Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

PROTOTIPO DE UNA HERRAMIENTA DE APOYO PARA LOS INFORMES CLÍNICOS DE UN HOSPITAL EN DISPOSITIVOS MÓVILES

Alumno: Adrián Pérez Ortega

Tutor: Prof. D. Francisco Daniel Pérez Cano

Cotutor: Prof. D. Juan José Jiménez Delgado

Dpto: Departamento de informática

Julio, 2022

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don FRANCISCO DANIEL PÉREZ CANO y Don JUAN JOSÉ JIMÉNEZ DELGADO, tutores del Trabajo Fin de Grado titulado: PROTOTIPO DE UNA HERRAMIENTA DE APOYO PARA LOS INFORMES CLÍNICOS DE UN HOSPITAL EN DISPOSITIVOS MÓVILES, que presenta ADRIÁN PÉREZ ORTEGA, autorizan su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Julio de 2022

El alumno:

Firmado por PEREZ ORTEGA ADRIAN - ***5350** el día 01/07/2022 con un certificado emitido por

ADRIÁN PÉREZ ORTEGA

Los tutores:

Firmado digitalmente por PEREZ CANO FRANCISCO DANIEL - 26051826W
Fecha: 2022.07.01 19:54:26 +02'00'

JIMENEZ DELGADO JUAN JOSE - 26216401N

Firmado digitalmente por JIMENEZ DELGADO JUAN JOSE - 26216401N
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26216401N, givenName=JUAN JOSE, sn=JIMENEZ DELGADO, cn=JIMENEZ DELGADO JUAN JOSE - 26216401N
Fecha: 2022.07.03 20:31:05 +02'00'

FRANCISCO DANIEL PÉREZ CANO

Y

JUAN JOSÉ JIMÉNEZ DELGADO

Índice

Índice Ilustraciones

Índice Tablas

1. Introducción

1.1 Motivación

El proyecto surge debido a los problemas que puede tener un médico al rellenar los informes clínicos de los pacientes ingresados, ya que en ciertas ocasiones el especialista, o alguno de sus ayudantes, se encarga de tomar notas sobre la visita a un paciente. Pero en la mayoría de las ocasiones, no se toma ninguna nota y los informes se hacen tras la visita a varias habitaciones o plantas por lo que se puede perder omitir algunos datos a la hora de generar el informe. Por ello, se ha decidido crear un prototipo de una aplicación móvil con la que el personal de cualquier centro hospitalario o clínica pueda tomar notas sobre los pacientes ingresados de la manera más eficiente y efectiva posible.

Además, a día de hoy hay pocas aplicaciones móviles especializadas para la toma de notas sobre pacientes ingresados en hospitales o clínicas privadas con las que el personal pueda trabajar cómodamente.

Por otro lado, la elección de este trabajo fin de grado se debe a mi interés personal de profundizar el conocimiento de varias tecnologías con las que no he trabajado a lo largo del grado.

1.2 Objetivo

El principal objetivo de este trabajo fin de grado es el de crear un prototipo de una aplicación móvil para el sistema operativo Android [1], ya que es el sistema operativo de dispositivos móviles más utilizado y demandado en la actualidad, que permita al personal de cualquier hospital o clínica poder tomar diversas notas de los pacientes que estén ingresados con el objetivo de facilitar la elaboración del informe clínico. Además, otro de los principales objetivos es el de proporcionar un prototipo que sea fácil de usar, intuitivo y que no requiera de mucho esfuerzo de aprendizaje.

Por otro lado, se deberán de obtener y definir los requisitos funcionales y de calidad que deberá de cumplir el prototipo, para ello se deberá de investigar qué aplicaciones existen a día de hoy y cuales son las necesidades específicas del cliente. También, se deberá de diseñar y desarrollar el prototipo de aplicación móvil en un lenguaje de programación, el cual nos permite satisfacer todas las necesidades y requisitos.

Por último, si el proyecto se desarrolla conforme lo planificado y sin retrasos se desarrollará una aplicación web que pueda dar soporte a la aplicación móvil con la que el personal podrá acceder desde su ordenador para gestionar las notas.

2. Análisis

2.1 Análisis preliminar

En el apartado de análisis preliminar mostraremos primero algunas aplicaciones móviles con las que se pueden tomar notas de los pacientes ingresados y posteriormente indicaremos cuales son las tecnologías elegidas para el desarrollo de nuestro prototipo de aplicación móvil.

2.1.1 Estado del arte

En este apartado analizaremos las aplicaciones móviles existentes que podrían ser utilizadas por los usuarios para tomar las notas de los pacientes ingresados.

2.1.1.1 Mis pacientes

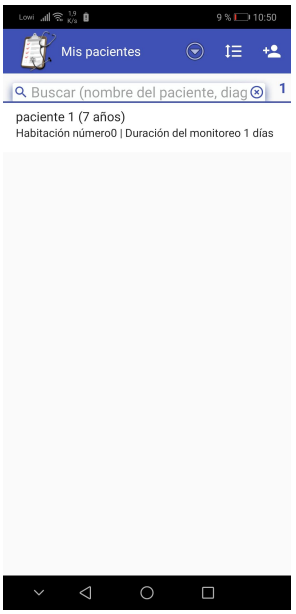


Ilustración 1: interfaz de Mis pacientes.

Mis pacientes es una aplicación móvil que está diseñada para poder las gestionar notas de un paciente. Esta aplicación permite almacenar los datos personales del paciente, observaciones sobre las pruebas de laboratorio y las fechas de las diferentes pruebas que se le han realizado.

Además nos permite crear recordatorios y almacenar los datos de la aplicación tanto en un fichero .zip en local o en Google Drive. No obstante la aplicación presenta varios inconvenientes, como pueden ser:

- Los campos de las notas a rellenar son muy ambiguos o innecesarios.
- Una navegación e interacción con la aplicación es poco intuitiva.
- Los datos que se introducen y almacenan en la aplicación son datos clínicos, por lo que una filtración de estos datos es bastante peligrosa.
- No se distingue a quién se le ha tomado la nota y en qué momento.

2.1.1.2 Mis notas

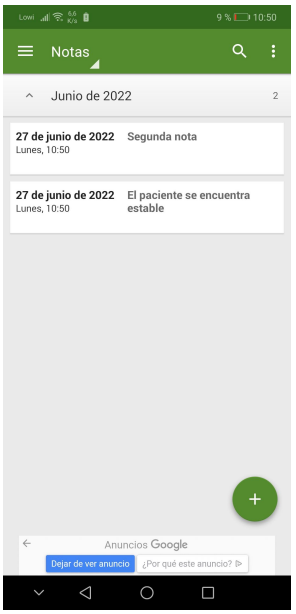


Ilustración 2: interfaz de Mis notas.

Mis notas es una aplicación que permite añadir, eliminar, editar o compartir notas simples, también podemos buscar texto escrito en las notas mediante una barra de búsqueda y almacenar las notas en un servidor si nos registramos en la aplicación, en caso de que no te registres solamente podrás guardar las notas en local.

La mayoría de aplicaciones para tomar notas son semejantes a ésta ya que suelen tener las mismas características y funciones. Estas aplicaciones se caracterizan por su simplicidad y su interfaz intuitiva para realizar las acciones. No obstante, el principal problema de este tipo de aplicaciones es que no están personalizadas para el uso del personal del hospital, debido a que tomar notas únicas sobre un paciente ingresado sería muy difícil, por el hecho de que no hay ningún campo para identificar la nota.

2.1.1.3 Evernote

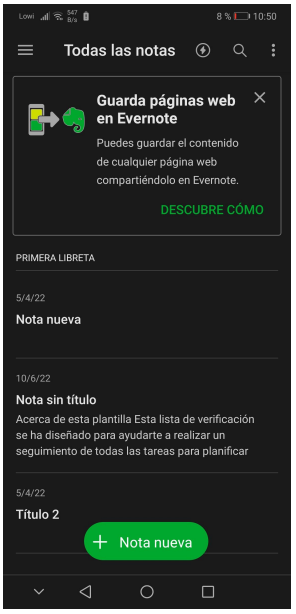


Ilustración 3: interfaz de Evernote.

Evernote es una aplicación móvil disponible en la Play Store y en App store, la cual es una aplicación bastante conocida y usada por parte de los usuarios, ya que tiene una gran valoración.

Evernote permite realizar las acciones habituales de un bloc de notas como son la de añadir, editar, eliminar, compartir notas o buscar texto en una barra de búsqueda, pero además añade otras funcionalidades extras como la de organizar las notas en libretas, adjuntar ficheros, audios o fotos a las notas. Aunque la funcionalidad más útil es la de crear una nota mediante plantillas, las cuales tienen grandes funcionalidades y campos en función del usuario.

No obstante, esta aplicación presenta ciertos inconvenientes para el uso en el ámbito médico. A pesar de que el usuario pueda personalizar la nota y los campos a añadir y mostrar, esto solo es válido para un único usuario.

2.1.2 Selección de tecnologías

En este apartado se describen todas las tecnologías que se han utilizado para el desarrollo del prototipo de la aplicación móvil.

2.1.2.1 MySQL

Ilustración 4: logo de MySQL.

MySQL [4] es un sistema gestor de bases de datos relacional, MySQL es uno de los sistemas más conocidos y usados gracias a su facilidad de uso y su simplicidad, debido a que pueden interactuar con el sistema a través de los comandos habituales utilizados en SQL. Respecto a su utilización en el proyecto, MySQL se utiliza para la persistencia de los datos relevantes de la aplicación. Además MySQL es de código abierto por lo que es una gran opción para ser utilizada en el ámbito académico y laboral sin tener que invertir dinero y obteniendo un sistema gestor de base de datos seguro y confiable.

No obstante, también existen otras alternativas a MySQL a utilizar en el prototipo. La mejor alternativa a MySQL sería MariaDB, el cual es también un sistema de gestión de bases de datos relacional, MariaDB tiene como base MySQL, debido a esto, MariaDB ofrece una gran capacidad de reemplazo respecto a MySQL, es decir se podría cambiar el gestor de la bases de datos del prototipo sin demasiada dificultad.

Otro de los factores que me llevaron a la elección de MySQL es que la mayoría de plataformas de servicio en la nube(PaaS) como son por ejemplo Heroku, Clever-Clouds o AWS de Amazon, ofrecen de forma gratuita el complemento de MySQL y en cambio otros gestores de bases de datos como PostgreSQL, MongoDB o MariaDB no tienen licencia gratuita.

2.1.2.2 Kotlin

Ilustración 5: logo de Kotlin.

Kotlin [5] es un lenguaje de programación de código abierto creado por la empresa JetBrains. Es un lenguaje de tipado estático ya que se desarrolla sobre la máquina virtual de Java(JVM) o sobre JavaScript.

Kotlin [6] proporciona soluciones fáciles a los problemas comunes que suelen aparecer al desarrollar una aplicación móvil con Java. Por ejemplo, en Kotlin no existen las excepciones de tipo NullPointerException, la cual es bastante común en Java, y nos obliga a tener controlados todos los errores que puedan ocurrir en el programa. Además, Kotlin tiene una curva de aprendizaje más baja que otros lenguajes, como C++, y ha pasado a ser el lenguaje oficial de programación en Android, sustituyendo a Java.

Por último, cabe destacar que a pesar de que el prototipo va a ser realizado únicamente para dispositivos Android, Kotlin nos permite crear una aplicación multiplataforma, similar al lenguaje de programación Dart, mediante la librería de Kotlin Multiplatform [7], no obstante el prototipo de aplicación que se ha desarrollado no incluye estas características debido a que el ámbito del proyecto ya era lo suficientemente grande.

2.1.2.3 Node.js

Ilustración 6: logo de node.js.

Node.js [8] en un entorno en tiempo de ejecución multiplataforma para la capa de servidor basado en JavaScript. Node.js está orientado a eventos asíncronos, por tanto este entorno se suele utilizar para crear aplicaciones escalables y que acepten solicitudes de cualquier tipo de dispositivo.

La elección de este entorno de ejecución se debe a las muchas ventajas que posee:

- La curva de aprendizaje de este torno es relativamente baja ya que el lenguaje que utiliza es JavaScript.

- Utilizar este entorno permite gran escalabilidad en la aplicación, ya que node.js nos permite crear con gran facilidad nuevos microservicios.
- Node.js proporciona el Node Package Manager (npm) [9], el cual es un gestor de módulos que podremos utilizar en cualquier proyecto realizado en este entorno.
- Node.js es más ligero y rápido a la hora de compilar el programa y desplegarlo que otras tecnologías, como es el caso del spring boot.
- Node.js utiliza un modelo de entrada/salida sin bloqueo, es decir se pueden realizar diferentes peticiones al backend y se crearán diferentes hilos para que ninguna tarea tenga que esperar a otra.

El entorno node.js se ha utilizado para el desarrollo del backend del prototipo de la aplicación. Dentro del entorno hemos utilizado el módulo express el cual es un framework que nos permite crear una API-REST de una manera sencilla y rápida.

Por otro lado, también existen varias alternativas para desarrollar el backend, como pueden ser Spring Boot, Ruby on Rails o Django. No obstante, se decidió utilizar node.js ya que es la mejor alternativa disponible debido a su modelo de entrada y salida de peticiones sin bloque, ya que nuestra aplicación móvil realizará llamadas al backend para acceder a la base de datos, por lo que no se necesita una gran capacidad de computación del backend.

2.1.2.4 Handlebars

Ilustración 7: logo de handlebars.

Handlebars [10] es un motor de plantillas de javascript que nos permite crear y formatear de manera rápida y efectiva código de HTML. Además, las plantillas que se crean son plantillas web responsive con lo que se podrán adaptar las vistas a diferentes dispositivos. Al utilizar handlebars podremos reutilizar código que se repita en diferentes vistas o secciones. Además al crear las vistas con handlebars

obtendremos un código muy organizado ya que podremos separar totalmente las interfaces del usuario en la página web del backend.

Alternativas a handlebars hay bastantes, como por ejemplo son JSP, JSF, o Angular. Sin embargo, se ha decidido utilizar handlebars ya que node.js permite integrarlo en el proyecto de forma muy sencilla mediante el framework express.

Se utilizará handlebars para la implementación de la página web de apoyo y soporte del prototipo de aplicación móvil.

2.1.2.5 Android Studio

Ilustración 8: logo de android studio.

Android Studio es un entorno de desarrollo integrado(IDE) oficial y más utilizado para el desarrollo e implementación de aplicaciones nativas en Android. Algunas de sus principales características son su sistema de compilación flexible basado en Gradle para automatizar la compilación del código, además Android Studio nos ofrece la posibilidad de utilizar y descargar varios emuladores de móvil, tablet, dispositivos wear e incluso de televisiones. También nos ofrece integración con GitHub así como de aplicaciones y servicios de terceros como son Firebase.

2.1.2.6 Firebase

Ilustración 9: logo de firebase.

Firebase [11] es una plataforma adquirida por Google en el año 2014, dedicada al desarrollo de aplicaciones web y móviles.

Con la plataforma Firebase se pueden realizar varias acciones, algunas de ellas son:

- Desarrollo e implementación de aplicaciones web o móviles.
- Permitir crear bases de datos no relacionales y alojarlas en la nube, almacenando los datos en formato JSON.
- Autentificación de usuarios mediante métodos personalizados o proveedores externos como son Google, Facebook o GitHub.
- Firebase ofrece un servicio llamado app check el cual consiste en ofrecer al usuario métodos para proteger su backend de ataques informáticos como son el phishing y mantener los recursos de la aplicación seguros.
- Firebase permite crear aplicaciones dedicadas exclusivamente al aprendizaje automático(machine learning).

En el caso del prototipo de aplicación móvil a desarrollar, se utilizará la plataforma de Firebase para la autentificación de usuarios mediante el proveedor de Google. Gracias a esto obtenemos un aumento en la seguridad de nuestro prototipo de aplicación, ya que evitamos el envío continuo de credenciales por parte del cliente y además no tenemos que almacenar el nombre de usuario ni la contraseña del usuario en la base de datos del servidor.

2.1.2.7 Visual Studio Code

Ilustración 10: logo de visual studio code.

Visual Studio Code es un editor de código desarrollado por Microsoft el cual se puede utilizar en prácticamente cualquier sistema operativo, se utilizará para el desarrollo e implementación del backend.

En Visual Studio Code se puede instalar el entorno de ejecución node.js y el npm (Node Package Manager) de forma muy sencilla y rápida.

2.1.2.8 Visual Paradigm

Ilustración 11: logo de visual paradigm.

Visual Paradigm es una herramienta de ingeniería del software utilizada para ayudar a los desarrolladores de software en las etapas del ciclo de vida de un proyecto de software. Con la herramienta se pueden crear diagramas para el diseño del sistema, como pueden ser: diagramas de casos de uso, de clase, de secuencia, de comunicación, etc. También permite diseñar interfaces de usuario para cualquier tipo de dispositivo, móvil, tablet, wear o televisión.

Se utilizará esta herramienta durante la etapa de diseño para crear los diagramas de clase, diagrama de casos de uso, diagrama de secuencia y una maqueta de la aplicación.

2.1.2.9 Docker Desktop

Ilustración 12: logo de docker.

Docker es una plataforma de código abierto diseñada para el desarrollo, implementación y despliegue de aplicaciones que se ejecutan en un contenedor de software.

El objetivo de docker es permitir ejecutar cualquier aplicación que está en un contenedor software, independientemente del sistema operativo que la máquina utilice, únicamente es necesario tener instalado Docker, por lo que el desarrollador sólo tiene que preocuparse en escribir el código sin pensar en qué tipo de sistema operativo puede ejecutarse.

En el proyecto se utiliza docker para tener instalado MySQL en local y poder probar durante las primeras fases del desarrollo el backend, para ello se utiliza un contenedor que contiene MySQL y se configura en función de los parámetros que se

necesitan, además docker nos permite configurar una aplicación creada con el contenedor para que se desinstale una vez que se ha apagado el ordenador.

La principal ventaja de utilizar la imagen de MySQL en docker es que tiene un peso mucho menor a MySQL y MySQL workbench por lo que lo hace más eficiente y cómodo. Además, con la imágen docker podremos instalar y desinstalar con gran facilidad el gestor de bases de datos.

2.1.2.10 DBeaver

Ilustración 13: logo de DBeaver.

DBeaver es un programa software utilizado para la administración de las bases de datos, relacionales y no relaciones.

Con DBeaver crearemos la base de datos del backend que utilizaremos durante el desarrollo del prototipo, en las primeras etapas del desarrollo crearemos la base de datos del servidor en la imagen del docker para utilizarla en local y posteriormente, cuando se vaya a desplegar crearemos una base de datos que estará alojada en un servidor. Se utilizará esta herramienta para crear una copia de la estructura de la base de datos.

2.1.2.11 GitHub

Ilustración 14: logo de GitHub.

GitHub es una plataforma para almacenar proyectos, normalmente proyectos de software, utilizando el sistema de control de Git. Gracias a Git podremos trabajar en el proyecto con o sin conexión, además nos permite crear varias ramificaciones sobre el proyecto en la que podremos crear y unir diferentes ramas, lo que resulta

bastante útil para trabajos en equipo, así como también podemos volver a commits anteriores del proyecto.

Para el desarrollo del proyecto se han creado 2 repositorios en GitHub, uno para la parte del frontend, el cual está escrito en el lenguaje de Kotlin y otro para la parte del backend, el cual está desarrollado con el entorno node.js.

Por otro lado, la elección de utilizar GitHub en vez de cualquier otra plataforma de control de versiones es debido a que las páginas de hosting gratuitas que se analizaron requerían que los proyectos a subir debían de estar en un repositorio en GitHub.

2.1.2.12 Heroku

Ilustración 15: logo de Heroku.

Heroku es una plataforma de servicios en la nube(PaaS) que permite desplegar aplicaciones en servidores y poder configurarlos.

La principal ventaja de Heroku es la capacidad de poder desplegar aplicaciones realizadas en los principales lenguajes de programación. Por otro lado, las principales características de Heroku son:

- **Heroku-CLI**: terminal de Heroku similar a la terminal de Git Bash que se utiliza para poder inicializar repositorios de Heroku y poder subirlos a la nube.
- **Dynos**: los dynos son contenedores virtuales de Linux que se encargan de la capacidad de procesamiento de la aplicación. Estos elementos son de los más importantes de la aplicación, cuando una aplicación falla y entra en un estado inconsistente los dynos dejan de funcionar y es por eso que una vez al día, Heroku comprueba si los dynos de la aplicación siguen funcionando, también puede el propio propietario de la aplicación reiniciar los dynos.
- **Integración con GitHub**: Heroku permite utilizar los repositorios que tienes en GitHub y subirlos al servidor.
- **Addons**: son complementos a la aplicación, no obstante, la mayoría de estos complementos son de pago, los addons más comunes son los gestores de las

bases de datos. No obstante, al interactuar con la herramienta hemos encontrado varios problemas a la hora de crear un addon.

Respecto a la utilización de Heroku en el proyecto desarrollado, se utiliza heroku, para alojar la parte del API de REST del backend.

2.1.2.13 Clever Cloud

Ilustración 16: logo de clever cloud.

Clever cloud es una plataforma de servicios en la nube(PaaS) igual que Heroku, tiene las mismas características que Heroku, exceptuando los dynos y el Heroku Cli.

Clever Cloud se utiliza en el proyecto para alojar la base de datos del servidor del prototipo de la aplicación móvil. La principal razón de utilizar clever cloud en vez de heroku es la facilidad con la que se pueden crear los complementos en clever cloud y gestionarlos a través de cualquier herramienta de administración de base de datos.

2.1.2.14 Gradle

Ilustración 17: logo de gradle.

Gradle es una herramienta que se utiliza para la automatización de la compilación, es el compilador oficial para Android tanto en Java como en Kotlin. Con esta herramienta compilamos el prototipo de aplicación móvil a desarrollar en el lenguaje Kotlin.

2.1.2.15 PostMan

Ilustración 18: logo de postman.

Postman es una herramienta que permite realizar peticiones a APIs Rest de una forma bastante sencilla y rápida. Postman se puede instalar en el ordenador como un programa software normal o como una extensión del navegador.

Esta herramienta será utilizada en la fase de pruebas, con el objetivo de comprobar el funcionamiento de la API-REST implementada.

2.2 Solución propuesta

Una vez realizado el análisis preliminar y elegido qué tecnologías vamos a utilizar, hemos decidido realizar un prototipo de aplicación móvil en la que solo exista un tipo de usuario y se pueda autenticar y registrar mediante una cuenta de Google. Los usuarios registrados podrán realizar las funcionalidades básicas de gestionar una nota de un paciente, como son añadir, editar, eliminar y compartir.

En el prototipo de aplicación a desarrollar se podrá añadir los datos básicos de un paciente que está ingresado, como son:

- **Planta**: planta en la que se encuentra el paciente.
- **Habitación**: habitación en la que se encuentra el paciente.
- **Cama**: cama en la que se encuentra el paciente.
- **Constantes vitales**: las constantes vitales del paciente.
- **Evolución**: la evolución del paciente.
- **Plan de actuación**: plan de actuación a seguir.

Además, se desarrollará un prototipo de aplicación móvil que permita al usuario trabajar sin conexión, ya que es posible que el usuario no siempre tenga la posibilidad de tener una conexión a internet disponible. Una vez que el usuario vuelve a tener una conexión disponible, podrá sincronizarse con el usuario de forma manual cuando lo desee. Por otro lado, el prototipo de aplicación a desarrollar

deberá ser compatible tanto para móviles como para tablets Android, con el objetivo de permitir al usuario decidir con qué dispositivo trabajar.

También se desarrollará un prototipo para una aplicación web sencilla, en la que el usuario pueda visualizar las notas que ha creado y de las que tiene acceso. Esta herramienta era considerada bastante útil por el cliente debido a que los informes se realizan desde el ordenador de un despacho y el hecho de poder ver la información, en el mismo dispositivo, en el que se está realizando el informe y poder copiar las anotaciones realizadas desde el teléfono móvil facilitaba la elaboración del informe.

Por último, la aplicación se llamará AnyNote y tendrá un logo asociado con el objetivo de hacerlo más reconocible y amigable a los usuarios.

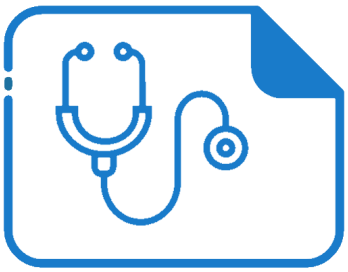


Ilustración 19: logo de la aplicación AnyNote.

2.3 Metodología

La metodología de desarrollo del software utilizada para desarrollar el prototipo de la aplicación móvil ha sido la teoría y filosofía de SCRUM [2] y el método Kanban [3].

2.3.1 SCRUM

SCRUM es un marco ligero de trabajo para el diseño, desarrollo y mantenimiento de proyectos software con cierta complejidad. Además, SCRUM utiliza un enfoque incremental e iterativo lo que resulta ideal para aquellos proyectos adaptativos en los que se podrán modificar requisitos u objetivos conforme se avanza en el proyecto. Con SCRUM se realizan entregas y revisiones periódicas del producto con el cliente para aumentar la eficiencia del equipo de desarrollo y obtener retroalimentación del cliente, ya que cada cierto tiempo se comprueba cómo va el desarrollo del producto. Al utilizar SCRUM obtenemos:

- **Transparencia**: Ya que todos los cambios o avances realizados en el proyecto serán visibles para todo el equipo de desarrollo.
- **Inspección**: En cada iteración se inspecciona el trabajo desarrollado anteriormente por lo que se pueden detectar en cada iteración fallos o problemas que se habían pasado por alto en la iteración pasada.
- **Adaptación**: Se podrá realizar cambios en la planificación del equipo de desarrollo de trabajo en función del proceso de desarrollo del producto, además se podrá modificar los requisitos satisfacer del proyecto en caso de que se quieran añadir nuevos requisitos o suprimir requisitos existentes, teniendo en cuenta que siempre se acordará y negociará con el cliente.

Dicho todo esto, para realizar el prototipo de la aplicación móvil se han realizado algunas variantes sobre la metodología SCRUM para poder adaptarla a un proyecto individual como este.

Lo primero de todo, el cliente con el que se interactúa durante el tiempo de desarrollo del proyecto será la Dra. Sara Ruíz Magaña, médica del parque tecnológico de ciencias de la salud de Granada, que nos ha proporcionado retroalimentación sobre aspectos de diseño de la aplicación así como de diferentes necesidades de la misma.

Respecto al equipo SCRUM, cambia respecto a la filosofía original ya que las personas con el rol de Product Owner y SCRUM Master pasarán a ser la misma persona. El SCRUM Master será el encargado de facilitar la interacción y colaboración con el cliente y el Product Owner será el responsable de la Product BackLog para maximizar el valor final del producto.

Por otro lado, los eventos de SCRUM también han sido adaptados para este proyecto individual. Los cambios son:

- La duración del sprint se ha establecido en 2 semanas, con una entrega donde se incluya una versión de la aplicación con un incremento en la funcionalidad.
- El sprint planning sigue la filosofía de SCRUM para decidir qué se puede hacer durante el Sprint y la forma de conseguirlo. No obstante, solo participa una persona al tratarse de un proyecto individual.
- El daily SCRUM se realiza cada día para establecer las tareas que se van a realizar en ese día y revisar lo que se hizo ayer.

- En el sprint review sigue igual que en la filosofía de SCRUM donde inspeccionamos el incremento obtenido del sprint y realizamos una retroalimentación con el cliente para mostrarle el producto. Además se obtienen y negocian nuevas adaptaciones a realizar sobre el prototipo
- El evento de sprint retrospective desaparece ya que al tratarse de un proyecto individual no se valora cómo se ha trabajado y comunicado el equipo con el objetivo de aplicar cambios para mejorar la eficiencia del mismo.

Por último, los artefactos de la metodología, son los elementos que representan el valor del producto y están diseñados para tener una gran transparencia sobre la información que se quiere transmitir, siguen manteniendo la esencia principal de la metodología, no obstante se han realizado pequeños cambios para adaptarlos al proyecto, estos artefactos son:

- **Product Backlog**: es el trabajo o tareas que quedan por hacer sobre el proyecto,además se irá modificando añadiendo o eliminando tareas. Además en la pila de producto está definido el Product Goal, el cual describe y explica cómo debe ser el estado final en el que se debe de encontrar el producto una vez se termine el proyecto. La evolución del Product Backlog pasará a ser el único miembro del proyecto y no del Product Owner.
- **Sprint BackLog**: son las tareas a realizar en un Sprint. El Sprint Backlog incluye la definición de Sprint Goal. Este término hace referencia al objetivo a cumplir y satisfacer durante el Sprint. Al Igual que el Product Backlog, la evolución y control del Sprint BackLog es responsabilidad del único integrante del proyecto y no del equipo de desarrolladores que es lo que está definido en SCRUM.
- **Incremento**: un incremento es una tarea que se ha realizado en el Sprint y aporta valor al producto final del Sprint.

2.3.2 Kanban

Kanban es un método que implementa la mentalidad Lean, la cual nos permite encontrar problemas y conflictos durante el desarrollo de las tareas con el objetivo de entregar el máximo valor posible del producto al cliente. Además, Kanban es un método que nos permite mejorar los procesos de desarrollo.

La principal ventaja de utilizar Kanban es la mejora de la visualización del flujo de trabajo del proyecto, debido a que podemos ver las tareas realizadas y las pendientes por hacer.

Para implementar el método Kanban. utilizaremos un tablero de tareas donde podamos ver en todo momento el estado en el que se encuentra la tarea y la prioridad de la misma. Para ello utilizaremos la aplicación Trello, en el tablero definiremos 4 listas:

- **Product BackLog**: lista que contendrá todas las tareas del producto para cumplir con el Product Goal.
- **Sprint Backlog**: lista que contendrá las tareas a realizar en el Sprint para cumplir con el Sprint Goal.
- **En progreso**: lista de tareas del Sprint que se está realizando y aún no están finalizadas.
- **Terminado**: lista de tareas que se han terminado.

2.4 Requisitos

2.4.1 Requisitos funcionales

Los requisitos funcionales describen los servicios que debe ofrecer el sistema y cómo debe comportarse el sistema en las diferentes situaciones y escenarios que pueden suceder. Los requisitos funcionales que debe cumplir el prototipo de aplicación móvil son:

- El sistema debe permitir iniciar sesión mediante Google.
- El sistema debe permitir al usuario cerrar sesión cuando desee.
- El sistema debe permitir al usuario añadir una nota.
- El sistema debe permitir al usuario editar una nota.
- El sistema debe permitir al usuario eliminar una nota.
- El sistema debe permitir al usuario compartir una nota.
- El sistema debe de mostrar una lista con todas las notas almacenadas en la base de datos local.
- El sistema debe permitir al usuario eliminar varias notas de la lista a la vez.
- El sistema debe permitir al usuario visualizar cualquier nota almacenada en la base de datos local.

- El sistema debe permitir al usuario ordenar la lista de notas por fecha, de forma ascendente y descendente.
- El sistema debe permitir al usuario ordenar la lista de notas por los valores de planta, habitación y cama de forma ascendente y descendente.
- El sistema debe permitir al usuario a buscar en una barra de búsqueda texto que aparezca en cualquier nota almacenada en la base de datos local.
- El sistema debe permitir al usuario buscar notas por los campos de planta, habitación y cama en la base de datos del servidor.
- El sistema debe permitir al usuario buscar notas por los campos de planta. habitación y cama en la base de datos local cuando no hay conexión a internet.
- El sistema debe permitir añadir notas cuando no hay una conexión a internet.
- El sistema debe permitir editar notas cuando no hay una conexión a internet.
- El sistema debe de permitir eliminar notas cuando no hay una conexión a internet.
- El sistema debe permitir al usuario cambiar de idioma.
- El sistema debe permitir al usuario cambiar a modo oscuro la aplicación.
- El sistema debe de sincronizarse solo cuando se inicie la aplicación, siempre y cuando haya conexión.
- El sistema debe permitir al usuario sincronizar el dispositivo móvil con el servidor.
- El sistema debe guardar la sesión del usuario para poder acceder a la aplicación cuando no tiene conexión sin tener que volver a iniciar sesión.

2.4.2 Requisitos de calidad

Los requisitos de calidad son aquellos requisitos que una funcionalidad o servicio que debe ofrecer el sistema, sino que describen la calidad que se espera que tenga los servicios ofrecidos por el sistema. Los requisitos de calidad que debe de cumplir el prototipo de aplicación móvil son:

- **Flexibilidad**: el sistema debe ser flexible para que en un futuro se pueda añadir nuevas funcionalidades o adaptar el comportamiento del sistema.
- **Robustez**: el sistema debe controlar las entradas inválidas al sistema y seguir funcionando con total normalidad. Por otro lado, ante un fallo del servidor, la

aplicación móvil debe funcionar correctamente y permitir al usuario trabajar en el modo sin conexión.

- **Usabilidad**: el sistema debe de ser fácil de usar e intuitivo, el tiempo máximo de aprendizaje hasta que el usuario sea capaz de dominar el sistema no debe superar un día.
- **Integridad**: para acceder a los datos del usuario en la aplicación móvil será necesaria autenticarse mediante una cuenta de Google.
- **Compatibilidad**: el sistema deberá adaptar la interfaz de usuario en función del dispositivo y del tamaño de su pantalla.

2.5 Historias de Usuario

El objetivo principal de definir las historias de usuario es obtener una explicación escrita por parte del cliente de lo que el software debería de hacer. Además, una historia de usuario nos proporciona el valor que aporta esa funcionalidad o mejora al producto.

Las historias de usuario se escriben en lenguaje natural siguiendo una estructura como [perfil], quiero, para:

- **Como [perfil]**: nos indica para quién va dirigida la historia de usuario.
- **Quiero**: se describen las intenciones que tiene el usuario al realizar la acción.
- **Para**: el objetivo que obtiene el usuario al realizar la acción.

Por otro lado, para realizar historias de usuario en un marco de trabajo ágil, se deberá de seguir el modelo *INVEST* [12], el cual es un método para garantizar la calidad de las historias de usuario y que cumplan una serie de características:

- **Independiente**: las historias de usuario deben de ser independientes entre sí, para que se puedan planificar y desarrollar en cualquier momento.
- **Negociable**: una historia de usuario no incluye detalles sobre cómo ha de implementarse la funcionalidad, por lo que las historias de usuario deben ser negociables en un futuro con el cliente.
- **Valiosa**: una historia de usuario debe aportar un valor al producto final. En nuestro proyecto, el rango del valor estará entre 1 y 5.
- **Estimable**: una historia de usuario se debe de estimar con facilidad el tiempo y esfuerzo que se le dedicará para realizarla. La variable que utilizaremos para definir el esfuerzo a realizar serán los puntos de historia, en el cual

supondremos que cada punto de historia corresponde a un día de una jornada laboral.

- **Pequeña**: una historia de usuario no puede tener un tiempo de realización mayor a lo que dura un Sprint.
- **Testeable**: una historia de usuario debe de tener los medios para ser validada y verificada por el cliente, es decir, el cliente debe de estar seguro de que la historia de usuario está finalizada.

Para establecer el valor de la historia de Usuario utilizaremos el método *MoScoW* [29], con el cual ordenaremos las historias de usuario en función del valor que aporta al producto.

Id	Título	Descripción	Estimación	Valor
1	Iniciar Sesión	Como usuario, quiero iniciar sesión, para acceder a la aplicación y a mis datos.	7	5
7	Ver una nota	Como usuario, quiero ver una nota, para visualizar los datos de un paciente.	5	5
3	Añadir una nota	Como usuario, quiero añadir una nota, para gestionar un paciente.	5	5
4	Editar una nota	Como usuario, quiero editar una nota, para modificar los datos de un paciente.	2	5
5	Eliminar una nota	Como usuario, quiero eliminar una nota, para no tener notas que no quiero.	5	5
13	Trabajar sin conexión	Como usuario, quiero trabajar sin conexión, para no depender de la conexión a internet.	6	5
14	Sincronizar con el servidor	Como usuario, quiero sincronizar mis datos con el servidor, para salvaguardar mis datos y acceder en cualquier dispositivo.	2	5
8	Visualizar una lista con las notas	Como usuario, quiero ver una lista con las notas, para facilitar la interacción.	4	4
10	Buscar notas	Como usuario, quiero buscar una nota, para buscar un paciente en concreto.	5	4
2	Cerrar sesión	Como usuario, quiero cerrar sesión, para	2	4
9	Ordenar la lista	Como usuario, quiero ordenar la lista, para facilitar la búsqueda de notas.	3	3
6	Compartir una nota	Como usuario, quiero compartir una nota, para que mis compañeros accedan a ella.	4	3

11	Cambiar el idioma	Como usuario, quiero cambiar el idioma, para que los compañeros puedan utilizar la aplicación	5	2
12	Cambiar a modo Oscuro	Como usuario, quiero cambiar a modo oscuro, para obtener mayor accesibilidad.	1	1

Tabla 1: historias de Usuario.

2.6 Planificación inicial

Una vez obtenidas las historias de usuario, en las cuales tenemos una estimación del esfuerzo necesario a invertir en la historia de usuario y el valor que aporta al producto final es el momento de planificar los Sprints que se van a realizar y las tareas que compondrán estos Sprints. Como se ha indicado en las historias de usuario, la duración de las tareas a realizar se medirá en días.

Id	Historia de usuario	Tareas	Precedentes	Duración
A	Configuración inicial del proyecto	1. Crear un proyecto Kotlin en Android Studio 2. Añadir al proyecto un repositorio en GitHub 3. Crear un proyecto en firebase para la autentificación de usuarios. 4. Crear un proyecto node.js.	-	1
B	Iniciar Sesión	1. Diseñar la ventana de inicio de sesión 2. Crear un proyecto en firebase para la autenticación con Google 3. Implementar la funcionalidad de inicio de sesión con Google en firebase. 4. Crear un proyecto en Facebook developers para vincularlo con firebase. 5. Implementar la funcionalidad con Facebook en firebase. 6. Implementar la ruta y la funcionalidad de inicio sesión en el backend.	A	7
C	Cerrar sesión	1. Implementar el cierre de sesión del usuario.	B	2
D	Añadir una nota	1. Diseñar las vistas para añadir una nota. 2. Implementar la funcionalidad de añadir las claves de la nota. 3. Implementar la funcionalidad de añadir los campos de la nota. 4. Implementar la ruta y la funcionalidad de añadir una nota en el backend.	A	5
E	Editar una nota	1. Implementar la funcionalidad de editar una nota. 2. Implementar la ruta y la funcionalidad de editar una nota en el backend.	D,F	2

F	Eliminar una nota	1. Implementar la funcionalidad de eliminar la nota que se está visualizando. 2. Implementar la funcionalidad de eliminar varias notas seleccionadas de la lista. 3. Implementar la ruta y la funcionalidad de eliminar una nota en el backend.	A	5
G	Compartir una nota	1. Implementar la funcionalidad de compartir una nota. 2. Implementar la ruta y la funcionalidad de compartir una nota en el backend.	A	4
H	Ver una nota	1. Diseñar la vista de visualizar una nota. 2. Implementar la funcionalidad de visualizar una nota.	I	5
I	Visualizar una lista con las notas	1. Diseñar la nota a mostrar en la lista. 2. Implementar la lista de notas en la pantalla principal.	A	4
J	Ordenar la lista	1. Implementar la funcionalidad de ordenar la lista mediante las fechas de actualización de las notas. 2. Implementar la funcionalidad de ordenar la lista mediante los valores de las claves(Planta - Habitación - Cama) de las notas.	I	3
K	Buscar notas	1. Implementar la funcionalidad de buscar texto en las notas mediante una barra de búsqueda. 2. Implementar la funcionalidad de buscar notas mediante un formulario. 3. Implementar la funcionalidad de buscar notas en servidor cuando haya conexión o en local cuando no. 4. Implementar la ruta y la funcionalidad de buscar notas en el backend.	I	5
L	Variables persistentes	1. Diseñar la ventana de configuración. 2. Implementar las variables que deberán de persistir para la configuración.	A	2
M	Cambiar el idioma	1. Implementar la funcionalidad para cambiar de idioma.	L	5
N	Cambiar a modo Oscuro	1. Implementar la funcionalidad para cambiar a un tema oscuro.	L	2
O	Trabajar sin conexión	1. Implementar la funcionalidad de iniciar la aplicación sin conexión cuando se ha iniciado sesión anteriormente. 2. Implementar la funcionalidad de añadir una nota sin conexión. 3. Implementar la funcionalidad de editar una nota sin conexión. 4. Implementar la funcionalidad de eliminar una nota sin conexión.	B	6

P	Sincronizar con el servidor	1. Implementar la funcionalidad para poder sincronizar cuando se desee con el servidor.	D,E,F	2

Tabla 2: definición de las tareas a realizar.

Una vez definidas las tareas a realizar en cada historia de usuario obtenemos que para realizar el prototipo de la aplicación móvil son necesarios 56 puntos de historia. Por tanto, en cada sprint se realizan una serie de tareas que aportarán cierto valor al producto final. Sabiendo esto definimos y planificamos los sprints a realizar.

Sprint	Fecha inicio	Fecha fin	Historias de Usuario	Valor
1	21/03/2022	3/04/2002	A,B,C	9
2	4/04/2022	17/04/2022	D,F	10
3	18/04/2022	1/05/2022	E,G,I	12
4	2/05/2022	15/05/2022	H,J,L	8
5	16/05/2022	29/05/2022	N,O,P	11
6	30/05/2022	12/06/2022	K,M	6

Tabla 3: sprints a realizar.

Respecto al valor final de cada sprint, será la suma del valor de cada tarea a realizar en él. Como se ha mencionado en el apartado de metodología, realizaremos Sprints de 2 semanas de duración, empezando en la semana posterior a la que se me asignó el trabajo fin de grado.

Como podemos ver en la tabla 3, se han priorizado aquellas tareas que aportan más valor al producto para alcanzar el objetivo final, ya que en caso de surgir retrasos o imprevistos durante el desarrollo del proyecto, tendríamos un producto con mayor valor. No obstante, tenemos que tener en cuenta las relaciones de precedencia por lo que al final las tareas con más valor del producto se han ido repartiendo en las diferentes iteraciones a realizar.

A continuación se muestra un diagrama de gantt para poder visualizar una vista global de todas las iteraciones realizadas en la línea temporal establecida. Además, podremos ver cómo se relacionan las tareas, la duración del proyecto será de 60 días laborables, siendo cada sprint de 10 días laborables.

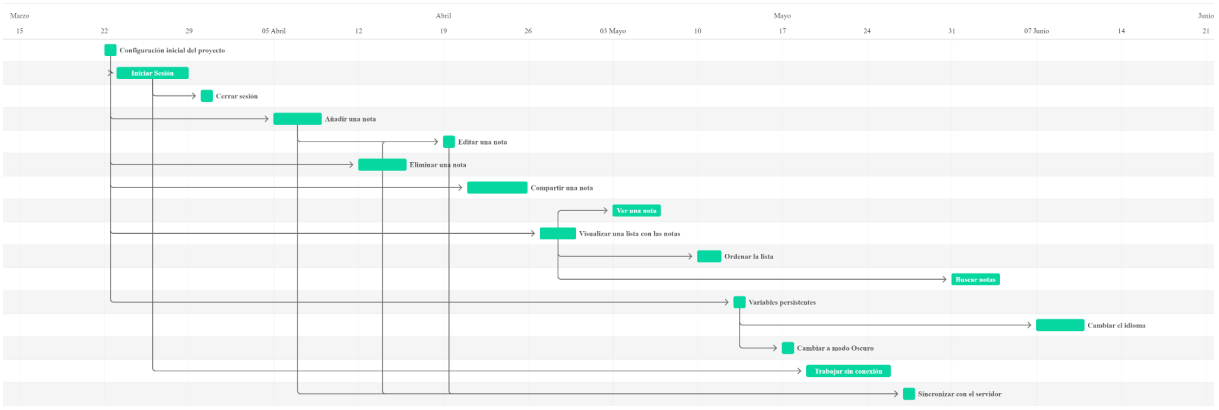


Ilustración 20: diagrama de gantt.

En caso de no poder visualizarlo correctamente puede ver la imágen en el siguiente enlace: https://drive.google.com/file/d/1t3OSTVMMAQID0i81MFXsVTSa523NAfnq/view?usp=sharing

Como podemos ver en el diagrama de Gantt, todas las tareas se realizan de forma secuencial y no hay ninguna que se realice de forma paralela, esto se debe a que en el proyecto solamente está trabajando una única persona. No obstante, con la obtención de historias de usuario y la posterior definición de las tareas a realizar, he definido unas tareas independientes que se podrían hacer perfectamente de forma paralela si el proyecto tuviese más personal.

2.6.1 Variaciones de la planificación inicial

Durante el periodo de desarrollo del proyecto han surgido alteraciones y modificaciones.

- En las primeras etapas del desarrollo del proyecto, se decidió que sería bastante útil implementar una página web que dé soporte a la aplicación móvil para que el médico pueda acceder a las notas desde el mismo ordenador en el que realiza los informes clínicos. Por lo tanto, no hay definidas tareas respecto a la página web, sin embargo estas tareas se realizaron al finalizar el sprint 6.
- Durante la realización de las tareas de la autentificación de usuario mediante Facebook, surgió el problema de que no permitía autenticar a usuarios reales sin tener disponible nuestra aplicación móvil en la Play Store de Google.

- Una vez finalizado el sprint 6, al obtener retroalimentación del cliente, se decidió dividir la presión arterial en 2 variables independientes para facilitar la toma de las constantes vitales del paciente.
- Durante la implementación de la página web surgieron problemas en la implementación de las operaciones de añadir, editar y eliminar una nota, por lo que no se pudo añadir estas funcionalidades a la página web final.

2.7 Estimación de costes

En este apartado se explicará cuales son los costes económicos de realizar el proyecto, para ello expondremos el coste de los componentes del proyecto.

El coste del software que utilizaremos en este proyecto será en su mayoría de licencia libre, solamente la licencia de visual paradigm que estamos utilizando es de pago, la licencia standard de visual paradigm tiene un coste de 19€/mes, dado que trabajamos en el proyecto durante 3 meses, el precio final será de 57 €.

Respecto al hardware que se ha utilizado para el desarrollo y la implementación del proyecto:

Hardware	Descripción	Coste	Vida útil
Ordenador portátil	Ordenador en el que se desarrollará el proyecto. Portátil HP 15s-fq2147ns, i3, 8GB, 256GB SSD	450 €	3 años
Dispositivo móvil	Huawei p20 lite de 5.85 pulgadas con Sistema operativo Android versión 9.	110 €	2 años
Tablet	Tablet Galaxy Tab A7 Lite de 8,7 pulgadas y Sistema Operativo Android versión 9.	100 €	2 años

Tabla 4: costes del hardware.

En la tabla superior se ha indicado los precios de los dispositivos físicos que se han utilizado, no obstante para cada uno de ellos deberemos calcular la amortización en el proyecto sobre su vida útil. Por tanto en los 3 meses que estaremos utilizando estos dispositivos, el ordenador portátil tendrá un coste de unos 37,5 €, el móvil tendrá un coste de 13,75 € y la tablet de 12,5 €. Es decir el coste de hardware para este proyecto será de 63,75 €.

Respecto al personal del equipo de trabajo:

Recurso humano	Coste	Porcentaje Dedicado	Total
Programador	2.340 € / mes	45 %	3.159 €
Analista	2.780 € / mes	20 %	1.668 €
Jefe de proyecto	3.070 € / mes	10 %	921 €
Administrador Base de Datos	2.500 € / mes	25 %	1.875 €
			7.623 €

Tabla 5: costes de los recursos humanos.

Para el proyecto ha realizar deberán de intervenir los 4 roles indicados en la tabla superior, debido a la planificación inicial y ha que el proyecto se trata de un proyecto individual no se ha podido dedicar el tiempo necesario en algunos roles, nos hemos centrado en la parte de programación y del gestión de la base de datos del proyecto.

Por otro lado, tenemos que tener en cuenta los gastos indirectos derivados del desarrollo del proyecto como son el internet, el agua, la electricidad, el alquiler del lugar de trabajo, posibles cambios que se produzcan a lo largo del proyecto o contratación de nuevo personal. Por tanto, indicaremos que los costes indirectos representarán un 15 % del coste total del proyecto. En conclusión, el coste final del proyecto será:

Recurso	Coste
Software	57 €
Hardware	63,75 €
Recursos humanos	7.623 €
Costes indirectos	1.161 €
Coste total del proyecto	8.904,75 €

Tabla 6: coste final del proyecto.

3. Diseño

Tras realizar la etapa de análisis y haber obtenido las historias de usuario y los requisitos funcionales y de calidad que debe tener el prototipo de aplicación móvil, se continuará con la etapa de diseño, en la cual mediante una serie de diagramas y un mockup mostraremos el modelo y las especificaciones del producto.

3.1 Diagramas E/R

Dado que en el prototipo de aplicación móvil a desarrollar podrá ser usada con conexión a internet o sin ella, se necesitarán definir dos diagramas de entidad - relación, el primero para la base de datos del servidor y el segundo para la base de datos local que tendrá el dispositivo móvil y deberá de gestionar las notas añadidas, editadas o eliminadas en el modo sin conexión.

3.1.1 Diagrama E/R - Servidor

Para la base de datos del servidor, se han definido 3 tablas.

- **Tabla Usuario**: La tabla Usuario, representa a un usuario que tiene acceso a la aplicación y a iniciado sesión como mínimo una vez. Los atributos de la tabla son:

Atributo	Tipo	Descripción
email	varchar(100)	Cuenta de Google con la que el usuario inicia sesión en la aplicación.
nombre	varchar(100)	Nombre del usuario.
fechaActualizado	varchar(100)	Fecha en la que se actualizó los datos del usuario por última vez, ya sea por el mismo o por una nota compartida.

Tabla 7: atributos de la entidad Usuario.

La clave primaria de esta tabla es el *email* ya que es única y no puede haber 2 usuarios diferentes con el mismo email.

Por otro lado, cabe destacar la elección del tipo de fechaActualizado como un Varchar pudiendo ser del tipo DateTime, no obstante esta elección se debe a la utilización de la norma ISO.8601 [13], la cual nos explica como debe de ser el formato de la fecha para su fácil migración entre distintas

plataformas y tecnologías. El formato escogido es YYYY/MM/DD hh:mm:ss, un ejemplo de una fecha en este formato sería: 2022/06/10 10:00:00. Gracias a esto nos aseguramos que todas las plataformas que se vayan a utilizar puedan obtener la fecha de actualización de usuario. Además con este formato se puede ordenar en función de la hora de la misma forma que si fuese un objeto del tipo DateTime.

- **Tabla Nota**: La tabla Nota, representa a una nota de un paciente que ha sido tomada por un Usuario, los atributos de la tabla son:

Atributo	Tipo	Descripción
planta	integer(10)	Planta en la que se toma la nota.
habitación	integer(10)	Habitación en la que se toma la nota.
cama	integer(10)	Cama en la que se toma la nota.
fechaActualizacion	varchar(100)	Fecha de la última actualización de la nota.
emailCreado	varchar(100)	Email del usuario que ha creado la nota
observaciones1	varchar(700)	Descripción de la evolución del paciente.
observaciones2	varchar(700)	Descripción del plan de actuación a seguir sobre el paciente.
ct_frecuenciaCardiaca	integer(10)	Valor de la constante vital de la frecuencia cardiaca.
ct_frecuenciaRespiratoria	integer(10)	Valor de la constante vital de la frecuencia respiratoria.
ct_temperatura	double(10)	Valor de la constante vital de la temperatura.
ct_presionArterial	integer(10)	Valor de la constante vital de la presión siastólica.
ct_presionArterial2	integer(10)	Valor de la constante vital de la presión diastólica.
compartida	bit	Variable booleana que indica si la nota está o no está compartida.

Tabla 8: atributos de la tabla Nota.

La clave primaria de está tabla es (planta - habitación - cama - fechaActualziación - emailCreado) ya que esta clave es única para una nota.

Por otro lado, el atributo emailCreado es una clave foránea de la tabla Usuario.

Respecto a los atributos no clave de la entidad, todos ellos pueden ser nulos, aunque no tiene sentido escribir una nota sin rellenar al menos uno de esos campos.

- **Tabla UsuarioNotaCrossRef**: La tabla UsuarioNotaCrossRef, representa una relación entre un Usuario y una nota a la que tiene acceso, los atributos de la tabla son:

Atributo	Tipo	Descripción
email	varchar(100)	Cuenta de Google con la que el usuario inicia sesión en la aplicación.
planta	integer(10)	Planta en la que se toma la nota.
habitación	integer(10)	Habitación en la que se toma la nota.
cama	integer(10)	Cama en la que se toma la nota.
fechaActualizacion	varchar(100)	Fecha de la última actualización de la nota.
emailCreado	varchar(100)	Email del usuario que ha creado la nota

Tabla 9: atributos de la entidad UsuarioNotaCrossRef.

La clave primaria de esta tabla son todos sus atributos ya que a su vez son claves foráneas de las 2 tablas definidas anteriormente.

Como podemos comprobar, las tablas definidas en el servidor están en la 3ª forma normal, ya que todas las tablas están en 2ª forma normal y cada uno de sus atributos no clave dependen únicamente de la clave, haciendo que no existan dependencias transitivas respecto a la clave.

Por último indicar que las tablas que poseen claves foráneas tendrán restricciones de eliminación y actualización en cascada(CASCADE) [14], con el objetivo de que cuando se elimine una nota de la base de datos, se eliminen todas las entidades UsuarioNotaCrossRef que hagan referencia a esa nota. De igual forma, cuando se elimine un usuario se eliminarán todas las notas y relaciones que están asociados a ese usuario.

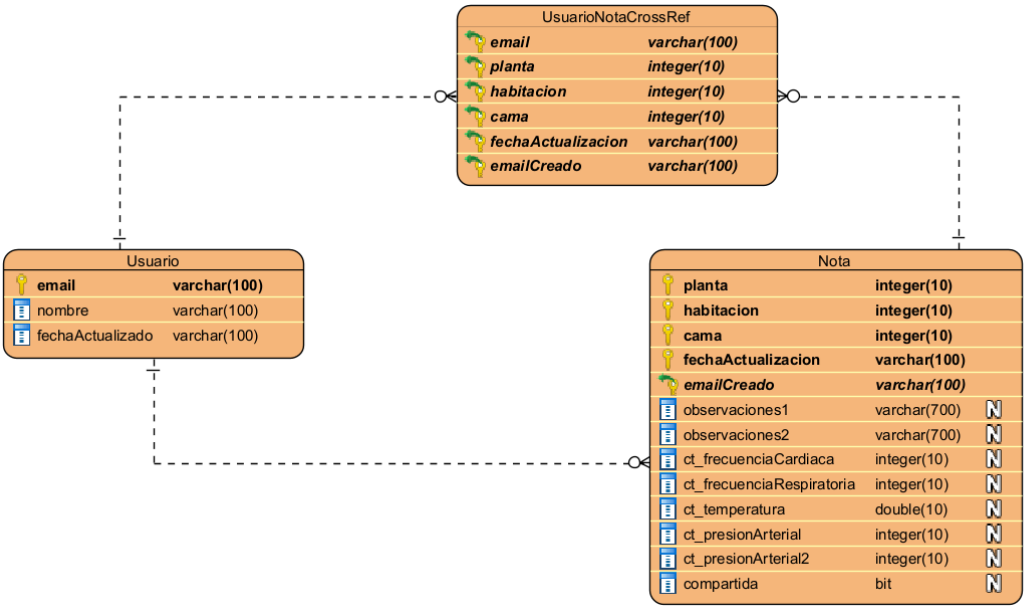


Ilustración 21: diagrama E/R del servidor.

3.1.2 Diagrama E/R - Local

Para la base de datos del servidor, se han definido 5 tablas.

- **Tabla Usuario**: la tabla Usuario, es igual a la tabla Usuario definida en el servidor.
- **Tabla Nota**: la tabla Nota, es igual a la tabla Nota definida en el servidor, solamente hay un cambio y es que el atributo emailCreado ya no pasa a ser clave foránea de la entidad Usuario, debido a que si se pusiera como clave foránea, no se podría obtener e introducir en la base de datos local aquellas notas que estén compartidas con el usuario, ya que se lanzaría una excepción al no estar registrado en la base de datos local el usuario creador de la nota.
- **Tabla UsuarioNotaCrossRef**: la tabla UsuarioNotaCrossRef, representa una relación entre una nota y un usuario.
- **Tabla NotaAdd**: la tabla NotaAdd, representa una nota que ha sido añadida o editada en local, pero no se ha podido añadir al servidor debido a que se estaba utilizando el prototipo de la aplicación sin una conexión a internet, o hubo un fallo al conectarse al servidor, los atributos de la tabla son:

Atributo	Tipo	Descripción
planta	integer(10)	Planta en la que se toma la nota.

habitación	integer(10)	Habitación en la que se toma la nota.
cama	integer(10)	Cama en la que se toma la nota.
fechaActualizacion	varchar(100)	Fecha de la última actualización de la nota.
emailCreado	varchar(100)	Email del usuario que ha creado la nota

Tabla 10: atributos de la entidad NotaAdd.

La clave primaria de la tabla son todos sus atributos, que a su vez son clave foránea de la tabla de Nota, ya que cuando la aplicación se vaya a sincronizar con el servidor, las notas a las que se hacen referencia en la tabla de NotaAdd serán añadidas al servidor.

- **Tabla NotaEliminada**: la tabla NotaEliminada, representa una nota que ha sido eliminada en local, pero no se ha podido eliminar en el servidor debido a que se estaba utilizando el prototipo de la aplicación sin una conexión a internet, o hubo un fallo al conectarse al servidor, los atributos de la tabla son:

Atributo	Tipo	Descripción
planta	integer(10)	Planta en la que se toma la nota.
habitación	integer(10)	Habitación en la que se toma la nota.
cama	integer(10)	Cama en la que se toma la nota.
fechaActualizacion	varchar(100)	Fecha de la última actualización de la nota.
emailCreado	varchar(100)	Email del usuario que ha creado la nota

Tabla 11: atributos de la entidad NotaEliminada.

La clave primaria de la tabla son todos sus atributos, que a su vez son clave foránea de la tabla de Nota, ya que cuando la aplicación se vaya a sincronizar con el servidor, las notas a las que se hacen referencia en la tabla de NotaEliminada serán eliminadas en el servidor.

Como podemos comprobar, las tablas definidas en la base de datos local están en la 3ª forma normal, ya que todas las tablas están en 2ª forma normal y cada uno de sus atributos no clave dependen únicamente de la clave, haciendo que no existan dependencias transitivas respecto a la clave.

Por último indicar que las tablas que poseen claves foráneas tendrán restricciones de eliminación y actualización en cascada(CASCADE), con el objetivo de que cuando se elimine una nota de la base de datos, se eliminen todas las entidades UsuarioNotaCrossRef y las entidades NotaAdd o NotaEliminada, que hagan referencia a esa nota. De igual forma, cuando se elimine un usuario se eliminarán todas las relaciones que están asociados a ese usuario.

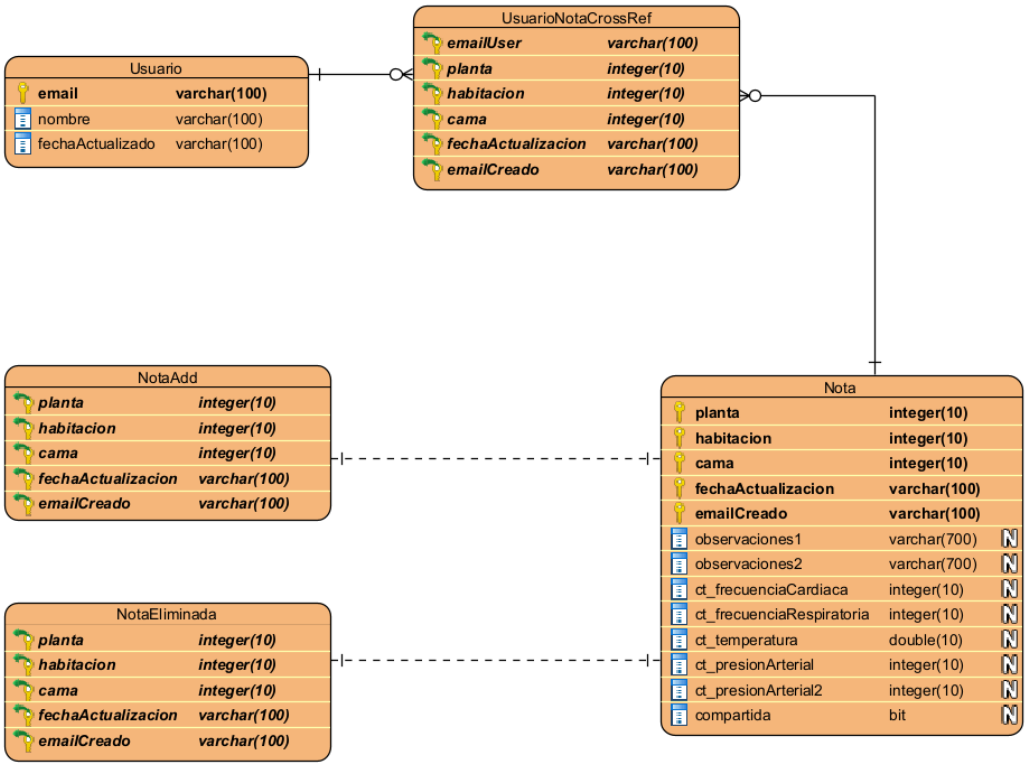


Ilustración 22: diagrama E/R del cliente móvil.

3.2 Diagramas de clases

3.2.1 Servidor

El diagrama de clases de la parte del backend estará compuesto por servicios API y por los objetos que devuelve o se le envían como payload de la petición. Diferenciaremos 3 partes en el diagrama de clases del backend, ya que tendremos que definir los servicios disponibles para cada una de las entidades declaradas en la base de datos del servidor.

A continuación, mostraremos las imágenes del diagrama de clases del servidor, en caso de que se quiera visualizar con mayor calidad se pueden ver las imágenes en la carpeta compartida del siguiente enlace:

https://drive.google.com/drive/folders/1rBnaD6l1JdmACHrvDeBBZMrcx-JxNzPo?usp=sharing

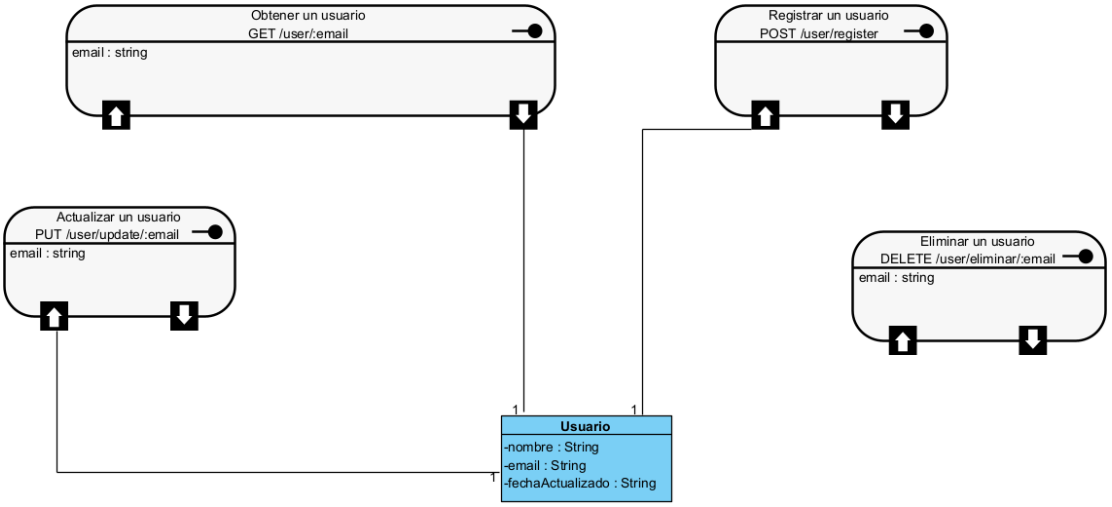


Ilustración 23: diagrama de clases del servidor del recurso Usuario.

La imágen superior muestra los servicios disponibles respecto al recurso del Usuario.

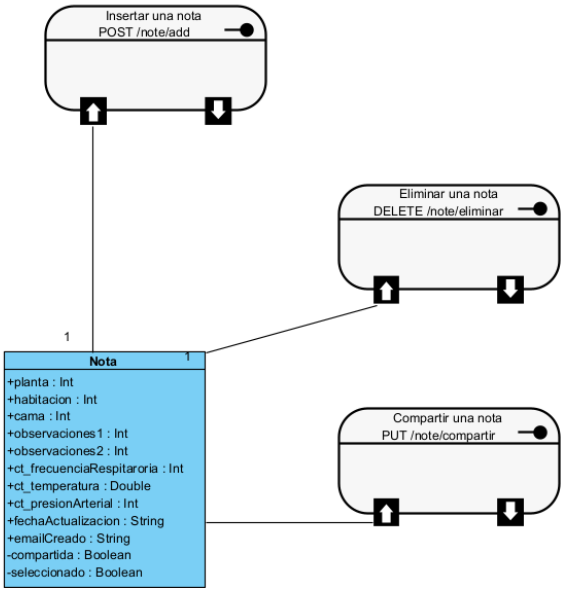


Ilustración 24: diagrama de clases del servidor del recurso Nota.

En la imágen superior podemos observar los servicios disponibles para el recurso de las notas.

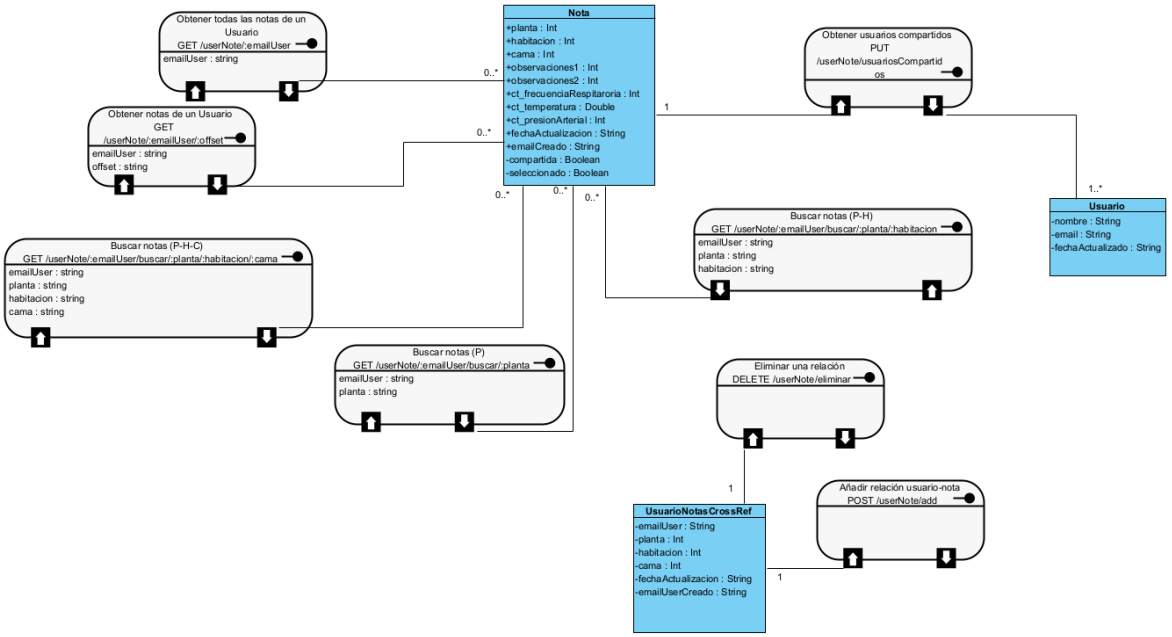


Ilustración 25: diagrama de clases del servidor del recurso UsuarioNotaCrossRef.

En la imagen superior se puede ver los diferentes servicios disponibles para el recurso de UsuarioNotaCrossRef.

Por otro lado, indicar que cada servicio de la API-REST devuelve siempre un código de respuesta para saber si la operación ha tenido éxito o ha ocurrido un error, estos códigos se han omitido de los diagramas. Los códigos son:

1. **200**- OK: código de respuesta cuando la operación ha tenido éxito
2. **400** - Bad Request: código de respuesta cuando hay un fallo en el servidor.
3. **404** - Not Found: código de respuesta cuando un recurso no existe.

3.2.2 Cliente

A continuación, mostraremos el diagrama de clases del proyecto de la aplicación móvil. Al tratarse de un diagrama de clases bastante grande, se ha tenido que dividir el diagrama en 3 imágenes para que se puedan distinguir las clases junto con sus atributos, funciones y relaciones. No obstante, el diagrama completo se puede visualizar y descargar como una imagen en el siguiente enlace, https://drive.google.com/file/d/1xe7zVogIHUr-jpMbmeB6tu1MHvZh-50t/view?usp=sharing

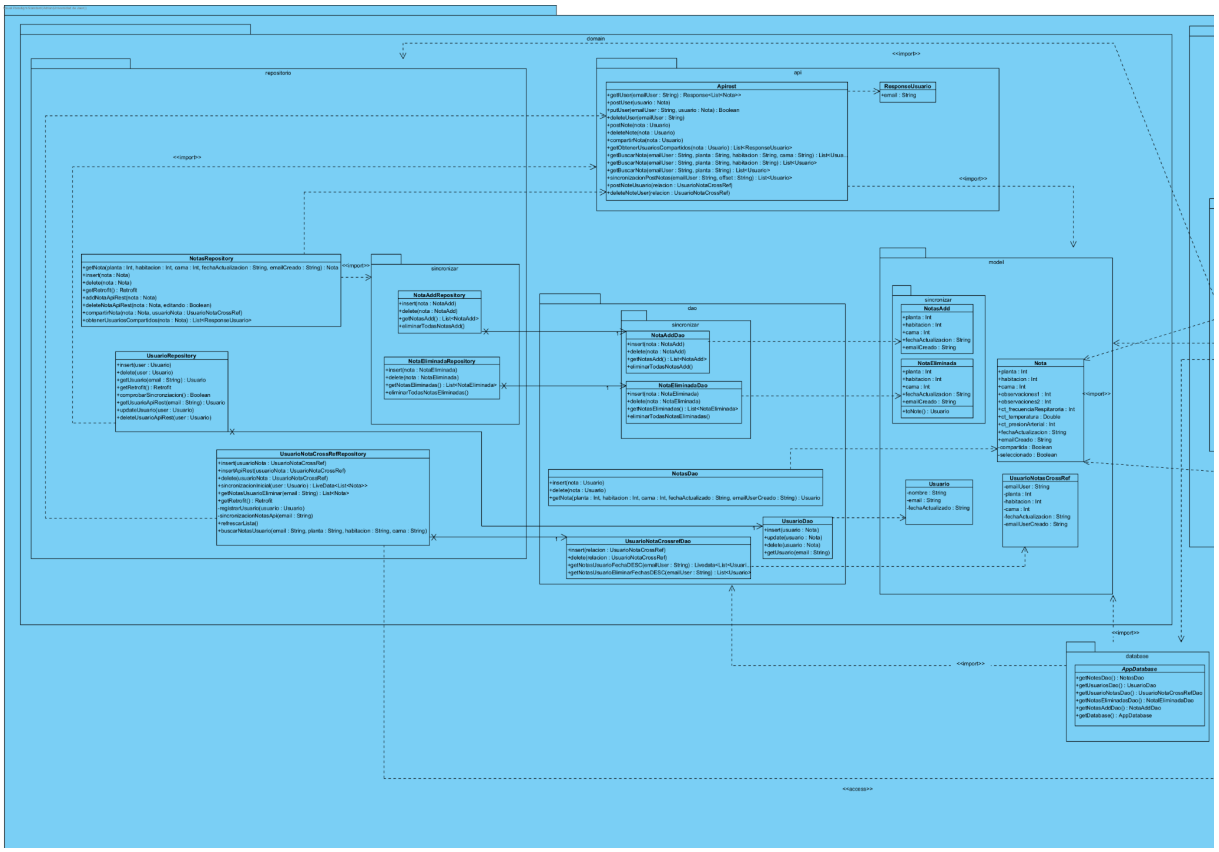


Ilustración 26: diagrama de clases del cliente móvil - parte 1.

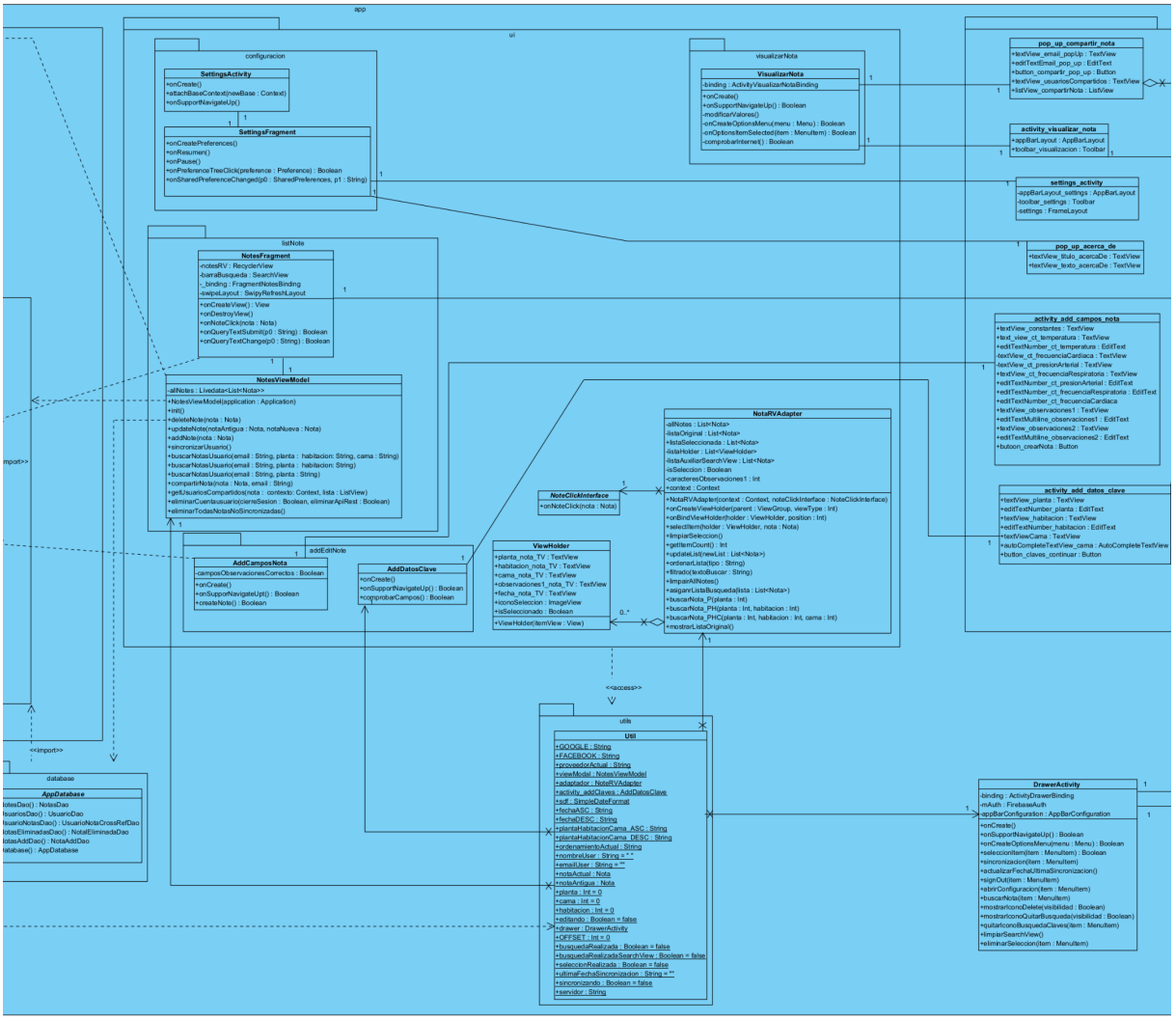


Ilustración 27: diagrama de clases del cliente móvil - parte 2.

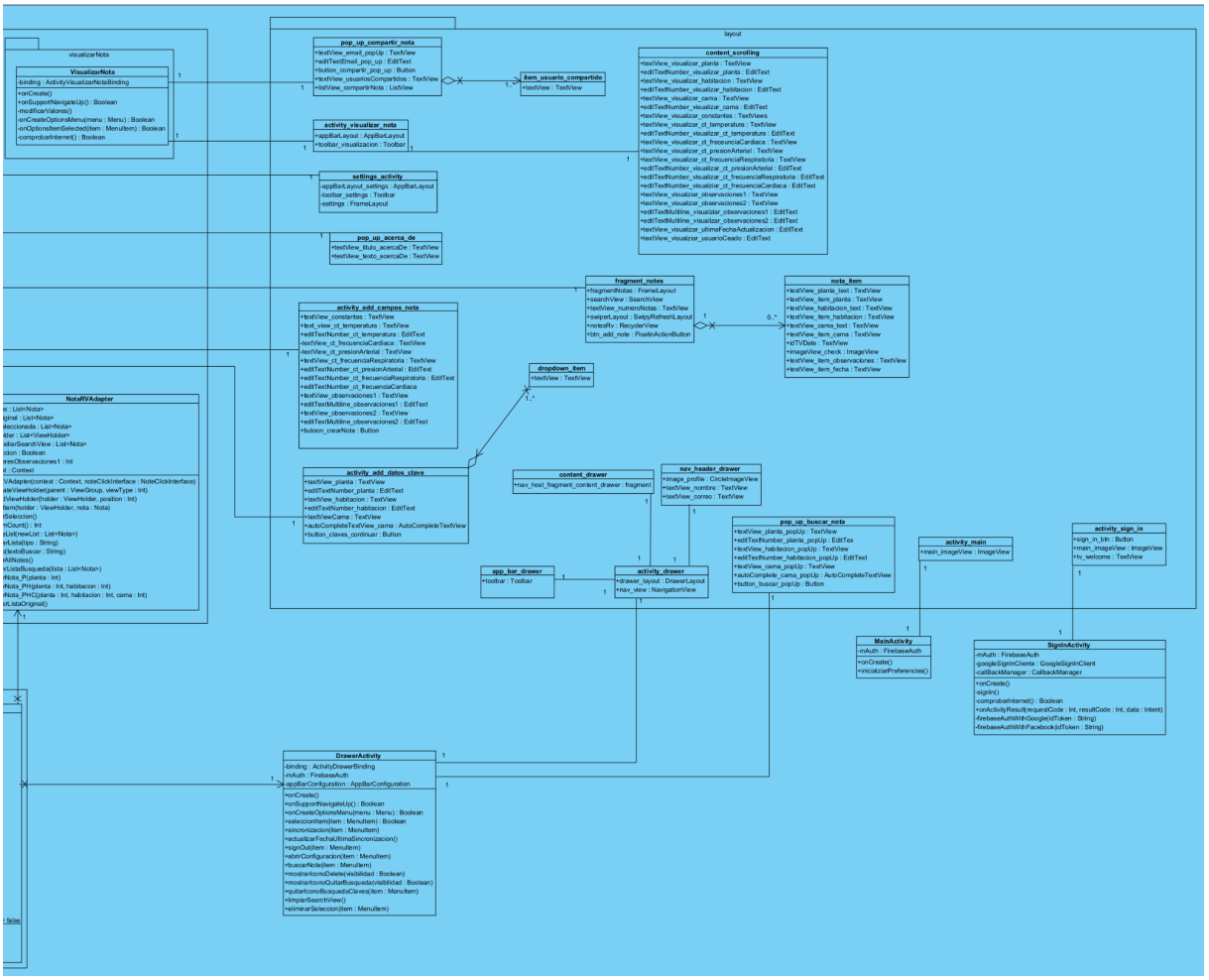


Ilustración 28: diagrama de clases del cliente móvil - parte 3.

Como podemos ver en las 3 imágenes que representan el diagrama de clases, hemos añadido al diagrama los paquetes a definir en el proyecto de Kotlin ya que se trata de un sistema bastante complejo. Al utilizar los paquetes intentaremos que las clases que se introduzcan en él estén fuertemente cohesionadas y poco acopladas entre sí. Los paquetes que se han creado son:

- **app**: Representa al proyecto Kotlin.
    - **domain**: Representa la lógica del negocio.
        - **repositorio**: Contiene los repositorios de la aplicación.
            - sincronizar: Contiene los repositorios auxiliares para la sincronización.
        - **dao**: Contiene los DAOs de la aplicación.
            - sincronizar: Contiene los DAOs auxiliares para la sincronización.
        - **model**: Contiene la entidades de la aplicación.

        - sincronización: Contiene las entidades auxiliares para la sincronización.
    - **api**: Contiene los objetos necesarios para acceder a la API-REST.
- **database**: Contiene la instancia de la base de datos local.
- **ui**: Contiene las clases que controlan las interfaces de usuario.
    - **configuracion**: Contiene las clases controladoras del fragmento de configuración.
    - listNote: Contiene las clases controladoras de la ventana principal.
    - **addEditNote**: Contiene las clases controladoras de la vista de añadir notas.
    - **visualizarNota**: Contiene la clase controlador de la vista de visualizar nota.
- **layout**: Contiene todas las interfaces de la aplicación.
- **utils**: Contiene una clase utilizada por varias clases de otros paquetes.

3.3 Diagrama de casos de uso

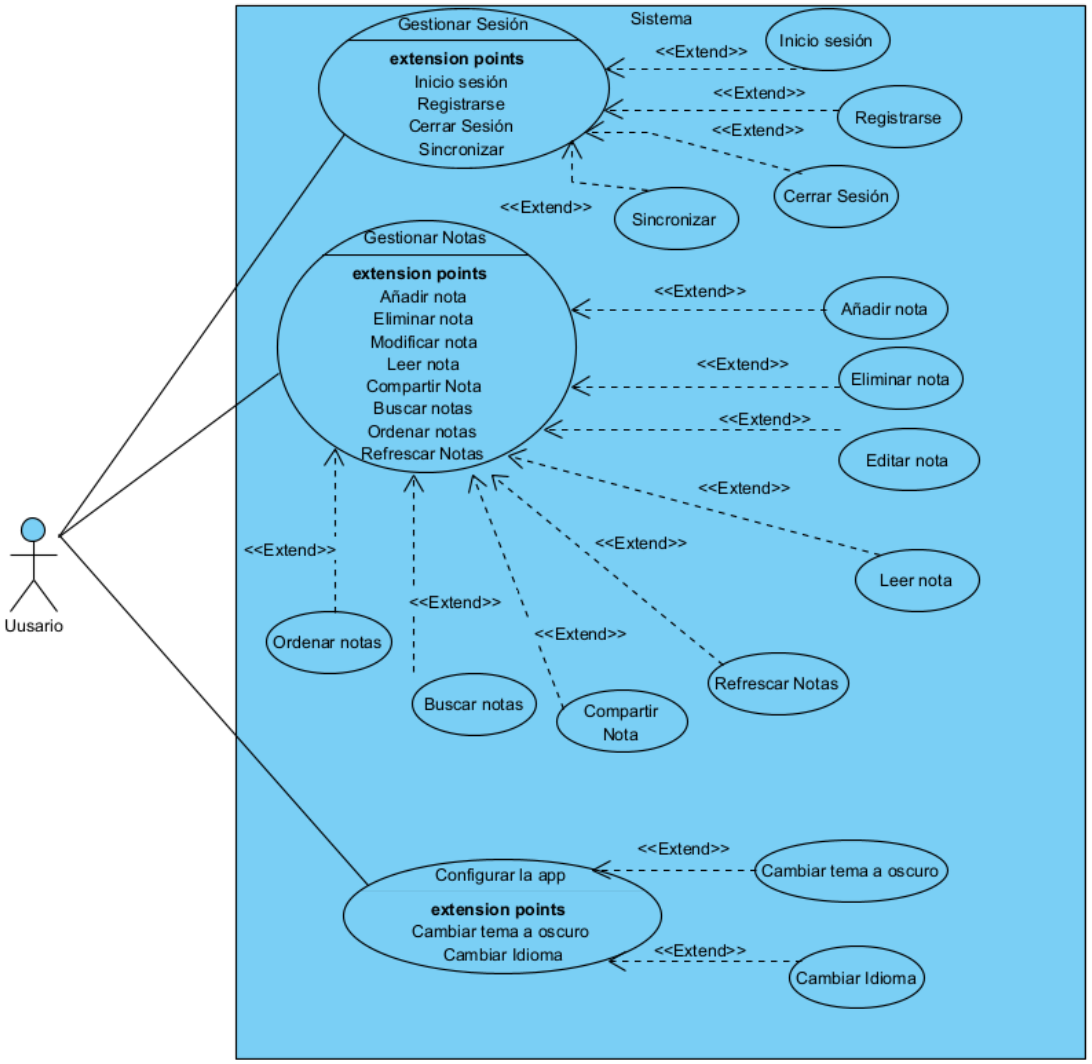


Ilustración 29: diagrama de casos de uso.

Como podemos ver en el diagrama de casos de uso, se presentan los casos de uso en función de las historias de usuario definidas en el capítulo 2.

3.4 Diagramas de secuencia

En este apartado mostraremos los diagramas de secuencia, los cuales nos permiten representar el intercambio de mensajes entre instancias del modelo y el comportamiento dinámico del prototipo de aplicación móvil. Los diagramas de secuencia a diferencia de los diagramas de comunicación muestran la interacción de las instancias que componen el prototipo, realizando diferentes casos de uso o funciones, haciendo gran énfasis en el ordenamiento temporal de los mensajes.

3.4.1 Inicio de sesión/ Registro Usuario/ Sincronización

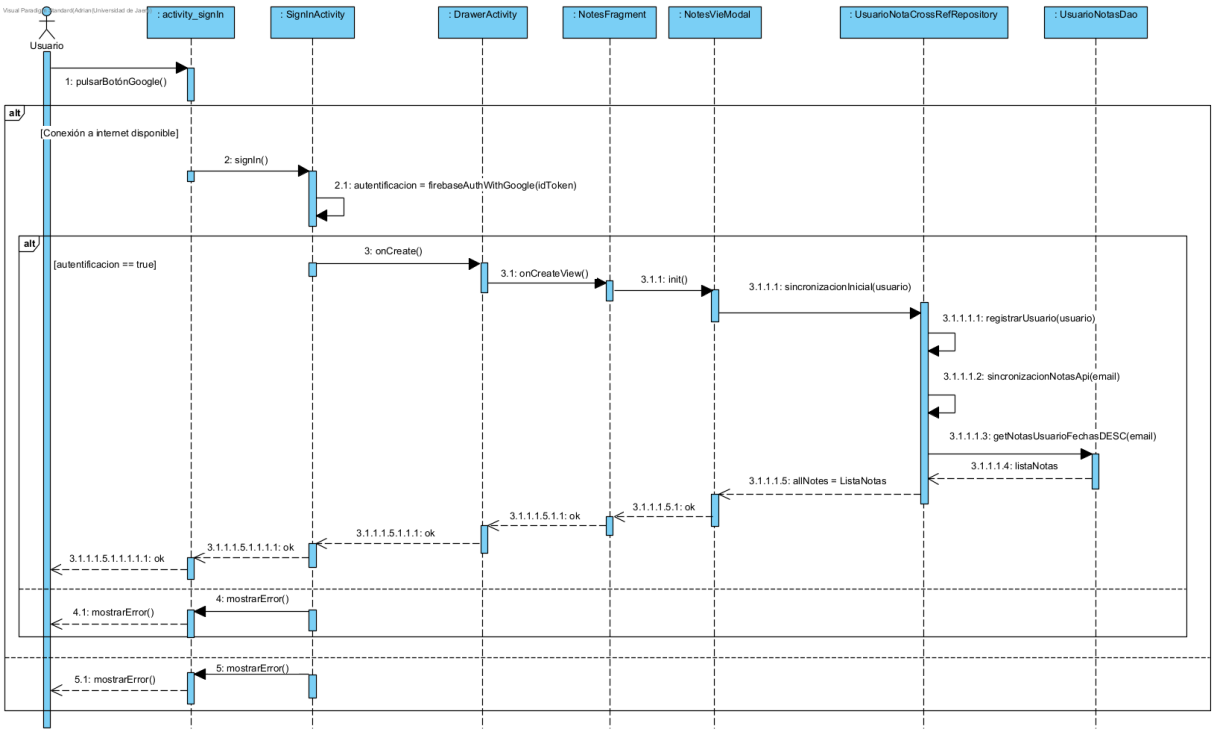


Ilustración 30: diagrama de secuencia de sincronización.

En la imagen superior podemos observar el diagrama de secuencia de los casos de uso de iniciar sesión, registro del usuario y de la sincronización.

Respecto a la interacción, el usuario pulsa el botón de Google en la interfaz y se comprueba que el usuario tiene conexión a internet, si el usuario tiene conexión a internet le aparecerá una ventana emergente para seleccionar o introducir la cuenta de Google y comenzar la autenticación. En caso contrario se le mostrará una alerta con la información del error.

Una vez que el usuario está autenticado, se redirigirá a la vista principal de la aplicación. La instancia de *NotesViewModel*, se encargará de gestionar todo el proceso de registro del usuario y de la sincronización con el servidor, para ello se accederá a los repositorios necesarios y se realizarán las llamadas correspondientes a la API-REST.

Por último, cuando se haya terminado la sincronización con el servidor se actualizará la lista que aparece en la vista principal para mostrar las notas recibidas por parte del servidor.

Para poder mostrar la interacción de las funciones de registrarUsuario() y sincronizacionNotasApi() se han realizado dos diagramas de secuencia separados.

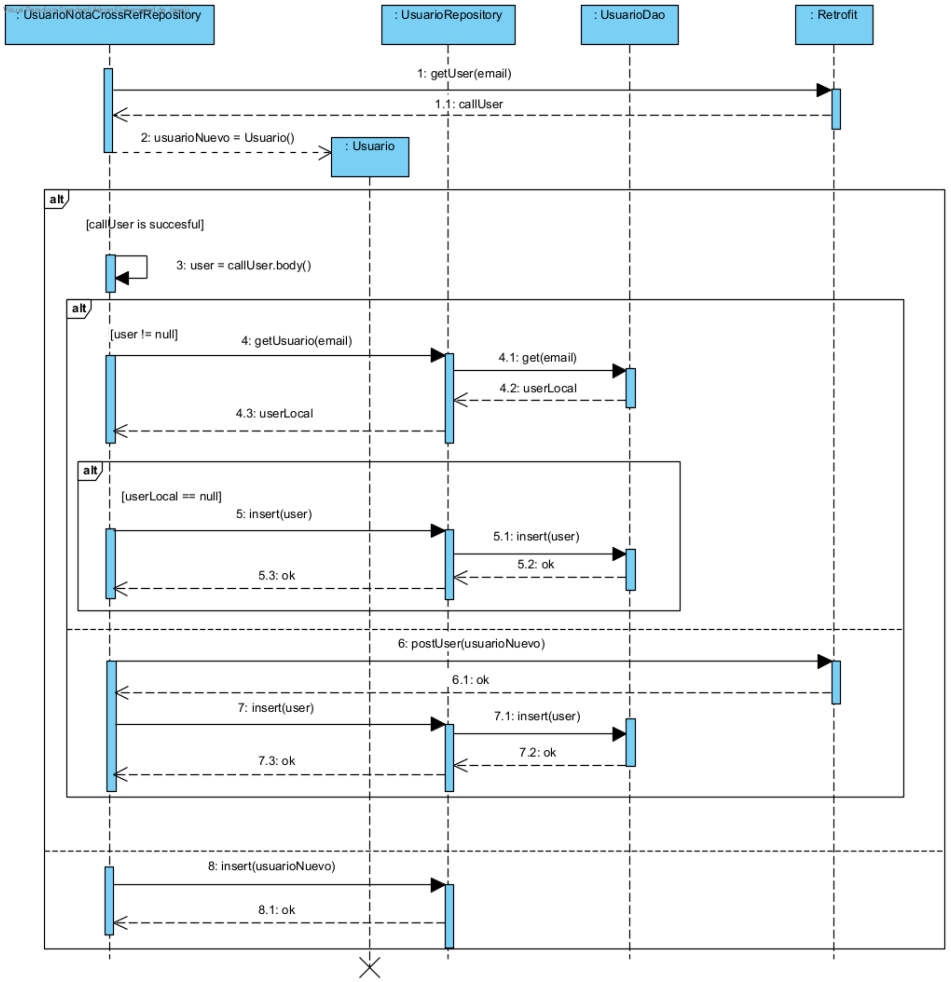


Ilustración 31: diagrama de secuencia de la función registrarUsuario.

Como podemos observar en el diagrama superior, se representa a la función de registrarUsuario() declarada en la clase de *UsuarioNotaCrossRefRepository*.

Lo primero de todo es realizar una llamada a la API-REST para obtener el usuario del servidor, para así comprobar si ya se ha registrado anteriormente. Comprobamos si nos ha devuelto un usuario y procedemos a comprobar si ya está registrado el usuario en la base de datos local. Si el usuario está registrado en la base de datos local, no se hace nada, pero en caso contrario introducimos el usuario del servidor en la base de datos local.

Por otro lado, si el usuario que nos ha devuelto el servidor no es válido, es decir no existe en el servidor, procedemos a introducir un usuario nuevo en la base de datos local y en la base de datos del servidor.

Una vez terminada la función de registrarUsuario(), se procederá a realizar la función de sincronizacionNotaApi().

El diagrama de secuencia de esta función resulta ser bastante grande debido a la complejidad de la función, por lo que se ha separado el diagrama de secuencia en 2 imágenes.

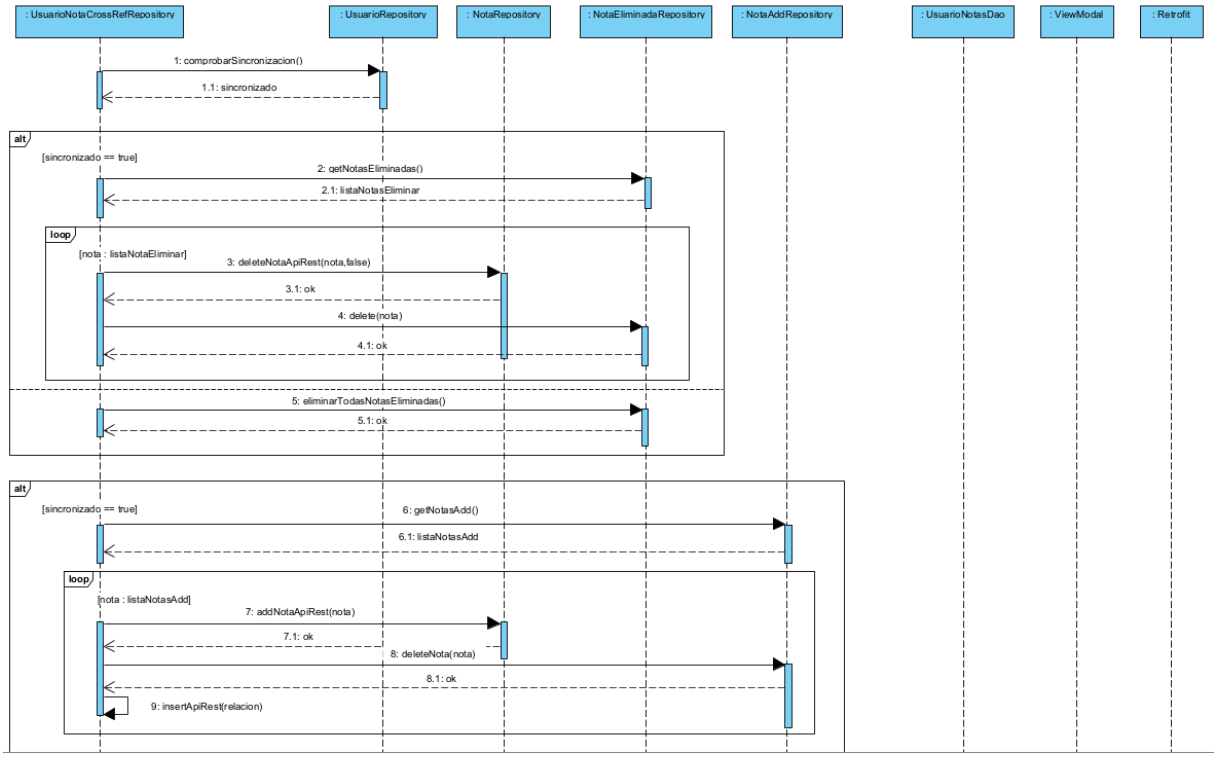


Ilustración 32: diagrama de secuencia de la función sincronizaciónNotasApi - parte 1.

Como podemos ver en la ilustración 32, corresponde a la sincronización de las notas gestionadas en el modo sin conexión.

Lo primero que se hace es comprobar si el usuario local tiene la misma fecha de Actualizado que el usuario del servidor, en caso de que tengan las mismas fechas se procederá a eliminar primero aquellas notas que se han eliminado en local y posteriormente se procederá a añadir las notas que están en la base de datos local y no en la base de datos del servidor. En caso de que las fechas sean distintas, se procederá a eliminar todos los datos de la base de datos local.

Una vez realizada las operaciones para añadir las notas creadas, eliminadas o editadas sin conexión procedemos a actualizar las fechas de actualización del usuario, tanto en local como en el servidor.

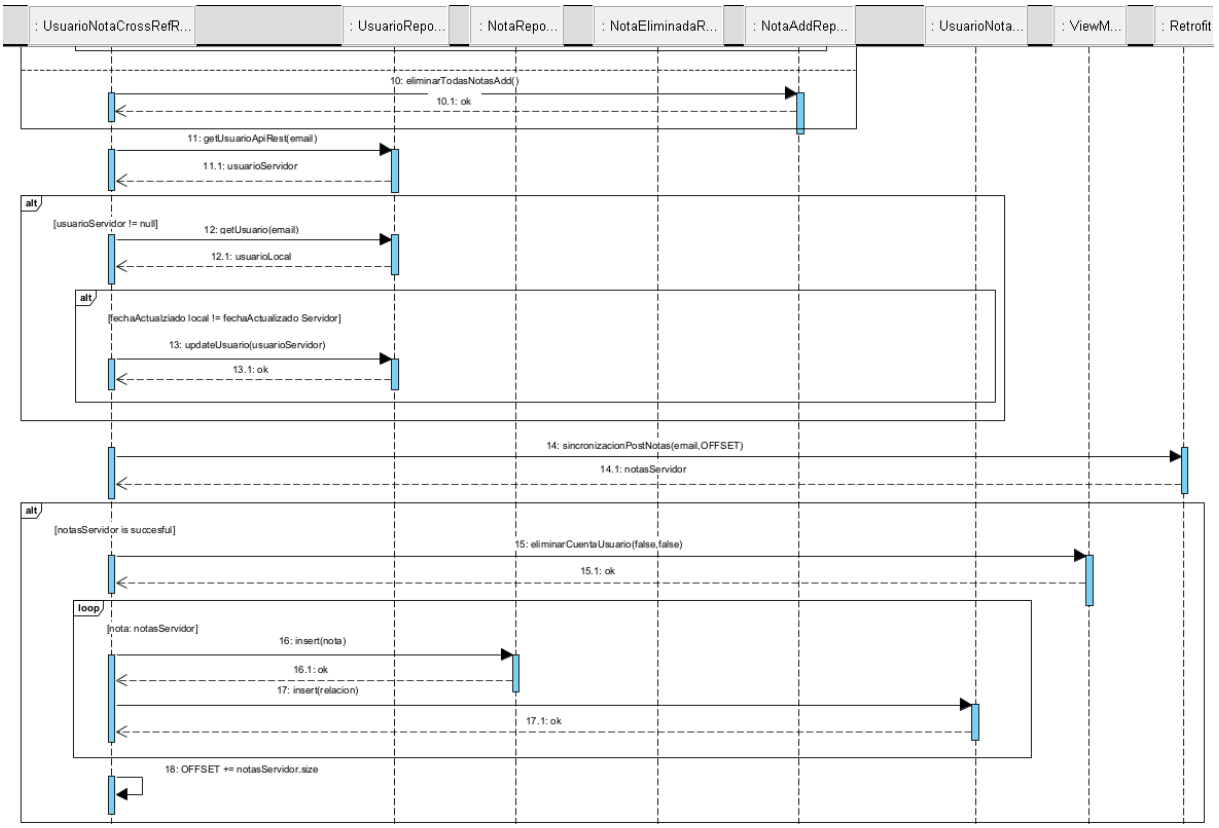


Ilustración 33: diagrama de secuencia de la función sincronizaciónNotasApi - parte 2.

Posteriormente actualizamos las fechas de actualización del usuario del servidor y del usuario de la base de datos local.

Una vez actualizadas las fecha de actualización del usuario procedemos a obtener las notas del servidor. Para ello, lo primero será realizar una llamada al servidor en la que le indiquemos que queremos obtener las notas del usuario.

Una vez obtenidas las notas del servidor procedemos a añadirlas en la base de datos local.

3.4.2 Cerrar sesión

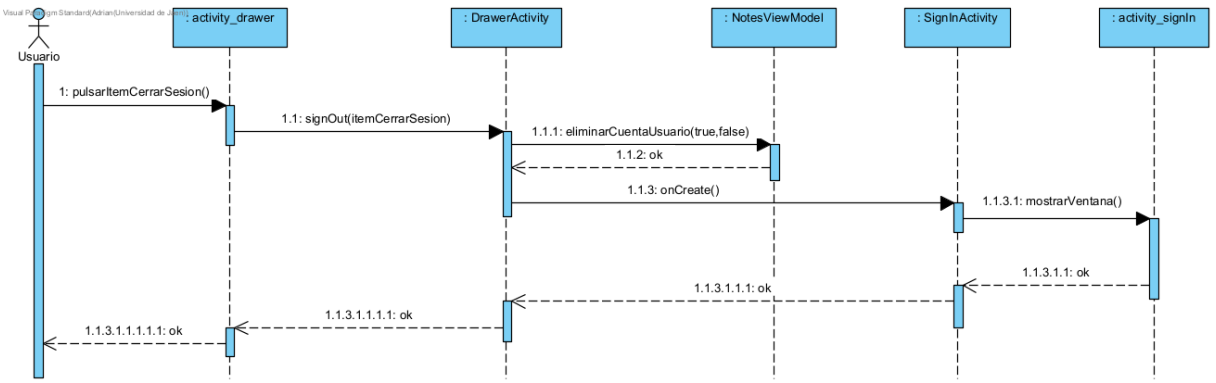


Ilustración 34: diagrama de secuencia de cerrar sesión.

En la imagen superior podemos observar el diagrama de secuencia del caso de uso de cerrar sesión.

Respecto a la interacción,el usuario pulsa el botón de cerrar sesión, el cual está dentro del menú desplegable

Posteriormente, se llamará a la instancia de *NotesViewModel* gestionar el cierre de sesión de usuario. Por último se redirigirá al usuario a la ventana de inicio de sesión.

3.4.3 Eliminar nota

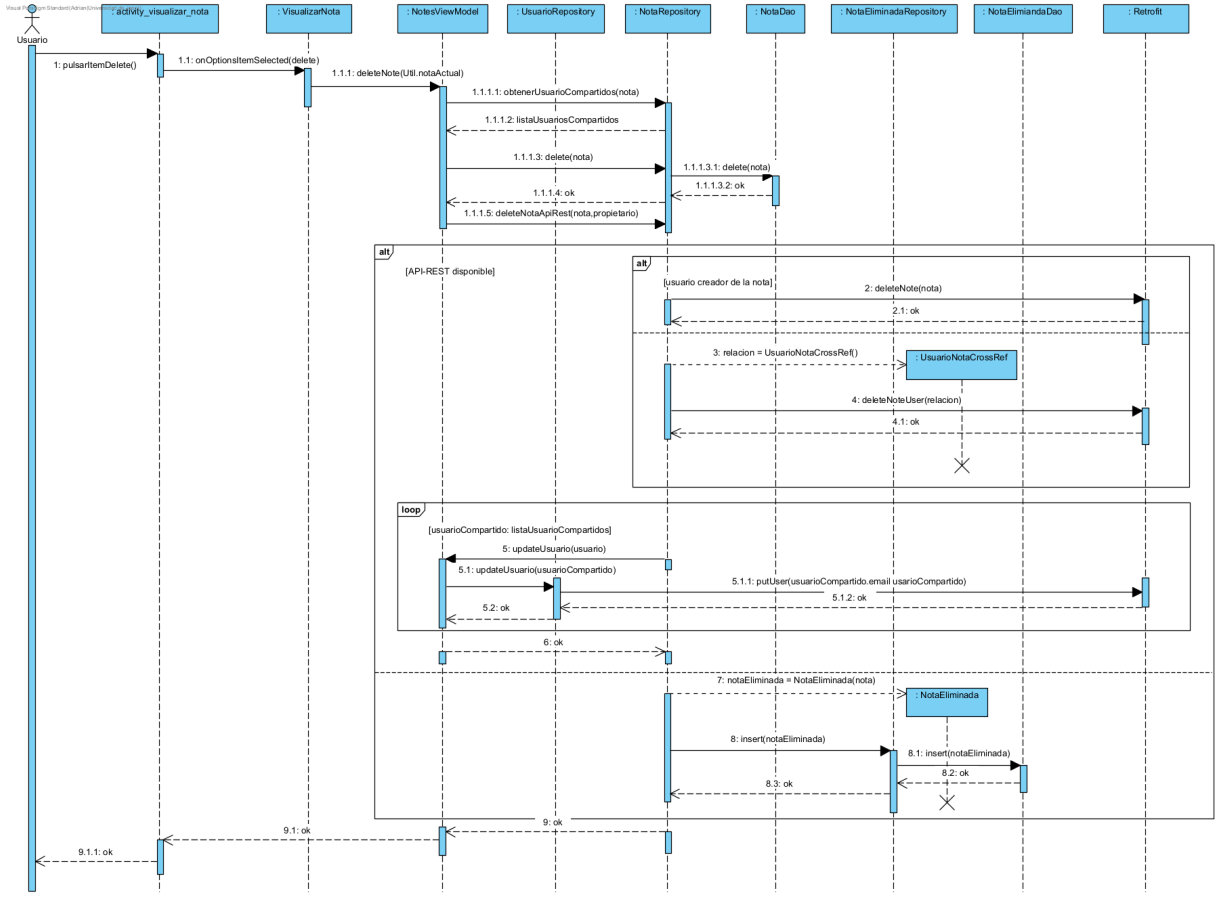


Ilustración 35: diagrama de secuencia de eliminar nota.

En la imagen superior podemos observar el diagrama de secuencia del caso de uso de eliminar una nota.

Respecto a la interacción, para eliminar una nota el usuario tiene 2 formas de hacerlo, pudiendo visualizar la nota y seleccionando el ítem de eliminar, o puede seleccionar una o varias notas en la lista que se muestra en la pantalla principal y una vez que están las notas seleccionadas se pulsa un icono de una papelera para eliminar las notas. El diagrama de secuencia que se muestra a continuación

representa el caso en el que se está visualizando una nota, pero sirve para ambos casos ya que solamente cambia en que en el caso de la selección de notas de la lista se cambia el item por un icono.

Lo primero que hace el usuario para eliminar una nota es visualizar la nota y pulsar el ítem de eliminar en la ventana de visualizar la nota. Una vez pulsado el ítem se llamará a *NotesViewModel* para que elimine la nota que se está visualizando.

A continuación se obtendrán todos los usuarios que tienen acceso a la nota para poder actualizar su fecha de actualización en el servidor. Una vez hecho esto se eliminará la nota en local.

Una vez eliminada la nota de la base de datos local, se intentará eliminar en el servidor, si el servidor está disponible y el usuario tiene una conexión a internet se eliminará la nota en el servidor solamente si el usuario es el creador de la nota, en caso de que no sea el creador solamente se le eliminará el acceso. Por último se actualizará la fecha de actualización a todos los usuarios que tenían acceso a la nota.

En caso de que no se haya podido eliminar la nota en el servidor, se creará una *NotaEliminada* para añadirla a la base de datos local y eliminar la nota del servidor la próxima vez que se vaya a sincronizar el dispositivo móvil con el servidor.

3.4.4 Añadir/Editar nota

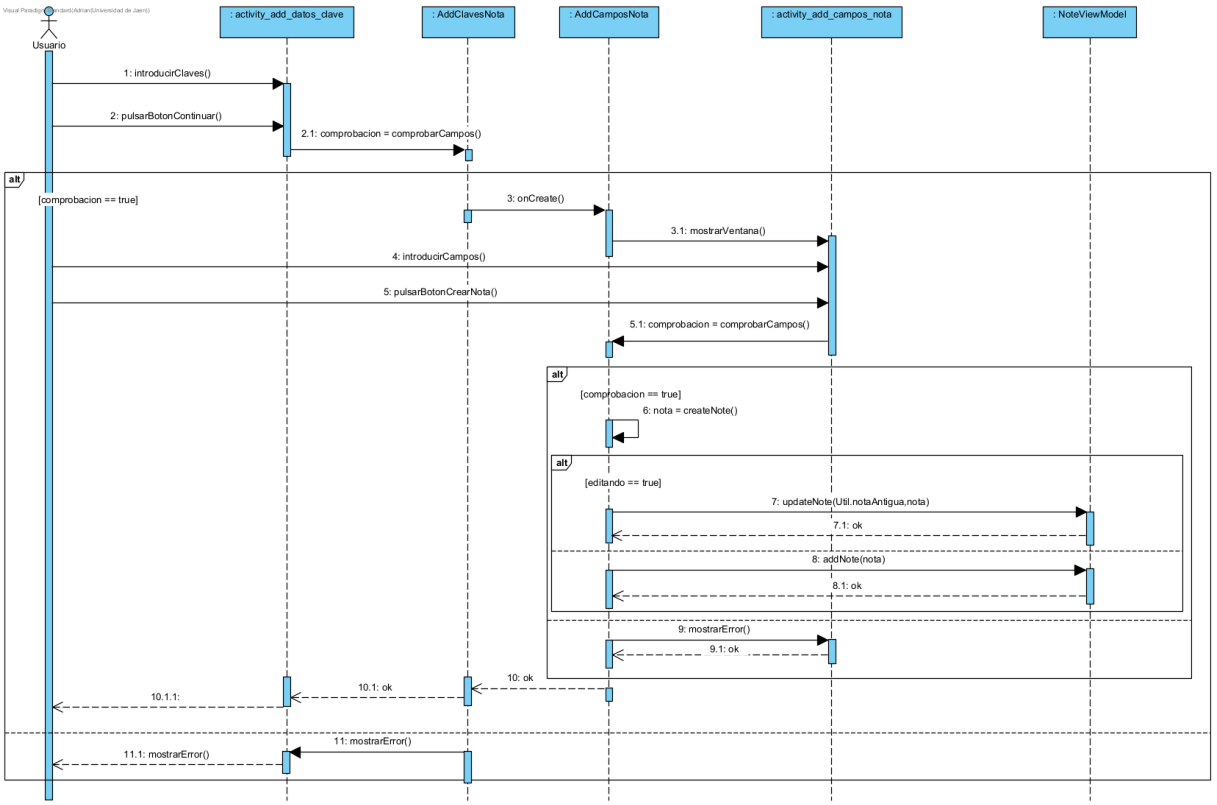


Ilustración 36: diagrama de secuencia de añadir/editar nota.

En la imagen superior podemos observar el diagrama de secuencia de los casos de uso de añadir y editar una nota. Respecto a la interacción, tanto para añadir una nota como para editarla son las mismas ventanas, la única diferencia es que cuando se edite una nota, aparecerán los campos de la ventana rellenados por los datos de la nota.

Lo primero que realiza el usuario es introducir las claves de la nota y pulsar el botón de continuar, cuando se pulse el botón de continuar, se comprobará en el controlador que los datos son correctos. Si los datos son correctos se pasará a la ventana de añadir los campos, si son incorrectos, se mostrará los errores.

En la ventana de añadir los campos, sigue la misma mecánica que la vista anterior, el usuario introduce los datos de la nota y pulsa el botón de crear una nota. Se comprobará si los datos son correctos o no, en caso de que no lo sean se mostrará los errores.

Por último, una vez que se ha comprobado que todos los datos son correctos se procederá a llamar al *NoteViewModel* para añadir o editar la nota.

Para poder mostrar la interacción de las funciones de addNote() y de updateNote() se han realizado dos diagramas de secuencia separados.

3.4.4.1 Añadir Nota

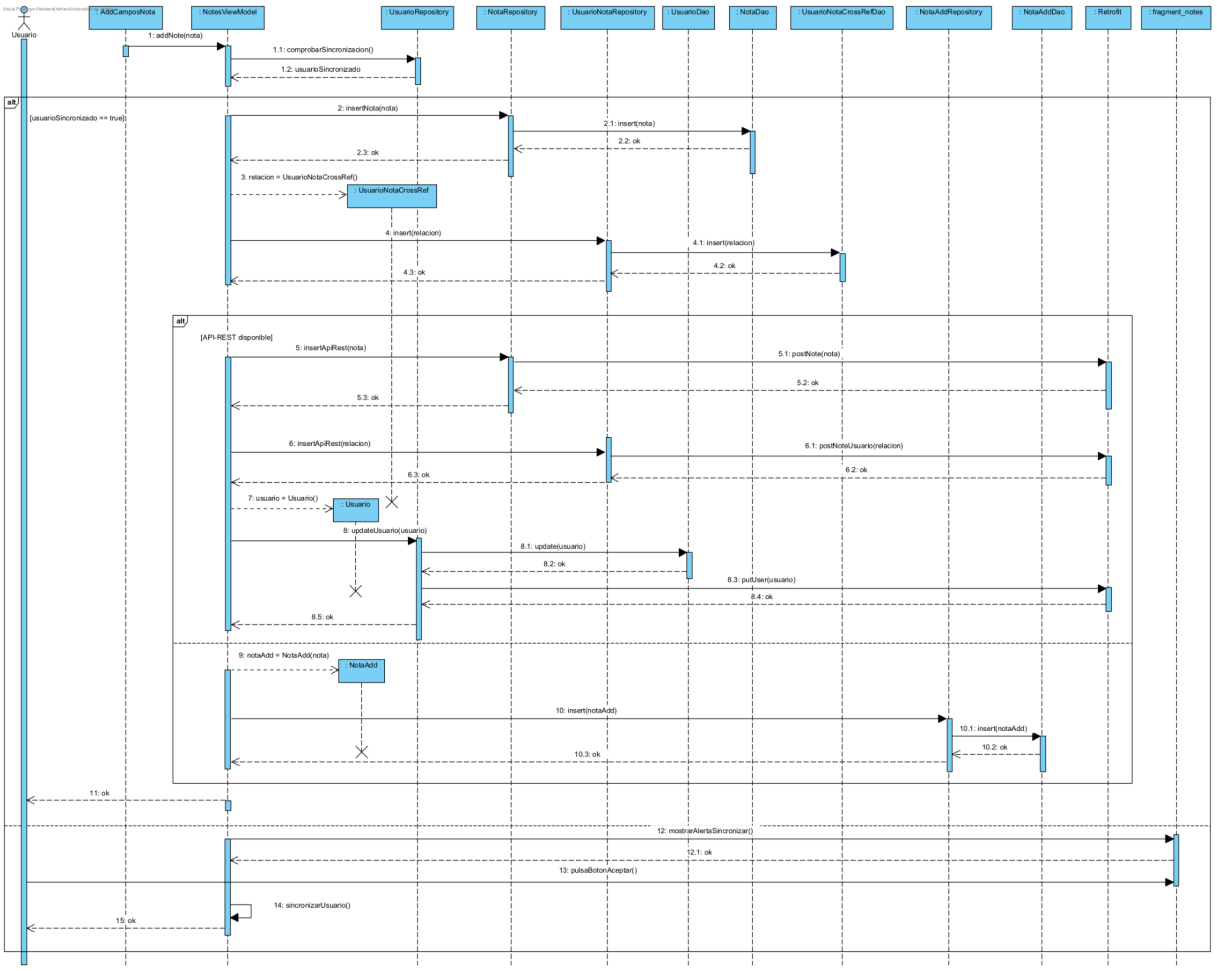


Ilustración 37: diagrama de secuencia de añadir nota.

En la imagen superior, podemos ver el diagrama de secuencia que hace referencia a la función de addNote().

Respecto a la interacción de la función, lo primero es comprobar si la aplicación está sincronizada con el servidor. Si la aplicación no está sincronizada se muestra una alerta para sincronizar el usuario con el servidor, si se acepta realizar la sincronización realiza la sincronización con el servidor, cuyo diagrama de secuencia ha sido explicado anteriormente.

Por otro lado, si la aplicación está sincronizada se procederá a insertar la nota en la base de datos local llamando a los repositorios correspondientes y a su

respectivos DAOs. Una vez insertada la nota en la base de datos local, se procederá a añadir la nota en la base de datos del servidor si el servidor está disponible.

Si el servidor está disponible, simplemente se añade la nota y la relación al servidor llamando a las funciones correspondientes de los repositorios y las llamadas correspondientes a la instancia de Retrofit. Posteriormente, se procederá a actualizar el usuario tanto en local como en el servidor para tener las fechas sincronizadas.

En caso de que la API-REST no esté disponible, se procederá a crear una instancia de *NotaAdd* para que posteriormente cuando se disponga de una conexión a internet o esté disponible la API-REST se pueda añadir.

3.4.4.2 Editar nota

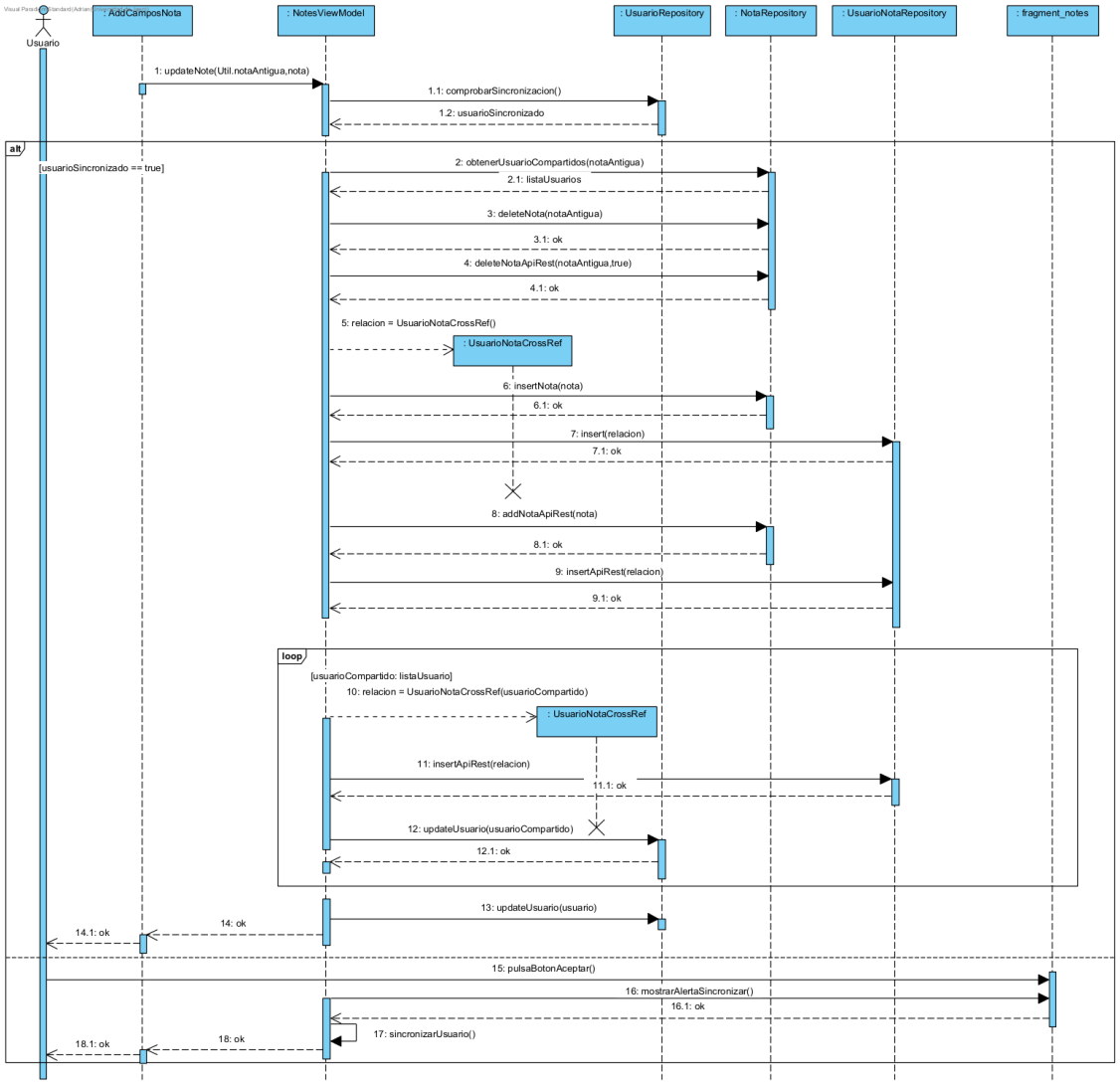


Ilustración 38: diagrama de secuencia de editar nota.

En la ilustración 38 se muestra el diagrama de secuencia de la función de editar una nota.

En este diagrama de secuencia se ha obviado la interacción al añadir una nota y eliminarla, ya que esa interacción ha sido definida en diagramas de secuencia anteriores.

Por otro lado, en este caso de uso también surge el escenario alternativo donde se trabaja sin conexión, en ese caso se realiza lo mismo que se ha explicado en los diagramas de secuencia de añadir una nota y el de eliminar una nota.

Respecto a la interacción de la función, lo primero es comprobar si la aplicación está sincronizada con el servidor, igual que se hacía en los diagramas anteriores.

Una vez que se ha comprobado que el usuario local está sincronizado con el servidor, se procederá a obtener todos los usuarios que tienen acceso a la nota, posteriormente se borrará la nota en el cliente y en el servidor, después se añadirá la nota y la relación nueva tanto en local como en el servidor y se actualizará el usuario para que esté sincronizado con el servidor.

Por último, se añadirá la relación de acceso en el servidor a todos los usuario que tenían acceso a la nota y se actualizarán los usuarios en el servidor para que estén sincronizados.

3.4.5 Compartir nota

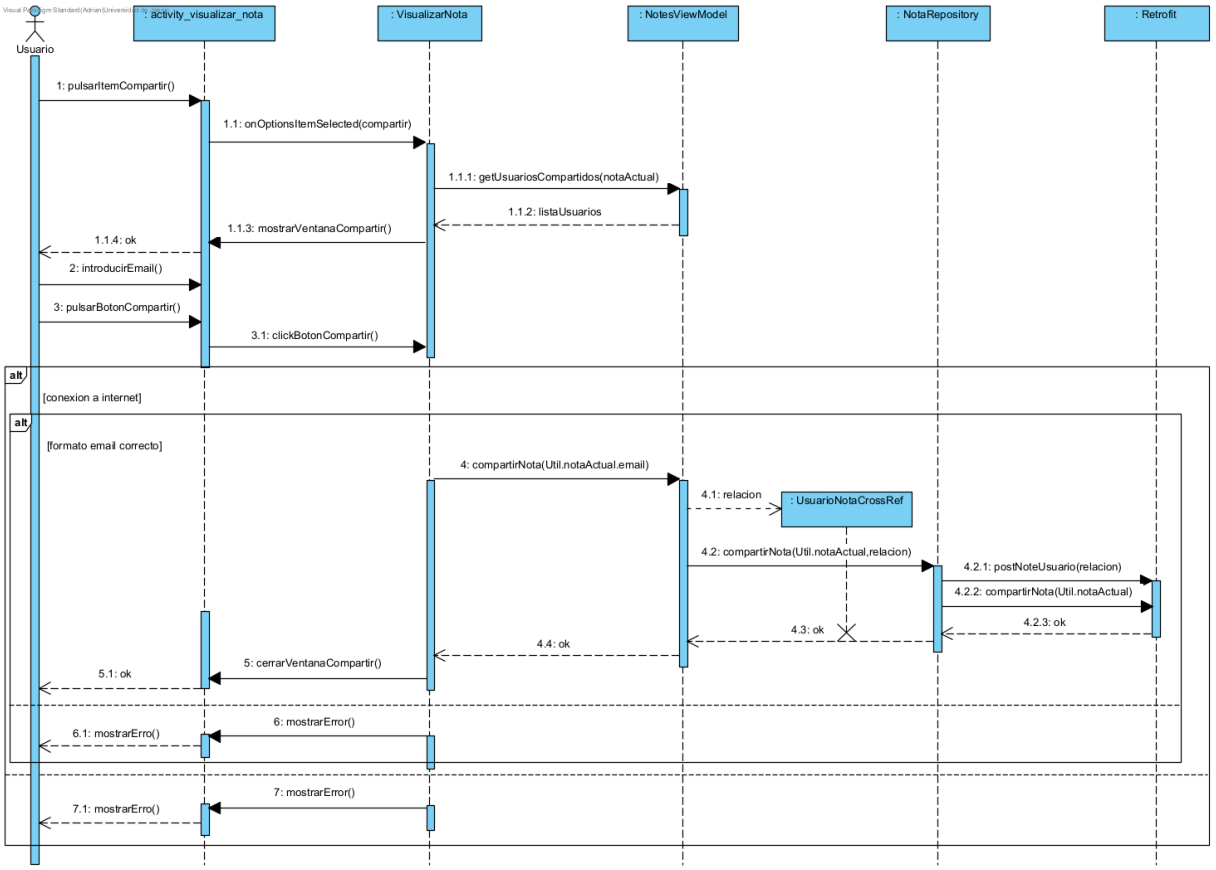


Ilustración 39: diagrama de secuencia de compartir nota.

En la imagen de arriba se muestra el diagrama de secuencia de la función de compartir una nota.

Lo primero de todo es pulsar el ítem de compartir que está en la vista de visualizar nota, cuando se pulsa, se llama controlador de la vista de la visualización de la nota para gestionar la selección del ítem de compartir una nota.

Posteriormente, se llamará a la instancia de *NotesViewModel*, para mostrar al usuario una ventana emergente en la que el usuario pueda ver los correos de los usuarios que tienen acceso y poder compartirla a otros usuarios.A continuación, el usuario introducirá un correo electrónico de un usuario de la aplicación y le dará al botón de compartir. Se comprobará que el usuario dispone de una conexión a internet y que el correo electrónico introducido sigue un formato correcto. En caso de que haya algún error, se mostrará al usuario el error que ha ocurrido.

Una vez que se ha comprobado todo, se llamará a la instancia *NotesViewModel*, para indicarle al repositorio correspondiente que se debe compartir la nota con un nuevo usuario.

3.4.6 Ver nota

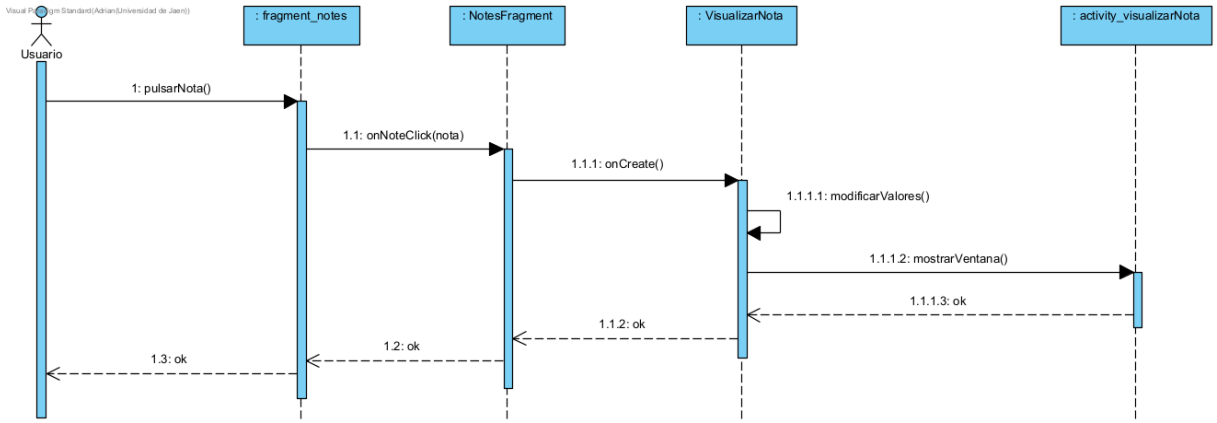


Ilustración 40: diagrama de secuencia de ver nota.

En la imagen superior se muestra el diagrama de secuencia del caso de uso de visualizar una nota.

Respecto a la interacción del diagrama, el usuario toca o hace un click en cualquier nota de la lista que se muestra en la pantalla principal. Una vez hecho esto, se llamará al controlador de la vista de notas para detectar el click en la nota y abrir una nueva vista para visualizar la nota.

Posteriormente, se modificará los valores a mostrar en la vista para que coincida con la nota seleccionada.

3.4.7 Buscar nota

Para poder buscar una o varias notas hay 2 formas de hacerlo, mediante la barra de búsqueda en la que se introduce texto y se filtran las notas de la base de datos local que contienen el texto introducido, o se puede buscar mediante un formulario en el que se introducen los campos de planta, habitación y cama de la nota, en el caso del formulario, la búsqueda se realiza siempre que se pueda en el servidor, si hay una conexión a internet y está el servidor disponible.

3.4.7.1 Buscar nota con barra de búsqueda

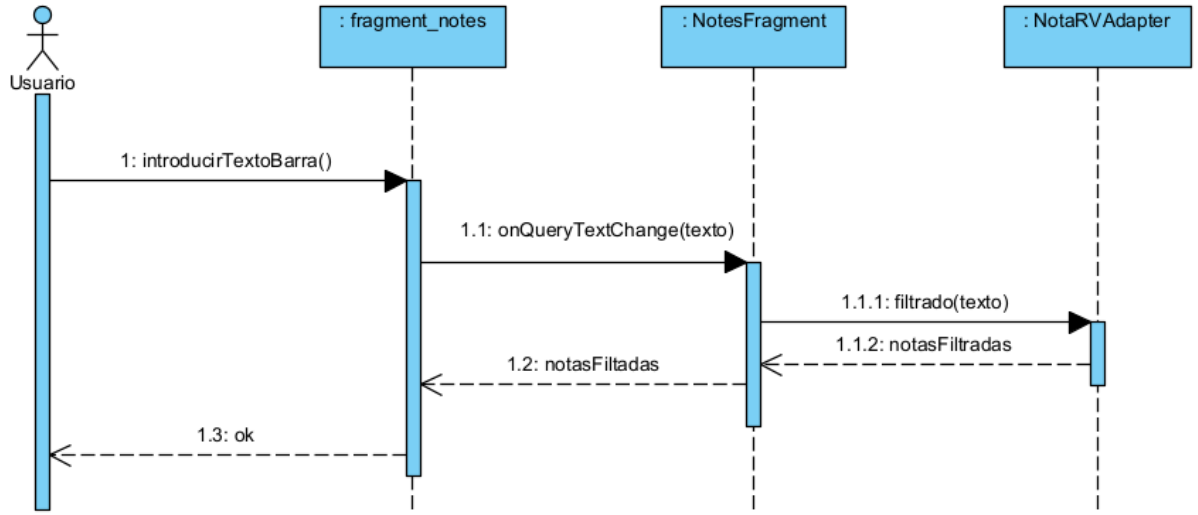


Ilustración 41: diagrama de secuencia de buscar notas en la barra de búsqueda.

El diagrama de secuencia de arriba hace referencia a la búsqueda de notas mediante la barra de búsqueda.

El usuario introduce texto en la barra de búsqueda y conforme se vaya cambiando el texto introducido se va llamando al controlador de la vista principal, posteriormente se llamará a la clase *NotaRVAdapter* para realizar el filtrado de las notas que hay en la base de datos local y contienen el texto introducido.

3.4.7.2 Buscar nota con formulario

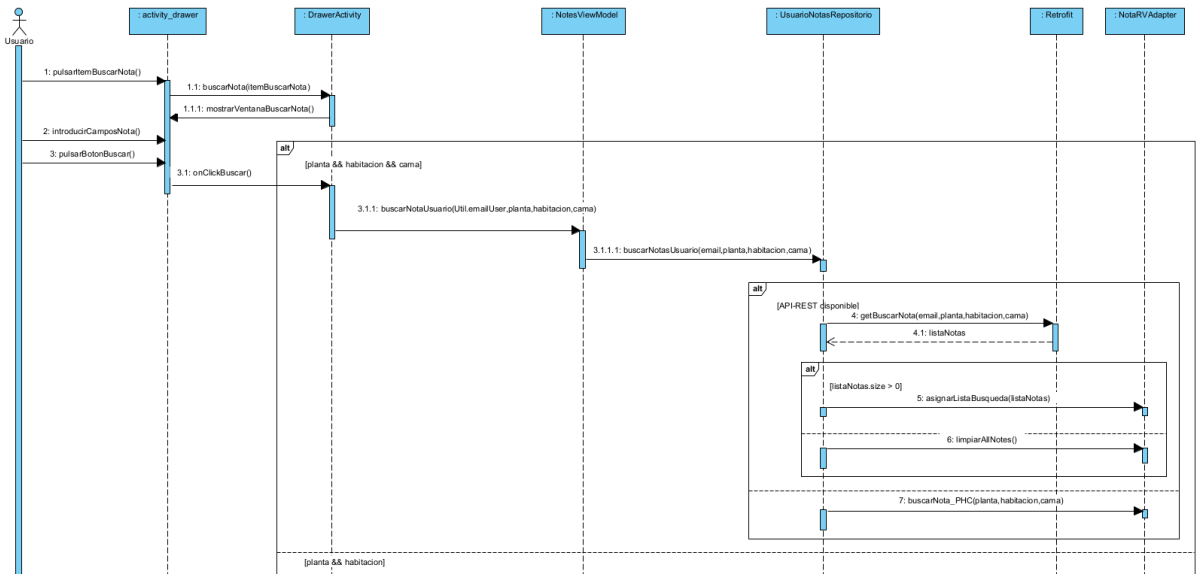


Ilustración 42: diagrama de secuencia de buscar notas mediante un formulario - parte 1.

En la imagen superior, se muestra la primera parte del diagrama de secuencia de buscar una nota mediante un formulario.

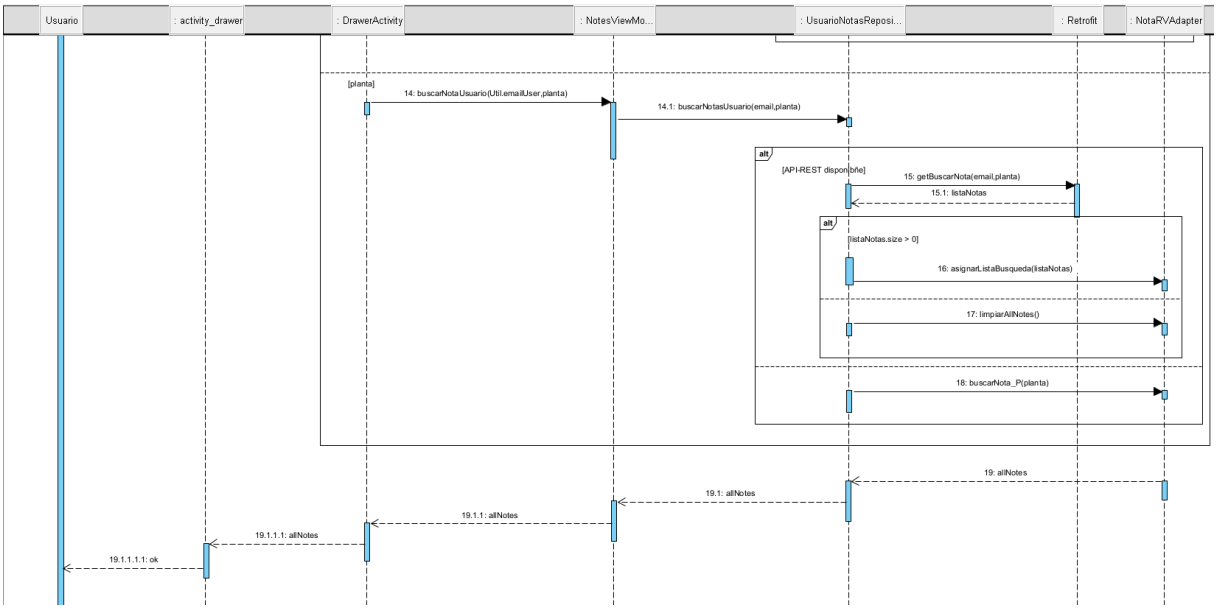


Ilustración 43: diagrama de secuencia de buscar notas mediante un formulario - parte 2.

En la imagen superior, se muestra la parte del diagrama de secuencia de buscar una nota mediante un formulario, donde solo se ha introducido la planta.

Respecto a la interacción de esta parte del diagrama, lo primero que se hace es que el usuario seleccione el ítem de buscar una nota. Cuando se selecciona se mostrará una ventana emergente con un formulario. El usuario podrá rellenar los campos que desee, con la única condición de que el campo de la planta es obligatorio. Una vez relleno el formulario, el usuario pulsa el botón de buscar.

A continuación, en función de los datos rellenados podrán suceder 3 posibles escenarios. No obstante, la lógica de los escenarios es la misma y lo único que cambia son los parámetros que se les pasa a las funciones sobrecargadas.

Por tanto, una vez que se pulsa el botón de buscar, se realizará una llamada a la clase *NotesViewModel* para que comunique al repositorio correspondiente que debe realizar una llamada a la API-REST.

Si la API-REST está disponible, se realizará la llamada y se obtendrá una lista de notas que contienen los datos suministrados y se actualizará la lista de notas que se le muestra al usuario.

En caso de que no esté disponible el servidor, se buscará las notas de la base de datos local, para ello se llamará a la clase del adaptador de la lista para que filtre las notas actuales por los campos suministrados.

Los otros 2 escenarios son idénticos al primero, solamente cambia el número de parámetros rellenados en el formulario.

3.4.8 Ordenar lista

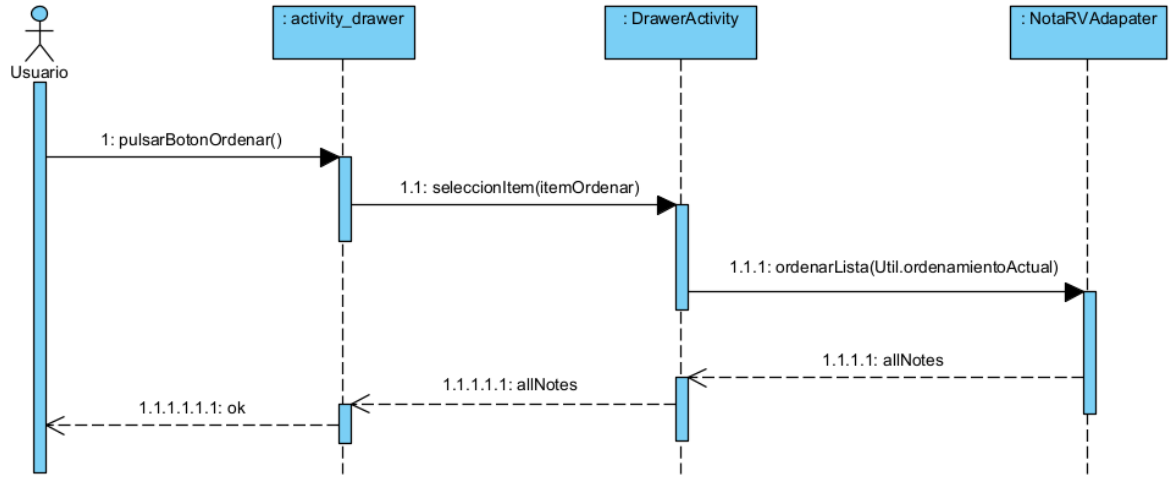


Ilustración 44: diagrama de secuencia de ordenar lista.

En la imagen superior se muestra el diagrama de secuencia del caso de uso de ordenar la lista de notas.

El usuario pulsa el ítem correspondiente a ordenar la lista según el tipo de ordenamiento que desee. Una vez pulsado el ítem, y sabiendo que tipo de ordenamiento quiere, se realizará una llamada a la clase *NotaRVAdapter* para ordenar la lista que se muestra en la vista principal.

3.4.9 Refrescar Lista

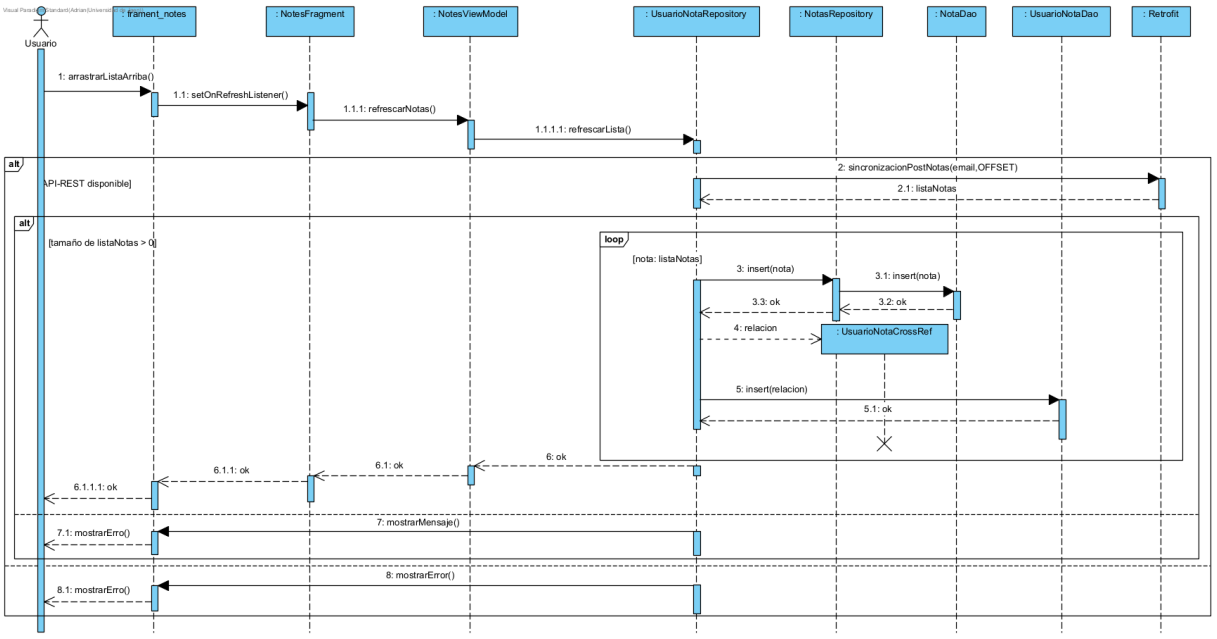


Ilustración 45: diagrama de secuencia de refrescar lista.

En la imagen superior se muestra el diagrama de secuencia del caso de uso de refrescar una lista.

El usuario arrastra el dedo desde el fondo de la pantalla hacia arriba para activar esta funcionalidad, una vez hecho esto se llamará al repositorio correspondiente para realizar la llamada a la API-REST y obtener nuevas notas.

Una vez que se ha recibido la lista de notas de la API-REST, añadimos estas nuevas notas a la base de datos local y actualizamos la vista principal del usuario.

Por otro lado, en caso de que no esté la API-REST disponible o no devuelva ninguna nota nueva se mostrarán los mensajes correspondientes para informar al usuario de lo sucedido.

3.5 Diseño de la interfaz

En este último apartado del capítulo de diseño, vamos a mostrar un mockup para poder visualizar todas las vistas que va a tener nuestro prototipo de aplicación móvil.

Las vistas diseñadas para el prototipo de aplicación móvil intentan ser lo más sencillas e intuitivas posibles para mejorar la usabilidad del prototipo.

3.5.1 Vista de la pantalla de carga de la aplicación

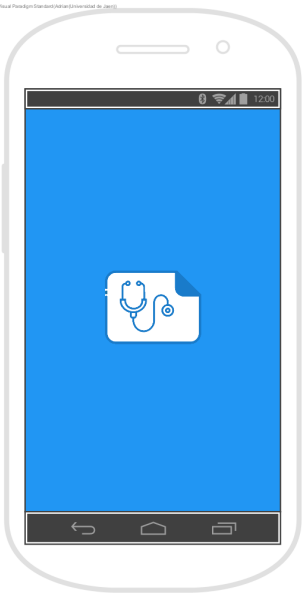


Ilustración 46: diseño de pantalla de carga.

Esta será la vista que se muestre siempre que se inicie la aplicación, es decir es la pantalla de carga de nuestra aplicación.

3.5.2 Vista de inicio de sesión

Ilustración 47: diseño de la vista de iniciar sesión.

Esta será la vista de inicio de sesión del usuario, como podemos ver aparece el logo de la aplicación, el nombre de la aplicación y el botón para iniciar sesión con Google o Facebook.

3.5.3 Vista del menú desplegable

Ilustración 48: diseño del menú desplegable lateral.

Como podemos observar en la vista de arriba, se trata de un menú bastante sencillo en el que hay una cabecera donde aparecen los datos del usuario. Debajo están los items para realizar distintas acciones.

3.5.4 Vistas de la pantalla principal

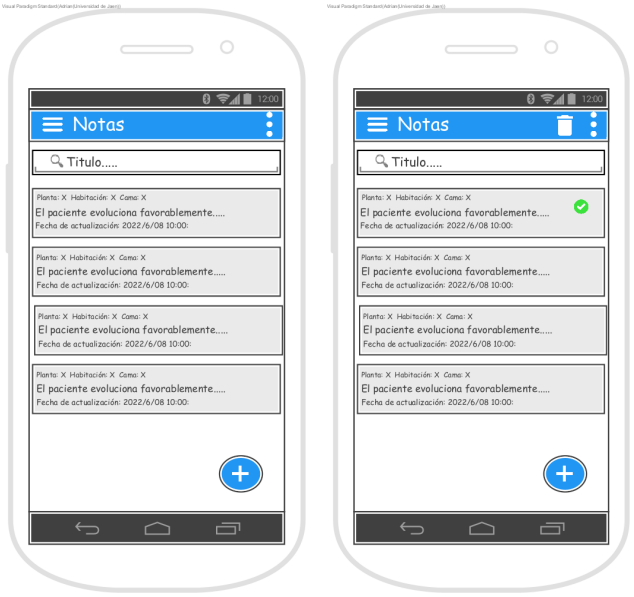


Ilustración 49:diseño de la vista principal.

Como podemos observar en la imagen de arriba, la vista de la pantalla principal está formada en su mayoría por la lista de notas, también aparece una barra de búsqueda encima de la lista y un botón en la esquina inferior derecha para añadir las notas.

Por otro lado, cuando se selecciona una nota para eliminarla aparece un icono de selección a la derecha de la nota que se ha seleccionado y aparece un icono de una papelera en la barra de configuración.

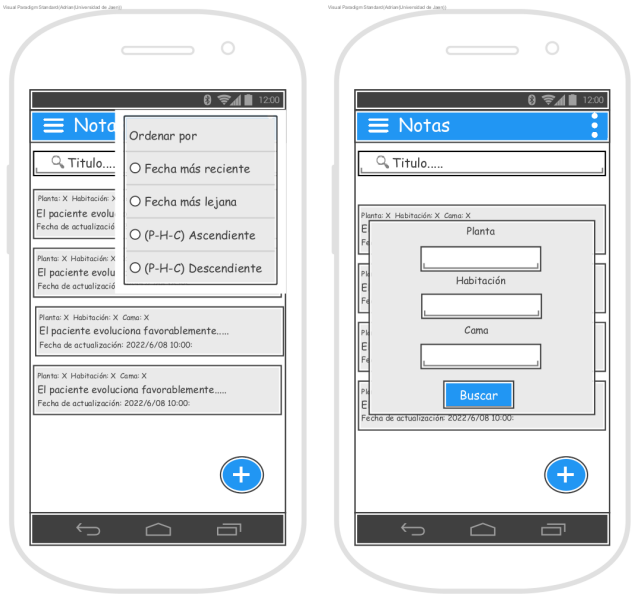


Ilustración 50: diseño de las opciones del menú de la vista principal.

Si se pulsa el menú de la barra de configuración aparecerán 2 opciones, la de ordenar la lista y la de buscar notas.

La imagen de arriba representa la forma en la que se tendrán que ver estas opciones cuando se seleccionen.

3.5.5 Vista de visualización de una nota

Ilustración 51: diseño de la vista de visualización de la nota.

Esta será la vista de visualización de cualquier nota. Como podemos ver, en la ventana de visualización aparecen todos los campos que tiene una nota. No

obstante, dependiendo del dispositivo utilizado, tal vez no sea posible visualizar todos los campos a la vez por lo que esta vista se podrá deslizar hacia abajo.

Al pulsar la opción de compartir la nota, aparecerá una ventana emergente que nos permitirá añadir el correo electrónico del usuario al que se la vamos a compartir, además nos mostrará una lista de los usuarios que tienen acceso a la nota.

3.5.6 Vistas de añadir una nota

Ilustración 52: diseño de las vistas para añadir/editar una nota.

Estas serán las vistas para añadir o editar una nota. En la imagen de la izquierda, podemos ver la vista en la que se introduce la planta, la habitación y la cama del paciente. Mientras que en la imagen de la derecha, podemos ver la vista en la que se introducen los demás campos de la nota.

3.5.7 Vista de configuración

Ilustración 53: diseño de la vista de configuración.

Estas serán las vistas para configurar el prototipo de aplicación móvil y obtener información sobre él.

En la opción de Idioma, podremos cambiar el idioma mediante un menú desplegable. En la opción de tema oscuro, habrá un botón de activación para cambiar a tema oscuro. Por último, al seleccionar la opción de Acerca de la aplicación, aparecerá una ventana emergente con información de la aplicación.

3.6 Plan de pruebas

Dado que estamos utilizando la metodología SCRUM, al realizar una historia de usuario deberemos indicar también cuáles son los criterios de aceptación. Los criterios de aceptación son breves descripciones sobre las expectativas de los usuarios al realizar las acciones que ofrece el prototipo. Las pruebas a realizar deberán cumplir estos criterios para que la prueba se considere exitosa.

Para el plan de pruebas nos centraremos en el aspecto y la interacción con la interfaz de usuario. Los criterios de aceptación para cada historia de usuario son:

Id	Título	Criterios de aceptación
1	Iniciar Sesión	1. Al iniciar sesión se redirigirá al usuario a la vista principal. 2. El usuario deberá poder ver el nombre, el correo electrónico

		y su imágen en el menú desplegable.
2	Cerrar sesión	1. Al cerrar sesión se redirigirá al usuario a la vista de inicio de sesión. 2. Una vez cerrada la sesión, si el usuario abre la aplicación se redirigirá a la ventana de inicio de sesión.
3	Añadir una nota	1. Al añadir una nota, la nota deberá de aparecer en la lista de notas de la vista principal.
4	Editar una nota	1. Al editar una nota, la nota editada deberá de aparecer en la lista de notas de la vista principal.
5	Eliminar una nota	1. Al eliminar una nota, la nota desaparecerá de la lista de notas.
6	Compartir una nota	1. Al compartir una nota con éxito aparecerá un mensaje en la parte baja de la vista de visualizar nota,que no indique que se ha compartido correctamente. 2. Si no pudiese compartir la nota, aparecerá un mensaje en la parte baja de la vista de visualizar nota. 3. Si no hubiese una conexión a internet, aparecerá un mensaje en el que se informe del problema.
7	Ver una nota	1. Al seleccionar una nota nos aparecerá en la vista de visualizar nota, todos los datos de esa nota.
8	Visualizar una lista con las notas	1. En la pantalla principal podremos ver todas las notas que hay en la base de datos local. 2. Para cada ítem podremos ver los campos de planta, habitación, cama, fecha de actualización y la evolución del paciente.
9	Ordenar la lista	1. Al seleccionar cualquier modo de ordenamiento, la lista que se muestra en pantalla se cambiará el orden de las notas sin que fuese necesario recrear la actividad.
10	Buscar notas	1. Al buscar una nota se mostrará en la lista las notas coincidentes.
11	Cambiar el idioma	1. Al cambiar el idioma en la ventana de configuración se cambiará el idioma en toda la aplicación. 2. Cada vez que se ejecute la aplicación, el idioma de la aplicación será el último seleccionado.
12	Cambiar a modo Oscuro	1. Al cambiar a modo oscuro se cambiará todos los temas de todas las vistas de la aplicación. 2. Cada vez que se ejecute la aplicación, el tema de la aplicación será el último
13	Trabajar sin conexión	1. El usuario podrá utilizar la aplicación aunque no tenga una conexión a internet, siempre y cuando haya iniciado sesión anteriormente.
14	Sincronizar con el servidor	1. En caso de que el cliente no esté sincronizado con el servidor, aparecerá un mensaje para que sincronice la aplicación. 2. En caso de no poder sincronizar con el servidor, aparecerá un mensaje en la parte baja de la vista principal indicando que no se ha podido sincronizar.

Tabla 12: plan de pruebas a realizar.

Por otro lado, se realizarán pruebas en local para probar el funcionamiento del backend antes de realizar el despliegue. Estas pruebas serán comentadas en el apartado 5.

4. Implementación

Tras realizar la etapa de diseño y haber mostrado los diferentes diagramas y la maqueta para mostrar el modelo y las especificaciones del producto, procederemos a explicar la forma en la que he implementado el prototipo de la aplicación.

4.1 Diagrama arquitectónico

El patrón arquitectónico utilizado para la implementación del prototipo de la aplicación móvil, es el patrón MVVM el cual es bastante similar al patrón tradicional MVC. El patrón MVVM se utiliza para separar la parte de las vistas con la parte del modelo, en este patrón se sustituye el elemento de Controller por el elemento ViewModel el cual será el intermediario entre las vistas y el modelo.

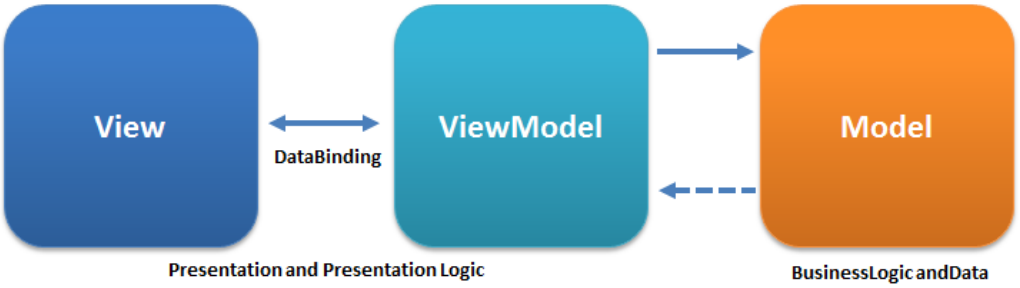


Ilustración 54: patrón MVVM.

Los elementos de este patrón son:

- **View**: Vista de la actividad.
- **ViewModel**: Intermediario entre la vista y el modelo.
- **Model**: Modelo de la aplicación el cual contiene la lógica de la aplicación.

El patrón MVVM nos proporciona varias ventajas en el desarrollo de aplicaciones móviles frente al patrón MVC, algunas ventajas son:

- Permite acceder a las vistas mediante un objeto llamado binding.
- El acceso a la información y actualización de la interfaz de usuario se realiza de forma automática.
- La implementación del componente de ViewModel es mucho más fácil que la creación del controlador tradicional en MVC.

4.2 Detalles de implementación

4..2.1 Backend

Para la implementación del backend hemos utilizado el entorno de ejecución de node.js [15] y el gestor de paquetes npm.

4.2.1.1 Fichero package.json

El fichero package.json se genera de forma automática al inicializar un proyecto con el gestor de paquetes npm, este fichero contiene información de la aplicación.

Ilustración 55: contenido del fichero package.json.

Como podemos ver en la imagen del contenido del fichero package.json, nos ofrece información sobre el proyecto, como el nombre o una breve descripción, pero la información más relevante que nos aporta son los módulos(dependencies) que utiliza el proyecto, los scripts declarados en el proyecto y el fichero inicial de la aplicación cuando se ejecuta. Respecto a los módulos que utilizamos en nuestro proyecto:

- **express**:framework de node.js [16] que nos permite gestionar las rutas y recursos de nuestra aplicación así como las peticiones de entrada y sus respuestas.
- **dotenv**: es un módulo que nos permite configurar las variables de entorno de la aplicación.

- **express-handlebars**: módulo que nos permite introducir la sintaxis de handlebars en el proyecto. Este módulo se utiliza en la implementación de la página web.
- **handlebars**: módulo que nos permite utilizar plantillas de handlebars para crear vistas de la página web utilizando handlebars.
- **firebase**: módulo que utilizamos para autentificar a un usuario de la aplicación.
- **mysql**: módulo para poder acceder a una base de datos que ha sido creada con MySQL.
- **body-parser**: módulo que nos permite recibir JSON en los cuerpos de las peticiones POST y PUT. Además nos permite enviar a los clientes información con el formato JSON.
- **nodemon**: nodemon es un módulo que nos permite reiniciar la aplicación cada vez que se realicen cambios en ella y se guarden. Este módulo ha sido utilizado a la hora del desarrollo del backend, ya que nos facilitaba bastante ver los cambios realizados, no obstante en el despliegue de la aplicación este módulo no se utiliza.

Por otro lado, podemos ver como se ha definido el script de server.js el cual tiene toda la inicialización del backend. A su vez, se ha seleccionado este fichero javascript como el inicial de la aplicación.

4.2.1.2 Creación de la base de datos con MySQL

Para la creación de la base de datos con MySQL hemos creado un addon en la plataforma Clever Cloud e indicado que utilizará el gestor de base de datos MySQL.

Una vez creada la base de datos, la plataforma nos proporciona todas las variables de entorno que necesitaremos para poder acceder a la base de datos. Los datos más importantes son el host, el nombre de la base de datos, el usuario que tiene acceso a la base de datos y la contraseña de este.

Una vez obtenidos los campos, nos conectamos a la base de datos a través de la aplicación de DBeaver para poder administrar la base de datos y crear las tablas definidas en la sección de diseño.

4.2.1.3 Inicialización del backend

El primer paso consiste en declarar las variables de entorno que utilizamos en el proyecto. Para ello se creó un fichero llamado .env-local en la raíz del proyecto. Las variables de entorno que se utilizan en el backend son:

- **PORT**: Puerto donde se escuchan las peticiones que llegan al backend. Esta variable local es utilizada durante el desarrollo del proyecto de forma local. El servidor, Clever Cloud, proporciona a la base de datos un puerto por defecto que no se puede cambiar, así que dejará de tener valor cuando se realice el despliegue final
- **DB_HOST**: Ubicación del servidor donde se aloja nuestro backend.
- **DB_USER**: Usuario de la base de datos.
- **DB_PASS**: Contraseña del usuario que tiene acceso a la base de datos.
- **DB_NAME**: Nombre de la base de datos.

Una vez declaradas las variables de entorno, se implementó el fichero database.js, que se encarga de gestionar la conexión con la base de datos.

src > helpers > JS database.js > ...

```
const mysql = require('mysql'); // Requerimos el módulo de mysql

// Creamos la conexión, es similar al createConecction
const pool = mysql.createPool({
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASS,
    database: process.env.DB_NAME,
    connectionLimit: 5
});



// Conectamos con la base de datos y comprobamos si hay errores
pool.getConnection((err, connection) => {
    if(err){
        if (err.code === 'PROTOCOL_CONNECTION_LOST'){
            console.error('Database connection lost');
        }
        if (err.code === 'ER_CON_COUNT_ERROR'){
            console.error('Database has too many connection');
        }
        if (err.code === 'ECONNREFUSED'){
            console.error('Database connection was refused');
        }
    }
    console.log("conexion exitosa")
    if(connection) connection.release();

    return;
});

module.exports = pool; // Exportamos el módulo para utilizar la base de datos

```

Ilustración 56: contenido del fichero database.js.

En este fichero, lo primero que hago es crear una conexión con la base de datos del servidor utilizando las variables de entorno declaradas anteriormente e indico el valor 5 como el número máximo de conexiones que se pueden crear para acceder a la base de datos, debido al tipo de servicio contratado en la plataforma Heroku que solo nos permite tener 5 conexiones conectadas a la vez a la API-REST.

Posteriormente declaramos la función de getConnection para poder obtener una conexión disponible. En caso de que no haya una conexión disponible, el dispositivo que ha realizado la petición no podrá utilizar la API-REST.

Una vez implementado el código necesario para la conexión con la base de datos, procedemos a implementar el fichero que se encarga de la inicialización de toda la aplicación, este fichero es server.js.

Lo primero de todo es declarar los módulos que necesitamos, posteriormente creamos una aplicación con express y la inicializamos para que acepte peticiones con un cuerpo del mensaje en formato JSON.

```
src > JS server.js > ...
1   const path = require('path');
2   const express = require('express'); // Módulo para crear una conexión
3   const dotenv = require('dotenv');
4   // Requerimos los módulos necesarios para crear la página
5   const exphbs = require('express-handlebars');
6   const bodyParser = require('body-parser');
7   dotenv.config({path: '.env-local'});
8
9   const PORT = process.env.PORT || '3001';
10
11  const app = express();
12
13  /**
14   * Middleware
15   */
16  app.use(express.json());
17  app.use(express.urlencoded({extended:false}));
18  app.use(bodyParser.urlencoded({
19      extended: true
20    }));
21  app.use(bodyParser.json());
22
```

Ilustración 57: inicialización de la API-REST.

Una vez creada la aplicación inicializamos las rutas accesibles a la aplicación. Estas rutas están definidas en otros ficheros de JavaScript, los cuales serán explicados en el siguiente apartado, por último indicamos que la aplicación está lista para recibir peticiones [17].

```
46
47  const userRouter = require('./routes/user');
48  const notasRouter = require('./routes/note')
49  const userNoteRouter = require('./routes/user_note')
50  const clienteWebRouter = require('./routes/clienteWeb')
51
52  app.use('/user',userRouter);
53  app.use('/note',notasRouter);
54  app.use('/userNote',userNoteRouter)
55  app.use('/notas',clienteWebRouter)
56
57  // Servidor escuchando
58  app.listen(PORT, () => {
59      console.log(`Aplicación escuchando en el puerto: ${PORT}`)
60  })
```

Ilustración 58: declaración de las rutas de la API-REST.

4.2.1.4 Rutas de la API de REST

El fichero note.js contiene las rutas sobre los recursos de la entidad nota.

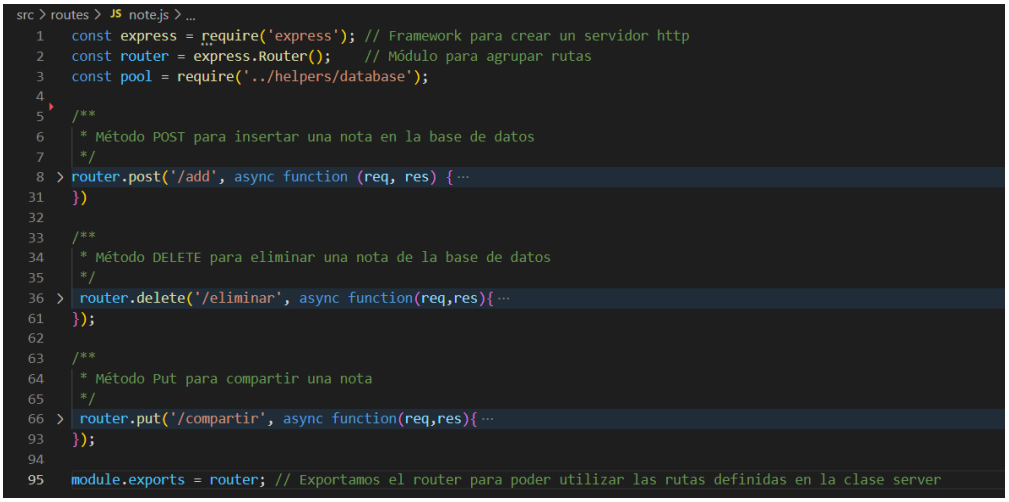


Ilustración 59: contenidos del fichero note.js.

Como se puede apreciar en la imagen superior, se han declarado 3 rutas para poder acceder a las notas. Estas rutas y métodos son:

Método	Ruta	Descripción
POST	/note/add	Ruta para insertar una nota en la base de datos.
DELETE	/note/eliminar	Ruta para eliminar una nota de la base de datos.
PUT	/note/compartir	Ruta para indicar que una nota es compartida.

Tabla 13: rutas declaradas en el fichero note.js.

El fichero user.js contiene las rutas del recurso de la entidad de Usuario.

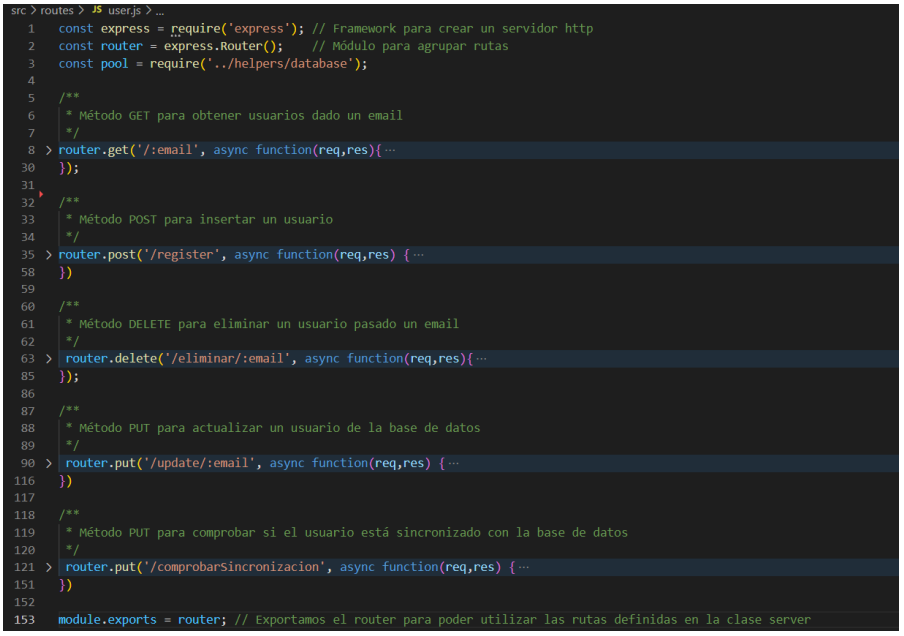


Ilustración 60: contenido del fichero user.js.

En la imagen superior, se puede observar cómo se han definido 5 rutas para poder acceder al recurso de los usuarios. Estas rutas y métodos son:

Método	Ruta	Descripción
GET	/user/email	Ruta para obtener el usuario con el email pasado como parámetro.
POST	/user/register	Ruta para añadir un usuario a la base de datos.
DELETE	/user/eliminar/email	Ruta para eliminar el usuario con el email pasado como parámetro.
PUT	/user/update/email	Ruta para actualizar la fecha de actualización del usuario pasado como parámetro.
PUT	/user/comprobarSincronizacion	Ruta para comprobar si un usuario está sincronizado con el servidor.

Tabla 14: rutas declaradas en el fichero user.js.

El fichero user_note.js contiene las rutas del recurso de la entidad de UsuarioNotaCrossRef.

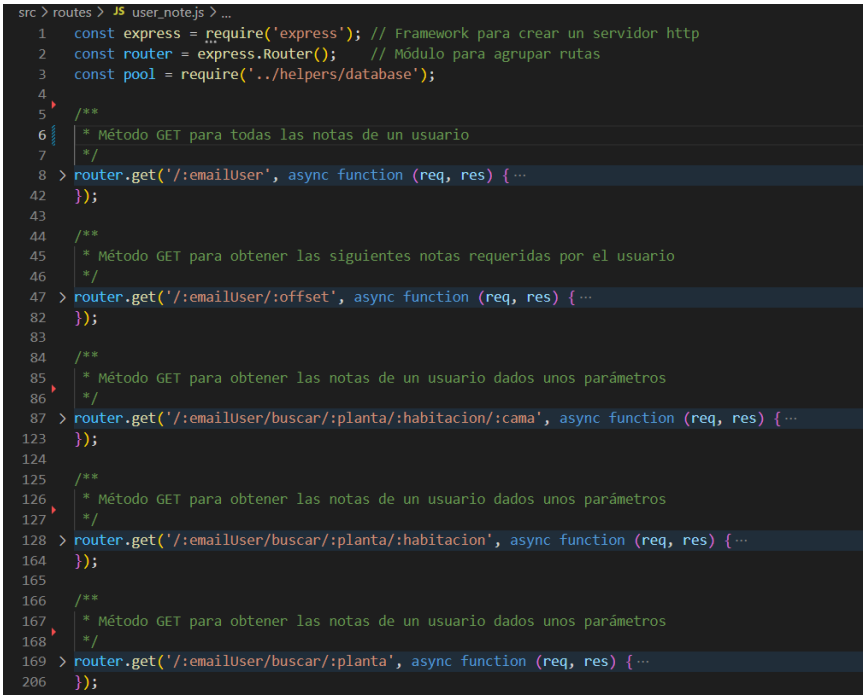


Ilustración 61: contenido del fichero user_note.js - parte 1.

```
208  /**
209   * Método POST para insertar una nueva relación
210   */
211  router.post('/add', async function (req, res) {···
234  })
235
236  /**
237   * Método DELETE para eliminar una relación en la base de datos
238   */
239  router.delete('/eliminar', async function (req, res) {···
264  });
265
266  /**
267   * Método PUT para obtener los usuarios compartidos
268   */
269  router.put('/usuariosCompartidos', async function(req,res){···
310  });
311
312  module.exports = router; // Exportamos el router para poder utilizar las rutas definidas en la clase server
```

Ilustración 62: contenido del fichero user_note.js - parte 2.

En las ilustraciones 61 y 62, muestran la declaración de 8 rutas accesibles para acceder a la relación entre los usuarios y las notas. Estas rutas y métodos son:

Método	Ruta	Descripción
GET	/userNote/emailUser	Ruta para obtener todas las notas a las que tiene acceso un usuario del email pasado como parámetro.
GET	/userNote/emailUser/offset	Ruta para obtener un número determinado de notas que tiene acceso un usuario,dependiendo del email.pasado como parámetro y el valor de offset pasado.
GET	/userNote/emailUser/buscar/planta/habitacion/cama	Ruta para buscar notas a las que tiene acceso un usuario según los parámetros pasados.
GET	/userNote/emailUser/buscar/planta/habitacion	Ruta para buscar notas a las que tiene acceso un usuario según los parámetros pasados.
GET	/userNote/emailUser/buscar/planta	Ruta para buscar notas a las que tiene acceso un usuario según los parámetros pasados.
POST	/userNote/add	Ruta para añadir una nueva relación usuario - nota a la base de datos.
DELETE	/userNote/eliminar	Ruta para eliminar una relación usuario - nota de la base de datos.
PUT	/userNote/usuariosCompartidos	Ruta para obtener todos los usuarios que tienen acceso a una cierta nota.

Tabla 15: rutas declaradas en el fichero user_note.js.

Por otro lado, las peticiones que se realizan a la base de datos son mediante la sintaxis de SQL, la mayoría de nuestras servicios realizan operaciones bastantes sencillas, no obstante en las operaciones que intervienen 2 tablas como son las

búsquedas de notas, las consultas son bastante largas por lo que cualquier error de sintaxis es una excepción MySQL.

```
/**
 * Método GET para obtener las siguientes notas requeridas por el usuario
 */
router.get('/:emailUser/:offset', async function (req, res) {
    try {
        // Obtenemos los parámteros del body
        const sqlQuery =
            'SELECT n.planta, n.habitacion, n.cama, n.fechaActualizacion, n.observaciones1, n.observaciones2, ' +
            'n.ct_frecuenciaCardiaca, n.ct_frecuenciaRespiratoria, n.ct_temperatura, n.ct_presionArterial, n.ct_presionArterial2, n.
            'FROM Nota n, UsuarioNotaCrossRef u_n ' +
            'WHERE u_n.email = ? ' +
            'AND u_n.planta = n.planta ' +
            'AND u_n.habitacion = n.habitacion ' +
            'AND u_n.cama = n.cama ' +
            'AND u_n.fechaActualizacion = n.fechaActualizacion ' +
            'AND u_n.emailCreado = n.emailCreado ' +
            'ORDER BY n.fechaActualizacion DESC ' +
            'LIMIT 10 OFFSET ?';

        pool.query(sqlQuery, [req.params.emailUser, parseInt(req.params.offset)], function (err, result, fields) {
            if (err)
                throw err;
            // Comprobamos si el resultado está vacio o no
            if (Object.keys(result).length === 0) {
                res.status(404).send('Recurso no encontrado')
            } else {
                res.status(200).json(result);
            }
        });
    } catch (error) {
        res.status(400).send(error.message);
    }
});
```

Ilustración 63: ejemplo de llamada a la base de datos con express.

4.2.2 Frontend

4.2.2.1 Creación del proyecto en Firebase

Para permitir la autenticación mediante Google, se ha creado un proyecto en Firebase [18]. Para ello nos vamos a la consola de Firebase y creamos un proyecto nuevo.

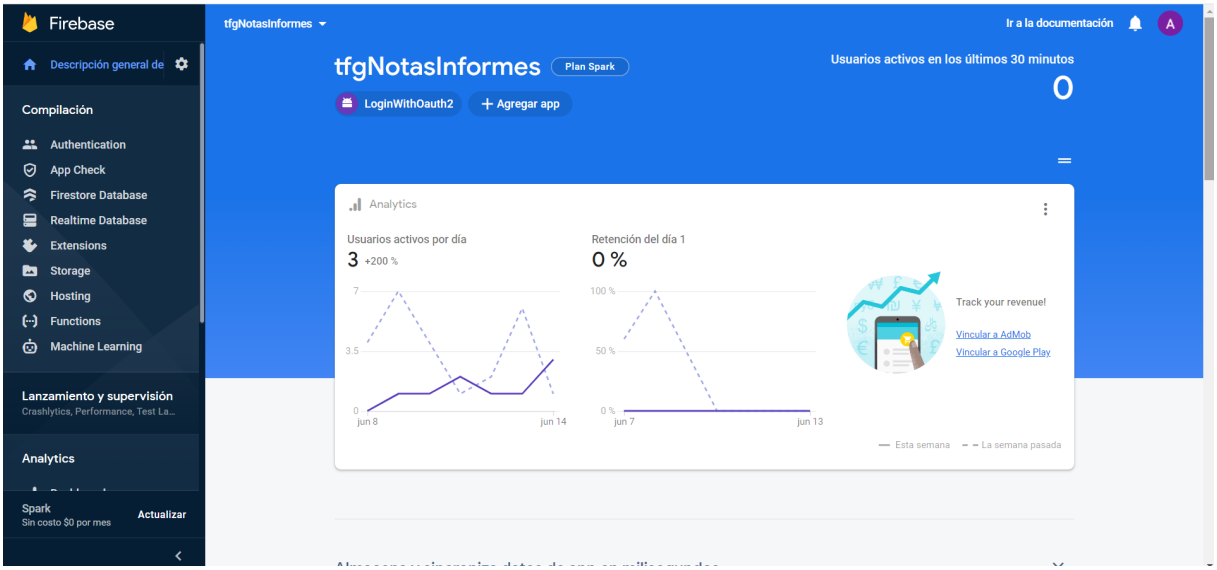


Ilustración 64: creación de un proyecto en firebase.

Una vez creado, hay que habilitar los proveedores con los que podremos autenticar a los usuarios. Como podemos ver en la imagen inferior, se han habilitado

los proveedores de Google y Facebook. No obstante, la autenticación por Facebook no ha sido posible implementarla ya que es necesario subir el prototipo a la plataforma Play Store.

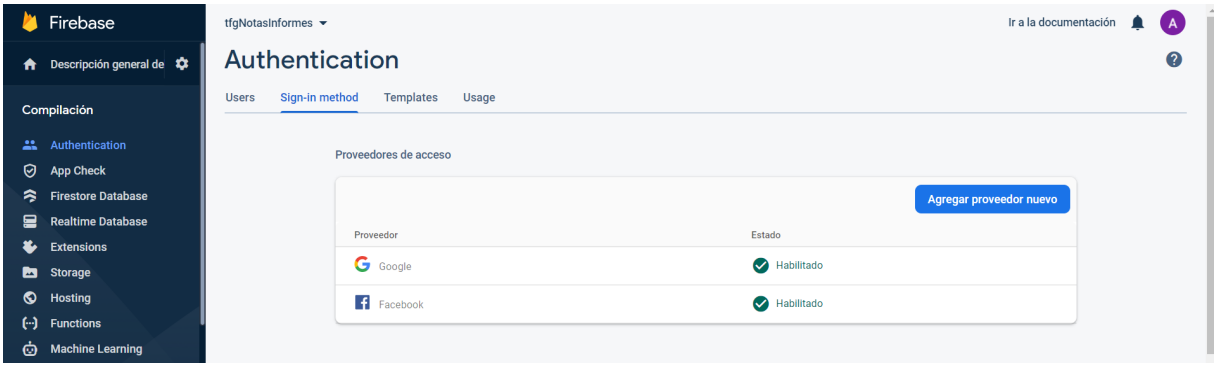


Ilustración 65: habilitación de los proveedores en firebase.

Una vez configurado el proyecto, hay que añadir el fichero google-service.json a la aplicación para habilitar el servicio de autenticación de Google.

4.2.2.2 Clase Util

Antes de comenzar a explicar cómo se han implementado las diferentes partes del frontend, es importante indicar que se ha creado la clase Util,que contiene atributos públicos y estáticos auxiliares necesarios para el desarrollo de la aplicación.

Gracias a esta clase se puede obtener en cualquier momento atributos compartidos por todas las clases como, por ejemplo, el tipo de ordenamiento que se está realizando, el usuario que ha iniciado sesión y muchos atributos más que nos facilitan la lógica de nuestro prototipo. Además, en esta clase también se declaran las instancias de NotesViewModel y de Drawer con el objetivo de poder acceder a su contexto y modificar la vista principal para mostrar mensajes en ella.

4.2.2.3 Implementación de la base de datos local

Para la implementación de la base de datos local [19] hemos utilizado la biblioteca Room, la cual ofrece una capa de abstracción para SQLite con la que puedo gestionar la base de datos.

Para utilizar la biblioteca Room basta con añadir las dependencias correspondientes en el fichero build.gradle del proyecto Kotlin. Los elementos principales de la base de datos son:

- Las entidades que se almacenarán en la base de datos.

- Los DAOs, es decir los objetos que utilizamos para acceder a la base de datos.
- Los repositorios, los cuales nos ofrecen una capa de abstracción adicional para poder acceder a los diferentes DAOs de nuestra aplicación. Además en este caso, desde los repositorios se hacen las llamadas a la API-REST.

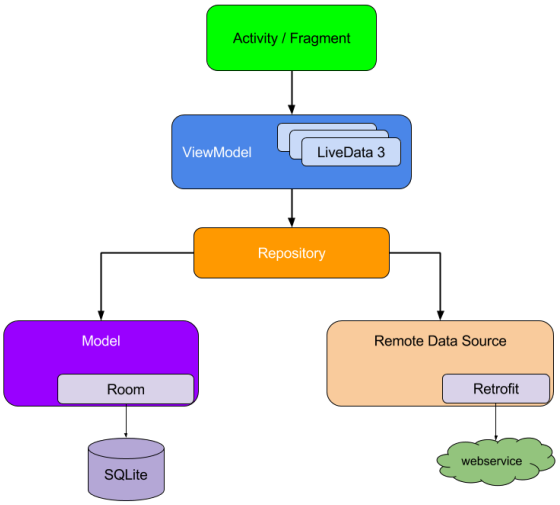


Ilustración 66: esquema del cliente móvil.

Para crear una entidad en Kotlin basta con hacer una clase del tipo data, cuya clase solo almacena datos y no tiene funciones, añadir la anotación *@Entity* a la clase y definir la clave primaria.

```kotlin
/**
 * Clase que representa a la entidad Usuario
 *
 * @param nombre Nombre del usuario
 * @param email Correo electrónico del Usurio
 * @param fechaActualizado Última fecha en la que se ha sincronizado con el servidor
 */
@Entity
data class Usuario(
    val nombre: String,
    @PrimaryKey val email: String,
    var fechaActualizado: String
)
```

Ilustración 67: declaración de una entidad en Kotlin.

Como podemos ver en la imagen de arriba, se ha implementado la entidad Usuario de una forma bastante sencilla gracias a las anotaciones que nos proporciona la biblioteca Room.

Por otro lado, si la entidad tiene una clave primaria compuesta y tiene claves foráneas, se declararán como parámetros de la anotación *@Entity*.

```kotlin
@Entity(
    primaryKeys = ["emailUser","planta","habitacion","cama","fechaActualizacion","emailUserCreado"],
    foreignKeys = [
        ForeignKey(
            entity = Usuario::class,
            parentColumns = ["email"],
            childColumns = ["emailUser"],
            onDelete = CASCADE
        ),
        ForeignKey(
            entity = Nota::class,
            parentColumns = ["planta","habitacion","cama","fechaActualizacion","emailCreado"],
            childColumns = ["planta","habitacion","cama","fechaActualizacion","emailUserCreado"],
            onDelete = CASCADE
        )
    ]
)
data class UsuarioNotaCrossRef(
    val emailUser: String,
    val planta: Int,
    val habitacion: Int,
    val cama: Int,
    var fechaActualizacion: String,
    var emailUserCreado: String
```

Ilustración 68: declaración de claves primarias y foráneas en Kotlin.

Como se puede observar en la imagen de arriba se representa la entidad *UsuarioNotaCrossRef* la cual tiene una clave primaria compuesta y 2 claves foráneas.

Las entidades de *Nota*, *NotaAdd* y *NotaEliminada* se implementan de la misma forma que las anteriores.

Por otro lado, en esta parte de la implementación han surgido varias alternativas a la hora de implementar las entidades, ya que la biblioteca Room, nos permite utilizar embeber objetos dentro de una entidad mediante la etiqueta *@Embedded,* no obstante no se ha utilizado esta opción ya que surgieron varios problemas al realizar las llamadas a la API-REST, ya que le pasábamos como cuerpo del mensaje una entidad con un objeto embebido.

Una vez que se ha implementado las entidades, se procederá a implementar la base de datos, la forma de implementar una base de datos local con Room es crear una clase abstracta que implementa la funcionalidad de la clase *RoomDatabase* e indicar que se trata de una base de datos mediante la anotación *@Database*.

```kotlin
@Database(
    entities = [Nota::class, Usuario::class, UsuarioNotaCrossRef::class, NotaEliminada::class, NotaAdd::class],
    version = 1,
    exportSchema = false
)
abstract class AppDatabase : RoomDatabase() {
    /** Función para obtener el DAO de las notas ...*/
    abstract fun getNotesDao(): NotasDao

    /** Función para obtener el DAO de los usuarios ...*/
    abstract fun getUsuarioDao(): UsuarioDao

    /** Función para obtener el DAO de las relaciones ...*/
    abstract fun getUsuarioNotasDao(): UsuarioNotaCrossRefDao

    /** Función para obtener el DAO de las notas a eliminar ...*/
    abstract fun getNotasEliminadasDao(): NotaEliminadaDao

    /** Función para obtener el DAO de las notas a añadir ...*/
    abstract fun getNotasAddDao(): NotaAddDao
```

Ilustración 69: declaración de la base de datos en Kotlin.

Como podemos ver en la imagen de arriba se indican cuales son las entidades que almacenará nuestra base de datos dentro de la directiva *@Database*. Por otro lado, la clase *AppDatabase* es la encargada de crear los DAOs de todas las entidades.

Por último, para hacer que solo se pueda crear una instancia de la clase, definiremos una función para obtener la instancia de la base usando el patrón Singleton.

```kotlin
        @Volatile
        private var INSTANCE: AppDatabase? = null

        /**
         * Función para obtener la base de datos local
         */
        fun getDatabase(context: Context): AppDatabase {
            // if the INSTANCE is not null, then return it,
            // if it is, then create the database
            return INSTANCE ?: synchronized( lock: this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                     name: "notas_database"
                ).build()
                INSTANCE = instance
                // return instance
                instance  ^synchronized
            }
        }
    }
}
```

Ilustración 70: utilización del patrón Singleton.

Como se muestra en la ilustración superior, definimos una función estática llamada getDatabase que devuelve la instancia de la base de datos local. Para forzar que solo pueda existir una única instancia utilizamos la anotación *@Volatile*.

A continuación, explicaremos cómo se han implementado los DAOs de las entidades.

Para crear un DAO lo único que hay que hacer es crear una interfaz e incluir la anotación *@Dao* de Room. Posteriormente se definen las funciones que tendrá el DAO y le incluimos a cada función la anotación correspondiente en función de lo que haga la función. Las anotaciones más utilizadas son:

- **@Insert:** anotación para indicar que se va a insertar un objeto de una entidad en la base de datos.
- **@Delete**: anotación para indicar que se va a eliminar un objeto de una entidad en la base de datos.
- **@Update**: anotación para indicar que se va a actualizar un objeto de una entidad en la base de datos.
- **@Query**: anotación que permite realizar consultas con la sintaxis de SQL.

```
/**
 * Interfaz que representa al DAO de los Usuarios
 */
@Dao
interface UsuarioDao {

    /** Función para insertar un nuevo Usuario ...*/
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(usuario: Usuario)

    /** Función para actualizar un Usuario en la base de datos local ...*/
    @Update
    suspend fun update(usuario: Usuario)

    /** Función para eliminar un Usuario en la base de datos local ...*/
    @Delete
    suspend fun delete(usuario: Usuario)

    /** Función para obtener un usuario concreto ...*/
    @Query( value: "Select * from Usuario WHERE email =:email")
    fun getUsuario(email: String): Usuario

}
```

Ilustración 71: declaración de un DAO en Kotlin.

Como podemos ver en la imagen superior, se ha implementado el DAO de la entidad Usuario. Para ello se ha definido una interfaz y se le ha incluido la anotación *@Dao*, además a cada función declaradas se le inserta la anotación correspondiente.

Respecto a la anotación *@Query*, se utiliza para definir funciones con la sintaxis SQL. Hay que tener cuidado con las funciones que devuelven objetos ya que en caso de no existir el objeto que se quiere obtener, se devuelve un valor nulo.

Los DAOs de las entidades *Nota*. *UsuarioNotaCrossRef*, *NotaAdd* y *NotaEliminada* se han implementado de la misma manera.

Por último, voy a comentar algunos aspectos sobre la implementación de los repositorios.

Un repositorio es una clase normal de Kotlin y no utiliza ninguna anotación de Room, los repositorios no son obligatorios pero al definirlos nos ofrecen más abstracción y modularidad en la aplicación. La principal ventaja de definir los repositorios en nuestro prototipo es que el repositorio decide cuándo se debe llamar a los DAOs o a la API-REST.

Para definir los repositorios basta con crear clases que tengan como parámetros el DAO de su entidad y los repositorios de las otras entidades que utilice.

```kotlin
/**
 * Repositorio para almacenar las relaciones entre Usuario y Nota
 *
 * @param usuariosNotasDao DAO de la relación
 * @param usuarioRepository Repositorio de las usuarios
 * @param notasRepository Repositorio de las notas
 * @param notasEliminadaRepository Repositorio de las notas a eliminar
 * @param notasAddRepository Repositorio de las notas a añadir
 */
class UsuarioNotaCrossRefRepository(
    private val usuariosNotasDao: UsuarioNotaCrossRefDao,
    private val usuarioRepository: UsuarioRepository,
    private val notasRepository: NotasRepository,
    private val notasEliminadaRepository: NotaEliminadaRepository,
    private val notasAddRepository: NotaAddRepository
) {
```

Ilustración 72: declaración de un repositorio en Kotlin.

Como podemos ver en la imagen superior, el repositorio de *UsuarioNotaCrossRef* se implementa como una clase normal y recibe como parámetros el DAO de su entidad y los repositorios que utiliza. Todos los demás repositorios se implementan de la misma forma.

4.2.2.4 Implementación de las Actividades

Una vez implementada la base de datos local, procedemos a explicar como se ha implementado las actividades del prototipo.

Las actividades son las clases que ejercen de controladores de las vistas de la aplicación. Para crear una vista y su controlador, utilizaremos las actividades por defecto de Android Studio ya que permiten crear una vista y vincularla automáticamente a su controlador. Las actividades implementadas son:

- **AddCamposNota**: controlador de la vista de añadir los campos opcionales de las notas.
- **AddDatosClave**: controlador de la vista de añadir las claves a la nota.
- **VisualizarNota**: controlador de la vista de visualizar una nota.
- **DrawerActivity**: controlador de la vista principal y del menú desplegable.
- **MainActivity**: controlador de la vista del splash screen.
- **SignInActivity**: controlador de la vista de inicio de sesión.

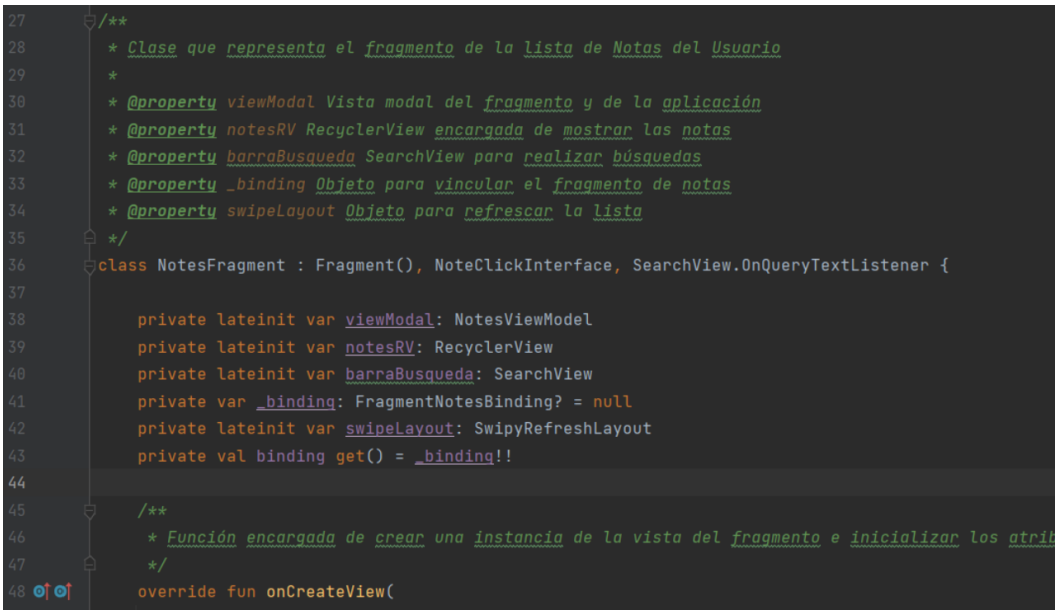
```
20  /**
21   * Clase que se encargará de la lógica y control de la vista de visualizar una nota
22   *
23   * @property binding Objeto para vincular la clase con su vista
24   */
25  class VisualizarNota : AppCompatActivity() {
26      private lateinit var binding: ActivityVisualizarNotaBinding
27
28      /**
29       * Método que se llama al inicializar la actividad para inicializar sus atributos
30       */
31      override fun onCreate(savedInstanceState: Bundle?) {
```

Ilustración 73: declaración de una actividad en Kotlin.

La ilustración 73 hace referencia a la implementación del controlador *VisualizarNota*. Como se puede observar, lo único que hace falta para implementar el controlador es implementar los métodos de la clase *AppComparActivity* y sobrecargar los métodos que se necesiten. En todos los casos, hay que sobrecargar el método de onCreate() para crear el controlador, mostrar la vista e inicializar los atributos si fuese necesario.

Por otro lado, la vista de la lista de notas y la configuración se han implementado como fragmentos, con el objetivo de crear un código más modular y ofrecer flexibilidad en caso que en el futuro se desee añadir nuevas funcionalidades.

Para crear un fragmento basta con hacer que implemente las funcionalidades de la clase Fragment y sobrecargar los métodos necesarios. Además se podrá añadir como atributos el objeto que implemente la vista modal para seguir el patrón MVVM.



```
/**
 * Clase que representa el fragmento de la lista de Notas del Usuario
 *
 * @property viewModal Vista modal del fragmento y de la aplicación
 * @property notesRV RecyclerView encargada de mostrar las notas
 * @property barraBusqueda SearchView para realizar búsquedas
 * @property _binding Objeto para vincular el fragmento de notas
 * @property swipeLayout Objeto para refrescar la lista
 */
class NotesFragment : Fragment(), NoteClickInterface, SearchView.OnQueryTextListener {

    private lateinit var viewModal: NotesViewModel
    private lateinit var notesRV: RecyclerView
    private lateinit var barraBusqueda: SearchView
    private var _binding: FragmentNotesBinding? = null
    private lateinit var swipeLayout: SwipyRefreshLayout
    private val binding get() = _binding!!

    /**
     * Función encargada de crear una instancia de la vista del fragmento e inicializar los atrib
     */
    override fun onCreateView(
```

Ilustración 74: declaración de un fragmento en Kotlin.

Como podemos ver en la imagen anterior, se ha implementado el fragmento *NotesFragment*, el cual contendrá todas las funcionalidades de la vista de la lista de notas.

Una vez que se ha explicado la implementación de los controladores se procederá a explicar la forma de creación de las vistas. Estas vistas están en la ruta res/layout, estas vistas tienen la extensión .xml.

Todas las vistas tienen en ellas componentes de layout [20] para que se adapten a los diferentes tipos de pantalla, los componentes layout utilizados en el prototipo de aplicación móvil son:

- **NestedScrollView**: layout que nos permitirá hacer nuestras vistas desplazables.
- **ConstraintLayout**: layout que nos permite modificar la posición de los elementos que estén dentro de él.
- **LinearLayout**: layout que nos permite representar los elementos de forma secuencial y horizontal.
- **FrameLayout**: layout que nos indica que el contenido que tiene está asociado a un fragmento.
- **AppBarLayout**: layout que nos permite incluir una barra de configuración superior en nuestra vista.

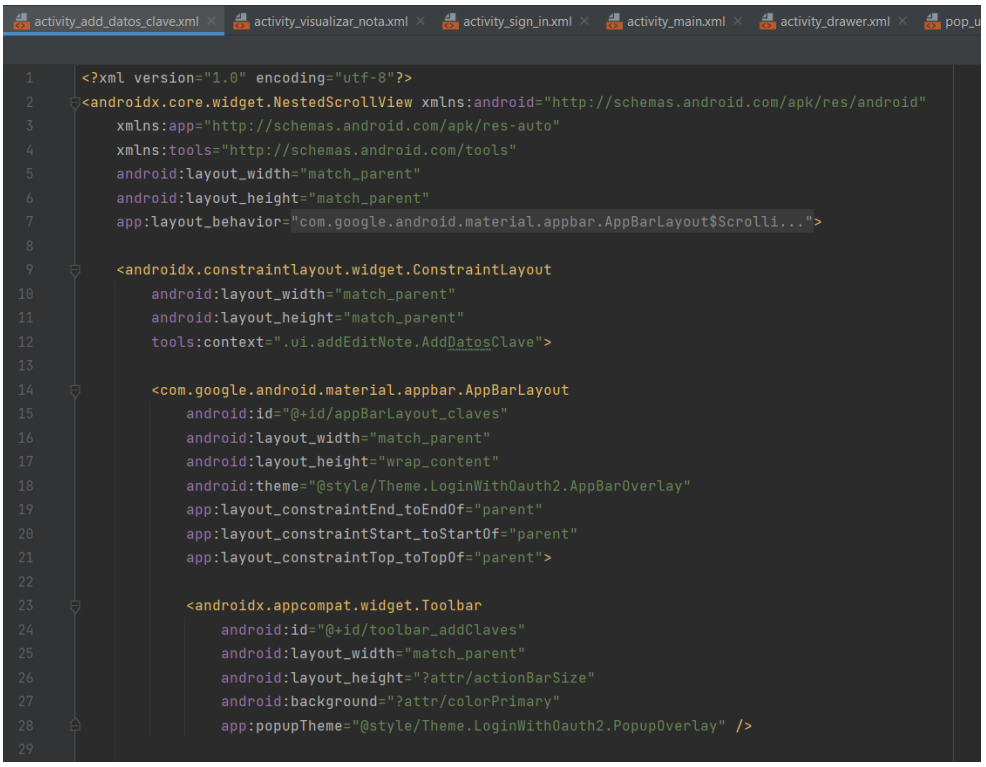


Ilustración 75: ejemplo de layout en el proyecto.

Los elementos que están dentro de los layouts son:

- **TextView**: etiquetas de texto para informar al usuario.
- **EditText**: campos editables que nos permiten introducir datos como los campos de la nota o los correos de los usuarios.
- **AutoCompleteTextView**: elemento que se usará en el menú desplegable para elegir la cama.
- **Button y FloatingActionButton**: botones para realizar acciones.
- **ImageView**: elemento para mostrar una imagen.
- **SearchView**: barra de búsqueda que será utilizada para buscar texto en las notas.
- **ListView**: lista estática que se utilizará para representar a los usuarios que tienen acceso a una nota.

Por otro lado, cabe destacar el uso del elemento *RecyclerView*, que permite representar una lista dinámica [21], con la que se pueden mostrar las notas en la pantalla principal. Esta lista, a diferencia de la ListView, permite detectar un toque sobre los elementos de la lista, también nos permite personalizar la visualización de los elementos que componen la lista y además es más eficiente para trabajar con listas dinámicas. Para implementar la *RecyclerView*, hay que declarar un adaptador,

el cual se define en el fichero NotaRVAdapter.kt, con el que se puede gestionar la lógica de la lista. Además, se le ha asignado a la lista de notas un elemento *SwipyRefreshLayout* que permite refrescar la lista.

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <com.orangegangsters.github.swipyrefreshlayout.library.SwipyRefreshLayout
        android:id="@+id/swiperLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:srl_direction="both">

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/notesRV"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            tools:listitem="@layout/nota_item">

        </androidx.recyclerview.widget.RecyclerView>
```

Ilustración 76: declaración de la recyclerView a utilizar en el proyecto.

4.2.2.5 Implementación vista modal

En nuestro prototipo hemos implementado una clase *ViewModel* [22] la cual contendrá todos los repositorios disponibles en la aplicación y será llamada por aquellas vistas que necesitan realizar un cambio en el modelo de la aplicación.

```
/**
 * Clase para representar la vista modal de las notas, esta clase sirve como una capa de comunicación
 * entre el repositorio y la interfaz de usuario
 *
 * @param application Aplicación/Contexto de la vista modal
 * @property allNotes Lista de todas las notas a mostrar en la lista
 * @property repositoryNotas Repositorio del que extraer las notas
 * @property repositoryUsuarios Repositorio del que extraer los usuarios
 * @property repositoryUsuariosNotas Repositorio del que extraer las relaciones de las entidades
 * @property repositoryNotaEliminada Repositorio de las notas eliminadas
 * @property repositoryNotaAdd Repositorio de las notas eliminadas
 */
class NotesViewModel(application: Application) : AndroidViewModel(application) {

    var allNotes: LiveData<List<Nota>>
    private val repositoryNotas: NotasRepository
    private val repositoryUsuarios: UsuarioRepository
    private val repositoryUsuariosNotas: UsuarioNotaCrossRefRepository
    private val repositoryNotaEliminada: NotaEliminadaRepository
    private val repositoryNotaAdd: NotaAddRepository
```

Ilustración 77: declaración de la clase NotesViewModel.

Además la clase de *NotesViewModel* tendrá una función para iniciar los repositorios a través de la instancia de la base de datos.

```
/**
 * Método para inicializar los DAOs, los repositorios y todas las notas de la base de datos loca
 */
init {
    val daoNotas = AppDatabase.getDatabase(application).getNotesDao()
    val daoUsuarios = AppDatabase.getDatabase(application).getUsuarioDao()
    val daoUsuariosNotas = AppDatabase.getDatabase(application).getUsuarioNotasDao()
    val daoNotasEliminadas = AppDatabase.getDatabase(application).getNotasEliminadasDao()
    val daoNotasAdd = AppDatabase.getDatabase(application).getNotasAddDao()

    // Obtenemos los repositorios
    repositoryNotaEliminada = NotaEliminadaRepository(daoNotasEliminadas)
    repositoryNotaAdd = NotaAddRepository(daoNotasAdd)
    repositoryNotas = NotasRepository(daoNotas, repositoryNotaEliminada, repositoryNotaAdd)
    repositoryUsuarios = UsuarioRepository(daoUsuarios)
    repositoryUsuariosNotas =
        UsuarioNotaCrossRefRepository(
            daoUsuariosNotas,
            repositoryUsuarios,
            repositoryNotas,
            repositoryNotaEliminada,
            repositoryNotaAdd
        )

    allNotes = liveData { }

    // Sincronizamos el usuario
    sincronizarUsuario()
}
```

Ilustración 78: inicialización de la clase NotresViewModel.

Por otro lado, la clase *NotesViewModel* tiene las funciones para añadir, eliminar, editar o sincronizar las notas. Todas estas funciones serán declaradas como corrutinas [30], con la finalidad de poder realizar llamadas asíncronas a la base de datos local y a la API-REST. Para declarar que una función es una corrutina se deberá de indicar el ámbito de ciclo de vida de la función, en nuestro caso será el viewModel para que cuando se destruya el objeto *NotesViewModel* se terminen todas las llamadas asíncronas realizadas por la clase, o también podemos declarar la función como suspend [31], la cual indicará que la función se realizará en un subproceso.

La funciones de añadir una nota, editar una nota y eliminar una nota se declaran en el ámbito de viewModel y no como un subproceso ya que en caso de que se realice alguna de estas acciones y seguidamente se cierre la aplicación y no se haya conseguido conectar con la API-REST, esta operación quedará invalidada. Si se da este caso en estas operaciones se crearían los objetos de *NotaAdd* y *NotaEliminada*.

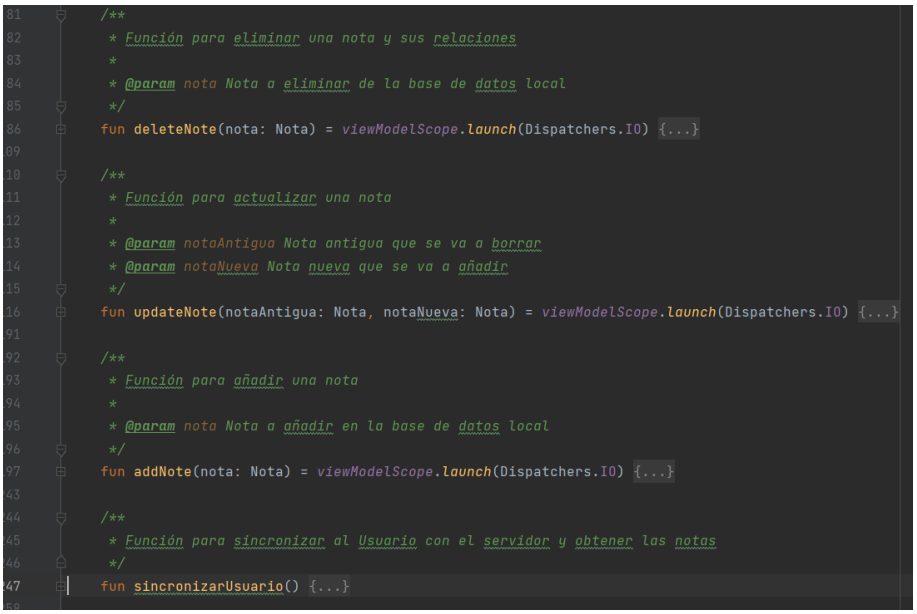


Ilustración 79: funciones de la clase NotesViewModel - parte 1.

Sin embargo, en esta parte de la implementación han surgido varios problemas ya que las funciones declaradas en el ámbito de viewModel no pueden devolver nada y no podríamos saber cuando ha tenido éxito una operación. La solución a este problema ha sido declarar varias partes de código mediante try y catch con el objetivo de que cuando se lance una excepción podamos manejarla.

También declaramos en el *NotesViewModel* las funciones para refrescar una lista y para buscar una lista en el servidor o en la base de datos local.

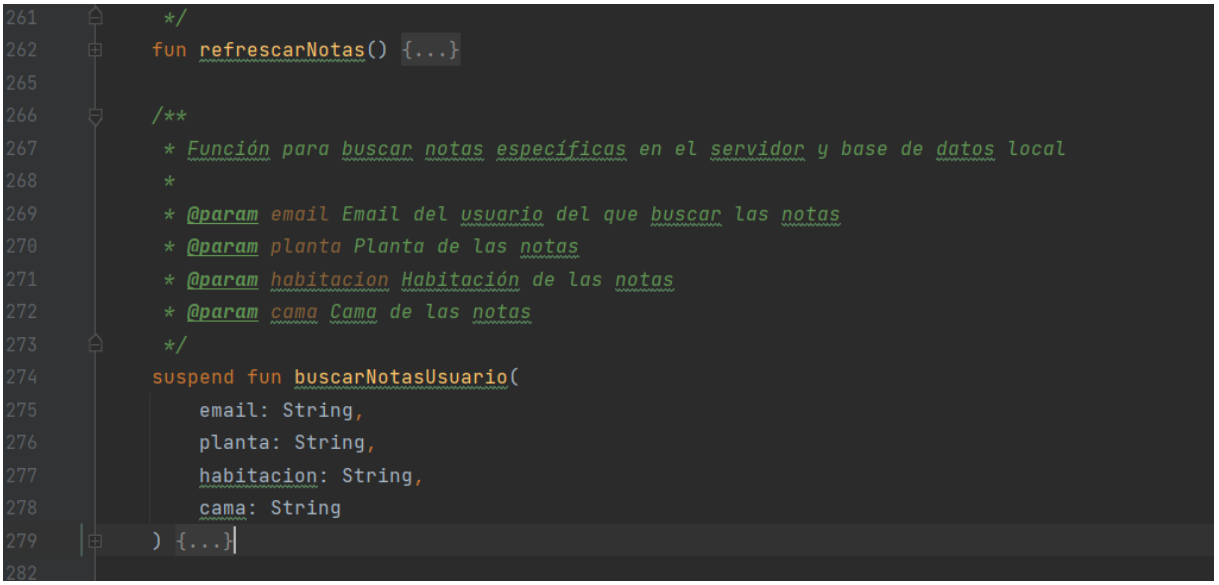
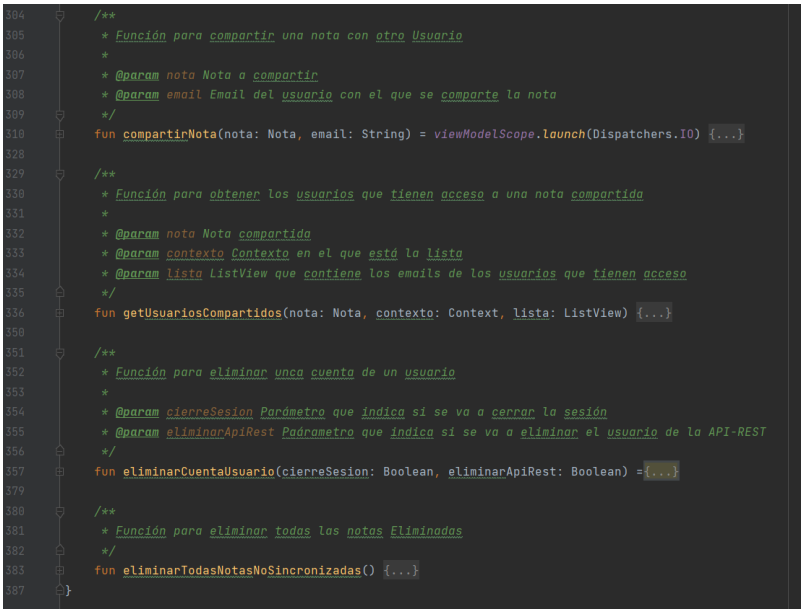


Ilustración 80: funciones de la clase NotesViewModel - parte 2.

Por último se declararán las funciones necesarias para compartir una nota, obtener los usuarios que tienen acceso a una nota, eliminar la cuenta de usuario local y eliminar los datos de la base de datos local.



```
    /**
     * Función para compartir una nota con otro Usuario
     *
     * @param nota Nota a compartir
     * @param email Email del usuario con el que se comparte la nota
     */
    fun compartirNota(nota: Nota, email: String) = viewModelScope.launch(Dispatchers.IO) {...}

    /**
     * Función para obtener los usuarios que tienen acceso a una nota compartida
     *
     * @param nota Nota compartida
     * @param contexto Contexto en el que está la lista
     * @param lista ListView que contiene los emails de los usuarios que tienen acceso
     */
    fun getUsuariosCompartidos(nota: Nota, contexto: Context, lista: ListView) {...}

    /**
     * Función para eliminar unca cuenta de un usuario
     *
     * @param cierreSesion Parámetro que indica si se va a cerrar la sesión
     * @param eliminarApiRest Paárometro que indica si se va a eliminar el usuario de la API-REST
     */
    fun eliminarCuentaUsuario(cierreSesion: Boolean, eliminarApiRest: Boolean) ={...}

    /**
     * Función para eliminar todas las notas Eliminadas
     */
    fun eliminarTodasNotasNoSincronizadas() {...}
}
```

Ilustración 81: funciones de la clase NotesViewModel - parte 3.

Respecto a la función de eliminarCuentaUsuario, esta función se llama cada vez que se cierre la sesión o se sincronice con el servidor y se encargará de eliminar los datos almacenados en la base de datos local, esta decisión de eliminar las notas de la base de datos local cada vez que se realice alguna de estas 2 acciones es debido a que si no la limpiamos, podríamos tener en la base de datos local notas las cuales no existen en la base de datos del servidor.

4.2.2.6 Implementación de Retrofit

A continuación, explicaremos cómo realizamos llamadas a la API-REST [23] alojada en un servidor desde el prototipo de aplicación móvil. Para realizar llamadas a API-REST tenemos que utilizar la biblioteca retrofit2, la cual se puede añadir al fichero de build.gradle.

Para ello, el primer paso es definir en el fichero Util un atributo estático con la url del servidor. Una vez hecho esto, se crea una interfaz para poder acceder a la API-REST.

En la interfaz hay que definir los métodos que son necesarios para realizar llamadas a la API y utilizaremos las anotaciones proporcionadas por la librería Retrofit para indicar el tipo de petición que es.

```kotlin
/**
 * Interfaz que representa la API-REST de nuestro BACK-END
 */
interface ApiRest {

    // Funciones del repositorio de Usuarios-----------------------------------------------------------

    /**
     * Método para obtener un Usuario
     *
     * @param emailUser Usuario del email a obtener
     * @return Response de una lista de usuarios
     */
    @GET( value: "user/{emailUser}")
    suspend fun getUser(@Path( value: "emailUser") emailUser: String): Response<List<Usuario>>

    /**
     * Método para registrar un Usuario en el servidor
     *
     * @param usuario Usuario a registrar
     */
    @POST( value: "user/register")
    suspend fun postUser(@Body usuario: Usuario)

```

Ilustración 82: declaración de la interfaz ApiRest.

Como podemos observar en la imagen de arriba, se ha implementado la interfaz de la API-REST y se han definido varios métodos. Los métodos declarados tienen varios componentes:

1. **Tipo de método**: definiremos qué tipo de petición realizamos, pudiendo ser GET, POST, PUT o DELETE, ya que son los métodos definidos en el backend.
2. **Ruta**: ruta del API-REST para poder acceder al recurso que se quiere.
3. **Parámetros**: los parámetros que le enviamos al servidor que a su vez son parte de la ruta, para ello utilizamos la anotación *@Path*
4. **Cuerpo del mensaje**: para indicar que un parámetro es el payload del mensaje utilizamos la anotación *@Body*

Para declarar una instancia de retrofit basta con crear este método en cada repositorio que realice llamadas a la API.

```
45      /**
46       * Método para crear una instancia de acceso a la API
47       *
48       * @return Instancia de Retrofit
49       */
50      private fun getRetrofit(): Retrofit {
51          return Retrofit.Builder()
52              .baseUrl(Util.servidor)
53              .addConverterFactory(GsonConverterFactory.create())
54              .build()
55      }
56
```

Ilustración 83: función para crear una instancia de Retrofit.

Para utilizar la interfaz definida anteriormente, debemos utilizar el objeto Retrofit de la siguiente manera.

```
val call = getRetrofit().create(ApiRest::class.java).getUser(Util.emailUser)
```

Ilustración 84: realización de una llamada mediante Retrofit.

Sin embargo, al implementar la interfaz de la API nos han surgido ciertos problemas:

- Los métodos GET no aceptan un cuerpo del mensaje por lo que funciones como buscar una nota en la que le pasas como cuerpo una nota son del tipo PUT aunque no se actualice nada en la base de datos del servidor.
- Cuando enviamos información al servidor, ya sea como parámetros de la ruta o como un cuerpo del mensaje, el nombre de los atributos debe de ser igual a las variables locales declaradas en las funciones del backend.
- Los datos que devuelve la API-REST son en formato JSON por lo que a no ser que se devuelvan todas las variables requeridos por el constructor del objeto se deberá de crear clases que representen la respuesta del servidor, es el caso de la clase *ResponseUsuario* la cual es la respuesta a la función de obtener los usuarios compartidos de una nota.

4.2.2.7 Implementación de recursos

Por último, definiremos los recursos que hemos utilizado en nuestra aplicación, estos recursos se encuentran en la carpeta res/values, en esta carpeta podemos encontrar:

- **arrays**: fichero que contiene los arrays estáticos de la aplicación, en nuestro caso contiene el array de idiomas y el array de la selección de cama.
- **colors**: fichero que contiene todos los colores que utilizamos en la aplicación.
- **dimens**: en esta carpeta están definidas las dimensiones según el dispositivo que utiliza la aplicación, hemos definido un fichero para dispositivos móviles, para dispositivos que superen un ancho de pantalla de 600dp y para dispositivos que superen un ancho de pantalla de 1240dp.
- **strings**: carpeta que contiene todos los idiomas que acepta nuestra aplicación, los idiomas son Español, Inglés y Francés.
- **themes**: carpeta que contiene los 2 temas disponibles de la aplicación, estos temas son el tema claro y tema oscuro. En cada uno de ellos hemos declarado los estilos que deben de seguir los elementos que componen las interfaces de usuario.

Respecto a los idiomas que ofrece la aplicación móvil, se podrá cambiar entre español, inglés y francés. Para implementar los idiomas bastará con definir un recurso string para cada uno y llamar al fichero como el código local del idioma. Por ejemplo, el fichero del idioma del inglés es nombrado como strings-en. Sin embargo, el idioma español es nombrado como strings sin código local ya que también será el idioma por defecto de la aplicación.

4.2.2.8 Colores elegidos

Como hemos visto anteriormente en la maqueta que muestra las vistas del prototipo de aplicación móvil, hemos elegido una paleta de colores formada por colores aptos para las personas que sufren alguna alteración en la visión de colores. La alteración de colores se puede clasificar en:

- **Deuteranopia**: Alteración de la visión al color rojo.
- **Tritanopia**: Alteración de la visión al color azul.
- **Protanopia**: Alteración de la visión al color verde.
- **Acromática**: Alteración de la visión en la que la personas solo ve en blanco y negro sin distinguir ningún color.

Por otro lado, al cambiar el tema a modo oscuro se cambian algunos colores de la aplicación, los colores elegidos para el prototipo son:

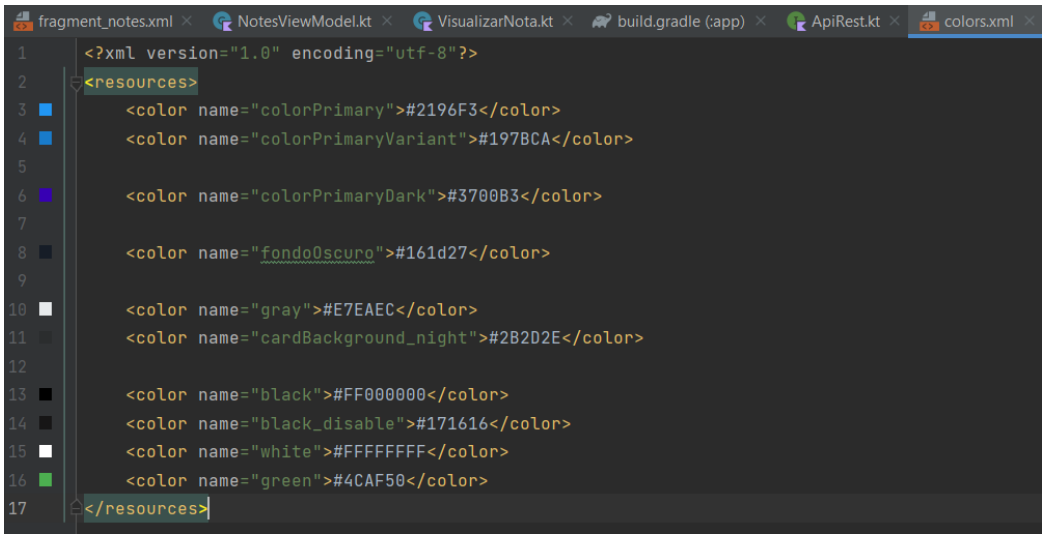


Ilustración 85: declaración de colores de la aplicación.

Como podemos ver en la imagen superior se han definido un total de 10 colores para los temas claro y oscuro.

Los colores elegidos pueden ser distinguibles por cualquier persona que sufre alguna de las alteraciones mencionadas anteriormente [28].

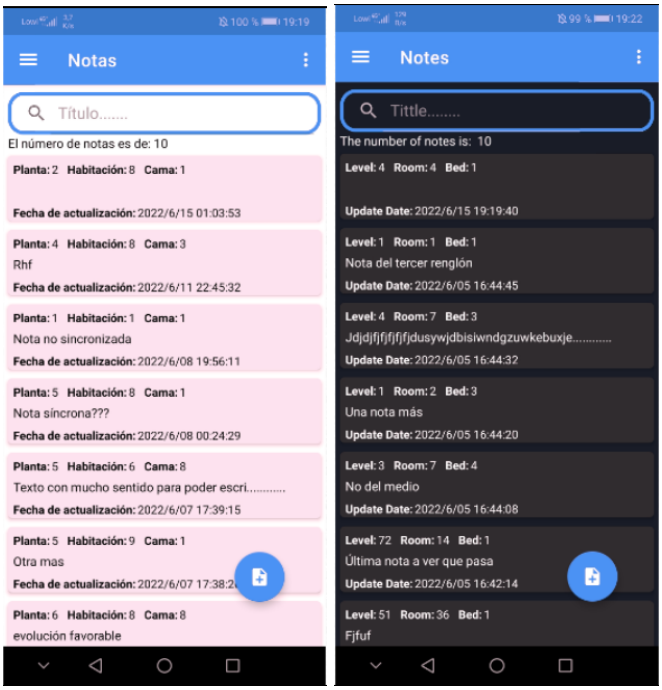


Ilustración 86: vista de la aplicación para personas con daltonismo del tipo deuteranopia.

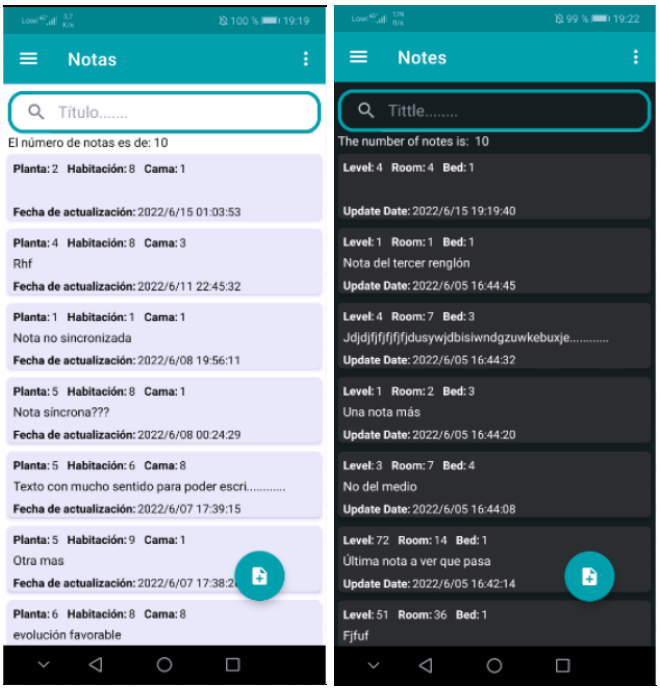


Ilustración 87: vista de la aplicación para personas con daltonismo del tipo tritanopia.

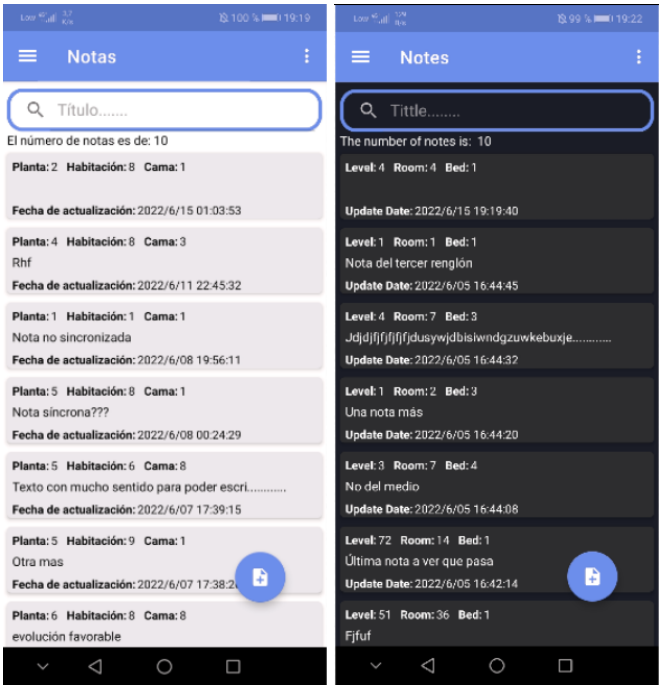


Ilustración 88: vista de la aplicación para personas con daltonismo del tipo protanopia.

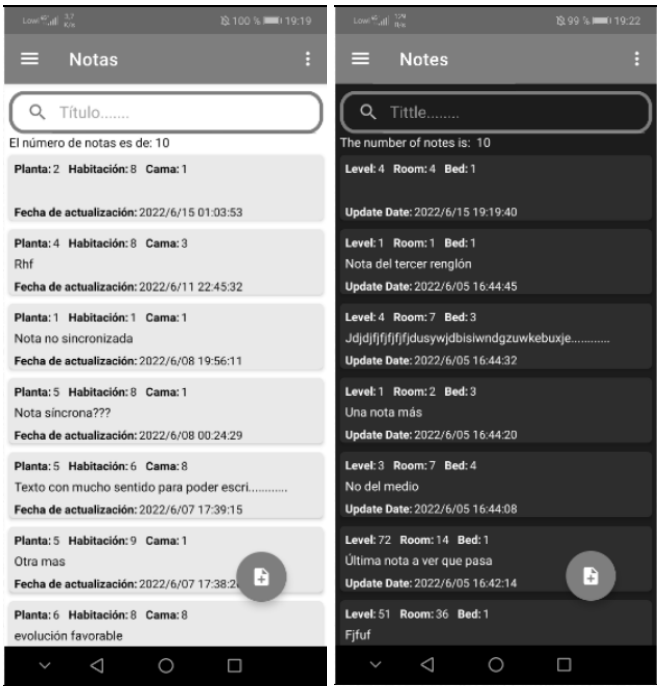


Ilustración 89: vista de la aplicación para personas con daltonismo del tipo acromática.

4.2.3 Sincronización

Como se ha explicado en la introducción, la aplicación deberá de permitir al usuario trabajar sin conexión a internet y posteriormente sincronizar los datos con el servidor. Esta parte del proyecto ha sido la más complicada debido a los continuos problemas que surgían durante la implementación.

La aplicación móvil se sincronizará automáticamente con el servidor cada vez que se ejecute la aplicación móvil. Para ello se han implementado varias funciones que gestionan la sincronización.

```
fun sincronizacionInicial(user: Usuario): LiveData<List<Nota>> {
    CoroutineScope(Dispatchers.IO).launch {  this: CoroutineScope
        launch { registrarUsuario(user) }.join()
        // Obtenemos las notas del backend
        sinicronizacionNotasApi(user.email)
    }
    return usuariosNotasDao.getNotasUsuarioFechasDESC(user.email)
}
```

Ilustración 90: función sincronizacionInicial.

En la imagen superior podemos observar las funciones de registrarUsuario y de sincronizaciónNotasApi, las cuales están declaradas en el repositorio

UsuarioNotaCrossRef. Uno de los principales problemas que surgieron en la implementación de la sincronización era que al realizar llamadas asíncronas a la API-REST, el backend no podía realizar las acciones de sincronización de las notas ya que se ejecutaban 2 hilos de forma paralela, por tanto la solución fue hacer que para obtener las notas del servidor, antes tiene que finalizar la tarea de registro del usuario, esto se consigue mediante la función join de la corrutina.

Otro de los problemas que surgieron en la implementación de la sincronización fue la capacidad de la API-REST para devolver un número determinado de notas. Ya que devolver todas las notas a las que tiene acceso un usuario puede ser bastante ineficiente, debido a que un usuario puede tener acceso a un grán número de notas pero lo más común es acceder a las notas más recientes. Para solucionar esto se crearon varias funciones, tanto en el frontend de la aplicación móvil como en el backend, que nos permite gestionar el número de notas a devolver.

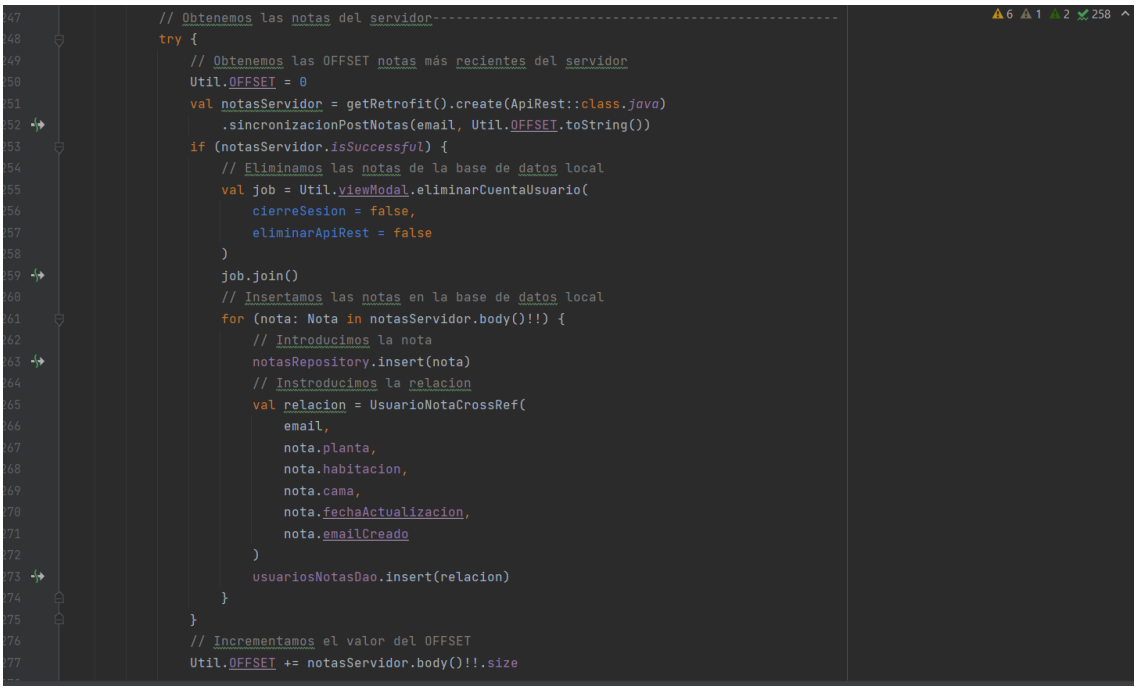
```
/**
 * Método GET para obtener las siguientes notas requeridas por el usuario
 */
router.get('/:emailUser/:offset', async function (req, res) {
    try {
        // Obtenemos los parámteros del body
        const sqlQuery =
            'SELECT n.planta, n.habitacion, n.cama, n.fechaActualizacion, n.observaciones1, n.observaciones2, ' +
            'n.ct_frecuenciaCardiaca, n.ct_frecuenciaRespiratoria, n.ct_temperatura, n.ct_presionArterial, n.ct_presionArterial2, ' +
            'FROM Nota n, UsuarioNotaCrossRef u_n ' +
            'WHERE u_n.email = ? ' +
            'AND u_n.planta = n.planta ' +
            'AND u_n.habitacion = n.habitacion ' +
            'AND u_n.cama = n.cama ' +
            'AND u_n.fechaActualizacion = n.fechaActualizacion ' +
            'AND u_n.emailCreado = n.emailCreado ' +
            'ORDER BY n.fechaActualizacion DESC ' +
            'LIMIT 10 OFFSET ?';

        pool.query(sqlQuery, [req.params.emailUser, parseInt(req.params.offset)], function (err, result, fields) {
            if (err)
                throw err;
            // Comprobamos si el resultado está vacio o no
            if (Object.keys(result).length === 0) {
                res.status(404).send('Recurso no encontrado')
            } else {
                res.status(200).json(result);
            }
        });
    } catch (error) {
        res.status(400).send(error.message);
    }
});
```

Ilustración 91: ruta del backend para obtener las notas de un usuario.

En la imagen superior se ha definido la ruta del método get para obtener un número determinado de notas. En esta función lo más importante es el valor offset, el cual será el número de notas actuales que tiene el usuario en la base de datos local. Además, el límite de notas que se devuelve es de 10, con el objetivo de no sobrecarga la respuesta del servicio.

Otro de los problemas que tuve a la hora de implementar la sincronización fue la gestión de la base de datos cuando se sincroniza con el servidor, esto se debe a que la base de datos local podría seguir almacenando notas que habían sido editadas o eliminadas pero que el servidor no nos había devolvido debido al límite de 10 notas impuesto. Para solucionar este problema decidí eliminar todas las notas almacenadas en la base de datos local cada vez que se realice una sincronización o que se cierre sesión.



```
// Obtenemos las notas del servidor----------
try {
    // Obtenemos las OFFSET notas más recientes del servidor
    Util.OFFSET = 0
    val notasServidor = getRetrofit().create(ApiRest::class.java)
        .sincronizacionPostNotas(email, Util.OFFSET.toString())
    if (notasServidor.isSuccessful) {
        // Eliminamos las notas de la base de datos local
        val job = Util.viewModal.eliminarCuentaUsuario(
            cierreSesion = false,
            eliminarApiRest = false
        )
        job.join()
        // Insertamos las notas en la base de datos local
        for (nota: Nota in notasServidor.body()!!) {
            // Introducimos la nota
            notasRepository.insert(nota)
            // Instroducimos la relacion
            val relacion = UsuarioNotaCrossRef(
                email,
                nota.planta,
                nota.habitacion,
                nota.cama,
                nota.fechaActualizacion,
                nota.emailCreado
            )
            usuariosNotasDao.insert(relacion)
        }
    }
    // Incrementamos el valor del OFFSET
    Util.OFFSET += notasServidor.body()!!.size
```

Ilustración 92: función de sincronizaciónNotasApi del cliente web.

El código de la imágen superior, está dentro de la función de sincrozaciónNotasApi, como podemos ver se llama a la función de eliminarCuentaUsuario con el objetivo de eliminar las notas de la base de datos local antes de añadir las notas obtenidas en el servidor.

En conclusión, la implementación de esta parte ha sido la más difícil del proyecto, sin embargo al permitir la utilización de la aplicación en un modo sin conexión aporta un gran valor al producto final y además resulta muy útil al usuario ya que habrá situaciones en las que el usuario no disponga de una conexión a internet.

4.2.4 Frontend página web

Como se ha especificado en las historias de usuario y en los requisitos del proyecto, inicialmente no se contemplaba la implementación de una aplicación web.

No obstante, esta parte del proyecto, se desarrolló después de terminar toda la implementación de la aplicación móvil y hubo poco tiempo dado la planificación inicial. Por tanto, la página web tiene una funcionalidad muy básica, en la que el usuario puede iniciar sesión mediante una cuenta de Google y ver las notas que el usuario tiene en una tabla.

4.2.4.1 Inicialización de la aplicación web

```
// Vistas que utilizaremos en la aplicación
app.set("views", path.join(__dirname, "views"));
app.engine(
  ".hbs",
  exphbs.engine({
    defaultLayout: "main",
    layoutsDir: path.join(app.get("views"), "layouts"),
    partialsDir: path.join(app.get("views"), "partials"),
    extname: ".hbs",
  })
);
app.set("view engine", ".hbs");

app.use(express.urlencoded({ extended: false }));
app.use(express.json());

// Carpeta publica de la aplicación que contiene los estilos CSS y las imágenes
app.use(express.static(path.join(__dirname, "public")));

// Rutas accesibles de nuestra API REST
app.get('/', (request, response) => {
    response.render('index');
})
```

Ilustración 93: inicialización de la página web.

En el fichero server.js inicializamos la aplicación web, indicando el motor de plantillas HTML que hemos utilizado para generar las vistas, la página inicial de la aplicación y la carpeta donde estarán los recursos públicos como son las imágenes o los estilos css.

Por otro lado, en el fichero clienteWeb.js se han implementado las operaciones que tiene la página web, sin embargo la única operación que se ha implementado con éxito es el de la visualización de las notas ya que las demás acciones no se han conseguido implementarlas debido a la falta de tiempo y diversos problemas de la tecnología de handlebars.

5. Pruebas y resultados

5.1 Pruebas

Dado que estamos utilizando la metodología SCRUM, se han realizado pruebas constantemente durante todas las iteraciones de desarrollo del producto. De esta forma se han detectado los errores de tareas realizadas en sprints pasados. Se han realizado pruebas tanto para el backend como para el frontend.

- Para probar el backend:
    1. El software PostMan para poder realizar peticiones a la API-REST en las primeras etapas de desarrollo del backend.
    2. Utilizaremos el administrador de base de datos DBeaver para comprobar durante todo el desarrollo del proyecto si se están almacenando los datos correctamente.
- Para probar el frontend:
    1. Durante todo el desarrollo utilizaremos los emuladores que nos proporciona Android Studio para comprobar el funcionamiento de nuestra aplicación en diferentes dispositivos.
    2. Comprobaremos que se puede navegar por todas las vistas de la aplicación.
    3. Generar interrupciones provenientes de otras aplicaciones para comprobar que la aplicación preserva el estado y no se cierra.
    4. Se comprobará el funcionamiento de la página web en diferentes navegadores.

Una vez que se ha realizado el despliegue del backend se han realizado varias pruebas por 3 usuarios externos en distintos dispositivos. Los usuarios que han probado el prototipo son:

- Un médico para obtener retroalimentación y comprobar la usabilidad de la aplicación.
- Dos ingenieros informáticos que trabajan en el ámbito de la salud.

Durante las pruebas no se han detectado bugs en la aplicación.

5.2 Resultados obtenidos

En este apartado mostraremos el resultado final de la aplicación móvil y de la página web.

Respecto a la aplicación móvil, se ha sido todo lo fiel posible a la maqueta presentada en el apartado de diseño.

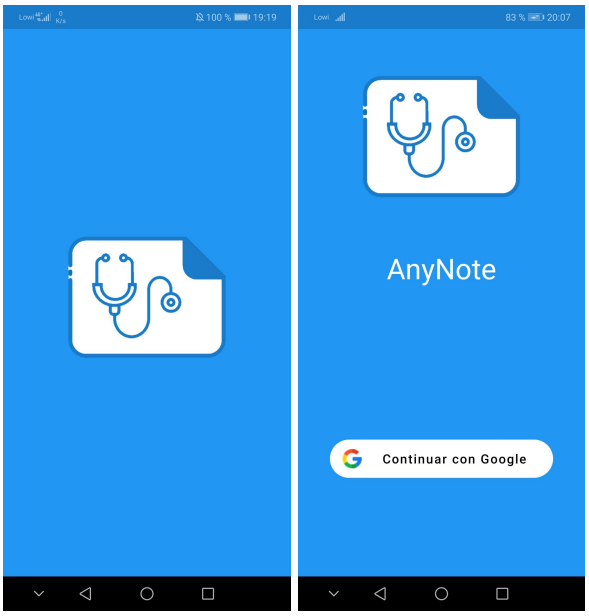


Ilustración 94: vistas de la pantalla de carga y de iniciar sesión.

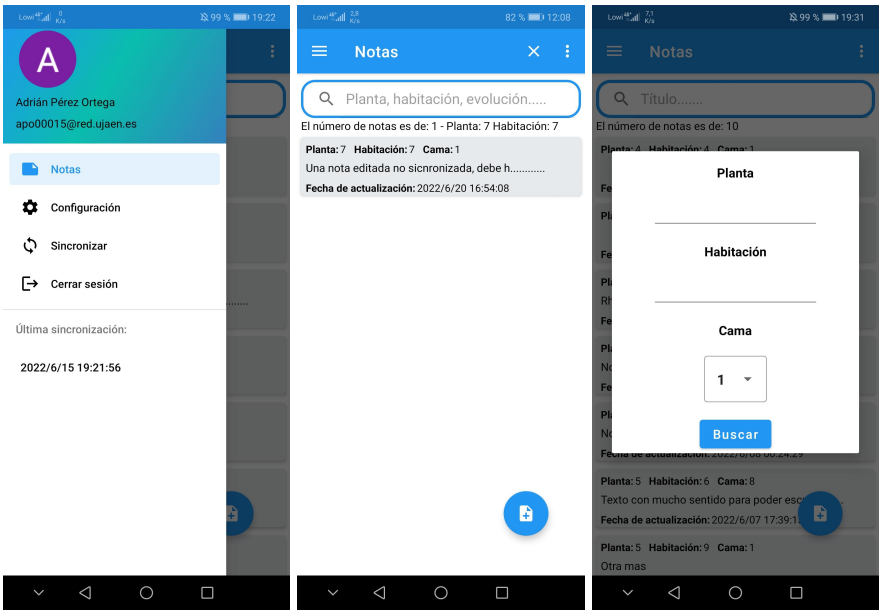


Ilustración 95: vistas de la pantalla principal de la aplicación.

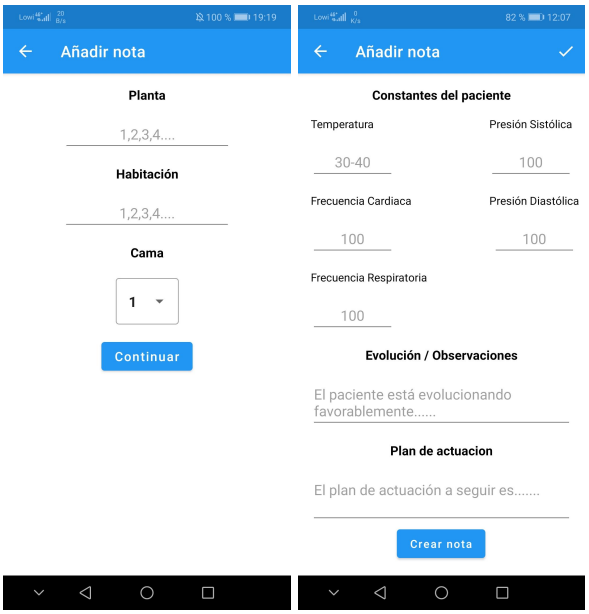


Ilustración 96: vistas de añadir/editar una nota.

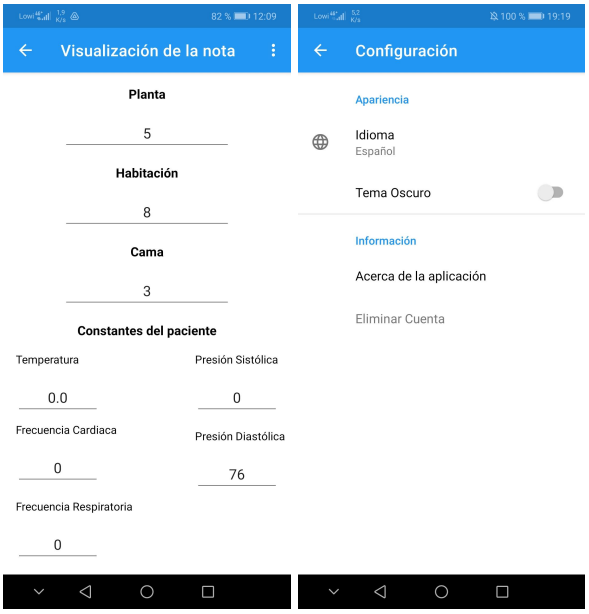


Ilustración 97: vistas de visualización de una nota y de la configuración.

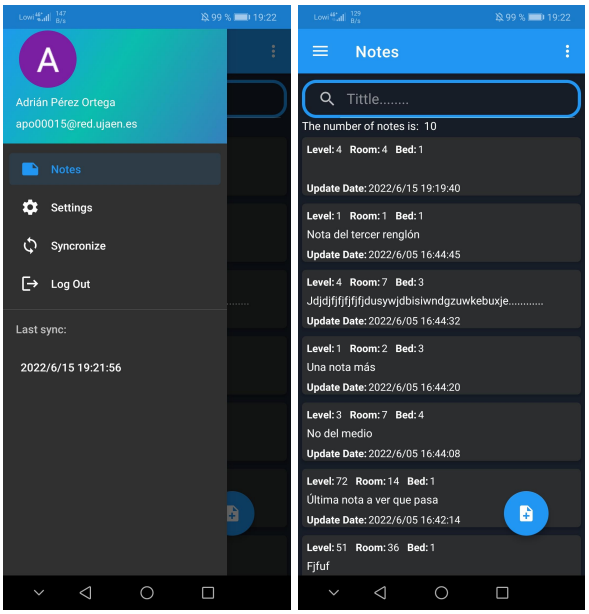


Ilustración 98: vistas de la pantalla principal en modo oscuro.

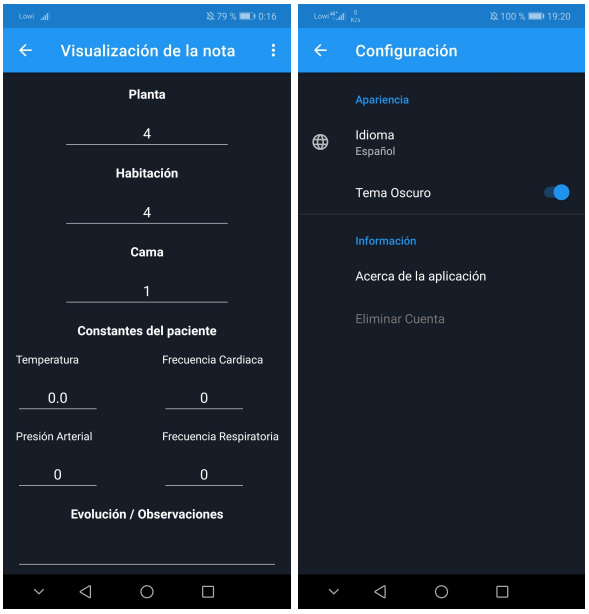


Ilustración 99: vistas de visualizar una nota y de configuración en modo oscuro.

En el siguiente enlace se puede ver un video en el que se muestra el funcionamiento de la aplicación móvil: https://drive.google.com/file/d/1P5NZDRq0m2EEBGUUHHw56zn6Znh_iDCh/view?usp=sharing

A continuación, mostraremos la aplicación móvil en una tablet con una pantalla de 1240 pixeles.

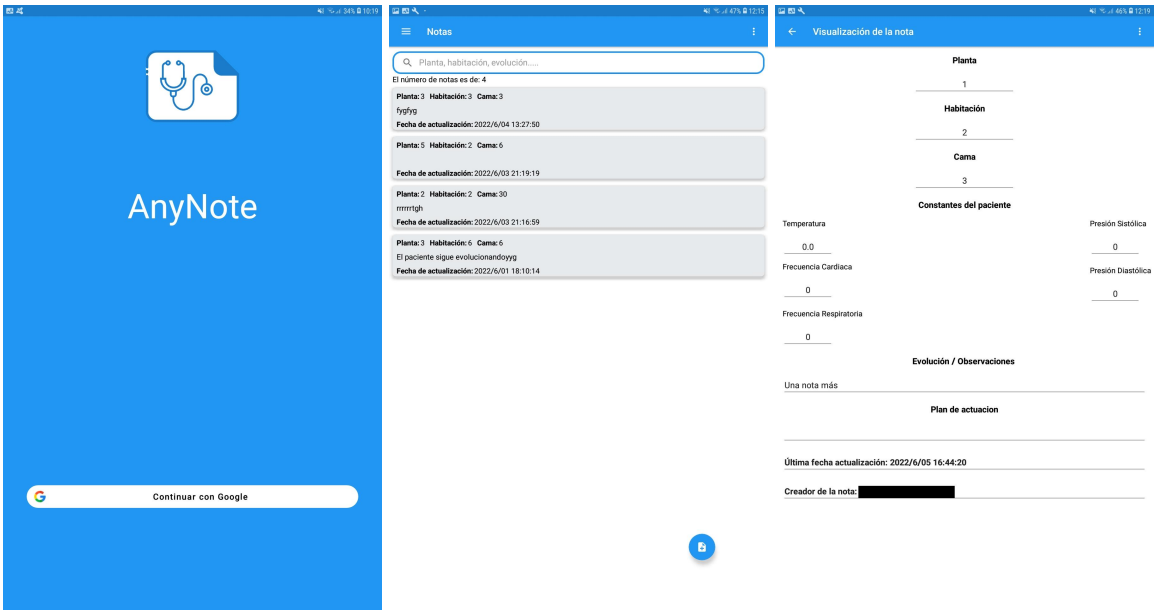


Ilustración 100: vistas de iniciar sesión, de la pantalla principal y de visualizar una nota en una tablet.

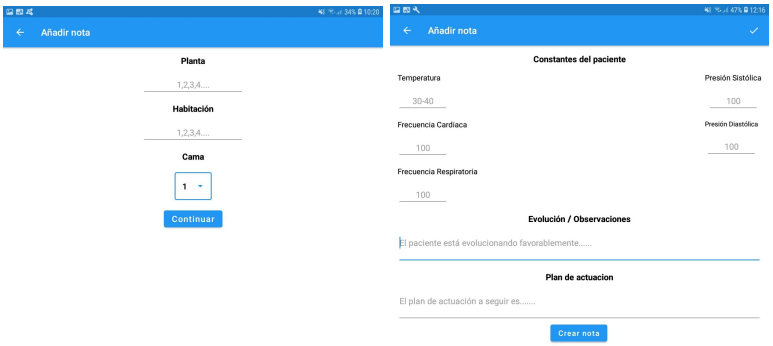


Ilustración 101: vistas de añadir una nota en una tablet.

Como podemos ver en las imágenes de arriba, la aplicación se adapta a las pantallas de las tablets.

Por último, mostraremos el resultado de la página web.

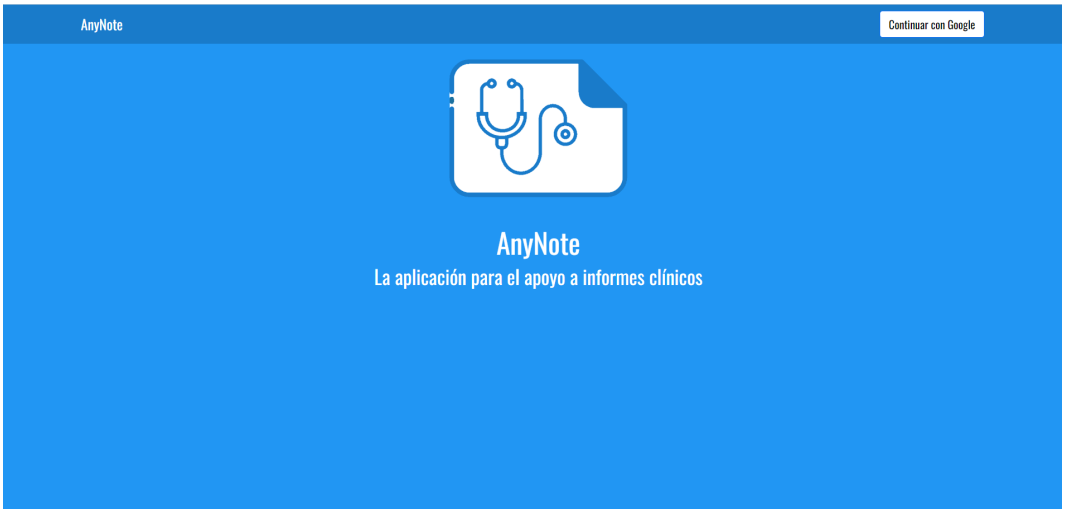


Ilustración 102: ventana principal de la página web.

Ilustración 103: ventana de visualización de las notas en la página web.

5.3 Evaluación de la aplicación

Para evaluar la calidad [24] de la aplicación nos centraremos en los siguientes aspectos:

5.3.1 Experiencia Visual

El prototipo de aplicación móvil posee una interfaz gráfica con la que el usuario puede interactuar de manera intuitiva y coherente.

- Para ello nos aseguramos que el prototipo de aplicación móvil desarrollada admite la navegación hacia atrás mediante un icono en la esquina superior izquierda o mediante el propio botón del teléfono móvil para volver para atrás.
- El prototipo de aplicación móvil se puede utilizar tanto en vertical como en horizontal. No obstante, la vista del splash screen y la vista de inicio de sesión están únicamente en vertical debido a que en el estado horizontal puede dar posibles fallos en dispositivos móviles con una pantalla pequeña al abrir la ventana de google para seleccionar la cuenta de usuario.
- La aplicación se encarga de mantener el estado de la sesión de usuario y de la propia aplicación, para ello permite la ejecución de la aplicación en segundo plano y restablece el estado al volver a la misma. Además, cuando se realiza un giro de la pantalla dentro de la aplicación, no se recrea la vista para no perder los datos introducidos.
- La aplicación mantiene el formato del texto cuando se cambia el idioma y representa todos los textos sin tener que cortarlos y con el suficiente espacio entre los diferentes elementos.

- La aplicación proporciona un modo oscuro para aumentar la visibilidad en las situaciones que hay poca luz.
- La aplicación proporciona una paleta de colores que pueden ser diferenciados por la gran mayoría de usuarios.
- La aplicación deberá permitir trabajar al usuario sin que necesite una conexión a internet.

5.3.2 Privacidad y seguridad

El prototipo de aplicación móvil deberá de gestionar de una manera segura los datos del usuario.

- La aplicación solicita la menor cantidad posible de datos privados, en nuestro caso únicamente se solicita el nombre, el correo electrónico y la foto de perfil del usuario de la cuenta de Google asociada al dispositivo.
- Los datos sensibles se almacenan en la propia base de datos local de la aplicación y no en ficheros separados.

6. Conclusiones y trabajo futuro

6.1 Conclusiones

En conclusión podemos decir que se ha cumplido con el objetivo final del producto ya que hemos implementado un prototipo de aplicación móvil la cual satisface todas las historias de usuario indicadas en el apartado 2.3. Además se ha desarrollado el prototipo de aplicación para que cualquier dispositivo Android pueda utilizarla.

Respecto a la planificación, se ha podido cumplir los plazos establecidos en la planificación inicial realizada, con lo que no hemos sufrido grandes retrasos ni imprevistos en las tareas imprescindibles del prototipo que pusieran en riesgo el desarrollo del proyecto, tanto es así que nos ha dado tiempo de crear un prototipo de aplicación web bastante sencilla con la que poder acceder a nuestras notas desde el navegador. No obstante, en la página web no he podido implementar todas las funcionalidades que me hubiera gustado debido a la falta de tiempo y a diversos problemas que tuve en la utilización de handlebars.

Por otro lado, uno de los grandes éxitos del proyecto ha sido conseguir desplegar el backend en un servidor de hosting, ya que a priori parecía una acción fácil, sin embargo me encontré con varias problemas a la hora de querer realizar el despliegue, debido a que en la mayoría de servidores no permitían utilizar complementos como son los gestores de bases de datos y esa es la razón por la que tuve que crear la base de datos en clever cloud y desplegar el backend en Heroku.

Por último, en lo personal estoy bastante satisfecho con el resultado final del proyecto ya que durante estos meses, en los que he estado trabajando en el proyecto, he podido aprender nuevas tecnologías de desarrollo de software como son el entorno de ejecución de node.js y el lenguaje de programación Kotlin, sabiendo que los conocimientos que he obtenido me van a ser de gran ayuda en un futuro.

6.2 Trabajo futuro

Sobre el prototipo de aplicación móvil, aún se pueden realizar varias cosas para añadir valor al producto.

La página web del prototipo de la aplicación AnyNote se debería de mejorar para permitir al usuario realizar todas las funcionalidades que tiene el prototipo de aplicación móvil.

Por otro lado, se podría utilizar la librería de Kotlin multiplatform para permitir utilizar la aplicación móvil en dispositivos móviles con sistema operativo IOS, pudiendo reutilizar la mayoría de código realizado.

Respecto a la propia funcionalidad de la aplicación móvil, se pueden realizar varias mejoras, por ejemplo aumentar el número de notas que el usuario reciba cuando sincronice la aplicación, no obstante esta funcionalidad va en función de la capacidad de la API-REST. También se podría añadir una sección o fragmento en el que solo se muestran las notas compartidas, para facilitar al usuario identificar cuales son las notas que tiene compartidas o le han compartido, sin tener que ir comprobando una por una. Por otro lado, se podría añadir la funcionalidad de eliminar cuenta de usuario, la cual ahora mismo está deshabilitada debido a que no se ha conseguido implementar con éxito.

Por último, se podría subir el prototipo de aplicación a Play Store para así poder conseguir la verificación de Google y poder incluir una política de privacidad para poder realizar la autenticación con Facebook.

Bibliografía

[1] Desarrolladores de Android [Online], Available: https://developer.android.com/design?hl=es-419 ,[Último acceso: 30-03-2022].

[2] Guía SCRUM [Online], Available: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-European.pdf ,[Último acceso: 29-03-2022].

[3] Kanban, Available: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban ,[Último acceso: 29-03-2022].

[4] MySQL, Available: https://openwebinars.net/blog/que-es-mysql/ ,[Último acceso: 5-04-2022].

[5] Kotlin, Available: https://kotlinlang.org/ ,[Último acceso: 10-06-2022].

[6] DawnGriffiths & David Griffiths, Head First Kotlin: A Brain-Friendly Guide, O'REILLY, 2019

[7] Kotlin Multiplatform, Available: https://kotlinlang.org/docs/multiplatform-mobile-integrate-in-existing-app.html#what-s-next ,[Último acceso: 10-06-2022].

[8] Características node.js, Available: https://openwebinars.net/blog/que-es-nodejs/ ,[Último acceso: 22-04-2022].

[9] NPM, Available: https://desarrolloweb.com/articulos/modulos-npm-nodejs.html ,[Último acceso: 22-03-2022].

[10] Handlebars, Available: https://desarrolloweb.com/manuales/manual-handlebars.html ,[Último acceso: 15-06-2022].

[11] Firebase, Availabe: https://firebase.google.com/ ,[Último acceso: 17-06-2022].

[12] Método INVEST, Available: https://www.scrummanager.net/bok/index.php?title=INVEST ,[Último acceso: 29-03-2022].

[13] Estándar fecha y hora, Available: https://en.wikipedia.org/wiki/ISO_8601 ,[Último acceso: 20-05-2022].

[14] Operaciones bases de datos, Available: https://www.sqlshack.com/es/eliminar-en-cascada-y-actualizar-cascada-en-la-clave-externa-de-sql-server/ ,[Último acceso: 10-05-2022].

[15] Introducción node.js, Available: https://desarrolloweb.com/manuales/manual-nodejs.html ,[Último acceso: 01-06-2022].

[16] Módulo Express; Available: https://expressjs.com/en/starter/hello-world.html ,[Último acceso: 15-04-2022].

[17] Enrutamiento API, Available: https://expressjs.com/en/guide/routing.html ,[Último acceso: 15-04-2022].

[18] Creación proyecto firebase, Available: https://firebase.google.com/docs/auth/android/google-signin?hl=es ,[Último acceso: 05-04-2022].

[19] Android Room, Available: https://developer.android.com/codelabs/android-room-with-a-view#0 ,[Último acceso: 25-04-2022].

[20] Layout, Available: https://developer.android.com/guide/topics/ui/declaring-layout ,[Último acceso: 03-05-2022].

[21] RecyclerView, Available: https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=es-419 ,[Último acceso: 03-05-2022].

[22] Implementación ViewMode, Available: https://developer.android.com/topic/libraries/architecture/viewmodel ,[Último acceso: 16-04-2022].

[23] Implementacion Retrofit, Available: https://cursokotlin.com/tutorial-retrofit-2-en-kotlin-con-corrutinas-consumiendo-api-capitulo-20-v2/ ,[Último acceso: 12-05-2022].

[24] Calidad de la aplicación, Available: https://developer.android.com/docs/quality-guidelines/core-app-quality?hl=es-419 ,[Último acceso: 17-06-2022].

[25] Vida útil de los ordenadores, Available: https://pcredcom.com/blog/computo/vida-util-del-portatil/ , [Último acceso: 15 - 04 - 2022]

[26] Vida útil de móviles y tablets, Available: https://revistaempresarial.com/tecnologia/consejos-para-alargar-la-vida-util-del-celular/ , [Último acceso: 15 - 04 - 2022]

[27] Sueldos de los recursos humanos, Available: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2022-5438 , [Último acceso: 15 - 04 - 2022]

[28] Color Blindess Simulator, Available: https://www.color-blindness.com/coblis-color-blindness-simulator/ , [Último acceso: 15 - 06 - 2022]

[29] Método MoScoW, Available: https://www.productplan.com/glossary/moscow-prioritization/ , [Último acceso: 20 - 06 - 2022]

[30] Corrutinas Kotlin, Available: https://developer.android.com/kotlin/coroutines?hl=es-419 , [Último acceso: 13 - 06 - 2022]

[31] Mejorar rendimiento de las corrutinas, Available: https://developer.android.com/kotlin/coroutines/coroutines-adv?hl=es-419 . [Último acceso: 13 - 06 - 2022]

Apéndice I. Manual de instalación del sistema

Para poder instalar la aplicación en un dispositivo android, bastará con instalar el fichero AnyNote.apk, la aplicación no requiere de permisos especiales.
Respecto a realizar una instalación local en un ordenador, el software que se ha utilizado y su versión para poder utilizar el prototipo de aplicación móvil o el prototipo de página web en el ordenador es:

1. Node.js con la versión 16.13.1
2. Docker Desktop con la versión 4.6.0
3. Android Studio con la versión 2020.3
4. Visual studio code con la versión 1.68.1
5. Gradle con la versión 7.3.3
6. JDK 11
7. DBeaver con la versión 22.0.0

No es necesario instalar el software anterior según un orden, no obstante es aconsejable tener instalado el JDK 11 antes de instalar el IDE Android Studio.

Apéndice II. Manual de Usuario

En este apéndice explicaremos cómo navegar por el prototipo de aplicación móvil y utilizar las distintas funcionalidades que nos ofrece.

La primera vista que nos aparecerá en el prototipo será la pantalla del splash screen durante 2 segundos.

Ilustración 104: pantalla de carga de la aplicación.

En caso de que el usuario no haya iniciado sesión te redirigirá a la ventana de inicio de sesión, en caso contrario te redirigirá a la vista principal.

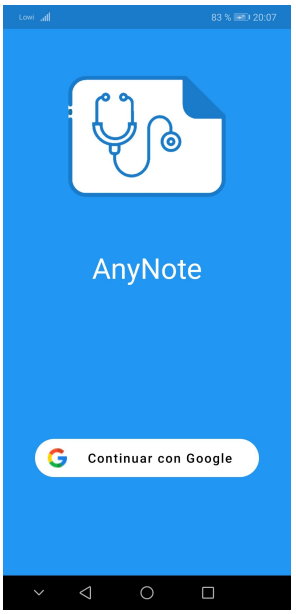


Ilustración 105: pantalla de iniciar sesión.

Una vez has iniciado sesión se te redirigirá a la ventana principal, la cual contendrá el menú desplegable y la lista de notas.

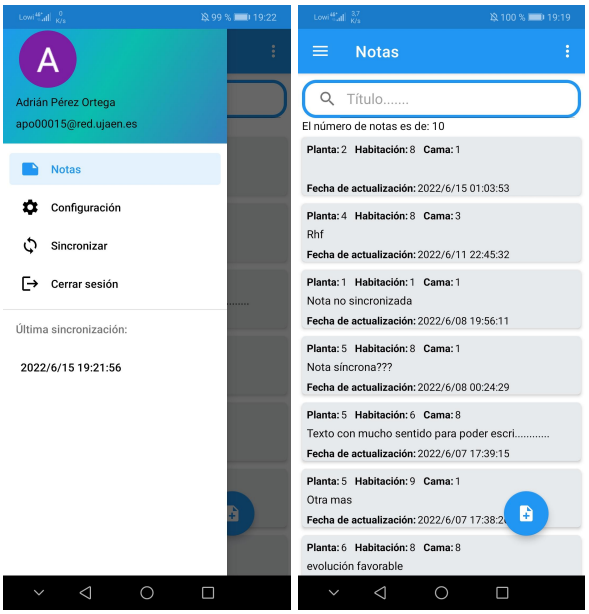


Ilustración 106: pantalla principal de la aplicación.

Una vez que estás en la ventana principal puedes realizar varias acciones:

1. Abrir el menú de la barra de configuración para ordenar la lista o buscar una nota.

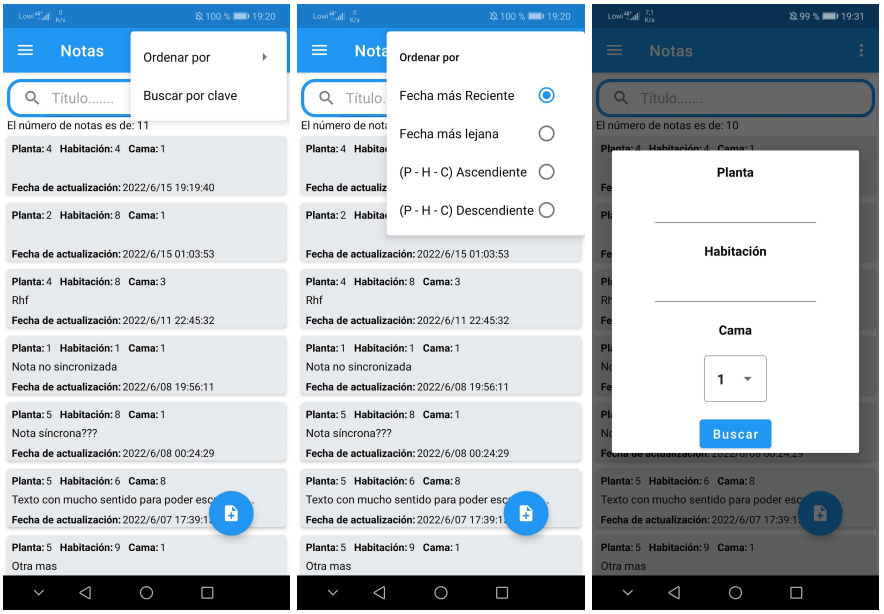


Ilustración 107: opciones en la pantalla principal.

Al seleccionar cualquier ítem del tipo de ordenamiento se cambia el orden de las notas en la lista. Por otro lado, al pulsar el ítem de buscar nota a

aparecerá una ventana emergente en la que se introducirá los campos de las notas, al pulsar el botón de buscar se actualizará la lista de notas a mostrar.

2. Realizar una búsqueda con la barra de búsqueda que hay en la pantalla principal para buscar texto coincidente en las notas que hay en la base de datos local.
3. Seleccionar una nota para eliminarla, para ello bastará con mantener pulsada una nota hasta que aparezca la imagen de selección, una vez que hay una nota seleccionada las demás notas se seleccionan mediante un simple toque.

Ilustración 108: selección de notas en la lista.

Una vez que las notas estén seleccionadas, aparecerá en la barra de configuración el icono de eliminar, si pulsas el icono se eliminarán las notas de la base de datos local y desaparecerán de la lista.

4. Al tocar una nota sin que esté activa la selección, se procederá a abrir la ventana de visualización de la nota, en la que podrás ver los campos de la nota.

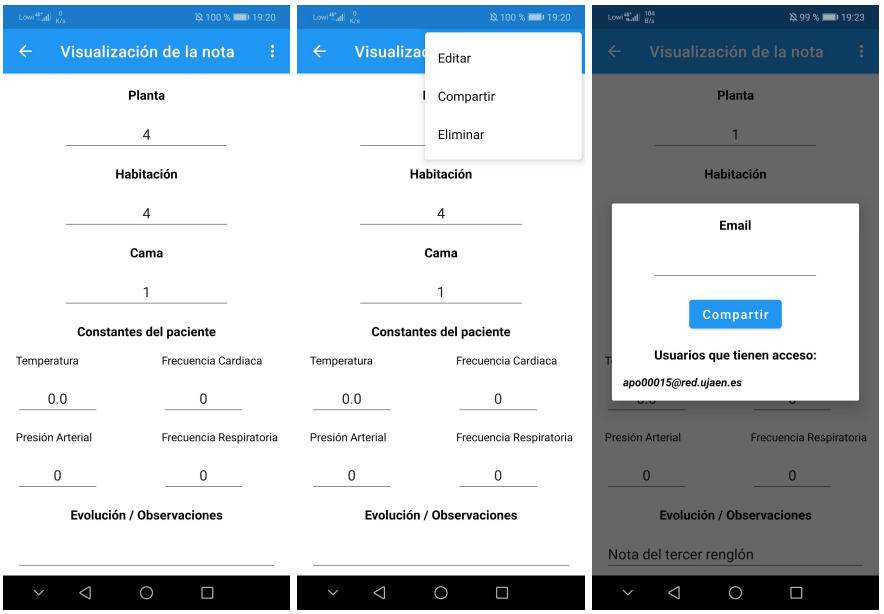


Ilustración 109: vistas de visualización de notas.

Como podemos ver en la ventana aparecerán los datos de la nota, para ver todos los datos bastará con desplazar la ventana hacia abajo. Por otro lado,, en esta ventana hay un menú en la barra de configuración en el que podemos realizar 3 acciones:

1. Editar: Si pulsas el ítem de editar se redirigirá a la ventana de añadir una nota.
2. Compartir: Si pulsas el ítem de compartir, te abrirá una ventana emergente en la que introducimos el email del usuario al que se la quieres compartir.
3. Eliminar: Si pulsas el ítem de eliminar, se eliminará la nota y te redirigirá a la ventana principal.

5. En la ventana principal aparecerá un botón flotante en la esquina inferior a la derecha, al pulsar este botón se redirigirá a la ventana de añadir una nota.

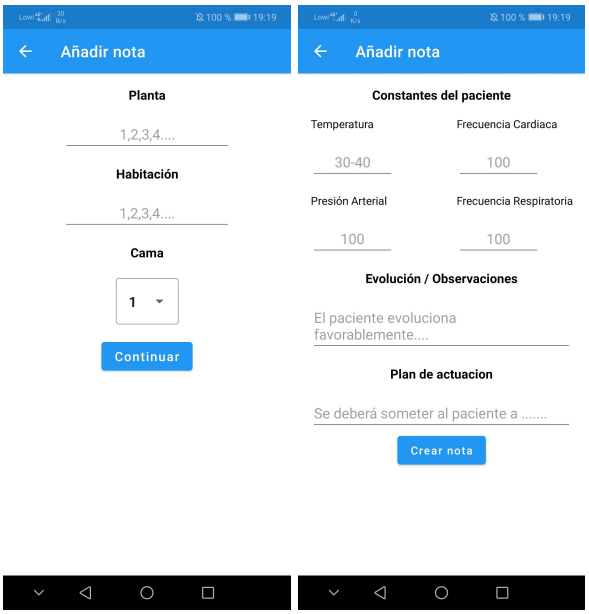


Ilustración 110: vistas de añadir una nota.

En la primera ventana aparecerán los campos de la clave de la nota, al introducirlos y pulsar el botón de continuar se te redirigirá a la ventana de añadir los campos opcionales a la nota, al pulsar el botón de crear nota se redirigirá a la ventana principal y aparecerá la nota creada o editada en la lista.

Por otro lado, explicaremos las acciones que podemos realizar en el menú desplegable.

1. Podemos pulsar el ítem de sincronizar y se sincronizará la aplicación móvil con el servidor, para tener las notas actualizadas.
2. Podemos pulsar el ítem de cerrar sesión para cerrar la sesión del usuario, al pulsarlo se redirigirá al usuario a la ventana de inicio de sesión.
3. Podemos pulsar el ítem de configuración para abrir una ventana en la que podremos cambiar ciertas características de la aplicación.

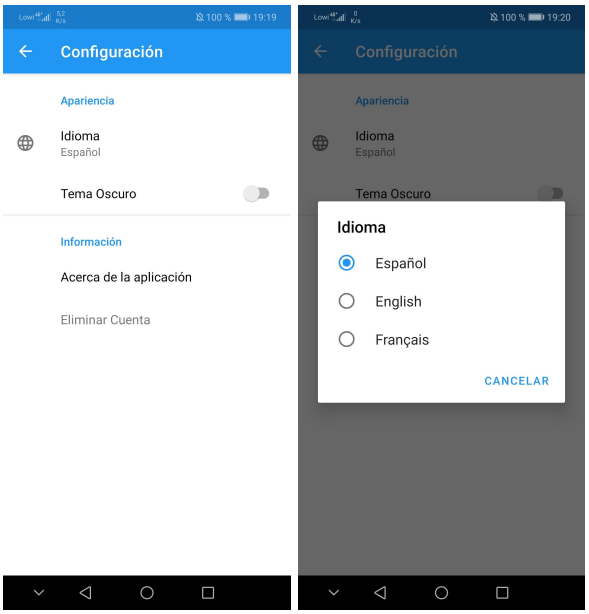


Ilustración 111: vistas de configuración.

Cuando seleccionas el modo oscuro de la aplicación así es como se ven algunas de las demás vistas.

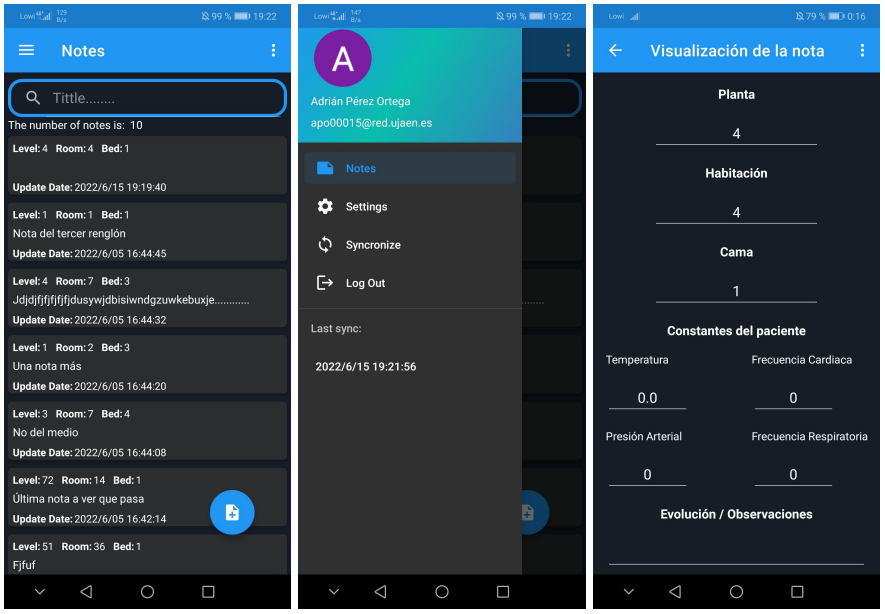


Ilustración 112: vistas en modo oscuro.

Por otro lado, indicar que cuando se comete algún error o se necesita informar al usuario sobre determinadas acciones, la aplicación mostrará varias alertas para informar al usuario. Algunas de las alertas que se mostrará a los usuarios son:

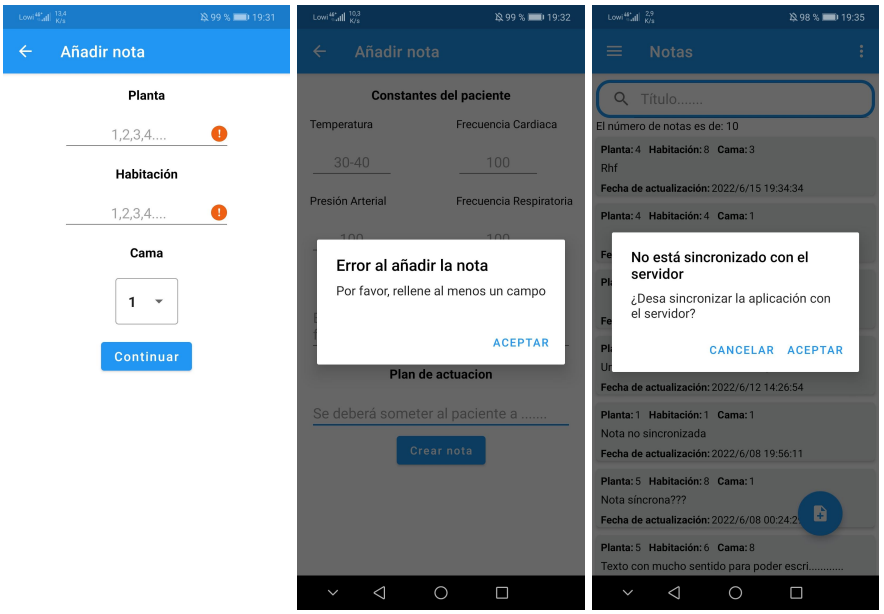


Ilustración 113: información de errores en la aplicación.

Por otro lado, para poder utilizar la página web solo hará falta un navegador y se podrá acceder desde la url: https://tfg-notas.herokuapp.com/

Apéndice III. Descripción del contenido entregado

A continuación comentaremos cuales son los archivos que se han adjuntado junto con la memoria del proyecto.

- **TFG_Perez_Ortega_Adrian.pdf**: memoria del proyecto.
- **ProyectoKotlin.zip**: archivo comprimido que contiene el código de la aplicación móvil.
- **ProyectoNodeJS.zip:** archivo comprimido que contiene el código del backend de la aplicación y el frontend de la página web.
- **Base_de_datos.txt**: fichero que contiene las sentencias para construir la base de datos del backend.
- **AnyNote.apk**: archivo para instalar la aplicación móvil en un dispositivo Android.