ULPGC

Escuela de Ingeniería Informática

MyTrainer

Desarrollo de aplicación móvil modular para la gestión eficiente entre clientes y entrenadores personales

Autor

David Iglesias Guerra

Grado en Ingeniería Informática - Tecnologías de la Información

Tutor

Dr. Francisco Alexis Quesada Arencibia

Ciencias de la Computación e Inteligencia Artificial

Junio de 2024 – Las Palmas de Gran Canaria

Resumen

Este proyecto consiste en el desarrollo de una aplicación móvil modular destinada a la gestión eficiente de entrenadores personales y sus clientes, MyTrainer.

La aplicación MyTrainer permite a los entrenadores organizar sus horarios de sesiones de entrenamiento y a los clientes reservar y pagar por dichas sesiones de manera flexible. Además del módulo principal de gestión de calendario.

Esta aplicación busca proporcionar una solución integral y de valor que facilite la interacción y gestión de entrenadores y clientes, mejorando la eficiencia y la experiencia de usuario en el ámbito del fitness.

Abstract

This project focuses on the development of a modular mobile application for the efficient management of personal trainers and their clients, MyTrainer.

The MyTrainer application allows trainers to organize their training session schedules and clients to conveniently book and pay for training sessions. In addition to the main calendar management module.

This application seeks to provide a comprehensive and valuable solution that facilitates the interaction and management of trainers and clients, improving efficiency and user experience in the fitness environment.

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1 Planteamiento

El mundo fitness está en tendencia y ahora más a través de las redes sociales. A partir de esto, el tutor de este trabajo plantea la idea del desarrollo de una herramienta que permita gestionar sesiones de entrenamiento y esto suponga una relación más personal e interactiva entre un entrenador personal y sus clientes.

Hoy en día, el mercado del fitness sigue estando en tendencia y creciendo constantemente, esto provoca que vayan apareciendo nuevos servicios y productos o figuras que permiten alcanzar los objetivos que cada persona considere para su salud y bienestar. Este crecimiento produce que haya una gran demanda de productos y servicios, donde una de ellas es la gestión de clientes por parte de un entrenador personal.

Después de un estudio inicial del estado del arte y varias sesiones de intercambio de ideas con el tutor, el autor del presente trabajo plantea el desarrollo de una aplicación móvil modular que esté diseñada para la gestión eficiente entre un entrenador personal y sus clientes.

La aplicación permite a los entrenadores añadir sus sesiones de manera única o periódica, creando un horario de entrenamiento, que servirá para que el cliente pueda ver las sesiones que tiene su entrenador y pagar por cada sesión, o pagar una subscripción que le permita un número de sesiones durante un tiempo, siendo el entrenador el que las define según sus necesidades.

Una de las claves más importantes en este contexto de entrenamiento personal, es conseguir más atención por parte de sus clientes, así como más fidelidad, al mismo tiempo, los clientes buscan y valoran lo flexible y fácil que puede ser el reservar sesiones, y que esto les permita tener un horario organizado y flexible.

El módulo principal de la aplicación es el calendario, desde aquí se permite a los entrenadores personales organizar su agenda, a su vez, el cliente podrá ver la disponibilidad del entrenador y reservar sesiones, esto incluye las opciones de pago,

permitiendo tanto a clientes como entrenador tener pagos mensuales, semestrales, o de cualquier otro tipo.

Un módulo de usuarios desde donde el entrenador pueda añadir a sus clientes que así se lo soliciten, con un código de invitación o un email, y, finalmente, un módulo de pagos, desde donde el entrenador puede crear sus productos y subscripciones según las necesidades con respecto a su horario.

1.2 Estado del arte

Una vez definido el contexto y la necesidad por la que surge esta aplicación se ha realizado un análisis de herramientas que ofrecen servicios similares a los mencionados, en concreto, hemos preferido analizar tantos servicios ofrecidos en la web/navegador como aplicaciones móviles.

Desde el punto de vista del cliente, obtiene valor la posibilidad de ver y gestionar un calendario, y reservar las sesiones necesarias, lo que reduce la necesidad de comunicarse con el entrenador, al conseguir esta gestión se genera una posibilidad que aporta valor a ambas partes: realizar pagos en sesiones individuales, o realizar packs, o suscripciones mensuales, anuales, etc.

1.2.1 Setmore

Figura 1. Logo de Setmore

Setmore [1] es una plataforma de software que permite gestionar y programar citas con el fin de ayudar a cualquier empresa o profesional organizar y administrar su horario de manera flexible.

Setmore se comercializa a través de una subscripción (gratis, pro, equipo), para cada una de estas subscripciones se tiene tanto un pago mensual como anual.

Para el registro del usuario es obligatorio un número de teléfono y ofrece la posibilidad de conectarse tanto a Google como a Microsoft.

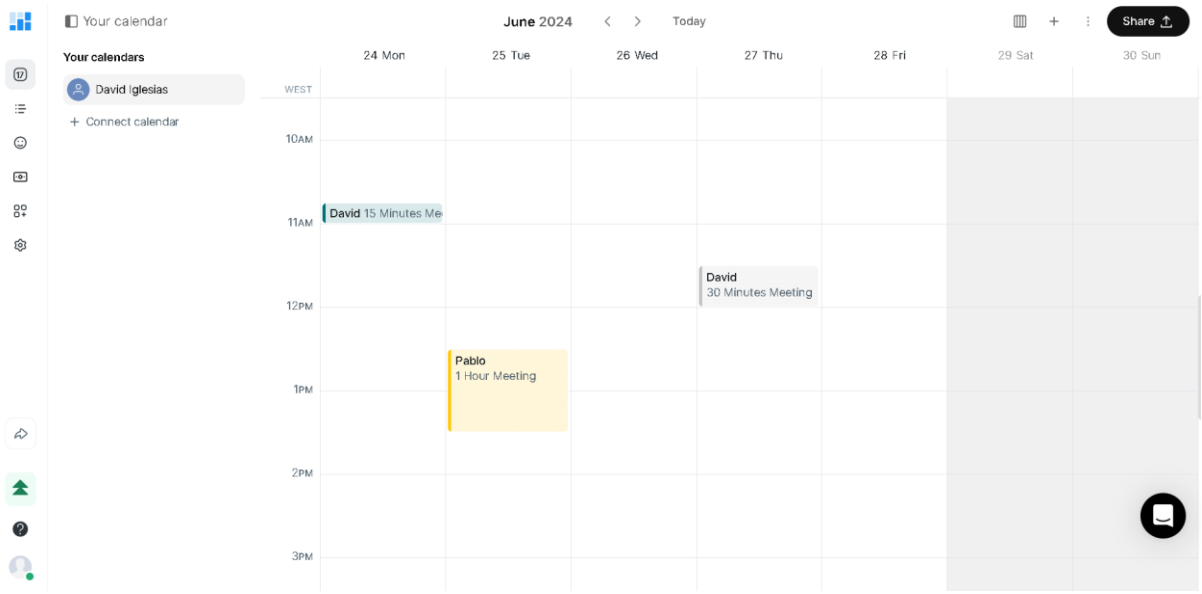


Figura 2. Dashboard de Setmore

En la figura anterior, podemos observar el *dashboard* de esta plataforma, que se carga desde el inicio de sesión con el calendario, donde se tiene una lista de todos los calendarios creados, dentro de cada uno de ellos, tenemos una vista global de los eventos que el organizador ha creado.

El organizador puede crear tanto sesiones *(figura 3)*, como servicios, como eventos, la aplicación distingue entre unas de ellas dependiendo del número de personas y el objetivo de estas.

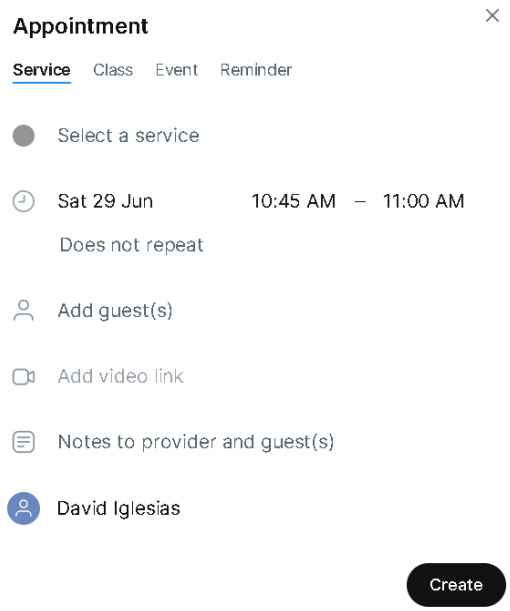


Figura 3. Creación de una cita en Setmore

Esta plataforma también disponible de una aplicación para iOS y Android.

Ofrece una sección donde gestionar los servicios propios de cada usuario, por ejemplo, en el caso de que sea un organizador, se tendrá una sección de clientes donde ver los datos de cada uno de ellos.

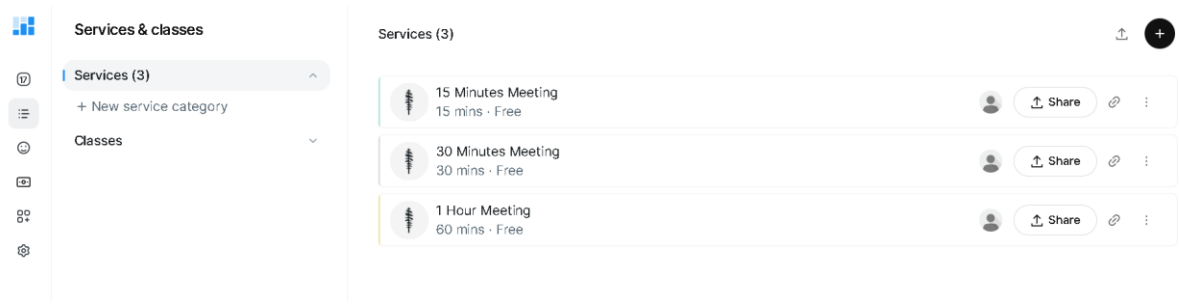


Figura 4. Listado de servicios en Setmore

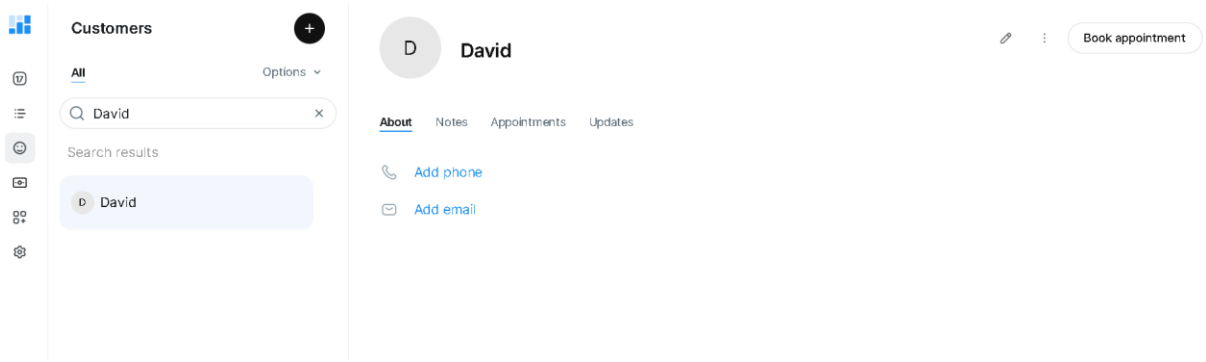


Figura 5. Listado de usuarios en Setmore

La plataforma además ofrece distintas integraciones con distintos servicios como puede ser Stripe, Zoom, Google Meet, Facebook, Instagram Bookings, etc….

Aspectos positivos	*Aspectos negativos*
Tiene un calendario de reservas intuitivo y fácil de usar.	Planes de suscripción con costo, aunque también tiene una versión gratuita muy limitada.
Ofrece la programación automatizada de citas y recordatorios.	Disponibilidad únicamente en inglés (no multi-idioma).
Acepta pagos en línea.	
Gestión de clientes y contactos.	
Generación de informes y análisis de actividades.	

Tabla 1. Aspectos positivos y negativos de Setmore

1.2.2 Harbiz

Figura 6. Logo de Harbiz

Harbiz [2] es una plataforma de software de entrenamiento dirigida a entrenadores personales que les ofrece la posibilidad de una gestión integral de su negocio y clientes. En la plataforma los profesionales pueden gestionar desde rutinas y planificaciones, hasta planes de comunicación, reservas y evolución física de los clientes.

La clave del producto se basa en el fitness y en todas las posibilidades que este sector ofrece. Harbiz intenta ofrecer un servicio/producto que aporte una solución exclusivamente dirigida hacia los profesionales de la salud/fitness, con la posibilidad de permitir a estos crear sus propios portales, etc.

Ofrece una interfaz bastante intuitiva y un diseño moderno, lo que la hace parecer menos antigua y más amigable en cuanto a la experiencia del usuario.

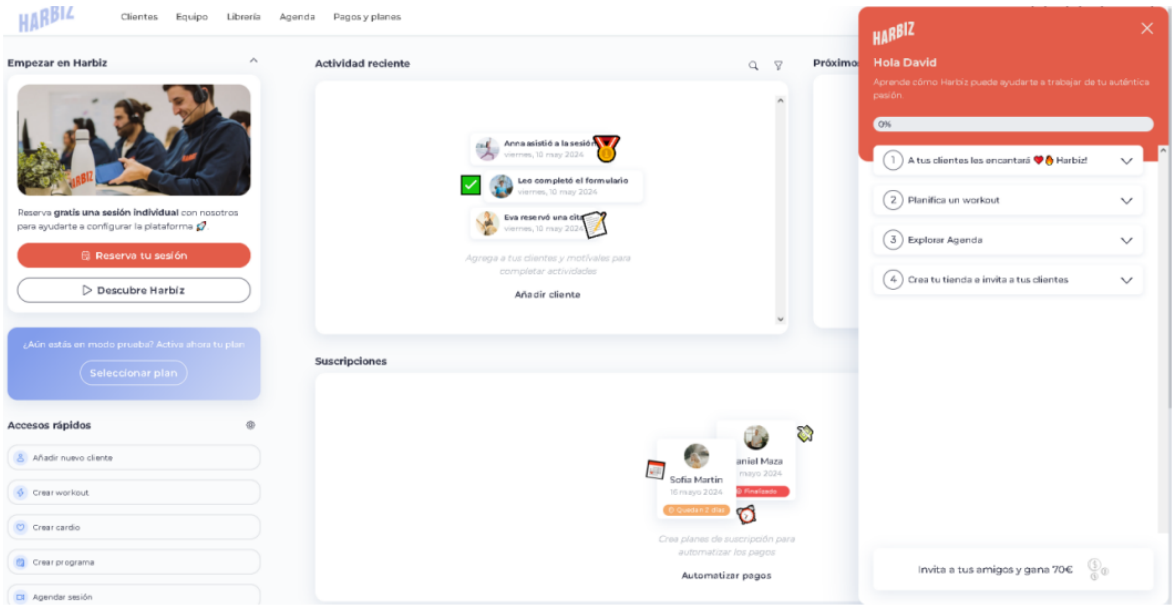


Figura 7. Dashboard de Harbiz

Otro punto importante de Harbiz es su enfoque en el mercado hispano, aunque existe la posibilidad de cambiar idioma dentro del producto, la mayoría de su público es hispanohablante.

Al igual que Setmore también tienen aplicación móvil para iOS y Android, pero en este caso, tienen una aplicación para el cliente (*Harbiz*) y otra para el entrenador personal *(Harbiz Manager)*.

Aspectos positivos	*Aspectos negativos*
Dispone de una posición líder en el sector de fitness y bienestar.	*Falta de información detallada sobre algunas funcionalidades clave.*
Ofrece soluciones digitales integrales para profesionales del sector.	*Es posible que se perciba "caro" para algunos usuarios.*
Facilita el uso y la adopción por parte de los usuarios.	*Preocupaciones sobre privacidad y seguridad de datos.*
Amplía la red de profesionales y empresas del sector.	*En algunos momentos, necesita de una mayor claridad en la navegación y estructura de la plataforma.*
Ofrece recursos educativos y de desarrollo profesional.	*Tienen limitaciones en la disponibilidad en ciertos mercados o regiones.*
Contribuye a la transformación digital del sector.	*Retraso en las respuestas a solicitudes de soporte.*

Tabla 2. Aspectos positivos y negativos de Harbiz

1.2.3 Wodbuster

Wodbuster [3] es una solución digital para centros deportivos con la que se pueden gestionar clases y tarifas dando la oportunidad a sus atletas de reservar fácilmente su entrenamiento y aumentar su motivación.

A su vez, ofrece un sistema de reservas flexible y fácil de usar, capaz de adaptarse a cada centro deportivo y sus atletas, manejan dos tipos de pagos, que serían las tarifas,

similar a una subscripción. y **bonos,** que correspondería a la reserva única de una sesión.

Tienen un módulo de gestión para los invitados, para ofrecer sesiones gratuitas de manera eficiente, ya sea gestionando las invitaciones a través de los administradores del centro o que los propios invitados se registren. Además, ofrece ventajas como el control de plazas para estos clientes.

También tiene un menú de informes detallados sobre ciertas métricas que consideran claves para desarrollar una gestión eficaz de un centro deportivo, algunas de estas serían el Ticket Medio, Evolución de los usuarios, Retención y Frecuencia de Pérdidas.

Similar a las dos plataformas mencionadas anteriormente, Wodbuster cuenta con una aplicación para iOS y Android.

Aspectos positivos	*Aspectos negativos*
Ofrece una gran biblioteca de ejercicios y rutinas de entrenamientos funcionales.	Los modelos de subscripción de pagos pueden ser demasiados costosos.
Tiene contenido educativo de calidad, como tutoriales y artículos.	Varios problemas en cuanto al rendimiento en la aplicación móvil.
Ofrece a los usuarios un seguimiento y análisis de su progreso de manera clara.	Problemas de actualizaciones entre centro deportivos.
Comunidad activa y motivadora.	

Tabla 3. Aspectos positivos y negativos de Wodbuster

1.2.4 Comparativa entre las distintas herramientas

Como hemos observado en los puntos anteriores, las distintas herramientas presentan características similares, pero en concreto, no cumplen con alguno de los requisitos que se plantean con el desarrollo del proyecto

A continuación, mostraremos una tabla comparando dichas herramientas:

Características	***Setmore***	***Harbiz***	***Wodbuster***
Tipo de herramienta	Herramienta de programación de citas y reservas en línea	Plataforma integral de gestión y operaciones para el sector de fitness y bienestar	Aplicación de entrenamiento funcional y bienestar
Servicios adicionales	- Procesamiento de pagos - Integración con plataformas y servicios externos	- Herramientas de marketing y ventas - Análisis de datos e informes	- Contenido educativo sobre técnicas de entrenamiento y nutrición - Comunidad de usuarios
Orientado a	Pequeños negocios y profesionales independientes	Profesionales y empresas del sector de fitness y bienestar	Entusiastas del crossfit y usuarios interesados en el entrenamiento funcional
Planes gratuitos	Sí, versión gratuita con funcionalidades básicas	Disponible durante 15 días, después no está disponible.	Sí, versión gratuita con funcionalidades limitadas
Funcionalidades unificadas en la misma plataforma	No, se enfoca principalmente en la programación de	Sí, solución integral para la gestión del negocio	Sí, concentrada en el entrenamiento funcional y el

	citas y reservas		bienestar
Notificaciones push	Sí, para recordatorios de citas y reservas	Sí, para mantener informados a los usuarios sobre eventos, promociones, etc.	Sí, para notificar a los usuarios sobre nuevos contenidos, desafíos, etc.

Tabla 4. Comparativa de las distintas herramientas

1.3 Motivación y objetivos de la solución propuesta.

Tras las herramientas mencionadas anteriormente, existen varias herramientas que pueden utilizarse como solución al problema comentado, aunque ninguna de ellas es del todo especifica y concreta como la propuesta.

En algunos casos, encontramos funcionalidades que no están en la misma plataforma, si no que los usuarios, clientes, entrenadores, tienen que usar aplicaciones distintas.

En muchas herramientas de este tipo no se contempla la necesidad que surge de tener una interacción completa entre entrenador y cliente.

Por lo tanto, la solución al problema consiste en el desarrollo de una herramienta que cubra ciertas necesidades, y esto es precisamente lo que ha motivado el desarrollo de esta aplicación.

- Debe tratarse de una aplicación multiplataforma donde compartan espacio tanto entrenadores como clientes.
- Desde su panel, los entrenadores deben poder crear y gestionar sus sesiones, incluyendo la posibilidad de:
    - Establecer su disponibilidad horaria.
    - Gestionar la inscripción de los clientes a las sesiones.
    - Generar y gestionar el calendario de sesiones automáticamente.
    - Asignar manualmente horarios a las sesiones.
    - Registrar manualmente el progreso de los clientes.
- Desde su panel, los clientes podrán gestionar todo lo relativo a sus sesiones, incluyendo la posibilidad de:

- Inscribirse a sesiones disponibles.
- Indicar la preferencia horaria.
- Proponer cambios en los horarios de las sesiones.

2. Competencias

Con el desarrollo de este proyecto hemos cubierto las siguientes competencias específicas relativas a la mención de Tecnologías de la Información.

Competencia	*Definición*	*Justificación*
TI03	*"Capacidad para emplear metodologías centradas en el usuario y la organización para el desarrollo, evaluación y gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, ergonomía y usabilidad de los sistemas."*	Uno de los principales elementos de la problemática mencionada anteriormente, es el poder ofrecer a los clientes y entrenadores una misma aplicación donde puedan gestionar y programar sus sesiones, además flexibilizar sus pagos, por ambas partes.
TI06	*"Capacidad de concebir sistemas, aplicaciones y servicios basados en tecnologías de red, incluyendo Internet, web, comercio electrónico, multimedia, servicios interactivos y computación móvil."*	Para desarrollar el proyecto, se diseñó y estructuraron una aplicación móvil multiplataforma integrada con varios servicios que se mencionarán después, como las bases de datos, el sistema de autenticación o el envío de notificaciones.

Tabla 5. Competencias específicas cubiertas

3. Normativa y legislación

3.1 Licencias de software

3.1.1 MIT

La licencia de software MIT [4] es una licencia de software libre permisiva, originada en el Instituto Tecnológico de Massachusetts (MIT). Impone muy pocas limitaciones, lo que le otorga una gran compatibilidad con otras licencias como, por ejemplo, la GNU GPL. Desde sus comienzos ha tenido una gran importancia y se suele emplear en proyectos de software libre.

De entre las herramientas y servicios empleados en este proyecto, se encuentran bajo esta licencia: GitHub.

3.1.2 GNU GPL

Conocida como GNU [5] *(General Public License*), es una licencia de derecho de autor ampliamente usada en el mundo del software libre y código abierto. Garantiza a los usuarios finales (personas, organizaciones, compañías) la libertad de usar, estudiar, compartir (copiar) y modificar el software, garantizado así los objetivos propuestos.

De entre las herramientas y servicios empleados en este proyecto, se encuentran bajo esta licencia: Git para el control de versiones

3.1.3 Apache License

La licencia Apache [6] (Apache License o Apache Software License para versiones anteriores a 2.0) es una licencia de software libre permisiva creada por la Apache Software Foundation (ASF).8 La licencia Apache requiere la conservación del aviso de derecho de autor y el descargo de responsabilidad, pero no es una licencia copyleft, ya que no requiere la redistribución del código fuente cuando se distribuyen versiones modificadas.

Las herramientas que hemos usado que se encuentran bajo esta licencia son Firebase.

3.1.4 BSD

La licencia BSD [7] es la licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution), un tipo del sistema operativo Unix-like. Es una

licencia de software libre permisiva como la licencia de OpenSSL o la MIT License. Esto está en contraste con las licencias copyleft, que tienen de reciprocidad requisitos de compartir-igual. Esta licencia tiene menos restricciones en comparación con otras como la GPL estando muy cercana al dominio público. La licencia BSD al contrario que la GPL permite el uso del código fuente en software no libre. La versión original ya se ha revisado y sus variantes son denominadas licencias BSD modificadas.

Las herramientas que hemos usado que se encuentran bajo esta licencia son: Flutter.

3.1.5 Microsoft Office 365

Microsoft 365 [8] es una línea de servicios por suscripción ofrecidos por Microsoft. El conjunto de soluciones se lanzó en 2014 con el nombre de Office 365.

Gracias a la ULPGC, los estudiantes disponemos de una subscripción que nos permite utilizar dichos servicios, para el desarrollo de este proyecto y la planificación se ha utilizado tanto Microsoft Teams para la planificación con el tutor, como Microsoft Word para la redacción de la memoria final.

3.1.6 CC

Las licencias Creative Commons [9] o CC están inspiradas en la licencia GPL (General Public License) de la Free Software Foundation, y comparten buena parte de su filosofía. La idea principal detrás de ellas es posibilitar un modelo legal ayudado por herramientas informáticas, para así facilitar la distribución y el uso de contenidos.

La herramienta para el diseño de la interfaz de esta aplicación ha sido Figma que se ve afectada por esta licencia.

3.2 Implicación de la Ley Orgánica de Protección de Datos personales y Garantía de los Derechos Digitales y del Reglamento General de Protección de Datos

Los datos que se solicitan al cliente o entrenador por parte de MyTrainer es posible que puedan verse afectados por la Ley Orgánica de Protección de Datos Personales y Garantía de los Derechos Digitales (LOPDGDD) y el Reglamento General de Protección de Datos (RPGD).

MyTrainer intenta limitar un mínimo necesario para el correcto funcionamiento de dicha aplicación, y en este caso, serán nombre, apellidos y email, según la RPGD son datos clasificados a nivel básico.

El tratamiento de estos datos se basa en poder identificar a cada cliente y entrenador que forma parte de la aplicación.

4 Metodología y planificación.

4.1 Metodología.

Para el desarrollo del proyecto se ha utilizado una metodología Scrum para garantizar la entrega de valor durante las distintas iteraciones que se han planificado.

Primero, vamos a definir en qué consiste la metodología ágil Scrum y tras esto, podemos justificar el por qué se ha escogido para este proyecto.

SCRUM es una metodología ágil de gestión de proyectos que se enfoca en el desarrollo iterativo e incremental de software. Se centra en la colaboración y la comunicación efectiva entre los miembros del equipo. El trabajo de desarrollo se divide en pequeñas iteraciones llamadas *sprints*. Cada sprint dura entre una y cuatro semanas y, al final de cada sprint, se entrega un incremento del software.

Esta metodología consiste en varios procesos, reuniones y herramientas que realizan tareas y eventos que cumplen regularmente con el objetivo o solución final o, en definitiva, aportar el máximo valor posible.

Existen varios elementos dentro de este marco, que se definen como “artefactos”, en este caso se trataría de un **Product Backlog**, que representa la lista de las tareas necesarias para conseguir el objetivo planificado.

Estas tareas van suministrándose al **Sprint Backlog**, estas son aquellas en las que el equipo se compromete a finalizar durante un período definido como **sprint o iteración**.

Y finalmente, el **Incremento,** que se refiere al resultado conseguido tras cada sprint/iteración, y que significa un avance real al logro del objetivo propuesto *(figura 8)*.

A su vez, también existen tres roles principales dentro de esta metodología:

- **Product Owner (PO)**: Es el propietario del producto y su responsabilidad principal es definir y priorizar el backlog del producto. Colabora estrechamente con el equipo de desarrollo y los *stakeholders* para decidir qué funcionalidades y características deben incluirse en el producto.
- **Scrum Master (SM)**: Actúa como facilitador del equipo y se encarga de asegurar que el proceso SCRUM se siga correctamente. Colabora con el equipo

para identificar y eliminar cualquier obstáculo que pueda afectar el progreso del proyecto.

- **Equipo de Desarrollo**: Es el grupo encargado de llevar a cabo las tareas y actividades necesarias para desarrollar el producto. Está compuesto por personas multifuncionales que realizan diversas tareas según sea necesario.

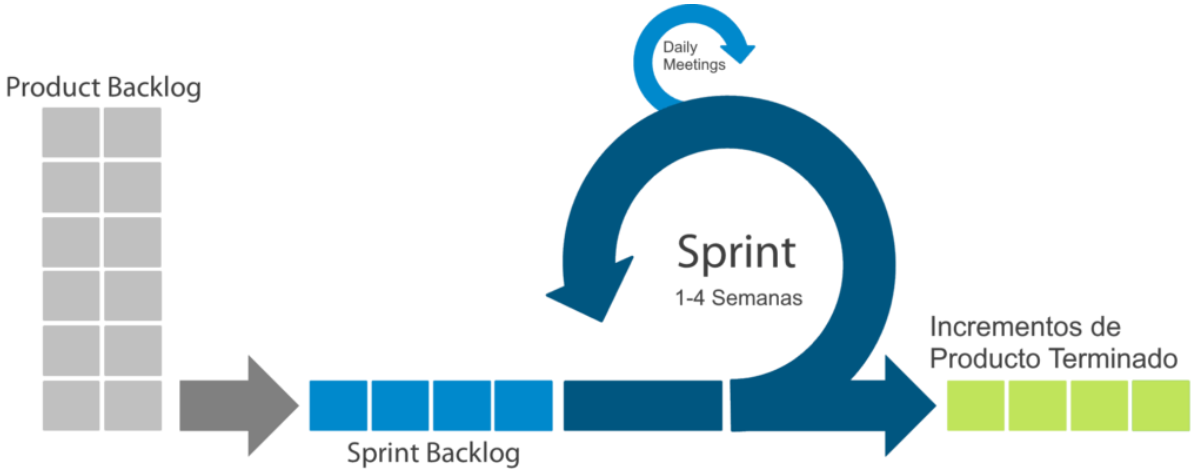


Figura 8. Flujo de trabajo dentro de la metodología SCRUM.

Lo que se pretende con esta metodología es llegar al resultado que se espera a la hora de planificarlo, si es cierto, que este tipo de marcos de trabajo ágil se suelen aplicar con equipos de un cierto número de personas, pero en mi caso, he considerado a mi tutor como un Product Owner (PO), que, durante ciertas semanas, ha estado observando el progreso y valor conseguido a través de cada iteración.

Al final, los valores reales del Scrum se basan en la efectividad, y el compromiso y la concentración, y es lo que se ha conseguido a través de la aplicación de esta metodología.

4.2 Planificación.

Durante la planificación del proyecto, se entregó una primera propuesta (TFT01) con las fases principales y una estimación en horas necesarias para su correcta realización, como se aprecia en la siguiente tabla.

Fases	***Duración Estimada*** ***(horas)***	***Duración Real*** ***(horas)***	***Tareas***
Estudio previo / Análisis	30	30	**Tarea 1.1:** Análisis de requisitos
			Tarea 1.2: Investigación de herramientas a utilizar
			Tarea 1.3: Investigación de funcionalidades ya cubiertas que ofrezcan los resultados que se buscan.
Diseño / Desarrollo / Implementación	210	220	**Tarea 2.1:** Diseño de arquitectura de la aplicación
			Tarea 2.2: Diseño del módulo base
			Tarea 2.3: Implementación de módulos
			Tarea 2.4: Funcionalidad de los módulos
Evaluación / Validación / Prueba	30	30	**Tarea 3.1:** Pruebas del sistema
			Tarea 3.2: Pruebas de usuarios
Documentación / Presentación	30	35	**Tarea 4.1:** Redacción de la documentación referente al proceso
			Tarea 4.2: Preparación y realización de la defensa del proyecto

Tabla 6. Planificación y estimación del desarrollo del proyecto.

En la tabla mencionada, se incluyó una nueva columna sobre la entregada en la primera propuesta, y representa la duración real de cada tarea de cada fase.

Como podemos observar, en general, no ha habido una gran desviación de horas respecto a lo planificado, solamente en las fases de diseño/desarrollo y documentación que presentan una leve diferencia, esto se debe a que, debido al uso de tecnologías que son familiares pero presentaban varios conflictos con ciertas funcionalidades que se habían definido y planificado desde un principio, por lo demás podemos considerar que se ha realizado una planificación coherente y que ha sido posible gracias al tutor, ya que desde un principio, nos reunimos y definimos claramente que es lo que buscábamos y pretendíamos obtener con este proyecto.

Además, la correcta planificación y seguimiento de la estimación inicial ha sido posible gracias a que ya se disponía una experiencia y dominio previo en cuanto a las tecnologías y herramientas que se han utilizado, en otro caso, esto supondría una mayor duración de tiempo ya que se tendrían que familiarizar con dichas herramientas, en nuestro caso, por ejemplo, ya se disponía de un conocimiento previo en tecnologías como Flutter o Firebase.

5 Herramientas y tecnologías utilizadas

Actualmente, existen numerosas herramientas y tecnologías que permiten desarrollar una aplicación móvil con flexibilidad y escalabilidad. Para nuestro proyecto, hemos seleccionado varias de ellas, y a continuación se describen cada una y el motivo de su elección.

5.1 Herramientas de diseño

- *Figma*

Figma [3] es una eficiente herramienta de diseño colaborativo para equipos. Explora ideas y recopila comentarios, crea prototipos realistas y agiliza el desarrollo de productos con sistemas de diseño, cuyo enfoque principalmente se centra en el UI/UX.

El empleo de esta herramienta se ha limitado al uso exclusivo de realizar unas vistas preliminares o mockups que suponían un planteamiento inicial a la hora del desarrollo del proyecto.

5.2 Herramientas para el desarrollo

5.2.1 Lenguajes

- Dart

Es un lenguaje de programación de código abierto, desarrollado por Google. El objetivo de Dart no es reemplazar JavaScript como el principal lenguaje de programación web en los navegadores web, sino ofrecer una alternativa más moderna, además de ser el lenguaje utilizado por Flutter, además de que toda la lógica de la aplicación, la interfaz de usuario y la interacción con los servicios de Firebase están implementadas en Dart.

- Kotlin

Es el lenguaje de programación para el desarrollo de aplicaciones Android nativas, a menudo utilizado en Android Studio, aunque estemos utilizando Flutter y Dart, se ha realizado algunas integraciones necesarias ajustar ciertos valores y añadir ciertos archivos para que esta se pueda integrar correctamente con los distintos servicios, en concreto, tanto a los de Firebase como a los de Stripe.

- Java

Similar a Kotlin, es el lenguaje de programación oficial para el desarrollo de aplicaciones Android nativas, también utilizado por Android Studio, y para nuestro caso, ha sido necesario cambiar un archivo de configuración para establecer correctamente la integración de pagos con Stripe.

- Javascript

Es [14] un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Para nuestro proyecto, es el lenguaje utilizado para implementar ciertas funciones de Firebase a través de las Firebase Functions, que aportan una lógica del lado del servidor.

5.2.2. Frameworks y librerías

- Android Studio

Es un entorno de desarrollo integrado *(IDE)* oficial para el desarrollo de aplicaciones Android, basado en IntelliJ IDEA.

Se ha escogido este IDE ya que ofrece ciertas funciones que en otros entorno son complicadas de replicar, una de ellas, sería el emulador del dispositivo totalmente configurable y funcional para el desarrollo íntegro y completo de aplicaciones nativas, además de incluir ciertas extensiones que faciliten el desarrollo con Flutter, como puede ser el Flutter Inspector el Flutter Live Performance, que provienen de la propia

extensión de Flutter para Android Studio, por estos motivos ha sido el IDE seleccionado para el desarrollo del proyecto.

- Flutter

Es un framework de desarrollo de UI de código libre creado por Google, este permite crear aplicaciones nativas de alto rendimiento para iOS y Android utilizando un único código base.

Se ha escogido debido a la flexibilidad y lo rápido que puede llegar a ser para el desarrollo multiplataforma de aplicaciones móviles, esto nos permite la posibilidad de generar la aplicación tanto para iOS como para Android.

Toda la interfaz de usuario y la lógica de la aplicación está implementadas usando Flutter.

- Firebase

Es una plataforma de desarrollo de aplicaciones móviles y web desarrollada por Google, que proporciona una variedad de herramientas y servicios para la autenticación, la base de datos en tiempo real, el almacenamiento, la analítica, y más.

Para este proyecto se ha utilizado para la autenticación de usuarios *(Firebase Authentication),* almacenamiento de datos en tiempo real *(Firestore Database),* funciones de servidor (Firebase Functions), y en un futuro podría utilizarse para realizar el hosting de dicha aplicación *(App Hosting)*.

Se ha escogido esta herramienta ya que es la presenta mayor compatibilidad y flexibilidad en cuánto aplicaciones móviles se refiere.

- Material Design *(Material 3)*

Material Design es un sistema de diseño creado y respaldado por diseñadores y desarrolladores de Google. Material.io proporciona una guía exhaustiva de UX y la implementación de componentes UI para Android, Flutter y la Web.

La última versión, Material 3, permite experiencias personalizadas, adaptativas y expresivas, abarcando desde colores dinámicos y una accesibilidad mejorada hasta bases para diseños en pantallas grandes y tokens de diseño.

- Cupertino Icons

Proporciona los iconos estilizados para iOS que se utilizan en las aplicaciones Flutter.

- Firebase Core

Librería esencial para inicializar y configurar Firebase en la aplicación, permitiendo la integración con otros servicios de Firebase como Firestore y Authentication

- Firebase Messaging

Gestiona la recepción de mensajes push y notificaciones en Firebase Cloud Messaging, facilitando la comunicación con los usuarios incluso cuando la aplicación está en segundo plano.

- Provider

Implementa el patrón Provider para la gestión de estado en Flutter, permitiendo una administración eficiente de los datos y la lógica de la aplicación.

- Google Sign In

Permite la autenticación de usuarios mediante Google Sign-In, integrado directamente con Firebase Authentication para una experiencia de inicio de sesión simplificada.

- Firebase Auth

Ofrece funcionalidades completas de autenticación de usuarios como registro, inicio de sesión y gestión de sesiones, utilizando los servicios de autenticación de Firebase

- Firebase UI Auth

Proporciona una interfaz de usuario prediseñada para flujos de autenticación en Firebase, facilitando la implementación de formularios de inicio de sesión y registro de usuarios.

- Firebase UI OAuth Google

Implementa el flujo de autenticación específico para Google dentro de Firebase UI, permitiendo a los usuarios autenticarse con sus cuentas de Google de manera segura.

- Cloud Firestore

Proporciona acceso y gestión de datos en tiempo real con Firestore, la base de datos NoSQL de Firebase, ideal para aplicaciones que requieren sincronización instantánea de datos entre dispositivos.

- SyncFusion Calendar

Componente avanzado de calendario de SyncFusion para Flutter, que permite la visualización y gestión de eventos de manera interactiva y personalizable.

- Firebase UI Localizations

Incluye localizaciones de idioma específicas para la interfaz de usuario de autenticación en Firebase, asegurando una experiencia consistente y localizada para los usuarios.

En nuestro caso, la aplicación estará en español exclusivamente.

- Google Nav Bar

Sirve para implementar una barra de navegación estilo Google para Flutter, facilitando la navegación entre pantallas y mejorando la experiencia del usuario con una navegación intuitiva

- Flutter Localizations

Paquete esencial para la internacionalización de la aplicación Flutter, permitiendo adaptar el contenido de la aplicación a diferentes idiomas y regiones.

- SyncFusion Localizations

Proporciona localizaciones específicas para los componentes de SyncFusion utilizados en la aplicación, asegurando una integración completa con la personalización del idioma.

- UUID

Es una librería que sirve para la generación de identificadores únicos (UUID), útil para asignar identificadores únicos a entidades y registros dentro de la aplicación.

- Google Fonts

Añade el uso de fuentes de Google Fonts en la aplicación, permitiendo una personalización con una gran variedad de opciones de fuentes.

- Flutter Launcher Icons

Sirve para la generación de iconos de la aplicación para Flutter, asegurando que los iconos sean generados y configurados correctamente para diferentes tamaños y plataformas.

- INTL

Proporciona las herramientas necesarias para manejar la internacionalización y el formateo de datos en Flutter, asegurando que la aplicación pueda adaptarse y presentar información según las preferencias locales del usuario.

- Stripe Payment

Integra Stripe para el procesamiento de pagos dentro de la aplicación, permitiendo realizar transacciones seguras y gestionar el flujo de pago de manera eficiente.

5.3 Herramientas organizativas y de control de versiones.

- GitHub y Git

GitHub [10] es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git.

Git [11] es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

GitHub, es la herramienta más utilizada en el ámbito de control de versiones y con el que se está familiarizado más, ofrece funciones de trabajo en equipo, gestión de tareas e integración continua, por eso y motivos, se ha usado para alojar el proyecto dentro de un repositorio y que así esté disponible para su revisión.

- Microsoft Word

Microsoft Word [12] es un software para procesamiento de textos desarrollado por Microsoft desde 1983 hasta hoy. Está incluido en el paquete de aplicaciones Microsoft Office, como parte del software de suscripción en línea Microsoft 365 y Works hasta su descontinuación.

6. Análisis

6.1 Usuarios

Dentro de la aplicación diferenciamos a dos tipos de usuarios: el **entrenador personal** y el **cliente**

Un entrenador personal es un usuario que:

- Está registrado y verificado para ser un entrenador por parte del administrador de la aplicación.
- Puede crear, editar y eliminar sesiones de entrenamiento.
- Puedo crear, editar y eliminar subscripciones o pagos a sus sesiones.
- Puede invitar a clientes a través de un código de invitación.
- Puede mostrar información adicional de su perfil, como certificaciones, antigüedad o número de clientes.
- Puede ver las sesiones de cada cliente y sus pagos.
- Puede establecer un número de clientes por sesión.
- Puede establecer una lista de espera, por si, algún cliente cancelase la sesión programada.

Un cliente es un usuario que:

- Está registrado, pero no ha realizado el proceso de verificación para ser un entrenador.
- Puedo asignarse a un entrenador a través de un código de invitación.
- Puede reservar sesiones de entrenamientos.
- Puede cancelar sesiones de entrenamientos con un tiempo máximo de preaviso antes de la sesión reservada.
- Puede ver sus sesiones realizadas.
- Puede ver los pagos realizados.
- Puede ver su información personal a través de su perfil.

6.2 Casos de uso

A continuación, se muestra los distintos diagramas de casos de uso que ilustran los distintos requisitos funcionales de la aplicación.

En el siguiente diagrama incluimos los casos de uso dirigidos al entrenador personal:

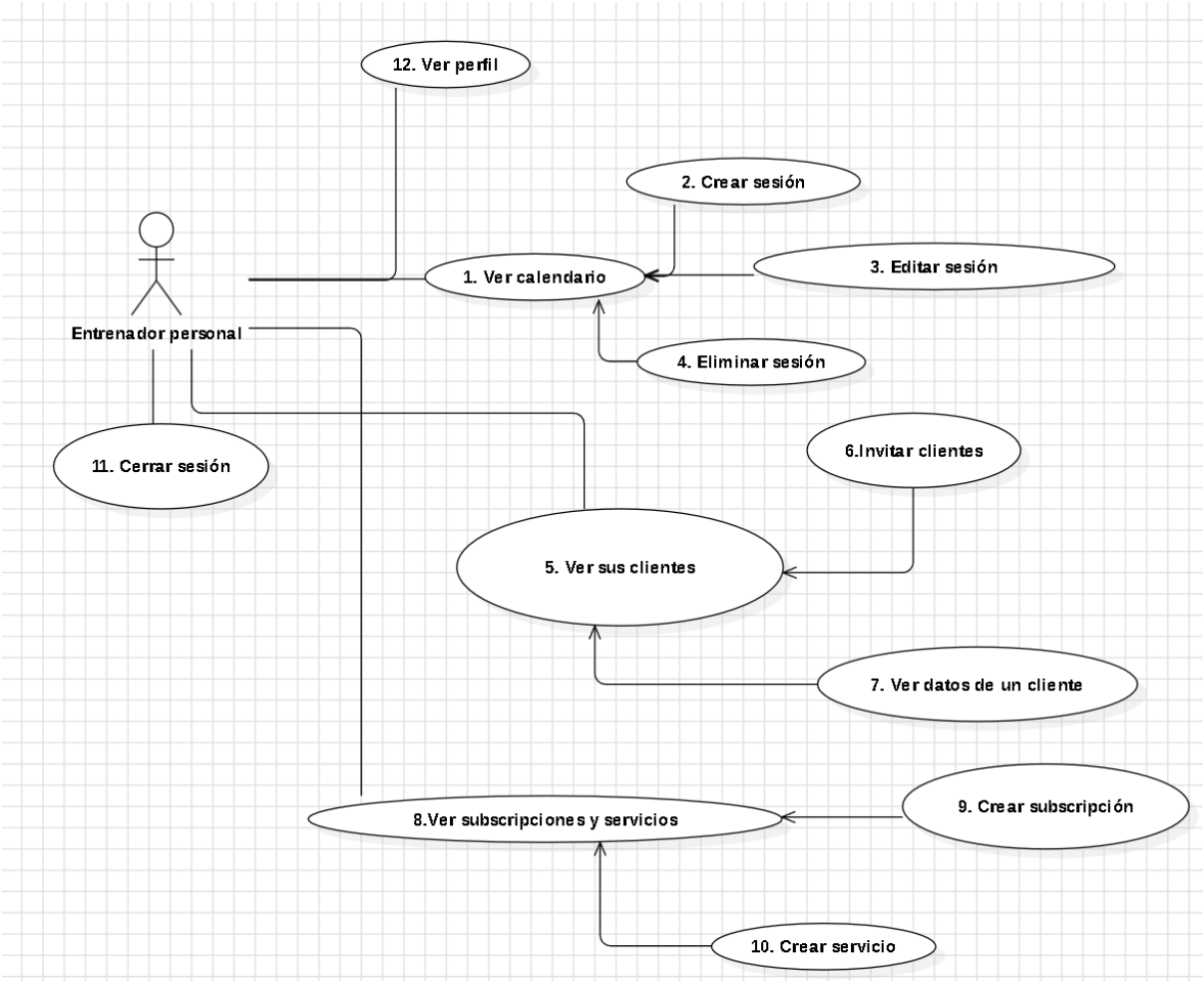


Figura 9. Casos de uso de un entrenador personal

En el siguiente diagrama incluimos los casos de uso dirigidos al cliente:

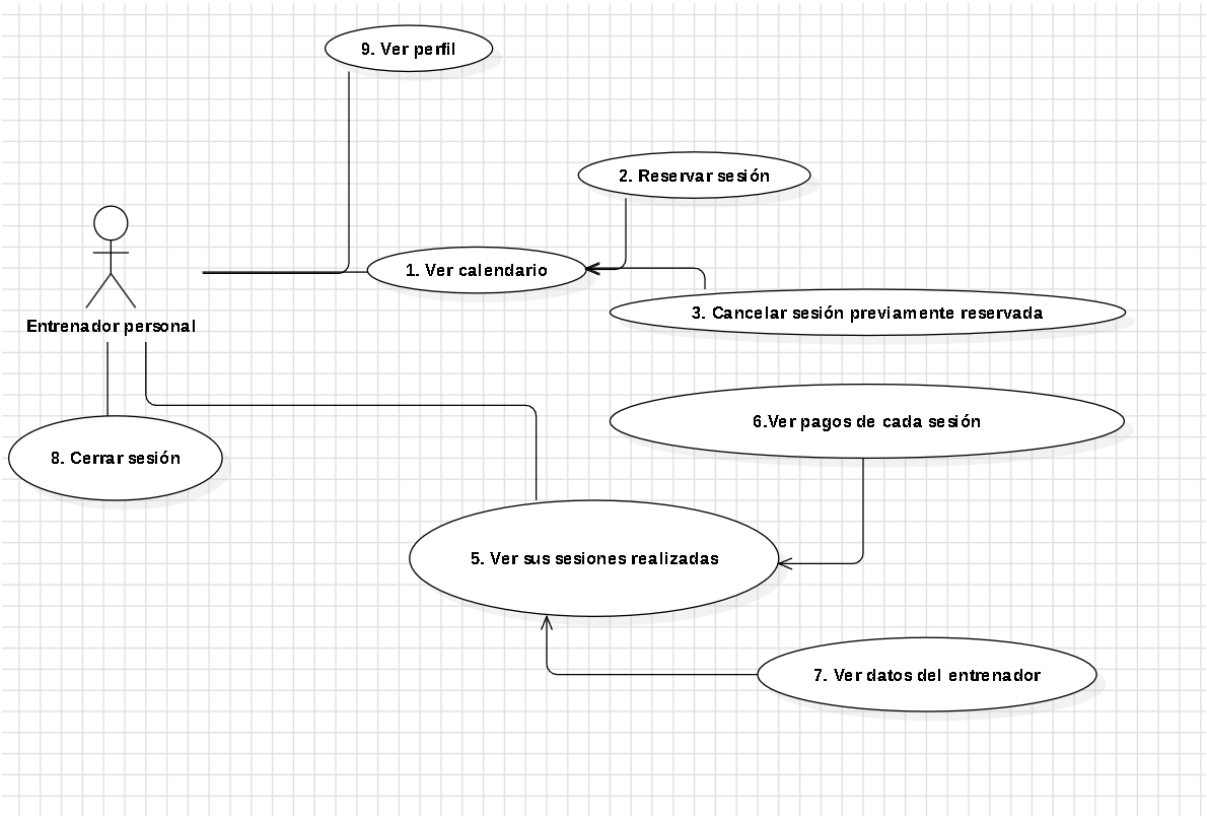


Figura 10. Casos de uso de un cliente

En la siguiente tabla incluimos una breve descripción de los diagramas de casos de uso anteriormente representados:

ID	Caso de uso	Actor principal	Descripción
1	Ver calendario	Entrenador personal	Permite al entrenador personal consultar el calendario de sesiones programadas para revisar y planificar sus actividades y citas con los clientes.
2	Crear sesión	Entrenador personal	Permite al entrenador personal crear una nueva sesión de entrenamiento, especificando detalles como fecha, hora, duración y tipo de sesión, y estableciéndole recurrencia en el caso de que lo desee

3	Editar sesión	Entrenador personal	Permite al entrenador personal modificar los detalles de una sesión de entrenamiento ya programada, como ajustar la fecha, hora o tipo de actividad.
4	Eliminar sesión	Entrenador personal	Permite al entrenador personal cancelar o eliminar una sesión de entrenamiento previamente programada.
5	Ver clientes	Entrenador personal	Permite al entrenador personal visualizar una lista de todos los clientes que están asignados.
6	Invitar clientes	Entrenador personal	Permite al entrenador personal enviar un código de invitación a nuevos clientes para que se registren en el sistema y se asignen a dicho entrenador y comiencen a usar sus servicios.
7	Ver datos de un cliente	Entrenador personal	Permite al entrenador personal acceder a la información detallada de un cliente específico, incluyendo historial de sesiones, preferencias y pagos.
8	Ver subscripciones y servicios	Entrenador personal	Permite al entrenador personal consultar las subscripciones y servicios que están disponibles en el sistema, así como el estado de las subscripciones actuales de los clientes.
9	Crear subscripción	Entrenador personal	Permite al entrenador personal establecer una nueva subscripción para un cliente, definiendo términos y condiciones como la duración y tipo de servicio.
10	Crear servicio	Entrenador personal	Permite al entrenador personal añadir nuevos servicios al sistema,

			describiendo los detalles y características de cada servicio ofrecido.
11	Ver perfil	Entrenador personal/Cliente	Permite tanto al entrenador personal como al cliente ver y editar su perfil personal, incluyendo información como nombre, contacto, preferencias y configuraciones.
12	Cerrar sesión	Entrenador personal/Cliente	Permite a los entrenadores personales y clientes cerrar su sesión actual en el sistema, asegurando que la cuenta quede segura después de su uso
13	Reservar sesión	Cliente	Permite al cliente solicitar y reservar una sesión de entrenamiento con un entrenador personal qué esté disponible en el calendario. El cliente puede seleccionar la sesión según la disponibilidad del entrenador
14	Cancelar sesión previamente asignada	Cliente	Permite al cliente cancelar una sesión de entrenamiento que había reservado anteriormente. Esto puede incluir la necesidad de proporcionar una razón para la cancelación y posiblemente seguir políticas de cancelación del sistema.
16	Ver sesiones realizadas	Cliente	Permite al cliente consultar el historial de sesiones de entrenamiento que ha completado. El cliente puede revisar detalles como fechas, tipos de sesiones y el progreso alcanzado.
17	Ver pagos de cada sesión	Cliente	Permite al cliente revisar la información de pagos relacionados con cada sesión

			de entrenamiento. Esto incluye ver los montos pagados, fechas de pago y posibles facturas o recibos.
18	Ver datos del entrenador asignado	Cliente	Permite al cliente consultar la información del entrenador personal con el que está trabajando. Esto puede incluir el perfil del entrenador, sus certificados, y detalles sobre su experiencia y especializaciones.

Tabla 7. Casos de uso de la aplicación

6.3 Requisitos funcionales

- Los usuarios, ya sean entrenadores o clientes, deben poder registrarse proporcionando su nombre, apellido, correo electrónico y contraseña.
- El sistema debe enviar un correo de verificación para confirmar la identidad del usuario.
- Los clientes deben poder realizar una verificación para convertirse en entrenador personal
- Los usuarios deben poder iniciar sesión usando su correo electrónico y contraseña.
- Si un usuario olvida su contraseña, el sistema debe ofrecer una opción para recuperarla.
- Los usuarios deben poder actualizar su perfil, incluyendo nombre, apellido, foto y descripción, así como sus datos de contacto.
- También deben poder ver la información de su perfil y los datos que han proporcionado.
- Los entrenadores deben crear sesiones de entrenamiento, especificando el nombre, la descripción, la duración, la fecha y la hora, y estableciendo si las sesiones seguirán recurriendo.
- Los entrenadores deben poder editar las sesiones que ya han creado para ajustar detalles como la descripción, duración, fecha y hora.

- En caso de necesidad, los entrenadores deben poder cancelar sesiones programadas y notificar a los clientes afectados.
- Los entrenadores deben tener la opción de configurar sesiones recurrentes, ya sean diarias, semanales o mensuales.
- Los entrenadores deben poder agregar clientes usando un código de invitación o un correo electrónico.
- Los entrenadores deben poder ver la información y el progreso de sus clientes.
- Los clientes deben poder ver las sesiones de entrenamiento que están disponibles para reservar.
- Deben poder reservar las sesiones de entrenamiento que les interesen.
- Los clientes también deben poder cancelar las reservas que hayan realizado.
- Los entrenadores deben poder crear diferentes planes de suscripción, definiendo precios y duración.
- Los clientes deben poder pagar por sesiones individuales, paquetes de sesiones o suscripciones a través de Stripe.
- Tanto los entrenadores como los clientes deben poder ver un calendario con todas las sesiones programadas.
- Los entrenadores deben poder actualizar este calendario para añadir nuevas sesiones o modificar las existentes.
- Los entrenadores deben poder enviar notificaciones a los clientes sobre nuevas sesiones, cambios de horario y recordatorios.
- Los clientes deben recibir notificaciones sobre nuevas sesiones, cambios de horario y recordatorios.
- Los clientes deben poder registrar y revisar su progreso en términos de sesiones asistidas y logros alcanzados.
- También deben poder ver estadísticas sobre su progreso y rendimiento en la aplicación.
- Solo los usuarios autenticados y autorizados deben poder acceder a funciones específicas de la aplicación.

6.4 Requisitos no funcionales

- El sistema contará con un manual de usuario
- La aplicación debe ser multiplataforma, tanto para Android como para iOS.
- Debe disponer de una interfaz sencilla e intuitiva, que no entorpezca la experiencia del usuario.
- El sistema mostrará distintos mensajes de información, aviso, o error de ciertos eventos relacionados con la acción del usuario.

7. Diseño

En este capítulo hablaremos sobre los objetivos del diseño, la organización de la aplicación y sus distintos componentes, y de cómo hemos implementado el enfoque centrado en el usuario para asegurar una experiencia óptima. La aplicación ha sido desarrollada siguiendo las directrices de Material Design 3 para asegurar una interfaz moderna, intuitiva y consistente, utilizando además la librería Firebase UI Auth, que nos proporciona interfaces referentes a la autenticación del usuario.

7.1 Objetivos del diseño

Desde el inicio del proyecto, se tenía clara la necesidad de resolver el problema de la gestión de entrenamientos personales, queríamos diseñar una aplicación que fuera: útil, sencilla, organizada, intuitiva y visualmente atractiva.

La planificación de diseño supone un ahorro clave del tiempo durante el desarrollo, ya que no tenemos que pensar o imaginar como deberían de ser esas vistas.

Para planificar este diseño, primero hay que definir el estilo, colores, formas, fuentes, figuras, etc.... que utilizará la aplicación, hay maneras de establecer estos valores de forma global para que se compartan entre toda la aplicación, que, en nuestro caso, es lo que hemos decidido.

Para definir un ámbito global de estilos, hemos tenido que definir dichos estilos en un archivo común donde posteriormente se importará los estilos necesarios en las vistas que lo necesiten.

Con esto, el resultado del diseño de la aplicación consistirá en un mismo criterio artístico, y, por lo tanto, el cliente como el entrenador personal no sentirán ninguna discrepancia entre las distintas vistas de la aplicación.

Para asegurar una experiencia agradable y comunicativa, se diseñaron interacciones suaves y respuestas visuales claras a las acciones del usuario. Las transiciones y animaciones siguen las recomendaciones de Material Design 3 para asegurar una experiencia fluida.

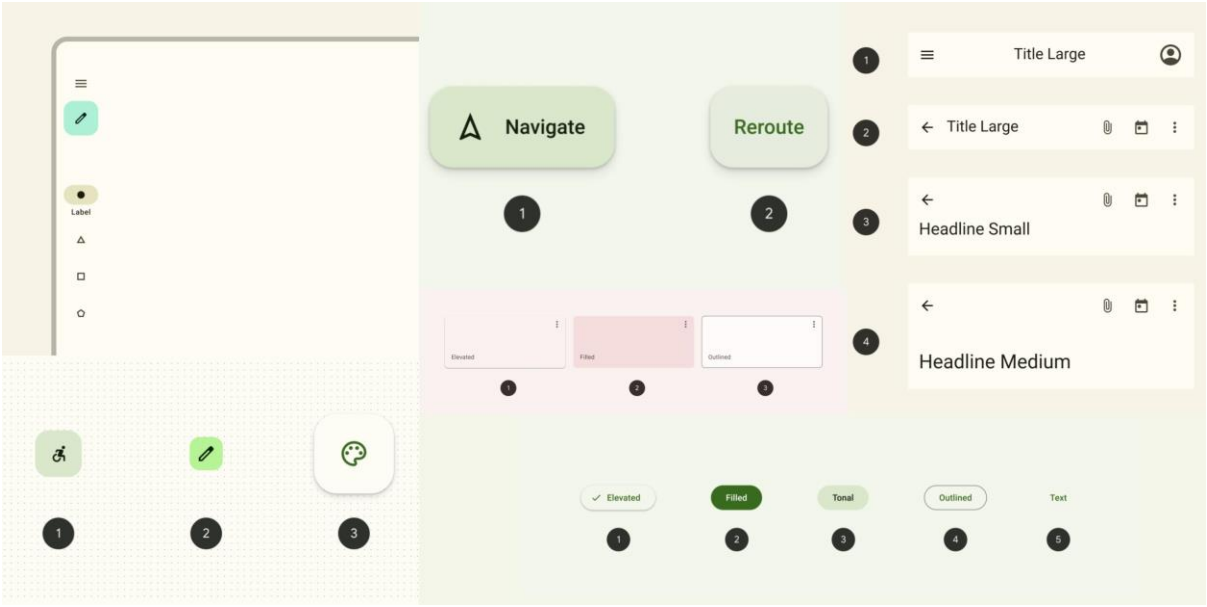


Figura 11. Ejemplos de widgets de Material Design 3

Muchas de estas vistas han sido reutilizadas desde que se ha generado las primeras vistas, como puede ser la del menú principal o calendario.

Nuestro objetivo principal era desarrollar una aplicación que resolviera eficientemente el problema de coordinar entrenamientos entre entrenadores personales y sus clientes. Para lograr esto, diseñamos una función que, basándose en las disponibilidades de los entrenadores y los objetivos de los clientes.

Siguiendo los criterios y necesidades del proyecto, uno de los primeros puntos a definir es el nombre de la aplicación: **MyTrainer**. Este nombre consiste en un simple juego de palabras en inglés que traducido literalmente es “Mi”, “Entrenador”, hemos decidido este nombre ya que es el mejor define y referencia la solución propuesta de la aplicación.

Una vez definido el nombre de nuestra aplicación, hay que representar a través de un logo, al ser una aplicación móvil, tendremos un logo que se utilizará para ser el icono de la aplicación dentro del dispositivo y para su futuro despliegue en alguna tienda de aplicaciones (App Store, Google Play Store), pero además tenemos un logo completo, que además tiene el nombre de la aplicación (*figura 13*).

A continuación, se muestra el logo (*figura 12*):

Figura 12. Logo de MyTrainer

Figura 13. Logo de MyTrainer con nombre.

Esta flexibilidad de logos permite su uso en diferentes vistas de la aplicación, como se podrá observar posteriormente.

Otra decisión a destacar es la paleta de colores escogida para la aplicación *(figura 14)*.

Para el diseño de nuestra aplicación móvil destinada a la gestión de entrenadores personales y clientes, hemos seleccionado una paleta de colores que no solo es estéticamente agradable, sino también funcional y alineada con los objetivos y la experiencia de usuario que deseamos proporcionar. La paleta de colores incluye los siguientes tonos: #1AA2F3 (color principal), #FFEF00, #7F93A7, #8B9DAF. A

continuación, explico la elección de estos colores y cómo contribuyen al diseño de la aplicación:

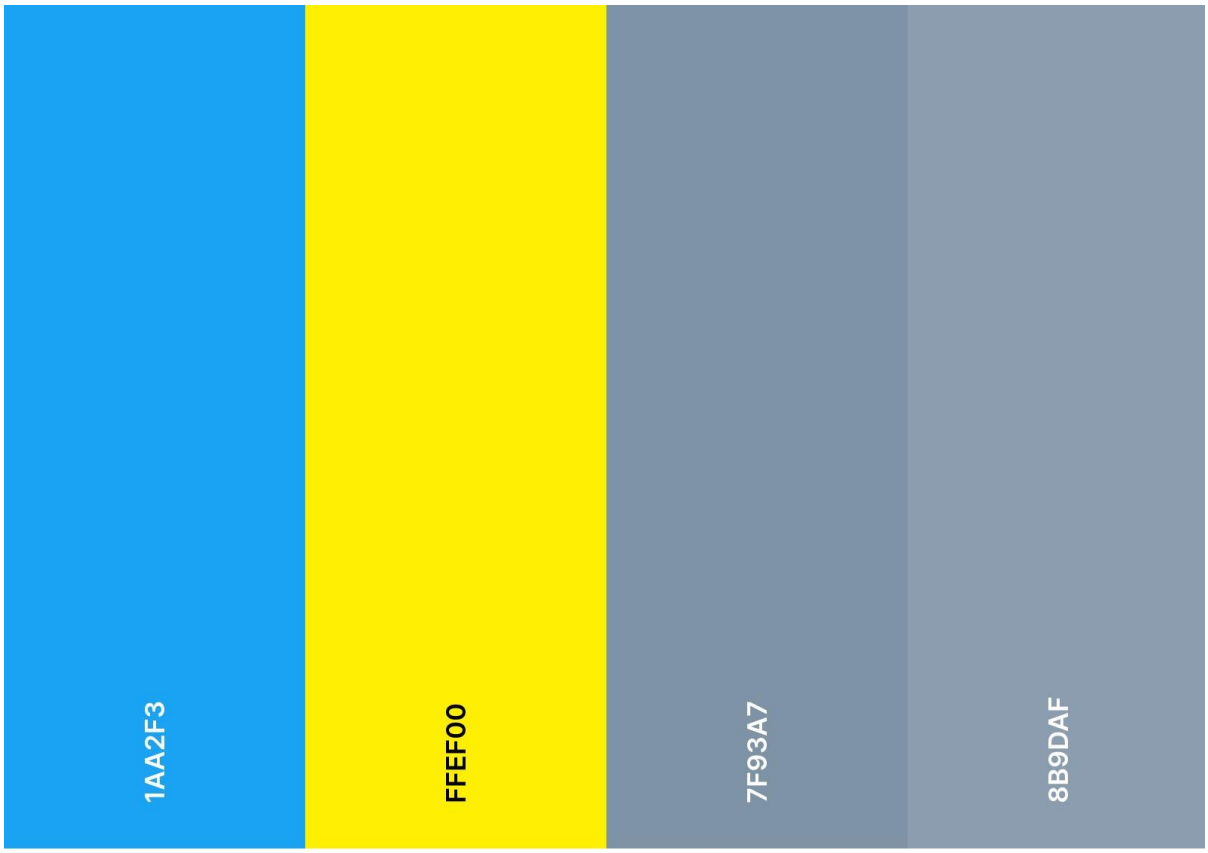


Figura 14. Paleta de colores creada para MyTrainer

Elegimos estos colores que se complementan, escogiendo como color primario el azul (1AA2F3), para crear una interfaz profesional y darle una apariencia coherente respecto al tema del proyecto, también se ha considerado el contraste de estos colores para que sean legibles y los elementos que usen dichos colores sean claros y concisos.

En cuanto al color, esto impacta de alguna manera al usuario transmitiendo sensaciones, lo que pretendemos con esa selección transmitir confianza, profesionalismo y calma, elementos importantes y significativos en la relación entre cliente y entrenador personal.

A su vez, esta paleta de colores nos permite ser flexible en cuanto distintas secciones de la aplicación, ya que nos permite utilizar distintos tonos para distinguir entre tipos de contenido y funcionalidad, sin perder la coherencia visual.

En conclusión, nuestra selección de colores pretende reforzar nuestra identidad de marca, que mejore la experiencia del usuario y transmita profesionalismo y confianza.

7.2 Diseño de *mockups*

Los mockups de MyTrainer fueron diseñados para ser lo más fieles posible a la idea final, permitiendo una transición suave a la etapa de desarrollo. Los componentes y herramientas utilizados se diseñaron con un propósito claro y siguiendo las guías de Material Design 3, en concreto, se ha reutilizado los componentes que ofrece un plugin de Material Theme Builder dentro de Figma.

Tras esto, comencemos a mostrar los distintos mockups de la aplicación:

Comenzamos con los mockups relacionados con la autenticación de usuarios, en concreto, el de *inicio de sesión (figura 15)*.

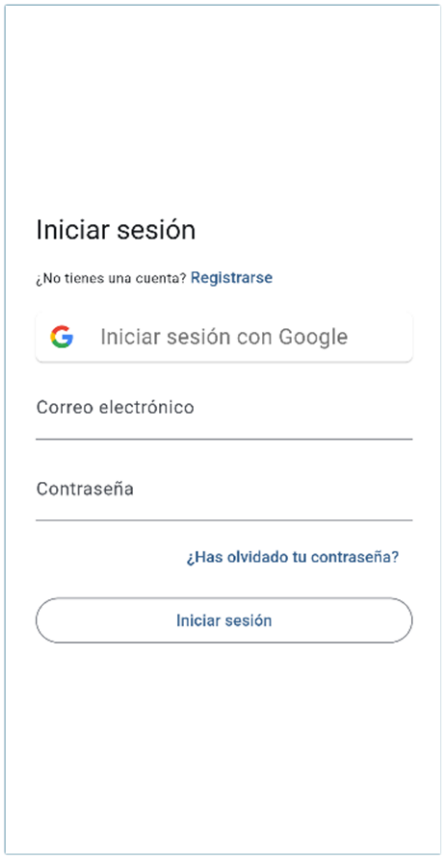


Figura 15. Mockup de inicio de sesión

Aquí tenemos los mockups tanto del registro del usuario como si un usuario olvida su contraseña y desea recuperarla *(figura 16 y 17)*.

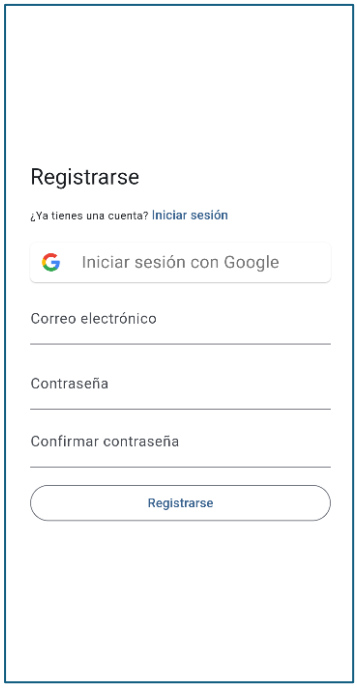


Figura 17. Mockup de registro

Figura 16. Mockup de recuperación de contraseña

- Al iniciar la sesión el usuario (cliente o entrenador), la aplicación redirigirá a dichos usuarios a la pantalla principal (*home*), esta vista será distinta tanto para un cliente como para los entrenadores (*figura 18 y 19*).

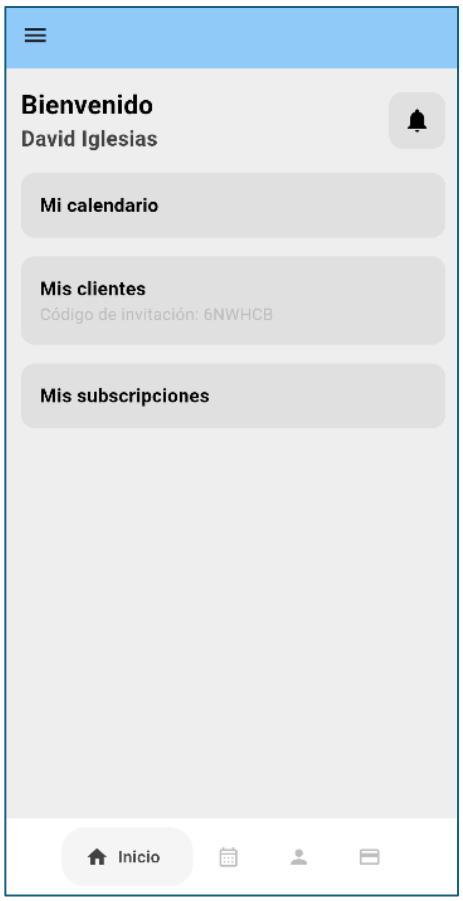


Figura 18. Mockup de home para un entrenador.

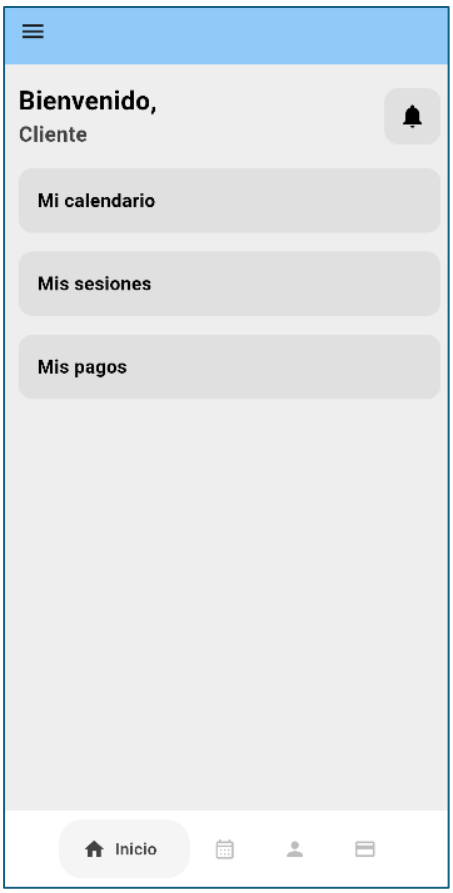


Figura 19.Mockup de home para un cliente.

- En la parte inferior de la pantalla, se encuentra una barra de navegación que contiene diferentes elementos de navegación. Estos elementos están disponibles para el usuario según su tipo, ya sea cliente o entrenador. Dependiendo de su tipo de usuario, podrán acceder a pantallas específicas. Así, los entrenadores tendrán acceso a ciertas pantallas, mientras que los clientes podrán acceder a otras *(figura 20)*.
- Esta barra de navegación se mantiene durante toda la navegación de la aplicación, excepto, en los formularios de sesión y subscripciones.

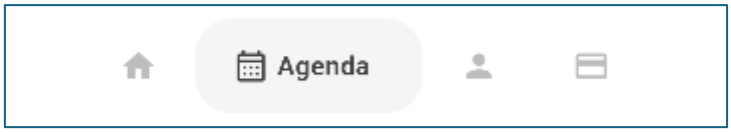
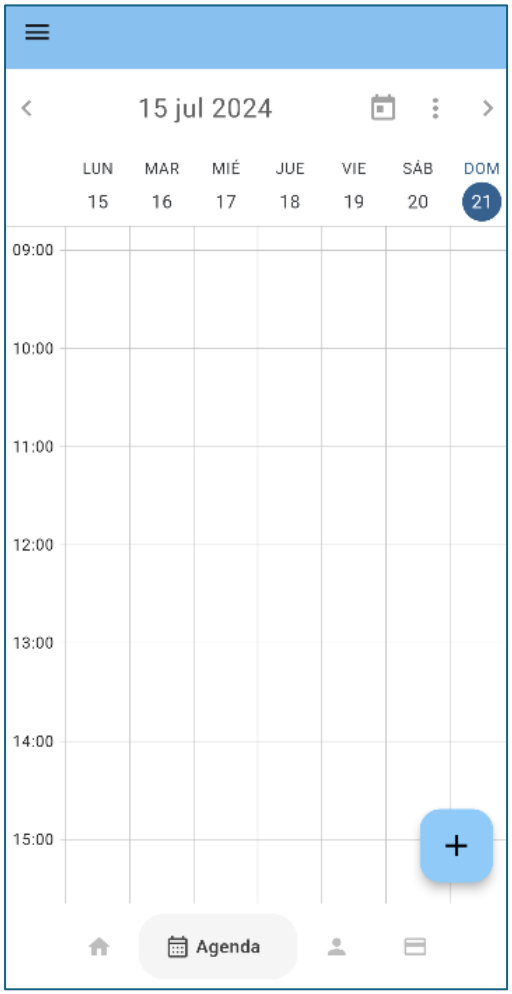


Figura 20. Barra de navegación

- A continuación, mostramos el mockup del calendario, donde se pueden visualizar los eventos creados por el entrenador personal asignado al cliente.
- Esta vista para el entrenador, le permite añadir sesiones desde el botón flotante, en cambio, para los clientes no aparecerá dicho botón (*figura 21*).

Figura 21. Mockup de la vista del calendario



- Esta vista representa el formulario donde el entrenador añade sus sesiones, introduciendo los valores que aparecen e indicando la recurrencia que se desee *(figura 22)*.

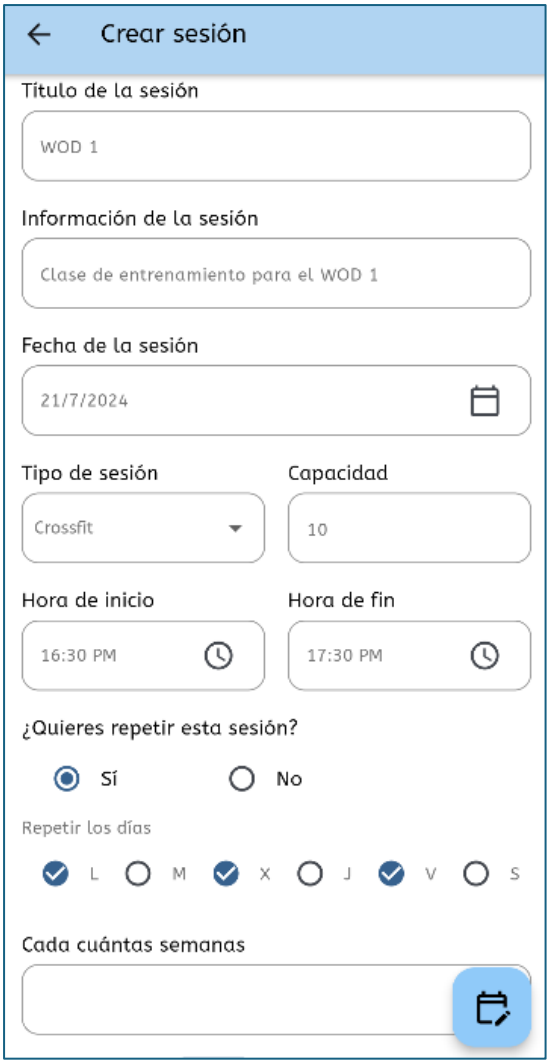


Figura 22. Mockup del formulario de sesión

- Como cliente asignado a un entrenador personal, puedo reservar una sesión de entrenamiento, para esto, tendremos que hacer clic en la sesión deseada y darle a Reservar.
- En esta vista, cada entrenador podrá ver los clientes que están asignados a él, a través del código de invitación que hemos generado al realizar la verificación de cliente a entrenador.
- Por otra parte, tenemos un menú lateral donde dependiendo si el usuario es un cliente, pues tendrá un botón que le permita enviar un correo electrónico a los “administradores” de la plataforma para que se inicie un proceso de verificación

en el cual el cliente pase a ser un entrenador (*figura 23*).

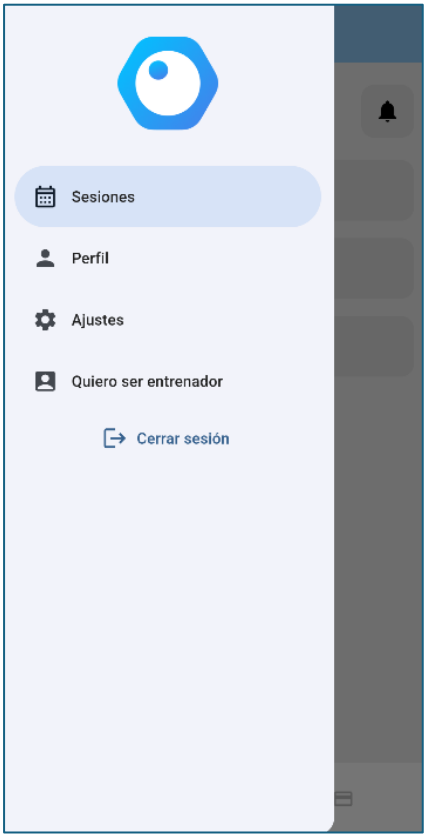


Figura 23. Menú lateral de la aplicación.

- Este menú lateral se mantiene en toda la aplicación, y funciona como una lista de "accesos directos" a distintas partes de la aplicación.
- Continuando, aquí podemos observar el listado de clientes que visualizará el entrenador (*figura 24*).

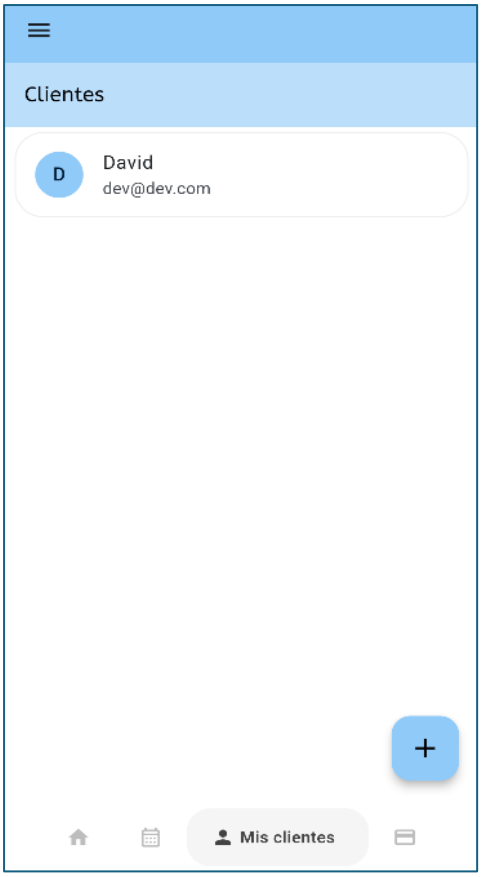


Figura 24. Mockup de listado de clientes de un entrenador.

- En esta vista, el entrenador podrá visualizar todos los clientes que están asignados a él, para asignar un cliente con un entrenador es necesario que este le comparta su código de invitación .
- Esta vista tiene como objetivo que el entrenador pueda visualizar todos los datos generados por sus clientes, ya sea registro de sesiones, pagos, información básica, información médica, altura, peso, etc.

- Para que un entrenador pueda invitar a un cliente lo puede hacer desde este menú, al hacer clic en el botón de añadir, aparecerá un modal *(figura 25)* tal que así:

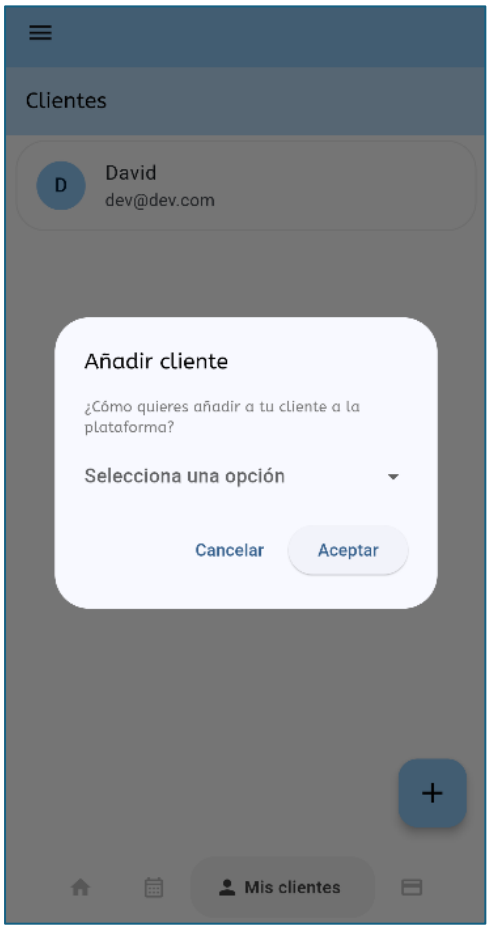


Figura 25. Mockup de dialogo para la invitación a cliente

- Desde este modal, tendrá dos opciones disponibles que son altamente similares, una de ellas es que puedes realizar una invitación a un cliente sin plan de pago, que posteriormente cuando el cliente entre en la aplicación tendrá que establecer dicho plan y realizar el pago, o con la otra opción que representaría invitar a dicho a cliente con un plan de pago ya establecido y que el cliente solo se encargue de introducir sus datos de pago.
- Hablando de pagos, el entrenador tiene una vista disponible donde puede configurar sus distintas subscripciones, planes de pagos, etc.

- Dichos productos se crean en la aplicación, que al disponer una integración con Stripe, estos mismo también se crean en dicho servicio, para así poder llevar un control seguro y eficiente de los pagos (*figura 26*).

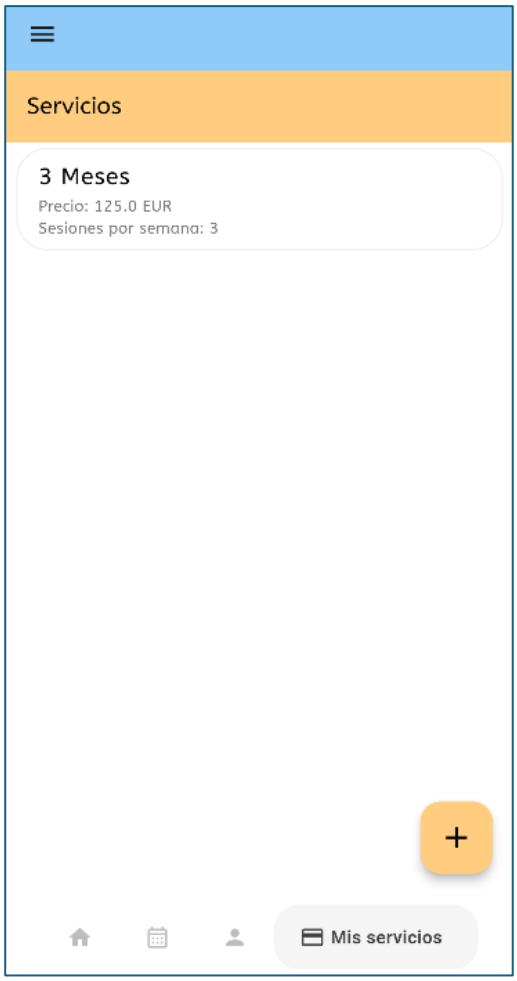


Figura 26. Mockup de listado de servicios por parte de un entrenador.

- La idea de crear la subscripción es que los clientes que están asignados a un entrenador dispongan de un número de sesiones por semana, para que así el entrenador disponga de una mejor gestión de sus clientes.

- Para crear dichas subscripciones, se utilizará el formulario configurado para ello *(figura 27)*.

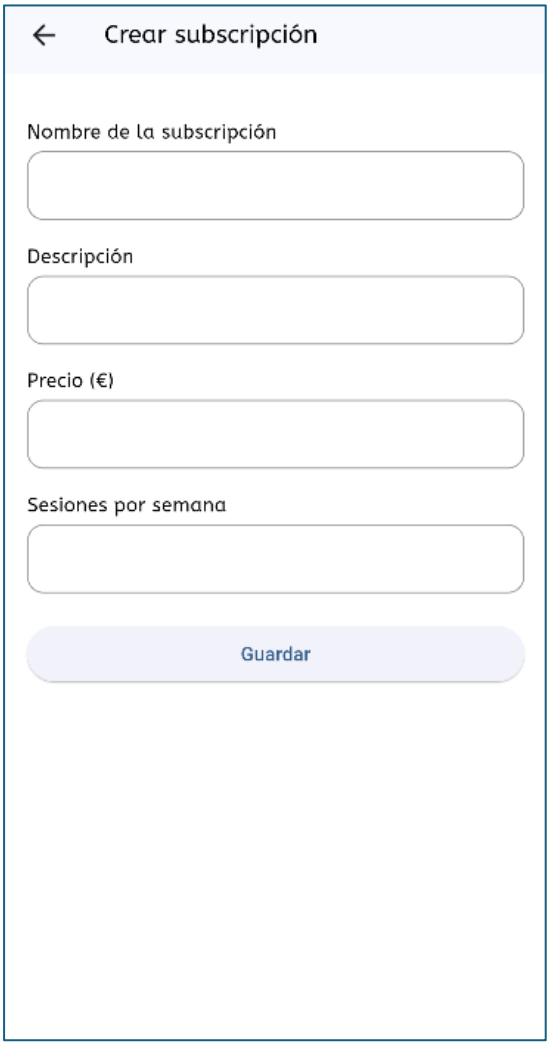


Figura 27. Formulario de creación de subscripción

En resumen, se ha mostrado los mockups principales de la aplicación.

Estos mockups proporcionan una visión completa y detallada de la aplicación MyTrainer, asegurando que tanto entrenadores como clientes tengan una experiencia de usuario fluida, intuitiva y eficiente en la gestión de sus sesiones de entrenamiento.

8. Desarrollo del proyecto

Para la planificación y desarrollo del proyecto se ha utilizado la metodología SCRUM, esto nos permitió trabajar de una manera ordenada y asegurarnos de que cada parte de la aplicación que se ha planeado, se integra de manera correcta. La primera fase del desarrollo se dedicó a crear una estructura base de la aplicación, estableciendo unos estándares para después en el futuro podemos desarrollar todo en un mismo contexto.

Tras esto, se siguío con la gestión de usuarios, implementando Firebase Authentication, definiendo los roles y permisos para cada usuario, e integrando la creación del usuario en Cloud Firestore e Stripe.

Después, se continuo con el módulo de calendario, que permite a los usuarios (*clientes y entrenadores personales*) ver, añadir, o reservar sesiones de manera sencilla, continuamos con el módulo de clientes, donde añadimos las funcionalidades de asignación a un cliente desde un entrenador, y finalmente se creó el módulo de pagos y subscripciones, garantizando así que las compras dentro de la aplicación se realizan de manera segura.

Cada una de estas fases fue planificada y ejecutada en *sprints,* lo que nos permitió revisar y ajustar el trabajo continuamente en base al *feedback* que recibíamos por parte del tutor, asegurando así que el desarrollo del proyecto cumple con las expectativas planificadas.

8.1 Arquitectura del proyecto

Nuestra aplicación está diseñada con una arquitectura basada en la nube con un enfoque modular y escalable, dividiendo el proyecto en varios componentes para facilitar su desarrollo y mantenimiento.

La arquitectura de esta aplicación se divide en tres componentes principales:

- Aplicación móvil *(frontend)*
- Firebase *(backend)*
- API de servicios externos (*Stripe*)

8.2 Aplicación móvil

La aplicación móvil representa el componente principal con el que los usuarios interactúan directamente, tanto entrenadores como clientes.

Se ha desarrollado utilizando Flutter, que consiste en un framwork de desarrollo multiplataforma que permite construir aplicaciones nativas para iOS y Android con un único código fuente.

Flutter [13] fue lanzado en mayo de 2017. Google lo usa internamente para aplicaciones como Google Pay y Google Earth. También se usa en otras empresas como ByteDance y Alibaba.

Los componentes importantes de Flutter incluyen:

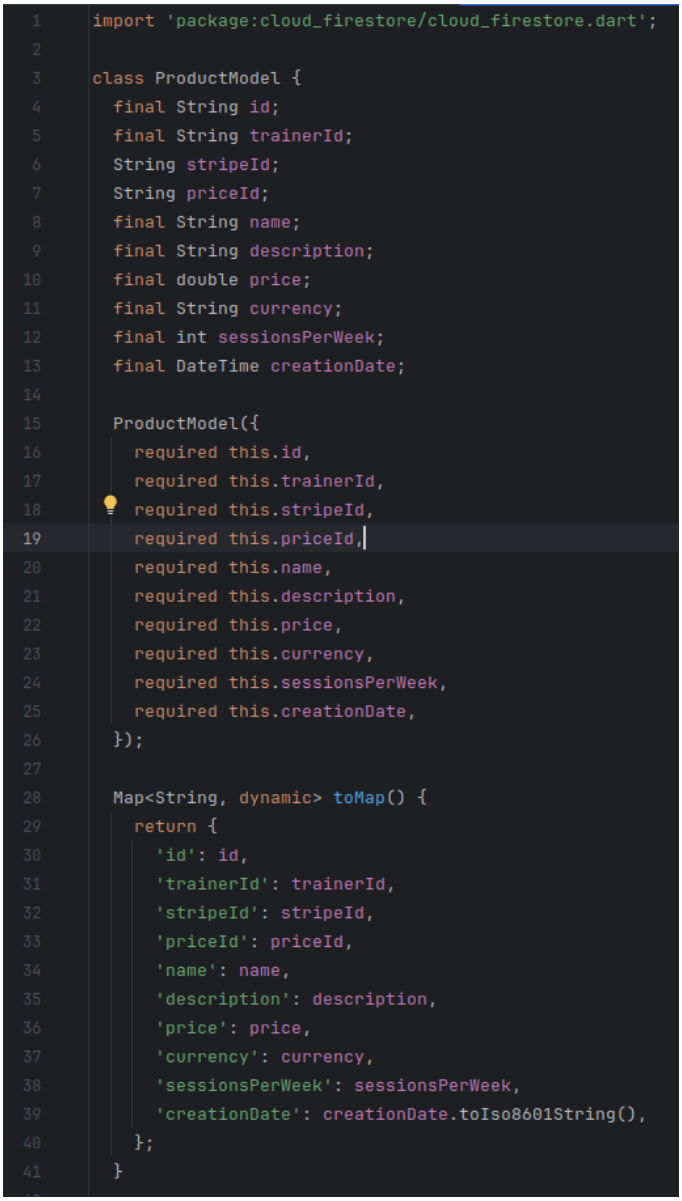
- **Dart platform**, las apps de Flutter están escritas en Dart (lenguaje de programación) y hace uso de muchas de las características más avanzadas.
- **Flutter engine**, que está escrito principalmente en C++, proporciona un soporte de bajo-nivel para renderización que utiliza Google Skia. Además, se vincula con SDKs de Android e iOS [14].
- **Foundation library**, está escrito en Dart, proporciona clases básicas y funciones las cuales suelen construir las aplicaciones que utilizan Flutter, como APIs para comunicar con el motor [15].
- **Design-specific widgets**, que permite contener dos conjuntos de widgets que conforman un lenguaje de diseño concreto. Material Design Widgets implementa el lenguaje de diseño de Google del mismo nombre, y Cupertino widgets imita el diseño de Apple iOS [16] [17].
- **Flutter Development Tools (DevTools)**

8.2.1 Estructura del proyecto

La estructura del proyecto se basa en el patrón de diseño Model-View-ViewModel *(MVVM)*, que promueve la separación de responsabilidades, mejorando la mantenibilidad y la capacidad de prueba.

- **Modelos**: Representan las estructuras de datos como usuarios, eventos, productos y pagos.

Respecto a los modelos, aquí tenemos un ejemplo de cómo lo declaramos:



```dart
import 'package:cloud_firestore/cloud_firestore.dart';

class ProductModel {
  final String id;
  final String trainerId;
  String stripeId;
  String priceId;
  final String name;
  final String description;
  final double price;
  final String currency;
  final int sessionsPerWeek;
  final DateTime creationDate;

  ProductModel({
    required this.id,
    required this.trainerId,
    required this.stripeId,
    required this.priceId,
    required this.name,
    required this.description,
    required this.price,
    required this.currency,
    required this.sessionsPerWeek,
    required this.creationDate,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'trainerId': trainerId,
      'stripeId': stripeId,
      'priceId': priceId,
      'name': name,
      'description': description,
      'price': price,
      'currency': currency,
      'sessionsPerWeek': sessionsPerWeek,
      'creationDate': creationDate.toIso8601String(),
    };
  }
```

Figura 28. Modelo de producto de la aplicación

- **Vistas**: Se encargan de la capa de presentación y la interfaz de usuario, respecto a las vistas, en la aplicación las creamos bajo el nombre de “screen” considerando que cada archivo representa una pantalla en la aplicación, a continuación, vamos a mostrar distintas vistas que se han utilizado para el desarrollo del proyecto, ya sea la vista de inicio de sesión, donde podemos observar las distintas funciones relativas a la creación de usuario, así como también la vista de calendario, su datasource y demás funciones que contiene la aplicación.

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.blue[100],
      shadowColor: Colors.black,
      title: Text(widget.event != null ? 'Editar sesión' : 'Crear sesión', style: appTitleStyle),
    ),  // AppBar
    body: SingleChildScrollView(
      padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 4.0),
      child: Form(
        key: _formKey,
        child: Column(
          children: <Widget>[
            InputField(label: 'Titulo de la sesión', hint: '', controller: _titleController),
            InputField(label: 'Información de la sesión', hint: '', controller: _infoController),
            InputField(
                label: 'Fecha de la sesión',
                hint: DateFormat.yMd('es_ES').format(_selectedDate),
                controller: _selectedDateController,
                widget: IconButton(
                    icon: Icon(Icons.calendar_today_outlined),
                    onPressed: () {
                      _getDateTime();
                    })),  // IconButton, InputField
            Row(...),  // Row
            Row(...),  // Row
            Column(...),  // Column
            Visibility(...),  // Visibility
          ],  // <Widget>[]
        ),  // Column
      ),  // Form
    ),  // SingleChildScrollView
    floatingActionButton: FloatingActionButton(
      backgroundColor: Colors.blue.shade200,
      onPressed: _saveEvent,
      child: Icon(Icons.edit_calendar_outlined),
    ),  // FloatingActionButton
  );  // Scaffold
}
```

Figura 29. Formulario de creación de sesiones

Este código (*figura 29*) crea una pantalla que permite a los entrenadores crear o editar una sesión.

El formulario está dividido en varias partes:

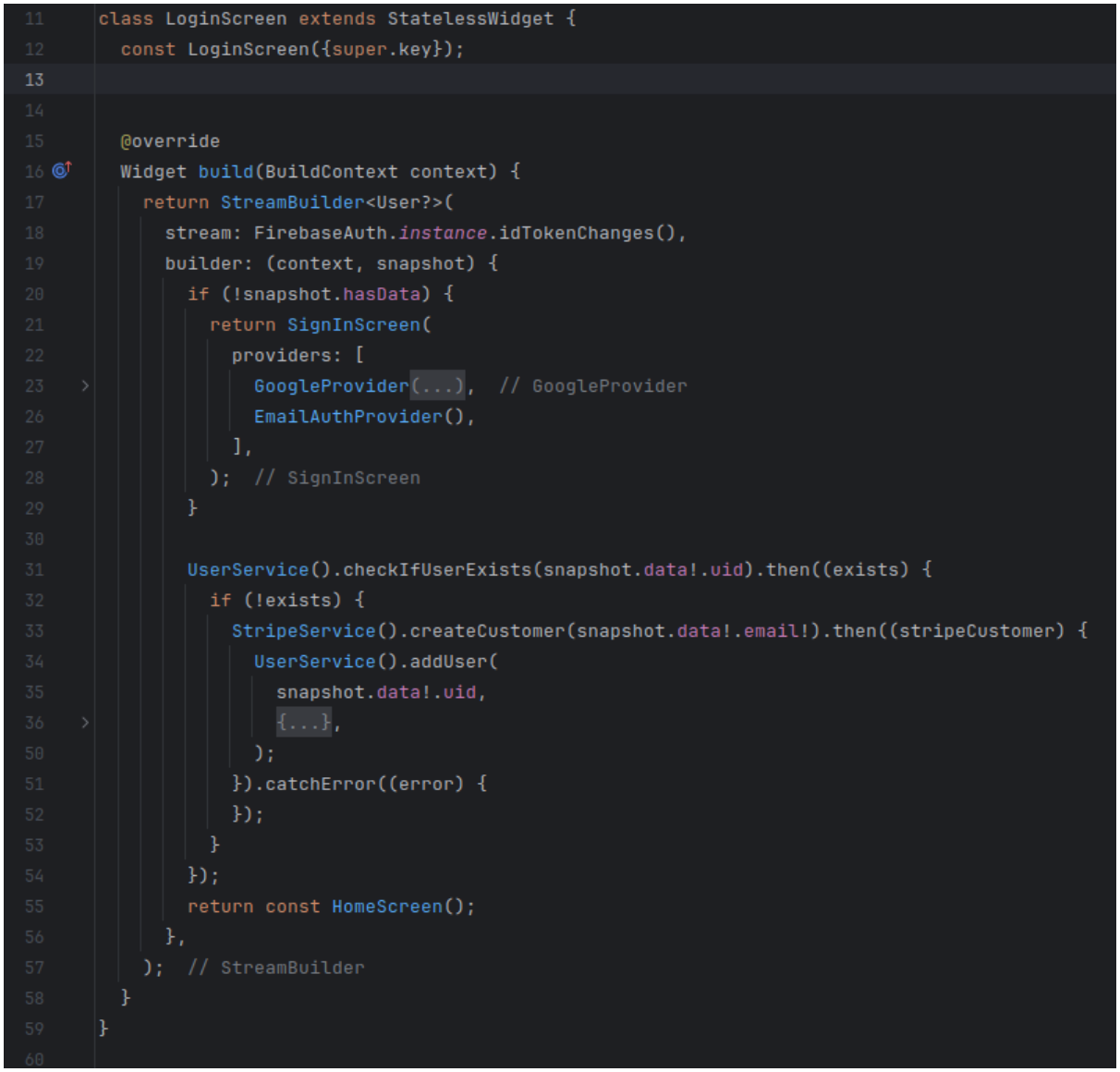
- AppBar, dependiendo si el entrenador está creando o editando una sesión, el título de la barra se cambiará para mostrar “Editar sesión” o “Crear sesión”.
- Formulario, si este tiene muchos campos que no caben en la pantalla, se puede desplazar verticalmente, los campos a introducir son título de la sesión, información de la sesión y la fecha, además de especificar la recurrencia si así se desea.
- FloatingActionButton, es el botón que confirma y guarda la sesión.

Aunque se muestre en la imagen parcialmente, la validación de los datos se realiza a través de la llave del formulario *_formKey*.

Respecto a la *figura 30*, la pantalla extiende de *StatelessWidget*, lo que significa que no mantiene un estado mutable, algo esencial para construir de manera estática.

El constructor de la interfaz tiene varias partes para tener en cuenta:

- StreamBuilder<User?>, escucha cambios en el estado de autenticación del usuario a través del stream *FirebaseAuth.instance.idTokenChanges()*. Este stream emite eventos cada vez que el estado cambio (*cuando un usuario inicia sesión o cierra sesión*).
- Snapshot, la propiedad *snapshot.hasData*, comprueba si el snapshot tiene datos, lo que indicaría que un usuario está autenticado.
- Si no hay datos *(!snapshot.hasData),* se muestra la pantalla de inicio de sesión SignInScreen, que permite a los usuarios autenticarse utilizando proveedores como Google y correo electrónico.
- Si el snapshot tiene datos, se extra el *uid* del usuario, se llama a *UserService().checkIfUserExists(snapshot.data!.uid),* para verificar si el usuario ya está en la base de datos, si no existe, se crea el nuevo cliente en Stripe usando el correo electrónico y se añade el nuevo usuario a Cloud Firestore, tras esto, la aplicación navega a HomeScreen.



```
class LoginScreen extends StatelessWidget {
  const LoginScreen({super.key});


  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream: FirebaseAuth.instance.idTokenChanges(),
      builder: (context, snapshot) {
        if (!snapshot.hasData) {
          return SignInScreen(
            providers: [
              GoogleProvider(...),  // GoogleProvider
              EmailAuthProvider(),
            ],
          );  // SignInScreen
        }

        UserService().checkIfUserExists(snapshot.data!.uid).then((exists) {
          if (!exists) {
            StripeService().createCustomer(snapshot.data!.email!).then((stripeCustomer) {
              UserService().addUser(
                snapshot.data!.uid,
                {...},
              );
            }).catchError((error) {
            });
          }
        });
        return const HomeScreen();
      },
    );  // StreamBuilder
  }
}
```

Figura 30. Código de la pantalla de inicio de sesión

Seguimos con la *figura 31*, la pantalla extiende de *State<HomeScreen>,* lo que significa que esta pantalla puede tener un estado mutable, o sea que se actualiza dinámicamente.

Esta pantalla tiene la siguiente propiedades y métodos:

- scaffoldKey: Una clave global para acceder al estado del Scaffold.
- _selectedIndex: Un índice entero que rastrea qué página está seleccionada actualmente.
- navigateBottomBar: Un método que se llama cuando el usuario selecciona un elemento de la barra de navegación inferior.

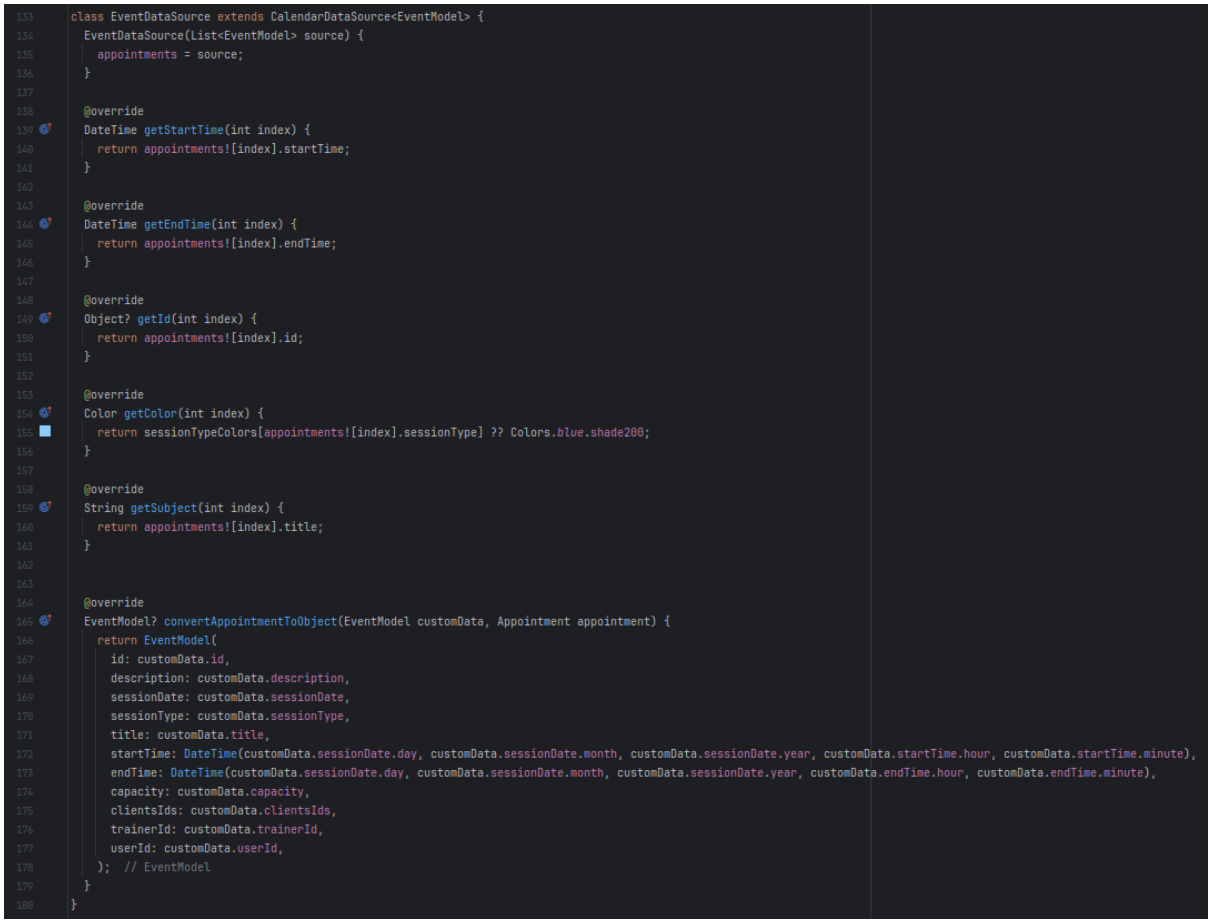
- _pages: Una lista de widgets que representan las diferentes páginas que se pueden mostrar en la pantalla principal.

Respecto a la creación de la interfaz, podemos identificar la estructura con la que se contruye:

- Scaffold: Es un contenedor que implementa la estructura básica de Material Design.
    - bottomNavigationBar: Barra de navegación inferior.
        - onTap: Define qué hacer cuando se selecciona un elemento de la barra de navegación inferior. Aquí se llama al método navigateBottomBar.
- body: Contenido principal de la pantalla, que muestra la página seleccionada actualmente basada en _selectedIndex.
- appBar: Barra de aplicación en la parte superior.
- drawer: Cajón de navegación lateral.
    - onDestinationSelected: Define qué hacer cuando se selecciona un elemento del cajón de navegación.
    - selectedIndex: El índice del elemento seleccionado actualmente.
    - children: Lista de widgets que representan los elementos del cajón de navegación.
        - Container: Un contenedor con una altura y un padding definidos que muestra una imagen.
        - map: Mapea cada elemento de navegación a un NavigationDrawerDestination, que define la apariencia y funcionalidad de cada elemento en el cajón de navegación.
        - SignOutButton: Un botón para cerrar sesión.

```dart
  State<StatefulWidget> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final GlobalKey<ScaffoldState> scaffoldKey = GlobalKey<ScaffoldState>();
  int _selectedIndex = 0;
  void navigateBottomBar(int index) {...}
  final List<Widget> _pages = [...];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      bottomNavigationBar: BottomNavbar(
        onTabChange: (index) => navigateBottomBar(index),
      ),  // BottomNavbar
      body: _pages[_selectedIndex],
      appBar: AppBar(...),  // AppBar
        drawer: NavigationDrawer(
          onDestinationSelected: navigateBottomBar,
          selectedIndex: _selectedIndex,
          children: <Widget>[
            Container(
              height: 150,
              padding: const EdgeInsets.all(8.0),
              child: Image.asset('assets/icon.png'),
            ),  // Container
            ...navigationItems.map(
                  (navigationItem) {
                return NavigationDrawerDestination(
                  label: Text(navigationItem.label, style: TextStyle(color: Colors.grey.shade900)),
                  icon: navigationItem.icon,
                  selectedIcon: navigationItem.selectedIcon,
                );  // NavigationDrawerDestination
              },
            ),
            SignOutButton(variant: ButtonVariant.text),
          ],  // <Widget>[]
        )  // NavigationDrawer
    );  // Scaffold
  }
}
```

Figura 31. Código de la pantalla home de la aplicación



```dart
class EventDataSource extends CalendarDataSource<EventModel> {
  EventDataSource(List<EventModel> source) {
    appointments = source;
  }

  @override
  DateTime getStartTime(int index) {
    return appointments![index].startTime;
  }

  @override
  DateTime getEndTime(int index) {
    return appointments![index].endTime;
  }

  @override
  Object? getId(int index) {
    return appointments![index].id;
  }

  @override
  Color getColor(int index) {
    return sessionTypeColors[appointments![index].sessionType] ?? Colors.blue.shade200;
  }

  @override
  String getSubject(int index) {
    return appointments![index].title;
  }


  @override
  EventModel? convertAppointmentToObject(EventModel customData, Appointment appointment) {
    return EventModel(
      id: customData.id,
      description: customData.description,
      sessionDate: customData.sessionDate,
      sessionType: customData.sessionType,
      title: customData.title,
      startTime: DateTime(customData.sessionDate.day, customData.sessionDate.month, customData.sessionDate.year, customData.startTime.hour, customData.startTime.minute),
      endTime: DateTime(customData.sessionDate.day, customData.sessionDate.month, customData.sessionDate.year, customData.endTime.hour, customData.endTime.minute),
      capacity: customData.capacity,
      clientsIds: customData.clientsIds,
      trainerId: customData.trainerId,
      userId: customData.userId,
    );  // EventModel
  }
}
```

Figura 32. Código de la fuente de datos utilizado por la pantalla del calendario.

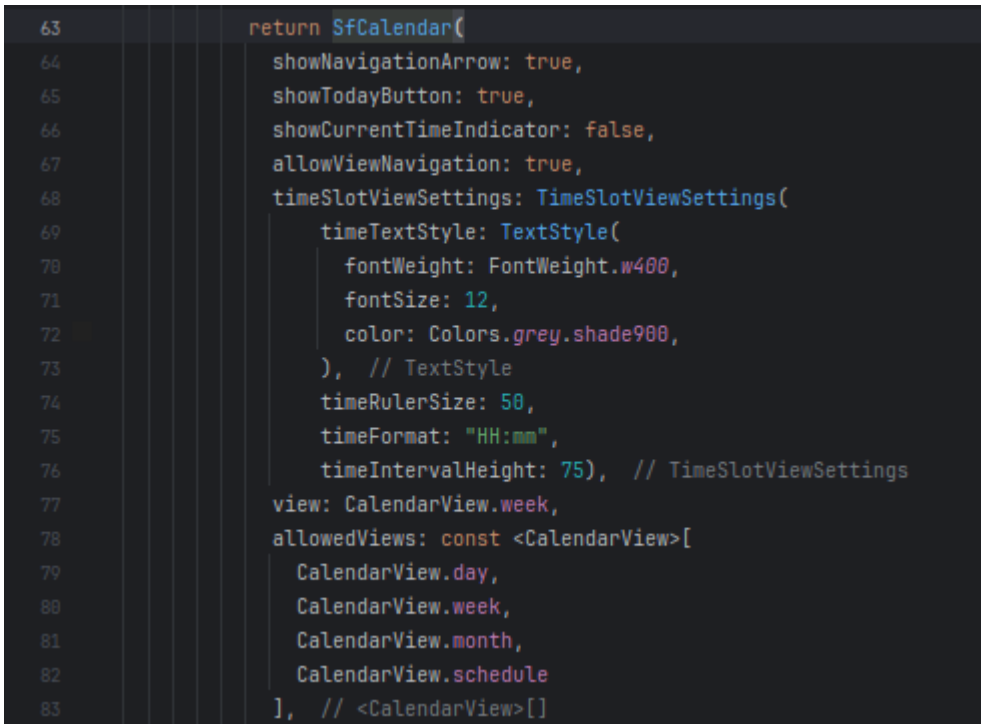
```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: FutureBuilder<void>(
      future: _initialLoad,
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return Center(child: CircularProgressIndicator());
        }
        if (snapshot.hasError) {
          return Center(child: Text('Error: ${snapshot.error}'));
        }

        CalendarController calendarController = CalendarController();
        return SfCalendar(...);  // SfCalendar
      }
    ),  // FutureBuilder
    floatingActionButton: FloatingActionButton(
      backgroundColor: Colors.blue.shade200,
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => AddSessionScreen(),
          ),  // MaterialPageRoute
        );
      },
      child: const Icon(Icons.add),
    ),  // FloatingActionButton
  );  // Scaffold
}
}
```

Figura 33.Código de la pantalla de calendario (parte 1)

Al igual que en las otras pantallas, la construcción de esta se inicia desde el método *build*, que mantiene la siguiente estructura:

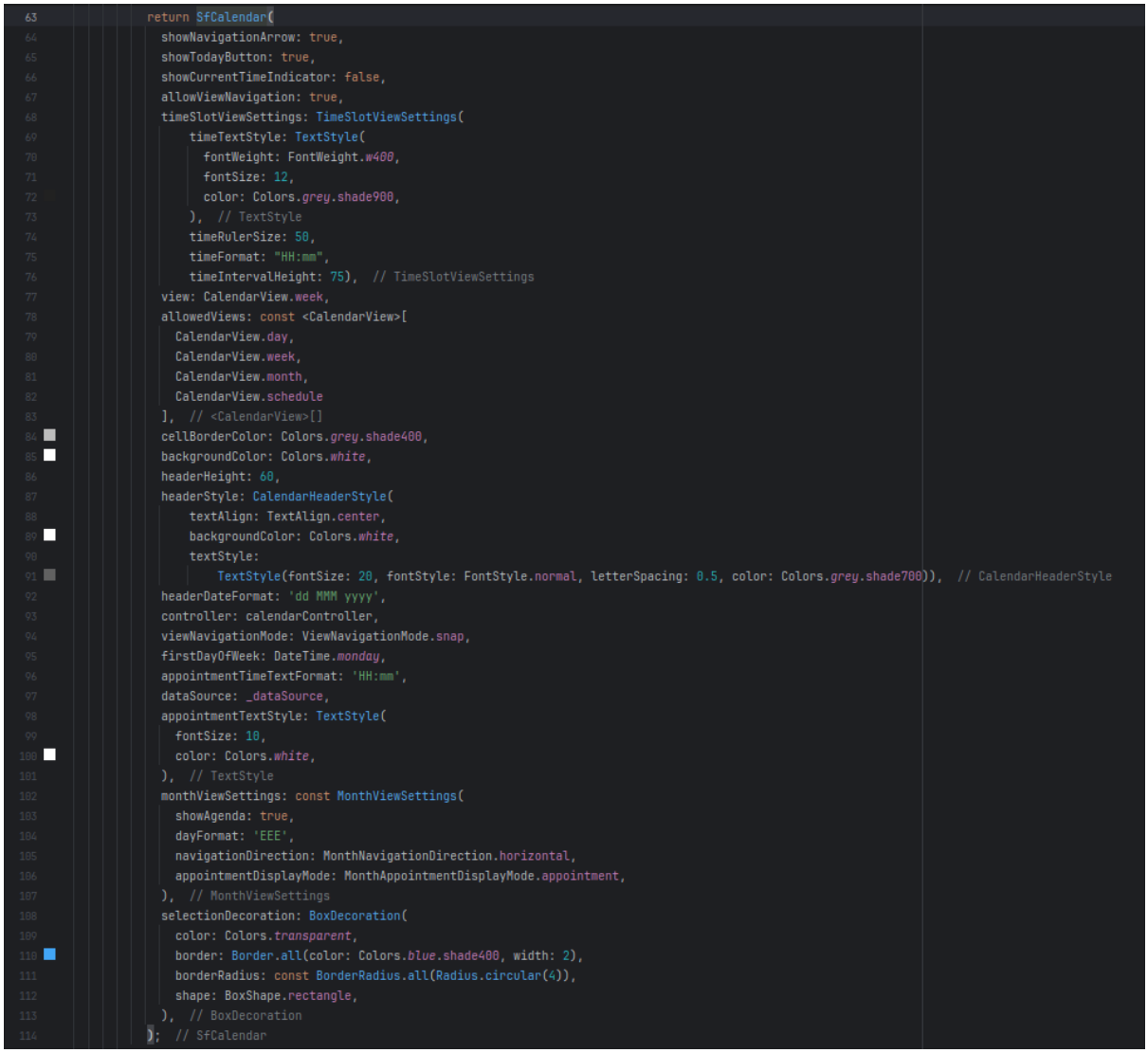
- Scaffold: Proporciona la estructura inicial de la pantalla.
- FutureBuilder: Maneja la carga asíncrona de datos antes de mostrar el calendario.
- ConnectionState.waiting: Muestra un indicador de progreso mientras se cargan los datos.
- snapshot.hasError: Muestra un mensaje de error si ocurre algún problema.
- CalendarController: Controlador para gestionar el estado del calendario.
- SfCalendar: Widget de Syncfusion para mostrar el calendario (configurado posteriormente).

- FloatingActionButton: Un botón flotante que abre la pantalla para añadir una nueva sesión al ser presionado.



```
63          return SfCalendar(
64            showNavigationArrow: true,
65            showTodayButton: true,
66            showCurrentTimeIndicator: false,
67            allowViewNavigation: true,
68            timeSlotViewSettings: TimeSlotViewSettings(
69                timeTextStyle: TextStyle(
70                  fontWeight: FontWeight.w400,
71                  fontSize: 12,
72                  color: Colors.grey.shade900,
73                ),  // TextStyle
74                timeRulerSize: 50,
75                timeFormat: "HH:mm",
76                timeIntervalHeight: 75),  // TimeSlotViewSettings
77            view: CalendarView.week,
78            allowedViews: const <CalendarView>[
79              CalendarView.day,
80              CalendarView.week,
81              CalendarView.month,
82              CalendarView.schedule
83            ],  // <CalendarView>[]
```

Figura 34. Código de la pantalla de calendario (parte 2)



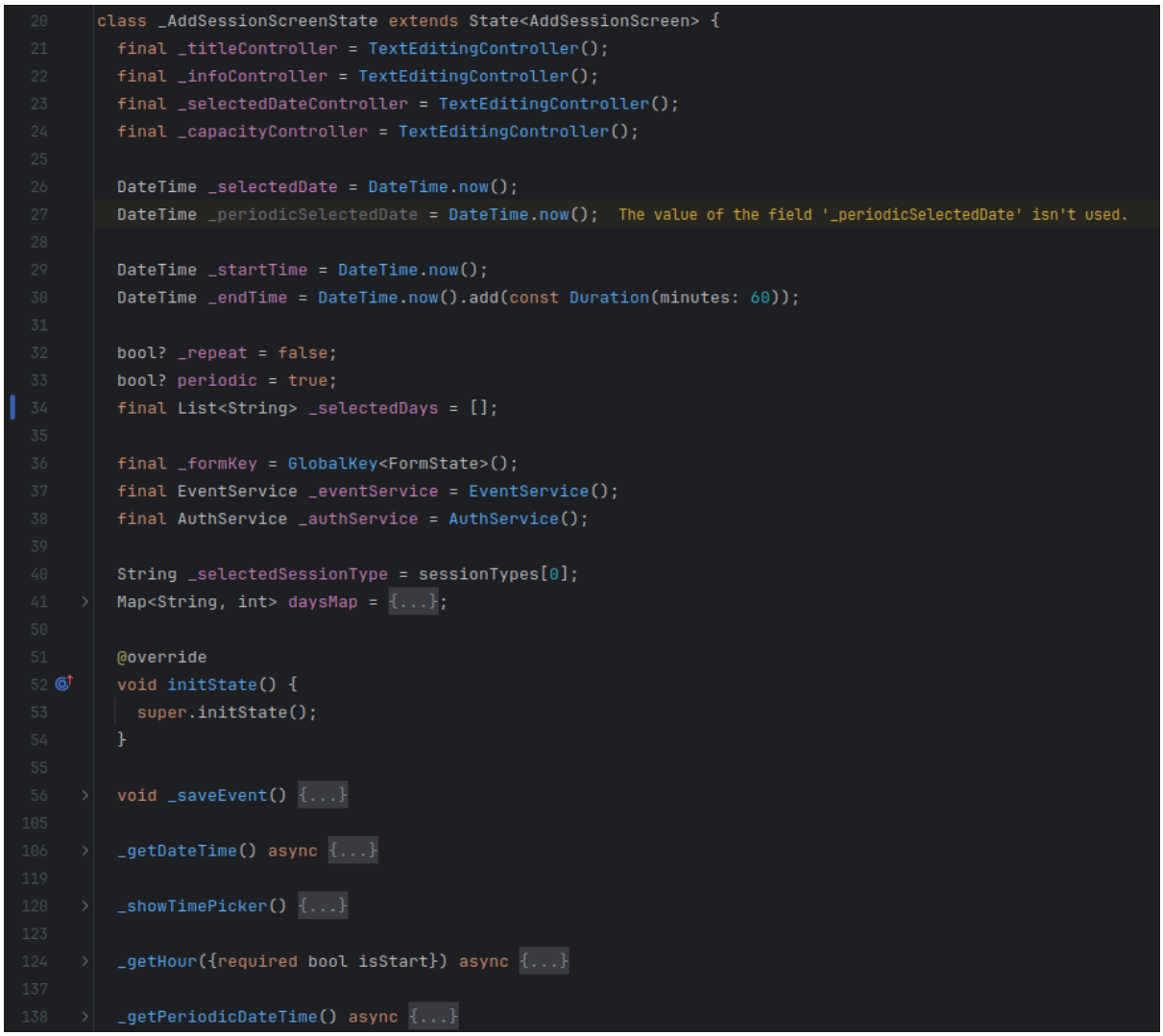
```
return SfCalendar(
  showNavigationArrow: true,
  showTodayButton: true,
  showCurrentTimeIndicator: false,
  allowViewNavigation: true,
  timeSlotViewSettings: TimeSlotViewSettings(
      timeTextStyle: TextStyle(
        fontWeight: FontWeight.w400,
        fontSize: 12,
        color: Colors.grey.shade900,
      ),  // TextStyle
      timeRulerSize: 50,
      timeFormat: "HH:mm",
      timeIntervalHeight: 75),  // TimeSlotViewSettings
  view: CalendarView.week,
  allowedViews: const <CalendarView>[
    CalendarView.day,
    CalendarView.week,
    CalendarView.month,
    CalendarView.schedule
  ],  // <CalendarView>[]
  cellBorderColor: Colors.grey.shade400,
  backgroundColor: Colors.white,
  headerHeight: 60,
  headerStyle: CalendarHeaderStyle(
      textAlign: TextAlign.center,
      backgroundColor: Colors.white,
      textStyle:
          TextStyle(fontSize: 20, fontStyle: FontStyle.normal, letterSpacing: 0.5, color: Colors.grey.shade700)),  // CalendarHeaderStyle
  headerDateFormat: 'dd MMM yyyy',
  controller: calendarController,
  viewNavigationMode: ViewNavigationMode.snap,
  firstDayOfWeek: DateTime.monday,
  appointmentTimeTextFormat: 'HH:mm',
  dataSource: _dataSource,
  appointmentTextStyle: TextStyle(
    fontSize: 10,
    color: Colors.white,
  ),  // TextStyle
  monthViewSettings: const MonthViewSettings(
    showAgenda: true,
    dayFormat: 'EEE',
    navigationDirection: MonthNavigationDirection.horizontal,
    appointmentDisplayMode: MonthAppointmentDisplayMode.appointment,
  ),  // MonthViewSettings
  selectionDecoration: BoxDecoration(
    color: Colors.transparent,
    border: Border.all(color: Colors.blue.shade400, width: 2),
    borderRadius: const BorderRadius.all(Radius.circular(4)),
    shape: BoxShape.rectangle,
  ),  // BoxDecoration
);  // SfCalendar
```

Figura 35. Código de la pantalla de calendario (parte 3)

Esta figura, hace referencia a la configuración del calendario, a continuación, definiré ciertos valores que se han empleado a la hora de implementar el calendario:

- showNavigationArrow: Muestra flechas de navegación.
- showTodayButton: Muestra un botón para ir a la fecha actual.
- showCurrentTimeIndicator: Desactiva el indicador de tiempo actual.
- allowViewNavigation: Permite la navegación entre vistas.
- timeSlotViewSettings: Configuraciones para la vista de intervalos de tiempo.
- view: Vista predeterminada semanal.
- allowedViews: Vistas permitidas: día, semana, mes y horario.
- cellBorderColor: Color de borde de las celdas.
- backgroundColor: Color de fondo del calendario.

- headerStyle: Estilo de la cabecera del calendario.
- appointmentTextStyle: Estilo del texto de las citas.
- monthViewSettings: Configuraciones para la vista mensual.
- selectionDecoration: Decoración de la selección en el calendario.



```
class _AddSessionScreenState extends State<AddSessionScreen> {
  final _titleController = TextEditingController();
  final _infoController = TextEditingController();
  final _selectedDateController = TextEditingController();
  final _capacityController = TextEditingController();

  DateTime _selectedDate = DateTime.now();
  DateTime _periodicSelectedDate = DateTime.now();  The value of the field '_periodicSelectedDate' isn't used.

  DateTime _startTime = DateTime.now();
  DateTime _endTime = DateTime.now().add(const Duration(minutes: 60));

  bool? _repeat = false;
  bool? periodic = true;
  final List<String> _selectedDays = [];

  final _formKey = GlobalKey<FormState>();
  final EventService _eventService = EventService();
  final AuthService _authService = AuthService();

  String _selectedSessionType = sessionTypes[0];
  Map<String, int> daysMap = {...};

  @override
  void initState() {
    super.initState();
  }

  void _saveEvent() {...}

  _getDateTime() async {...}

  _showTimePicker() {...}

  _getHour({required bool isStart}) async {...}

  _getPeriodicDateTime() async {...}
```

Figura 36. Código de la pantalla de calendario (parte 4)

Las anteriores figuras *(figura 32, 33, 34, 35, 36)* son el código que implementa la pantalla del calendario, esta pantalla permite a los usuarios ver el calendario en varias vistas, navegar entre fechas y añadir nuevas sesiones a través del formulario mencionado anteriormente.

- **View-Models**: Encapsulan la lógica de negocio, gestionan el estado y facilitan la interacción entre los modelos y las vistas.

```dart
class AuthViewModel extends ChangeNotifier {
  final AuthService _authService = AuthService();
  User? get currentUser => _authService.currentUser;
  UserModel? get currentUserModel => _authService.currentUserModel;
  Stream<User?> get authStateChanges => _authService.authStateChanges;
  String? errorMessage;

  Future<UserModel?> getCurrentUser() async {
    return _authService.currentUserModel;
  }

  Future<void> signUp(String email, String password) async {
    try {
      String? userId = await _authService.register(email, password);
      if (userId == null) {
        errorMessage = 'Error en el registro';
      }
    } catch (e) {
      errorMessage = 'Error en el registro: $e';
    }
    notifyListeners();
  }

  Future<void> signIn(String email, String password) async {
    try {
      String? userId = await _authService.signInWithEmailAndPassword(email, password);
      if (userId == null) {
        errorMessage = 'Error en el inicio de sesión';
      }
    } catch (e) {
      errorMessage = 'Error en el inicio de sesión: $e';
    }
    notifyListeners();
  }

  Future<void> signOut() async {
    await _authService.signOut();
    notifyListeners();
  }
}
```

Figura 37. View-model del servicio de autenticación de la aplicación

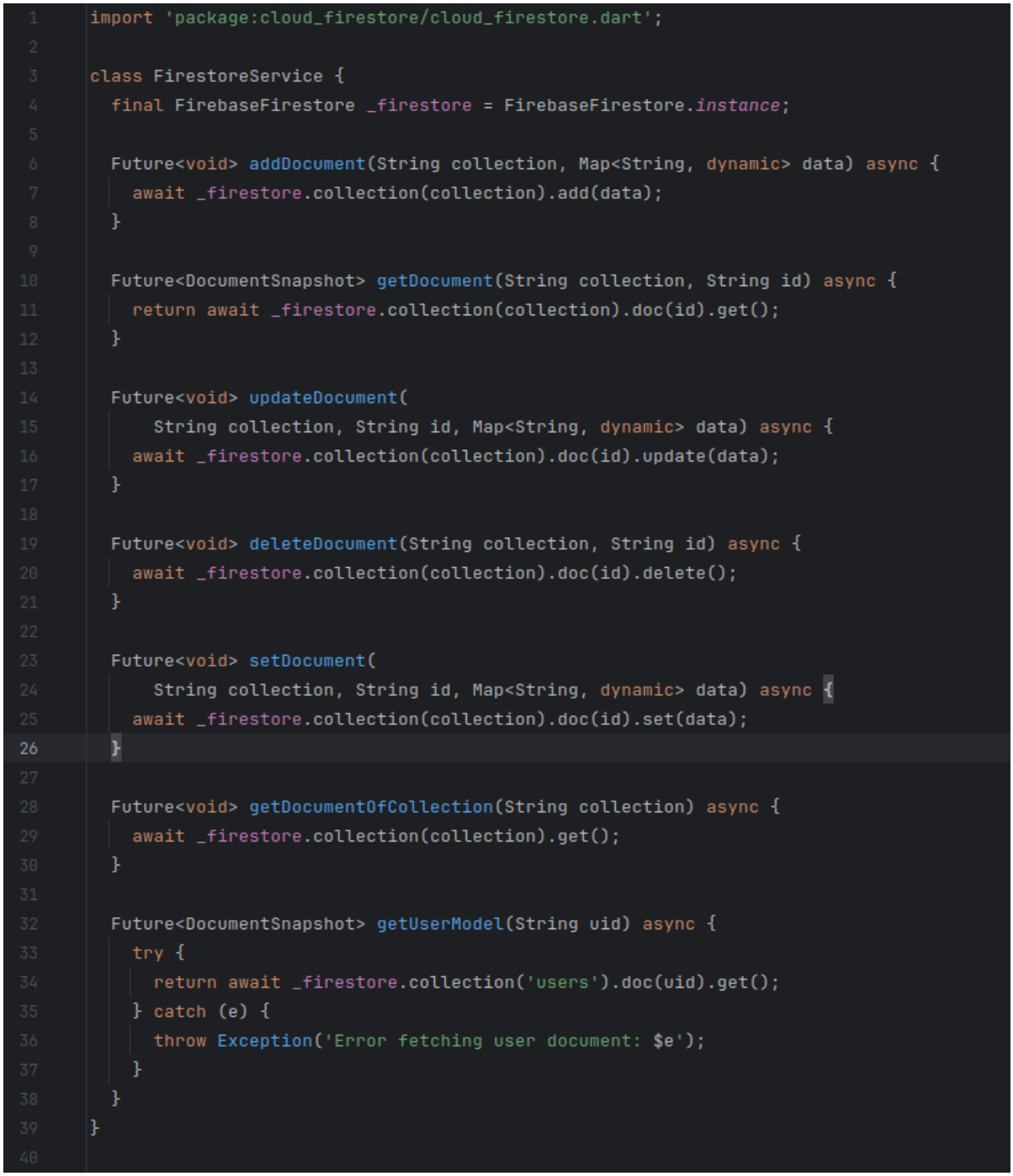
- **Servicios**: Manejan la comunicación con la API y la persistencia de datos.

```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';
import 'package:my_trainer/services/user_service.dart';

import '../models/user_model.dart';

class AuthService with ChangeNotifier {
  User? _user;
  UserModel? _userModel;
  final FirebaseAuth _auth = FirebaseAuth.instance;

  Stream<User?> get authStateChanges => _auth.authStateChanges();

  AuthService() {
    _auth.authStateChanges().listen(_onAuthStateChanged);
  }


  User? get currentUser => _user;
  UserModel? get currentUserModel => _userModel;

  Future<void> _onAuthStateChanged(User? user) async {
    _user = user;
    if (user != null) {
      UserService().getUser(user.uid).then((userModel) {
        _userModel = userModel;
      });
    }
    notifyListeners();
  }
```

Figura 38. Servicio de autenticación de la aplicación

```
31
32      Future<String?> register(String email, String password) async {
33        try {
34          UserCredential userCredential =
35              await _auth.createUserWithEmailAndPassword(
36            email: email,
37            password: password,
38          );
39          _user = userCredential.user;
40          notifyListeners();
41          return _user!.uid;
42        } catch (e) {
43          return null;
44        }
45      }
46
47      Future<String?> signInWithEmailAndPassword(
48          String email, String password) async {
49        try {
50          UserCredential userCredential = await _auth.signInWithEmailAndPassword(
51            email: email,
52            password: password,
53          );
54          _user = userCredential.user;
55          notifyListeners();
56          return _user!.uid;
57        } catch (e) {
58          return null;
59        }
60      }
61
62      Future<void> signOut() async {
63        await _auth.signOut();
64      }
65    }
66
```

Figura 39. Servicio de autenticación de la aplicación



```dart
import 'package:cloud_firestore/cloud_firestore.dart';

class FirestoreService {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<void> addDocument(String collection, Map<String, dynamic> data) async {
    await _firestore.collection(collection).add(data);
  }

  Future<DocumentSnapshot> getDocument(String collection, String id) async {
    return await _firestore.collection(collection).doc(id).get();
  }

  Future<void> updateDocument(
      String collection, String id, Map<String, dynamic> data) async {
    await _firestore.collection(collection).doc(id).update(data);
  }

  Future<void> deleteDocument(String collection, String id) async {
    await _firestore.collection(collection).doc(id).delete();
  }

  Future<void> setDocument(
      String collection, String id, Map<String, dynamic> data) async {
    await _firestore.collection(collection).doc(id).set(data);
  }

  Future<void> getDocumentOfCollection(String collection) async {
    await _firestore.collection(collection).get();
  }

  Future<DocumentSnapshot> getUserModel(String uid) async {
    try {
      return await _firestore.collection('users').doc(uid).get();
    } catch (e) {
      throw Exception('Error fetching user document: $e');
    }
  }
}
```

Figura 40. Código del servicio de Cloud Firestore

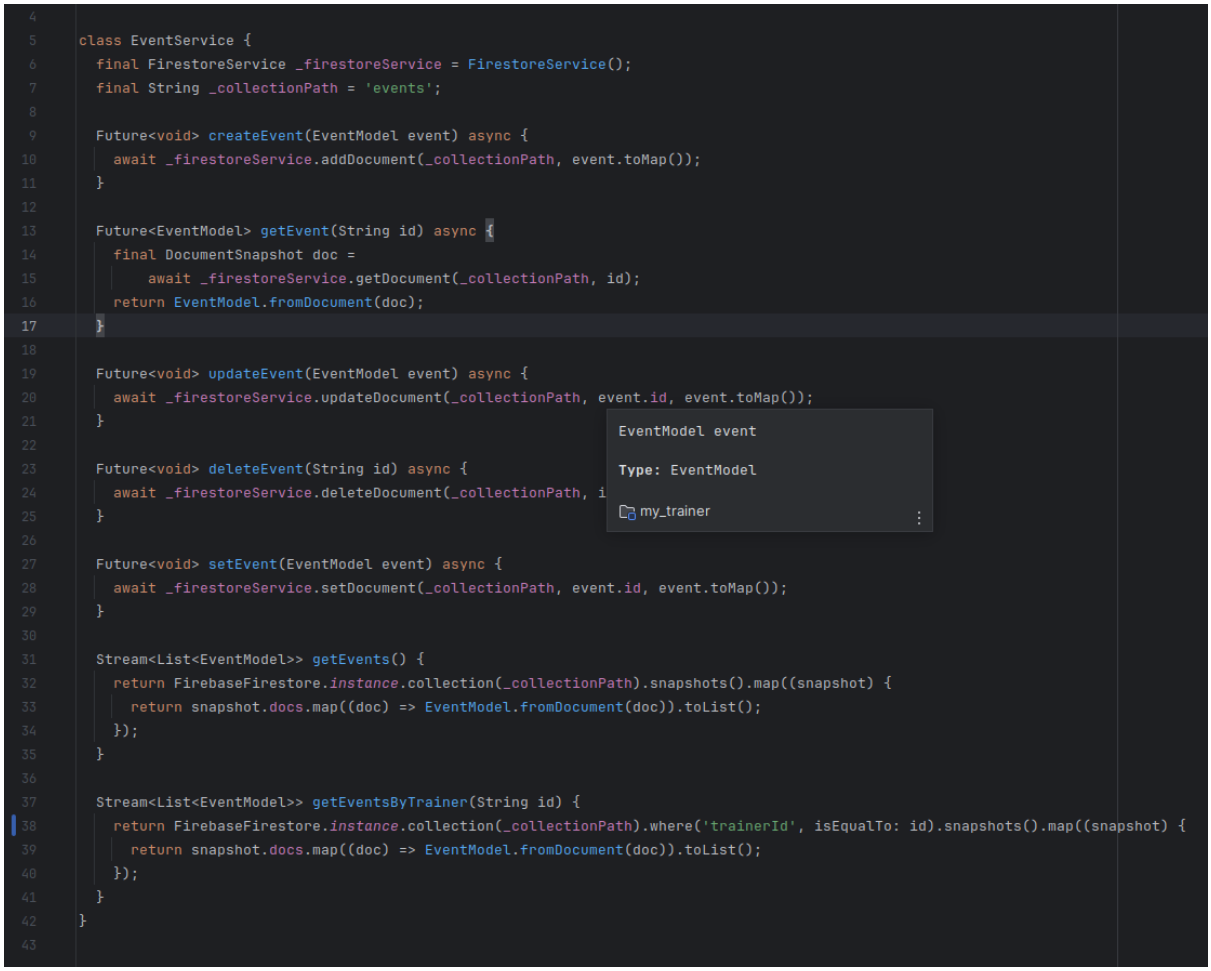
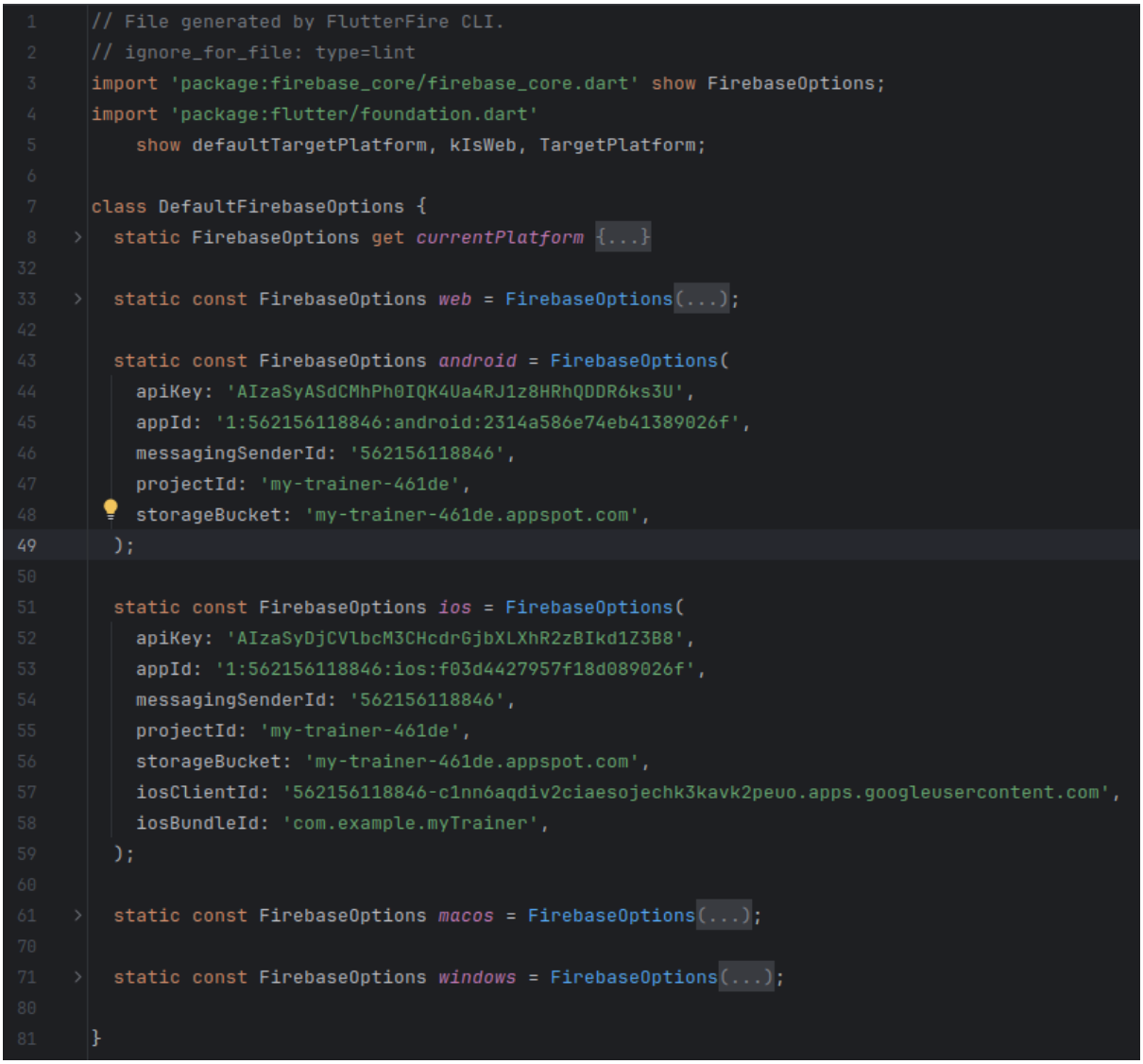


Figura 41. Código del servicio de Firestore relacionado con la colección "events"

El código de la aplicación móvil está organizado de manera modular para facilitar su mantenimiento, escalabilidad y futuros desarrollos. A continuación, se presenta la estructura principal de carpetas y archivos:

- */android, /ios, /linux, /macos, /web, /windows*: Carpetas generadas automáticamente por Flutter que contienen la configuración específica para cada plataforma, permitiendo compilar la aplicación de manera nativa en cada una de ellas, dentro de cada una de ellas es necesario integrar el archivo de configuración del proyecto de Firebase, es un archivo tal que así:



```dart
// File generated by FlutterFire CLI.
// ignore_for_file: type=lint
import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;

class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {...}

  static const FirebaseOptions web = FirebaseOptions(...);

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyASdCMhPh0IQK4Ua4RJ1z8HRhQDDR6ks3U',
    appId: '1:562156118846:android:2314a586e74eb41389026f',
    messagingSenderId: '562156118846',
    projectId: 'my-trainer-461de',
    storageBucket: 'my-trainer-461de.appspot.com',
  );

  static const FirebaseOptions ios = FirebaseOptions(
    apiKey: 'AIzaSyDjCVlbcM3CHcdrGjbXLXhR2zBIkd1Z3B8',
    appId: '1:562156118846:ios:f03d4427957f18d089026f',
    messagingSenderId: '562156118846',
    projectId: 'my-trainer-461de',
    storageBucket: 'my-trainer-461de.appspot.com',
    iosClientId: '562156118846-c1nn6aqdiv2ciaesojechk3kavk2peuo.apps.googleusercontent.com',
    iosBundleId: 'com.example.myTrainer',
  );

  static const FirebaseOptions macos = FirebaseOptions(...);

  static const FirebaseOptions windows = FirebaseOptions(...);

}
```

Figura 42. Archivo de configuración de Firebase

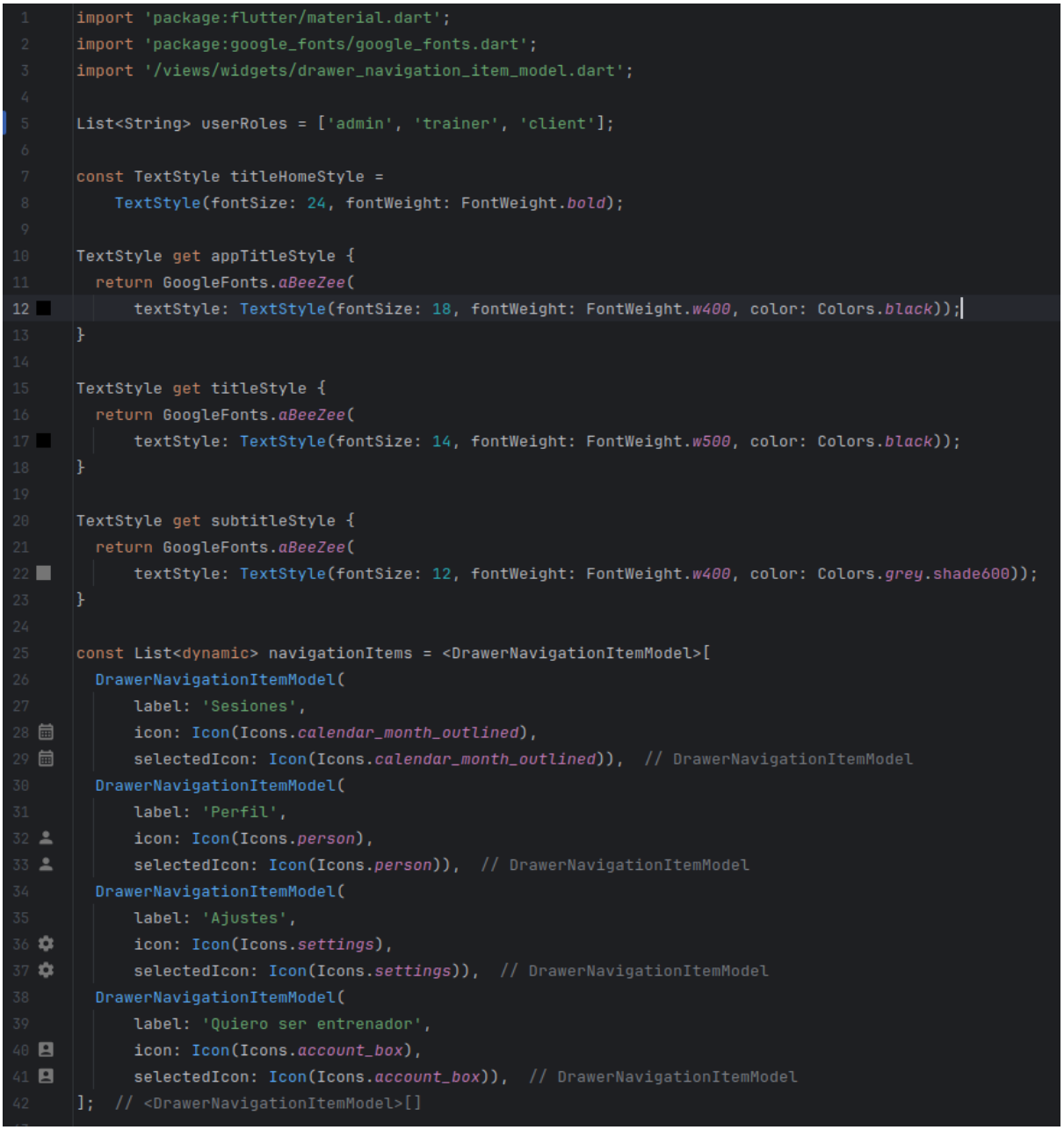
Figura 43. Archivo de configuración para Android de Firebase

- */assets*: Carpeta que contiene recursos estáticos como imágenes, iconos y otros archivos multimedia utilizados en la aplicación.
- */functions*: Carpeta que contiene las Cloud Functions de Firebase, utilizadas para la lógica de backend y la integración con servicios externos como Stripe.
- */test*: Carpeta destinada a las pruebas unitarias y de integración de la aplicación.
- *.gitignore*: Archivo de configuración que indica a Git qué archivos o directorios deben ser ignorados y no subidos al repositorio.
- *.metadata:* Archivo que contiene metadatos sobre el proyecto.
- README.md: Archivo que proporciona información sobre el proyecto, su propósito y cómo configurarlo y ejecutarlo.
- *analysis_options.yaml*: Archivo de configuración que define las reglas de análisis estático para el código Dart.
- *flutter_launcher_icons.yaml*: Archivo de configuración para personalizar los iconos de la aplicación en Flutter.

```yaml
flutter_launcher_icons:
  android: "launcher_icon"
  ios: true
  image_path: "assets/icon.png"
  min_sdk_android: 21 # android min sdk min:16, default 21
  web:
    generate: true
    image_path: "assets/icon.png"
    background_color: "#hexcode"
    theme_color: "#hexcode"
  windows:
    generate: true
    image_path: "assets/icon.png"
    icon_size: 48 # min:48, max:256, default: 48
  macos:
    generate: true
    image_path: "assets/icon.png"
```

Figura 44. Archivo de iconos de la aplicación

- *.env:* Archivo que contiene variables de entorno para configurar la aplicación, como claves API y otros parámetros de configuración sensibles.
- *pubspec.lock*: Archivo generado automáticamente que bloquea las versiones específicas de las dependencias utilizadas en el proyecto.
- *pubspec.yaml:* Archivo de configuración principal del proyecto Flutter, donde se especifican las dependencias y otra configuración del proyecto.
- */lib:* Carpeta principal donde reside el código fuente de la aplicación.
    - */models:* Contiene las definiciones de los modelos de datos que se utilizan en la aplicación. Estos modelos representan las entidades principales como entrenadores, clientes, sesiones y reservas.
    - */services:* Contiene los servicios que interactúan con Firebase y otros sistemas externos, como el servicio de notificaciones y la integración con Stripe
    - */utils:* Almacena utilidades y constantes que son utilizadas en diferentes partes de la aplicación, facilitando la reutilización del código



```dart
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import '/views/widgets/drawer_navigation_item_model.dart';

List<String> userRoles = ['admin', 'trainer', 'client'];

const TextStyle titleHomeStyle =
    TextStyle(fontSize: 24, fontWeight: FontWeight.bold);

TextStyle get appTitleStyle {
  return GoogleFonts.aBeeZee(
      textStyle: TextStyle(fontSize: 18, fontWeight: FontWeight.w400, color: Colors.black));
}

TextStyle get titleStyle {
  return GoogleFonts.aBeeZee(
      textStyle: TextStyle(fontSize: 14, fontWeight: FontWeight.w500, color: Colors.black));
}

TextStyle get subtitleStyle {
  return GoogleFonts.aBeeZee(
      textStyle: TextStyle(fontSize: 12, fontWeight: FontWeight.w400, color: Colors.grey.shade600));
}

const List<dynamic> navigationItems = <DrawerNavigationItemModel>[
  DrawerNavigationItemModel(
      label: 'Sesiones',
      icon: Icon(Icons.calendar_month_outlined),
      selectedIcon: Icon(Icons.calendar_month_outlined)),  // DrawerNavigationItemModel
  DrawerNavigationItemModel(
      label: 'Perfil',
      icon: Icon(Icons.person),
      selectedIcon: Icon(Icons.person)),  // DrawerNavigationItemModel
  DrawerNavigationItemModel(
      label: 'Ajustes',
      icon: Icon(Icons.settings),
      selectedIcon: Icon(Icons.settings)),  // DrawerNavigationItemModel
  DrawerNavigationItemModel(
      label: 'Quiero ser entrenador',
      icon: Icon(Icons.account_box),
      selectedIcon: Icon(Icons.account_box)),  // DrawerNavigationItemModel
];  // <DrawerNavigationItemModel>[]
```

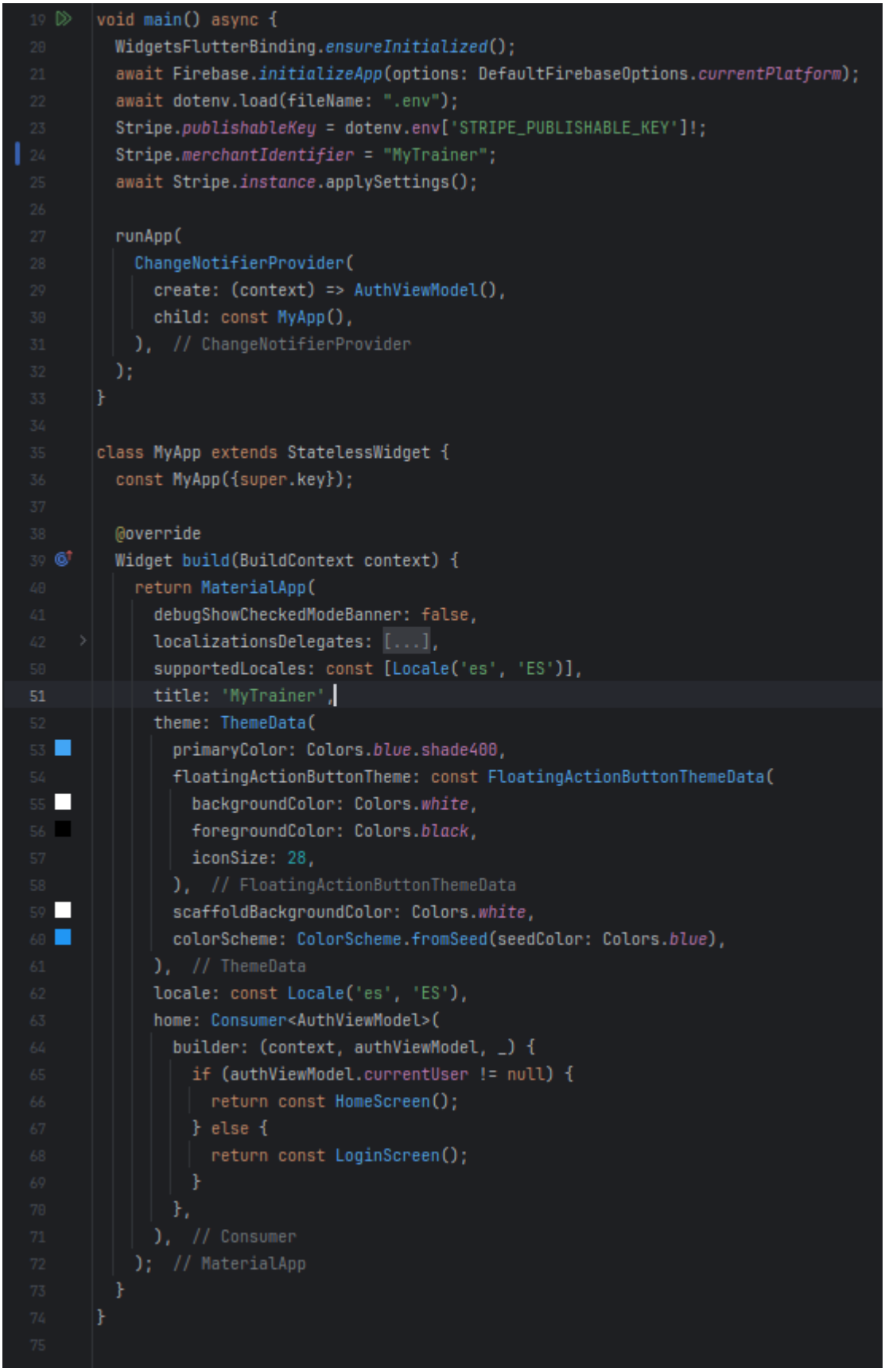
Figura 45. Código del archivo "constants" de la aplicación

Aquí se definen los estilos de texto generales y los elementos de navegación que utilizará el *drawer* dentro de la aplicación, los estilos de textos están personalizados utilizando Google Fonts para mantener una apariencia profesional. Los elementos de navegación proporcionan opciones para navegar entre las distintas pantallas de la aplicación.

```dart
import 'dart:math';

import 'package:uuid/uuid.dart';

String generateUniqueUid(String baseUid) {
  var uuid = Uuid();
  return uuid.v5(Uuid.NAMESPACE_URL, baseUid);
}

String generateUniqueInvitationCode() {
  const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
  final random = Random();
  return String.fromCharCodes(
    Iterable.generate(6, (_) => chars.codeUnitAt(random.nextInt(chars.length))),
  );  // String.fromCharCodes
}
```

Figura 46. Código del archivo "functions" de la aplicación

- */views_models:* Incluye los proveedores que gestionan el estado de la aplicación y la lógica de negocio relacionada con los datos. Utilizamos el patrón Provider para la gestión del estado.
- */views:* Agrupa las pantallas o vistas principales de la aplicación. Cada archivo representa una pantalla específica, como el calendario, la gestión de sesiones y las reservas.
- *main.dart:* Archivo principal de la aplicación que inicializa y configura las diferentes partes del proyecto.



```dart
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
  await dotenv.load(fileName: ".env");
  Stripe.publishableKey = dotenv.env['STRIPE_PUBLISHABLE_KEY']!;
  Stripe.merchantIdentifier = "MyTrainer";
  await Stripe.instance.applySettings();

  runApp(
    ChangeNotifierProvider(
      create: (context) => AuthViewModel(),
      child: const MyApp(),
    ),  // ChangeNotifierProvider
  );
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      localizationsDelegates: [...],
      supportedLocales: const [Locale('es', 'ES')],
      title: 'MyTrainer',
      theme: ThemeData(
        primaryColor: Colors.blue.shade400,
        floatingActionButtonTheme: const FloatingActionButtonThemeData(
          backgroundColor: Colors.white,
          foregroundColor: Colors.black,
          iconSize: 28,
        ),  // FloatingActionButtonThemeData
        scaffoldBackgroundColor: Colors.white,
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue),
      ),  // ThemeData
      locale: const Locale('es', 'ES'),
      home: Consumer<AuthViewModel>(
        builder: (context, authViewModel, _) {
          if (authViewModel.currentUser != null) {
            return const HomeScreen();
          } else {
            return const LoginScreen();
          }
        },
      ),  // Consumer
    );  // MaterialApp
  }
}
```

Figura 47. Archivo main.dart de la aplicación.

8.3 Firebase

Firebase proporciona una plataforma de backend que facilita la autenticación *(Firebase Authentication),* almacenamiento en tiempo real *(Firestore Database)*, notificaciones push, funciones en la nube, etc. A continuación, se describen los servicios utilizados de Firebase por *"MyTrainer"*.

8.3.1 Firebase Authentication

Es el encargado de gestionar el registro e inicio de sesión de usuarios. Permite también a los usuarios autenticarse a través de distintos proveedores, en nuestro caso, puedo hacerlo a través de Google Sign-In.

Posteriormente, observaremos como al registrar un usuario y este, al iniciar sesión por primera vez, creamos la misma instancia del usuario tanto en la Firebase Database para almacenar distintos datos que necesitamos, y a su vez, también creamos al usuario a través de la API de Stripe como cliente, esto le permitirá en el caso del cliente pagar por sus servicios, y en el del entrenador, pues crear sus distintas subscripciones que se representarán como productos en Stripe.

8.3.2 Cloud Firestore

Cloud Firestore [18] es una base de datos flexible y escalable para desarrollo móvil, web y de servidores de Firebase y Google Cloud. Al igual que Firebase Realtime Database, mantiene sus datos sincronizados en las en tiempo real y ofrece soporte sin conexión para dispositivos móviles y web para que pueda crear aplicaciones que funcionen independientemente de la red o la conexión a Internet, este tipo de base de datos nos permitirá ofrecer una mejor experiencia al usuario, tanto al cliente como al entrenador al poder sincronizar ya sea eventos o subscripciones en tiempo real.

Al tratarse de una base de datos no relacional presenta ciertas características para tener en cuenta:

- Modelo de datos basado en documentos
    - Firestore utilizado un tipo de modelo de datos donde considera que las tablas son "colecciones" y en ellas se almacenan "documentos". Cada

documento es un diccionario de clave-valor y puede contener datos anidados.

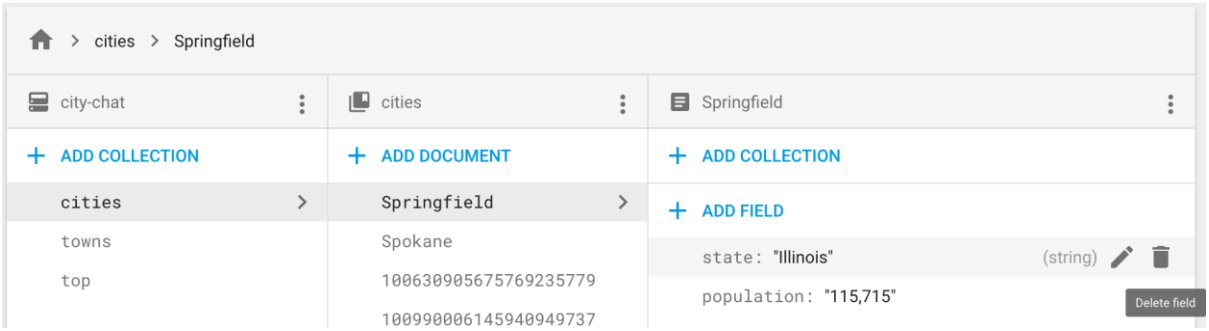


Figura 49. Ejemplo de colección en Firestore

- Flexibilidad de estructura
    - Los documentos en Firestore pueden tener estructuras diferentes y no es necesario definirlas antes de insertarlas. Esto permite que haya documentos en una misma colección que tengan campos distintos lo cual no es posible en una base de datos relacional.
- Consulta y almacenamiento en tiempo real
    - Firestore ofrece sincronización en tiempo real, esto significa que los cambios que se produzcan en los datos se reflejarán instantáneamente en todas las aplicaciones que estén conectadas, como habíamos mencionado anteriormente, esto nos permite mejorar la experiencia del usuario a través de los eventos y subscripciones.
- Desacoplamiento de datos
    - Los documentos no requieren de relaciones entre ellos, aunque podemos realizar búsquedas y consultas anidadas, no es necesario definir ningún tipo de relación entre las distintas colecciones como se haría en una base de datos relacional.

Respecto a la estructura de la base de datos, se muestra en la siguiente tabla:

Colección	Documento	Campos
users	userId	availableSlots*: Número de clientes disponibles para asignar.
		clientsIds[]*: Conjunto de datos donde se almacenan los userId de cada cliente.
		invitationCode*: Código de invitación de un entrenador.
		profilePictureUrl: URL donde se muestra la foto de perfil del usuario.
		registrationDate: Fecha de registro.
		role: Rol de usuario, posibles valores: “client”, “trainer”.
		sessionIds[]: Colección de los sessionIds de las sesiones creadas por el entrenador.
		trainerId*: ID del entrenador para identificar sus sesiones.
		uid: Tiene el mismo valor que userId.
		subscriptionStatus: “active” o “inactive”
		fcmToken: Puede ser una colección o un valor, y representa el token de dispositivo por el cual ha iniciado sesión.
		stripeId: ID del cliente en Stripe.
events	eventId	capacity: Capacidad iniciar y por asignar de la sesión
		date: Fecha de la sesión
		description: Descripción de la sesión
		endTime: Fecha hora de fin de la sesión
		startTime: Fecha hora del inicio de la sesión
		id: eventId

		sessionType: Información de tipo de sesión
		studentIds[]: Colección de los IDs de los clientes asignados a la sesión
		title: Título de la sesión
		trainerId: ID del entrenador que ha creado la sesión
products	productId	creationDate: Fecha de creación del producto
		currency: Divisa del precio
		description: Descripción del producto
		id: productId
		name: Nombre del producto
		price: Precio del producto
		priceId: ID del precio creado en Stripe
		sessionsPerWeek: Nº de sesiones disponibles del producto
		stripeId: ID del producto creado en Stripe
		trainerId: ID del entrenador que ha creado el producto.
payments	paymentId	id: ID del pago
		userId: ID del usuario que ha realizado el pago
		amount: Precio del pago realizado
		currency: Divisa del pago
		status: Estado del pago
		timestamp: Fecha de creación del pago
		productId: Id del producto
subscriptions	subId	id: ID de la subscripción
		userId: ID del usuario
		productId: ID del producto
		startDate: Fecha de inicio de la subscripción
		endDate: Fecha de fin de la subscripción
		status: Estado de la subscripción

Tabla 8. Estructura de la base de datos **Solo para entrenadores personales.*

Las colecciones están interrelacionadas para mantener la coherencia y facilitar las consultas. A continuación, se detalla cómo se relacionan las colecciones principales:

- **users** y **events**: Cada evento tiene un **trainerId** que se refiere al **userId** del entrenador que creó el evento.
- **users** y **products**: Cada producto tiene un **trainerId** que se refiere al **userId** del entrenador que ofrece el producto.
- **users** y **payments**: Cada pago tiene un **userId** que se refiere al cliente que realizó el pago.
- **users** y **subscriptions**: Cada suscripción tiene un **userId** que se refiere al cliente suscrito y un productId que se refiere al producto adquirido.
- **products** y **subscriptions**: Cada suscripción tiene un **productId** que se refiere al producto asociado a la suscripción.
- **events** y **subscriptions**: Implícitamente, los eventos que un cliente puede reservar están relacionados con sus suscripciones activas.

Para asegurar los datos en Cloud Firestore, se implementan reglas de seguridad que restringen el acceso a los datos según el rol del usuario y otras condiciones específicas

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {

    match /users/{userId} {
      allow read: if request.auth.uid == userId;
      allow write: if request.auth.uid == userId;
    }

    match /events/{eventId} {
      allow read: if request.auth != null;
      allow write: if request.auth.token.role == 'trainer' && request.auth.uid == resource.data.trainerId;
    }

    match /products/{productId} {
      allow read: if request.auth != null;
      allow write: if request.auth.token.role == 'trainer' && request.auth.uid == resource.data.trainerId;
    }

    match /payments/{paymentId} {
      allow read, write: if request.auth.uid == resource.data.userId;
    }
  }
```

Figura 50. Reglas de seguridad de Cloud Firestore

8.3.3 Firebase Cloud Messaging (*FCM*)

FCM nos permite enviar notificaciones push a los usuarios. Se utiliza para notificar a los clientes sobre sesiones próximas o actualizaciones importantes.

Con FCM [19], puede notificar a una aplicación cliente que hay nuevos correos electrónicos u otros datos disponibles para sincronizar. Puede enviar mensajes de notificación para impulsar la reincorporación y retención de los usuarios

La configuración de FCM incluye la integración con el backend para enviar mensajes y notificaciones.

Para la integración de FCM se ha seguido los siguientes pasos:

1. Habilitar FCM en el proyecto Firebase
2. Al igual que todos los servicios de Firebase, hay que incluir el archivo *google-services.json* (para Android) y *GoogleServiceInfo.plist* (para iOS).
3. Agregamos las dependencias necesarias, que en este caso es *firebase_messaging*.
4. Se inicializa Firebase en el archivo *main.dart,* mencionado anteriormente en la estructura del proyecto.
5. Se configura FCM para manejar la recepción de mensajes en primer y segundo plano, para ello creamos un servicio que al inicializar pregunte por los permisos para las notificaciones al dispositivo, si el usuario las autoriza, el servicio queda establecido.
6. Este servicio contará con la inicialización y obtención del token FCM, la actualización de dicho token, el almacenamiento del token en base de datos, el envío de notificaciones a dispositivos, una gestión para los tokens que sean inválidos, y subscripciones a temas, que, en nuestro caso, será a entrenadores.
7. Ahora cada vez que un usuario se inicie en la aplicación, podemos obtener su token de FCM y almacenarlo en la base de datos para su posterior uso, hay que tener en cuenta que este token puede cambiar y por lo tanto siempre es importante actualizarlo cada vez que cambie.

Dicho servicio nos permitirá tanto enviar notificaciones a usuarios concretos a través de sus dispositivos, como poder enviar notificaciones a temas, que, en nuestro caso, crearemos un tema por cada entrenador creado en la aplicación, y cada vez que un usuario se asigne a un entrenador, este también se asignará a dicho tema.

8.4 Stripe

Stripe [20] es una plataforma de pagos en línea que permite a las empresas aceptar pagos, gestionar suscripciones y realizar transferencias de dinero a nivel mundial. Stripe se destaca por su robusta API, que facilita la integración de diversas funcionalidades de pago en aplicaciones móviles y web.

Esto son los principales componentes de Stripe:

- **Dashboard**: Una interfaz gráfica donde los administradores pueden ver y gestionar transacciones, clientes, suscripciones y disputas.
- **Stripe API**: Una interfaz de programación de aplicaciones que permite a los desarrolladores interactuar programáticamente con los servicios de Stripe.
- **Webhooks**: Mecanismo de comunicación que permite a Stripe enviar notificaciones en tiempo real a tu aplicación cuando ocurren eventos importantes (por ejemplo, pagos completados, suscripciones canceladas).

Estas son las principales funcionalidades que te permite la API de Stripe:

- **Gestión de Pagos**: Creación y gestión de intenciones de pago y pagos reales.
- **Gestión de Suscripciones**: Creación y administración de planes de suscripción y suscriptores.
- **Gestión de Clientes**: Creación y actualización de perfiles de clientes.
- **Gestión de Productos y Precios**: Creación y actualización de productos y precios.

Para realizar dicha integración en la aplicación se han tenido en cuenta los siguientes pasos:

1. Primero, hay que crear una cuenta en Stripe y configurarla.
2. Stripe proporciona dos tipos de claves para utilizar y autenticar las solicitudes que se le envía a la API, una de ellas son claves de pruebas que sirven para el

entorno de desarrollo y otras que son claves en vivos que sirven para entornos de producción.

3. Al registrar un nuevo usuario en la aplicación, se crea automáticamente un perfil de cliente en Stripe, de esta manera podemos gestionar los pagos y las subscripciones de dicho usuario, al crearse dicho cliente Stripe devuelve una respuesta con un ID del cliente creado, este dato también lo almacenaremos en base de datos.
4. Antes de que un cliente pueda realizar un pago de una sesión o de una subscripción, dicho pago o producto tiene que estar creado previamente en Stripe. Cuando un cliente cree una intención de pago, Stripe se asegura que los fondos estén disponibles y que el pago esté autorizado.
5. Los entrenadores pueden gestionar y ofrecer subscripciones de sus sesiones a sus clientes, con esto, el cliente puede seleccionar un plan de subscripción, y enviar una solicitud a Stripe para la creación de la subscripción.
6. Stripe enviará notificaciones cuando ocurran eventos importantes relacionados con el usuario, ya sea pagos completados o subscripciones canceladas.

La integración con Stripe nos permite proporcionar a los usuarios una experiencia de pago fluidas y segunda, permitiendo a los entrenadores y clientes enfocarse en lo más importantes, qué es el entrenamiento.

9. Pruebas unitarias

Una de las fases de la planificación del desarrollo del proyecto consiste en el empleo de pruebas unitarias que permiten verificar y ofrecer un código de calidad.

En nuestro caso, las pruebas unitarias son pruebas que automatizamos para que verifiquen el funcionamiento de código, como pueden ser funciones dentro de un servicio, construcciones de widgets, etc.

Gracias a las pruebas unitarias podemos detectar errores que vayan surgiendo con el desarrollo, así como para verificar que en efecto se ha realizado correctamente la función o el módulo que se pretendía implementar.

También nos permite mejorar la calidad del código y facilitar su futuro mantenimiento, básicamente son esenciales y necesarios para cualquier desarrollo de calidad de cualquier proyecto.

Para realizar las pruebas unitarias, al tratarse del framework Flutter, podemos utilizar herramientas como *flutter_test* o *mockito*.

9.1 Pruebas de Widgets

Los widgets son la base de la interfaz de usuario en Flutter, por lo tanto, hemos realizado pruebas unitarias para verificar que los widgets se construyen correctamente y realizar las funciones e interactúan de la manera que se desea.

Por lo tanto, nuestro objetivo es el de asegurar que cada widget no tiene ningún problema de renderizado (*RenderError*) y se comporte de la manera adecuada.

Utilizando *flutter_test,* se han escrito las pruebas *(figura 51)* para comprobar que los widgets renderizan el contenido deseado y responden a las acciones del usuario.

```dart
class MockUserService extends Mock implements UserService {}
class MockAuthViewModel extends Mock implements AuthViewModel {}

void main() {
  late MockUserService mockUserService;
  late MockAuthViewModel mockAuthViewModel;

  setUp(() {
    mockUserService = MockUserService();
    mockAuthViewModel = MockAuthViewModel();
  });

  Widget createWidgetUnderTest() {
    return MultiProvider(
      providers: [
        Provider<UserService>(create: (_) => mockUserService),
        ChangeNotifierProvider<AuthViewModel>(create: (_) => mockAuthViewModel),
      ],
      child: MaterialApp(
        home: UsersScreen(),
      ),  // MaterialApp
    );  // MultiProvider
  }

  testWidgets('UsersScreen renders correctly', (WidgetTester tester) async {
    // Arrange
    when(mockAuthViewModel.getCurrentUser())
        .thenAnswer((_) async => UserModel(trainerId: 'trainer1', uid: '', email: '', s
    when(mockUserService.getUsersByTrainer('trainer1'))
        .thenAnswer((_) => Stream.value([
      UserModel(displayName: 'User1', email: 'user1@example.com', uid: '', stripeCustom
      UserModel(displayName: 'User2', email: 'user2@example.com', uid: '', stripeCustom
    ]));  // Stream.value

    // Act
    await tester.pumpWidget(createWidgetUnderTest());
    await tester.pumpAndSettle();

    // Assert
    expect(find.text('Clientes'), findsOneWidget);
    expect(find.byType(ListTile), findsNWidgets(2));
    expect(find.text('User1'), findsOneWidget);
    expect(find.text('user1@example.com'), findsOneWidget);
    expect(find.text('User2'), findsOneWidget);
    expect(find.text('user2@example.com'), findsOneWidget);
  });
```

Figura 51. Código de pruebas unitarias de widgets.

9.2 Pruebas de servicios con Firebase

En este proyecto, se utilizaron varios servicios de Firebase, incluyendo Cloud Firestore para el almacenamiento de datos y Firebase Authentication para la gestión de Usuarios, así como Cloud Messaging para el envío de notificaciones.

En este caso, nuestro objetivo es asegurar que los servicios de Firebase funcionan correctamente tanto la base de datos, la autenticación de los usuarios como el envío de notificaciones.

Para el desarrollo de las pruebas *(figura 52)*, se han creado *mocks* para simular el comportamiento de dichos servicios, con *mockito*, se crearon objetos de los servicios, permitiendo realizar las pruebas sin necesidad de acceder al entorno real de Firebase.

```
class MockFirestoreService extends Mock implements FirestoreService {
  void main() {
    const collection = 'tests';
    group('FirestoreService', () {
      test('addDocument', () async {
        final firestoreService = FirestoreService();
        final mockFirestore = MockFirestoreService();
        when(mockFirestore.addDocument(collection, {'id': 'id'})).thenAnswer((_) async {});
        await firestoreService.addDocument('collection', {});
        verify(mockFirestore.addDocument('collection', {}));
      });

      test('getDocument', () async {
        final firestoreService = FirestoreService();        final mockFirestore = MockFirestoreService();
        when(mockFirestore.getDoc(collection, 'id')).thenAnswer((_) async {});
        await firestoreService.getDocument('collection', 'id');
        verify(mockFirestore.getDocument('collection', 'id'));
      });

      test('updateDocument', () async {
        final firestoreService = FirestoreService();
        final mockFirestore = MockFirestoreService();
        when(mockFirestore.updateDocument(collection, 'id', {'id': 'id'})).thenAnswer((_) async {});
        await firestoreService.updateDocument('collection', 'id', {});
        verify(mockFirestore.updateDocument('collection', 'id', {}));
      });

      test('deleteDocument', () async {
        final firestoreService = FirestoreService();
        final mockFirestore = MockFirestoreService();
        when(mockFirestore.deleteDocument(collection, 'id')).thenAnswer((_) async {});
        await firestoreService.deleteDocument('collection', 'id');
        verify(mockFirestore.deleteDocument('collection', 'id'));
      });

      test('setDocument', () async {
        final firestoreService = FirestoreService();
        final mockFirestore = MockFirestoreService();
        when(mockFirestore.setDocument(collection, 'id', {'id': 'id'})).thenAnswer((_) async {});
        await firestoreService.setDocument('collection', 'id', {});
        verify(mockFirestore.setDocument('collection', 'id', {}));
      });

      test('getDocumentOfCollection', () async {
        final firestoreService = FirestoreService();
        final mockFirestore = MockFirestoreService();
        when(mockFirestore.getDocumentOfCollection(collection)).thenAnswer((_) async {});
        await firestoreService.getDocumentOfCollection('collection');
        verify(mockFirestore.getDocumentOfCollection('collection'));
      });
    });
  }
}
```

Figura 52. Código de pruebas de integración con Firebase

9.3 Pruebas de integración con Stripe

La integración de Stripe en la aplicación permite a los cliente y entrenadores realizar y gestionar pagos y subscripciones de manera segura.

Nuestro objetivo será que los procesos de pagos y la validación de las tarjetas de crédito funcionen correctamente y sin errores.

Esto se ha desarrollado al igual que manera que las pruebas realizadas a los servicios de Firebase, se ha utilizado *mocks* para recrear el comportamiento que tiene el servicio de Stripe, y con *mockito* se crearon objetos que permiten realizar estas pruebas y simular los flujos de pagos en transacciones que no son reales.

10. Aportaciones

10.1 Entorno socioeconómico

El objetivo principal del proyecto es el de solucionar un problema de comunicación que existe entre los entrenadores personales y sus clientes, un problema que cada vez es más común puesto que el mundo del fitness está en constante evolución y ya el número de personas que está involucradas en cierta manera con este ámbito crece año tras año.

Hemos alcanzado la idea y la solución que habíamos planeado desde un principio, y podemos afirmar que hemos desarrollado una aplicación que mejora la comunicación y gestión entre los clientes y entrenadores personales, esta mejora la conseguimos al ofrecer a los usuarios una aplicación sencilla y limpia que conste de un concepto básico, pero a la vez funcional qué es el de reservar una sesión o programar un calendario de sesiones con su entrenador personal.

Como hemos visto, al principio de la memoria, hay numerosas opciones que cumplen en cierta parte la solución que buscamos, pero ninguna de ellas es tan concreta como la nuestra, por lo tanto, podemos destacar que hemos creado un producto innovador que ha cumplido los requisitos tan específicos con los cual se ha planeado.

10.2 Entorno personal

En cuanto a lo personal, la motivación de este proyecto fue en principio una idea que me comentó mi tutor, al tratarse de una idea que está relacionado en cierta parte con una de mis mayores aficiones que es hacer deporte, y en concreto, ir al gimnasio, pues me resultaba apasionado y gratificante la idea de poder desarrollar una aplicación así.

Estoy contento porque creo que hemos podido alcanzar el objetivo que nos habíamos propuesto, además, de lo aprendido y reforzado durante el desarrollo, sobre todo con Flutter, qué es un framework que utilizo en mi puesto laboral y con la integración que he realizado con la nube, qué también es algo bastante común en mi empleo y por el cual siempre está bien aprender y reforzar conocimientos.

11. Resultados, conclusiones y futuras mejoras.

11.1 Resultado y conclusiones

Tras la finalización del proyecto, podemos decir que hemos conseguido los objetivos que nos planteamos desde un principio, desarrollando así una aplicación que ofrezca una solución a la gestión de entrenadores personales.

Ahora, los clientes y entrenadores personales poseen una aplicación que puede:

- Planear un horario de sesiones introduciendo datos informativos, como el tipo, duración, descripción, capacidad, etc.
- Disponer de un calendario donde se muestren todas las sesiones disponibles de un entrenador y que el cliente que esté asignado a él pueda reservarla haciendo uso de este.
- Servir para notificar a los usuarios de próximas sesiones, cambios de horas, sesiones canceladas, etc., en tiempo real.
- Permitir a un entrenador la creación de productos y subscripciones en una plataforma de pagos como puede Stripe.

Además, hemos podido asegurar los siguientes puntos:

- **Seguridad y fiabilidad**: La integración con Firebase y Stripe nos asegura que tantos los datos del usuario como las transacciones económicas se manejan con alto estándares de seguridad, lo que genera confianza en los usuarios
- **Facilidad de uso**: Gracias al diseño realizado con Material Design 3, hemos creado una interfaz de usuario intuitiva en cuanto a su navegación y uso.
- **Flexibilidad y adaptabilidad**: La aplicación está diseñada modularmente de manera que puede adaptarse a diferentes necesidades en el ámbito del fitness, e implementar dichas funcionalidades sin afectar al diseño original del sistema.
- **Beneficio para los usuarios**: Los entrenadores pueden gestionar a sus clientes de una manera más efectiva, optimizando el uso de su tiempo a través del módulo de calendario dándoles así una mayor comodidad.

Hemos demostrado ser un proyecto exitoso en términos de diseño, implementación y funcionalidad. Al utilizar arquitecturas modernas y tecnologías avanzadas como el

desarrollo en la nube, se ha creado una aplicación que no solo satisface las necesidades iniciales, sino que, además, de una aplicación que tiene un potencial de evolución y mejora impresionante.

11.2 Futuras mejoras

La aplicación que se ha desarrollado tiene una infinidad de funcionalidades que se pueden implementar y serían muy interesantes para el usuario, de hecho, el desarrollo modular nos permite crear un contexto de negocio en el cual el usuario sea similar en cuanto a datos en distintos módulos de las aplicaciones y así ofrecerle distintos servicios.

Algunos de los nuevos módulos que se podían implementar son:

- Módulo de análisis y estadísticas, con el que proveer a los usuarios con insights detallados sobre su rendimiento y progreso
- Módulo de comunicación, que incluiría un chat de mensajería instantánea similar a WhatsApp en el que se pudieran hacer videollamadas, y la creación de foros y grupos, donde compartir ideas y experiencias.
- Módulo de dietas y nutrición, para complementar los planes de entrenamiento.
- Módulo de evaluación física, para permitir que el entrenador pueda evaluar a su cliente en base a su condición física y tener un registro de su evolución
- Módulo de fidelidad, para aumentar la motivación y el compromiso del cliente con el entrenador.
- Módulo de Marketplace, donde se pueda ofrecer un espacio para la venta y compra de productos y servicios relacionados con el fitness.

12. Bibliografía

[1] Software gratuito de programación de citas en línea. Setmore. Consultado en julio de 2024. https://www.setmore.com/es

[2] App para Profesionales Fitness & Salud. Harbiz. Consultado en julio de 2024. https://harbiz.io/

[3] Software de gestión y reservas. Wodbuster. Consultado en julio de 2024. https://wodbuster.com/

[4] Wikipedia. (s.f.). Licencia MIT. Consultado en julio de 2024, de https://es.wikipedia.org/wiki/Licencia_MIT

[5] Wikipedia. (s.f.). GNU General Public License. Consultado en julio de 2024, de https://es.wikipedia.org/wiki/GNU_General_Public_License

[6] Wikipedia. (s.f.). Apache License. Consultado en julio de 2024, de https://es.wikipedia.org/wiki/Apache_License

[7] Wikipedia. (s.f.). Licencia BSD. Consultado en julio de 2024, de https://es.wikipedia.org/wiki/Licencia_BSD

[8] Wikipedia. (s.f.). Microsoft 365. Consultado en julio de 2024, de https://es.wikipedia.org/wiki/Microsoft_365

[9] Wikipedia. (s.f.). Creative Commons. Consultado en julio de 2024, de https://es.wikipedia.org/wiki/Creative_Commons

[10] Wikipedia. (s.f.). GitHub. Consultado en julio de 2024, de https://es.wikipedia.org/wiki/GitHub

[11] Wikipedia. (s.f.). Git. Consultado el 2 de julio de 2024, de https://es.wikipedia.org/wiki/Git

[12] Wikipedia. (s.f.). Microsoft Word. Consultado el 2 de julio de 2024, de https://es.wikipedia.org/wiki/Microsoft_Word

[13] Wikipedia. (s.f.). Flutter. Consultado el 2 de julio de 2024, de https://es.wikipedia.org/wiki/Flutter_(software)

[14] «Technical Overview - Flutter». flutter.io. Consultado el 13 de diciembre de 2017.

https://flutter.io/technical-overview/

[15] «Foundation library - Dart API». docs.flutter.io (en inglés). Archivado desde el original el 13 de diciembre de 2017. Consultado el 13 de diciembre de 2017.

https://web.archive.org/web/20171213143153/https://docs.flutter.io/flutter/foundation/foundation-library.html

[16] «Material Design Widgets - Flutter». flutter.io (en inglés). Consultado el 13 de diciembre de 2017.

https://docs.flutter.dev/ui/widgets/material

[17] «Cupertino (iOS-style) Widgets - Flutter». flutter.io (en inglés). Consultado el 13 de diciembre de 2017.

https://docs.flutter.dev/ui/widgets/cupertino

[18] «Cloud Firestore». Firebase Google. Consultado el 4 de julio de 2024.

https://firebase.google.com/docs/firestore?hl=es

[19] «Firebase Cloud Messaging». Firebase Google. Consultado el 4 de julio de 2024.

https://firebase.google.com/docs/cloud-messaging?hl=es

[20] “Stripe”. Stripe. Consultado el 4 de julio de 2024.

https://stripe.com/es