Universidad Zaragoza
1542

Trabajo Fin de Grado

Aplicación para el Control de la Asistencia COVID

COVID Attendance Control App

Autor

D. Javier Ramos Marco

Directora

Dª. Piedad Garrido Picazo

Escuela Universitaria Politécnica de Teruel

2022

Tabla de contenidos

Índice de Figuras

Resumen y Palabras Clave

En este documento se expone el desarrollo del Trabajo Fin de Grado de Ingeniería Informática (GII) con título “Aplicación para el control de la asistencia COVID” de la Escuela Universitaria Politécnica de Teruel (EUPT)
Este TFG ha consistido en la realización de una aplicación móvil multiplataforma mediante el uso del Framework UI Flutter y del lenguaje de programación Dart desarrollados por Google. Esta APP facilita a los docentes de la EUPT el control de la asistencia de sus alumnos a cada una de sus asignaturas de manera sencilla y rápida mediante la subida a través de un Excel del listado de alumnos para cada asignatura, seleccionando posteriormente los días que se tiene clase y de esta forma poder pasar lista en cada sesión cómodamente.
Finalmente se tiene la posibilidad de generar un PDF mensual con las asistencias de todos los alumnos, el cual es requerido que los docentes firmen. Para guardar los datos de la aplicación se ha utilizado Cloud Firestore y la autenticación de usuarios se ha realizado mediante el servicio Authenticacion proporcionado también por Firebase. Aparte, se ha utilizado Bloc como gestor de estados para la aplicación y controlar así las interacciones de los usuario con la misma.
Esta memoria consta de ocho apartados más un conjunto de anexos que se enumeran a continuación: Introducción y Objetivos, Estado del Arte, Análisis, Diseño, Tecnologías utilizadas, Pruebas, Licencia Software y Documental y un último apartado de Conclusiones y Trabajo Futuro.

Palabras Clave

Flutter, Aplicación móvil, Framework, Firebase, Bloc

Summary and Key Words

This document presents the development of the Computer Engineering Final Degree Project (GII) entitled " COVID Attendance Control App" of the Polytechnic University School of Teruel (EUPT).
This TFG has consisted in the development of a multiplatform mobile application using the UI Flutter Framework and the Dart programming language developed by Google.
This APP makes it easier for EUPT teachers to control the attendance of their students to each of their subjects in a simple and fast way by uploading through an Excel the list of students for each subject, then selecting the days they have class and thus being able to call roll in each session comfortably. Finally, it is possible to generate a monthly PDF with the attendance of all students, which teachers are required to sign. Cloud Firestore has been used to store the data and the user authentication has been done through the Authenticacion service also provided by Firebase. In addition, Bloc has been used as a state manager for the application to control user interactions with it.
This report consists of eight sections plus a set of annexes which are listed below: Introduction and Objectives, State of the Art, Analysis, Design, Technologies used, Tests, Software License and Documentation and a last section of Conclusions and Future Work.

Key Words

Flutter, Mobile App, Framework, Firebase, Bloc

1. Introducción y Objetivos

El ser humano tiene una necesidad por el control y esto se refleja también en el control de la asistencia a cualquier lugar y en cualquier situación. Se controla la asistencia a clase de los alumnos, la asistencia al trabajo de los empleados, la asistencia a citas médicas, espectáculos, así como muchos más ejemplos ya que esto permite no solo llevar un control para saber quién ha asistido como quién no, sino también realizar diversos estudios posteriores con estos datos.

Este control de la asistencia puede ser realizado tanto de manera perceptible cuando se tiene que firmar un documento o se tiene que reconocer una entrada o ticket de como imperceptible mediante visión artificial, cámaras IP, sensores infrarrojos en grandes superficies, supermercados y otras tiendas donde la entrada es libre y debido a la gran cantidad de personas que entran a lo largo del día, estos sistemas son lo más eficientes. [1]

La necesidad de llevar un control de la asistencia se ha visto además reforzada debido a la existencia de una pandemia llamada COVID, que ha hecho necesario saber con quién se ha estado en contacto y en concreto, la universidad es un lugar importante para la vida de los estudiantes donde pasan una gran cantidad de horas estudiando y por lo tanto se juntan con muchas personas.

A pesar de las distintas medidas de seguridad que se han ido implementando (uso de mascarillas, distancia de seguridad, ventilación…) no se han impedido los contagios.

Debido a todo esto y a la probabilidad de futuras enfermedades que afecten de forma global al día a día, es más necesario que nunca contar con una forma sencilla y rápida de llevar un control de la asistencia.

En la EUPT se solía hacer este registro en papel, pero debido a la digitalización de la sociedad, la forma de realizarlo ha ido cambiando con el tiempo gracias a la aparición de nuevas tecnologías lo cual ha dado resultado a diversas maneras de llevar un control de la asistencia entre las que destacan:

- El uso de tarjetas electrónicas con tecnología RFID (Radio Frequency Identification) las cuales consisten en que el estudiante pasa la tarjeta en un lector y se registra su asistencia. Su principal ventaja es la facilidad de uso ya que sólo hay que pasar una tarjeta para que se registre la asistencia. Sus principales desventajas son que no se puede estar seguro de si ha sido el propietario de la tarjeta el que la ha pasado y que se depende de la red eléctrica por lo que, en caso de apagones o falta de conexión, no funcionarán.
- La utilización del GPS del móvil de forma que cuando se está cerca del sitio que se ha determinado, se registra la asistencia. La ventaja es que el registro de la asistencia se hace de forma automática pero siempre que se tenga la ubicación del teléfono activada. Como desventajas se tienen que comentar la falta de precisión sobre todo dentro de edificios, así como la posibilidad de falsificar la ubicación real.
- A través del Wifi de forma que cuando una persona se conecta a un Wifi determinado se registra como una asistencia. Las ventajas son similares al GPS ya que el registro de la asistencia se hace de forma automática, pero es necesario tener el Wifi activo.
  En cuanto a las desventajas son, la necesidad de routers para cada clase (2 o 3 para que sea eficiente), la necesidad de tener conexión a internet, así como la posibilidad de que el alumno simplemente se acerque a clase para que el móvil detecte el Wifi y no acuda a clase.
- Mediante el escaneo de Códigos QR. Esta medida es bastante peligrosa pues sólo hace falta una foto al código QR para que se pueda escanear desde cualquier lugar y por lo

tanto el control de la asistencia perderá todo el sentido. Una solución sería generar códigos QR para cada clase lo cual conlleva un gasto de tiempo. [2], [5]

Como se ha visto, la mayoría de estas formas de controlar la asistencia requieren de la participación del estudiante y no son del todo fiables ya que, de una forma u otra, se puede engañar haciendo creer que se ha ido a clase cuando no ha sido así o puede que no se registre correctamente la asistencia. Por lo que la responsabilidad de pasar lista suele recaer en el profesor, que sabe al 100% quién ha asistido a clase y quién no, evitando así posibles fallos a la hora de controlar la asistencia.

El objetivo general del TFG consiste en el análisis, diseño, implementación y correspondiente juego de pruebas, de una aplicación móvil para el control de la asistencia con los siguientes objetivos específicos:

1. Cargar el listado de los alumnos de cada asignatura a través de un Excel generado desde la aplicación de Sistema de Gestión Académica (sia.unizar.es) de la Universidad de Zaragoza (UZ)
2. Permitir que los docentes seleccionen cuando tienen clase de cada asignatura durante el periodo de tiempo que ésta dura.
3. Mostrar a los docentes cuando tienen cada asignatura y permitir pasar lista a los alumnos para cada día, así como modificarla.
4. Generar al final del mes para cada asignatura un PDF el cual los docentes tienen que firmar y remitir a la secretaría de la EUTP para su control y registro.

Para finalizar con este apartado, hay que comentar que la memoria se estructura ocho apartados. En el primer capítulo se presenta esta propuesta, explicando por qué se va a hacer y los objetivos que se quieren conseguir. En el segundo capítulo se hace un análisis del estado del arte, donde se investigan las tecnologías que se usan principalmente para el desarrollo de este tipo de aplicaciones, así como se analizan varios sistemas similares. En el tercer capítulo, se encuentra la fase de análisis donde se establecen los requisitos que tiene que cumplir la aplicación y se diseñan los casos de uso, y los diagramas de secuencia y actividad, para que se vea de forma más clara el funcionamiento de la aplicación. En el cuarto capítulo se encuentra la fase diseño donde se establece la arquitectura, así como los primeros bocetos del aspecto que tendrá en el futuro. En el quinto capítulo se encuentra la fase de desarrollo de la APP, aquí se explican las tecnologías y programas que se han utilizado para llevar a cabo este proceso, así como los distintos pasos su implementación.
En el sexto capítulo se encuentra la fase de pruebas donde se comprueba que la APP cumple los requisitos que se indican en el tercer capítulo. En el séptimo capítulo se detallan la licencia software y documental. En el octavo capítulo se habla de las conclusiones y trabajos futuros para la aplicación.

2. Estado del Arte

En este apartado se va a realizar un análisis de las principales tecnologías que se utilizan a la hora de desarrollar aplicaciones móviles, así como una selección de varios ejemplos de aplicaciones para el control de la asistencia en el aula con el fin de ver qué es lo que ofrecen y qué es lo que las diferencian de esta propuesta.

Después de buscar y analizar diversas aplicaciones para el control de la asistencia, se ha comprobado que éstas están siempre disponibles para al menos una de las siguientes plataformas sino son ambas:

- Para Plataformas móviles mediante Apps
- Para ordenadores a través del navegador mediante aplicaciones web

Por lo que se van a estudiar estas dos opciones.

2.1. Aplicaciones Web

Las aplicaciones web se basan en la estructura Cliente-Servidor (C/S), esto quiere decir que se ejecutan dentro del navegador web del usuario (cliente) y se almacenan en un servidor remoto haciendo que la aplicación sea independiente del sistema operativo. [3]

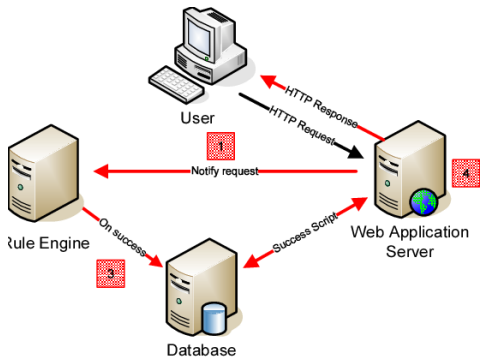

FIGURA 1. FUNCIONAMIENTO DE UNA APLICACIÓN WEB

Como se puede ver en la imagen, el usuario interactúa con la aplicación web mediante peticiones en las cuales se envía distinta información y se genera una página acorde a esto. Este funcionamiento ofrece una serie de ventajas y desventajas en cuanto a la hora de desarrollar aplicaciones web. [4]

Las ventajas de desarrollar aplicaciones web son:

- Ahorro de tiempo ya que con un solo desarrollo se llega a todos los usuarios
- No ocupan espacio en el ordenador.
- Facilidad de uso, pues no hay que instalar programas.
- Se pueden acceder desde cualquier dispositivo ya que todos llevan incorporado un navegador web.
- Las actualizaciones de la página (añadir nuevas funcionalidades, cambiar el aspecto…) son globales e instantáneas para todos ya que el código es prácticamente el mismo para todos.

Las desventajas encontradas son:

- Hay que pagar para tener un dominio en Internet donde alojar la página.

- Requieren siempre de conexión a Internet, sobre todo si dependen de servicios de terceros.
- No están adaptadas al dispositivo del usuario, por lo que la experiencia de uso puede que no sea la mejor.
- Son más lentas que las aplicaciones nativas pues dependen del navegador.

Dentro de las aplicaciones web se pueden distinguir 2 grupos: estáticas y dinámicas

Aplicaciones web estáticas

Es lo que se conoce como sitio web, se tratan de páginas web escritas con HTML, CSS y JavaScript en las que la interacción con el usuario es escasa o nula y su actualización es compleja por lo que se usan para contenidos que no se van a actualizar como blogs, portfolios o páginas de presentación de empresas

Aplicaciones web dinámicas

En este caso el usuario sí que interactúa con el sitio de manera que cada interacción con el sitio requiere de una respuesta por parte del servidor. Las aplicaciones web suelen contar con dos partes: backend y frontend. [6], [7]

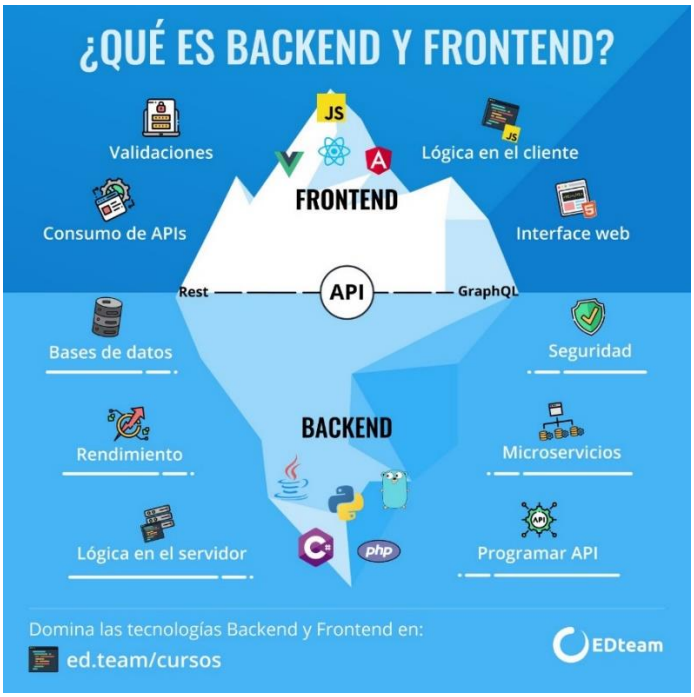

FIGURA 2 - BACK-END Y FRONT-END

- El front-end es la parte visible del sitio a través del navegador y con la que el usuario interactúa.
- El back-end es la parte dinámica donde se gestiona el funcionamiento interno del sitio, es decir, donde se gestionan los contenidos mediante bases de datos, se muestra una página u otra según elija el usuario, se procesa la información o se llaman a servicios de terceros entre otras cosas.

En cada una de estas partes trabajan distintas tecnologías:

- En el front-end destaca el uso de HTML (Hypertext Markup Language) , CSS (Cascading Style Sheets) y Javascript
- En cuanto al backend la variedad de tecnologías disponibles es muy diversa ya que como se ha comentado, en esta parte es donde se realizan la mayoría de las tareas. Teniendo ejemplos como MySQL o MariaDB para bases de datos, Express.js, Node.js para crear las APIs, PGP (Pretty Good Privacy) para la encriptación y ASP (Active Server Pages) para la lógica del servidor entre otros.

2.2. Aplicaciones móviles

Actualmente, casi todo el mundo tiene un teléfono móvil [5] y el número de aplicaciones móviles disponibles no para de crecer de forma que casi todo se puede hacer desde este dispositivo y aproximadamente un 88% del tiempo que se está usando el teléfono móvil se navega por alguna aplicación. [14]

La existencia de distintos tipos de sistemas operativos (iOS y Android principalmente) hace que haya diversas estrategias a la hora de desarrollar una aplicación móvil. [8]

Aplicaciones móviles nativas

Las aplicaciones nativas son aquellas que se desarrollan específicamente para un sistema operativo concreto. Esto hace que se tenga que conocer el lenguaje específico para cada sistema operativo como Java para Android o Swift para iOS, así como que se tenga que contar con distintos equipos de desarrollo para cada sistema lo que hace que se necesite más personal y los costes sean más elevados. Como ventaja, esta aproximación permite aprovechar al máximo los recursos de cada sistema y proporcionar una mejor experiencia al usuario. Para este tipo de aplicación destaca el uso de Java, Kotilin, Swift o Objective-C.

Aplicaciones Híbridas

Este tipo de aplicaciones permite con un solo código, desarrollar aplicaciones para los distintos sistemas operativos. Las ventajas de este tipo de aplicaciones son su rápido desarrollo pues con un solo código se puede llegar a todos los sistemas operativos. Esto hace también que sean más fáciles de mantener pues solo se tiene un código y por lo tanto cualquier cambio en la APP llega a todos los sistemas a la vez. Pero también tiene desventajas pues son más lentas que las aplicaciones nativas y no se aprovechan los recursos de cada sistema al máximo. Para este tipo de aplicaciones destaca el uso de Ionic, React Native o Flutter

Aplicaciones Web

Se trata de aplicaciones a las que se accede mediante un navegador por lo que son independientes del sistema operativo que se utilice para móvil u ordenador. Sus ventajas son un desarrollo rápido pues con un solo código se pueden usar en cualquier dispositivo y A un bajo coste, ya que no hay que tener un equipo de desarrollo para cada plataforma móvil. Este tipo de aplicaciones suelen utilizar HTML y CSS y JavaScript para la interfaz gráfica (UI) y PHP o Python para gestionar la lógica (Backend). Sus desventajas se centran en que se requiere tener siempre acceso a la web, los tiempos de respuesta son mayores, así como el acceso a los recursos del dispositivo es limitado. [10]

Aplicaciones Web Progresivas

Se trata de aplicaciones web pero que el usuario interpreta como una aplicación de escritorio o móvil pues tienen un aspecto similar y al instalarlas se crea un acceso en el dispositivo. Este tipo de aplicaciones se desarrollan con las mismas tecnologías que las aplicaciones web y aparte, destaca el uso de los Services Workes que serían similares a los servicios de Windows o a los deamons en Linux y que se instalan en el navegador, con los cuales se puede hacer que la aplicación lance notificaciones, guarde cache o funcione offline. Sus ventajas son que no requieren descarga, ocupan poco espacio y se pueden usar offline, permite enviar notificaciones al usuario y usan el protocolo HTTPS lo que proporciona más seguridad a parte de las ventajas explicadas previamente para las aplicaciones web Las desventajas son que no están disponibles para todos los dispositivos (en iOS solo a partir de la versión 11.3), no pueden acceder a muchas características del teléfono y por lo tanto no tienen las mismas funcionalidades que una aplicación nativa. [11], [12]

2.3. Aplicaciones similares a la propuesta

En esta sección se van a analizar varias herramientas que se utilizan para llevar un control de la asistencia tanto a través de móviles como mediante la web. Como no se han encontrado aplicaciones específicas para el control del COVID, el estudio se va a centrar en aplicaciones para el control de la asistencia en el aula. [13], [15], [16]

Se ha comprobado que hay bastantes aplicaciones que ofrecen la posibilidad de llevar un control de la asistencia, pero suelen tener demasiadas funcionalidades fuera del ámbito de aplicación o no cumplen con lo deseado por la aplicación a desarrollar.

Alexia

Se trata de una suite educativa con una gran variedad de opciones, tanto para los profesores (pasar lista, establecer unidades didácticas, subir contenido…) como para las familias (informar sobre sus hijos, responder a los profesores…). Es una aplicación compleja con muchas funcionalidades que requiere de una formación para sacarle el máximo provecho. [17]

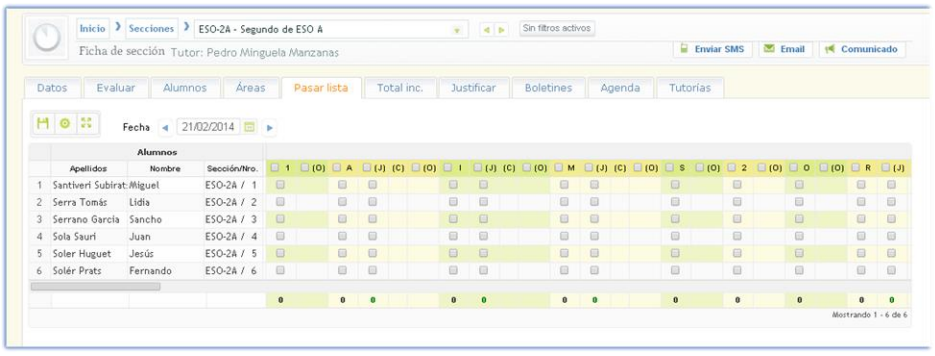

FIGURA 3 - PASAR LISTA EN ALEXIA

Dinantia

Aplicación móvil tanto para Android y iOS como para web. Permite llevar un control de la asistencia, comunicación directa con padres o alumnos e incluso hacer test desde la aplicación. [18]

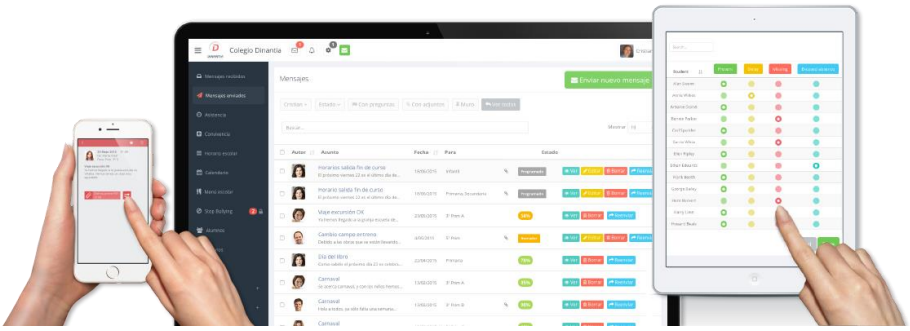

FIGURA 4 - EJEMPLO DE LA APLICACIÓN DIANTIA

Attendance Taker

Aplicación móvil gratuita para Android. Es una aplicación sencilla que permite crear clases y añadir alumnos para pasar lista e incluye la generación de informes en Excel o PDF. [19]

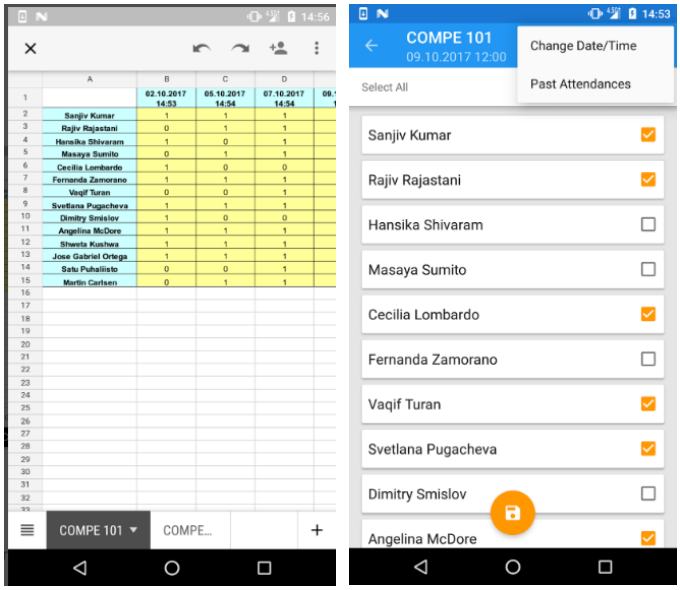

FIGURA 5 - INTERFAZ DE ATTENDANCE TRAKER PARA PASAR LISTA Y EXCEL DE LAS ASISTENCIAS

Pasalista

Aplicación móvil para Android. Permite crear listas de alumnos tanto manualmente como desde Excel, así como generar reportes en PDF o Excel, pero es de pago o con publicidad. [20]

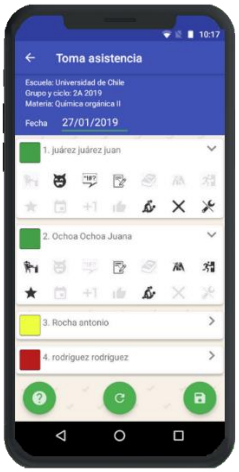

FIGURA 6 - EJEMPLO DE APLICACIÓN PASALISTA

A continuación, se presenta una tabla con las distintas herramientas comparando sus funcionalidades con las que se desea que tenga la aplicación a desarrollar en este TFG.

Aplicación	Sistemas Disponibles	Cargar alumnos desde fichero	Tiene calendario con las clases del día	Pasar Lista	Generar PDF con el formato deseado	Editar Lista de Asistencia	Modalidad
Pasalista	Android	Se pueden cargar alumnos desde CSV o TXT, pero no acepta el formato del fichero proporcionado por sia.unizar.es	No tiene, hay que añadir las listas a mano	Si	Permite generar PDF	Si	Con anuncios/ De pago
Attendance Tracker	Android	No, se añaden a mano.	No, hay que pasar lista para cada día a mano	Si	No, genera Excel con el número total de alumnos presentes	Si	Gratuito
Dinantia	iOS, Android, Windows	Si, pero con un formato determinado distinto al deseado	No, hay que pasar lista cada día	Si	Generar Excel, pero con las ausencia y asistencias totales	Si	De pago
Alexia	Navegador, Android y iOS (pero con funcionalidad limitada)	No lo indica	Tiene un calendario, pero las clases hay que crearlas individualmente	Si	Genera un PDF para cada alumno	Si	De pago

FIGURA 7 - COMPARATIVA DE HERRAMIENTAS

Una vez se ha realizado el análisis de las tecnologías y se han analizado varias aplicaciones similares a la propuesta de este TFG, se ha visto que la mayoría de las aplicaciones, aunque ofrecen funcionalidades similares a las de la propuesta, como son: cargar datos desde Excel y generar PDF, presentan el problema de que los archivos no se ajustan al formato que necesitan los docentes de la EUPT y además son servicios de pago o con anuncios. Tampoco se tiene la posibilidad de crear las sesiones para las asignaturas de forma dinámica y que se puedan visualizar en un calendario para cada mes, sino que el profesor tiene que seleccionar específicamente la asignatura en el día para crear una lista de asistencia, de las aplicaciones analizadas la única que cuenta con un calendario es Alexia, pero se tienen que añadir las sesiones manualmente.
La única propuesta bastante similar es Pasalista, a pesar de que no aceptaba el formato del Excel proporcionado a los docentes mediante sia.unizar.es.

Por lo que se ha optado por realizar un desarrollo de aplicación híbrida ya que permite una implementación rápida para todos los sistemas móviles, aprovecha la mayoría de los recursos

que los dispositivos ofrecen e incluye el acceso al almacenamiento del dispositivo que es lo que se necesita en este desarrollo. [22]

En concreto, para realizar la aplicación, se ha escogido el lenguaje de programación Dart junto con el UI Flutter ambos desarrollados por Google, Flutter es un UI Framework y actualmente es uno de los más utilizados ya que ha ido ganando popularidad rápidamente y permite con un mismo código desarrollar tanto aplicaciones móviles como para la web y escritorio, abarcando así las 2 principales plataformas que se utilizan, ordenadores y móviles. Además, proporciona un rendimiento superior a las otras opciones, lo que ha hecho que finalmente se opte por esta opción. [23]

Para la base de datos, se ha escogido Cloud Firestore, un sistema gestor de bases de datos no relacional y Authentication para el control de usuarios entre otros servicios y además está desarrollada por Google al igual que Flutter y Dart por lo que se va a integrar perfectamente con la APP. [30]

3. Análisis

Todo proyecto de ingeniería del software tiene una serie de etapas dentro de su ciclo de vida las cuales son necesarias para un correcto desarrollo.

La primera etapa es el Análisis, en la cual se detallarán los requisitos, casos de uso que deberá cumplir la aplicación de manera que se obtenga un sistema con las características que se van a necesitar y evitar así desarrollar aspectos innecesarios que supondrían un elevado coste para el proyecto. [26]

3.1. Análisis de Requisitos

A continuación, se detallan los requisitos que tiene que cumplir la aplicación propuesta en este TFG. Estos requisitos se dividen en dos tipos: Requisitos Funcionales (RF), que son los requisitos que definen la funcionalidad de la aplicación y los Requisitos No Funcionales (RNF) que hacen referencia a las características generales o restricciones que tiene que cumplir el sistema que se está desarrollando como pueden ser: la fiabilidad, seguridad o escalabilidad de la aplicación de acuerdo con los aspectos reflejados en el libro Ingeniería de software de Ian Sommerville [26].

En primer lugar, se van a detallar los RF:

Acceso y salida del sistema

Conjunto de requerimientos que permiten a un usuario acceder y salir del sistema

- RF01. La aplicación debe permitir dar de alta un nuevo usuario siempre que el correo electrónico no esté asociado ya a una cuenta.
- RF02. La aplicación debe permitir al usuario iniciar sesión mediante Google.
- RF03. La aplicación debe permitir dar de baja a un usuario siempre que se haya registrado en la aplicación cumpliendo el RF01.
- RF04. La aplicación debe permitir el acceso de un usuario si éste introduce un correo electrónico y contraseña correcto que estén registrados según el RF01.
- RF05. La aplicación debe permitir a un usuario autenticado cerrar sesión.
- RF06. La aplicación debe permitir al usuario restablecer su contraseña si tiene una cuenta previamente creada según el RF01.

Acciones generales del usuario

- RF07. El usuario podrá añadir una nueva lista de alumnos mediante la subida de un fichero Excel.
- RF08. El usuario podrá crear o eliminar asignaturas.
- RF09. La aplicación permitirá al usuario seleccionar los días de la semana en los que se dan clases de cada asignatura.
- RF10. El usuario podrá limitar la fecha de inicio y de fin de la asignatura
- RF11. El usuario podrá eliminar una asignatura siempre que esta esté previamente creada.
- RF12. La aplicación mostrará para el día actual las asignaturas que tiene el usuario de acuerdo con los días indicados en el RF09.
- RF13. La aplicación permitirá al usuario cambiar el modo de vista entre día, semana y mes.

- RF14. El usuario podrá pasar lista para cada asignatura en las fechas que se han indicado en el RF09.
- RF15. La aplicación permitirá al usuario elegir entre presente o no presente para cada alumno de cada asignatura en la fecha seleccionada.
- RF16. El usuario podrá modificar la lista de asistencia, para una fecha concreta, siempre que esta exista previamente de acuerdo con el RF11.
- RF17. El usuario podrá eliminar una lista de asistencia para una asignatura siempre que exista previamente dicha lista de asistencia.
- RF18. La aplicación deberá mostrar al usuario la opción de generar un PDF para una asignatura concreta indicando el mes para el que se va a generar.
- RF19. La aplicación generará un PDF mostrando la asistencia de los alumnos a una asignatura de acuerdo con lo definido en el RF16.
- RF20. El usuario podrá descargarse los PDF generados de acuerdo con el RF19.

A continuación, se procede a comentar los RNF:

Eficiencia

- RNF01. La aplicación tardará menos de 2 minutos en cargar la lista de alumnos a partir del Excel proporcionado, siempre que éste tenga el formato adecuado.
- RNF02. La creación de una nueva asignatura con sus fechas se podrá hacer en menos de 5 pasos o 5 minutos.
- RNF03. El usuario tardará menos de 1 minuto en acceder a los contenidos de la aplicación.
- RNF04. La aplicación tardará menos de 10 segundos en cargar la lista de alumnos para una asignatura y fecha concreta.
- RNF05. El usuario tardará menos de 45 segundos en pasar lista para el día actual
- RNF06. La aplicación tardará menos de 5 segundos en mostrar las listas de asistencia para una asignatura.
- RNF07. La aplicación no se bloqueará cuando se estén realizando operaciones.

Seguridad

- RNF08. No será expuesta ningún tipo de información sensible, así como guardada en posibles sitios vulnerables.
- RNF09. Se garantizará que los datos del usuario cumplen el nivel básico de seguridad según el RGPD.

Usabilidad

- RNF08. La aplicación móvil deberá tener un diseño responsive, es decir, adaptarse a todas las resoluciones de pantalla.
- RNF09. El tiempo de aprendizaje del sistema para un usuario medio deberá ser menor a 1 hora.
- RNF10. El sistema debe proporcionar mensajes de error informativos y orientados a usuarios finales.
- RNF11. Aparecerá un icono de cargando cuando se realicen operaciones internas que duren demasiado.
- RNF12. Las distintas pantallas contarán con mecanismos de vuelta a la pantalla anterior.

Disponibilidad

- RNF13. El sistema tiene que tener una disponibilidad del 99,8% de las veces en que un usuario intente acceder a él.

Integridad

- RNF14. El sistema debe asegurar la integridad de los datos.

Del producto

- RNF14. La aplicación será compatible con Android 8.0 o posteriores.
- RNF16. La aplicación será compatible con iOS 8 o posteriores.
- RNF17. La aplicación no podrá ocupar más de 100 megabits.

Externos

- RNF18. El sistema deberá operar los datos de manera que cumpla el RGPD. [24]
- RNF19. El sistema será gratuito y de código abierto.

De dominio

- RNF20. Será necesario el uso de internet para el funcionamiento de la aplicación.
- RNF21. La aplicación no consumirá más de un 40% de la RAM del teléfono.

Para la fase de Análisis de la aplicación propuesta, se han tenido en cuenta los aspectos reflejados en la ISO 25010-ISO/IEC 25000, centrándose en la Adecuación Funcional, Compatibilidad, Eficiencia, Usabilidad, Fiabilidad y Seguridad. [25]

FIGURA 8 - MODELO DE CALIDAD DEL PRODUCTO DEFINIDO POR LA ISO/IEC 25010

3.2. Caso de Uso, Secuencia y Actividad

Para el desarrollo de estos diagramas, se han seguido las directrices del lenguaje unificado modelado (UML), respaldado por el Object Management Group (OMG). [27], [64]

El diagrama de casos de uso es la descripción de una secuencia de actividades que deberá realizar alguien o algo para llevar a cabo un proceso.

Mediante los casos de uso se representan los RF.

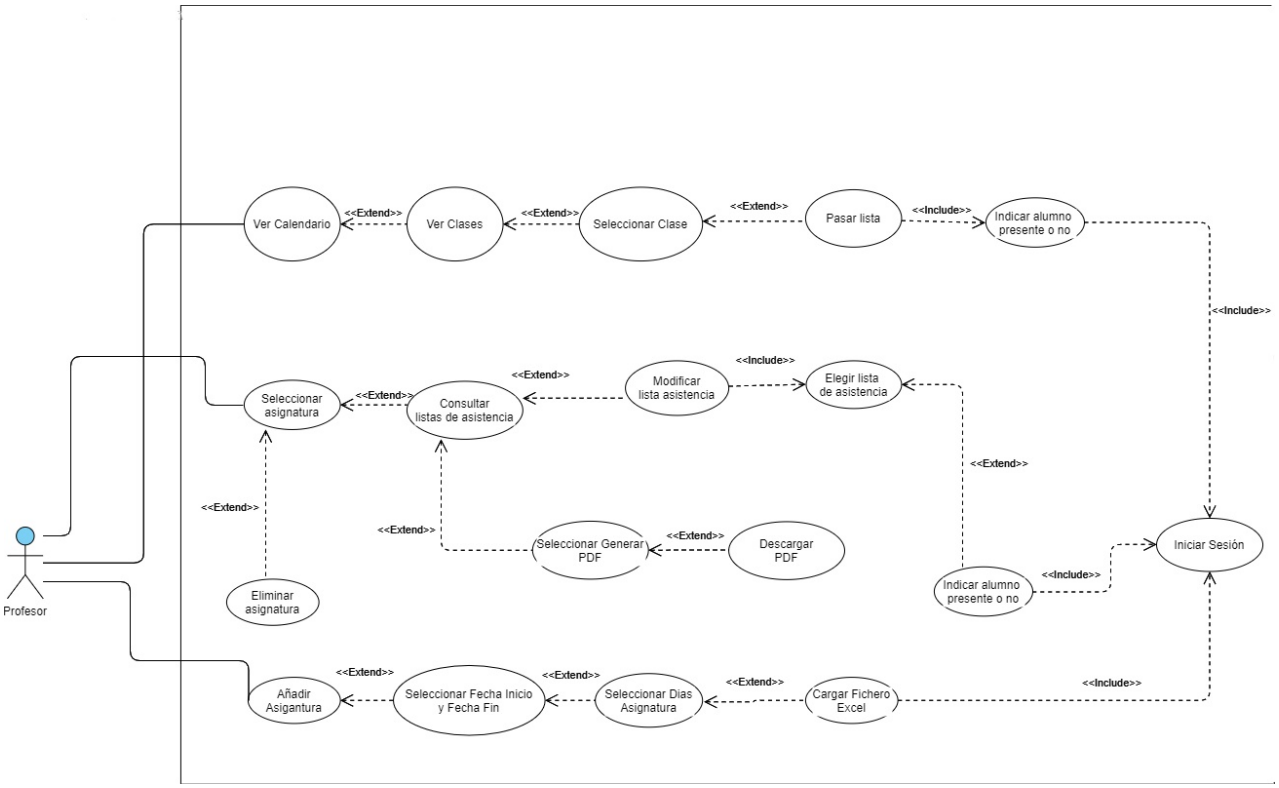

FIGURA 9 - CASOS DE USO

Se ha realizado un diagrama de secuencia indicando cómo sería el proceso para pasar lista de una asignatura, para la fecha actual, considerando que el usuario ya ha iniciado sesión previamente.

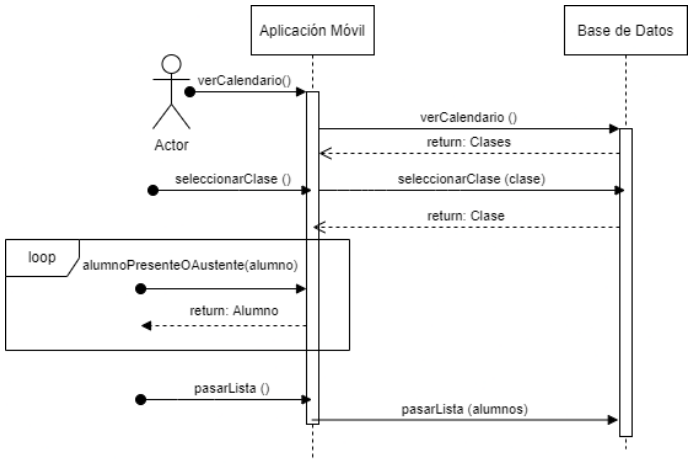

FIGURA 10 - DIAGRAMA DE SECUENCIA PARA PASAR LISTA

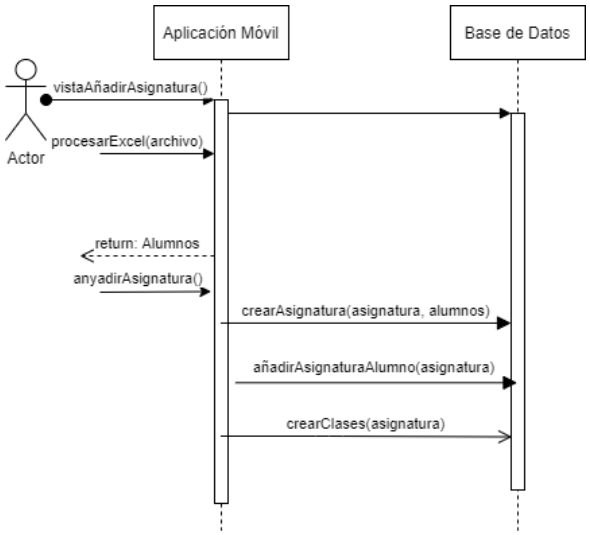

FIGURA 11 - DIAGRAMA DE SECUENCIA PARA AÑADIR ASIGNATURA

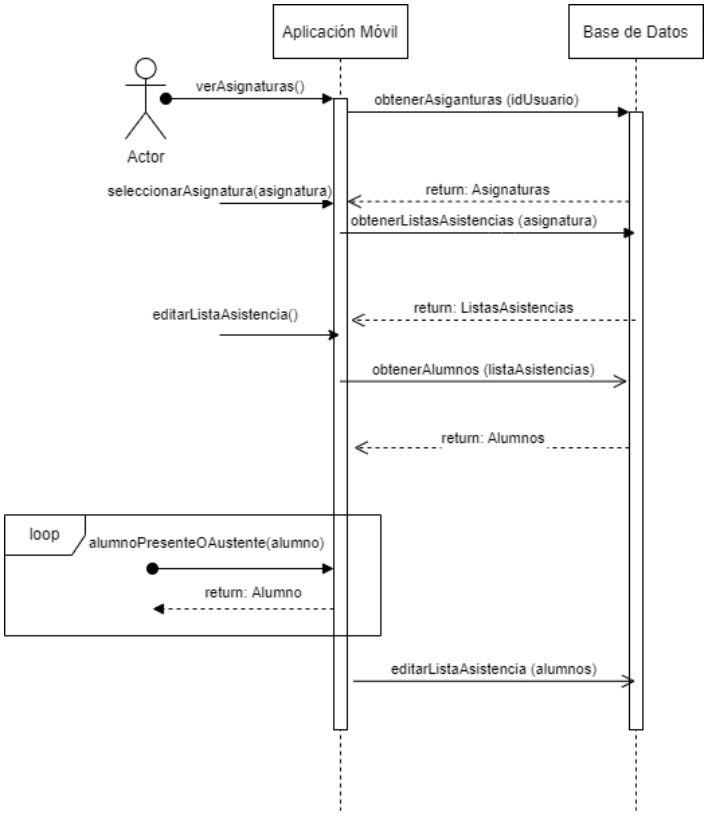

FIGURA 12 - DIAGRAMA DE SECUENCIA PARA EDITAR LISTA DE ASISTENCIAS

Los diagramas de actividades muestran el flujo de un proceso desde su inicio hasta su fin, indicando el comportamiento de cada sistema que interviene en dicha actividad.

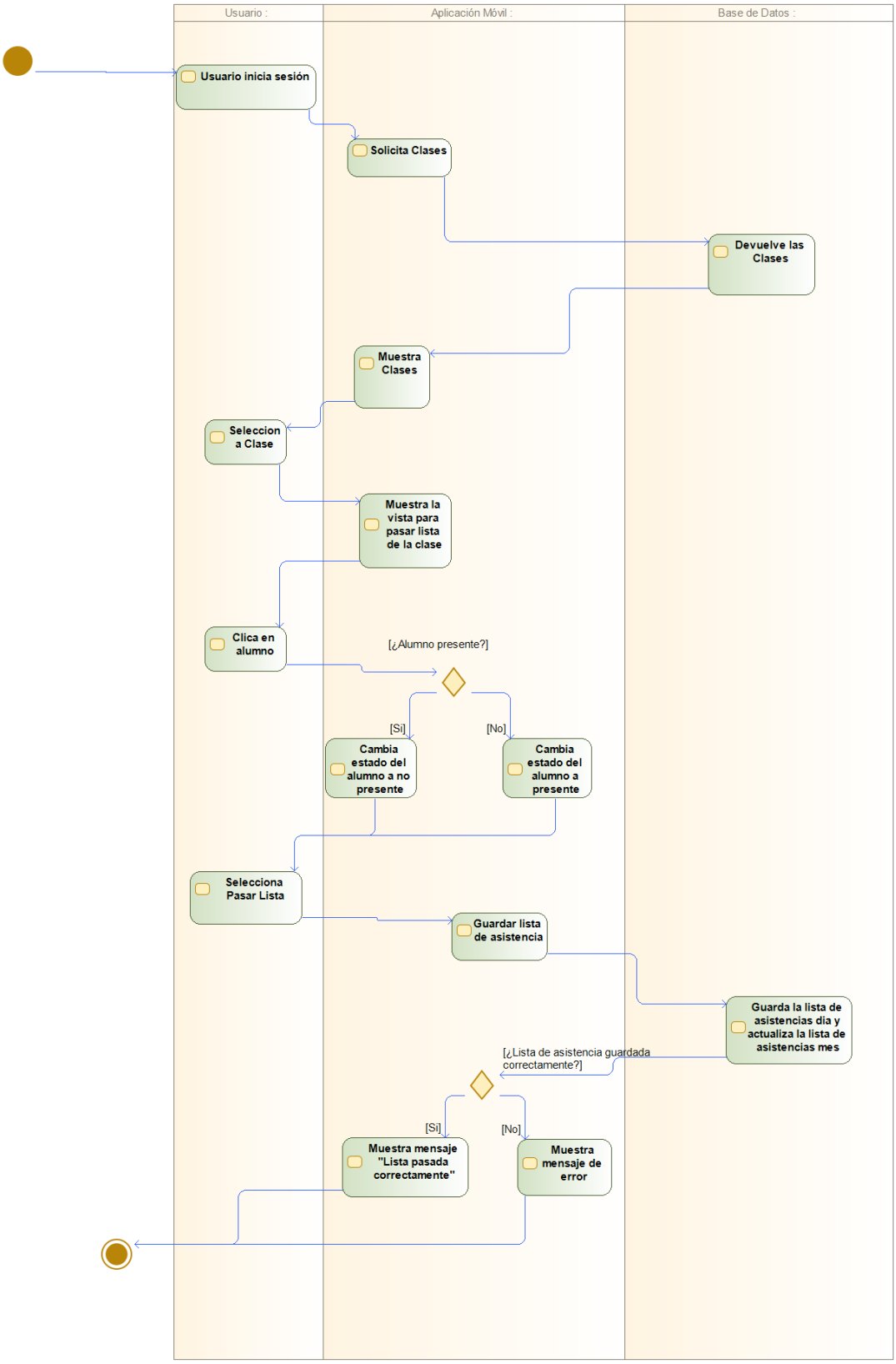

FIGURA 13 - DIAGRAMA DE ACTIVIDAD DE PASAR LISTA

4. Diseño

Una vez que se ha realizado la fase de análisis y se conocen los requisitos que tienen que cumplir la aplicación, hay que pasar a hablar del diseño del sistema indicando cómo se va a realizar la arquitectura tanto del front-end como del back-end, así como un primer vistazo a cómo será la interfaz de la aplicación.

4.1. Patrón de diseño

El patrón de diseño que se va a utilizar para desarrollar esta aplicación es el Modelo-Vista-VistaModelo (MVVM) una arquitectura muy utilizada en aplicaciones móviles que permite separar lo máximo posible la interfaz (Vista) de la lógica de la aplicación (Modelo) ya que en Flutter la parte principal es la Vista.

La capa ModeloVista actúa como intermediario entre la Vista y el Modelo para transferir datos entre ellos. Cuando el usuario solicita datos mediante la Vista, el ModeloVista informa al Modelo y éste (ModeloVista) lo notifica a la Vista para que los muestre.

La capa de Vista representa la interfaz del usuario mediante los Widgets. En esta capa se reciben los eventos del usuario y estos se redirigen a la capa de ModeloVista para hacer los procesos necesarios y cuando finalice se actualizará la Vista.

A continuación, se aporta una imagen que ilustra el comportamiento de este patrón de diseño [33]

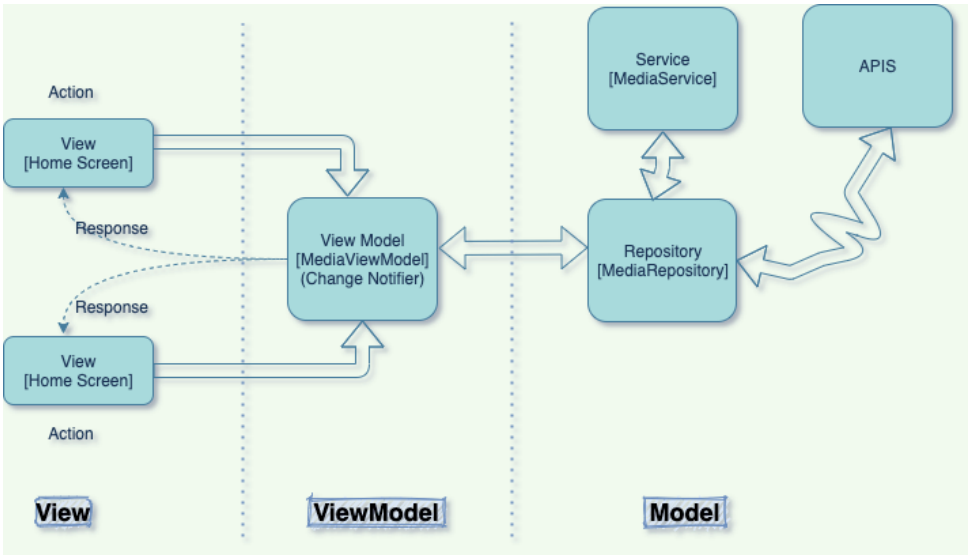

FIGURA 14 - PATRÓN MVMM

4.2. Arquitectura

La aplicación va a seguir la arquitectura de capas, que se utiliza para aislar la lógica de negocio (lo que es la capa de Dominio) de las partes exteriores de la aplicación, de forma que un cambio en las partes externas no suponga cambios importantes en la lógica de negocio. [34], [35], [36]

La arquitectura de capas se ha divido en:

- Presentación: Es la interfaz gráfica de la aplicación, que ha sido desarrollada mediante el Framework Flutter de Google que hace uso del lenguaje Dart.

- Dominio: Contiene los modelos, entidades, los cuales son independientes de la API o base de datos de los que se obtienen y no van a cambiar nunca, así como las reglas de negocio.
- Aplicación: Es la capa que comunica la capa de Presentación con la de Infraestructura, en el caso de la APP para este TFG se ha usado Bloc, un paquete de Dart para separar la vista del dominio.
- Infraestructura: Es la capa que se comunica con el exterior, en el caso de la aplicación para este TFG, se comunicaría con Firebase, en concreto el servicio Cloud Firestore que es una base de datos NoSql con la que se trabajará para tratar los datos de las asignaturas, los alumnos y las listas de asistencia mediante el uso del formato JSON (JavaScript Object Notation) que es un formato sencillo para intercambiar información entre servidores y aplicaciones basado en clave-valor.

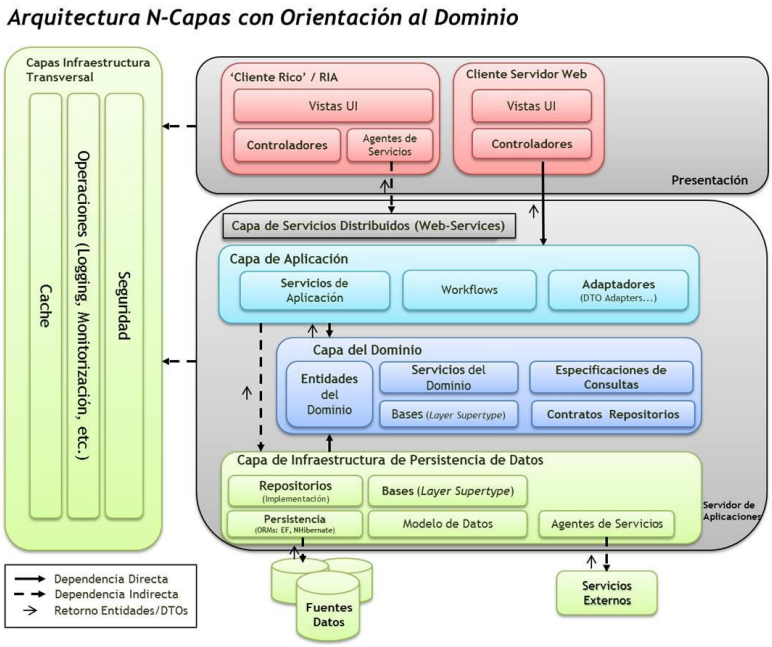

FIGURA 15 - ARQUITECTURA POR CAPAS

Con esta arquitectura, se busca no depender de una tecnología concreta, sino que de forma sencilla se pueda cambiar de una tecnología a otra sin tener que modificar demasiado el resto de la aplicación, para esto se hace uso de los repositorios (en la capa de infraestructura) que son los que se encargan de obtener los datos.

4.3. Prototipos

Como se ha indicado, la aplicación cuenta tanto con una parte front-end de software con la que interaccionan los usuarios y una capa back-end que consiste en una base de datos no-sql Firebase (Cloud Firestore)

Mediante el diseño de prototipos se va a tener una idea aproximada de cómo se verá la aplicación, cómo se van a implementar las distintas funcionalidades y se tendrá una idea más clara de la aplicación.

Para diseñar estos prototipos se ha utilizado la herramienta Balsamiq Mockups.

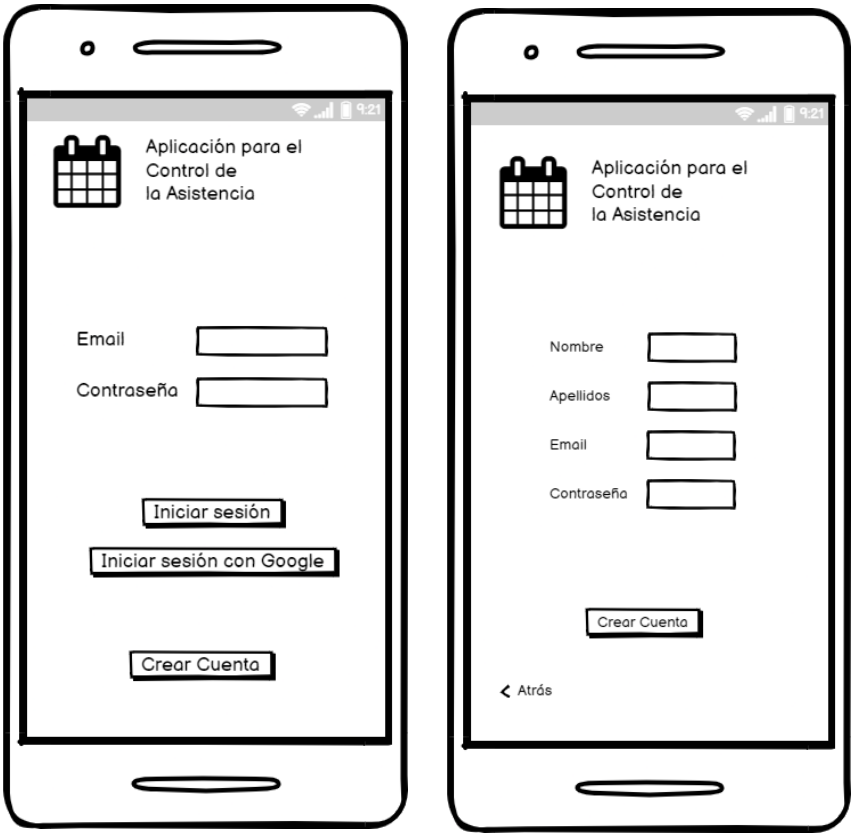

FIGURA 16 – PROTOTIPO PARA INICIAR SESIÓN Y PROTOTIPO PARA CREAR CUENTA

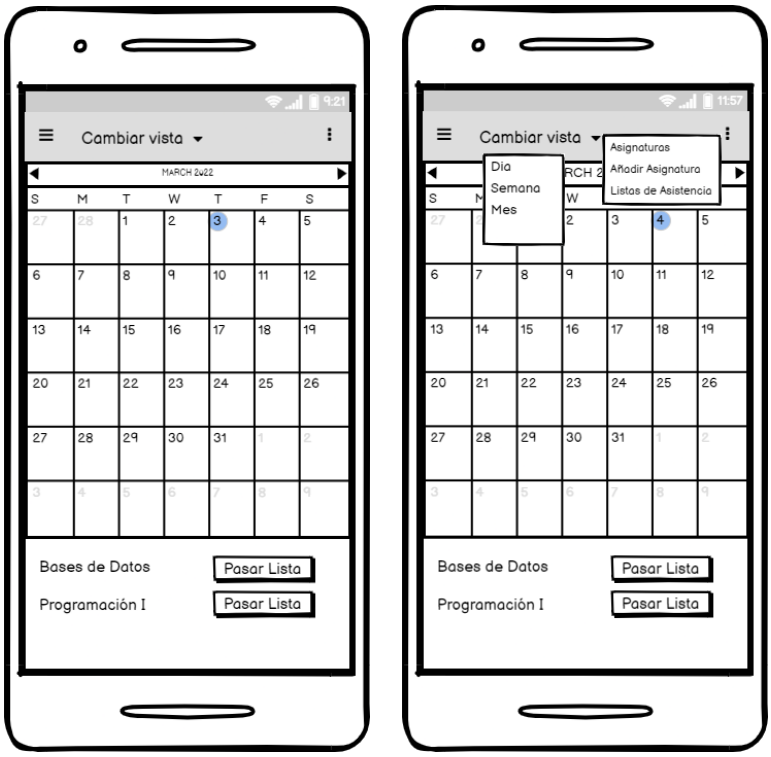

FIGURA 17 - PROTOTIPO PARA VISUALIZAR MES Y PROTOTIPO VISTA MES CON MENÚS DESPLEGADOS

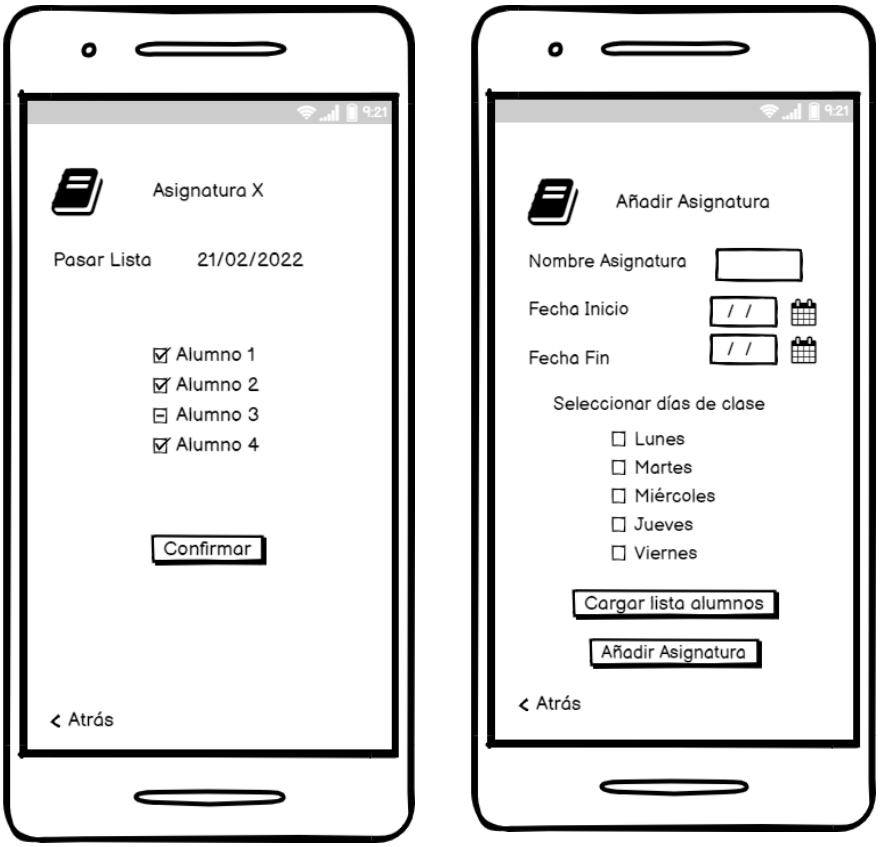

FIGURA 18 – PROTOTIPO PARA PASAR LISTA EN EL DÍA ACTUAL Y PROTOTIPO PARA AÑADIR ASIGNATURA

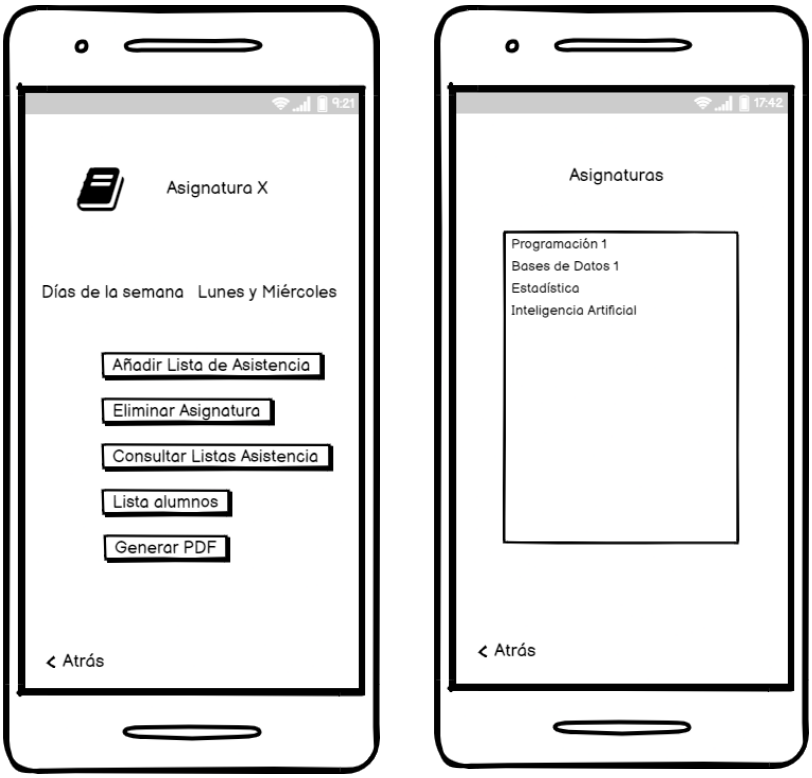

FIGURA 19 - PROTOTIPO DE LA VISTA DE UNA ASIGNATURA Y PARA LISTAR ASIGNATURAS

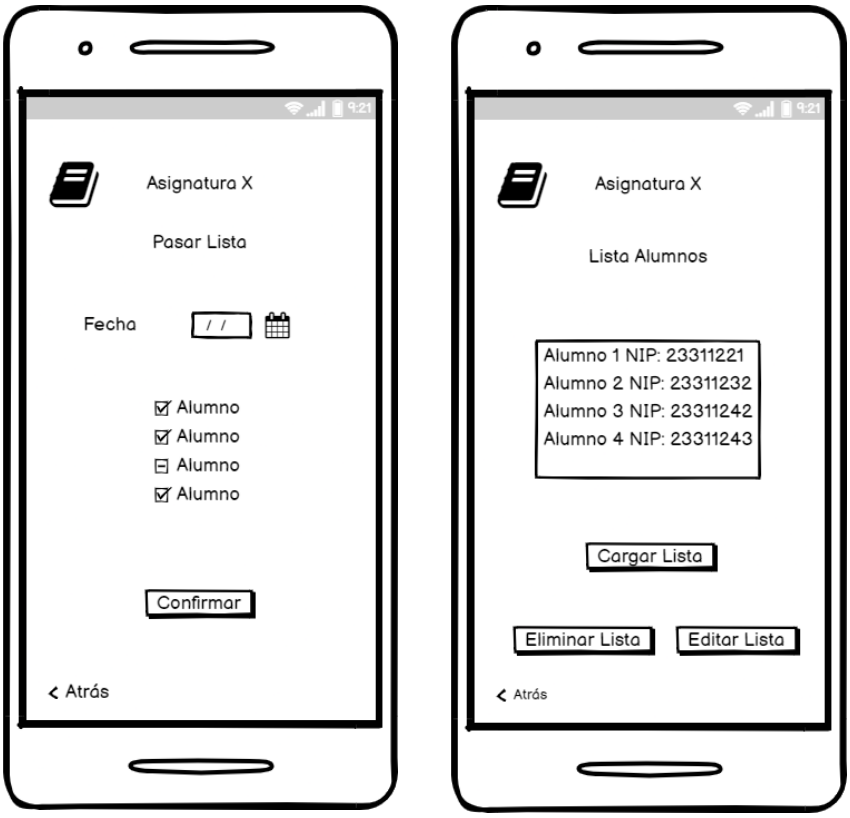

FIGURA 20 - PROTOTIPO PARA PASAR LISTA Y PROTOTIPO LISTA ALUMNOS PARA ASIGNATURA

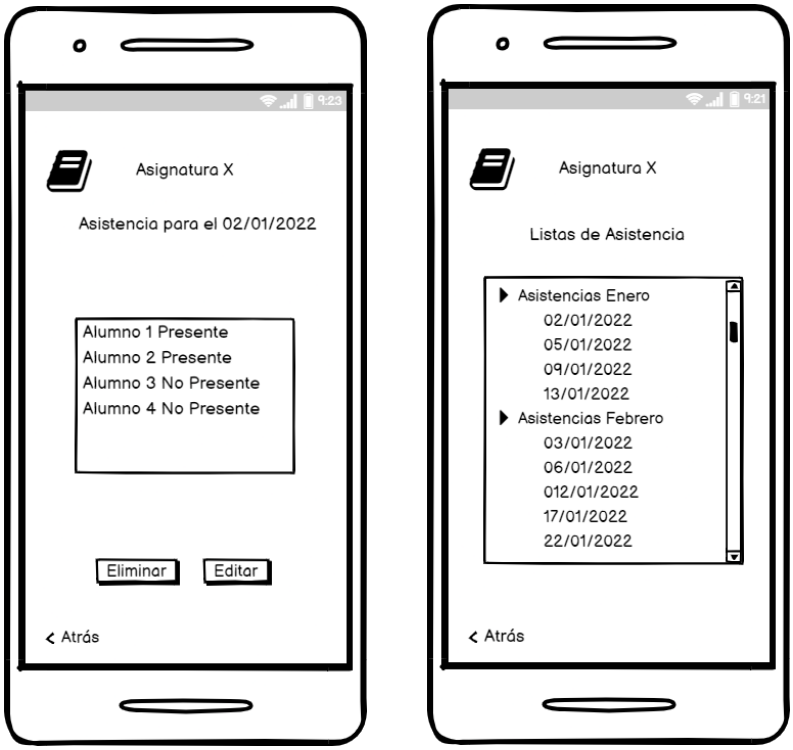

FIGURA 21 PROTOTIPO LISTA ASISTENCIA PARA ASIGNATURA Y PROTOTIPO LISTAS DE ASISTENCIA

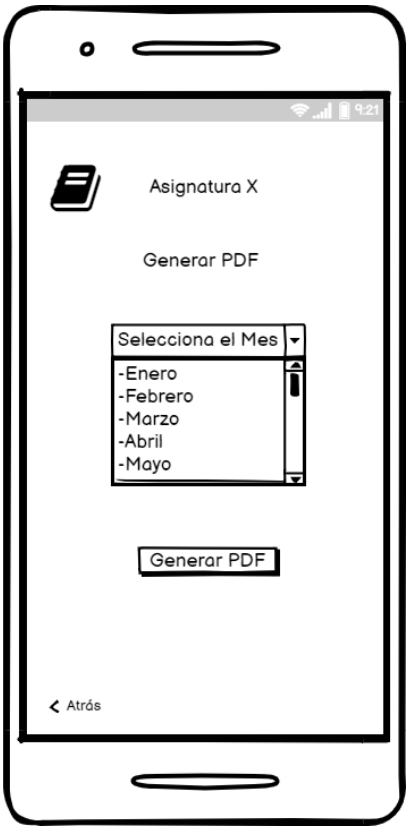

FIGURA 22 - PROTOTIPO PARA GENERAR PDF

4.4. Base de Datos

Es necesario hacer un diseño previo para la base de datos ya que esto facilitará luego las cosas a la hora de realizar la aplicación.

La base de datos elegida para la aplicación es una base de datos NoSQL (Not only SQL) o no relacional donde los datos se organizan en documentos agrupados en colecciones. [37], [38]

Las ventajas de este tipo de bases de datos son:

- Escalabilidad: En las bases de datos no relacionales la escalabilidad es sencilla pues solo hay que añadir más servidores (crecimiento horizontal), a diferencia de las relacionales.
- Bajos requerimientos: Para tener una base de datos no relacional, no se necesitan servidores con gran potencia.
- Los datos se pueden almacenar de distintas formas, como puede ser por columnas, clave-valor, documento, etc.
- Versatilidad: Añadir nuevos campos a una colección (una tabla en la base de datos relacional) es muy sencillo y no hay que cambiar nada en el resto de la estructura.

Las desventajas son:

- Atomicidad: Una de las características principales de las bases de datos relacionales, es la atomicidad, es decir que las operaciones de agregar, borrar o modificar datos, se hacen de forma completa o no se hacen, por lo que, si falla la operación, los datos modificados se vuelven a dejar como estaban. Esto no pasa en las bases de datos no

relacionales y debido a esto, los datos pueden ser diferentes para cada uno de los nodos lo que puede llevar a problemas a la hora de hacer consultas.

Al ser una base de datos no relacional, no está sujeto a las mismas reglas que las bases de datos relacionales por lo que no es necesario que las tablas estén normalizadas para evitar problemas futuros de manipulación, pues como se ha visto, en las bases de datos no relacionales esto no es un problema, pero se ha intentado modelar de forma que haya consistencia en los datos y sea más fácil a la hora de realizar consultas.

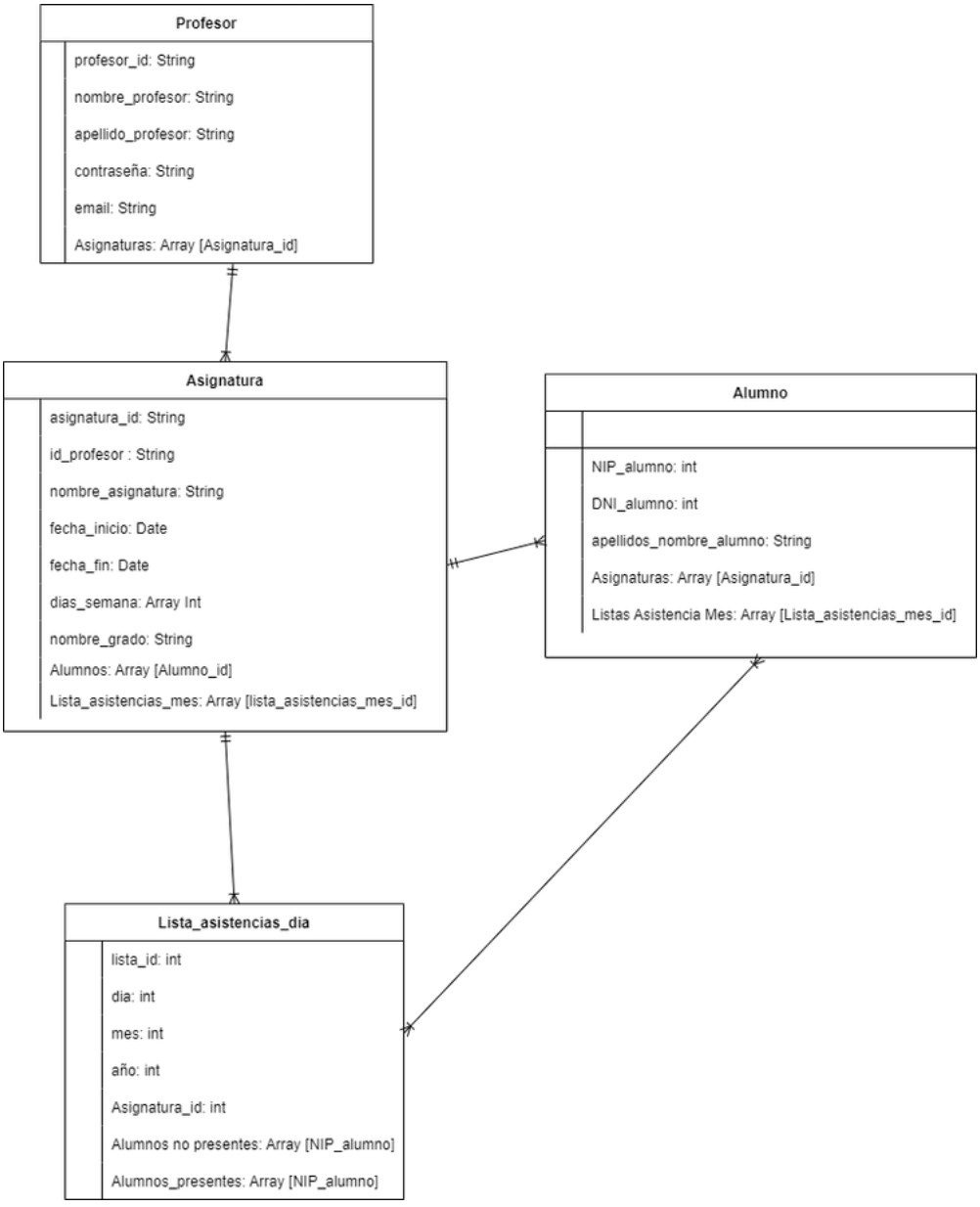

FIGURA 23 - ESQUEMA DE BASE DE DATOS

Una vez se ha hecho el análisis y diseño de la aplicación hay que pasar a la parte del desarrollo, la más extensa pues es donde la aplicación se hará realidad. Gracias al trabajo realizado en las fases previas se tiene más o menos una idea de cómo va a funcionar la APP, los requisitos que tiene que cumplir y su diseño.

5. Tecnologías Utilizadas

Antes de pasar a redactar los aspectos relevantes de la implementación, se van a indicar y explicar las tecnologías utilizadas para poder llevar a cabo dicha implementación.

5.1. Front-End

Para desarrollar el front-end de la aplicación se ha hecho uso del entorno de desarrollo Visual Studio 2022 con el editor de código Visual Studio Code, usando Dart como lenguaje de programación y Flutter como UI Framework para poder desarrollar aplicaciones multiplataforma.

Visual Studio 2022 y Visual Studio Code

Visual Studio 2022 es el IDE de Microsoft con una gran variedad de opciones como autocompletado de código, historial de cambios, trabajo en línea o depuración del código. Es un editor de código multiplataforma lanzado en 2015 que trabaja con diversos lenguajes, y ofrece soporte a la depuración, corrección de errores y ejemplos de código. Su principal característica es la gran variedad de plugins gratuitos de los que se dispone para facilitar la tarea al desarrollador. [39], [40]

Dart

Se trata de un lenguaje de programación orientado a objetos desarrollado por Google y lanzado en 2013, tanto para el desarrollo de programas del lado del cliente como aplicaciones web o móviles. Su lenguaje de programación es similar a otros como Kotlin, Swift o TypeScript lo que hace que los desarrolladores no tengan muchos problemas a la hora de aprenderlo. [41]

Sus principales características son: a) Programación asíncrona permite seguir ejecutando la aplicación mientras se está haciendo otra cosa, normalmente se usa para cuando se está esperando una información como leer de un fichero o guardar información. b) Hot Reload esto permite ver los cambios que se hacen en la aplicación reflejados de manera instantánea en el emulador o dispositivo que se esté utilizando para comprobar el funcionamiento de la aplicación. c) Compilación Dart cuenta con dos tipos de compilación: la compilación AOT (Ahead of Time) que marca los errores cuando se compila el código para después ejecutarlo y JIT (Just in Time) que permite ver los cambios instantáneamente gracias a Hot Reload y por lo tanto, solucionar errores de forma rápida, d) Sound Null Safety Una de sus principales características es Sound Null Safety, incluida en la versión 2.9, que significa que las variables nunca van a ser nulas por defecto a no ser que se especifique lo contrario. Evitando errores en el programa [42] y e) Gestor de paquetes Dart tiene un gestor de paquetes llamado Pub en el que otros desarrolladores pueden crear sus propios paquetes y compartirlos con la comunidad para que estos los utilicen en sus proyectos.

Flutter

Flutter, es un UI Framework desarrollado por Google en 2018. Cuando se dice que es un UI Framework se entiende que son una serie de herramientas que se pueden utilizar para desarrollar aplicaciones nativas de forma fácil, sencilla y rápida. Está orientado a la interfaz gráfica y esto se debe a que todo en Flutter es un Widget. Permite con el mismo código realizar tanto aplicaciones móviles como web y de escritorio comprendiendo así las 2 principales plataformas: ordenadores y móviles.

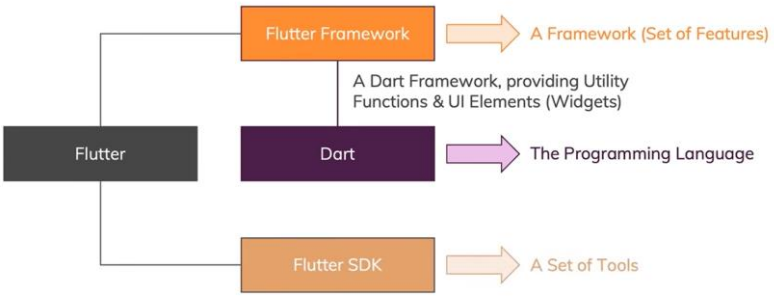

FIGURA 24 - ESQUEMA DE FLUTTER

Flutter hace uso del lenguaje de programación Dart también diseñado por Google lo que hace que se adapten perfectamente entre ellos. Esto permite entre otras cosas que las animaciones y transiciones corran a 60 fps (fotorgramas por segundo viéndose más fluidas.

Una de las principales ventajas de Flutter es su rendimiento, permitiendo velocidades similares a las de una aplicación nativa por delante de otras tecnologías multiplataforma como la popular React Native. Esto se debe a que Flutter, a diferencia de lo que ocurre con React Native, no tiene que hacer uso de "bridges", para convertir sus elementos creados con JavaScript a código nativo del sistema en el que se ejecutan, mediante el uso de JSON lo que hace que cuando se cuentan con muchos elementos, se pueden producir cuellos de botella. [43], [30]

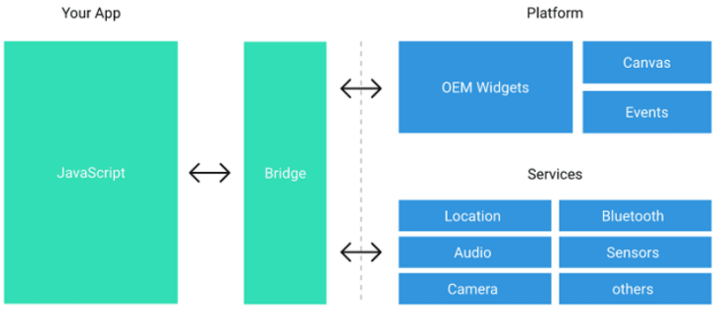

FIGURA 25 - CONVERTIR ELEMENTOS EN REACT NATIVE

Flutter dibuja los elementos sobre su propia herramienta gráfica llamada Skia, algo similar a lo que hacen las aplicaciones de videojuegos como Unity. [30]

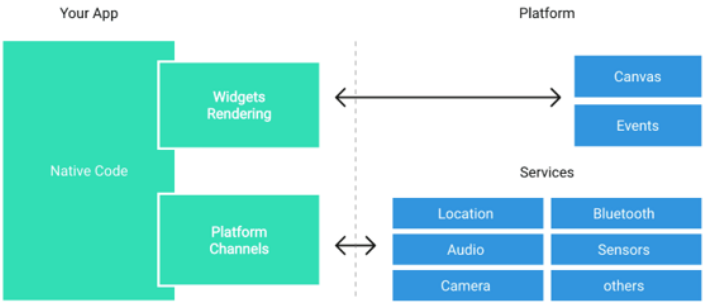

FIGURA 26 - CONVERTIR ELEMENTOS EN FLUTTER

De esta forma, la comunicación de la aplicación con el dispositivo se ve reducida lo que hace que las velocidades sean mayores y además la interfaz gráfica no se vea afectada por los cambios del sistema operativo pues no se depende del código nativo. Así mismo, también compila su código a ARM (Advanced RISC Machine) que es una arquitectura RISC (Reduced Instruction Set Computer) y se caracteriza por que los procesadores con esta arquitectura usan instrucciones pequeñas toman menos tiempo en comparación con la otra arquitectura de procesadores CICS (Computador con Conjunto de Instrucciones Complejas) que usa instrucciones complejas, o también puede compilarlo a librerías nativas haciendo que su rendimiento sea mayor. [47]

Bloc (Business Logic Component)

Flutter proporciona distintas formas de gestionar el estado de la aplicación como Provider, GetX, pero Bloc es la forma más completa de todas de gestionar el estado y la que nos permite una mayor escalabilidad, además de que se adapta perfectamente al patrón de diseño MVVM que se ha seleccionado para esta propuesta. [45], [46]

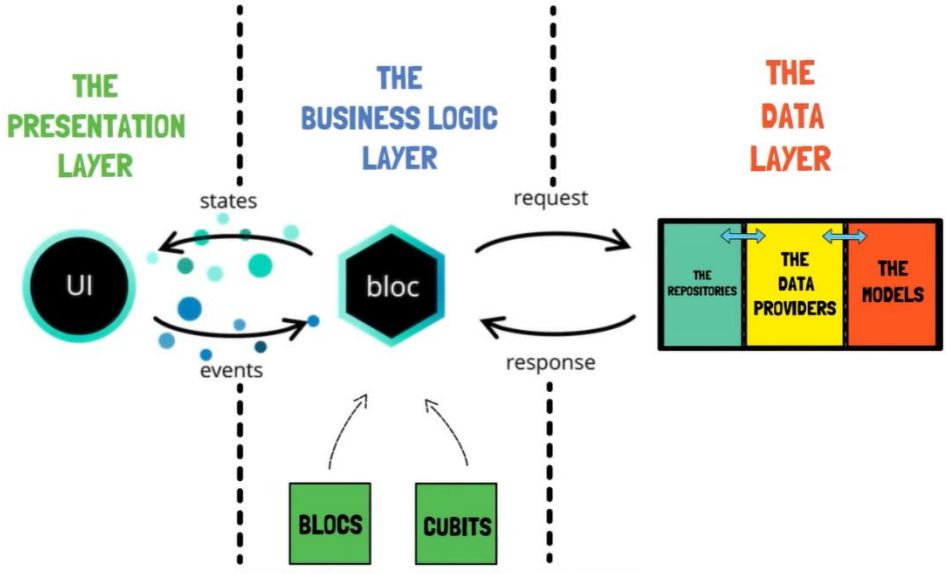

FIGURA 27 - ARQUITECTURA DE BLOC

Bloc es un paquete para Dart que permite separar la lógica de negocio de la capa de presentación (la interfaz gráfica) y de la capa de datos.

¿Cómo se comunican las capas?

La comunicación de Bloc con la capa de presentación se hace mediante eventos y estados. Los eventos se producen cuando el usuario toca algún elemento de la interfaz gráfica, este evento lo recibe Bloc que devolverá un estado el cual puede contener datos o no, que ha obtenido de la capa de datos, para que la interfaz se actualice de acuerdo con ese nuevo estado.

La comunicación de Bloc con la capa de datos se produce cuando previamente le ha llegado un evento de la interfaz y ésta requiere de datos para poder mostrar al usuario o tiene que hacer algún procedimiento, con lo que Bloc hace una petición a la capa de datos, concretamente a los repositorios, para obtener dichos datos.

La capa de datos se ha dividido en 3 partes

1. Repositorios: se comunican con Bloc y se encargan de obtener los datos permitiendo cambiar entre los distintos proveedores sin que esto afecte al resto de nuestro código.
2. Proveedores de datos: se encargan de obtener los datos de forma directa de distintas fuentes externas como son APIs, BBDD… y los transforman a los Modelos especificados para la aplicación.
3. Modelos: es una clase que se define en la aplicación para representar los datos.

5.2. Back-end

Para la **base de datos** se ha escogido Cloud Firestore una base de datos NoSql (no relacional) que es un servicio integrado en Firebase. [48]

Se ha escogido una base de datos NoSQL orientado a documentos y no una relacional debido a que la mayoría de las aplicaciones móviles utilizan este tipo de bases y además se integra perfectamente con Flutter ya que desde la página web de Firebase se indica cómo con unos pocos comandos se puede configurar Firebase en la aplicación Flutter propuesta en este TFG.

Al usar una base de datos no relacional, la estructura de los datos pasa a un segundo plano pues se pueden tener elementos duplicados y lo que le importa actualmente al usuario es la velocidad con la que se le muestra la información, y para esto, las bases de datos no relacionales son superiores a las relacionales en lo que se refiere a velocidad de lectura que es la operación que más va a hacer el usuario, pero no en tanto en la de escritura que es algo que no se hará tanto.

Otro punto a favor de las bases de datos no relacionales es la escalabilidad, mientras que una base de datos relacional está en un solo servidor y la única forma de escalarla, es hacerlo verticalmente con más memoria en dicho servidor, las bases de datos no relacionales se escalan de manera horizontal distribuyéndolas en más servidores lo que además hace que tengan una mayor disponibilidad pues si se cae un servidor, se tendrán otros disponibles.

A la hora de adaptar la base de datos conceptualizada en el apartado 4.5 de Diseño a la aplicación, se ha detectado la necesidad de incorporar 2 tablas adicionales, la de Clases para representar las sesiones que se imparten en cada asignatura y así mostrarlas en el calendario de la aplicación, y la tabla ListaAsistenciasMes para guardar todas las ListaAsistenciasDia de ese mes y que luego fuese más fácil obtener los datos a la hora de generar el PDF de asistencias.

Además, la tabla Clase tiene el listado de Alumnos ya incluido, para no tener que volver a pedirlo cuando se quiera pasar lista, y reducir así el número de lecturas, esto mismo se ha hecho para la tabla Asignatura la cual lleva las listasAsistenciasMes y si se quiere consultar alguna asistencia o generar un PDF, sólo se lee una vez de la base de datos.

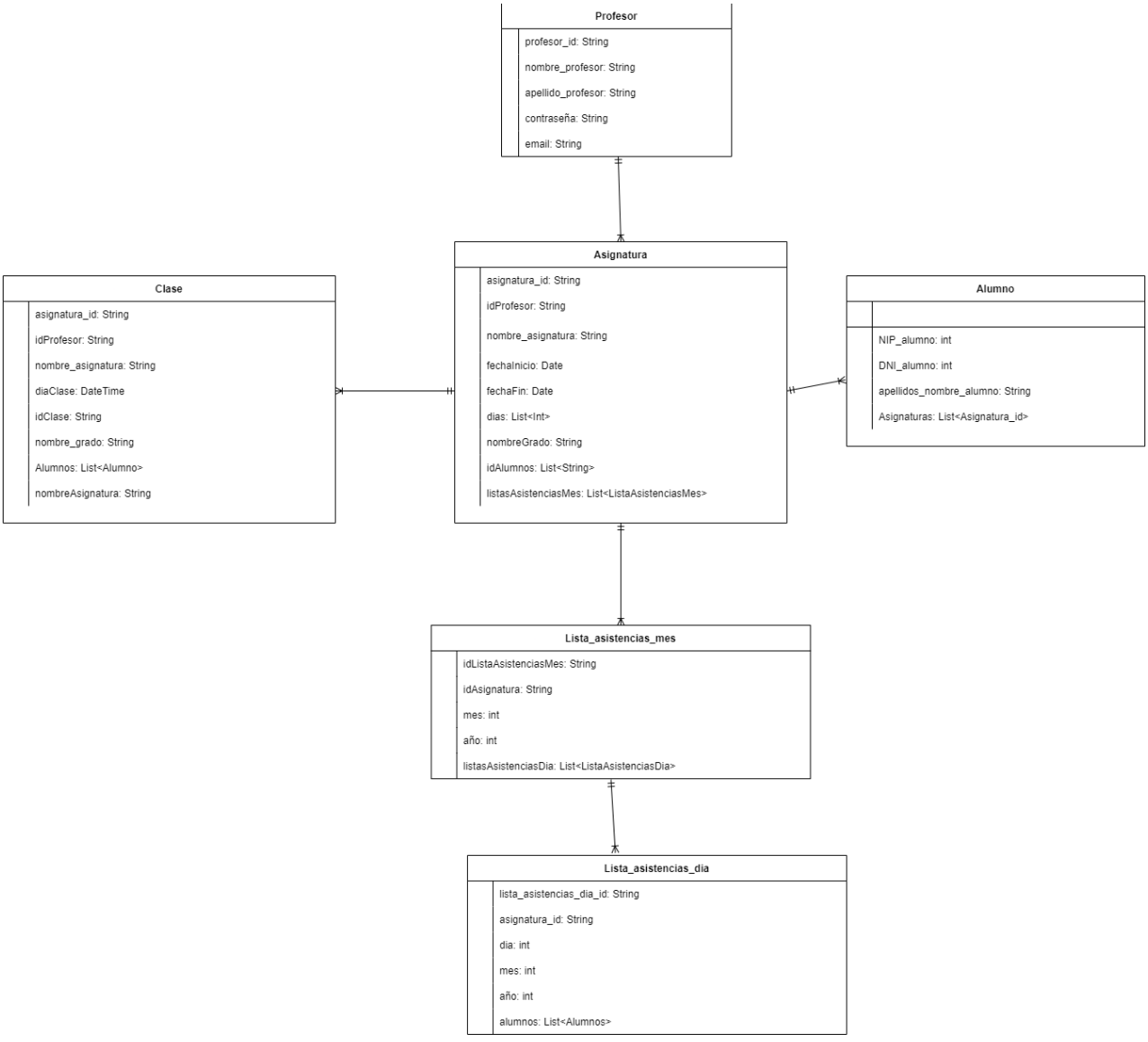

FIGURA 28 - VISTA ESQUEMA CONCEPTUAL FINAL

A parte de Cloud Firestore, Firebase proporciona otros servicios que son de utilidad y uno de ellos, es **Authentication**. [49]

Authentication permite con pocas líneas de código y de forma sencilla, gestionar el control y acceso de usuarios mediante distintos métodos como email/contraseña, servicios de terceros como Google, Facebook… así como restablecer contraseñas, verificación del usuario por SMS y otras funcionalidades. Al usar Authentication, la tabla de Profesor desaparece de la base de datos Cloud Firestore ya que los usuarios son gestionados por Authentication.

5.3. Otras tecnologías

Aparte de las tecnologías nombradas previamente, también se usarán archivos Excel para cargar los datos de los alumnos a la aplicación y archivos PDF para la descarga de datos.

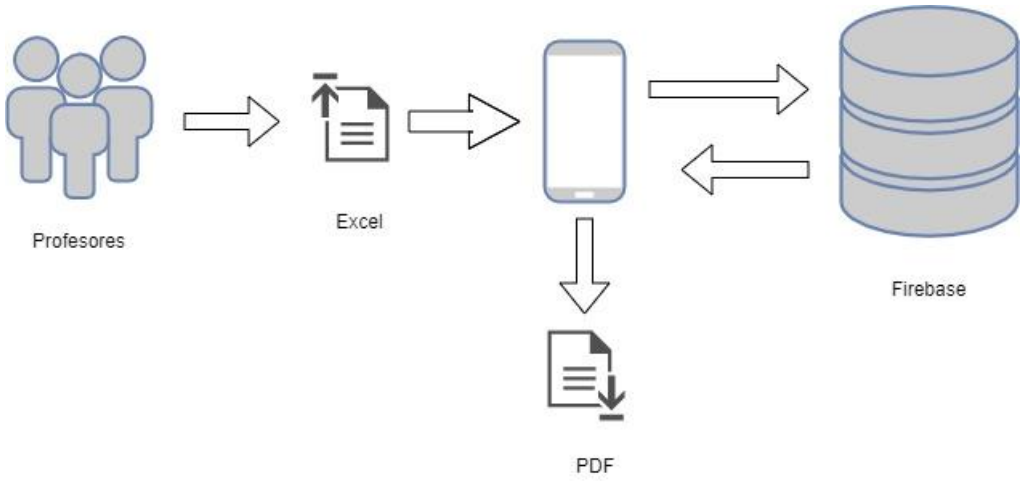

FIGURA 29 - TECNOLOGÍAS UTILIZADAS EN LA APP

6. Desarrollo de la Aplicación

En esta fase, se utilizan las tecnologías descritas previamente, así como el análisis que se ha realizado para realizar el desarrollo de la aplicación en un entorno real.

A continuación, se describen las funcionalidades de la aplicación en su versión final.

6.1. Apertura de la aplicación

Cuando el usuario abre la aplicación, verá una serie de pantallas donde se le explican las funcionalidades que tiene la aplicación. Mientras el usuario haya iniciado sesión esta pantalla no volverá a aparecer y una vez que cierre sesión o borre la cuenta, volverá a verlas. Para realizar la introducción se ha usado el paquete intro_slider. [50]

6.2. Autenticación de los usuarios

Lo primero que se ha implementado en la aplicación ha sido el control de autenticación de los usuarios mediante Firebase Authentication permitiendo la autenticación del usuario tanto por email y correo como por Google SignIn de forma sencilla.

Cuando se abre la aplicación, se comprueba si el usuario ya estaba autenticado previamente y no ha cerrado sesión, en cuyo caso se accede directamente a la vista del calendario. Si no está autenticado, se le mostrará la pantalla para iniciar sesión desde la que puede iniciar sesión mediante correo y contraseña o con Google (en este último caso no hace falta registrarse). Si el usuario no tiene una cuenta, puede ir a la vista de registro para crear una cuenta nueva, aquí se comprueba que el usuario ha introducido una contraseña de al menos 8 caracteres y se pide que la vuelva a introducir para confirmar que se acuerda de la contraseña.

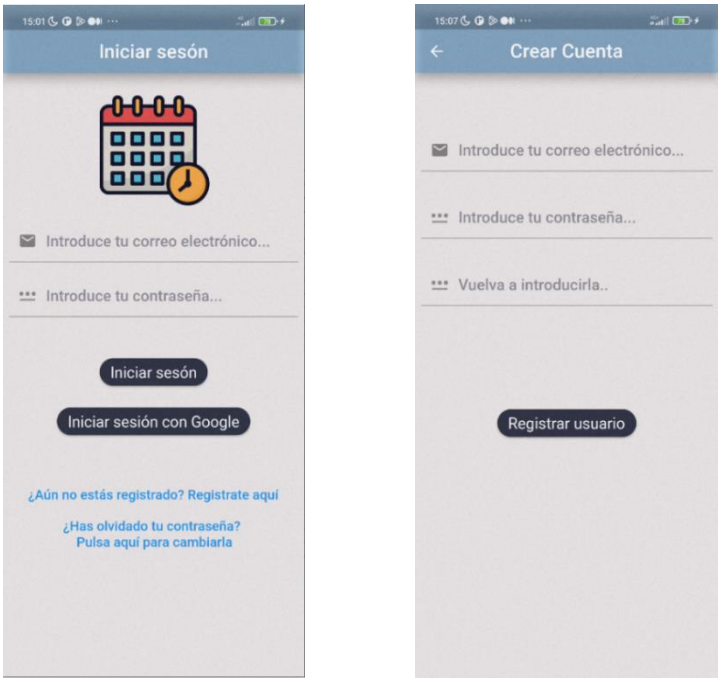

FIGURA 30 - VISTA DE INICIAR SESIÓN, CREAR CUENTA Y VERIFICAR CORREO

Para controlar que el usuario no deja ningún campo vacío, así como que los datos que se introducen están en el formato válido, se ha utilizado el paquete flutter_form_builder [51] que ha simplificado la implementación de estas comprobaciones.

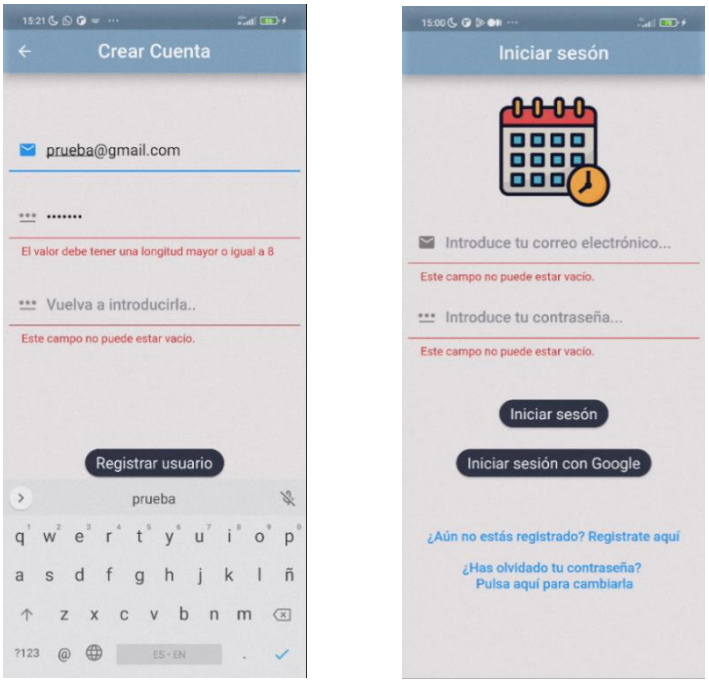

FIGURA 31 - COMPROBACIÓN DE CAMPOS DE TEXTO EN INICIAR SESIÓN Y REGISTRAR

6.3. Verificar correo electrónico

Para asegurarnos que el usuario es el propietario del correo electrónico con el que se ha registrado, se envía a dicho correo un email de verificación y el usuario no podrá iniciar sesión hasta que no lo haya confirmado.

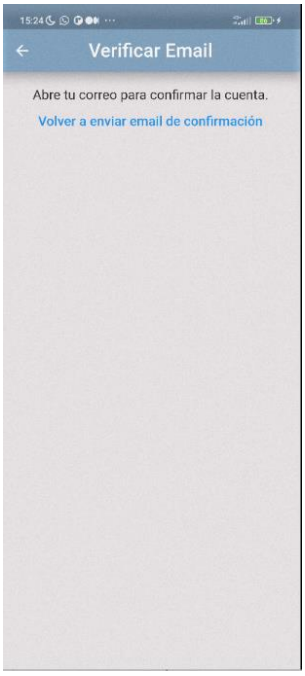

FIGURA 32 - VISTA DE LA VERIFICACIÓN DEL CORREO

6.4. Nueva contraseña

En caso de que se le haya olvidado la contraseña al usuario, puede seleccionar en la vista de Iniciar sesión, la opción de "¿Has olvidado tu contraseña?" La cual lleva a la vista para obtener una nueva contraseña mediante el correo electrónico. Una vez se pulsa en "Contraseña nueva" llegará un correo para cambiar la contraseña.

FIGURA 33 - VISTA DE CAMBIAR CONTRASEÑA

6.5. Calendario y vista Clases

Una vez que el usuario se ha autenticado, se le muestra el calendario en el mes actual y en la parte inferior aparecen las clases del día actual.

El usuario puede seleccionar otros días del mes para ver qué clases tiene en dicho día, así como desplazarse a otros meses.

Cuando se ha pasado lista para una asignatura en un día concreto, el botón de esa clase estará deshabilitado para evitar que se vuelva a pasar lista de nuevo. Para la creación del calendario se ha usado el paquete table_calendar. [52]

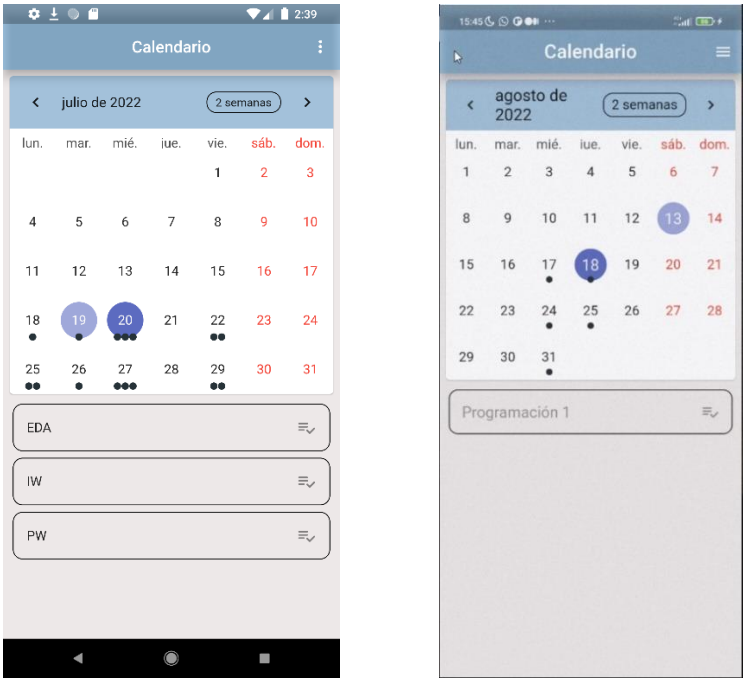

FIGURA 34 - VISTA DEL CALENDARIO Y CLASES Y VISTA DE CLASE DESHABILITADA

6.6. Crear Asignatura, importar alumnos desde Excel y crear clases

La creación de la asignatura se controla al igual que el inicio de sesión y registro mediante formularios para comprobar que no se dejan campos vacíos y que la Fecha de fin de la asignatura no es anterior a la de inicio.

Al seleccionar “Cargar lista de alumnos”, se abre el selector de archivos del teléfono en cuestión, permitiendo seleccionar el archivo .xlsx del que se desea cargar los alumnos.

Para convertir el archivo Excel a datos a la APP, se ha usado el paquete Excel [53] y una vez que se tienen los alumnos cargados, se puede añadir la asignatura.

Por último, se crean las Clases para la Asignatura que son los eventos que se van a mostrar en el calendario.

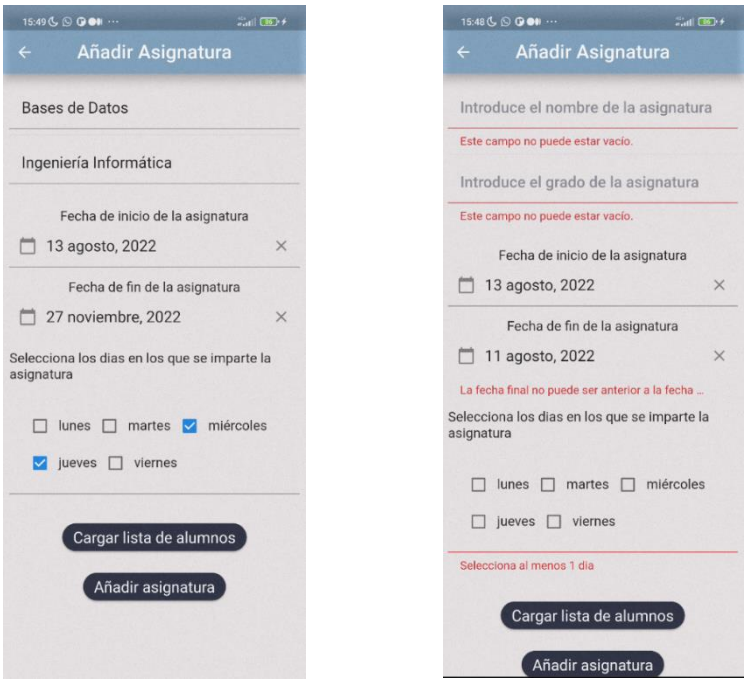

FIGURA 35 - VISTA DE AÑADIR ASIGNATURA

6.7. Pasar Lista

Desde la vista del calendario, cuando el usuario selecciona una de las asignaturas que aparecen en la parte inferior, se abre una nueva ventana para pasar lista en el día que estaba seleccionado.

Por defecto, todos los alumnos aparecen como presentes y al seleccionarlos cambian a no presentes, si se vuelven a seleccionar estarán otra vez como presentes.

Una vez que se ha pasado lista, el botón para pasar lista (de ese día) se deshabilita para evitar que el usuario vuelva a pasar lista.

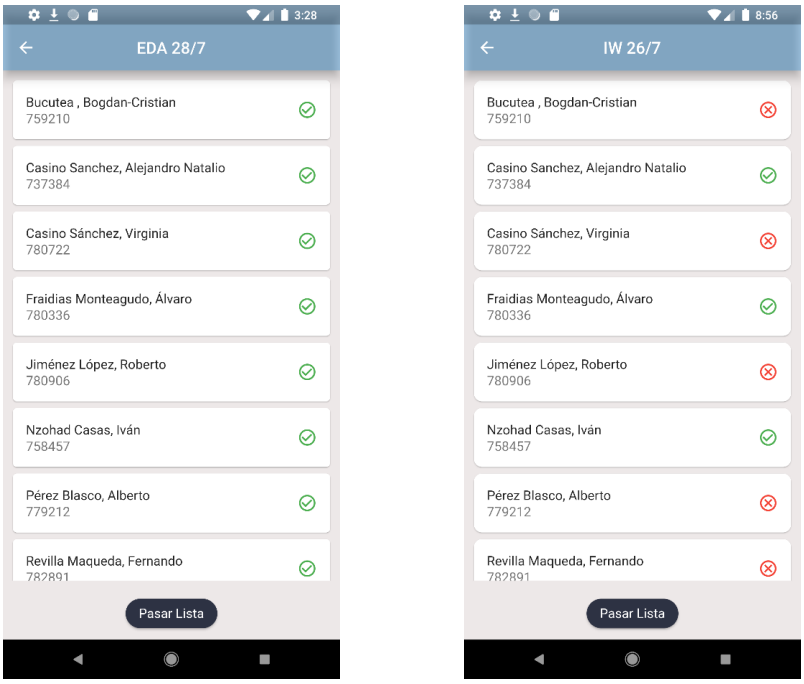

FIGURA 36 - VISTA PASAR LISTA

6.8. Listado de asignaturas y eliminar asignaturas

Desde la vista del calendario, se tiene en la parte superior derecha un menú para acceder a las asignaturas, una vez dentro, se ven las asignaturas ordenadas por año.

Si se desplaza la asignatura a la derecha se puede borrar dicha asignatura, esto hace que se borren las clases y listas de asistencia que se tengan asociadas a dicha asignatura y se borra la asignatura del alumno.

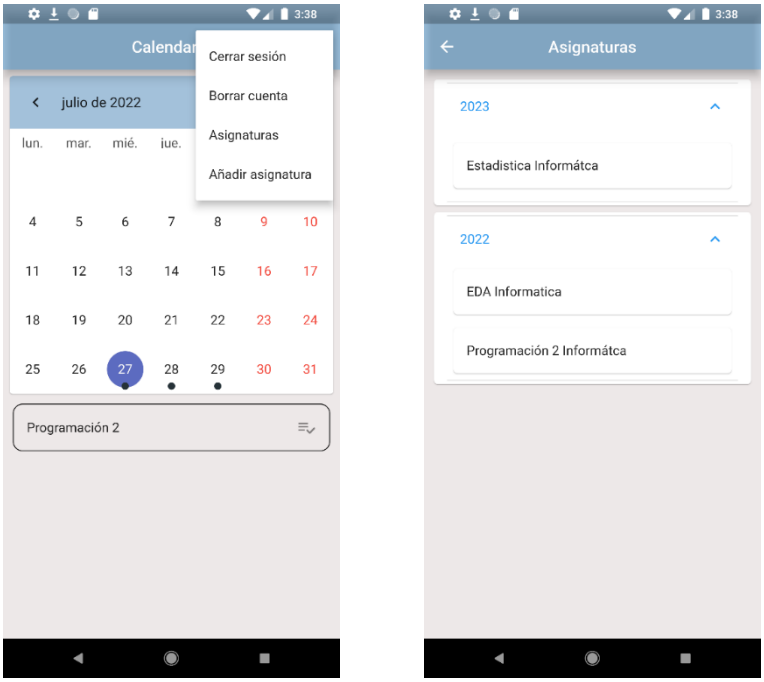

FIGURA 37 - VISTA MENÚ OPCIONES Y VISTA LISTA ASIGNATURAS

FIGURA 38 - VISTA BORRAR ASIGNATURA

6.9. Opciones de asignatura, consultar y editar listas de asistencia

Una vez se está dentro de la asignatura, se tienen 2 opciones:

- Generar las listas de asistencia
- Generar PDF

Si se accede a generar las listas de asistencia, se pueden ver las listas de asistencia para dicha asignatura agrupadas por mes, posteriormente se pueden desplegar para ver las listas de asistencia de cada día y si se desplaza a la derecha en el día, se puede editar dicha lista de asistencias.

FIGURA 39 - VISTA OPCIONES ASIGNATURAS Y LISTAS DE ASISTENCIAS

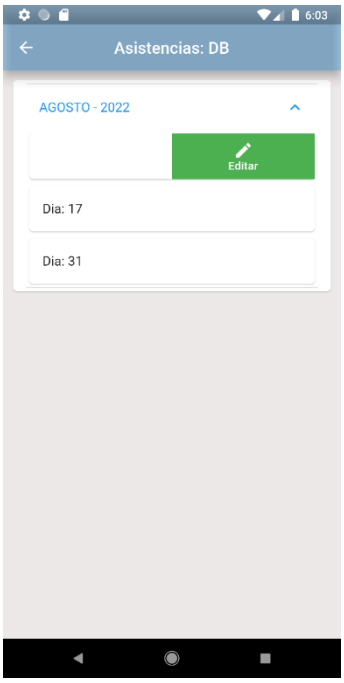

FIGURA 40 - VISTA PARA EDITAR LISTA ASISTENCIA

6.10. Generar PDF

En este caso, se selecciona la opción de Generar PDF, que mostrará el listado de meses disponibles para generar el PDF. Para que aparezca un mes en el cual se puede generar un PDF, se tiene que haber pasado lista en al menos una ocasión para dicho mes. Para generar el PDF se ha usado el paquete pdf [54]

Cuando se selecciona el mes, se mostrará la previsualización del PDF, mediante el uso del paquete printing [55] y se tiene la opción de guardarlo en el dispositivo móvil o imprimirlo.

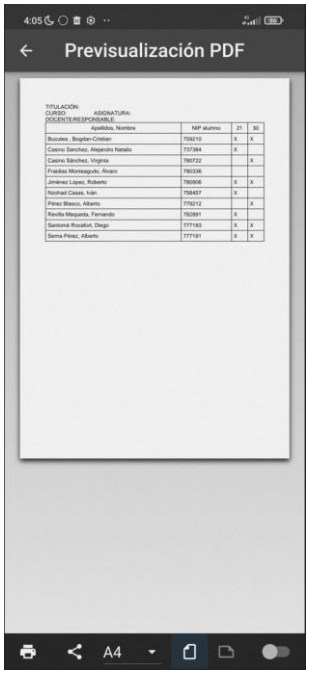

FIGURA 41 - VISUALIZACIÓN PDF

La aplicación se encuentra disponible en: https://github.com/720707/TFG_JRAMOS.

7. Pruebas

La fase de pruebas es un apartado indispensable en cualquier proyecto software con el fin de comprobar el correcto funcionamiento de la aplicación, así como descubrir posibles errores que se hayan pasado por alto a la hora de realizar una aplicación.

Además, gracias al uso de pruebas, se puede garantizar que los cambios que se realizan en alguna parte de la APP no provoquen fallos inesperados en otros lugares y en caso de que esto ocurriese, las pruebas permiten identificar de forma rápida donde ocurren.

En este caso, se van a realizar 3 tipos de pruebas para confirmar que la aplicación funciona correctamente:

- Pruebas de unidad
- Pruebas de widgets
- Pruebas de rutas

7.1. Pruebas de unidad

Las pruebas de unidad como su nombre indica, se utilizan para comprobar el correcto funcionamiento de una unidad, que en este caso es un método y en Flutter tienen el siguiente aspecto. [56], [58], [60]

FIGURA 42 - ESQUELETO DE UNA PRUEBA DE UNIDAD

Está divido en 3 partes conocidas, por sus nombres en inglés:

- La primera parte llamada en inglés “arrange”, es donde se definen las variables y se indica el resultado que darán las llamadas a los servicios que se han simulado.
- La segunda parte es “Act” donde se llama a la función que se va a testear
- Y la tercera parte “expect” donde se comprueba que los resultados son los deseados

Mocks (Simular servicios)

Al crear aplicaciones, se suelen utilizar servicios de terceros como pueden ser APIs, Bases de Datos… y si hubiera que utilizarlas a la hora de realizar los test, podrían surgir varios problemas o dificultades que hiciesen que los test fallasen por causas ajenas a nuestra APP

como puede ser un fallo de conexión a internet, que el dato que se intenta introducir exista previamente en la base de datos… Para solucionar estos problemas se han utilizado diversos paquetes que permite "simular" las llamadas a funciones indicando cual es el resultado que se espera recibir como Mocktail para simular de forma genérica, FakeFirebaseFirestore para simular una base de datos Cloud Firestore o MockFirebaseAuth para simular la autenticación en Firebase.

Las **pruebas en Bloc** [61] se han hecho mediante el paquete bloc_test ya que, al haber utilizado bloc para gestionar los estados, este paquete facilita la creación de las pruebas orientadas a Bloc.

Las pruebas tienen el siguiente aspecto:

```
blocTest<NombreBloc, EstadoBloc>(
  'Emite [Estado] cuando se llama a EventoBloc',
  setUp: () {
    when(() => mockRepositorio.tarea())
      .thenAnswer((_) => respuesta);
  },
  buid: () => NombreBloc();
  act: (bloc) => bloc.add(EventoBloc),
  expect: () => const <EstadoBloc>[Estado],
);
```

FIGURA 43 - ESQUELETO DE PRUEBA PARA BLOC

En ellos se distinguen las siguientes partes:

- Primero se indica el bloc y estados que va a soportar y una breve descripción de la prueba que se va a realizar.
- SetUp: Aquí se define las llamadas que se realizan en el test simulando como se ejecutaría el evento y los datos que se obtendrían tras cada llamada mediante la función thenAswer()
- Build: Aquí se pasa el bloc que se ha especificado al inicio de la prueba
- Act: Se indica el evento de bloc que se manda
- Expect: Se indica el resultado que se espera, que suele ser un valor o un atributo, en este caso al estar usando bloc, lo que se espera es un estado o lista de estados que se van a ir emitiendo al procesar el evento que le ha llegado en el act.

7.2. Pruebas de Widgets

Otra parte importante de una aplicación móvil es el apartado visual ya que es imprescindible que la aplicación se vea correctamente y que el usuario pueda interactuar con ella sin problemas. [59]

Para comprobar esto, se han utilizado las pruebas de widgets, con las que se ha podido analizar que los distintos widgets que forman las pantallas aparecen de forma correcta, se puede interactuar correctamente con ellos escribiendo texto, presionándolos o haciendo que manden eventos a Bloc entre otras acciones.

7.3. Pruebas de Rutas

Para comprobar la correcta navegación entre pantallas [61], se ha utilizado el paquete mockingjay [62] que ha permitido "simular" la navegación entre pantallas para de esa forma, comprobar que las rutas que se han definido funcionan correctamente y que se envían los eventos para cambiar de ventana como son *pushNamed()*, *popAndPushNamed()*, etc.

8. Licencia Software y Documental

FIGURA 44 - GFDL

FIGURA 45 - GPL

9. Conclusiones y Trabajo Futuro

Tras la realización del trabajo, se ha obtenido una aplicación en la cual, los usuarios ven un calendario con las clases que tienen pendientes en ese día y pueden pasar lista a sus alumnos, pueden añadir nuevas asignaturas y añadir alumnos mediante un fichero Excel y finalmente pueden generar un PDF con el listado de asistencias para un mes.

La realización de este trabajo me ha permitido aprender un nuevo lenguaje de programación que no conocía previamente que es Dart y el uso del Framework Flutter, con los cuales se pueden realizar aplicaciones multiplataforma, lo cual es cada vez más necesario por la necesidad de estar conectados en todo momento ya sea con un ordenador, un móvil o una Tablet y por lo tanto cualquier aplicación tiene que estar disponible en todos estos dispositivos.

También, se ha podido realizar una aplicación desde cero estando involucrado en todas sus partes del desarrollo (backend y frontend) y darse cuenta de la importancia de tener bien estructurada la base de datos puesto que la estructura inicial sufrió varios cambios a lo largo del desarrollo para poder adaptarse a las necesidades de la aplicación como es la creación de la tabla Clases, necesaria a la hora de poder mostrar las clases en el calendario.

Esto me ha permitido ser consciente de la importancia de una estructuración correcta de nuestro proyecto a la hora de realizar la aplicación para poder añadir nuevas funcionalidades de forma sencilla a lo cual se ha estado obligado desde el principio al usar el paquete Bloc para gestionar los estados. Esto también ha influido a la hora de poder realizar pruebas ya que la realización de pruebas es algo que se había hecho previamente de forma escasa y no con tanta profundidad como en esta aplicación, para todos los elementos de la misma.

Gracias a esto, se ha visto su importancia, ya que con una buena serie de pruebas se está más cómodo ante cambios ya que se puede corroborar de forma sencilla y rápida si los cambios han producido fallos en el código.

Por otro lado, al usar paquetes de terceros como table_calendar para la vista del calendario o pdf para generar el pdf, se ha facilitado el desarrollo de la aplicación, pero esto hace que se dependa de paquetes que pueden quedar desactualizados lo cual obligaría a buscar alternativas o desarrollarla los nuestros desde cero.

La aplicación no se ha podido probar en iOS puesto que se necesita tener un Mac para poder usar Xcode y configurar así la aplicación, pero por lo que se ha visto no tendría que haber ningún problema en usar esta aplicación en iOS.

La aplicación actualmente cumple los objetivos fijados para este trabajo, pero se pueden añadir nuevas funcionalidades como son:

- La obtención directa de los datos de los alumnos de forma directa desde la web (sia.unizar.es) de manera que ya no sería necesario descargar el fichero Excel en el dispositivo.
- También sería interesante añadir la hora en las clases para poder enviar una notificación al móvil cuando se acerque la hora de la sesión.
- Hacer uso de una base de datos local como MySQL pues actualmente se usa Firebase y aunque tiene un servicio gratuito, está limitado por el número de usos y en caso de que la aplicación fuese usada por mucha gente se tendría que pagar.

Referencias Bibliográficas

[1] Miguel Fragoso, “Sistema de control de aforo de alta precisión” [Online]. Disponible: https://sonotrack.com/blog/sistema-control-de-aforo-personas/

[2] “Technologies used in the Attendance management” [Online]. Disponible: https://www.zimyo.com/insights/technologies-used-in-the-attendance-management/

[3] 5 Most Popular Student Attendance Solutions [Online]. Disponible: https://www.seatssoftware.com/2019/05/21/5-most-popular-student-attendance-solutions/

[4] Federico MIchele Facca, A schema representing the architecture of the ReActive Web System. [Online]. Disponible: https://www.researchgate.net/figure/A-schema-representing-the-architecture-of-the-ReActive-Web-System_fig2_220940082

[5] Raquel Maluenda de Vega, Tipos de desarrollo de aplicaciones web: ejemplos y características [Online]. Disponible: https://profile.es/blog/desarrollo-aplicaciones-web/

[6] DATAREPORTAL, Digital around the world [Online]. Disponible: https://datareportal.com/global-digital-overview

[7] ¿Qué es Backend y Frontend? [Online]. Disponible: https://ed.team/comunidad/que-es-backend-y-frotend

[8] Front-End y Back-End - Definición, Concepto y Qué es [Online]. Disponible: https://www.definicionabc.com/tecnologia/frontend-backend.php

[9] David Bernal González, Principales tipos de aplicaciones móviles: ventajas, desventajas y ejemplos [Online]. Disponible: https://profile.es/blog/tipos-aplicaciones-moviles-ventajas-ejemplos/

[10] Aplicaciones web: Ventajas y desventajas - ¿Cuáles son las que más se desarrollan? [Online]. Disponible: https://thecloud.group/aplicaciones-web-ventajas-desventajas/

[11] Jaime S. Nielfa, Web App: Qué Es, Ventajas, Características y Ejemplos [Online]. Disponible: https://scoreapps.com/blog/es/web-app/

[12] Progressive Web Apps: Advantages and Disadvantages [Online]. Disponible: https://brainhub.eu/library/progressive-web-apps-advantages-disadvantages/

[13] Progressive Web Apps: Core Features, Architecture, Pros and Cons [Online]. Disponible: https://www.altexsoft.com/blog/engineering/progressive-web-apps/

[14] rosario3, Cinco apps para controlar la asistencia a clase [Online]. Disponible: https://www.rosario3.com/noticias/5-apps-para-controlar-la-asistencia-a-clase-20170815-0014.html

[15] Mobile App Download Statistics & Usage Statistics (2022) [Online]. Disponible: https://buildfire.com/app-statistics/#:~:text=There%20are%202.87%20million%20apps,on%20the%20Google%20Play%20Store

[16] Educación 3.0, 17 herramientas para el control de asistencia en el aula [online]. Disponible: https://www.educaciontrespuntocero.com/recursos/herramientas-control-asistencia/

[17] Alexia manuales [Online]. Disponible: http://www.cospa-agilmic.com/newsletter/Manuales/manuales_accesos.html

[18] Dinantia [Online]. Disponible: https://www.dinantia.com/es

[19] Google Play [Online]. Disponible: https://play.google.com/store/apps/details?id=com.ferid.app.classroom&hl=en&gl=US

[20] Goople Play [Online]. Disponible: https://play.google.com/store/apps/details?id=com.caracterizate.pasalista&hl=en&gl=US
[21] CleverTap, “What Are the Different Types of Mobile Apps? And How Do You Choose?” [online]. Disponible: https://clevertap.com/blog/types-of-mobile-apps/
[22] Abel Camarena, “Aplicaciones Web Progresivas y Service Workers” [Online]. https://www.espai.es/blog/2018/09/aplicaciones-web-progresivas-y-service-workers/
[23] Saurabh Barot, “Best Hybrid App Development Framework in 2022” [Online]. Disponible: https://aglowiditsolutions.com/blog/best-hybrid-app-development-framework/
[24] Reglamento general de protección de datos [Online]. Disponible: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A02016R0679-20160504
[25] ISO/IEC 25010 [Online]. Disponible: https://iso25000.com/index.php/normas-iso-25000/iso-25010
[26] Sommerville, Ian. Ingeniería del software / Ian Sommerville ; traducción, María Isabel Alfonso Galipienso ... [et al.] . 7ª ed., reimpr. Madrid [etc.] : Pearson Educación, 2009
[27] UML [Online]. Disponible: https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado
[28] Flutter development: advantages and disadvantages [Online]. Disponible: https://sannacode.com/blog/advantages-and-disadvantages-using-flutter
[29] Flutter vs Native vs React-Native: Examining performance [Online]. Disponible: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980
[30] Andrii Bondarenko, React Native vs Flutter: Detailed Comparison [Online], Disponible: https://stormotion.io/blog/react-native-vs-flutter-detailed-comparison/
[31] Google, Firebase [Online]. Disponible: https://firebase.google.com/products-build
[32] Goolge, “Almacena y entrega contenido con facilidad “[Online] https://firebase.google.com/products/storage
[33]Jitesh Mohite “Flutter: MVVM Arquirectura” [Online]. Disponible: https://medium.com/flutterworld/flutter-mvvm-architecture-f8bed2521958
[34] Introducción a DDD y arquitectura hexagonal con un ejemplo de aplicación en Java [Online]. Disponible: https://picodotdev.github.io/blog-bitix/2021/02/introduccion-a-ddd-y-arquitectura-hexagonal-con-un-ejemplo-de-aplicacion-en-java/
[35] Arquitectura en Capas [Online]. Disponible: https://www.ecured.cu/Arquitectura_en_Capas
[36] ASP.NET MVC arquitectura DDD(Domain Driven Design) [Online]. Disponible: https://es.stackoverflow.com/questions/41889/asp-net-mvc-arquitectura-ddddomain-driven-design
[37] Sara Martín. “Bases de datos NoSQL: Guía definitiva” [Online]. Disponible: https://pandorafms.com/blog/es/bases-de-datos-nosql/
[38] “Base de datos no relacional. ¿Qué es? Características y ejemplos” [Online]. Disponible: https://ayudaleyprotecciondatos.es/bases-de-datos/no-relacional/
[39] “Visual Studio 2022” [Online]. Disponible: https://visualstudio.microsoft.com/es/vs/
[40] “Visual Studio Code” [Online]. Disponible: https://esflutter.dev/docs/development/tools/vs-code
[41] “Dart”. [Online]. Disponible: https://en.wikipedia.org/wiki/Dart_(programming_language)

[42] Bob Nystrom. “Understanding null safety” [Online]. Disponible: https://dart.dev/null-safety/understanding-null-safety
[43] “Flutter vs Dart” [Online]. Disponible: https://blog.back4app.com/flutter-vs-dart/#What_is_Dart
[44] Flutter. “List of state management approaches“ [Online]. Disponible: https://docs.flutter.dev/development/data-and-backend/state-mgmt/options
[45] “Getting Started with Flutter Bloc Pattern” [Online]. Disponible: https://www.mitrais.com/news-updates/getting-started-with-flutter-bloc-pattern/
[46] Flutterly . “The Best Flutter Bloc Complete Course - Visualise, Understand, Learn & Practice Bloc Concepts” [Online]. Disponible: https://www.youtube.com/watch?v=THCkkQ-V1-8
[47] “Arquitectura ARM” [Online]. Disponible: https://es.wikipedia.org/wiki/Arquitectura_ARM
[48] Google, “Firebase Documentation” [Online]. Disponible: https://firebase.google.com/docs/firestore
[49] Google, “Firebase Authentication” [Online]. Disponible: https://firebase.google.com/docs/auth
[50] intro_slider 3.0.10 [Online]. Disponible: https://pub.dev/packages/intro_slider
[51] flutter_form_builder 7.6.0 [Online]. Disponible: https://pub.dev/packages/flutter_form_builder
[52] table_calendar 3.0.6 [Online]. Disponible: https://pub.dev/packages/table_calendar
[53] excel 1.1.5 [Online]. Disponible: https://pub.dev/packages/excel
[54] pdf 3.8.3 [Online]. Disponible: https://pub.dev/packages/pdf
[55] printing 5.9.2 [Online]. Disponible: https://pub.dev/packages/printing
[56] Flutter, “Testing”[Online]. Disponible: https://docs.flutter.dev/cookbook/testing
[57] “Testing In Flutter: Widget Test”[Online]. Disponible: https://dhruvnakum.xyz/testing-in-flutter-widget-test
[58] “Flutter Testing Guide for Beginners - Part 1: Unit Tests & Setup” [Online]. Disponible: https://www.youtube.com/watch?v=hUAUAkIZmX0&t=1s
[59] “Flutter Testing Guide for Beginners – Part 2: Widget & Integration Tests” [Online]. Disponible: https://www.youtube.com/watch?v=Ghqry5dtgH4
[60] “Flutter Unit Test – Fundamentals” [Online]. Disponible: https://www.youtube.com/watch?v=UAAAwFH4A2w&t=1782s
[61]“OktoberTest · Testing con bloc_test package “ [Online]. Disponible: https://www.youtube.com/watch?v=RKSOoybqAyA
[62] mockingjay 0.3.0 [Online]. Disponible: https://pub.dev/packages/mockingjay
[63] get_it 7.2.0 [Online]. Disponible: https://pub.dev/packages/get_it
[64] BOOCH, G. et al.UML : el lenguaje unificado de modelado : guía del usuario. 2ª ed. [s. l.]: Addison-Wesley, 2012. ISBN 9788478290765.

ANEXOS

Anexo I. Inyección de Dependencias

Para la inyección de dependencias (pasar las dependencias de una clase por parámetros al constructor), se ha usado el paquete get_it, con el cual se puede declarar si una clase va a ser *Singleton* o *Factory*.

Primero se inicializa *GetIt* [63] con *GetIt.instance* y posteriormente se indica cómo se va a inicializar cada dependencia con *sl.registerLazySingleton* para que sea un *Singleton*, pero que solo se registrará cuando se necesite utilizar y en los parámetros del constructor de la clase se pasa simplemente *sl()*, los parámetros que recibe la clase en el constructor también se tienen que declarar de la misma forma para que funcione correctamente.

```
final sl = GetIt.instance;
void inicializar() {
  inicializarRepostiorio();
  inicializarProveedores();
  inicializarUilidades();
  inicializarServiciosTerceros();
}

void inicializarRepostiorio() {
  sl.registerLazySingleton<RepositorioAutenticacion>(
      () => RepositorioAutenticacion(proveedorAutenticacion: sl()));
  sl.registerLazySingleton<RepositorioAlumnos>(
      () => RepositorioAlumnos(proveedorAlumnos: sl()));
  sl.registerLazySingleton<RepositorioAsignaturas>(
      () => RepositorioAsignaturas(proveedorAsignaturas: sl()));
  sl.registerLazySingleton<RepositorioClases>(
      () => RepositorioClases(proveedorClases: sl()));
  sl.registerLazySingleton<RepositorioListasAsistencia>(
      () => RepositorioListasAsistencia(proveedorListasAsistencia:
sl()));
}

void inicializarUilidades() {
  sl.registerLazySingleton<GeneradorClases>(() => GeneradorClases());
  sl.registerLazySingleton<GeneradorPdf>(() => GeneradorPdf());
  sl.registerLazySingleton<ProcesadorExcel>(() => ProcesadorExcel());
```

```
}


void inicializarProveedores() {
  sl.registerLazySingleton<ProveedorAlumnos>(
      () => ProveedorAlumnosFirebase(firestore: sl()));
  sl.registerLazySingleton<ProveedorAutenticacion>(
      () => ProveedorAutenticacionFirebase(firebaseAuth: sl()));
  sl.registerLazySingleton<ProveedorAsignaturas>(
      () => ProveedorAsignaturasFirebase(firestore: sl()));
  sl.registerLazySingleton<ProveedorClases>(
      () => ProveedorClasesFirebase(firestore: sl()));
  sl.registerLazySingleton<ProveedorListasAsistencia>(
      () => ProveedorListasAsistenciaFirebase(firestore: sl()));
}


void inicializarServiciosTerceros() {
  sl.registerLazySingleton<FirebaseFirestore>(() =>
FirebaseFirestore.instance);
  sl.registerLazySingleton<FirebaseAuth>(() => FirebaseAuth.instance);
}
```

ANEXO 1 – EJEMPLO DE INYECCIÓN DE DEPENDENCIAS

Anexo II. Paso de parámetros entre pantallas y navegación entre pantallas

El paso de parámetros entre pantallas ha sido importante a la hora de reducir el número de consultas a la base de datos. De esta forma, al ir al listado de asignaturas y obtener dichas asignaturas, se obtenían también las listas de asistencia de manera que al ir a la siguiente pantalla para consultar cada lista de asistencia y al volver a atrás para ver otra lista de asistencia distinta, no se realizaban nuevas consultas a la base de datos, sino que se usaban los mismos valores obtenidos previamente.

Para pasar los parámetros, al llamar a la función *pushNamed()* se tiene la propiedad arguments que permite pasar valores entre pantallas, si se quieren pasar varios argumentos se tiene que crear una clase con la que pasarlos como se ha hecho con *ArgumentosPantalla.*

```
class _ConsultaListasMesesParaPdfVistaState
    extends State<ConsultaListasMesesParaPdfVista> {
  @override
  Widget build(BuildContext context) {
```

```
    return Scaffold(
      appBar: MiAppBar(
        titulo: "PDF - ${widget._asignatura.nombreAsignatura}",
        ruta: opcionesAsignaturasRuta,
        argumentos: ArgumentosPantalla(
            asignatura: widget._asignatura, usuario: widget._usuario),
      ),
      body: BlocListener<AsignaturaBloc, EstadoAsignatura>(
        listener: (context, state) {
          if (state is PdfGenerado) {
            Navigator.of(context).pushNamed(visorPDFRuta,
                arguments: ArgumentosPantalla(
                    usuario: widget._usuario,
                    pdf: state.pdf,
                    asignatura: widget._asignatura,
                    listaAsistenciasMes: state.listaAsistenciaMes));
          } else if (state is AsignaturaError) {
            MiSnackBar.informacionSnackBar(context, state.mensaje);
          }
        },
```

ANEXO 2 – EJEMPLO DE NAVEGACIÓN ENTRE PANTALLAS CON PUSHNAMED()

Posteriormente, según la ruta se comprueba que los argumentos que se pasan son los correctos y se muestra la vista correcta, en caso de error se muestra una vista de fallo.

```
case visorPDFRuta:
        if (args is ArgumentosPantalla) {
          Uint8List pdf = args.pdf!;
          UsuarioAutenticado usuario = args.usuario;
          Asignatura asignatura = args.asignatura!;
          ListaAsistenciasMes listaAsistenciaMes = args.listaAsistenciasMes!;
          return MaterialPageRoute(
            builder: (_) => VistaPreviaPdf(
              pdf: pdf,
              asignatura: asignatura,
              usuario: usuario,
              listaAsistencias: listaAsistenciaMes,
```

```
            ),
          );
        }
        return _rutaErronea();
      default:
```

ANEXO 3 – FICHERO DE RUTAS

Anexo III. Bloc

BlocProvider

Para poder utilizar un Bloc que se ha definido en la aplicación, se tiene que proporcionar al árbol de Widgets de la APP (la interfaz gráfica) y para eso se utiliza BlocProvider.
BlocProvider permite el uso de un Bloc a los Widgets que se crean después del Widget donde se instancia, pero no a sí mismo o a Widgets anteriores. [45]

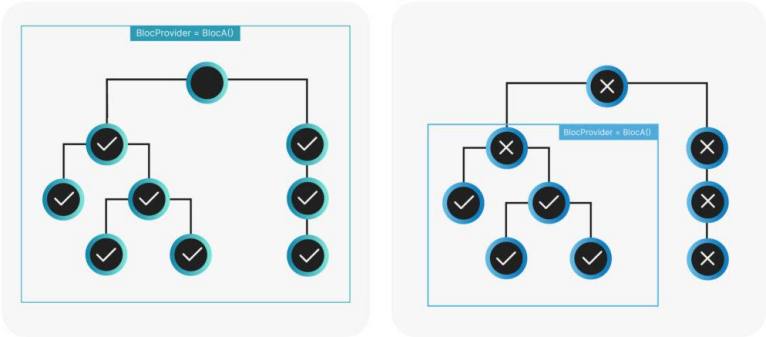

FIGURA 46 - FUNCIONAMIENTO DE BLOCPROVIDER

En el siguiente código se ve un ejemplo del uso de BlocProvider para definir un Bloc, se puede ver como se define AutenticacionBloc y se crea mediante *create: (context) =>,* se usa la inyección de dependencias que se ha visto en el primer anexo.
En el caso de la aplicación desarrollada para el TFG, se han creado los Blocs en el widget principal para que de esta forma esté disponible en toda la aplicación.

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  // This widget is the root of your application.
  @override

Widget build(BuildContext context) {
    final GeneradorDeRutas generadorDeRutas = GeneradorDeRutas();
    return MultiBlocProvider(
      providers: [
        BlocProvider<AutenticacionBloc>(
```

```
          create: (context) => AutenticacionBloc(
              repositorioAutenticacion: sl<RepositorioAutenticacion>()),
        ),
        BlocProvider<CalendarioBloc>(
          create: (context) => CalendarioBloc(
            repositorioClases: sl<RepositorioClases>(),
          ),
        ),
        BlocProvider<AsignaturaBloc>(
          create: (context) => AsignaturaBloc(
            repositorioAlumnos: sl<RepositorioAlumnos>(),
            repositorioAsignaturas: sl<RepositorioAsignaturas>(),
            repositorioClases: sl<RepositorioClases>(),
            repositorioListasAsistencia:
sl<RepositorioListasAsistencia>(),
            generadorClases: sl<GeneradorClases>(),
            generadorPDF: sl<GeneradorPdf>(),
            procesadorExcel: sl<ProcesadorExcel>(),
          ),
        ),
```

ANEXO 4 – CREACIÓN DE BLOC CON BLOCPROVIDER

Estados Bloc

En el siguiente código se muestra un ejemplo de cómo se declaran los estados en Bloc. Primero se declara el estado genérico que es una clase abstracta y luego los distintos estados que se utilizarán.
En este caso, se ven los estados *EstadoCargando* y *SesionIniciada*

```
@immutable
abstract class EstadoAutenticacion extends Equatable {
  const EstadoAutenticacion();
}

//Estado que se produce cuando se está haciendo alguna operación
@immutable
class EstadoCargando extends EstadoAutenticacion {
  final String? texto;
  const EstadoCargando({
```

```
    this.texto = 'Espera un momento',
  });
  @override
  List<Object?> get props => [];
}
//Estado que se produce cuando el usuario está iniciando sesión
@immutable
class SesionIniciada extends EstadoAutenticacion {
  final UsuarioAutenticado usuarioAutenticado;
  const SesionIniciada(this.usuarioAutenticado);
  @override
  List<Object?> get props => [];
}
```

ANEXO 5 – FICHERO DE ESTADOS EN BLOC

Eventos Bloc

En este código, se muestra cómo se declaran los eventos en Bloc, al igual que con los estados, se declara una clase abstracta y luego los distintos estados que extienden de dicha clase, en este ejemplo se muestran dos eventos, *ComprobarEstadoUsuario* e *IniciarSesion* al cual se le pasan parámetros.

```
@immutable
abstract class EventoAutenticacion extends Equatable {
  const EventoAutenticacion();
}
//Evento para comprobar el estado en el que está el usuario (si ha
iniciado sesión
//si no ha iniciado sesión)
@immutable
class ComprobarEstadoUsuario extends EventoAutenticacion {
  const ComprobarEstadoUsuario();
  @override
  List<Object> get props => [];
}
//Evento que se produce cuando el usuario pulsa el botón de iniciar
sesión
@immutable
class IniciarSesion extends EventoAutenticacion {
```

```
  final String correo;
  final String password;
  const IniciarSesion({required this.correo, required this.password});
  @override
  List<Object> get props => [correo, password];
}
```

ANEXO 6 – FICHERO DE EVENTOS EN BLOC

Para enviar un evento desde la vista a Bloc, se utiliza:

```
  context.read<NombreBloc>().add(EventoBloc());
```

ANEXO 7 – CÓDIGO PARA ENVIAR UN EVENTO A BLOC

Ejemplo de código que envía un evento para eliminar una asignatura:

```
void eliminarAsignatura({required String idAsignatura}) {
      context.read<AsignaturaBloc>().add(EliminarAsignatura(
          idAsignatura: idAsignatura, idUsuario: _usuario.id));
```

ANEXO 8 – ENVIAR EVENTO PARA ELIMINAR ASIGNATURA

Gestión de eventos y estados en Bloc

En el código siguiente se puede ver como Bloc gestiona los eventos que recibe, así como los estados y operaciones que va realizando en cada estado.

En este caso, se tiene el evento *IniciarSesion* donde se emite el estado *EstadoCargando* para que la interfaz muestre una pantalla de carga y se comprueba si el usuario tiene una sesión, luego si tiene el correo sin verificar se emite el estado *SinVerificarCorreo*.

En caso de error también se emite un estado para informar al usuario.

```
class AutenticacionBloc extends Bloc<EventoAutenticacion,
EstadoAutenticacion> {
  final RepositorioAutenticacion _repositorioAutenticacion;
  //El usuario al principio no está autenticado
  AutenticacionBloc(
      {required RepositorioAutenticacion repositorioAutenticacion})
      : _repositorioAutenticacion = repositorioAutenticacion,
        super(const NoAutenticado()) {
    //Cuando el usuario pulsa el botón de Iniciar sesión se manda un
evento a Bloc
    on<IniciarSesion>((event, emit) async {
      emit(const EstadoCargando());
```

```
      try {
        await _repositorioAutenticacion.iniciarSesion(
            email: event.correo, password: event.password);
        final usuario = repositorioAutenticacion.usuarioActual;
        emit(SesionIniciada(usuario!));

        final user = _repositorioAutenticacion.usuarioActual;

        if (user!.emailVerificado != true) {
          emit(const SinVerificarCorreo());
        }
      } on Exception catch (e) {
        if (e is UsuarioAutenticadoNoEncontrado) {
          emit(ErrorAutenticacion(
              excepcion: e, mensaje: textoUsuarioNoEncontrado));
        } else if (e is PasswordIncorrecta) {
          emit(ErrorAutenticacion(
              excepcion: e, mensaje: textoPasswordIncorrecta));
        } else {
          emit(ErrorAutenticacion(
              excepcion: e, mensaje: textoExcepcionGenerica));
        }
        emit(const NoAutenticado());
      }
    });
```

ANEXO 9 – FICHERO DE BLOC

Bloc Listener

BlocListener se coloca en la interfaz gráfica para reaccionar a los estados emitidos por Bloc y mostrar distinta información según el estado que se reciba.

En este caso, si se recibe el estado *PdfGenerado*, se va a la pantalla para previsualizar el PDF y si ha habido un error, se recibe el estado *AsignaturaError* para mostrar un *SnackBar* informando al usuario.

```
class _ConsultaListasMesesParaPdfVistaState
    extends State<ConsultaListasMesesParaPdfVista> {
  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: MiAppBar(
        titulo: "PDF - ${widget._asignatura.nombreAsignatura}",
        ruta: opcionesAsignaturasRuta,
        argumentos: ArgumentosPantalla(
            asignatura: widget._asignatura, usuario: widget._usuario),
      ),
      body: BlocListener<AsignaturaBloc, EstadoAsignatura>(
        listener: (context, state) {
          if (state is PdfGenerado) {
            Navigator.of(context).pushNamed(visorPDFRuta,
                arguments: ArgumentosPantalla(
                    usuario: widget._usuario,
                    pdf: state.pdf,
                    asignatura: widget._asignatura,
                    listaAsistenciasMes: state.listaAsistenciaMes));
          } else if (state is AsignaturaError) {
            MiSnackBar.informacionSnackBar(context, state.mensaje);
          }
        },
        child: Center(
          child: Padding(
            padding: const EdgeInsets.all(8.0),
            child: Column(
              children: [
                Expanded(
                    child:
widget._asignatura.listasAsistenciasMes.isEmpty
                        ? const Center(
                            child: Text(textoNoHayPDFs),
                          )
                        : ListView(
                            scrollDirection: Axis.vertical,
                            shrinkWrap: true,
```

```
                            children:
widget._asignatura.listasAsistenciasMes
                                .map((listaAsistenciasMes) {
                              String mes = DateFormat.MMMM().format(
                                  listaAsistenciasMes
                                      .listaAsistenciasDia.first.fecha);
                              return ElementoLista(
                                titulo:
                                    '${mes.toString().toUpperCase()} -
${listaAsistenciasMes.anyo}',
                                asignatura: widget._asignatura,
                                listaAsistenciasMes: listaAsistenciasMes,
                              );
                            }).toList(),
                          )),
              ],
            ),
          ),
        ),
      ),
    );
  }
}
```

ANEXO 10 – FICHERO PARA CONSULTAR LOS MESES EN LOS QUE SE PUEDEN GENERAR PDF

Bloc Consumer

BlocConsumer se coloca en la interfaz gráfica y se utiliza para reconstruir los widgets de la interfaz según el estado que se reciba.

En este código, cuando se recibe el estado *AsignaturasUsuarioObtenidas*, se muestra una lista con las asignaturas obtenidas, para ello, se reconstruye la interfaz gráfica.

```
child: BlocConsumer<AsignaturaBloc, EstadoAsignatura>(
        listener: (context, state) {
          if (state is AsignaturaEliminadaCorrectamente) {
            context.read<ListasAsistenciaBloc>().add(
                EliminarListasAsistenciaDiayMes(
                    idAsignatura: state.idAsignatura));
```

```
          } else if (state is AsignaturaError) {
            MiSnackBar.informacionSnackBar(context, state.mensaje);
          }
        },
        builder: (context, state) {
          if (state is AsignaturasUsuarioObtenidas) {
            asignaturas = state.asignaturas;
            return Scaffold(
                appBar: MiAppBar(
                  titulo: "Asignaturas",
                  ruta: calendarioRuta,
                  argumentos: widget._usuario,
                ),
                body: asignaturas.isEmpty
                    ? const Center(child: Text(noHayAsignaturasCreadas))
```

ANEXO 11 – EJEMPLO DE BLOC CONSUMER

Anexo IV. Interfaz gráfica

En este ejemplo, se muestra cómo se declara una interfaz gráfica con distintos componentes como es el *AppBar*, *ListView* y *FormBuilder*.

```
class InicioSesionVista extends StatelessWidget {
  final _formKey = GlobalKey<FormBuilderState>();
  final String titulo;
  InicioSesionVista({Key? key, required this.titulo}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final altura =
        MediaQuery.of(context).size.height -
MediaQuery.of(context).padding.top;
    final espacio = altura > 650 ? espacioM : espacioS;
    final esTeclado = MediaQuery.of(context).viewInsets.bottom != 0;
    return Scaffold(
      appBar: AppBar(
        centerTitle: true,
```

```
      title: Text(titulo),
      automaticallyImplyLeading: false,
    ),
    body: ListView(
      shrinkWrap: true,
      padding: const EdgeInsets.all(8.0),
      children: <Widget>[
        FormBuilder(
          key: _formKey,
          autovalidateMode: AutovalidateMode.always,
          child: Column(
            children: <Widget>[
              SizedBox(height: espacio),
              if (!esTeclado)
                Image.asset(
                  'assets/images/calendar.png',
                  width: 150,
                  height: 150,
                ),
              SizedBox(height: espacio),
              CampoTextoFormulario(
                pista: introducirCorreo,
                key: const Key('correo'),
                icono: const Icon(Icons.email),
                nombreCampo: "correo",
                tipoTeclado: TextInputType.emailAddress,
                validador: FormBuilderValidators.compose(
                  [
                    FormBuilderValidators.email(),
                    FormBuilderValidators.required(),
                  ],
                ),
              ),
              SizedBox(height: espacio),
              CampoTextoFormulario(
```

```
                  pista: introducirPassword,
                  icono: const Icon(Icons.password),
                  key: const Key('password'),
                  nombreCampo: "password",
                  esPassword: true,
                  validador: FormBuilderValidators.compose(
                    [
                      FormBuilderValidators.required(),
                      FormBuilderValidators.minLength(8),
                    ],
                  ),
                ),
                SizedBox(height: 3 * espacio),
                IniciarSesionBoton(formKey: _formKey),
                SizedBox(height: espacio),
                BotonGenerico(
                  texto: iniciarSesionGoogle,
                  onPressed: () {
                    context.read<AutenticacionBloc>().add(
                          const IniciarSesionConGoogle(),
                        );
                  },
                ),
                SizedBox(height: espacio * 3),
                const BotonIrARuta(ruta: registroRuta, texto:
registrarBoton),
                const BotonIrARuta(
                    ruta: nuevaPasswordRuta, texto: cambiarPassword),
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```

ANEXO 12 – FICHERO PARA LA VISTA DE INICIAR SESIÓN

Anexo V. Pruebas

Prueba unitaria

Ejemplo de código para realizar una prueba unitaria.

Se simula la obtención de las clases del usuario y se comprueba que los datos que se obtienen son correctos y que solo se llama una vez a la función para obtener las clases con *verifyNoMoreInteractions()*.

```
  test('Obtener Clases Dia', () async {
      //arrange
      UsuarioAutenticado usuario = const UsuarioAutenticado(
          email: 'prueba@gmail.com', emailVerificado: false, id:
'43443');
      when(() => mockProveedorClases.obtenerClasesDia(
          diaClase: DateTime.utc(2022),
          idUsuario: usuario.id)).thenAnswer((_) async => clases);
      //act
      final resultado = await repositorioClases.obtenerClasesDia(
          diaClase: DateTime.utc(2022), idUsuario: usuario.id);
      //assert
      expect(resultado, isA<List<ClaseEntidad>>());
      expect(resultado, equals(clases));
      verify(() => mockProveedorClases.obtenerClasesDia(
          diaClase: DateTime.utc(2022), idUsuario: usuario.id));
      verifyNoMoreInteractions(mockProveedorClases);
    });
```

ANEXO 13 – PRUEBA PARA OBTENER CLASES DÍA

Prueba unitaria a Firebase

Ejemplo de Prueba unitaria a Firebase. Se usa FakeFirebaseFirestor para simular la base de datos en Firebase.

En este código se simula que se añade una asignatura a un alumno y se comprueba que se ha añadido de forma correcta mediante *expect()* indicando el resultado esperado.

```
void main() {
  final FakeFirebaseFirestore fakeFirebaseFirestore =
FakeFirebaseFirestore();
```

```
  final ProveedorAlumnosFirebase proveedorAlumnosFirebase =
      ProveedorAlumnosFirebase(firestore: fakeFirebaseFirestore);


  group('Proveedor alumnos firebase test', () {
    test('Anyadir Asignatura A Alumno', () async {
      final CollectionReference mockCollectionReference =
fakeFirebaseFirestore
          .collection(proveedorAlumnosFirebase.collecion.path);
      await mockCollectionReference
          .doc(alumnoDTO.nipAlumno)
          .set(alumnoDTO.toJson());


      await proveedorAlumnosFirebase.anyadirAsignaturaAAlumno(
          idAlumno: alumnoDTO.nipAlumno, idAsignatura: '76776');

      final alumno =
    await proveedorAlumnosFirebase.obtenerAlumno(alumnoDTO.nipAlumno);

      expect(alumno.asignaturas, ['2342342', 'Matematicas', '76776']);
    });
```

ANEXO 14 – PRUEBA PARA AÑADIR ASIGNATURA A ALUMNO

Prueba de Bloc

Ejemplo de Código para testear un bloc:

```
group('Registrar Usuario', () {
      blocTest<AutenticacionBloc, EstadoAutenticacion>(
        '''Emite [EstadoUsuarioSinVerificarCorreo] cuando se manda el
        evento RegistrarUsuario''',
        setUp: () {
          when(() => mockRepositorioAutenticacion.registrarUsuario(
              email: usuario.email,
              password: '1234')).thenAnswer((_) async =>
Future<void>.value);

          when(() =>
mockRepositorioAutenticacion.enviarEmailVerificacion())
              .thenAnswer((_) async => Future<void>.value);
        },
```

```
        build: () => autenticacionBloc,
        act: (bloc) =>
            bloc.add(RegistrarUsuario(password: '1234', correo:
usuario.email)),
        expect: () => <EstadoAutenticacion>[
          const EstadoCargando(),
          const SinVerificarCorreo()
        ],
      );
```

ANEXO 15 – PRUEBA DE BLOC PARA REGISTRAR USUARIO

Prueba de Widgets

Ejemplo de un test para widgets, en los 2 *when()* se indica el estado inicial de Bloc que tendrá ese widget.

Con await tester.pumpWidget() se simula la pantalla con los widgets que se va a testear y se pueden simular acciones sobre el widget como presionar con *tap()*.

En todo momento, hay que indicar el widget sobre el que se va a actuar y para ello se usa *find*, pudiéndolo encontrar por tipo de widget, por un cierto texto, clave…

Con *expect()* se indica la condición de éxito que se quiere comprobar para que el test haya sido exitoso ya que se encuentra el widget que se quería.

```
testWidgets(
      "Despliega listas de asistencias al pulsar en un mes",
      (WidgetTester tester) async {
        when(() => listasAsistenciaBloc.state)
            .thenReturn(ListasAsistenciaInitial());
        when(() => asignaturaBloc.state).thenReturn(AsignaturaInitial());

        await tester.pumpWidget(testWidget);

        expect(find.byType(MiAppBar), findsNWidgets(1));
        expect(find.byType(ListadoListasAsistencia), findsWidgets);
        //El Slidable de cada lista de asistencia no es visible por
defecto
        expect(find.byType(Slidable), findsNothing);
        await tester.tap(find
            .byKey(Key(asignaturaConLista
                .listasAsistenciasMes.first.idListaAsistenciasMes))
```

```
            .first);

        await tester.pump(const Duration(milliseconds: 100));
        //Despúes de pulsar el widget ListadoListasAsistencia,
        //los Slidable son visibles
        expect(
            find.byKey(Key(asignaturaConLista.listasAsistenciasMes.first
                .listaAsistenciasDia.first.idListaAsistenciasDia)),
            findsWidgets);
      },
    );
```

ANEXO 16 – PRUEBA PARA DESPLEGAR LAS LISTAS DE ASISTENCIAS EN UN MES SE VISUALIZA DE FORMA CORRECTA

Pruebas de Rutas

En este código, con el *when()* se simula el comportamiento que tendrá la aplicación, en el primer *when()* se indica el estado inicial de Bloc que tendrá ese widget y en el segundo *when()* se simula la navegación entre pantallas.

Se comprueba que se hace la navegación a la ruta correcta con el verify() comprobando que se ha llamado al método popAndPushNamed() que es uno de los métodos de navegación entre pantallas.

```
testWidgets(
      "Volver atrás",
      (tester) async {
        when(() => asignaturaBloc.state).thenReturn(AsignaturaInitial());
        when(
          () => navigator.popAndPushNamed(calendarioRuta, arguments:
usuario),
        ).thenAnswer((_) async {
          return CalendarioyClasesVista;
        });
        await tester.pumpWidget(testWidget);

        await tester.pump(const Duration(milliseconds: 200));
        await tester.tap(find.byIcon(Icons.arrow_back));

        await tester.pumpAndSettle();
```

```
        verify(() =>
                navigator.popAndPushNnamed(calendarioRuta, arguments:
usuario))
            .called(1);
      },
    );
```

ANEXO 17 – TEST DE WIDGET PARA COMPROBAR LA NAVEGACIÓN AL VOLVER ATRÁS EN UNA VISTA

Anexo VI. Problemas encontrados

Archivo Excel Cifrado

El principal problema que se ha encontrado a la hora de realizar la aplicación es que el fichero Excel que se proporciona a los profesores con el listado de alumnos, tiene el NIP del alumno cifrado por lo que, al principio al intentar obtener los datos del Excel en la app, en el caso del NIP salían valores extraños con un formato de fecha lo cual no tenían sentido.

```
I/flutter ( 6603): Los valores del excel son: 4027-11-07T00:00:00.000
I/flutter ( 6603): Los valores del excel son: 73439634J
I/flutter ( 6603): Los valores del excel son: Santomé Rocafort, Diego
I/flutter ( 6603): La fila del excel es esto: [Data(4027-11-05T00:00:00.000, 0, 9, null, listado_30253_40_
326_1), Data(25361541Q, 1, 9, null, listado_30253_40_326_1), Data(Serna Pérez, Alberto, 2, 9, null, listad
o_30253_40_326_1)]
I/flutter ( 6603): Los valores del excel son: 4027-11-05T00:00:00.000
I/flutter ( 6603): Los valores del excel son: 25361541Q
I/flutter ( 6603): Los valores del excel son: Serna Pérez, Alberto
```

ANEXO 18 – ALUMNOS CON NIPS CIFRADOS

Tras diversos intentos, se vio que editando los valores de los Nips en el Excel a mano sí que se obtenían de forma correcta y al preguntar a mi tutora Piedad Garrido, comentó lo expuesto anteriormente.

```
I/flutter ( 6603): Los valores del excel son: 73439634J
I/flutter ( 6603): Los valores del excel son: 777183
I/flutter ( 6603): Los valores del excel son: Santomé Rocafort, Diego
I/flutter ( 6603): La fila del excel es esto: [Data(25361541Q, 0, 9, null, listado_30253_40_326_1), Data(7
77181, 1, 9, null, listado_30253_40_326_1), Data(Serna Pérez, Alberto, 2, 9, null, listado_30253_40_326_
1)]
I/flutter ( 6603): Los valores del excel son: 25361541Q
I/flutter ( 6603): Los valores del excel son: 777181
I/flutter ( 6603): Los valores del excel son: Serna Pérez, Alberto
I/flutter ( 6603): El json del excel es excel
D/EGL_emulation( 6603): eglMakeCurrent: 0xe3133240: ver 3 0 (tinfo 0xdb2510e0)
```

ANEXO 19 – ALUMNOS CON NIPS SIN CIFRAR

Este problema no tiene una solución óptima ya que obliga al usuario a editar el Excel a mano.