MIHIPOTECA APP
MYMORTGAGE APP

TRABAJO FIN DE GRADO
CURSO 2022-2023
CONVOCATORIA: JUNIO 2023

AUTORES

SERGIO ALEJO GARCÍA	GRADO EN INGENIERÍA DEL SOFTWARE
RICARDO CARAZO PÉREZ	GRADO EN INGENIERÍA DEL SOFTWARE
JORGE MORALES LÓPEZ	GRADO EN INGENIERÍA DEL SOFTWARE
CARLOS SOBRADOS RISCO	GRADO EN INGENIERÍA INFORMÁTICA

TUTOR
MANUEL NÚÑEZ GARCÍA

FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

RESUMEN

Nuestro trabajo de fin de grado se llama MiHipoteca App y consiste en una aplicación móvil para comparar distintas opciones de hipoteca en función del tipo de interés y vinculaciones que proponen los distintos bancos. El objetivo es, ofrecer al cliente las mejores ofertas y que pueda compararlas y guardarlas en caso de que le interesen. Además, el usuario podrá introducir los datos de una hipoteca y realizar un seguimiento de ella a través de la aplicación.

La causa por la que decidimos hacer este trabajo es para ofrecer a las personas que no tienen demasiados conocimientos sobre hipotecas la mejor opción para ellos sin necesidad de realizar un estudio pormenorizado e individualizado de las distintas opciones. Aunque ya existen otros simuladores de hipotecas, no es tan frecuente que tengan ambas funcionalidades juntas de manera clara y sencilla.

Comenzamos nuestro proyecto realizando un plan de negocio para determinar la viabilidad y ver en que podíamos destacar con respecto a la competencia. Hemos desarrollado una aplicación nativa para Android utilizando el lenguaje Java y XML, y el IDE *Android Studio.* También hemos utilizado Python para la implementación de la API *(Application Programming Interface*) que obtiene información de las distintas ofertas de hipotecas.

Palabras clave

Hipoteca, cálculos financieros, aplicación móvil, *Android Studio,* API, *web scraping,* servidor, seguimiento hipoteca.

ABSTRACT

Our final degree project is called MiHipoteca App and it consists of a mobile application to compare different mortgage options depending on the interest rate and products proposed by the different banks. The objective is to offer the client the best offers so he can compare and save them in case they interest him. In addition, the user will be able to enter the data of a mortgage and keep track of it through the application.

The reason why we decided to do this work is to offer people who do not have much knowledge about mortgages the best option for them without the need to carry out a detailed and individualized study of the different options. Although other mortgage simulators already exist, it is not so frequent that they have both functionalities together in a clear and simple way.

We started our project by making a business plan to determine the feasibility and see what we could stand out from the competition. We have developed a native Android application using the Java language and XML, and the Android Studio IDE. We have also used Python for the implementation of the API (Application Programming Interface) that obtains information from the different mortgage offers.

Keywords

Mortgage, financial calculations, mobile app, *Android Studio,* API, *web scraping,* server, mortgage monitoring.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

En este capítulo introductorio, se explicarán las razones por las que escogimos el tema del TFG, los objetivos iniciales propuestos, de qué manera se organizó el plan de trabajo y por último la estructuración de la memoria, junto con la explicación sobre cómo acceder al código.

1.1 Motivación

A la hora de escoger nuestro tema de TFG, estábamos interesados en aprender a desarrollar una aplicación móvil y vimos en esta propuesta la oportunidad de llevarlo a cabo. La propuesta consistía en un comparador de ofertas de hipotecas junto con el posible seguimiento de tus propias hipotecas. Vimos en este tema la posibilidad de aprender cómo funcionan y pensamos que sería útil ya que es una decisión que es probable que todos tomemos en algún momento.

1.2 Objetivos

El objetivo final de nuestro trabajo es combinar el seguimiento y la comparación de ofertas de hipotecas en una sola aplicación funcional e intuitiva. Para llevar a cabo este proceso definimos los siguientes objetivos:

- Gestión de usuarios para el uso de la aplicación.
- Comparar ofertas en función de unos datos de entrada introducidos por el usuario en la aplicación.
- Realizar el seguimiento de una hipoteca registrada por el usuario.
- Operaciones y detalles gráficos relacionadas con la evolución de las hipotecas del usuario

1.3 Plan de trabajo

Nos hemos basado en una metodología ágil similar a *Extreme Programming* (XP) ya que sus principios son la calidad del código, la estrecha colaboración de los integrantes del grupo al tener cierta experiencia trabajando juntos. También se basa en principios como la retroalimentación constante ya que cada semana intentamos tener una reunión para poner en común los avances de cada integrante del equipo. Además, hemos optado por la programación en parejas, dividiéndonos los dos principales objetivos de la aplicación. Por un lado, una pareja se centró en el seguimiento de las hipotecas incluyendo operaciones como amortizaciones anticipadas, cuadros de amortización mensual y anual, gráficos detallados con información, etc. La otra pareja se centró en la obtención de las ofertas de hipotecas proporcionadas por los bancos, de datos de utilidad para la comparación de hipotecas y de la configuración del servidor utilizado como intermediario en la obtención de estos datos.

1.4 Estructura de la memoria

Nuestra memoria está estructurada en varios capítulos. En primer lugar, comenzamos explicando el plan de negocio realizado para tener una primera visión de nuestro trabajo de fin de grado en cuanto a su posición en su sector de mercado. También detallamos partes importantes de un plan de negocio tales como propuesta de valor, análisis de competencia, análisis DAFO y un estudio económico financiero que nos ha ayudado a ponernos en distintos escenarios futuros y ver así la rentabilidad o viabilidad de la empresa. La memoria continúa con el capítulo 3, donde nos hemos centrado en explicar los requisitos y funcionalidades necesarias que contiene nuestra aplicación. En esta sección hemos incluido los casos de uso principales necesarios para que el usuario pueda navegar por la aplicación. También hemos incluido en esta sección unos apartados explicando las pantallas más importantes de la aplicación junto con unos bocetos iniciales realizados. En el cuarto capítulo explicamos la arquitectura utilizada y detallamos de manera profunda herramientas, patrones de diseño, técnicas utilizadas y

dificultades encontradas tanto en la parte de *back-end* como en *front-end.* En el quinto capítulo de este proyecto, presentamos nuestras conclusiones tras la realización del mismo. Además, ofrecemos una visión sobre las posibles mejoras y funcionalidades que se podrían implementar en el futuro y que, debido a restricciones de tiempo, no pudimos desarrollar en esta ocasión. Por último, la parte final de la memoria se compone de las contribuciones personales de cada miembro del trabajo junto con la bibliografía y el anexo.

1.5 Acceso a la aplicación y al código

El código de la aplicación se encuentra alojado en un repositorio en GitHub al cual se puede acceder y descargar mediante el siguiente enlace.

https://github.com/Seryiii/TFG-MiHipotecaApp

También hemos realizado un vídeo explicando el funcionamiento de la aplicación el cual se encuentra accesible a través del siguiente enlace.

link al vídeo

Chapter 1 – Introduction

In this introductory chapter, the reasons why we chose the topic of the TFG will be explained, the initial objectives proposed, how the work plan was organized and finally the structure of the memory together with the explanation on how to access the code.

1.1 Motivation

At the time of choosing our TFG theme, we were interested in learning how to develop a mobile application, and we saw in this proposal the opportunity to carry it out. The proposal consisted of a mortgage offer comparator together with the possibility of monitoring your own mortgages. We saw in this topic the advantage of learning how they work, and we thought it would be useful since it is a decision that we are all likely to make at some point.

1.2 Goals

The ultimate goal of our work is to combine the monitoring and comparison of mortgage offers in a single functional and intuitive application. To carry out this process, we define the following objectives:

- User management for the use of the application.
- Compare offers based on input data entered by the user in the application.
- Keep track from a mortgage registered by the user.
- Operations and graphic details related to the evolution of the user's mortgages.

1.3 Work plan

We have based ourselves on an agile methodology similar to Extreme Programming (XP) since its principles are the quality of the code, close collaboration from the group members having some experience working together before. It is also based on principles such as constant feedback, since every week we tried to have a meeting to share the progress from each member of the team. In addition, we have opted for programming in pairs, dividing the two main goals of the application. On the one hand, one couple focused on monitoring mortgages including operations related to them such as early amortizations, monthly and annual amortization tables, detailed information charts, etc. The other couple focused on obtaining mortgage offers provided by banks, useful data for the comparison of mortgages and the configuration of the server used as an intermediary in obtaining this data.

1.4 Manuscript Structure

Our memory is structured in several chapters. In the first place, we begin by explaining the business plan carried out to have a first vision of our final degree project in terms of its position in its market sector. We also detail important parts of a business plan such as value proposition, competition analysis, SWOT analysis and a financial economic study which has helped us to put ourselves in different future scenarios in order to see the profitability or viability of the company. The memory continues with chapter 3, where we have focused on explaining the requirements and necessary functionalities that our application contains. In this section, we have included the main use cases necessary for the user to navigate in the application. We have also included in this part some sections explaining the most important screens of the application together with some initial sketches made. In the fourth chapter we explain the architecture used and we detail in depth the tools, design patterns, techniques used and difficulties encountered both in the *back-end* and in the *front-end*. In the fifth chapter of this project, we present our conclusions after carrying it out. In addition, we offer an insight into possible

improvements and functionalities that could be implemented in the future and that, due to time constraints, we were unable to develop on this occasion. Finally, the final part of the memory consists of the personal contributions of each member of the work together with the bibliography and the annex.

1.5 Access to the application and code

The application code is hosted in a GitHub repository which can be accessed and downloaded through the following link.

https://github.com/Seryiii/TFG-MiHipotecaApp

We have also made a video explaining all the features of the application which is accessible through the following link.

video link

Capítulo 2 - Plan de negocio

Para ver la viabilidad y rentabilidad de la aplicación, hemos desarrollado un plan de negocio incluyendo las partes más importantes del mismo que nos ha ayudado a organizar el proyecto y la hipotética empresa. A continuación, se detalla el plan de negocio desarrollado describiendo cada una de sus partes y los estudios realizados.

2.1 Descripción de MiHipoteca App

MiHipoteca App es una aplicación móvil que proporciona a los usuarios una experiencia completa en el ámbito de las hipotecas. Nuestra misión es ayudar al usuario para que entienda de manera sencilla cómo funciona una hipoteca y las mejores ofertas en función de sus preferencias. Nuestra visión es poder llegar al mayor número de usuarios posibles para ayudar a tomar las mejores decisiones.

Los valores elegidos que queremos que defienda nuestra empresa son los siguientes:

- Calidad
- Compromiso
- Competitividad
- Trabajo en equipo
- Responsabilidad

2.2 Propuesta de valor

Escoger una hipoteca es una acción determinante en la vida de una persona y es necesario ofrecer una aplicación la cual brinde soporte para poder elegir las mejores condiciones posibles de cara a una hipoteca. Con nuestro buscador podrá tener al alcance en unos pocos pasos, las mejores ofertas de cara a acceder a la vivienda de sus sueños.

Nuestro producto está dirigido a personas mayores de edad con un nivel de ahorros considerable, ingresos mensuales y con la idea de obtener una vivienda.

Nuestros socios clave para llevar a cabo nuestro proyecto, van a ser anunciantes y empresas como Google.

Las principales fuentes de ingresos van a ser anuncios por medio de la aplicación, y el posible uso de cuentas premium, en las que los usuarios que decidan usarla obtendrán mayores beneficios y funcionalidades.

Para diferenciarnos de la competencia nos vamos a centrar en la innovación respecto a otras aplicaciones que aportan funcionalidades similares a las de nuestra aplicación. Nuestra principal diferencia se basa en reunir en una sola aplicación tanto la funcionalidad de comparar las mejores ofertas de hipotecas, como el seguimiento de una hipoteca.

Como primer paso ofreceremos nuestro producto a nivel nacional, y, poco a poco, nos iremos expandiendo internacionalmente.

2.3 Análisis de la competencia

Este apartado es uno de los más importantes del plan de negocio, ya que permite identificar las aplicaciones que nos pueden hacer competencia en el mercado, pudiendo conocer sus debilidades y fortalezas, y así adaptar nuestra aplicación para conseguir destacar en el mercado y que los usuarios la escojan como primera opción.

- **Simulador de préstamos**: permite la visualización de cierta información de hipotecas introducidas por el usuario ofreciendo distintos gráficos que hace más entendibles las cantidades a pagar por el comprador. El diseño es poco intuitivo y claro.
- **Idealista**: una de las principales aplicaciones de comparación de ofertas de hipotecas de diferentes bancos en función de la información introducida por el

usuario. También ofrece una información básica de la cuota y gráficos poco detallados.

- **Gibobs**: muy útil para llevar el seguimiento de varias hipotecas a la vez, pero ofrece poca información de ellas. Es una aplicación muy fácil de entender, pero con poca funcionalidad. Permite que las hipotecas sean fijas o variables.
- **Rastreator**: muy útil para comparar diferentes hipotecas ofrecidas por diversos bancos en función de la información proporcionada por el usuario, permitiendo filtrar y ordenar en función de diferentes atributos como bancos, tae, cuota…
- **Calculadora hipotecas Karl**: ofrece un seguimiento de una hipoteca introducida por el usuario, dando la opción de incluir una amortización anticipada de plazo, pero con un diseño pobre y muy poco intuitivo.
- **MiHipoteca**: esta aplicación no se encuentra en la *Play Store*, sino que se trata de una APK externa. Permite la visualización de cuadros de amortización muy detallados y claros, en función de los datos introducidos por un usuario de una determinada hipoteca fija o variable. El diseño es bastante mejorable.

	MiHipotecaApp	Simulador préstamos	Idealista	Gibobs	Rastreator	Calculadora hipotecas karl	MiHipoteca
Elegir tipo de interés al comparar (fija, variable o mixta)	✓	✗	✓	✗	✓	✗	✗
Ver más detalles de la oferta	✓	✗	✓	✗	✓	✗	✗
Guardar ofertas	✓	✗	✗	✗	✗	✗	✗
Ordenar por banco, tae, ... (filtros)	✓	✗	✓	✗	✓	✗	✗
Guardar varias hipotecas	✓	✗	✗	✓	✗	✗	✗
Gráficos con información detallada	✓	✓	✓	✓	✗	✓	✗
Cuadros de amortización	✓	✓	✓	✗	✗	✓	✓
Amortizaciones anticipadas	✓	✗	✗	✗	✗	✓ (solo de plazo)	✓
Hipotecas variables o mixtas en función del euribor histórico	✓	✗	✓ (no mixta)	✓ (no mixta)	✗	✗	✓ (no mixta)

Tabla 1: Análisis de competencia

2.4 Análisis de los clientes

El **cliente objetivo** de nuestra aplicación consiste en personas de entre 25 y 40 años que cuenten con una fuente de ingresos mensual y un porcentaje de ahorro considerable.

La **segmentación** será de tipo contextual, es decir, variará en función del comportamiento del usuario, según sus ahorros y las distintas características que quieren para su hipoteca.

El **nicho principal** de usuarios que usarán nuestra aplicación será de personas interesadas en conseguir su primera hipoteca y que se adapte a sus necesidades y requerimientos. También podemos diferenciar **nichos secundarios** como personas que simplemente quieren informarse de cómo funcionan los distintos tipos de hipotecas o posibles inversores, que buscan adquirir un inmueble para su explotación.

Las **técnicas de captación** que usaremos serán:

- Campañas de publicidad en las diferentes redes sociales.
- Envío masivo de correos.

La forma de **comunicación** que vamos a utilizar para llegar a la gente y que conozcan nuestra aplicación será a través de anuncios de Google en páginas web o en redes sociales. Para llevar a cabo estas campañas de marketing, necesitaremos ciertos recursos tecnológicos como programas de edición para crear anuncios personalizados.

También llevaremos a cabo distintas **estrategias de fidelización** para que los usuarios usen habitualmente la aplicación y no solamente para buscar una sola vez. Algunas de estas estrategias son:

- Seguimiento de la hipoteca contratada de los usuarios registrados.
- Notificaciones al correo electrónico o a la app móvil en cambios que puedan afectar a la hipoteca (Euríbor).
- Noticias relacionadas con las hipotecas que puedan interesar al cliente.

- Nuevas hipotecas ofertadas por diferentes bancos que ofrezcan condiciones que se ajusten más a sus necesidades.

2.5 Estudio económico financiero

El tiempo necesario para la creación de MiHipoteca App, sería de 4 meses de desarrollo, tras los cuales, empezaría a generar ingresos. La inversión inicial que necesita nuestra empresa para empezar a desarrollar la aplicación teniendo en cuenta todos los posibles primeros gastos sería de:

- **Sueldos** de los cuatro programadores encargados de la creación y el soporte de MiHipoteca App durante los 4 meses de desarrollo con un salario de 1.950€ brutos al mes por programador. Los 8 meses restantes contrataremos a un encargado de mantenimiento por 1.300€ al mes brutos.
- Para alojar nuestra aplicación en un **servidor**, hemos tenido en cuenta varios tipos y empresas en función de las prestaciones ofertadas y el precio. Al final nos hemos decidido por alojarla en un servidor de *Amazon Web Service* por 30€ al mes.
- Por último, para darnos a conocer, vamos a llevar a cabo dos tipos de **campañas publicitarias**:
    - Campaña de Twitter: invertiremos un total de 420€, que se traduce en 100.000 visualizaciones de nuestra publicidad, ya que Twitter cobra 4,20€ por cada 1000 visualizaciones.
    - Campaña *Google Ads*: invertiremos un total de 400€, aprovechando una oferta que indica que con los primeros 400€ invertidos, Google regala otros 400€ en publicidad. Teniendo en cuenta que, por cada 1.000 visualizaciones, Google cobra 1,2€, conseguiremos un total de 666.000 visualizaciones de nuestra publicidad. Después del

primer año ya no contaríamos con esa oferta por lo que el coste total de publicidad aumentaría en 400€.

El total de los gastos para esta primera inversión en cuanto al primer año, será de 33.540€.

Los ingresos que vamos a obtener periódicamente con MiHipoteca App, se van a dividir en tres tipos:

- **Suscripción premium**: donde el usuario que pague 2,95€ al mes podrá obtener una serie de privilegios frente a los usuarios que no lo hagan. Algunos de estos privilegios son: seguimiento más detallado de sus hipotecas, posibilidad de guardar ofertas ilimitadas, eliminación completa de los anuncios y seguimientos de hipotecas ilimitados.
- **Anuncios en la aplicación**: otra forma de ingresos va a ser mostrando una serie de anuncios de otras aplicaciones o productos. Los ingresos serían de 3€ por cada 1.000 impresiones
- **Acuerdos con bancos**: negociaremos con bancos para que en función de lo que paguen, sus ofertas de hipotecas aparezcan más arriba y más recomendadas (siempre que cumplan lo que el cliente quiere, claro está). El precio por cada 1.000 visualizaciones de esas ofertas del banco con el que hemos negociado este "plan", sería de 2,90€.

Por otra parte, para mantener esta aplicación funcionando correctamente, una vez ya se haya completado el desarrollo, sería necesario contratar a un encargado de mantenimiento los 8 meses restantes por un total de 1.300€ brutos al mes, que llevará a cabo tareas de mantenimiento, soporte y pequeñas actualizaciones o parches.

A continuación, nos planteamos tres escenarios diferentes (optimista, pesimista, esperado) para ver como iría nuestro proyecto en distintas situaciones. Incluimos unas imágenes y unas tablas para representar la información de manera más clara y visual.

En el segundo año, en cuanto a los costes de personal, en el escenario optimista tendríamos dos trabajadores de mantenimiento con un salario de 1.950€ brutos al mes, en el caso del escenario esperado, un solo encargado de mantenimiento con un salario de 1.625€ brutos y en el pesimista, debido a las grandes pérdidas del primer año, mantendremos el salario del único encargado de mantenimiento en 1.300€ brutos.

Para cada escenario hemos tenido en cuenta distintos porcentajes explicados en la siguiente tabla:

	% Nuevos usuarios Google y Twitter	**% Usuarios que se hacen premium**	**% Nuevos usuarios boca a boca**		**% Usuarios que abandonan la aplicación**
	Año 1 y 2	**Año 1 y 2**	**Año 1**	**Año 2**	**Año 2**
Optimista	3%	5%	3.000 usuarios	8.000 usuarios	30%
Esperado	2%	4%	1.500 usuarios	3.000 usuarios	50%
Pesimista	1%	3%	500 usuarios	1.250 usuarios	70%

Tabla 2: Porcentajes aplicados estudio económico financiero

Escenario Optimista

1er año

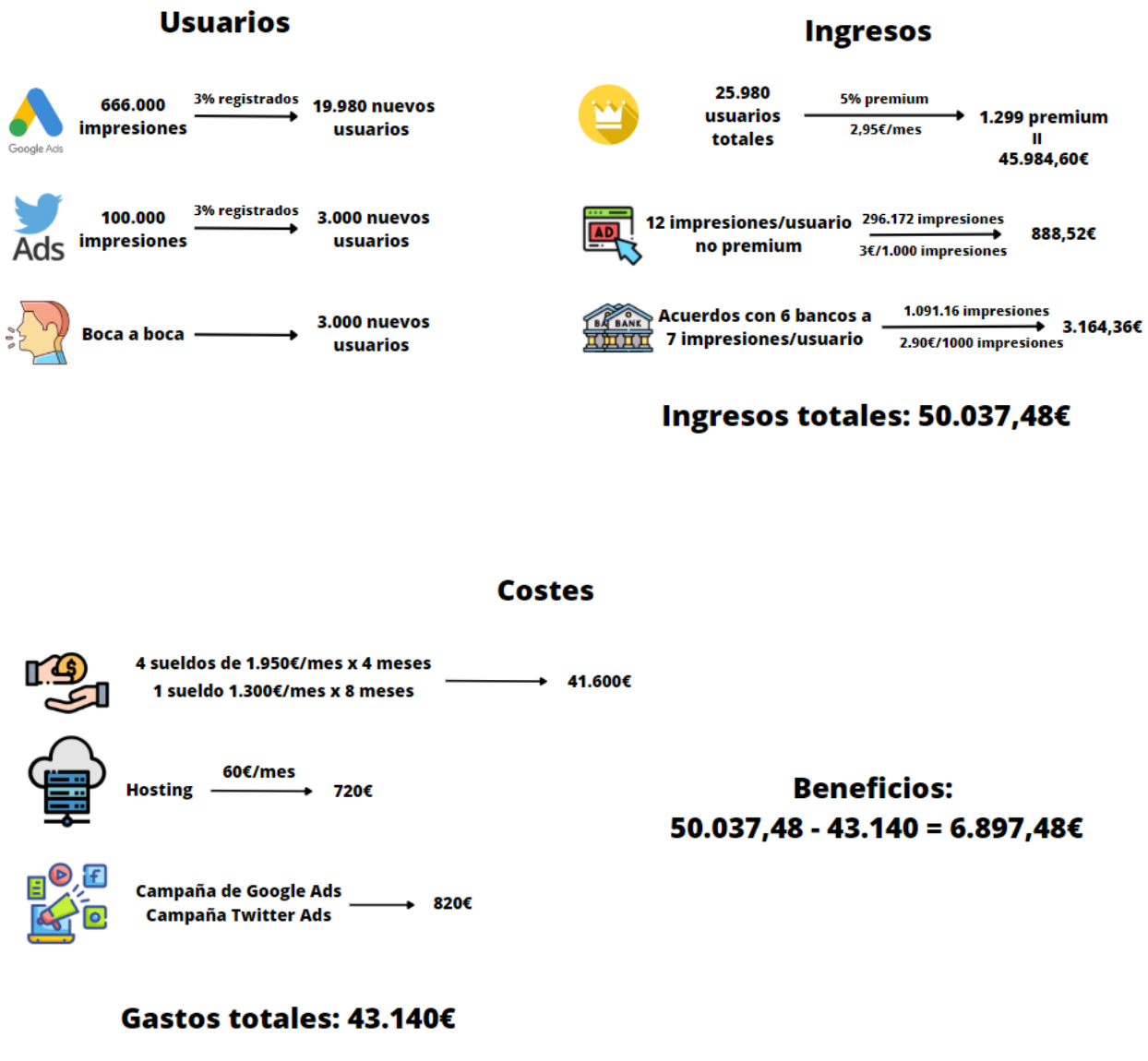

Ilustración 1: Beneficios y Costes Escenario Optimista año 1

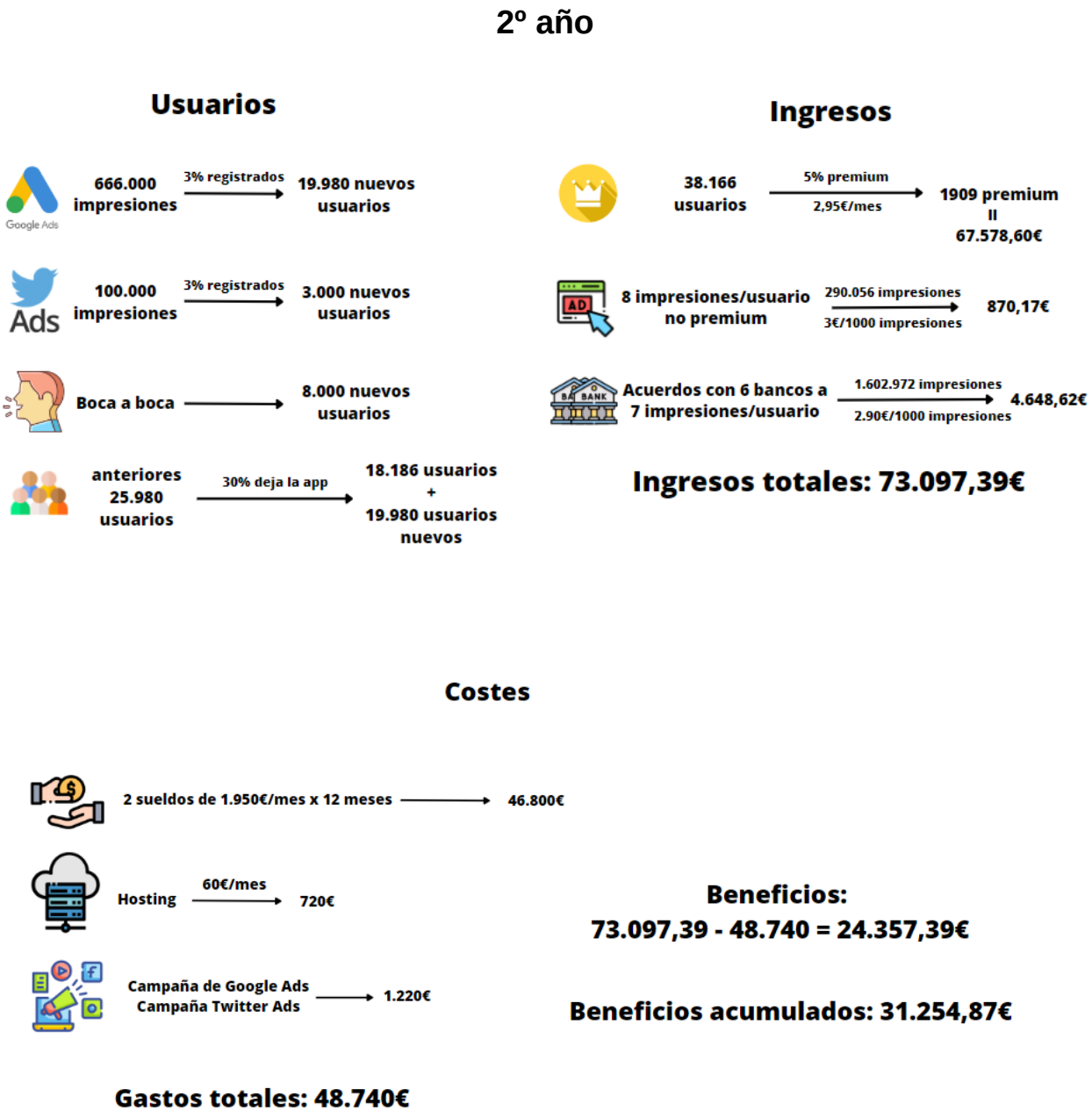

Ilustración 2: Beneficios y Costes Escenario Optimista año 2

Años	1er Año	2º Año
Ingresos	50.037,48€	73.097,39€
Costes Fijos	43.140€	48.740€
Costes Variables	0€	0€
Beneficios / Pérdidas	6.897,48€	24.357,39€
Punto Muerto	43.140€	48.740€
Pérdidas/Beneficios Ac.	6.897,48€	31.254,87€

Tabla 3: Escenario Optimista

Escenario Esperado

1er año

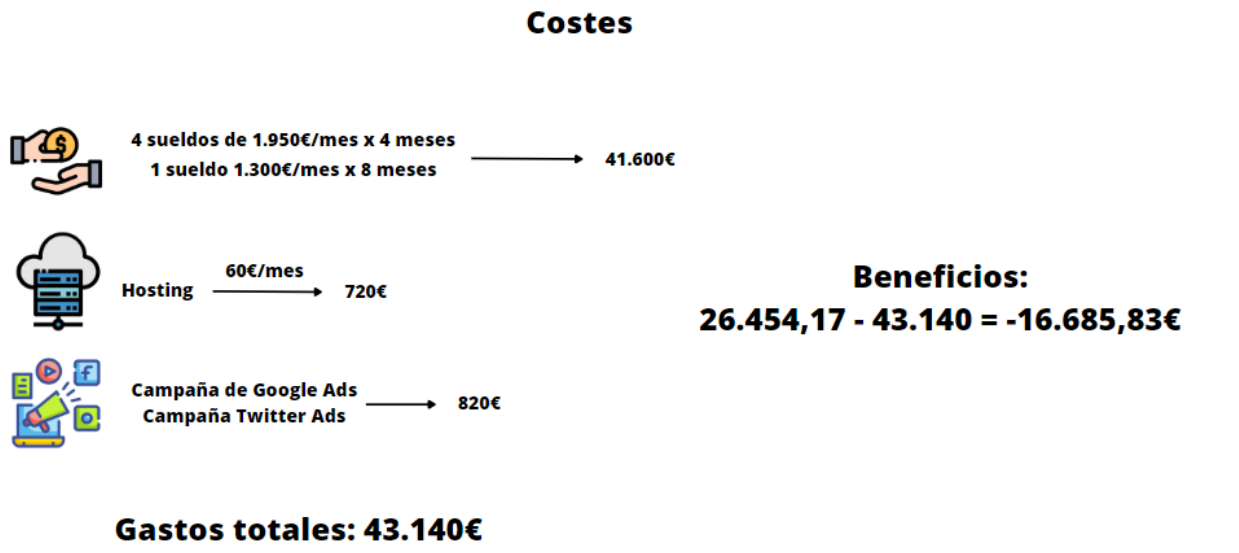

Ilustración 3: Beneficios y Costes Escenario Esperado año 1

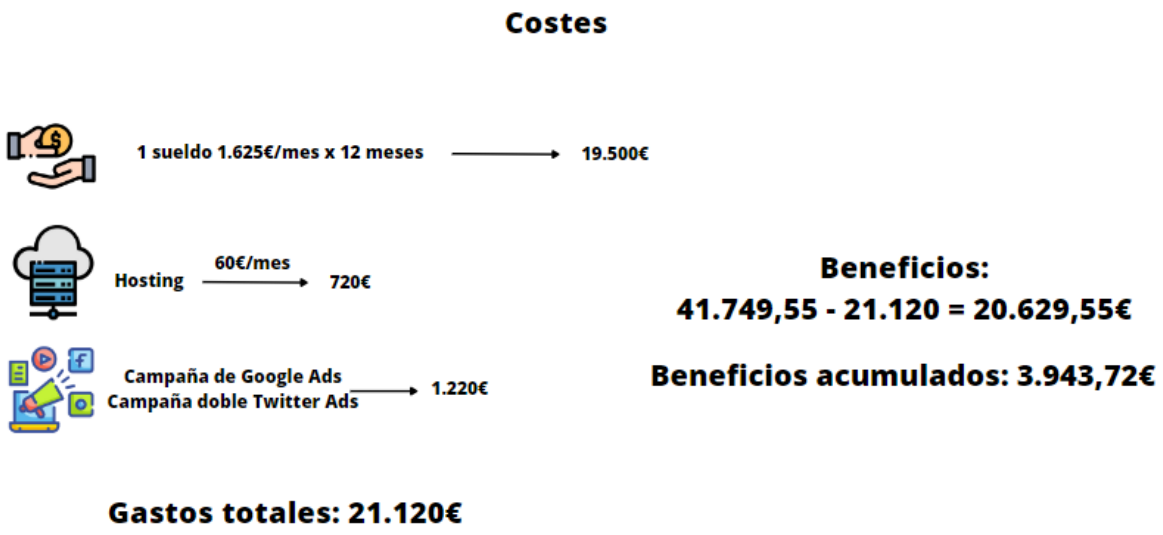

Ilustración 4: Beneficios y Costes Escenario Esperado año 2

Años	1er Año	2º Año
Ingresos	26454,17€	41.749,55€
Costes Fijos	43.140€	21.120€
Costes Variables	0€	0€
Beneficios / Pérdidas	-16.685,83€	20.629,55€
Punto Muerto	43.140€	21.120€
Pérdidas/Beneficios Ac.	-16.685,83€	3.943,72€

Tabla 4: Escenario Esperado

Escenario Pesimista

1er año

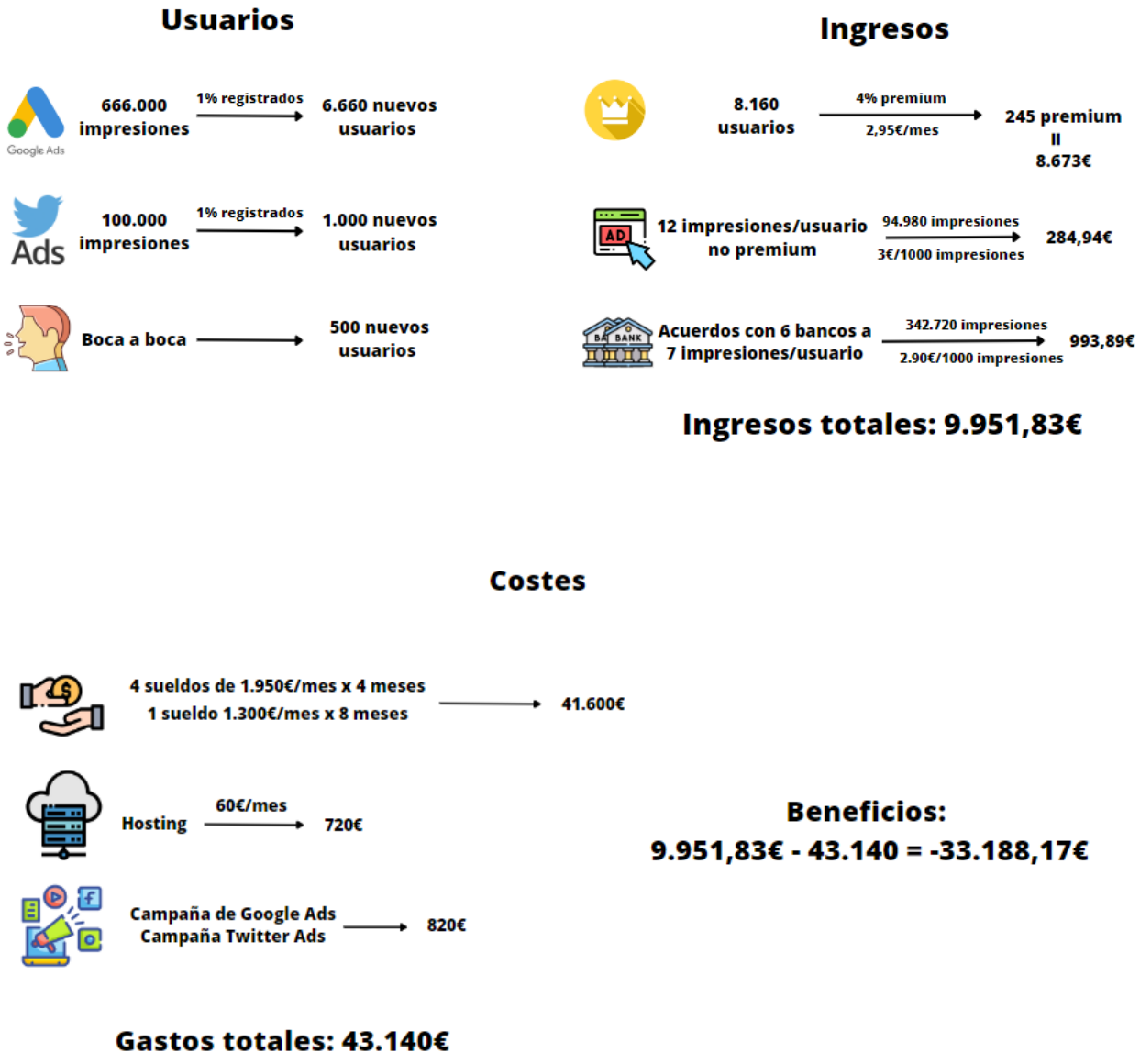

Ilustración 5: Beneficios y Costes Escenario Pesimista año 1

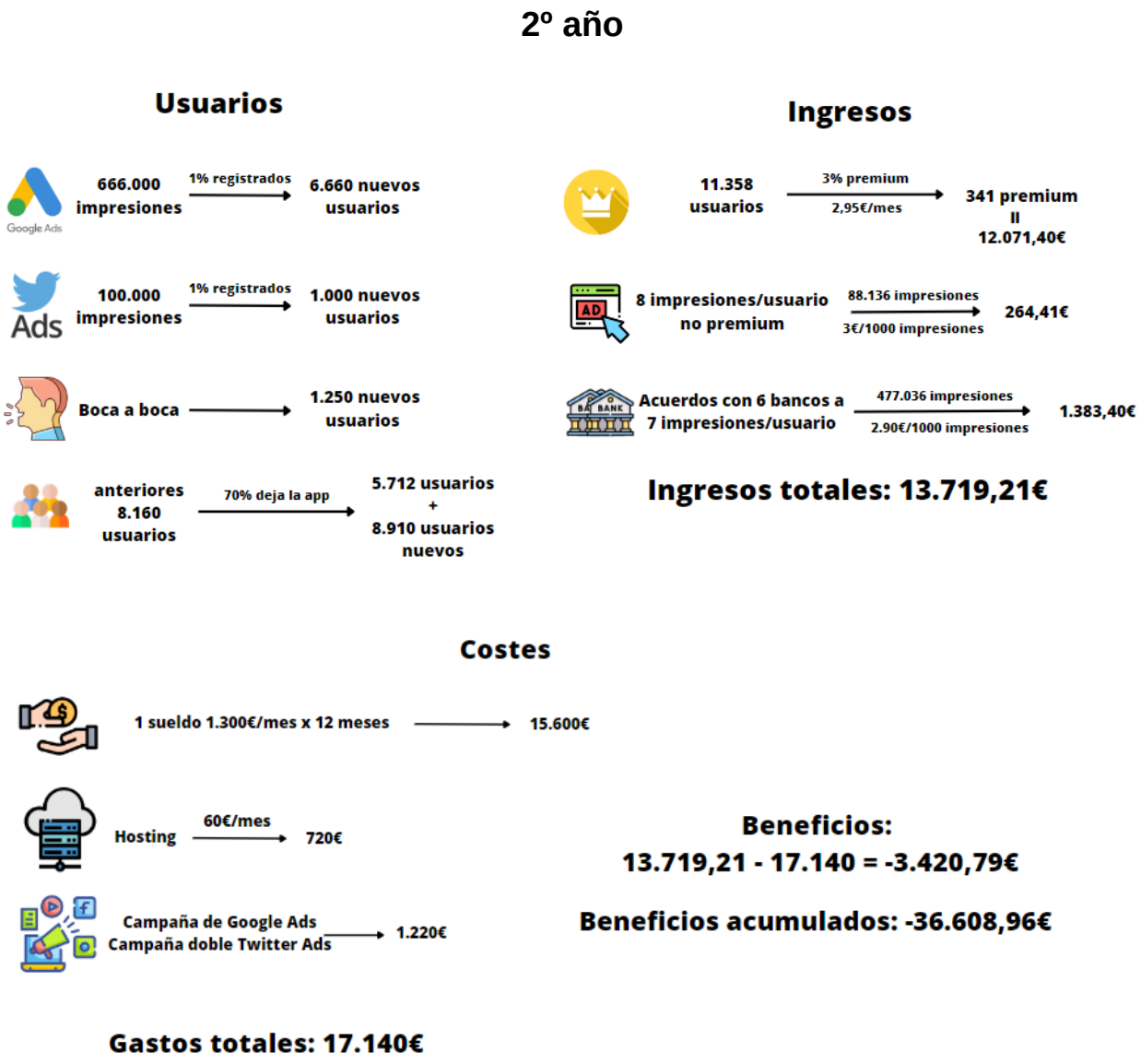

Ilustración 6: Beneficios y Costes Escenario Pesimista año 2

Años	1er Año	2º Año
Ingresos	9.951,83€	13.719,21€
Costes Fijos	43.140€	17.140€
Costes Variables	0€	0€
Beneficios / Pérdidas	-33.188,17€	-3.420,79€
Punto Muerto	43.140€	17.140€
Pérdidas/Beneficios Ac.	-33.188,17€	-36.608,96€

Tabla 5: Escenario Pesimista

Tras analizar los tres hipotéticos escenarios, hemos observado una diferencia considerable de beneficios entre el escenario optimista y el pesimista. En cambio, en el esperado se mantiene un cierto equilibrio, incurriendo en pérdidas en el primer año, pero obteniendo tras el segundo año unos beneficios de 3.943,72€ indicándonos que el proyecto podría ser rentable.

2.6 Análisis DAFO

El análisis DAFO es una herramienta muy útil para conocer la situación de una empresa. La sigla DAFO, significa Debilidades, Amenazas, Fortalezas y Oportunidades, que sirven para identificar factores internos y externos de nuestra empresa.

Análisis interno

- **Fortalezas**: como aplicación nos diferenciamos de la competencia, porque combinamos las dos funcionalidades principales de seguimiento y comparación de una hipoteca, debido a que la inmensa mayoría de las empresas solo implementa una de ellas, otorgándonos así una posición aventajada respecto a nuestros competidores.

- **Debilidades:** debido a la falta de recursos, conocimientos y de personal, no podemos competir directamente con las grandes empresas especializadas en este sector, debido a nuestra inexperiencia no podemos contactar con las entidades bancarias para que nos proporcionen recursos para la continua integración de nuestra aplicación.

Análisis externo

- **Oportunidades:** debido a la inflación del sector inmobiliario en los últimos años, las múltiples complicaciones para conseguir una hipoteca con unas condiciones que se ajusten a las necesidades del consumidor y el auge de las aplicaciones tecnológicas hacen que la población se vea más interesada en una aplicación que agrupe todos los gastos y ofertas relacionadas con el entorno inmobiliario y que esté siempre a su disposición.
- **Amenazas:** actualmente debido a la constante incertidumbre, la sociedad está más desmotivada en adquirir una hipoteca y prefiere el alquiler de una vivienda. Produciendo así una posible pérdida del cliente objetivo de nuestra aplicación.

  Por otro lado, al contar con escasos recursos, en el caso de que la aplicación tuviese éxito, las grandes empresas podrían fijarse en la combinación de estas dos grandes funcionalidades dejando nuestra aplicación atrás en cuanto al mercado se refiere.

Capítulo 3 - Propuesta y diseño

En primer lugar, se describen las funcionalidades básicas de la aplicación junto con algunos de los casos de uso más importantes. Se han incluido tablas y diagramas de actividad para facilitar la comprensión de los mismos. A continuación, explicamos los bocetos iniciales diseñados, la estructura principal y los colores escogidos de la aplicación. Finalmente incorporamos una explicación de las distintas vistas de la aplicación detallando su contenido.

3.1 Funcionalidades básicas y casos de uso

Esta sección está dividida en tres subsecciones. En la primera se detallan los requisitos relacionados con el usuario, la segunda contiene los requisitos de sistema y la tercera incluye los casos de uso principales para que el usuario inicie su experiencia en la aplicación. El resto de casos de uso se incluyen en el anexo del documento.

3.1.1 Requisitos de usuario

En esta subsección definiremos las necesidades y requerimientos para el usuario final de nuestra aplicación, en ella detallamos las tareas que podrá realizar cada tipo de usuario. En primer lugar, definimos los tipos de usuarios los cuales son:

- Usuario no registrado.
- Usuario registrado con cuenta normal.
- Usuario registrado con cuenta premium.

Los **usuarios no registrados** constan de los siguientes requisitos:

- No podrán acceder a realizar un seguimiento de una hipoteca.
- Los usuarios no registrados no contarán con las funcionalidades de gestión de su propio usuario al no tener una cuenta registrada en la aplicación.

- Podrán comparar las posibles ofertas, pero sin la posibilidad de guardarlas.

Los **usuarios registrados con cuenta normal**:

- Podrán registrar sus hipotecas para realizar el seguimiento, pero con un límite de registros.
- Podrán acceder a todas las funcionalidades de modificación de usuario proporcionadas por la aplicación.
- Una vez iniciado el seguimiento, alguna información como la visualización de gráficos no estará disponibles.
- Podrán comparar ofertas y guardarlas para posteriormente visualizar y comparar sus ofertas preferidas.

Los **usuarios registrados con cuenta premium**:

- Podrán registrar sus hipotecas para realizar el seguimiento sin límite de registros.
- Podrán acceder a todas las funcionalidades de modificación de usuario proporcionadas por la aplicación.
- Una vez iniciado el seguimiento, se habilita la visualización de gráficos.
- Podrán guardar ofertas ilimitadas.

3.1.2 Requisitos de sistema

Por otro lado, enfocándonos en las necesidades de la aplicación, definimos una serie de requisitos de sistema:

- La aplicación será compatible con dispositivos Android, con una versión Android 11 o superior o a través de un emulador Android.
- Tanto para la funcionalidad de comparar ofertas como para el uso de la base de datos, será necesario tener acceso a internet.

- El servidor que aloja la API para realizar peticiones debe de estar disponible las 24 horas del día, los 7 días de la semana.
- La aplicación debe tener medidas de encriptación para asegurar la gestión de contraseñas de los usuarios.
- La aplicación debe cumplir con el Reglamento General de Protección de Datos.

3.1.3 Casos de uso

En esta sección hemos decidido incluir los casos de uso principales describiendo tareas específicas realizadas por el usuario hacia el sistema y la respuesta que este ofrece. Hemos escogido los casos de uso de registrar usuario, nuevo seguimiento y comparar hipotecas, ya que son los principales para empezar a utilizar la aplicación.

Registrar usuario

Descripción	Registra a un nuevo usuario en la BD
Necesita	*Firebase Authentication, Firebase Firestore*
Actor	Usuario
Entrada	Nombre usuario, correo, contraseña, imagen de perfil, suscripción
Salida	ID generado por la BD o mensaje de error
Origen	Interfaz de usuario (Vista → Inicio de sesión)
Destino	*Firebase Authentication, Firebase Firestore*
Precondición	No haya un usuario con el correo introducido
Postcondición si éxito	Dar de alta al nuevo usuario en *Firebase Authentication*, se crea un nuevo documento en la colección de usuarios en *Firebase Firestore*
Postcondición si fallo	Mensaje de error, redirección al formulario de registro

Tabla 6: Caso de uso registrar usuario

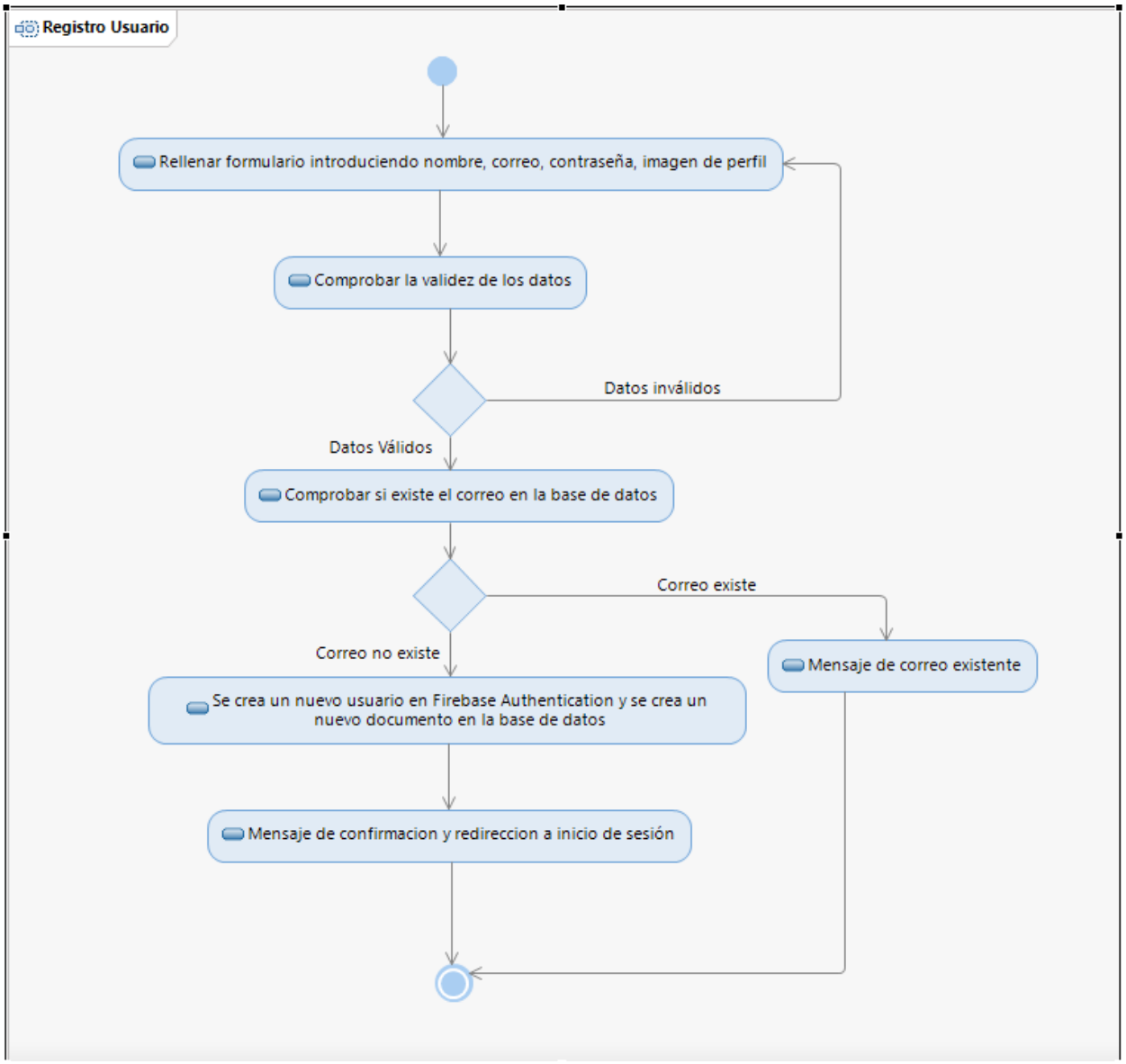

Ilustración 7: Diagrama de actividad registro usuario

Nuevo seguimiento

Descripción	Nuevo seguimiento
Necesita	Base de datos, aplicación
Actor	Usuario *logueado* en la aplicación
Entrada	Rellenar el formulario nuevo seguimiento: ● Comunidad autónoma ● Banco asociado ● Tipo de vivienda ● Antigüedad de vivienda (nueva | segunda mano) ● Precio de la vivienda ● Cantidad abonada ● Plazo en años ● Fecha de inicio ● Tipo de hipoteca ● Porcentajes dependiendo del tipo de hipoteca ● Gastos de la hipoteca ● Nombre de la hipoteca
Salida	Dar de alta el nuevo seguimiento en la base de datos
Origen	Interfaz de usuario
Destino	Base de datos
Precondición	Usuario *logueado* en la aplicación
Postcondición si éxito	Seguimiento registrado en base de datos y redirección a vista de mis hipotecas con el nuevo seguimiento
Postcondición si fallo	No se guarda el seguimiento y se notifica al usuario de que los datos introducidos no son correctos

Tabla 7: Caso de uso nuevo seguimiento

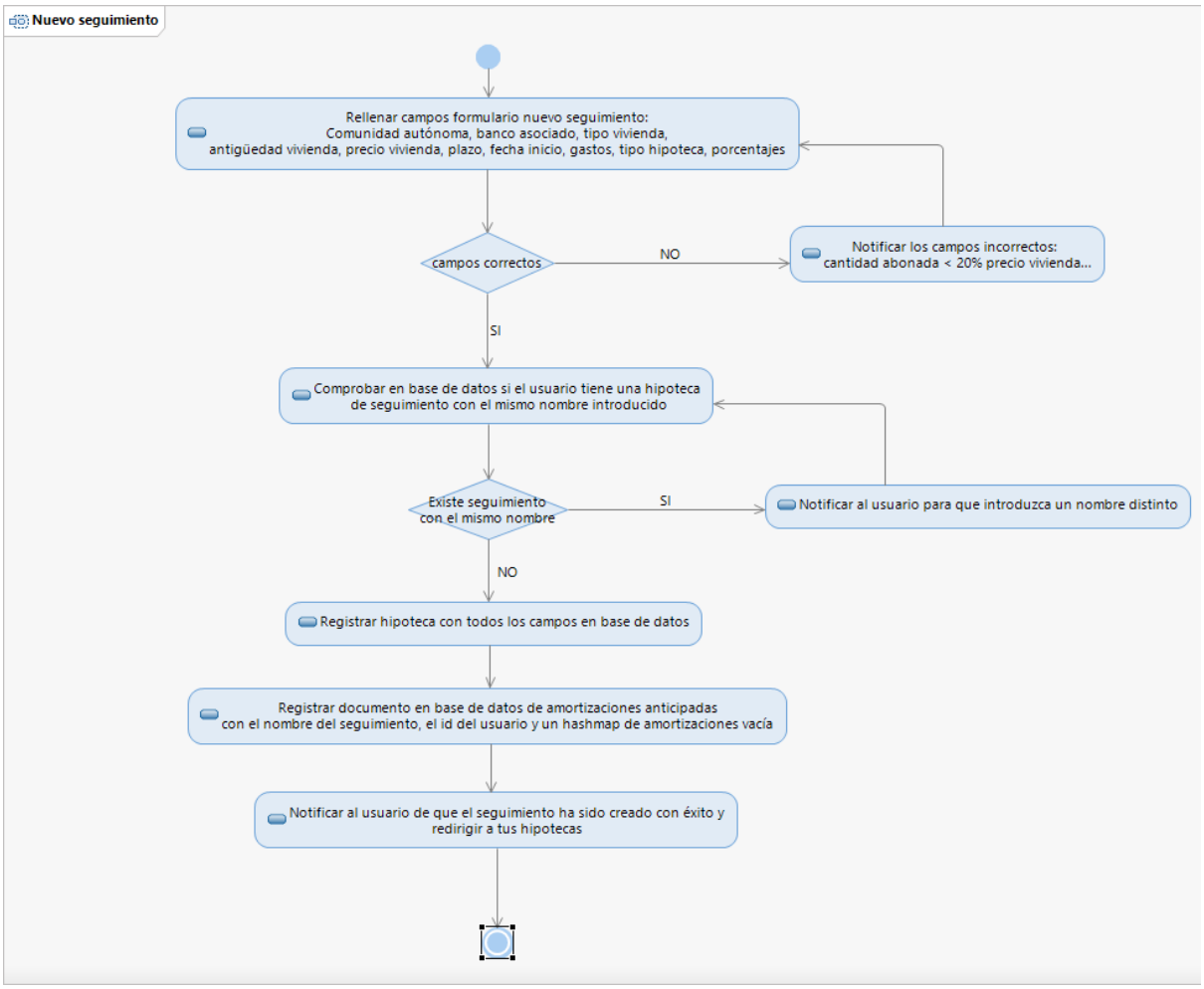

Ilustración 8: Diagrama de actividad nuevo seguimiento

Comparar hipotecas

Descripción	Comparar hipotecas
Necesita	Base de datos, aplicación, conexión con aplicación web
Actor	Usuario: ● Caso 1 → registrado en aplicación ● Caso 2 → no registrado en aplicación
Entrada	Rellenar formulario comparación hipoteca: ● Tipo de vivienda ● Provincia ● Antigüedad vivienda ● Precio vivienda ● Cantidad abonada por comprador ● Plazo a pagar el préstamo ● Ingresos mensuales ● Búsqueda detallada
Salida	Caso 1 → lista ofertas con opción de guardado Caso 2 → lista ofertas sin opción de guardado
Origen	Interfaz de usuario
Destino	Interfaz de usuario
Precondición	Caso 1 → usuario *logueado* en aplicación Caso 2 → usuario no registrado en aplicación
Postcondición si éxito	Muestra la lista con las ofertas obtenidas
Postcondición si fallo	Mensaje de error, redirección a la vista de rellenar formulario

Tabla 8: Caso de uso comparar hipotecas

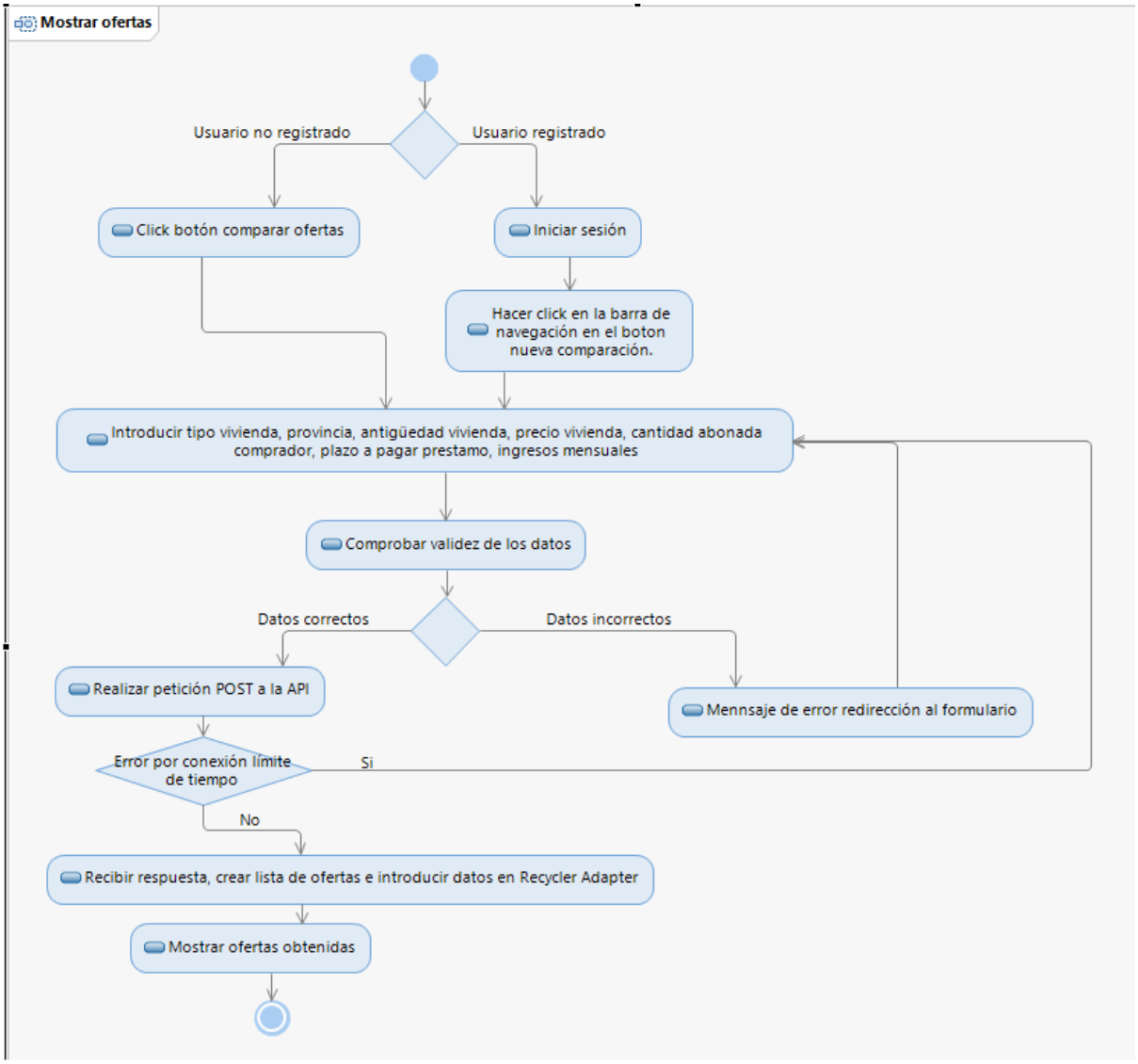

Ilustración 9: Diagrama de actividad comparar hipotecas

3.2 Bocetos iniciales

Para la creación de las vistas de nuestra aplicación realizamos varias fases de investigación con el fin de hacer una aplicación fácil de entender para el usuario y que éste se pudiese adaptar a ella en el menor tiempo posible sin necesidad de tutorial. Para ello realizamos unos primeros bocetos en papel y los pusimos en común entre los

miembros del equipo para ver qué partes de cada dibujo escoger indicando las ventajas y los inconvenientes de cada una de las vistas.

Una vez elegidos los componentes de las principales vistas, elegimos la herramienta *Figma* para crear nuestros propios *mockups* para digitalizar esos bocetos y comprobar si eran correctos vistos en una pantalla. *Figma* es una herramienta *software* online gratuita que permite crear diferentes proyectos individuales o grupales para elaborar *mockups* de las vistas de una aplicación, tanto web como móvil. Además, también se pueden crear conexiones entre las diferentes vistas para simular el uso de un usuario final de la aplicación. Esto, junto a la facilidad de uso y la cantidad de componentes posibles que ofrece *Figma*, hace que sea una herramienta software clave en el diseño de nuestra aplicación, ya que nos ha permitido probar diferentes diseños de vista usando gran cantidad de componentes, sin necesidad de programarlos. También nos ha servido para elegir los diferentes colores de los componentes de la aplicación, los símbolos asociados a algunos de los botones, la proporcionalidad de los elementos de las vistas, etc. Los diseños realizados con *Figma* pueden consultarse en el anexo al final del documento.

Ilustración 10: Logotipo Figma

Para elegir la tonalidad de la aplicación hemos realizado una pequeña investigación sobre la psicología del color, llegando a la conclusión de que la gama de colores que más se adecuaba a nuestra aplicación son los tonos azules. Esto es debido a que este color está relacionado, entre otras cosas, con la verdad, la seriedad, el modernismo y el desarrollo tecnológico, además de la relajación y la seguridad.

Desde nuestro punto de vista, estos matices son los que queremos transmitir con nuestra aplicación, y una de las formas de hacerlo es usando este tipo de colores en ella, ya que,

por lo que hemos estudiado de diferentes fuentes, el color de una aplicación móvil o web, dictamina las primeras impresiones de los usuarios al entrar en ella. También hemos elegido un tema de aplicación oscuro, porque queremos conseguir que algunos elementos destaquen y así logramos captar una mayor atención del usuario. Además, investigando, hemos leído varios artículos en los que confirman que el modo oscuro usado en los móviles mejora la experiencia del usuario, protegiendo la vista de este o disminuyendo la batería del dispositivo más lentamente.

3.3 Explicación de pantallas

En esta sección, vamos a explicar detalladamente las pantallas más importantes de nuestra aplicación. Está estructurada en cuatro pantallas principales (Mis Hipotecas, Nuevo Seguimiento, Comparar Hipotecas y Mi Perfil), que contienen las funcionalidades más importantes. Para acceder a cada una de ellas, tenemos un componente llamado *bottom navigation menu* en la parte inferior, explicado de manera más detallada en el apartado “4.2.1 Estructura *front-end*”.

La pantalla principal de la aplicación se llama Mis Hipotecas y en la parte superior contiene un título informativo junto con el avatar escogido por el usuario. A continuación, aparece un título y un emoticono haciendo referencia a las ofertas guardadas por el usuario pudiendo acceder a ellas haciendo clic en ambos elementos.

Por último, se encuentra la información en forma de tarjeta de todas las hipotecas de seguimiento que haya registrado el usuario con el logotipo del banco asociado, el título del seguimiento y el tipo de la hipoteca. Cuando se hace clic en una de ellas, se habilitan tres opciones por medio de un *pop-up menu* y se puede visualizar, editar o eliminar el seguimiento. La última tarjeta o primera, en caso de no tener seguimientos registrados, contiene un botón para añadir un nuevo seguimiento.

Ilustración 11: Vista Mis Hipotecas

Antes de continuar con la siguiente pantalla principal, vamos a explicar una pantalla secundaria, “Visualizar Hipoteca Seguimiento”, a la que se accede cuando se pulsa en una tarjeta y se selecciona en el menú la opción de visualizar, ya que contiene bastante información útil para el usuario. En primer lugar, aparece información sobre el tipo de hipoteca, banco asociado, dinero restante por pagar incluyendo tanto capital como intereses y tiempo restante.

Ilustración 12: Vista Visualizar Seguimiento 1

Después aparecen dos botones, el primero redirige a otra pantalla mostrando tanto el cuadro de amortización mensual como el anual y el otro botón redirige a otra pantalla permitiendo realizar cualquier tipo de amortización anticipada.

Ilustración 13: Vista Cuadro de Amortización

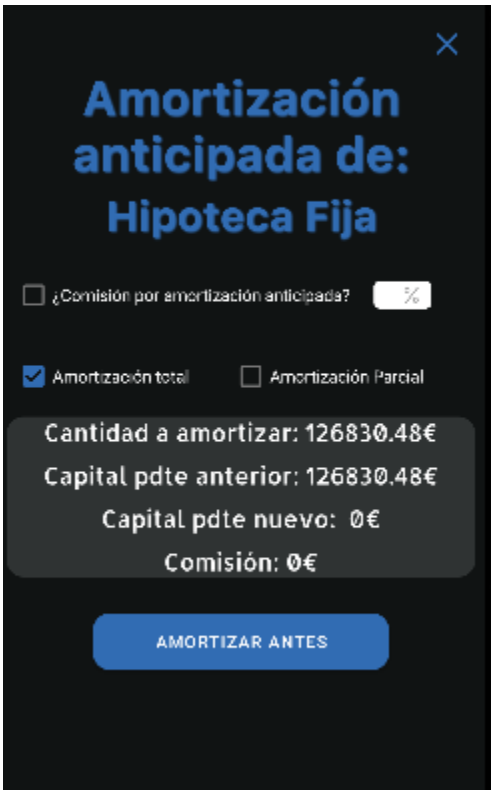

Ilustración 14: Vista Amortización anticipada

Siguiendo con la información que contiene la pantalla “Visualizar”, aparece una sección con toda la información relacionada con la siguiente cuota mensual. En primer lugar, aparece el nombre del mes de la siguiente cuota, la cantidad, el porcentaje, el capital junto con los intereses desglosados y el número de cuotas pagadas. En caso de tener una amortización anticipada en la siguiente cuota, incluiríamos un apartado indicando la cantidad de capital amortizado y un botón para eliminarla.

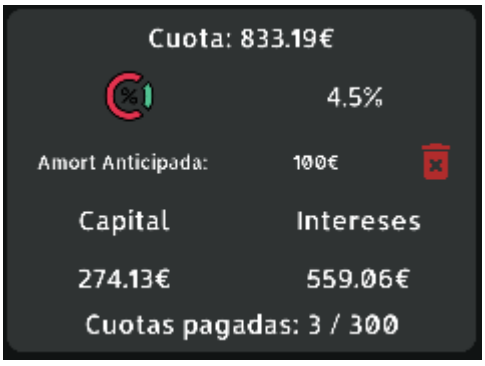

Ilustración 15: Vista Visualizar Seguimiento 3

Gracias a la incorporación del botón eliminar amortización anticipada, el usuario puede ver cómo afecta esa amortización a su hipoteca y en caso de no interesarle, borrarla.

Seguidamente la pantalla contiene otra sección con dos botones para visualizar dos gráficos distintos, uno relacionado con los gastos totales de la hipoteca y otro que muestra información comparando intereses con el capital, la cuota y las vinculaciones.

Ilustración 16: Vista Visualizar Seguimiento 4

El primero es un gráfico de sectores que muestra los gastos totales de la hipoteca dividiéndolos en función de vinculaciones, otros gastos relacionados con gestoría, comisiones, tasación, registro, … y luego otros dos sectores relacionados con impuestos que dependiendo de si es una vivienda nueva se aplica el IVA y en caso de vivienda de segunda mano se aplica el ITP (impuesto transmisiones patrimoniales) y por otro lado el AJD (impuesto de actos jurídicos documentados).

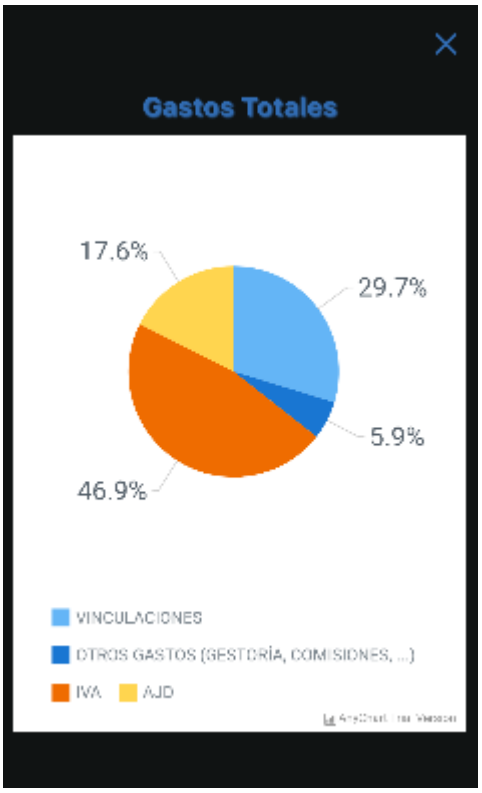

Ilustración 17: Vista Gráfico Gastos Totales

El segundo es un gráfico de líneas en el que se puede ver la evolución anual de la cuota, intereses, capital amortizado y vinculaciones, para poder ver así el punto en el que el usuario empieza a pagar más capital que intereses y cómo varían tanto cuotas como las vinculaciones a lo largo de la vida de la hipoteca.

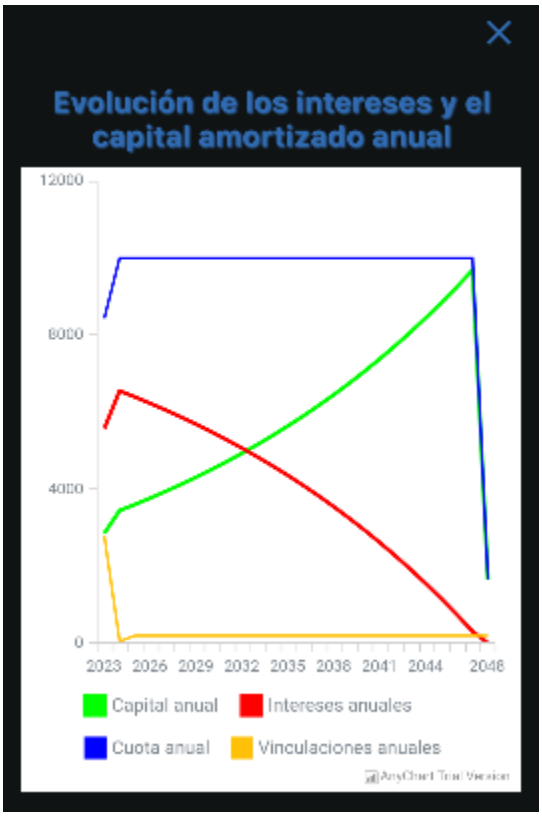

Ilustración 18: Vista Gráfico Capital vs Intereses

Llegando a la parte final incluimos un tercer gráfico de sectores con información relacionada con el dinero aportado en comparación con el dinero por financiar. Dividimos el gráfico en cuatro sectores: capital amortizado, capital pendiente, intereses pagados e intereses pendientes. También completamos este gráfico con una tabla en su parte inferior para ver las cifras de manera más clara.

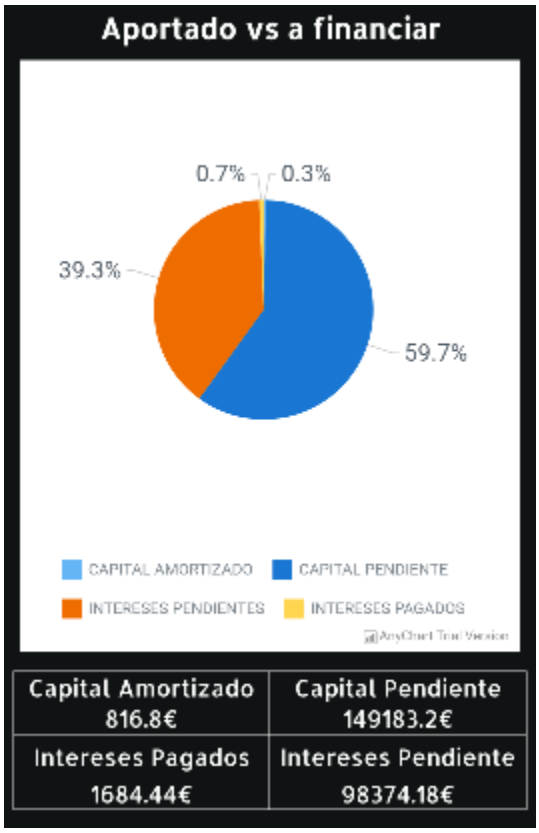

Ilustración 19: Vista Visualizar Seguimiento 5

Para acabar con la pantalla “Visualizar”, la más extensa de nuestra aplicación, en caso de tener la hipoteca de seguimiento amortizaciones anticipadas, incluimos una tabla con la información de las mismas: número de cuota en la que se realizó, tipo, capital amortizado, meses reducidos (en caso de no ser de plazo se muestran tres guiones) y comisión total en caso de tenerla. Por otro lado, se muestra un texto indicando al usuario que no tiene amortizaciones anticipadas.

Ilustración 20: Vista Visualizar Seguimiento 6

Continuando con la pantalla “Nuevo Seguimiento”, que contiene un formulario pidiendo al usuario toda la información necesaria que necesitamos para hacer el seguimiento de una hipoteca: precio de la vivienda, cantidad abonada, tipo de hipoteca, porcentajes aplicados entre otros.

Ilustración 21: Vista Nuevo Seguimiento

Seguimos con la pantalla principal “Comparar Hipotecas”, en la cual, el usuario rellena un formulario parecido al de nuevo seguimiento, pero en este caso para realizar una búsqueda comparando las distintas ofertas de los bancos.

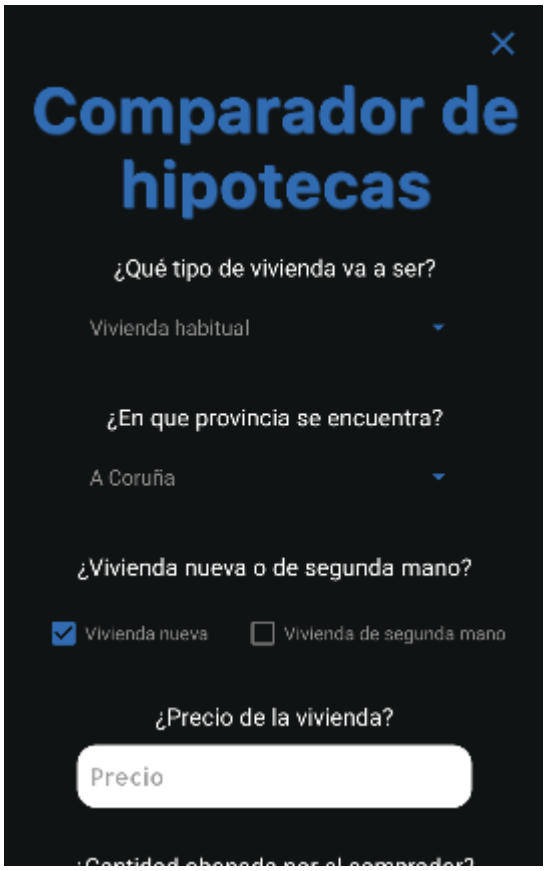

Ilustración 22: Vista Comparar Hipotecas

Además, en esta pantalla hemos incluido un switch por si el usuario quiere obtener unas ofertas con un mayor nivel de detalle incluyendo vinculaciones y escoge las que tienen mejor TAE y cuota.

Una vez pulsado el botón de comparar ofertas, se redirige a la siguiente actividad, “Mostrar Ofertas”, en la que se listan mediante unas tarjetas toda la información acerca de las ofertas obtenidas. Esta información procede de realizar una petición POST a nuestra API, que realiza la técnica de *web scraping* sobre la página Rastreator recogiendo la información y devolviéndola como respuesta en un objeto JSON a la aplicación móvil.

Además, esta vista ofrece aplicar una serie de filtros, filtrar por banco dejando las tarjetas del banco seleccionado. Más abajo encontramos ordenar por TAE y por cuota, mostrando las tarjetas y ordenándolas de mayor a menor de arriba a abajo.

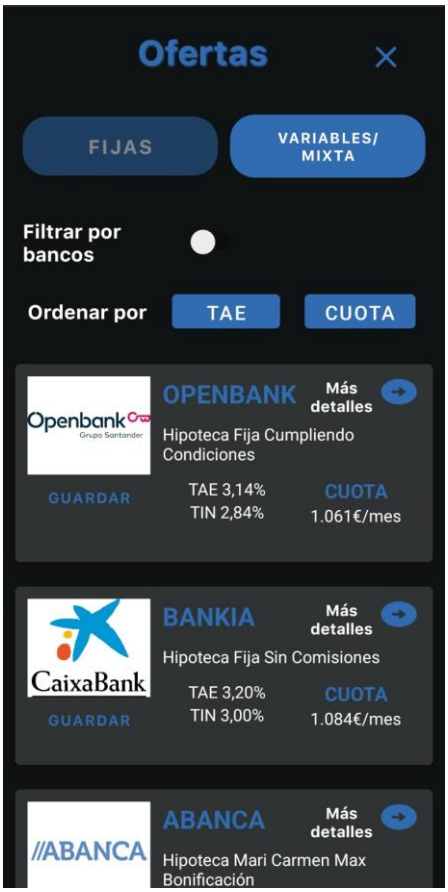

Ilustración 23:Mostrar Ofertas

Finalmente, la última página principal sería la de “Mi Perfil”, que contiene todas las opciones relacionadas con el usuario como modificar perfil, pasar a premium, cerrar sesión, eliminar cuenta, etc.

Ilustración 24: Vista Mi Perfil

Capítulo 4 - Arquitectura

En este capítulo, se explica de manera detallada la arquitectura utilizada tanto en la parte de *back-end* como en *front-end.* Se incluyen herramientas elegidas, algunos patrones de diseño y dificultades encontradas durante el desarrollo.

4. 1 Backend

En la parte de *back-end* explicamos la base de datos escogida tras probar otras opciones inicialmente y cómo se estructura la información en la misma. Además, tuvimos que desarrollar una API para hacer peticiones desde *Android Studio* y por medio de *web scraping* recoger información necesaria para las ofertas de las hipotecas. Para alojar dicha API, fue necesario un servidor y hemos reservado una subsección explicando cómo se configuró para poder hacer las peticiones de forma remota.

4.1.1 Lenguajes utilizados

4.1.1.1 Java vs Kotlin

A la hora de desarrollar una aplicación por medio de *Android Studio* nos encontramos con dos principales opciones: utilizar el lenguaje de programación Java o utilizar Kotlin. Para decidir qué lenguaje íbamos a utilizar, vimos que los cuatro miembros teníamos mucha experiencia y conocimientos de Java adquiridos en la carrera, mientras que del lenguaje Kotlin ninguno teníamos conocimientos.

Comparando ambas opciones, Kotlin es un lenguaje de programación nacido en 2010 por *JetBrains*, y tiene como objetivo reemplazar al lenguaje Java. Algunas de las ventajas que ofrece Kotlin es que su curva de aprendizaje es corta debido a su simple sintaxis y cuenta con un excelente soporte para *Android Studio*.

Por otro lado, Java es un lenguaje orientado a objetos independiente de plataforma y cuenta con un recolector de basura que permite liberar y optimizar la memoria. Aunque

ambas opciones eran buenas, debido a la diferencia en cuánto a conocimientos entre ambos lenguajes, nos acabamos decidiendo por Java para desarrollar nuestra aplicación.

4.1.1.2 Python

Python es un lenguaje de programación que, en nuestro caso, utilizamos para la realización de los *scripts* y la creación de una API que explicaremos posteriormente. Este nos ofrece una gran cantidad de librerías que nos permiten de manera sencilla y rápida poder realizar las técnicas de *web scraping* y el formateo de objetos JSON.

Además, estas herramientas proporcionan una gran cantidad de funcionalidades y abstracciones que hacen que la creación de una API sea mucho más fácil y rápida que si se hiciera desde cero.

4.1.1.3 *Shell* de *Bash* (Unix)

La *Shell* de Unix es un intérprete de comandos que proporciona una interfaz de usuario para acceder al sistema operativo de Unix/Linux, el servidor proporcionado por la universidad en *Jupyter*, utiliza la *shell* de *bash* como intérprete para el sistema operativo, lo que hace la *shell* es leer los comandos que escribes en la terminal, y ejecuta las acciones correspondientes en el sistema operativo. *Shell* ofrece una gran cantidad de comandos que nos han servido para modificar permisos, instalar y configurar servicios como *Google Chrome* con sus licencias o *Chromedriver*, necesarios para implementar las funcionalidades de nuestra aplicación.

4.1.2 Base de datos

Para la base de datos investigamos distintas opciones para ver cuál sería la más adecuada para nuestro proyecto. En primera instancia, decidimos utilizar una base de

datos relacional SQL, debido a que los cuatro miembros estabamos más familiarizados a trabajar con este tipo de bases de datos.

4.1.2.1 Base de datos relacional con Xampp, librería *Volley* y *Web Service*

La primera opción seleccionada consistía en crear una base de datos relacional local en *XAMPP* y por medio de un *web service* en PHP, que actuando de intermediario entre *Android Studio* y la base de datos, hiciese las operaciones y devolviese los datos correspondientes a *Android Studio*. Las peticiones al *web service*, se harían utilizando una librería de Android llamada *Volley*. Tras plantear esta opción y empezar a adaptar el código en *Android Studio*, nos dimos cuenta de que la conexión entre *Android Studio* y el *web service*, se hacía por medio del protocolo HTTP*,* el cual no es muy seguro y entonces decidimos investigar sobre una plataforma muy potente llamada *Firebase* que ofrecía distintos servicios para base de datos y era muy útil en el ámbito de desarrollo móvil mejorando la calidad y el rendimiento de las aplicaciones desarrolladas.

4.1.2.2 Firebase

Tras dedicar un tiempo a investigar qué era realmente *Firebase*, decidimos cambiar y utilizar esta plataforma ya que nos solucionaba el problema de utilizar HTTP y nos sería más fácil para gestionar los usuarios de la aplicación.

Hemos utilizado dos servicios que proporciona *Firebase.* Por un lado, hemos utilizado *Firebase Authentication,* servicio encargado de guardar los datos de los usuarios en la nube de forma segura. De todas las opciones que ofrecía escogimos la autenticación por medio de un correo electrónico y una contraseña.

Para guardar los datos generados en nuestra aplicación decidimos escoger *Firebase Firestore,* una base de datos NoSQL flexible y escalable. La base de datos de *Firestore,* se basa en un modelo de documentos y colecciones. Los datos se almacenan en

documentos JSON, que contienen información sobre un elemento en particular. Estos documentos se organizan en colecciones.

Para este servicio tuvimos que formarnos primero para entender el funcionamiento de las bases de datos NoSQL y luego configuramos el proyecto para incorporar las dependencias necesarias para la plataforma *Firebase* junto con *Firestore* y *Authentication*.

Tras escoger esta segunda opción, no se volvió a cambiar, ya que funcionaba rápido y de manera segura.

4.1.2.3 Estructuración de los datos

Para la estructura de la base de datos de la aplicación, se crean cinco colecciones detalladas a continuación:

- **Usuarios**: colección que incluye toda la información de un usuario registrado en la aplicación. Sus campos son:
  - correo: campo único que identifica al usuario.
  - password: contraseña del usuario.
  - nombre: nombre del usuario.
  - avatar: campo entero para cargar una foto (avatar) escogido por el usuario.
  - premium: booleano que indica si el usuario es un usuario premium o un usuario normal.
- **Hipotecas_seguimiento**: colección que incluye todos los campos necesarios de una hipoteca para posteriormente hacer su seguimiento. Los campos que contiene son:

- nombre_hipoteca: nombre único para la hipoteca dentro de las hipotecas del usuario.
- precio_vivienda: precio total de la vivienda.
- cantidad_abonada: ahorro aportado por el usuario.
- comunidad_autónoma: comunidad autónoma de la vivienda.
- fecha_inicio: fecha de cuando se inicia la hipoteca, el mismo día del mes de la fecha de inicio se realizarán los pagos de cada cuota.
- plazo_anios: número de años que va a durar la hipoteca.
- tipo_vivienda: se indica si es vivienda de régimen general o de protección oficial.
- arrayVinculacionesAnual: array con la cantidad que se paga de vinculaciones por año. Cada nuevo año se rellena con los gastos de vinculaciones correspondientes por si hubiera alguna variación.
- idUsuario: UID del usuario al que corresponde la hipoteca introducida.
- antiguedad_vivienda: se indica si es vivienda de primera o segunda mano.
- banco_asociado: se guarda un *string* indicando el banco asociado para luego cargar el logotipo del banco.
- totalGastos: se guarda el total de gastos que corresponden a gestoría, notaría, tasación, registro y comisiones.
- tipo_hipoteca: el tipo de hipoteca puede ser fijo, variable o mixta.
- porcentaje_fijo: en caso de ser una hipoteca fija, en otro caso valdría cero.
- revisión_anual: booleano que en caso de ser true indica revisión anual y en otro caso revisión cada seis meses.

- duración_primer_porcentaje_variable: en las hipotecas variables, por lo general se suele aplicar un primer porcentaje fijo antes de empezar con el Euríbor y el diferencial, por lo que, en caso de que se aplique, se indica la duración en meses y si no en caso contrario se pone a cero.
- primer_porcentaje_variable: primer porcentaje aplicado en la hipoteca variable en caso de que exista.
- porcentaje_diferencial_variable: diferencial que se va aplicar junto con el Euríbor en la hipoteca variable.
- anios_fija_mixta: duración en años de la parte fija en una hipoteca mixta.
- porcentaje_fijo_mixta: porcentaje fijo aplicado en una hipoteca mixta.
- porcentaje_diferencial_variable: diferencial que se va aplicar junto con el Euríbor en la hipoteca mixta.

- **Amortizaciones_anticipadas**: colección que incluye todas las amortizaciones anticipadas realizadas en una hipoteca de seguimiento. Contiene los siguientes campos:
  - idUsuario: UID del usuario que se utiliza en conjunto con el nombre de la hipoteca para identificar cada una de las hipotecas de seguimiento.
  - nombre_hipoteca: nombre de la hipoteca de seguimiento al que pertenecen las amortizaciones.
  - amortizaciones_anticipadas: es un *Hashmap* que contiene como clave el número de cuota en la que se realiza una amortización anticipada y como valor depende del tipo de amortización:

1. Amortización anticipada total y amortización anticipada parcial de reducción de cuota: se guarda una lista con dos valores; el primero para indicar el tipo de amortización (“total” o “parcial_cuota”) y la segunda indicando la cantidad total de capital de dicha amortización.
2. Amortización anticipada parcial de reducción de plazo: se guarda una lista con tres valores; el tipo (“parcial_plazo”), la cantidad de capital de dicha amortización y el número de meses reducidos.

   Se guarda también un campo con el porcentaje de la comisión aplicada. En caso de no tener comisión, el porcentaje sería cero.

- **Impuestos**: colección que se utiliza para calcular los impuestos en función de la comunidad autónoma. Contiene los siguientes campos:
  - AJD_HIPOTECA: impuesto de actos jurídicos documentados que se aplica para obras de segunda mano.
  - AJD_OBRA_NUEVA: impuesto de actos jurídicos documentados que se aplica para obras nuevas.
  - ITP: impuesto aplicado cuando se trata de una vivienda de segunda mano.
  - IVA: impuesto aplicado en caso de vivienda nueva y de régimen general.
  - IVA_PO: impuesto aplicado en caso de vivienda nueva y de protección oficial.
- **Euríbor**: colección creada y rellenada por medio de un script de *web scraping* que contiene todos los valores del Euríbor por mes y año desde el año 1999 hasta el año y mes actual. El día 1 de cada mes se ejecuta un

script para cargar el valor del Euríbor del mes actual. Los campos de esta colección son:

- anio: año correspondiente al Euríbor.
- mes: mes correspondiente al valor del Euríbor.
- valor: valor del Euríbor del mes y año correspondiente.

- **Ofertas_guardadas**: colección que guarda la información de una oferta de hipoteca guardada por el usuario una vez hecha la comparación de hipotecas. Contiene los siguientes campos:
  - banco: banco asociado a la oferta de la hipoteca.
  - tipo: tipo de la hipoteca: fija, variable o mixta. Algunos campos que contiene la oferta varían en función del tipo:
    - Fija:
      - cuota: cantidad mensual fija de la hipoteca.
      - tin: porcentaje aplicado durante la hipoteca.
    - Variable / Mixta:
      - cuota_x: cuota de los primeros años de la hipoteca.
      - cuota_resto: cuota del resto de los años.
      - tin_x_anios: porcentaje TIN aplicado para los primeros años.
      - tin_resto: información que muestra el porcentaje diferencial junto con el Euríbor.
  - desc: título de la oferta de la hipoteca.
  - idUser: id del usuario que está *logueado* y guarda la oferta.
  - nombreOferta: nombre elegido por el usuario para identificar la oferta.

- tae: porcentaje TAE de la oferta.
- vinculaciones: información de las vinculaciones que tiene la oferta, en caso de no tener se guarda con un *string* vacío.

4.1.3 Servidor

4.1.3.1 Configuración del servidor

Para la configuración del servidor, el cual estaba implementado en Linux, tuvimos que aplicar los conocimientos aprendidos en asignaturas cursadas durante la carrera, como Ampliación de Sistemas y Redes, Seguridad en Redes y prácticas en empresa como ingeniero de sistemas. Nos proporcionaron un servidor con el puerto 5.000 abierto, con Python 3 instalado, requisitos para poder tener nuestra API funcionando en el servidor. Además, nos facilitaron una carpeta compartida en el usuario de Ricardo, al que pudiéramos acceder con cada uno de los usuarios que componen el grupo.

Para el correcto funcionamiento del servidor y de la aplicación alojada en ella, tuvimos que instalar varios programas y ejecutables, como *Chromedriver*, Google Chrome o la licencia de uso de Google Chrome y librerías de Python usadas para la API, más el posterior cambio de permisos para que pudiésemos acceder todos los integrantes del proyecto a dichos recursos. También se tuvo que hacer la importación de credenciales de la base de datos de *Firebase*, para poder interactuar con ella, añadiendo entradas, o leyendo datos de la misma.

4.1.3.2 Ansible

Ansible es un motor *open source* que automatiza los procesos para preparar la infraestructura, gestionar la configuración, implementar las aplicaciones y organizar los sistemas, entre otros procedimientos de TI. Con la automatización que ofrece Ansible, puede instalar los sistemas de software, automatizar las tareas diarias, preparar la

infraestructura, mejorar la seguridad y el cumplimiento, ejecutar parches en los sistemas y compartir la automatización en toda la empresa. En nuestro proyecto hemos utilizado esta herramienta principalmente para mejorar la seguridad de nuestra aplicación. Específicamente lo hemos utilizado para cifrar las credenciales de la base de datos, y que no sean visibles de cara a cualquier persona ajena al desarrollo del proyecto. Utilizamos una herramienta que nos provee Ansible, cómo es *ansible-vault*, herramienta con la cual conseguimos cifrar las credenciales de la base de datos, y que el fichero en el que están alojadas sea ilegible, ejecutando el comando *ansible-vault encrypt* [ruta al fichero a encriptar], solicitando una contraseña de encriptación, que se pida cuando se quiera desencriptar el archivo. Para que dichas credenciales se puedan seguir utilizando, las mismas se desencriptan ejecutando el comando *ansible-vault decrypt* como primer paso en el momento que se ejecuta el *script* que pone a funcionar la API, de manera que solo estén descifradas las claves en el momento de lectura de las credenciales.

```python
from ansible.parsing.vault import VaultLib

archivo_encriptado = '/srv/home/salejo/TFG/mihipotecaapp-4b30a-firebase-adminsdk-uubno-841ef0be20.vault'

# Ejecutar el comando de desencriptación utilizando ansible-vault
proceso = subprocess.run(
    ["ansible-vault", "decrypt", archivo_encriptado, "--output=-"],
    input=clave_vault.encode(),
    capture_output=True,
)

# Verificar el resultado del proceso
if proceso.returncode == 0:
    contenido_desencriptado = proceso.stdout.decode()

    # Convertir el contenido desencriptado a formato JSON
    contenido_json = json.loads(contenido_desencriptado)

    # Aquí puedes trabajar con el contenido JSON como desees
    credentials_json = json.loads(contenido_desencriptado)
    credentials_firebase=credentials.Certificate(credentials_json)

    firebase_admin.initialize_app(credentials_firebase)
    db = firestore.client()
```

Ilustración 25:Desencriptado para lectura de credenciales de Firestore

4.1.4 Herramientas utilizadas

4.1.4.1 *Android Studio*

Android Studio es un entorno de desarrollo integrado (IDE), que nos permite escribir, depurar y probar diferentes proyectos en un solo lugar, además tiene un emulador de dispositivos que nos permite probar nuestra aplicación en diferentes tamaños y resoluciones de pantalla, lo que nos ayuda a asegurarnos de que MiHipoteca App se vea bien en diferentes dispositivos. A destacar también la compatibilidad del entorno con Java, lenguaje que conocemos y dominamos, cosa que nos ha agilizado el proceso de desarrollo de la aplicación. Asimismo, *Android Studio* incluye una gran cantidad de bibliotecas de soporte que nos han permitido agregar funcionalidades avanzadas a nuestra aplicación. Algunas de ellas han sido: *Awesome Validator*, que es una biblioteca de validación de formularios, que utilizamos para la validación de entradas de datos en la aplicación y *Anychart*, que es una biblioteca que nos ha ayudado a incorporar gráficos y cuadros estadísticos durante el desarrollo de la aplicación.

Otra de las bibliotecas compatible con este entorno de desarrollo, que nos ha ayudado a lograr los requisitos de nuestro proyecto, es la biblioteca *Volley*, la cual permite hacer peticiones HTTP, ayuda a la gestión de operaciones de red, como la ejecución y enrutamiento de las solicitudes de red.

En conclusión, debido a todas las ventajas que nos ofrece *Android Studio*, ha sido el principal entorno de desarrollo que hemos utilizado durante todo el desarrollo del TFG.

4.1.4.2 *Visual Studio Code*

Visual Studio Code es un entorno de desarrollo moderno y altamente personalizable que nos permite escribir y depurar código de manera eficiente. Su interfaz intuitiva, amplia compatibilidad con lenguajes de programación y su amplia gama de extensiones hacen que nos haya sido muy cómodo desarrollar gran parte del trabajo relacionado con *web*

scraping, además de la integración con la terminal, que nos ha sido muy útil para levantar la versión beta de la API en local, y ejecutando test sobre la funcionalidad de la misma.

4.1.4.3 PyCharm

PyCharm es un Entorno de Desarrollo Integrado (IDE) orientado a la programación en Python, nos ofrece una amplia variedad de herramientas y funcionalidades que nos facilitan el desarrollo de aplicaciones, ayudándonos así a trabajar de una manera más rápida y eficiente.

Durante nuestra experiencia aplicando técnicas de *web scraping* por medio de *Selenium* en lenguaje Python, hemos encontrado en *PyCharm* una herramienta muy útil y completa que nos ha permitido trabajar con mayor comodidad. Las características que destacamos de este entorno son el autocompletado de código, la integración con sistemas de control de versiones, herramientas para refactorizar código y la capacidad y facilidad de hacer *debugging*.

Además, *PyCharm* nos ofrece la posibilidad de trabajar con distintos frameworks y tecnologías relacionadas con Python, como *ApScheduler*, *Flask*, *Selenium* entre otras. Esto nos ha facilitado la práctica de *web scraping*, ya que muchas de las tecnologías nombradas anteriormente han sido necesarias para llevar a cabo dicha práctica.

4.1.4.4 Google Drive

Además, con el objetivo de mantener todos nuestros documentos almacenados de forma ordenada, decidimos utilizar Google Drive. Esta elección se basó tanto en nuestro conocimiento con la plataforma como en su capacidad para editar los documentos en tiempo real, lo cual resultó sumamente beneficioso para nuestro proyecto. De esta manera, pudimos colaborar de manera efectiva y mantener nuestros archivos organizados en carpetas.

También hemos utilizado Google Drive para hacer el seguimiento de los avances de cada miembro, asignarnos nuevas tareas y tener un sitio centralizado donde se refleje el estado del desarrollo del proyecto en todo momento.

4.1.4.5 GitHub

En nuestro proyecto era necesario tener un sistema de control de versiones para poder trabajar los cuatro miembros de forma colaborativa. Decidimos escoger GitHub debido a la gran cantidad de funciones que nos ofrecía y ya que era fácil trabajar con este sistema debido a la previa experiencia que nos ha proporcionado la carrera.

GitHub es una plataforma para alojar proyectos utilizando el sistema de control de versiones de Git. Es ampliamente utilizado por empresas tecnológicas y en la comunidad de software libre y código abierto. GitHub ofrece un historial completo de los cambios realizados en el código de un proyecto y puedes revertir los cambios a un punto concreto si deseas volver a una versión anterior. En nuestro proyecto utilizamos la herramienta *GitHub desktop*, ya que es muy útil para subir los cambios realizados haciendo tan solo unos clics y también la herramienta proporcionada por *Android Studio* de Git.

4.1.5 Web Scraping

Web scraping es una técnica de extracción de datos de páginas web de manera automatizada que nos ha permitido la obtención de varios datos como el Euríbor, el TIN y TAE ofrecidos por los distintos bancos.

4.1.5.1 *Selenium*

Selenium es una herramienta que se utiliza para automatizar la interacción con sitios web y, por lo tanto, puede ser utilizada para realizar *web scraping.* Gracias a esta herramienta nos ha permitido seleccionar los distintos elementos del código HTML, podemos seleccionar un elemento en función de su id, nombre de la clase o el XPATH que es un

lenguaje que nos permite identificar los elementos utilizando una sintaxis de ruta de acceso, ya que muchos de estos a veces no tienen id único con el que poder seleccionarlo o tienen la misma clase que otros elementos por lo que si queremos seleccionar un elemento en concreto y varios elementos tienen la misma clase no podremos acceder a él de una manera rápida y efectiva y, posteriormente, poder interactuar con el elemento.

```
var_precio = datos["precio_vivienda"]
input_precio = WebDriverWait(driver,10).until(EC.presence_of_element_located((By.XPATH,"(//input[@placeholder='Ej: 250.000€'])[1]")))
input_precio.send_keys(Keys.CONTROL + "a")
input_precio.send_keys(var_precio)
```

Ilustración 26: Selección de elemento con Selenium

4.1.5.2 *BeautifulSoup*

Por otro lado, tenemos *BeautifulSoup* otra librería que nos permite extraer información de las páginas web. Sin embargo, a diferencia de *Selenium*, *BeautifulSoup* no puede interactuar con los elementos de una página web, es decir, no puede hacer clic en botones, enviar formularios o realizar otras acciones interactivas. Esto significa que, si se necesita interactuar con una página web para extraer información, *BeautifulSoup* no es suficiente, pero nos ha sido muy útil como complemento para poder visualizar el código HTML de las páginas web sobre todo de cara a la integración de la API con el servidor, debido a la falta de interfaz gráfica del mismo.

4.1.5.3 *Flask*

Para la creación de la API hemos utilizado un *framework* de Python que nos permite una creación de una aplicación web de manera rápida y sencilla ya que no se necesitan más librerías. Para poder comunicarnos con esta a través de *Android Studio* lanzamos peticiones tanto GET como POST en función de las necesidades requeridas y esta las recibe ejecutando el código que se encuentra en la ruta especificada. Devolviendo, así como respuesta un objeto JSON con los datos solicitados. Las rutas definidas en la API

son para obtener tanto el Euríbor histórico como el mensual y el diario, además de obtener las ofertas ofrecidas por los bancos.

4.1.5.4 *ApScheduler*

Para poder programar la ejecución de funciones que se ejecuten diariamente, o el primer día de cada mes, hemos añadido las bibliotecas de Python *ApScheduler*, que viene de *flask_apscheduler*, donde, una vez iniciada esta aplicación, se podrían añadir con *ApScheduler*, *jobs*, que se ejecutarán con una cierta frecuencia, llamando así a otras funciones alojadas en el script de la API.

```
scheduler=APScheduler()
scheduler.add_job(id="euribor-historico", func=euriborHistorico, trigger="cron", day="1", hour="11")
scheduler.add_job(id="euribor_diario", func=euriborDiario, trigger="cron", day_of_week="mon-fri", hour="12")
scheduler.init_app(app)
scheduler.start()
```

Ilustración 27: ApScheduler configuración

4.1.6 Patrones de diseño

4.1.6.1 *Adapter*

El patrón *Adapter* se encarga de convertir la interfaz de un objeto, para que pueda ser comprendida por otro objeto. El motivo principal por el que hemos implementado este patrón es para la creación de listas dinámicas en las vistas que requerían de esta implementación. Hemos implementado este patrón, para que tras hacer *web scraping*, y obtener las ofertas de las hipotecas, hacer de ellas un objeto entendible para el cliente, entre otras funcionalidades.

Las vistas en las que hemos implementado este patrón, han sido desarrolladas mediante el uso del elemento *RecyclerView* propio de Android, el cuál explicamos a continuación.

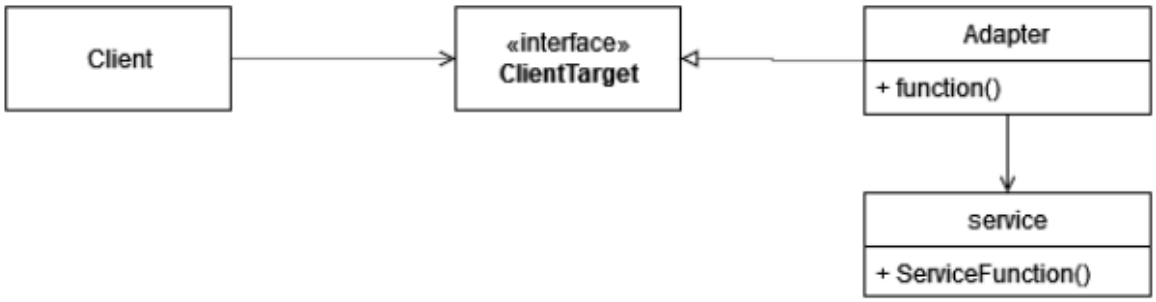

Ilustración 28: Patrón Adapter

Un *widget* proporcionado por *Android Studio* que nos permite visualizar una lista de elementos pudiendo hacer *scroll* en la pantalla. Se utiliza para enumerar las ofertas recibidas por el script de Python a través de una lista, permitiendo visualizar cada una de ellas de manera personalizada, flexible y con un mayor rendimiento a diferencia de otros widgets como el ListView ya que este utiliza el patrón de diseño ViewHolder para reciclar y reutilizar las vistas de los elementos. También es utilizado a la hora de mostras las hipotecas asociadas a cada usuario.

Para poder implementar el *RecyclerView* es necesario crear los siguientes subelementos que nos permitirán visualizar todos los elementos de la lista y poder recorrer cada uno de ellos:

1. Por un lado, tendremos que crear la vista donde queremos alojar nuestro *RecyclerView* para mostrar la lista.
2. Crearemos la vista de cómo queremos que se visualice cada uno de los elementos de la lista.
3. Crearemos un adaptador para el *RecyclerView*, *Android Studio* nos proporciona una opción suya, pero si queremos personalizarlo, que será la mayoría de los casos, lo tendremos que crear extendiendo la clase con *RecyclerView*.Adapter que nos proporciona tres métodos *onCreateViewHolder*, *onBindViewHolder* y *getItemCount*. Estos métodos sirven para proporcionar la vista que hemos creado de los elementos a cada posición de la lista proporcionada al constructor.

4. Hacer una clase *viewholder* que nos servirá para identificar los elementos como un *TextView* de la vista de un elemento de la lista. Una vez configurados cada uno de ellos podremos combinarlos con *card views* u otros elementos para dar más personalización a la parte del *front-end*.

4.1.6.2 *Transfer*

Uno de los patrones utilizados en nuestro proyecto, ha sido el patrón transferencia o *transfer*. Lo hemos utilizado para transportar toda la información en una sola llamada. Algunos ejemplos de los objetos *transfer* utilizados en nuestro proyecto son: Oferta, Usuario y Euríbor, los cuales no proveen lógica de aplicación ni validaciones.

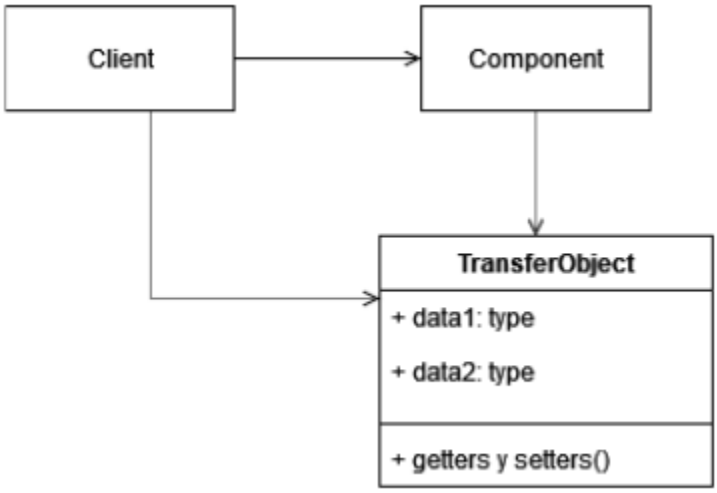

Ilustración 29: Patrón Transfer

4.1.6.3 *Singleton*

El patrón *singleton* garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella. Es muy útil ya que permite un acceso controlado a la única instancia. En nuestro proyecto los servicios de *Firebase* utilizan un *singleton* y a parte nosotros hemos implementado uno propio para el uso de la librería *Volley*. Este, nos

ahorra tiempo en las peticiones al solo crear una cola de peticiones a la que se van añadiendo las peticiones hechas en las distintas clases de la aplicación, en vez de generar una cola por cada petición que hiciésemos en cada una de las clases.

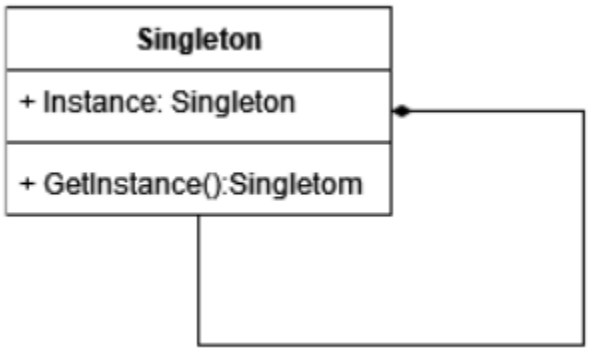

Ilustración 30: Patrón Singleton 1

```
public static synchronized  VolleySingleton getInstance(Context context) {
    if(instance==null){
        instance=new VolleySingleton(context);
    }
    return instance;

}
```

Ilustración 31: Patrón Singleton 2

```
2 usages  Seryiii
public RequestQueue getRequestQueue() {
    if(requestQueue==null){
        requestQueue= Volley.newRequestQueue(context.getApplicationContext(),new HurlStack());
    }
    return requestQueue;
}
3 usages  Seryiii
public <T> void addToRequestQueue(Request<T> request) { getRequestQueue().add(request); }
}
```

Ilustración 32: Patrón Singleton 3

4.1.7 Dificultades encontradas

4.1.7.1 Aprendizaje desarrollo móvil

Inicialmente ninguno de los cuatro miembros tenía conocimientos sobre como programar aplicaciones móviles, por lo que tuvimos que formarnos y aprender cómo se estructuraba un proyecto en *Android Studio* y cómo se programaba en este IDE. Para formarnos vimos varios tutoriales de YouTube y realizamos pruebas haciendo un proyecto sencillo para familiarizarnos con el entorno.

4.1.7.2 Técnicas de *web scraping*

Para poder realizar esta técnica y aprender un poco más de ella, nos tuvimos que informar por un lado a través de tutoriales de YouTube donde se explicaban las nociones básicas de cómo poder moverse a través de una página web gracias al código HTML. Nos explicaban cómo poder escoger los distintos elementos, seleccionándolos en función de su id, nombre de la clase o el XPATH que es un lenguaje que nos permite identificar los elementos de una manera mucho más sencilla ya que muchos de estos a veces no tienen id único con el que poder seleccionarlo o tienen la misma clase que otros elementos por lo que si queremos seleccionar un elemento en concreto y varios elementos tienen la misma clase, no podremos acceder a él de una manera rápida y efectiva.

4.1.7.3 *Socket* vs API

La implementación de estas dos herramientas surge por la necesidad de permitir la comunicación entre la aplicación móvil desarrollada en Java en el entorno de *Android Studio*, y los *scripts* de *web scraping* escritos en Python.

Por un lado, nos planteamos la solución por medio de un *Socket*, que es una abstracción de bajo nivel que nos permite la comunicación entre procesos a través de una red. Es una forma de establecer una conexión punto a punto entre dos sistemas y permitir el intercambio de datos de manera bidireccional, funcionalidades que nos interesaban debido a las necesidades de nuestra aplicación, ya que requerimos el paso de información desde un formulario en nuestra aplicación de Android a un script de Python y viceversa. Los *sockets* se utilizan principalmente para la comunicación entre aplicaciones que se ejecutan en diferentes plataformas o sistemas, y son muy útiles para el desarrollo de aplicaciones de red, como servidores web o sistemas de mensajería instantánea. La principal ventaja de los *sockets* y por lo que fue la primera idea para implementar la funcionalidad comentada en el párrafo anterior, es que los *sockets* ofrecen un control muy preciso sobre la conexión y los datos que se intercambian, lo que los hace muy flexibles y adaptables a diferentes situaciones.

Por otro lado, al ver la complejidad de implementar un *socket*, decidimos empezar a implementar una API para llevar a cabo la comunicación entre los dos sistemas que queríamos conectar. Las API ofrecen una forma más estructurada y controlada para intercambiar información entre aplicaciones y se utilizan principalmente para acceder a servicios web o bases de datos. Una API define un conjunto de operaciones que una aplicación puede realizar en otra, y proporciona una serie de protocolos y formatos de datos estandarizados. Las ventajas que nos ofrece implementar una API son el acceso a datos de otras aplicaciones o servicios de una forma más fácil y estandarizada, lo que las hace muy útiles para la integración de sistemas y la creación de aplicaciones más complejas.

Por lo tanto, en lo que respecta a las diferencias entre los sockets y las APIs, la más destacable es que los *sockets* son una herramienta de bajo nivel que proporciona un control más preciso y flexible sobre la comunicación de datos, mientras que las APIs ofrecen una interfaz más estructurada y fácil de usar, pero a costa de proporcionar una menor flexibilidad, pero en nuestro caso la requerida para implementar las funcionalidades necesarias por nuestra aplicación.

Al final, decidimos que la solución más viable a implementar era el uso de la API, debido a su forma más estandarizada y estructurada de intercambiar datos, aparte de tener una mayor facilidad de ser implementada esta opción.

4.1.7.4 Tiempo de espera

Aun así, durante esta solución, fuimos encontrando problemas en su desarrollo, como fueron la creación de dos hilos mientras se realizaba *web scraping*, ejecutándose así cada petición a la API dos veces. Encontramos el problema, que era producido por la librería *Volley*, al crear un hilo secundario, que también ejecuta la interfaz, además del hilo principal de la aplicación. Otro de los problemas encontrados durante la realización de la API, fue el tiempo que esperaba la aplicación por medio del hilo principal antes de intentar realizar las siguientes operaciones, este tiempo viene dado por defecto por *Volley*, y es de 2,5 segundos; problema que solventamos con la especificación y modificación de la política de manejo de errores y de reintentos de *Volley*, añadiendo el tiempo necesario para que se llevase a cabo la técnica de *web scraping*. El primer problema lo resolvimos especificando los reintentos al valor 0, y el segundo problema, especificando la cantidad de segundos a esperar en 60.000.

```
request.setRetryPolicy(new DefaultRetryPolicy(
        initialTimeoutMs: 60000, // segundos
        0, // 1 reintentos
        DefaultRetryPolicy.DEFAULT_BACKOFF_MULT));
```

Ilustración 33: RetryPolicy

4.1.7.5 Cálculos matemáticos

Para realizar la parte de seguimiento de hipotecas, la cual permite a un usuario llevar un seguimiento real y automatizado en el tiempo de una o varias hipotecas, primero hay que entender las partes claves de toda hipoteca y que el usuario debe conocer:

- **Plazo (normalmente en años)**: duración de la hipoteca.
- **Precio de la vivienda**
- **Cantidad aportada**
- **Tipo**: puede ser fijo, variable o mixto. Esto determina el valor de los intereses a lo largo del tiempo.
- **Intereses**: la parte proporcional de cada pago que se atribuye al banco con el que se ha contratado el crédito. Varía dependiendo del tipo, si es variable o mixta, en la parte variable, es un diferencial más el Euríbor correspondiente al mes en el que se calcula.

A partir de estos datos se puede calcular toda la información útil para los usuarios, relacionada con las hipotecas, usando diversas fórmulas.

Algunas de estas fórmulas son:

- **Cuota mensual**: $C = \frac{P * i}{1-(1+i)^{-n}}$ donde:
  - C: es la cuota mensual.
  - P: es la cantidad prestada.
  - i: es la tasa de interés mensual, expresada en términos decimales.
  - n: es el número total de pagos.

En caso de que el interés sea negativo, se establece el interés aplicado a 0 y la fórmula sería $\frac{Capital\ Pendiente}{Nº\ de\ cuotas\ restantes}$

- **Cuota mensual de un pago específico**: devuelve la cuota correspondiente a ese número de cuota, manteniendo la misma cuota en caso de reducción de plazo y sin que en el siguiente pago haya una revisión. En caso de revisión, se actualiza la nueva cuota con los parámetros correctos (porcentaje aplicado, capital pendiente y cuotas restantes).
- **Capital amortizado mensual**: $CapAmortMensual = cuotaMensual - (capPdte * i)$
- **Obtener intereses mensuales**: $CapPdte * i = CapPdte * \frac{\%}{12}$
- **Capital pendiente actual**: consiste en que, desde un capital inicial total, ir restando el capital amortizado de cada una de las cuotas hasta un número de pago específico. Para obtener este valor hay que tener en cuenta el tipo de hipoteca, ya que si es variable o mixta tiene una primera parte "fija", pero después, en la parte variable, los intereses dependen del Euríbor de cada mes donde se haga la revisión (anual o cada seis meses). En el caso de las variables y las mixtas, se realiza una estimación del capital restante en función del Euríbor actual ya que no podemos saber el Euríbor de las cuotas futuras.
- **Dinero restante**: consta del capital pendiente junto con los intereses en un punto determinado de la hipoteca. Para realizar los cálculos, tenemos en cuenta para todos los tipos de hipoteca que la cuota no varía ya que, este cálculo en el caso de las hipotecas fijas es real ya que el porcentaje no cambia, pero en las variables y en las mixtas, al no tener el Euríbor futuro, hacemos una estimación en función del Euríbor actual junto con el diferencial, indicándoselo al usuario en todo momento. En caso de encontrarse en la parte "fija", tendríamos en cuenta las cuotas fijas restantes y simularíamos el resto en función del Euríbor actual.
- **Plazo actual**: devuelve el plazo actual teniendo en cuenta las amortizaciones anticipadas de reducción de plazo y si la hubiese de reducción total. También tenemos la opción de calcular el plazo en un número determinado de cuota.

A la hora de ajustar la hipoteca a la fecha actual, hemos tenido que realizar varias operaciones de cálculo de fechas, como, por ejemplo, obtener los años y meses restantes, coger el número de cuotas pagadas hasta el día de hoy u obtener el número de cuota en enero, usado para rellenar correctamente el cuadro de amortización mensual. Si la hipoteca no comienza en el mes de enero o no finaliza en diciembre, es necesario realizar ajustes en el número de filas del cuadro de amortización mensual correspondiente a ese año en particular. Además, se debe calcular el número de años que abarca la hipoteca teniendo en cuenta estas variaciones en las fechas de inicio y finalización. Por ejemplo, si el plazo de la hipoteca es de 25 años, pero el inicio o finalización no se encuentran en los meses mencionados, el período real de la hipoteca puede ser de 26 años.

Otra funcionalidad extra que implementamos, es dar la posibilidad al usuario de llevar un seguimiento de las vinculaciones asociadas con su hipoteca dinámicamente. Para ello, cuando se introduce una nueva hipoteca, el usuario tiene la opción de rellenar un campo de “vinculaciones anuales” que corresponderá con el dinero invertido en este tipo de gastos cada año desde el inicio de la hipoteca hasta la actualidad. Esta información se almacena en un array en la base de datos en la que cada posición es el año de hipoteca y el valor, equivale a las vinculaciones asociadas a ese año. También el usuario podrá editar los gastos de cada año en el apartado de “editar hipoteca”.

Además, el día siguiente al que la hipoteca cumple años, se mostrará un diálogo en el que el usuario podrá ingresar los gastos invertidos en vinculaciones ese año.

También damos la posibilidad al usuario de amortizar anticipadamente una cantidad de dinero una vez al mes con la finalidad de reducir su cuota o su plazo, teniendo efecto este pago en la próxima cuota, pero pudiendo previsualizar el cambio que supone. Esto añade un cálculo extra en cada función de la lógica, comprobando si se han producido amortizaciones anticipadas y, de haberse producido, ajustando el capital pendiente. En el caso de que se produzca una de plazo también se reducirá la duración de la hipoteca.

Por otro lado, si la amortización es total, simplemente el capital pendiente pasaría a ser cero, y se actualizará el plazo final al número de cuotas pagadas + 1.

Para implementar la lógica de las amortizaciones anticipadas parciales distinguimos entre los dos tipos:

- **Reducción de plazo**: a la hora de reducir el plazo, comparamos el capital introducido por el usuario a amortizar con el capital de las próximas cuotas. De esta manera obtenemos el número de meses a reducir con esa cantidad de dinero. En caso de que sea la última cuota, el usuario no podrá amortizar. El dinero que "sobra" de la amortización se resta a la última cuota para ajustar con el capital pendiente.
    - Fija: se mantiene la cuota fija y el plazo se actualiza restando el número de cuotas amortizadas anticipadamente.
    - Variable o mixta: si es en la parte fija o el primer periodo de las variables, si lo tiene, actúa como en las hipotecas fijas, por el contrario, si es en la variable, los cambios se ven reflejados en cuanto a la reducción del capital pendiente, pero la cuota sigue igual hasta que se realiza la siguiente revisión, donde se actualiza la cuota con el nuevo plazo y el Euríbor mensual junto con el diferencial.
- **Reducción de cuota**: el capital introducido por el usuario es restado al capital pendiente lo que provoca que la cuota disminuya.
    - Variable o mixta: si se va a amortizar dentro de la parte fija, se disminuye el capital pendiente, por lo que la cuota se reduce. Si, por otra parte, se encuentra en la parte variable, se actualiza la cuota con el plazo actual de la hipoteca, es decir, si se han reducido cuotas en ese mismo periodo antes de la siguiente revisión, se usaría ese nuevo plazo.

Para comprobar la veracidad de estos cálculos nos hemos apoyado en un simulador online del Banco de España proporcionado por el tutor del proyecto, incluido en la bibliografía.

4.1.7.6 Servidor

Los problemas encontrados durante la implementación del servidor empezaron, tras la descarga de los diferentes programas y las importaciones de los ficheros que poseían las funcionalidades de *web scraping* y la propia configuración de la API. Los problemas relacionados con estos archivos, fueron la configuración de permisos sobre las distintas carpetas y archivos, por lo que, viendo los usuarios y grupos alojados en el sistema, observamos que teníamos un grupo formado por todos los integrantes del grupo, por lo que configuramos recursivamente todo lo alojado en la carpeta TFG compartida que teníamos.

Los siguientes problemas llegaron de la mano de las primeras ejecuciones de navegador, donde vimos que el mismo no tenía interfaz gráfica, por lo que tuvimos que modificar las opciones de las funciones que realizaban *web scraping*, poniendo el navegador en modo sin interfaz, pudiendo así el script empezar a interactuar con los elementos HTML.

El siguiente error que apareció durante la ejecución de la API, era el navegador al que intentaba acceder cuando se hacía una petición, el servidor venía con *Chromium* instalado y este era el navegador al que intentaba acceder. Al ser el navegador instalado en la ruta protegida donde están almacenados los ejecutables en el servidor, a la que nosotros no podíamos acceder. Sin embargo, nosotros hacíamos las peticiones con *Google Chrome*, navegador que utilizamos para el correcto uso de la aplicación *Chromedriver*.

Para solucionar este problema, encontramos la solución de especificar la ruta de instalación del ejecutable de *Google Chrome* en las opciones de *Chromedriver*.

El error explicado anteriormente fue muy similar al problema encontrado con la ejecución del *Chromedriver*, y fue solucionado de la misma forma.

A partir de este momento las solicitudes empezaron a funcionar, pero los scripts implementados y mandados en local, seguían dando errores, por lo que tuvimos que empezar a ver otra vez, con bibliotecas como PDB, que nos permitieron ir depurando el código y hacer *troubleshooting* por las líneas de código, donde vimos que la estructura HTML variaba en los cambios de ventana con y sin interfaz gráfica, superponiéndose botones y paneles a otros botones donde queríamos hacer clic; por lo que tuvimos que realizar varios cambios tanto en los scripts de obtención del Euríbor, como el de extracción de las comparaciones de hipotecas.

4.2 *Front-end*

En la parte del *front-end* se empieza detallando cómo se estructuró el diseño y cuál fue la idea de aplicación escogida comparando entre varios formatos utilizados por diferentes aplicaciones. También se explica el lenguaje utilizado y las dificultades que nos surgieron durante el diseño de las distintas actividades y fragmentos.

4.2.1 Estructura *front-end*

Para realizar la estructura del *front-end* de MiHipoteca App nos hemos inspirado en las aplicaciones más punteras del mercado, ya que presentan una gran lógica visual y facilidad a los usuarios. La aplicación usa la vista de *mainActivity.java* como *landing page* o página de destino, ofreciendo al usuario la posibilidad de crearse un perfil, para intentar convertir a los usuarios en clientes potenciales y que así disfruten de las posibilidades que eso brinda dentro de la aplicación.

Podemos distinguir dos tipos de estructuras en las vistas de la aplicación, primero encontramos actividades simples donde solo hay un foco de atención, la propia vista. Esto sucede en páginas como *registrarse.java, iniciarSesion.java, mainActivity.java, etc.*

Por otra parte, están las actividades que se dividen en dos partes, un menú flotante en la parte inferior con el que puedes navegar por las diferentes actividades principales de la aplicación en función del icono en el que pulses.

Ilustración 34: Menú inferior

De izquierda a derecha, los símbolos indican:

- **Mis hipotecas**: página principal donde ver las hipotecas de seguimiento asociadas a tu perfil.
- **Nuevo seguimiento de hipoteca**
- **Nueva comparación de hipotecas**
- **Mi perfil**: acceso a las distintas opciones del perfil de usuario.

Este menú es muy útil para el fácil “movimiento” de los clientes por las principales funcionalidades de la aplicación. Además de este elemento, estas vistas tienen un foco principal donde aparece la funcionalidad de las mismas. Algunas de las páginas donde utilizamos esta distribución son *paginaPrincipal.java* y *perfil.java,* entre otras*.*

También la aplicación ofrece servicios, y funcionalidades diferentes en función de si un usuario es premium o no, es decir, algunas de las vistas sufren modificaciones atendiendo a esta condición del usuario. Un ejemplo es la propia vista de *PasarPremium.java*, que, si el usuario no paga la suscripción mensual, muestra sus ventajas, precio y demás información sobre la suscripción junto con un botón para iniciar los trámites de suscripción, mientras que, si es premium, indica que el usuario cuenta

con esta característica y aparece un botón para iniciar los trámites de cancelación de la suscripción.

4.2.2 Lenguaje XML

Para llevar a cabo la parte del diseño de las diferentes vistas de la aplicación *Android Studio,* utiliza el lenguaje de marcado extensible (XML por sus siglas en inglés), que se basa en agrupaciones de diferentes componentes como botones, *textViews* o *editTexts*. Esto facilita mucho el diseño de estas actividades ya que puedes juntar componentes de diferentes formas, como por ejemplo en *LinearLayouts*, *CardViews* o *Scrolls*, para que cumplan la funcionalidad que buscamos. A su vez, cada componente tiene múltiples atributos que puedes ir editando para que se adapten a las necesidades, como el color, las dimensiones, la visibilidad, etc. Además, al ser una herramienta muy extendida, hemos contado con numerosos estudios y nuevos componentes, a parte de los básicos de *Android Studio*, creados por personas con el fin de aumentar las posibilidades a la hora de programar aplicaciones Android. Algunos de estos nuevos componentes son los gráficos utilizados en el seguimiento de las hipotecas, las tablas del cuadro de amortizaciones o las *CardViews*.

4.2.3 Dificultades encontradas:

4.2.3.1 Aprendizaje diseño móvil

Inicialmente el aprendizaje sobre cómo implementar el diseño de nuestra aplicación se basó en la visualización de los diferentes elementos en aplicaciones comunes del mercado, fijándonos en cómo consiguen hacer ver al usuario para qué sirve cada elemento de una vista fácilmente, sin necesidad de tutorial y sin perder la simplicidad y el estilo. Decidimos usar ciertos elementos comunes en aplicaciones de uso cotidiano como el menú inferior de navegación, que permite el fácil movimiento entre las cuatro funcionalidades de la aplicación, o los *RecyclerViews*, que permiten ver listas de elementos del mismo tipo, como ofertas de hipotecas, de una manera muy visual y limpia.

Una vez teniendo esto y aplicando los conocimientos adquiridos en asignaturas como DSI (Diseño de Software Interactivo), AW (Aplicaciones Web) o IW (Ingeniería Web), y usando *Figma* para diseñar en primera instancia las vistas, pudimos empezar a desarrollar cada vista en *Android Studio*, usando *XML*. Al principio era algo tedioso, pero probando y formándonos con videos y tutoriales online, supimos adaptarnos y entender cómo funcionaba.

Otro factor que nos ayudó bastante fue enseñar la aplicación a amigos y familiares para que, desde un punto de vista externo, nos ayudasen a encontrar fallos en el diseño o nos localizasen elementos o símbolos que daban lugar a confusiones, mientras que desde nuestra perspectiva de desarrolladores no veíamos.

Capítulo 5 - Conclusiones y trabajo futuro

5.1 Conclusiones

Tras realizar el trabajo, hemos adquirido una gran cantidad de conocimientos acerca del mundo de las hipotecas y detalles muy técnicos que pueden servirnos de gran ayuda para nuestro futuro a la hora de contratar una. También hemos conseguido englobar dentro de un mismo proyecto la mayoría de herramientas que una empresa profesional utiliza, como una base de datos remota, un servidor con su respectiva configuración, o una API, entre otras.

Además, hemos aprendido mucho sobre cómo trabajar de manera efectiva en equipo, siguiendo una metodología ágil (*Extreme Programming*) con la cual no teníamos experiencia ninguno de los miembros. Estos conocimientos adquiridos nos van a ser de gran ayuda en el ámbito laboral ya que actualmente la mayoría de empresas adoptan este tipo de metodologías para su trabajo diario.

Consideramos que la elaboración del plan de negocio ha sido uno de los aspectos más cruciales en la creación de esta aplicación. A través de él, hemos logrado identificar las funcionalidades en las que debíamos enfocarnos para destacar nuestras fortalezas en un mercado de aplicaciones similares. Además, hemos evaluado la viabilidad de nuestro *startup* mediante el estudio económico financiero, permitiéndonos visualizar cómo podrían ser los primeros años en distintos escenarios.

En cuanto a la profunda investigación realizada para encontrar el funcionamiento de todos los tipos de hipotecas, amortizaciones anticipadas y gastos que conllevan las hipotecas, nos hemos dado cuenta que esa información no está disponible de manera muy clara y nos ha sido complicado el proceso de aprendizaje.

Para comparar los resultados con casos reales, los mayores problemas surgieron con las hipotecas variables y mixtas, debido a la escasez de simuladores o aplicaciones que calculen cuotas y cuadros de amortización de una hipoteca variable o mixta.

La creación de esta aplicación ha sido de gran valor para nuestro desarrollo y aprendizaje en el ámbito laboral. Nos ha brindado la oportunidad de mejorar nuestras habilidades en el desarrollo de aplicaciones, comprender mejor las necesidades de los usuarios, administrar adecuadamente nuestro tiempo y trabajar de manera efectiva en equipo. En resumen, esta experiencia ha sido muy enriquecedora y nos ha preparado para enfrentar futuros retos profesionales con mayor confianza y destreza.

5.2 Trabajo futuro

A continuación, se explican las funcionalidades a desarrollar en un futuro, necesarias para generar ingresos y crecer exponencialmente como empresa.

- **Chat**: funcionalidad en desarrollo, en la que el usuario pueda resolver sus dudas sobre el funcionamiento de la aplicación y que sean respondidas por una inteligencia artificial, o en su defecto, si el usuario prefiere, pueda establecer una conversación con uno de los administradores.
- **Blog**: funcionalidad en desarrollo, que incluye noticias informativas o actuales sobre el funcionamiento de las hipotecas para que los usuarios obtengan conocimientos de manera sencilla.
- **Incluir anuncios**: incorporar en la aplicación publicidad para los usuarios no premium y así conseguir una de las principales fuentes de ingresos.
- **Ofrecer una pasarela de pago**: acoplar a la aplicación un medio de pago para obtener así la fuente principal de ingresos de la aplicación.
- **Compartir ofertas**: funcionalidad por la que se conseguiría captar más clientes compartiendo ofertas interesantes realizando una búsqueda en nuestra aplicación.
- **Inicio con cuenta de Google:** implementar la posibilidad de iniciar sesión a la aplicación sin necesidad de registrar una nueva cuenta, utilizando la de Google.

- **Cambiar el servidor**: migrar el proyecto a un servidor con más recursos para mejorar la eficiencia de la aplicación.
- **Hacer ofertas sin web scraping**: contactar con posibles proveedores de datos de bancos o con los propios bancos para no tener que obtener esta información de otro comparador.
- **Convenios con bancos**: acuerdos con bancos para que obtener mejor posicionamiento de sus ofertas en nuestra aplicación.

Chapter 5 - Conclusions and future work

5.1 Conclusions

After carrying out the work we have acquired a great amount of knowledge about the world of mortgages and very technical details that can be of great help for our future when hiring one. We have also managed to include within the same project most of the tools which a professional company uses, such as a remote database, a server with its respective configuration, or an API among others.

We have also learned a lot about how to work effectively as a team, following an agile methodology (Extreme Programming) with which none of the members had experience. This acquired knowledge will be of great help in the workplace, since most companies currently adopt this type of methodologies for their daily work.

We consider that the preparation of the business plan has been one of the most crucial aspects in the creation of this application. Through it, we have been able to identify the functionalities that we should focus on to highlight our strengths in a market of similar applications. In addition, we have evaluated the viability of our startup through an economic-financial study, allowing us to visualize what the first years could be like in different scenarios.

Regarding the in-depth research carried out to find out how all types of mortgages work, early amortizations and expenses that mortgages entail, we have realized that this information is not available in a very clear way and the learning process has been complicated for us.

To compare the results with real cases, the biggest problems arose with variable and mixed mortgages, due to the scarcity of simulators or applications that calculate installments and amortization tables of a variable or mixed mortgage.

The creation of this application has been of great value for our development and learning in the workplace. It has given us the opportunity to improve our app development skills, better understand user needs, properly manage our time, and work effectively as a team.

In summary, this experience has been very enriching and has prepared us to face future professional challenges with greater confidence and skill.

5.2 Future work

The features to be developed in the future which are necessary to generate income and grow exponentially as a company are explained below.

- **Chat**: feature under development, in which the user can solve their doubts about the operation of the application and have them answered by artificial intelligence, or failing that, if the user prefers, they can establish a conversation with one of the administrators.
- **Blog**: feature under development, including current or informative news about how mortgages work to make it easy for users to gain knowledge.
- **Include ads**: include advertising into the application for non-premium users and thus achieve one of the main sources of income.
- **Offer a payment gateway**: attach a means of payment to the application in order to obtain the main source of income for the application.
- **Sharing offers**: feature by which it would be possible to attract more customers by sharing interesting offers by performing a search in our application.
- **Sign in with Google account**: implement the possibility to sign in to the application without registering a new account, using the Google account.
- **Change server**: Migrate the project to a server with more resources to improve the efficiency of the application.
- **Make offers without web scraping**: contact possible bank data providers or the banks themselves to avoid having to obtain this information from another comparator.
- **Agreements with banks**: agreements with banks to obtain a better positioning of their offers in our application.

Capítulo 6 - Contribuciones Personales

Sergio Alejo García

Al inicio del TFG trabajé conjuntamente con mis tres compañeros para decidir las principales funcionalidades que queríamos implementar en la aplicación. Para ello, elaboramos un plan de negocio para nuestra aplicación en el que estudiamos la viabilidad del proyecto mediante un estudio económico financiero en el que supusimos distintos escenarios, las oportunidades dentro del nicho de mercado en el que nos encontrábamos y la identificación de nuestro público objetivo.

Una vez elaborado el plan de negocio procedimos a investigar sobre cómo gestionan los distintos bancos las ofertas de las hipotecas.

Para ello, cada uno de nosotros simuló una hipoteca en dos o tres bancos distintos, empleando siempre los mismos parámetros; y analizando las comisiones, vinculaciones y gastos que ofrecían dichos bancos.

Posteriormente elaboramos la SRS (Especificación de Requisitos de Software), documento donde especificamos los objetivos a cumplir durante todo el desarrollo de la aplicación, y las condiciones para llevarlos a cabo. Dicho documento nos ha servido como base para implementar las funcionalidades de la aplicación.

A continuación, empezamos con la parte de codificación en *Android Studio*, entorno de desarrollo elegido para implementar la aplicación.

Esto nos permitió aumentar nuestros conocimientos sobre el funcionamiento de *Android Studio*, entendiendo cómo interactúan las distintas partes de este, tanto en el *back-end* como en el *front-end*. Asimismo, durante esta etapa, pudimos comprender el funcionamiento de *Firestore Database* y su integración con Java.

Realizados estos primeros pasos, identificamos claramente las dos funcionalidades principales de la aplicación: seguimiento de las hipotecas y comparación de hipotecas.

Entre los cuatro, decidimos que la mejor forma de abordar ambas funcionalidades era dividiéndonos en parejas: Carlos y Ricardo, por un lado; y Jorge y yo por el otro. A nosotros nos correspondió la parte de comparación de hipotecas, centrada en la parte de obtención de datos.

Para realizar esta comparación de hipotecas, decidimos realizar la técnica de *web scraping*, técnica que era totalmente desconocida por ambos, pero que nos despertaba mucha curiosidad. Tras estar investigando, encontramos la herramienta de *Selenium*, la cual nos pareció que se adecuaba mucho a las necesidades funcionales que requeríamos, por lo que decidimos empezar a realizar *web scraping*, empleando esta herramienta.

Nuevamente, con el objetivo de abarcar un mayor número de bancos, decidimos dividírnoslos entre Jorge y yo, correspondiéndome a mí a sacar información de bancos como Bankinter o BBVA.

Tras realizar los *scripts* previamente mencionados, observamos que esta no era la forma óptima de realizar la comparación de hipotecas. Por tanto, procedimos a desarrollar un comparador común para todos los bancos, en el que salieron opciones de páginas web como Idealista o Rastreator, fue esta última sobre la que se decidió aplicar *web scraping*.

A su vez, apreciamos que se necesitaba desarrollar un *script* para sacar el Euríbor histórico de cada mes y el Euríbor diario; así como implementar un *script* que el uno de cada mes actualizase el Euríbor medio del mes anterior. Ambos *scripts* se hicieron por medio de *web scraping*.

Para obtener el Euríbor histórico, realicé *web scraping* sobre la página de Idealista sacando la información de una tabla que contiene el Euríbor medio por mes desde enero del año 1999. Una vez obtenidos estos datos, pude implementar que, si no existía la colección del euríbor al iniciarse la aplicación, se llamase a una clase que cargase el Euríbor histórico.

Para lograr la conexión entre la aplicación y los distintos *scripts* que realizaban *web scraping*, propusimos dos implementaciones distintas.

Por un lado, elaboramos un *socket* que se conectaba entre las dos partes. Mientras que, por el otro, desarrollamos un API en *Flask*, a la cual se mandaban peticiones desde la aplicación de Android mediante el uso de la librería *Volley*, que permite hacer peticiones HTTP.

Para optimizar el desarrollo de dicha conexión, yo fui el encargado de implementar el *socket*, mientras que Jorge se encargó de implementar la API.

Tras observar las facilidades que aportaba *Flask*, así como las continuas complicaciones que existían con el *socket*, decidimos optar por la API como método final, terminando su implementación entre los dos.

Durante este proceso nos encontramos diversos fallos al hacer la solicitud a la API, problemas que solucionamos entre Jorge y yo tras una profunda investigación sobre el funcionamiento de las peticiones *Volley*.

Tras finalizar todo lo anteriormente comentado, decidimos solicitar un servidor en el que pudiésemos hacer las peticiones desde un punto centralizado. Para ello, nuestro tutor nos puso en contacto con una de las personas responsables del Departamento de Sistemas Informáticos y Computación de la Universidad Complutense de Madrid para que nos proporcionara el servidor.

Una vez lo obtuvimos, me encargué de conseguir su correcto funcionamiento aplicando habilidades aprendidas en asignaturas de la carrera cómo ASR (Ampliación de Sistemas y Redes), así como conocimientos aprendidos durante mis prácticas en empresas, en las que pude trabajar como Ingeniero de Sistemas.

Conseguí que se pudiesen hacer peticiones con normalidad, instalando programas como Google Chrome o el *Chromedriver* (controlador de Google Chrome). También me encargué de instalar las distintas librerías de Python necesarias para el correcto funcionamiento de la API; así como las licencias y los permisos necesarios para que cada uno de nosotros pudiera acceder a todos los recursos compartidos.

Posteriormente, y con la ayuda de Jorge, me encargué de modificar los scripts de *web scraping* para lograr que funcionasen en un servidor sin interfaz gráfica, ya que

cambiaban algunos de los elementos HTML accedidos mediante *Selenium* y la forma de actuar de las páginas web.

Durante el tiempo que tardaron en proporcionarnos el servidor y posteriormente a que el mismo estuviese funcional, estuve implementando un chat que pudiese usar el usuario para darnos opiniones e intentar aclarar sus dudas. Este chat contesta gracias al uso de inteligencia artificial y en caso de que el usuario requiera de información más específica, le permite contactar con nosotros.

Por último, junto con mis compañeros, terminé de redactar diferentes apartados de la memoria como la configuración del servidor, el uso de *Android Studio*, el funcionamiento de *ApScheduler* y los apartados dirigidos a la parte del servidor entre otros.

Ricardo Carazo Pérez

Mi aportación en este TFG se ha centrado en el plan de negocio, el *front-end*, el *back-end* y la memoria. Las primeras semanas de trabajo se ocuparon en identificar, junto con mis compañeros, el tipo de aplicación que íbamos a desarrollar y las funcionalidades básicas que contendría. El primer paso que decidimos abordar fue el plan de negocio realizándolo todos de manera conjunta.

Lo siguiente fue informarnos sobre el funcionamiento de todos los tipos de hipotecas y cómo tenían la información guardada en cada banco. Para ello, nos dividimos para buscar información de hipotecas tanto fijas como variables y mixtas, y entender así qué vinculaciones ofrecían y cómo se reducían los porcentajes en función de las vinculaciones escogidas. Yo me encargué de buscar información sobre ING y EVO Banco.

Una vez tuve claro cómo funcionan las hipotecas y lo que ofrecían los distintos bancos, pusimos en común nuestras búsquedas y decidimos redactar entre todos los requisitos y funcionalidades en lenguaje natural de manera más detallada.

Después de tener los requisitos, pasamos a decidir qué base de datos utilizar. En mi caso, tenía una plantilla de una aplicación con una base de datos con XAMPP junto con un *web service* en PHP y propuse a mis compañeros utilizar esta opción ya que iba a ser rápida teniendo un ejemplo como guía. Me encargué personalmente de incluir las clases necesarias y una vez lo pude probar, nos dimos cuenta de que el *web service* creado para hacer de intermediario entre *Android Studio* y la base de datos utilizaba HTTP y decidimos cambiar esta opción por Firebase ya que nos ofrecía una comunicación segura y era fácil de conectarla a la aplicación.

El siguiente paso fue crear el proyecto de la aplicación, vincularlo con Github y conectarnos a la plataforma de Firebase junto con los servicios requeridos: *Firebase Firestore* y *Firebase Authentication*. De forma paralela, Carlos fue diseñando unos *mockups* en Figma y cuando nos comentó que los tenía acabados, realizamos todos los

miembros de forma conjunta las vistas más generales de la aplicación y el *back-end* de gestión de usuarios.

Cuando todos nos familiarizamos con el entorno, nos dimos cuenta de que había dos funcionalidades grandes (seguimiento de hipotecas y comparación de ofertas), que llevarían bastante trabajo desarrollarlas, por lo que decidimos dividirlas. Carlos y yo desarrollamos la parte de seguimiento.

Primeramente, tuvimos que investigar los cálculos y fórmulas que se necesitan para llevar a cabo el seguimiento de los tres tipos de hipotecas. Para entenderlo y tenerlo de forma más visual, realicé en papel un caso de una hipoteca fija de ING con datos reales para comprobarlo realizando todas las cuentas.

Teniendo claro el funcionamiento, me puse junto con Carlos a estructurar el código de manera que acabamos definiendo cuatro clases con la mayoría de la funcionalidad de las hipotecas. En primer lugar, una clase general "*HipotecaSeguimiento.java*" que contenía las operaciones comunes a todas las hipotecas como: calcular la cuota mensual, operaciones de ajuste de fechas para mostrar información de tiempo restante, número de cuotas pagadas, etc.

Por otro lado, tres clases que heredan de ésta, y que son: *HipotecaSegFija.java*, *HipotecaSegVariable.java* e *HipotecaSegMixta.java*, las cuales realizan las operaciones en función del tipo de hipoteca. En este apartado nos surgieron varias dificultades. En primer lugar, decidimos llevar el seguimiento de una hipoteca acorde con el calendario actual, lo que supuso tener que realizar bastantes cuentas complejas con fechas. Otro de los problemas que nos surgieron, fue plantear el seguimiento sin la opción de amortizar antes ya que decidimos hacerla posteriormente y tuvimos que reajustar todo el código.

Para adaptar el código que ya teníamos desarrollado, nos informamos sobre cómo funcionaban las amortizaciones anticipadas y debido a la complejidad junto con la falta de información disponible, preguntamos a nuestro tutor y nos proporcionó un simulador

del Banco de España donde pudimos probar varios casos y comprobarlo con una fuente fiable.

Por último, mi contribución en la memoria se ha basado en rellenar apartados como: base de datos, Github, Java vs Kotlin entre otros. Además, fuimos completando la memoria entre todos los miembros realizando varias partes de forma conjunta; por ejemplo, los cálculos matemáticos los describimos entre Carlos y yo ya que fuimos los que desarrollamos esa parte.

Jorge Morales López

Al proyecto MiHipoteca App realizado con mis compañeros, he aportado las siguientes contribuciones. Durante las primeras fases del proyecto, me dediqué a formarme en Android Studio mediante tutoriales y documentación disponible en Internet, familiarizándome con los principales *widgets* de la aplicación y realizando pequeñas pruebas para practicar con la herramienta.

Junto con el resto de los miembros del grupo, nos propusimos investigar las principales características de una hipoteca, sus elementos y las fórmulas necesarias para enfrentarnos al proyecto. Así, distribuimos diferentes bancos para recopilar información acerca de cada uno. Yo me encargué de aportar los datos necesarios sobre los bancos Santander y Openbank.

Además, trabajamos en equipo para elaborar el primer plan de negocio de nuestra empresa, analizando los clientes objetivo y la viabilidad del proyecto, entre otros aspectos importantes.

Después, nos reunimos para definir los principales requisitos de la aplicación y plasmarlos en un borrador, que, aunque no fuese definitivo, sirviera como guía para la implementación de las funcionalidades. Fue entonces cuando decidí trabajar en los casos de uso de las principales funcionalidades de la aplicación, aunque al ser el inicio del proyecto, los casos de uso fueron más bien un pequeño guión para implementarlas en un futuro.

En cuanto a la base de datos, comenzamos con una base de datos relacional mediante XAMPP y PHPMyAdmin, pero al descubrir Firebase, una base de datos no relacional, decidimos migrar el proyecto a esta plataforma ya que se podían realizar consultas a través de HTTPS (conexión segura). De esta manera, me informé sobre cómo conectar nuestra app a Firebase a través de la documentación de s página principal.

Una vez elegida la base de datos y definidas las principales funcionalidades, Carlos realizó los primeros bocetos en Figma y los tomamos como ejemplo para implementar las primeras vistas de la aplicación. Las vistas más generales las hicimos juntos, ya que ninguno dominaba del todo el lenguaje XML, retroalimentándonos cada uno con lo que habíamos aprendido de manera individual. Esta tarea nos llevó unas semanas debido a la falta de familiarización con la herramienta.

Después de estas tareas, dividimos el proyecto en dos funcionalidades principales: realizar el seguimiento de una hipoteca y obtener las ofertas de los bancos de la manera más eficiente posible; siendo esta última, la tarea que implementé junto con Sergio. Me encargué de investigar de dónde podíamos obtener la información de las ofertas ofrecidas por los bancos. En un principio traté de obtener los datos mediante la técnica *web scraping* de las páginas web de cada uno de ellos, pero descubrí que era más eficiente utilizar comparadores como Rastreator. Desarrollé un *script* que recogía los datos rellenados por el usuario en el formulario de comparación y con la herramienta *Selenium,* poder automatizar el proceso de recolección de los datos de Rastreator.

Una vez la obtención de las ofertas de los bancos estuvo lista, me dediqué a investigar cómo conectar la aplicación móvil con el *script* a través de una API creada en Python con el *framework* de Flask, lo que permitió ejecutar el *script* realizando una petición POST.

Después desarrollé tanto el *front-end* como el *back-end* de la vista, que permitiría a los usuarios visualizar las ofertas de los bancos y presentar los resultados de manera ordenada y legible.

Junto con Sergio, solucioné diversos problemas que tuvimos cuando lanzábamos las peticiones a la API desde el emulador del móvil: añadiendo, por un lado, una serie de funciones para que solo se ejecutase un hilo al realizar la petición y, por otro, aumentar el tiempo de espera para obtener la respuesta.

Ejecutábamos la API de manera local, por lo que esto nos resultó un problema, así que fue necesario solicitar un servidor a la universidad para que esta se mantuviera activa todo el tiempo y poder hacer las peticiones desde cualquier lado.

Sergio se encargó de la configuración del servidor y una vez listo, adaptamos el código del *script* de la obtención de datos; ya que el servidor no cuenta con una interfaz gráfica y cargaba los elementos en la página con algunas diferencias respecto a Windows.

Finalizada también esta tarea, comenzamos a rellenar la memoria. He contribuido realizando: los requisitos de usuario y sistema, la mayoría de los casos de uso, la explicación de algunas secciones acerca del *back-end*, incluyendo Selenium, Python, *web scraping* y el uso de estas en nuestro proyecto. Además, me he encargado de explicar el *widget RecyclerAdapter* entre otros y, junto con Sergio, realizar toda la explicación que conlleva el conectar la aplicación móvil con los *scripts* en Python.

Por último, hemos realizado de manera conjunta todo lo respectivo a las partes comunes de la memoria.

Carlos Sobrados Risco

Mi contribución a este proyecto ha consistido en diferentes etapas, cada una adaptada a la fase en la que se encontraba el propio TFG. En primera instancia, una vez decididas las funcionalidades básicas de la aplicación a desarrollar, junto con mis compañeros, nos encargamos de elaborar un primer plan de negocio para conocer las posibilidades de encajar en el mercado que tenía nuestra aplicación. Para ello, como se puede observar en el "Capítulo 2: plan de negocio", realizamos varias secciones, como la comparación de nuestro proyecto con las aplicaciones relacionadas más punteras del mercado, para hacernos una idea de sus fortalezas y debilidades, o el estudio económico financiero, donde supusimos varios escenarios, optimista, esperado y pesimista, en los cuales obtuvimos una cantidad diferente de ingresos para ver la viabilidad de nuestra empresa a lo largo de los dos primeros años.

Una vez llevada a cabo esta parte, para entender mejor cómo funcionaban las ofertas de hipotecas propuestas por los diferentes bancos, nos dividimos dos o tres bancos para cada uno y así investigar esto, rellenando una serie de pautas anteriormente propuestas entre todos, como comisiones, posibles vinculaciones, gastos de gestión o simular una hipoteca igual para todos los bancos para observar sus diferencias.

Después, decidimos elaborar un documento donde fijamos una serie de requisitos y funcionalidades que debía tener *MiHipoteca App* para así entender mejor el funcionamiento total de cada apartado de la aplicación. Este documento fue muy útil y ha acabado siendo bastante fiel a la realidad actual de la aplicación.

Más adelante, sugerí una serie de bocetos muy esquemáticos y simples a papel al resto de mis compañeros para decidir un poco las principales vistas y después me encargué de realizar los *mockups* de la aplicación de manera más detallada. Estos diseños han sufrido bastantes cambios visuales hasta el día de hoy, aunque en esencia y funcionalidad son parecidos.

Seguidamente, comenzamos con la adaptación al entorno de desarrollo *Android Studio* haciendo las vistas más generales de la aplicación como *IniciarSesión.java*,

Registrarse.java o *MainWindow.java* para ir cogiendo soltura tanto en la parte de *front-end* como en el *back-end*. Una vez habíamos hecho estas primeras actividades, seguimos con otras vistas más generales que no requerían de más conocimientos, ya que con lo que habíamos investigado y aprendido podíamos realizarlas, como *InfoPerfilUsuario.java* o *PasarPremium.java*.

Después de tener esto, nos repartimos por parejas con un sorteo, las dos funcionalidades grandes de la aplicación, como son la comparación de ofertas y el seguimiento de hipotecas, donde a Ricardo y a mí nos tocó la segunda opción.

Para realizar esto, primero tuvimos que aprender cómo funcionaban los diferentes tipos de hipotecas, saber qué impuestos se aplican a cada una de ellas y de que dependían, qué eran las amortizaciones anticipadas y que tipos hay, etc. Seguidamente, aplicamos lo aprendido al código creando una clase principal general, *HipotecaSeguimiento.java*, de la cual heredan tres clases que correspondían con cada tipo de hipoteca, *HipotecaSegFija.java*, *HipotecaSegVariable.java* e *HipotecaSegMixta.java*, cada una con sus atributos correspondientes, y a partir de ahí fuimos creando los métodos necesarios para calcular cada apartado de una hipoteca y llevar así su seguimiento diferenciando nuevamente entre cada tipo de hipoteca.

Debido a que optamos porque el usuario pudiese llevar un seguimiento que fuese cambiando día a día si así lo requería, se nos presentaron una serie de dificultades relacionadas con las fechas y con el Euríbor para las hipotecas variables y mixtas.

También decidimos no implementar desde un inicio las amortizaciones anticipadas, lo que nos hubiese ahorrado muchos problemas, ya que cuando nos pusimos a organizar y programar esta funcionalidad, tuvimos que cambiar la mayoría de los métodos de las cuatro clases mencionadas anteriormente. Para implementar esto generamos un *HashMap* de amortizaciones anticipadas para cada hipoteca de seguimiento donde almacenamos la información de cada una de ellas. Esta funcionalidad dio lugar a varios problemas y cálculos delicados, ya que surgían ciertas dudas que nos costó resolver por

nuestra cuenta y que tuvimos que preguntar al tutor del proyecto y a personas externas familiarizadas con el tema.

También me encargué de realizar la funcionalidad de que el usuario pueda introducir cada año el total de gastos en vinculaciones de ese año para que se vea reflejado en los gráficos del seguimiento.

Por último, mi aportación a la memoria de este proyecto ha sido la realización de algunos apartados, como explicar la estructura del *front-end*, el uso del lenguaje *XML*, los bocetos iniciales o los cálculos matemáticos utilizados junto con Ricardo, entre otros, y la maquetación y revisión del mismo documento junto a los demás compañeros de proyecto.

Bibliografía

1. Ionos. (2022, mayo). *Páginas web, dominios, hosting*. Disponible: https://www.ionos.es/
2. Inboost Marketing. (2023). *Cuánto cuesta la publicidad en redes sociales*. Disponible: https://inboost.marketing/cuanto-cuesta-la-publicidad-en-redes-sociales-precios-publicidad-en-redes-sociales/
3. Google. *Precio Google Ads*. Disponible: https://ads.google.com/intl/es_es/getstarted
4. Google. *Pujas por impresión de anuncios*. Disponible: https://support.google.com/google-ads/answer/2630842?hl=es#:~:text=siempre%20es%201%2C20%20%E2%82%AC,los%20emplazamientos%20de%20ese%20sitio
5. Inmoversion. (2022, noviembre). *Qué tipos de comisión tiene una hipoteca*. Disponible: https://inmoversion.es/tipos-de-comisiones-hipoteca/
6. Arquitasa. (2021, agosto). *Los gastos de la hipoteca y quién debe pagarlos*. Disponible: https://arquitasa.com/arqtualidad/pago-gastos-hipoteca/
7. Yanina Muradas, OpenWebinars. (2021, enero). *Kotlin vs java*. Disponible: https://openwebinars.net/blog/kotlin-vs-java/
8. Miquel Riera, HelpMyCash. (2023, mayo). *Calcula cuánto pagarías por tu hipoteca*. Disponible: https://www.helpmycash.com/hipotecas/#calcula-cuanto-pagarias-por-tu-hipoteca
9. Lucía Gastón Lorente, BBVA. (2023). *ITP, AJD e IVA: los impuestos de las viviendas que debes conocer si vas a comprar casa en España*. Disponible: https://www.bbva.com/es/salud-financiera/itp-ajd-e-iva-los-impuestos-las-viviendas/

10. Android Developers. (2023). *Android para desarrolladores*. Disponible: https://developer.android.com/?hl=es-419
11. Usuario *agarwalkeshav8399*, GeeksForGeeks. (2023, abril). *Bottom Navigation Bar in Android*. Disponible: https://www.geeksforgeeks.org/bottom-navigation-bar-in-android/
12. Firebase. (2023, febrero). *Conecta tu app a Firebase*. Disponible: https://firebase.google.com/docs/database/android/start?hl=es-419
13. Github. *Material Components Android Studio*. Disponible: https://github.com/material-components/material-components-android/blob/master/docs/components/TextField.md
14. AnyCharts. (2023). *AnyChart Android Charts*. Disponible: https://www.anychart.com/technical-integrations/samples/android-charts/
15. Banco de España. *Amortización parcial anticipada de un préstamo (hipotecario o personal).* Disponible: https://app.bde.es/asb_www/es/amortizacion.html#/principalAmortizacion
16. Juan Ribón, Nadia Pérez y Pablo Vega, Finanzas Roams. (2023, abril). *Cuota de la hipoteca: ¿cómo se calcula? Tipos y ejemplos*. Disponible:https://finanzas.roams.es/hipotecas/como-calcular-cuota/
17. ING. *Hipotecas NARANJA. Calcula tu hipoteca 100 % online y descubre tu precio personalizado.* Disponible: https://www.ing.es/landings/hipotecas/simula-hipoteca
18. Asociación Hipotecaria Española. *Simulador hipotecario: Para ayudarle a elegir su préstamo*. Disponible: http://www.ahe.es/bocms/sites/ahenew/simulador-hipotecario/
19. Idealista. (2023). *Simulador de hipotecas*. Disponible: https://www.idealista.com/hipotecas/simulador-hipotecas/

20. Rastreator. (2023). *Comparador de hipotecas*. Disponible: https://hipotecas.rastreator.com/datos-comparativa
21. Estefanía González, Kelisto.es. (2023, abril). *Hipoteca mixta: qué es, cómo funciona y cuáles son las mejores.* Disponible: https://www.kelisto.es/hipotecas/consejos-y-analisis/hipoteca-mixta
22. Songhao Wu, Towards Data Sciencie. (2020, junio). *Web Scraping Basics*. Disponible: https://towardsdatascience.com/web-scraping-basics-82f8b5acd45c
23. Selenium. *Locating Elements*. Disponible: https://selenium-python.readthedocs.io/locating-elements.html
24. SelectorsHub. (2022). *Free Productivity Booster Tools For Testers*. Disponible:https://selectorshub.com/
25. Bailey Maybray, blog hubspot. (2022, agosto). *Color Psychology: How To Use it in Marketing and Branding*. Disponible: https://blog.hubspot.com/the-hustle/psychology-of-color
26. Flask. *Flask Documentation*. Disponible: https://flask.palletsprojects.com/en/2.3.x/
27. Alex Grönholm, APScheduler. *User guide Installing APScheduler*. Disponible: https://apscheduler.readthedocs.io/en/stable/userguide.html
28. Firebase. (2023, abril). *Importa y exporta datos*. Disponible: https://firebase.google.com/docs/firestore/manage-data/export-import?hl=es-419
29. Gloria Laura, BitFactura. (2022, abril). Modo oscuro: ¿qué es y cuáles son sus ventajas? Disponible: https://bitfactura.com/modo-oscuro-que-es-y-cuales-son-sus-ventajas

30. Jaume Vicent, TreceBits. (2020, octubre). *Por qué es bueno utilizar el «modo oscuro» de las apps*. Disponible: https://www.trecebits.com/por-que-es-bueno-utilizar-el-modo-oscuro-de-las-apps/

31. RedHat. (2023). *Esquema de particionamiento recomendado*. Disponible: https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/6/html/installation_guide/s2-diskpartrecommend-x86

32. Python. (2023). The Python Debugger. Disponible: https://docs.python.org/3/library/pdb.html

Anexo

Bocetos iniciales

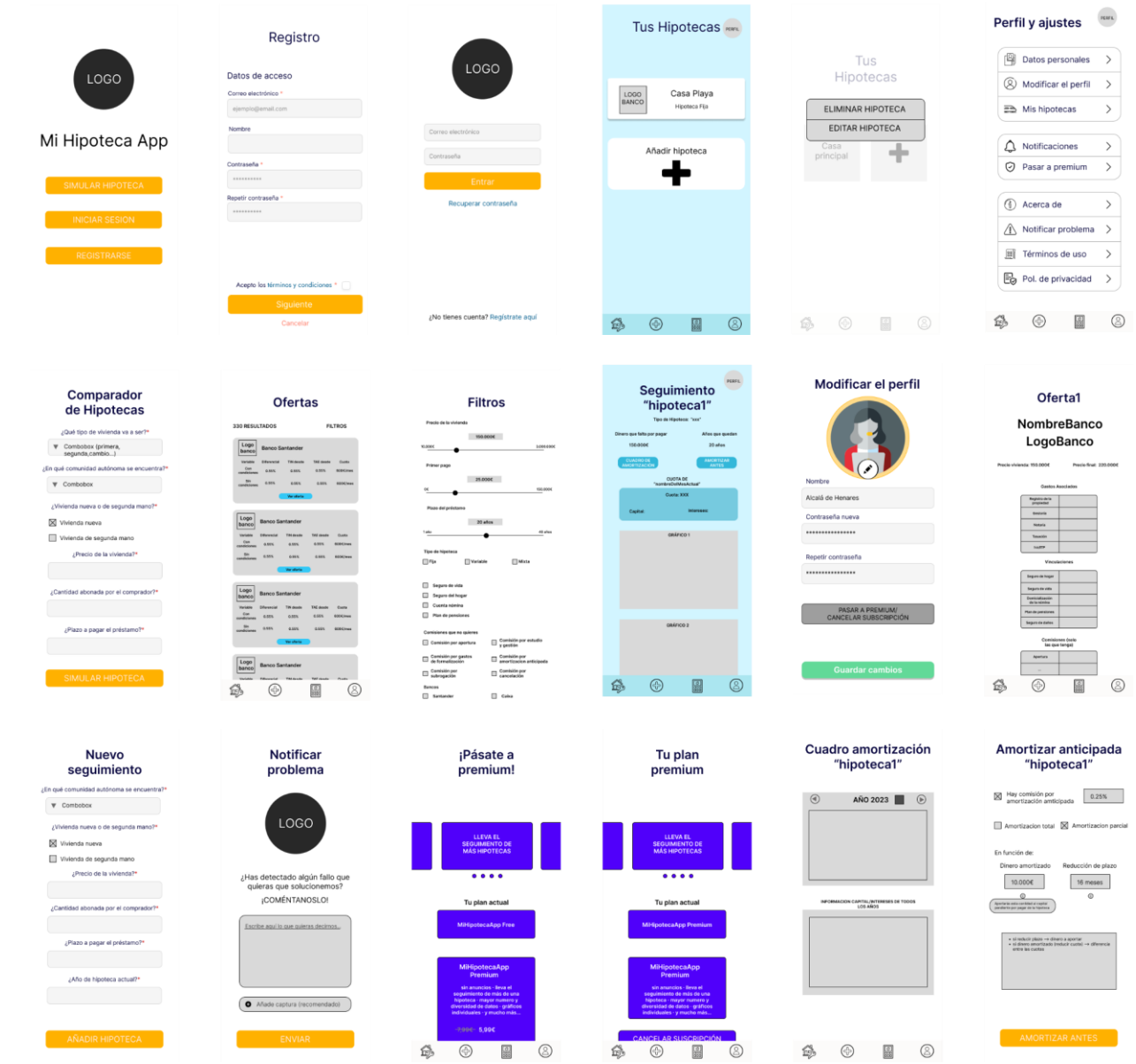

Ilustración 35: Diseños *Figma*

Casos de uso

Inicio de sesión

Descripción	Inicio de sesión por parte del usuario en la aplicación
Necesita	*Firebase Authentication*, aplicación
Actor	Usuario
Entrada	Correo y contraseña
Salida	Redirección a la página principal o mensaje de error
Origen	Interfaz de usuario (Vista → Inicio de sesión)
Destino	Interfaz de usuario (Vista → Página Principal)
Precondición	Correo exista en la *Firebase Authentication* y la contraseña coincida
Postcondición si éxito	Redirección a la página principal del usuario
Postcondición si fallo	Mensaje de error, redirección al formulario inicio de sesión

Tabla 9: Caso de uso inicio de sesión

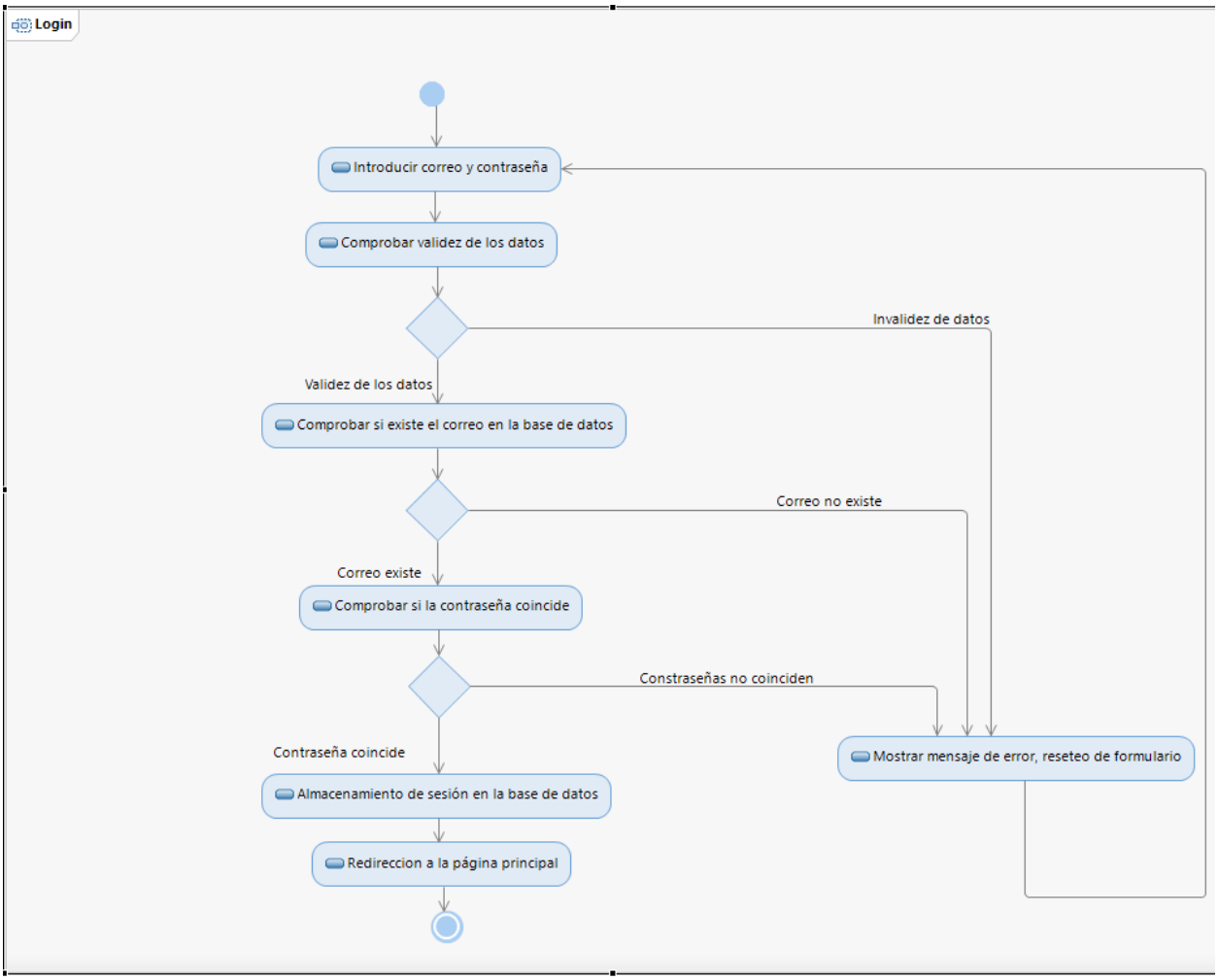

Ilustración 36: Diagrama de actividad inicio de sesión

Cambio de suscripción

Descripción	Cambio de tipo suscripción de usuario
Necesita	Base de datos, aplicación
Actor	Usuario *logueado* en la aplicación
Entrada	Clic en botón de cambio de suscripción
Salida	Cambio de suscripción realizado
Origen	Interfaz de usuario (Vista → Pasar premium)
Destino	Interfaz de usuario (Vista → Modificar datos usuarios)
Precondición	Usuario haya iniciado sesión
Postcondición si éxito	**Paso de normal a premium** → Confirmación de compra y actualización de campo “premium” a True en el documento del usuario de la colección de usuarios **Paso de premium a normal** → Confirmación de actualización y se cambia el campo “premium” a false en el documento del usuario de la colección de usuarios
Postcondición si fallo	Mensaje de error

Tabla 10: Caso de uso cambio de suscripción.

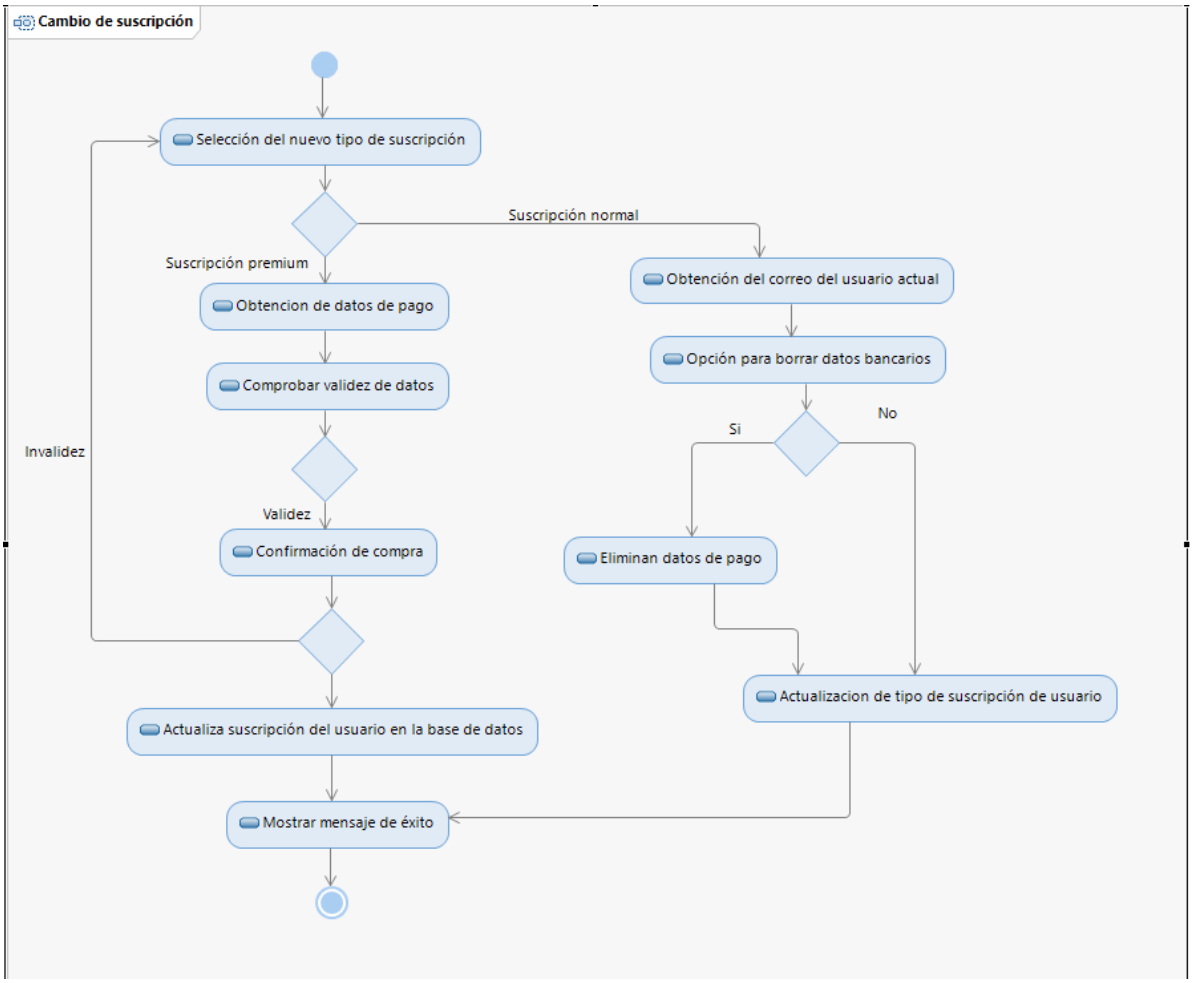

Ilustración 37: Caso de uso Cambio de suscripción.

Eliminar usuario

Descripción	Eliminación usuario
Necesita	Base de datos, aplicación
Actor	Usuario *logueado* en la aplicación
Entrada	Clic en botón eliminar usuario
Salida	Dar de baja el usuario en la base de datos y borrar todos sus datos: hipotecas y ofertas guardadas
Origen	Interfaz de usuario
Destino	Base de datos
Precondición	Usuario *logueado* en la aplicación
Postcondición si éxito	Redirección a página de inicio de sesión de la aplicación y notificación de usuario eliminado con éxito
Postcondición si fallo	Mensaje de error, no se realiza la eliminación del usuario

Tabla 11: caso de uso eliminar usuario

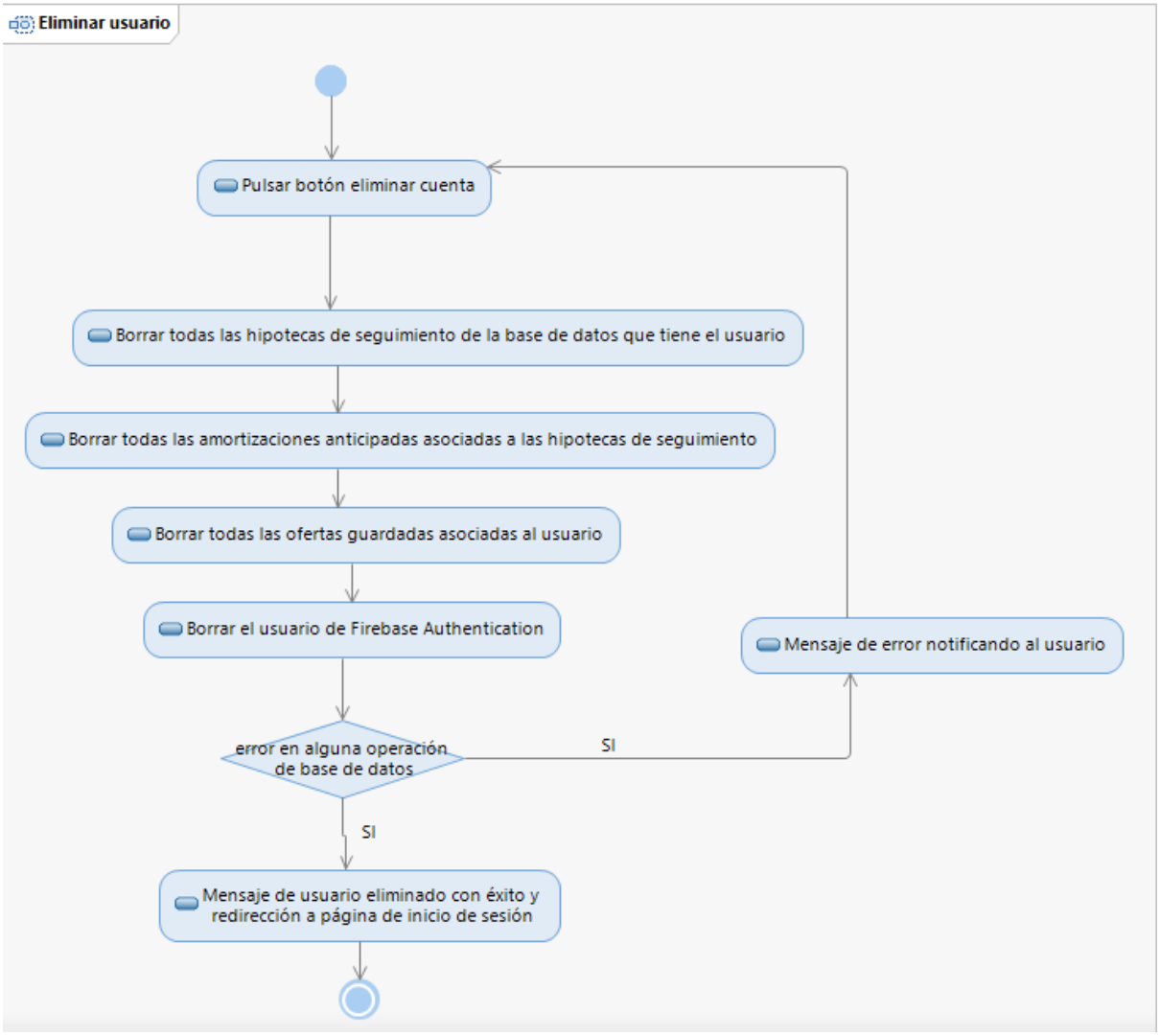

Ilustración 38: Caso de uso Eliminar usuario