ugr | Universidad de Granada

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Aplicación móvil y en la nube para la gestión de un banco de dientes

Autor
Javier Romera Trujillos

Director
Héctor Pomares Cintas

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Departamento de Arquitectura y Tecnología de Computadores

—

Granada, septiembre de 2018

Aplicación móvil y en la nube para la gestión de un banco de dientes

Autor
Javier Romera Trujillos

Director
Héctor Pomares Cintas

Aplicación móvil y en la nube para la gestión de un banco de dientes

Javier Romera Trujillos

Palabras clave: Android, BaaS, Firebase, banco de dientes.

Resumen

El objetivo de este trabajo es diseñar e implementar una aplicación móvil Android que conecte con un servidor en la nube. Sus principales características serán la creación de piezas dentales, por parte de los dentistas en base a las piezas físicas disponibles en sus respectivas clínicas, y su solicitud por los estudiantes de odontología que las requieran.

Este proyecto ha surgido gracias al encargo de un especialista en odontología.
A lo largo del documento se analiza y sintetiza la información extraída en las reuniones y contactos establecidos con él determinando los elementos básicos que poseen las herramientas.

En cuanto a la aplicación solicitada, es necesario que sea de fácil acceso y que permita a los usuarios añadir o editar cada uno de las variables que componen las piezas dentales de la manera más simple posible en función del tipo de usuario que hayan seleccionado.
Para su correcta implementación se especifican y detallan los diagramas seguidos, la estructura implementada y las interfaces diseñadas.

Respecto al servidor utilizado, este requiere poseer principalmente la capacidad de albergar una base datos para las piezas dentales y un almacén de archivos para las imágenes asociadas.
Se listan tanto las alternativas como la solución finalmente seleccionada, Firebase, un BaaS del cual se detallan las distintas herramientas utilizadas, su comunicación con la aplicación móvil y las desventajas encontradas durante el desarrollo.

Finalmente, con el propósito de comprobar el rendimiento del servidor a las peticiones de los usuarios, se analizan los tiempos de respuesta que se experimentan al realiza alguna operación desde el dispositivo móvil y los de las funciones implementadas en el backend del servidor.

Mobile and cloud application for managing a bank of teeth

Javier Romera Trujillos

Keywords: Android, BaaS, Firebase, bank of teeth.

Abstract

The purpose of this work is to design and implement an Android mobile application that connects with a cloud server which allows the creation of dental pieces by the dentists based on the physical pieces availables in their respective clinics and their request by the students of odontology that need them.

This work has been developed thanks to the order of a dental specialists. Throughout the document the information extracted in the meetings and contacts established with him are analyzed and synthesized, determining the basic elements that the tools possess.

Regarding the application requested, it is necessary to be easily accessible and allow users to add or edit each of the teeth variables in the simplest way possible depending on the type of user they have selected.
For its correct implementation, the diagrams followed, the structure implemented and the interfaces designed are specified and detailed.

The server used requires to have the capacity to contain a database for the dental pieces and a file store for the associated images.
The alternatives are listed as well as the finally selected solution, Firebase, a BaaS which details the different tools used, the communication with the mobile application and the disadvantages found during the development.

Finally, in order to check the server's performance to the users' requests, the response times that are experienced when performing some operations from the mobile device and those from the functions implemented in the server's back-end are analyzed.

Yo, **Javier Romera Trujillos**, alumno de la titulación Grado en ingeniería informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76626639T, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Javier Romera Trujillos

Granada a 05 de septiembre de 2018 .

D. **Héctor Pomares Cintas,** Profesor del Departamento Arquitectura y Tecnología de Computadores de la Universidad de Granada

Informa

Que el presente trabajo, titulado **Aplicación móvil y en la nube para la gestión de un banco de dientes** ha sido realizado bajo su supervisión por **Javier Romera Trujillos** , y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 05 de septiembre de 2018 .

El director:

Héctor Pomares Cintas

Agradecimientos

A la familia.

Índice de contenido

Capítulo 1. Introducción

En la actualidad, el uso de bases de datos está extendido a todos los ámbitos de la vida. Permiten recoger y organizar datos que serían casi imposibles de gestionar sin ellos.

Para el mercado sanitario existen numerosas de ellas que almacenan desde informes de los pacientes hasta datos sobre los órganos que se almacenan.

En este proyecto se desarrolla una base de datos donde se almacenan las piezas dentales que cada dentista posea y decida incluir con el objetivo de que los estudiantes de odontología puedan acceder a ellos y solicitarlos.

Esto permite que los estudiantes no tengan que llamar ni desplazarse por las distintas clínicas de la provincia en busca de una determinada pieza dental para sus prácticas. Sólo será necesario solicitarlas a través de la aplicación y acercarse a la clínica indicada para recogerlas.

Los usos que se pueden dar a los dientes recogidos desde las clínicas por los estudiantes incluyen:

- Reimplantes. En el caso de que algún usuario pierda un diente puede ser reimplantado de nuevo. Para ello hay que conocer el propietario de la pieza dental.
- Estudio de las características. Permite que los estudiantes apliquen las técnicas aprendidas sobre diferentes tipos de piezas.
- Uso en técnicas de estética. En el caso de que algunas piezas dentales de un paciente resulten dañadas se pueden recurrir al banco de datos para obtener una equivalente que permita reparar el daño sufrido.

El futuro de este trabajo también podrá estar relacionado con las solicitudes de piezas dentales que estén preservadas mediante criogenización. Esta técnica consiste en almacenar las piezas en nitrógeno líquido a una temperatura que esté por debajo de los 196 grados lo que permite almacenar en perfecto estado las piezas durante largos periodos de tiempo.
Su principal uso está relacionado con la obtención de las células madres de la pulpa con los siguientes propósitos:

- Regeneración de órganos.
- Regeneración de tejidos.
- Reparación de huesos.

1.1. Origen del proyecto

Agradecer a la experta jurista y especialista en odontología Dña. María Isabel Navarro Moreno por el encargo del proyecto que ha dado lugar a este trabajo.

El propósito era poder cumplir con la petición del experto en odontología, que deseaba una aplicación móvil que permitiese almacenar piezas dentales y que los alumnos pudiesen solicitarlos.

Para obtener información detallada sobre las peticiones del especialista se establecieron varias comunicaciones con él:

- El día 26 de octubre de 2017 se realizó una reunión donde se aclararon una serie de puntos con el propósito de obtener la información básica para determinar el tipo de trabajo que se iba a desarrollar.

- El día 6 de mayo de 2018 hubo un intercambio de mensajes mediante una aplicación de mensajería móvil para detallar los parámetros básicos de las piezas dentales.

- El día 30 de mayo de 2018 se volvió a producir un intercambio de mensajes mediante una aplicación de mensajería móvil para aclarar ciertos puntos tratados en la anterior comunicación.

Es conveniente comentar que la obtención y posterior síntesis de la información proporcionada no fue una tarea sencilla.
Se ha observado que algunos expertos en materias ajenas al mundo de la informática pueden no saber transmitir la información de lo que desean de la manera más sencilla para aquellos que sí lo conocen.
Estas complicaciones han sido comunes durante el desarrollo de este trabajo, ya que cada vez que se necesitaba establecer contacto con el experto se producían:

- En la primera reunión hubo que recoger grandes cantidades de información a través de una charla poco detallada y más tarde reordenarla para obtener un esquema simplificado de los datos y el funcionamiento básico de la aplicación que se necesitaba.
- En las siguientes comunicaciones, ante preguntas detalladas, se recibieron numerosos audios de diversa duración donde se comentaban de manera no muy específica las posibles soluciones. Hubo que escuchar los audios numerosas veces hasta obtener la información deseada.

1.2. El proyecto

Este proyecto solicitado tiene como objetivo principal el desarrollar una aplicación a la que se acceda desde un dispositivo móvil y que permita almacenar piezas dentales o solicitarlas en función del tipo de usuario (dentista o estudiante respectivamente).

Para ello habrá que diseñar tanto una aplicación móvil como una base de datos mediante el modelo software cliente-servidor.

La aplicación deberá poseer distintas pantallas para permitir a los usuarios registrase y gestionar aquellas acciones referidas a los datos contenidos en la base de datos.

La base de datos deberá, como mínimo, permitir almacenar datos en ella, editarlos y borrarlos cuando un usuario registrado tenga credenciales para ello.

Tras un estudio inicial, se determinó la necesidad de crear dos grupos de usuarios cada uno con funciones diferentes:

- Los dentistas (pertenecientes a sus respectivas clínicas) que podrán:

    - Añadir piezas dentales.
    - Editar las piezas que hayan creados.
    - Eliminarlas de la base de datos.
- Los estudiantes que podrán:
    - Solicitar las piezas dentales.

Para gestionar esos datos, la base de datos deberá permitir almacenar, editar y borrar los siguientes conjuntos:

- Las piezas dentales.
- Los perfiles de los usuarios de tipo dentista y los datos de su clínica.
- Los perfiles de los usuarios de tipo estudiante.

1.3. Artículos en revistas

Para tener una mayor perspectiva acerca de los bancos dentales se ha realizado una búsqueda de diferentes artículos que tratasen sobre ello.

A continuación, se comentarán los artículos encontrados y las diferencias con el objetivo de este proyecto:

- **Biobanco de dientes humanos para investigación en odontología.** [inf 01]

  Artículo que detalla las especificaciones legales y de sanidad necesarias para la creación de un banco de dientes humanos para el estudio odontológico en la Facultad de Odontología de la Universidad Nacional de Colombia.

  Este artículo se aleja del propósito de este trabajo debido a que su objetivo es la reparación en niños exclusivamente y no se trata la posibilidad de aplicarlo a dientes permanentes. Tampoco se incluye la posibilidad de solicitarlos mediante algún tipo de aplicación ni la necesidad de realizar estudios sobre ellos.

- **Banco de dientes: una alternativa para la rehabilitación de dientes temporales anterosuperiores.** [inf 02]

  Artículo que trata sobre la creación de un banco de dientes temporales donde se puedan extraer piezas para aplicar reparaciones dentales en niños.

  Este artículo se acerca más a la idea de este proyecto, pero no especifica el requisito de crear una aplicación que permita gestionar solicitudes o acceder a sus características de cada una de las piezas dentales del banco.

1.4. Mercado español

En España no existen dichos bancos dentales. Lo más próximo a ellos son los llamados bancos de células madres donde el almacenamiento de piezas dentales se realiza mediante la criopreservación

(conservación en nitrógeno líquido en una temperatura en torno a los -196º) con el propósito de conseguir que las funciones vitales de las células contenidas en la pulpa no se vean deterioradas con el paso del tiempo. [inf 06]

A principios de año existían dos de ellas: Dencells y Criodental Biopharma, sin embargo, esta última cerró a principios de mayo [inf 5].

Dencells [inf 03]

Empresa cuyo propósito es la conservación de las piezas dentales, tanto deciduas como permanentes, de sus clientes con el objetivo futuro de extraer de ellas las células madres contenidas en la pulpa y de ahí poder reparar daños en los tejidos, corazón, cerebro, etc del propietario.

El almacenamiento de las piezas dentales se realizará tras un estudio de la viabilidad de la pieza extraída y se podrá prolongar a lo largo de 25 años.

Los costes van desde los 450 € hasta los 2000 € dependiendo si se decide pagar tras contratar el servicio o tras realizar la prueba de viabilidad de la pieza dental [inf 04].

La diferencias con este proyecto se encuentran en que es un servicio de pago y que carece de utilidad para aquellos estudiantes que deseen obtener piezas dentales para la realización de experimentos o trabajos universitarios ya que la conservación dental para trabajo o investigación externa a la empresa no es su objetivo.

1.5. Planificación y presupuesto

Para la planificación del proyecto se ha dividido su desarrollo en 11 procesos agrupados en 4 fases principales. En la figura 1.1. se muestra el periodo de realización de cada una de las fases y el trabajo realizado y pendiente.

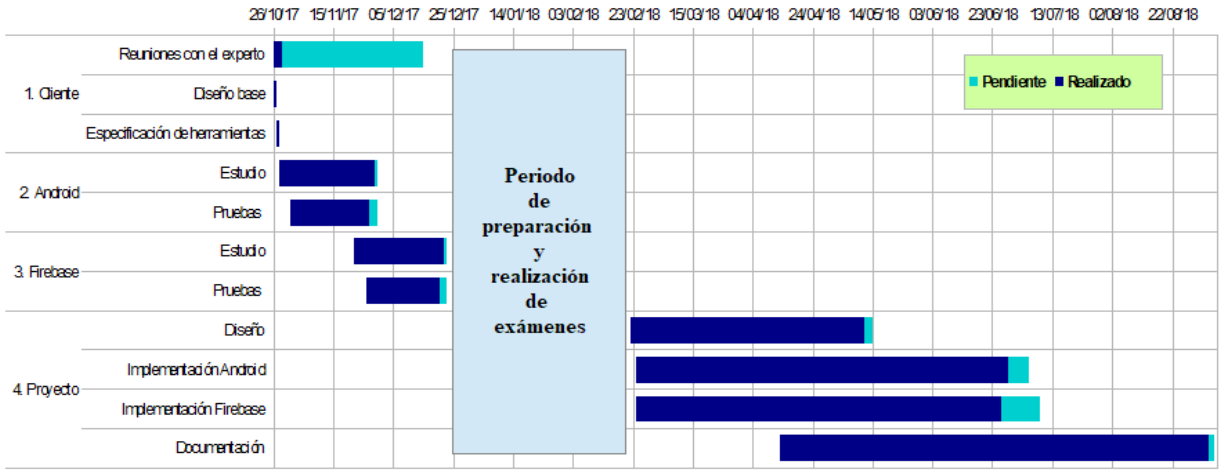


Figura 1.1. Diagrama de Gantt del proyecto

En la tabla 1.2. se establece la comparativa entre las horas previstas y las horas realizadas en cada uno de los procesos:

Fase	Proceso	Horas previstas	Horas realizadas	Porcentaje
Cliente	Reuniones	20 horas	3 horas	15,00%
	Diseño base	7 horas	7 horas	100,00%
	Esp. Herramientas	4 horas	4 horas	100,00%
Android	Estudio	100 horas	98 horas	98,00%
	Pruebas	87 horas	87 horas	100,00%
Firebase	Estudio	100 horas	94 horas	94,00%
	Pruebas	50 horas	48 horas	96,00%
Proyecto	Diseño	81 horas	78 horas	96,29%
	Implementación	100 horas	98 horas	98,00%
	Documentación	80 horas	78 horas	97,50%
Total		629 horas	595 horas	94,59%

Tabla 1.1. Datos de fases realizadas

A estos datos hay que añadir las diversas reuniones con el director del trabajo las cuales están en torno a las 14 horas en 9 reuniones diferentes.

Respecto al presupuesto para desarrollo de este trabajo, se han realizado una serie de cálculos aproximados los cuales determinan que está en torno a los 12.566 €.
En la siguiente tabla se detallan cada uno de los aspectos considerados:

Material necesario	Coste	Total
Ordenador	250 €	250 €
Dispositivo móvil	120 €	120 €
Alquiler del local (9 meses)	350 €/mes	3.150 €
Consumo de agua (9 meses)	28,25 €/mes	254 €
Electricidad (9 meses)	44,7 €/mes	402,3 €
Precio curso de aprendizaje	500 €	500 €
Trabajo autónomo (6 meses)	11,5 €/hora	11.040 €
Total requerido		15.716,3 €

Tabla 1.2. Coste del trabajo realizado.

1.6. El documento realizado y consideraciones previas

Para facilitar la lectura, en los siguientes apartados del documento se mostrarán en negrita aquellas

palabras que tengan relación con el servidor o herramientas externas al trabajo sobre la aplicación del cliente y en cursiva aquellas palabras que hagan referencia a código, métodos o funciones dentro de ella.

También se especificarán los requerimientos, los diagramas seguidos y la estructura desarrollada para cumplir con el proyecto.

Es conveniente indicar que el resultado final se trata de un prototipo. Esto se debe a que durante largos periodos de tiempo ha sido imposible contactar con el experto por lo que gran parte del diseño de la aplicación y de los elementos almacenados se han basado en las ideas obtenidas en la primera reunión.

Tras la última comunicación que se tuvo con él se observó que algunos de los valores que se usaban para el almacenamiento de las piezas dentales no eran los adecuados. Debido a la falta de tiempo sólo se sustituyeron algunos de ellos.

A continuación, se muestran los valores que se deberían desarrollar en una versión final[1]:

1. **Tipo de dentadura**:
   - Temporales.
   - Permanentes (jóvenes).
   - Permanentes (adultos).
2. **Pieza dental** (usando la numeración FDI y en función del tipo de dentadura seleccionada).
3. **Decoloración**:
   - Grado alto.
   - Grado medio.
   - Grado bajo.
4. **Estado**:
   - Sellador de fosas y fisuras.
   - Traumatismos.
   - Daños en corona.
   - Daños en raíces.
   - En perfecto estado.
5. **Dientes supernumerarios** (sólo en caso de que los haya):
   - Dens in dente superior.
   - Dens in dente inferior.
6. **Endoncia** (sólo para dientes permanentes):
   - Sí.
   - No.
7. **Cavidades tipo Black:**
   - Ninguno
   - Tipo I.
   - Tipo II.
   - Tipo III.

1 La versión utilizada para el prototipo se puede consultar en el apartado *Descripción de los actores y de los datos a almacenar* del Capítulo 2.

    - Tipo IV.
    - Tipo V.
8. **Tratamiento pulpar** (en función del tipo de dentadura):
    - Temporal: ninguno, pulpotomía o pulpectomía.
    - Permanente (jóvenes): ninguno, tratamiento pulpar directo, apicogénesis o inducción a la picoformación.
    - Permanente (adultos): Ninguno o endodoncia.
9. **Caries**
    - Mesial.
    - Distal.
    - Oclusal/incisal.
    - Gingival.
    - Vestibular.
    - Palatina.
    - Lingual.
    - Ninguna
10. **Obturación**:
    - No posee.
    - Composite.
    - Amalgama (opción sólo permitida en caso de seleccionar alguna cavidad de tipo Black).
11. **Conservación**:
    - Hipoclorito de sodio
    - Clorhexidina
    - Sin conservación.
12. **Porcentaje de pulpa.**
13. **Porcentaje de corona.**
14. **Porcentaje de dentina.**
15. **Fecha de extracción de la pieza.**
16. **Fecha límite para la destrucción de la pieza.**

Capítulo 2. Análisis de requisitos

En este capítulo se mostrarán los requisitos obtenidos tras realizar una síntesis de los datos obtenidos durante la reunión con el experto y su posterior análisis.

2.1. Objetivos del sistema

OBJ-1. El sistema debe permitir el registro y el acceso a los usuarios que desean darse de alta en la plataforma.

OBJ-2. El sistema debe permitir la alteración de los datos personales de los usuarios registrados.

OBJ-3. El sistema debe ser capaz de permitir que un usuario registrado elimine su perfil.

OBJ-4. El sistema debe ser capaz de almacenar y gestionar la información de las piezas dentales suministradas por el usuario dentista.

OBJ-5. El sistema debe permitir la inclusión, gestión y eliminación de la información de las piezas dentales suministradas por el usuario dentista.

OBJ-6. El sistema debe permitir el acceso a la información de las piezas dentales por parte de los usuarios registrados.

OBJ-7. El sistema debe permitir alterar parte de la información de las piezas dentales por parte de los usuarios estudiantes.

OBJ-8. El sistema debe estar disponible independientemente del acceso a los recursos que hagan otros usuarios.

2.2. Requisitos funcionales

RF-1. Gestión de usuarios de tipo "dentista". Se permitirá dar de alta/baja a cualquier usuario de tipo dentista del sistema, consultar y/o modificar cualquiera de sus datos así como gestionar las piezas dentales que decida añadir al sistema.

> **RF-1.1.** Alta de usuario. Se registrará en el servidor el usuario de tipo "dentista" así como su información correspondiente.
>
> **RF-1.2.** Alterar la información del usuario. Se permitirá la alteración de los datos almacenados asociados al usuario.
>
> **RF-1.3.** Consultar información del usuario de tipo dentista. Se permitirá acceder a la información del propio usuario.

RF-1.4. Baja de usuario. Se eliminará la toda la información asociada al usuario.

RF-1.5. Gestión de piezas dentales. Se permitirá añadir piezas dentales, alterar sus datos y eliminarlas del sistema.

RF- 1.5.1. Incluir pieza dental. Se podrá incluir una pieza dental así como sus datos.

RF-1.5.2. Alterar información de una pieza dental. Se permitirá alterar la información de las piezas dentales añadidas por el usuario.

RF-1.5.3. Eliminación de las piezas dentales. Se podrá eliminar las piezas dentales que hayan sido añadidas al sistema siempre y cuando no hayan sido solicitadas por un usuario del tipo "estudiante".

RF-2. Gestión de usuarios de tipo "estudiante". Se permitirá dar de alta/baja a cualquier usuario de tipo estudiante del sistema, consultar y/o modificar cualquiera de sus datos así como solicitar las piezas dentales que existan en el sistema.

RF-2.1. Alta del usuario. Se registrará en el servidor el usuario de tipo "estudiante" así como su información correspondiente.

RF-2.2. Alterar la información del usuario. Se podrá alterar los datos almacenados asociados al usuario.

RF-2.3. Consultar información del usuario. Se permitirá acceder a la información del propio usuario.

RF-2.4. Baja de usuario. Se eliminará la toda la información asociada al usuario.

RF-2.5. Visualizar información de las piezas dentales. Se podrá obtener los datos asociados a las piezas dentales que hayan en el sistema.

RF-2.6. Búsqueda de piezas dentales. Se realizarán búsquedas personalizadas de piezas dentales incluidas en el sistema.

RF-2.7. Solicitud de piezas dentales. Se podrá solicitar las piezas dentales que aún no tengan un usuario de tipo "estudiante" asignado.

RF-2.8. Visualizar información de los usuarios de tipo "dentista". Se podrá visualizar la información referida a los usuarios de tipo "dentista" cuando de consulten las piezas dentales creadas en el sistema.

RF-3. Gestión de piezas dentales. El sistema gestionará el estado de las piezas dentales incluidas.

RF-3.1. El sistema gestionará el control de las fechas de actualización de las piezas dentales en función de la fecha del servidor.

RF-3.2. El sistema gestionará el control de las imágenes asociadas a los datos de las piezas

dentales indicando si existe dicha imagen o no.

RF-3.3. Se podrá comprobar la información asociada a las piezas dentales.

RF-3.4. Se permitirá alterar la información asociada a las piezas dentales.

RF-3.5. Se permitirá eliminar de forma definitiva toda la información asociada a las piezas dentales.

2.3. Requisitos no Funcionales

Usabilidad

RN-1. Se proporcionará una ayuda en dentro de la pestaña información de la aplicación.

Fiabilidad

RN-2. Para prever caídas del sistema se realizarán copias de seguridad internas.

Rendimiento

RN-3. El sistema deberá realizar búsquedas rápidos a los datos almacenados. La latencia en la recuperación de los datos almacenados no puede superar los 5 segundos en condiciones óptimas.

RN-4. El sistema deberá permitir el envío rápido de los datos obtenidos. La latencia en el envío de datos no puede superar los 5 segundos en condiciones óptimas.

Soporte

RN-5. Los datos mostrados a los usuarios deben ser adaptados para ser compatibles con un sistema Android cuya versión del SDK mínima sea 19.

Restricciones de implementación

RN-6. Se debe usar el lenguaje de programación compatible con el desarrollo de aplicaciones para Android.

RN-7. Se debe usar un SDK compatible con el desarrollo de aplicaciones para Android.

Restricciones de interfaz

RN-8. El sistema permitirá el envío de información de los usuarios de tipo estudiante a los usuario de tipo "dentista"

Requisitos Físicos

RN-9. El sistema necesita un servidor para almacenar la base de datos, las imágenes subidas, así como un sistema de programación backend y un sistema de monitorización.

RN-10. El sistema deberá poder realizar sin problemas el envío y recepción de los datos a través de la red cuando las conexiones superen los 8Mbps.

RN-11. El sistema deberá implicar una carga menor al 50% en dispositivos con una CPU

inferior a 800 Mhz.

RN-12. El sistema deberá funcionar para dispositivos que usen una RAM de 100 MB.

RN-13. El sistema deberá funcionar para dispositivos con un almacenamiento de caché mínimo de 32 KB.

2.4. Requisitos de información

RI-1. Cuenta de usuario - Dentista.
Información del usuario dentista.
Contenido: nombre, apellido, DNI, Fecha de nacimiento, ciudad, población, dirección, teléfono, correo de contacto, clínica, número de colegiado, dientes creados.
Requisitos asociados: RF-1, RF-1.1, RF-1.2, RF-1.3, RF-1.4, RF-2.8, RF-4, RF-4.1, RF-4.2 y RF-4.3.

RI-2. Cuenta de usuario - Estudiante.
Información del usuario estudiante.
Contenido: nombre, apellido, DNI, Fecha de nacimiento, ciudad, población, dirección, teléfono, correo de contacto, facultad, dientes solicitados.
Requisitos asociados: RF-2, RF-2.1, RF-2.2, RF-2.3 y RF-2.4, RF-4 (RF-4.1, RF-4.2 y RF-4.3).

RI-3. Piezas dentales.
Información relativas a las piezas dentales incluidas por los dentistas.
Contenido: tipo de pieza dental, número asignado (Nomenclatura dental universal), tipo de sangre, edad, género, número de raíces, color, tipo de conservación, estado (raíces y corrona), tratamiento pulpar realizado, enfermedades (dentales y propietario), porcentajes (esmalte, dentina y pulpa), imagen, fecha de alteración, identificadores (comprador, creador y diente) y estado (creación, compra y borrado).
Requisitos asociados: RF-1.5, 1.5.1, 1.5.2, 1.5.3, RF-2.5, RF-2.6, RF-2.7, RF-3, RF-3.1, RF-3.2, RF-3.3, RF-3.4 y RF-3.5.

2.5. Diagramas de casos de uso

En este aparatado se detallarán los distintos diagramas de casos de uso diseñados y se dará una breve explicación sobre su funcionamiento.

2.5.1. Subsistema de gestión de usuarios.

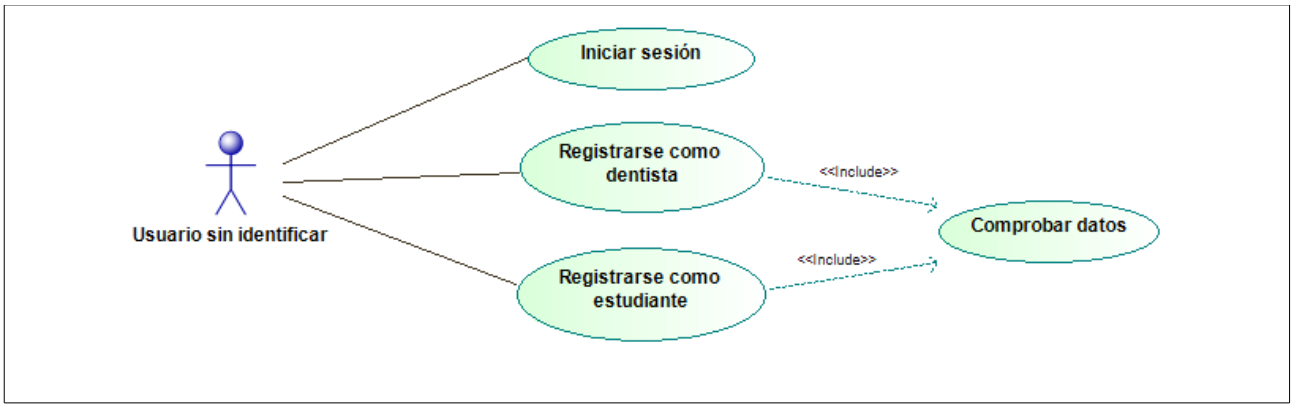


Figura 2.1.

La figura 2.1 muestra el comportamiento inicial que del sistema para los usuarios no registrados.

Se les debe permitir registrarse como estudiantes o como dentistas e iniciar sesión una vez se haya creado la cuenta.

2.5.2. Subsistema de gestión de perfil.

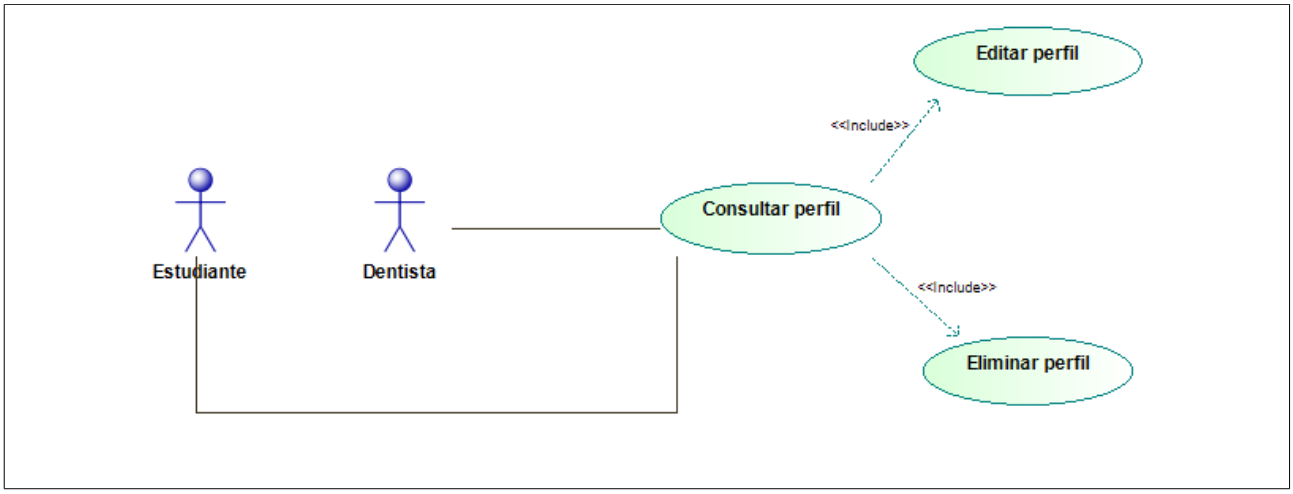


Figura 2.2

En la figura 2.2 se observa como para cada usuario que haya iniciado sesión en el sistema se le debe permitir consultar su perfil y editarlo.
También podrá eliminar dicho perfil lo que eliminará su cuenta del sistema.

2.5.3. Subsistema de gestión del usuario dentista.

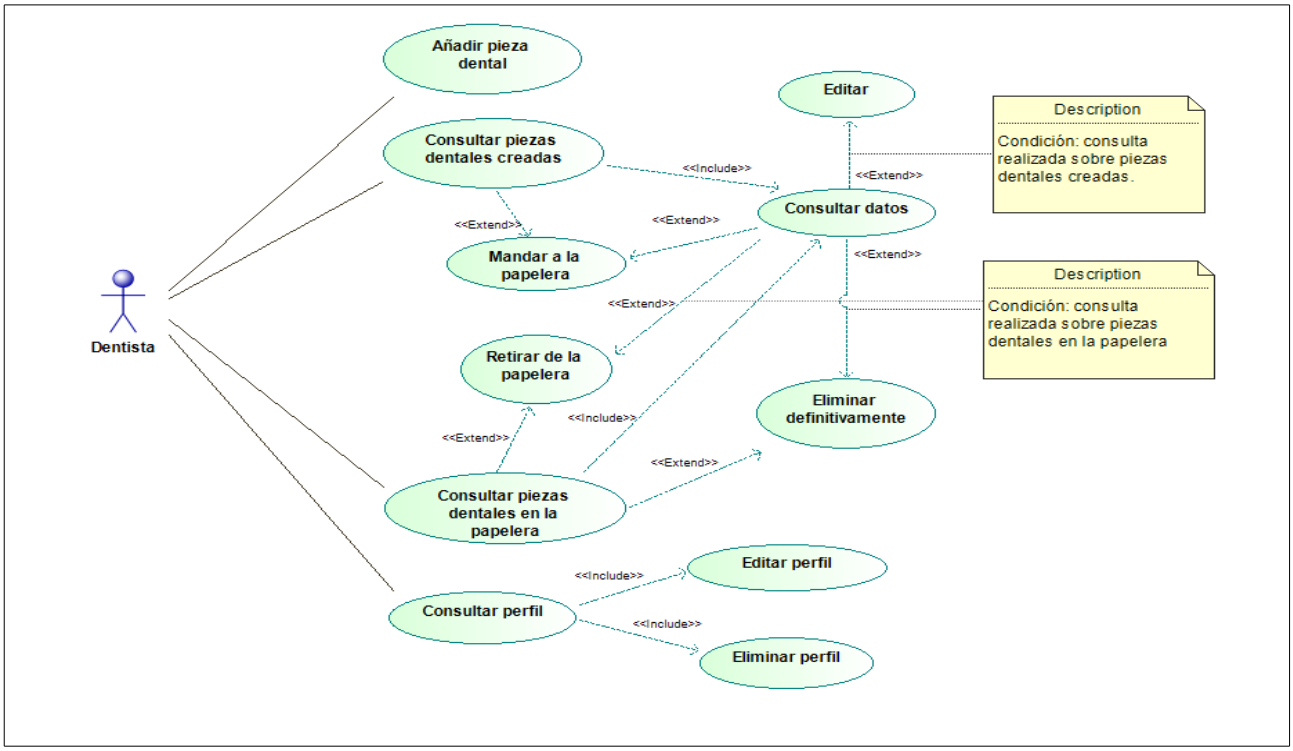


Figura 2.3

La figura 2.3 muestra las distintas posibilidades que se le otorgan al usuario una vez que se identifique como dentista en el sistema.
El dentista deberá poder crear piezas dentales nuevas, ver las que ya están creadas y editarlas o consultar aquellas que han sido solicitadas por algún estudiante.

2.5.4. Subsistema de gestión del usuario estudiante.

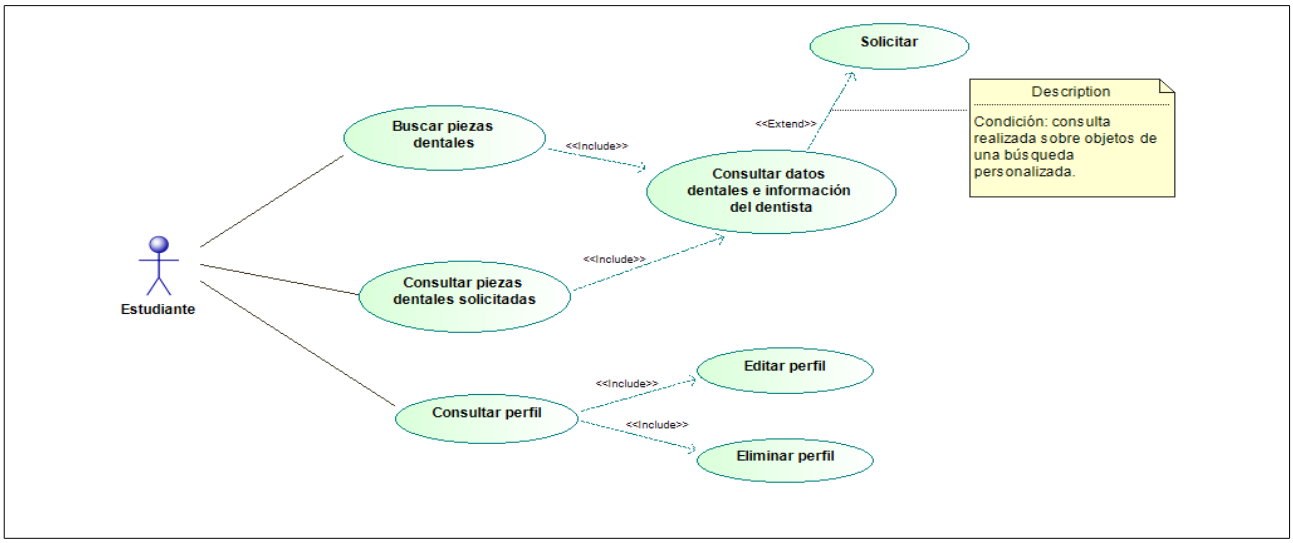


Figura 2.4

La figura 2.4 indica el funcionamiento del sistema cuando el usuario que ha iniciado la sesión es un estudiante.
Se le debe permitir realizar búsquedas de las piezas dentales que hay en el servidor las cuales podrá solicitar y consultar las que ya ha solicitado.

2.6. Descripción de los actores y de los datos a almacenar

Actor	**Usuario sin identificar**				
Descripción	Representa a un usuario que podrá registrarse en la aplicación o acceder con unos credenciales creados anteriormente.				
Características	Este actor representa a un cliente del sistema antes de pasar a ser un usuario registrado del sistema o registrase en él.				
Relaciones	Ninguna				
Referencias					
Autor	Javier Romera	Fecha	22/06/18	Versión	1.0

Actor	**Dentista**				
Descripción	Representa a un usuario registrado del tipo dentista.				
Características	Este actor representa a un usuario registrado en el sistema cuyo tipo es "dentista".				
Relaciones	Especialización de un usuario no registrado				
Referencias					
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Atributos		
Nombre	**Descripción**	**Tipo**
name	Nombre del usuario.	Cadena
surname	Apellido del usuario.	Cadena
dni	Documento de identidad del usuario	Entero
birth	Fecha de nacimiento del usuario.	Cadena
town	Población del usuario.	Cadena
city	Ciudad del usuario.	Cadena
address	Dirección del usuario.	Cadena
phone	Teléfono de contacto del usuario.	Long

clinic	Nombre de la cínica en la que opera el usuario.	Cadena
collegiate_number	Número de colegiado del usuario.	Long
contact_email	Correo electrónico de contacto del usuario.	Cadena
teeth_counter	Contador numérico de las piezas dentales creadas por el usuario.	Entero
search_limit	Límite de objetos mostrados por pestaña.	Entero

Actor	**Estudiante**				
Descripción	Representa a un usuario registrado del tipo estudiante.				
Características	Este actor representa a un usuario registrado en el sistema cuyo tipo es "estudiante".				
Relaciones	Especialización de un usuario no registrado				
Referencias					
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Atributos		
Nombre	**Descripción**	**Tipo**
name	Nombre del usuario.	Cadena
surname	Apellido del usuario.	Cadena
dni	Documento de identidad del usuario	Entero
birth	Fecha de nacimiento del usuario.	Cadena
town	Población del usuario.	Cadena
city	Ciudad del usuario.	Cadena
address	Dirección del usuario.	Cadena
phone	Teléfono de contacto del usuario.	Long
faculty	Nombre de la facultad donde estudia el usuario.	Cadena
contact_email	Correo electrónico de contacto del usuario.	Cadena
teeth_counter	Contador numérico de las piezas dentales solicitadas por el usuario.	Entero
search_limit	Límite de objetos mostrados por búsqueda.	Entero

Actor	**SGBD**
Descripción	Representa a la plataforma que permite la gestión, la lectura y la escritura en la

	base de datos.				
Características	Actor de gran importancia ya que se encarga del correcto funcionamiento de las funcionalidades del sistema.				
Relaciones	Dentista, Estudiante.				
Referencias					
Autor	Javier Romera	Fecha	22/06/18	Versión	1.0

Datos a almacenar	**Piezas dentales**				
Descripción	Representa a una pieza dental.				
Características	Son los datos que representan a una pieza dental que se añade al sistema.				
Relaciones	Sólo pueden ser incluidas por los usuarios de tipo dentista.				
Referencias					
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Atributos		
Nombre	**Descripción**	**Tipo**
id	Identificador de la pieza dental en el servidor.	Cadena
type	Determina el tipo de pieza dental seleccionada.	Cadena
tooth_number	Número de la pieza dental seleccionada.	Entero
blood_type	Tipo de sangre de la pieza dental.	Cadena
age	Edad de la pieza dental.	Entero
gender	Género sexual del propietario de la pieza dental.	Cadena
roots	Cantidad de raíces de la pieza dental.	Entero
color	Color representativo de la pieza dental.	Cadena
conservation	Tipo de conservación aplicada a la pieza dental.	Cadena
roots_state	Estado de las raíces de la pieza dental.	Cadena
crown_state	Estado de la corona de la pieza dental.	Cadena
pulp_treatment	Tratamientos pulpares realizados a la pieza dental.	Cadena
tooth_diseases	Enfermedades de la pieza dental.	Map (Cadena, Booleano)
owners_diseases	Enfermedades del propietario de la pieza dental.	Map (Cadena, Booleano)
others_tooth_diseases	Enfermedad de la pieza dental no incluida en el Map	Cadena

	tooth_diseases.	
others_owners_diseases	Enfermedad del propietario no incluida en el Map owners_diseases.	Cadena
enamel	Porcentaje de esmalte de la pieza dental.	Double
dentin	Porcentaje de dentina de la pieza dental.	Double
pulp	Porcentaje de pulpa de la pieza dental.	Double
image	Indica si la pieza dental tiene una imagen asignada.	Booleano
date	Fecha de alteración de los datos de la pieza dental	Date
creator	Identificador del creador de la pieza dental.	Cadena
buyer	Identificador del solicitante de la pieza dental.	Cadena
sold	Indica si la pieza dental ha sido solicitada por un estudiante.	Booleano
new_creation	Indica si la pieza ha sido revisada por el dentista una vez creada.	Booleano
new_buy	Indica si la pieza ha sido revisada por el estudiante. Sólo cambia cuando la pieza ha sido solicitada.	Booleano
delete	Indica si la pieza está o no en la lista de basura.	Booleano

2.7. Casos de uso

Caso de Uso	**Registrarse como estudiante**		**CU-1**		
Actores	Usuario sin identificar				
Tipo	Primario, Esencial				
Referencias					
Precondición	El usuario no está registrado en el sistema.				
Postcondición	Se registra en el sistema el nuevo usuario.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Registrar a un usuario no identificado en el sistema.

Resumen
El usuario sin identificar accederá a esta ventana donde rellenará un formulario. Si todo está correcto se le dará de alta en el sistema.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario accede a la ventana de registro como estudiante.	2	Muestra el formulario de registro.
3	Rellena el formulario.	4	Comprueba el valor de los datos.
5	Pulsa el botón de enviar el formulario.	6	Envía los datos del usuario al servidor.
		7	Recibe la confirmación de creación de la cuenta de usuario (usuario y contraseña).
		8	Se envía los datos personales del usuario.
		9	Se confirma la creación de la cuenta personal asociada y se le notifica la la correcta creación al usuario.

Curso alterno	
4a	Si el usuario no introduce correctamente alguno de los campos del formulario, los datos no se almacenarán.
6a	Si el envío falla, se le notificará el error al usuario con un mensaje por pantalla.
7a	Si se ha producido algún error en la creación, se le notificará el error al usuario mediante un mensaje por pantalla y se cancelará la creación.
8a	Si el envío falla, se le notificará el error al usuario con un mensaje por pantalla.
9a	Si se ha producido algún error en la creación, se le notificará el error al usuario mediante un mensaje por pantalla.

Caso de Uso	**Registrarse como dentista**		**CU-2**		
Actores	Usuario sin identificar				
Tipo	Primario, Esencial				
Referencias					
Precondición	El usuario no está registrado en el sistema.				
Postcondición	Se registra en el sistema el nuevo usuario.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Registrar a un usuario no identificado en el sistema.

Resumen
El usuario sin identificar accederá a esta ventana donde rellenará un formulario. Si todo está correcto se le dará de alta en el sistema.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario accede a la ventana de registro como dentista.	2	Muestra el formulario de registro.
3	Rellena el formulario.	4	Comprueba el valor de los datos.
5	Pulsa el botón de enviar el formulario.	6	Envía los datos del usuario al servidor.
		7	Recibe la confirmación de creación de la cuenta de usuario (usuario y contraseña).
		8	Se envía los datos personales del usuario.
		9	Se confirma la creación de la cuenta personal asociada y se le notifica la la correcta creación al usuario.

Curso alterno	
4a	Si el usuario no introduce correctamente alguno de los campos del formulario, los datos no se almacenarán.
6a	Si el envío falla, se le notificará el error al usuario con un mensaje por pantalla.
7a	Si se ha producido algún error en la creación, se le notificará el error al usuario mediante un mensaje por pantalla y se cancelará la creación.
8a	Si el envío falla, se le notificará el error al usuario con un mensaje por pantalla.
9a	Si se ha producido algún error en la creación, se le notificará el error al usuario mediante un mensaje por pantalla.

Caso de Uso	**Iniciar sesión**		**CU-3**		
Actores	Usuario sin identificar				
Tipo	Primario, Esencial				
Referencias					
Precondición	El usuario tiene una cuenta en el sistema.				
Postcondición	Se le permitirá el acceso al usuario al sistema.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir acceder a un usuario no identificado a su cuenta en el sistema.

Resumen
El usuario sin identificar introducirá su correo y su contraseña en la ventana de inicio de sesión. Si su relación correo-contraseña es correcta se le permitirá acceder al sistema.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario accede a la ventana de inicio de sesión.	2	Muestra el formulario de registro.
3	Rellena el formulario.	4	Comprueba los datos del formulario.
5	Pulsa el botón de inicio de sesión.	6	Envía los datos (usuario y contraseña) al servidor.
		7	Recibe tanto la confirmación de conexión permitida como los datos personales del usuario y se envía el id del usuario para determinar su tipo de cuenta.
		8	Se obtiene la respuesta del servidor y se le notifica la y se muestra la ventana correspondiente en función de su tipo de usuario.

Curso alterno	
4a	Si el usuario no introduce correctamente alguno de los campos del formulario, los datos no serán almacenados y se le notificará al usuario.
6a	Si el envío falla, se le notificará el error al usuario con un mensaje por pantalla.
7a	Si la recepción falla, se le notificará el error al usuario con un mensaje por pantalla.
7b	Si el envío falla, se le notificará el error al usuario con un mensaje por pantalla.
8a	Si se ha producido algún error en el acceso, se le notificará el error al usuario mediante un mensaje por pantalla.

Caso de Uso	**Añadir pieza dental**	**CU-4**
Actores	Dentista	
Tipo	Primario, Esencial	
Referencias		

Precondición	El usuario ha iniciado sesión como dentista.				
Postcondición	Se incluirán en el sistema los datos de la pieza dental y su imagen.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir que el usuario de tipo dentista cree una pieza dental que será incluida en el sistema.

Resumen
El dentista rellenará una formulario indicando los datos fundamentales de la pieza dental a incluir. así como una imagen representativa de dicha pieza.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El dentista pulsa el botón de creación de pieza dental.	2	Muestra un diálogo con las tres opciones disponibles: • Temporales • Permanentes (joven) • Permanentes (adulto)
3	Selecciona uno de los campos.	4	Muestra la pantalla de creación de piezas dentales.
5	Rellena el formulario de creación.		
6	Pulsa el botón de enviar.	7	Se envían al servidor los datos de la pieza dental.
		8	Recibe la confirmación de inclusión y envía al servidor los datos de la imagen.
		9	Recibe la confirmación y muestra un mensaje de que la pieza ha sido añadida correctamente.

Curso alterno	
3a	Si no selecciona nada, el usuario se mantendrá en la ventana actual.
7a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
8a	Si no se han podido añadir los datos al servidor, se le notificará al usuario el error mediante un mensaje por pantalla.
9a	Si no se ha podido añadir la imagen en el servidor, se le notificará al usuario el error mediante un mensaje por pantalla.

Caso de Uso	Consultar piezas dentales creadas	CU-5			
Actores	Dentista				
Tipo	Primario, Esencial				
Referencias					
Precondición	El usuario ha iniciado sesión como dentista.				
Postcondición	Se listarán las piezas dentales asociadas al dentista y que no han sido solicitadas.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Obtener las piezas dentales creadas por el dentista que están disponibles para ser solicitadas.

Resumen
El dentista obtendrá la lista de piezas dentales creadas por él mismo y que aún no han sido seleccionadas por el usuario.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario pulsa el botón de lista de piezas dentales.	2	Envía al servidor la petición de obtener los datos cuya id sea la del dentista y que no hayan sido solicitados.
		3	Recupera los datos asociados y muestra la lista de piezas dentales.

Curso alterno	
2a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
3a	Si se ha producido algún error en la recepción de los datos, se le notificará el error al usuario mediante un mensaje por pantalla.
3b	Si no se obtiene ningún dato, se le notificará la carencia de piezas dentales al usuario mediante un mensaje por pantalla.

Caso de Uso	Consultar datos	CU-6
Actores	Dentista	
Tipo	Primario, Esencial	

Referencias		CU-5, CU-9			
Precondición	Haber seleccionado un objeto de la lista de piezas dentales buscadas.				
Postcondición	Se mostrarán los datos de la pieza dental seleccionada.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Obtener los datos de una determinada pieza dental.

Resumen
El usuario obtendrá los datos individuales de una determinada pieza dental seleccionada de una lista de piezas.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario pulsa sobre uno de los objetos de la lista de piezas dentales.	2	Muestra una ventana con los valores de la pieza dental.

Curso alterno	
2a	Si se ha producido algún error, la ventana se cierra y devuelve al usuario a la ventana anterior.

Caso de Uso	**Editar**		**CU-7**		
Actores	Dentista				
Tipo	Primario, Esencial				
Referencias		CU-6			
Precondición	El usuario de tipo dentista ha incluido una pieza dental. Que el diente no haya sido solicitado por un usuario de tipo estudiante.				
Postcondición	Se alterarán los datos en la pieza dental.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir que el usuario de tipo dentista pueda alterar los datos de una pieza dental creada previamente.

Resumen
El dentista podrá editar los datos previos de una pieza dental creada por él mismo. Dichos valores se actualizarán en el servidor nada más alterarlos, siempre y cuando sean correctos.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El dentista selecciona uno de las piezas dentales creadas.	2	Muestra la pantalla de edición de la pieza dental.
3	Edita el campo deseado y acepta el nuevo valor.	4	Comprueba el valor del campo.
		5	Se envía el nuevo dato al servidor.
		6	Se recibe la notificación y se muestra por pantalla la correcta edición.

Curso alterno	
4a	Si el dato es incorrecto, su valor no será alterado.
5a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
6a	Si no se ha podido alterar el dato en el servidor, se le notificará al usuario el error mediante un mensaje por pantalla.

Caso de Uso	**Mandar a la papelera pieza dental creada**		**CU-8**		
Actores	Dentista				
Tipo	Primario, Esencial				
Referencias		CU-5, CU-6			
Precondición	El usuario de tipo dentista ha incluido una pieza dental. Que el diente aparezca en la lista de dientes creados.				
Postcondición	Se alterarán los datos en la pieza dental.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir que el usuario de tipo dentista pueda mandar a la papelera las piezas dentales creadas.

Resumen
El dentista podrá mandar a la papelera una pieza dental creada por él mismo que no haya sido

solicitada del sistema.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El dentista selecciona una o varias piezas dentales y pulsa el botón de enviar a la papelera.	2	Muestra un mensaje de confirmación de borrado.
3	Confirma el envío.	4	Envía al servidor la petición de cambio del estado del dato.
		5	Se recibe la notificación y se muestra por pantalla el envío.

Curso alterno	
2a	Si el usuario cancela el mensaje, se mantiene la ventana actual.
4a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
5a	Si el envío de la pieza se ha realizado desde la ventana de consulta datos, se devolverá al usuario a la ventana previa.
5b	Si no se ha podido alterar el dato en el servidor, se le notificará al usuario el error mediante un mensaje por pantalla.

Caso de Uso	**Consultar piezas dentales en la papelera**	**CU-9**
Actores	Dentista	
Tipo	Primario, Esencial	
Referencias		
Precondición	El usuario de tipo dentista ha enviado a la papelera una pieza dental.	
Postcondición	Se listarán las piezas dentales enviadas a la papelera por el dentista.	
Autor	Javier Romera **Fecha** 22/06/18 **Versión** 1.0	

Propósito
Obtener las piezas dentales que han sido enviadas a la papelera.

Resumen
El dentista obtendrá la lista de piezas dentales creadas que ha enviado a la papelera.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El dentista pulsa el botón de papelera.	2	Envía al servidor la petición de obtener los datos cuya id sea la del dentista y que estén en la papelera.
		3	Recupera los datos asociados y muestra la lista de piezas dentales.

Curso alterno	
2a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
3a	Si se ha producido algún error en la recepción de los datos, se le notificará el error al usuario mediante un mensaje por pantalla.
3b	Si no se obtiene ningún dato, se le notificará la carencia de piezas dentales al usuario mediante un mensaje por pantalla.

Caso de Uso	**Retirar de la papelera**		**CU-10**		
Actores	Dentista				
Tipo	Primario, Esencial				
Referencias		CU-9, CU-6			
Precondición	El dentista ha enviado una pieza dental a la papelera.				
Postcondición	Se retirará la pieza dental de la papelera.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir que el usuario de tipo dentista pueda retirar de la papelera las piezas dentales.

Resumen
El dentista podrá retirar de la papelera las piezas dentales que se hayan enviado previamente. Permitiendo que se puedan volver a mostrar en la lista de creadas o que los estudiantes puedan solicitarlas.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El dentista selecciona una o varias piezas que quiere retirar de la papelera y pulsa el	2	Envía al servidor la petición de actualización.

	botón de extraer.		
		3	Se recibe la notificación, se actualiza la lista y se muestra en un mensaje la correcta eliminación.

Curso alterno	
2a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
3a	Si no se ha podido actualizar la pieza dental del servidor, se le notificará al usuario el error mediante un mensaje por pantalla.

Caso de Uso	**Eliminar definitivamente pieza dental**		**CU-11**		
Actores	Dentista				
Tipo	Primario, Esencial				
Referencias		CU-9, CU-6			
Precondición	El usuario de tipo dentista ha enviado a la papelera una pieza dental.				
Postcondición	Se borrará definitivamente la pieza dental del sistema.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir que el usuario de tipo dentista pueda eliminar definitivamente los dientes creados.

Resumen
El dentista podrá eliminar del sistema una pieza dental creada por él mismo que no haya sido solicitada.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El dentista selecciona una o varias piezas a eliminar y pulsa el botón de eliminar definitivamente.	2	Muestra un mensaje de confirmación de eliminación.
3	Confirma la eliminación.	4	Envía al servidor la petición de borrado.
		5	Se recibe la notificación y se muestra por pantalla la eliminación.

Curso alterno	
2a	Si el usuario cancela el mensaje, se mantiene la ventana actual.
4a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
5a	Si la eliminación de la pieza se ha realizado desde la ventana de consulta datos, se devolverá al usuario a la ventana previa.
5b	Si no se ha podido eliminar la pieza dental del servidor, se le notificará al usuario el error mediante un mensaje por pantalla.

Caso de Uso	**Buscar piezas dentales**		**CU-12**		
Actores	Estudiante				
Tipo	Primario, Esencial				
Referencias					
Precondición	El usuario ha iniciado sesión como estudiante.				
Postcondición	Se listarán las piezas dentales según los criterios de búsqueda.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Obtener las piezas dentales en función de una búsqueda.

Resumen
El estudiante obtendrá una lista de piezas dentales en función de los parámetros que indique.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario pulsa el botón de búsqueda.	2	Muestra la ventana con las distintas opciones de limitar la búsqueda.
3	Selecciona los valores de búsqueda y pulsa el botón de buscar.	4	Solicita al servidor una búsqueda con los datos indicados.
5	Rellena el formulario de búsqueda.	6	Recupera los datos del servidor y muestra las piezas dentales obtenidas.

Curso alterno	
4a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.

6a	Si se ha producido algún error en la recepción de los datos, se le notificará el error al usuario mediante un mensaje por pantalla.
6b	Si no se obtiene ningún dato, se le notificará la carencia de piezas dentales buscadas al usuario mediante un mensaje por pantalla.

Caso de Uso	**Consultar piezas dentales solicitadas**		**CU-13**		
Actores	Estudiante				
Tipo	Primario, Esencial				
Referencias		CU-12			
Precondición	El usuario ha iniciado sesión como estudiante.				
Postcondición	Se listarán las piezas dentales que el estudiante haya solicitado.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Obtener las piezas dentales que hayan sido solicitadas.

Resumen
El estudiante obtendrá la lista de piezas dentales que haya solicitado previamente.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario pulsa el botón solicitados.	2	Solicita al servidor la lista de piezas dentales que tengan como solicitante el id del estudiante.
		3	Recupera los datos del servidor y muestra las piezas dentales obtenidas.

Curso alterno	
2a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
3a	Si se ha producido algún error en la recepción de los datos, se le notificará el error al usuario mediante un mensaje por pantalla.
3b	Si no se obtiene ningún dato, se le notificará la carencia de piezas dentales buscadas al usuario mediante un mensaje por pantalla.

Caso de Uso	**Consultar datos dentales e información del dentista**		**CU-14**		
Actores	Estudiante				
Tipo	Primario, Esencial				
Referencias		CU-15			
Precondición	El usuario ha seleccionado un objeto de un lista de búsqueda.				
Postcondición	Se mostrarán los datos de una pieza dental y a la información del creador.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Mostrar los datos de una pieza dental y de su creador.

Resumen
El estudiante obtendrá los datos de una pieza dental y los del dentista que la añadió al sistema.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El estudiante pulsa sobre un elemento de la lista de piezas dentales buscadas.	2	Envía una petición de obtención al servidor de los datos de la pieza dental y de su creador.
		3	Recibe la información del servidor y la muestra en una ventana nueva.

Curso alterno	
2a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
3a	Si se ha producido algún error en la recepción de los datos, se le notificará el error al usuario mediante un mensaje por pantalla.

Caso de Uso	**Solicitar**	**CU-15**
Actores	Estudiante	
Tipo	Primario, Esencial	
Referencias	CU-14	
Precondición	El usuario ha realizado un búsqueda de piezas dentales.	
Postcondición	Se cambiará los datos de la pieza dental indicando que ha sido solicitada.	

Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Solicitar una pieza dental disponible.

Resumen
El estudiante solicitará una pieza dental de las disponibles y se alterarán sus datos en el servidor añadiendo el nuevo estado y al solicitante.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El estudiante pulsa el botón de solicitar.	2	Comprueba que la pieza dental no ha sido alterada durante su acceso.
		3	Envía una petición de alteración al servidor de los datos de la pieza dental para incluir al solicitante.
		4	Recibe la confirmación de alteración del servidor y muestra un mensaje por pantalla.

Curso alterno	
2a	Si la pieza ha sido alterada, se le notificará al usuario el error mediante un mensaje por pantalla.
3a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
4a	Si se ha producido algún error en la recepción de los datos, se le notificará el error al usuario mediante un mensaje por pantalla.

Caso de Uso	**Consultar perfil**	**CU-16**			
Actores	Dentista, Estudiante				
Tipo	Primario, Esencial				
Referencias					
Precondición	El usuario ha iniciado sesión como dentista.				
Postcondición	Se mostrarán los datos asociados al usuario.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir que el usuario vea sus datos de perfil.

Resumen
El usuario obtendrá los datos de su perfil incluyendo los cambios más recientes en él.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario pulsa el botón de perfil.	2	Envía al servidor la id del usuario y su tipo.
3		3	Recupera los datos de usuario y los muestra en una ventana nueva.

Curso alterno	
2a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
3a	Si se ha producido algún error en la recepción de los datos, se le notificará el error al usuario mediante un mensaje por pantalla.

Caso de Uso	**Editar perfil**		**CU-17**		
Actores	Dentista, Estudiante				
Tipo	Primario, Esencial				
Referencias		CU-16			
Precondición	El usuario ha iniciado sesión.				
Postcondición	Se alterarán los datos del perfil de usuario en el sistema.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir que el usuario pueda alterar su perfil.

Resumen
El usuario registrado podrá cambiar los datos de su perfil en el sistema.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario pulsa sobre el botón de editar	2	Muestra en un formulario los datos de

	perfil.		perfil del usuario.
3	Edita un campo y acepta el cambio	4	Comprueba si el valor nuevo es correcto.
		5	Envía al servidor una petición de alteración del dato indicado.
		6	Recibe la confirmación, cambia el valor del dato en el formulario y le muestra al usuario la confirmación.

Curso alterno	
4a	Si el valor es incorrecto, el valor no se altera y se vuelve al punto 2.
5a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
6a	Si se ha producido algún error en la inclusión, se le notificará el error al usuario mediante un mensaje por pantalla.

Caso de Uso	**Eliminar perfil**		**CU-18**		
Actores	Dentista, Estudiante				
Tipo	Primario, Esencial				
Referencias		CU-16			
Precondición	El usuario ha iniciado sesión.				
Postcondición	Se eliminarán los datos de usuario en el sistema.				
Autor	Javier Romera	**Fecha**	22/06/18	**Versión**	1.0

Propósito
Permitir que el usuario pueda eliminar su cuenta.

Resumen
El usuario registrado podrá eliminar su cuenta personal del sistema, así como sus datos de perfil.

Curso normal			
Acción del Actor		**Acción del sistema**	
1	El usuario pulsa sobre el botón de eliminar perfil.	2	Muestra un mensaje de confirmación.
3	Confirma la eliminación.	4	Envía al servidor la petición de eliminación del sistema.
		5	Recibe la confirmación de eliminación y

			devuelve al usuario a la pantalla inicial.

Curso alterno	
3a	Si no se confirma, se mantiene al usuario en la ventana actual.
4a	Si se ha producido algún error en el envío, se le notificará el error al usuario mediante un mensaje por pantalla.
5a	Si se ha producido algún error en la eliminación, se le notificará el error al usuario mediante un mensaje por pantalla.

2.8. Glosario de términos

- **Dentista**: Usuario de la aplicación móvil del tipo dentista que puede subir los datos de las piezas dentales.
- **Estudiante**: Usuario de la aplicación del tipo estudiante que puede solicitar las piezas dentales subidas por los dentistas.
- **Android**: Sistema Operativo de Google usado en el lado del cliente.
- **Android Studio**: Es el entorno de desarrollo integrado oficial de Android.
- **Firebase**: Servidor web de aplicaciones de Google.
- **Administrador**: Programador que se encarga de gestionar el servidor de Firebase.
- **Actividad** (Activity): componente de la aplicación de Android que tiene asociada una ventana donde interactúan los usuarios.
- **Fragmento** (Fragment): representa un comportamiento o una parte de la interfaz de usuario en una Activity.
- **Diálogo** (Dialog): pequeña ventana emergente que le puede pedir al usuario que acepte o escriba algo en ella.
- **Vista** (view): representa el espacio rectangular de la pantalla móvil donde se dibujan los objetos que serán usados por la actividad o el fragmento asociado.
- **Itent**: objeto que sirve de enlace con otro objeto en tiempo de ejecución.
- **Adaptador (adapter)**: interfaz que asocia a una vista una lista cuyos elementos, de manera independiente, están asociados a otro objeto vista.
- **Snackbar**: Pequeña ventana de notificación cuadrada que aparece, por lo común, en la parte inferior de la vista. Es una versión mejorada de Toast que permite incluir botones de acción.
- **Log**: objeto de la API Log que permite generar mensajes en tiempo de ejecución que serán mostrados en Android Studio.
- **XML**: Lenguaje de marcado extensible usado en los archivos que no contienen código Java de Android.
- **Interfaz** (layout): archivo XML donde se almacena la estructura visual asociada a la vista.
- **Gradle**: sistema de compilación de Android Studio que compila recursos y código fuente de la aplicación y los empaqueta en un archivo APK.
- **Manifiesto** (Manifest): documento XML donde se define el nombre de aplicación e información general, así como los permisos y recursos que usará la aplicación de manera global y, de manera específica, sus actividades.

Capítulo 3. Base tecnológica

3.1. Posibles soluciones

Se necesitan dos condiciones para llevar a cabo el desarrollo solicitado: una aplicación para el cliente que resulte cómoda y fácil de utilizar y de acceder y un proveedor de recursos cuya administración sea completamente ajena al usuario y con la cual conecte la aplicación para el intercambio de datos.

Como se puede observar, el objetivo final es desarrollar un modelo software que cumpla con la arquitectura cliente-servidor.

3.1.1. El cliente

Para el lado del cliente se necesita una herramienta cuyo uso esté extendido entre una gran cantidad de usuarios y que sea fácil de usar sin tener grandes conocimientos sobre su manejo.

Para alcanzar el objetivo de esta tarea existen distintas soluciones se entre las cuales destacan: diseñar algunas páginas web almacenadas en servidor local o remoto, programar una aplicación para la Plataforma Universal de Windows (UWP) o o desarrollar una aplicación móvil (en iOS o Android).

3.1.1.1. Página web para dispositivos móviles

El desarrollo de una página web enfocada a los dispositivos móviles puede ser una gran alternativa ya que cumple con todos los requisitos y simplificaría la gestión de las conexiones entre la página web y el servidor donde se encuentre la base de datos porque, en principio, ambas establecerían conexiones de manera opaca al usuario.

La documentación para el desarrollo web (HTML, CSS, PHP, etc) no supondría un problema debido a que es de fácil acceso al haber gran cantidad de ella en diferentes blogs y foros de Internet.

A pesar de ello, obliga al usuario a acceder a un navegador web y desde ella acceder a la dirección indicada. Dos pasos que pueden resultar tediosos si se pretende usar la herramienta desde un dispositivo móvil.

3.1.1.2. La aplicación UWP [inf 07]

Desarrollar una aplicación para la Plataforma Universal de Windows permitiría usarla tanto en ordenadores y móviles que tengan como sistema operativo a Windows 10.
Su programación y diseño sería la misma tanto en la versión de escritorio como en la versión móvil debido a que usa APIs comunes de UWP.

Su principal problema es que limita su uso a aquellos dispositivos que tengan a Windows 10 como sistema operativo y, en dispositivos móviles, este no supone nada más que el 0,47% de todos los disponibles en el mercado. [inf 09]

3.1.1.3. Aplicación móvil (iOS y Android)

La programación de aplicaciones móviles ha experimentado un gran crecimiento en los últimos tiempos. Estas permiten a los usuarios acceder fácilmente a un sinfín de actividades en unos pocos segundos las cuales pueden gestionar abundante cantidad de información en pocos segundos.

La gran cantidad de documentación disponible en Internet así como los blogs y foros donde las comunidades de programadores se reúnen para solucionar problemas permite afrontar su desarrollo de una manera cómoda.

El problema principal está en el hecho de que los dos principales sistemas operativos para los que se desarrollan aplicaciones (Android e iOS) usan tecnologías y lenguajes diferentes lo que obliga a decantarse por una de ellas.
También se obliga a que la aplicación deba conectar con un servidor web que la provea de los datos necesarios, por lo que se obliga a programar y gestionar esas conexiones.

3.1.2. El servidor

Su prepósito principal será el de almenar los datos y responder a las peticiones que se envíen desde la aplicación creada. Todo ello se debe hacer mediante canales seguros de comunicación.

3.1.2.1. Servidor web REST

Para desarrollar los requerimientos del servidor una buena idea es el desarrollo de un servidor que use una API REST.
REST permite establecer comunicaciones entre la aplicación y el servidor mediante el protocolo HTTP lo que permite transferencias de información en formato JSON o XML entre ellas de manera sencilla.

Su problema radica en la necesidad de programar tanto la infraestructura del servidor, así como la seguridad que se debe aplicar a los datos almacenados en el servidor (nadie puede acceder a ellos sin tener las credenciales correctas) y a las comunicaciones entre el cliente y el servidor mediante el protocolo HTTPS.

3.1.2.2. PaaS

Otra alternativa es el uso de una plataforma de servicios en la nube que permita la creación de una aplicación sin tener que concentrarse en la infraestructura del servidor lo que permite desplegar aplicaciones de manera más rápida y sencilla.

Su problema radica en que los protocolos de seguridad para las transferencias entre el cliente y el servidor se deben programar y que la aplicación estará almacenada en una plataforma la cual no podemos controlar y que limitará su uso al plan de pago que se haya seleccionado.

Algunos ejemplos son: Amazon EC2, Heroku, Microsoft Azure o Cloud Foundry.

3.1.2.3. BaaS [inf 10]

En la actualidad y gracias a la popularidad de los servicios en la nube y el desarrollo de aplicaciones móviles, ha ido apareciendo un nuevo tipo de servicio que permite la relación entre las dos de manera cómoda y sencilla, el llamado “Backend as a Service". Los servidores que proporcionan este servicio ofrecen una serie de aplicaciones ya desarrolladas en las cuales apenas se requiere de configuración o programación lo que permite tener a disposición del usuario una plataforma completa en poco minutos.

Entre sus herramientas más comunes se encuentran aquellas que permiten almacenar datos, usar servicios de análisis o autentificar a los usuarios de la plataforma para asegurar la confidencialidad e integridad de los datos.

El gran problema que poseen radica en el hecho de que sus herramientas apenas pueden ser configuradas, obligando a adaptarse a ellas, así como comprender que los datos que se almacenen en estos servidores estarán sometidos a una protección de datos independiente el usuario.

Algunas de estas plataformas son: MOCA, KONA o Firebase.

3.2. Desarrollo de una solución

Como uno de los mayores objetivos es el que la aplicación del cliente sea nativa a alguno de los sistemas operativos móviles disponibles, sea accesible por el mayor número de personas posibles y también sea sencilla de usar se ha descartado la posibilidad de diseñar una página web decantándose por el desarrollo de una aplicación móvil.

	Web	**WUP**	**iOS**	**Android**
Conocimiento previo	Sí	No	No	No
Conocimiento del lenguaje	Sí	No	No	Sí
Aplicació nativa	No	Sí	Sí	Sí
Mercado móvil	Alto	Bajo	Medio	Alto
Dispositivo para pruebas	Sí	Sí	No	Sí

Tabla 3.1. Tabla comparativa de las herramientas móviles.

El sistema operativo sobre el que trabajar se ha seleccionado tras comprobar que actualmente el más usado por los dispositivos móviles es Android con un 75% de la cuota del mercado. [inf 09]

Para su desarrollo se deberá instalar el SDK y el IDE de Android y optar por programar en uno de los dos lenguajes oficiales: Java o Kotlin. [inf 08]
Debido a su familiaridad, por uso uso durante la carrera, se ha optado por usar Java para el completo desarrollo de la aplicación.

Al decidirse por esta tipo de aplicación, se necesitará de un servidor que proporcione de los recursos necesarios para almacenar y gestionar los datos. Para esa función se ha optado por usar el BaaS Firebase (Google LLC.).

	REST	**PaaS**	**BaaS**
Conocimiento previo	No	Sí	No
Encriptación de la BD	No	No	Sí
Múltiples herramientas	No	No	Sí
API para móvil	No	No	Sí
Condiciones legales	No	Sí	Sí

Tabla 3.2. Tabla comparativa de las opciones para los servidores.

Puesto que su desarrollo, al ser una tecnología nueva que abarca muchas posibilidades, está en pleno ascenso y que proporciona una serie de APIs para establecer comunicaciones seguras entre sus herramientas y las aplicaciones móviles para Android (misma compañía) ha influenciado para que sea la alternativa escogida.

Su único escollo está en el hecho de que muchas de sus herramientas están aún en fase beta por lo que su uso y rendimiento está sujeto a cambios. Algunos de estos pueden producirse sin previo aviso lo que dificulta su gestión obligando al desarrollador a estar constantemente atento a nuevos cambios.

3.2.1. Android [and 01]

Para programar en Android se han de tener conocimientos básicos acerca de los componentes básicos que toda aplicación tiene, así como entender sus ciclos de vida:

- Actividades.
- Servicios.
- Proveedores de contenido.
- Receptores de mensajes.

Para la iniciar las actividades, los servicios o los receptores de mensajes será necesario el uso del objeto *Intent* que sirve para crear un mensaje de activación de los distintos componentes que se indiquen de manera asíncrona. En el caso de los proveedores de servicio será necesario el uso del objeto *ContentResolver*.
Los métodos para la ejecución de cada uno de los componentes son:

- Actividades: s*tartActivity()* y *startActivityForResult()*. Este último busca obtener algún tipo de respuesta antes de que se destruya la actividad lanzada.

- Servicios: *startService()*.
- Proveedor de contenido: *query()*.
- Receptores de mensajes:*sendBroadcast()*, *sendOrderedBroadcast()* o *sendStickyBroadcast()*.

A continuación, se detallarán los aspectos más importantes de cada uno de ellos y también de otros cuyo uso ha sido necesario para el desarrollo de este trabajo.

3.2.1.1. Actividades [and 02]

Las actividades son las clases básicas de Android y serán lo primero que se muestre al iniciar una aplicación (a excepción de la ventana de carga donde se muestra el logotipo). A ellas se les asigna una ventana donde se incluirá todo el contenido con el que el usuario de la aplicación podrá interactuar (interfaz de usuario) y cuyo comportamiento deberá ser programado por el desarrollador.

Cada actividad implementada podrá lanzar otras actividades, todas ellas, incluida la inicial, se irán almacenando en un **pila** cuyo funcionamiento se basa en el concepto LIFO (último en entrar, primero en salir). El acceso a esta pila está permitido por lo que se podrá alterar su contenido mediante los métodos que se proporcionan.

Se puede **enviar información** de una actividad padre a otra invocadora añadiendo al intent creado los parámetros que se requieran mediante un identificador textual antes de ejecutarlo.
Los siguientes códigos ejemplifican su comportamiento:

```
Intent intent = new Intent(getContext(), Clase.class);
intent.putExtra("data", data);
startActivity(intent);
```

Código 3.1. Envío de datos a una nueva actividad.

```
toothData = (ToothData) getIntent().getIntExtra("data");
```

Código 3.2. Recogida de los datos enviados en el código 1.

Para recoger información desde la actividad lanzada hacia la padre se deberá ejecutar la actividad con *startActivityForResult(intent)* y recoger los datos mediante *onActivityResult(int requestCode, int resultCode, Intent data)*.

Para su desarrollo hay que tener presente que todas las actividades que se creen heredan de una clase **Activity** principal y cuyo ciclo de vida es asignado a cada una de ellas. Este ciclo no es alterable y una mala implementación de ellos podría provocar que la aplicación deje de funcionar.

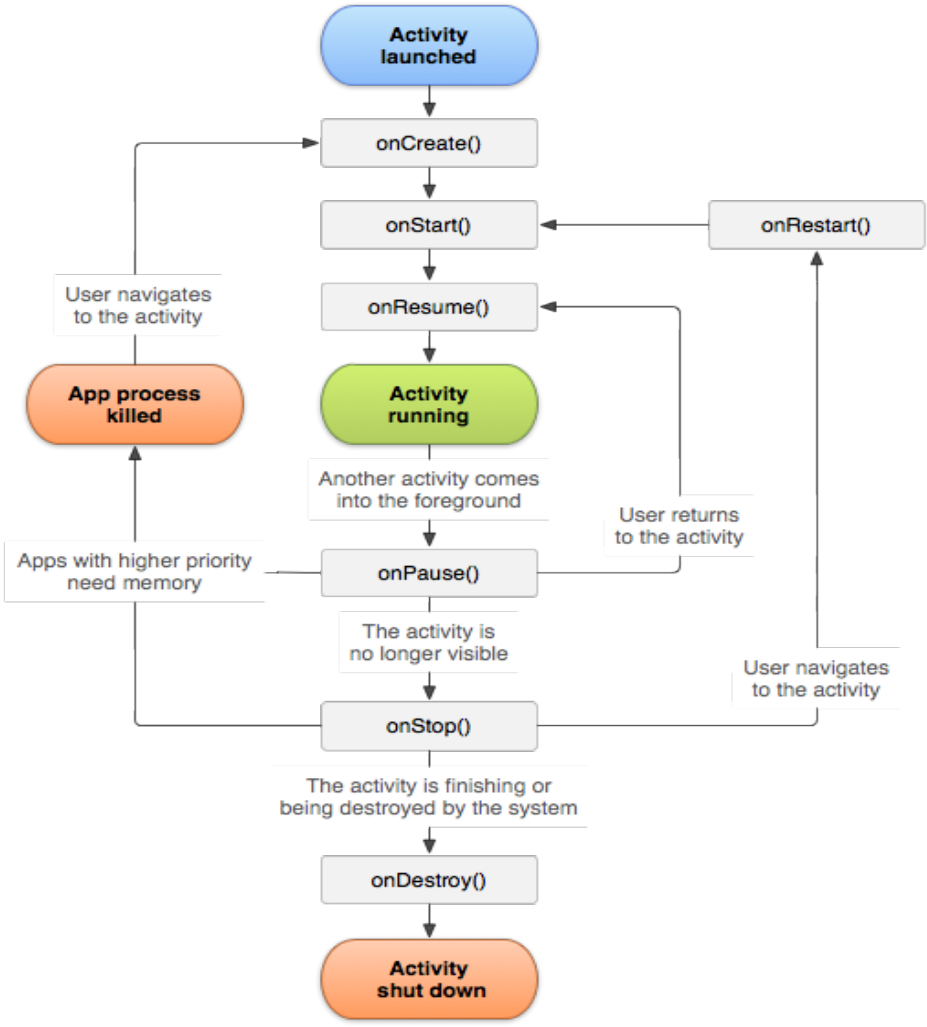


Figura 3.1. Ciclo de vida de una actividad.

Los métodos de la figura 3.1 son los siguientes:

- *onCreate()*: donde se indica la interfaz que se usará en la ventana y especifica el comportamiento de los objetos contenidos en ella.
- *onStart()*: es llamada nada más mostrarse la ventana creada al usuario.
- *onPause():* se llama cuando se notifica que se va a ejecutar otra ventana que sustituye la actual.
- *onResume()*: es llamada cuando una actividad vuelve a recuperarse de la pila. Siempre es el siguiente método después de *onPause()*.
- *onStop()*: se llama una vez la actividad ha sido sustituida y ya no es visible. Esto se puede deber a que ha sido destruida o a que ha sido sustituida por otra.
- *onRestart()*: se llama justo cuando la actividad se detiene y no cuando vuelve a inicializarse.
- *onDestroy()*: es el último método al que se llama justo antes de que la actividad sea destruida.

3.2.1.2. Servicios [and 03]

Permiten la ejecución en segundo plano de componentes de la aplicación y sin la necesidad de interacción por parte del usuario.

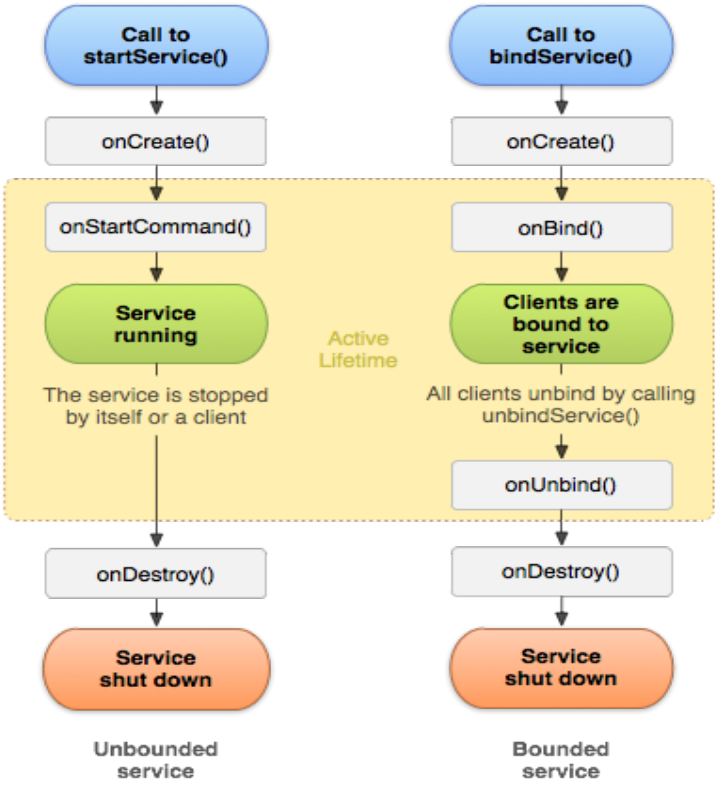


Figura 3.2. Ciclos de vida de los servicios.

Algunos ejemplos de uso pueden ser: gestionar datos del usuario, recuperar fechas del calendario, lanzar música, almacenar datos de la aplicación dentro de la memoria del dispositivo o establecer comunicaciones con algún servidor.
Existen dos tipos de servicios:

- Servicio iniciado: se crea cuando un componente lo llama y se ejecuta de manera indefinida hasta que se detiene (por sí mismo o por otro componente)
- Servicio de enlace: el servicio se crea mediante un componente cliente que lo usará hasta que este notifique que ya no le es necesario y el sistema lo destruya. Ofrece una interfaz que permite que los componentes interactúen con él.

Como se muestra en la figura 3.2, por cada forma de lanzar un servicio existe un ciclo de vida distinto. Para iniciar los servicios iniciados, se usa el método *startService()* y para los servicios de enlace, *bindService()*.

Los métodos de un servicio de inicio son :

- *onCreate()*: se llama nada más llamar al servicio y determina su configuración inicial.
- *onStartCommand()*: es llamado para entregarle el objeto Intent.

- *onDestroy():* se llama cuando se decide destruir el servicio.

Los métodos de un servicio de enlace son :

- *onCreate():* se llama nada más llamar al servicio y determina su configuración inicial.
- *onBind()*: es llamado para entregarle el objeto Intent y enlazarlo con el servicio.
- *onUnbind()*: llamado para desenlazar el servicio.
- *onDestroy()*: se llama cuando se decide destruir el servicio.

3.2.1.3. Proveedores de contenido [and 04]

En Android existe una base de datos interna donde se almacena toda la información relativa a las aplicaciones cuya gestión se realiza mediante SQLite. Los proveedores de contenido permite que las distintas aplicaciones instaladas puedan acceder y compartir los datos de ella siempre que les sea necesario.

Un ejemplo de esto podría ser la lista de contactos que posee un usuario en su teléfono móvil a la cual acceden tanto la aplicación de contactos como las que son de mensajería.

Los accesos a los datos se realizan mediante el uso de identificadores de recursos uniformes (URI) y se podrán añadir limitaciones de uso para evitar que puedan ser accedidos por otras aplicaciones.

También se pueden crear proveedores de contenido para almacenar y recoger información en el sistema de archivos o en algún servidor Web.

3.2.1.4. Receptores de mensajes [and 05]

Se trata de un componente que gestiona todas las notificaciones del sistema como pueden ser aquellas que indican la cantidad de batería disponible, la conexión con algún dispositivo, el estado de la pantalla, etc.

También permite a una aplicación lanzar sus propios mensajes de notificación para que otras lo reciban o indicarle al usuario la realización de alguna de ellas mediante una notificación en la barra de estado.

3.2.1.2. Fragmentos [and 06]

Son clases que permiten la gestión de las interfaces de usuario de las actividades. Su uso está completamente relacionado con el de estas últimas ya que son necesarias para la ejecución de los fragmentos.

Cada fragmento será gestionado mediante una pila de fragmentos LIFO que estará asignada a la actividad anfitriona y que si esta es elimina la pila también será destruida.

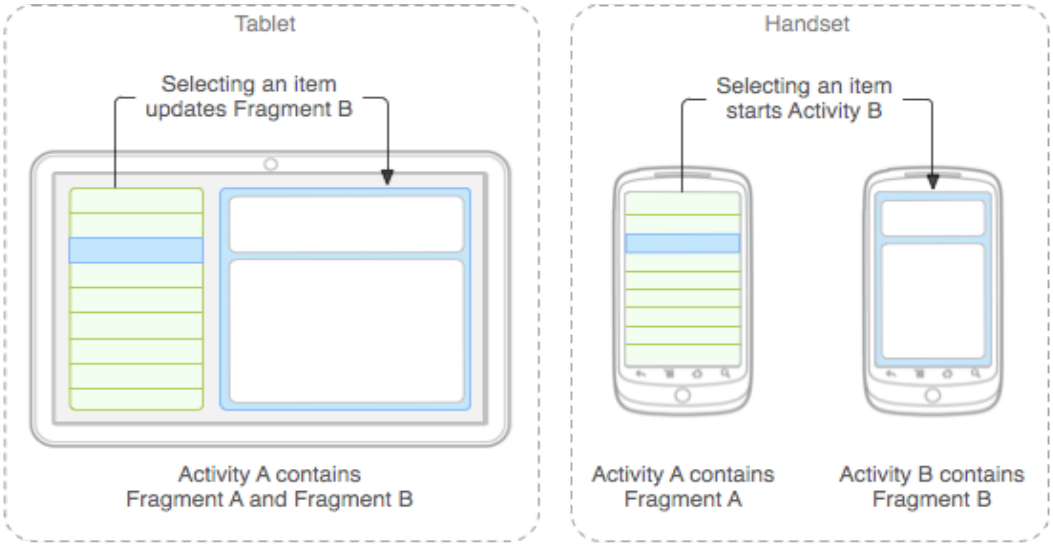


Figura 3.3. Ejemplo de funcionamiento de los fragmentos.

Pueden ser ejecutados por distintas actividades que no guarden relación entre sí y pueden ser ejecutadas varias a la vez permitiendo interfaces multipanel como se muestra en la figura 3.3.

Para enviar información a uno de ellos antes de su creación hay que crear un método estático (dentro de su clase) que reciba las variables deseadas y que devuelva dicho fragmento. Dentro de él se incorporarán las variables necesarias mediante un objeto Bundle asociado.
El siguiente ejemplo muestra su funcionamiento:

```
public static FragmentoCreado newInstance(int data {
    FragmentoCreado fragment = new FragmentoCreado();
    Bundle args = new Bundle();
    args.putInt("entero", data);
    fragment.setArguments(args);
    return fragment;
}
```

Código 3.3. Creación de una instancia con receptor de datos.

Al igual que las actividades también presentan un ciclo de vida que se ha de seguir en todo momento:

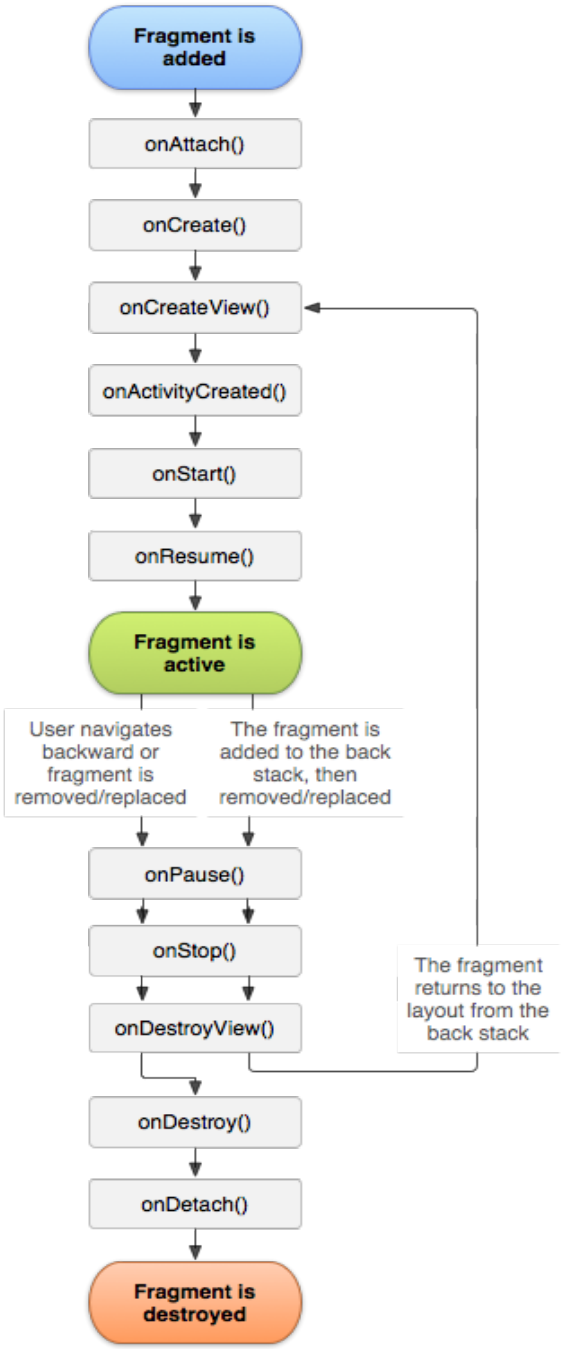


Figura 3.4. Ciclo de vida de un fragmento.

Los métodos de la figura 3.4 son los siguientes:

- *onAttach()*: se llama cuando se asocia un fragmento a una actividad.
- *onCreate()*: donde se indica la interfaz que se usará en la ventana y especifica el comportamiento de los elementos contenidos en ella.
- *onCreateView()*: se llama para crear la interfaz de usuario que usará el fragmento. Se debe devolver en un objeto View.
- *onActivityCreate()*: es llamado sólo cuando el método *onCreateView* realice una llamada a la

actividad anfitriona.
- *onStart()*: es llamada nada más mostrarse la interfaz creada al usuario.
- *onResume()*: está asociada al comportamiento de la actividad anfitriona. Es llamada cuando la actividad es recuperada de la pila.
- *onPause()*: está asociada al comportamiento de la actividad anfitriona. Se llama cuando se notifica que se va a abandonar la actividad actual.
- *onStop()*: se llama una vez que el fragmento ha sido sustituido y ya no es visible.
- *onDestroyView()*: se llama para eliminar la jerarquía de vistas del fragmento.
- *onDestroy()*: se llama justo antes de que el fragmento sea destruido.
- *onDetach()*: se llama para eliminar la asociación del fragmento con la actividad .

3.2.1.5. Diálogos [and 07]

Los diálogos son pequeñas ventanas emergentes que se muestran en la interfaz de usuario. Pueden mostrar mensajes de texto o permitir la selección de elementos de una lista o la inserción de texto mediante teclado.

Es una subclase de la clase fragmento que puede ser utilizada tanto por ellas como por las actividades y permite ser llamada tantas veces como sea necesario.

También indicar que tienen el mismo ciclo de vida y usan la misma pila que los fragmentos tienen asignada en la actividad invocadora. Esto obliga que su uso se tenga que realizar con cuidado y gestionando cada uno de los elementos contenidos en ella.

3.2.1.5. Diseños (XML) [and 08]

Cada una de las interfaces de usuario de la aplicación consta de dos partes: una actividad o fragmento que la gestione y un layout donde se almacenen la estructura con los contenidos visuales.

Los layouts son archivos creados mediante el lenguaje de marcado extendible (XML) que permiten diseñar y crear todos los elementos que definen la estructura de las interfaces de usuario, así como definir cada uno de sus atributos (color, tamaño, posición en pantalla, etc).

Los XML también son usados para gestionar cadenas de texto, colores, iconos o tamaños dentro de la aplicación. Estos archivos se crean y organizan en directorios por defecto, una vez el proyecto de la aplicación se crea, dentro de la carpeta *res*:

- **Directorio drawable**: contiene a los objetos e imágenes que se usarán.
- **Directorio layout**: incluye todos los layouts usados en la aplicación.
- **Directorio menu**: incluye a los archivos de la barra de herramientas y de menú que se usarán el layout del menú.
- **Directorio mipmap**: especifica las imágenes que se usarán en la app como “iconos iniciadores”. Se usa un directorio distinto a drawable por motivos de resolución.
- **Directorio values**: contiene archivos relativos a las cadenas de texto, las listas, los colores,

las dimensiones y los estilos usados.

Cada uno de los elementos contenidos en cualquier archivo XML puede ser referenciado por la clase que la gestiona mediante identificadores. Esto permite que se pueda alterar dinámicamente los parámetros de cada uno de los elementos en función de las necesidades.

3.2.1.7. Interfaces de escucha (Listeners)

La forma más rápida y eficaz de establecer comunicación entre fragmentos (normales y diálogos) o entre estos y las actividades es mediante el uso de escuchas.

Las escuchas no son más que interfaces que se definen en la clase que generará el mensaje que deberán estar implementadas en la clase receptora.

En el caso de que las escuchas sean hacia una actividad padre las escuchas han de implementarse de la siguiente manera:

```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    try {
        listener = (DialogFragment.DialogListener) context;
    }catch (ClassCastException e) {
        Log.w("ERROR", "listener failure", e.getCause());
    }
}

public interface DialogListener{
    void applyDialog(String dialog);
}
```

Código 3.4. Creación de un listener para una actividad padre.

En el caso de que las escuchas sean hacia un fragmento padre las escuchas han de implementarse de la siguiente manera:

```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    try {
        listener = (DialogFragment.DialogListener)
getTargetFragment();
    }catch (ClassCastException e) {
        Log.w("ERROR", "listener failure", e.getCause());
    }
}

public interface DialogListener{
    void applyDialog(String dialog);
}
```

Código 3.5. Creación de un listener para un fragmento padre.

Este sistema es semejante al que se usa para recibir los eventos de entrada de cada uno de los componentes de la interfaz. [and 09]

3.2.1.8. El manifiesto [and 10]

El manifiesto es un documento en XML que define la información vital que usará la aplicación. Este documento especifica el nombre del proyecto, define su identificador, los recursos que usará y los permisos que serán necesarios para su funcionamiento (acceso a datos sensibles del dispositivo). También se definen las características de cada una de las actividades de las que consta la aplicación especificando sus nombres, los recursos que usan e incluso si es necesario almacenar los datos de la aplicación en caso de que se rote la pantalla asignada.

Este archivo de nombre AndroidManifest.xml es vital para cualquier proyecto y no puede ser eliminado o alterado de manera inapropiada porque provocaría errores de compilación o que, una vez instalada en un dispositivo, la aplicación no funcione correctamente.

3.2.1.9. Las desventajas de Android

La documentación oficial.
Hay que tener presente que, dependiendo de la versión web que se use como guía para Android (*android.com* o *developer.android.com)* o de si la página está en su versión original o en otro idioma, la información y la forma de programar algunas funcionalidades cambian.

También se puede observar que la documentación no abarca toda la complejidad de Android, la API de Android en algunos casos carece de la suficiente información, y que resulta necesario realizar búsquedas en blogs o en foros (*stackoverflow.com* o *android-developers.googleblog.com*, por ejemplo) para obtener más detalles.

Esto puede ser un problema cuando el programador es novato en esta tecnología, busca información sobre algún aspecto importante y se encuentra en la tesitura de escoger un método u otro de diseño para progresar.

Los ciclos de vida.
Cada una de las actividades, fragmentos o servicios que se implementen tienen asignado un ciclo de vida que no puede ser alterado ni modificado.
Es por ello que todos aquellos métodos programados que desarrollen algún tipo de gestión han de ejecutarse a través de estos y deben ser robustos ante cualquier parada, pausa o destrucción que se produzca.

Seguir y comprender estos ciclos es vital para el correcto desarrollo del proyecto y obliga a analizar el comportamiento de cada línea de código incorporada a cada uno de esos métodos cuando se crean.

El SDK
Al iniciar el proyecto de una aplicación hay que definir cuál será el SDK mínimo para el que funcionará. Esta decisión no es insustancial porque podría suponer un problema en el futuro cuando, durante el desarrollo, se quiera aplicar algún tipo de código que sólo funciona en las versiones más recientes, ya que algunos métodos disponibles en estas no están disponibles para las anteriores.

Para solucionarlo se pueden realizar aplicar dos soluciones:
- Programar toda la aplicación usando los métodos proporcionados por el SDK mínimo indicado. Esta solución no provocará fallos en las versiones más actuales, pero no será la óptima para ellas.
- Programar para cada una las diferentes versiones comprobando cuál está instalada en el dispositivo móvil. Es lo más óptimo, pero obliga a escribir grandes cantidades de código.

3.2.2. Firebase [fir 01]

Firebase es un servidor BaaS en la nube que nos permite conectar conectar nuestra aplicación con sus herramientas de manera sencilla e intuitiva gracias a las APIs que nos proporciona para Android.

Las que han tenido alguna utilidad para este proyecto son:
- **Authentication**: Permite el registro de usuarios en el servidor.
- **Realtime Database**:Base de datos NoSQL.
- **Cloud Firestore**: Base de datos NoSQL en fase beta.
- **Cloud Storage**: Servicio de almacenamiento de archivos multimedia y documentos.
- **Cloud Functions**: Permite la programación del backend del servidor.
- **Analytics**: proporciona estadísticas sobre el uso y los usuarios de la aplicación.
- **Crashlytics**: Realiza informes sobre los errores registrados en la aplicación.
- **Performance Monitoring**: Realiza informes sobre el rendimiento de la aplicación.
- **Test Lab**: Ejecuta la aplicación sobre distintos dispositivos móviles para comprobar su rendimiento.

También es recomendable que según el paquete que se seleccione para el desarrollo en el servidor se tendrán más o menos cuotas en el uso de las herramientas. [fir 14]
Estos límites incluyen entre otras:
- El número de peticiones, accesos y búsquedas en las herramientas.
- Memoria utilizada por las bases de datos y el almacenamiento de archivos.
- El número de funciones ejecutadas

Debido al uso de la versión gratuita (Spark) durante el desarrollo del proyecto se han encontrado una serie de limitaciones que han incidido en el uso y configuración de las distintas herramientas.

A continuación, se detallarán cada una de las herramientas del servidor y las desventajas encontradas en ellas.

3.2.2.1. Authentication [fir 02]

Esta funcionalidad nos permitirá registrar distintos usuarios en la plataforma mediante un usuario y una contraseña.

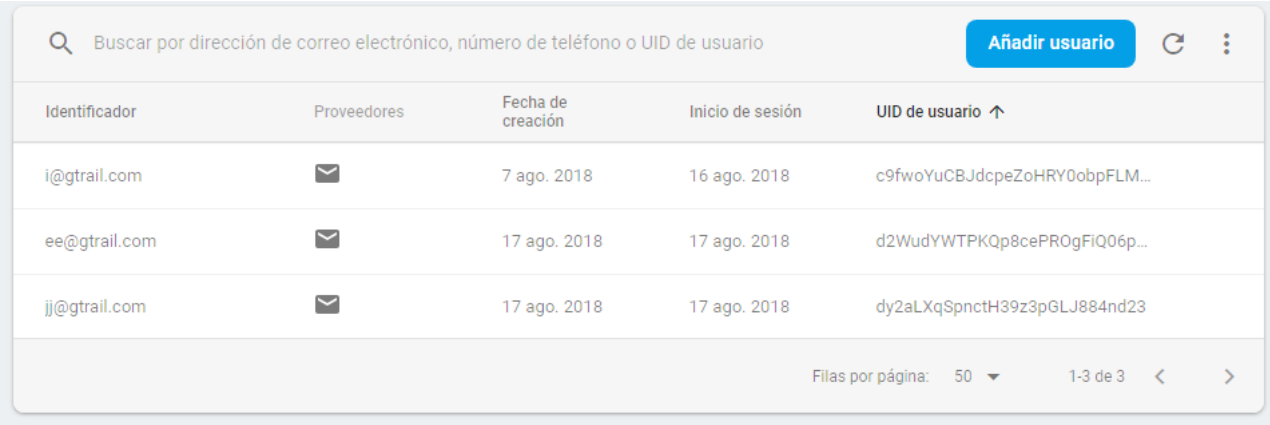


Figura 3.5. Ventana de gestión de usuarios.

El usuario está limitado a ser una cuenta correo, un número de teléfono válido o una cuenta de identidad federada como Google, Facebook, Twitter o GitHub.
Esto certifica la existencia del usuario y su correspondiente asociación con la cuenta.

La contraseña que se debe indicar no conlleva grandes limitaciones, sólo es obligatorio que su longitud sea como mínimo de seis caracteres[1].

Una vez creada la cuenta se le asociará un identificador único que puede ser útil si se quiere que el usuario tenga algún tipo de cuenta dentro de alguna de las bases de datos disponibles.

Junto a la cuenta se crea también una base de datos muy limitada cuyo uso no es obligatorio donde se incluyen el correo, el nombre, el teléfono y una imagen del usuario.

En la figura 3.5 se puede observar cómo se han creado varias cuentas mediante los correos electrónicos y cómo se les ha asignado un identificador.

También nos permite gestionar los correos electrónicos que serán enviados a los usuarios cuando se registren para confirmar la cuenta, decidan cambiar de correo asociado o la contraseña.

3.2.2.1.1. Desventajas

La base de datos interna
La pequeña base de datos que asigna a cada usuario sólo se envía una vez por cada inicio de sesión que se realice. Esto obliga a que las aplicaciones que muestren datos del propio perfil al usuario a que este inicie sesión, escriba los datos que desee, cierre la sesión y vuelva a abrirla para que estos

1 Este es el único dato de las cuentas de usuario al que el administrador del servidor no podrá acceder, pero sí podrá editar o enviar al usuario un mensaje para su alteración.

nuevos datos sean visibles.
Demasiados pasos para una simple actualización del perfil.

3.2.2.2. Realtime Database [fir 03]

Es la primera base de datos NoSQL, cuyos datos se almacenan en formato JSON, habilitada en Firebase.

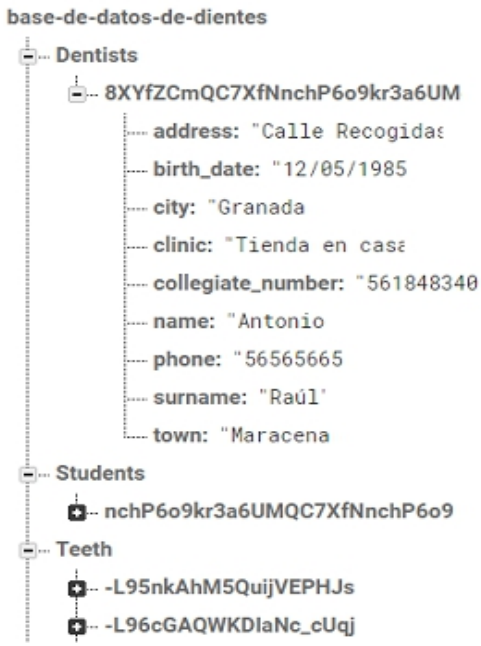


Figura 3.6. Datos almacenados en la herramienta.

Está en su versión final y no se tiene previsto ningún tipo de mejora o aumento de sus funcionalidades debido a la aparición de **Firestore** que representa una versión mejorada de ella.

Permite la creación de reglas para establecer la seguridad de los accesos y de los datos incorporados mediante un documento en JSON.

El único aspecto encontrado en el que mejora a **Firestore** se basa en la posibilidad de comprobar el estado de conexión entre el servidor y la aplicación antes de enviar los datos. Esto es muy útil para evitar ejecutar conexiones asíncronas con el servidor que serán inútiles y consumirán recursos.

3.2.2.2.1. Desventajas

Consultas simples
Su mayor desventaja reside en que el acceso a los datos se encuentra limitado a consultas simples. Aquellas que sean compuestas, como buscar por un tipo dentro de los valores de las variables no se puede realizar.

En un principio esta fue la base de datos utilizada en el proyecto, pero al descubrir este problema se decidió optar por Cloud Firestore.

3.2.2.3. Cloud Firestore [fir 04]

Es una base de datos NoSQL cuyos datos son almacenados en formato JSON.

La distribución de los datos contenidos se basa en colecciones que contienen documentos que a su vez contienen a los datos. Estos datos pueden ser del tipo:

- Number
- String
- Date
- Boolean
- Object
- Array
- Null
- Timestamp
- Geopoint
- Reference

Firestore traduce automáticamente los tipos de cada una de las variables enviadas a su equivalente dentro de la base de datos.

Cada colección y documento tendrá como identificador una cadena en formato UTF-8 que no podrá superar los 1500 bytes ni contendrá el carácter especial "/".

Permite la creación de reglas para establecer la seguridad de los accesos y de los datos incorporados mediante un documento en JSON.

Las consultas de los elementos de la base de datos se pueden realizar mediante distintos métodos, dependen del estilo que se desee utilizar para la recepción de ellos. Principalmente, los métodos de solicitud de datos varían en función de si se desea un elemento específico en formato *Map* (DocumentSnapshot) u *Object* (QuerySnapshot) o si se pretende recuperar varios de elementos a la vez. En el caso de este último se devolverá el conjunto de documentos en un objeto *Query.* [fir 12]

También permitirá realizar consultas compuestas sobre la base de datos, lo que permite realizar búsquedas sobre los datos contenidos dentro de los documentos y obtener el conjunto de elementos resultante.

Para el envío de datos (creación o actualización) existen dos alternativas: enviar los datos en formato *Map* o en un *Object* diseñado por el programador. Las dos son igual de eficientes y están implementadas para no producir fallos en Firestore.

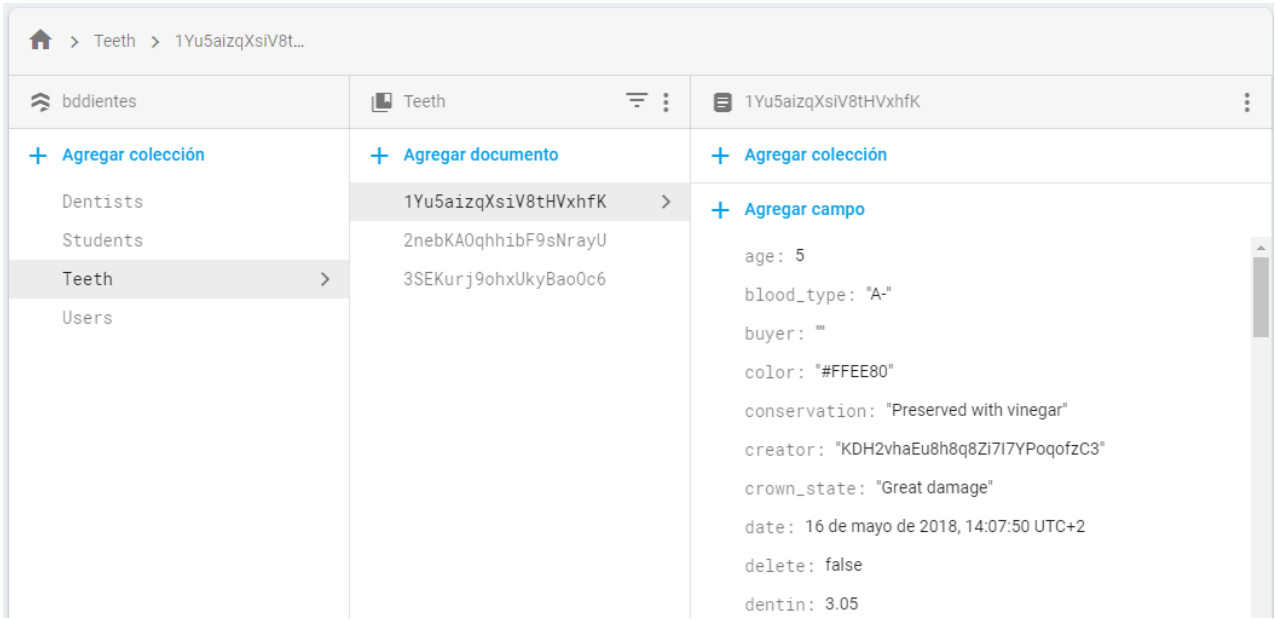


Figura 3.7. Pantalla para la gestión de los datos almacenados.

Durante la creación de nuevos documentos se generará un identificador único para la referencia a esa creación. Será decisión del programador si desea usar esa cadena de texto, ya que su existencia sólo estará disponible durante la ejecución del código, una vez que se elimine la referencia a esa creación el identificador desaparecerá.

3.2.2.3.1. Desventajas

El formato de los objetos
Al obtener un objeto desde el servidor si este está en formato Map, para acceder de manera individual a cada uno de los elementos contenidos se deberá realizar un casting de cada uno de ellos. Si durante ese casting su usa un tipo equivocado con respecto al tipo con el que se subió al servidor se podría producir una conversión errónea que, en algunas situaciones, no se notifica. Esto puede provocar que se produzcan situaciones extrañas y problemáticas al tratar las nuevas variables. Un ejemplo podría producirse cuando al descargar un elemento del servidor con variables de tipo *Number* se asignan a variables de tipo entero sin tener presente que los datos descargados fueron almacenados desde variables con formato *Double*.

Consultas compuestas
Las consultas compuestas para elementos que sean iguales a uno determinado no entrañan complejidad. El problema aparece cuando se tratan de elementos mayores o menores, ya que Firebase obliga a indexar cada una de las combinaciones de búsqueda posible.

Para su funcionamiento es necesario que se especifique cada una de las posibles combinaciones tanto en la herramienta, para que se realice una indexación de los elementos indicados, como en el código de la aplicación concatenando cada uno de ellos en diferentes líneas de código.

Esto supone una gran complejidad para el programador debido a que si tiene n elementos (siempre

que sean superiores a uno) deberá configurar cada una de las posibilidades (∑(n^3 – 1)) tanto en el servidor como en la aplicación.

Actualizaciones sin búsquedas
Firestore no permite realizar actualizaciones de datos mediante búsquedas en sus documentos.
Para ello es necesario realizar una búsqueda con los parámetros que interesen y obtener los resultados y lanzar una petición al servidor para realizar la actualización sobre ellos.
Esto puede provocar errores en el contenido de los datos cuando entre la obtención de los resultados de una búsqueda y su posterior actualización sus datos en el servidor han cambiado siendo diferentes a los indicados al buscarla.

Superar el límite de objetos
Si por algún motivo se alcanza el límite de objetos que puede contener un documento (en el momento del desarrollo de este trabajo el límite estaba en 2000 elementos y las subcolecciones en 100) dejará de estar disponible y no se podrá almacenar más en él. Esto es un problema si se necesita de un conjunto que almacene grandes cantidades de elementos, ya que en poco tiempo un se podría llenar impidiendo el correcto funcionamiento de la aplicación.
Es posible que sea necesario pensar en alguna forma de ir cambiando de documento una vez se alcance su límite.

3.2.2.4. Cloud Storage [fir 05]

Es un servicio de almacenamiento de los archivos multimedia y documentos que desee el usuario sobre los depósitos de la plataforma de **Cloud Storage**.

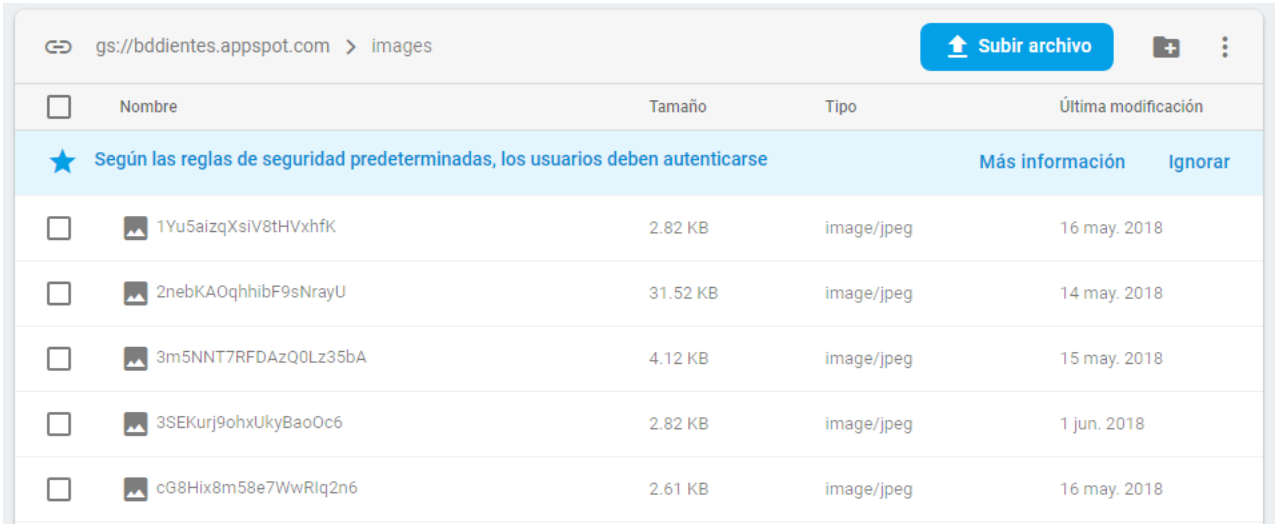


Figura 3.8. Contenido de la carpeta *images*.

Su estructura de almacenamiento es semejante al de las estructuras jerárquicas de directorios de cualquier sistema operativo: existe un directorio raíz al que se pueden añadir archivos o nuevos directorios. En la figura 3.8 se puede observar cómo en el directorio raíz existe una carpeta imágenes que alberga distintas imágenes.
También permite la creación de reglas para establecer la seguridad de los accesos y de los archivos

mediante un documento en JSON.

Las descargas de estos elementos podrán ser gestionados mediante:

- Almacenamiento en la memoria interna. Para ello el archivo se debe almacenar en un array de bytes o en un objeto *URI* que posea el suficiente tamaño.
- Descargar en un archivo local. Habrá que crear un archivo temporal con suficiente espacio para copiar en él el archivo.
- Descarga en caché (sólo imágenes). Es el recurso recomendado por los desarrolladores de Google y que consiste en la gestión de las imágenes mediante el uso de la biblioteca Glide. Permite almacenar en caché datos de manera prolongada mientras se esté dentro de la aplicación o de manera temporal (el almacenamiento de las imágenes estará asociado al ciclo de vida de la clase usada).

3.2.2.4.1. Desventajas

Limitaciones de la base de datos
Los límites de las base de datos de Firebase (1GiB en la versión gratuita) obliga a tener un control exhaustivo sobre el tamaño de los archivos que se suban a la herramienta con el fin de evitar que se supere el límite disponible y se impida el acceso a ella.

Sobreescritura
Cuando se sube un archivo cuyo nombre y dirección ya está ocupada por otro archivo este será sustituido por el nuevo de manera automática y sin aviso alguno. Esto puede suponer un problema si se almacena algún tipo de archivo sensible y se permite almacenar en la misma dirección sin control alguno sobre los nombres que se asignen a los nuevos archivos.

3.2.2.5. Cloud Functions [fir 06]

Es un servicio que permite el programar el backend del servidor para gestionar su comportamiento ante las acciones producidas sobre herramientas de gestión y almacenamiento de datos o archivos.

A diferencia de las demás herramientas de Firebase esta necesita que el programador disponga del entorno de ejecución de node.js sobre el equipo donde se vaya a programar para la instalación de la herramienta y los archivos asociados a nuestra cuenta.

Una vez instalada, se descargarán los ficheros asociados a nuestra cuenta de servidor y será sobre el fichero index.js donde se escribirá el código necesario para la gestión requerida.

Esto permite lanzar eventos no visibles al usuario de la aplicación móvil y reducir la cantidad de código ejecutable de la aplicación.
Un ejemplo podría ser cuando un usuario de una aplicación realiza una petición al servidor para la creación de un nuevo objeto, el código contenido en **Cloud Functions** también realice una actualización del número de objetos creados por ese usuario. Si esto se realizase desde el código del dispositivo móvil se deberían realizar dos peticiones: una para la creación y otra para la actualización.

Una parte importante de esta herramienta es la capacidad de crear y aceptar peticiones **HTTPS.** Gracias a que pueden ser llamadas directamente no es necesario realizar ninguna operación sobre las herramientas para ejecutarlo lo que permite realizar todas las operaciones dentro de su código interno.

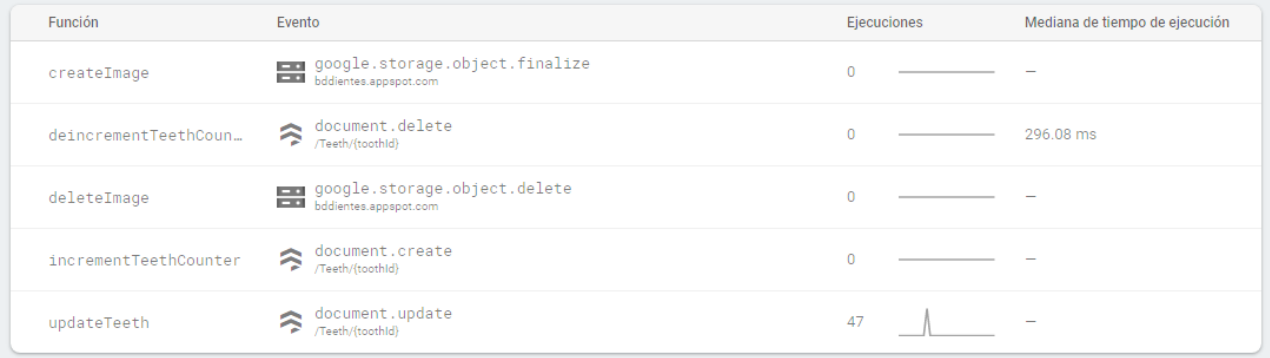


Figura 3.9. Información sobre las funciones implementadas.

Dentro del servidor se podrá obtener información sobre el número de veces que han sido ejecutadas (figura 3.9) , el tiempo que han tardado en llevar a cabo su función y, si se han producido fallos, los detalles sobre ellos.

3.2.2.5.1. Desventajas

Instalación
A diferencia de las otras herramientas, esta obliga al administrador a descargar los numerosos archivos de configuración de la herramienta en el ordenador desde que el se decida gestionarlos.

Entorno Node.js
A pesar de que en las demás elementos de **Firebase** apenas se requiere tener conocimiento de programación en esta es obligatorio tener conocimientos de **node.js** para gestionar los métodos asociados a las distintas herramientas y evitar la ejecución de bucles dentro del servidor (lo que podría consumir los recursos disponibles por día). También se obliga a entender la forma de gestión que proporciona Google mediante los ejemplos proporcionados en la plataforma o en **Github**.

Bucles
Hay que tener presente que cuando se crea algún tipo función que ejecuta la actualización de datos sobre una herramienta tras recibir una actualización procedente de ella se entrará en un bucle infinito. Será entonces necesario crear algún tipo de método que revise si se está repitiendo la función y salir de ella.

Direcciones dentro de las funciones
Se ha observado que en algunas ocasiones una función puede no llegar a leer correctamente una dirección de la base de datos indicada.
Aunque este error sólo se ha producido en un par de ocasiones ha obligado a revisar toda la base de datos en busca de los datos no actualizados.

Instancias
Aunque se asegura que en su versión final se lanzarán todas las instancias necesarias para dar servicio a cada una de las peticiones en estos momentos (versión beta) no se cumple. Esto dificulta el correcto funcionamiento del servidor obligando al administrador a controlarlo constantemente hasta que su versión final sea lanzada.

Limitaciones de la herramienta
Las limitaciones asignadas a la cuenta tanto de instancias como de CPU hay que valorarlas ya que cuando se llegan al límite marcado dejarán de funcionar todas las funciones creadas. En algunos casos, si sólo es de instancias por segundo, se impedirá la ejecución hasta pasado unos 100 segundos y entonces se volverán a ejecutar, pero si es por limitaciones de CPU la reanudación se realizará cuando empiece un nuevo día.
Estas limitaciones dependerán del tipo de cuenta que se posea, pero se ha observado que las reanudaciones de las funciones no funcionan correctamente porque algunas ejecuciones no llegan a realizarse completamente.

Actualizaciones
La herramienta puede caerse o actualizarse sin previo aviso lo que puede producir grandes problemas si gran parte del uso de las herramientas dependen del buen estado de esta.
Esto obliga al administrador a realizar revisiones periódicas para certificar su correcto funcionamiento[1].

Bloqueo de funciones
La configuración del tiempo, memoria y CPU máxima permitida o el bloqueo de funciones (muy útil cuando una función entra en un bucle infinito) está dentro de una pantalla de configuración de aplicaciones ajena a la de administración de Firebase. Para acceder a esta pantalla hay que entrar desde **Google Cloud Platform** a la dirección: https://console.cloud.google.com/ y seleccionar el proyecto creado.

3.2.2.6. Analytics [fir 07]

Herramienta que permite centralizar la información general sobre los eventos producidos por los usuarios dentro de la aplicación. Los datos estarán distribuidos en distintas pestañas donde se nos mostrarán en gráficas para facilitar su entendimiento.

Las gráficas nos mostrará toda la información sobre la aplicación que nos puede resultar relevante acerca de los usuarios (figura 3.10), diferentes accesos, tiempo de uso, las partes más usadas de la aplicación o aquellas que más recursos consumen.
También permite crear paquetes de información personalizados para enviar al servidor en el caso de que los análisis por defecto resulten insuficientes.

Normalmente todos estos datos serán enviados al servidor aproximadamente cada hora, aunque el primer envío se realiza a los pocos segundos desde el inicio de la aplicación.

1 Durante el mes de julio del año 2018 se produjeron varias actualizaciones que obligaron a reeditar todo el código implementado en las funciones.

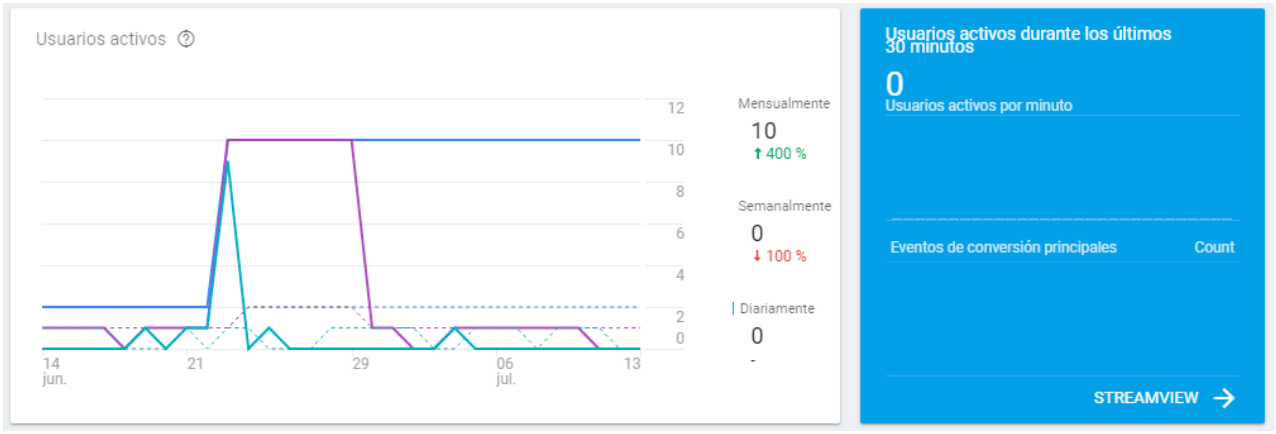


Figura 3.10. Ventana de usuarios activos.

3.2.2.6.1. Desventajas

Exportar datos
Existe un botón que permite recoger en formato CSV los datos de la herramienta, el problema reside en que esos datos no muestran información relevante, se limitan a detallar si la recogida de datos de ciertas instancias está habilitada y el número de accesos a ellas omitiendo aquellos datos que permiten el análisis de parámetros complejos que se muestran en las gráficas de la herramienta.

Esto obliga a que para obtener tener un análisis completo y complejo de los datos de nuestra aplicación sea necesario acceder vía web a la herramientas de la plataforma

Documentación
En ocasiones los parámetros que muestra Firebase pueden llegar a ser difíciles de entender ya que almacena la información sobre ciertos aspectos bajo nombres cuyo conocimiento es nulo. Tampoco ayuda que la documentación no aporte gran información sobre estos datos, obligando al administrador a aprender de ellos bajo el uso de la experiencia.

Lentitud en la recepción de datos
Cuando el usuario decide utilizar un paquete de datos diseñado por él mismo el envío de estos al servidor puede realizarse tras periodos de tiempo considerablemente largos.

```
Bundle bundle = new Bundle();
bundle.putString(FirebaseAnalytics.Param.CONTENT_TYPE,
"DentistData_fragment");
bundle.putString(FirebaseAnalytics.Param.ITEM_ID, "set Values");
mFirebaseAnalytics.logEvent(FirebaseAnalytics.Event.SELECT_CONTENT,
bundle);
mFirebaseAnalytics.setUserProperty("age", userData.getBirth());
```

Código 3.6. Código para enviar nuevos datos a Analytics.

```
06-21 17:05:36.649 2160-10241/? V/FA-SVC: Upload scheduled in approximately ms: 1551452
06-21 17:05:36.679 2160-10241/? V/FA-SVC: Scheduling upload with GcmTaskService
    Scheduling task with Gcm. time: 1551452
```

Figura 3.11. Tiempo necesario para el envío de datos.

Dichas esperas pueden llegar a ser bastantes molestas si se desea obtener los datos en tiempos moderadamente aceptables.
Este problema se puede observar en la figura 3.11 donde el tiempo para el envío al servidor de los datos del código 3.6 puede llegar a tardar unos 1551452 milisegundos (25 minutos aproximadamente).

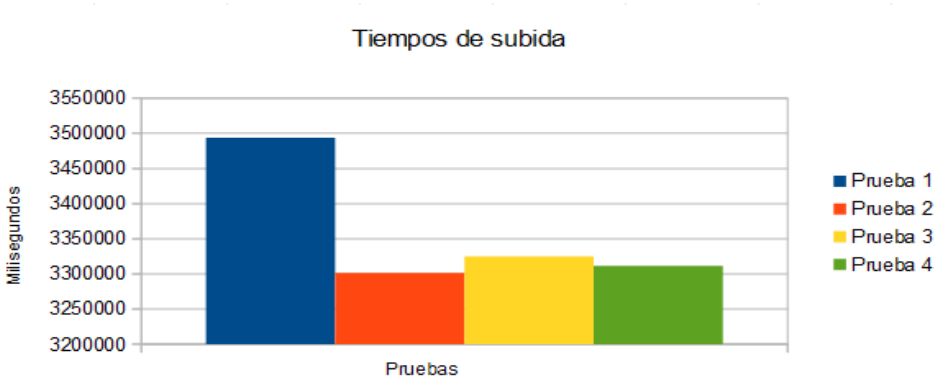


Figura 3.12. Tiempos obtenidos para el envío.

Para mostrar con más detalle este problema, en la figura 3.12 se muestran los resultados de diferentes pruebas realizadas con la herramienta donde se observa que el tiempo para el envío de datos suele oscilar entre los 3300000 y los 3500000 milisegundos.

3.2.2.7. Crashlytics [fir 08]

Herramienta que recoge las excepciones producidas en la aplicación y nos las muestra en el servidor. Es la versión actual del antiguo **Crash Reporting** que dejó de estar disponible a principios de 2018. [fir 09]

Mediante una serie de pestañas, la herramienta nos proporciona información en tiempo real sobre las excepciones reportadas mediante estadísticas y gráficas como se muestra en la figura 3.14.

También nos indicará datos relevantes como son la actividad o fragmento donde se produjo, la fecha y la línea de código donde se produjo el error y nos proporcionará información relativa a ellos para evitarlos en un futuro y mantener la estabilidad de la aplicación.

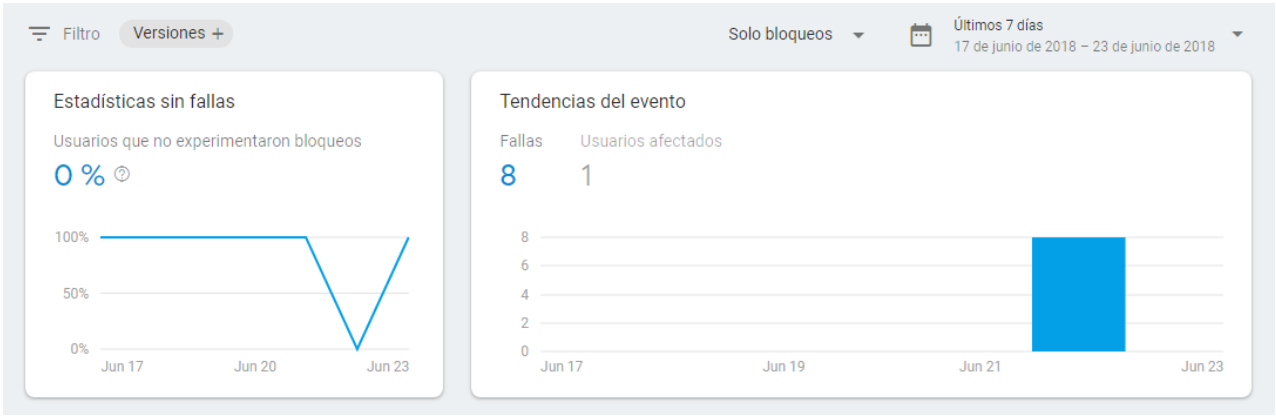


Figura 3.13. Ventana de estadísticas de errores.

Si se integra con la herramienta **Analytics** se puede conseguir un mayor nivel de detalle sobre las excepciones. Lo que nos ayuda a identificar a los tipos de usuarios y sus acciones antes del error.

3.2.2.7.1. Desventajas

Limitaciones de la información recibida
Aunque no es necesario, en ocasiones la información obtenida puede resultar limitada o no nos proporciona aspectos interesantes acerca del error.
Para solucionarlo se deberán personalizar los informes de fallos mediante la creación y el envío de mensajes cuando se produzca el error.
Los mensajes se crearán gracias a un objeto cuyos métodos permitirán asignar a unas claves determinadas los valores que sean de interés.

3.2.2.8. Performance Monitoring [fir 10]

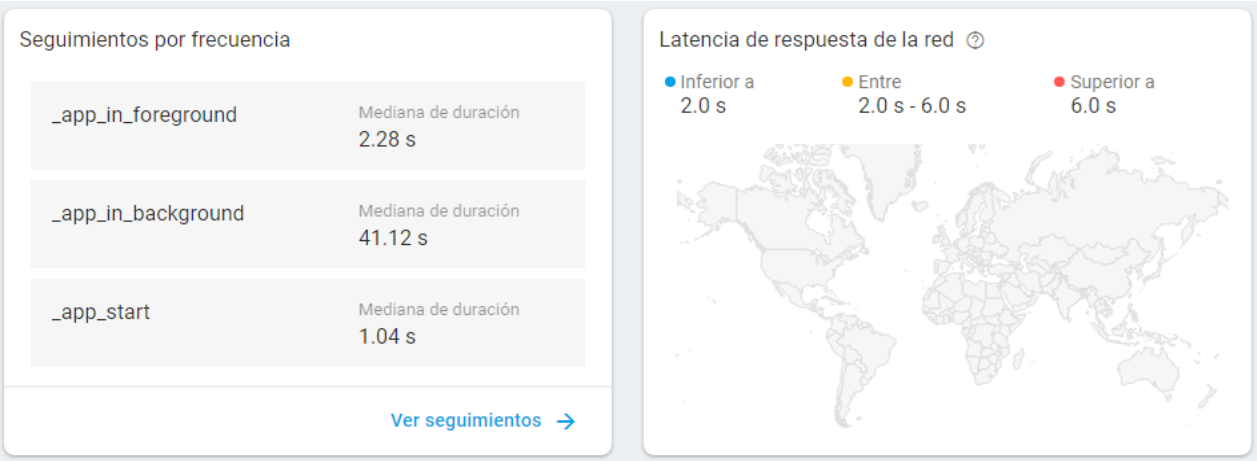


Figura 3.14. Ventana para el análisis de tiempos.

Notifica aquellas latencias producidas en la aplicación tanto en el código interno como en las

conexiones de red que realice.

Permite tener constancia del código ejecutado, la duración y la procedencia del usuario que ha experimentado las latencias.

3.2.2.8.1. Desventajas

Lentitud en la actualización de notificaciones
Las esperas para poder visualizar las actualizaciones de los informes suele ser larga. En algunos casos se ha tenido que esperar más de 12 horas para recibir dichos informes del día lo que hace difícil realizar un seguimiento en tiempo real.

3.2.2.9. Test Lab [fir 11]

Es una herramienta que nos permite ejecutar nuestra aplicación sobre una amplia gama de dispositivos móviles de distintos fabricantes, tanto virtuales como físicos, dentro de la plataforma y que apareció durante el 2018.

Las opciones que nos permiten van desde la selección del dispositivo móvil deseado, el tipo de orientación, comprobar accesos privados y ejecución de vínculos URL.

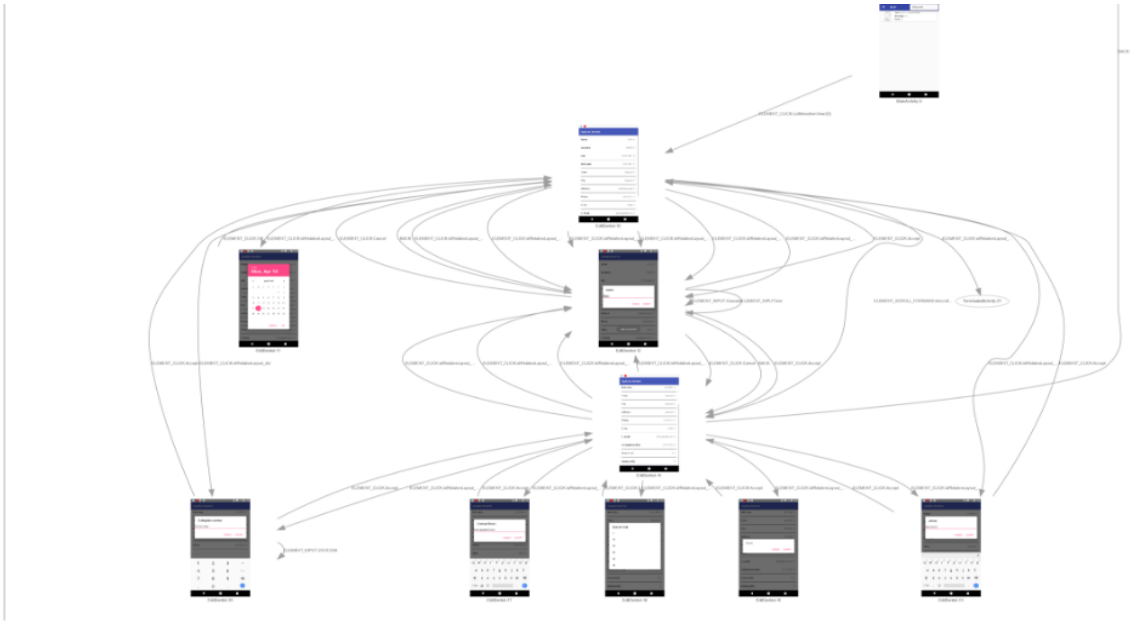


Figura 3.15. Ventana de seguimiento de acciones.

Las pruebas se basan en comprobaciones reiterativas de los distintos elementos dentro de la aplicación (en la figura 3.15 se puede observar) para establecer si son capaces de responder correctamente ante cualquier ejecución o, en el caso de cajas de texto, si pueden almacenar valores según los rangos indicados.

Figura 3.16. Ventana de errores.

Una vez terminada la prueba definida se mostrarán una serie de pestañas donde entre ellas se podrá observar el procedimiento seguido dentro de la aplicación, el uso que hace de los recursos físicos y un informe detallado en el caso de que se haya producido un error (figura 3.16).

Para ejecutar esta herramienta sólo será necesario subir al servidor el archivo APK de la aplicación e indicar el tipo de prueba a realizar.

3.2.2.9.1. Desventajas

Limitaciones en el número de pruebas
Puede ser interesante realizar numerosas pruebas sobre una aplicación un día determinado.
El problema radica en que tanto para los planes **Spark** (gratuito) como **Flame** las ejecuciones de pruebas están limitadas a 10 dispositivos virtuales y 5 físicos por día y tampoco se permite ejecutar más de 4 pruebas a la vez.
Esto impide que el desarrollador realice todas las pruebas necesarias para su aplicación en un sólo día.

3.2.2.10. Otras herramientas

Existen otras funcionalidades de Firebase con las que no se han llegado a trabajar debido a que no eran necesarias para la correcta realización del trabajo. Aquí se comentan brevemente:

- **Predicitions:** mediante anális permite predecir el comportamiento futuro de un grupo de usuarios. [fir 13]
- **AdMob**: Muestra anuncios de millones de anunciantes de Google
- **AdWords**: Sirve para aumentar las instalaciones de las aplicaciones, obtener la información sobre esos usuarios y realizar eventos publicitarios orientados.
- **BigQuery**: Analiza los datos de la aplicación cuando se producen ciertos eventos. Su uso requiere de la contratación de una cuenta de pago.

- **DoubleClick** Digital Marketing: Analiza el impacto de las campañas de marketing al ver y pulsar sobre publicidad dentro de la aplicación. Requiere de vinculación con una cuenta DoubleClick.
- **Google Play**: realiza un seguimiento de las descargas y compras de las aplicaciones creadas y sobre fallos producidos en ellas. Requiere vincularlo a una cuenta a Google Play.
- **Slack**: Envía al equipo del administrador alertas importantes que se produzcan en las herramientas de Firebase.

3.2.2.11. Las copias de seguridad

Firebase proporciona la capacidad de realizar copias de seguridad de las herramientas utilizadas de manera opaca al usuario. Estas sirven para proteger el trabajo del administrador de errores internos relacionados con los servidores de Google.

Pese a esto, el desarrollador no dispone de ningún instrumento que permita realizar copias de seguridad de ninguna de las herramientas que proporciona.

Esto hecho obliga a que se tengan que definir reglas robustas para sus bases de datos y almacenamiento de archivos y asegurar la integridad de los datos almacenados. También es recomendable desarrollar métodos en **Cloud Functions** que aseguren dichos datos aplicando análisis más exhaustivos sobre esos datos.

Como alternativa, las comunidades de desarrolladores, mediante la herramienta *node.js client* que permite conectar con las herramientas del servidor, han creado diferentes aplicaciones que permiten realizar copias de seguridad los datos contenidos en Firebase y restaurarlos si se precisa[1].

3.2.3. La seguridad

3.2.3.1. Conexión cliente-servidor [fir 15]

Toda la información que se envíe a través de Internet desde una aplicación móvil hacia la plataforma Firebase o viceversa se transmite mediante el protocolo seguro de transferencia de hipertexto (**HTTPS**). Este cifrado representa una mejora en el el protocolo **HTTP** que permite el envío de información en hipertexto de manera segura gracias a los certificados de seguridad SSL/TTL.

3.2.3.2. Las bases de datos

Todo los contenidos que se almacenen en el servidor están cifrados. **Firebase** se encarga de cifrar dicha información una vez se recibe, antes de almacenarla, mediante el protocolo AES de 256 bits

1 Ejemplo de herramienta para Firestore: https://www.npmjs.com/package/firestore-backup

cuya clave de cifrado de manera regular es cifrada mediante un conjunto de claves maestras para mayor seguridad.
Aunque se puede modificar para dar acceso a todo el mundo, este tipo de cifrado permite que en un principio sólo el administrador de la plataforma y aquellos usuarios que posean credenciales de acceso en la herramienta **Authentication** puedan acceder a los datos descifrados.

Las contraseñas también se cifran con otra clave con el propósito de que nadie pueda acceder a las cuentas de los usuarios. Sin embargo, el administrador de la plataforma puede acceder a la posibilidad de alterar las contraseñas de los usuario desde la propia plataforma web.

Por su parte, Google asegura que los datos de sus servidores son seguros debido al uso de un cortafuegos con numerosas reglas para cada petición (que el usuario puede configurar), una supervisión exhaustiva de los contenidos y a las supervisiones de auditorías externas que comprueban que cumplen con los estándares de seguridad necesarios. [fir 16]

3.2.3.3. Ciclo de vida de las conexiones

Por último, las peticiones que se realicen desde la aplicación hacia el servidor pueden ser permanentes si no se sale de la clase que las ejecuta o no se cierra la aplicación (lo que elimina toda clase de conexión con el servidor). Para evitar esto es necesario indicar que sus ciclos de vida estarán relacionados con el de la actividad o fragmento asociado.

3.2.3.4. Protección de datos

Como todo proyecto de estas características se ha de velar por la seguridad de los datos almacenados, el uso fraudulento que se pueda dar de ellos, la correcta comprensión del usuario final acerca de su uso y tratamiento o la propiedad intelectual de los recursos utilizados.

Es por ello que **Firebase** cumple con los requisitos específicos como *procesador de datos* estipulados en el Reglamento general de protección de datos (GDPR) de la Unión Europea. [fir 17]

La gestión del administrador como *controlador de datos* tanto en el servidor como en la aplicación móvil también se compromete con el cumplimiento de los requisitos de GDRP.

Capítulo 4. Diseño e implementación

Aclarados las herramientas a utilizar y los requisitos de desarrollo, ahora nos centraremos en el diseño utilizado para realizar la aplicación.

4.1. El servidor

Para el servidor, el uso de Firebase estuvo claro desde casi el comienzo del desarrollo. A penas cuando se llevaba una semana analizando los distintos servidores se decidió por su uso gracias a su gran versatilidad. Esto queda patente en los diagramas donde se deja claro desde un principio que el servidor será ese y no otro.

Su desarrollo no ha sido complejo, pero sí dificultoso debido a las carencias que muestran las documentaciones oficiales obligando a realizar numerosas pruebas para certificar la forma en la que gestiona los datos de cada herramienta.

Las herramientas que se usaron al comienzo fueron **Authentication**, **Realtime Database** y **Storage**, pero debido a sus carencias en cuanto a las consultas compuestas, mencionado en su apartado de desventajas, la segunda fue sustituida por **Firestore**.

Después de ese primer desarrollo, se continuó con herramientas como **Cloud Functions**, cuyo implementación fue complicada debido al desconocimiento del lenguaje Javascript del que hace uso.

Finalmente, se aplicaron las herramientas de análisis (**Analytics**, **Crashlytics** y **Performance Monitoring**) cuyo funcionamiento sólo implicaban un pequeño conocimiento sobre su funcionamiento y la instalación de ellas.

Debido a que antes incluso de llegar a programar o estudiar en profundidad otra alternativa se decidió por el uso de este servidor, el estudio y la forma gestionar datos se definió bajo su comportamiento lo que queda patente en los diagramas actividad usados en este capítulo.

4.1.1. Authentication

Se ha configurado para permitir sólo la creación de cuentas mediante correos electrónicos.
También se han establecido una serie de mensajes para su envío automático cuando se cree una nueva cuenta y para notificar cambios en las contraseñas o en los correos electrónicos.
En el caso de que no se acepte el enlace contenido en los mensajes de confirmación no se podrá acceder a la aplicación.

Por simplicidad y debido a que cada vez que se inicia sesión en esta herramienta se envían de manera automática los datos personales asociados al usuario, se ha decidido utilizar la variable correspondiente al *nombre* para añadir una cadena de texto que sirva como identificador para

determinar si la cuenta es de tipo dentista o estudiante.

4.1.2. Storage y Firestore

Para poder relacionar los datos de **Firestore** con las cuentas creadas en **Authentication**, como si de claves primarias de una base de datos SQL se tratase, se usan los identificadores que se asignan a los usuarios creados en esta última como identificadores en los documentos que almacenan sus perfiles en **Firestore**.

Para relacionar cada una de las imágenes con su pieza dental correspondiente se usa el identificador asignado para la pieza en **Firestore** como nombre del archivo.
Esto permite recuperar los datos de una pieza aleatoria y poder buscar su imagen asociada.

Para **Firestore** se ha diseñado un estructura que consta de las siguientes colecciones donde se irán añadiendo los documentos relativos a ellos:

- Teeth: para las piezas dentales.
- Dentists: para los perfiles de los dentistas.
- Students: para los perfiles de los estudiantes.

Los elementos enviados y descargados de cada una de estas secciones se hará mediante objetos de las clases *ToothData*, *DentistUserData* y *StudentUserData* respectivamente.

4.1.3. Cloud Functions

Para gestionar los datos recibidos se han programado los siguientes métodos:

- Incrementar contador. Aumenta el contador del dentista cuando crea una pieza nueva o el del estudiante si este solicita una pieza. Se ejecuta tras una operación de creación en Firebase.
- Disminuir contador. Disminuye el contador del dentista al eliminarse alguna pieza dental creada por él. dentales creadas del dentista. Se ejecuta tras una operación de actualización en Firebase.
- Crear imagen. Si una imagen es creada se localiza la pieza dental asignada y se actualiza su variable *image* a *true*. Se ejecuta tras una operación de creación en **Storage**.
- Borrar imagen. Si una imagen es eliminada se localiza la pieza dental asignada y se actualiza su variable *image* a *false*. Se ejecuta tras una operación de eliminación en **Storage**.
- Actualizar pieza dental. Actualizará las fechas de las piezas dentales creadas y de aquellas que sufran una actualización a la hora del servidor. Se ejecuta tras una operación de actualización en Firebase.
- Borrar usuario. Una vez un usuario registrado decida eliminar su cuenta este método actualizará los datos del perfil relacionado (**Firestore**) cambiando el valor de la variable *delete* a *true* y creando una nueva variable *delete_date* con la fecha en la que se produjo la baja. Se ejecuta tras una operación de eliminación en Authentication.

4.1.4. Analytics, Crashlytics y Performance Monitoring

Estas herramientas no necesitan nada más que realizar sus correspondientes instalaciones en la aplicación y , si se requiere, configurarlas para que recojan y envíen los datos extras que se necesiten.

Sólo en el caso de Analytics y Crashlytics se han añadido algunas configuraciones extra para incluir más detalles sobre los fragmentos más usados y aquellas consultas a Firebase que devuelvan datos erróneos.

4.1.5. Reglas

El servidor Firebase nos permite gestionar un gran número de herramientas, pero sólo en aquellas en las que se almacena algún tipo de información indicará que es útil especificar una serie de reglas para evitar la escritura de datos erróneos.

Para la herramienta **Storage** se han definido dos reglas:

- Una que impida que cualquier usuario no registrado pueda acceder a los datos.
- Otra que impida incluir archivos cuyo tamaño supere 1 MiB.

Para la herramienta **Firestore** se han definido la siguiente regla:

- Una que impida que cualquier usuario no registrado pueda acceder a los datos

4.1.6. Las peticiones al servidor

Para que los usuarios de la aplicación puedan usar las distintas herramientas del servidor antes hay que conseguir un token que permita al servidor descifrar la clave privada de cifrado usada para los elementos contenidos de Firebase. Para ello es necesario estar iniciar sesión en la cuenta asociada a la herramienta Authentication y que el servidor nos envíe nuestros datos personales junto con el token (Figura 4.1).

Una vez obtenido el token, las peticiones al servidor se realizarán de manera muy sencilla:

1. Se instalará la API de la herramienta a utilizar.
2. Se realizarán las solicitudes a la herramienta mediante un mensaje asíncrono.
   Dependiendo de la herramienta usada, el mensaje puede contener tanto la dirección de la ubicación de los datos, así como los valores a introducir si se quiere realizar alguna escritura.
3. Firebase devolverá un mensaje de confirmación de acción y, si se ha solicitado una recuperación de datos, el objeto u objetos con los datos encontrados. En el caso de que se haya producido algún problema en el servidor devolverá un objeto con el error.

Un ejemplo con el mecanismo básico se puede ver en la figura 4.2.

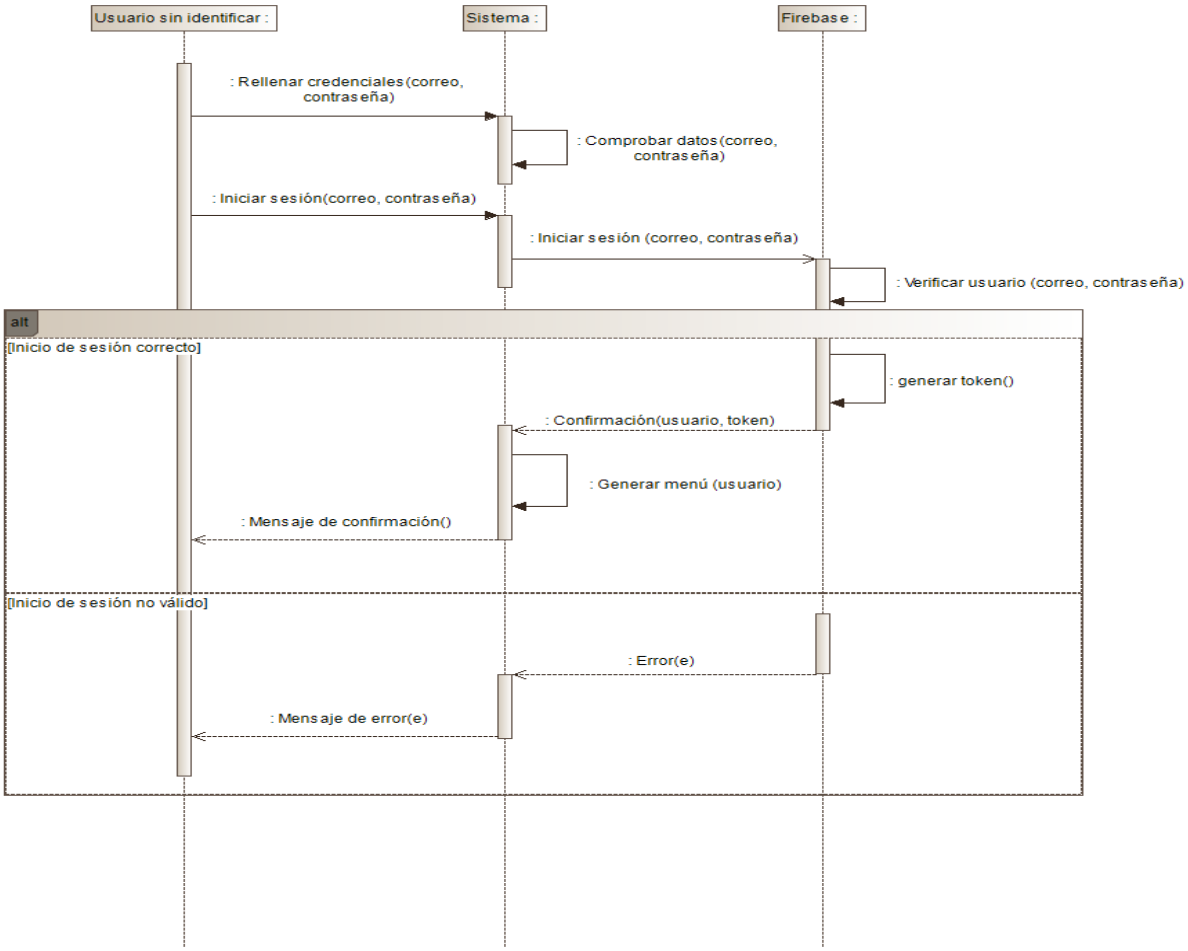


Figura 4.1. Diagrama de inicio de sesión.

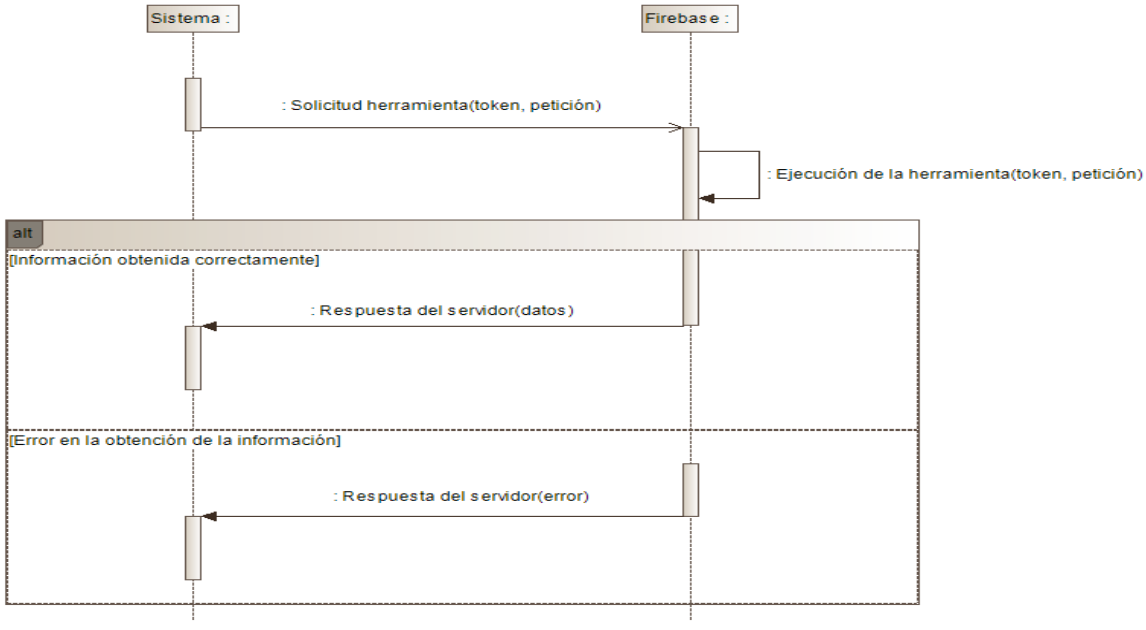


Figura 4.2. Diagrama de solicitud a una herramienta de Firebase.

4.2. La aplicación Android

Para conseguir alcanzar los objetivos marcados ha sido necesario aprender durante varios meses los mecanismos básicos de la programación en Android. Tras este periodo que duró varios meses se comenzó a desarrollar la aplicación final.

En un primer momento se creó una aplicación donde se usaban únicamente actividades para gestionar y diálogos.
A medida que se avanzaba el desarrollo se comenzó a entender el uso de los fragmentos lo que obligó a rediseñar toda la estructura diseñada hasta entonces.

Esta última versión comienza con una actividad de inicio de sesión desde la cual se generarán las actividades necesarias para mostrar cada uno de los apartados especificados en los casos de uso.[1]

4.2.1. Pantalla de inicio

Se trata del primer apartado del que se debe encargar la aplicación.
Busca simplificar el acceso al usuario así como evitar que nadie que no esté registrado en la aplicación pueda acceder a las datos contenidos en las herramientas utilizadas.

Las tres funciones básicas que debe contener son las siguientes:

- Iniciar sesión.
- Registrar a los usuarios dependiendo del tipo de usuario que deseen ser.
- Permitir el cambio en sus cuentas de la contraseña y el correo electrónico.

El propósito es que en la misma ventana al usuario pueda acceder a un apartado donde pueda introducir sus credenciales y acceder a su sesión o, en caso de no poseer una cuenta, pulsar una opción donde se le permita registrarse.

También se ha incluido la posibilidad de alterar la contraseña y el correo electrónico relacionados con la cuenta creada desde este apartado. Esto se debe a que cuando se editan estos valores en una sesión activa los cambios no se aplican hasta que esta se cierra y para evitar cerrar la sesión activa de un usuario estando dentro de su menú correspondiente de manera agresiva se ha preferido crear este pequeño apartado.

Para desarrollar estas características se ha optado por implementar una actividad de inicio de sesión, diseño por defecto, donde se pueda escribir los credenciales correspondientes y que tenga un enlace a otra actividad que muestre el formulario de registro.

1 Todos los diagramas utilizados para la aplicación (clase, actividad y paquetes) se encuentra en el Anexo.

4.2.1.1. Inicio de sesión

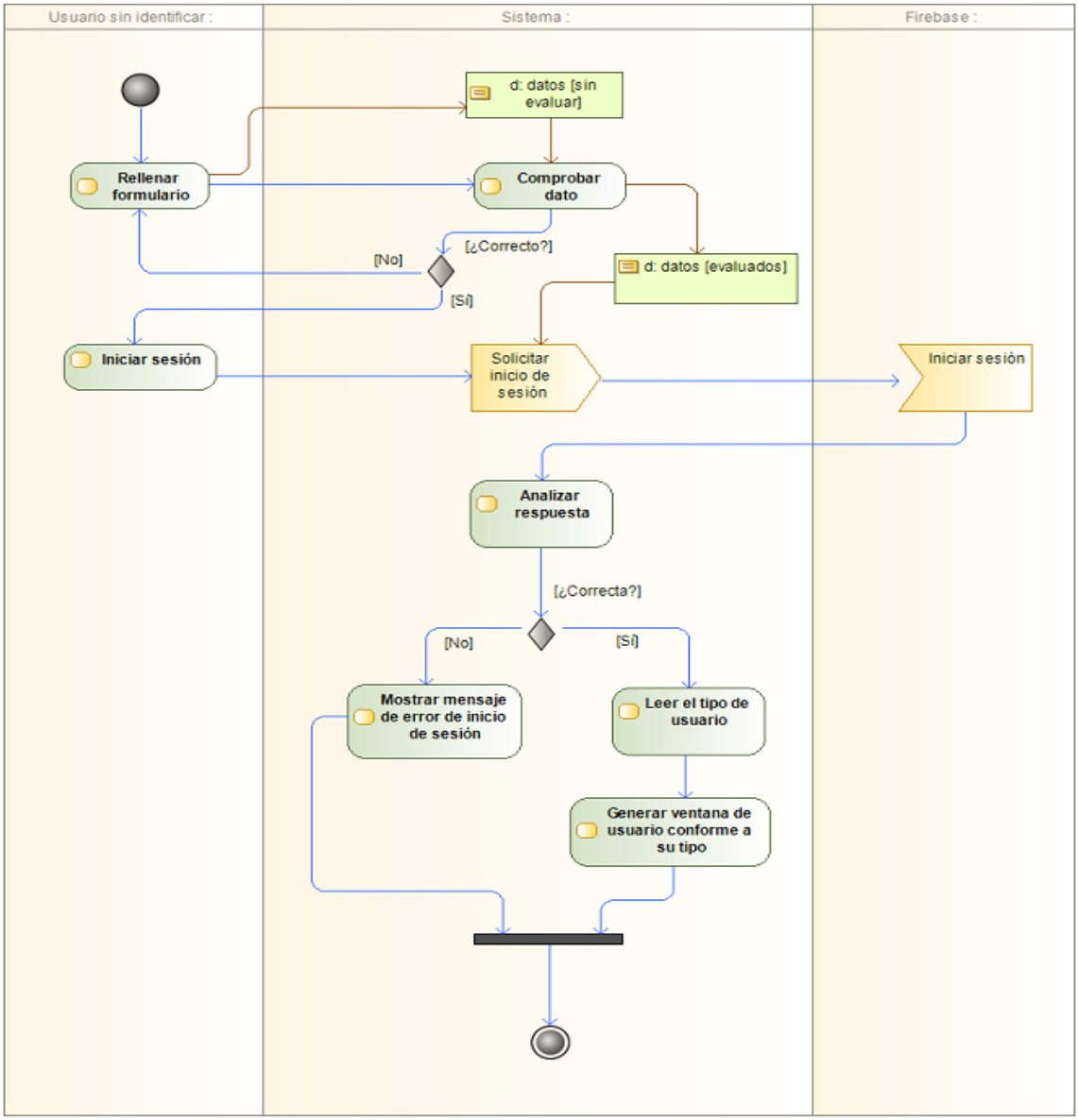


Figura 4.3. Diagrama de actividad para el inicio de sesión.

Para esta parte se requerirán de dos accesos al servidor:

- Uno donde se compruebe los credenciales del usuario y nos los devuelva.
- Otro donde se compruebe el tipo de usuario que tiene asociada la cuenta.

En el primer caso, el acceso se hará a la herramienta **Authentication**, donde se comprueba si el usuario existe o no. Si la cuenta existe, se continua con el siguiente paso, pero si no es así, se le notificará al usuario que los credenciales no son válidos y se cancelará el inicio de la sesión.

En el segundo,se hará uso de la base de datos **Firestore** donde, gracias al identificador que nos dará

Authentication, se obtendrá el documento asociado.

4.2.1.2. Registro

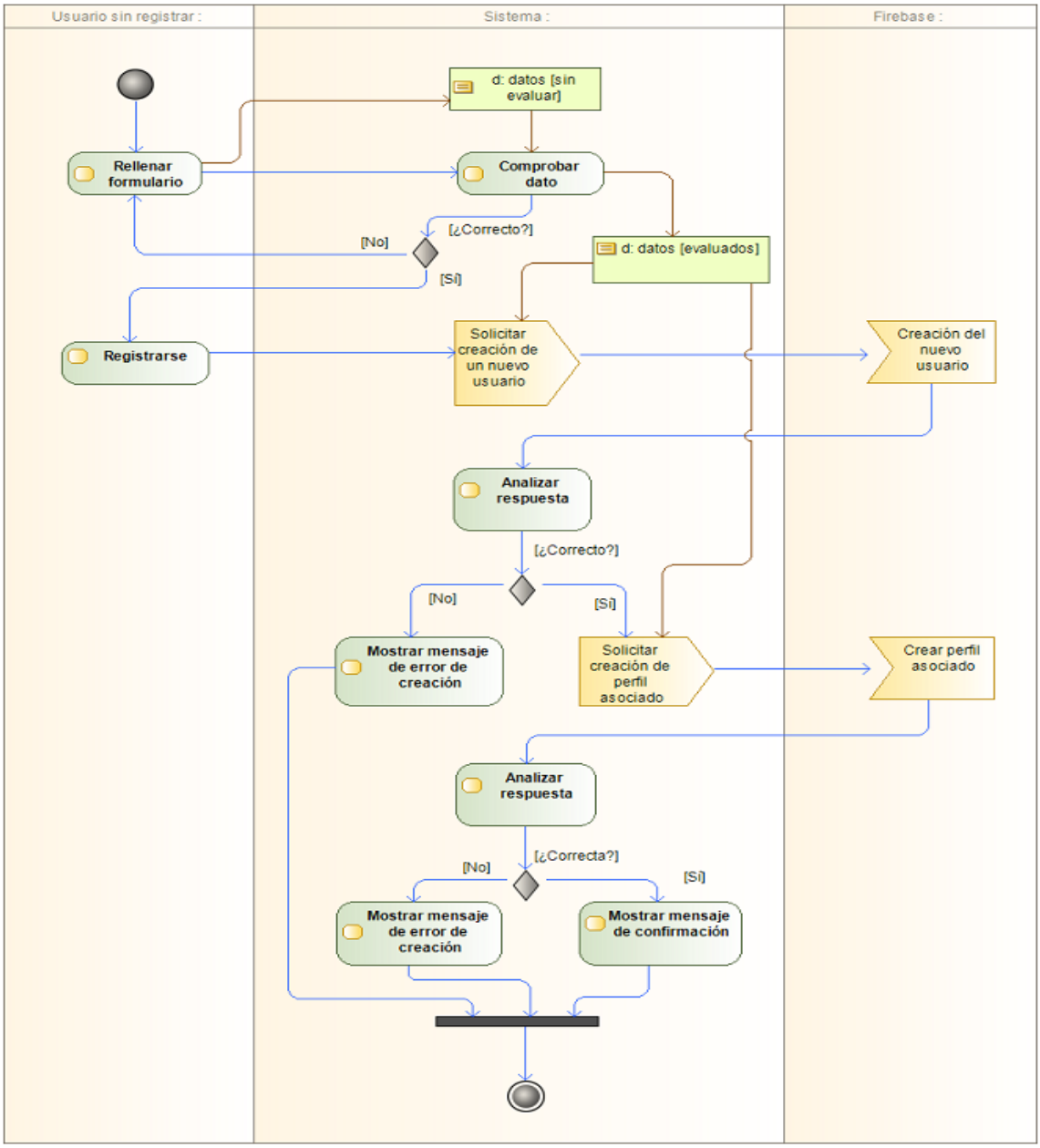


Figura 4.4. Diagrama de actividad para el registro.

Para el registro se necesitarán de dos accesos al servidor:

- Uno para la creación de la cuenta.
- Otro para incluir los datos relativos al perfil de usuario.

El primer acceso, su uso se hace a través de la herramienta **Authentication** donde se creará una nueva cuenta de usuario lo que asignará al usuario un identificador único dentro del servidor.

El segundo acceso, se trata de la creación de datos en **Firestore** para almacenar la información personal del usuario.1 Para ello, se crea un nuevo documento con toda la información indicada al que se le asigna como nombre el identificador generado en el paso anterior.

4.2.1.3. Cambios en correo electrónico y contraseña

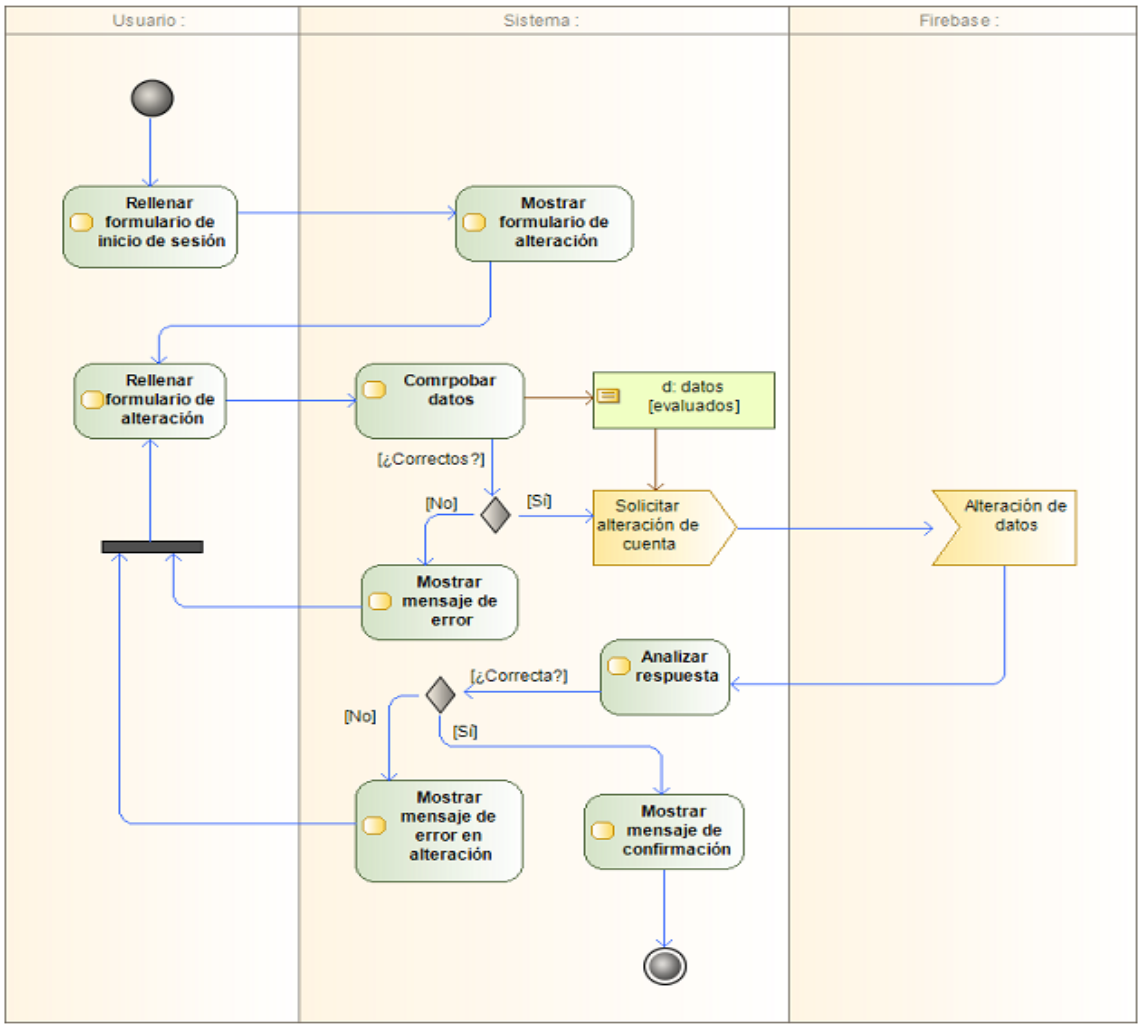


Figura 4.5. Diagrama de edición de datos de la cuenta.

Para permitir alterar los datos de la cuenta el usuario primero debe indicar si desea alterar su correo electrónico o su contraseña. Una vez escogida la opción deberá iniciar sesión y luego escribir dos veces el nuevo valor que desea.
Para esto sólo es necesario establecer tres conexiones con la herramienta **Authentication**:

- Una para el inicio de sesión.
- Otra para efectuar la alteración.
- La última para cerrar la sesión una vez se salga de la pantalla de edición (se haya completado o no)

En la figura 4.5. se puede ver el ejemplo de funcionamiento para cualquiera de las dos opciones, ya que ambas funcionan del mismo modo.

4.2.1.4. La estructura implementada

Tras analizar los puntos anteriores se puede entender que la solución más práctica es que nada más iniciar la aplicación se muestre una actividad cuya interfaz incluya un espacio para el inicio de sesión y las opciones para el registro y edición de parámetros.

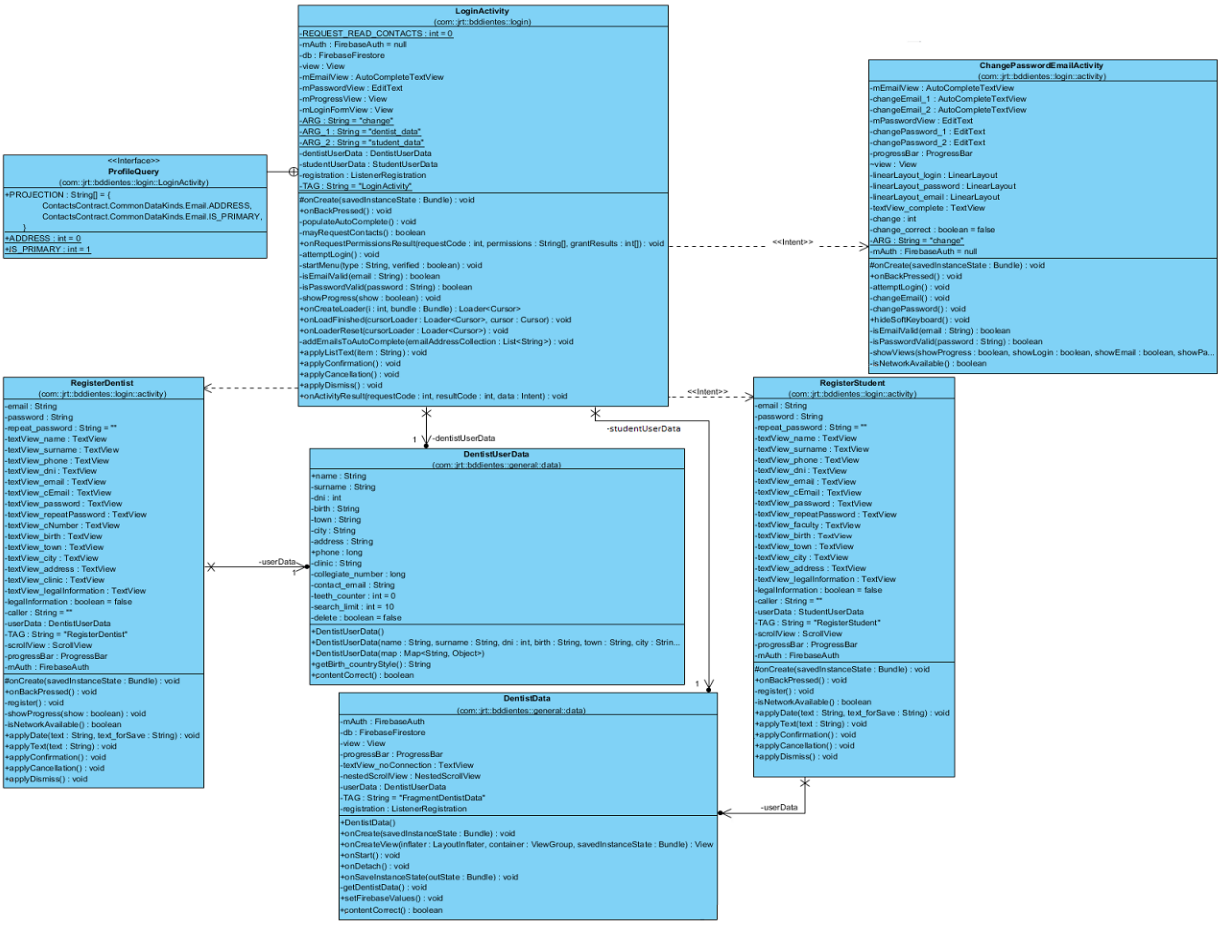


Figura 4.6. Diagrama de las clases para el inicio.

Para este propósito se han utilizado tres actividades (clases):

- Una que muestre la ventana de inicio de sesión y que será la principal de la aplicación.
- Una que muestre una ventana con un registro.
- Una que muestre una ventana para alterar los credenciales de la cuenta.

Para la primera ventana se ha decidido usar la actividad por defecto que **Android Studio** nos ofrece para los inicios de sesión y que incluye una estructura básica.

La actividad ha sido modificada, ampliando su comportamiento para conectar con el servidor **Firebase** y realizar las siguientes funcionalidades:

- Iniciar sesión al introducir correctamente los credenciales y lanzar el menú correspondiente a su tipo de usuario.
- Dar acceso a la actividad de registro, en función del tipo de usuario seleccionado, donde se mostrará un formulario que deberá rellenar y confirmar para crear una cuenta en **Athentication** y el perfil asociado en **Firestore**.
- Comprobar al iniciar sesión si el usuario aún no ha aceptado el mensaje de confirmación que se envía al crear la cuenta. Eso lanzará un diálogo donde se le preguntará al usuario si desea que se vuelva a enviar el mensaje.
  Independientemente de lo que decida, la sesión se cerrará.
- Permitir alterar el correo electrónico y la contraseña de su cuenta mediante una actividad que pedirá primero iniciar sesión y luego indicar lo nuevos valores. Independientemente de si el usuario decide modificar esos parámetros o no, una vez salga de esta actividad se cerrará la sesión.

4.2.1.5. Interfaz de usuario

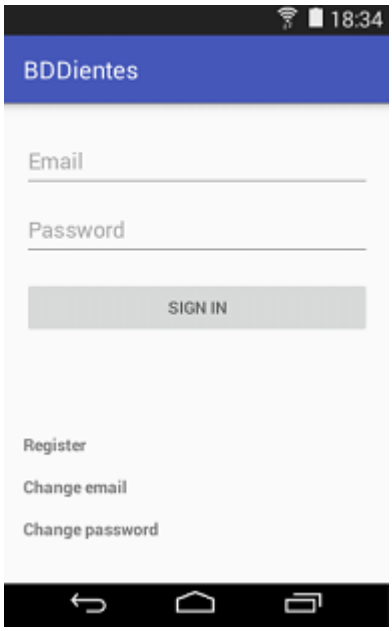


Figura 4.7.

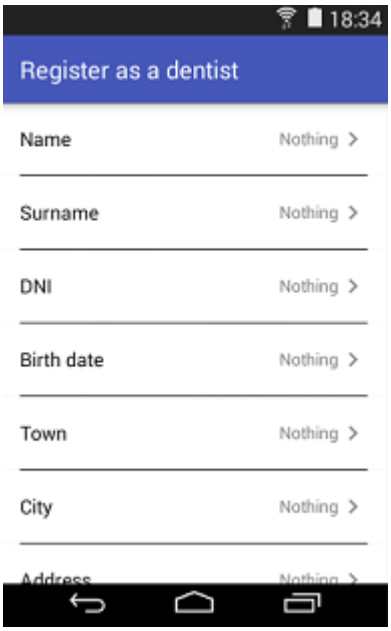


Figura 4.8.

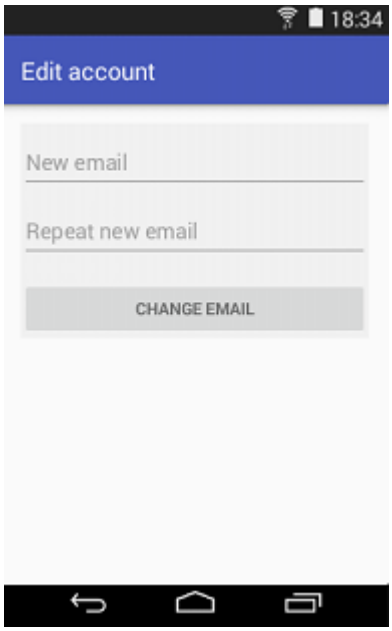


Figura 4.9.

Cada una de las actividades tiene una interfaz asignada:

- Para el inicio de sesión se ha diseñado una que permita acceder al inicio de sesión y a cada uno de los aspectos indicados anteriormente mediante cajas de texto (figura 4.7.)
- Para el registro tanto del estudiante como del dentista se han diseñado dos formularios distintos (figura 4.8.).
- Para el cambio de contraseña o correo electrónico se usa la misma interfaz mostrando en primer momento un cuadro de inicio de sesión y luego sólo el cuadro de diálogo de edición

asignado a la opción seleccionada en la pantalla inicial (figura 4.9.).

4.2.2. La versión para el dentista

En esta parte se explicará los detalles relativos a aquellas interfaces a las que sólo podrá acceder el usuario de tipo dentista.

4.2.2.1. El menú

Para permitir al usuario acceder a cada una de las opciones disponibles se ha optado por crear una ventana con un menú integrado.
Este dispondrá de las siguientes opciones:

- Perfil del usuario.
- Listado de piezas dentales creadas y sin seleccionar.
- Listado de piezas dentales que han sido solicitadas.
- Listado de piezas dentales eliminadas.
- Información.
- Cerrar sesión y volver a la ventana de inicio de sesión.

Para el perfil y los listados se hará uso de la herramienta **Firestore** para recuperar toda la información contenida, realizando búsquedas tanto a los perfiles como a las piezas dentales.

La descarga de las imágenes se hará mediante la herramienta **Storage**.

La ventana de información no necesitará ningún tipo de conexión ya que sólo mostrará información legal y el manual de la aplicación.

Cerrar sesión obligará a enviar una petición a la herramienta **Authentication** para que desconecte al usuario del servidor.

4.2.2.1.1. La estructura implementada

Para el menú se ha creado una actividad llamada *MenuDentist* que usa el diseño por defecto de la actividad menú a la que se le han ido añadiendo las funcionalidades necesarias para gestionar los siguientes elementos:

- La ejecución de los fragmentos relativos a las opciones del menú.
- La ejecución de un fragmento (el perfil de usuario) que servirá como interfaz inicial y final en cada acceso a esta actividad. El propósito es conseguir que el usuario tenga presente que esa opción y su fragmento asociado representen la interfaz principal del menú.
- El despliegue de un diálogo de confirmación cuando se pulse el botón de cerrar sesión o al pulsar sobre el botón de atrás estando en la interfaz principal que al ser aceptada enviará la petición a la herramienta **Authentication** y permitirá volver al inicio de sesión.
- La pila de fragmentos cada vez que se ejecute uno de los fragmentos iniciales con el

propósito de liberar la memoria del dispositivo.

- La ejecución de las actividades de edición de perfil e inclusión de una nueva pieza dental. Para esta última primero se lanzará un diálogo para seleccionar una de las distintas dentaduras posibles para enviársela a la actividad[1].

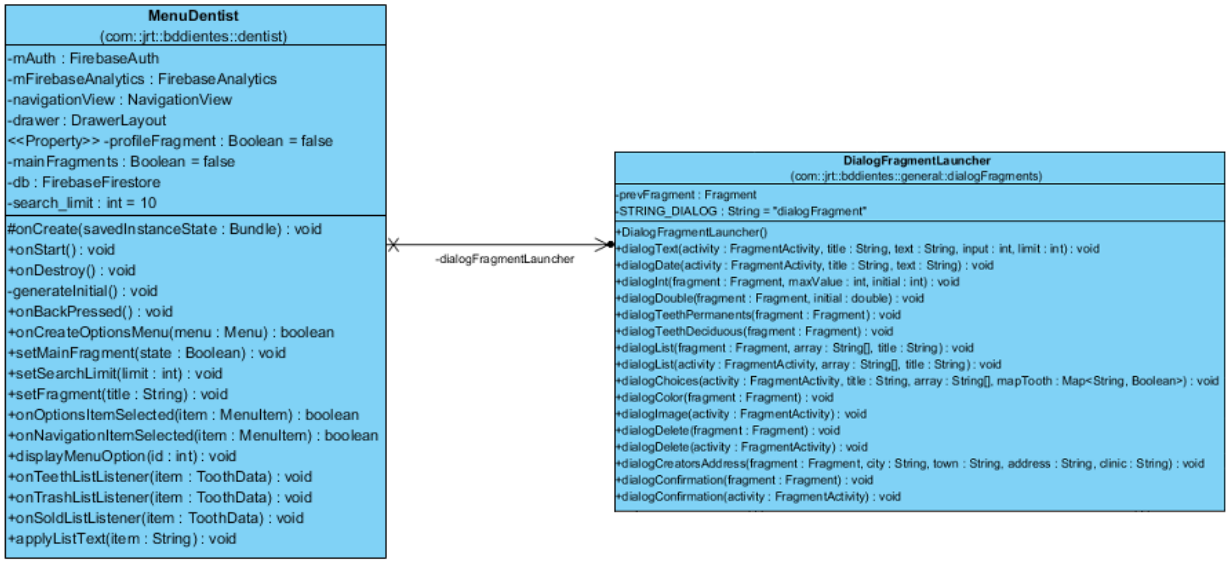


Figura 4.10. Estructura de la clase menú.

4.2.2.1.2. Interfaz de usuario

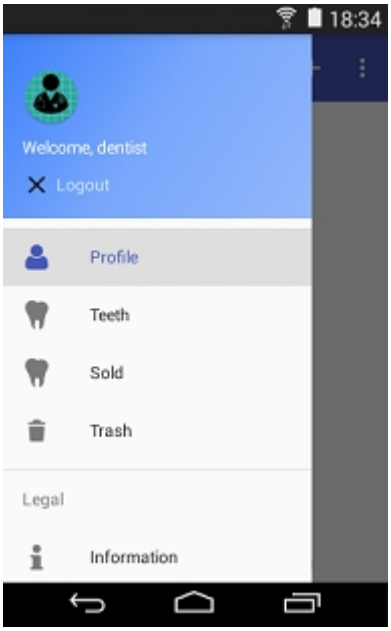


Figura 4.11. Interfaz menú del tipo dentista.

1 El propósito es que la opción seleccionada varíe el contenido que muestra.

Para representar todo lo indicado se ha usado la interfaz de usuario de menú por defecto al que se le han incorporado los elementos necesarios:

- Las distintas opciones.
- La imagen de un dentista en la parte superior.
- El mensaje de bienvenida.
- La opción de cerrar sesión.

4.2.2.2. Las opciones del menú (fragmentos iniciales)

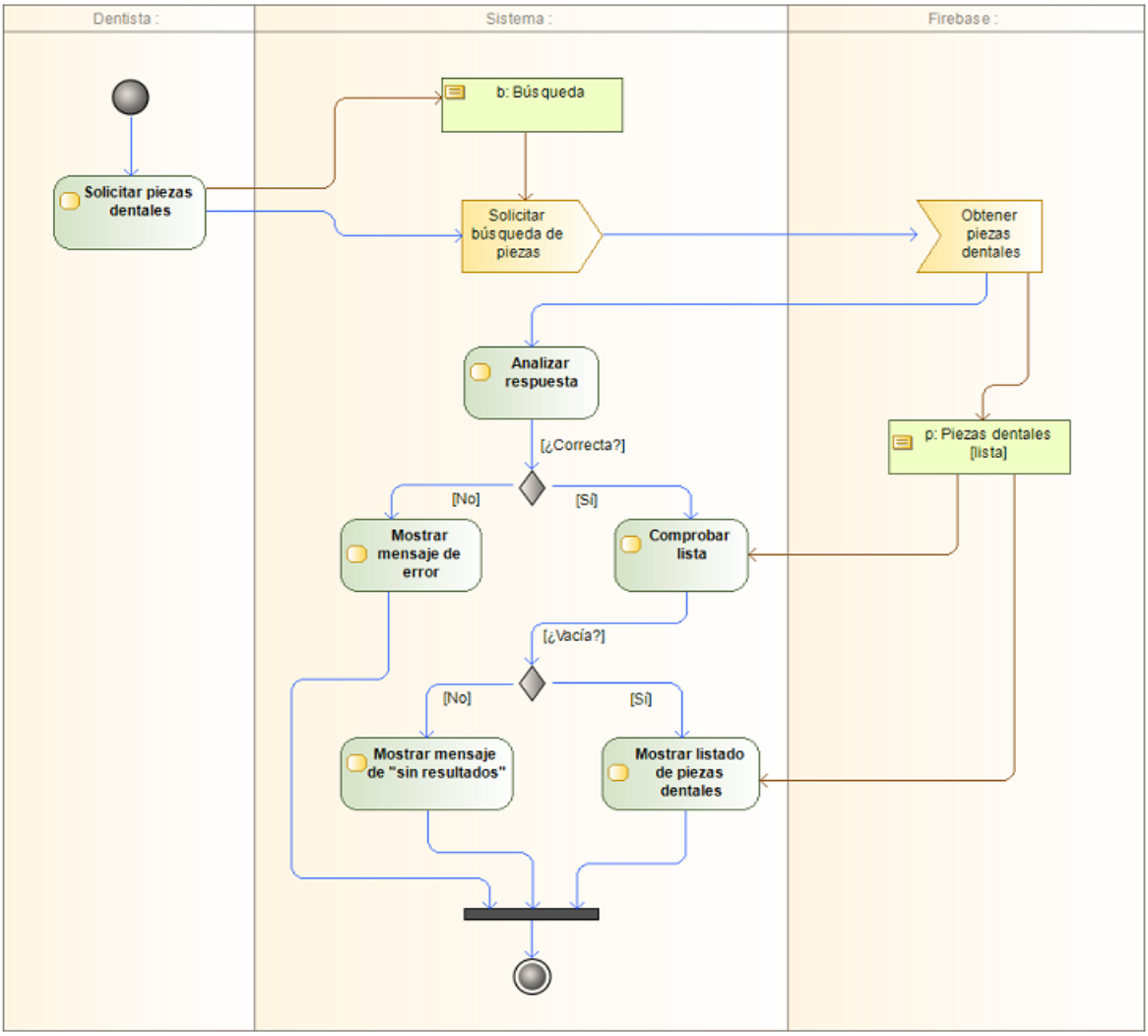


Figura 4.12. Diagrama de actividad para la solicitud de piezas dentales.

Para cada comunicación se requiere gestionar un comportamiento y una escucha a los datos del servidor de manera distinta.

Para descargar el perfil del usuario será necesario acceder a **Firestore** y buscar en él el usuario

mediante el identificador que le ha otorgado **Authentication**.

Para mostrar las listas será necesario descargar una colección de datos de **Firestore** cuya cantidad dependerá de la variable *search_limit* del perfil del usuario.

En la figura 4.12 se puede comprobar el funcionamiento que tendrá la aplicación cada vez que el usuario acceda a una de las opciones que impliquen algún tipo de listado dental. [1]

4.2.2.2.1. Estructura implementada

Como se puede observar en la figura 4.13, para los fragmentos con listados será necesario el uso del objeto *ListData* para almacenar el listado de piezas dentales recuperadas del servidor.

Para mostrar correctamente cada uno de los elementos de las filas del listado será necesario el uso de un adaptador para configurar el diseño de cada una.

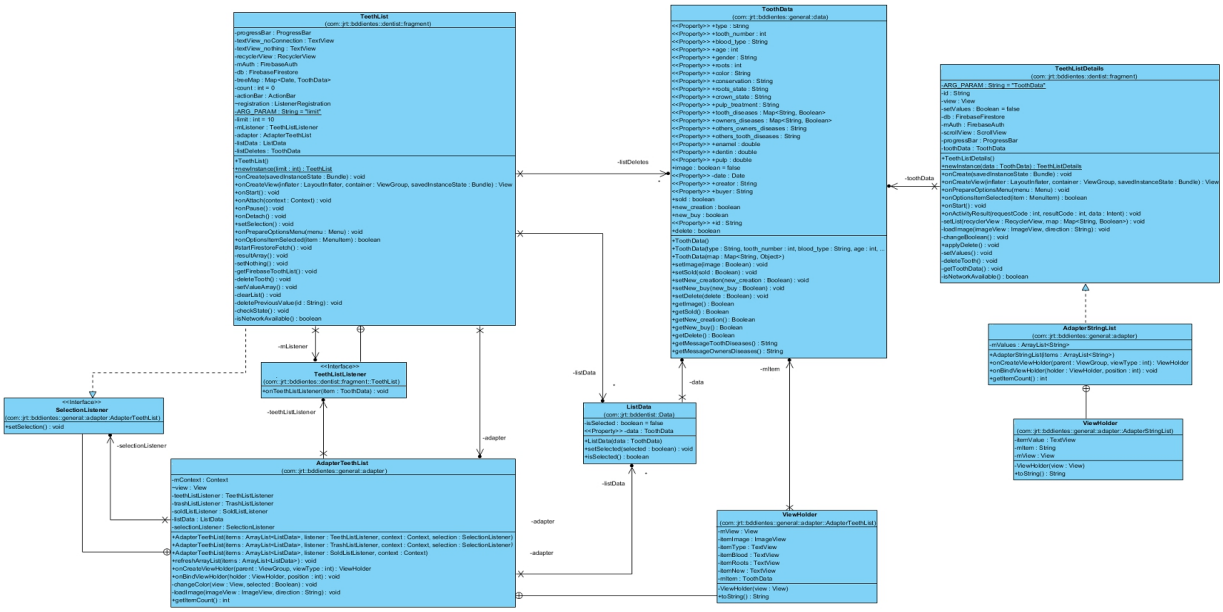


Figura 4.13. Estructura para el fragmento *TeethList*.

Para el fragmento relativo al perfil será necesario un objeto *DentistUserData* para almacenar y mostrar los valores contenidos.

Cada uno de los fragmentos que acceden al servidor tienen la capacidad de reiniciarse. El propósito es permitir al usuario volver a ejecutar la búsqueda de elementos en el servidor en las situaciones en las que se haya producido algún tipo de error.[2]

A continuación, se detallan los fragmentos que se han usado y su principales características.

1 El resto de diagramas de clase se encuentran en el Anexo.
2 Para ello será necesario deslizar la interfaz de usuario hacia abajo.

Perfil de usuario (Profile). (Figura 4.14)

- Es el primer y último fragmento que se le muestra al usuario dentro del menú.
- Se mostrará la información de perfil asociada al usuario tras una realizar una consulta de sus datos en la herramienta **Firestore**. Esta se realizará en la ejecución de *onStart()*.
- La escucha implementada será permanente y estará asociada al ciclo de vida de la actividad invocadora.
- El valor de la variable *search_limit* del perfil será enviada a la actividad invocadora para limitar los tamaños de búsqueda.
- Una vez obtenidos los datos se asignarán a los elementos visuales de la interfaz.
- Su escucha siempre está en funcionamiento y su vida está asociado a la de la actividad invocante.

Listado de piezas dentales creadas y sin seleccionar (TeethList). (Figura 4.15)

- Se realizará una búsqueda de piezas dentales donde el valor de la variable *buy* sea *false* y cuyo creador sea igual al usuario actual (la variable *creator* debe contener el mismo identificador asociado al usuario actual).
- Para almacenar y mostrar cada uno de los elementos recuperados se hará uso de un lista de objetos *ListData*.
- Cada uno de los elementos de la lista podrá ser seleccionado. El propósito es permitir que el usuario pueda enviarlos a la papelera. Para ello se ejecutará un bucle que enviará tantas actualizaciones al servidor como objetos seleccionados hayan donde el parámetro *delete* de las pieza pasará a ser *true*.
- Debido a la necesidad de recoger las alteraciones en los datos mostrados, la escucha implementada para los datos buscados es permanente. La vida de esta está asociada a la del fragmento.

Listado de piezas dentales que han sido solicitadas (SoldList). (Figura 4.16)

- Se realizará una búsqueda de piezas dentales donde el valor de la variable *buy* sea *true* y cuyo creador sea igual al usuario actual (la variable *creator* debe contener el mismo identificador asociado al usuario actual).
- Para almacenar y mostrar cada uno de los elementos recuperados se hará uso de un lista de objetos *ListData*.
- Debido a la necesidad de recoger las alteraciones en los datos mostrados, la escucha implementada para los datos buscados es permanente. La vida de esta está asociada a la del fragmento.

Listado de piezas dentales eliminadas (TrashList). (Figura 4.17)

- Se realizará una búsqueda de piezas dentales donde el valor de la variable *delete* sea *true* y cuyo creador sea igual al usuario actual (la variable *creator* debe contener el mismo identificador asociado al usuario actual).
- Para almacenar y mostrar cada uno de los elementos recuperados se hará uso de un lista de objetos *ListData*.
- Cada uno de los elementos de la lista podrá ser seleccionado. El propósito es permitir que el usuario pueda recuperalos de la papelera (enviarlos de vuelva a *TeethList*). Para ello se ejecutará un bucle que enviará tantas actualizaciones al servidor como objetos seleccionados hayan donde el parámetro *delete* de las pieza pasará a ser *false*.

- También se incluye la posibilidad de eliminar todos o sólo los elementos seleccionados de manera permanente. Esto se realiza mediante un bucle que irá mandando las peticiones necesarias para eliminar los datos del servidor.
- Debido a la necesidad de recoger las alteraciones en los datos mostrados, la escucha implementada para los datos buscados es permanente. La vida de esta está asociada a la del fragmento.

Información (information).[1]

- Se limita a mostrar la ayuda sobre la aplicación, derechos de autor e información legal.

4.2.2.2.1. Interfaz de usuario

En este apartado se muestran las interfaces de usuario diseñadas para cada uno de los fragmentos.

En el caso de que se haya producido algún problemas (los definidos u otros relacionados con los datos recogidos) se ha incluido la posibilidad de deslizar la pantalla hacia abajo con el objetivo de recargar la interfaz y toda la información contenida en él.

La visualización de imágenes se ha realizado mediante **Glide**. Se ha optado por permitir sólo su almacenamiento temporal en la caché para evitar que se produzca un desbordamiento en ella.

Para el listado de piezas dentales creadas y sin seleccionar, se ha incluido una pequeña etiqueta dentro del objeto que notifica si una pieza ha sido creada recientemente y aún no ha sido visualizada.

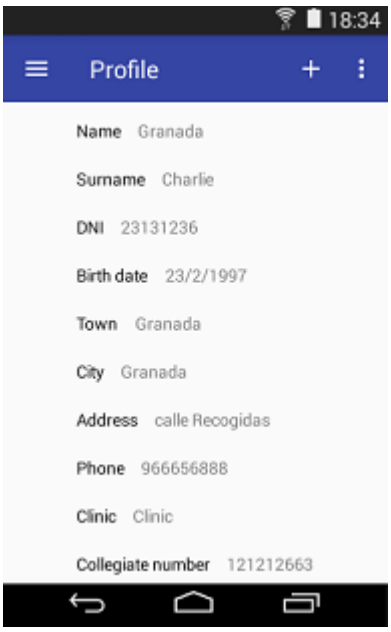


Figura 4.14. Ventana de perfil.

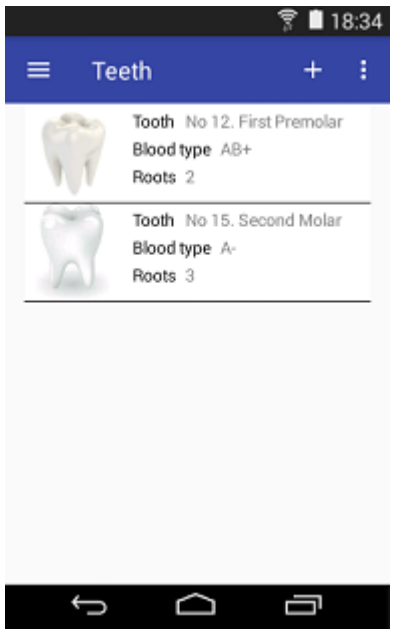


Figura 4.15. Ventana de listado de piezas

1 En este documento no se muestra ningún ejemplo de esta interfaz debido a su extrema sencillez: un mensaje de texto sobre una interfaz básica.

dentales creadas y sin solicitar.

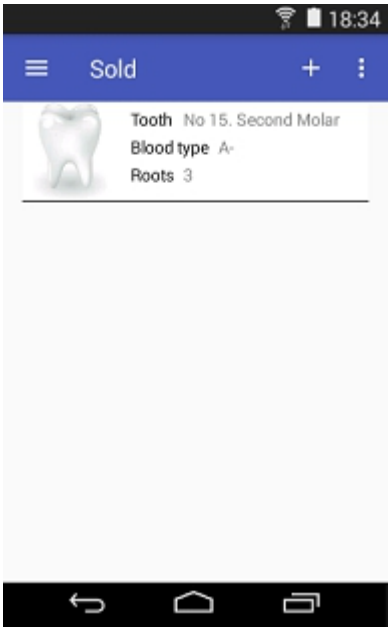


Figura 4.16. Ventana de piezas dentales solicitadas.

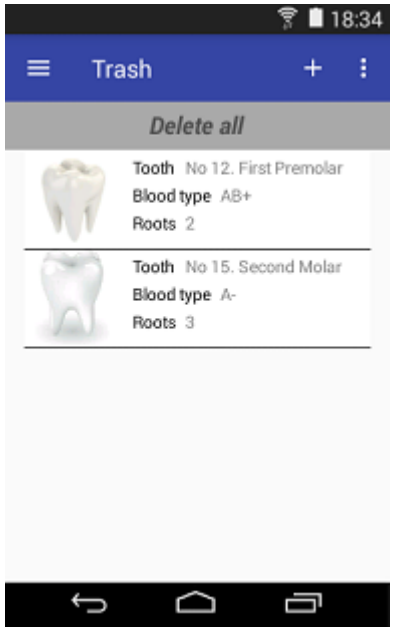


Figura 4.17. Ventana de piezas enviados a la papelera.

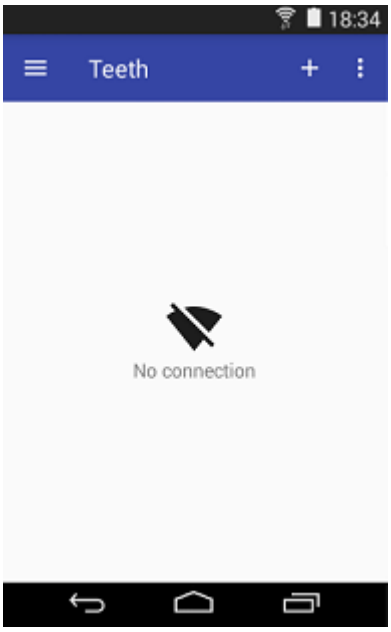


Figura 4.18. Ventana de no conexión.

Figura 4.19. Ventana de inexistencia de piezas dentales.

Por último, para todos estos diseños se han implementado una serie de “pantallas” dentro de las

interfaces de cada fragmento para informar al usuario sobre ciertos eventos[1]:

- La falta de conexión del dispositivo. (figura 4.18.).
- La no existencia de elementos dentro de los parámetros (Figura 4.19.).

4.2.2.3. Información detallada (Fragmentos secundarios)

Para mostrar la información detallada de cada una de las piezas seleccionadas es necesario del uso de un fragmento que reciba el objeto *ToothData* correspondiente y lo muestre por pantalla.

Se han creado un fragmento para visualizar los detalles por cada listado creado.
Las relaciones entre ellos son las siguientes:

- *TeethList → TeethListDetails*
- *SoldList → SoldListDetails*
- *TrashList → TrashListDetails*

4.2.2.3.1. Estructura implementada

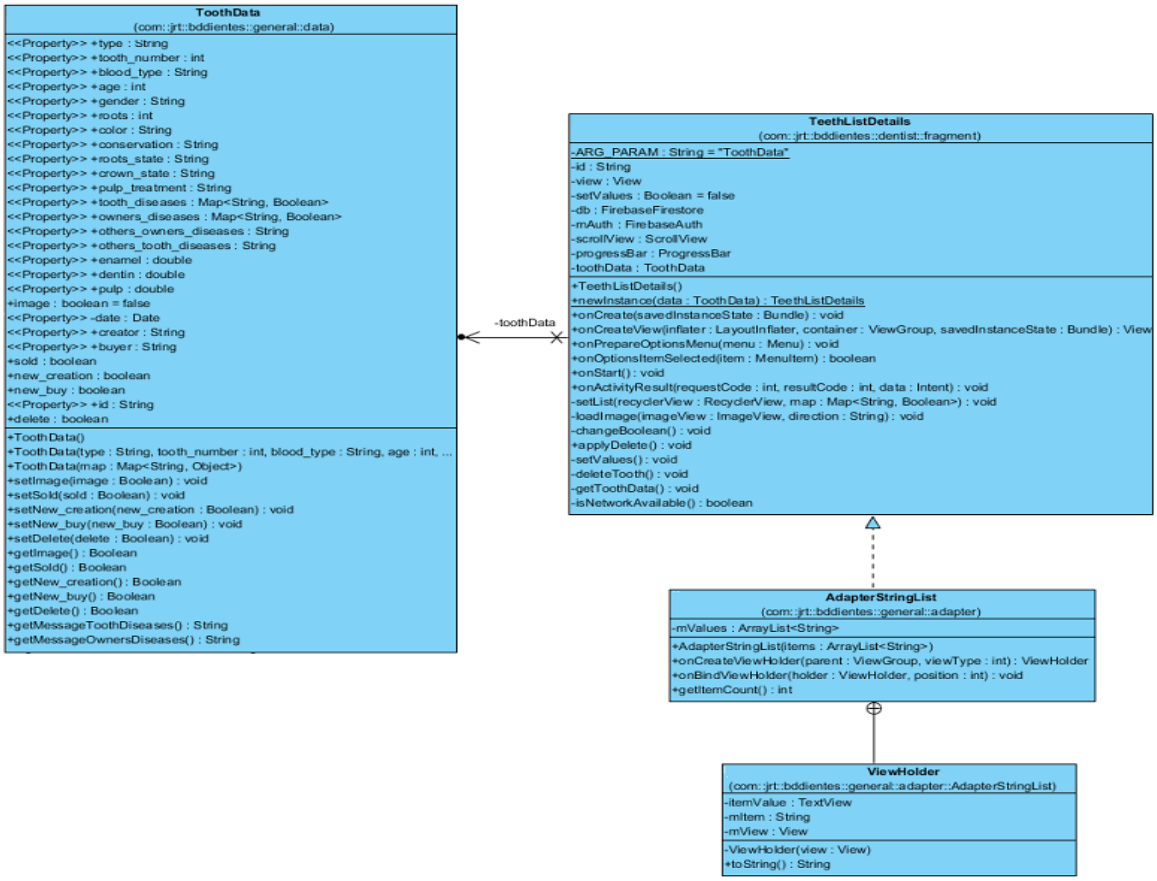


Figura 4.20. Estructura para *TeethListDetails*.

1 Estas interfaces son comunes a todos los listados de la aplicación independientemente de la versión usada.

Todos los fragmentos poseen el mismo diseño, sólo *TeethListDetails y TrashListDetails* poseen características diferentes.

TeethListDetails

- Permite la edición de una pieza dental. Para ello se ejecutará la actividad de editar pieza dental con el objetivo que devuelva un resultado. Si el resultado obtenido es positivo, el fragmento realizará una consulta al servidor para obtener los datos de la pieza actualizados.
- Permite enviar la pieza a la papelera. Se realizará mediante una actualización al servidor indicando que la variable *delete* pasa a ser *true*.

TrashListDetails

- Permite recuperar la pieza de la papelera. Se realizará mediante una actualización al servidor indicando que la variable *delete* pasa a ser *false*.
- Permite eliminar definitivamente la pieza. Se realizará mediante una solicitud de eliminación de la pieza al servidor. Tras ello se devolverá al fragmento anterior.

4.2.2.3.2. Interfaz de usuario

En este apartado sólo se mostrará una de las tres interfaces implementadas debido a que todas ellas son iguales en contenido.

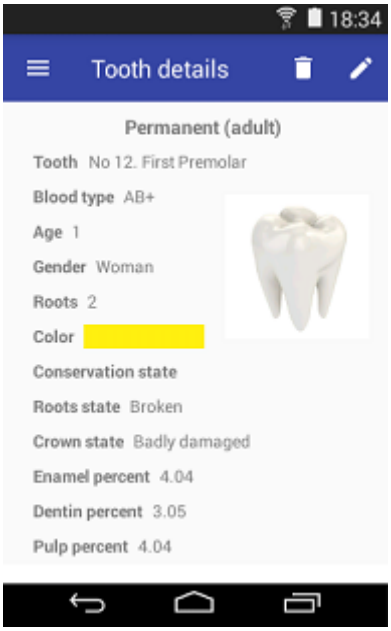


Figura 4.21. Interfaz para mostrar todos los datos de la pieza dental.

Cada una de las interfaces contará con un título que determinará el tipo de pieza dental seleccionada, sus datos (aquellos relativos a gestiones internas no se muestran) y la imagen asociada (se descargará de nuevo mediante **Glide** usando las mismas limitaciones de caché que en las listas).

En función de si el elemento “Others” ha sido seleccionado en alguna de las listas de enfermedades, se mostrará o no el aparatado y contenido de la variable que introdujo el usuario.

4.2.2.4. Añadir una pieza dental

El propósito de este aparatado es definir cómo el usuario dentista puede incluir nuevas piezas dentales a la plataforma.

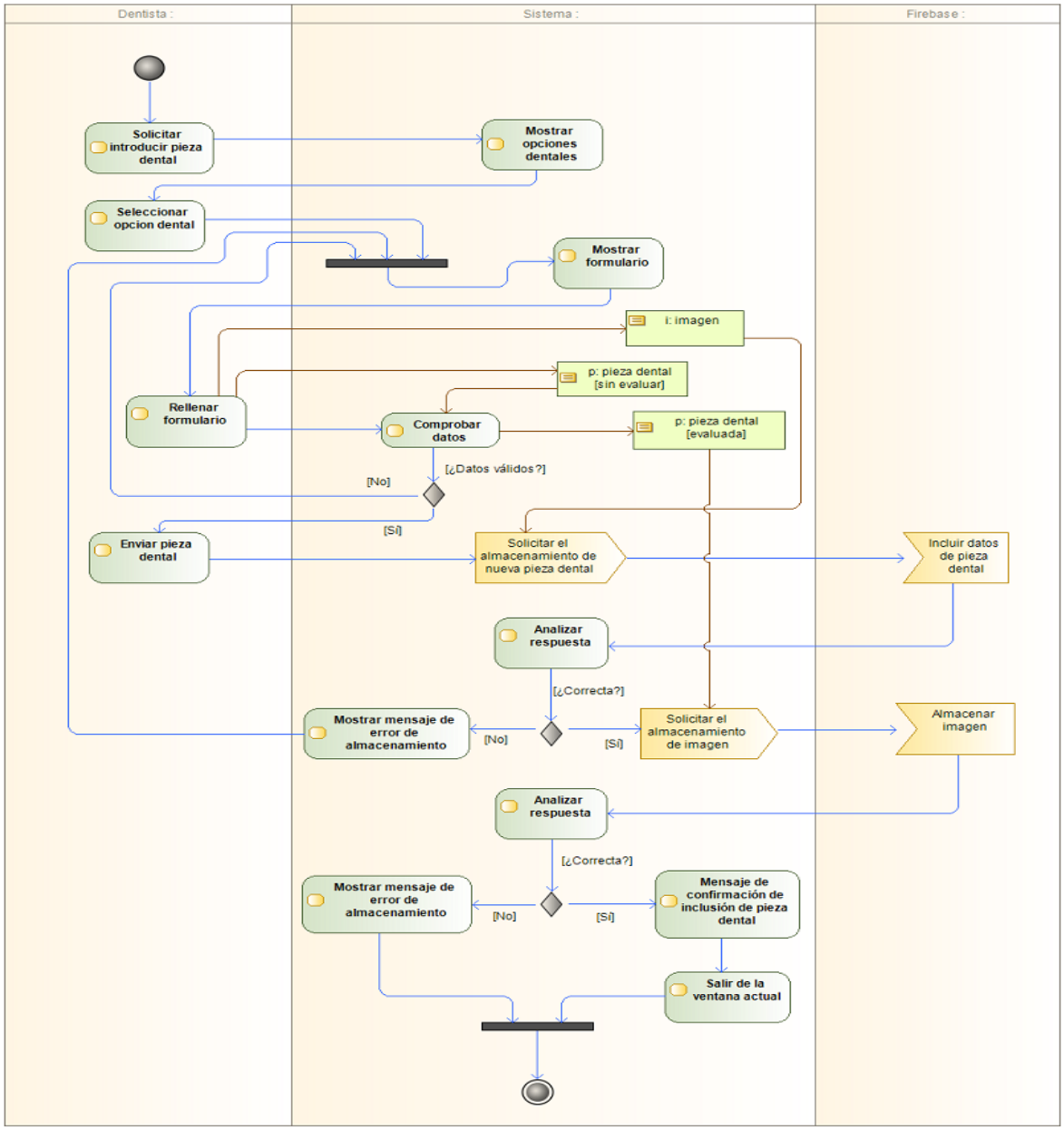


Figura 4.22. Diagrama de actividad para la creación de una pieza dental.

Como se observa en la figura, se necesitan cuatro pasos para crear una pieza dental en el sistema:

- Rellenar el formulario de información de la pieza dental.

- Seleccionar una imagen.
- Comprobar los valores.
- Incluir los datos asociados en el servidor.
- Incluir la imagen en el servidor.

Para los dos primeros puntos se ha diseñado una interfaz con un formulario cuyos contenidos serán modificables mediante diálogos.

Debido a que los valores de las piezas dentales no pueden ser aleatorios, muchos los diálogos poseen listas con los valores posibles.
Para los diálogos que permitan incluir texto se ha programada una serie de filtros para evitar malos usos.

Una vez completado todo el formulario, todos los valores se habrán añadido a un objeto *ToothData* que se enviará a la herramienta **Firestore** para la creación del nuevo elemento. Como para cada nuevo almacenamiento se crea un identificador único, se ha usado este para definir el nombre del documento de la pieza en el servidor.

Incluir la imagen en el servidor requiere establecer comunicación con la herramienta **Storage** y crear una nueva pieza cuyo nombre será el identificador que nos ha proporcionado el documento de **Firestore** creado antes.

4.2.2.4.1. La estructura implementada

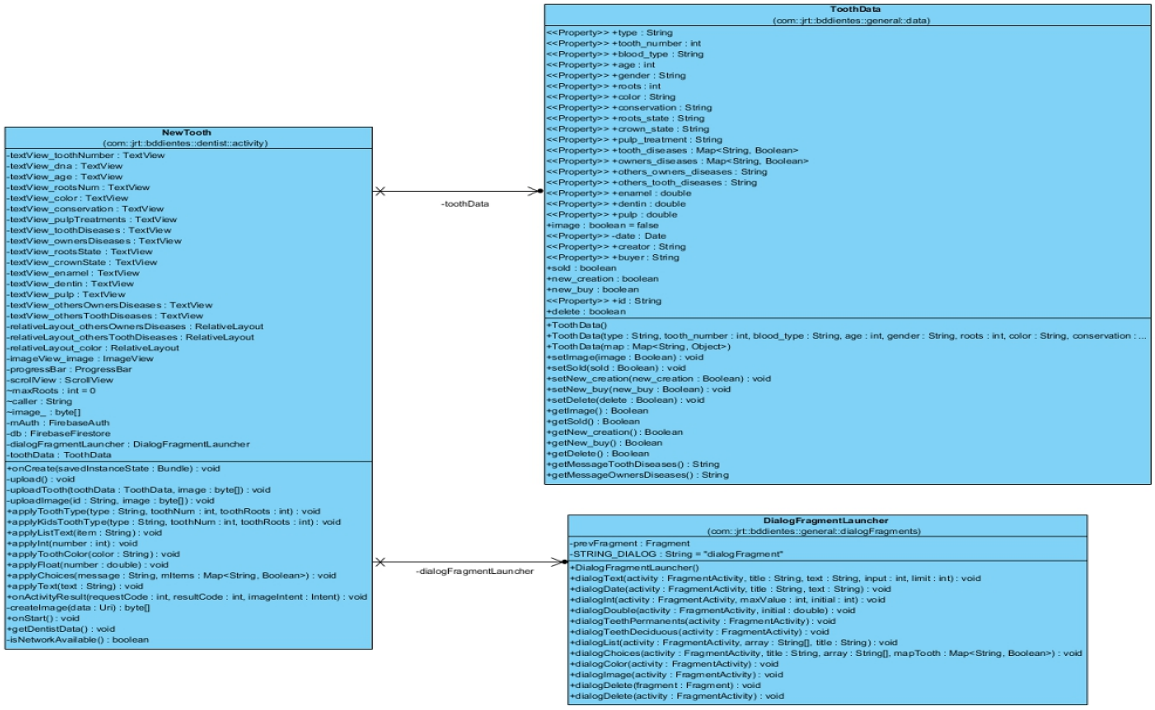


Figura 4.23. Estructura para la creación de una pieza dental.[1]

1 Por falta de espacio no se incluyen ni la clases de los diálogos relacionados ni sus distintas interfaces,

En esta estructura se realizan las siguientes funcionalidades:

- Lanzar cada uno de los diálogos asociados al formulario.
- Muestra u oculta el la opción de incluir una enfermedad específica en función de si en las enfermedades dentales o del propietario se ha incluido la opción “Others”.
- Se recogerá el elemento tipo que se ha seleccionado en el diálogo de incialización y según su valor se mostrará un diálogo para la dentadura distinto.
- Según el valor que se indique en el número de pieza dental, el máximo de raíces máximas permitidas variará.
- Se revisa el contenido devuelto por los diálogos textuales.
- Cada uno de los valores obtenidos por los diálogos se incorporarán a un objeto *ToothData*.
- La imagen será recogida mediante el método *onActivityResult()* que permite obtener el valor seleccionado en diálogo de imagen. Una vez obtenida, pasará por el método *createImage()* donde se editará su calidad para reducir su tamaño.
- Para la creación de la pieza en el servidor se incluirán en el objeto los valores necesarios y no accesibles por el usuario, luego se creará la pieza dental mediante el objeto *ToothData* y, cuando se obtenga la confirmación, se subirá la imagen[1].

Para esta actividad se ejecutan los siguientes diálogos[2]:

- DialogFragmentTeeth_permanent_activity.
- DialogFragmentTeeth_deciduous_activity.
- DialogFragmentColor_activity.
- DialogFragmentInt_activity.
- DialogFragmentDouble_activity.
- DialogFragmentList_activity.
- DialogFragmentChoices_activity.
- DialogFragmentText_activity.
- DialogFragmentImage.

4.2.2.4.2. Interfaz de usuario

El objetivo buscado para esta interfaz ha sido mostrar el formulario de una manera sencilla y poco abrumadora para el usuario.

Cada opción desplegará su diálogo correspondiente y la elección establecida quedará registrada en el cuadro de texto (Si muestra “Nothing” implica que aún no se ha indicado nada).

La opción para seleccionar la imagen de la pieza dental también está dentro del formulario. Por defecto sólo mostrará un fondo que deberá ser pulsado para lanzar el diálogo de imagen. Cuando se haya seleccionado una, el fondo cambiará para mostrar la imagen.

1 La creación no se realizará si el dispositivo no está conectado a la red. Solución obtenida de [sta 01].
2 Para obtener más información sobre los diálogos ir a la sección Diálogos de este capítulo.

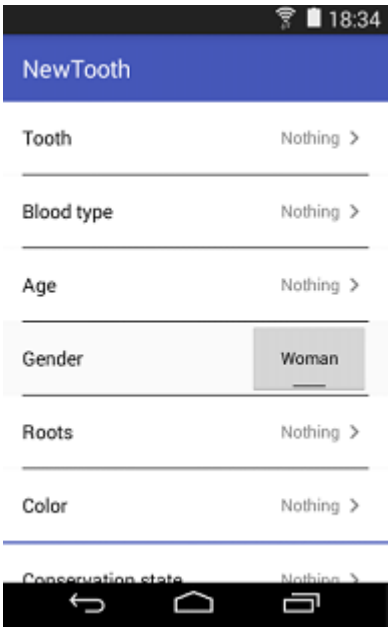


Figura 4.24. Parte superior del formulario.

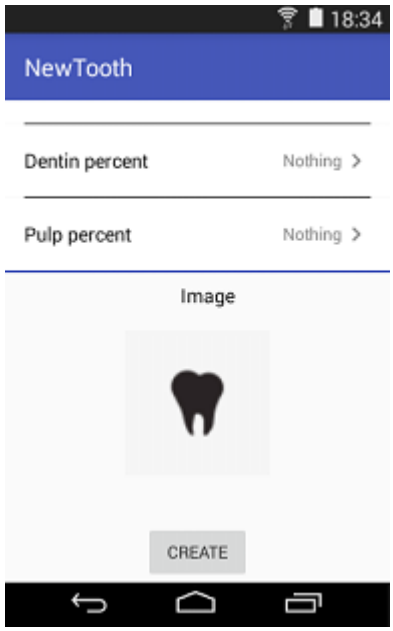


Figura 4.25. Parte inferior del formulario.

Si se produce algún evento importante dentro del código un *Snackbar* con un mensaje en su interior se lo hará saber al usuario.

4.2.2.5. Editar una pieza dental

Para poder desarrollar esto es necesario realizar una serie de tareas:

- Comprobar los datos nuevos introducidos.
- Comprobar el estado de la pieza en el momento actual.
- Añadir todo el contenido nuevo a un objeto *Map.*
- Enviar al servidor los nuevos datos.

El primer punto se podrá gestionar mediante los controles implementados tanto dentro como en fuera de los diálogos.

El segundo requerirá de la descarga de nuevo de la pieza en **Firestore** para revisar si ha sido solicitada por alguien. En tal caso, se anularía la edición.

Para el tercer punto se añadirá en un Map el nombre del elemento a alterar, su nuevo valor y la variable date a null[1]. Al usar este elemento se reduce la carga del elemento a enviar al servidor.

Para el último punto, completar la actualización de la pieza indicada, se deberá indicar la dirección de ella dentro de la herramienta **Firestore**.

En el caso de que se actualice una imagen, se realizará la alteración de datos necesario y se enviará a la herramienta **Storage** mediante la dirección indicada en la variable *id*.

1 Esto sirve para activar en **Cloud functions** la función que añade a esa variable la fecha y hora del servidor.

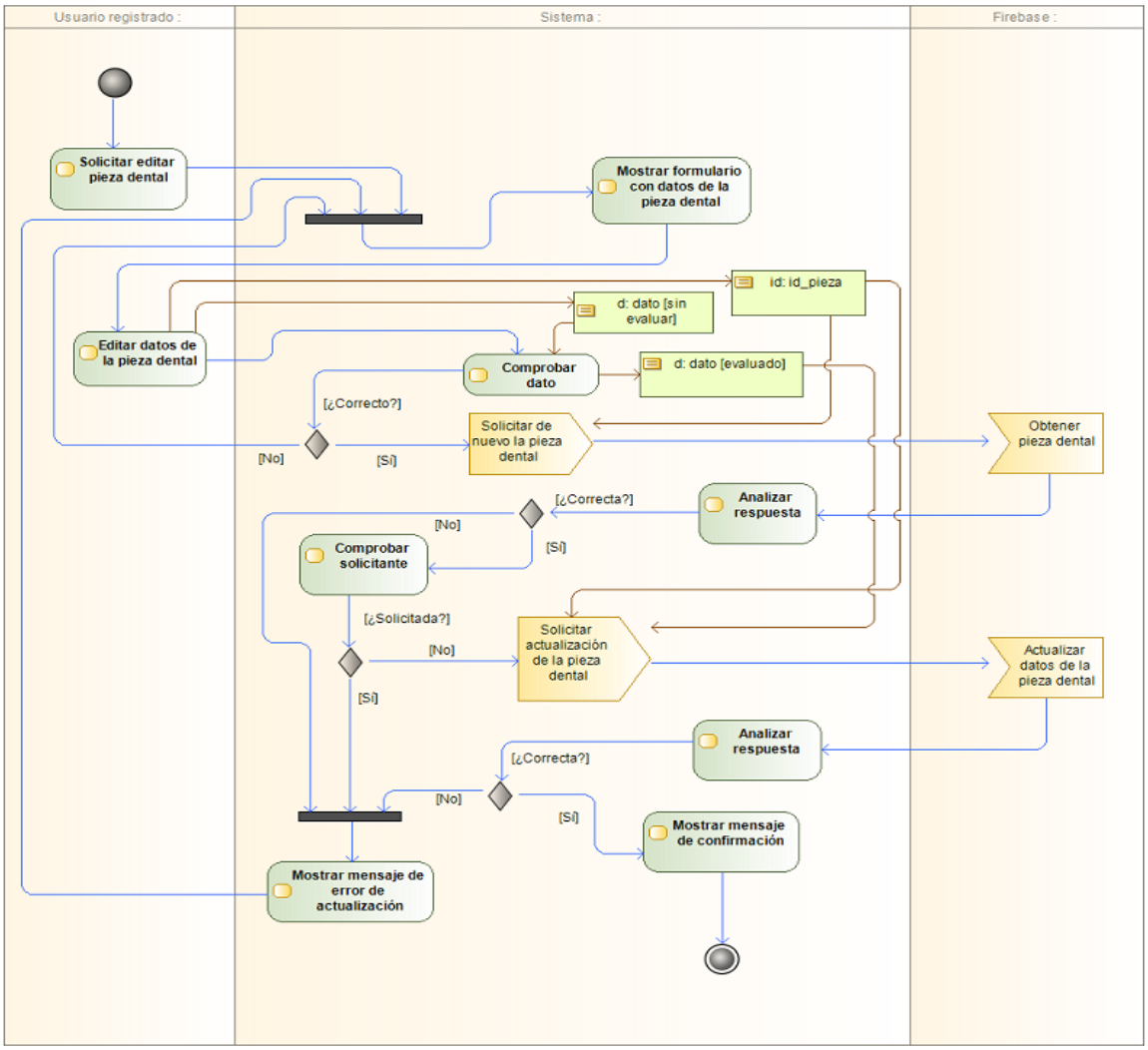


Figura 4.26. Diagrama de actividad para la edición de una pieza dental.

4.2.2.5.1. La estructura implementada

Para diseñar esta estructura se ha utilizado una actividad semejante a la de añadir una pieza dental. En este caso el objeto *ToothData* utilizado ya estará inicializado con los valores de la pieza seleccionada en *TeethListDetails* y son mostrados en cada apartado del formulario.

Las características más importantes de esta estructura son:

- Los valores de los diálogos dependerán del valor tipo de la pieza editada. Ese valor no podrá ser editado.
- Por cada nueva edición se comprobará el nuevo valor y se descargará la pieza de nuevo para comprobar si se puede realizar la actualización. En caso positivo, se enviará al servidor un objeto *Map* con el contenido nuevo[1].
- En el caso de la imagen, primero se ejecutará la función *createImage()* para reducir su

1 La edición no se realizará si el dispositivo no está conectado a la red. Solución obtenida de [sta 01].

calidad, luego se descargará la pieza dental y, si no hay problema, se subirá la nueva imagen.

- Sólo si se ha realizado alguna actualización de manera satisfactoria, al finalizar la actividad se enviará un código de confirmación al fragmentos invocador.

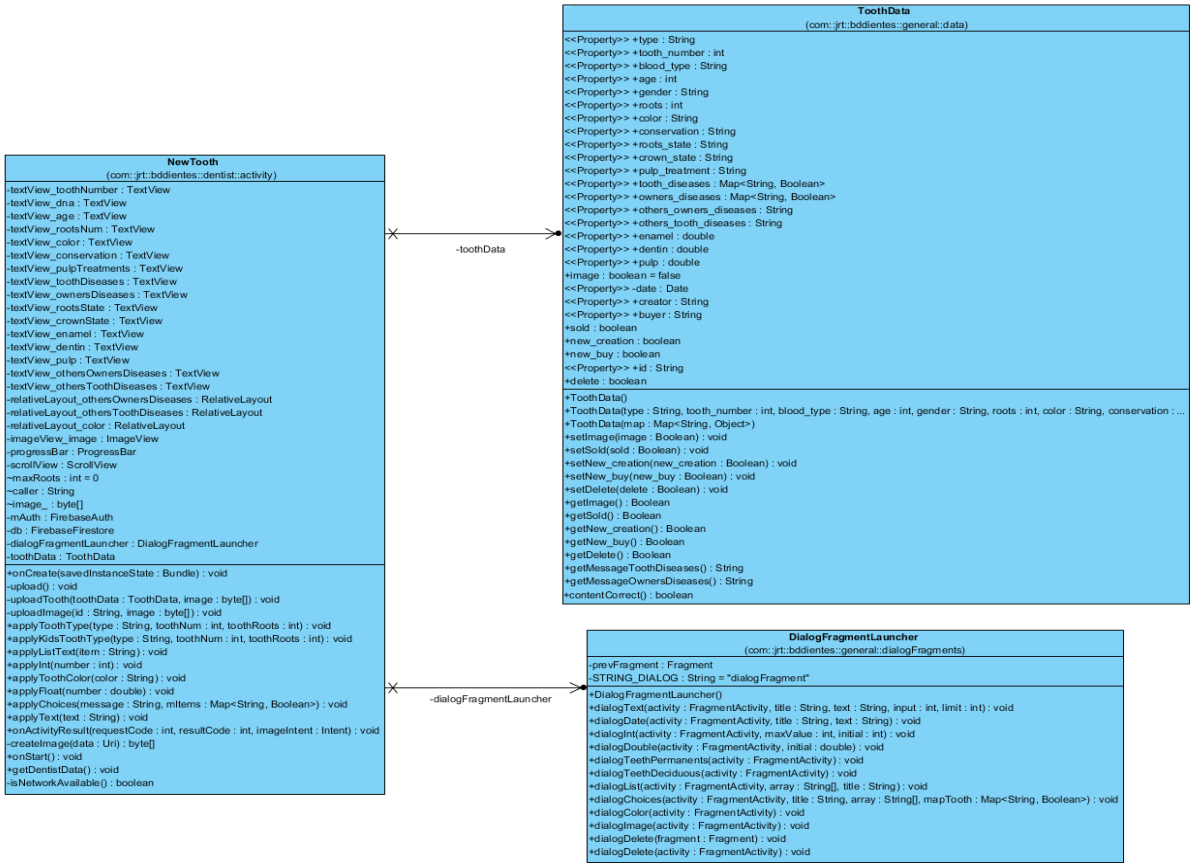


Figura 4.27. Estructura para la actualización de piezas dentales.

Para esta actividad se ejecutan los mismos diálogos[1] para la edición que en *NewTooth*:

- DialogFragmentTeeth_permanent_activity.
- DialogFragmentTeeth_deciduous_activity.
- DialogFragmentColor_activity.
- DialogFragmentInt_activity.
- DialogFragmentDouble_activity.
- DialogFragmentList_activity.
- DialogFragmentChoices_activity.
- DialogFragmentText_activity.
- DialogFragmentImage.

1 Para obtener más información sobre los diálogos ir a la sección Diálogos de este capítulo.

4.2.2.5.2. Interfaz de usuario

Se puede observar que se trata de la misma interfaz que en la clase *NewTooth*, pero con los datos de la pieza dental ya asignados a cada uno de los elementos del formulario.

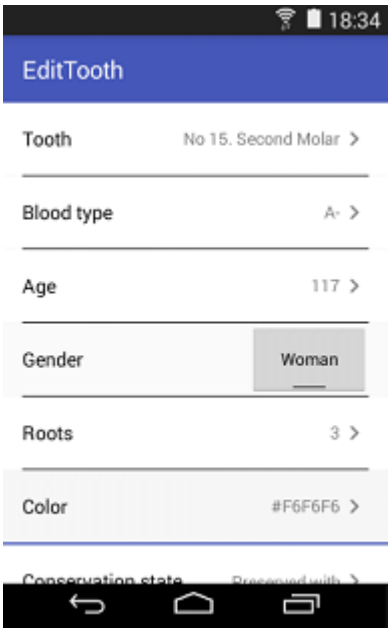


Figura 4.28. Parte superior del formulario.

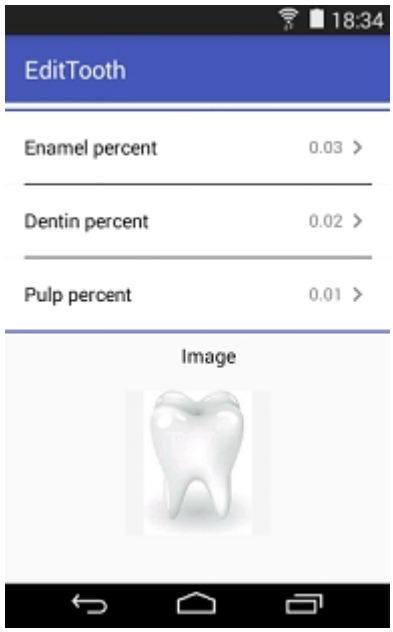


Figura 4.29. Parte inferior del formulario.

Para este formulario la imagen mostrada tanto al inicio como después de la actualización es descargada directamente de la herramienta **Storage** mediante **Glide**. Su propósito es tener la certeza de que la imagen ya está en el servidor.

Si se produce algún evento importante dentro del código un *Snackbar* con un mensaje en su interior se lo hará saber al usuario.

4.2.3. La versión para el estudiante

En esta parte se explicará los detalles relativos a aquellas interfaces que sólo podrá visitar el usuario de tipo estudiante.

4.2.3.1. El menú

Al igual que en el caso del dentista, el usuario de tipo estudiante necesita de un menú que le permita acceder a los distintos aparatos de la aplicación. Cada uno de ellos mostrado en una pantalla distinta del resto.

Para este menú el usuario debe poder acceder a las siguientes opciones:

- Mostrar su perfil de usuario.
- Realizar búsquedas de piezas dentales.
- Mostrar el listado de las piezas dentales que ya ha solicitado.
- Mostrar la información legal y la ayuda de la aplicación.
- Permitir salir de la sesión y volver a a la pantalla de inicio.

4.2.3.1.1. La estructura implementada

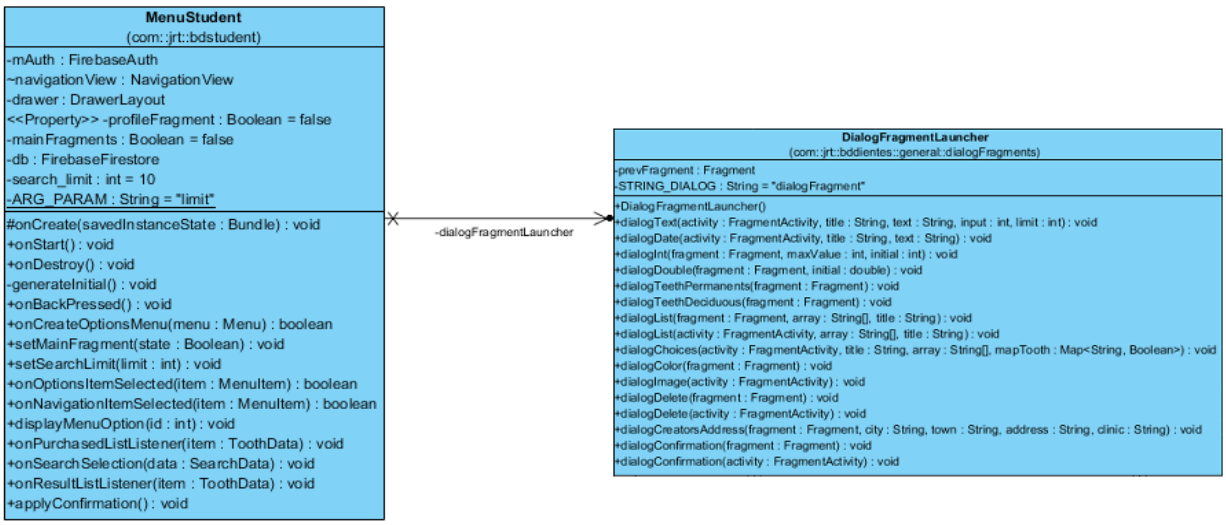


Figura 4.30. Estructura para el menú de usuario del estudiante.

Para el menú se ha creado una actividad llamada *MenuStudent* que usa el diseño por defecto de la actividad menú a la que se le han ido añadiendo las funcionalidades necesarias para gestionar los siguientes elementos:

- La ejecución de los fragmentos relativos a las opciones del menú.
- La ejecución de un fragmento (el perfil de usuario) que servirá como interfaz inicial y final en cada acceso a esta actividad. El propósito es conseguir que el usuario tenga presente que esa opción y su fragmento asociado representen la interfaz principal del menú.
- El despliegue de un diálogo de confirmación cuando se pulse el botón de cerrar sesión o al pulsar sobre el botón de atrás estando en la interfaz principal que al ser aceptada enviará la petición a la herramienta **Authentication** y permitirá volver al inicio de sesión.
- La pila de fragmentos cada vez que se ejecute uno de los fragmentos iniciales con el propósito de liberar la memoria del dispositivo.
- La ejecución de la actividades de edición de perfil.

4.2.3.1.2. Interfaz de usuario

Al igual que en el caso del dentista, para representar todo lo indicado anteriormente se ha usado la interfaz de menú por defecto al que se le han incorporado las opciones necesarias.

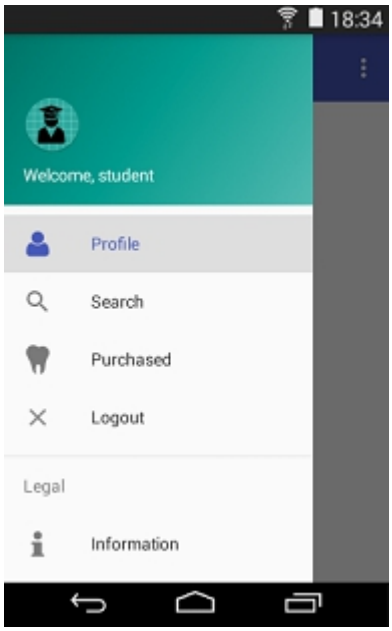


Figura 4.31. Interfaz del menú de estudiante.

4.2.3.2. Las opciones del menú (fragmentos iniciales)

Cada uno de los fragmentos indicados en el apartado anterior tendrán un comportamiento distinto. Para ello

Para los listados de piezas dentales y la obtención del perfil se establecerán escuchas permanentes con cada uno de los documentos asociados de la herramienta **Firestore** con el propósito de recibir en tiempo real los cambios realizados en sus parámetros.

En el caso de las búsquedas que puede realizar el usuario, para obtener un listado de elementos determinado, el fragmento deberá mostrar un formulario donde podrá seleccionar cada uno de los valores que la pieza debe contener y un botón de búsqueda que pulsará una vez haya definido lo que desee.

La pantalla de información simplemente mostrará el texto correspondiente a la ayuda y la información legal.

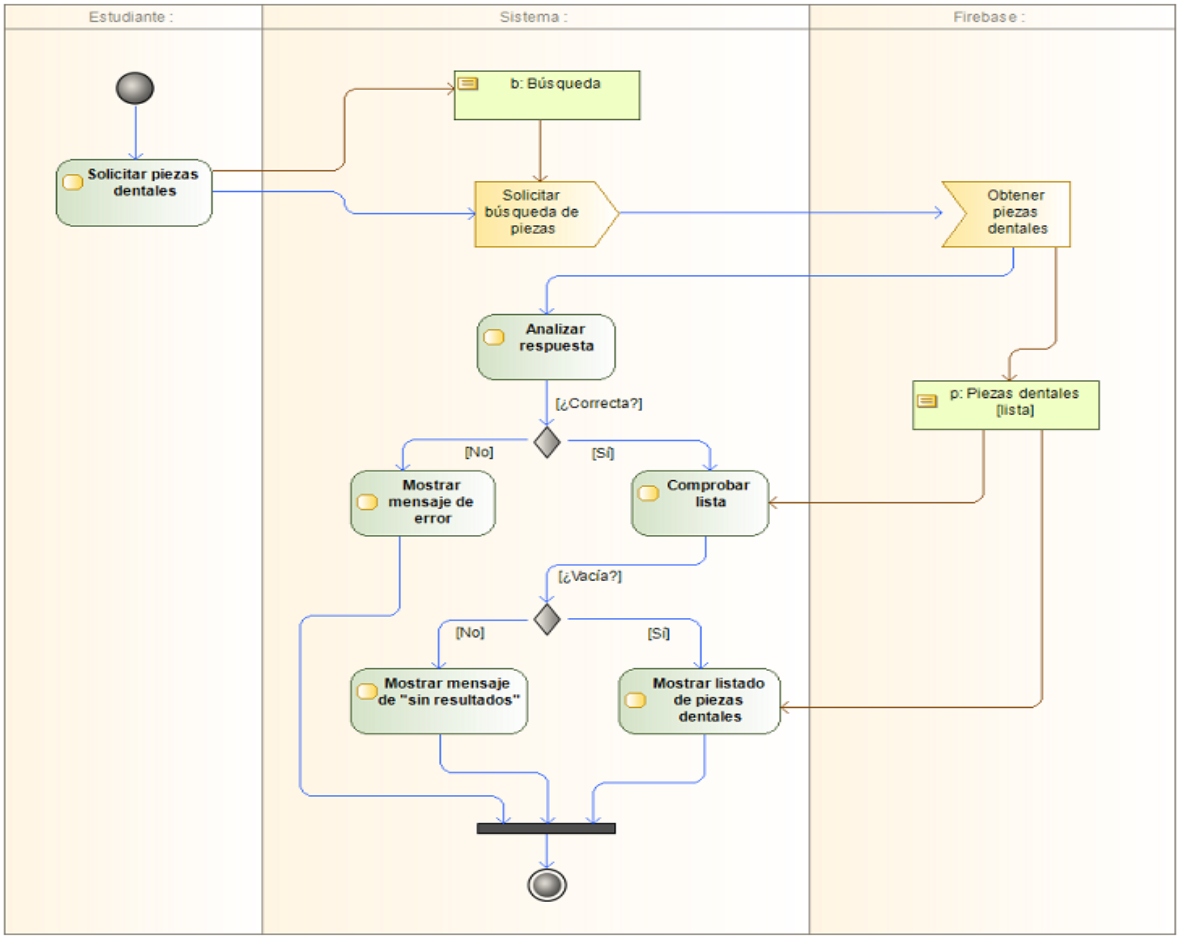


Figura 4.32. Diagrama para la obtención de piezas dentales.

4.2.3.2.1. Estructura implementada

Para la creación de los fragmentos que contengan algún listado de piezas dentales se requiere del acceso a la clase *ListData* para el almacenamiento de los elementos descargados así como la necesidad de implementar las escuchas a las interfaces programadas para permitir a la clase *MenuStudent* manejar los objetos seleccionados[1].

En el caso del fragmento con perfil será necesario el uso del objeto StudentUserData para almacenar y mostrar el contenido descargado.

Todos estos elementos serán buscados y recogidos durante la ejecución del método *onStart()*.

Para todos los fragmentos que contengan elementos descargados se ha habilitado el uso de un deslizador cuyo código interno sirve para destruir y volver a ejecutar el fragmento. Su propósito es que en caso de que haya habido algún problema en la recuperación de los datos se pueda volver a ejecutarlo deslizando la pantalla hacia abajo.

1 La edición no se realizará si el dispositivo no está conectado a la red. Solución obtenida de [sta 01].

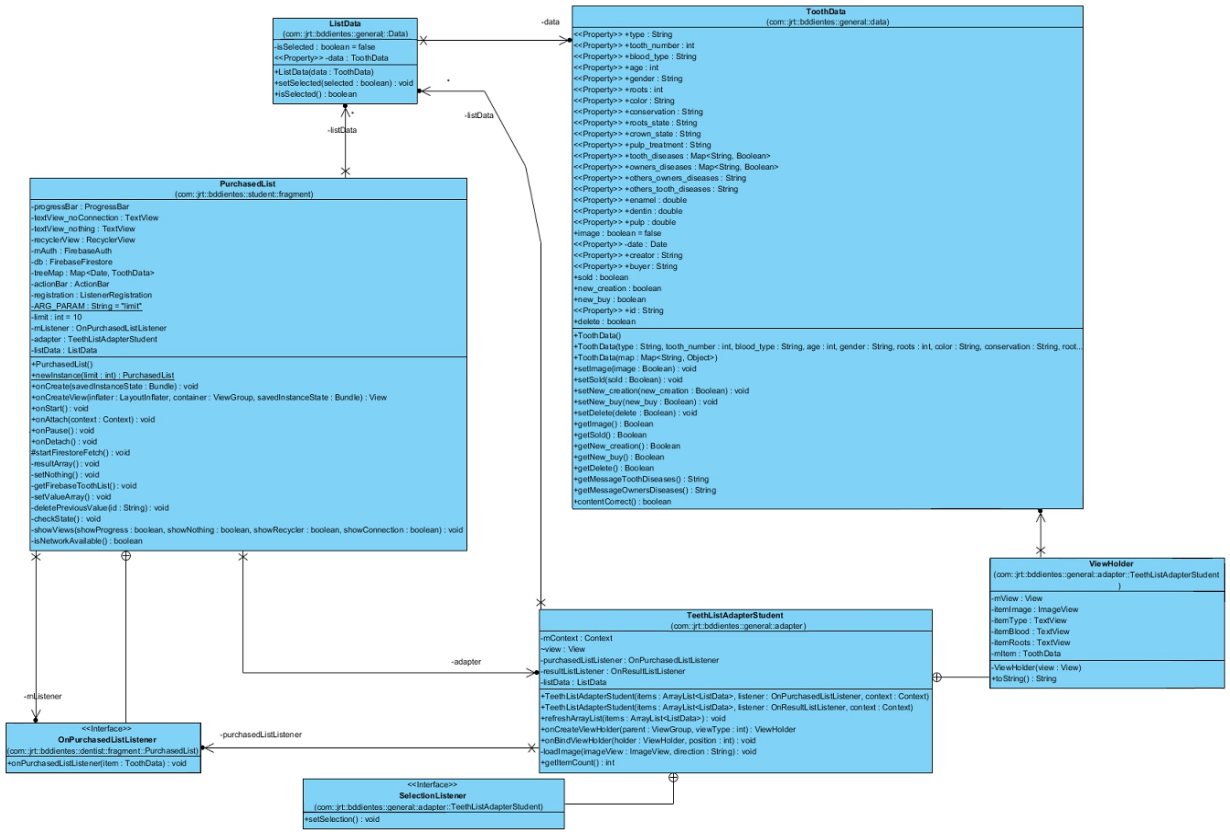


Figura 4.33. Estructura de la opción *PurchasedList*.

Los diferentes fragmentos diseñados son los siguientes:

Perfil de usuario (Profile). (Figura 4.34.)

- Es el primer y último fragmento que se le muestra al usuario dentro del menú.
- Se mostrará la información de perfil asociada al usuario tras una realizar una consulta de sus datos en la herramienta **Firestore**. Esta se realizará en la ejecución de *onStart()*.
- La escucha implementada será permanente y estará asociada al ciclo de vida de la actividad invocadora.
- El valor de la variable *search_limit* del perfil será enviada a la actividad invocadora.
- Una vez obtenidos los datos se asignarán a los elementos visuales de la interfaz.
- Su escucha siempre está en funcionamiento y su vida está asociado a la de la actividad invocante.

Búsqueda de piezas dentales (Search). (Figura 4.35.)

- Muestra un formulario que permite definir qué parámetros se deben buscar en la base de datos.
- Los valores de algunas opciones estarán limitados por los elementos seleccionados previamente.
  Por ejemplo: del tipo de dentadura dependerán el número de pieza dental y los tratamientos pulpares y del número de pieza dental dependerán las raíces máximas que se pueden seleccionar.
- También se permite realizar búsquedas sin especificar ningún valor en el formulario.

- No tiene implementada ninguna escucha ya que este fragmento sólo es de selección de parámetros.
- Los diálogos asociados son los siguientes[1]:
  - DialogFragmentTeeth_permanent_fragment..
  - DialogFragmentTeeth_deciduous_fragment..
  - DialogFragmentColor_fragment.
  - DialogFragmentInt_fragment..
  - DialogFragmentDouble_fragment..
  - DialogFragmentList_fragment..

Listado de piezas dentales seleccionadas (PurchasedList). (Figura 4.36.)

- Se realizará una búsqueda de piezas dentales donde el valor de la variable *buy* sea *true* y cuyo solicitante sea igual al usuario actual (la variable *buyer* debe contener el mismo identificador asociado al usuario actual).
- Para almacenar y mostrar cada uno de los elementos recuperados se hará uso de un lista de objetos *ListData*.
- Debido a la necesidad de recoger las alteraciones en los datos mostrados, la escucha implementada para los datos buscados es permanente. La vida de esta está asociada a la del fragmento.

Información (Information) [2].

- Se limita a mostrar la ayuda sobre la aplicación, derechos de autor e información legal.

4.2.3.2.2. Interfaz de usuario

La única interfaz distinta a las de la versión del dentista es la de búsqueda. En ella se ha incorporado un formulario donde se puede indicar el valor de la variable que se desee y un botón flotante para que ejecuta la llamada a la interfaz implementada en la actividad menú.

Para los diseños para el perfil y el listado de piezas dentales solicitadas se han implementado una serie de “pantallas” dentro de las interfaces de cada fragmento para informar al usuario sobre ciertos eventos:

- La falta de conexión del dispositivo.
- La no existencia de elementos dentro de los parámetros.

1 Para obtener más información sobre los diálogos ir a la sección Diálogos de este capítulo.
2 Debido a su diseño sencillo no se mostrará la interfaz asociada.

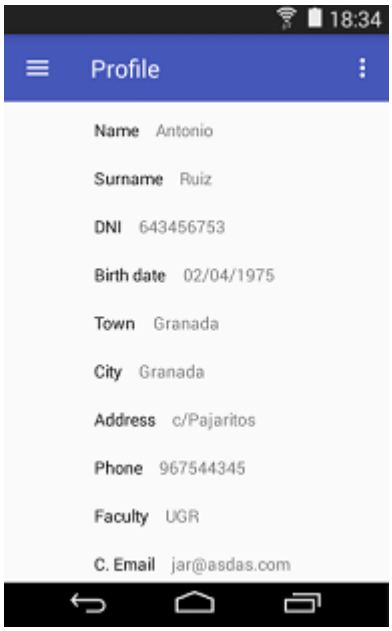


Figura 4.34. Perfil de usuario.

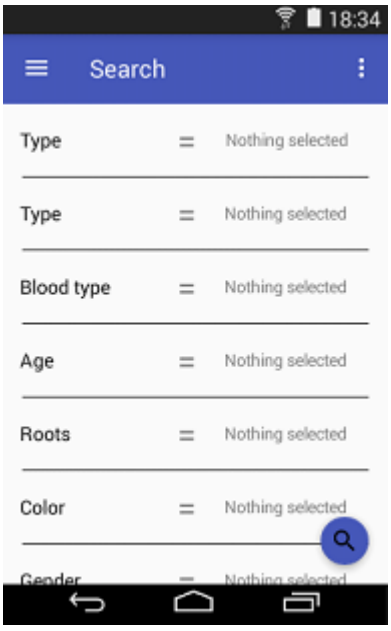


Figura 4.35. Búsqueda de piezas.

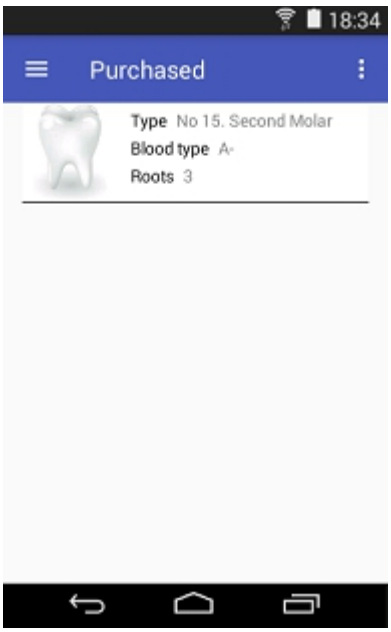


Figura 4.36. Listado de piezas solicitadas.

4.2.3.3. La búsqueda (Fragmentos secundarios)

Este es el paso intermedio entre la opción de buscar del menú y el fragmento que muestra la información detallada de una pieza dental seleccionada.

Este fragmento tomará el objeto *SearchData* enviada desde el menú (recogidos del fragmento de búsqueda) e irá comprobando si existe cada uno de sus campos para realizar la búsqueda correspondiente.

Una vez seleccionados todos elementos con los que se va a realizar la búsqueda, el procedimiento es igual que el se sigue cuando se realiza una búsqueda predeterminada en cualquier otro fragmento con un listado, la única diferencia radica en el hecho de que en este caso, al manejar tantas opciones de búsqueda distintas, la solicitud enviada a **Firestore** será mucho más compleja.

El mayor problema encontrado y por lo que se ha optado por permitir sólo búsquedas sencillas (igual que) es debido a la complejidad que entraña el desarrollarlas cuando se realizan con elementos complejos (mayor o menor que).
Cuando una búsqueda contiene uno de estos elementos se obliga a que cada una de las combinaciones posibles se incluyan en el servidor de **Firestore** y a que en la aplicación se definan dentro de una misma línea de código concatenada. En el caso de búsquedas con muy pocos elementos se podría desarrollar sin ninguna dificultad, pero, en el caso de esta aplicación, en este apartado se especifican más de 20 elementos por lo que habría que definir cada una de las más de 40000 posibilidades posibles.

4.2.3.3.1. Interfaz de usuario

La interfaz implementada para obtener los resultados de la búsqueda es la misma que se utiliza para cualquier listado de la aplicación.

La única diferencia puede encontrase es que la barra de progreso que se muestra mientras se obtienen los resultados del servidor puede permanecer en pantalla durante más tiempo que en otros fragmentos. Se debe a que los tiempos de búsqueda del servidor pueden verse afectados en función de la cantidad de elementos indicados o de los datos que haya en el servidor[1].

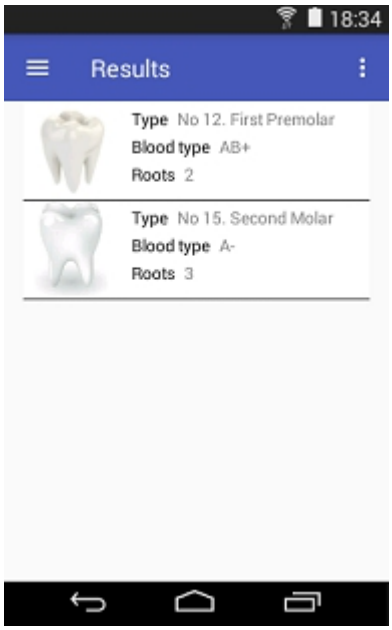


Figura 4.37. Interfaz del listado de piezas dentales buscadas.

También se han implementado una serie de "pantallas" dentro de su interfaz para informar al usuario sobre ciertos eventos:

- La falta de conexión del dispositivo.
- La no existencia de elementos dentro de los parámetros.

4.2.3.4. Información detallada (Fragmentos secundarios)

Para mostrar la información detallada de la pieza dental obtenida se ha creado dos fragmentos con pequeñas variaciones entre ellas.

Las relaciones entre los fragmentos previos y los actuales son las siguientes:

- *SearchList → SearchListDetails*
- *PurchasedList → PurchasedListDetails*

1 Se puede consultar los tiempos de duración de cada búsqueda en el Capítulo 5.

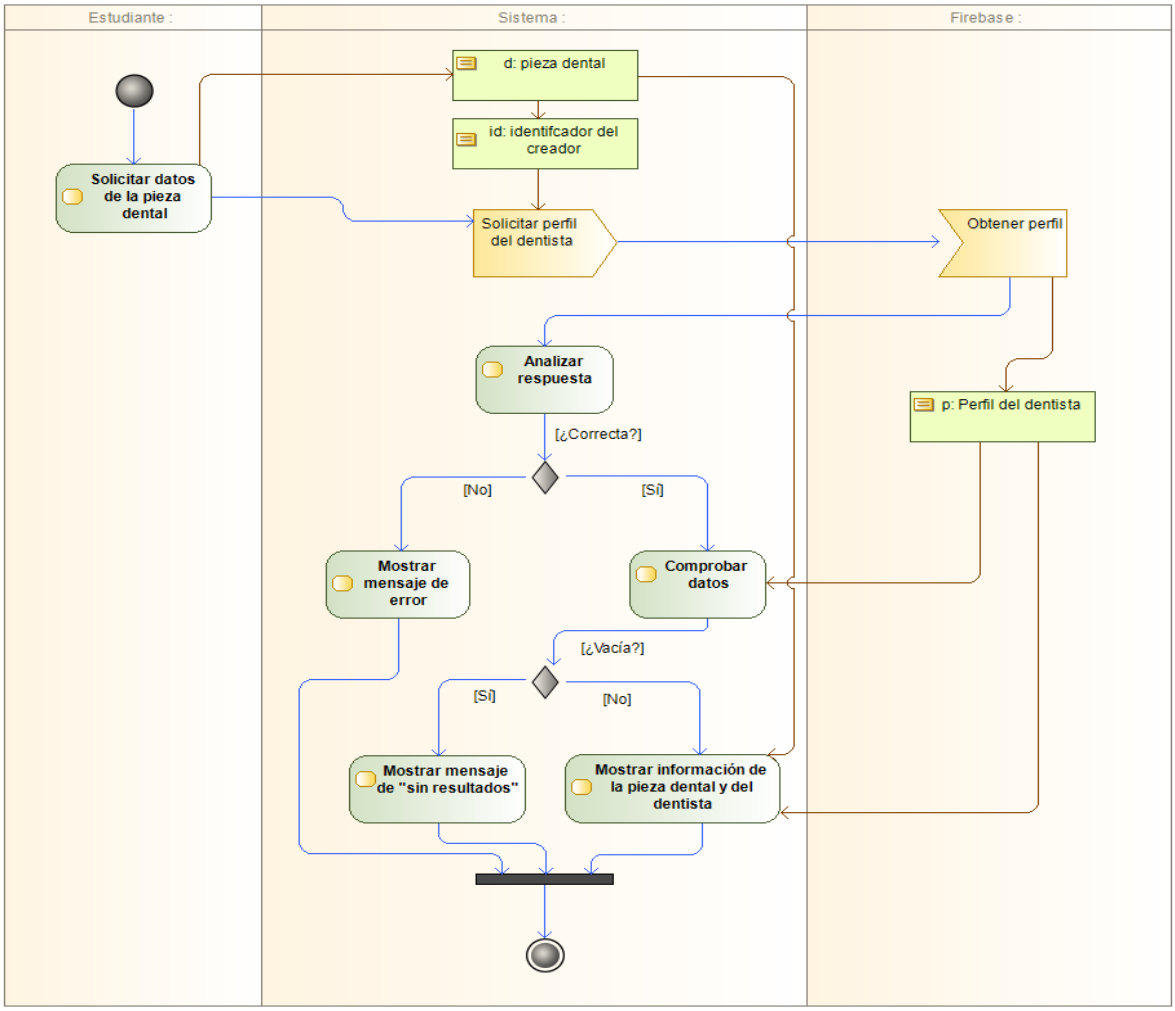


Figura 4.38. Diagrama para la obtención de los datos de una pieza.

Como es necesario mostrar también datos del perfil del dentista será necesario realizar una petición a la herramienta **Firestore** para recuperarlo y almacenarlo en un objeto *DentistUserData*.
La dirección en el servidor de su perfil vendrá en el contenido de la variable *creator* del objeto *ToothData.*

4.2.3.4.1. La estructura implementada

Una vez se haya descargado el perfil del dentista, se mostrarán los parámetros más relevantes para el usuario. Si se decide por pulsar sobre la dirección se ejecutará un diálogo donde se muestra su localización exacta en un mapa. Esto se ha implementado mediante las herramientas **Geocoder** y **GoogleMaps**. El primero permite obtener de una dirección escrita sus coordenadas geográficas y el segundo muestra la posición de esas coordenadas en un mapa geográfico[1].

Ambos fragmentos poseen el mismo diseño, pero en *SearchListDetails* se han incorporado una opción nueva.

1 Para más información consultar el aparatado *Uso de Geocode y Google maps* del Anexo.

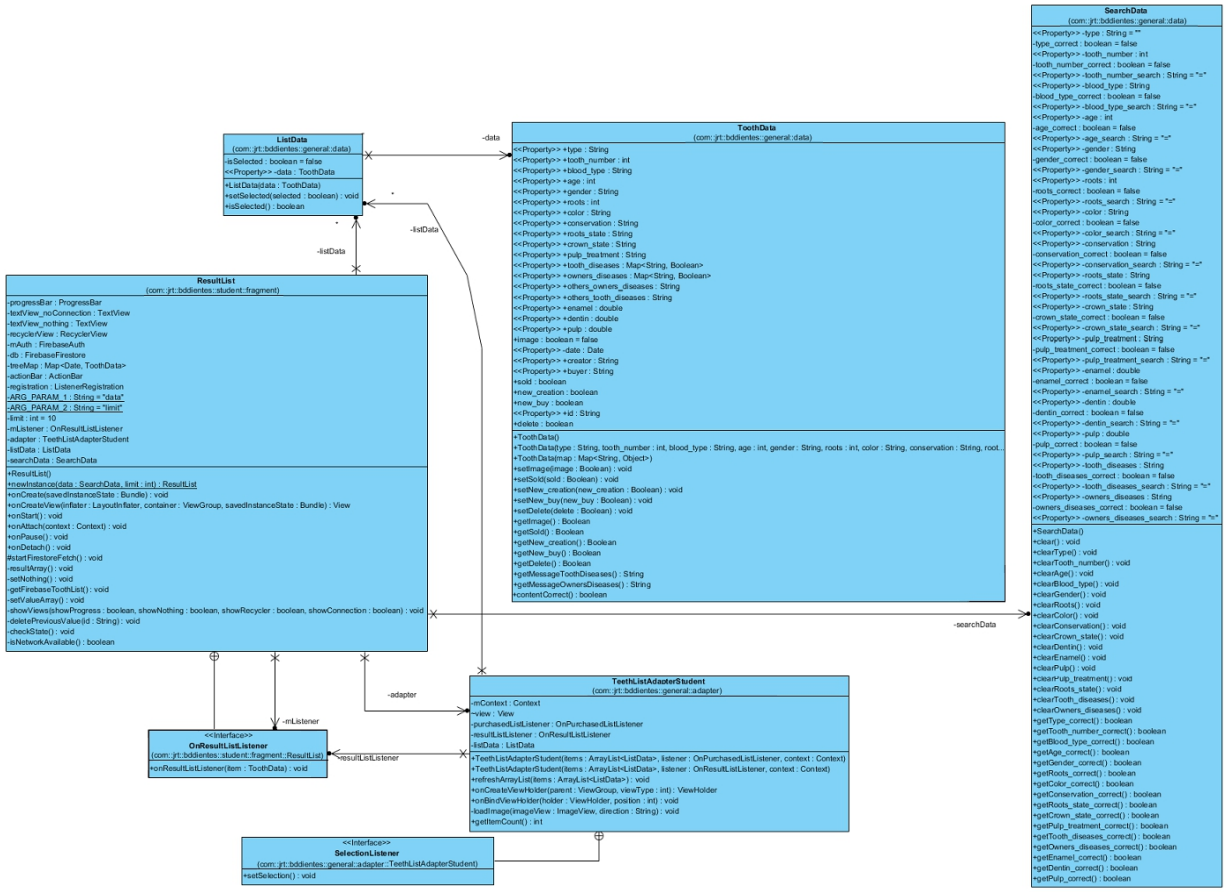


Figura 4.39. Estructura para *ResultListDetails*.

SearchListDetails

- Permite al usuario solicitar la pieza dental. Para ello se descargará de nuevo la pieza dental y se comprobará que sigue sin haber sido solicitada. Una vez realizada la comprobación, se lanzará una petición de actualización al servidor para indicar que la pieza está solicitada por el usuario actual.[1]
- El procedimiento para la solicitud de la pieza sólo podrá ejecutarse cuando el perfil del dentista se haya descargado correctamente y si su variable *delete* está a *false*.
- Los errores producidos durante la solicitud serán notificados al usuario y si el error que corresponde con la de actualización a **Firestore** falla dos o más veces se mostrará un mensaje donde se insta a contactar con el administrador.

4.2.3.4.2. Interfaz de usuario

Para ambos fragmentos se usa la misma interfaz, pero sólo en el caso de *SearchListDetails* se mostrará el botón de solicitud (figura 4.41).

1 La solicitud no se realizará si el dispositivo no está conectado a la red. Solución obtenida de [sta 01].

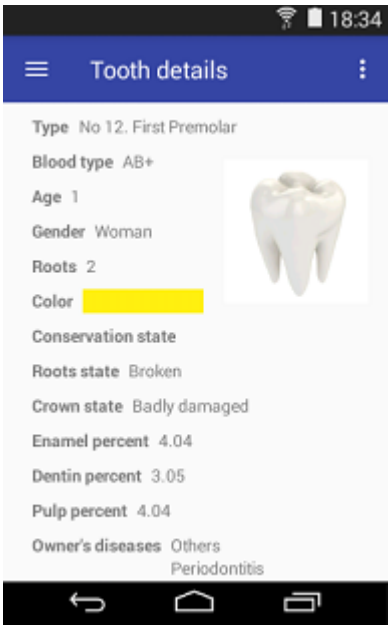


Figura 40. Parte superior.

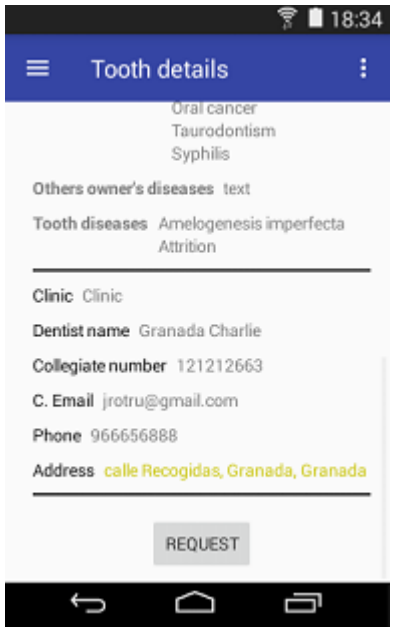


Figura 41. Parte inferior.

4.2.4. Editar perfil

Esta opción es común a ambas versiones de la aplicación por lo que sólo se explicará en este apartado.

Para poder alterar el perfil antes será necesario mostrar los datos contenidos en un formulario semejante al de registro. Con el propósito de que el usuario entienda que está en una sección distinta a la del menú, esta ventana se creará mediante una actividad.
Debido a la posibilidad de que el usuario pueda acceder a esta actividad antes de haber descargado los datos de perfil desde el fragmento perfil, al inicializarla se realizará la descarga de datos del servidor **Firestore**.

Lo primero que se debe hacer para permitir la edición es comprobar que el dato escrito es correcto. En el caso de que lo sea se establecerá una comunicación con la herramienta del servidor **Firestore** para solicitar alterar el valor de la variable indicada.

Para evitar enviar grandes cantidades de datos al servidores las alteraciones de parámetros se añadirán a un objeto *Map* que contendrá el nombre de la variable y su nuevo valor.

Por último, es esta ventana donde se mostrará la posibilidad de darse de baja de la aplicación lo que enviará una solicitud de eliminación de cuenta a la herramienta **Authentication** y se devolverá al usuario a la ventana de inicio de sesión.

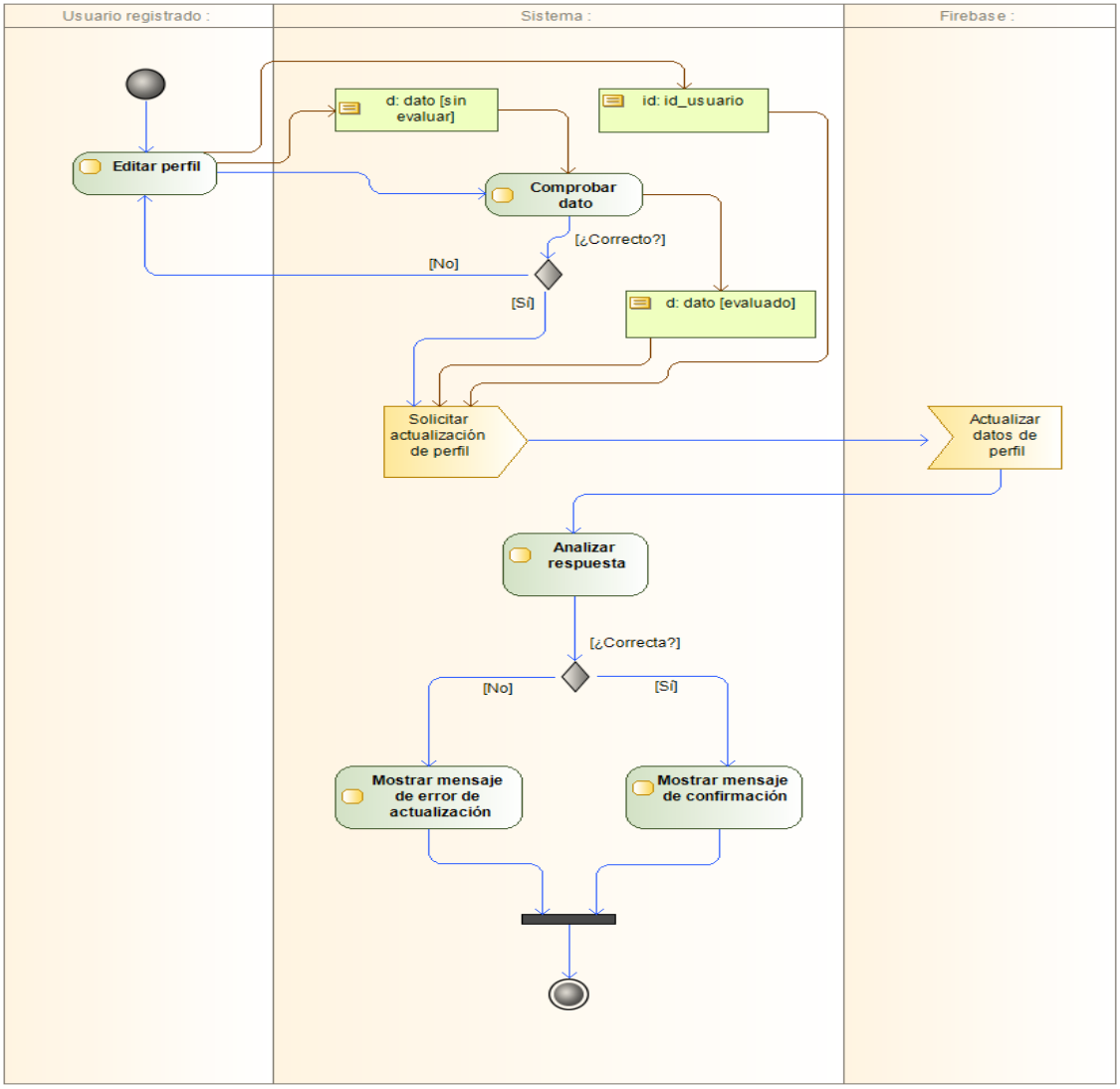


Figura 4.42. Diagrama de edición de perfil.

4.2.4.1. La estructura implementada

Esta actividad realizará las siguientes funciones:

- Descargará los datos de perfil al iniciarse en un objeto *DentistUserData o StudentUSerData.*
- Los valores que contengan se asignarán a cada uno de los apartados del formulario.
- Lanzará los diálogos oportunos por cada sección.
- Analizará los datos introducidos y, en caso de ser correctos, solicitará la actualización.
- En el caso de que el usuario solicite darse de baja, se mostrará un mensaje de confirmación que deberá aceptar. Una vez confirmado se producirá la baja y se devolverá el usuario a la pantalla de inicio de sesión.

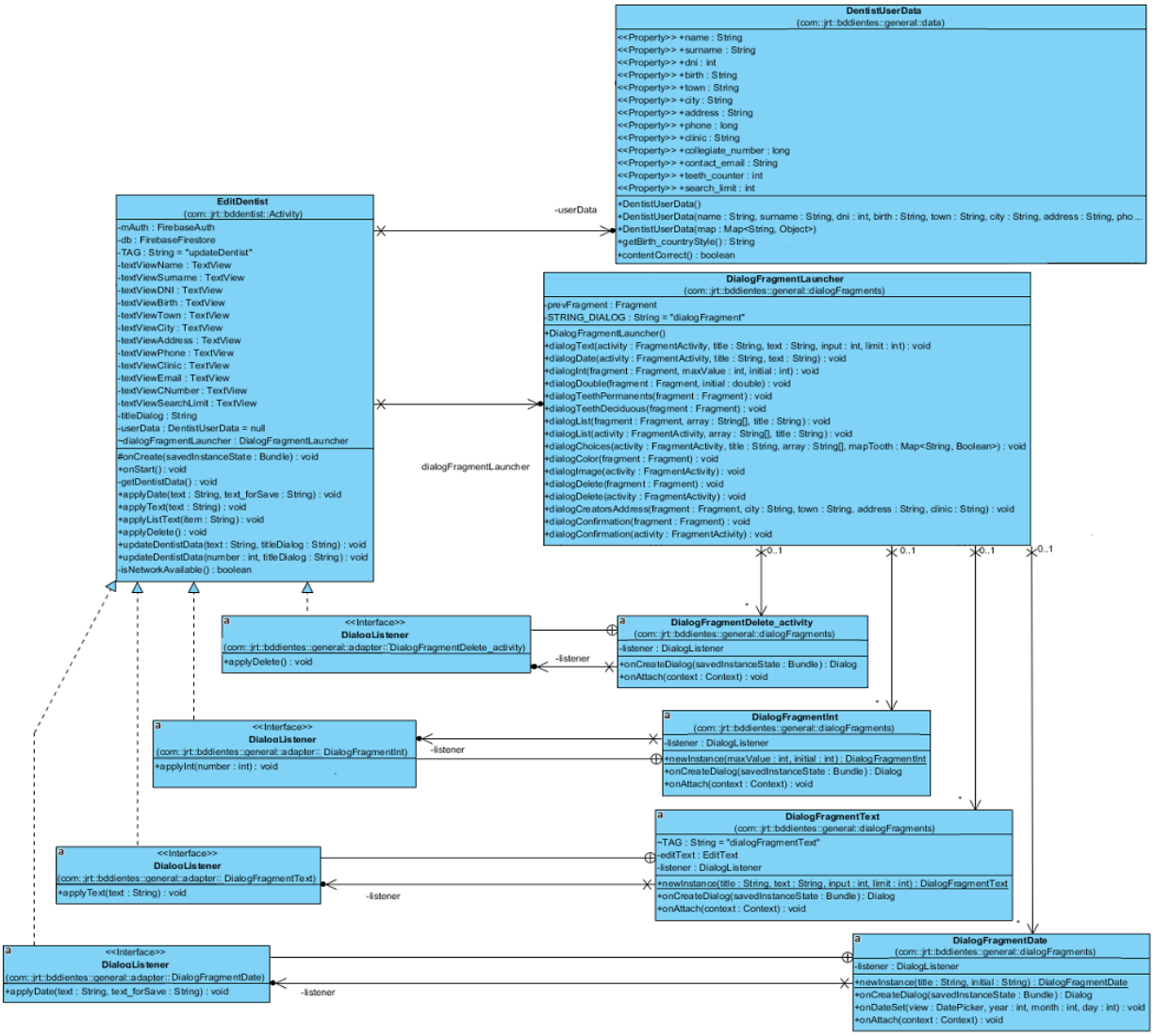


Figura 4.43. Estructura de edición de perfil para el dentista.[1]

En este aparatado se usan los siguientes diálogos[2]:

- DialogFragmentInt_activity.
- DialogFragmentText_activity.
- DialogFragmentDate_activity.
- DialogFragmentDelete_activity.

1 Este es el único diagrama donde se mostrará el funcionamiento de las interfaces y sus relaciones con la clase que controladora.
Para ver el modelo del estudiante ir al Anexo.

2 Para obtener más información sobre los diálogos ir a la sección *Diálogos* de este capítulo.

4.2.4.2. Interfaz de usuario

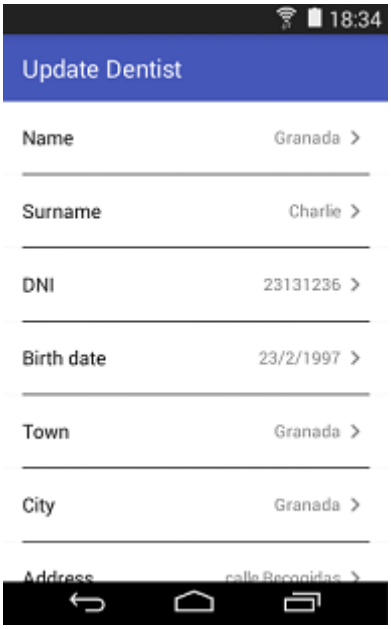


Figura 4.44. Interfaz para la edición del dentista.

Para esta interfaz se ha usado el mismo diseño que en el registro de usuario. La diferencia radica en que esta cargará los valores del perfil recuperados del servidor.

4.2.5. Diálogos

Una de las partes más importantes durante el desarrollo de la aplicación han sido los diálogos. Gracias a ellos se ha permitido tanto gestionar toda la información introducida mediante teclado como reducir el código total del proyecto.

Como se puede observar en la figura 4.45, se han diseñado múltiples diálogos que permiten gestionar cualquier tipo de entrada o mensaje necesario en la aplicación. La gran mayoría de ellos están duplicados con el objetivo de que exista una versión para ejecutar en las actividades y otra para los fragmentos[1].

Con el objetivo de conseguir que todos los diálogos diseñados estén agrupados y se pueda acceder a ellos de una manera cómoda y sencilla se ha creado una clase *DialogLauncher* desde la cual se lanza cada uno de los diálogos en función del método y los parámetros indicados.

Todos los diálogos tienen implementadas distintas interfaces para enviar el resultado seleccionado por el usuario a la clase que los ha ejecutado (esta deberá tenerlos implementados para que funcionen).
El único que no lo tiene implementada una interfaz es *DialogFragmentImage* debido a que cuando se selecciona uno de los elementos de la lista que muestra (cámara y carpeta multimedia) ejecutan

1 Al ser tan complicado incluir en el diagrama todos los diálogos, sólo se muestra una genérica de cada uno.

una nueva actividad que tiene como objetivo conseguir una imagen. Para obtener esa respuesta es necesario incluir en el código de la actividad ejecutante el método *onActivityResult()* que es donde se enviará el elemento seleccionado.

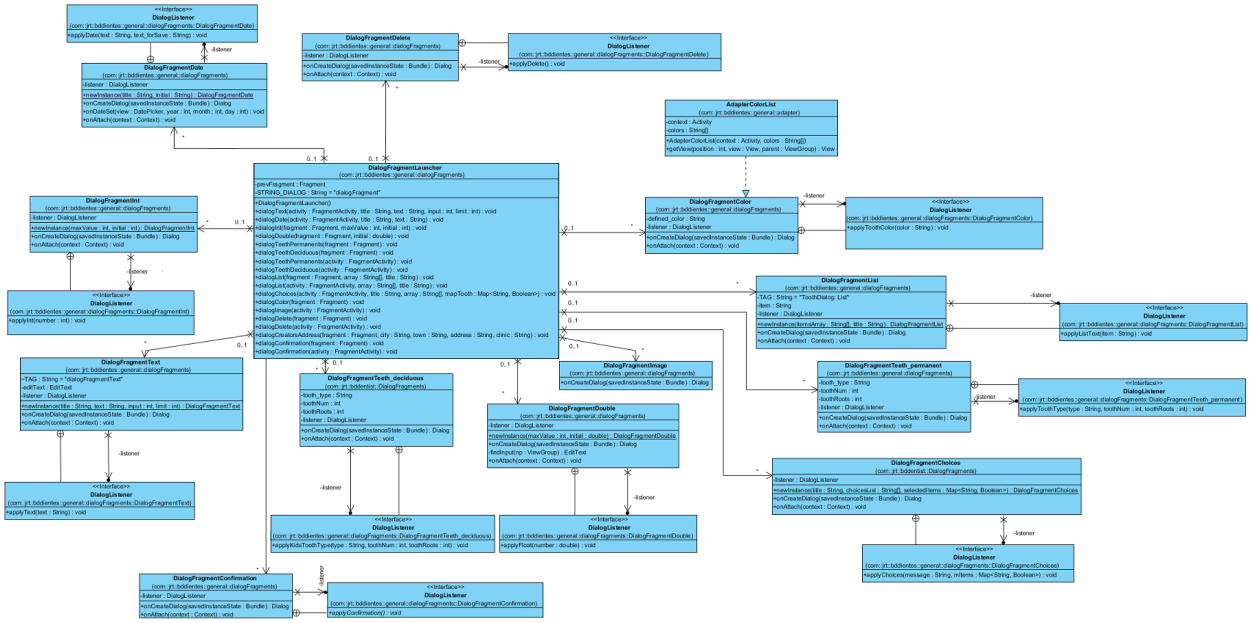


Figura 4.45. Estructura de los diálogos.

Los distintos diálogos creados son:

DialogFragmentTeeth_deciduous (figura 4.56)

- Muestra todos las piezas dentales temporales disponibles para seleccionar.
- Usa una interfaz que muestra una dentadura para piezas dentales temporales a las cuales se le han ido colocando botones invisibles. Cuando el usuario pulse una de ellas se sustituirá la imagen por una nueva donde la pieza dental estará marcada con el color amarillo.
- Para más detalles también se muestra una pequeña cantidad de información sobre la pieza seleccionada.
- Se devolverá los datos relativos al número de la pieza, la cadena de texto a mostrar y el número máximo de raíces que puede tener.

DialogFragmentTeeth_permanent (figura 4.57)

- Muestra todos las piezas dentales permanentes disponibles para seleccionar.
- Usa una interfaz que muestra una dentadura para piezas dentales permanentes a las cuales se le han ido colocando botones invisibles. Cuando el usuario pulse una de ellas se sustituirá la imagen por una nueva donde la pieza dental estará marcada con el color amarillo.
- Para más detalles también se muestra una pequeña cantidad de información sobre la pieza seleccionada.
- Se devolverá los datos relativos al número de la pieza, la cadena de texto a mostrar y el número máximo de raíces que puede tener.

DialogFragmentList (figura 4.48)

- Recibe una lista de cadenas de texto que mostrará al usuario.
- Sólo uno de los elementos podrá ser seleccionado y será el que se devuelva.

DialogFragmentColor (figura 4.51)
- Muestra una lista con una serie de colores.
- Para mostrar el color de cada uno de los elementos ha sido necesario crear un adaptador que permita su gestión.
- Devuelve una cadena de texto con el valor seleccionado.

DialogFragmentDate.
- Muestra un selector de fechas.
- La fecha seleccionada se devolverá en una cadena de texto cuya estructura es día/mes/año.

DialogFragmentDelete (figura 4.47)
- Muestra un mensaje para confirmar una eliminación.
- Se ha implementado sólo una interfaz para recoger los casos de confirmación.
- El propósito es tener un diálogo exclusivopara eliminaciones tanto de perfil como de piezas dentales.

DialogFragmentInt (figura 4.52)
- Muestra un selector de números enteros.
- Permite introducir el número por teclado.

DialogFragmentDouble (figura 4.53)
- Muestra dos selectores de números: uno para la parte entera y otro para la decimal.
- Debido a problemas con la parte decimal ha sido necesario diseñar una cadena de texto para indicar los valores posibles.
- Devuelve el número completo tras concatenar cada parte.
- Permite indicar los números mediante teclado[1].

DialogFragmentChoices (figura 4.50)
- Se muestra una lista dada y se permite seleccionar de múltiples elementos de ella.
- Recibe el *Map* con los objetos a mostrar y su estado de selección.
- Devuelve un *Map* con el contenido y una cadena de texto con los valores seleccionados.

DialogFragmentConfirmation (figura 4.46)
- Muestra un mensaje que podrá ser aceptado o no.
- Tiene implementados también las opciones de cancelar y desechar el diálogo lo que permite tener un mayor control de las acciones del usuario.

DialogFragmentText (figura 4.49)
- Muestra un ventana con una caja para la inserción de texto.
- El formato de teclado mostrado variará en función del tipo que se indique al crearla.
- Debido a que los distintos teclados para texto permiten la entrada de todo tipo de caracteres

1 Para solucionar el problema del teclado cuando a un selector se le pasa una cadena de texto ha sido necesario recurrir a la solución propuesta en [sta 03].

(alfabéticos, número y símbolos) ha sido necesario diseñar un filtro donde se especifiquen cuáles de ellos pueden ser aceptados permitiendo sólo aquellos que sean alfabéticos[1].

DialogMap (figura 4.58)

- Muestra la posición en un mapa de una dirección dada.
- Se ha usado **Geocoder** para traducir una dirección textual a sus coordenadas geográficas y **Google maps** para situar dichas coordenadas en un el mapa[2].
- Debido a la delicadeza de mostrar un mapa en un diálogo (el más mínimo error en el posicionamiento puede producir una excepción) se ha añadido código necesario para que en el caso de que no se encuentre ninguna dirección se busque por una dirección limitada a ciudad y población o, si todo falla, mostrar simplemente el mapa estándar.

4.2.5.1. Interfaces diseñadas

En este apartado se muestran algunos ejemplos de los distintos diálogos implementados.

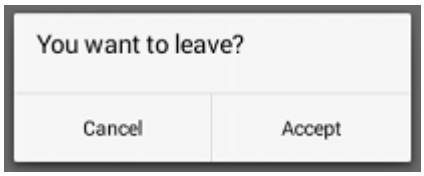


Figura 4.46. Diálogo de confirmación.

Figura 4.47. Diálogo de eliminación.

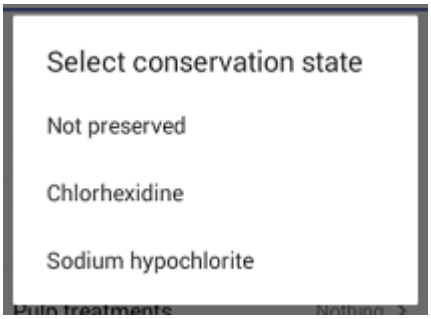


Figura 4.48. Diálogo para listas.

Figura 4.49. Diálogo de escritura.

1 Este problema se ha solucionado mediante la solución propuesta en [sta 02].
2 Para saber cómo realizar su instalación en una aplicación Android consultar el apartado *Uso de Geocoder y Google maps* del Anexo.

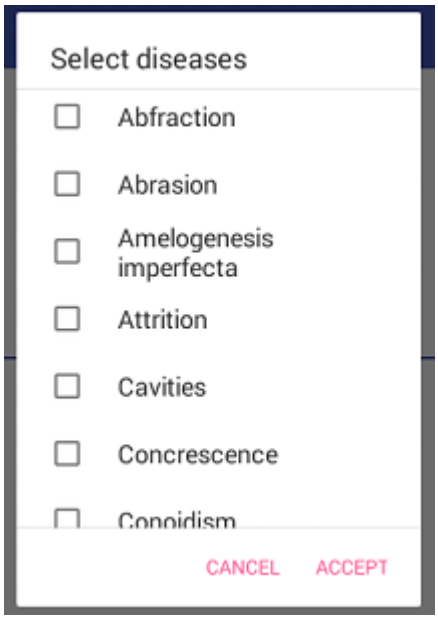


Figura 4.50. Diálogo de elección múltiple.

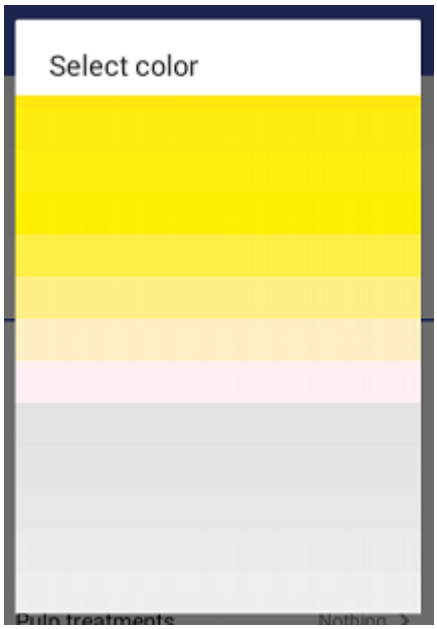


Figura 4.51. Diálogo de listado de colores.

Figura 4.52 Diálogo numérico simple.

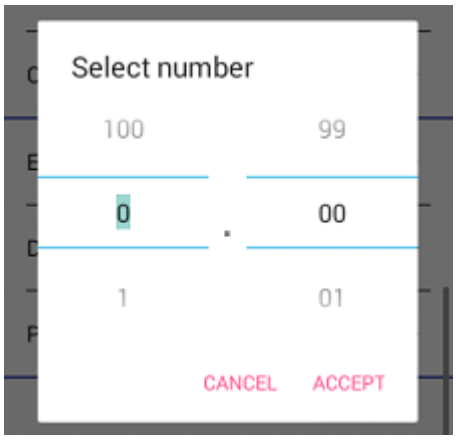


Figura 4.53. Diálogo numérico con decimales.

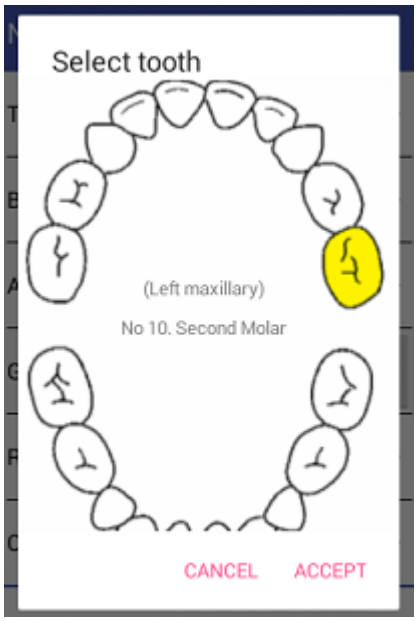


Figura 4.56. Diálogo de selección de piezas dentales temporales.

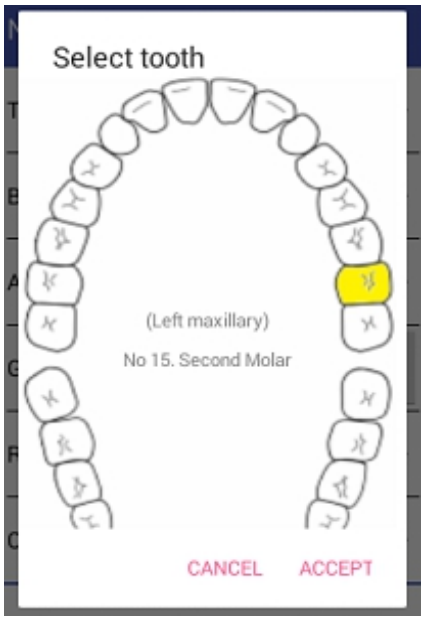


Figura 4.57.Diálogo de selección de piezas dentales permanentes.

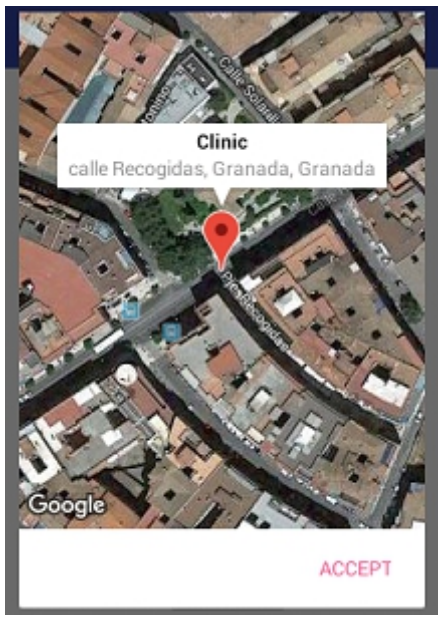


Figura 4.58. Diálogo que muestra la localización de una dirección en Google maps.

4.2.6. La barra de herramientas

Con el propósito de permitir al usuario acceder a las distintas partes de la aplicación se han diseñado diversos botones en la barra de herramientas.

Dependiendo de la interfaz (actividad o fragmento) en la que se encuentre el usuario la barra cambiará para mostrar las nuevas opciones a las que puede acceder o para mostrar algún tipo de información extra (cuando se selecciona algún objeto de un listado).

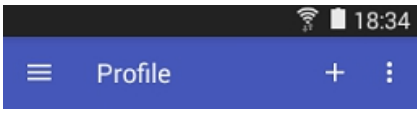


Figura 4.55.

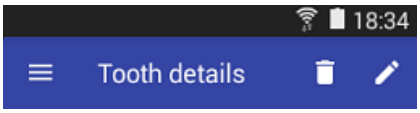


Figura 4.56.

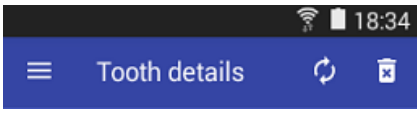


Figura 4.57.

Figura 4.58.

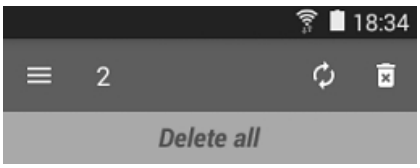


Figura 4.59.

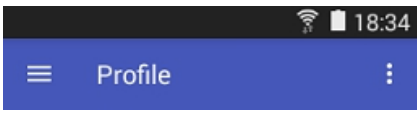


Figura 4.60.

Para la versión del dentista la barra muestra:

- Figura 55: opciones para incluir una nueva pieza dental o acceder a la edición del perfil.
  Esta es la barra de herramientas por defecto.

- Figura 56: opciones para enviar a la papelera la pieza dental o acceder a la ventana de edición.
  Sólo para la lista de dientes creados y que no han sido solicitados.

- Figura 57: opciones para extraer de la papelera una pieza dental o eliminarla definitivamente del servidor.
  Sólo para la lista de dientes creados y en la papelera.

- Figura 58: la cantidad de elementos seleccionados de la lista y la opción de enviarlos a la papelera.
  Esta versión sólo aparece al seleccionar un elemento de la lista de dientes creados y que no han sido solicitados.

- Figura 59: muestra la selección de un objeto y la opción de restaurarlo de la papelera o borrarlo definitivamente del servidor. La opción de *Delete all* hace lo mismo que la anterior salvo por el hecho de que elimina todo el contenida de la papelera.
  Esta versión sólo aparece al seleccionar un elemento de la lista de dientes creados y en la papelera.

Para la versión del estudiante la barra muestra:

- Figura 60: muestra la opción de de acceder a la edición de perfil.

4.2.7. Los objetos de almacenamiento

Existen cuatro clases diseñadas para almacenar y leer los datos que nos devuelve el servidor. Cada uno de esos objetos tienen diseñados los métodos *get* y *set* necesarios para obtener y editar los objetos.

Para esta tarea se han diseñado cuatro clases:

DentistUserData.
- Almacena la información de perfil del dentista.
- Posee una función que permite asignar valores a las variables desde datos de un *Map*.
- La función *getBirth_countryStyle()* permite mostrar la fecha de nacimiento del usuario en función del idioma y país indicados en el dispositivo móvil.[1]
- También se ha creado un método para comprobar si todas sus variables tienen asignadas algún valor.

StudentUserData.
- Almacena la información de perfil del estudiante.
- Posee una función que permite asignar valores a las variables desde datos de un *Map*.
- La función *getBirth_countryStyle()* permite mostrar la fecha de nacimiento del usuario en función del idioma y país indicados en el dispositivo móvil.
- También se ha creado un método para comprobar si todas sus variables tienen asignadas algún valor.

ToothData.
- Almacena la información de la pieza dental.
- Posee una función para comprobar que todos sus campos tienen asignados algún valor.

ListData.
- Almacena un objeto *ToothData* y una variable booleana.
- Es utilizada para poder determinar qué piezas dentales de los listados han sido seleccionadas por el usuario.

SearchData.
- Almacena los datos para la búsqueda.
- Debido a que en un principio se consideró que las consultas pudiesen realizarse para valores iguales, menores o mayores se ha dejado implementado la posibilidad de incluir una cadena para especificar qué operación se aplica a cada valor.

1 Sólo cambia el diseño en el caso de que el país sea Estados Unidos y el idioma el inglés. Esto se ha realizado mediante el uso del objeto **Locale** y consultando los códigos ISO indicados en [inf 11].

Capítulo 5. Pruebas

En este capítulo se mostrarán los tiempos de ejecución para cada función implementada en **Cloud Functions** y aquellas herramientas de **Firebase** de las que hace uso el usuario de la aplicación móvil. Para ello se han realizado diferentes mediciones en distintas situaciones con el objetivo de encontrar cualquier tipo de variación conforme a ellas.

Un dato interesante a conocer es que si el servidor **Firebase** ha estado ocioso durante bastante tiempo al realizar una nueva petición el tiempo de respuesta puede llegar a alargarse hasta el doble de lo normal.

5.1. Rendimiento de Cloud Functions

Para esta prueba se han realizado varias mediciones del tiempo de ejecución de las funciones implementadas en **Cloud Functions**.
Un detalle importante es que las funciones que hacen uso de búsquedas dentro de las herramientas de almacenamiento se han visto muy poco afectadas por la cantidad de elementos incluidos en ellas.

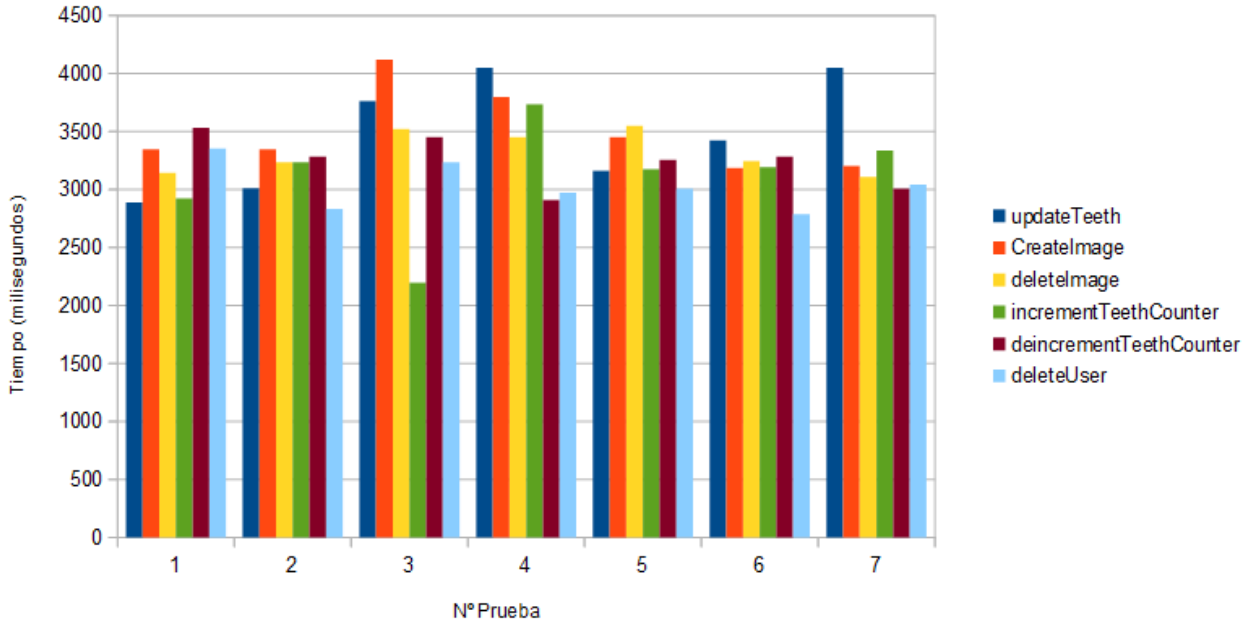


Figura 5.1. Mediciones de las funciones implementadas.

En las distintas mediciones realizas de la figura 5.1 las gráficas de las funciones muestran unos tiempos muy parecidos entre ellos, siendo *updateTeeth* y *createImage* las que mayores tiempos han obtenido. Esto certifica que los tiempos para aquellas funciones que no son extremadamente complejas suelen estar por debajo de los 60 segundos que Firebase tiene marcado como máximo

antes de cancelar la ejecución de la función.

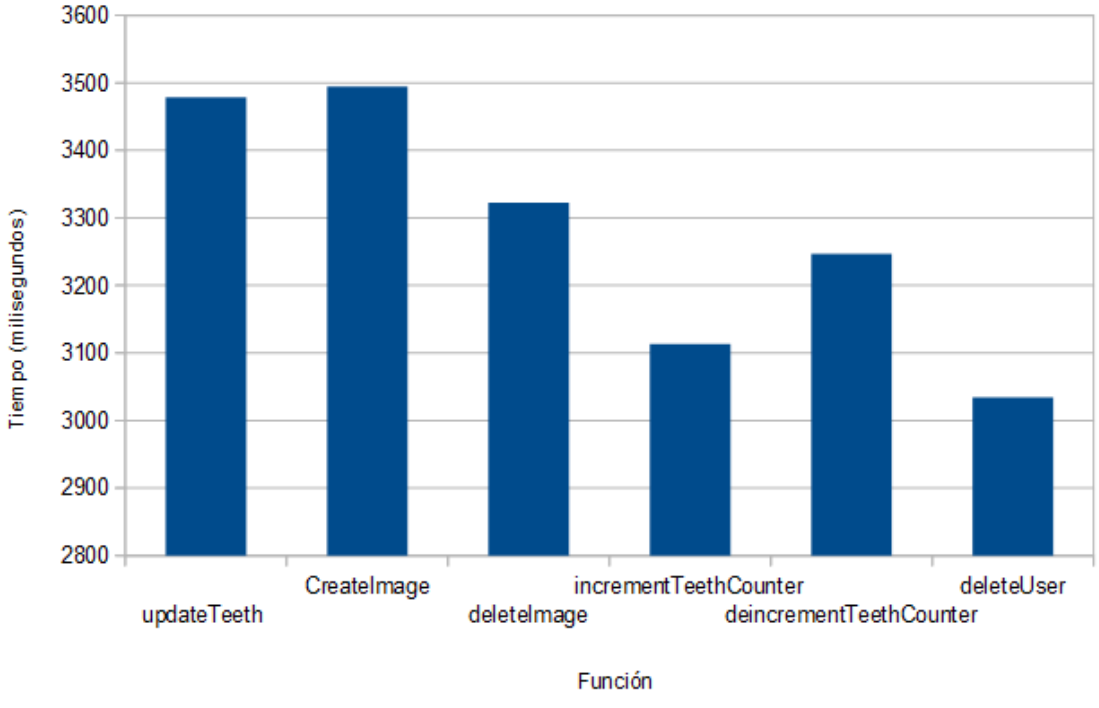


Figura 5.2. Medias de las funciones implementadas.

Gracias a la figura 5.2 se puede observar que los valores medios de cada una de las funciones anteriores está entre los 3000 y los 3500 milisegundos.

Aunque los valores que aquí se muestran son los comunes para cada una de las funciones se ha observado que en raras ocasiones cuando el servidor Firebase soporta altas cargas los tiempos aumentan. En esos momentos la ejecución de una función puede llegar a prolongarse hasta tres veces más de lo que tardaría con una carga normal.

5.2. Rendimiento de las peticiones

En este apartado se ha comprobado la velocidad de respuesta para las peticiones de los usuarios a las herramientas **Firestore** y **Storage**.

Para completar correctamente este análisis se han ido introduciendo piezas dentales (datos e imagen) al servidor hasta alcanzar la cifra de 2000 (número máximo de documentos dentro de una colección).Cada vez que se incluían 50 elementos nuevos se han ejecutado las operaciones que se muestran en la figura 5.3 comprobando sus tiempos de ejecución.

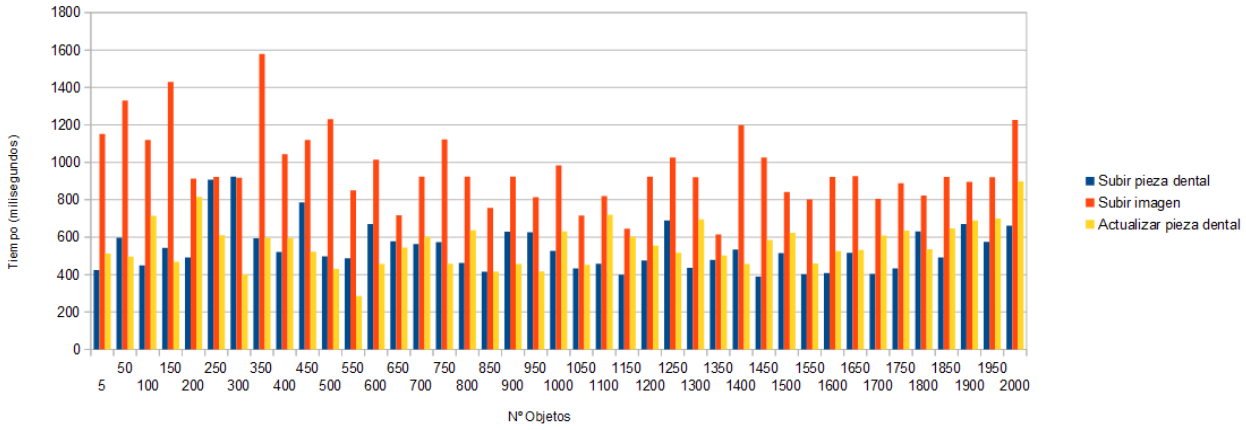


Figura 5.3. Tiempos de ejecución de las funciones de la derecha.

Como se puede observar, para las peticiones mostradas en la figura 5.3 los tiempos suelen ser más o menos constantes sin importar la cantidad de elementos introducidos en el servidor. Esto se debe a que todas ellas se ejecutan tras realizar la operación de creación, actualización o eliminación relacionada con su herramienta y, como reciben los datos previos y posteriores a esas ejecuciones, no deben hacer ninguna búsqueda en los datos del servidor.

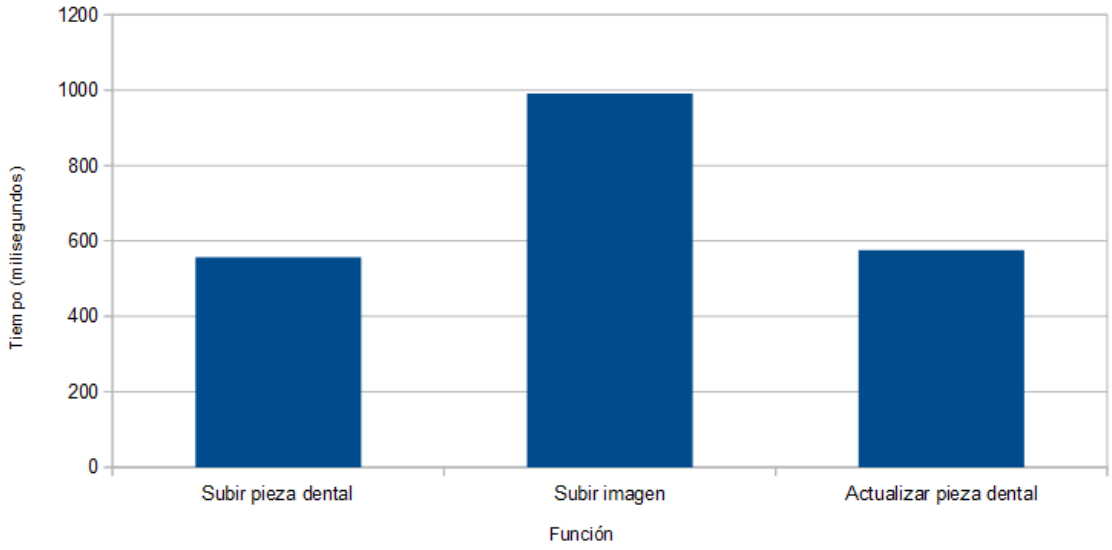


Figura 5.4. Medias de las funciones de la figura 5.3.

En la gráfica de las medias de la figura 5.4 las funciones *Subir pieza dental* y A*ctualizar pieza dental* (ambas para **Firestore**) poseen unos tiempos de ejecución muy parecidos entre sí y sólo *Subir imagen* ***(Storage)*** implica un mayor tiempo de ejecución debido al mayor peso de los datos tratados.

5.3. Rendimiento de las búsquedas

Al igual que en el apartado anterior, se han ido comprobando los tiempos de respuesta cada 50 elementos subidos al servidor. Las búsquedas analizadas implican a 4, 9 y 19 elementos cada una.

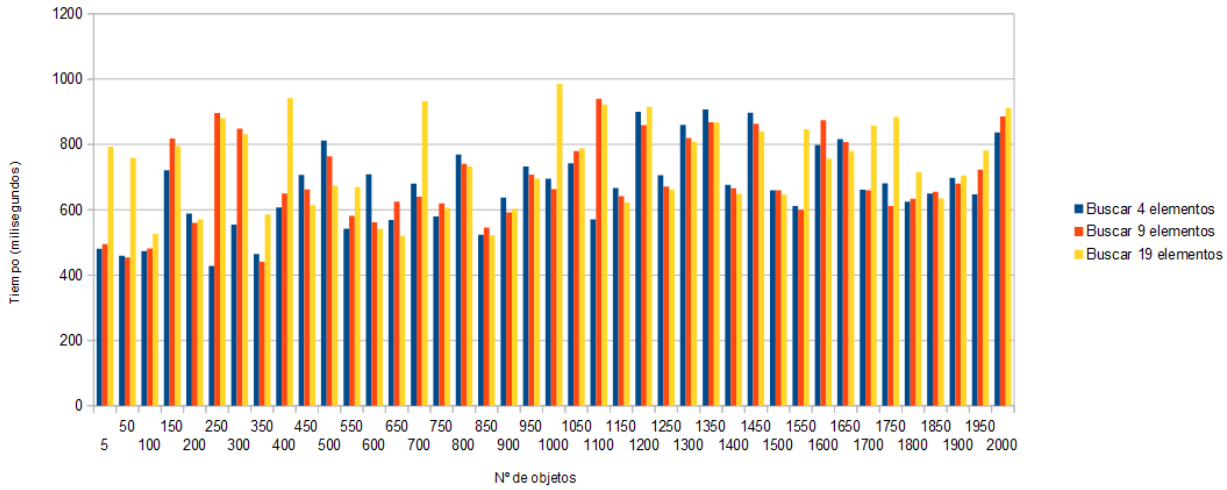


Figura 5.5. Tiempos de las distintas búsquedas.

En la gráfica se muestra cómo a pesar de mantener unos tiempos de respuesta semejantes en todo momento existe un ligero aumento del tiempo a medida que se introducen una mayor cantidad de elementos en el servidor. Este aumento no es muy notable debido a que la velocidad de respuesta de **Firebase** depende de la carga soportada o de si está sufriendo algún tipo de modificación interna.

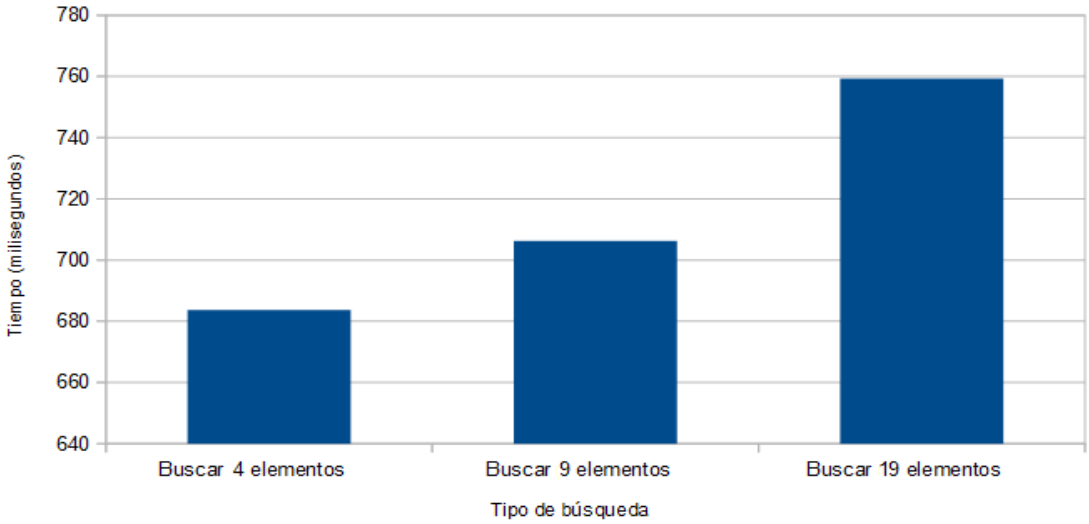


Figura 5.6. Medias para las distintas búsquedas.

En la figura 5.6 se puede observar cómo los tiempos de respuesta son mayores en función del

número de elementos buscados. Aún así, sus variaciones no son tan grandes como para limitar los elementos buscados dentro de la aplicación.

5.4. Rendimiento del inicio de sesión

Para este apartado se han realizado 3 medidas para 2, 10, 20 y 30 usuarios dentro de la herramienta Authentication.

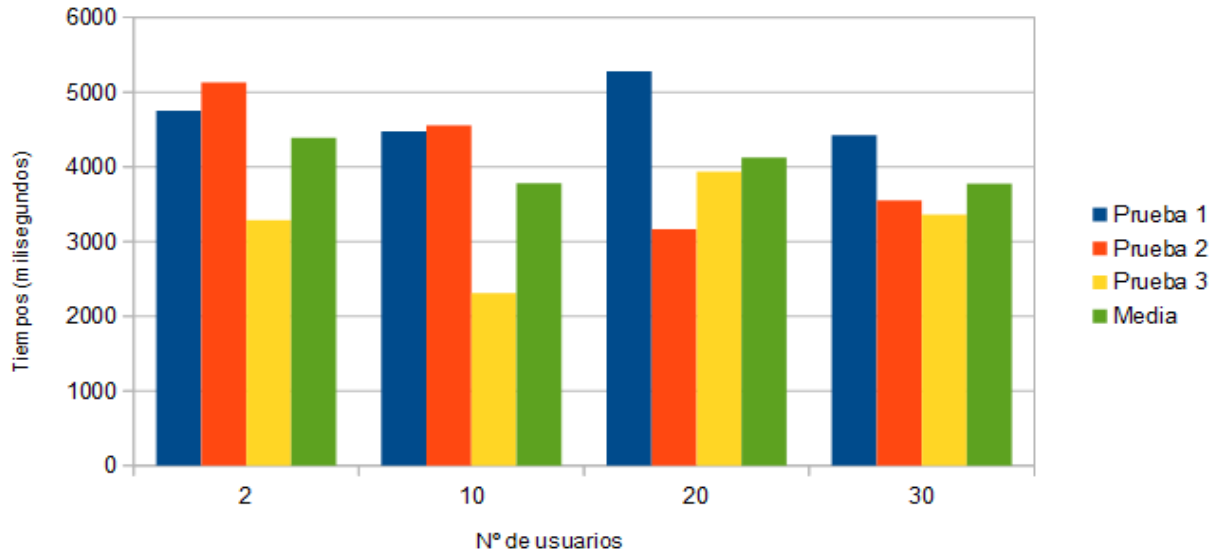


Figura 5.7. Tiempos de inicio de sesión.

En este caso, la herramienta **Authentication** no suele dar unos tiempos más o menos regulares. Aunque si se realizan varias pruebas y se calculan sus medias se pueden obtener unos resultados semejantes el problema reside en que en ocasiones un inicio de sesión puede tardar escaso tiempo mientras que otro puede alargarse demasiado.

Capítulo 6. Conclusiones y desarrollo futuro

En este capítulo se expondrán los motivos por los que se considera que se han alcanzado los objetivos marcados al comienzo del trabajo y se indicarán las mejoras que podrían realizarse en un futuro.

6.1. Conclusiones

Uno de los objetivos de este TFG era desarrollar la capacidad de atender a la petición de un cliente y poder alcanzar los objetivos que pidiese.
Se ha experimentado con lo que sería el trabajo síntesis y extracción de datos a través de diálogos con él y la dificultad que puede llegar a suponer trabajar con ellos si no están disponibles durante largos periodos de tiempo.

Uno de estos problemas se puede observar en el uso del prototipo de pieza dental que se diseñó en base a unos pocos datos obtenidos de la reunión inicial y a los conocimientos propios y ajenos que se consultaron.

A pesar de estas circunstancias, todos los elementos básicos que se solicitaron están incluidos en la aplicación por lo que considero que los objetivos marcados en este aspecto se han logrado.

En cuanto al desarrollo de la aplicación, la programación de aplicaciones móviles era un asunto que me despertaba gran interés y que tras pasar largas horas desarrollando diferentes versiones (primero una de prueba, luego una versión final basada en actividades y, finalmente, la versión final) creo que he alcanzado los conocimientos necesarios para el trabajo sobre ellas. Aún así, considero que aún me quedan numerosas cosas por comprender y mejorar, cosa que espero mejorar en el futuro.

Sobre el servidor, aunque en un principio se había considerado diseñar un servidor REST, la aparición de el servidor de Firebase dio un nuevo enfoque al trabajo.
Desarrollar una aplicación que use recursos de una tecnología tan reciente ha sido de gran ayuda para conocer y aprender a gestionar estos servicios.
Teniendo en cuenta que Firebase parece haber llegado para quedarse y liderar una nueva generación de servidores de aplicaciones móviles (BaaS) (su relación con Android es tan precisa y fácil de manejar que no sería de extrañar que sea el servidor que la gran mayoría de desarrolladores de aplicaciones del mundo, profesionales o novatos, usen para sus aplicaciones en el futuro), ha sido una gran idea utilizarlo y comprender cómo funcionan.

El único problema encontrado durante el uso del servidor radica en su estado en permanente evolución.
Hasta que no se desarrolle una versión final completamente estable no sería conveniente utilizarla para desarrollos grandes que requieran de un rendimiento óptimo.

Por todo lo expuesto, considero que he alcanzado los conocimientos que pretendía conseguir cuando decidí solicitar este TFG.

6.2. Desarrollo futuro

En este apartado se incluirán todas aquellas ideas que no sea han podido desarrollar en la versión final del proyecto por falta de tiempo.

Cambiar el formato de las piezas dentales.
Como ya se ha comentado en el capítulo 1 de este trabajo, la versión utilizada para los datos de las piezas dentales es un prototipo. Es necesario implementar los cambios necesarios para incluir los datos de la pieza final y alterar cada uno de los apartados de los formularios. También será necesario rediseñar las funciones desplegadas en **Cloud Functions** relacionadas con las piezas dentales para que se apliquen a las nuevas.

Eliminar piezas
Actualmente la función *DeleteUser* implementada en **Cloud Functions** sólo actualiza el perfil contenido en **Firestore** de un usuario que se ha dado de baja en su cuenta de **Authentication**.
Es necesario añadir en esta un mecanismo para la búsqueda de las piezas dentales creadas y no solicitadas y, una vez obtenido el resultado, realizar la eliminación de cada una de ellas.

Mejorar las consultas a las piezas dentales.
La aplicación usa escuchas permanentes para recuperar las piezas dentales que se busquen. Se ha diseñado de esta manera debido a que es la solución más útil para obtener los cambios producidos en los datos en tiempo real.

El problema radica en que esas escuchas consumen más recursos de red que un acceso común.

Se podría mejorar mediante consultas simples de tamaño fijo que se ejecuten una vez para la obtención inicial y otra cuando se notifiquen que se han producido ediciones los valores de las piezas (sólo para el caso de los dentistas).

Esto también podría permitir la creación de un método que permitiese seguir obteniendo resultados del servidor cuando se alcance el final del listado volviendo a solicitar al servidor otra lista partiendo desde la última pieza recibida.

Mejorar las reglas de las herramientas de Firebase.
Actualmente, las reglas establecidas sobre cada una de las herramientas no son lo suficientemente robustas para evitar que los usuarios que consigan crackear la aplicación puedan introducir datos erróneos.
Actualmente es la propia aplicación la que implementa las limitaciones de acceso a cada una de las base de datos asociadas y siempre espera obtener los resultados correctos por parte del servidor.

En el caso de que pudiesen evitar esas limitaciones, se podrían alterar o añadir datos corruptos para nuevos documentos relativos a las piezas dentales, los datos de perfil de los usuarios o introducir datos sin relación alguna con el fin de alcanzar el límite permitido para el tamaño de la base de datos.

Mejorar las peticiones al servidor.
La aplicación realiza consultas y descargas a las herramientas utilizando las funciones de conexión por defecto ofrecidas para cada una de ellas.
Cuando se completa correctamente una operación sobre ellas, **Cloud Functions** ejecuta las funciones implementadas que tratan esos datos en base a la acción realizada (creación, actualización o eliminación en la mayoría de los casos).

El problema radica en que varios elementos distintos subidos a una misma herramienta pueden requerir gestiones diferentes lo que obliga a que cada función creada compruebe cada una de esas consultas para ver si posee las condiciones necesarias para ejecutar su código.
También existe el riesgo de que una función de actualización ejecute otra actualización lo que obliga a controlarla mediante condiciones para evitar bucles infinitos.

La solución que se puede aplicar consistiría en la eliminación de tosas esas funciones que controlan las acciones y sustituirlas por funciones **HTTPS** (también se definen dentro de **Cloud Functions)** que realicen todo los necesario para llevar a cabo cada una de las funcionalidades del servidor de manera independiente las unas de las otras.
Esto permitiría simplificar el código de la aplicación, ya que todas las comprobaciones de datos antes de leer o escribir se ejecutarían dentro del código de la función.
Tampoco se producirían bucles debido a que son funciones independientes de las acciones que el usuario realice en las herramientas.

Realizar limpieza mensual
Firebase no permite ejecutar funciones cada cierto tiempo, pero sí permite que se reciban solicitudes desde otros servidores.
Para permitir la eliminación de elementos que lleven cierto tiempo en el servidor será necesario:

- Implementar una función **HTTPS** que ejecute las operaciones necesarias para los perfiles de usuario en estado de eliminación, las piezas dentales asociadas a ellos y para aquellas que fueron solicitadas tiempo atrás.
- Crear o utilizar un servidor que permita el envío de peticiones **HTTPS** a la dirección de la función indicada en el punto anterior cada mes.

Realizar pagos
Como se ha podido observar a lo largo del documento, muchas de las variables utilizadas para el almacenamiento de piezas dentales usan nombres relativos a su compra. Esto se debe a que un fase inicial del desarrollo se contempló la posibilidad de realizar compras mediante la plataforma **PayPal** (sistema para el pago en línea).

En un futuro se podría incluir la posibilidad de realizar compras mediante esa herramienta una vez se pulse algún botón de confirmación de pago dentro de la aplicación. Esta leería los datos de las cuentas de cada uno de los usuarios gracias a un documento en **Firestore** que albergase sus datos para **PayPal**.

Búsquedas compuestas complejas.
Debido al problema de tener que especificar cada combinación posible tanto en el servidor como en

el código de la aplicación cuando las consultas compuestas incluyen una o varias operaciones "mayor que" o "menor que" en el proyecto final no se ha desarrollado la posibilidad de incluirlas.

Sería interesante que algún valor de los elementos de las búsquedas de piezas dentales pudiesen contener uno o dos de estas operaciones (al menos para aquellos que incluyan valores numéricos). Deberían ser muy pocas porque sería prácticamente imposible configurar la aplicación y el servidor para un gran número de ellas.

Cliente de correo.
La implementación del backend del servidor envía correos automáticamente a los dentistas cada vez que un estudiante solicita una pieza dental. Para ello ha sido necesario incorporar a los credenciales de la cuenta de **Gmail** a **Cloud Functions.**
Debido a las limitaciones de 500 correos electrónicos por día es posible que en un futuro sea necesario incorporar otra herramienta que permita mayores cantidades. Será necesario crear una función que envíe peticiones a un servidor que se encargue de enviar los correos. El gran problema de esto es que las peticiones **HTTPS** no está permitidas en la versión gratuita de Firebase por lo que sería necesario contratar una versión de pago.

Actualizaciones del servidor.
Firebase es un servidor de aplicaciones en la nube muy reciente por lo que su desarrollo está en constante evolución.
No es un asunto menor debido a que todas las herramientas y sus funcionalidades pueden estar sujetas a cambios o incluso ser descartadas lo que obligaría a reestructurar toda la administración realizada.
Esto se ha observado durante el desarrollo de este trabajo en varias ocasiones:

- Cuando, sin recibir notificación alguna, la herramienta **CrashAnalytics** sustituyó a **Crash Reporting**.
- Durante las actualizaciones que se realizaron sobre **Cloud Functions** en el mes de julio de 2018 que obligaron a rediseñar el código utilizado hasta entonces porque las funciones anteriores dejaban de funcionar.
- Cambios producidos en la pantalla de administración web al cambiar de lugar o retirar y añadir nuevas funcionalidades.

Es por ello que, si el desarrollo se continua con este servidor, será necesario realizar constantes verificaciones de su estado para prevenir errores.

Incluir idiomas
La versión desarrollada de la aplicación sólo tiene disponible el inglés como idioma. Por ello, sería recomendable incluir otros idiomas en función de los parámetros del dispositivo para los diferentes usuarios. Para su inclusión es necesario crear una nueva carpeta el directorio *res* cuyo nombre sea *values-idioma* donde idioma será el código ISO 3166-1 alpha-2 del idioma deseado e incluir un archivo *strings.xml* con todas las nuevas cadenas.

Bibliografía

Enlaces para información

[inf 01] Lina C Gonzáles Pita, Margarita Viviana Úsuga Vacca, Carolina Torres Rodríguez y Edgar Delgado Mejía. (2014). *Biobanco de dientes humanos para investigación en odontología*. Bddigital. Recuperado de: https://revistas.unal.edu.co/index.php/actaodontocol/article/view/44602

[inf 02] Débora Vasconcelos Pereira, Marina Luisa Garbarino Giora, Renata de Oliveira Mattos Graner, José Carlos Imparato Pettorossi y Norailus Pérez Navarro. *Banco de dientes: una alternativa para la rehabilitación de dientes temporales anterosuperiores*. Revista Cubana de Estomatología.
Recuperado de: http://scielo.sld.cu/scielo.php?pid=S0034-75071997000200010&script=sci_arttext&tlng=pt

[inf 03] Dencells. (2015). *Células Madre Dentales*.
Recuperado de: http://www.dencells.com/celulas-madre/por-que-conservarlas/

[inf 04] Dencells. (2015). *Contratación*.
Recuperado de: http://www.dencells.com/contacto/contratacion/

[inf 05] eInforma. (2018). *Criodental Biopharma Sociedad Limitada*.
Recuperado de:
https://www.einforma.com/servlet/app/portal/RANK/prod/ETIQUETA_EMPRESA/nif/6YQbXQdO_Ko9TOqByCE_xg

[inf 06] Aixa. (2016). *¿Qué es la criopreservación? ¿Cómo se realiza?*.
Recuperado de: http://www.nosabesnada.com/otras-curiosidades/81821/que-es-la-criogenesis/

[inf 07] Microsoft Corporation. (2018). *¿Qué es una aplicación para Plataforma universal de Windows (UWP)?*.
Recuperado de: https://www.androidcentral.com/android-history

[inf 08] Sergio Delgado. (2017). *Kotlin, ¿otra moda más? (Spoiler: no).*
https://www.paradigmadigital.com/dev/kotlin-otra-moda-mas-spoiler-no/

[inf 09] Statcounter. (2018). *Mobile Operating System Market Share Worldwide - June 2018*.
Recuperado de: http://gs.statcounter.com/os-market-share/mobile/worldwide

[inf 10] Cristian Spoiala. (2015). *Cloud offering: Comparison between IaaS, PaaS, SaaS, BaaS*.
Recuperado de: https://assist-software.net/blog/cloud-offering-comparison-between-iaas-paas-saas-baas

[inf 11] ISO. (2013). *Country code*.
Recuperado de: https://www.iso.org/obp/ui/#search

Enlaces para Android

[and 01] Google LLC. (2018). *Aspectos fundamentales de la aplicación*.
Recuperado de: https://developer.android.com/guide/components/fundamentals?hl=es-419

[and 02] Google LLC. (2018). *Actividades*.
Recuperado de: https://developer.android.com/guide/components/activities?hl=es-419

[and 03] Google LLC. (2018). *Servicios*.
Recuperado de: https://developer.android.com/guide/components/services?hl=es-419

[and 04] Google LLC. (2018). *Proveedores de contenido*.
Recuperado de: https://developer.android.com/guide/topics/providers/content-providers?hl=es-419

[and 05] Google LLC. (2018). *Aspectos fundamentales de la aplicación*.
Recuperado de: https://developer.android.com/guide/components/fundamentals?hl=es-419

[and 06] Google LLC. (2018). *Fragmentos*.
Recuperado de: https://developer.android.com/guide/components/fragments?hl=es-419

[and 07] Google LLC. (2018). *Cuadros de diálogo*.
Recuperado de: https://developer.android.com/guide/topics/ui/dialogs?hl=es-419

[and 08] Google LLC. (2018). *Diseños*.
Recuperado de: https://developer.android.com/guide/topics/ui/declaring-layout?hl=es-419

[and 09] Google LLC. (2018). *Eventos de entrada*.
Recuperado de: https://developer.android.com/guide/topics/ui/ui-events?hl=es-419

[and 10] Google LLC. (2018). *Manifiesto*.
Recuperado de: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419

[and 11] Google LLC. (2018). *Android Debug Bridge*.
Recuperado de: https://developer.android.com/studio/command-line/adb?hl=es-419

Enlaces para Firebase

[fir 01] Google LLC. (2018). *Agrega Firebase a tu proyecto de Android*.
Recuperado de: https://firebase.google.com/docs/android/setup

[fir 02] Google LLC. (2018). *Firebase Authentication*.
Recuperado de: https://firebase.google.com/docs/auth/

[fir 03] Google LLC. (2018). *Firebase Realtime Database*.
Recuperado de: https://firebase.google.com/docs/database/

[fir 04] Google LLC. (2018). *Cloud Firestore*.
Recuperado de: https://firebase.google.com/docs/firestore/

[fir 05] Google LLC. (2018). *Cloud Storage*.
Recuperado de: https://firebase.google.com/docs/storage/

[fir 06] Google LLC. (2018). *Cloud Functions para Firebase*.
Recuperado de: https://firebase.google.com/docs/functions/

[fir 07] Google LLC. (2018). *Cloud Analytics para Firebase*.
Recuperado de: https://firebase.google.com/docs/analytics/?hl=es-419

[fir 08] Google LLC. (2018). *Firebase Crashlytics*.
Recuperado de: https://firebase.google.com/docs/crashlytics/

[fir 09] Google LLC. (2018). *Firebase Crash Reporting*.
Recuperado de: https://firebase.google.com/docs/crash/?hl=es-419

[fir 10] Google LLC. (2018). *Firebase Performance Monitoring*.
Recuperado de: https://firebase.google.com/docs/perf-mon/

[fir 11] Google LLC. (2018). *Firebase Test Lab*.
Recuperado de: https://firebase.google.com/docs/test-lab/

[fir 12] Google LLC. (2018). *firebase.database.Query.*
Recuperado de: https://firebase.google.com/docs/reference/js/firebase.database.Query

[fir 13] Google LLC. (2018). *Firebase Predictions*.
Recuperado de: https://firebase.google.com/docs/predictions/

[fir 14] Google LLC. (2018). *Cuotas y límites.*
Recuperado de: https://firebase.google.com/docs/firestore/quotas

[fir 15] Google LLC. (2018). *Server-Side Encryption*.
Recuperado de: https://cloud.google.com/firestore/docs/server-side-encryption

[fir 16] Google LLC. (2018). *Seguridad de Google Cloud Plataform*.
Recuperado de: https://cloud.google.com/security/

[fir 17] Google LLC. (2018). *Privacidad y seguridad en Firebase*.
Recuperado de: https://firebase.google.com/support/privacy/?hl=es-419

Enlaces a Stack Overflow

[sta 01] Peter O. (2016). *How to check internet access on Android? InetAddress never times out*.
Recuperado de: https://stackoverflow.com/questions/1560788/how-to-check-internet-access-on-android-inetaddress-never-times-out

[sta 02] Srikanth. (2017). *EditText allow only alphabets, digits of all languages*.
Recuperado de: https://stackoverflow.com/questions/41936266/edittext-allow-only-alphabets-digits-of-all-languages

[sta 03] Flexo. (2018). *Android number picker keyboard type*.
Recuperado de: https://stackoverflow.com/questions/16793414/android-number-picker-keyboard-type

Anexo

1. Manual de usuario

Pantalla principal
Opciones disponibles:

- *Inicio de sesión*. Será necesario introducir unos credenciales correctos y haber aceptado el correo de confirmación de cuenta.
- *Registro*. Permite registrarse como estudiante o dentista.
- *Cambio de correo electrónico o contraseña*.

Registro: Para ello será necesario pulsar la opción de registro dentro de la pantalla de inicio de sesión y escoger una de las dos opciones (estudiante o dentista). Se debe rellenar cada uno de los apartados del formulario que se mostrará y pulsar el botón de registrarse. Si el registro se ha completado de manera correcta se mostrará un mensaje por pantalla.

Cambio de correo electrónico o contraseña: Se deberá seleccionar la opción correspondiente tras pulsar la opción en la ventana principal. Una vez que se acceda a la nueva, introducir los credenciales e iniciar sesión. Tras ello, se mostrará un formulario de cambio que se deberá rellenar y aceptar. La correcta alteración de los datos provocará que se envíe un correo al usuario.

Versión para el usuario de tipo dentista
Opciones del menú:

- *Perfil de usuario.* Muestra el perfil del usuario actual.
- *Dientes creados*. Listado de piezas dentales que han sido creadas por el usuario.
- *Dientes que han sido solicitados*. Listado de piezas dentales que han sido solicitadas por algún estudiante.
- *Dientes que han sido enviados a la papelera*. Listado de piezas dentales que el usuario ha enviado a la papelera desde la pantalla de dientes creados.
- Información acerca de la aplicación. Muestra información acerca de la aplicación y la universidad.
- *Cerrar sesión*. Desconecta al usuario del servidor y lo devuelve a la pantalla de inicio de sesión.

Editar perfil: Pulsando el botón de la barra de herramientas se podrá acceder a la edición del perfil.

Enviar a papelera un diente: Se podrá mantener pulsada la pieza dental desde la pantalla de dientes creados y esperar a que aparezca la nueva barra de herramientas desde donde podrá la opción de enviar a la papelera (permite seleccionar varios elementos) o acceder a su ventana de datos detallada y pulsar el botón de la barra de herramientas.
Para realizar el paso inverso habrá que acceder a la ventana de dientes enviados a la papelera y realizar el mismo procedimiento pulsado el botón de "recuperar".

Añadir piezas dentales: Para ello el usuario deberá pulsar el botón más de la barra de herramientas. Esto mostrará un formulario que deberá rellenar completamente incluyendo también una imagen.

Una vez completado, al pulsar sobre el botón de inclusión se mostrará la pantalla de carga hasta que todo haya finalizado correctamente (se notifica con un mensaje en pantalla)

Editar piezas dentales: Es necesario que el usuario acceda a la información detallada de una pieza dental dentro de la categoría de piezas creadas. Una vez allí, deberá pulsar el botón superior de la barra de herramientas lo que mostrará una nueva ventana donde se mostrará el formulario de añadir piezas dentales ya rellenado con los datos de la pieza seleccionada. Podrá seleccionar cualquier opción y alterarla, lo que mostrará un mensaje de confirmación o de error si se ha producido algún fallo.

Eliminar definitivamente una pieza dental: Es necesario acceder a la ventana de piezas dentales enviadas a la papelera pulsar la pieza dental deseada y esperar a que aparezca la nueva barra de herramientas desde donde podrá la opción de eliminar definitivamente (permite seleccionar varios elementos). También podrá realizarse desde la ventana de de datos detallados de la misma sección y pulsar el botón de la barra de herramientas.

Versión para el usuario de tipo estudiante
Opciones del menú:

- *Perfil de usuario. Muestra el perfil del usuario actual.*
- *Dientes solicitados*. Piezas dentales que han sido solicitadas por el usuario.
- *Buscar dientes*. Rellenar un formulario de búsqueda para las piezas dentales.
- *Información*. Muestra información acerca de la aplicación y la universidad.
- *Cerrar sesión*. Desconecta al usuario del servidor y lo devuelve a la pantalla de inicio de sesión.

Editar perfil: Pulsando el botón de la barra de herramientas se podrá acceder a la edición del perfil.

Buscar un diente: Para ello habrá que rellenar el formulario que se muestra en la ventana de búsqueda de dientes y pulsar el botón de buscar. Una vez se obtengan los resultados se mostrarán en un listado.

Solicitar diente: Previamente se habrá realizado un búsqueda previamente y se habrá seleccionado una de las piezas dentales mostradas. Dentro de su ventana de datos detallados se deberá pulsar el botón de solicitar en la parte inferior.

Para ver ejemplos del funcionamiento se puede consultar el siguiente enlace:
https://www.youtube.com/channel/UC4y4o5sSGOGiUtlCbznuMKw/videos

2. Acceso a ADB

ADB [and 11] es una herramienta que permite mediante un conjunto de comandos propios ejecutar distintas operaciones en un móvil Android desde el ordenador [and 08].

Para poder acceder a ella en Windows, desde una consola de comandos hay que:

1. Ir a la ubicación C:\Users\%usuario %\AppData\Local\Android\Sdk\platform-tools

2. Introducir el siguiente comando: adb shell.

3. Métodos de más de 64KB

En el caso de que la aplicación use métodos que superen los 64 kB se producirán fallos que impedirán su instalación en los dispositivos móviles.
Para solucionarlo hay que:

1. Dentro de build.gradle a nivel de aplicación añadir la línea multiDexEnabled true en *android -> defaultConfig*

2. Si la versión del sdk mínimo usado está por debajo de la 20 hay que incluir en build.gradle a nivel de aplicación la línea implementation 'com.android.support:multidex:1.0.3' en *dependencies*.

3. Dentro del manifiesto incluir la línea android:name= "android.support.multidex.MultiDexApplication" dentro de la etiqueta *application*.

4. Volver a reconstruir el proyecto.

4. Uso de Geocoder y Google maps

Para poder usar **Google maps** con el propósito de obtener la ubicación de una determinada dirección escrita en una cadena de texto es necesario traducir dicha dirección en coordenadas geográficas mediante la biblioteca **Geocoder**.

El uso de **Geocoder** requiere de una serie de pasos:

1. Obtener la llave que permitirá usarla.
   Para ello es necesario ir a la siguiente dirección https://cloud.google.com/maps-platform/#get-started, seguir los pasos indicados y obtenerla.
   En función del tipo de cuenta que se tenga en la plataforma (gratuita o de pago) se podrán traducir más o menos direcciones.

2. Incluir la llave en el manifest del proyecto.
   Dentro de la etiqueta *application* se incluye:
   ```
   <meta-data
       android:name="com.google.android.geo.API_KEY"
       android:value="LA_KEY_DADA"/>
   ```

3. Reconstruir el proyecto para poder usar **Geocoder**.

5. Bocetos previos

En este apartado se muestran los bocetos básicos que se diseñaron para las distintas pantallas de la aplicación tras un estudio previo a la implementación final.

Figura 5.1.

Figura 5.2.

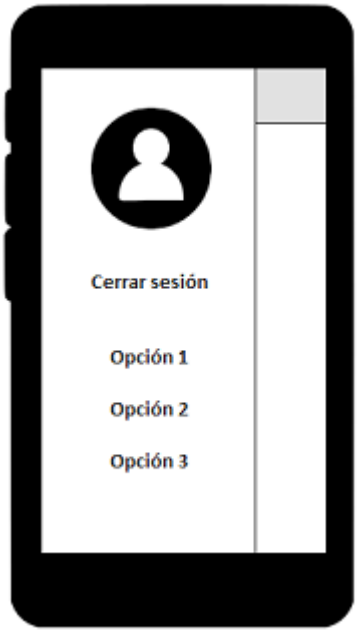


Figura 5.3.

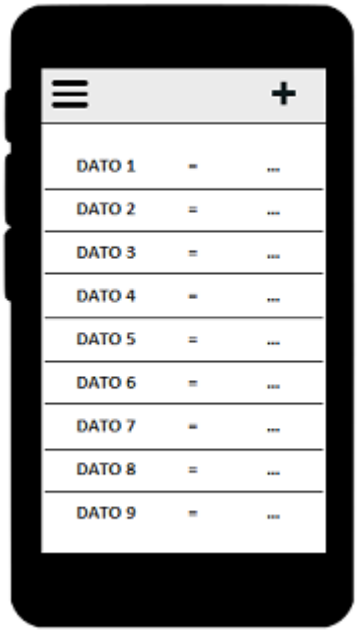


Figura 5.4.

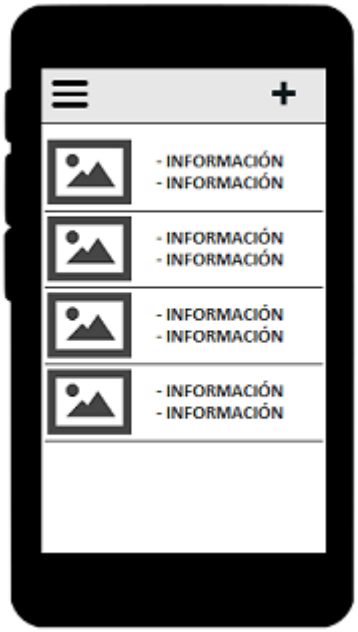


Figura 5.5.

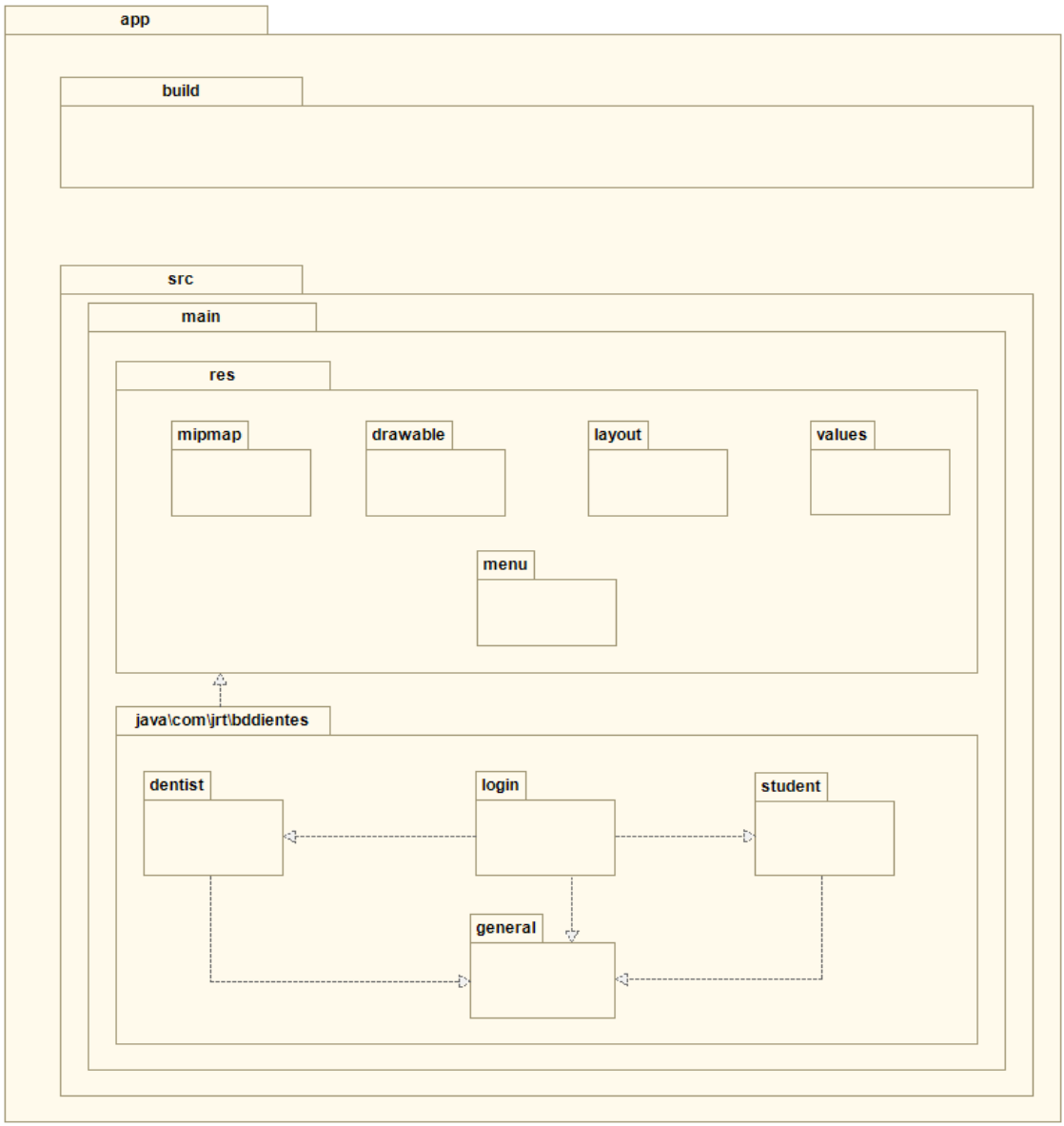
Figura 5.6.

- **Figura 5.1**: interfaz inicial de la aplicación.
- **Figura 5.2**: interfaz una vez el usuario haya iniciado sesión correctamente.
- **Figura 5.3**: interfaz para el menú de usuario.

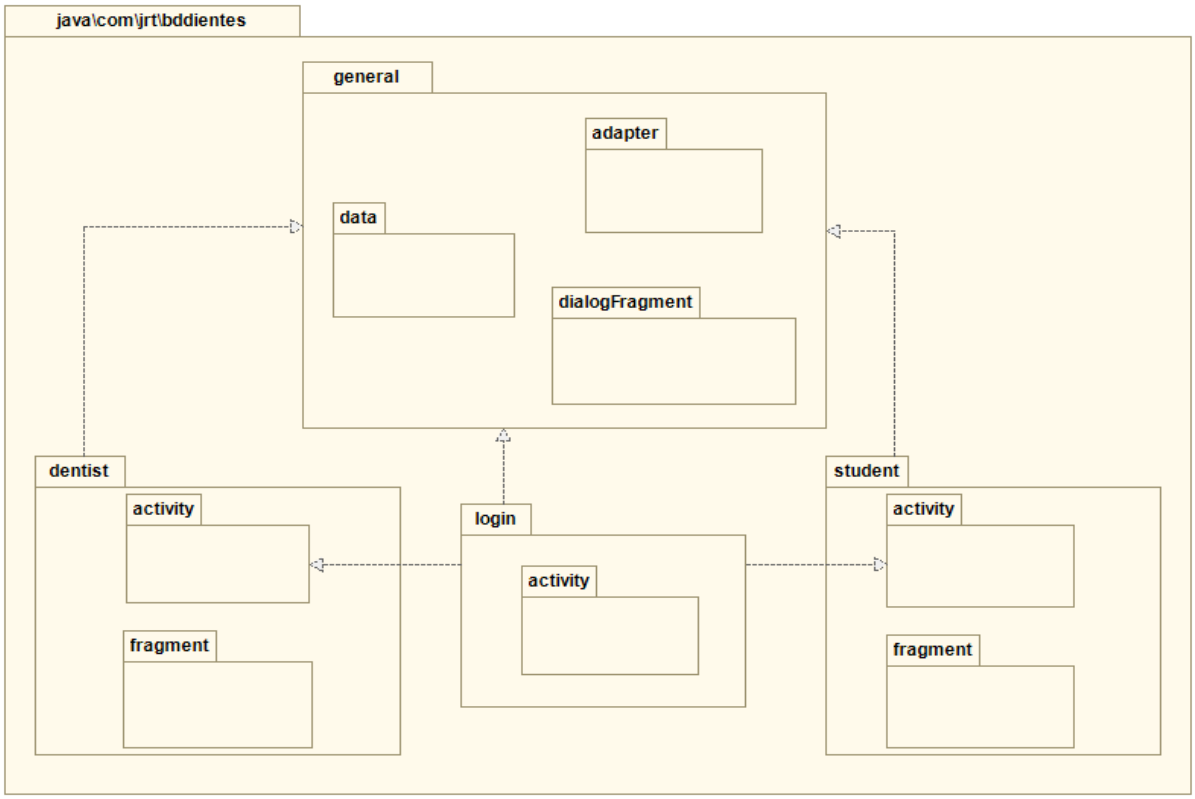
- **Figura 5.4**: interfaz para los formularios o las búsquedas.
- **Figura 5.5**: interfaz para los resultados de las búsquedas de piezas dentales.
- **Figura 5.6**: interfaz para mostrar detalladamente los datos de la pieza dental seleccionada tras realizar una búsqueda.

6. Diagramas de paquetes

6.1. Diagrama de paquetes global

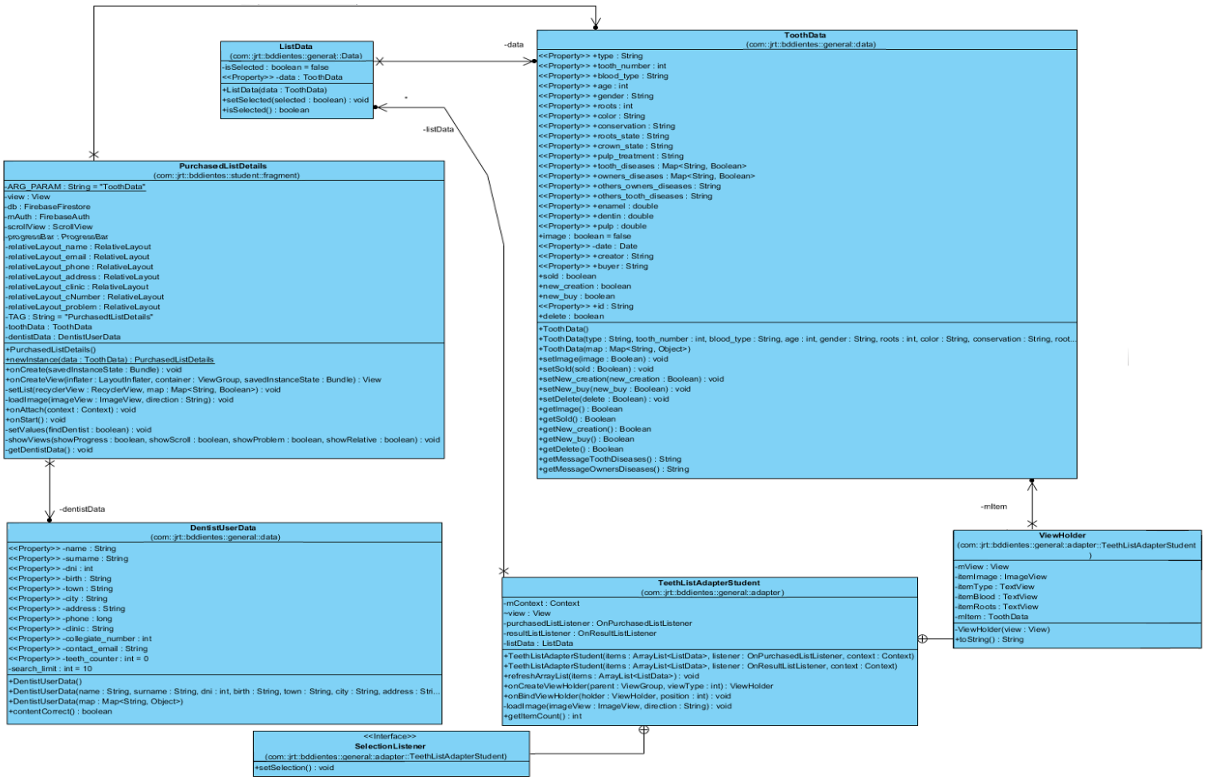


6.2. Diagrama de paquetes detallado

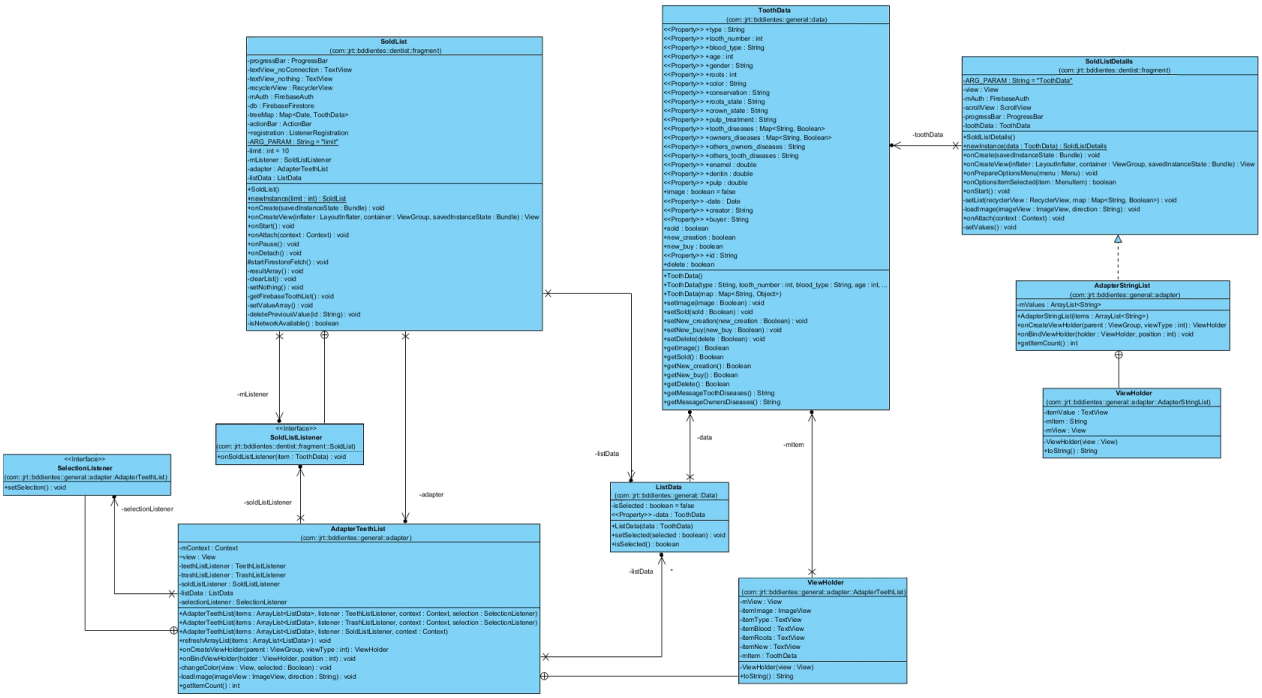


7. Diagramas de clases

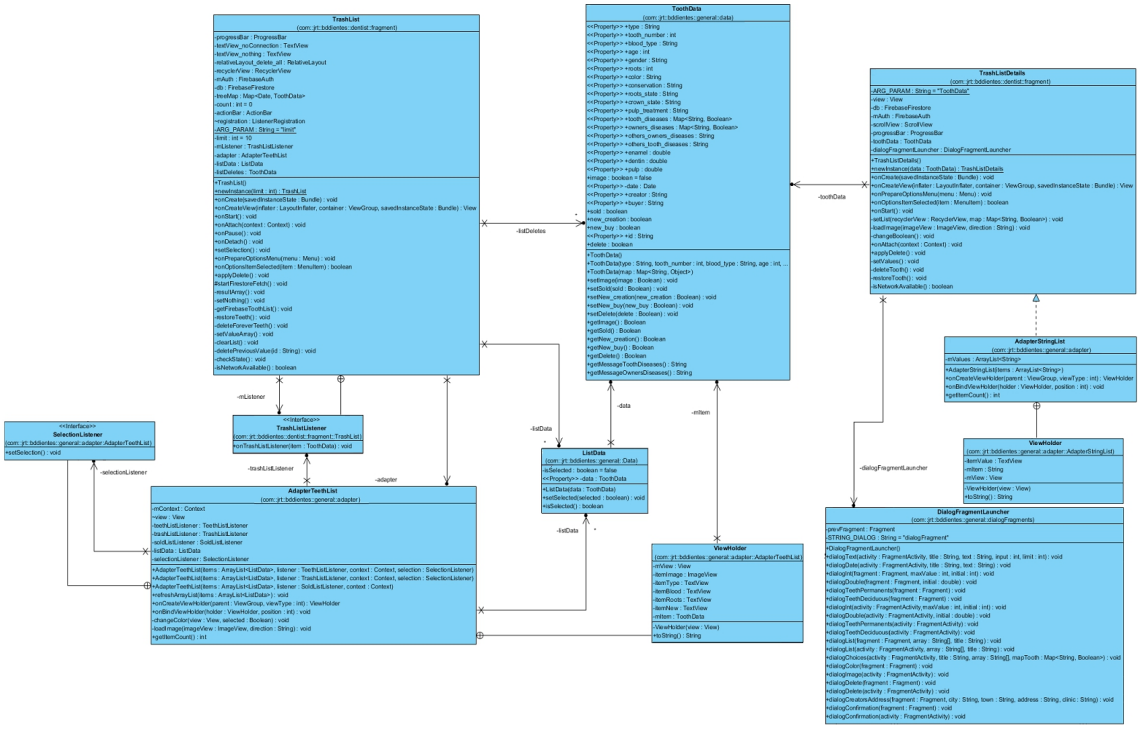
7.1. Diagrama de clases para *PurchasedList* (Dentista)



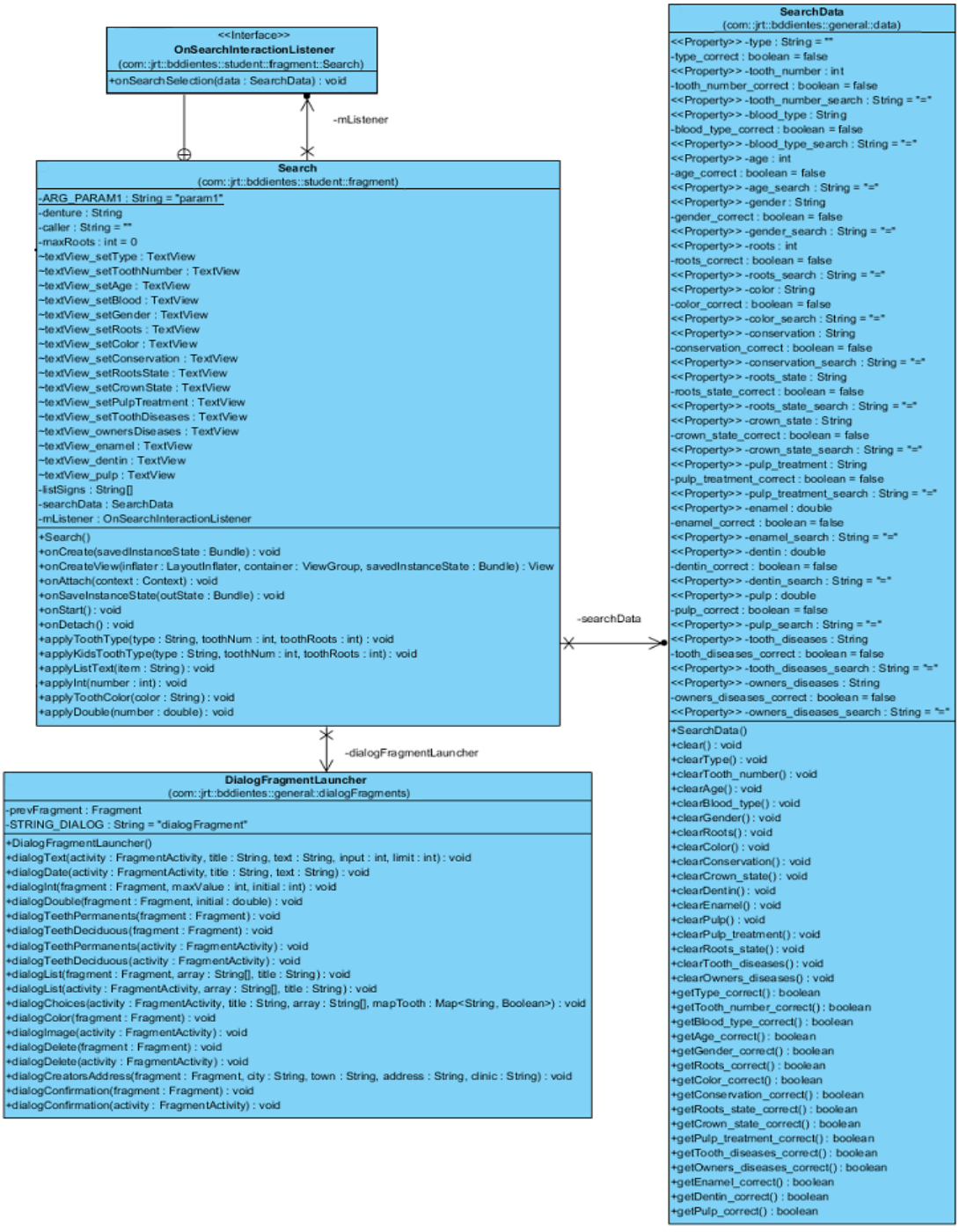
7.2. Diagrama de clases para *SoldList* (Dentista)



7.3. Diagrama de clases para *TrashList* (Dentista)

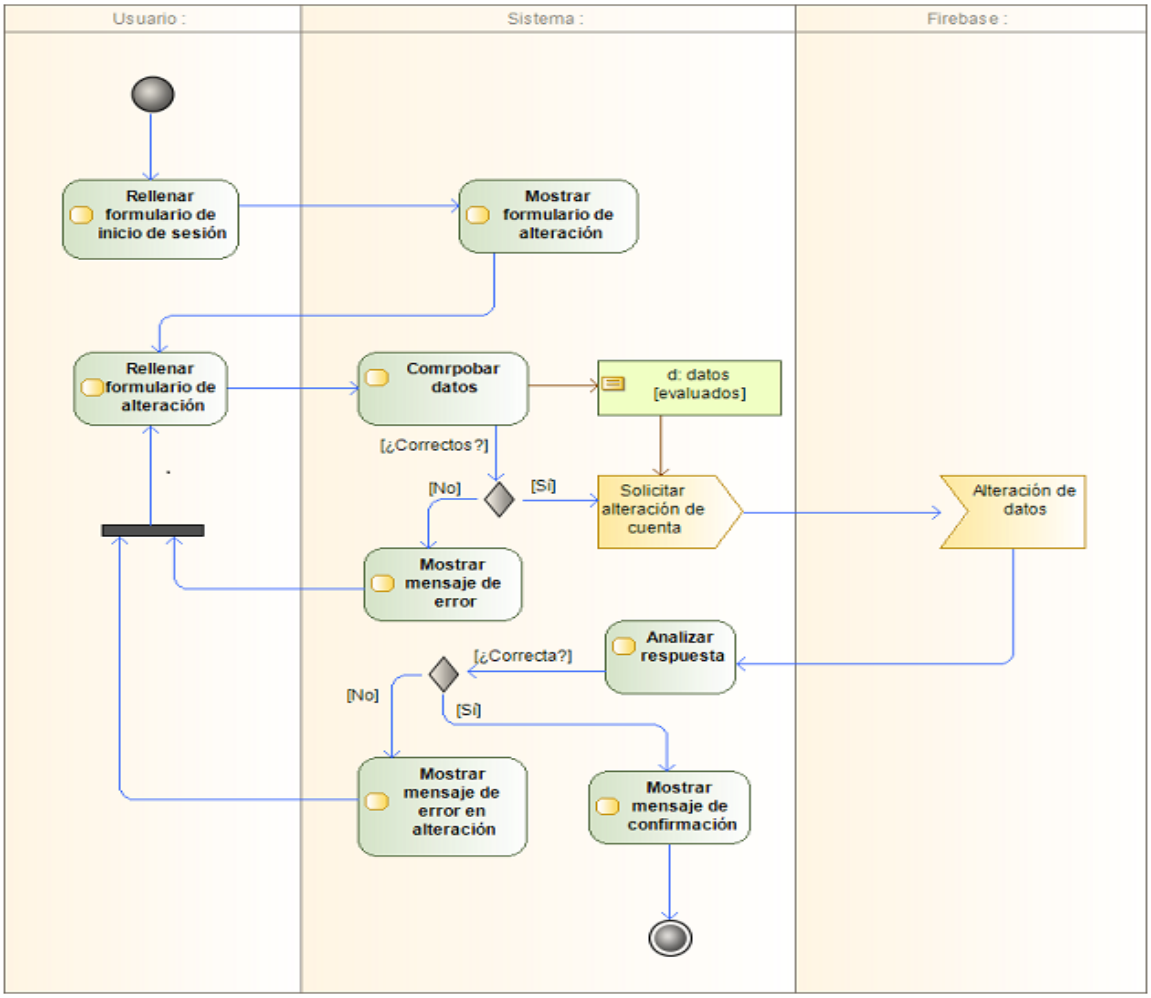


7.4. Diagrama de clases para *Search* (Estudiante)



8. Diagramas de actividades

8.1. Diagrama de actividades para alteración del perfil de usuario



8.2. Diagrama de actividades para la actualización del estado en papelera (Dentista)