDb;
17
18     @Override
19     public ListWithCount<SessionExercises> getAll(int page, int pageSize) {
20         return sessionExercisesDaoDb.getAll(page, pageSize);
21     }
22
23     @Override
24     public Optional<SessionExercises> findById(int sessionId) {
25         return sessionExercisesDaoDb.findById(sessionId);
26     }
27
28     @Override
29     public void save(SessionExercises sessionExercises) {
30         sessionExercisesDaoDb.save(sessionExercises);
31     }
32
33     @Override
34     public ListWithCount<SessionExercises> getAllBySessionId(int sessionId, int page, int pageSize) {
35         return sessionExercisesDaoDb.getAllBySessionId(sessionId, page, pageSize);
36     }
37
38     @Override
39     public void delete(int id) {
40         sessionExercisesDaoDb.delete(id);
41     }
}
```

Figura 59. Implementación del repositorio de Sesiones-Ejercicios.

Este método de ejemplo (línea 33), llama a la interfaz de su DAO (Figura 60), la cual es, consecuentemente, implementada (Figura 61):

```
1  package com.bs24.demo.persistence.dao.db;
2
3  import com.bs24.demo.domain.model.ListWithCount;
4  import com.bs24.demo.domain.model.SessionExercises;
5
6  public interface SessionExercisesDaoDb extends GenericDaoDb<SessionExercises> {
7      ListWithCount<SessionExercises> getAllBySessionId(int sessionId, int page, int pageSize);
8  }
```

Figura 60. Interfaz del DAO de Sesiones-Ejercicios.

BeSport24Training

```
1 package com.bs24.demo.persistence.dao.db.jpa;
2
3 import com.bs24.demo.domain.model.ListWithCount;
4 import com.bs24.demo.domain.model.SessionExercises;
5 import com.bs24.demo.persistence.dao.db.SessionExercisesDaoDb;
6 import com.bs24.demo.persistence.dao.db.jpa.entity.SessionExercisesJPA;
7 import com.bs24.demo.persistence.dao.db.jpa.entity.SessionJPA;
8 import com.bs24.demo.persistence.dao.db.jpa.mapper.SessionExercisesJPAMapper;
9 import com.bs24.demo.persistence.dao.db.jpa.mapper.SessionJPMapper;
10 import com.bs24.demo.persistence.dao.db.jpa.repository.SessionExercisesJPARespository;
11 import lombok.RequiredArgsConstructor;
12 import org.springframework.data.domain.Page;
13 import org.springframework.data.domain.PageRequest;
14 import org.springframework.data.domain.Pageable;
15 import org.springframework.stereotype.Repository;
16
17 import java.util.List;
18 import java.util.Optional;
19
20 @Repository
21 @RequiredArgsConstructor
22 public class SessionExercisesDaoDbImpl implements SessionExercisesDaoDb {
23
24     private final SessionExercisesJPARespository sessionExercisesJPARespository;
25
26     @Override
27     public List<SessionExercises> getAll() {
28         return List.of();
29     }
30
31     @Override
32     public ListWithCount<SessionExercises> getAll(int page, int size) {
33         Pageable pageable = PageRequest.of(page, size);
34         Page<SessionExercisesJPA> sessionExercisesJPAPage = sessionExercisesJPARespository.findAll(pageable);
35         return new ListWithCount<>(
36             sessionExercisesJPAPage.stream()
37                 .map(SessionExercisesJPAMapper.INSTANCE::toSessionExercises)
38                 .toList(),
39             sessionExercisesJPAPage.getTotalElements()
40         );
41     }
42
43     @Override
44     public Optional<SessionExercises> findById(long id) {
45         return sessionExercisesJPARespository.findById(id)
46             .map(SessionExercisesJPAMapper.INSTANCE::toSessionExercises);
47     }
48
49     @Override
50     public long insert(SessionExercises sessionExercises) {
51         return 0;
52     }
53
54     @Override
55     public void update(SessionExercises sessionExercises) {
56
57     }
58
59     @Override
60     public void delete(long id) {
61         sessionExercisesJPARespository.deleteById(id);
62     }
63
64     @Override
65     public long count() {
66         return 0;
67     }
68
69     @Override
70     public SessionExercises save(SessionExercises sessionExercises) {
71         SessionExercisesJPA sessionExercisesJPA = SessionExercisesJPAMapper.INSTANCE.toSessionExercisesJPA(sessionExercises);
72         return SessionExercisesJPAMapper.INSTANCE.toSessionExercises(sessionExercisesJPARespository.save(sessionExercisesJPA));
73     }
74
75     @Override
76     public ListWithCount<SessionExercises> getAllBySessionId(int sessionId, int page, int pageSize) {
77         Pageable pageable = PageRequest.of(page, pageSize);
78         Page<SessionExercisesJPA> sessionExercisesJPAPage = sessionExercisesJPARespository.findBySession_SessionId(sessionId, pageable);
79         return new ListWithCount<>(
80             sessionExercisesJPAPage.stream()
81                 .map(SessionExercisesJPAMapper.INSTANCE::toSessionExercises)
82                 .toList(),
83             sessionExercisesJPAPage.getTotalElements()
84         );
85     }
86 }
```

Figura 61. Implementación del DAO de Sesiones-Ejercicios.

```
75     @Override
76     public ListWithCount<SessionExercises> getAllBySessionId(int sessionId, int page, int pageSize) {
77         Pageable pageable = PageRequest.of(page, pageSize);
78         Page<SessionExercisesJPA> sessionExercisesJPAPage = sessionExercisesJPARespository.findBySession_SessionId(sessionId, pageable);
79         return new ListWithCount<>(
80             sessionExercisesJPAPage.stream()
81                 .map(SessionExercisesJPAMapper.INSTANCE::toSessionExercises)
82                 .toList(),
83             sessionExercisesJPAPage.getTotalElements()
84         );
85     }
```

Figura 62. Método concreto del ejemplo en el DAO de Sesiones-Ejercicios.

En este caso concreto, la interfaz (Figura 60) hereda todos los métodos del DAO genérico, pero a su vez contiene un método concreto para este DAO, el cual es el del método que estamos empleando. Por tanto, la implementación (Figura 61, Figura 62) contiene todos los métodos del DAO genérico por la herencia más el método propio del DAO que implementa.

De nuevo, el DAO llamará al repositorio de JPA, que se encargará de la llamada a la fuente de datos:

```
1 package com.bs24.demo.persistence.dao.db.jpa.repository;
2
3 import com.bs24.demo.persistence.dao.db.jpa.entity.SessionExercisesJPA;
4 import org.springframework.data.domain.Page;
5 import org.springframework.data.domain.Pageable;
6 import org.springframework.data.jpa.repository.JpaRepository;
7
8 import java.util.List;
9
10 public interface SessionExercisesJPARespository extends JpaRepository<SessionExercisesJPA, Long> {
11     Page<SessionExercisesJPA> findBySession_SessionId(int sessionId, Pageable pageable);
12 }
```

Figura 63. Interfaz del repositorio JPA de Sesiones-Ejercicios.

Aquí es donde radica una de las ventajas de hacer uso de JPA, ya que nos permite, en una sola línea de código, establecer la consulta a la base de datos que queremos hacer. Evidentemente, debemos realizarlo según las normas que dicta JPA, pero en el momento en el que nos acostumbramos, la realización de las sentencias se realiza rápidamente y con menor margen de error que cuando lo realizábamos mediante JDBC.

Front End

Finalmente, pasamos al *frontend*, aquello con lo cual va a interactuar directamente el usuario final. Esta es la parte más diferenciadora de este trabajo final de grado ya que emplea dos tecnologías muy poco vistas en Desarrollo de Aplicaciones Web y Desarrollo de Aplicaciones Multiplataforma, ya que son tecnologías de Apple: Swift y SwiftUI. Consecuentemente, este *front* va a ser móvil nativo y no web o aplicación web progresiva.

Por ello, el primer paso resultaba crucial: establecer la arquitectura que era más acorde a esta parte del proyecto. De esta manera, fue necesario establecer una fase del proyecto centrada en la investigación de arquitecturas empleadas en Swift y, al emplear el *framework* más reciente, aquellas más adecuadas para SwiftUI.

En Swift, la arquitectura limpia se puede implementar siguiendo los mismos principios generales, adaptados al ecosistema de iOS. Las principales arquitecturas en Swift son las siguientes:

MVC

Siglas de Model-View-Controller.

En SwiftUI no encaja bien porque no hay controladores de vista como en UIKit. Las vistas y la lógica se mezclan mucho, por lo que es difícil de escalar.

El model son los datos que se utilizan para obtener la información que se va a representar en las vistas.

La vista es la representación visual del modelo que queremos mostrar en las pantallas de nuestra aplicación.

Por último, el controlador es el mediador. Conecta la View con el Model y controla qué lógica ejecutar. Es la pieza más importante y la menos reusable.

REDUX

Inspirada en Redux de JavaScript. Propone un estado único global (@*ObservableState*) más acciones y *reducers*.

Ventajas:

- Muy estructurada.
- Ideal para apps grandes o con lógica compleja.
- Testable.
- Predecible.

Desventajas:

- Verbosa.
- Gran curva de aprendizaje.

VIPER

Siglas de View-Interactor-Presenter-Entity-Router. VIPER se utiliza para organizar el contenido de una aplicación en módulos. Dichos módulos tienen una responsabilidad específica y son reutilizables. Asimismo, es fácilmente testeable.

- View: es la vista que acabaremos mostrando por pantalla.
- Presenters: tiene una doble misión.
 - o Se encarga de recibir acciones de la View (como cuando se da al botón de login con usuario y contraseña).
 - o También se encarga de presentar los datos a la View.
- Interactor: es quien recibe la información que le pide el Presenter.
 - o Es donde se encuentra la lógica de negocio.
 - o Es muy común que sea una petición HTTP o a una BBDD.
 - o Una vez recibe la respuesta de la petición, envía la respuesta al presenter para que decida qué hacer con esa información.
- Router: permiten navegar a otras pantallas dentro de nuestra aplicación.
 - o Son los encargados de recibir una orden del presenter indicando dónde queremos navegar.
 - o Una vez en el otro módulo, encontramos los mismos elementos en dicho módulo.
- Entities: son los modelos. Es el modelo con el que trabaja el interactor.

MVP

De las siglas Model-View-Presenter.

El Model representa los datos y la lógica de negocio. Tiene la responsabilidad de guardar y de manipular los datos de la aplicación. Por ejemplo, hace las peticiones HTTP, acceder y persistir a la BBDD, etc.

La View es lo que vemos por pantalla.

El Presenter es el intermediario entre el modelo y la vista. Para poder comunicar la información que le llega del modelo a la vista se utilizan los protocolos (Delegation pattern).

Al estar desacoplado, permite que el código sea reutilizable, que el código sea testeable y que la aplicación sea escalable.

En esta arquitectura, el Presenter conoce a la View, a diferencia del ViewModel y el Model en la arquitectura MVVM.

Se usa en UIKit y Android tradicional, pero no en SwiftUI ni JetPack Compose.

MVVM

Model-View-ViewModel (MVVM) es un patrón arquitectónico que Apple utiliza para diseñar aplicaciones de iOS construidas con su *framework* SwiftUI.

Este patrón se divide en tres niveles diferentes: la capa de negocio, la interfaz gráfica y la lógica de presentación.

Componentes de MVVM para SwiftUI en aplicaciones para iOS:

Se basa en 3 capas:

- Model: es la capa de acceso a datos que se encarga de gestionar la información que maneja la aplicación y que en Apple se amplía también a las definiciones de los propios datos. Hace también las peticiones (no en el mismo struct de la estructura del objeto del modelo).
- View: esta capa es la encargada de ofrecer en la pantalla la información para el usuario. De modo que, son las estructuras que se implementan con el protocolo ‘View’. La información se la proporciona el ViewModel.
- View Model: es una capa intermediaria entre el modelo y la vista que contiene toda la lógica de presentación de una pantalla que utiliza el protocolo “Observable Object”.
 - Realiza las acciones en las que desembocan las interacciones del usuario con la vista (como pulsar un botón).
 - Modifica el estado de la vista.

Beneficios de la arquitectura MVVM:

- Permite que las diferentes apps sean fáciles de leer, de mantener y de testear.
- Permite reutilizar componentes.

MVVM + Iterator

Iterator es un patrón de diseño que se usa para recorrer colecciones de objetos sin exponer su estructura interna.

En Swift, ya usamos el patrón Iterator casi sin darnos cuenta, porque `ForEach`, `List` o `.map` están usando este patrón.

En contexto MVVM + SwiftUI, se emplea cuando:

- Iteras elementos en tu View para mostrarlos.
- Separas la lógica de iteración dentro del ViewModel si necesitas personalizarla.

En SwiftUI rara vez necesitamos escribir un iterador a mano, ya que `ForEach`, `List`, `.map`, `.filter` ya lo realizan automáticamente.

En resumidas cuentas, sería filtrar en el ViewModel en vez de en el View.

Ventajas:

- Separación de responsabilidades: la View solo muestra, no decide qué mostrar.
- Reutilización: puedes usar el mismo ViewModel en distintas vistas con distintos criterios.
- Testabilidad: puedes testear la lógica de filtrado sin necesitar una vista.
- Limpieza: tu vista es más clara.

Así pues, la arquitectura que vamos a emplear es la de MVVM, en la cual se aplicará el Iterator en los `ForEach`, `List` o `.map`.

Posteriormente, era el turno de realizar la instalación de Xcode y comenzar el proyecto.

Para ello, al crear el proyecto se estableció la versión mínima de iOS sobre la cual funcionar, la cual es iOS 17 debido a que introdujo grandes cambios en SwiftUI, por

BeSport24Training

lo que podemos contar con prácticamente todas las novedades de cara a utilizar una aplicación más optimizada y con mayor rendimiento. También se establecieron los dispositivos soportados, incluyendo iPhone, iPad, macOS y visionOS y la orientación según el dispositivo:

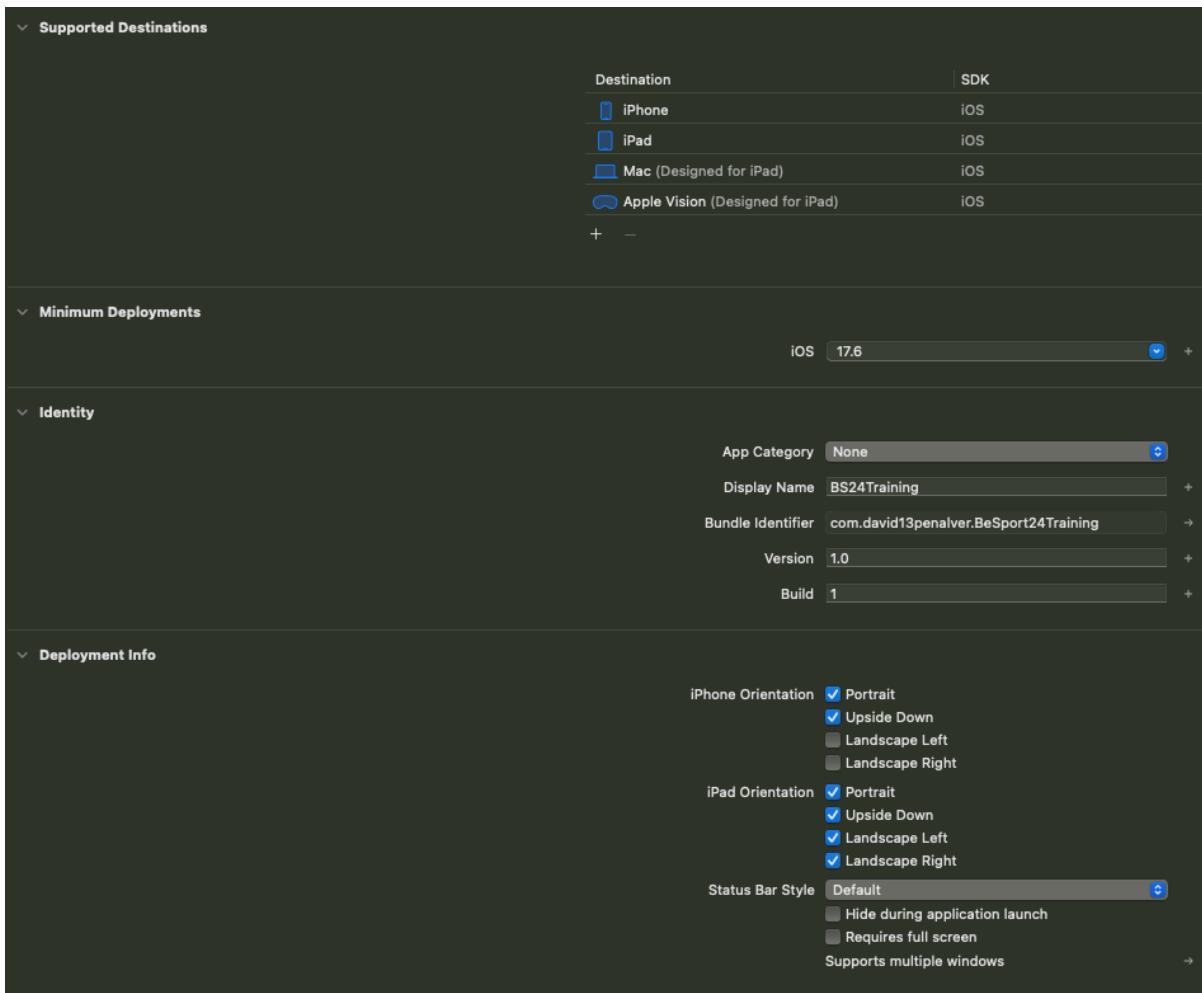


Figura 64. Información general del proyecto.

El siguiente paso fue crear la estructura del proyecto, la cual, como hemos mencionado, sigue la estructura de MVVM + Interactor. A ello, se le añade una capa extra llamada API que, como su nombre dice, se encarga de la conexión con la API.

En cuanto al Interactor, es la capa que mapea del modelo de negocio al DTO y viceversa. El ViewModel es el encargado de la lógica de negocio del frontend. La View es la vista que podremos visualizar desde nuestro dispositivo.

BeSport24Training

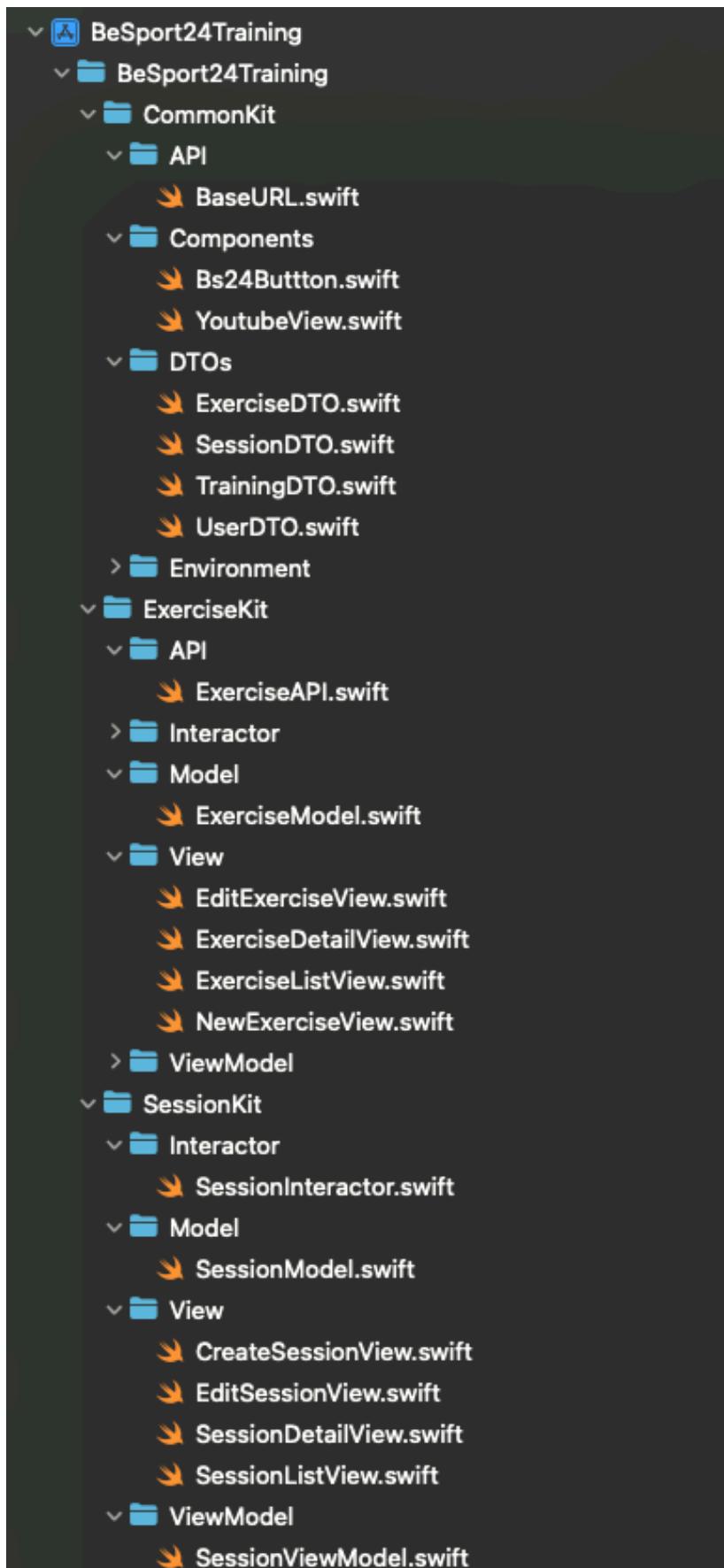


Figura 65. Scaffolding del frontend.

BeSport24Training

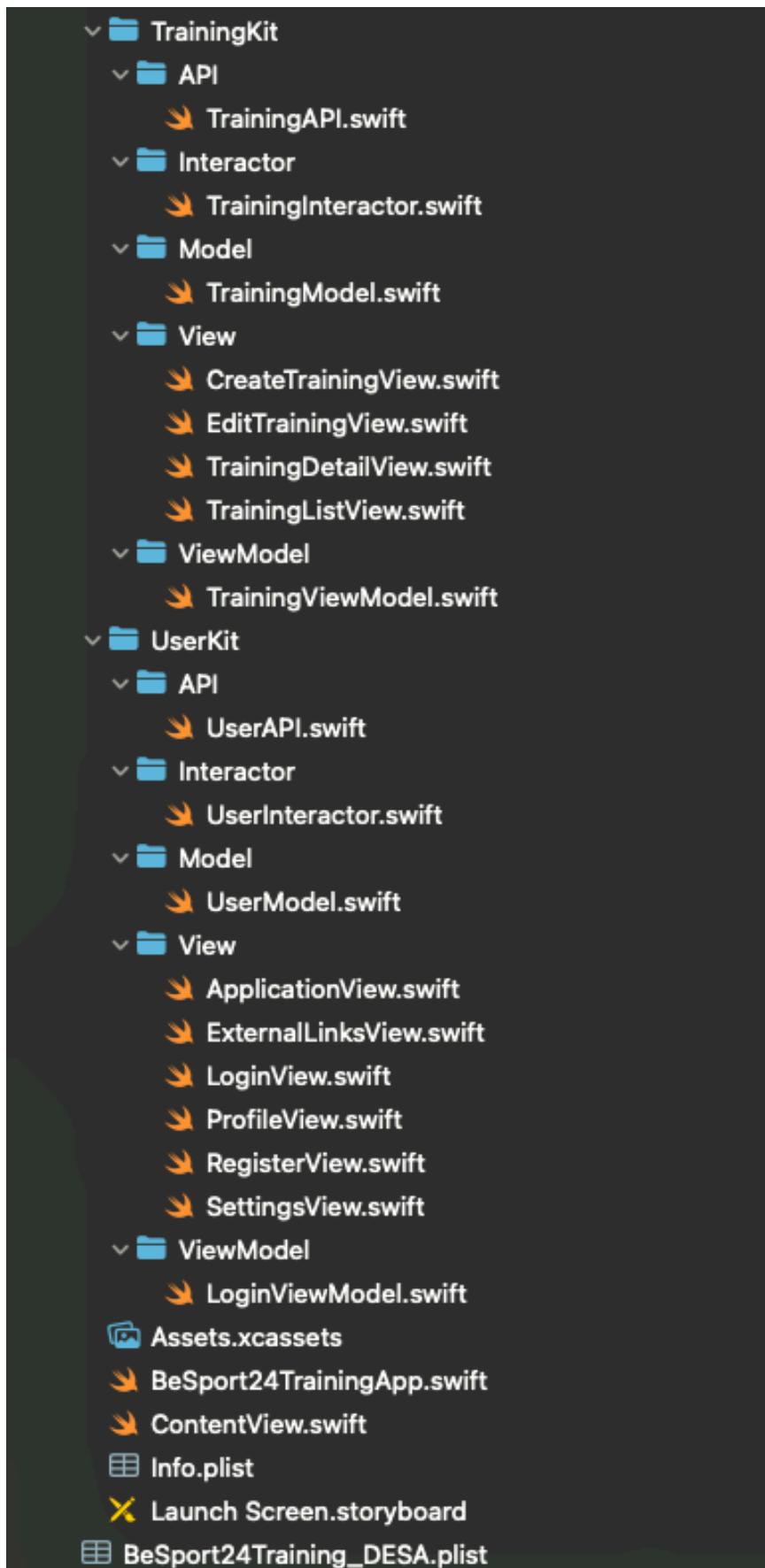


Figura 65. Scaffolding del frontend (continuación).

El siguiente paso fue crear la pantalla de inicio de la aplicación, conocido como Launch Screen, que es la pantalla que se muestra cuando arranca la aplicación, siendo la siguiente:



Figura 66. Launch Screen de la aplicación.

Y el ícono de la aplicación que se verá en el cajón de aplicaciones o en la pantalla de inicio:



Figura 67. Icono de la aplicación.

Tras ello, se realizó el CRUD de cada uno de los apartados de la aplicación. Explicaremos todo ello mediante la parte de Ejercicios, como en el ejemplo del back.

Respecto a la API, tenemos el fichero `ExerciseAPI`, en el cual se encuentra el protocolo y la implementación. El protocolo es el equivalente a las interfaces en Swift.

BeSport24Training

```
1 //  
2 //  ExerciseAPI.swift  
3 //  BeSport24Training  
4 //  
5 //  Created by David Peñalver Navarro on 2/5/25.  
6 //  
7  
8 import Foundation  
9  
10 protocol ExerciseAPIProtocol {  
11     func getAllExercises() async throws -> [ExerciseDTO]  
12     func findExerciseById(id: Int) async throws -> ExerciseDTO  
13     func createExercise(dto: ExerciseDTO) async throws  
14     func updateExercise(dto: ExerciseDTO) async throws  
15     func deleteExercise(id: Int) async throws  
16 }  
17  
18 struct ExerciseListResponse: Codable {  
19     let data: [ExerciseDTO]  
20 }  
21  
22 final class ExerciseAPIImpl: ExerciseAPIProtocol {  
23     func getAllExercises() async throws -> [ExerciseDTO] {  
24         guard let url = URL(string: "\(BaseUrl.baseURL)exercises") else { throw URLError(.badURL) }  
25  
26         let (data, _) = try await URLSession.shared.data(from: url)  
27  
28         let decodedResponse = try JSONDecoder().decode(ExerciseListResponse.self, from: data)  
29         return decodedResponse.data  
30     }  
31  
32     func findExerciseById(id: Int) async throws -> ExerciseDTO {  
33         guard let url = URL(string: "\(BaseUrl.baseURL)exercises/\(id)") else { throw URLError(.badURL) }  
34  
35         let (data, _) = try await URLSession.shared.data(from: url)  
36  
37         let decodedResponse = try JSONDecoder().decode(ExerciseDTO.self, from: data)  
38         return decodedResponse  
39     }  
40 }
```

Figura 68. *ExerciseAPI*.

```
22 final class ExerciseAPIImpl: ExerciseAPIProtocol {
23     func createExercise(dto: ExerciseDTO) async throws {
24         guard let url = URL(string: "\(BaseUrl.baseURL)exercises") else { throw URLError(.badURL) }
25
26         var request = URLRequest(url: url)
27         request.httpMethod = "POST"
28         request.setValue("application/json", forHTTPHeaderField: "Content-Type")
29         request.httpBody = try JSONEncoder().encode(dto)
30
31         (_, _) = try await URLSession.shared.data(for: request)
32     }
33
34     func updateExercise(dto: ExerciseDTO) async throws {
35         guard let id: Int = dto.id,
36               let url = URL(string: "\(BaseUrl.baseURL)exercises/\(id)") else {
37             throw URLError(.badURL)
38         }
39
40         var request = URLRequest(url: url)
41         request.httpMethod = "PUT"
42         request.setValue("application/json", forHTTPHeaderField: "Content-Type")
43         request.httpBody = try JSONEncoder().encode(dto)
44
45         (_, _) = try await URLSession.shared.data(for: request)
46     }
47
48     func deleteExercise(id: Int) async throws {
49         guard let url = URL(string: "\(BaseUrl.baseURL)exercises/\(id)") else {
50             throw URLError(.badURL)
51         }
52
53         var request = URLRequest(url: url)
54         request.httpMethod = "DELETE"
55
56         (_, _) = try await URLSession.shared.data(for: request)
57     }
58
59 }
60
61 }
```

Figura 68. ExerciseAPI (continuación).

Tal y como podemos apreciar, en Swift es común, cuando solo se tiene una implementación de la interfaz, tener ambas en el mismo archivo. En el caso de que se empleara en más de un archivo, ya se extraería a uno propio y se seguiría la misma estructura que hemos empleado en Java de tener el protocolo en la carpeta de ExerciseAPI y, también en dicha carpeta, la carpeta de ExerciseAPIImpl con las implementaciones de la API. Lo mismo ocurriría con la estructura ExerciseListResponse, que solamente se usa en dicho archivo y por ello se deja conjuntamente.

Avanzando con el fichero, encontramos ya las implementaciones. Aquí, como tenemos aspectos comunes a todos los ficheros de API, se han extraído al paquete Common. En concreto, se trata de la URL a emplear. De este modo, sería fácilmente modificable emplear una u otra según el entorno empleado, ya que, si observamos la Figura 65, encontramos archivos de configuración diferentes. El archivo info.plist

BeSport24Training

es el correspondiente al entorno de producción, mientras que BeSport24Training_DESA es el del entorno de desarrollo. Volviendo al archivo común, sería este:

```
1 //  
2 //  BaseURL.swift  
3 //  BeSport24Training  
4 //  
5 //  Created by David Peñalver Navarro on 2/5/25.  
6 //  
7  
8 import Foundation  
9  
10 struct BaseURL {  
11     static var baseURL: String = "http://192.168.1.53:8080/api/"  
12 }  
13
```

Figura 69. Fichero BaseURL.

Todos aquellos datos que fueran comunes, se meterían en esa carpeta y, mientras no fueran muy abultados, podrían estar en ese fichero renombrándolo a un nombre más genérico.

Visto ello, solo nos queda ver las diferentes implementaciones de los métodos, las cuales siguen una estructura común:

- En primer lugar, se crea la función con el nombre y los parámetros que recibe.
- A continuación, se indica que es una función asíncrona y que puede lanzar excepciones.
- Finalmente, se indica aquello que retorna, en el caso de que tenga que retornar algo.

Dentro de los métodos, la funcionalidad es similar:

- En primer lugar, se crea una guarda, la cual se encarga de crear la constante de la URL en el caso de que esta sea correcta. Si no existe, lanza directamente la excepción pertinente.
- Seguidamente, se crea la variable de la petición.
- Finalmente, se retorna aquello que se requiere en los métodos que lo requieren.

No obstante, aquí encontramos diferencias entre las funciones GET y las restantes.

En las GET, tras crear la guarda, se hace directamente la petición de dicha URL y se decodifica el JSON obtenido. Por último, se retorna la petición, que iría al interceptor. Para que ello sea más sencillo, el DTO empleado tiene la misma estructura que el data del JSON recibido:

```
1 // ExerciseDTO.swift
2 // BeSport24Training
3 //
4 //
5 // Created by David Peñalver Navarro on 1/5/25.
6 //
7
8 struct ExerciseDTO: Codable, Identifiable {
9     var id: Int? { exerciseId }
10    let exerciseId: Int
11    let name: String
12    let imageURL: String
13    let videoURL: String?
14    let isGlobal: Bool?
15    let description: String?
16    let exerciseType: ExerciseTypeDTO
17    let exerciseSubtype: ExerciseSubtypeDTO
18
19    init(exerciseId: Int, name: String, imageURL: String, videoURL: String?, isGlobal: Bool?, description: String?, exerciseType: ExerciseTypeDTO,
20         exerciseSubtype: ExerciseSubtypeDTO) {
21        self.exerciseId = exerciseId
22        self.name = name
23        self.imageURL = imageURL
24        self.videoURL = videoURL
25        self.isGlobal = isGlobal
26        self.description = description
27        self.exerciseType = exerciseType
28        self.exerciseSubtype = exerciseSubtype
29    }
30
31 struct ExerciseTypeDTO: Codable {
32     let exerciseTypeId: Int
33     let name: String
34 }
35
36 struct ExerciseSubtypeDTO: Codable {
37     let exerciseTypeId: Int
38     let name: String
39 }
40
```

Figura 70. Estructura del DTO.

En cuanto a las peticiones PUT, POST y DELETE, el proceso difiere. En estos casos, se crea una variable de la petición, a la cual se le añade la URL y, posteriormente, el tipo de método. En los casos PUT y POST, se añade la cabecera del tipo que contenido que sea tipo JSON y, en la siguiente línea, el cuerpo de la petición. Por último, al igual que en todos los métodos, se envía la petición al *backend*.

Tras ello, pasamos al Interactor, que se encargará de mapear del DTO (Figura 70) al objeto de dominio (Figura 71):

```
8 import Foundation
9
10 struct ExerciseDescription: Codable {
11     let what_is: String
12     let muscles_involved: String
13     let steps: [String]
14 }
15
16 struct ExerciseModel: Encodable, Sendable, Identifiable {
17     let exerciseId: Int?
18     let name: String
19     let imageURL: String?
20     let videoURL: String?
21     let isGlobal: Bool?
22     let description: String?
23     let typeName: String
24     let subtypeName: String
25
26     var id: Int? { exerciseId }
27
28     var parsedDescription: ExerciseDescription? {
29         guard let description, let data = description.data(using: .utf8) else { return nil }
30         return try? JSONDecoder().decode(ExerciseDescription.self, from: data)
31     }
32
33     init(from dto: ExerciseDTO) {
34         self.exerciseId = dto.exerciseId
35         self.name = dto.name
36         self.imageURL = dto.imageURL
37         self.videoURL = dto.videoURL
38         self.isGlobal = dto.isGlobal
39         self.description = dto.description
40         self.typeName = dto.exerciseType.name
41         self.subtypeName = dto.exerciseSubtype.name
42     }
43
44     init(exerciseId: Int, name: String, imageURL: String?, videoURL: String?, isGlobal: Bool?, description: String?, typeName: String, subtypeName: String) {
45         self.exerciseId = exerciseId
46         self.name = name
47         self.imageURL = imageURL
48         self.videoURL = videoURL
49         self.isGlobal = isGlobal
50         self.description = description
51         self.typeName = typeName
52         self.subtypeName = subtypeName
53     }
54 }
```

Figura 71. Estructura del objeto de dominio.

Como podemos apreciar en ambas figuras, existen variables que contienen un interrogante. Ello se debe a que son variablesopcionales, por lo que recibirán un valor del *back* o, si no lo reciben, serán *nil*, el equivalente a *null*. Ello permite, como ventaja, que con un mismo modelo podamos recibir distintos tipos de información, como podría ser el JSON que recibamos de la lista o el que recibamos del detalle, que contendrá más campos. Por el contrario, debemos tener cuidado con esos *nil*, ya que, en el caso de no *unwrappear* correctamente estas propiedades, puede provocar un *crash* en la aplicación.

Ahora sí, la estructura del Interactor es la siguiente:

BeSport24Training

```
1 //  
2 //  ExerciseInteractor.swift  
3 //  BeSport24Training  
4 //  
5 //  Created by David Peñalver Navarro on 1/5/25.  
6 //  
7  
8 protocol ExerciseInteractorProtocol {  
9     func getAllExercises() async throws -> [ExerciseModel]  
10    func findExerciseById(id: Int) async throws -> ExerciseModel  
11    func createExercise(exercise: ExerciseModel) async throws  
12    func updateExercise(exercise: ExerciseModel) async throws  
13    func deleteExercise(id: Int) async throws  
14 }  
15  
16 class ExerciseInteractorImpl: ExerciseInteractorProtocol {  
17  
18     private let api: ExerciseAPIProtocol  
19  
20     init(api: ExerciseAPIProtocol = ExerciseAPIImpl()) {  
21         self.api = api  
22     }  
23 }  
24  
25 @APIActor  
26 extension ExerciseInteractorImpl {  
27  
28     func getAllExercises() async throws -> [ExerciseModel] {  
29         let dtos = try await api.getAllExercises()  
30         return dtos.map { ExerciseModel(from: $0) }  
31     }  
32  
33     func findExerciseById(id: Int) async throws -> ExerciseModel {  
34         let dto = try await api.findExerciseById(id: id)  
35         return ExerciseModel(from: dto)  
36     }  
37  
38     func createExercise(exercise: ExerciseModel) async throws {  
39         var exerciseTypeDTO: ExerciseTypeDTO!  
40         var exerciseSubtypeDTO: ExerciseSubtypeDTO!  
41  
42         switch exercise.typeName {  
43             case "Resistance":  
44                 exerciseTypeDTO = .init(exerciseTypeId: 1, name: "Resistance")  
45             case "Cardio":  
46                 exerciseTypeDTO = .init(exerciseTypeId: 2, name: "Cardio")  
47             case "Mobility":  
48                 exerciseTypeDTO = .init(exerciseTypeId: 3, name: "Mobility")  
49             default:  
50                 fatalError("Unsupported exercise type")  
51         }  
52  
53         switch exercise.subtypeName {  
54             case "Chest":  
55                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 1, name: "Chest")  
56             case "Back":  
57                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 2, name: "Back")  
58             case "Shoulders":  
59                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 3, name: "Shoulders")  
60             case "HIIT":  
61                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 4, name: "HIIT")  
62             case "SIT":  
63                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 5, name: "SIT")  
64             case "LISS":  
65                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 6, name: "LISS")  
66             case "Static Stretching":  
67                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 7, name: "Static Stretching")  
68             case "Dynamic Stretching":  
69                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 8, name: "Dynamic Stretching")  
70             case "Balistic Stretching":  
71                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 9, name: "Balistic Stretching")  
72             case "Passive Stretching":  
73                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 10, name: "Passive Stretching")  
74             case "Leg":  
75                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 11, name: "Leg")  
76             case "Core":  
77                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 12, name: "Core")  
78             case "Biceps":  
79                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 13, name: "Biceps")  
80             case "Glutes":  
81                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 14, name: "Glutes")  
82             case "Calves":  
83                 exerciseSubtypeDTO = .init(exerciseSubtypeId: 15, name: "Calves")  
84 }
```

figura 72. Estructura del Interactor.

BeSport24Training

```
38     func createExercise(exercise: ExerciseModel) async throws {
39         case "Quadriceps":
40             exerciseSubtypeDTO = .init(exerciseTypeId: 16, name: "Quadriceps")
41         case "Hamstrings":
42             exerciseSubtypeDTO = .init(exerciseTypeId: 17, name: "Hamstrings")
43         case "Triceps":
44             exerciseSubtypeDTO = .init(exerciseTypeId: 18, name: "Triceps")
45         default:
46             fatalError("Unsupported exercise subtype")
47     }
48
49     let dto = ExerciseDTO(exerciseId: -1, name: exercise.name, imageURL: exercise.imageURL ?? "", videoURL: exercise.videoURL, isGlobal: exercise.isGlobal, description: exercise.description, exerciseType: exerciseTypeDTO, exerciseSubtype: exerciseSubtypeDTO)
50
51     try await api.createExercise(dto: dto)
52 }
53
54 func updateExercise(exercise: ExerciseModel) async throws {
55     var exerciseTypeDTO: ExerciseTypeDTO!
56     var exerciseSubtypeDTO: ExerciseSubtypeDTO!
57
58     switch exercise.typeName {
59     case "Resistance":
60         exerciseTypeDTO = .init(exerciseTypeId: 1, name: "Resistance")
61     case "Cardio":
62         exerciseTypeDTO = .init(exerciseTypeId: 2, name: "Cardio")
63     case "Mobility":
64         exerciseTypeDTO = .init(exerciseTypeId: 3, name: "Mobility")
65     default:
66         fatalError("Unsupported exercise type")
67     }
68
69     switch exercise.subtypeName {
70     case "Chest":
71         exerciseSubtypeDTO = .init(exerciseTypeId: 1, name: "Chest")
72     case "Back":
73         exerciseSubtypeDTO = .init(exerciseTypeId: 2, name: "Back")
74     case "Shoulders":
75         exerciseSubtypeDTO = .init(exerciseTypeId: 3, name: "Shoulders")
76     case "HIIT":
77         exerciseSubtypeDTO = .init(exerciseTypeId: 4, name: "HIIT")
78     case "SIT":
79         exerciseSubtypeDTO = .init(exerciseTypeId: 5, name: "SIT")
80     case "LISS":
81         exerciseSubtypeDTO = .init(exerciseTypeId: 6, name: "LISS")
82     case "Static Stretching":
83         exerciseSubtypeDTO = .init(exerciseTypeId: 7, name: "Static Stretching")
84     case "Dynamic Stretching":
85         exerciseSubtypeDTO = .init(exerciseTypeId: 8, name: "Dynamic Stretching")
86     case "Balistic Stretching":
87         exerciseSubtypeDTO = .init(exerciseTypeId: 9, name: "Balistic Stretching")
88     case "Passive Stretching":
89         exerciseSubtypeDTO = .init(exerciseTypeId: 10, name: "Passive Stretching")
90     case "Leg":
91         exerciseSubtypeDTO = .init(exerciseTypeId: 11, name: "Leg")
92     case "Core":
93         exerciseSubtypeDTO = .init(exerciseTypeId: 12, name: "Core")
94     case "Biceps":
95         exerciseSubtypeDTO = .init(exerciseTypeId: 13, name: "Biceps")
96     case "Glutes":
97         exerciseSubtypeDTO = .init(exerciseTypeId: 14, name: "Glutes")
98     case "Calves":
99         exerciseSubtypeDTO = .init(exerciseTypeId: 15, name: "Calves")
100    case "Quadriceps":
101        exerciseSubtypeDTO = .init(exerciseTypeId: 16, name: "Quadriceps")
102    case "Hamstrings":
103        exerciseSubtypeDTO = .init(exerciseTypeId: 17, name: "Hamstrings")
104    case "Triceps":
105        exerciseSubtypeDTO = .init(exerciseTypeId: 18, name: "Triceps")
106    default:
107        fatalError("Unsupported exercise subtype")
108    }
109
110    let dto = ExerciseDTO(exerciseId: exercise.id ?? 0, name: exercise.name, imageURL: exercise.imageURL ?? "", videoURL: exercise.videoURL, isGlobal: exercise.isGlobal, description: exercise.description, exerciseType: exerciseTypeDTO, exerciseSubtype: exerciseSubtypeDTO)
111    print(dto)
112    try await api.updateExercise(dto: dto)
113 }
114
115 func deleteExercise(id: Int) async throws {
116     try await api.deleteExercise(id: id)
117 }
118
119
120 }
```

Figura 72. Estructura del Interactor (continuación).

Tal y como podemos apreciar, este archivo se encarga de mapear aquellos datos que van de nuestro modelo de dominio al *backend* y viceversa.

Para ello, se inyecta la dependencia de la API tal y como hacíamos el año pasado manualmente en Java en las clases de César Guijarro. A su vez, como el protocolo

solo es empleado por una sola implementación, los colocamos en el mismo archivo de Swift.

Por ello, en los métodos GET tan solo tiene que hacer la llamada a la API y mapear el objeto al modelo de dominio. En el caso de la lista, mediante un mapeo con programación funcional. En el caso del detalle, con un simple mapeo. La ventaja de estos mapeos es que se realizan en el propio init, que sería el equivalente al constructor de Java. Si retrocedemos a las Figura 70 y Figura 71, podemos ver que una de las formas de inicializar estos objetos es introduciendo como parámetro el objeto desde el cual queremos mapear; por lo que se realiza de manera rápida y efectiva sin necesidad de crear nuevos ficheros en nuestra aplicación que se dediquen específicamente a este proceso.

El método del DELETE simplemente envía el identificador del ejercicio que queremos eliminar.

Finalmente, los métodos POST y PUT sí que realizan una conversión del parámetro que se introduce, pues hay diferencias en el modelo de dominio y en el DTO respecto al apartado de tipo y subtipo de ejercicio. Esto se ha hecho con el simple objetivo de mostrar cómo en el Interactor puede haber cierta lógica en el caso de que los modelos de negocio y el DTO no coincidan.

El siguiente paso es cuando llegamos al ViewModel, que se encarga de la lógica, como el servicio en Java. A continuación, podemos analizar el ViewModel de los ejercicios:

BeSport24Training

```
4  // Created by David Peñalver Navarro on 1/5/25.
5  // 
6  //
7
8 import Foundation
9
10 @MainActor
11 final class ExerciseViewModel: ObservableObject {
12
13     @Published var exercise: ExerciseModel?
14     @Published var exercises: [ExerciseModel] = []
15
16     @Published var showAlert: Bool = false
17     @Published var alertMessage: String = ""
18
19     private let interactor: ExerciseInteractorProtocol
20
21     init(interactor: ExerciseInteractorProtocol = ExerciseInteractorImpl()) {
22         self.interactor = interactor
23     }
24
25     func getAllExercises() async {
26         do {
27             let response = try await interactor.getAllExercises()
28             exercises = response
29         } catch {
30             print("Error loading exercises: \(error.localizedDescription)")
31         }
32     }
33
34     func findExerciseById(id: Int) async {
35         do {
36             let response = try await interactor.findExerciseById(id: id)
37             exercise = response
38         } catch {
39             print("Error loading exercise: \(error.localizedDescription)")
40         }
41     }
42
43     func createExercise(name: String, imageURL: String, videoURL: String, isGlobal: Bool, description: String, typeName: String, subtypeName: String) {
44         async {
45             let trimmedName = name.trimmingCharacters(in: .whitespacesAndNewlines)
46             let trimmedImageURL = imageURL.trimmingCharacters(in: .whitespacesAndNewlines)
47             let trimmedVideoURL = videoURL.trimmingCharacters(in: .whitespacesAndNewlines)
48             let trimmedDescription = description.trimmingCharacters(in: .whitespacesAndNewlines)
49             let trimmedTypeName = typeName.trimmingCharacters(in: .whitespacesAndNewlines)
50             let trimmedSubtypeName = subtypeName.trimmingCharacters(in: .whitespacesAndNewlines)
51
52             if trimmedName.isEmpty {
53                 alertMessage = "Name is required"
54                 showAlert = true
55                 return
56             } else if trimmedImageURL.isEmpty {
57                 alertMessage = "Image URL is required"
58                 showAlert = true
59                 return
60             } else if trimmedVideoURL.isEmpty {
61                 alertMessage = "Video URL is required"
62                 showAlert = true
63                 return
64             } else if trimmedDescription.isEmpty {
65                 alertMessage = "Description is required"
66                 showAlert = true
67                 return
68             } else if trimmedTypeName.isEmpty {
69                 alertMessage = "Type Name is required"
70                 showAlert = true
71                 return
72             } else if trimmedSubtypeName.isEmpty {
73                 alertMessage = "Subtype Name is required"
74                 showAlert = true
75                 return
76             }
77
78             let newExercise = ExerciseModel(
79                 exerciseId: 0,
80                 name: trimmedName,
81                 imageURL: trimmedImageURL,
82                 videoURL: trimmedVideoURL,
83                 isGlobal: isGlobal,
84                 description: trimmedDescription,
85                 typeName: trimmedTypeName,
86                 subtypeName: trimmedSubtypeName
87             )
        }
```

Figura 73. ViewModel de los ejercicios.

```
88     do {
89         try await interactor.createExercise(exercise: newExercise)
90     } catch {
91         print(error.localizedDescription)
92     }
93 }
94
95 func updateExercise(name: String, imageURL: String, videoURL: String, isGlobal: Bool, description: String, typeName: String, subtypeName: String
96     async {
97     let trimmedName = name.trimmingCharacters(in: .whitespacesAndNewlines)
98     let trimmedImageURL = imageURL.trimmingCharacters(in: .whitespacesAndNewlines)
99     let trimmedVideoURL = videoURL.trimmingCharacters(in: .whitespacesAndNewlines)
100    let trimmedDescription = description.trimmingCharacters(in: .whitespacesAndNewlines)
101    let trimmedTypeName = typeName.trimmingCharacters(in: .whitespacesAndNewlines)
102    let trimmedSubtypeName = subtypeName.trimmingCharacters(in: .whitespacesAndNewlines)
103
104   if trimmedName.isEmpty {
105       alertMessage = "Name is required"
106       showAlert = true
107       return
108   } else if trimmedImageURL.isEmpty {
109       alertMessage = "Image URL is required"
110       showAlert = true
111       return
112   } else if trimmedVideoURL.isEmpty {
113       alertMessage = "Video URL is required"
114       showAlert = true
115       return
116   } else if trimmedDescription.isEmpty {
117       alertMessage = "Description is required"
118       showAlert = true
119       return
120   } else if trimmedTypeName.isEmpty {
121       alertMessage = "Type Name is required"
122       showAlert = true
123       return
124   } else if trimmedSubtypeName.isEmpty {
125       alertMessage = "Subtype Name is required"
126       showAlert = true
127       return
128   }
129
130   let newExercise = ExerciseModel(
131     exerciseId: exercise?.id ?? 0,
132     name: trimmedName,
133     imageURL: trimmedImageURL,
134     videoURL: trimmedVideoURL,
135     isGlobal: isGlobal,
136     description: trimmedDescription,
137     typeName: trimmedTypeName,
138     subtypeName: trimmedSubtypeName
139   )
140
141   do {
142       try await interactor.updateExercise(exercise: newExercise)
143   } catch {
144       print(error.localizedDescription)
145   }
146 }
147
148 func deleteExercise(id: Int) async {
149     do {
150         try await interactor.deleteExercise(id: id)
151     } catch {
152         print(error.localizedDescription)
153     }
154 }
155 }
```

Figura 73. ViewModel de los ejercicios (continuación).

Esta es, como hemos comentado, la parte de la aplicación (en este caso concreto del apartado de ejercicios) que controla la lógica y que, con sus modificaciones, altera o modifica los datos que se muestran en la vista. Por tanto, cubre los casos prácticos que se llevan en todas ellas y que tienen que ver con los ejercicios: la muestra del listado de ejercicios, la muestra del detalle de un ejercicio, la creación de un ejercicio, la edición de uno existente y la eliminación de un ejercicio existente.

Por ello, contiene a la variable del array de ejercicios y la del ejercicio y también la opción de que se muestre el error y el mensaje a mostrar. A continuación, como en el caso anterior, la inyección de dependencias. Finalmente, los métodos que venimos

BeSport24Training

comentando a lo largo del CRUD. En ellos, cabe destacar las validaciones que se realizan en los métodos de PUT y POST de cara a que los datos que se envíen al servidor sean los correctos. Asimismo, es aquí donde se recogen los `catch` de los errores que se puedan lanzar desde el resto de métodos.

Finalmente, hay que destacar también que las variables que comentábamos en el párrafo anterior son `@Published` ya que van a ser modificadas por la vista. Para que ello sea posible, el `ViewModel` tiene que conformar el protocolo `ObservableObject`.

Por último, llegamos ya a la `View` o vista, aquello con lo que interactúa el usuario de la aplicación. Una de las ventajas de Xcode es que contamos con el Canvas, motivo por el cual podemos ver prácticamente en tiempo real en la misma aplicación cómo van a responder nuestros cambios. En este mismo canvas podemos hacer que se muestre la opción del modo claro y/o del modo oscuro, la orientación del dispositivo o el tamaño de los componentes debido a las opciones de accesibilidad de Apple. Asimismo, también podemos configurar el tipo del dispositivo, como podría ser el modelo concreto de iPhone o iPad. De este modo, rápidamente podemos establecer la vista óptima para cada tipo de dispositivo en función a todos estos parámetros:

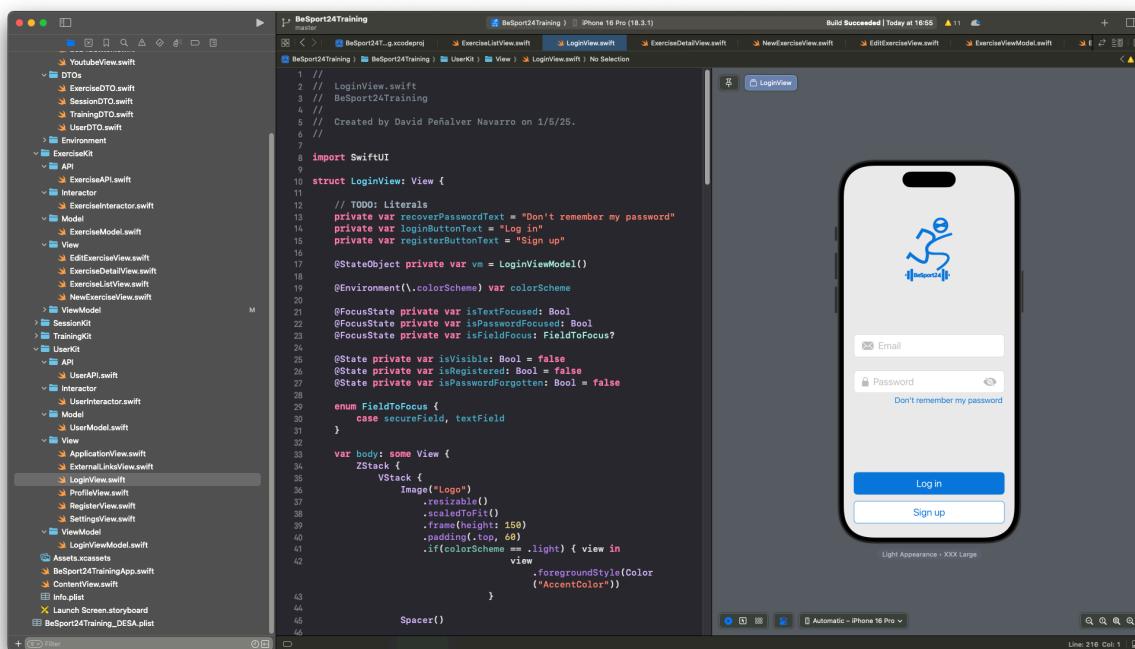


Figura 74. Ejemplo de cómo se ve Xcode cuando estamos con una vista mostrando la vista del login en un iPhone 16 Pro.

Estas vistan pueden ser de unas pocas decenas de líneas a cientos de ellas. Por ejemplo, la vista del *login* contiene más de 200, mientras que la de la lista de ejercicios se queda en 60.

Para hacernos una idea de cómo pueden ser estas vistas, veremos dos ejemplos: el listado de ejercicios y la creación de uno nuevo.

Respecto al listado de ejercicios, el código es el siguiente:

```
1 // ExerciseListView.swift
2 // BeSport24Training
3 //
4 // Created by David Peñalver Navarro on 1/5/25.
5 //
6
7 import SwiftUI
8
9 struct ExerciseListView: View {
10
11     @StateObject var vm: ExerciseViewModel
12
13     @State private var searchText = ""
14     @State private var showingNewExerciseView = false
15
16     var filteredExercises: [ExerciseModel] {
17         if searchText.isEmpty {
18             return vm.exercises
19         } else {
20             return vm.exercises.filter { $0.name.localizedCaseInsensitiveContains(searchText) }
21         }
22     }
23
24
25     var body: some View {
26         NavigationStack {
27
28             List(filteredExercises) { exercise in
29                 NavigationLink(destination: ExerciseDetailView(exercise: exercise, vm: ExerciseViewModel())) {
30                     Text(exercise.name)
31                 }
32             }
33             .task {
34                 await vm.getAllExercises()
35             }
36             .navigationTitle("Exercises")
37             .searchable(text: $searchText, prompt: "Search for name...")
38             .toolbar {
39                 ToolbarItem(placement: .navigationBarTrailing) {
40                     Button(action: {
41                         showingNewExerciseView = true
42                     }) {
43                         Image(systemName: "plus")
44                     }
45                 }
46             }
47             .fullScreenCover(isPresented: $showingNewExerciseView) {
48                 NewExerciseView(isPresented: $showingNewExerciseView, vm: vm)
49             }
50             .refreshable {
51                 await vm.getAllExercises()
52             }
53         }
54     }
55
56     #Preview {
57         ExerciseListView(vm: ExerciseViewModel())
58     }
59 }
```

Figura 75. Vista del listado de ejercicios.

Gracias a la inclusión del *NavigationStack* en SwiftUI a partir de iOS 16, la creación de menús es mucho más sencilla que antes. De este modo, mostrar un listado en el cual poder acceder a cada una de las opciones se logra con unas pocas líneas de código. Es por ello que, en esta vista, casi todo el código son las opciones de dicho *NavigationStack* y las del buscador.

Para ello, las variables propias que emplea la vista se deben crear previamente a la variable `body`, que es la que usa SwiftUI por defecto para mostrar, con `some View`, la vista en nuestro dispositivo, en el canvas y en el simulador; ya que las vistas son variables.

De este modo, destacamos que se emplea la variable precedida de `@StateObject` para referirnos a que recibe los datos del `ViewModel` y que, además, están actualizados. Las variables `@State` son las variables propias de la vista, que mantienen conectados los datos con lo que puede editar el usuario, tal y como sería en un *Two Way Data Binding* de Angular. La variable restante es la que será usada en la lista y que recibe los parámetros del texto de la búsqueda para mostrar los ejercicios que coincidan con ella o todos en el caso de que dicho campo de texto se encuentre vacío.

Dentro del `NavigationStack`, destacamos los modificadores, que permiten lo siguiente:

- El modificador `.task` es el que se emplea para las tareas que necesitan ejecutar métodos asíncronos. Debido a la naturaleza de este modificador, aquello que se encuentra dentro se realiza antes de cargar la vista.
- El modificador `.navigationTitle` se encarga, evidentemente, del nombre que se mostrará en la búsqueda.
- El modificador `.searchable` es el que crea, de manera sencilla y nativa, el buscador de este elemento, aspecto que se ha facilitado considerablemente a versiones anteriores de SwiftUI. En él, se introduce la variable que se emplea para mostrar los resultados y el mensaje para cuando está vacía. Por defecto, también incluye un botón de borrar y de cancelar el proceso como veremos al final de estas explicaciones.
- El modificador `.toolbar` muestra los elementos que rodean aquello que está dentro del `NavigationStack`. En este caso, es el símbolo de “+” que servirá para crear un nuevo ejercicio. Cada uno de estos elementos es un `ToolbarItem`, que contiene entre paréntesis la localización en la cual se va a situar.
- El modificador `.fullScreenCover` es el que se lanza cuando se pulsa sobre el “+” comentado anteriormente, ya que solo se ejecuta cuando su valor es verdadero, el cual se configura así mediante el botón del previo. Como

veremos más adelante, la vista que se abre contiene la opción e cambiar esta variable a falso, hecho que cerrará inmediatamente la vista ya que, como podemos ver, se le envía dicho parámetro. Es una vista que aparece de abajo a arriba y que ocupa toda la pantalla. Existen otras opciones como `.sheet`, que abren una ventana que ocupa la práctica totalidad y de la cual se puede salir deslizando hacia abajo.

- El modificador `.refreshable` es el que se logra al deslizar de arriba abajo la vista y hace la función de recargar, tal y como sucede también en Android. La acción que realiza es la de refrescar los datos.

Una de las ventajas de usar estos elementos es que son nativos de Swift y SwiftUI, por lo que serán actualizados con la nueva versión de iOS que promete grandes cambios en la interfaz de usuario.

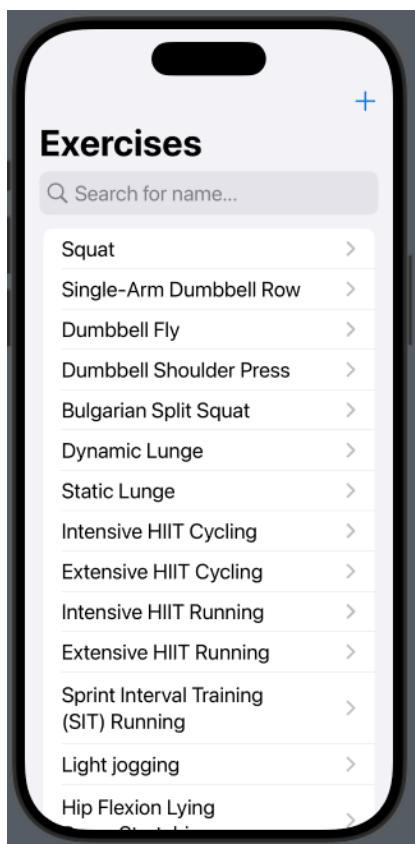


Figura 76. Vista del listado de ejercicios (en modo claro).

Por último en este apartado, detallaremos el apartado de la vista de creación de un ejercicio, ya que presenta una complejidad mayor.

BeSport24Training

```
1 //  
2 //  NewExerciseView.swift  
3 //  BeSport24Training  
4 //  
5 //  Created by David Peñalver Navarro on 4/5/25.  
6 //  
7  
8 import SwiftUI  
9  
10 struct NewExerciseView: View {  
11  
12     @Binding var isPresented: Bool  
13  
14     @ObservedObject var vm = ExerciseViewModel()  
15  
16     @State private var name: String = ""  
17     @FocusState private var isNameFocused: Bool  
18     @State private var imageURL: String = ""  
19     @FocusState private var isImageURLFocused: Bool  
20     @State private var videoURL: String = ""  
21     @FocusState private var isVideoURLFocused: Bool  
22     @State private var isGlobal: Bool?  
23     @FocusState private var isIsGlobalFocused: Bool  
24     @State private var description: String = ""  
25     @FocusState private var isDescriptionFocused: Bool  
26     @State private var typeName: String = ""  
27     @FocusState private var isTypeNameFocused: Bool  
28     @State private var subtypeName: String = ""  
29     @FocusState private var isSubtypeNameFocused: Bool  
30  
31     var body: some View {  
32  
33         NavigationStack {  
34             VStack {  
35                 nameForm  
36                     .padding()  
37                 imageURLForm  
38                     .padding()  
39                 videoURLForm  
40                     .padding()  
41                 isGlobalForm  
42                     .padding()  
43                 descriptionForm  
44                     .padding()  
45                 typeNameForm  
46                     .padding()  
47                 subtypeNameForm  
48                     .padding()  
49  
50                 Button("Create") {  
51                     Task {  
52                         await vm.createExercise(name: name, imageURL: imageURL, videoURL: videoURL, isGlobal: isGlobal ?? false, description:  
53                             description, typeName: typeName, subtypeName: subtypeName)  
54                         await vm.getAllExercises()  
55                         isPresented = false  
56                     }  
57                     .buttonBorderShape(.roundedRectangle)  
58                     .tint(Color("AccentColor"))  
59                     .alert(vm.alertMessage, isPresented: $vm.showAlert) {  
60                         Button("OK", role: .cancel) {}  
61                     }  
62                     .padding(.horizontal, 24)  
63                 }  
64                 .navigationTitle("New Exercise")  
65                 .toolbar {  
66                     ToolbarItem(placement: .topBarTrailing) {  
67                         Button {  
68                             isPresented = false  
69                         } label: {  
70                             Image(systemName: "xmark")  
71                             .imageScale(.large)  
72                         }  
73                     }  
74                 }  
75             }  
76         }  
77     }  
78 }
```

Figura 77 vista de la creación de un ejercicio.

BeSport24Training

```
79  private var nameForm: some View {
80      HStack {
81          Image(systemName: "text.bubble")
82              .foregroundStyle(isNameFocused ? Color("FocusColor") : Color("GrayColor"))
83          TextField("", text: $name, prompt: Text("Name of the exercise").foregroundColor(Color("GrayColor")))
84              .foregroundStyle(Color("BlackColor"))
85              .focused($isNameFocused)
86              .autocorrectionDisabled()
87              .keyboardType(.emailAddress)
88              .textContentType(.emailAddress)
89              .autocapitalization(.none)
90      }
91      .padding(.vertical, 12)
92      .padding(.horizontal)
93      .overlay(
94          RoundedRectangle(cornerRadius: 8)
95              .stroke(isNameFocused ? Color("FocusColor") : Color("GrayColor"), lineWidth: 1)
96      )
97      .background(Color("WhiteColor"))
98      .cornerRadius(8)
99  }
100
101 private var imageURLForm: some View {
102     HStack {
103         Image(systemName: "photo")
104             .foregroundStyle(isImageURLFocused ? Color("FocusColor") : Color("GrayColor"))
105         TextField("", text: $imageURL, prompt: Text("Image URL").foregroundColor(Color("GrayColor")))
106             .foregroundStyle(Color("BlackColor"))
107             .focused($isImageURLFocused)
108             .autocorrectionDisabled()
109             .keyboardType(.emailAddress)
110             .textContentType(.emailAddress)
111             .autocapitalization(.none)
112     }
113     .padding(.vertical, 12)
114     .padding(.horizontal)
115     .overlay(
116         RoundedRectangle(cornerRadius: 8)
117             .stroke(isImageURLFocused ? Color("FocusColor") : Color("GrayColor"), lineWidth: 1)
118     )
119     .background(Color("WhiteColor"))
120     .cornerRadius(8)
121 }
122
123 private var videoURLForm: some View {
124     HStack {
125         Image(systemName: "person.crop.square.badge.video")
126             .foregroundStyle(isVideoURLFocused ? Color("FocusColor") : Color("GrayColor"))
127         TextField("", text: $videoURL, prompt: Text("Video URL").foregroundColor(Color("GrayColor")))
128             .foregroundStyle(Color("BlackColor"))
129             .focused($isVideoURLFocused)
130             .autocorrectionDisabled()
131             .keyboardType(.emailAddress)
132             .textContentType(.emailAddress)
133             .autocapitalization(.none)
134     }
135     .padding(.vertical, 12)
136     .padding(.horizontal)
137     .overlay(
138         RoundedRectangle(cornerRadius: 8)
139             .stroke(isVideoURLFocused ? Color("FocusColor") : Color("GrayColor"), lineWidth: 1)
140     )
141     .background(Color("WhiteColor"))
142     .cornerRadius(8)
143 }
144
145 private var isGlobalForm: some View {
146     VStack {
147         Text("Is a global exercise?")
148             .foregroundStyle(isIsGlobalFocused ? Color("FocusColor") : Color("GrayColor"))
149         HStack {
150             Image(systemName: "globe")
151                 .foregroundStyle(isIsGlobalFocused ? Color("FocusColor") : Color("GrayColor"))
152             Picker("Is a global exercise?", selection: Binding($isGlobal, replacingNilWith: false)) {
153                 Text("True").tag(true)
154                 Text("False").tag(false)
155             }
156             .pickerStyle(.segmented)
157             .foregroundColor(Color("BlackColor"))
158         }
159         .padding(.vertical, 12)
160         .padding(.horizontal)
161         .overlay(
162             RoundedRectangle(cornerRadius: 8)
163                 .stroke(isNameFocused ? Color("FocusColor") : Color("GrayColor"), lineWidth: 1)
164         )
165         .cornerRadius(8)
166     }
167 }
```

Figura 77 vista de la creación de un ejercicio (continuación).

BeSport24Training

```
169 private var descriptionForm: some View {
170     HStack {
171         Image(systemName: "text.page")
172             .foregroundStyle(isDescriptionFocused ? Color("FocusColor") : Color("GrayColor"))
173         Textfield("", text: $description, prompt: Text("Description").foregroundColor(Color("GrayColor")))
174             .foregroundStyle(Color("BlackColor"))
175             .focused($isDescriptionFocused)
176             .autocorrectionDisabled()
177             .keyboardType(.emailAddress)
178             .textContentType(.emailAddress)
179             .autocapitalization(.none)
180     }
181     .padding(.vertical, 12)
182     .padding(.horizontal)
183     .overlay(
184         RoundedRectangle(cornerRadius: 8)
185             .stroke(isDescriptionFocused ? Color("FocusColor") : Color("GrayColor"), lineWidth: 1)
186     )
187     .background(Color("WhiteColor"))
188     .cornerRadius(8)
189 }
190
191 private var typeNameForm: some View {
192     VStack {
193         Text("What type of exercise is it?")
194             .foregroundStyle(istypeNameFocused ? Color("FocusColor") : Color("GrayColor"))
195         HStack {
196             Image(systemName: "figure.skiing.downhill")
197                 .foregroundStyle(istypeNameFocused ? Color("FocusColor") : Color("GrayColor"))
198             Picker("Exercise type?", selection: $typeName) {
199                 Text("Resistance").tag("Resistance")
200                     .background(.blue)
201                 Text("Cardio").tag("Cardio")
202                 Text("Mobility").tag("Mobility")
203             }
204             .pickerStyle(.segmented)
205         }
206         .padding(.vertical, 12)
207         .padding(.horizontal)
208         .overlay(
209             RoundedRectangle(cornerRadius: 8)
210                 .stroke(isSubtypeNameFocused ? Color("FocusColor") : Color("GrayColor"), lineWidth: 1)
211         )
212         .cornerRadius(8)
213     }
214 }
215 private var subtypeNameForm: some View {
216     VStack {
217         Text("Main musculature involved")
218         HStack {
219             Image(systemName: "figure")
220                 .foregroundStyle(isSubtypeNameFocused ? Color("FocusColor") : Color("GrayColor"))
221             Picker("Subtype", selection: $subtypeName) {
222                 ForEach(exerciseSubtypeOptions) { option in
223                     Text(option.name).tag(option.name)
224                 }
225             }
226             .pickerStyle(.navigationLink)
227             .frame(maxWidth: .infinity, alignment: .leading)
228             .padding(.leading, 4)
229         }
230         .padding(.vertical, 12)
231         .padding(.horizontal)
232         .overlay(
233             RoundedRectangle(cornerRadius: 8)
234                 .stroke(isSubtypeNameFocused ? Color("FocusColor") : Color("GrayColor"), lineWidth: 1)
235         )
236         .cornerRadius(8)
237     }
238 }
239
240 struct ExerciseSubtypeOption: Identifiable {
241     let id = UUID()
242     let name: String
243 }
244
245 let exerciseSubtypeOptions: [ExerciseSubtypeOption] = [
246     .init(name: "Chest"),
247     .init(name: "Back"),
248     .init(name: "Shoulders"),
249     .init(name: "HIIT"),
250     .init(name: "SIT"),
251     .init(name: "LISS"),
252     .init(name: "Static Stretching"),
253     .init(name: "Dynamic Stretching"),
254     .init(name: "Ballistic Stretching"),
255     .init(name: "Passive Stretching"),
256     .init(name: "Leg"),
257     .init(name: "Core"),
258     .init(name: "Biceps"),
259     .init(name: "Glutes"),
260     .init(name: "Calves"),
261     .init(name: "Quadriceps"),
262     .init(name: "Hamstrings"),
263     .init(name: "Triceps")
264 ]
265
266 }
267
268 #Preview {
269     NewExerciseView(isPresented: .constant(true))
270 }
271
272 extension Binding where Value == Bool? {
273     init(_ source: Binding<Bool?>, replacingNilWith defaultValue: Bool) {
274         self.init(
275             get: { source.wrappedValue ?? defaultValue },
276             set: { newValue in source.wrappedValue = newValue }
277         )
278     }
279 }
280 }
```

Figura 77 vista de la creación de un ejercicio (continuación).

De nuevo, gracias al uso de `NavigationStack` en SwiftUI, la creación de formularios estructurados se ha simplificado notablemente. En esta vista, el objetivo es permitir al usuario introducir los datos necesarios para registrar un nuevo ejercicio, como podemos intuir.

Antes del `body`, se definen múltiples variables que permiten recoger la información introducida por el usuario. Estas variables están decoradas con `@State` para que SwiftUI mantenga una conexión bidireccional entre los campos del formulario y el estado de la vista, tal y como hemos visto anteriormente. A ello se une `@FocusState` para gestionar el foco de los campos, lo que permite mejorar la usabilidad al controlar qué campo está activo o resaltado en cada momento de manera nativa.

Por otro lado, la variable `@Binding var isPresented` se utiliza para que la vista pueda ser cerrada desde dentro, al establecerla como `false`. Esta es la forma que hemos comentado previamente para cerrar esta vista y volver al listado de ejercicios.

Dentro del `NavigationStack`, se encuentran todos los formularios divididos por secciones mediante propiedades privadas (`nameForm`, `imageURLForm`, etc.), lo que ayuda a mantener el código organizado. Cada formulario consiste, en general, un `HStack` que contiene un ícono y un campo de texto o selector, con un diseño uniforme, asegurando así una experiencia visual coherente con el resto de la app. El hecho de que sean propiedades privadas permite que se describan en otros lugares de la vista, dejando el `body` limpio de cara a poder mantener y leer el código más fácilmente.

En cuanto a dichas variables, comentaremos solamente tres de ellas, de cara a resaltar los aspectos más importantes:

nameForm

Es la encargada de renderizar el campo de texto donde el usuario debe introducir el nombre del nuevo ejercicio.

El formulario está compuesto por un `HStack` que incluye un ícono representativo y un `TextField`. El campo de texto está enlazado bidireccionalmente a la variable de estado `@State private var name`, permitiendo que cualquier cambio hecho por el usuario se almacene automáticamente en la propiedad de cara a cuando se pulse en

Crear para que el ViewModel reciba el valor más actualizado en su método `createExercise()`.

Para mejorar la experiencia de usuario, el campo implementa `@FocusState`, una propiedad introducida en versiones recientes de SwiftUI, que permite detectar cuándo el campo está activo. En función de ello, se modifica dinámicamente el color del borde del formulario y del ícono, utilizando los colores definidos en el `ColorSet` al principio de la creación de la aplicación, lo cual aporta una retroalimentación visual clara e inmediata al usuario.

Además, se aplican estilos adicionales como bordes redondeados, relleno (padding) y desactivación de la autocorrección y capitalización automática, ya que se espera un valor exacto y sensible a mayúsculas y minúsculas.

isGlobalForm

Esta propiedad representa un componente clave para decidir si el ejercicio será global (que involucra a más de una articulación) con el valor `true` o analítico (que involucra a solamente una articulación de nuestro organismo) con el valor `false`. Esta elección se realiza mediante un *Picker* con estilo `.segmented`, que permite elegir de forma rápida entre las opciones dos opciones.

La variable que almacena este valor es `@State private var isGlobal: Bool?`, lo que implica que inicialmente puede no tener valor. No obstante, SwiftUI no permite trabajar directamente con opcionales en *Picker*, por lo que he tenido que crear una extensión de *Binding* personalizada que convierte automáticamente `nil` en un valor por defecto (`false`) y permite escribir y leer correctamente en la variable original.

El formulario mantiene una consistencia visual con el resto de formularios: bordes redondeados, cambio dinámico de color en función del foco y fondo blanco personalizado.

typeNameForm

Esta propiedad se utiliza para seleccionar el tipo general de ejercicio, una clasificación que agrupa los ejercicios según su objetivo físico principal. Las tres opciones

disponibles son: Resistance (fuerza), Cardio (resistencia) y Mobility (movilidad). Esta elección se guarda en la variable `@State private var typeName`.

El selector se construye mediante un *Picker* con estilo `.segmented`, lo que permite una interacción directa y visualmente clara para el usuario, al igual que sucede con el de si es un ejercicio global. Cada opción tiene su etiqueta correspondiente y un valor asociado mediante `.tag`, que es el que se emplea en el Interactor a la hora de mapear de *Exercise* a *ExerciseDTO*.

Como en los formularios anteriores, se usa un *HStack* con un ícono (en este caso "figure.skiing.downhill") y se aplican estilos de diseño para mantener la coherencia visual.

La del subtipo de ejercicio es similar, aunque usa un estilo de *Picker* para que sea más intuitivo para el usuario.

Botones de crear y cancelar

Ya por último tenemos dos botones, el de crear el ejercicio y el de la X para cancelar este proceso.

El botón de crear, ubicado en la parte inferior del formulario, al pulsarlo ejecuta una tarea asíncrona mediante *Task*. Dentro de ella se llama al método `createExercise()` del *ViewModel*, se recargan los datos con `getAllExercises()` y se cierra la vista. Esto permite que el nuevo ejercicio aparezca directamente en la lista tras la creación, sin necesidad de reiniciar la aplicación.

El botón de cerrar, implementado como un ícono "xmark" de SF Fonts en la barra de navegación (`.toolbar`), permite salir sin guardar. Asimismo, como al cerrar la vista esta se destruye, al volver a tratar de crear un ejercicio, tendremos los campos en blanco para poder escribir de nuevo aquello que necesitemos para tener nuestro nuevo ejercicio satisfactoriamente.

A continuación, vemos cómo sería dicha vista:

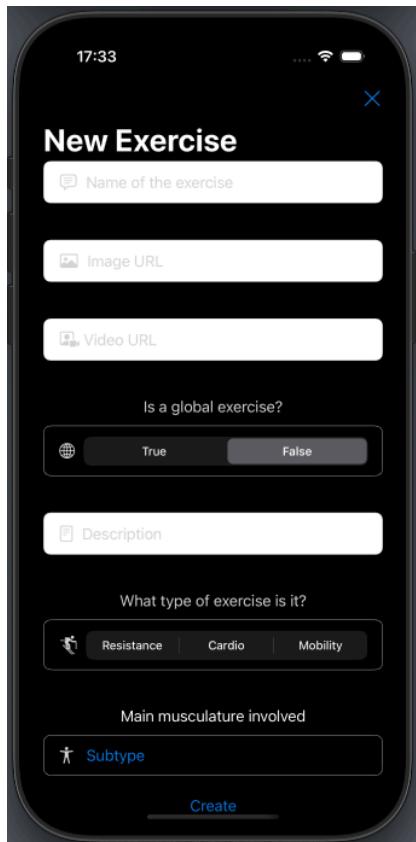


Figura 78. Vista de la creación de un ejercicio (en modo oscuro).

Manual de usuario

A continuación, se muestran unas guías maestras de cómo utilizar la aplicación para extraerle el máximo potencial posible.

Sección 1: Login

En la pantalla de login, tenemos la opción de registrarnos si no tenemos cuenta o de iniciar sesión si ya la tenemos. En ambas pantallas, tenemos validaciones de que se rellenen el campo del correo electrónico y de la contraseña.



Figura 79. Pantalla de inicio de sesión.

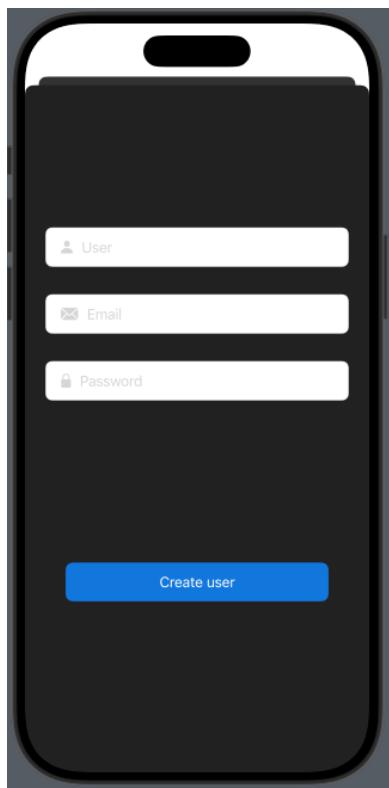


Figura 80. Pantalla de registro.

Sección 2: Perfil

Una vez dentro de la aplicación, pulsando en la pestaña de ajustes, podremos ir a las opciones de configuración de nuestro perfil.

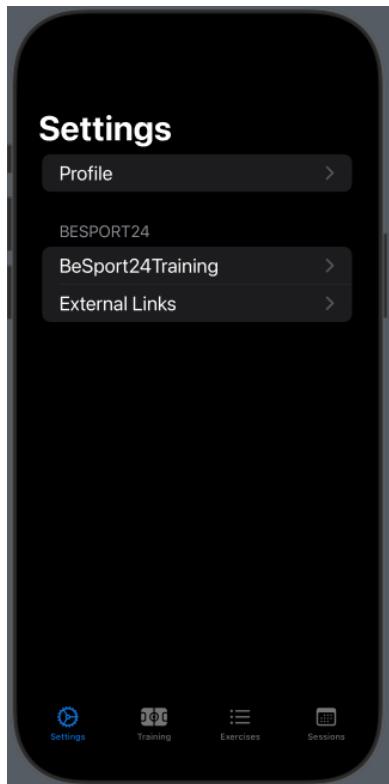


Figura 81. Pestaña de ajustes.

En las opciones de BeSport24Training podremos ver datos de la aplicación, mientras que en los enlaces externos, las redes sociales de BeSport24 (los cuales se abren en el navegador predeterminado al clicar sobre ellos o en su aplicación si la tenemos instalada):

BeSport24Training

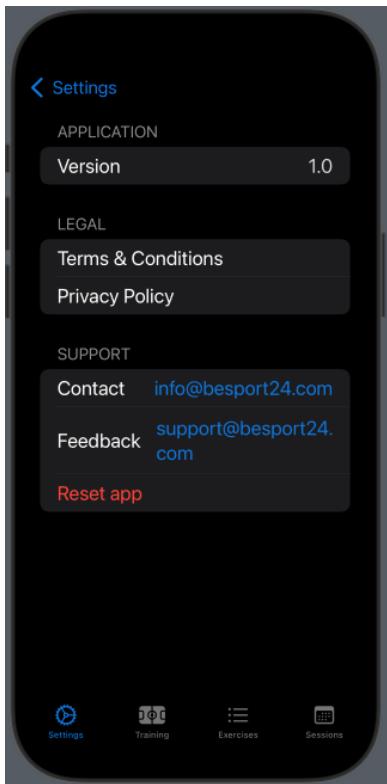


Figura 82. Datos de la aplicación.

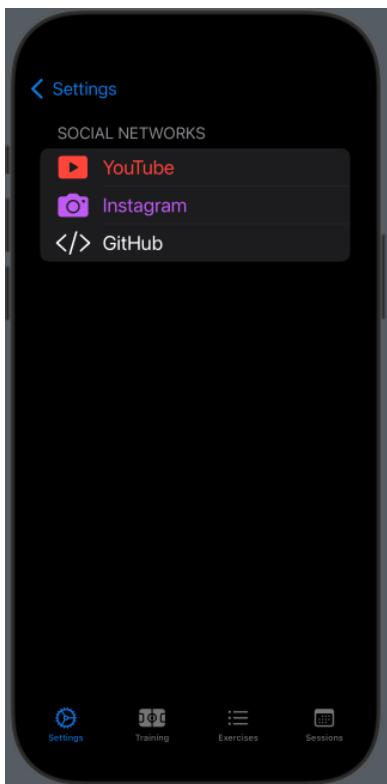


Figura 83. Enlaces externos.

BeSport24Training

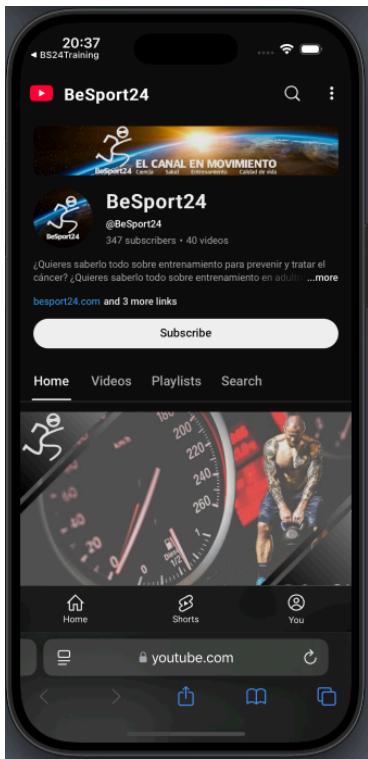


Figura 84. Navegador abierto tras pulsar uno de los enlaces.

La opción restante es la de ir a nuestro perfil, donde podremos visualizar nuestros datos, con las opciones de editar o de borrar el perfil.



Figura 85. Vista del perfil.

Sección 3: Ejercicios

Pulsando sobre la pestaña de ejercicios accederemos a la lista de ejercicios existentes en la base de datos de la aplicación:

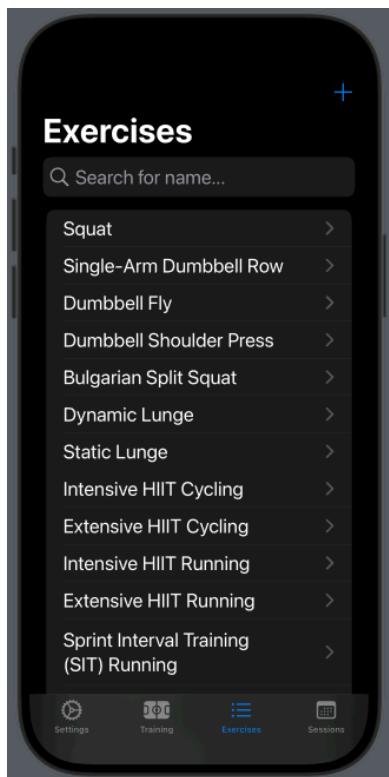


Figura 86. Listado de ejercicios.

En la parte superior tenemos un buscador que nos permite filtrar por el nombre del ejercicio:

BeSport24Training



Figura 87. Listado de ejercicios filtrado.

Si pulsamos sobre el botón “+”, podremos crear un nuevo ejercicio rellenando las distintas opciones:

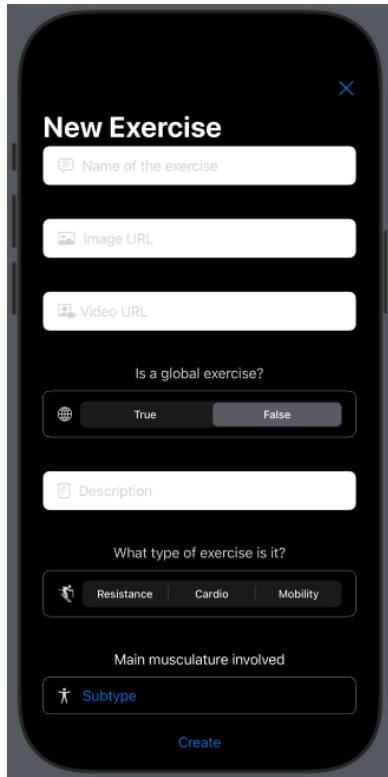


Figura 88. Creación de un nuevo ejercicio.

Si pulsamos sobre un ejercicio existente, accederemos a los detalles del mismo:

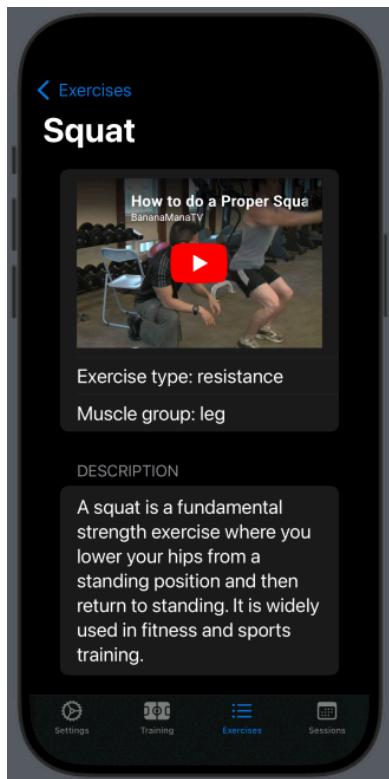


Figura 89. Detalles de un ejercicio.

Pulsando sobre el vídeo, podemos ver la explicación del propio ejercicio en el reproductor nativo de nuestro dispositivo:

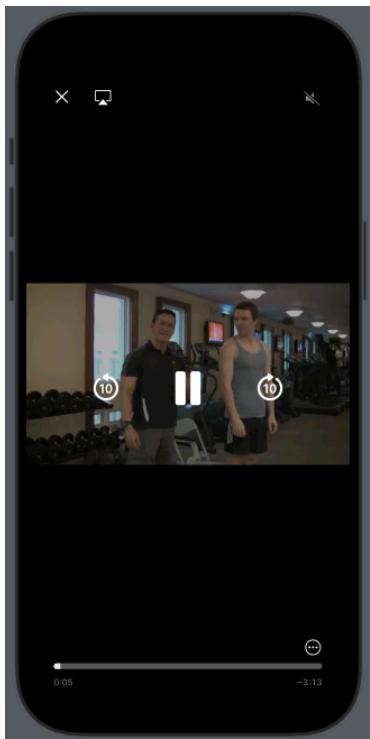


Figura 90. Reproducción del vídeo del ejercicio.

Yendo al fondo del detalle, tenemos las opciones de editar y de borrar ejercicio. En la de editar, se abrirá una vista en la cual editar los valores del ejercicio:

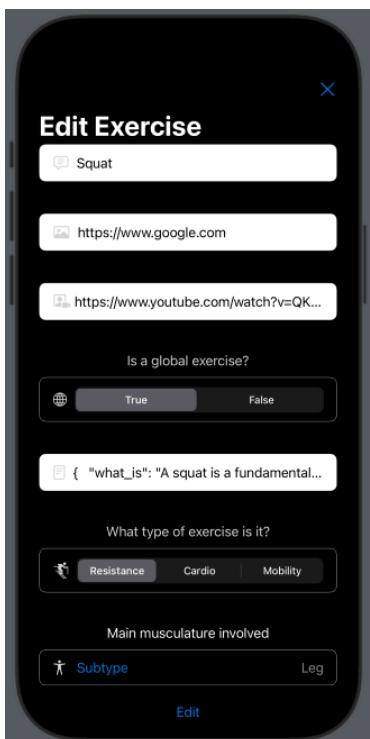


Figura 91. Edición de ejercicio.

Mientras que la opción de borrar se encargará de eliminar dicho ejercicio de la base de datos.

Sección 4: Sesiones

En el apartado de sesiones, los métodos son los mismos que en los ejercicios, pudiendo ver el listado completo, un ejercicio en concreto, crear un ejercicio y editar o borrar uno existente. Al pulsar en dicho apartado, se mostrará automáticamente el listado de sesiones que tenemos creadas:

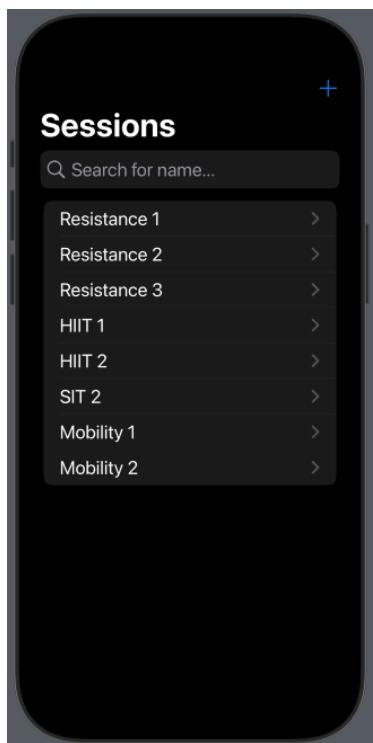


Figura 92. Listado de sesiones.

En el botón “+”, podremos crear una nueva sesión:

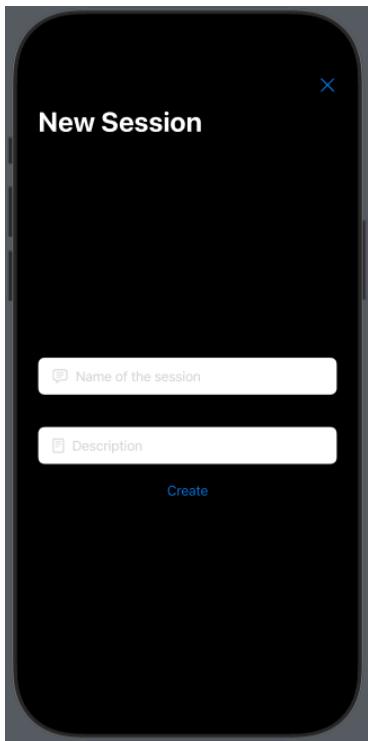


Figura 93. Creación de una sesión.

Si pulsamos en una de las sesiones creadas, se muestran los detalles propios de la sesión y, además, el listado de ejercicios que conforman dicha sesión, de cara a tener claro todo aquello que hay que realizar en la misma:

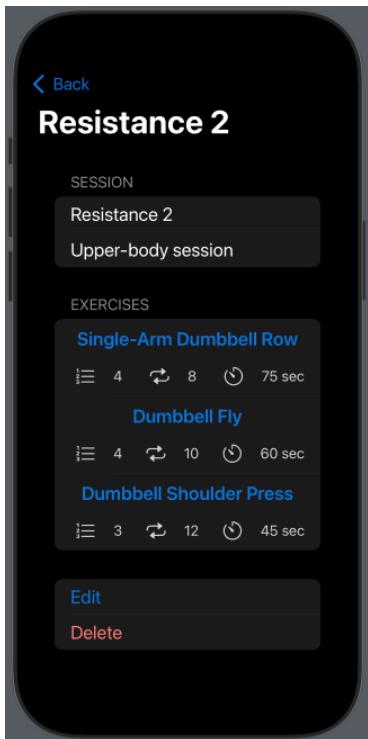


Figura 94. Detalles de una sesión.

Al ver una sesión, si no sabes cómo se ejecuta un ejercicio, puedes ir a la pestaña de ejercicios a buscarlo y, al volver a la de la sesión, permanecerá en el punto en el que estaba previamente.

Aquí, podemos editar la sesión, abriendose un menú similar al de edición de ejercicios.



Figura 95. Edición de una sesión.

Por el contrario, si pulsamos en el botón de borrar, eliminaremos dicha sesión.

Sección 5: Entrenamientos

Tanto al iniciar la aplicación, como si pulsamos en el ícono de entrenamientos, podremos ver los distintos entrenamientos que hemos realizado:

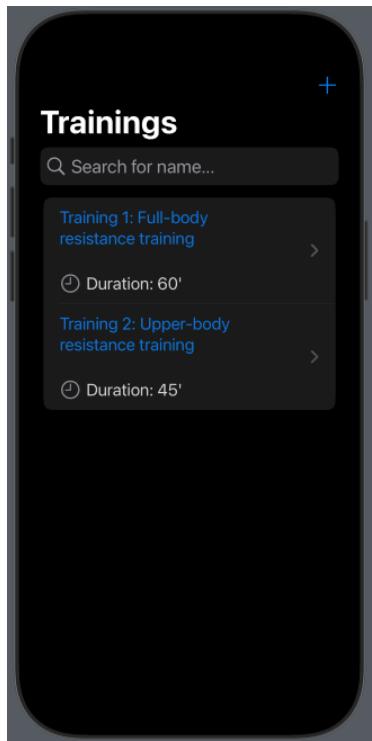


Figura 96. Listado de entrenamientos.

Si pulsamos en el botón “+”, podremos crear un nuevo entrenamiento.

Si deseamos ver los detalles de cómo ha sido una de nuestras sesiones de entrenamiento, pulsaremos sobre ella:

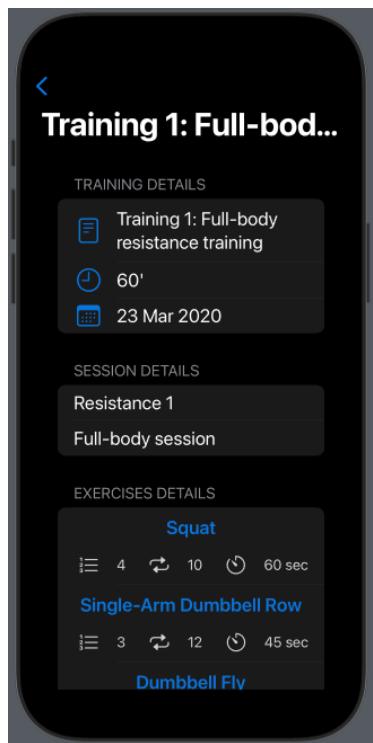


Figura 97. Detalles de un entrenamiento.

Al igual que con las sesiones, al ver un entrenamiento, si quieras asegurarte de cómo es la sesión que quieras ejecutar, puedes ir a la pestaña de sesiones a buscarla y, al volver a la del entrenamiento, permanecerá en el punto en el que estaba previamente.

Finalmente, en el botón de editar, como en los casos previos, podremos editar en este caso el entrenamiento en el que estamos, mientras que si pulsamos en eliminar, borraremos dicho registro.

Propuestas de mejora

La lista de futuras ampliaciones del proyecto es la siguiente:

- Sesiones de entrenamiento privadas y públicas.
- Equipamiento.
- Historial de progreso.
- Programas semanales.
- Multi idioma.
- Entrenador-deportista.
- Foco en privacidad.

A continuación, se muestra cada apartado explicado detalladamente para poder seguir más fácilmente el hilo de este apartado del trabajo:

- Sesiones de entrenamiento privadas y públicas.

Actualmente, las sesiones creadas en la aplicación son únicamente visibles por el propio usuario. Como mejora, se propone añadir una opción para configurar la visibilidad de cada sesión.

De esta manera, los usuarios podrán decidir si desean mantener sus entrenamientos en privado o compartirlos con otros miembros de la comunidad. Esta funcionalidad fomentaría tanto la privacidad como la interacción entre usuarios, permitiendo descubrir nuevas rutinas o inspirarse en los entrenamientos de otros.

Además, también permitiría que la propia aplicación contara con sesiones de entrenamiento con validez científica de cara a los usuarios que no cuenten con los conocimientos para elaborar sus propias sesiones de trabajo.

- Equipamiento.

En este apartado, la idea sería introducir en la base de datos una relación N-N en la cual se estipula el material que se emplea en cada ejercicio. Así pues, cada usuario o entrenador de la aplicación podría seleccionar los ejercicios que desea hacer en base al material del que dispone, ya sea entrenando en su casa, al aire libre o en un gimnasio.

- Historial de progreso.

La idea de este apartado es contar con gráficas en las cuales visualizar el progreso en los distintos ejercicios. Para que se entienda mejor, pondré dos ejemplos:

En el press banca, llegar a los 100kg es un objetivo que tienen muchos hombres. Poder contar con un gráfico en el cual se ve el progreso del ejercicio puede ayudarnos a comprobar el progreso que vamos realizando, evaluar que el programa de entrenamiento es exitoso o a mantener la motivación, entre muchos otros análisis.

En el caso del entrenamiento cardiovascular, el parámetro del VO2max nos puede prestar claves importantes de cara a saber el estado de nuestra salud general, ya que es un gran medidor de salud cardiovascular. Poder ver cómo evoluciona puede ayudarnos también de cara a prepararnos carreras como la Media Maratón de Valencia, pudiendo ver si el tiempo en el que queremos correrla es factible.

- Programas semanales.

Actualmente podemos crear sesiones, pero no establecer un calendario en el cual estipular qué sesión debemos realizar cada día. Esto es de gran ayuda a la hora de la programación del entrenamiento, ya que podremos centralizar todo el proceso en una misma aplicación y no tener que depender de terceros.

- Multi idioma.

Otra opción interesante es la de admitir más idiomas. Este es un apartado que he tenido que tener en cuenta en la empresa, ya que los idiomas han aumentado considerablemente, por lo que conozco como poder llevarlo a cabo. No obstante, con el objetivo que tengo en mente, conllevaría la modificación de la base de datos.

- Entrenador-deportista

Este es el apartado más ambicioso, ya que permitiría la creación de distintos roles: el entrenador y el deportista. Esto permitiría que expertos en ejercicio contaran con una aplicación integral en la cual desarrollar los planes de entrenamiento y que los deportistas solo tuvieran que seguir el planning. Debido a la envergadura de este proyecto, es una idea remota que por ahora no planteo llevar a la realidad.

- Foco en privacidad

Finalmente, mi mayor deseo es que la aplicación corra en local sin tener que enviar o recibir datos del exterior. Esto permitirá que la aplicación pueda funcionar sin una conexión a internet o sin depender de tener activo un servidor. Además, le da un plus de privacidad ya que los datos estarían encriptados en el propio dispositivo. Asimismo, se habilitaría una opción para exportar los datos por si se cambia de dispositivo.

Debido a que prácticamente desde el primer día estoy metido en el desarrollo del proyecto de la empresa en la cual estoy cursando las prácticas, no me es posible realizar todas estas funciones por la falta de tiempo. No obstante, y debido a que es una aplicación que quiero desarrollar para poder llevar el registro de mis entrenamientos, planeo ir desarrollando estas funciones poco a poco.

Valoración personal

Tras finalizar el desarrollo de esta aplicación como parte del Trabajo de Fin de Grado, considero que esta experiencia ha supuesto un gran crecimiento como programador, ya que me ha permitido consolidar conocimientos y aplicarlos con nuevas tecnologías, facilitando mi incorporación en la dinámica de la empresa de prácticas.

Aprendizaje técnico

Desde un punto de vista técnico, uno de los mayores retos fue adentrarme en Swift y SwiftUI. Se trataba de un lenguaje, un framework, una arquitectura y una manera de plasmar los datos del *backend* de una manera totalmente diferente a lo he venido a estar acostumbrado estos dos años de formación. Al principio, esta forma de construir interfaces me resultó algo abstracta, especialmente en comparación con otros frameworks más imperativos al ser declarativo, pero una vez asimilado, la velocidad para construir las vistas y el flujo de datos fue mucho más rápida. Al mismo tiempo, el tener que *debuggear* en la empresa me ha permitido acelerar en gran medida la resolución de errores.

Asimismo, el uso de la arquitectura MVVM (Model-View-ViewModel) resultó clave para estructurar la lógica de la aplicación. Separar la vista del modelo me permitió mantener el código más ordenado, reutilizable y testeable, algo esencial ya que es una aplicación que quiero seguir ampliando este año de cara a tener un mejor portfolio. De hecho, aprender testing en Swift es uno de los objetivos que tengo.

Diseño de la experiencia de usuario

Uno de los aspectos más satisfactorios del desarrollo fue diseñar la experiencia de usuario (UX) desde cero en un entorno totalmente diferente como es el ecosistema de Apple. Aprendí a utilizar diferentes componentes como *NavigationStack*, *Picker*, *TextField*, *Toolbar*, o *FullScreenCover*, y a gestionar interacciones como `@FocusState` o `.refreshable`.

Me esforcé especialmente en mantener una coherencia estética en todos los formularios, con uso de colores definidos en un ColorSet, iconos representativos y

contenedores reutilizables con bordes redondeados. El resultado fue una interfaz moderna, clara y adaptada a los estándares actuales de iOS, lo cual considero uno de los mayores logros del proyecto; que al mismo tiempo es nativa de cara a que los futuros cambios de iOS 19 afecten lo menor posible a este proyecto.

Superación de dificultades

Como en todo desarrollo real, me encontré con diversos obstáculos que me obligaron a investigar, consultar documentación oficial y resolver errores de forma autónoma. Algunos errores eran triviales (como el uso incorrecto de un *@Binding* o una URL mal formada debido a los opcionales de iOS), mientras que otros requerían un análisis más profundo, como los errores de preview en Xcode o problemas al sincronizar datos entre vistas con diferentes instancias del ViewModel al emplear instancias distintas.

Aprendí a trabajar con lógica condicional en la validación de formularios, mostrar mensajes de alerta personalizados, y evitar duplicación de lógica gracias a la creación de componentes reutilizables. Todo ello me dio una perspectiva más realista sobre cómo se desarrolla software en entornos profesionales.

Aun así, la mayor dificultad que tuve que afrontar fue la del tiempo, ya que, por prácticas y entrenamientos, me es bastante escaso. No obstante, me ha ayudado a seguir afinando mi capacidad para organizarme y afrontar contratiempos.

Reflexión final

Este proyecto no ha sido simplemente una entrega académica: ha sido una simulación real del ciclo completo de desarrollo de una aplicación móvil, desde la fase de diseño y estructura de datos hasta la implementación de lógica, integración con *backend* y aprendizaje de lenguajes y frameworks completamente nuevos.

Me siento especialmente orgulloso de haber creado una aplicación funcional, con persistencia de datos, navegación clara, formularios validados, filtros dinámicos y una interfaz cuidada. Sé que aún hay mucho por aprender y mejorar, pero sin duda este trabajo ha supuesto un gran paso hacia mi madurez profesional como desarrollador.

Como conclusión, considero que este TFG ha cumplido con creces su propósito: no solo como demostración de mis competencias, sino también como impulso para continuar aprendiendo y evolucionando en el mundo del desarrollo de aplicaciones. SwiftUI me ha abierto un nuevo camino y me motiva a seguir explorando tecnologías modernas, patrones de arquitectura y mejores prácticas en el ámbito del desarrollo móvil y *full stack*. Asimismo, me sirve como base de cara a que este proyecto sea una referencia en mi portafolio como desarrollador *full stack* fusionando iOS y Java.

Puntos a destacar del proyecto

A lo largo del desarrollo de esta aplicación móvil para la gestión de entrenamientos, se han abordado numerosos aspectos técnicos y organizativos que merece la pena destacar, tanto por su complejidad como por su aporte en el proceso de aprendizaje y en la calidad final del proyecto.

Desde las fases iniciales del proyecto, se priorizó la organización y la planificación. Ello me ha permitido ser más eficiente en el momento de comenzar a actuar, teniendo las cosas claras desde el principio. El haber planteado con antelación los esquemas de la base de datos, los flujos de navegación y la estructura general de carpetas ha sido clave para evitar ambigüedades y mantener el control en las distintas fases del desarrollo.

Asimismo, este proyecto sigue una arquitectura por capas bien definida, separando claramente el modelo de dominio, la lógica de negocio y el acceso a datos. Además, se emplea un enfoque modular y escalable, lo que facilita el mantenimiento y la evolución futura del sistema; ya que la idea es seguir complementándolo con las propuestas de mejora ya mencionadas.

Otro aprendizaje fue el de profundizar en el funcionamiento de JPA, aspecto que me proporcionará gran valor en el ámbito profesional, ya que es el sistema empleado en la empresa para acceder a la base de datos.

El diseño de la base de datos relacional con MariaDB también fue un componente esencial. Planifiqué correctamente las tablas, claves foráneas y relaciones, lo que permitió una integración fluida con el backend y una respuesta coherente desde los endpoints.

En cuanto al *frontend*, destaco el uso de un lenguaje y un ecosistema que rara vez se puede ver en un ciclo de formación profesional, por lo que puede abrirme las puertas a un nicho de mercado altamente cotizado debido a la menor oferta de desarrolladores en este ámbito.

A nivel de integración, ha sido muy enriquecedor ver cómo los datos viajaban desde la base de datos relacional hasta una vista final en el dispositivo móvil, atravesando

las capas del *backend*, las llamadas a API y la aplicación móvil. Este flujo completo me ha dado una visión transversal del desarrollo de software que pocas veces se alcanza en un entorno educativo.

Aparte de los aspectos técnicos, valoro especialmente la mejora de habilidades transversales como la resolución de errores, la autonomía para investigar las problemáticas que iban surgiendo y la capacidad de estructurar el código de manera limpia, modular y fácilmente ampliable.

En definitiva, este proyecto ha sido una experiencia de desarrollo integral, en la que he actuado como analista, diseñador, programador y usuario real de la aplicación. Me ha permitido aplicar conocimientos previos, adquirir otros nuevos y trabajar con herramientas y lenguajes de actualidad que me acercan al entorno profesional real. Me siento satisfecho no solo con el resultado funcional, sino con todo el aprendizaje técnico y organizativo que ha implicado. Sin duda, esta aplicación representa un hito importante en mi trayectoria como futuro desarrollador.

One More Thing

Durante muchos años, Apple popularizó la expresión *One More Thing* como un recurso especial en sus presentaciones. Esta frase, empleada por Steve Jobs y retomada en ocasiones por su sucesor Tim Cook, servía para anunciar al final del evento una sorpresa, una funcionalidad inesperada o un producto disruptivo que nadie se esperaba, generando un momento memorable para los asistentes.

Inspirándome en esta tradición, he querido cerrar este proyecto con mi propio “One More Thing”: la aplicación no solo está disponible para iPhone, sino que también es plenamente funcional en iPad.

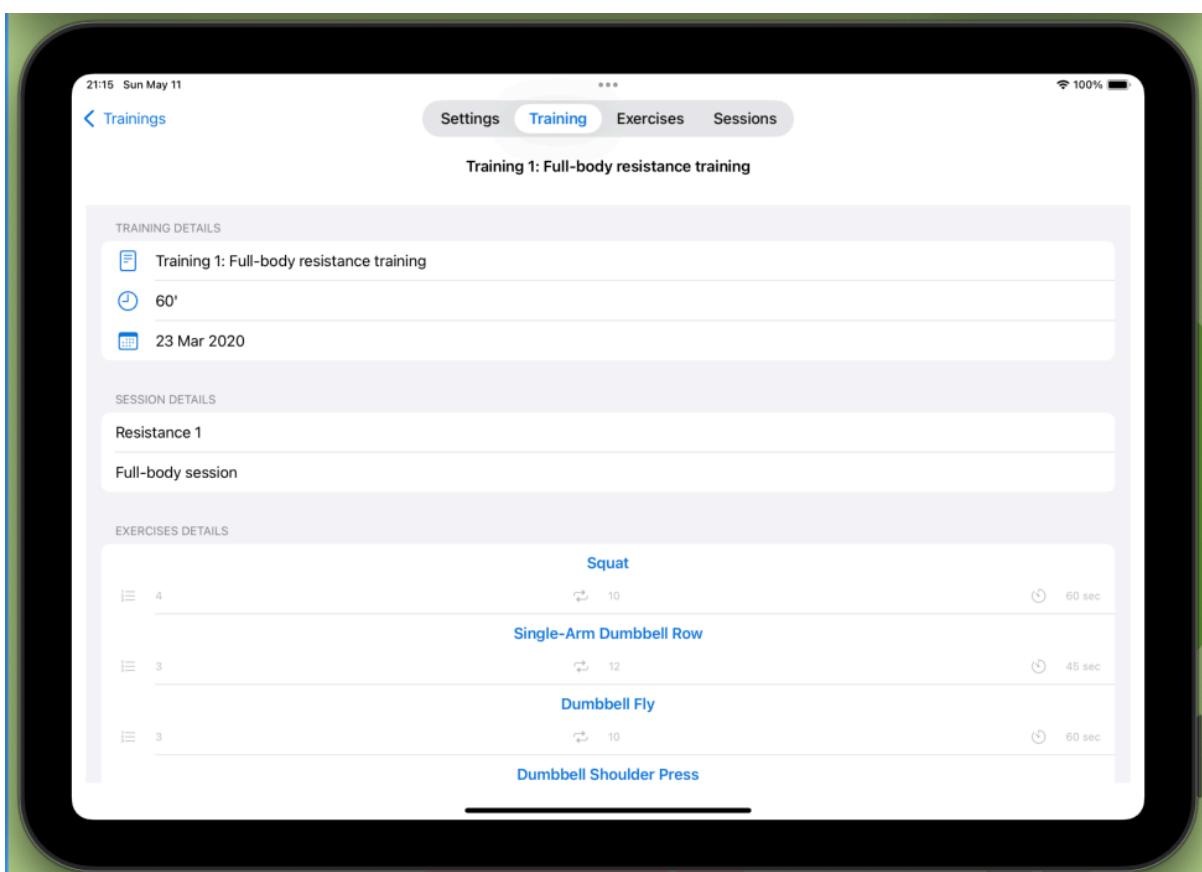


Figura 98. BeSport24Training corriendo en un iPad en modo horizontal.

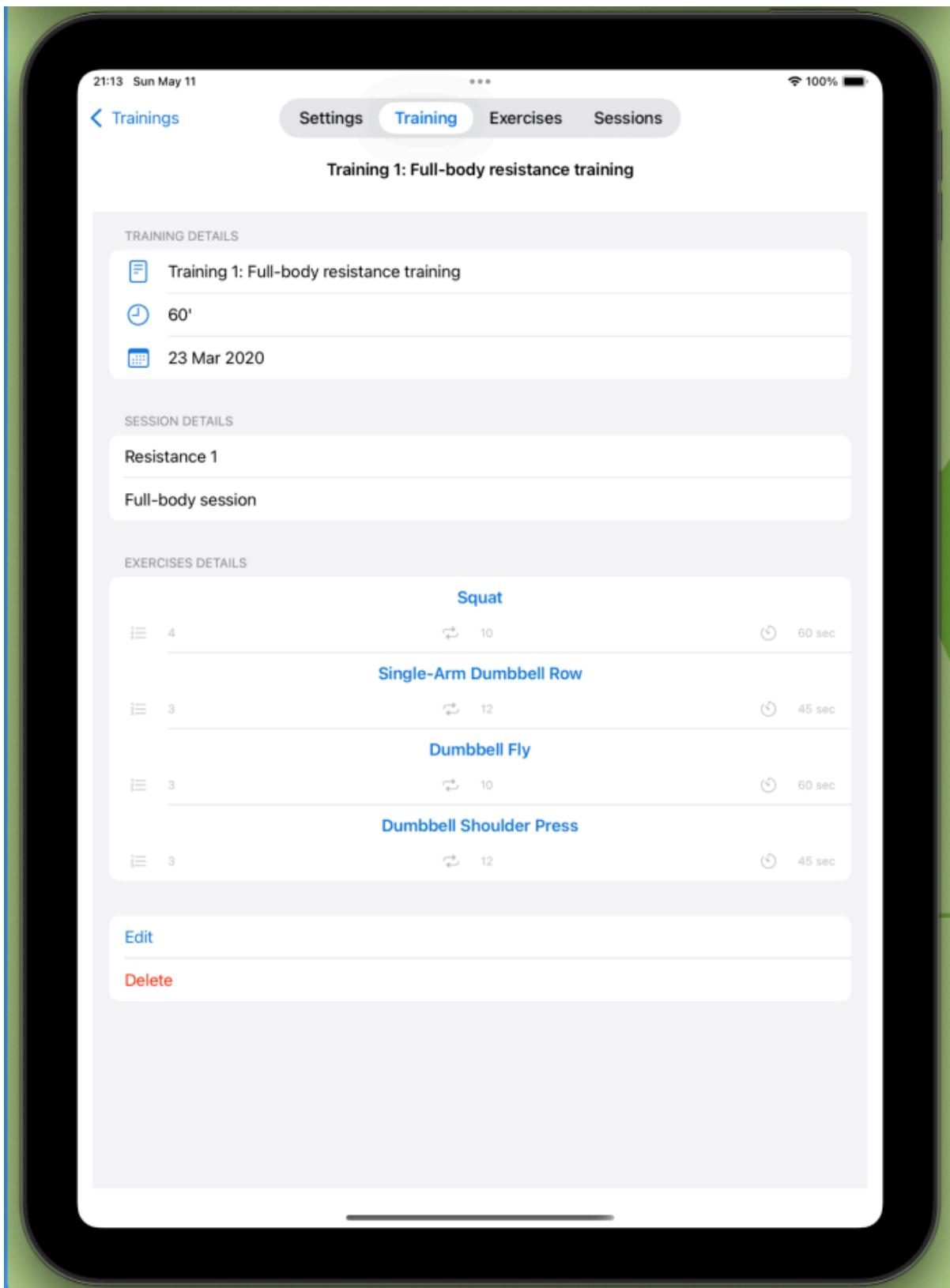


Figura 99. BeSport24Training corriendo en un iPad en modo vertical.

BeSport24Training

Además, contamos con un segundo One More Thing. BeSport24Training funciona en macOS:

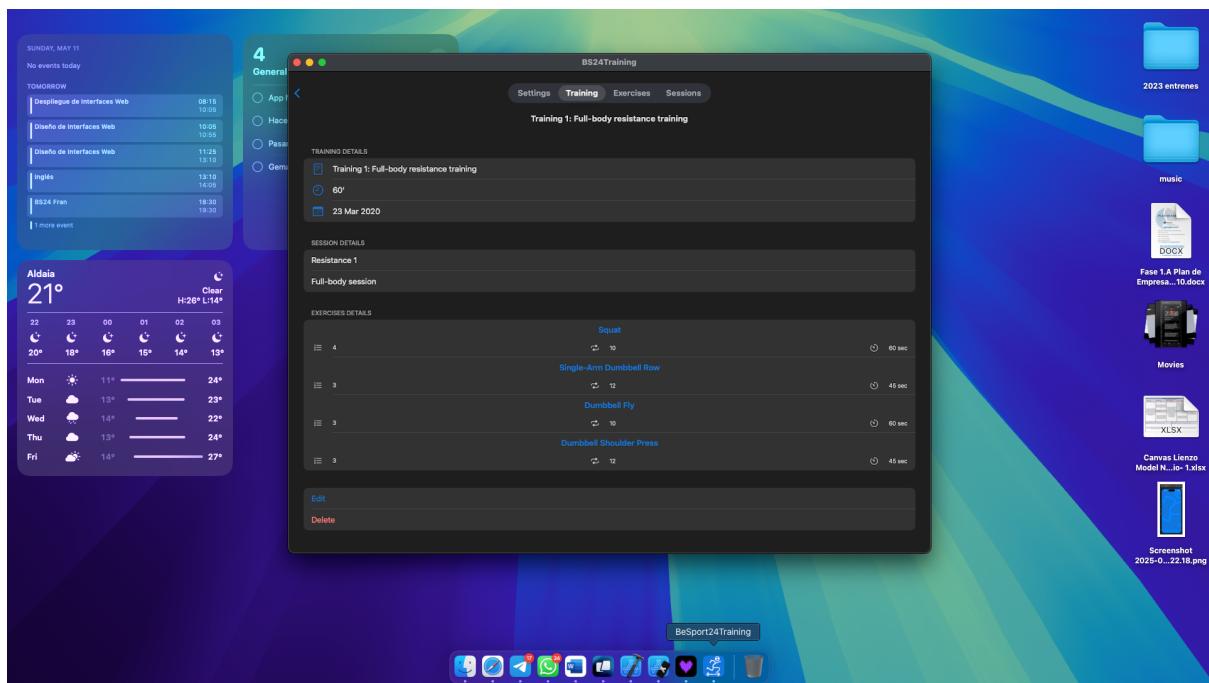


Figura 100. BeSport24 corriendo en un Mac.

Referencias

Swift:

- Swift Fundamentals: Mastering the Basics with Swift & iOS – Stephan Dowless:
<https://www.udemy.com/course/swift-fundamentals/?couponCode=LETSLEARNNOW>
- Aprendiendo Swift 5.5 – Julio César Fernández Muños (Apple Coding Academy):
<https://www.udemy.com/course/comenzando-con-swift/>

SwiftUI:

- Swift desde Cero – Brais Moure:
https://youtube.com/playlist?list=PLNdFk2_brsRetB7LiUfpnlclBe_1iOS4M&si=vCzHW54baML3w6Sz
- Curso de programación SwiftUI y Xcode – SwiftBeta:
https://youtube.com/watch?v=H0kihMIApn4&list=PLeTOFRUxkMcrJIZf4NnJQsL_ZNIq9JWQy
- SwiftUI Crypto App – Swiftful Thinking:
https://www.youtube.com/watch?v=TTYKL6CfbSs&list=PLvvDm4Vfkphbc3bg_y_LpLRQ9DDfFGcFu

Arquitecturas Swift/SwiftUI:

- Arquitecturas Swift – SwiftBeta:
https://www.youtube.com/watch?v=LHJHgpD5cFA&list=PLeTOFRUxkMcpY5fjTNVXWz_O1MAYYdVQA&index=7
- MVVM – Apple Coding Academy:
<https://acoding.academy/mvvm-y-la-arquitectura-del-framework-swiftui-para-el-desarrollo-de-aplicaciones-ios/>
- MVVM – Paradigma Digital:
<https://www.paradigmadigital.com/dev/arquitectura-mvvm-swiftui-combine/>
- MVVM – SwiftBeta:
https://www.youtube.com/watch?v=dPFrz0PxQTY&list=PLeTOFRUxkMcpY5fjTNVXWz_O1MAYYdVQA&index=1&pp=iAQB

BeSport24Training

- MVC – SwiftBeta:
https://www.youtube.com/watch?v=dPFrX0PxQTY&list=PLeTOFRUxkMcpY5fjTNVXWz_O1MAYYdVQA&index=2
- VIPER – SwiftBeta:
https://www.youtube.com/watch?v=0GjEFSVsulg&list=PLeTOFRUxkMcpY5fjTNVXWz_O1MAYYdVQA&index=4
- VIPER – SwiftBeta:
https://www.youtube.com/watch?v=ZRqvWGO0S3w&list=PLeTOFRUxkMcpY5fjTNVXWz_O1MAYYdVQA&index=5
- MVP – SwiftBeta:
https://www.youtube.com/watch?v=U5ozYLO4KVs&list=PLeTOFRUxkMcpY5fjTNVXWz_O1MAYYdVQA&index=7

Java:

- Apuntes César: <http://cesguiro.es/doku.php>
- Spring Initializr: <https://start.spring.io>
- CORS: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS>

MariaDB:

- Documentación oficial: <https://mariadb.org>