



CreepyRabbit

Proyecto final DAM-Dual

ALUMNO: Elena Cirstea

CICLO: Desarrollo de aplicaciones Multiplataforma

GRUPO: DAM-DUAL

MÓDULO: Proyecto final

CURSO: 2020/2021

ÍNDICE

ANÁLISIS PREVIO	2
Descripción del producto o proyecto a realizar	2
Descripción de la empresa	2
Justificación del proyecto	2
Planificación del proyecto	4
ANÁLISIS Y DISEÑO	9
Diagrama de casos de uso	9
Diagrama de clases	11
Diagrama de secuencia de aplicación	12
Mapa de navegación de la aplicación	14
Diagrama de E-R/relacional y scripts	15
Guía básica de estilos	17
Diseño de interfaces	19
CONFIGURACIÓN Y DESARROLLO DEL SOFTWARE	24
Desglose de la tecnología utilizada	24
Implementación	26
Pruebas	32
Memoria económica	36
ABSTRACT DEL PROYECTO	38
BIBLIOGRAFÍA	39
WEBGRAFÍA	39
HERRAMIENTAS UTILIZADAS	39

ANÁLISIS PREVIO

Descripción del producto o proyecto a realizar

CreepyRabbit es una aplicación para el sistema operativo Android que se lanza en fase de proyecto piloto. Ofrece contenido multimedia y está orientada al entretenimiento de un público específico que se deleita con historias de terror, específicamente las conocidas con el nombre de *creepypastas*. Se entiende por este concepto las historias de terror, de tamaño variado, recogidas y compartidas a través de internet y que tienen el propósito de inquietar al lector mediante la duda de su veracidad. Empezaron a surgir en la década de los 1990 mediante correos en cadena, aunque se desconoce el origen exacto de este tipo de narraciones. A partir de 2008 surgieron dos páginas web que recogían este contenido de forma explícita. Hoy en día se pueden encontrar de forma escrita o narradas. Al igual, se puede encontrar la misma historia en distintos idiomas.

Descripción de la empresa

La aplicación móvil es un encargo de un cliente privado. El cliente se puso en contacto para manifestar su deseo de contratación de servicios que pudiesen hacer real una idea suya. Se han llevado a cabo varias reuniones en las cuales se han ido ajustando los requisitos y el cliente ha dado su opinión respecto a los avances.

Justificación del proyecto

El objetivo principal de la aplicación móvil es ofrecer un espacio en el cual el usuario pueda escuchar historias. El público objetivo son personas mayores de 16 que tienen un nivel intermedio-alto del idioma inglés. El mensaje que se quiere transmitir con esta aplicación es que las historias pueden unir gente de distintos países e idiomas, pero que comparten el mismo gusto por el misterio y la inquietud que sienten con una historia *creepypasta*.

Para poder llevar a cabo la propuesta del cliente, se ha realizado un pequeño estudio de mercado para poder ofrecer el mejor producto al cliente. Según el estudio realizado, se han encontrado distintas aplicaciones específicas en [Google Play Store](#).

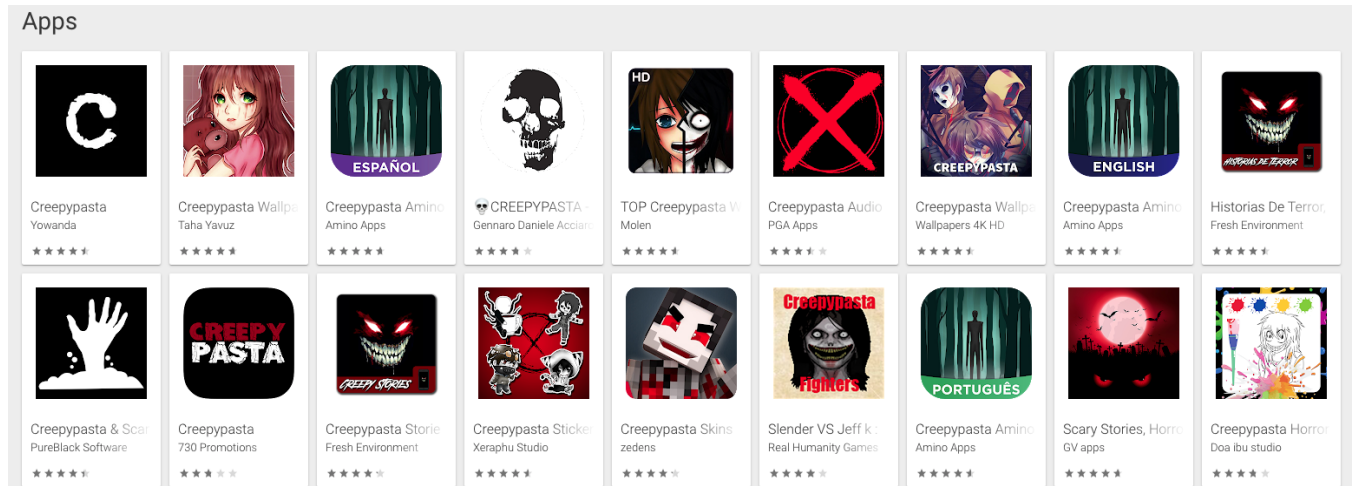
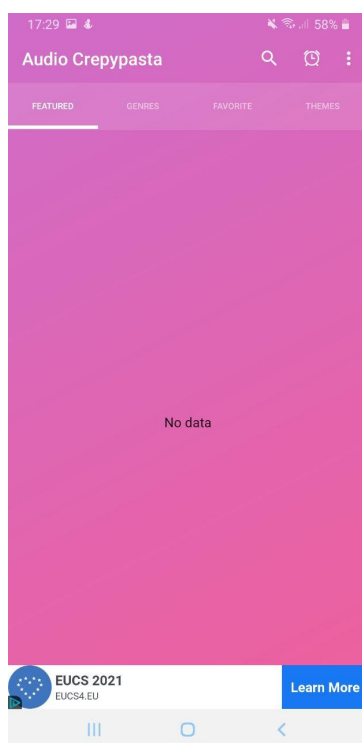
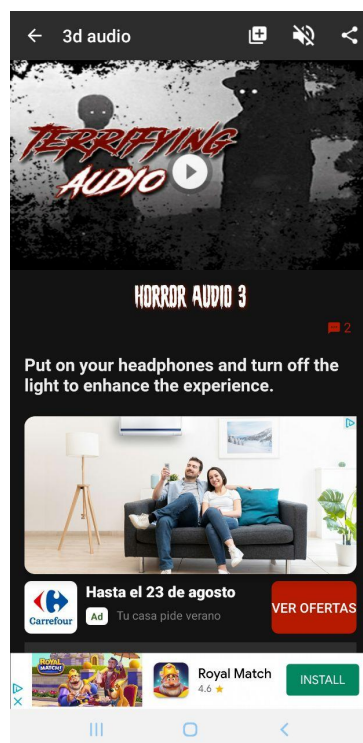


Figura 1.1 : listado aplicaciones en Google Play con la misma temática

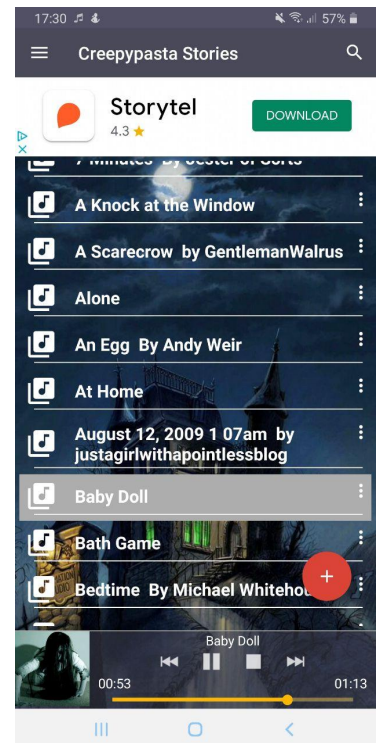
Del listado que ofrece Google Play Store, la gran mayoría de las aplicaciones ofrecen entradas de texto con estas historias. No obstante, las siguientes aplicaciones manejan contenido multimedia:



1



2



3

-
1. [Creepypasta Audio Stories. Horror Audio Books](#)
 2. [Creepypasta Stories](#)
 3. [Audio Creepypasta collection. Horror-scary stories](#)

Al igual, otros sitios que presentan parcialmente el contenido que el cliente desea plasmar en la aplicación móvil, serían canales específicos de YouTube que suben vídeos narrando estas historias o podcasts en la aplicación Spotify.

Según el breve análisis del mercado actual, se ha llegado a la siguiente conclusión: la aplicación móvil del cliente debería destacar por la integración de diverso contenido de múltiples autores, aportando al mismo tiempo una interfaz de usuario moderna e intuitiva, simple y que el diseño no inspire terror. Se entiende que al ser un tema de entretenimiento tan específico, a veces es difícil que los nuevos creadores se den a conocer, sobre todo al existir un mercado tan cerrado y escaso, por lo tanto esta aplicación móvil podría ser una perfecta oportunidad para llegar a más público, tanto para los autores como para los narradores.

Planificación del proyecto

El proyecto se llevará a cabo en un único equipo y por una única persona. La metodología que se va a utilizar en el desarrollo del presente proyecto es **modelo evolutivo**. Se ha decidido emplear esta metodología porque es iterativa, permitiendo desarrollar versiones cada vez más completas e integrando los posibles nuevos cambios que requiera el cliente o las tecnologías.

Las fases del proyecto son las siguientes:

1. ANÁLISIS DE REQUISITOS

Durante esta primera fase se ha realizado una entrevista inicial con el cliente en la cual ha manifestado sus necesidades y condiciones. Después de la mencionada entrevista, se han podido asentar las especificaciones de requisitos del sistema, divididas en funcionales y no funcionales.

Los requisitos **funcionales** serían los siguientes:

- La aplicación móvil se diseñará para la plataforma de Android.
- Deberá permitir a los usuarios registrados poder escuchar el contenido multimedia en su dispositivo
- Los usuarios registrados podrán crear y guardar listas de reproducciones, como por ejemplo una lista de audios favoritos
- Debe emplear una API mínima 26

En cuanto a los requisitos **no funcionales**, es deseable que la aplicación cumpla los siguientes:

- *Eficiencia*: el servidor debe ser capaz de procesar el contenido de N audios por segundo y responder en menos de 5 segundos al usuario con la carga del mismo contenido.
- *Seguridad*: los usuarios podrán reportar cualquier audio que no sea adecuado o esté identificado de manera errónea.
- *Utilización*: la interfaz de usuario debe ser muy intuitiva y simple.

Al final de cada reunión con el cliente se ha redactado un documento para reflejar todos los cambios acordados con el cliente. Cabe mencionar también que cada uno de los diseños ha sido aprobado por el cliente.

2. DISEÑO

El diseño tendrá como objetivo buscar un estilo equilibrado y coherente durante toda la interfaz. Se ajustará según los recursos disponibles y los criterios que pida el cliente.

3. CODIFICACIÓN

En esta parte se va a tratar de crear un código que presente modularidad, corrección, fácil de leer, eficiente y portable. La organización de la codificación del proyecto se va a realizar de la siguiente manera:

Front-end: se desarrollará una aplicación móvil en Android Studio con Kotlin.

Back-end: se creará un microservicio en Java para el manejo de la información de los usuarios y la autenticación.

Gestor de bases de datos: MongoDB, Firestore y FireCloud

4. PRUEBAS.

Se prueban las distintas opciones para detectar errores y se depuran. Las pruebas serán unitarias y de integración.

PRUEBAS UNITARIAS: consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente).

PRUEBAS DE INTEGRACIÓN: se realizan una vez que se han realizado con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo con todas sus partes interrelacionadas.

PRUEBAS MANUALES: las personas encargadas ejecutan manualmente los casos de prueba sin usar ninguna herramienta de automatización.

5. DOCUMENTACIÓN

De todas las etapas, se documenta y guarda toda la información.

6. EXPLOTACIÓN.

Se prueba la aplicación en distintos emuladores disponibles a través de Android Studio.

7. MANTENIMIENTO.

Se mantiene el contacto con el cliente para la actualización y modificación de la misma en un futuro.

Para poder llevar a cabo todo toda la organización se ha empleado un diagrama de Gantt en el cual se especifica una aproximación orientativa del proyecto. Se han propuesto unas fechas para las distintas partes del proyecto y una organización que se ajusten a los hitos. A continuación, se muestra el diagrama de Gantt en la *Figura 1.2*. Cabe mencionar que algunos aspectos se han ido modificando a lo largo de todo el proyecto, aunque no estén recogidos específicamente.

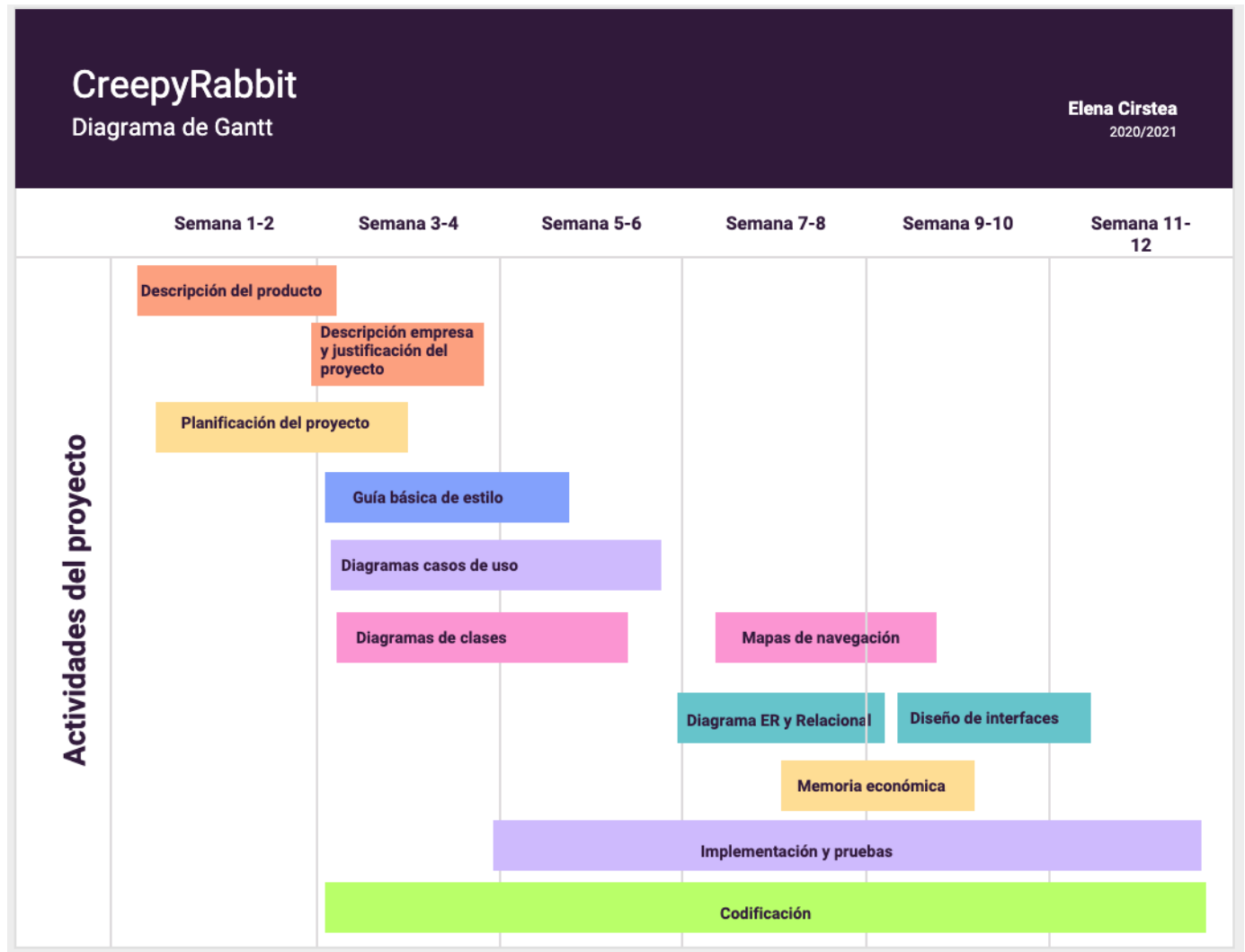


Figura 1.2: diagrama de Gantt

También se ha utilizado otra herramienta de gestión de proyectos llamada [Jira Software](#). Esta herramienta permite la creación de un tablero en el cual se pueden añadir tarjetas que recogen distintas tareas y se pueden mover las tarjetas según sus estados. En la *Figura 1.3* se muestra un ejemplo de cómo se emplea esta herramienta para el proyecto.

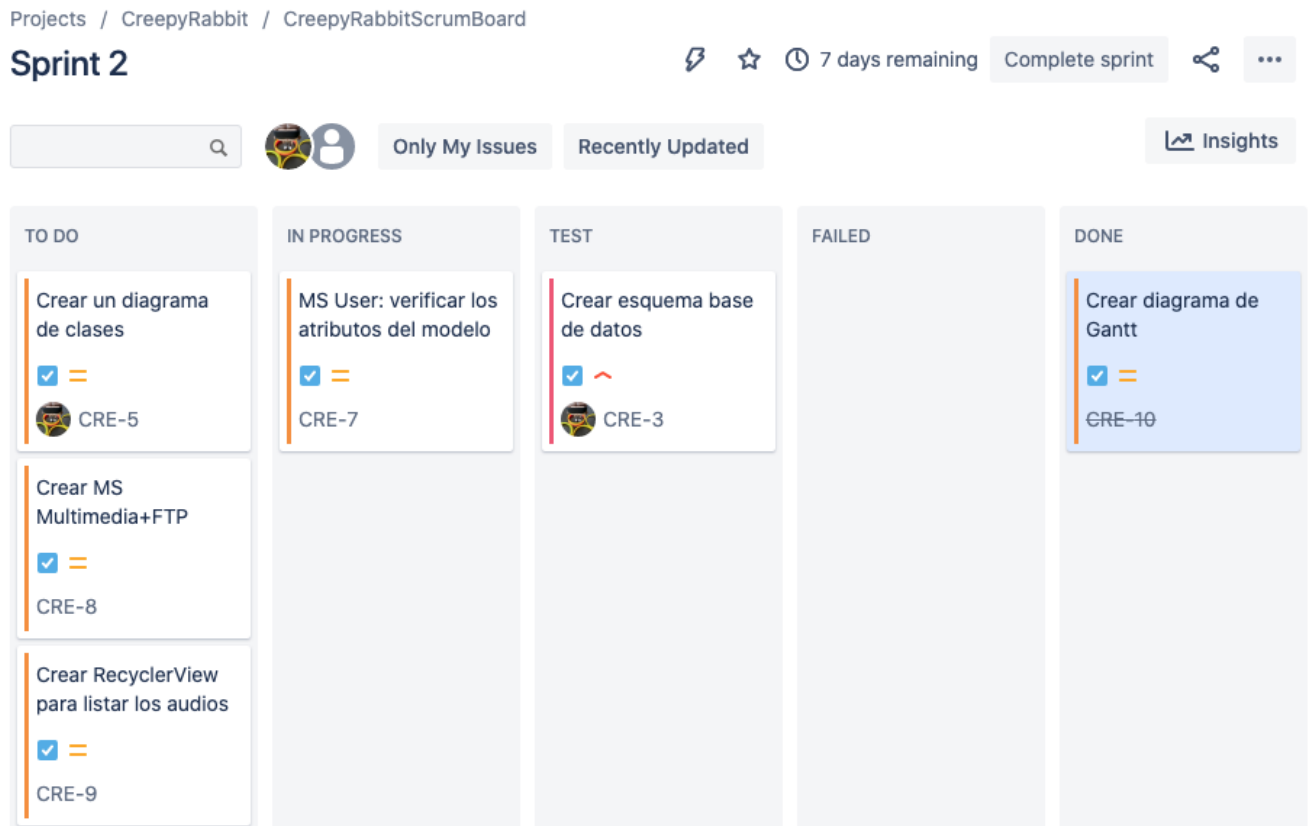


Figura 1.3: herramienta Jira para la organización de las tareas

ANÁLISIS Y DISEÑO

Diagrama de casos de uso

Los diagramas de casos de uso representan cómo interactúan los diferentes actores (roles que tiene un usuario) en un sistema para cada caso de uso. Es decir, definen qué acciones puede realizar cada actor dentro de un sistema. Los diagramas de casos de uso documentan el comportamiento de un sistema desde el punto de vista del usuario. Por lo tanto, los casos de uso determinan los requisitos funcionales del sistema.

En la *Figura 2.1* se puede observar que un usuario, al iniciar la aplicación móvil, tiene la posibilidad de registrarse, es decir, crear una cuenta de usuario a partir de la información que es requerida para el proceso. Una vez que está registrado, podrá autenticarse mediante el inicio de la sesión y las acciones que están relacionadas con su cuenta de usuario son las siguientes: deshabilitar su cuenta mediante baja, actualizar información de perfil, actualizar preferencias o enviar una queja o sugerencia.

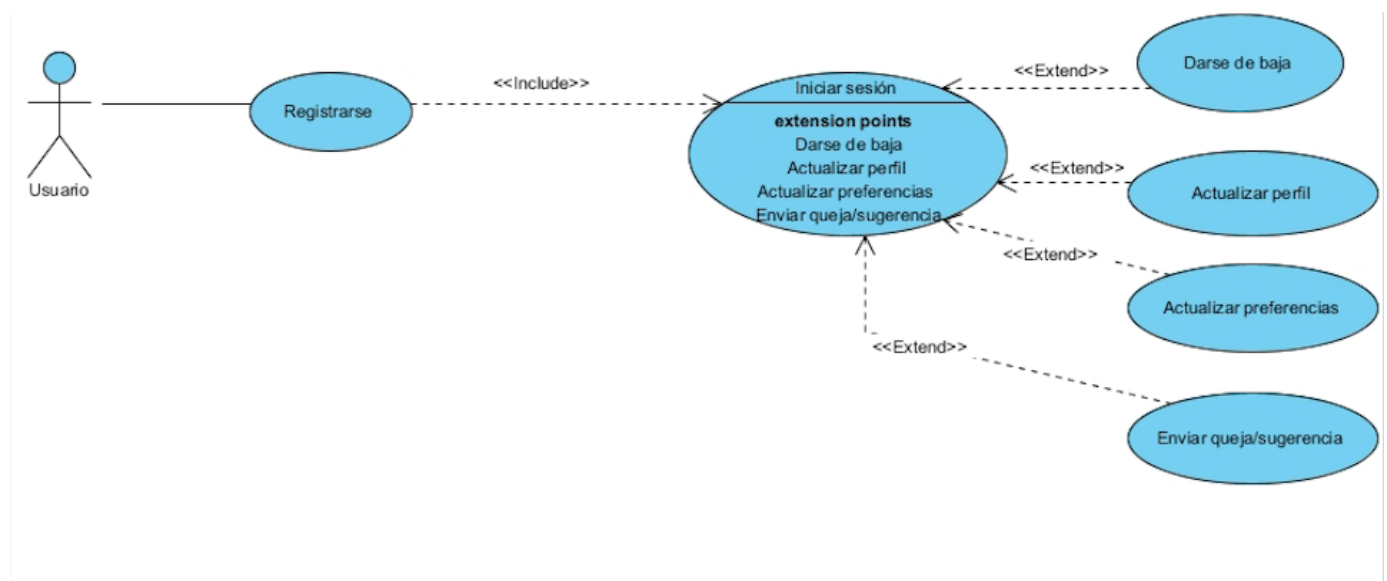


Figura 2.1: casos de uso para registro e inicio de sesión

La *Figura 2.2* representa las acciones que un usuario autenticado en el sistema puede llevar a cabo en la aplicación. En primer lugar, el usuario podrá listar y escuchar los audios listados previamente. Al igual, este listado se puede realizar por diferentes criterios. En segundo lugar, está la posibilidad de crear una lista de audios según las preferencias del usuario y/o modificar la mencionada lista.

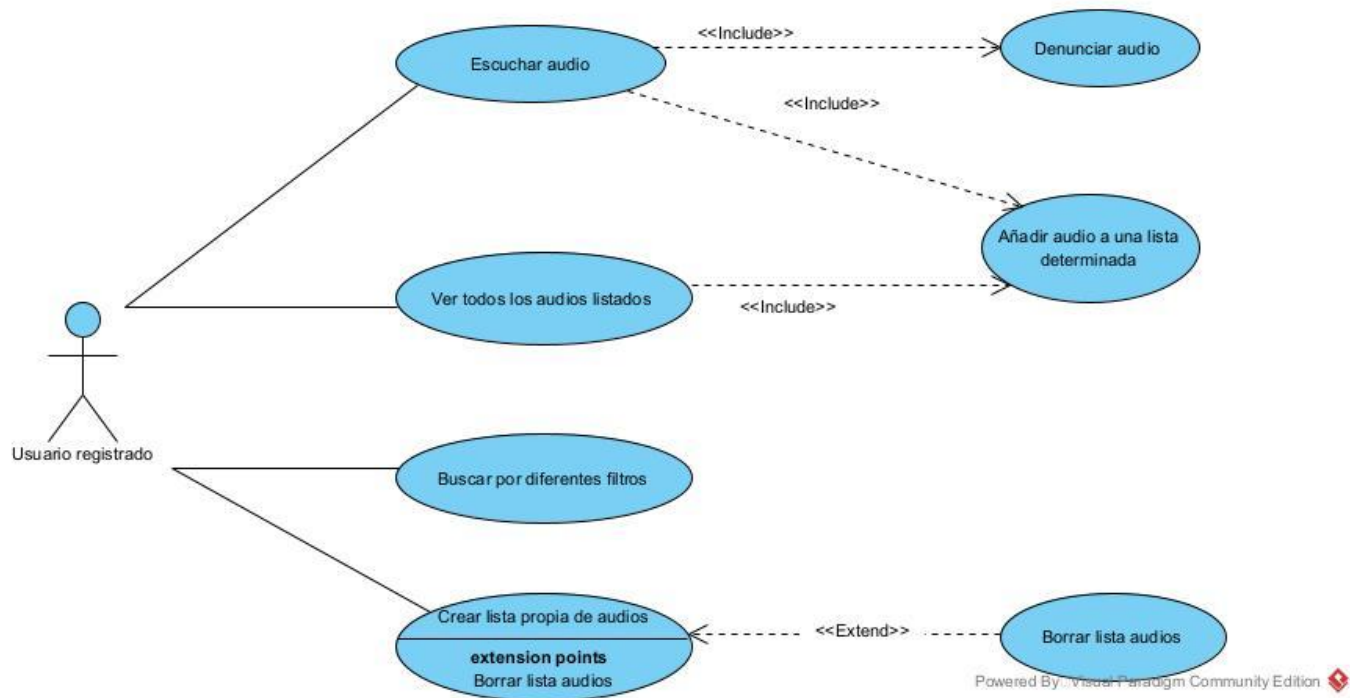


Figura 2.2: acciones que puede realizar un usuario registrado

Por último, se pueden mencionar las siguientes conclusiones sobre los diagramas de casos de uso:

- Cada caso de uso está relacionado, como mínimo, con un actor.
- Cada caso de uso es iniciado por un actor.
- Cada caso de uso lleva a un resultado relevante.

Diagrama de clases

Un diagrama de clases en Lenguaje Unificado de Modelado (UML) es un tipo de diagrama de estructura estática que muestra la organización de un sistema mostrando las clases del sistema, sus atributos, operaciones (o métodos), y las relaciones entre los objetos. No obstante, el presente proyecto trabaja con bases de datos no relacionales (NoSQL), por lo tanto la representación a través de diagramas de clases requiere unos ajustes.

Shin et al. (2017) presentan la siguiente propuesta de mapeo para poder realizar una representación de los diagramas de clases:

Diagrama de clase UML	Modelo de datos para documentos
Clase	Colección
Atributo	Columna en el documento
Asociación	Referencias incrustadas

La mayoría de los sistemas NoSQL utilizan el formato JSON (JavaScript Object Notation) para representar la información que almacenan. Un objeto JSON está formado por un conjunto de pares clave-valor. El tipo de un valor JSON puede ser de un tipo primitivo (Number, String, Boolean), un objeto, un array de valores o null para indicar que no existe valor para una clave. Los documentos JSON no conforman a un esquema sino que incluyen tanto los datos como su estructura.

En la *Figura 2.3* se muestra un diagrama de clases que abarca 3 colecciones: *Usuario*, *Favoritos* y *Multimedia*. Cada clase tiene una serie de acciones que se pueden llevar a cabo.

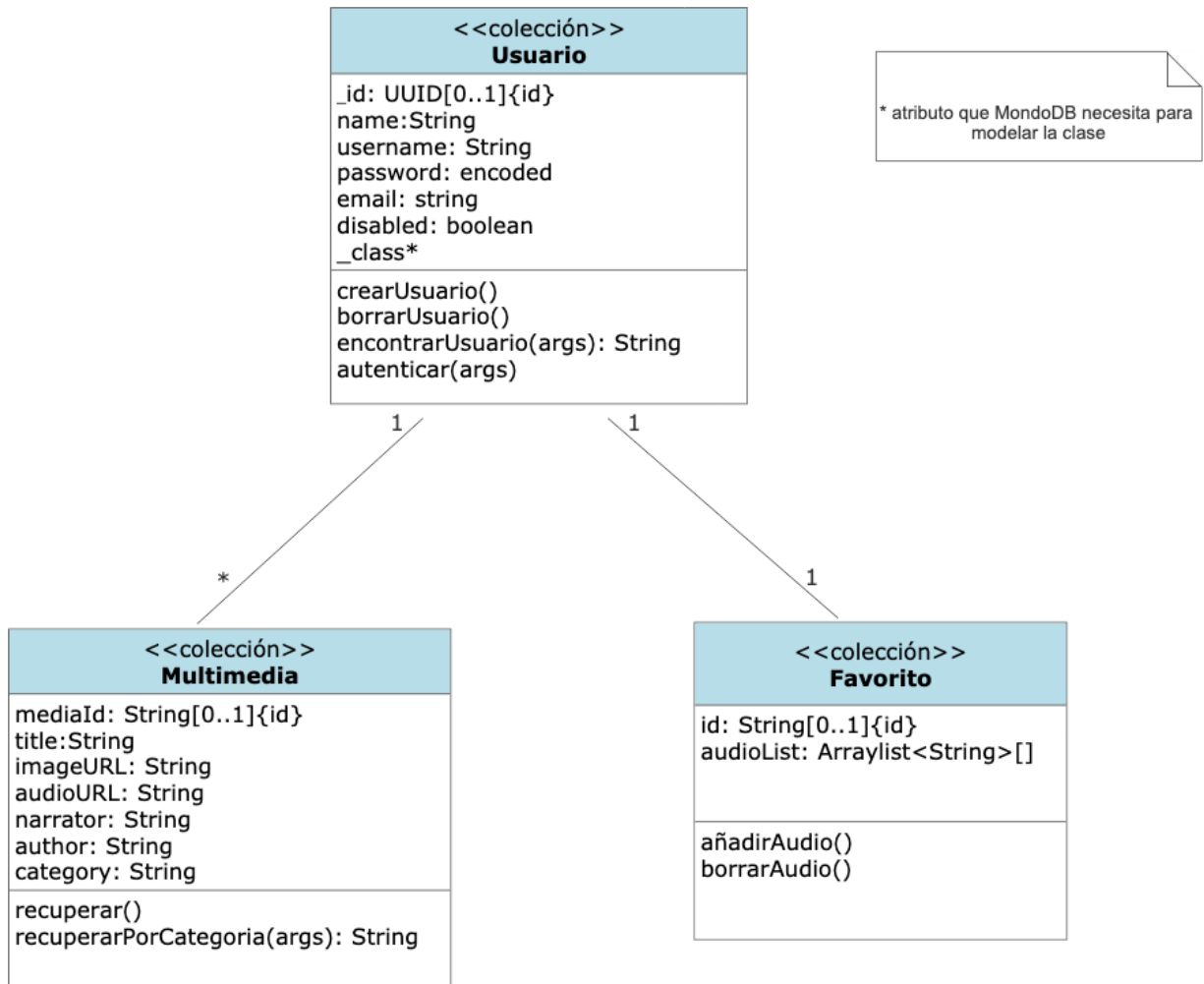


Figura 2.3: diagrama de clases

Diagrama de secuencia de aplicación

Para poder ofrecer una mejor visión del funcionamiento de la aplicación, se ha optado también por una representación mediante un tipo de diagrama de interacción. Los diagramas de interacción (interaction diagrams) pertenecen a la categoría de diagramas de comportamiento del lenguaje unificado de modelado. El tipo de diagrama escogido es el diagrama de secuencia, ya que describe el comportamiento dinámico del sistema de información, haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos.

Con el objetivo de conseguir una representación lo más clara posible, se ha dividido la actividad en varios diagramas, que se exponen a continuación. En la *Figura 2.4* aparecen representados los eventos de entrar en el sistema con unas credenciales. En primer caso, el actor introduce unas credenciales (usuario y contraseña). Estos datos llegan a la instancia de *Retrofit* y esta misma hace una petición *POST* al microservicio de usuarios mediante una API REST que el propio microservicio expone.

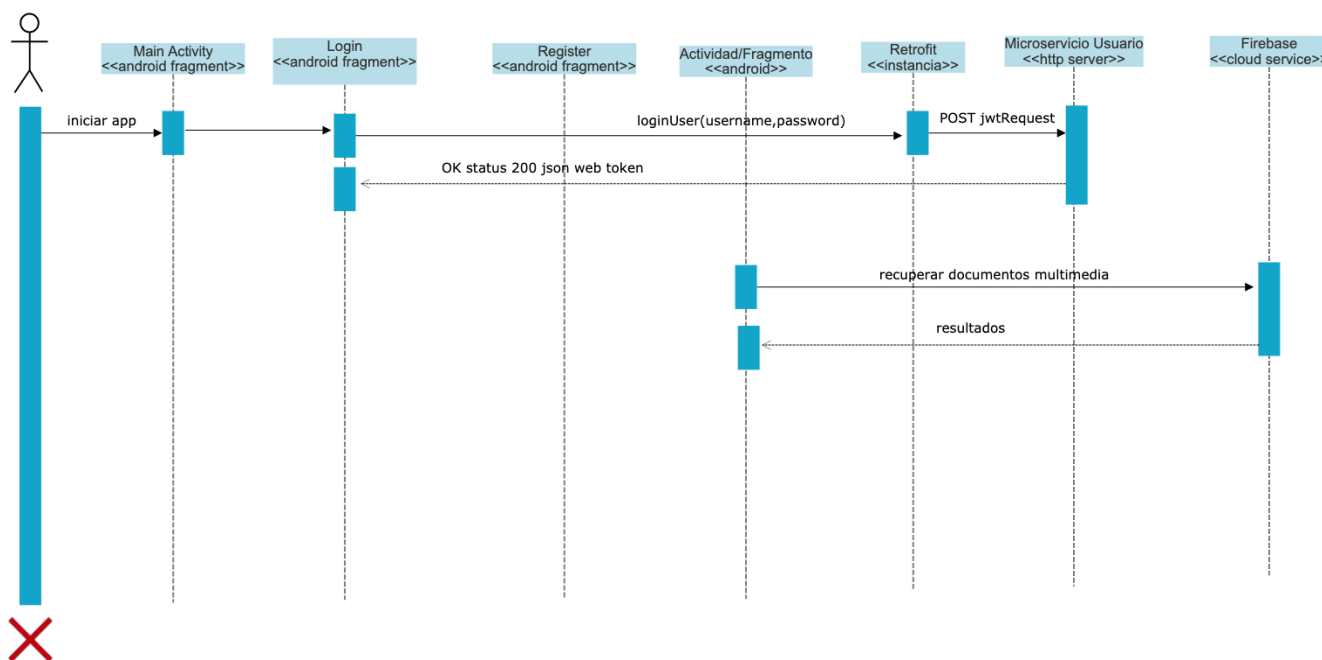


Figura 2.4: diagrama de secuencia de aplicación

A continuación, el microservicio devuelve una respuesta, en este caso un JSON WEB TOKEN, aunque también podría devolver una respuesta distinta en el caso de que las credenciales no sean correctas.

En el segundo caso, se hace una petición al servicio de *Firestore*, del cual se recuperan unos registros específicos según unos criterios concretos (véase el apartado 3 para más detalles). En los dos casos, las peticiones serán asíncronas.

Mapa de navegación de la aplicación

Se puede definir un mapa de navegación como una representación gráfica de la organización de la información de una estructura. Esta representación puede ser completa o resumida y está dirigida a orientar al usuario durante el recorrido de la aplicación o para facilitarle un acceso directo a un contenido que le interese de primera mano.

En la *Figura 2.5* aparece reflejado el mapa de navegación de la aplicación. Cuando se lanza el programa, el usuario puede entrar con sus credenciales o registrarse. Una vez esté autenticado, tiene la posibilidad de listar todos los audios o listarlos por categoría. Otra opción que dispone es editar su perfil o ponerse en contacto con los desarrolladores.

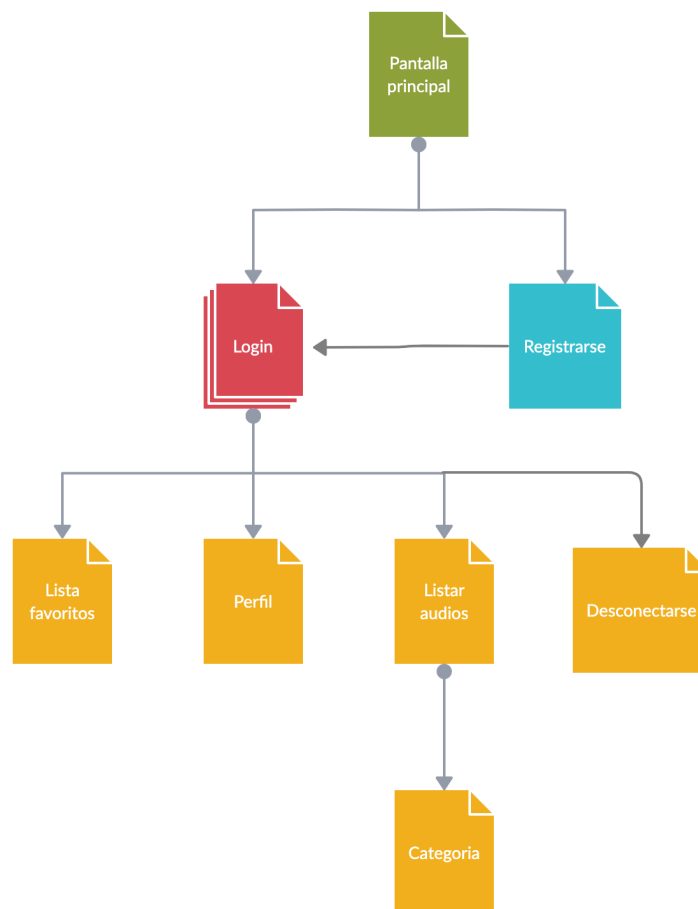


Figura 2.5: mapa de navegación de la aplicación

Diagrama de E-R/relacional y scripts

El modelo Entidad-Relación es un proceso de diseño de la base de datos en el cual se obtiene el esquema conceptual. En este modelo se definen todos los datos del problema y sus relaciones. El modelo conceptual más conocido es el modelo E/R propuesto por P. Chen (1976).

Es un modelo orientado a conceptos que no toma en cuenta la estructura final de los datos sino los objetos que se representan y las relaciones entre ellos. Se suele realizar como paso previo al modelo lógico, es decir, primero se realiza el modelo entidad relación para representar la realidad que se quiere transformar en una base de datos y a partir de él se pasará al modelo lógico. Por último, al tratarse de un modelo conceptual, no está orientado a ningún sistema físico concreto.

En la *Figura 2.6* se puede observar el diagrama de Entidad-Relación del proyecto. Los rectángulos representan las entidades, mientras que los rombos son las relaciones que unen las entidades con otras. En las elipses se especifican los atributos de las entidades. En cada relación hay una cardinalidad que define el tipo de relación que tienen las entidades entre ellas: uno a uno(1:1), uno a muchos(1:N) y muchos a muchos(N:M).

Las entidades, en este caso, serían las de USUARIO y AUDIO. La entidad usuario tendría asociada a su vez una entidad débil (LISTA FAVORITOS).

Tal como se ha ido mencionando, el presente proyecto está diseñado para trabajar con NoSQL, por lo tanto, la creación de las distintas bases de datos se lleva a cabo directamente a través del código, sin necesidad de un *script* concreto para la creación.

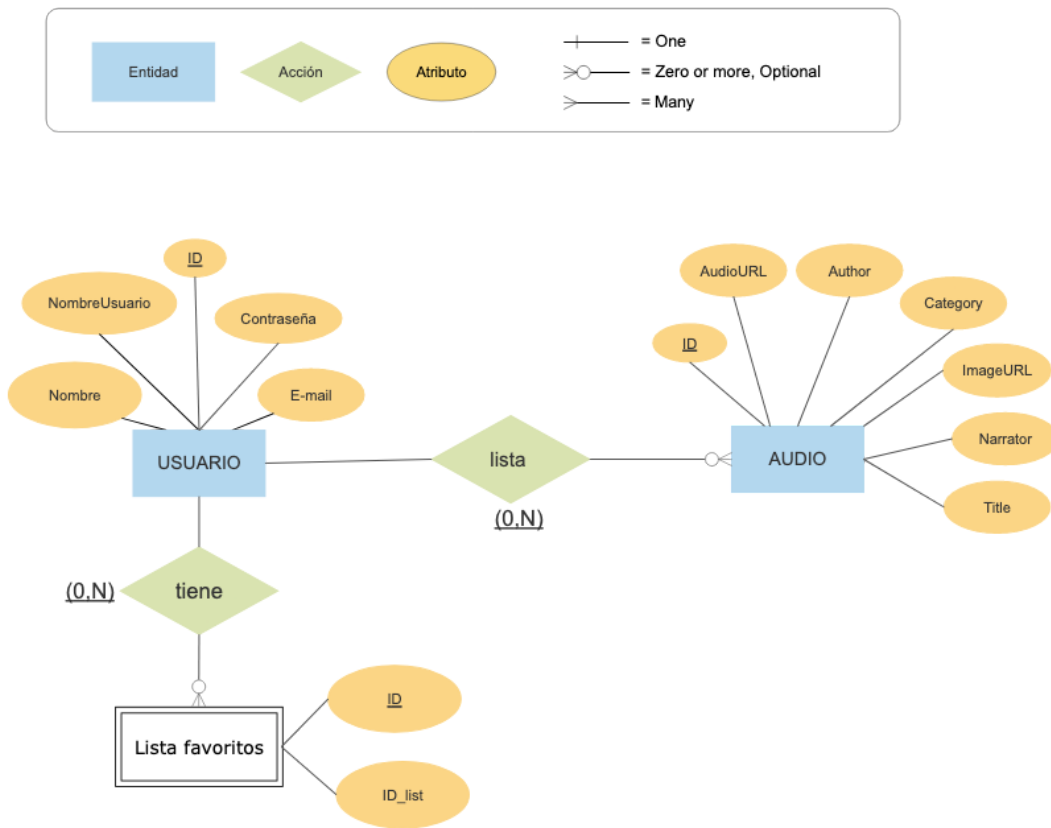
Diagrama Entidad-Relación
CreepyRabbit

Figura 2.6: diagrama entidad-relación

En el proyecto no hay la necesidad de creación de unos *scripts* específicos, ya que el microservicio hace uso de un *framework* que posibilita la creación de documentos a través de anotaciones en el código. Un ejemplo de un documento de la base de datos es el siguiente:

```
{
  "_id" : UUID,
  "name" : String,
  "username" : String,
  "password" : Encoded password,
  "email" : String,
  "created" : ISODate("2021-08-31T09:16:20.548Z"),
  "_class" : "com.ecirstea.user.model.User"
}
```

Figura 2.7: ejemplo documento en MongoDB para la colección Usuario

Guía básica de estilos

Una guía básica de estilo es un documento que recoge las directrices de los estilos que tendrá la aplicación móvil. Entre estas directrices se encuentran aspectos como el uso de los colores mediante una paleta ya establecida, las fuentes de texto que se utilizarán, el logotipo y sus variaciones, el tipo de maquetación y las imágenes, entre otros aspectos. Una paleta de colores está formada por un conjunto de colores que se combinan entre ellos de una manera armónica. Sus valores se representan en distintos formatos, como pueden ser valores hexadecimales o en valores RGB.

En la *Figura 2.8* se puede observar la paleta a utilizar en el presente proyecto.

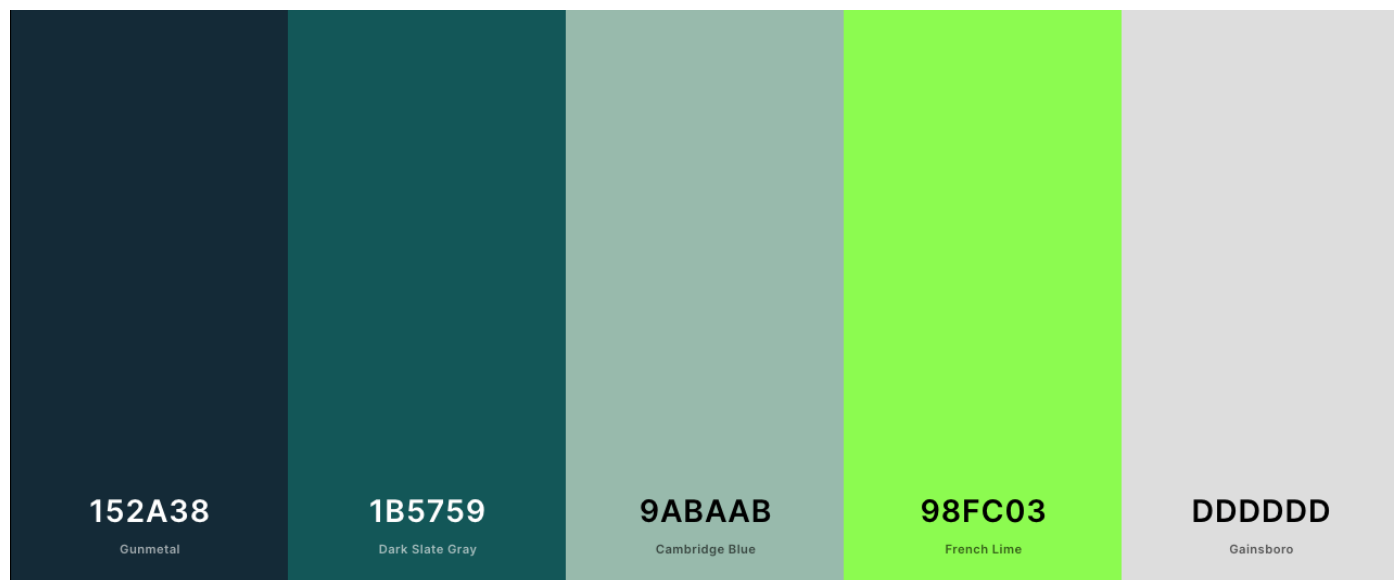


Figura 2.8: paleta de colores

En cuanto a la tipografía, se utilizarán las siguientes fuentes, tal cual se puede observar en las figuras siguientes:

[Poppins Medium:](#)

Medium 500

Almost before we knew it, we had left th

[Aldrich Regular:](#)

Regular 400

Almost before we knew it, we had left

[Montserrat:](#)

Light 300

Almost before we knew it, we had left th

La aplicación móvil tiene un logo que la define, de color negro y acorde con la temática. Su utilización debe darse con colores claros.



Figura 2.9: logo aplicación móvil

Otro aspecto a tener en cuenta a la hora de realizar una guía de estilo son los botones utilizados a lo largo de la aplicación móvil, que aplican el diseño de [Material Design para Android](#). En este caso, serían los siguientes:

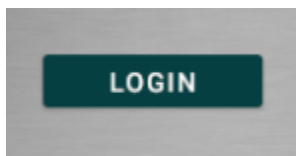


Figura 2.19: botón fondo oscuro



Figura 2.11: botón fondo claro

En resumen, la guía de estilo tiene como objetivo crear consistencia a lo largo de la aplicación móvil, unas directrices que puedan servir para futuros diseños y sobre todo mantener una coherencia en todo momento de la navegación.

Diseño de interfaces

A la hora de diseñar una aplicación móvil, se realiza una guía visual o esquema de las pantallas que representan el esqueleto de la aplicación en sí. A esta guía se le conoce con el nombre de *wireframe*. El objetivo del *wireframe* es mostrar el ordenamiento de una aplicación de manera esquemática, incluyendo elementos de la interfaz y sistemas de navegación. A continuación se muestran las pantallas en formato de *wireframe*.

En la *Figura 2.12*, la primera pantalla representa el punto de acceso a la aplicación: el usuario ya registrado puede introducir su nombre de usuario junto a su contraseña o tiene la opción de registrarse. El botón *Sign in* permite el acceso a la aplicación, mientras que el botón *Continue* guarda el usuario en el sistema.

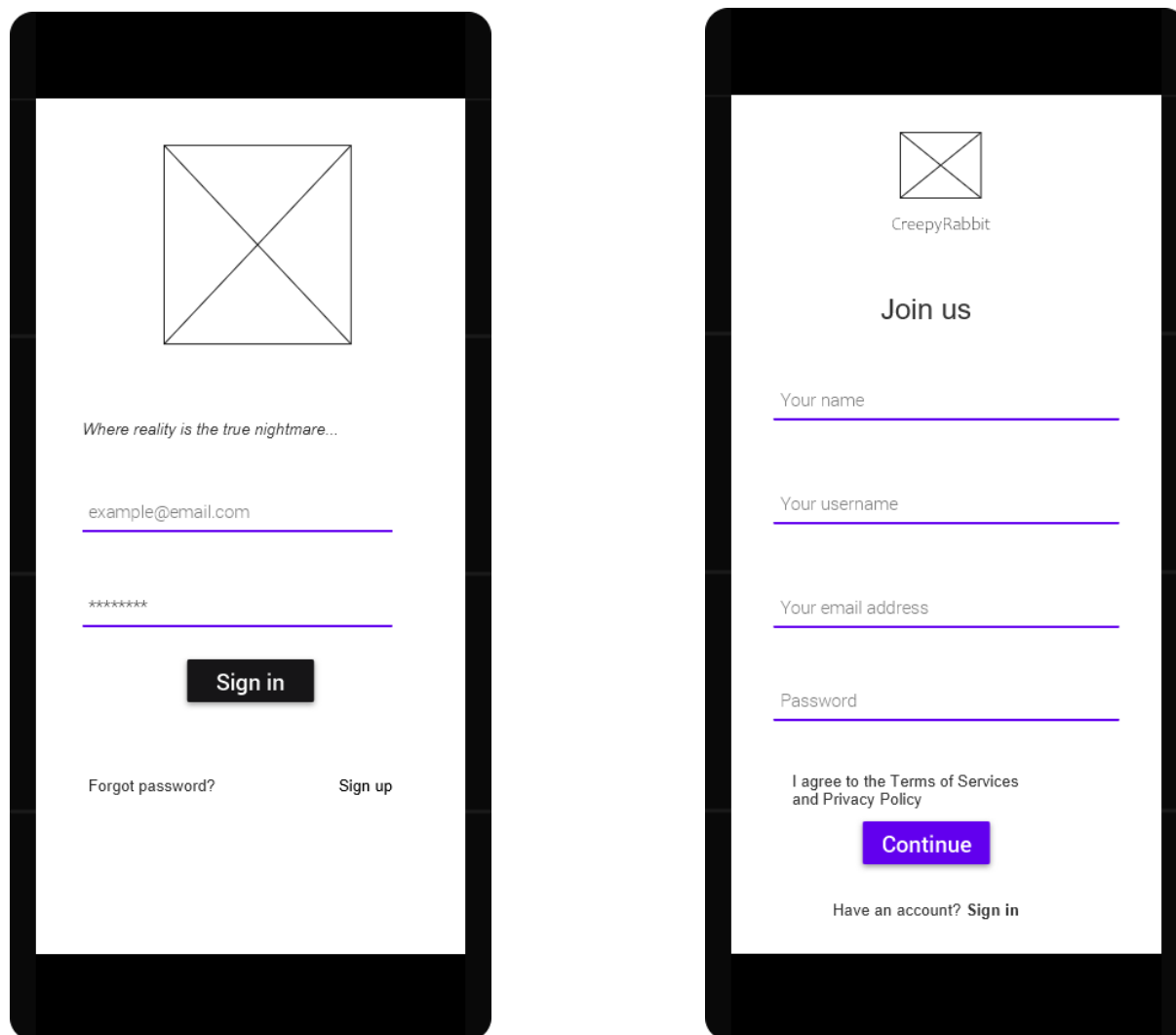


Figura 2.12: pantallas de inicio de sesión y registro de usuarios

En la *Figura 2.13*, en la primera pantalla (izquierda) aparece desplegada la barra de navegación, situada en la parte superior de la app para mostrar información y acciones de la pantalla actual. En la segunda pantalla (derecha) aparece la pantalla principal con un botón que llevaría directamente al listado de audios marcados como *Favoritos*.

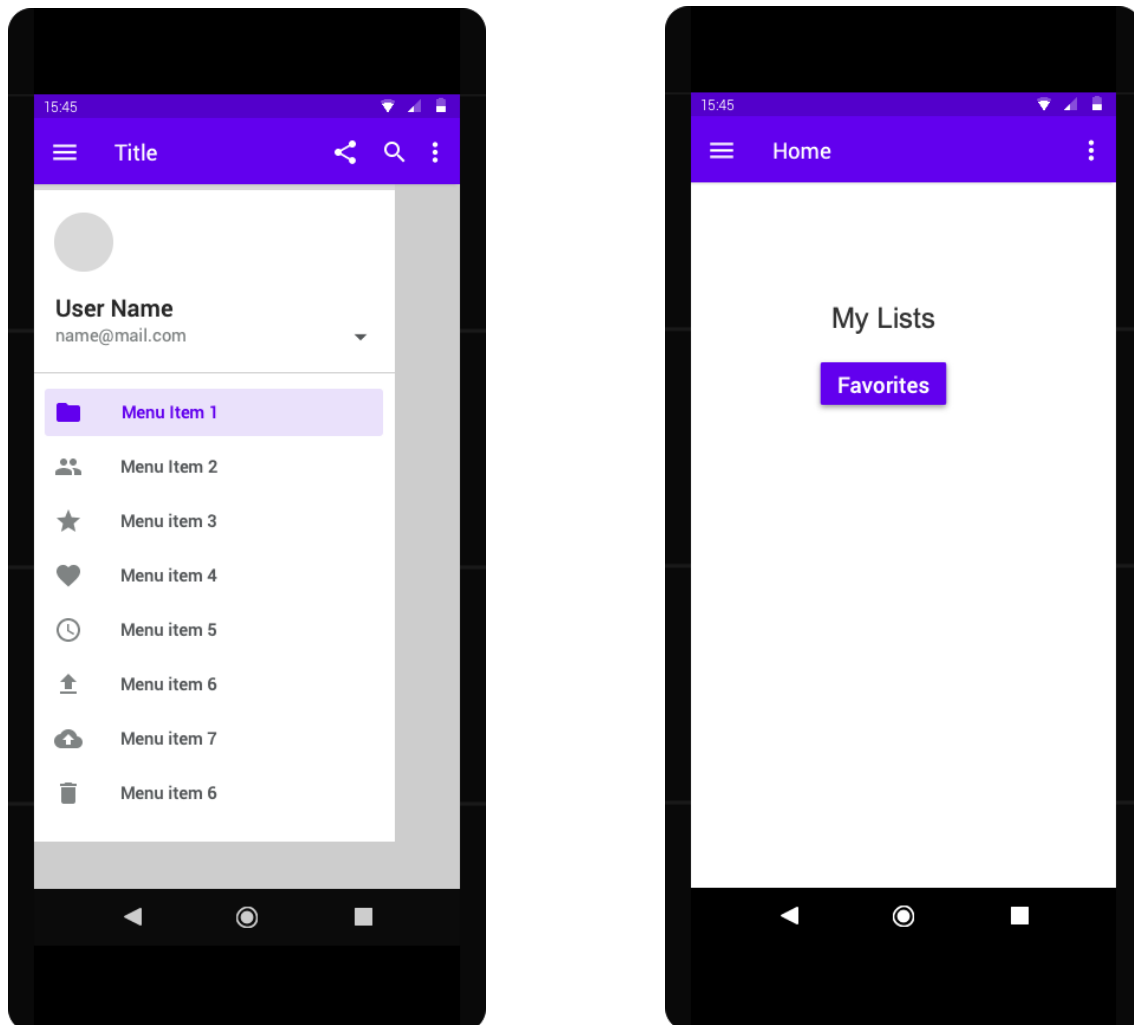


Figura 2.13: pantalla para la navegación y la página *home*

En la *Figura 2.14* se observa en la parte izquierda la pantalla para la modificación/actualización de la información de usuario. El botón **Save** permite modificar estos datos en la base de datos. En la pantalla derecha aparece el formulario de contacto a través del cual el usuario puede ponerse en contacto con los desarrolladores para cualquier inquietud o queja que tenga.

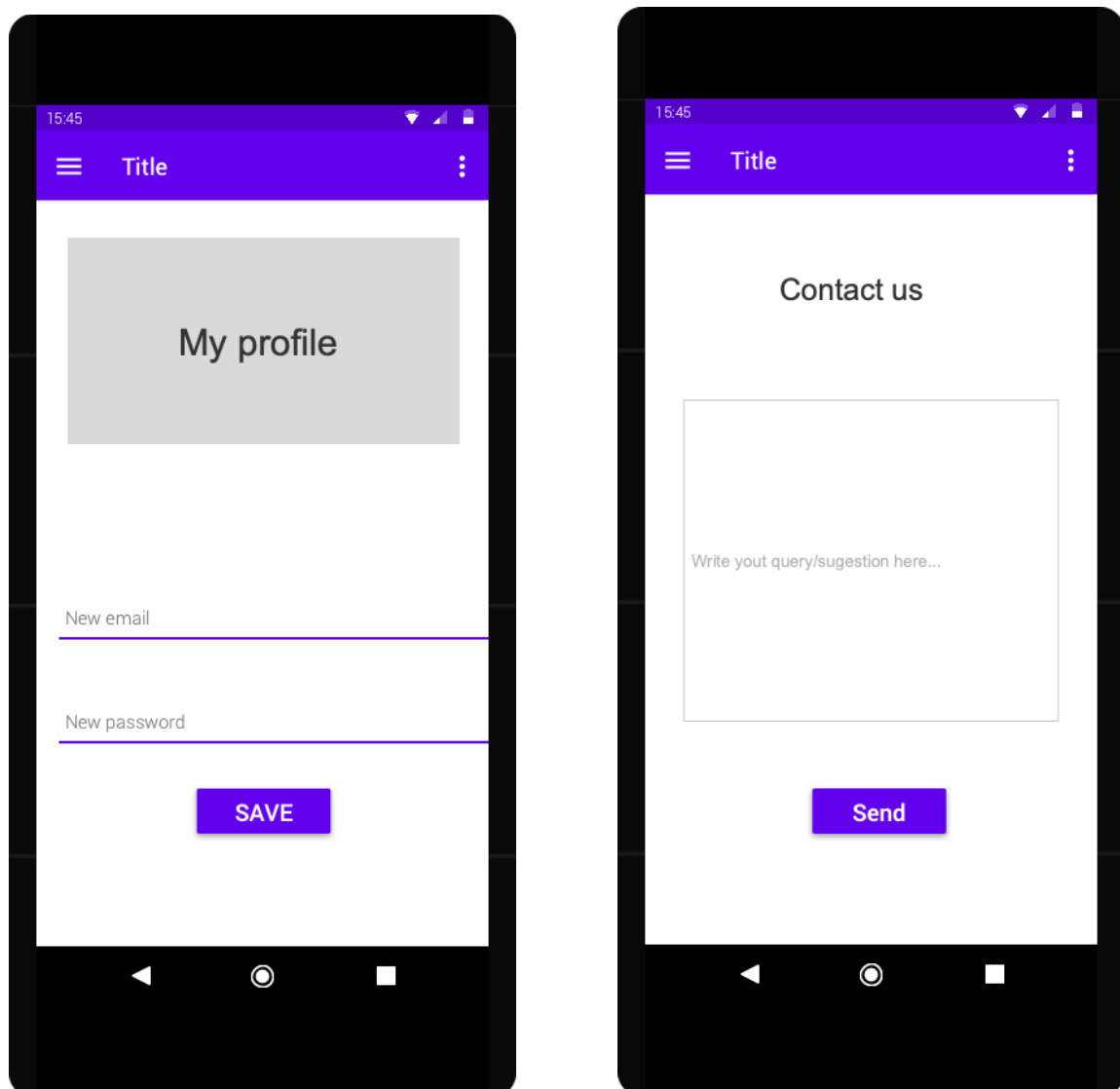


Figura 2.14: pantalla de modificación información del usuario y formulario de contacto

En la *Figura 2.15* aparece en la izquierda el listado de los audios disponibles en la aplicación. En la parte derecha el detalle de reproducción de un audio. El botón *Play* permite la reproducción del audio y *Pause* pondría el audio en pausa. Al igual, hay una barra que muestra el estado de la reproducción junto al tiempo que ha transcurrido y el total.

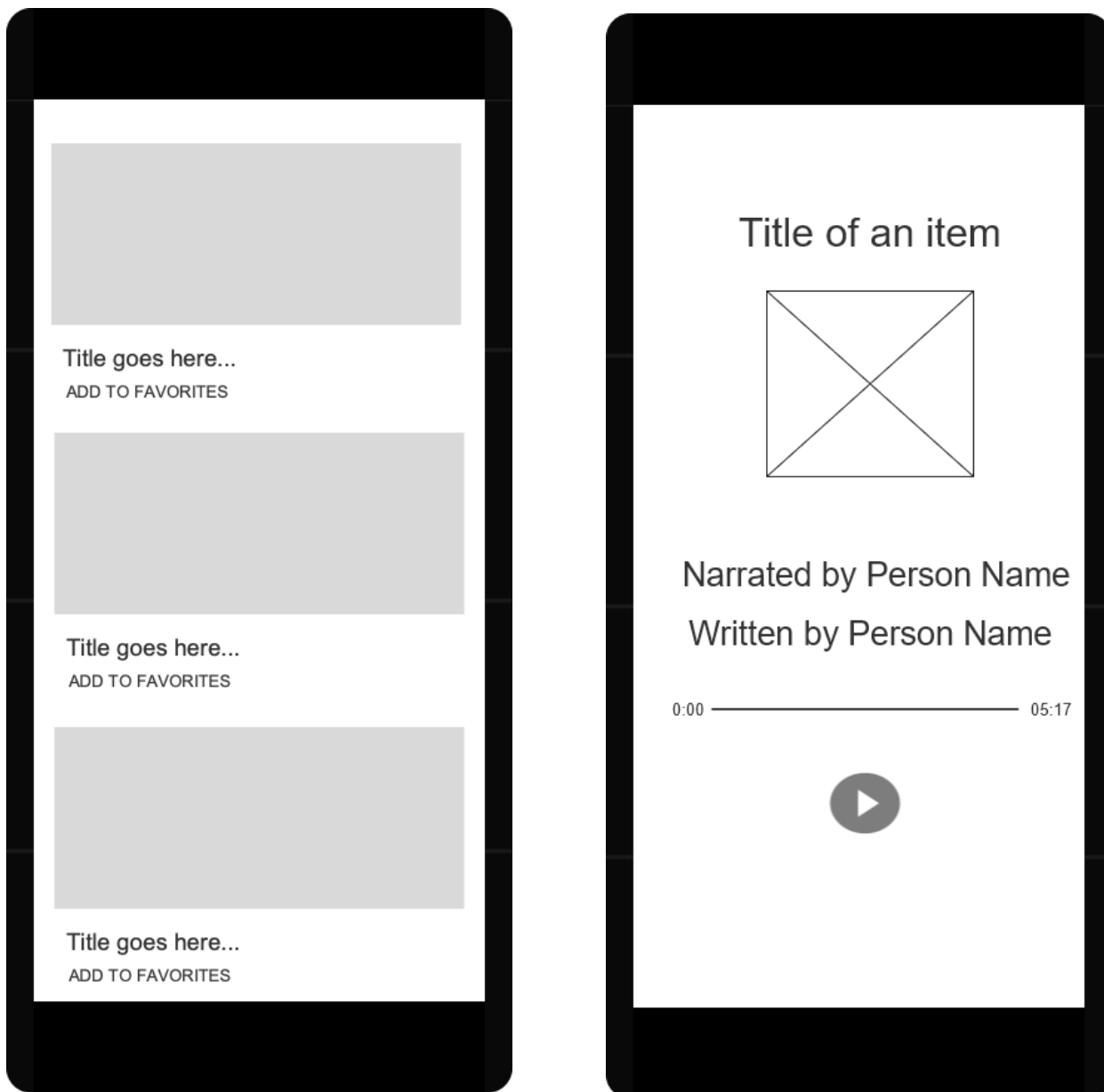


Figura 2.15: pantalla de listado de los audios y detalle de reproducción de un audio

CONFIGURACIÓN Y DESARROLLO DEL SOFTWARE

Desglose de la tecnología utilizada

Hardware:

- Equipo: MacBook Pro 13-inch

Procesador: 2,3 GHz Dual-Core Intel Core i5

Memoria: 16GB MHz DDR3

Graficos: Intel HD Graphics 3000 512MB

- Recursos humanos: un diseñador/programador

Tecnologías:

Para el desarrollo del presente proyecto se han adoptado varias tecnologías según muestra la *Figura 3.1* y que se va a detallar a continuación:

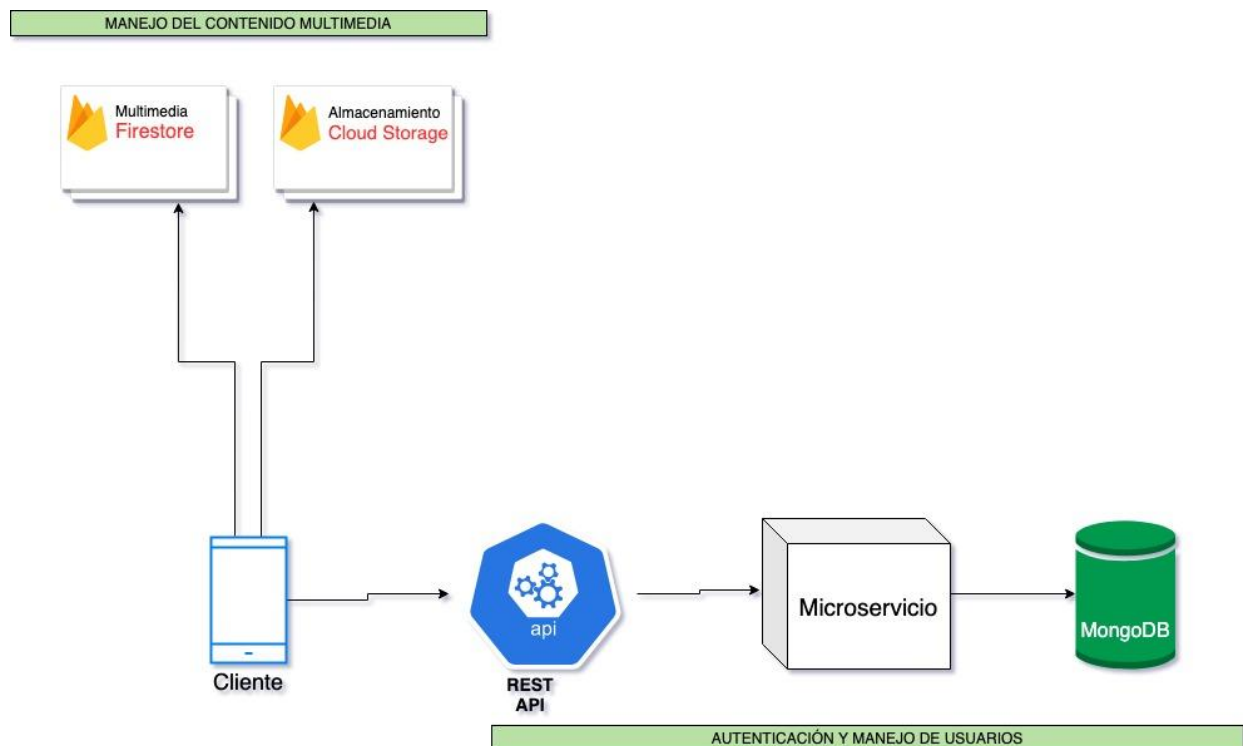


Figura 3.1: arquitectura del proyecto

En primer lugar, la parte cliente se ha desarrollado como una aplicación para el sistema operativo Android, utilizando como lenguaje de programación [Kotlin](#). Es un lenguaje tipado estático que corre sobre la máquina virtual de Java. La aplicación tiene como objetivo SDK mínimo la API 26 y utiliza [Gradle](#).

Gradle es un paquete de herramientas de compilación avanzadas, para automatizar y administrar el proceso de compilación y, al mismo tiempo, definir configuraciones de compilación personalizadas y flexibles, como por ejemplo las dependencias requeridas por el proyecto.

En segundo lugar, el cliente interactúa con dos fuentes de datos distintas: por una parte un microservicio y por otra con una instancia de Firebase.

MICROSERVICIO - USUARIOS: *Spring Boot*

Para el manejo de todos los datos relacionados con la información de los usuarios, incluida la autenticación, se ha creado un microservicio con el módulo del *framework* Spring llamado **Spring Boot**. Este módulo sigue el camino del paradigma de software llamado «convención sobre programación¹». Es una infraestructura ligera que elimina la mayor parte de las tareas que se encargan de configurar las aplicaciones basadas en Spring. El microservicio es un servidor sin estado que recibe unas peticiones a unos *endpoints* predeterminados

CLOUD FIRESTORE: base de datos para los datos multimedia

Cloud Firestore es una base de datos NoSQL, flexible y escalable para el desarrollo en dispositivos móviles, entre otros. Destaca porque se pueden usar consultas para recuperar documentos individuales específicos o para recuperar todos los documentos de una colección que coinciden con los parámetros de la consulta.

CLOUD STORAGE: almacenamiento para fotos y audios

Cloud Storage para Firebase es un servicio de almacenamiento de objetos potente, simple y rentable.

¹ https://es.wikipedia.org/wiki/Convenci%C3%B3n_sobre_configuraci%C3%B3n

Implementación

Creación e implementación de la aplicación móvil con Kotlin

La aplicación móvil (frontend) se ha desarrollado utilizando como entorno de desarrollo Android Studio 4.2.2 con la siguiente plataforma de Java: VM: OpenJDK 64-Bit Server VM.

Para poder realizar las llamadas al microservicio de usuarios, se ha utilizado una librería para Android llamada Retrofit que permite hacer las llamadas de red, obtener el resultado y convertirlo de forma automática a un objeto de tipo concreto, facilitando de esta manera la realización de peticiones a un API y procesar la respuesta.

Por otra parte, para poder acceder a los datos de Firestore y FireCloud, se ha tenido que implementar lo siguiente:

- Se vincula el proyecto en Android a Firestore mediante la consola
- Se añaden las siguientes dependencias de Gradle en el proyecto de Android:

```
implementation platform('com.google.firebase:firebase-bom:28.3.0')
```

```
implementation 'com.google.firebase:firebase-firestore-ktx'
```

- Inicializar la instancia de Firestore a través de código

Creación e implementación del microservicio con Spring Boot y Java:

Como entorno de desarrollo, se ha optado por IntelliJ IDEA 2021.1.3 (Community Edition) y con la siguiente plataforma de Java: VM: OpenJDK 64-Bit Server VM.

Spring ofrece una modalidad de crear un esqueleto de un proyecto de forma rápida. Accediendo al enlace <https://start.spring.io/> se pueden configurar los datos necesarios para poder generar un proyecto a partir del cual se puede empezar a codificar (*Figura 3.2*). Una vez que se genera la plantilla, se abrirá con el IDE y se podrá desplegar.



Project
☐ Maven Project ☒ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (M2) ☐ 2.5.5 (SNAPSHOT) ☒ 2.5.4
☐ 2.4.11 (SNAPSHOT) ☐ 2.4.10

Project Metadata

Group

com.ecirstea.user

Artifact

Microservice-users

Name

Microservice-users

Description

Microservice for user management

Package name

com.ecirstea.user.Microservice-users

Packaging

☒ Jar ☐ War

Java

☐ 16 ☒ 11 ☐ 8

Figura 3.2: generación de un proyecto esqueleto con Spring Boot

Para conectar la parte del cliente con el microservicio, se emplea un servicio REST. Representational State Transfer (REST) es una interfaz para conectar varios sistemas basados en el protocolo HTTP que permite obtener y generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como puede ser JSON, en este caso. REST se apoya en el protocolo HTTP, por lo tanto los verbos que utiliza son exactamente los mismos, permitiendo realizar peticiones GET, POST, PUT y DELETE.

REST permite una gran serie de ventajas, entre ellas destacan la separación de parte cliente del servidor. Permite el diseño de un microservicio orientado a un dominio (DDD), es totalmente independiente de la plataforma, así que se puede hacer uso de REST tanto en Windows, Linux o Mac. Al igual, la API puede hacerse pública si se requiere conseguir más visibilidad. En este caso, la API está expuesta a través de [Swagger UI](#) (Figura 3.3).

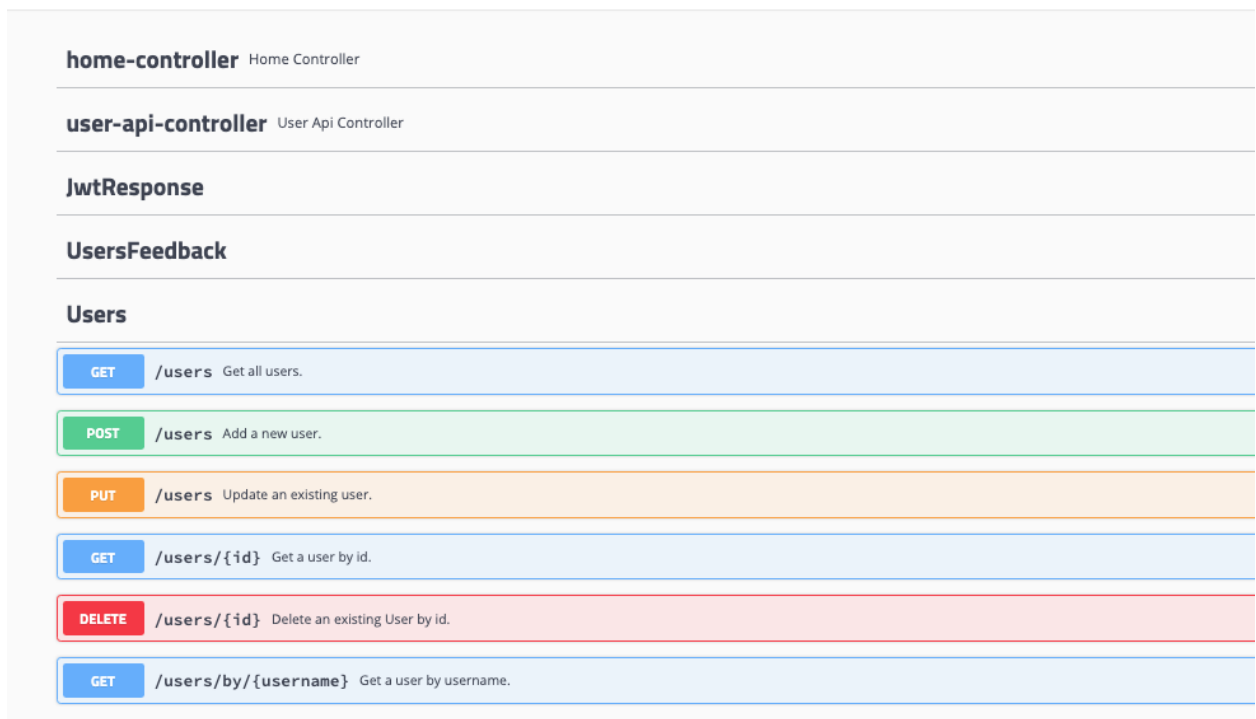
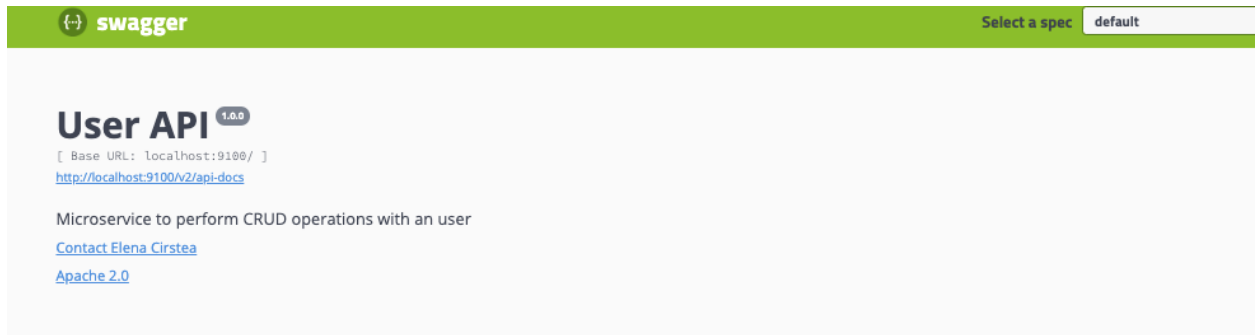
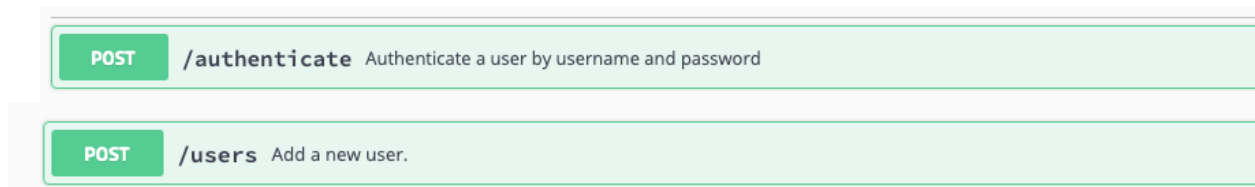


Figura 3.3: endpoints expuestos para el microservicio de usuarios

El microservicio expone las siguientes llamadas para la autenticación y creación de usuarios. Estas llamadas no requieren ningún tipo de autorización.



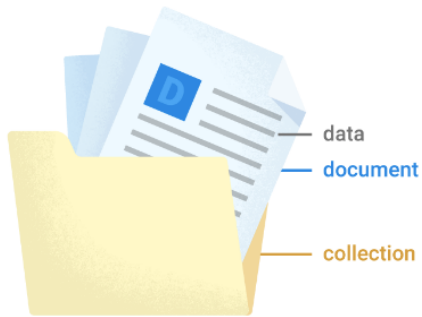
El resto de las peticiones requieren un [Json Web Token](#) para la autorización de acceder al recurso. Un JSON Web Token es un *token* de acceso estandarizado en el [RFC 7519](#) que permite el intercambio seguro de datos entre dos partes. Está compuesto por las siguientes partes:

HEADER.PAYLOAD.SIGNATURE



Fuente imagen: <https://openwebinars.net/blog/que-es-json-web-token-y-como-functiona/>

El microservicio implementa su propia base de datos, en este caso es una base de datos NoSQL. Las bases de datos no relacionales guardan estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida. El sistema de base de datos no relacional elegido para el microservicio es [MongoDB](#). MongoDB trabaja con colecciones de documentos.



Fuente imagen: https://firebase.google.com/docs/firestore#how_does_it_work

Para poder hacer uso de MongoDB en el microservicio, se ha optado por su implementación a través de Spring Data MongoDB, ya que facilita mucho el proceso de trabajo con entidades de datos.

En este proyecto, la entidad modelo es la clase Java de *User*. La anotación **@Document** indica a Spring Data MongoDB que será una colección en la base de datos. La anotación **@Id** indica a Spring que este campo debería de emplearse como el identificador único del documento/entidad. Para el manejo de datos se ha creado un repositorio que realiza todas las operaciones necesarias.

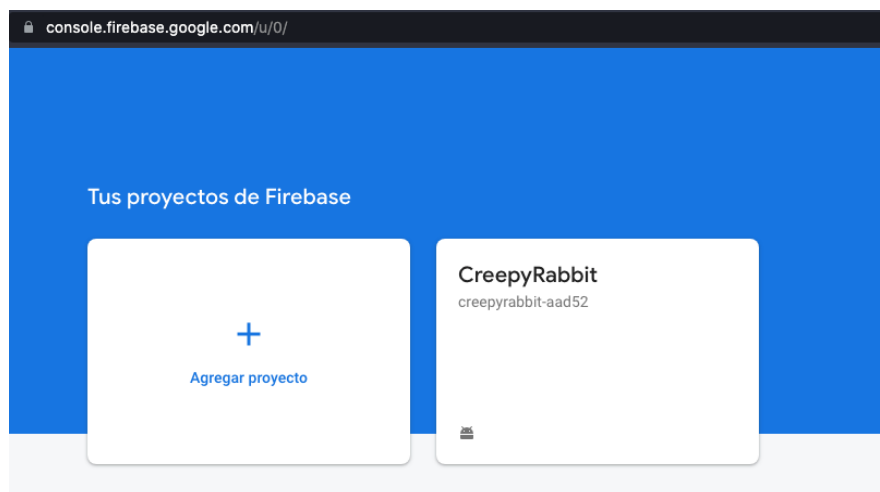
```
@Document(collection = "User")
public class User {
    @Id
    @Indexed(unique = true)
    private UUID id;

    @NotBlank(message = "Name is mandatory")
    @ApiModelProperty(position = 1)
    private String name;
```

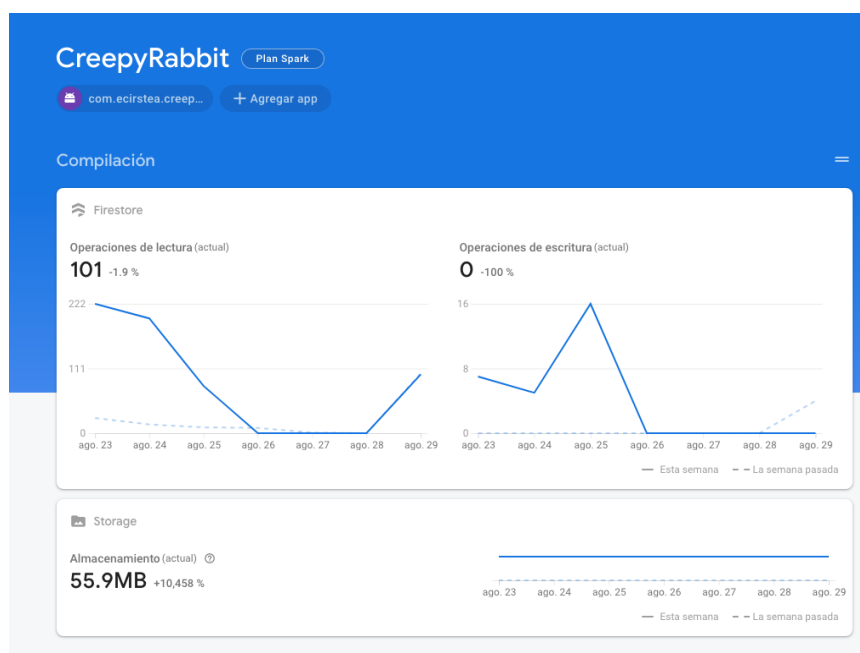
Figura 3.4: clase de Java para la creación de una entidad en la base de datos

Consola de Firestore

La consola de Firebase es una suite de herramientas integrada con servicios de Google. Proporciona una plataforma de administración para Android. Para poder hacer uso de ella, se tiene que tener el proyecto vinculado en la consola, tal como se muestra a continuación:



Una vez que el proyecto está vinculado, la consola ofrece acceso a todos los datos necesarios, como por ejemplo el uso de datos que hace la aplicación o el tamaño de almacenamiento.



Pruebas

El testing de software es una de las actividades más importantes y fundamentales en el desarrollo de un proyecto, ya que posibilita los procesos, métodos de trabajo y herramientas necesarias para garantizar la calidad de cualquier desarrollo. Hay varios tipos de pruebas de acuerdo a su ejecución: pruebas manuales y pruebas automatizadas de software.

En las pruebas manuales de software, el examinador ejecuta los casos de prueba sin ayuda de herramientas automáticas. El proceso suele incluir la verificación de todas las características especificadas en los documentos de requisitos. Asimismo, esto también implica probar el software teniendo en cuenta la perspectiva del usuario final. Este tipo de pruebas requiere de conocimiento y experiencia profunda, así como habilidades lógicas y analíticas.

En las pruebas de software, la automatización de pruebas consiste en el uso de software especial para controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados. En el proyecto se hacen dos tipos de pruebas de software: unitarias y de integración.

El objetivo de las pruebas unitarias es comprobar si los componentes se comportan de acuerdo a los requerimientos, mientras que en las pruebas de integración se testean los módulos en grupo.

En el presente proyecto se han llevado a cabo tanto pruebas manuales como automatizadas.

- **PRUEBAS MANUALES DE SOFTWARE**

Las pruebas manuales se han realizado sobre todo en la interfaz gráfica de la aplicación móvil. Se ha probado que los formularios de introducción de datos contemplan los casos donde el usuario introduce datos erróneos y que los botones cumplen con sus funciones.

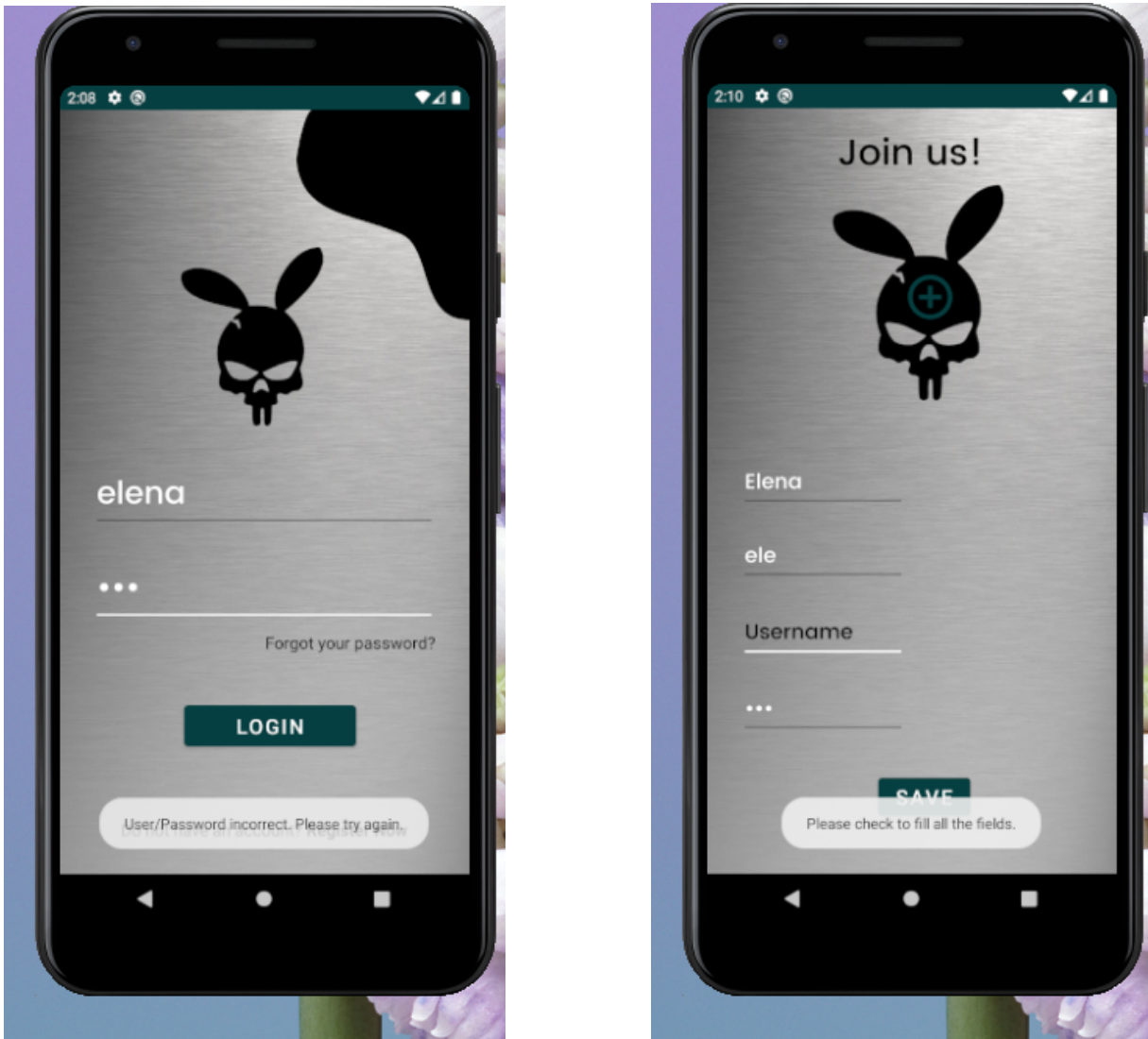


Figura 3.5: mensaje avisando al usuario que algún dato no es correcto

- PRUEBAS AUTOMATIZADAS DE SOFTWARE

PRUEBAS DE INTEGRACIÓN

En el caso del microservicio realizado con Spring Boot y codificado en Java, se ha optado por realizar pruebas de integración utilizando una clase de Spring *MVC Test Framework*, en concreto **MockMVC**. Se ha realizado un test para cada *endpoint* que expone la API y también se han realizado test complementarios como por ejemplo en el caso de que se intenta acceder a un recurso que no existe.

Los test se ejecutan dentro de un contenedor de **Docker**, creando una instancia de una base de datos de MongoDB y almacenando los datos sólo durante la ejecución de las pruebas.

```
@Test
@Order(7)
@DisplayName("Find all user")
void findAll() throws Exception
{
    ResultActions result = mockMvc
        .perform(MockMvcRequestBuilders
            .get( urlTemplate: "/users")
            .header( name: "authorization", userToken)
            .accept(MediaType.APPLICATION_JSON)
            .contentType(MediaType.APPLICATION_JSON)
            .characterEncoding(String.valueOf(StandardCharsets.UTF_8))
        )
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.content().contentType(MediaType.APPLICATION_JSON));

    List<User> list = objectMapper.readValue(result.andReturn().getResponse().getContentAsString(),
        objectMapper.getTypeFactory().constructCollectionType(List.class, User.class));

    System.out.println("GET ALL (SIZE): " + list.size());
}
```

Figura 3.6: test de integración para recuperar todos los usuarios de la base de datos

Una vez que se han ejecutado los test, Gradle genera un informe dónde se pueden ver los resultados de los test para cada método probado y los tiempos de ejecución.

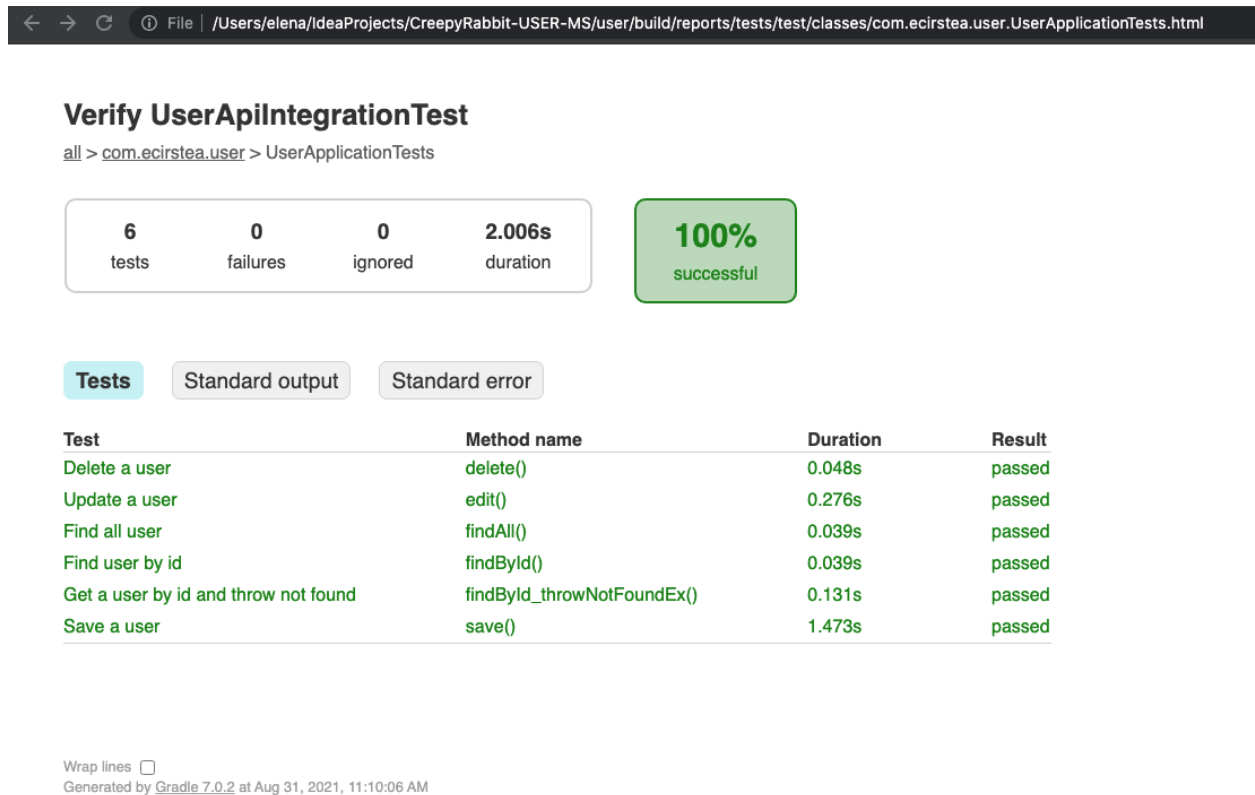


Figura 3.7: informe de Gradle con los resultados de las pruebas

PRUEBAS UNITARIAS

Las pruebas unitarias se han llevado a cabo con JUnit 4 en Kotlin en la aplicación móvil. Se ha optado por testear los inputs que introduce el usuario, desarrollándose para cada uso un caso de prueba.

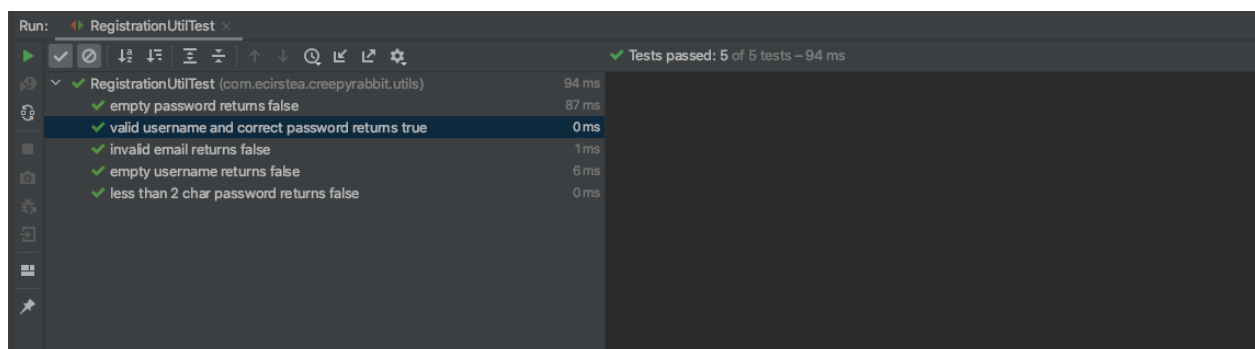


Figura 3.8: casos de pruebas unitarias para tratar los datos introducidos por el usuario

Memoria económica

Para la realización del conjunto de la aplicación móvil, el cliente ha comunicado el presupuesto que disponía y se han ajustado las funcionalidades del proyecto y el gasto según el tiempo empleado.

Coste económico del proyecto:

Distribución orientativa de los gastos del desarrollo			
Tarea	Horas empleadas	Precio hora	Total
Análisis y diseño	20	9,50€	190
Documentación	15	9,00€	135
Codificación	70	12,00€	840
Pruebas	15	13,00€	195
			Impuestos: 285,6€
			TOTAL: 1645,6€

El alojamiento del microservicio de usuario no supone un coste extra al proyecto, ya que se almacena en el propio equipo del desarrollador, no obstante, para el futuro habrá que plantearse su migración a *Amazon Web Services* u algún otro servicio de alojamiento en la nube.

En cuanto a los costes del mantenimiento de Firestore y Fire Storage, los servicios están contemplados en el plan *Spark* que ofrece Google. Este plan no conlleva coste alguno aunque sí está limitado el número de transacciones que se puede realizar al día. En este plan gratuito se ajusta a la característica de proyecto piloto e incluye lo siguiente:

Cloud Firestore	
Stored data	1 GiB total
Network egress	10 GiB/month
Document writes	20K writes/day
Document reads	50K reads/day
Document deletes	20K deletes/day

Figura 3.9 : Límite operaciones permitidas en Cloud Firestore con el plan Spark

Cloud Storage ?	
GB almacenados	5 GB
GB descargados	1 GB/día
Operaciones de carga	20,000/día
Operaciones de descarga	50,000/día
Varios depósitos por proyecto	×

Figura 3.10: Límite operaciones permitidas en Cloud Storage con el plan Spark

ABSTRACT DEL PROYECTO

Horror stories have become a very popular literary genre in recent years. Among them, we find a subgenre called creepypastas, which consists of short stories, based or not on real events, aimed at frightening and entertaining readers. Currently, such stories can be found all over the internet, however, for the time being, there are few sites intended to store them and have an exquisite pool of such stories.

One of the main objectives of this project has been, precisely, to create such a place, where these stories could be listened; in addition to allowing several fans of the genre to share their stories within the community. This platform, besides, has been developed using innovative technologies that give the final product great scalability potential as well as functionality. Android with Kotlin was used as a tool for the frontend part and a microservice with Java for the backend, also using a NoSQL database (MongoDB). Finally, the communication between them was made through API REST petitions.

BIBLIOGRAFÍA

Shin, K., Hwang, C. y Jung, H. (2017). NoSQL Database Design Using UML Conceptual Data Model Based on Peter Chen's Framework. *International Journal of Applied Engineering Research*, Volumen 12 (número 5), 632-636.
https://www.ripublication.com/ijaer17/ijaerv12n5_12.pdf

WEBGRAFÍA

Consultado el 16/08/2021

<https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/>

<https://manuel.cillero.es/doc/metodologia/metrica-3/tecnicas/diagrama-de-clases/>

<https://modeling-languages.com/discovery-and-visualization-of-nosql-database-schemas/>

Consultado frecuentemente:

<https://developer.android.com/>

HERRAMIENTAS UTILIZADAS

Creación del diagrama de Gantt:

<https://infograph.venngage.com/>

Creación diagrama clases y secuencias:

<https://cloud.smartdraw.com/>

Creación diagrama arquitectura:

<https://app.diagrams.net/>