TecnoCampus
Escola Superior Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

Grado en Ingeniería Informática de Gestión y Sistemas de Información

APLICACIÓN EDUCACIÓN VIAL

Memoria

ALEX GONZÁLEZ JUBANY
TUTORA: CATALINA JUAN NADAL

CURSO ACADÉMICO: 2022

TecnoCampus
Mataró-Maresme

Abstract

The goal of this project is to develop an application based on vial security concepts. The focus group of this particular project are children. It aims towards the idea that the parents or the people in charge of those children have the ability to control and manage the application as well as being able to suggest proposals, and report incidences. To achieve the main goals of this project, different preconditions and use cases are defined. Likewise, the study and analysis of technologies has been a key factor in order to boost and enhance factors such as the validity, consistency and applicability of the project.

Resum

Aquest projecte té com a objectiu el desenvolupament d'una aplicació mòbil basada en conceptes de la seguretat viaria. Està orientada al públic infantil i pretén que els tutors o responsables dels usuaris puguin controlar i gestionar l'aplicatiu, així com proposar suggeriments i reportar incidències. Per aconseguir els objectius establerts es defineixen un conjunt de requisits i casos d'ús, a més de l'estudi i l'anàlisi de tecnologies que aportin estabilitat i escalabilitat al projecte.

Resumen

Este proyecto tiene como objetivo el desarrollo de una aplicación móvil basada en conceptos de la seguridad vial. Está orientada al público infantil y pretende que los tutores o responsables de los usuarios puedan controlar y gestionar el aplicativo, así como proponer sugerencias y reportar incidencias. Para lograr los objetivos establecidos se definen un conjunto de requisitos y casos de uso, además del estudio y el análisis de tecnologías que aporten estabilidad y escalabilidad al proyecto.

Índice

Índice de figuras

Índice de tablas

Glosario de términos

TIC Tecnología de la información y Comunicación

SDK Software development kit

RAD Rapid Application Development

SSRN Social Science Research Network

JVM Java Virtual Machine

IDE Integrated Development Environment

BSON Binary JSON

JSON JavaScript Object Notation

DBaaS Database as a Service

VPC Virtual Private Cloud

API Application Programming Interfaces

iOS iPhone Operating System

SQL Structured Query Language

MVVM Model View ViewModel

MVC Model View Controller

MVP Model View Presenter

UI User Interface

XML eXtensible Markup Language

AMP Áreas Marinas Protegidas

1. Introducción

1.1 Objeto del proyecto

Actualmente, la tecnología es un mecanismo que aporta gran valor y conocimiento a nuestro día a día, permitiendo entender y agilizar el proceso de aprendizaje en entornos y tareas de alta complejidad.

Uno de los atractivos entre el público infantil reside en los dispositivos móviles y tabletas, permitiendo brindar a los educadores y familias infinidad de mecanismos para nutrir de conocimiento a sus infantes de forma sencilla, intuitiva, lúdica y dinámica.

Con ese fin, se pretende desarrollar un prototipo de aplicación para que el público infantil pueda aprender educación vial mediante juegos, pudiendo aportar a los niños la capacidad de identificar señales, marcas, instrumentos o símbolos que permiten prevenir y minimizar accidentes o contratiempos en la vía pública. Para eso se usarán mecanismos y juegos que refuerzan al usuario a recordar e interiorizar estos aspectos.

El *"target"* del usuario será entre dos y seis años, esto conlleva la adaptación de los juegos a las distintas habilidades que presenta el usuario en cuestión. Para eso se tendrá que investigar acerca del desarrollo motor, cognitivo y de lenguaje de los niños en esta etapa de desarrollo y como va a ser diseñada la aplicación para que ésta tenga una experiencia de usuario adecuada a sus capacidades.

El "*target*" de cliente serán los padres de los usuarios que consuman la aplicación. Van a tener acceso un apartado parental donde poder controlar los datos de los jugadores y una sección de sugerencias para realizar propuestas, recomendaciones o consejos y así poder escalar la aplicación teniendo en cuenta las opiniones del cliente.

La sincronización de datos entre dispositivos, usuarios y backend, permite acceder al entorno des de cualquier dispositivo aportando flexibilidad, seguridad y control al cliente.

1.2 Contexto

Actualmente, el mercado de las aplicaciones se encuentra en auge, es por eso que se va a realizar una aplicación enfocada a una temática de la cual aún no hay un monopolio. Aun así, se han seleccionado algunas aplicaciones acerca de la seguridad vial.

- **Tránsito para niños (dos dimensiones):**

  Aplicación educativa que trata de un "niño" que va por la ciudad y se va encontrando obstáculos en su camino, donde el niño va a tener que tomar decisiones para avanzar.

  El rango de "target" es muy elevado (tres-doce años), y hay parte del arte que no corresponde con la de nuestro país, el contenido audiovisual resulta poco atractivo.

  Contiene anuncios emergentes, si pagas no aparecen.

Fig. 1.1 Captura de pantalla juego Tránsito para niños. Fuente: Elaboración propia.

- **Seguridad Vial Panda-Seguridad (dos dimensiones):**

  El personaje viaja con su madre y narra los comportamientos correctos, pidiendo al usuario la su interacción en algunos momentos. Se trata de una historia interactiva.

  Juego desarrollado por una empresa de videojuegos "BabyBus" con un listado de aplicaciones educativas amplio.

Fig. 1.2 Captura de pantalla juego Panda-Seguridad. Fuente: Elaboración propia

- **AgenteJr: Seguridad vial (tres dimensiones):**

  Esta aplicación está más orientada a regular el flujo de tránsito de la ciudad. Parece un juego bastante complicado para las edades a las que se recomienda y las imágenes en tres dimensiones pueden derivar a la confusión de los usuarios.

Fig. 1.3 Captura de pantalla juego AgenteJr. Fuente: Elaboración propia

A continuación, se detalla cuáles han sido aquellos defectos o imperfecciones más relevantes en las aplicaciones para poder confeccionar la idea y redactar unos buenos objetivos.

Se ha detectado que la gran mayoría de aplicaciones usan una historia interactiva para aplicar conceptos y objetos. La señalización y los semáforos son los principales protagonistas.

Muestran los comportamientos incorrectos para que el niño rectifique las conductas mediante refuerzos positivos y añaden sonidos para que el usuario identifique las acciones.

La navegación entre pantallas no suele ser mayor de una o dos interacciones.

No se suele determinar un rango muy ajustado de edad, lo cual supone una complicación para los tutores, ya que no alcanzan a saber si la aplicación se va a adecuar a las capacidades del niño.

Carecen bastante de minijuegos clásicos con los que además de aprender lleguen adquirir habilidades como la de memorización. También resulta interesante un apartado con todos los elementos interactivos para reforzar el aprendizaje al usuario.

Parecen no dar mucha importancia a los elementos significativos de la aplicación, sobrecargan demasiado la interfaz y puede desembocar a confusión.

Finalmente, un aspecto importante es, la falta de la especificación del target al que van dirigidas estas aplicaciones. No lo dejan claro y el rango suele ser de 3 a 12 años, lo cual supone un problema, ya que dependiendo de la edad se tienen unas habilidades u otras.

1.3 Antecedentes

Hoy en día es una obviedad que los teléfonos móviles y tabletas se han convertido en nuestro máximo aliado de cara a muchos aspectos. Uno de ellos es el aprendizaje tanto para los más mayores como los más pequeños.

Android es el sistema operativo que resulta más atractivo y que está experimentando mayor crecimiento entre la población, en concreto, los más pequeños. Por lo tanto, va a ser la plataforma para la cual vamos a enfocar el proyecto para poder medir resultados.

A enero de 2022 las ratios de Andrid están sobre un 78% Android y un 21% iOS, según los datos de Statcounte [1].

Según estudios de implantación de TICs en la educación realizados por Rideout & Saphir en 2013, un 72% de los niños entre dos y ocho años utilizan asiduamente los dispositivos móviles [2].

Para el desarrollo de nuestra aplicación se tendrá en cuenta como principal usuario un niño entre dos y seis años de edad, el cual va a ser **objeto para el estudio del diseño**.

Para un correcto diseño se han analizado las aplicaciones existentes y añadido contenido de estudios acerca del desarrollo de aplicaciones para público infantil.

Uno de los principales focos se presenta con un buen diseño visual e interactivo, ya que será la fuente principal de comunicación entre el usuario y el software, acompañado de mensajes audiovisuales y multimedia.

En relación con el estudio *"Modelo teórico para el diseño y evaluación de la calidad en las apps infantiles (0-8 años)"* [2], *l*as características más importantes que requiere una app diseñada para público infantil son:

- **Atención y percepción.**
- **Simplicidad visual e interactiva.**
- **Distribución visual.**

Con estos tres aspectos se pretende contrastar información con un libro titulado *"Design for kids digital products for playing and learning"* [3] que realiza un estudio a fondo sobre qué tipo de diseño es adecuado para los distintos rangos de edad.

Para ello se van a diferenciar dos grupos dentro de nuestra aplicación, los de dos a cuatro años y los de cuatro a seis, criterio de diferenciación que se aconseja en gran parte de los estudios.

1.4 Necesidades de información

Para este proyecto se requieren conocimientos acerca de los siguientes aspectos:

- Diseño para público infantil.
- Arte, arquitectura y navegación adecuada.
- Clasificación de contenidos de aplicaciones móviles.
- Requisitos de permiso para la creación de aplicaciones para público infantil [4].
- Desarrollo front-end en Android Studio.

- MongoDB Atlas Cloud Database
- Desarrollo con MongoDB Realm SDK Java.
- Atlas App Services

2. Objetivos y alcance

2.1 Objetivos

Los principales objetivos a cumplir son:

- Realizar una aplicación potencialmente **escalable**.
- Lograr una **herramienta fácil** de proporcionar por los tutores o padres a los niños y que sea **correctamente consumible** por el target especificado.
- Acceso des de **distintos dispositivos** con los **datos sincronizados**.
- Incorporar minijuegos e iconos relacionados con la seguridad vial con el fin de enseñar **conceptos** y **señales**.
- **Ofrecer** un almacén de **imágenes** y **contenido multimedia**.
- **Implementar** apartado parental con formulario de **sugerencias**.

Se plantean los siguientes objetivos a cumplir para obtener el prototipo final deseado.

2.1.1 Objetivos del producto

- **Aplicación Android**
    - Ofrecer una aplicación con juegos adaptativos.
    - Garantizar una interficie “user friendly”.
    - Ofrecer refuerzos positivos para acelerar el aprendizaje.
    - Interactuar con los elementos o señales principales.
    - Ofrecer contenido multimedia.
    - Ofrecer apartado parental.
    - Ofrecer un apartado de sugerencias para tener comunicación con el cliente.

- Acceso al aplicativo con o sin internet con sincronización de datos.

2.1.2 Objetivos del cliente

- **Objetivos del tutor:**
    - Obtener herramienta de enseñanza.
    - Proporcionar entretenimiento.
    - Digitalizar la enseñanza.
    - Fomentar la seguridad vial.
- **Objetivos del usuario:**
    - Aprender acerca de la seguridad vial.
    - Entretenerse.
    - Interactuar.

2.1.3 Público potencial o target

Los principales beneficiarios son:

- **Usuarios:**
    - Perfil demográfico: niños/as entre dos y seis años.
    - Perfil sociocultural: cultura occidental.
    - Perfil digital: Perfil bajo. Solo interacción simple.
- **Tutores:**
    - Perfil demográfico: mayores de veinte años.
    - Perfil sociocultural: cultura occidental.

- Perfil digital: Público con mínimo de experiencia en la navegación por aplicaciones móviles.

2.2 Alcance

El producto final se obtiene como resultado del diseño y el desarrollo del aplicativo.

Se pretende obtener un aplicativo escalable con la capacidad de tener los datos sincronizados entre dispositivos, usuarios y el backend.

El usuario debe ser capaz de interactuar de forma autónoma por el aplicativo, con el fin de jugar a juegos sencillos con temática de seguridad vial y aprender algunos de los objetos y símbolos con fichas informativas.

Por parte de los tutores deben tener acceso des de cualquier lugar al aplicativo y la posibilidad de aportar ideas o reportar incidencias en el apartado de sugerencias, así como controlar el proceso del usuario.

3. Metodología

Se ha optado por una metodología ágil, dejando de lado las tradicionales como waterfall o RAD (diseño rápido de aplicaciones).

Actualmente, las nuevas formas de gestionar los proyectos nos brindan gran cantidad de beneficios como flexibilidad a la hora de afrontar cambios, más enfoque en el producto, comunicación con el cliente y demás aspectos que actualmente se adaptan a esta sociedad tan cambiante.

Al ser un proyecto desarrollado por una persona, la metodología ágil va a permitir dividir el proyecto en partes más manejables y facilitar los diferentes procesos.

Agile se basa en un desarrollo iterativo e incremental, donde las necesidades del proyecto permiten reformular algunos requisitos y soluciones.

Cada iteración incluye una planificación, análisis de requerimientos, diseño, codificación, pruebas, documentación y despliegue.

Se han consultado los principios del software ágil en un "paper" de SSRN [5], para enfocar, gestionar y desarrollar el proyecto.

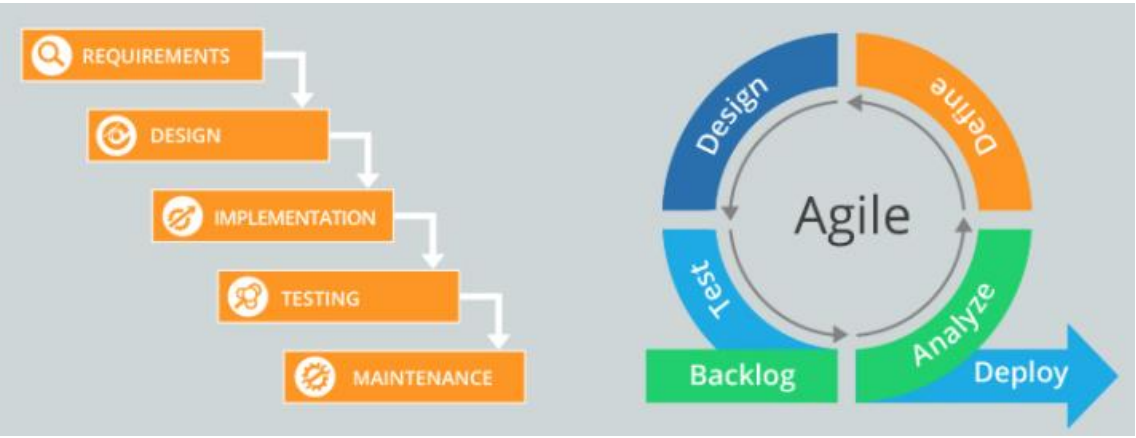

Fig. 3.1 Waterfall vs Agile. Fuente: https://www.digite.com/es/blog-es/pasar-de-cascada-a-agil-con-kanban

Se pretende completar las tareas “objetivo” realizando sprints entre una y dos semanas, pudiendo dividir la complejidad, y en caso de no finalizar alguna subtarea o se tengan que realizar cambios, se pasará al siguiente sprtint.

Esta metodología nos ofrece ventajas muy útiles para el proyecto en cuestión:

- Permite priorizar las tareas de un proceso. Pudiendo identificar cuáles son más importantes y cuales resultan ser innecesarias.
- Establecer y definir objetivos de forma constante y eficiente-
- Obtención de feedback de los clientes e usuarios durante el desarrollo del proyecto.

La plataforma Github Inc. será donde se va a alojar el proyecto, y el sistema de control de versiones va a ser Git.

4. Definición de requerimientos funcionales y tecnológicos

4.1 Definición de requerimientos funcionales

Para realizar una correcta toma de requerimientos se ha optado por escoger un método muy interesante, las "cinco W" [6]. Esta metodología permite entender de forma detallada las necesidades y los objetivos detrás de cada requerimiento.

- **What:** ¿Qué se necesita?
- **Why:** ¿Por qué es necesario este requisito/funcionalidad en el proyecto?
- **Who:** ¿Quién va a ser el objetivo?
- **Where:** ¿Dónde se va a usar?
- **When:** ¿Cuándo se va a utilizar?

A continuación, se van a detallar los requerimientos que se han definido. Al utilizar una metodología Agile estos van a poder cambiar, incluso van a poder aparecer nuevos requisitos durante el desarrollo.

Registrar un cliente en la aplicación

- **What:** Interfaz para registrar un nuevo cliente y guardar las credenciales el sistema.
- **Why:** Para poder tener acceso al aplicativo y tener un usuario.
- **Who:** Los tutores de los usuarios.
- **Where:** Aplicación móvil.
- **When:** Al iniciar la aplicación sin registro previo.

Registrar un usuario en la aplicación

- **What:** Haber creado un cliente anteriormente y guardar el nombre en el sistema.
- **Why:** Para poder jugar.
- **Who:** El usuario.
- **Where:** Aplicación móvil.
- **When:** Después de crear un cliente y antes de iniciar el juego.

Comprobar la existencia de un cliente

- **What:** Dadas las credenciales de un cliente saber si existe en el sistema.
- **Why:** Para poder acceder a la cuenta des de otro dispositivo.
- **Who:** Un cliente identificado.
- **Where**: Aplicación móvil.
- **When:** Cuando se pretenda acceder a la cuenta.

Proporcionar un sistema para modificar el avatar

- **What**: Modificar el avatar de un usuario.
- **Why:** Para favorecer la identificación.
- **Who:** El usuario.
- **Where:** Aplicación móvil.
- **When:** Cuando el usuario quiera.

Proporcionar un sistema de minijuegos

- **What:** La aplicación tiene que contener minijuegos para entretener y enseñar educación vial.
- **Why:** Para que los usuarios se sientan atraídos por el aplicativo.
- **Who:** El usuario.

- **Where:** Aplicación móvil.
- **When:** Cuando el usuario quiera.

Proporcionar elección de rango de edad

- **What:** Contener un apartado para seleccionar el rango de edad.
- **Why:** Para adaptar los juegos a las capacidades del usuario.
- **Who:** El usuario.
- **Where:** Aplicación móvil.
- **When:** Cuando el cliente o el usuario quieran.

Visualizar puntuaciones del usuario

- **What:** Poder consultar la puntuación.
- **Why:** Para transmitir feedback positivo al usuario.
- **Who:** El usuario.
- **Where:** Aplicación móvil.
- **When:** Cuando el usuario haya jugado.

Proporcionar un sistema de galería de ítems

- **What:** Galería de ítems para consultar los elementos del juego.
- **Why:** Para reforzar el aprendizaje de los elementos.
- **Who:** El usuario.
- **Where:** Aplicación móvil.
- **When:** Cuando el usuario acceda a la sección.

Proporcionar un sistema de sugerencias

- **What:** Permitir mediante un formulario dejar sugerencias del juego.
- **Why:** Para poder obtener feedback de los tutores y poder seguir escalando el aplicativo.
- **Who:** El cliente.
- **Where:** Aplicación móvil.
- **When:** Cuando el cliente tenga una cuenta y decida sugerir.

4.2 Definición de requerimientos tecnológicos

Para el desarrollo de la aplicación se requieren los siguientes requerimientos tecnológicos.

Android Studio Developer

El entorno de desarrollo usado es Android Studio, basado en la herramienta de IntelliJ IDEA, cuenta con un buen editor de códigos y gran cantidad de funciones que aceleran la productividad durante el desarrollo. Cuenta con flexibilidad de sistemas de compilación y un emulador de gran rapidez.

Las principales ventajas que se destacan del resto de IDEs son la ejecución del aplicativo en tiempo real, simulación de distintos dispositivos y tablets, la creación de elementos gráficos de forma sencilla, y es el IDE oficial de Android lo que asegura un correcto funcionamiento del software.

Una de las grandes desventajas es la gran cantidad de recursos que demanda, lo que implica desarrollar con un dispositivo con buenas especificaciones técnicas.

Lenguaje de programación: JAVA

Para la programación de apps en Android, es necesario que el código pueda compilarse y correrse en Java Virtual Machine (JVM) [10].

Actualmente un lenguaje que ha llamado mucho la atención en la comunidad es Kotlin, capaz de compilar a bytcode Java y nuevas ventajas que dejan a Java anticuado.

Una de las principales fortalezas es la interoperabilidad que tiene con Java, es decir, pueden permanecer en el mismo proyecto y que siga compilando sin problemas.

Java es uno de los lenguajes más utilizados en el mundo y sigue siendo el lenguaje oficial de Android.

El lenguaje es distribuido y orientado al objeto, cuenta con un recolector de basura que permite liberar y optimizar memoria, es multiplataforma y contiene altos niveles de seguridad.

Su amplia biblioteca de documentación y gran comunidad, resulta para los programadores una comodidad, lo que pude facilitar la incorporación de nuevos desarrolladores en un futuro.

Tipo de base de datos: MongoDB Atlas (NoSql)

MongoDB se fundamenta en una base de datos NoSQL, en vez de guardar los datos en tablas, guarda los datos en estructura BSON, esto permite hacer integración de datos de forma más fácil y rápida.

En MongoDB una base de datos es un contenedor de colecciones. Cada uno tiene su conjunto de archivos en el sistema de archivos. En general la estructura es-ta formada por Key/Value y los documentos no tienen un esquema predefinido, lo que implica una fácil adaptación de los capos durante el desarrollo.

Por lo que MongoDB Atlas, la base de datos como servicio (DBaaS) [11], es la que facilitara la implementación, utilización y escalabilidad de la BBDD en MongoDB en la nube.

Se caracteriza por tener varios niveles de seguridad uno de ellos el control de acceso mediante instancias de la base de datos que se implementan en una nube privada virtual (VPC) única para garantizar el aislamiento de red, también implementa listas blancas de IP, cifrado en tránsito y otras características más.

Al usar una estructura de instancias de base de datos distribuida geográficamente permite asegurar el correcto funcionamiento en cualquier parte y evitar un punto único de falla.

MongoDB Atlas permite escalar horizontalmente y verticalmente de forma fácil gracias a su elasticidad y el aprovisionamiento de recursos en el momento que se necesite.

Contiene un conjunto de servicios integrados que facilitan el desarrollo. Algunos de ellos son el servicio de base de datos en múltiples nubes, la generación de gráficos, APIs, disparadores, funciones, búsquedas sin necesidad de un motor adicional y la sincronización de dispositivos.

Para el desarrollo del aplicativo se han contemplado los servicios de base de datos y el de sincronización de dispositivos. Los cuales proporcionan las herramientas necesarias para cumplir con los requerimientos de la aplicación.

Motor de base de datos Realm [7]

Para el desarrollo de la aplicación, se requiere de unos requisitos para que esta sea funcional en cualquier circunstancia.

Realm es un motor de base de datos usado para el desarrollo de aplicaciones móviles tanto para Android como para sistema iOS. Este motor nos permite solucionar algunos problemas comunes dentro del desarrollo de aplicativos.

- En una app las conexiones se pueden perder, o los dispositivos pueden apagarse en cualquier momento.
- Los esquemas de datos API back-end y la base de datos deben ser consistentes.
- Las vulnerabilidades de seguridad deben ser fácilmente localizables.
- La consistencia entre el almacenamiento de base de datos y la memoria del aplicativo.
- Compatibilidad en múltiples sistemas operativos.

Gracias a las distintas bibliotecas y marcos, se pueden resolver de forma aislada los distintos obstáculos que se presentan al desarrollar una aplicación:

- **Almacenamiento local:** Realm Database se ejecuta en los dispositivos del cliente. Se puede almacenar, acceder y actualizar los datos de forma sencilla.

- **Fiabilidad de red**: El funcionamiento inicial es en local, no en la red. Al activar ***Atlas Device Sync***, Realm Database sincroniza los datos con ***Atlas App Services*** a través de la red en un subproceso en segundo plano, resolviendo los conflictos de forma coherente en cada cliente.

- **Interfaz de usuario reactiva**: Los ***"Live Objects"*** u objetos vivos, reflejan los últimos datos almacenados en ***Realm Database***. Realm ofrece la posibilidad de suscribirse a cambios para mantener la interfaz de usuario actualizada.

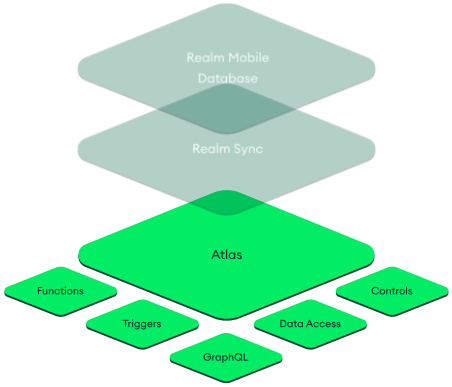

Fig. 4.1 Elementos para el funcionamiento de REALM SYNC. FUENTE: elaboración propia

Requerimientos a tener en cuenta

Va a ser importante los siguientes requerimientos para poder obtener un producto lícito:

- Proteger los datos.

- Cumplir las normativas de Play Store [4].

- Mantenimiento.

En cuanto a la versión de Android que se va a requerir va a ser como mínimo la Android 5 Lollipop, ya que estudios confirman que versiones anteriores han quedado prácticamente obsoletas [9].

Se enfoca únicamente para Android, porque como ha podido verse, es la que más se usa actualmente y en esta fase de desarrollo nos interesa tener un control máximo sobre la aplicación y en un futuro poder escalarla hacia una app híbrida. Las tecnologías escogidas permiten migrar la aplicación a híbrida en el momento que se requiera.

5. Desarrollo

5.1 Definición de casos de uso

CASO DE USO	REGISTRAR UN CLIENTE
Actor	Cliente
Pre condición	El cliente debe tener acceso a internet. El cliente no debe existir en la base de datos.
Descripción	El sistema permite registrar un nuevo usuario. Se le requiere el correo electrónico y una contraseña.
Ejecución normal	1. El cliente no ha iniciado sesión. 2. El cliente se encuentra en la pantalla de inicio de sesión. 3. El cliente introduce los datos. 4. El cliente presiona el botón de crear cuenta. 5. El sistema comprueba la validez de los datos. 6. El sistema registra al cliente. 7. El sistema redirige al cliente a la pantalla inicial.
Post condición	El cliente se encuentra registrado y en la pantalla principal del aplicativo.
Excepción	El cliente introduce mal los datos para el registro. El sistema no consigue registrar los datos.

Tabla 5.1 Caso de uso del registro de un cliente. Fuente: Elaboración propia.

CASO DE USO	INICIAR SESIÓN EN EL APLICATIVO
Actor	Cliente
Pre condición	El cliente debe tener acceso a internet. El cliente debe tener una cuenta El cliente no debe estar en ninguna sesión.
Descripción	El sistema permite iniciar sesión con el correo electrónico y la contraseña.
Ejecución normal	1. El cliente no ha iniciado sesión. 2. El cliente se encuentra en la pantalla de inicio de sesión. 3. El cliente introduce los datos. 4. El usuario presiona el botón iniciar sesión. 5. El sistema comprueba la validez de los datos. 6. El sistema redirige al usuario a la pantalla inicial.
Post condición	El usuario se encuentra en la pantalla principal del aplicativo.
Excepción	El usuario introduce mal los datos para el registro. El sistema no consigue acceder.

Tabla 5.2 Caso de uso de inicio de sesión en el aplicativo. Fuente: Elaboración propia

CASO DE USO	MODIFICAR AVATAR
Actor	Usuario
Pre condición	El usuario debe tener acceso a internet. El usuario debe estar registrado en la aplicación. El usuario debe estar en el apartado del Avatar
Descripción	El sistema permite modificar la foto del avatar.
Ejecución normal	El usuario presiona la imagen del avatar y se modifica.
Post condición	El usuario ha modificado su avatar.
Excepción	

Tabla 5.3 Caso de uso de modificación de avatar. Fuente: Elaboración propia

CASO DE USO	ENVIAR SUGERENCIA
Actor	Cliente
Pre condición	El cliente debe tener acceso a internet en algún momento. El cliente debe estar registrado en la aplicación. El cliente debe estar en la apartado parental.
Descripción	El sistema permite enviar sugerencias.
Ejecución normal	1. El cliente presiona en el icono para acceder al apartado parental. 2. El sistema proporciona un formulario. 3. El cliente rellena el formulario. 4. El cliente presiona el botón de enviar la sugerencia. 5. El sistema guarda y envía los datos. 6. El sistema refresca el formulario.
Post condición	El cliente ha enviado una sugerencia.
Excepción	El cliente introduce mal los datos. El sistema no consigue enviar los datos.

Tabla 5.4 Caso de uso del envío de sugerencia. Fuente: Elaboración propia

CASO DE USO	MODIFICAR DIFICULTAD DE LOS JUEGOS
Actor	Cliente
Pre condición	El cliente debe estar registrado en la aplicación. El cliente debe estar en el apartado parental.
Descripción	El sistema permite modificar la dificultad de los juegos según dos rangos de edad. De dos a cuatro años y de cuatro a seis.
Ejecución normal	1. El cliente presiona en el icono para acceder al apartado parental. 2. El sistema proporciona los datos del jugador y el nivel de dificultad. 3. El cliente selecciona el nivel de dificultad. 4. El sistema guarda los datos.
Post condición	El cliente ha modificado la dificultad de los juegos.
Excepción	

Tabla 5.5 Caso de uso modificación dificultad de juegos. Fuente: Elaboración propia

CASO DE USO	EMPEZAR PARTIDA
Actor	Usuario
Pre condición	El usuario debe estar registrado en la aplicación. El usuario debe estar en el apartado principal.
Descripción	El sistema permite acceder y jugar a minijuegos.
Ejecución normal	1. El usuario presiona en la imagen de jugar. 2. El sistema proporciona la interfaz donde se muestran los juegos. 3. El usuario selecciona un juego. 4. El sistema ejecuta el minijuego.
Post condición	El usuario se encuentra en una partida.
Excepción	

Tabla 5.6 Caso de uso de inicio de partida. Fuente: Elaboración propia

CASO DE USO	VISUALIZAR GALERÍA
Actor	Usuario
Pre condición	El usuario debe estar registrado en la aplicación. El usuario debe estar en el apartado principal.
Descripción	El sistema permite acceder al apartado galería para mostrar el contenido multimedia.
Ejecución normal	1. El usuario presiona en la imagen de galería. 2. El sistema proporciona la interfaz donde se muestran las imágenes. 3. El usuario selecciona una imagen. 4. El sistema muestra la imagen. 5. El usuario vuelve a presionar. 6. El sistema muestra la interfaz de las imágenes de nuevo.
Post condición	El usuario ha visualizado una imagen en la galería.
Excepción	

Tabla 5.7 Caso de uso apartado galería. Fuente: Elaboración propia

CASO DE USO	CERRAR SESIÓN
Actor	Cliente
Pre condición	El cliente debe estar registrado en la aplicación. El cliente debe estar en el apartado parental.
Descripción	El sistema permite cerrar la sesión activa.
Ejecución normal	1. El cliente presiona el icono de cerrar sesión. 2. El sistema despide al usuario con un mensaje y cierra sesión. 3. El sistema ofrece la pantalla de inicio de sesión.
Post condición	El usuario ha cerrado sesión de su cuenta.
Excepción	

Tabla 5.8 Caso de uso cierre de sesión. Fuente: Elaboración propia

5.2 Diseño

Se describe el proceso de diseño de la aplicación para satisfacer los requisitos. Pretende proporcionar una idea de cómo se ha desarrollado el software y el motivo de las tecnologías usadas.

5.2.1 Arquitectura de la Aplicación

La aplicación se basa en una arquitectura Cliente-servidor. Donde el cliente es una estación de trabajo que solicita servicios al servidor y este es una máquina que actúa como depósito de datos y se encarga de dar respuesta a las solicitudes del cliente.

Esta arquitectura nos permite integrar distintos clientes con sistemas operativos distintos, facilitar el uso de interfaces gráficas interactivas, integrar nuevas tecnologías y hacer crecer la infraestructura de forma estable.

Por la parte del Cliente la base de datos Realm Database almacena datos en reinos: colecciones de objetos de reino heterogéneos. Se puede entender cada reino como una base de datos. Cada objeto en un reino es equivalente a una fila en una tabla de base de datos SQL o un documento MongoDB.

MongoDB Realm simplifica la sincronización de datos. La sincronización funciona bidireccionalmente, moviendo datos entre Realm Mobile Database en el lado del cliente y MongoDB Atlas en el back-end.

En esta imagen se muestra el funcionamiento de la aplicación en el momento de solicitar datos des de Atlas App Services (Cliente) a Atlas Device Sync (Servidor).

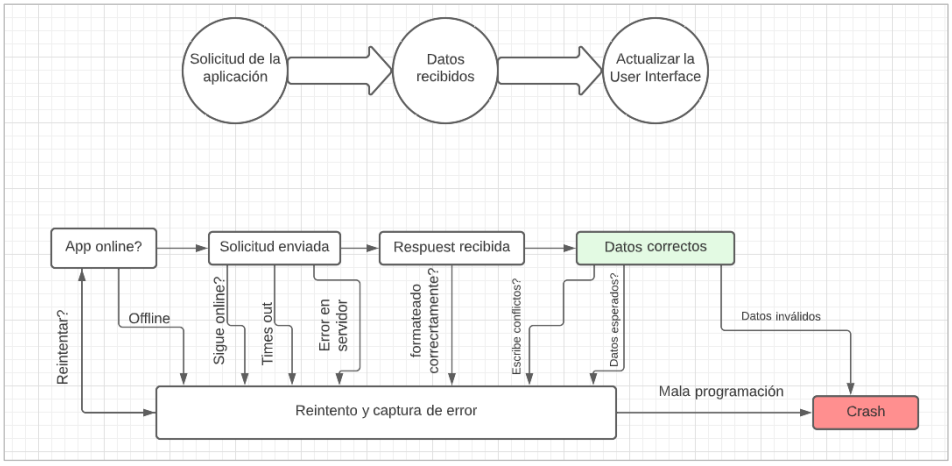

Fig. 5.1 Funcionamiento de la sincronización de datos. Fuente: Elaboración propia

En cuanto al servidor, gracias al servicio que ofrece MongoDB Atlas junto con Atlas App Services, las caídas de los servidores, los problemas de red, la inconsistencia de datos o la resolución de conflictos, no suponen un problema, ya que el mismo es capaz de lidiar con estas problemáticas.

El mantenimiento de los servidores tampoco supone un problema, ya que MongoDB ofrece distintas herramientas para gestionar y controlar el tráfico, así como aumentar las necesidades de espacio en el momento que se crea oportuno.

5.2.2 Arquitectura del Servidor

El proveedor de base de datos en el servidor va a ser MongoDB Atlas. Una base de datos de documentos que ofrece una gran variedad de ventajas.

Su modelo de datos intuitivo y la flexibilidad que ofrece para que el modelo de datos evolucione, resulta un gran atractivo.

El servicio Atlas Device Sync permite la consistencia de datos entre el almacenamiento local y el backend de MongoDB Atlas manteniendo los datos actualizados y precisos.

5.2.3 Arquitectura de Software del Cliente

El diseño de software usado es el Model View ViewModel (MVVM) [12], el cual permite separar la lógica empresarial de la interfaz de usuario (UI). Esta separación entre lógica de

la aplicación y la interfaz de usuario ayuda a abordar diferentes problemas de desarrollo y facilita la prueba, el mantenimiento y la evolución de la aplicación.

- **View:** Es la parte de la UI, los XML, las activities y los fragments. Estos no realizarán las acciones, se suscribien al view model y este les dirá cuando y como pintar.

- **ViewModel:** Sirve de enlace entre el Modelo y la Vista. Las vistas se suscriben a sus respectivos viewModels y al detectar que se modifica el modelo, este lo notifica a la vista.

- **Model:** Esta capa es la responsable de la abstracción de las fuentes de datos en este caso de Realm Database. Model y ViewModel trabajan juntos para obtener y guardar datos.

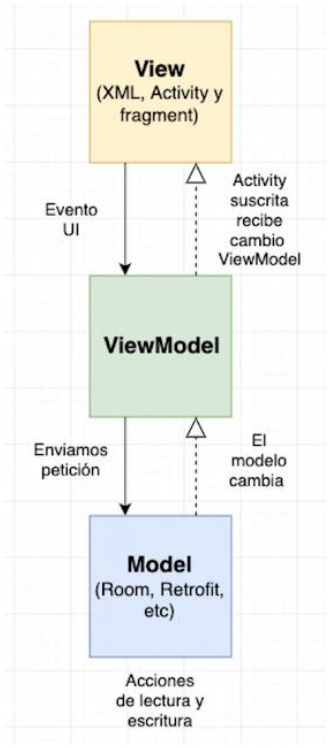

Fig. 5.2 Esquema MVVM. Fuente: Elaboración propia

Este patrón resulta ser muy parecido al MVP (Modelo-Vista-Presentador), ya que ViewModel desempeña la función de presentador. MVVM resuelve algunos inconvenientes respecto al MVP:

1. ViewModel no tiene referencia a la Vista.

2. La relación entre View y ViewModel es de muchos a 1.

3. No hay métodos de activación para actualizar la vista.

Para implementar esta arquitectura se ha usado la biblioteca DataBinding [13]. Es una biblioteca de compatibilidad que permite vincular los componentes de la UI de los diseños a las fuentes de datos la app.

La biblioteca de vinculación de datos proporciona clases y métodos para observar fácilmente los datos en busca de cambios. Permite hacer que objetos, campos o colecciones sean observables.

Para el uso de MVVM, la comunicación de ViewModel a View se realiza con solo LiveData [14]. De esta manera para la gestión de flujo de datos se usa LiveData y observers.

LiveData es una clase de contenedor de datos observable. A diferencia de un observable normal, es consicente del ciclo de vida, es decir, respeta el ciclo de vida de otros componentes de la aplicación, como fragmentos, actividades o servicios.

Este conocimiento garantiza que LiveData solo actualice los observadores de componentes de la aplicación que se encuentran en un estado de ciclo de vida activo.

Básicamente habrá un Observer que se suscriba a Observable para recibir las notificaciones de los últimos datos o los cambios de estado.

Ya no tenemos que preocuparnos por cancelar la suscripción onPause o onDestroy. Además, una vez que Observer se reanude, se notificará de inmediato los últimos datos del LiveData.

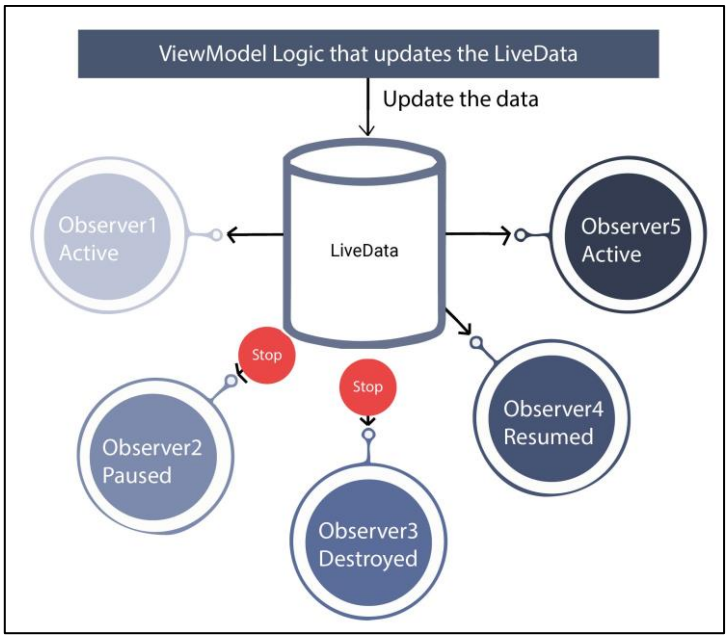

Fig. 5.3 Lógica ViewModel con LiveData. Fuente:
https://www.innominds.com/blog/introduction-to-livedata-in-android

Esta arquitectura aporta las siguientes ventajas al proyecto.

- La lógica de negocio está desacoplada de la interfaz de usuario.
- Realizar pruebas unitarias para el modelo y para vista-modelo, sin hacer referencia a la vista.
- Mantenimiento simple de los sistemas.

5.2.4 Diseño y navegación de pantallas

Esta sección muestra la navegación entre pantallas y el diseño implementado para cada funcionalidad.

5.2.4.1 Registro y acceso

Fig. 5.4 Pantalla registro y acceso. Fuente: Elaboración propia

5.2.4.2 Pantalla Inicial

La primera pantalla que aparece al iniciar sesión. El signo de exclamación de la parte superior es para acceder al apartado parental.

Fig. 5.5 Pantalla inicial. Fuente: Elaboración propia

5.2.4.3 Modificar Avatar

Fig. 5.6 Pantalla modificar avatar. Fuente: Elaboración propia

5.2.4.4 Pantalla de minijuegos

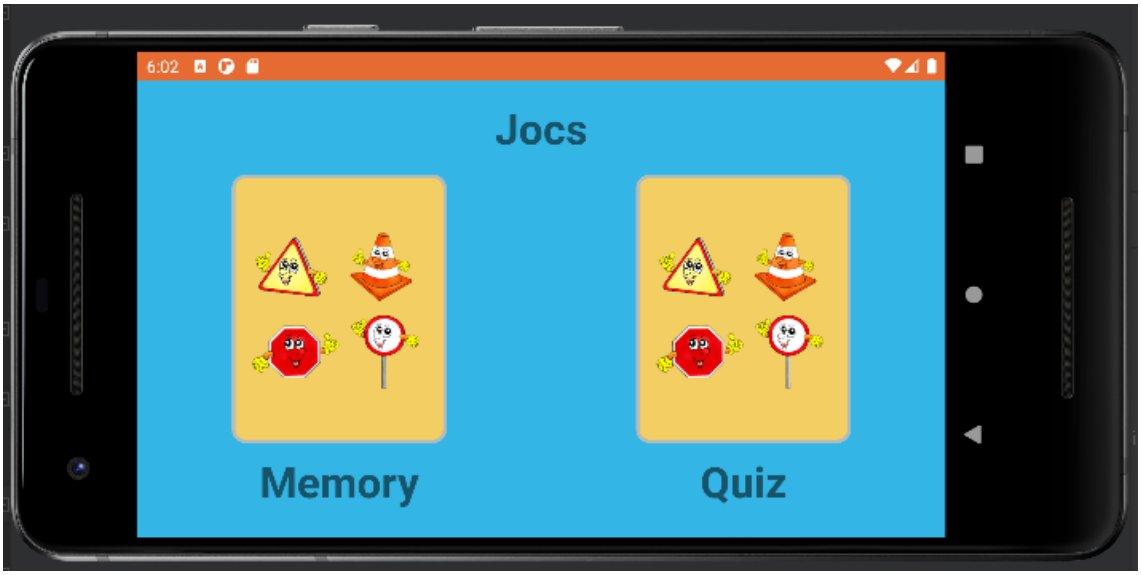

Fig. 5.7 Pantalla minijuegos. Fuente: Elaboración propia

5.2.4.5 Galería de ítems

Fig. 5.8 Pantalla galería de ítems. Fuente: Elaboración propia

5.2.4.6 Gestión parental y sugerencias

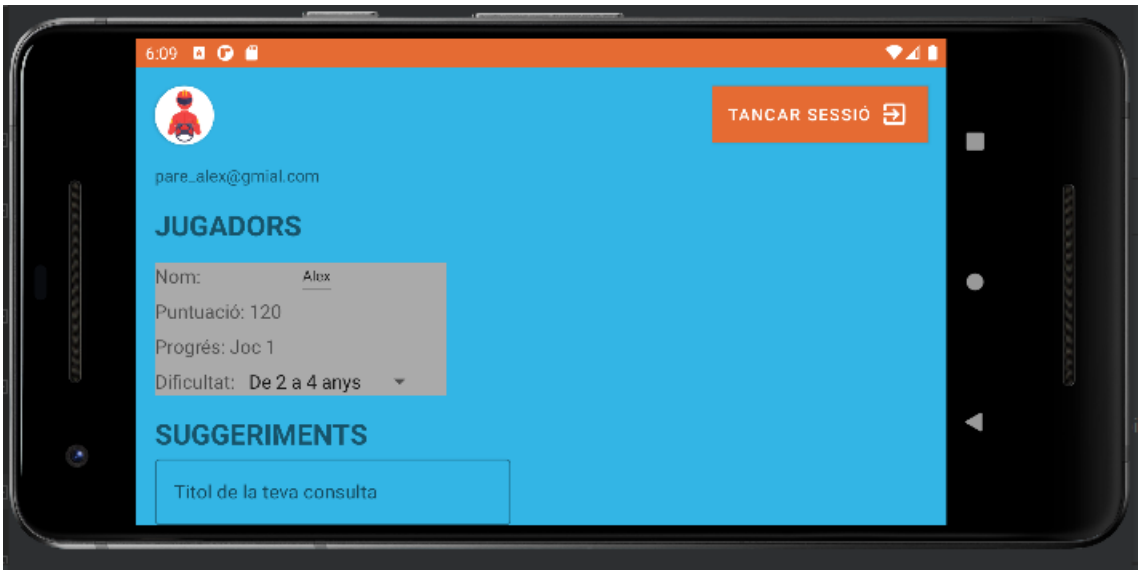

Fig. 5.9 Pantalla gestión parental y sugerencias. Fuente: Elaboración propia

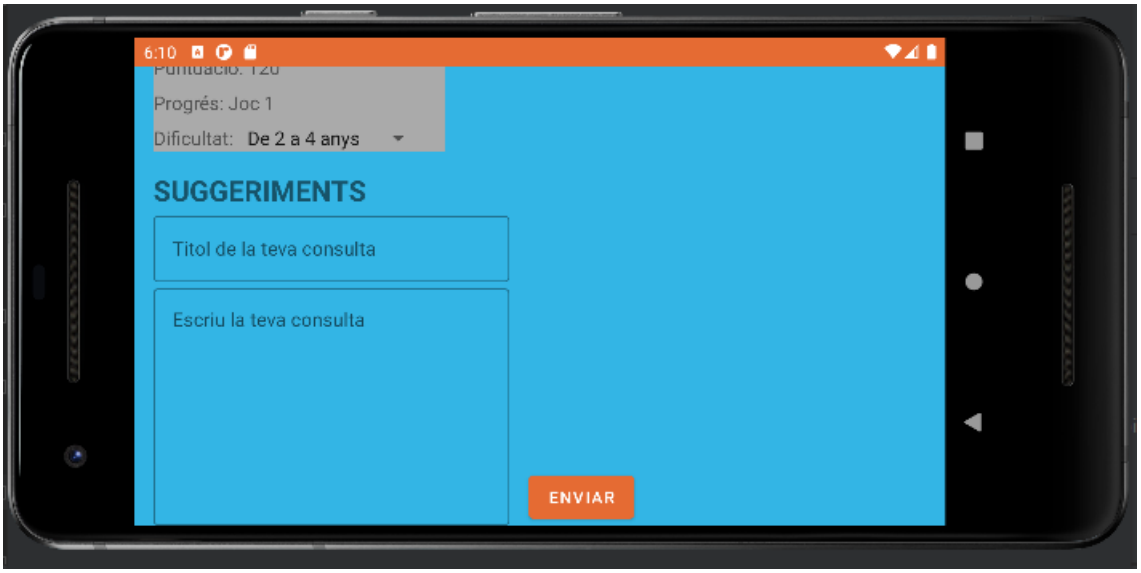

Fig. 5.10 Pantalla gestión parental y sugerencias 2. Fuente: Elaboración propia

5.2.5 Diseño de base de datos en MongoDB Atlas

En relación a la base de datos Backend MongoDB Atlas va a contener un Cluster, el cual permite escalar la horizontalmente en distintos servidores.

Dentro de los Cluesters se crean las Collections, que en base de datos SQL se les llama entidades o tablas.

El diagrama de clases usado para el aplicativo es el siguiente. Resulta ser un dominio bastante sencillo, lo que nos ha permitido trabajar de forma cómoda con todos los componentes, así como asentar una base útil y escalable.

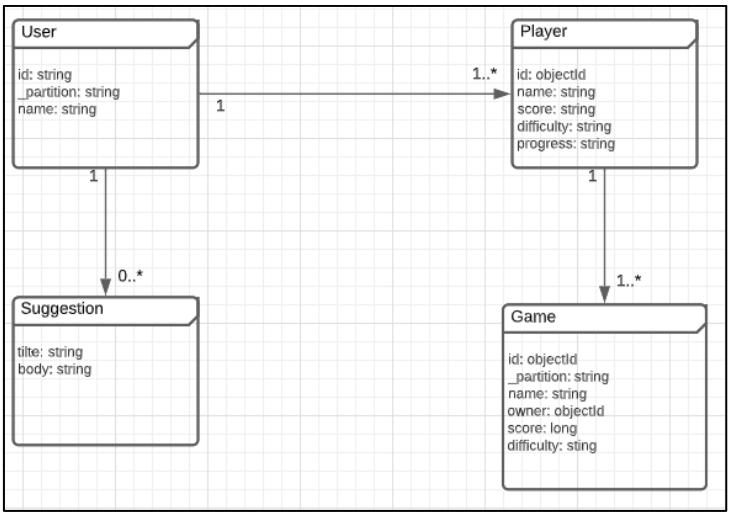

Fig. 5.11 Diseño UML base de datos. Fuente: Elaboración propia

5.3 Configuración Backend MongoDB Realm

Como ya se ha comentado anteriormente, Realm es un motor de base de datos que nos permite mantener los datos sincronizados en tiempo real entre el almacenamiento local (Realm Mobile Database) y el back-end (MongoDb Atlas).

También permite modificar el diseño de datos durante el desarrollo facilitando la escalabilidad de la aplicación.

Para ello, primero de todo se requiere de una cuenta en MongoDB inicialmente con la versión gratuita que ya nos permite almacenar datos y crear la aplicación Realm.

En el momento que se requiera se podrá cambiar a un plan dedicado y plantear-se la funcionalidad de copias de seguridad diarias.

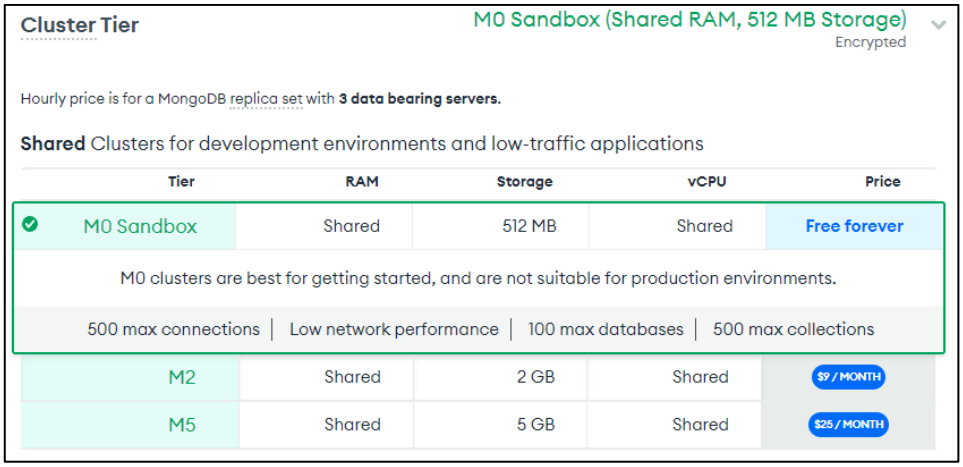

Fig. 5.12 Niveles de clusters. Fuente: Elaboración propia

Simplemente con rellenar las secciones del proveedor Cloud, la región y el nombre ya se puede crear el Cluser.

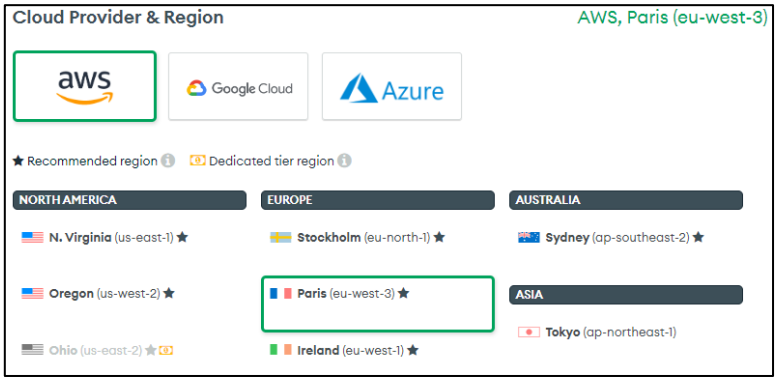

Fig. 5.13 Proveedor de Cloud y región. Fuente: Elaboración propia

Tras esperar unos minutos para que se configure el entorno, ya se puede ver el Cluster creado con su información.

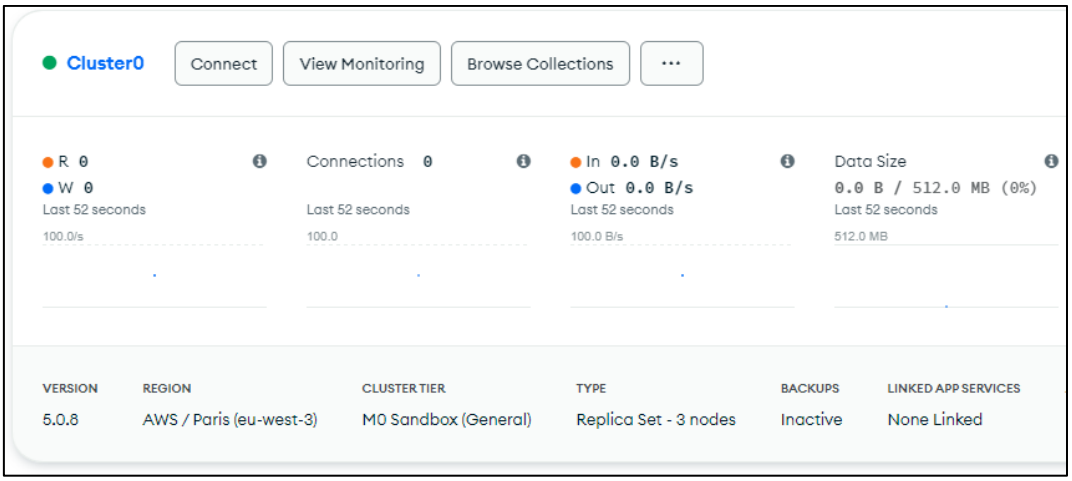

Fig. 5.14 Pantalla principal MongoDB. Fuente: Elaboración propia

Una vez dentro de MongoDB se accede al apartado Realm y aparece el siguiente formulario. Donde se rellena el nombre de la aplicación y que base de datos se va a usar, en este caso, un "Cluster" existente en MongoDB Atlas, una oferta de base de datos como servicio NoSQL en la nube pública, accesible des de cualquier navegador web.

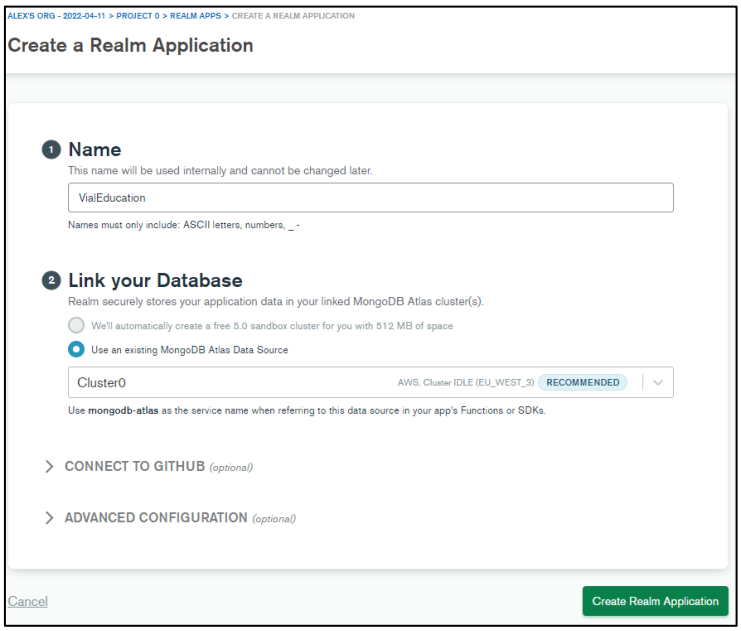

Fig. 5.15 Creación de aplicación Realm. Fuente: Elaboración propia

Una vez creada la aplicación es necesario ir al apartado Sync para activar la sincronización.

Se procede a seleccionar el tipo de sincronización, en este caso la basada en particiones, la cual permite elegir un solo campo en todas las colecciones para dividir los datos de Atlas en particiones según el valor que se indique en el campo.

Las particiones vinculan objetos en Realm Database a documentos en MongoDB. Gracias a la clave de partición la aplicación cliente crea objetos en el reino sincronizado. Cuando los objetos se sincronizan, este valor pasa a ser un campo en los documentos de MongoDB, determinando los permisos del cliente a los documentos.

Es importante activar el modo desarrollo para definir y modificar modelos de datos en esta etapa.

A continuación, se selecciona el *clúster* que se quiere sincronizar y se introduce el nombre de la base de datos en la cual queremos que se generen las nuevas colecciones.

Es necesario escoger una Clave de Partición, esta divide los datos de las colecciones en Realms, objetos que facilitan el acceso a ellos des de la aplicación y permite definir permisos de lectura y escritura. Es importante que esta clave se encuentre en el esquema JSON de cualquier colección que se quiera sincronizar.

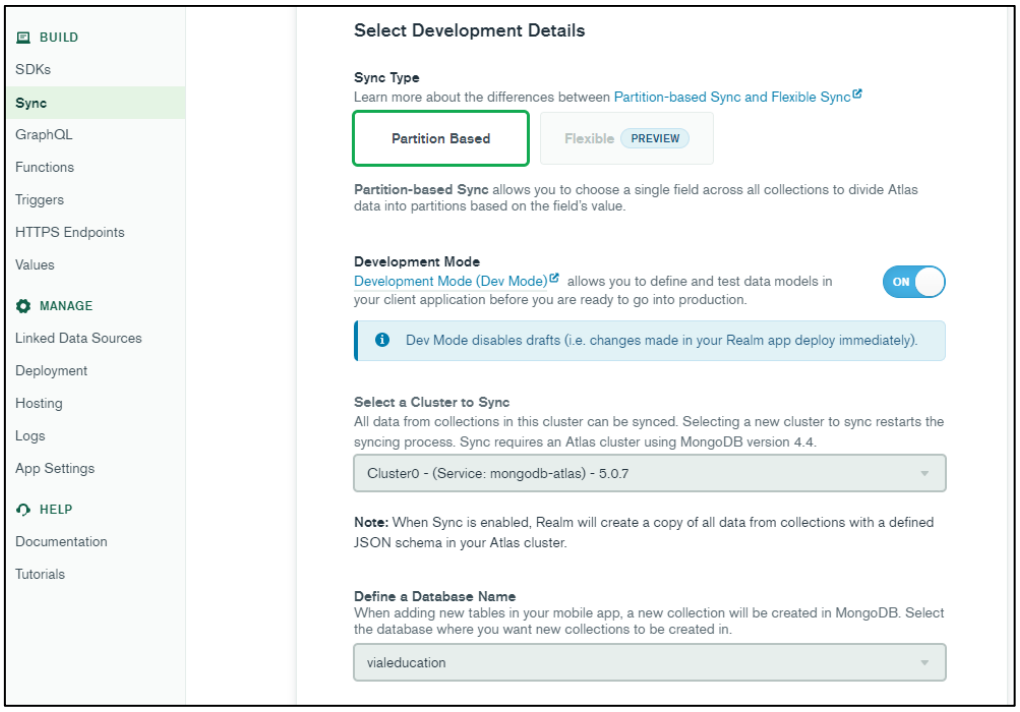

Fig. 5.16 Realm Sync detalles de desarrollo. Fuente: Elaboración propia

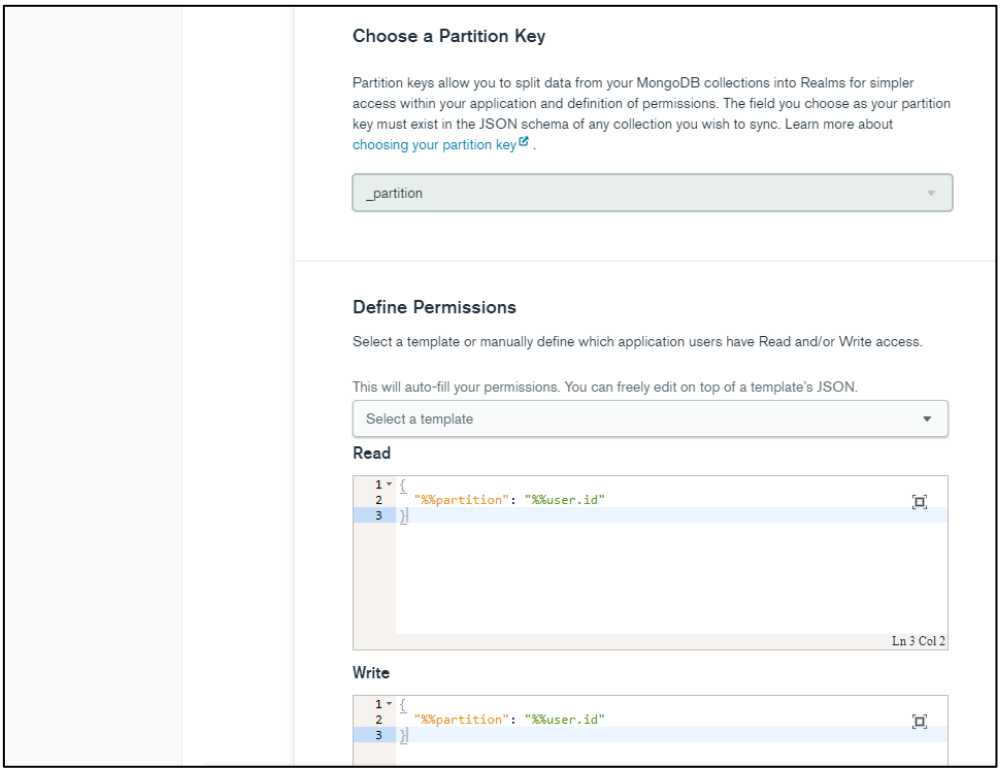

Fig. 5.17 Realm Sync clave de partición y permisos. Fuente: Elaboración propia

Una vez finalizada la configuración, establecemos cuáles van a ser los mecanismos de autenticación para que los usuarios accedan a la aplicación Realm, se debe activar el proveedor Email/Password, aunque también permite otros mecanismos como Google, API Key, Funciones personalizadas, etc.

Atlas App Services proporciona un conjunto de servicios de nube administrados que incluyen Atlas DeviceSync, funciones de nube sin servidor, reglas de acceso a datos declarativos y más.

Se usa App Services para escribir y alojar una aplicación en un entorno de nube administrado y respaldado por MongoDB Atlas.

Se incluyen servicios con varios requisitos comunes de back-end para facilitar el desarrollo. En este caso se ha considerado la utilización de *Realm Users and Authentications*.

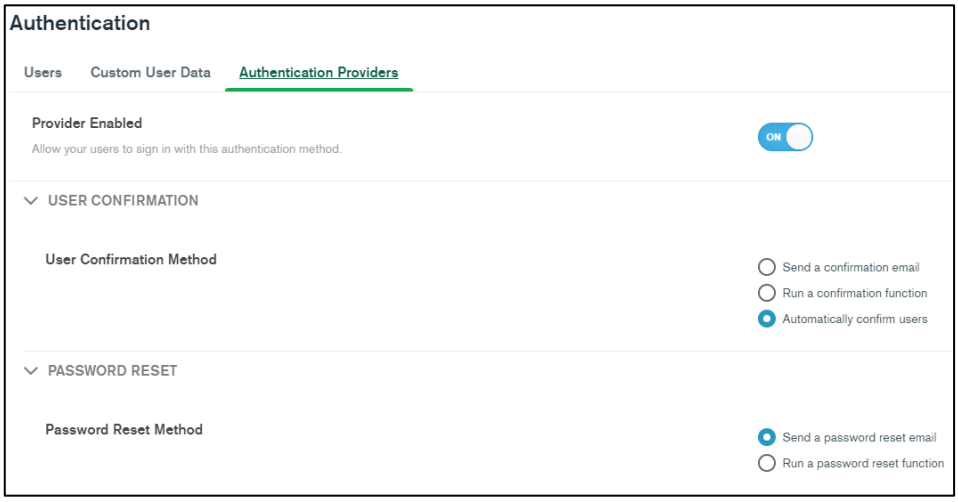

Fig. 5.18 Usuarios Realm y autenticación. Fuente: Elaboración propia

Finalmente, será necesario dar acceso al desarrollador. En el apartado Acceso de Red se debe añadir la dirección IP para poder conectarse al *cluster*.

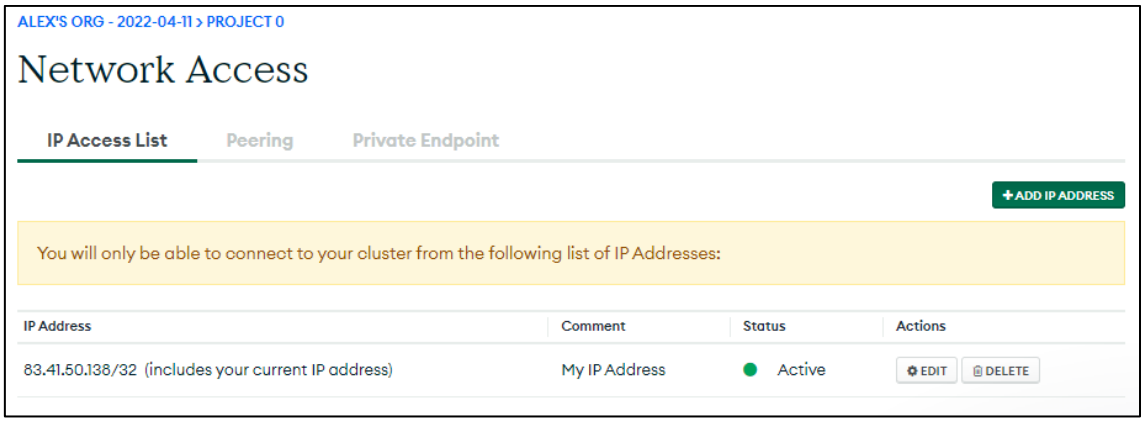

Fig. 5.19 Acceso desarrollador al cluster. Fuente: Elaboración propia

Ahora ya se puede conectar la aplicación de Android a la aplicación Realm que se ha configurado. Primero de todo se configura el fichero *build.gradle* del repositorio raíz, en este se definen las dependencias que se aplican a todos los módulos del proyecto.

En este proyecto se requiere la siguiente configuración, donde se incluye la dependencia *io.realm:realm-gradle-plugin:10.9.0* y otros paquetes para facilitar el desarrollo.

```
buildscript {
    ext.kotlin_version = '1.5.30'
    repositories {
        google()
        jcenter() // required for android-adapters realm recyclerview adapter library
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:7.0.2'
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
        classpath "io.realm:realm-gradle-plugin:10.9.0"
    }
}

allprojects {
    repositories {
        google()
        mavenCentral()
        jcenter() // required for android-adapters realm recyclerview adapter library
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Fig. 5.20 Dependencias build.gradle project. Fuente: Elaboración propia

En cuanto al archivo *build.gradle* de configuración a nivel de módulo que permite configurar ajustes de compilación para cada módulo específico en el que se encuentre.

Se requiere del plugin *realm-android*, el compilador SDK es el 31 y el mínimo el 22.

Para la construcción de la aplicación se requiere del identificador de la aplicación Realm. Para ello, en *buildTypes* introducimos el ID que aparece en MongoDB Realm.

```
android {
    compileSdk 31

    defaultConfig {
        applicationId "cat.tecnocampus.tfg.alexgonzalezjubany.vialeducation"
        minSdk 22
        targetSdk 31
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        def appId = "vialeducationapplication-hxyvf"  // Replace with proper Application ID
        debug {
            buildConfigField "String", "MONGODB_REALM_APP_ID", "\"${appId}\""
        }
        release {
            buildConfigField "String", "MONGODB_REALM_APP_ID", "\"${appId}\""
            minifyEnabled false
            signingConfig signingConfigs.debug
        }
    }
```

Fig. 5.21 Configuración módulo buid.gradle. Fuente: Elaboración propia.

En relación a la compilación, se realiza con Java 18. Se añade *sourceCompatibilit* y *targetCompatibility*.

```
compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
```

Fig. 5.22 Versión de compilación. Fuente: Elaboración propia

Importante activar la sincronización con realm.

```
realm {
    syncEnabled = true
}
```

Fig. 5.23 Activación Realm en módulo build.gradle. Fuente: Elaboración propia

Finalmente, se deben añadir las siguientes dependencias. Son librerías externas o módulos que facilitan el proceso de desarrollo.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jdk7:$kotlin_version"
    implementation "io.realm:android-adapters:4.0.0"
    implementation "androidx.recyclerview:recyclerview:1.2.1"
    implementation 'androidx.core:core-ktx:1.6.0'
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.6.0'
    implementation 'com.google.android.gms:play-services-auth:20.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'
    implementation 'androidx.navigation:navigation-fragment-ktx:2.4.2'
    implementation 'androidx.navigation:navigation-ui-ktx:2.4.2'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

Fig. 5.24 Dependencias para el proyecto. Fuente: Elaboración propia

Como podemos ver tenemos Kotlin, posible lenguaje para desarrollar aplicaciones en este sistema operativo.

Android ofrece dependencias como “recyclerview”, para crear listas dinámicas, “android.material”, para hacer una interfaz más bonita, “navigation”, para gestionar la navegación entre fragmentos. Y demás librerías que facilitan el proceso de desarrollo.

5.4 Implementación de funcionalidades

5.4.1 Registro de usuario Realm

Una vez ya está el backend de Realm en funcionamiento y el proyecto configurado para tener acceso a los recursos, se debe crear la interficie apropiada para que el usuario introduzca los datos y poder registrarlo.

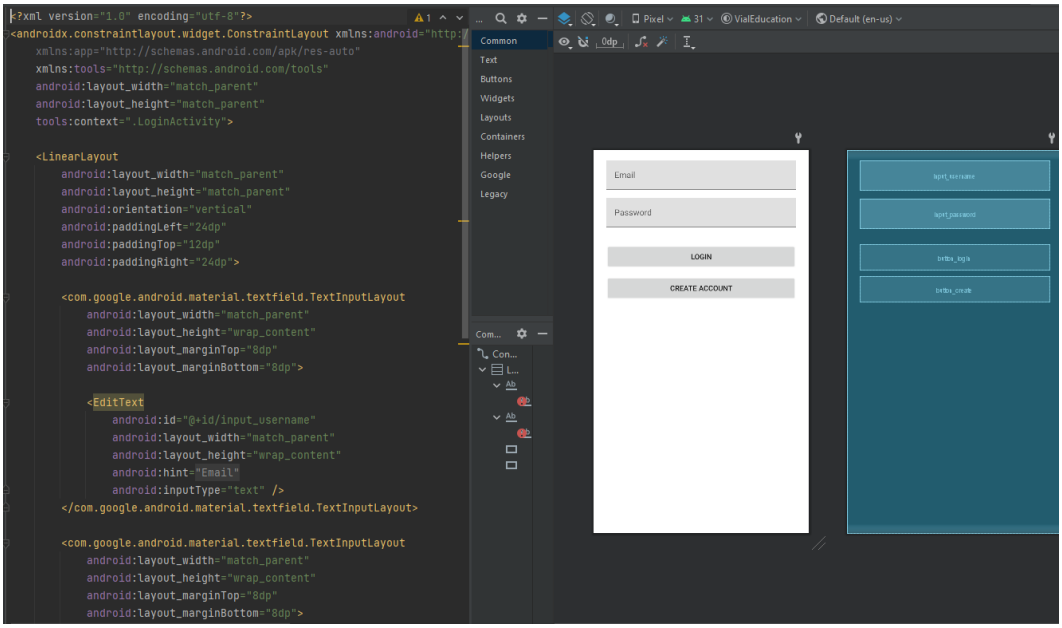

Fig. 5.25 UI para crear cuenta o iniciar sesión. Fuente: Elaboración propia

Una vez el frontEnd está listo y el usuario puede introducir sus datos, se ha de programar la lógica que permite registrar el usuario en cuestión a la aplicación Realm.

Es tan sencillo como leer los datos que ha introducido el usuario y dependiendo de que botón presione se ejecutará una transacción u otra.

En el caso de Create Account (Crear Cuenta) ejecutar la siguiente transacción usando la App Realm que se ha creado. *registerUserAsync*

```
public void register(String username, String password) {
    if (!validateCredentials(username, password)) {
        _viewState.postValue(new LoginObject(LOADING, arg1: null, arg2: null));
        _viewState.postValue(new LoginObject(ERROR, arg1: "Invalid username or password", arg2: null));
        return;
    }

    vialApp.getEmailPassword().registerUserAsync(username, password, it -> {
        if (!it.isSuccess()) {
            _viewState.postValue(new LoginObject(ERROR, it.getError().getErrorMessage(), arg2: null));
        } else {
            _viewState.postValue(new LoginObject(REGISTER_SUCCESS, username, password));
        }
    });

}
```

Fig. 5.26 Transacción Realm para crear un nuevo usuario. Fuente: Elaboración propia

Si los datos no son correctos, la misma aplicación informa al usuario del error y permite volver a ingresar de nuevo los datos.

Para que este mecanismo funcione adecuadamente, se debe configurar un Trigger (desencadenador) y una Función.

El trigger se encargará de que, en el momento de recibir la petición de crear un nuevo usuario, ejecute la función que registra este usuario en la aplicación.

Name	Trigger Type	Source / Provider(s)	Action Type	Schedule	Linked Function	Status
onNewUser	Authentication	Email/Password	Create		createNewUserDocument	Enabled

Fig. 5.27 Desencadenante para el proveedor Email/Password. Fuente: Elaboración propia.

La función que ejecuta es la "*createNewUserDocument*", la cual va a introducir en la base de datos el objeto usuario con los valores correspondientes.

```
exports = async function createNewUserDocument({user}) {
  const cluster = context.services.get("mongodb-atlas");
  const users = cluster.db("vialeducation").collection("User");
  return users.insertOne({
    _id: user.id,
    _partition: `user=${user.id}`,
    name: user.data.email,
  });
};
```

Fig. 5.28 Función createNewUserDocument. Fuente: Elaboración propia

Los usuarios contienen un id como clave primaria, el email y la partición de la cual forma parte.

En caso de que el usuario presione el botón de Login (Iniciar sesión), la transacción que se ejecuta es *loginAsync.* Esta comprueba la validez del usuario, si los datos son incorrectos la aplicación informa al usuario de que algún dato no es correcto. En caso contrario, el usuario ingresa al aplicativo donde se muestran distintos accesos como el menú de juegos, la galería, la gestión de perfil de usuario o el acceso parental.

```
public void login(String username, String password) {

    if (!validateCredentials(username, password)) {
        _viewState.postValue(new LoginObject(LOADING,  arg1: null,  arg2: null));
        _viewState.postValue(new LoginObject(ERROR,  arg1: "Invalid username or password",  arg2: null));
        return;
    }

    Credentials emailPasswordCredentials = Credentials.emailPassword(username, password);
    vialApp.loginAsync(emailPasswordCredentials, it -> {
        if (!it.isSuccess()) {
            _viewState.postValue(new LoginObject(ERROR, it.getError().getErrorMessage(),  arg2: null));
        } else {
            _viewState.postValue(new LoginObject(LOGIN_SUCCESS, username, password));
        }
    });
}
```

Fig. 5.29 Transacción Realm para verificar la existencia de un usuario. Fuente: Elaboración propia

5.4.2 Crear, Leer, Actualizar, borrar documentos con Realm Sync

Prácticamente todas las funcionalidades requieren de manejo de datos en local y la propia sincronización con el backend de Mongo DB. Por ello se muestra cuáles son las transacciones utilizadas para satisfacer el requerimiento de estas funcionalidades.

Como ya se ha comentado, realm funciona principalmente en local, pero al activar Atlas Device Sync estos datos se pueden sincronizar con el backend de MongoDB Atlas. Esta función se basa en el supuesto de que la conectividad no siempre es estable, lo llaman *offline-first*.

El funcionamiento de “fuera de línea primero” se basa en detectar los cambios en el dominio local (cliente) y Realm SDK envía lo antes posible los cambios al servidor. Del mismo modo, Realm SDK recibe los cambios del servidor y los integra en el dominio local. Es capaz de seguir trabajando sin preocuparse de la conectividad en tiempo real.

El SDK de Java permite el acceso a recursos de red y disco de dos maneras: **sincrónica** o **asincrónica**. Mientras que las solicitudes sincronizadas bloquean la ejecución hasta que la solicitud devuelve éxtito o fallo, las asincrónicas asignan una devolución de llamada y continúan con la ejecución en la siguiente linea de código.

Principalmente las transacciones de lectura y escritura deben realizarse mediante transacciones asincrónicas en el subproceso de la interfaz. Esto se debe a que realizar estas transacciones de forma sincrónica pude llevar a una interfaz lenta o que no responda.

De todas formas, si es necesario se pude configurar de forma sincrónica.

Las llamadas asincrónicas para ejecutar transacciones en un reino devuelven una instancia de RealmAsyncTask donde se puede controlar el éxito o la falla de la transacción.

Para poder abrir un reino sincronizado se requiere de una configuración para controlar los detalles de la sincronización con Atlas App Services. Tales como la partición, el tiempo que debe esperar antes de que se agote el tiempo de espera de una solicitud, si permite lecturas o escrituras sincrónicas en la interfaz de usuario, y más.

```
user = vialApp.currentUser();

partition = "user="+user.getId();
SyncConfiguration config = new SyncConfiguration.Builder(user, partition)
        .build();

Realm.setDefaultConfiguration(config);

userRealm = Realm.getDefaultInstance();
```

Fig. 5.30 Configurar y abrir Realm. Fuente: Elaboración propia

En userRealm se encuentra el reino abierto donde se podrán realizar las transacciones.

5.4.2.1 Crear

Para la creación de un nuevo documento, por ejemplo, un Jugador, se requiere de un objeto jugador y con el bloque de escritura de realm lo añadimos.

Es importante que esta transacción se realize en un hilo de ejecución distinto al de la interfaz. Para ello se puede usar la clase Thread, este recibe un objeto Runnable al cual se debe implementar usando el método “run()”.

```
public void createPlayer(){
    new Thread(new Runnable() {
        @Override
        public void run() {
            Player player = new Player( name: "DefNickName");
            userRealm.executeTransaction (transactionRealm -> {
                transactionRealm.insert(player);
            });
        }
    });
    userRealm.close();
}
```

Fig. 5.31 Creación de documento con Realm Sync. Fuente: Elaboración propia

Al finalizar la transacción es necesario cerrar el reino abierto con el método “close()”.

5.4.2.2 Leer

Se puede recuperar una colección con todos los elementos de un reino y filtrar esa colección con un filtro. Para que la aplicación funcione correctamente se debe realizar un un hilo de ejecución distinto al de la interfaz.

```
RealmResults<Player> players = userRealm.where(Player.class).findAll();
Player playerThatBeginWithA = players.where().beginsWith( fieldName: "name",  value: "A").findFirst();
```

Fig. 5.32 Lectura de colección y filtro con RealmSync. Fuente: Elaboración propia

5.4.2.3 Actualizar

Para modificar un Jugador, se debe usar una transacción de escritura tal que así. En este caso se recoge el jugador que está actualmente jugando y se modifica la puntuación.

```
userRealm.executeTransaction( transactionRealm -> {
    Player innerOtherPlayer = transactionRealm.where(Player.class)
                                .equalTo( fieldName: "_id", currentPlayer.getId())
                                .findFirst();
    innerOtherPlayer.setScore(innerOtherPlayer.getScore() + gameScore);
});
```

Fig. 5.33 Actualización de objeto en una colección con RealmSync. Fuente: Elaboración propia

5.4.2.4 Borrar

Finalmente si se requiere eliminar cualquier objeto de la colección simplemente llamando a la tarea “deleteFromRealm()” en una transacción de escritura se elimina.

```
userRealm.executeTransaction( transactionRealm -> {
    Player delCurrentPlayer = transactionRealm.where(Player.class)
            .equalTo( fieldName: "_id", currentPlayer.getId())
            .findFirst();
    delCurrentPlayer.deleteFromRealm();
});
```

Fig. 5.34 Eliminar objeto de una colección con RealmSync. Fuente: Elaboración propia

El control interno de conflictos consiste en lo siguiente. Des de la perspectiva de Atlas App Services (servicio en MongoDB Atlas), los conjuntos de cambios pueden llegar en cualquier momento que lo permita la conectividad. No hay garantía de que un conjunto de cambios con marca de tiempo anterior de un cliente llegue antes que un conjunto de cambios con marca de tiempo posterior de otro cliente. Atlas App Services mantiene un historial de transacciones por dominio para lidiar con la naturaleza desordenada de los mensajes.

Básicamente la última escritura es la que gana, pero hay otras técnicas más sofisticadas como la transformación operativa que sirve para aplicar un control de concurrencia en entornos de edición corporativa.

6. Conclusiones

En este apartado se argumentan las conclusiones obtenidas del trabajo realizado, la ingeniería de software aplicada y del producto final obtenido, mostrando las dificultades encontradas, las limitaciones y los logros obtenidos. Sin olvidar las tareas que siguen en proceso de desarrollo.

6.1 Conclusiones del trabajo

El desarrollo de un aplicativo para público infantil supone de un estudio de las necesidades y capacidades del usuario, así como el control de sus responsables o tutores dentro de la misma.

Al ser un producto muy ambicioso se establecieron un número determinado de funcionalidades a desarrollar para asentar una base del aplicativo y comprobar el correcto funcionamiento de las tecnologías escogidas.

El estudio previo, los objetivos y los requerimientos de las funcionalidades, ayudaron a determinar que tecnologías nos permitían además de desarrollar un producto, poder ofrecer una arquitectura con gran escalabilidad para poder seguir desarrollando de forma fácil y eficiente.

Se descartó la opción de desarrollar una aplicación hibrida ya que según estudios los dispositivos Android son los más usados en el mercado y el desarrollo híbrido suponía una curva de aprendizaje muy elevada, así que se optó por el sistema operativo Android que cuenta con su propio IDE Android Studio el cual contiene mucha documentación y permite crear aplicaciones con diseños backend y frontend sofisticados.

Inicialmente se pensó en desarrollar un aplicativo el cual solo funcionase sin acceso a internet, lo que suponía de utilizar bases de datos en el dispositivo local. Al plantear distintas funcionalidades, algunas se vieron afectadas por la necesidad de acceder a internet, lo que implicó buscar que tecnologías ofrecían un sistema robusto, estable y que cumpliera las necesidades.

MongoDB ofrecía una gran cantidad de servicios que nos permitían dar respuesta a los requerimientos de las funcionalidades. Por ello se tuvieron que investigar todas estas funcionalidades y servicios hasta encontrar la que mejor se adaptaba.

Finalmente, MongoDB Atlas ofrece distintos servicios en el que se encuentra el motor de base de datos en la nube y Device Sync, el servicio que permite la sincronización de datos entre dispositivos des de cualquier lugar.

Sin un conocimiento previo de estas tecnologías la curva de aprendizaje dentro del proyecto aumentó de forma significativa, pero aseguraba cumplir con los objetivos, y así desarrollar un producto que al finalizar el periodo del trabajo tuviera unas bases sólidas, con la posibilidad de formar un equipo e incrementar la productividad del desarrollo.

Concluimos que la parte de investigación y estudio de las diferentes opciones, aportó tecnologías que ofrecen muchos mecanismos de escalabilidad lo que era uno de los principales objetivos.

6.2 Conclusiones de la ingeniería de Software

Para el diseño de ingeniería de Software se analizaron las distintas arquitecturas que Android Studio recomienda para un correcto funcionamiento.

Una de las problemáticas que se detectaron es la necesidad de conocimiento del entorno para desarrollar de forma consistente y con buenas practicas, lo que supuso bastante tiempo de aprendizaje para obtener una aplicación funcional.

Andorid recomienda usar su lenguaje oficial Kotlin, lo que suponía aprender de nuevo este lenguaje. Esto incrementaba significativamente el aprendizaje que se requería, así que se optó por desarrollar en Java ya también aporta beneficios dentro del proyecto.

Se valoraron distintas arquitecturas como Modelo Vista Presentador (MVP), Modelo Vista Controlador(MVC) y Modelo Vista Vista-Modelo(MVVM). En este análisis pudimos ver cuál es la que nos aportaba más valor y una mayor base para su posterior escalabilidad.

Tanto MVP como MVVM resultaron proporcionar mejores resultados que MVC, ya que permite dividir la aplicación en componentes modulares y con propósitos mejor definidos.

Ambos, sin embargo, agregan más complejidad a nuestra aplicación. Para esta aplicación sería suficiente con MVC pero no nos permitía esta flexibilidad y modulación de los componentes, por lo que se optó por usar MVVM.

El resultado de usar MVVM con data binding ha permitido seguir un modelo de programación más reactivo y con un número de líneas de código considerable, así como separar la lógica empresarial de la interfaz de usuario (UI). Esta separación ayuda a abordar diferentes problemas de desarrollo y facilita la prueba, el mantenimiento y la evolución de la aplicación

Se han podido aplicar las buenas prácticas que recomienda Andoid para que el proyecto sea mantenible y escalable de forma sencilla y eficiente.

Los recursos como tamaños, strings, formatos, temas etc. Se han separado en distintos XML para poder usar estándares para todas las funcionalidades de la aplicación.

Se ha dotado de Navigation manager para determinar la navegación entre Activity y Fragments.

Las dependencias han quedado separadas y con versiones en variables para su correcto seguimiento y actualización.

Concluimos que la ingeniería de Software aplicada, cumple con los requisitos planteados y se ha obtenido un diseño acorde con las necesidades, tanto técnicas como las orientadas al usuario.

6.3 Conclusiones del producto

Se concluye que se ha obtenido un producto donde las funcionalidades más importantes se han logrado, pero el proyecto no se da por finalizado, sigue en fase de desarrollo.

Se ha subestimado el tiempo de aprendizaje lo que ha llevado a no dejar por finalizada la funcionalidad de los minijuegos.

Gracias al aprendizaje obtenido y el diseño de software empleado, los próximos springs se van a desarrollar con un periodo más reducido de tiempo. Como ya se ha comentado, se ha

creado un producto que siente las bases para que las próximas adaptaciones se realicen de la forma más eficiente posible.

7. Ampliaciones

Gracias a las arquitecturas usadas esta aplicación se puede ampliar hacia múltiples posibilidades.

MongoDB Realm ofrece distintos métodos de identificación y de recuperación de cuenta. Se puede implementar el acceso mediante Google, Facebook, etc.

Gracias a la flexibilidad y la sincronización de datos se pueden separar los dos fronends del tutor/responsable y la del usuario. Así el usuario no tendrá de opción a acceder al apartado parental y será exclusivamente el responsable el que gestione la información y los datos.

Con esta misma idea, se pueden llegar a realizar juegos en entornos colaborativos. Es decir, tanto juegos en tiempo real, como juegos por turnos con otros jugadores sin necesidad de tener conexión en tiempo real, ya que en el momento que coja conexión los datos ser sincronizan.

Se puede orientar hacia la educación escolar permitiendo que los tutores controlen los jugadores, así como tener un apartado de vistas para personas con discapacidades o afecciones, y que sea el tutor el que gestione las interfaces según los requisitos de los alumnos.

Hay una gran cantidad de opciones a implementar que desmarcarían esta aplicación sobre las que actualmente permanecen en el mercado.

Hay muchos proyectos donde muestran como han implementado su aplicación Realm con la estrategia de particiones y se puede ver la gran cantidad de posibilidades que permite.

Por ejemplo, en 2020 el programa de protección marina WildAid creó una aplicación para que los oficiales la usen mientras patrullan por el mar Áreas Marinas Protegidas por todo el mundo. La aplicación se llama O-FISH y sirve para que los agentes del orden busquen y creen informes de abordaje mientras están en el mar, patrullando las AMP y abordando embarcaciones para su inspección.

Esta aplicación está orientada a no tener conexión debido a las dificultades en alta mar, esto supuso el caso de uso perfecto para la base de datos Realm y MongoDB Realm Sync.

Este proyecto O-FISH [15] está publicado y es una buena fuente de inspiración para ver en qué casos de uso se encuentra esta tecnología, así como aprender el funcionamiento y como resuelven las problemáticas.

8. Bibliografia

[1] Statcounter GlobalStats [consulta: 5 enero de 2022]

Disponible en https://gs.statcounter.com/os-market-share/mobile/spain/.

[2] RCUB, Revistes científiques de la Universitat de Barcelona [consulta: 7 enero 2022]

Disponible en https://revistes.ub.edu/index.php/der/article/view/14497/pdf

[3] Levin, Debra: Design for kids Rosenfeld Media Broklyn, New York 2014

[4] Support Google [consulta: 11 enero 2022]

Disponible en https://support.google.com/googleplay/android-developer/answer/9893335?hl=es

[5] SSRN Papers: Agile Methodology Software Development Adaptability Challenges in Corporate Organization [consulta: 12 enero 2022]

Disponible en https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3851349

[6] University of Nebraska-Lincoln [Consulta: 15 enero 2022]

Disponible en https://its.unl.edu/bestpractices/remember-5-ws

[7] MongoDB [Consulta: 21 febrero 2022]

Disponible en https://www.mongodb.com/es/realm/mobile/sync

[8] Infobae [consulta: 17 enero 2022]

Disponible en https://www.infobae.com/america/tecno/2021/11/23/esta-es-la-version-de-android-mas-utilizada-del-mundo

[9] Ministerio de Cultura y deporte [consulta: 2 febrero 2022]

Disponible en https://www.culturaydeporte.gob.es/cultura/propiedadintelectual/la-propiedad-intelectual

[10] Oracle [consulta 12 febrero]

Disponible en https://docs.oracle.com/javase/specs/jvms/se7/html/

[11] MongoDB [consulta 15 enero]

Disponible en https://www.mongodb.com/docs/atlas/

[12] GreeksForGeeks [consulta 2 febrero]

Disponible en https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/

[13] Android Developer [consulta 2 febrero]

Disponible en https://developer.android.com/topic/libraries/data-binding

[14] Android Developer [consulta 2 febrero]

Disponible en https://developer.android.com/topic/libraries/architecture/livedata

[15] MongoDB [consulta 6 marzo]

Disponible en https://www.mongodb.com/developer/products/realm/realm-data-architecture-ofish-app/#the-o-fish-application