UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Ingeniería del Software

Talking Pics – Habla sin Palabras

Talking Pics – Talk without Words

Realizado por
Raquel Portales Luna

Tutorizado por
Gabriel Jesús Luque Polo

Departamento
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga

MÁLAGA, Junio 2021

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA

UNIVERSIDAD DE MÁLAGA

Ingeniería del Software

Talking Pics –Habla sin Palabras
Talking Pics–Talk without Words

Realizado por
Raquel Portales Luna

Tutorizado por
Gabriel Jesús Luque Polo

Departamento
Departamento de Lenguajes y Ciencias de la Computación Universidad de Málaga

MÁLAGA, Junio 2021

Dedico este trabajo a mis padres, por ser quienes me han apoyado día a día. Gracias por vuestros consejos que me ayudaron y guiaron a lo largo de la carrera y de mi vida.

Gracias a mi hermano, porque sus risas y su compañía cuando realizaba este proyecto me ayudaron a seguir adelante.

A mi pareja, que ha estado conmigo, literalmente, día a día conmigo en la universidad apoyándome. Estoy orgullosa de haber conseguido realizar esta etapa junto a ti.

A mi grupo de amigas, con las que he compartido grandes momentos desde antes de realizar las carreras con las que soñábamos durante años.

A mis compañeros y amigos, que han hecho que el paso por la universidad sea mucho más fácil gracias a ellos.

Y a mi tutor, por la ayuda y los conocimientos que me ha brindado al realizar este Trabajo de Fin de Grado.

Resumen

Hoy en día, un gran grupo de personas tiene algún tipo de discapacidad que no les permite comunicarse con mecanismos habituales. Esto hace que tengan que buscar un método más sencillo para ellos, que se adapte a sus necesidades. En hospitales, es común usar plantillas con las que los pacientes con estas capacidades pueden señalar lo que necesitan en ese momento o lo que quieren comunicar. Sin embargo, este método puede llegar a ser bastante lento.

Por ello, este proyecto busca desarrollar una aplicación que ayude a personas en esta situación, permitiendo formar frases más complejas de una forma más rápida y menos tediosa. Actualmente no existen aplicaciones que realicen este tipo de tareas, por lo que Talking Pics no tiene ningún competidor en este sector.

Las tecnologías principales en este proyecto han sido Spring, para poder crear el backend del sistema, React Native, para desarrollar el frontend, y Python, para crear una API de predicciones de texto. Aparte de estas tecnologías, se han usado algunas otras como balsamiq para hacer el diseño o Latex, para escribir esta memoria.

Palabras clave: Aplicación, Móvil, Comunicación, Discapacidad

Abstract

Nowadays, a huge amount of people have some type of disability that does not allow them to communicate with common mechanisms. Because of this, they have to look for a simpler method for them, that can adapt to their needs. In hospitals, it is really common to use plastic boards so that patients with these disabilities can point out what they need at that moment or they want to communicate.

Thus, this project seeks to develop an application that will be able to help people in this situation, allowing them to form more complex phrases in a faster and less tedious way. At the present, there is not any system that performs this kind of tasks, so Talking Pics does not have any competitor in this sector.

The main technologies used in this project were Spring, which was used to create the system's backend, React Native, which was used to develop the frontend, and Python to create a prediction API. Apart from theses technologies, Balsamiq was used to make the design or Latex to write this document.

Keywords: App, Smartphone, Communication, Disability

Índice

1 Introducción

En esta sección se explicará la motivación principal de este TFG, además de sus objetivos. Finalmente se mencionará la organización de esta memoria.

1.1. Motivación

Según la Organización Mundial de la Salud, la definición de discapacidad es "la interacción entre las personas que tienen algún problema de salud y factores personales y ambientales"[25]. En el mundo, actualmente existen más de 1000 millones con algún tipo de discapacidad, sin embargo en este proyecto nos centraremos especialmente en las discapacidades comunicativas que una persona podría tener.

Desde hace años se utilizan tablas con pictogramas, tal y como se muestra en la imagen 1, para poder comunicarse con personas con dificultades para expresarse. Estas tablas, aunque útiles, son lentas de usar y no permiten construir frases de cierta complejidad.

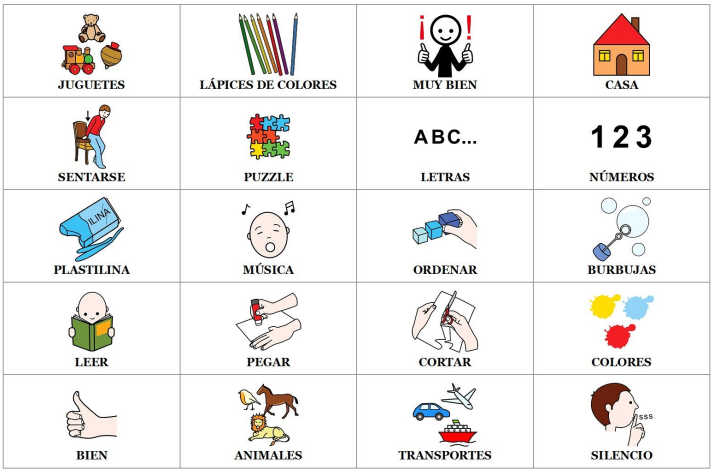


Figura 1: Pictogramas

La motivación de este TFG es crear una solución alternativa, en este caso una aplicación

móvil, a estas tablas de forma que se puedan crear frases complejas de una forma sencilla. De esta forma, tanto los cuidadores o personas externas, como las personas que tengan estas dificultad podrán comunicarse mucho más rápido que con las tablas, además de ser una solución que puedes llevar a cualquier parte.

1.2. Objetivos

El resultado de este TFG y, por lo tanto el objetivo principal de este, será una aplicación móvil desarrollada para dispositivos Android que permitirá al usuario con problemas de comunicación, relacionarse de una manera sencilla y rápida por medio de pictogramas. La idea es que facilite la comunicación de manera bidireccional.

Por un lado, el usuario con problemas de comunicación (que no es capaz de usar textos o palabras) seleccionará las imágenes que concuerden con lo que él quiere comunicar, mientras que la aplicación se encargará de recomendarle posibles imágenes que podrían estar relacionadas con las que seleccionó anteriormente.

Por otro lado, la aplicación también nos permitirá tomar una fotografía de una pequeña frase (por ejemplo, una señal, cartel o indicación) y que esta sea traducida a pictogramas, de forma que esta sea más sencilla de entender para el usuario.

Para personalizar aun más el uso de la aplicación para el usuario, este podrá almacenar sus propias imágenes con sus respectivos significados, para poder usarlas al construir sus frases de imágenes.

1.3. Organización de la memoria

Este TFG se divide en varias secciones. Primero se explicarán las diferentes tecnologías que se han usado a lo largo del proyecto para desarrollar la aplicación.

Tras esto, se procederá a explicar la metodología usada y como se han repartido las tareas a lo largo del tiempo.

En la sección de desarrollo, se explica el proceso para desarrollar las dos APIs que se han necesitado para este proyecto:

1. Una pequeña API que nos permite obtener predicciones de palabras gracias a un modelo de IA.

2. La API principal de la aplicación, que se conecta con la base de datos y otras APIs externas, sirviendo de intermediaria, además de proporcionar toda la lógica de la aplicación.

Finalmente, se explicarán las conclusiones de este proyecto y las lineas futuras que podrían tener. Además, en los anexos se incluyen el Manual de Instalación y el Manual de Usuario.

2
Tecnologías

En esta sección se explicarán las diferentes tecnologías que se han usado a lo largo del proyecto. Esta sección ha sido dividida según el uso que le hemos dado a las diferentes tecnologías. Primero explicaremos las herramientas utilizadas para desarrollar el backend, concretamente para la API de predicciones y la API principal, para luego proceder a desarrollar las tecnologías usadas en el frontend y en la memoria.

2.1. API para obtener predicciones

Esta API se ha desarrollado en Python (cuyo logotipo puede verse en la figura 2a), el cual es un lenguaje que busca que el código sea lo más sencillo de entender. Junto con Python (cuyo logotipo puede verse en la figura 2b), se ha usado Flask, el cual es un framework de desarrollo de APIs o páginas web y cuya filosofía es escribir el menor número de lineas de código.

Puesto que la API ha desarrollar no era demasiado complicada ni necesitaba muchos puntos de acceso, se eligieron estas tecnologías para hacer el desarrollo mucho mas sencillo. Además, desarrollar esta API con Python nos permite que en un futuro se pueda ampliar este proyecto, y así poder desarrollar nuestro propio modelo de predicciones.

(a) Logotipo Python

(b) Logotipo Flask framework

Figura 2: Tecnologías para API de predicciones

El modelo de predicciones que se ha usado es BETO [7], el cual es un modelo que se ha entrenado con un corpus de 2996016962 palabras en español. Concretamente se ha usado la versión *Uncased*, es decir, que no tiene en cuenta ni mayúsculas ni minúsculas para predecir la siguiente palabra. Se ha usado este modelo debido a lo sencillo que es usarlo, ya que con una simple llamada a su endpoint podemos obtener una predicción para la frase que le pasemos por parámetro.

Finalmente, para subir esta API a la nube, se ha usado Heroku, usando un Procfile que indica las especificaciones necesarias para desplegar la aplicación, como por ejemplo en este caso, que tuvimos que declarar que esta era una aplicación web. Heroku es conocido por su sencillez, ya que simplemente con añadirle el archivo Procfile mencionado anteriormente podemos subir una aplicación web o una API a su plataforma.

Figura 3: Logotipo de Heroku

Al ser una aplicación Python, para que Heroku funcione correctamente tendremos que añadir un fichero 'requirements.txt', el cual incluirá todos los paquetes que se han instalado para este proyecto

2.2. API principal

Esta API se ha desarrollado en Java, con uso del framework Spring [1] (cuyos logotipos pueden verse en las figuras 4a y 4b). Este es un framework que nos permite realizar APIs en Java, entre otras funcionalidades. Mediante las anotaciones que nos proporciona Spring, podremos crear los diferentes controladores y servicios que ofrecera nuestra aplicación. Un ejemplo de esto es la anotación *@Controller*, la cual se indica al declarar una clase. De esta forma, Spring sabe que en esta clase se desarrollaran endpoints y que se podrán llamar según la dirección indicada en las anotaciones.

Además Spring nos proporciona un inicializador [3] de proyectos, en el cual nosotros indicaremos los paquetes que necesitamos, a parte de la versión de Spring y Java que necesitaremos, de forma que nos permitirá descargar el proyecto para que podamos editarlo y ejecutarlo en nuestra máquina.

(a) Logotipo de Java

(b) Logotipo de Spring

Figura 4: Tecnologías para API principal

Esta API se ha conectado con una base de datos en Firebase [23] (cuyo logotipo puede verse en la figura 5), para poder almacenar los datos de las imágenes añadidas por los usuarios y sus frases favoritas. Se decidió usar Firebase debido a la sencillez de su base de datos, ya que es una base de datos NoSQL, organizada por documentos. Esto implica que muchas veces tendremos elementos repetidos, por ejemplo en los documentos de *Images*, tendremos también almacenado el usuario que ha subido esa imagen, aparte de añadirlo en los documentos de *Users*.

Figura 5: Logotipo de Firebase

Para poder implementar la funcionalidad de traducir frases, una funcionalidad que se describirá más adelante en la sección de Desarrollo, necesitábamos que a partir de una imagen se pudiera obtener el texto de ella. Para ello se ha usado la herramienta de Google Vision [5] (6).

Para usar esta herramienta en Spring, se tiene que instalar el paquete de Google Vision [11], el cual nos permite enviar URLs de imágenes a partir de unas credenciales JSON que están almacenadas en el sistema. Este paquete nos devuelve un *String*, el cual hay que limpiar de caracteres especiales como los saltos de linea y organizarlo en un array de String, para que la API de Arasaac pueda obtener un pictograma para cada una de las palabras en esa imagen.

Figura 6: Logotipo de Google Vision

Arasaac (cuyo logotipo se puede ver en la figura 7) es una API desarrollada por el Centro Aragonés para la Comunicación Aumentativa y Alternativa [6], la cual pone a nuestra disposición numerosos pictogramas que podemos usar para desarrollar aplicaciones para personas con habilidades comunicativas reducidas. Esta API también nos permite buscar pictogramas según el idioma de nuestro usuario, sin embargo esta funcionalidad no esta implementada, si no que se incluirá en lineas futuras.

Figura 7: Arasaac

Arasaac nos permite usar diferentes endpoints, pero los que se han usado en esta aplicación han sido los siguientes:

- https://api.arasaac.org/api/pictograms/all/{locale} - Nos permite obtener todos los pictogramas que hay almacenados en arasaac según el código de idioma que pongamos en locale.
- https://api.arasaac.org/api/pictograms/{idPictogram} - Podemos obtener un pictograma segun el id que le pasemos en la url.
- https://api.arasaac.org/api/pictograms/{locale}/search/{searchText} - Nos permite obtener todos los pictogramas que contengan la palabra que le pasemos en la url en searchText. Se tiene que especificar el idioma que estamos usando
- https://api.arasaac.org/api/pictograms/{locale}/bestsearch/{searchText} - Busca los pictogramas que contengan exactamente el texto pasado en searchText en el idioma especificado en locale.

Para poder probar las diferentes funcionalidades que se iban desarrollando se usaron dos tecnologías diferentes. La primera es Postman, la cual se usó principalmente para comprobar que las operaciones POST, PUT, DELETE y GET añadían, actualizaban, eliminaban y obtenían

los usuarios de la base de datos correctamente. Por otro lado, se uso JUnit en Java. Principalmente esta tecnología se uso para poder comprobar que las operaciones GET de la API devolvían el código de estado correctamente o si el tipo MIME de la respuesta era el correcto.

(a) Logotipo de Postman

(b) Logotipo de JUnit

Figura 8: Pruebas para API principal

Finalmente, tal y como se uso en la otra API, se subió el proyecto a Heroku, indicando que era una aplicación web.

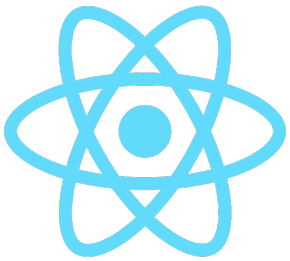
2.3. Interfaz de la aplicación

Balsamiq es una herramienta de diseño de Wireframes, la cual nos permite simular la interacción con el usuario para movernos de una ventana a otra. El problema de esta herramienta es que no es totalmente gratuita, por lo que para usarla se tiene que crear un proyecto, el cual será eliminado en los 30 días siguientes.

Aunque tenga ese punto negativo, la herramienta nos permite diseñar rápidamente como queremos que se vea nuestra aplicación, por lo que era perfecta para este proyecto.

Para implementar el frontend se ha usado React Native [20] con ayuda del framework Expo [22] (cuyos logotipos se pueden ver en las figuras 9b y 9a).

(a) Expo



(b) React Native

Figura 9: Tecnologías para el Frontend

React Native está basado en el lenguaje de JavaScript, y nos permite crear componentes para reutilizar en las diferentes ventanas que tenga nuestra aplicación. Además, mediante el estilo Flex que podemos añadir a nuestros componentes, estos pueden ser responsive, de forma que se adapten según el tamaño del dispositivo en el que estemos usando nuestra aplicación.

Este proyecto se desarrollo sin ningún conocimiento de React Native, por lo que esto añadió bastante complejidad. Sin embargo, la razón por la que se quiso usar React Native es por que actualmente es uno de los frameworks más usados para el desarrollo de aplicaciones móviles. Empresas como Pinterest o Facebook lo usan en sus apps.

Junto con React Native, se ha usado Expo, un framework que proporciona diferentes funcionalidades y librerías a proyectos de aplicaciones móviles o web. Expo nos permite, además de probar nuestra aplicación de forma sencilla mediante la aplicación móvil Expo Go, añadir otras funcionalidades que de otra forma serían mucho más complicadas de conseguir, como por ejemplo el inicio de sesión con Google.

Expo también nos ha permitido usar funcionalidades como la de acceder a la cámara o biblioteca del dispositivo del usuario, para que pueda añadir una imagen a nuestra aplicación, o la de crear una APK a partir de nuestro proyecto.

Además, se han usado múltiples bibliotecas, como React Native Navigation para poder navegar entre las diferentes pantallas de la aplicación y pasar argumentos entre ellas; o React Native Elements, que proporciona componentes que no se encuentran en la biblioteca oficial de React Native.

Para poder almacenar las imágenes que suba el usuario se ha usado cloudinary. Para ello

tenemos que crearnos una cuenta y asi obtener nuestra API Key que nos permitirá almacenar archivos. Desde la interfaz, cuando el usuario selecciona una imagen esta se guarda en cloudinary accediendo a su endpoint.

Figura 10: Logotipo de Cloudinary

2.4. Memoria

Para desarrollar este documento se ha usado LaTeX, que es un lenguaje de marcado el cual nos permite escribir documentos con formato. Para poder crear un documento LaTeXse ha usado la herramienta Overleaf (cuyo logotipo se puede ver en la figura 11), en la cual se pueden desarrollar este tipo de documentos de forma mucho más sencilla que en un editor de texto, ya que no requiere instalar nada y además está disponible en cualquier dispositivo al ser una aplicación web.

Figura 11: Logotipo de Overleaf

3
Metodología

En esta sección se explicará cual ha sido la metodología elegida, además de ver qué características principales tiene y cómo se ha adaptado al trabajo de una sola persona.

3.1. ¿Qué es Scrum?

Scrum [2] es un marco de trabajo el cual nos permite desarrollar un proyecto de forma que los problemas que se encuentren durante todo el proceso sean mucho más fáciles de desarrollar, permitiéndole ofrecer al cliente una aplicación de mucha más calidad que si, por ejemplo, se desarrollara con el marco de trabajo en cascada.

Scrum es mucho más liviano que otros marcos de trabajo, además de permitir adaptarnos a la situación que tenga en ese momento el proyecto. También, al estar siempre en contacto con el cliente, nos permite hacer una aplicación más acorde a las necesidades del cliente.

3.2. Características principales

Las principales características de Scrum son las siguientes:

- Transparencia - Todos los participantes del proyecto conocen la situación del proyecto, cuales son los problemas que se están teniendo y cuales son los sprints que se han terminado.

- Inspección - Las personas implicadas en el proyecto probar las versiones o artefactos que se realizan, para estar seguros de que es lo que realmente el cliente pide.

- Adaptación - Tal y como se menciono anteriormente, Spring tiene la posibilidad de adaptarse a las situaciones que encuentren los participantes del proyecto.

El equipo de Scrum consta de 3 roles principales. El Scrum Master será el encargado de dirigir el proyecto y comprobar que todo esta en orden, el Product Owner que representa al cliente, y el equipo de desarrollo.

Además Scrum incluye diferentes tipos de eventos los cuales ayudan a organizar el proyecto.

- Sprint - Es una unidad de tiempo que puede ser de una semana hasta un mes. Este Sprint tendrá una planificación y un objetivo que estará completado al terminar.
- Sprint Planning - Describe como se planea abordar cada uno de los sprints. Se realiza una reunión de máximo 8 horas en la que se planifica detalladamente que se planea hacer en ese Sprint.
- Sprint Goal - Es básicamente el objetivo del sprint, lo que se espera conseguir al final de este.
- Daily Scrum - Es una reunión diaria de unos 15 minutos que realiza el equipo de desarrollo para organizarse e informar al equipo que se dedicará a hacer cada persona ese día.

3.3. Scrum en el desarrollo de Talking Pics

Debido a que este proyecto era de una sola persona, se tuvo que adaptar Scrum para que fuera aplicable a este proyecto. En este caso no se hacían reuniones diarias, si no que al principio del día se comprobaba que era lo que quedaba por hacer del Sprint y se actualizaba la tabla de tareas. Esta tabla de tareas se organizó en una pared de Posts-Its, imitando la metodología Kanban que sirve para organizar tareas de una sola persona.

Es cierto que en Scrum, los Sprint son de una cantidad de tiempo fija, de forma que siempre son de una semana o varias. Sin embargo, debido a que a veces era demasiado trabajo para una sola persona, a veces había que alargar un poco ese Sprint para que se pudiera completar el objetivo fijado.

Cuando se empezó a desarrollar el proyecto, primero se obtuvieron todos los requisitos necesarios para la aplicación y sus correspondientes casos de uso. Además, se obtuvo un modelo de clases, para visualizar como se organizaría la base de datos.

Tras haber obtenido los requisitos, se procedió a realizar el diseño de todas las vistas (GUIs/Interfaces) de la aplicación con la herramienta Balsamiq.

La implementación de la aplicación se dividió en Sprints de una semana aproximadamente, en los que se desarrollaban funcionalidades de la API y/o de la interfaz. En total hubo 7 Sprints. Al final de estos, además de realizar pequeñas pruebas con Postman para comprobar que todo funcionaba correctamente, se realizaban actualizaciones en los modelos del proyecto, ya que el modelo inicial no pudo describir con total exactitud las diferentes acciones que se iban a poder realizar en la aplicación y todos los datos que iban a ser necesarios almacenar en la base de datos.

Finalmente, tras haber realizado toda la implementación y las pruebas, se mostró la aplicación al tutor para que diera su visto bueno y así dar paso al desarrollo de la memoria y entrega del proyecto.

Durante todo el proyecto se estuvo en contacto con el tutor para detallar el estado del proyecto, los problemas encontrados y decidir cuales serían los siguientes pasos para el próximo Sprint.

En la Figura 12 se puede ver cual ha sido el diagrama de Gantt del proyecto. En el se muestra que el desarrollo ha estado dividido en varios Sprints que han podido variar de tamaño según las necesidades de ese momento. En cada uno de los Sprints se iba avanzando en el desarrollo de la aplicación, implementando primero el backend y posteriormente el frontend. Para el backend se han usado 3 sprints, y el resto han sido para el frontend. También se realizaban pruebas al final de cada sprint, además de arreglar los errores que hubieran ocurrido durante el desarrollo.

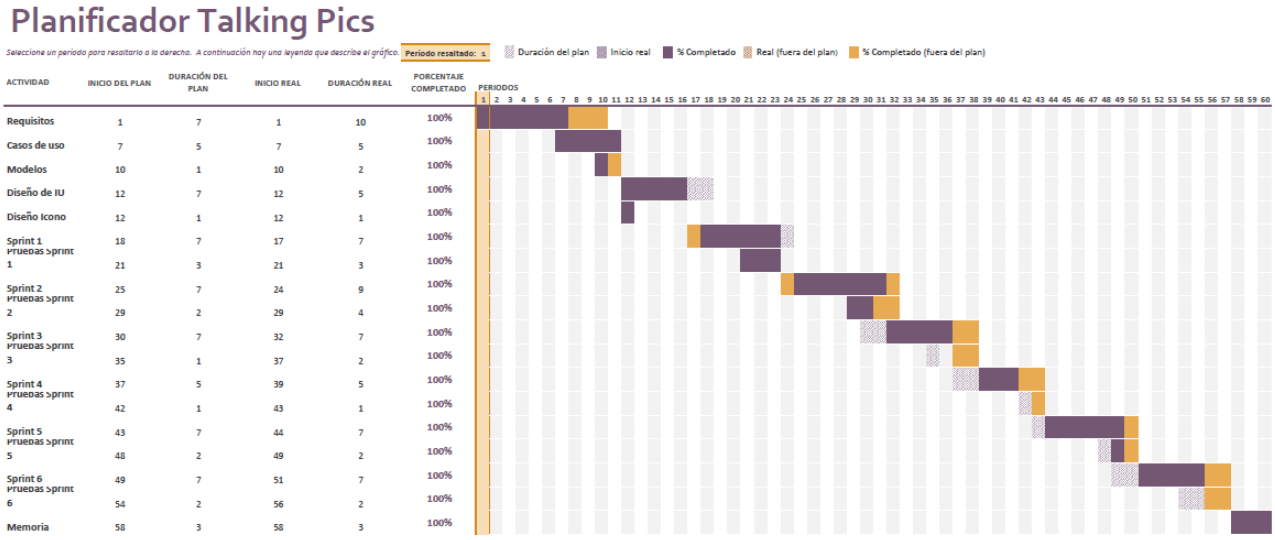


Figura 12: Diagrama de Gantt

4
Desarrollo

En esta sección se explicarán las diferentes fases del desarrollo, desde la obtención de requisitos hasta las pruebas realizadas para este proyecto.

4.1. Obtención de requisitos

Aquí se enumerarán los diferentes requisitos funcionales y no funcionales que se han obtenido para este proyecto, además de los respectivos casos de uso.

4.1.1. Requisitos funcionales

- RF-USER-01: El usuario podrá iniciar sesión en la aplicación mediante su cuenta de Google.
- RF-USER-02: El usuario podrá cerrar sesión en la aplicación.
- RF-USER-03: El usuario podrá seleccionar entre una gran variedad de imágenes para formar una frase.
- RF-USER-04: El usuario podrá buscar una imagen con un significado en concreto.
- RF-USER-05: El usuario podrá eliminar una de las imágenes seleccionadas de la frase de imágenes.
- RF-USER-06: El usuario podrá tomar una foto dentro de la aplicación.
- RF-USER-07: El usuario podrá añadir imágenes de su galería o cámara y asignarles un significado.
- RF-USER-08: El usuario podrá eliminar las imágenes que ha añadido manualmente a la aplicación.

- RF-USER-09: El usuario podrá modificar el significado de las imágenes que ha añadido a la aplicación.
- RF-IMG-01: Al finalizar de formar una “frase de imágenes”, se le mostrarán todas las imágenes seleccionadas.
- RF-IMG-02: Al finalizar de formar una “frase de imágenes”, se le mostrará al usuario la frase en texto que ha formado.
- RF-IMG-03: El usuario podrá almacenar sus “frases de imágenes” favoritas.
- RF-IMG-04: El usuario podrá eliminar sus “frases de imágenes” favoritas.
- RF-IMG-05: El sistema recomendará imágenes para añadir a continuación, tal y como haría un teclado con texto predictivo.
- RF-TXT-01: Las imágenes que tome el usuario de un texto se traducirán a “frases de imágenes” para el mejor entendimiento del usuario.

4.1.2. Requisitos no funcionales

Requisitos de aspecto

- RNF-INT-01: La interfaz se diseñará con la siguiente paleta de colores:
    1. #5D736B
    2. #B4BFB7
    3. #D99D8F
    4. #F2F2F2
    5. #262626
- RNF-INT-02: La interfaz deberá ser “responsive”, de forma que se adapte a diferentes dispositivos móviles de forma adecuada.

Requisitos de funcionamiento

Seguridad critica

Figura 13: Paleta de colores

- RNF-SC-01: El sistema debe cumplir con el RGPD.

Requisitos operacionales

Entorno físico

- RNF-EF-01: El sistema será operativo en móviles con sistema operativo Android.

Entorno tecnológico

- RNF-ENT-01: El sistema debe ser una aplicación móvil.
- RNF-ENT-02: Las imágenes se almacenarán en formato PNG, JPG o JPEG.

Aplicaciones relacionadas

- RNF-AR-01: El sistema estará relacionado con la API de ARASAAC.
- RNF-AR-02: La aplicación estará relacionada con el sistema de identificación de Google
- RNF-AR-03: La aplicación estará relacionada con una API de reconocimiento de texto en imágenes.

Requisitos de documentación

Manual de usuario

- RNF-MU-01: La aplicación contará con un manual de usuario.

- RNF-MU-02: El manual de usuario podrá ser descargado desde la aplicación.
- RNF-MU-03: El manual de usuario estará en formato PDF.

Guía de instalación y configuración

- RNF-GIC-01: Se entregará una guía de instalación junto con el proyecto.

4.1.3. Casos de uso

En esta sección se verán los diferentes casos de uso creados y los requisitos que están relacionados con ellos. En el diagrama de casos de uso de la Figura 14 se muestran todos los diferentes escenarios que puede realizar un actor. Los casos de uso están representados por los códigos que se indican más adelante en las tablas.

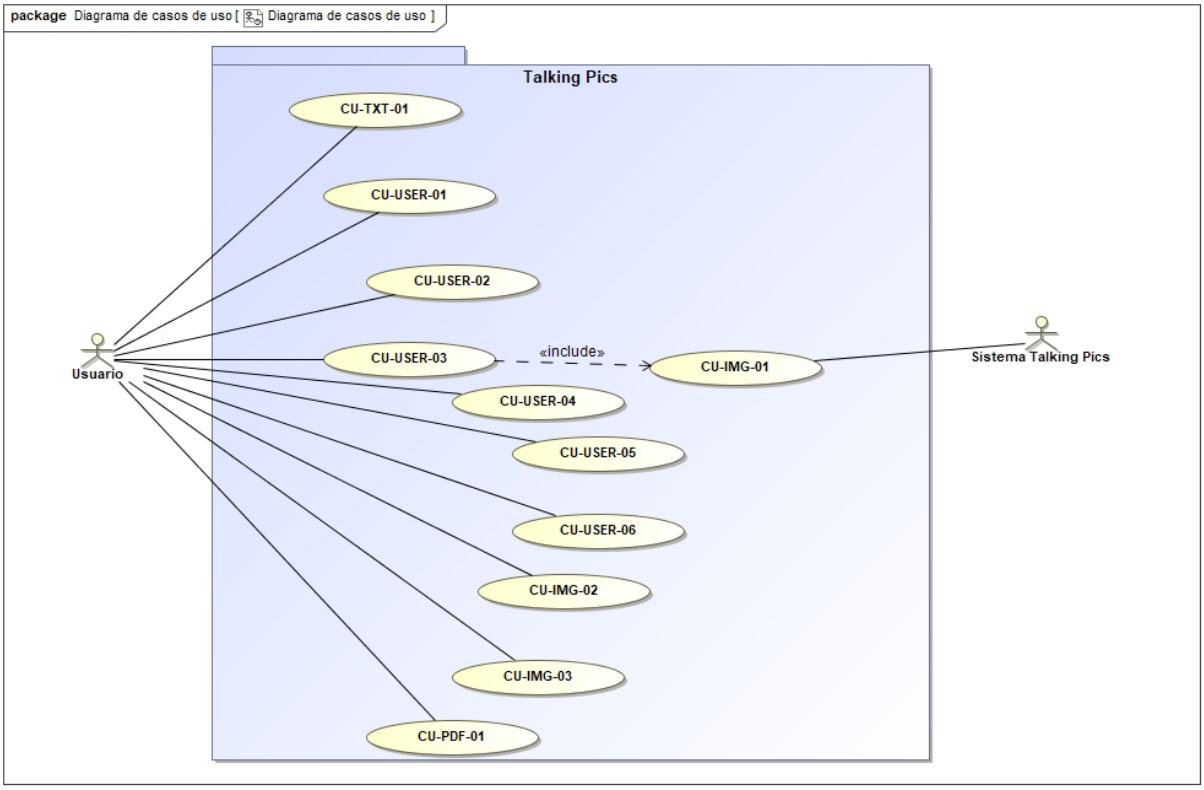


Figura 14: Diagrama de casos de uso

Código	CU-USER-01
Descripción	Inicio de sesión en la aplicación
Precondición	El usuario no ha iniciado sesión.
Postcondición	El usuario está dentro de la aplicación con una sesión iniciada.
Prioridad	Media
Requisitos relacionados	RF-USER-01, RNF-SC-01, RNF-AR-02.
Escenario principal	
1. El usuario abre la aplicación Talking Pics. 2. La aplicación le indica al usuario que debe iniciar sesión con Google. 3. Se le mostrarán al usuario las diferentes cuentas de Google con las que puede iniciar sesión 4. El usuario seleccionará una cuenta 5. El usuario habrá iniciado sesión 6. El usuario será redirigido a la pantalla principal de la aplicación	
Escenario alternativo	
▪ 5.a.1. No se ha conseguido iniciar sesión ▪ 5.a.2. El usuario será redirigido a la página de inicio	

Cuadro 1: CU-USER-01

Código	CU-USER-02
Descripción	Cierre de sesión en la aplicación
Precondición	El usuario está en la aplicación con una sesión iniciada.
Postcondición	El usuario ha terminado su sesión en la aplicación.
Prioridad	Media
Requisitos relacionados	RF-USER-02, RNF-SC-01, RNF-AR-02
Escenario principal	
1. El usuario selecciona en el menú de la aplicación “Cerrar sesión” 2. La sesión del usuario se cierra 3. El usuario es redirigido a la ventana inicial	
Escenario alternativo	
▪ 2.a.1. No se ha conseguido cerrar la sesión ▪ 2.a.2. Se indica al usuario que ha ocurrido un error al cerrar su sesión	

Cuadro 2: CU-USER-02

Código	CU-USER-03
Descripción	Creación de frase de imágenes
Precondición	El usuario tiene la sesión iniciada
Postcondición	El usuario ha creado una frase de imágenes
Prioridad	Alta
Requisitos relacionados	RF-USER-03, RF-USER-04, RF-USER-05, RF-IMG-05, RNF-AR-01

Escenario principal

1. El usuario selecciona “Crear una nueva frase de imágenes”
2. El usuario introducirá una palabra en el buscador
3. Aparecerán imágenes que coincidan con esa palabra
4. El usuario seleccionará una de las imágenes
5. En una sección más abajo, aparecerán imágenes recomendadas para añadir a la frase de imágenes
6. Se repetirá el proceso hasta que el usuario haya formado su frase de imágenes
7. El usuario pulsará el botón de finalizar

Escenario alternativo

- 4.a.1. El usuario pulsa de nuevo en una de las imágenes seleccionadas
- 4.a.2. La imagen será eliminada de la frase de imágenes
- 7.a.1. El usuario pulsa el botón .ªtrás"
- 7.a.2. Se le redirigirá a la página de inicio

Cuadro 3: CU-USER-03

Código	CU-IMG-01
Descripción	Presentación de frase de imágenes
Precondición	El usuario acaba de crear una frase de imágenes y tiene la sesión iniciada
Postcondición	Se mostrará al usuario la frase de imágenes formada
Prioridad	Alta
Requisitos relacionados	RF-IMG-01, RF-IMG-02
Escenario principal	
1. Se le muestra al usuario todas las imágenes, por orden, que han sido seleccionadas 2. Justo debajo se le mostrará la frase en texto que ha sido formada	
Escenario alternativo	
-	

Cuadro 4: CU-IMG-01

Código	CU-USER-04
Descripción	Fotos propias del usuario
Precondición	El usuario tendrá la sesión iniciada
Postcondición	Una imagen tomada del usuario será añadida a la aplicación
Prioridad	Media
Requisitos relacionados	RF-USER-06, RF-USER-07, RNF-ENT-02
Escenario principal	
1. El usuario selecciona “Añadir una nueva foto” 2. La aplicación preguntará si prefiere hacerlo desde la cámara o la galería 3. El usuario elige la cámara 4. El usuario toma una nueva foto 5. La aplicación le muestra la foto que ha tomado y le pregunta por su significado 6. El usuario introduce un significado 7. El usuario pulsa aceptar 8. La imagen y su significado son almacenadas	
Escenario alternativo	

- 3.a.1. El usuario elige la galería
- 3.a.2. El usuario selecciona una foto de su galería
- 3.a.3. La aplicación le muestra la foto que ha elegido y le pregunta por su significado
- 3.a.4. El usuario introduce un significado
- 3.a.5. El usuario pulsa aceptar
- 3.a.6. La imagen y su significado son almacenadas

Cuadro 5: CU-USER-04

Código	CU-USER-05
Descripción	Eliminación de imágenes del usuario
Precondición	El usuario tiene la sesión iniciada y tiene alguna imagen con significado almacenada
Postcondición	Una foto con significado ha sido eliminada
Prioridad	Media
Requisitos relacionados	RF-USER-08
Escenario principal	
1. El usuario selecciona “Visualizar mis imágenes añadidas” 2. La aplicación redirige al usuario a una ventana donde puede visualizar que imágenes él ha añadido manualmente 3. El usuario selecciona una imagen 4. El usuario pulsa “Eliminar”	
Escenario alternativo	
-	

Cuadro 6: CU-USER-05

Código	CU-USER-06
Descripción	Modificación de imágenes del usuario
Precondición	El usuario tiene la sesión iniciada y tiene alguna imagen con significado almacenada
Postcondición	Una foto con significado ha sido modificada
Prioridad	Media
Requisitos relacionados	RF-USER-09
Escenario principal	
1. El usuario selecciona “Visualizar mis imágenes añadidas” 2. La aplicación redirige al usuario a una ventana donde puede visualizar que imágenes él ha añadido manualmente 3. El usuario selecciona una imagen 4. El usuario pulsa modificar 5. La aplicación le redirige a una ventana donde podrá modificar el significado de la imagen 6. El usuario introduce el nuevo significado de la imagen 7. El usuario pulsa “Guardar”	
Escenario alternativo	
▪ 7.a.1. El usuario pulsa “Cancelar ▪ 7.a.2. El usuario será redirigido a la ventana donde se encuentran todas las imágenes almacenadas	

Cuadro 7: CU-USER-06

Código	CU-IMG-02
Descripción	Almacenamiento de frases favoritas
Precondición	El usuario tiene la sesión iniciada y ha terminado de crear una frase de imágenes
Postcondición	El usuario ha almacenado una frase de imágenes
Prioridad	Media
Requisitos relacionados	RF-IMG-03
Escenario principal	
1. El usuario pulsará el botón de “Almacenar como frase favorita” 2. La frase será automáticamente almacenada	
Escenario alternativo	
▪ 2.a.1. La frase no se ha podido almacenar ▪ 2.a.2. Se mostrará un mensaje de error al usuario iniciada	

Cuadro 8: CU-IMG-02

Código	CU-IMG-03
Descripción	Eliminar frases favoritas
Precondición	El usuario ha iniciado sesión y tiene frases favoritas almacenadas
Postcondición	Una frase favorita ha sido almacenada
Prioridad	Media
Requisitos relacionados	RF-IMG-04
Escenario principal	
1. El usuario se dirigirá a “Frases favoritas” 2. Se le mostrarán todas las frases favoritas almacenadas 3. El usuario seleccionará una frase favorita 4. El usuario pulsará “Eliminar” 5. La frase será eliminada	
Escenario alternativo	
▪ 5.a.1. La frase no pudo ser eliminada ▪ 5.a.2. Se mostrará un mensaje de error al usuario iniciada	

Cuadro 9: CU-IMG-03

Código	CU-TXT-01
Descripción	Texto a Imágenes
Precondición	El usuario tiene la sesión iniciada
Postcondición	Una frase ha sido traducida a frase de Imágenes
Prioridad	Media
Requisitos relacionados	RF-TXT-01, RNF-AR-03
Escenario principal	
1. El usuario selecciona “Traducir frase a frase de imágenes” 2. El usuario toma una foto 3. La aplicación traduce la imagen a frase de imágenes	
Escenario alternativo	
▪ 3.a.1. No se ha podido traducir la frase de imágenes ▪ 3.a.2. Se mostrará un error ▪ 3.a.3. Se redirigirá al usuario a la pantalla principal	

Cuadro 10: CU-TXT-01

Código	CU-PDF-01
Descripción	Manual de usuario
Precondición	El usuario tiene la sesión iniciada
Postcondición	El usuario descarga el manual de usuario
Prioridad	Baja
Requisitos relacionados	RNF-MU-01, RNF-MU-02, RNF-MU-03
Escenario principal	
1. El usuario selecciona “Más información” 2. El usuario es redirigido a una ventana donde hay más información sobre la aplicación 3. El usuario selecciona “Manual de usuario” 4. El manual de usuario es descargado como PDF	
Escenario alternativo	
▪ 4.a.1. El manual de usuario no se ha podido descargar correctamente ▪ 4.a.2. Se muestra un error al usuario	

Cuadro 11: CU-PDF-01

4.2. Diseño

En esta sección se mostrará el modelo de clases del proyecto, de la base de datos y el diseño que se realizó para la interfaz.

4.2.1. Modelo de clases

El modelo de clases, tal y como la implementación, ha ido mejorando a lo largo de los Sprints.

Al principio, era un modelo muy sencillo, tal y como se puede ver en la Figura 15. Simplemente mostraba los datos que iban a tener las entidades, sin tener ninguna operación que representara los Endpoints que iba a tener la API. Tampoco mostraba los Endpoints que se iban a usar para llamar a la API de predicciones o para usar la herramienta de Google Vision. Estas herramientas se han explicado en la sección de API principal y los endpoints desde donde se usan se explicaran en la sección de Desarrollo.

El modelo constaba de 5 clases y una interfaz. Lo que se buscaba con esta interfaz era que las frases de imágenes estuvieran compuestas de una lista de imágenes. En el modelo definitivo se podrá apreciar que, finalmente, se prescindió de esta interfaz por razones que se explicarán más adelante. Las clases de este modelo son:

- *Users* - Clase para almacenar los usuarios, que constaba de *_id*, un atributo *dificil* el cual sirve para indicar si la pantalla principal va a estar en un formato u otro, tal y como se explica en la sección de 4.2.3, y un *email*. La clase *Users* esta relacionada con las clases *ImagePhrases* representando las imágenes que han sido almacenadas por este usuario, y con la clase *ImagePhrases* tiene dos relaciones, *Almacenadas* y *Creadas* que representan las frases que han sido almacenadas en la base de datos y las que ha sido creadas por el usuario respectivamente.

- *SavedImages* - Clase para almacenar las imágenes guardadas por el usuario en la base de datos. Consta de *_id*, una *url* donde esta almacenada la imagen, y *significado*.

- *ImagePhrases* - Clase para almacenar las frases de imágenes que guarda el usuario como favoritas. Para ello se tiene un array de *Imagenes*, *meaning* (significado de la frase), un *_id* y un booleano que nos indica si es favorita o no.

- *ARASAACImages* y *Keywords* - Son dos clases que nos permiten obtener los pictogramas que obtendremos de la API de Arasaac. Los atributos que aparecen en estas clases son los que nos devuelve Arasaac cuando le preguntamos sobre un pictograma.

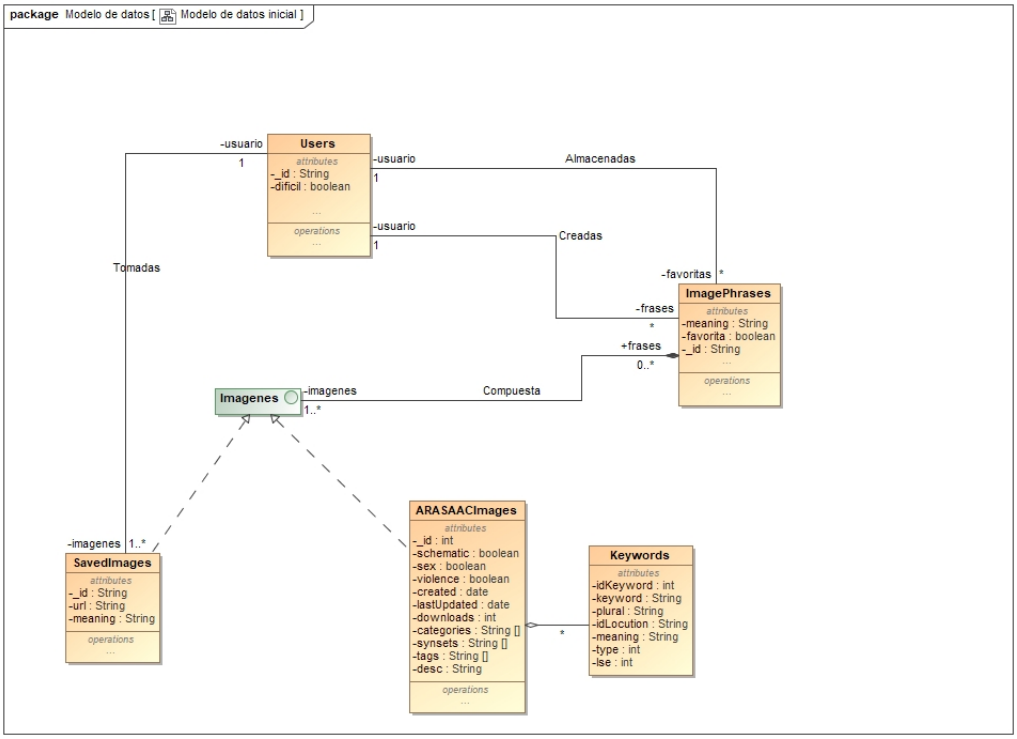


Figura 15: Modelo de clases inicial

Tras haber realizado varias iteraciones y completando Sprints, se obtuvo el siguiente modelo (Figura 16). En este modelo si que se representan los diferentes Endpoints que iba a tener nuestra API, además de añadir las clases para obtener las predicciones y llamar a Google Vision. También se añaden todos los atributos necesarios para las diferentes entidades.

Tal y como se ha mencionado antes, en este modelo hemos prescindido de la interfaz Imágenes y hemos eliminado la relación entre las entidades *ImagePhrases* y *SavedImages*. La razón para esto es por que, como el objetivo es no almacenar las imágenes obtenidas de la API de Arasaac, bastaría con tener almacenado solo el id y hacer una llamada a la API externa para obtener la imagen con esa información. Sin embargo, en Java no se puede crear un array de dos tipos diferentes, por lo que se optó por crear la lista de *Strings images* en la clase ImagePhrases, la cual se encargaría de almacenar los *id*s de las imágenes que forman esa frase, ya sean de las imágenes almacenadas en nuestra base de datos o en la API externa de Arasaac. En este modelo, las clases creadas son:

- *Users* - Almacena los mismos atributos que en el modelo anterior (15), pero el atributo *email* se ha sustituido por el id proporcionado por Google y de esta forma no tendremos el email, que es un dato sensible, almacenado en la base de datos. Se han añadido las diferentes operaciones, que representan los endpoints (4), que se pueden realizar sobre esta clase usuario. La clase *Users* esta relacionada con las clases *ImagePhrases* representando las imágenes que han sido almacenadas por este usuario, y con la clase *ImagePhrases* tiene dos relaciones, *Almacenadas* y *Creadas* que representan las frases que han sido almacenadas en la base de datos y las que ha sido creadas por el usuario respectivamente.

- *SavedImages* - Almacena los mismos atributos que en el modelo anterior (15) y se han añadido los diferentes endpoints (4) que se podran usar sobre esta clase.

- *ImagePhrases* Tiene los mismos atributos que en el modelo anterior (15), sin embargo ya no se tiene almacenada una lista de imágenes, si no que se ha almacenado una lista de *Strings*. La razón de esto se ha explicado en el párrafo anterior (4.2.1). Además se han añadido los diferentes endpoints (4)

- *ARASAACImages* y *Keywords* - Estas clases siguen con los mismos atributos, simplemente se les ha añadido los enpoints que se explicarán más adelante en la sección de Desarrollo

- *Prediction* - Es la clase que se usará para obtener las predicciones de las frases. Esta clase no tiene atributos, solamente enpoints (4).

- *GoogleVision* - Es la clase que se usará para obtener el texto de las imágenes, solamente tiene endpoints(4).

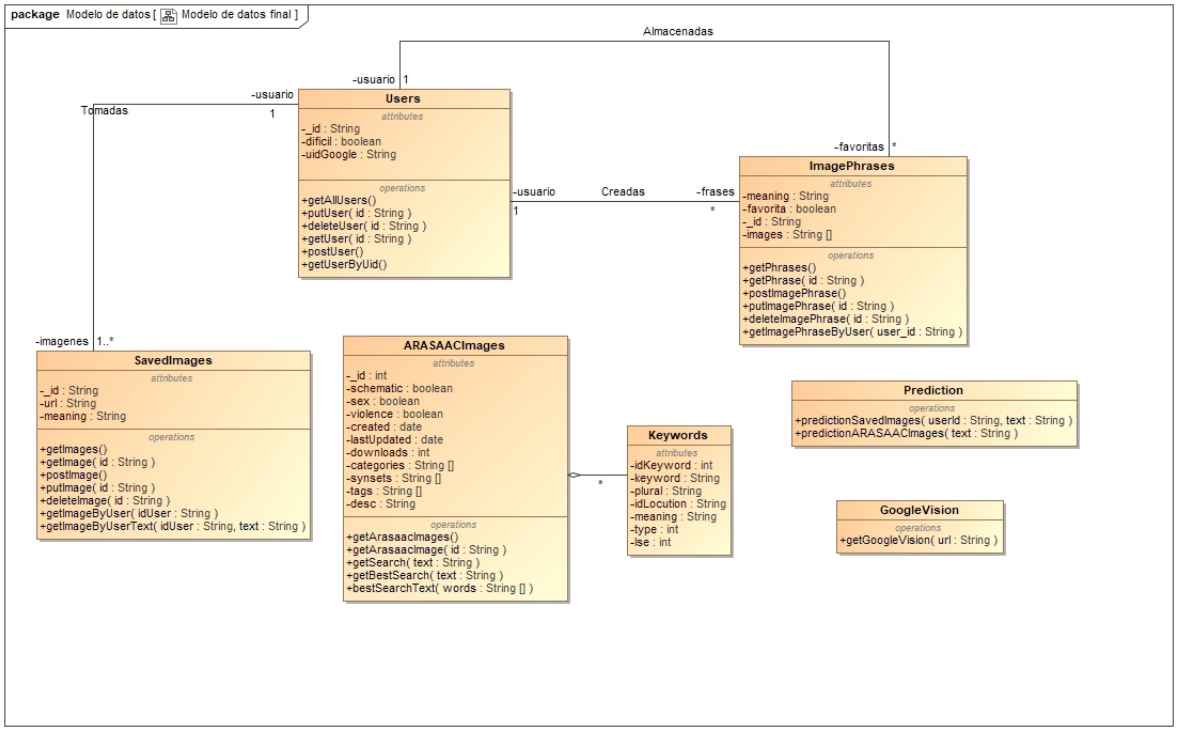


Figura 16: Modelo de clases final

4.2.2. Modelo de base de datos

Al inicio del proyecto, el modelo de la base de datos era tan simple que no se almacenaba ninguna información del usuario, tal y como se puede apreciar en la Figura 17. También se tenía una relación entre las imágenes almacenadas y las frases de imágenes, pero como se explicó anteriormente, esto no era viable debido al modo en el que funcionaba Java, ya que hacía que extraer o subir datos fuera mucho más complicado.

En la base de datos, inicialmente, se iban a almacenar las siguientes entidades:

- *Users* - el único atributo que se pensaba almacenar en un principio es un _id. Aparte, esta clase esta relacionada con las clases *SavedImages* y *ImagePhrases*, que indica que imágenes y frases se han almacenado para cada usuario

- *SavedImages* - En esta clase se iba a almacenar los _id, url donde está la imagen y el significado de esta. Esta clase está relacionada con *ImagePhrases*, ya que las frases de imágenes van a estar compuestas por las imágenes almacenadas en la base de datos.

- *ImagePhrases* - En esta entidad se almacena el significado, si la frase es favorita o no, y

el _id.

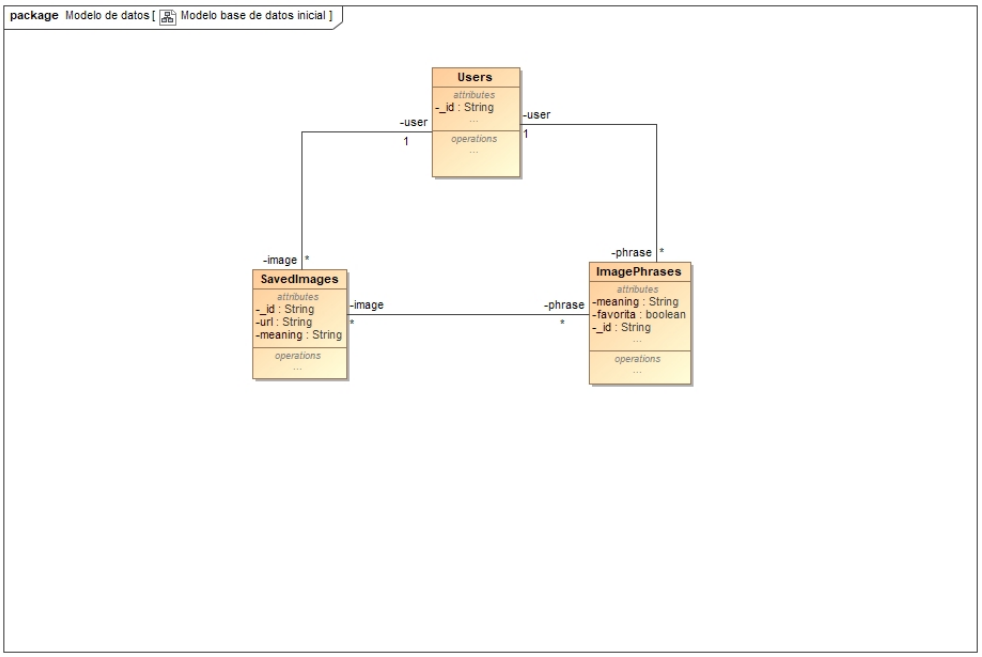


Figura 17: Modelo de base de datos inicial

Tras actualizar realizar varias iteraciones, se llegó a la conclusión de que la mejor manera de almacenar los datos era la mostrada en la Figura 18, por lo que se eliminó la relación entre imágenes y frases y se creo el array *images* dentro de *ImagePhrases*.

Además, en la entidad de usuario se añadieron dos atributos más:

1. *dificil*: este atributo sirve para indicar si la pantalla principal de la aplicación va a estar en modo sencillo (sin frases, solo iconos) o en modo difícil. Esta funcionalidad se explicara en la sección de Diseño interfaz

2. *uidGoogle*: hemos almacenado el uid que nos proporciona Google cuando iniciamos sesión con el usuario, ya que de este modo podemos ver si el usuario ya existía previamente en nuestra aplicación o es un usuario nuevo, además de así poder obtener su *_id* y todas sus imágenes o frases almacenadas.

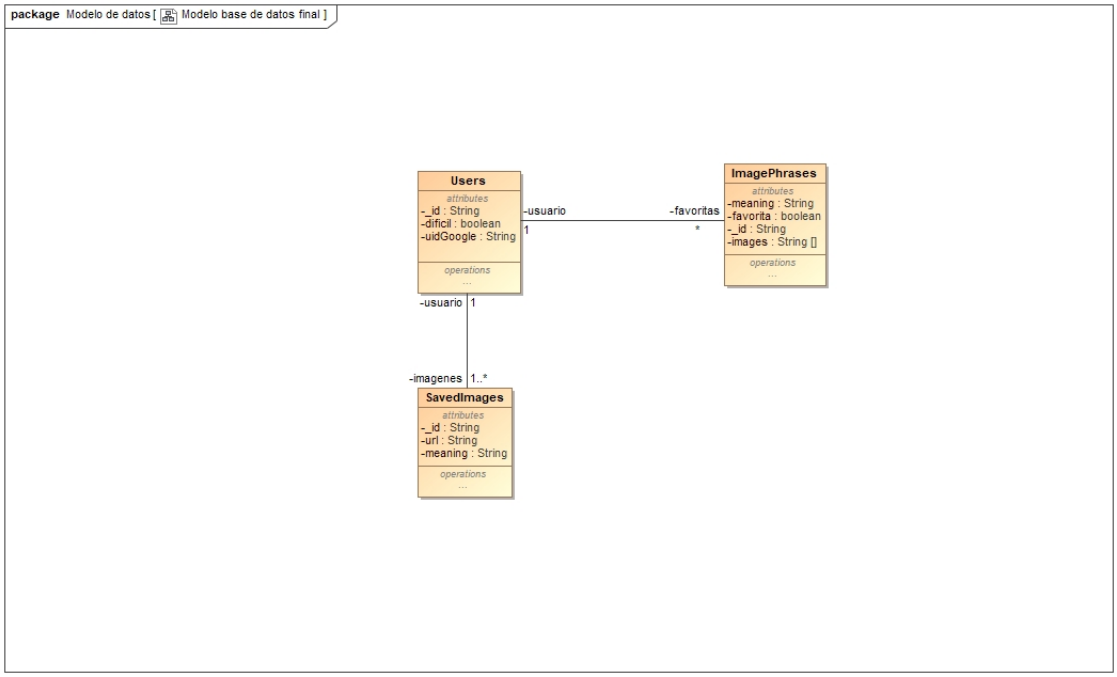


Figura 18: Modelo de base de datos final

4.2.3. Diseño interfaz

El diseño de la interfaz se realizó usando la herramienta Balsamiq.

Al inicio del diseño, se pensó que la mejor idea era que para todas las pantallas existiera una sustituta con iconos en vez de texto. Sin embargo, a lo largo del desarrollo se hicieron las pantallas tan sencillas que finalmente solo se necesitó hacer una versión fácil de la página principal de la aplicación.

A continuación se muestra el diseño de las pantallas que finalmente se usaron al desarrollar la aplicación. **Inicio de sesión**

En la figura 19 se muestra el diseño que se encontrará el usuario nada más entrar en la aplicación que le permitirá iniciar sesión.

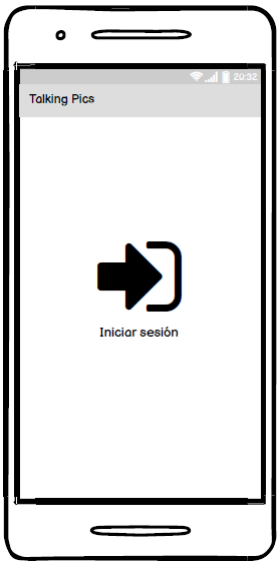
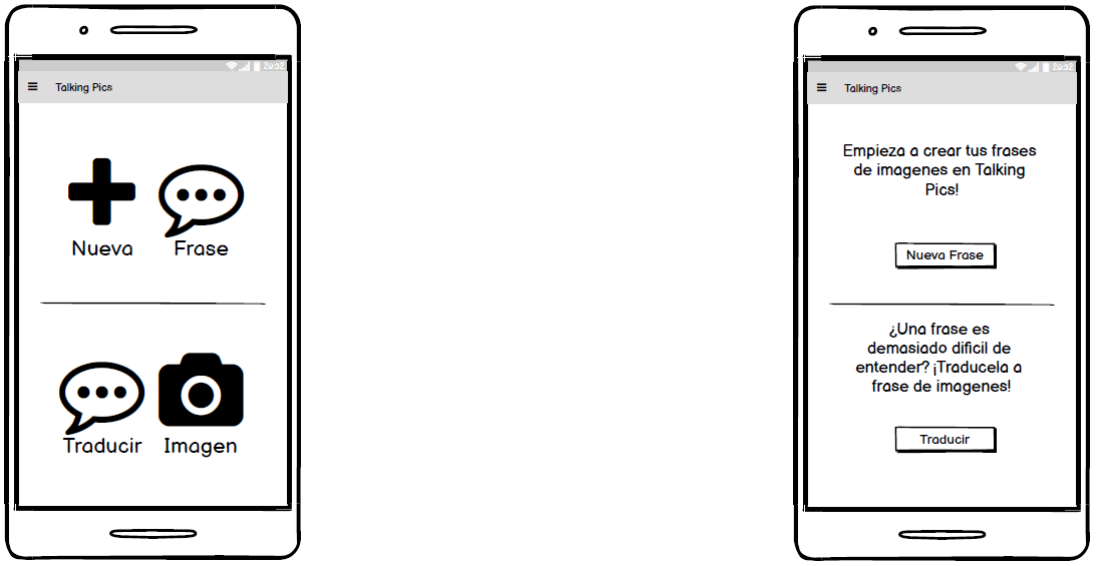


Figura 19: Diseño inicio de sesión

Ventana principal

Aquí se muestran las ventanas (20) que se mostrarán al usuario según tenga almacenado en la base de datos un *false* o un *true* en el atributo *dificil*. Este atributo lo que nos definirá es, básicamente, como se muestra esta ventana al usuario. Desde esta ventana el usuario puede navegar pulsando los botones 'Nueva Frase' o 'Traducir'.



(a) Diseño principal fácil

(b) Diseño principal difícil

Figura 20: Diseño ventana principal

Construir frase

En esta pantalla (21) podrán construir una frase de imágenes añadiendo o eliminando imágenes. Cuando seleccione una imagen, al principio de la galería aparecerán nuevas imágenes que la aplicación le recomienda añadir.

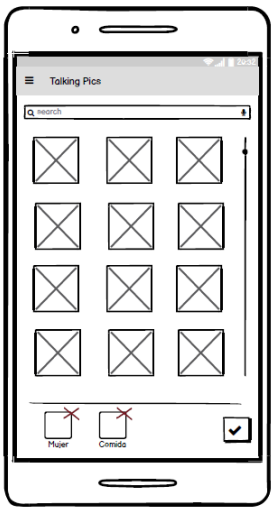


Figura 21: Diseño ventana construir frase

Frase construida

En esta ventana (23) se muestra la frase que ha construido el usuario y le permite añadir esta frase a favoritos.

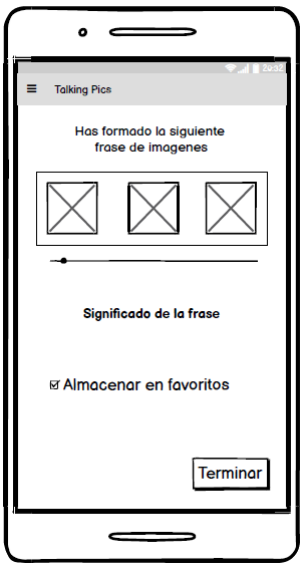


Figura 22: Diseño ventana frase construida

Traducir imagen

Desde esta ventana podrá traducir el texto que aparezca en una imagen a un pictograma por cada palabra.

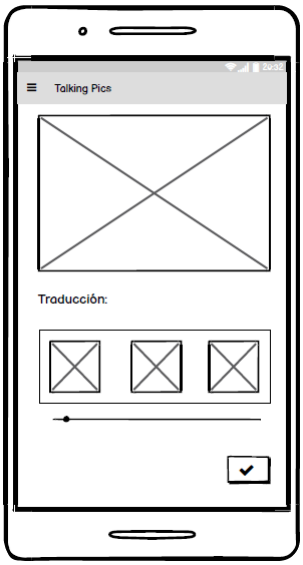


Figura 23: Diseño ventana traducir imagen

Menú

Desde este menú (24) el usuario podrá navegar entre las diferentes funcionalidades de la aplicación. Al principio se pensaba añadir un perfil a la aplicación, pero finalmente se prescindio de esta funcionalidad.

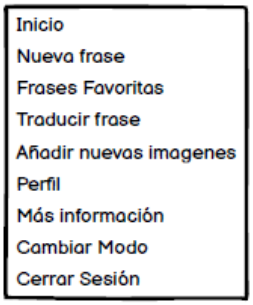


Figura 24: Diseño menú

Frases favoritas

En esta pantalla (25) se muestran las frases favoritas que tiene almacenadas el usuario.

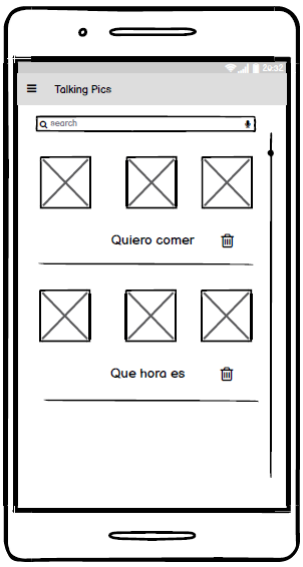


Figura 25: Diseño ventana frases favoritas

Imágenes almacenadas

En esta ventana se muestran las imágenes personalizadas que tiene el usuario almacenadas. Desde aquí podrá eliminar y editar las imágenes.

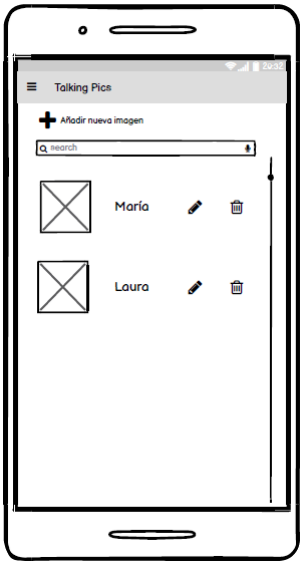


Figura 26: Diseño ventana imágenes almacenadas

Añadir/Editar imagen almacenada

En esta ventana (27) podrá editar el significado de sus imágenes almacenadas.

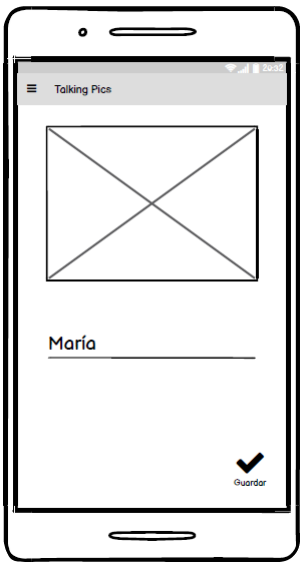


Figura 27: Diseño ventana editar imágenes

Más información

En esta ventana (28) se mostrará información adicional de la aplicación.

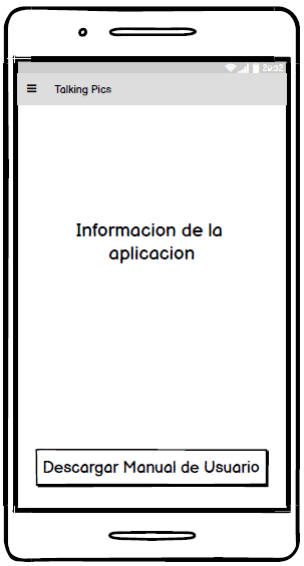


Figura 28: Diseño ventana información

Cambiar modo

En esta ventana (29) se muestra como se pretendía cambiar de modo. Al principio del proyecto, para cambiar de modo simplemente se iba a mostrar un modal, pero finalmente se decidió crear dos ventanas para cambiar de modo. Una para cuando se quiera cambiar de fácil a difícil, donde tendrá que confirmar la acción el usuario escribiendo la palabra CAMBIAR, y otra para cambiar de difícil a fácil, donde el usuario solo tendrá que pulsar un botón.

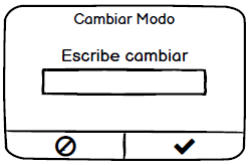


Figura 29: Diseño ventana cambiar modo

4.2.4. Icono

Con este icono (30) se ha querido representar la finalidad de la aplicación, poder comunicarse mediante imágenes. Por eso el poner dos nubes de dialogo con imágenes dentro.

El icono fue diseñado usando la paleta de colores mencionada en los requisitos no funcionales.

Figura 30: Icono Talking Pics

4.3. Implementación

4.3.1. API predicciones

Esta sencilla API fue desarrollada en Python con uso de Flask. Consta de un único Endpoint, con el que podremos obtener varias predicciones para la siguiente palabra de un texto.

Se puede acceder a este Endpoint mediante el enlace https://flask-api-predictions.herokuapp.com. El único parámetro que tenemos que pasar a este Endpoint es 'text', el cual es una cadena/string, que será el texto que utilizaremos para obtener una predicción. Un ejemplo de esto sería https://flask-api-predictions.herokuapp.com/?text=Para%20solucionar%20los

En la implementación del endpoint (4.3.1) se puede ver que primero obtiene el texto que le pasan por parámetro. Si el texto que se pasa es nulo o vacío, el endpoint devolverá directamente un array vacío. Si no es vacio o nulo, al final del texto se añadirá el String '[MASK]'. Esto indicará al modelo que la palabra que tiene que predecir se encuentra en esa posición. En nuestro caso, esta máscará estará siempre al final de la frase, por que la funcionalidad que buscamos es que le recomiende la siguiente palabra que tiene que añadir el usuario. Tras esto hace una llamada a la API del modelo, el cual calcula cual puede ser la siguiente palabra y devuelve los tokens que ha obtenido, indicando la probabilidad que tiene cada uno de ellos.

```
1
2   @app.route("/", methods=["GET"])
3   def predictions():
4
5       text = request.args.get('text')
6
7       if text is None:
8           return {"tokens" : []}
9       else:
10          text += " [MASK]"
11
12          data = json.dumps({"inputs": text})
13          response = requests.request("POST", API_URL, headers=headers,
    data=data)
14
15          return {"tokens":json.loads(str(response.content.decode("utf-8"
```

```
))))}
```

Listing 1: Endpoint Predicciones

La API usada que contiene el modelo se puede acceder desde el enlace "https://api-inference.huggingface.co/models/dccuchile/bert-base-spanish-wwm-uncased". En un principio, el modelo iba a ser desarrollado completamente desde 0, y de hecho se inició el desarrollo, pero debido a su alta complejidad y que no era un objetivo prioritario en el TFG, finalmente se descartó este desarrollo y se decidió hacer una llamada a este modelo a través de su API.

Es cierto que esta llamada se podría haber realizado desde la API principal que explicaremos más adelante. Sin embargo, en la página oficial del modelo se recomendaba que recomendaba de que la llamada se hiciera desde Python. Además haberla desarrollado en este lenguaje nos da pie a mejoras que se explicarán en la sección de Lineas Futuras.

Finalmente, para poder publicar esta aplicación se uso Heroku. Para ello se tuvo que crear un fichero 'requirements.txt' el cual contiene todos los paquetes que son necesarios para este proyecto, y el fichero Procfile que es necesario para indicarle a Heroku la configuración que va a tener esta aplicación. En este caso es una aplicación web, lo cual se denota de la siguiente forma:

```
1       web: gunicorn flask_api:app
2
```

Listing 2: Procfile Python

4.3.2. API principal

Esta API fue desarrollada en Java con el framework Spring. Las clases creadas fueron divididas en varios paquetes. Se explicaran cada uno de ellos y las clases que contienen.

Paquete firebase

Contiene las clases FirebaseInizialization y Token. La ultima clase se encarga de dar forma a los Tokens que se obtienen al comprobar si el accessToken que nos envian desde el frontend es valido o no.

El metodo validate token de la clase FirebaseInizialization es el que se encarga de llamar a la url https://www.googleapis.com/oauth2/v1/tokeninfo?access_token=token para comprobar si es válido o no.

Por ultimo, el metodo initialization de la clase FirebaseInizialization se encarga de obtener la conexión con la base de datos en la nube a partir de un archivo JSON que tendremos almacenado en la raíz del proyecto. Este archivo JSON se consigue al crear un proyecto web en la consola de firebase.

Paquete exceptions

Este paquete contiene una única clase, ApiException. El código de esta clase es el siguiente:

```
1       @ResponseStatus(code =org.springframework.http.HttpStatus.
    BAD_REQUEST, reason = "Invalid token")
2       public class ApiException extends Exception{
3
4           public ApiException() {
5               super();
6           }
7
8           public ApiException(String message) {
9               super(message);
10          }
11      }
12
```

Listing 3: Código exceptions

Paquete helper

Este paquete contiene una única clase, Helper, la cual tiene un solo método. Este método se encarga de comprobar si el id de usuario que pasamos por parámetro existe en nuestra base de datos. La cabecera del método es la siguiente:

```
1       public static boolean usuarioExiste(String userId){
2           ...
3       }
4
```

Listing 4: Codigo cabecera usuarioExiste

Paquete entities

Este paquete contiene las clases necesarias para crear objetos de las entidades almacenadas en la base de datos o que obtendremos a partir de llamar a otras APIs.

Clase ARASAACImage y KeyWord

Estas dos clases se usan para obtener objetos al hacer la llamada a la API de Arasaac. La clase Arasaac contiene un atributo List<Keyword>, por lo que estas clases están relacionadas, tal y como se mostró en el modelo de clases de la API.

Clase Prediction y Token

Estas clases se usan para obtener objetos al hacer una llamada a la API de predicciones. La clase Prediction contiene un atributo List<Token>por lo que estas dos clases estan relacionadas.

Clase User

Gracias a esta clase, podemos dar formato a los usuarios que obtenemos de la base de datos de Firebase. Está desarrollada según el modelo de la base de datos.

Clase SavedImage

Esta clase sirve para dar formato a las imágenes almacenadas en la base de datos. Tiene los atributos especificados en el modelo de la base de datos.

Clase ImagePhrase

La clase ImagePhrase sirve para dar formato a las frases de imagenes que obtenemos de la base de datos, con los atributos especificados en el modelo de la base de datos.

Paquete services

Clase GoogleVisionService

Esta clase contiene un solo método para inicializar Google Vision desde un JSON que contiene nuestras credenciales.

Clases ImageARASAACService, ImagesPhraseService, PredictionService, SavedImageService, UserService

Estas clases contienen los diferentes métodos que se van a llamar desde los controladores.

Paquete controllers

Clase GoogleVisionController

Este controlador tiene un único Endpoint que se puede llamar desde el enlace https://talking-pics.herokuapp.com/api/v1/cloud-vision. A este Endpoint se le tiene que mandar los siguientes argumentos:

1. url ->URL de la imagen de la que se quiere sacar el texto.

2. accessToken ->Token de acceso del usuario para poder usar los Endpoint.

Clase ImageARASAACController

Este controlador tiene 5 endpoints:

1. Get All Images - Se llama desde la url https://talking-pics.herokuapp.com/api/v1/arasaac-images pasando los siguientes argumentos:

   a) accessToken

   b) page ->es un entero que se manda para obtener las diferentes páginas de los datos.

2. Get Image - Se llama desde la url https://talking-pics.herokuapp.com/api/v1/arasaac-images/{idImage}. Este Endpoint solo recibe un parámetro, accessToken.

3. Get Search - Gracias a este endpoint se pueden obtener todas las imágenes que contienen la palabra o frase que le enviamos por parametro. Se accede desde la url https://talking-pics.herokuapp.com/api/v1/arasaac-images/search, añadiendo los siguientes parámetros:

   a) locale ->código del idioma que vamos a usar

   b) searchText ->el texto que se va a buscar

   c) accessToken

4. Get Best Search - Con este Endpoint obtenemos las imágenes que son iguales al texto que pasamos por parámetro. Se accede con la url https://talking-pics.herokuapp.com/api/v1/arasaac-images/best-search y se pasan los siguientes parámetros:

   a) locale

   b) searchText

   c) accessToken

5. Get Best Search Text - Con este Endpoint obtenemos una imagen para cada una de las palabras que se pasan por parámetros. Se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/arasaac-images/best-search-text y se tienen que pasar los siguientes parametros:

a) words ->array de palabras que se van a buscar

b) accessToken

Clase ImagesPhraseController

Este controlador sirve para obtener las frases de imágenes en la base de datos

1. Get ImagesPhrases - Obtiene todas las frases de imágenes de la base de datos. Se accede desde el enlace https://talking-pics.herokuapp.com/api/v1/images-phrases y se tiene que pasar el parámetro accessToken

2. Get ImagesPhrases - Obtiene la información de una frase en concreto y se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/images-phrases/{phraseId} y se tiene que pasar el parámetro accessToken

3. Get ImagesPhrases by User - Obtiene la información de las frases según su usuario. Se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/images-phrases/by-user y tienen que pasarse los siguientes argumentos:

   a) user_id

   b) accessToken

4. Post ImagesPhrases - Es un Endpoint para almacenar una frase y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/images-phrases, pasando el parámetro accessToken y un body con la información del usuario en formato JSON

5. Put ImagesPhrases - Es un Endpoint para actualizar la información de una frase y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/images-phrases/{phraseId}, pasando el parámetro accessToken y un body con la información de la frase en formato JSON

6. Delete ImagesPhrases - Es un Endpoint para eliminar una frase y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/images-phrases/{phraseId}, pasando el parámetro accessToken.

Clase PredictionController

Este controlador sirve para obtener predicciones para una frase, ya sea obteniendo las imágenes de la API de Arasaac o de las imágenes almacenadas del usuario.

1. Get SavedImage Prediction - Se accede desde el enlace https://talking-pics.herokuapp.com/api/v1/predictions/saved-images pasando los parámetros:

    a) text ->frase de la cual se podrá predecir la siguiente palabra

    b) accessToken

    c) userId

Clase SavedImageController

Este controlador sirve para obtener las imágenes almacenadas en la base de datos

1. Get SavedImage - Obtiene todas las imágenes de la base de datos. Se accede desde el enlace https://talking-pics.herokuapp.com/api/v1/saved-images y se tiene que pasar el parámetro accessToken

2. Get SavedImage - Obtiene la información de una imagen en concreto y se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/saved-images/{imageId} y se tiene que pasar el parámetro accessToken

3. Get SavedImage by User - Obtiene la información de una frase según su usuario. Se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/saved-images/by-user y tienen que pasarse los siguientes argumentos:

    a) user_id

    b) accessToken

4. Get SavedImage by Text - Obtiene la información de una imagen de un usuario en concreto según su significado. Se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/saved-images/by-user-text y tienen que pasarse los siguientes argumentos:

    a) user_id

    b) accessToken

    c) text ->significado que se quiere buscar

5. Post SavedImage - Es un Endpoint para almacenar una imagen y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/saved-images, pasando el parámetro accessToken y un body con la información del usuario en formato JSON

6. Put SavedImage - Es un Endpoint para actualizar la información de una imagen y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/saved-images/{imageId}, pasando el parámetro accessToken y un body con la información del usuario en formato JSON

7. Delete SavedImage - Es un Endpoint para eliminar una imagen y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/saved-images/{imageId}, pasando el parámetro accessToken.

Clase UserController

Este controlador se usa para acceder a los usuarios almacenados en la base de datos.

1. Get Users - Obtiene todos los usuarios de la base de datos. Se accede desde el enlace https://talking-pics.herokuapp.com/api/v1/users y se tiene que pasar el parámetro accessToken

2. Get User - Obtiene la información de un usuario en concreto y se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/users/{userId} y se tiene que pasar el parametro accessToken

3. Get User by UidGoogle - Obtiene la información de un usuario según su uid de Google. Se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/users/by-uid y tienen que pasarse los siguientes argumentos:

   a) uid

   b) accessToken

4. Post User - Es un Endpoint para almacenar un usuario y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/users, pasando el parámetro accessToken y un body con la información del usuario en formato JSON

5. Put User - Es un Endpoint para actualizar la información de un usuario y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/users/{userId}, pasando el parámetro accessToken y un body con la información del usuario en formato JSON

6. Delete User - Es un Endpoint para eliminar un usuario y se accede mediante la url https://talking-pics.herokuapp.com/api/v1/users/{userId}, pasando el parámetro accessToken.

Clase UserManualController

Este controlador tiene un único endpoint, el cual se accede mediante el enlace https://talking-pics.herokuapp.com/api/v1/predictions/saved-images/user-manual con el parámetro accessToken. Mediante este Endpoint se obtendrá un PDF que contiene el manual de usuario.

OpenAPI

La información completa de la API se puede consultar en el siguiente enlace https://talking-pics.herokuapp.com/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config#

4.3.3. Interfaz

La interfaz se realizó con el marco de trabajo de React Native. Esta parte del desarrollo se fue realizando en varias partes. Cabe destacar que esta fue una de las partes más complicadas de este proyecto, ya se empezó a desarrollar sin tener ningún conocimiento de React ni React Native.

Primero se crearon las estructuras de las interfaces, sin crear ninguna llamada a la API ni alguna funcionalidad, solo la estructura de cada una de las ventanas por separado. Para muchas de estas pantallas se tuvieron que usar imágenes de stock y así poder comprobar que funcionaban correctamente. Las pantallas se construyeron a partir de diferentes componentes que se iban reutilizando según las necesidades de las pantallas.

Tras tener las pantallas estructuradas, se procedió a añadir la navegación entre pantallas gracias a la librería React Navigation. Las navegaciones que se han añadido en esta aplicación son:

- Navegación de pila - Se van 'apilando' las ventanas una encima de otra, de forma que podemos volver atrás.
- Navegación de drawer - Gracias a esta navegación se pudo añadir un menú arrastrando la izquierda de la pantalla, desde donde se puede acceder a todas las funciones de la aplicación.

Lo difícil de la navegación es que se tenían que anidar las diferentes navegaciones para que

se pudieran usar las dos simultáneamente. Finalmente se consiguió anidando la navegación drawer dentro de la de pila.

Posteriormente, se comenzó a añadir diferentes funcionalidades, pantalla a pantalla. Se comenzó con las más sencillas, que son las de mostrar las imágenes almacenadas y frases favoritas. Después, se añadió la funcionalidad de la pantalla de Traducir una imagen. Finalmente, se terminaron de desarrollar las ventanas para construir una frase de imágenes.

Tras haber desarrollado todas estas ventanas, se pudo refinar la aplicación en conjunto, además de añadir la opción de Cambiar Modo, Cerrar sesión e Iniciar sesión. Cabe mencionar que para añadir las funcionalidades de manejo de sesiones se tuvo que instalar el framework de Expo, el cual permite añadir estas y otras muchas funcionalidades, como el acceso a la cámara o a los ficheros para permitir la descarga de archivos. Se tuvieron varios problemas al cambiar el proyecto a expo, ya que no hay forma de cambiarlo automáticamente de aplicación React Native a React Native + Expo, por lo que se tuvo que crear un proyecto Expo vacío y copiar los archivos que fueran necesarios para que funcionase la aplicación.

Lo ultimo que se añadió a la aplicación fue la opción de descargar el manual de usuario, puesto que este no se había desarrollado hasta que fue escrita la memoria. Finalmente, despues de haber añadido la opcion de descargar un pdf, se compiló la aplicación para obtener un APK con el siguiente comando:

```
1       expo build:android -t apk
```

4.4. Pruebas

En esta sección se explicarán las diferentes pruebas que se realizaron a lo largo del proyecto.

4.4.1. JUnit

En JUnit se realizaron diferentes pruebas para comprobar que las operaciones GET funcionaban correctamente. Para ello se hicieron pruebas que comprobaran que el código que devolvía el Endpoint es el correcto, además de corroborar que el 'Mime type' que devuelve es el correcto. Aquí veremos ejemplos de pruebas para obtener todas las imágenes almacenadas en la base de datos. Tenemos que tener en cuenta que estos tests funcionan gracias a un token de acceso que es proporcionado por Google y que caduca despues de 60 minutos. Si ha pasado

ese tiempo en los tests se tendrá que modificar el

Prueba con un token inválido

Con esta prueba se consigue comprobar que si mandamos un token que no existe o que ha caducado por parametro, la API nos devolvera un código 400, indicandonos que este token es invalido y que tenemos que iniciar sesión. Para ello desde todos los metodos de la API se comprueba la autenticidad del token haciendo una llamada a la url https://www.googleapis.com/oauth2/v1/tokeninfo?access_token={token}.

```
1      @Test
2   public void
    givenAccessTokenInvalid_whenSavedImagesIsRetrieved_then400IsReceived()
    throws IOException {
3
4       // Given
5       HttpUriRequest request = new HttpGet('https://talking-pics.
    herokuapp.com/api/v1/saved-images?accessToken=' + accessTokenIncorrect )
    ;
6
7       // When
8       HttpResponse httpResponse = HttpClientBuilder.create().build().
    execute( request );
9
10      // Then
11      assertTrue(httpResponse.getStatusLine().getStatusCode() == 400);
12   }
13
```

Listing 5: Prueba invalid token

Prueba con un token válido

Con esta prueba se comprueba que mandando un token válido por parámetro se obtenga un código 200.

```
    @Test
 public void
givenAccessTokenValid_whenSavedImagesIsRetrieved_then200IsReceived()
throws IOException {

    // Given
    HttpUriRequest request = new HttpGet("https://talking-pics.
herokuapp.com/api/v1/saved-images?accessToken=" + accessTokenCorrect );

    // When
    HttpResponse httpResponse = HttpClientBuilder.create().build().
execute( request );

    // Then
    assertTrue(httpResponse.getStatusLine().getStatusCode() == 200);
 }

```

Listing 6: Prueba Token Válido

Prueba MIME type

Finalmente, se comprueba que el tipo que devuelve el endpoint. En este caso podemos ver que es de tipo 'application/json'.

```
    @Test
 public void
 SavedImagesDefaultResponseContentTypeIsJson() throws IOException {

    String jsonMimeType = "application/json";
    HttpUriRequest request = new HttpGet("https://talking-pics.
herokuapp.com/api/v1/saved-images?accessToken=" + accessTokenCorrect );

    HttpResponse response = HttpClientBuilder.create().build().execute(
 request );

```

```
10        String mimeType = ContentType.getOrDefault(response.getEntity()).
    getMimeType();
11
12        assertEquals( jsonMimeType, mimeType );
13     }
14
```

Listing 7: Prueba Mime Type

Estos tipos de pruebas se han realizado para todos los Endpoints GET que contiene la aplicación. Tal y como se puede ver en las siguientes imágenes, todos los tests dieron resultados satisfactorios.

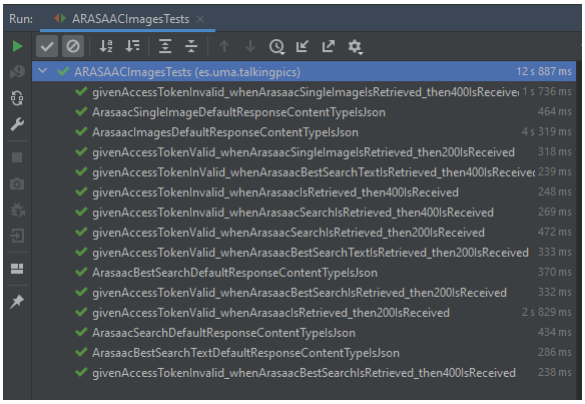


Figura 31: Arasaac Tests

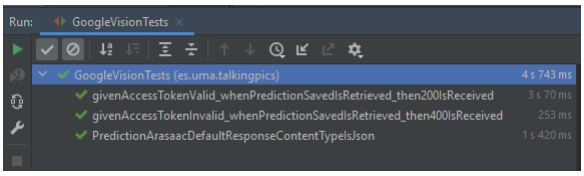


Figura 32: Google Vision tests

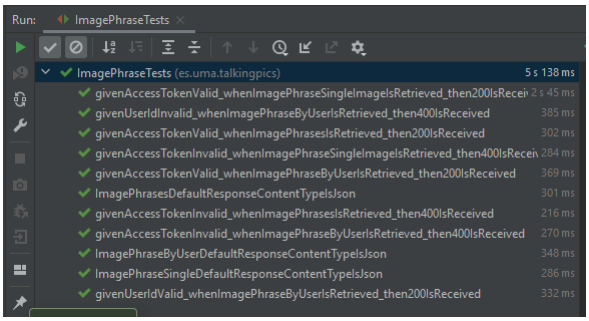


Figura 33: Image phrase tests

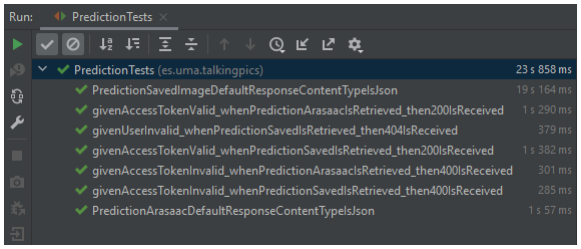


Figura 34: Prediction tests

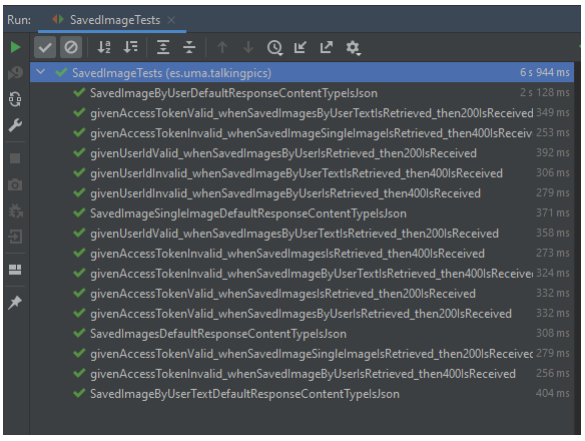


Figura 35: Saved Images tests

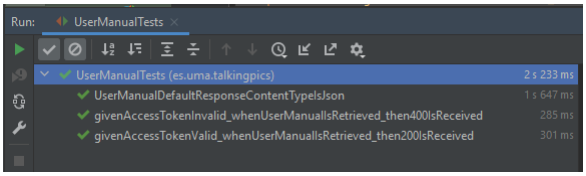


Figura 36: User Manual tests

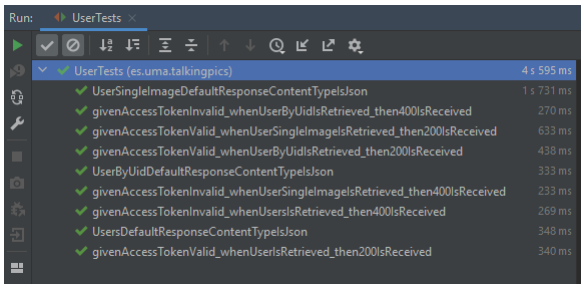


Figura 37: User Tests

4.4.2. Postman

En Postman se realizaron pruebas para comprobar que las operaciones CRUD de todas las entidades funcionaban correctamente. Estas pruebas se realizaron cuando el proyecto estaba en local y se iban desarrollando los Endpoints, y cuando el proyecto ya estaba publicado en Heroku. Aquí se puede ver la estructura de los paquetes y requests en Postman.

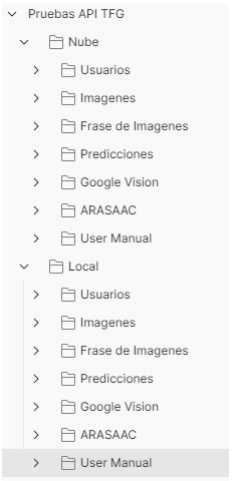


Figura 38: Estructura Postman

Para ver como se han realizado las pruebas nos centraremos en el paquete Usuarios de la carpeta Nube. Aunque es cierto que las operacion de DELETE no se utiliza en este proyecto, se ha añadido en el caso de que sea necesario para lineas futuras. Las requests que se han realizado a la API en este caso son las siguientes:

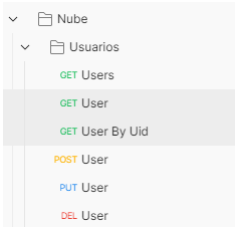


Figura 39: Paquete usuarios

Get Users

Con esta request obtendremos todos los usuarios almacenados en nuestra base de datos. Al ejecutarlo desde Postman obtenemos el siguiente resultado.

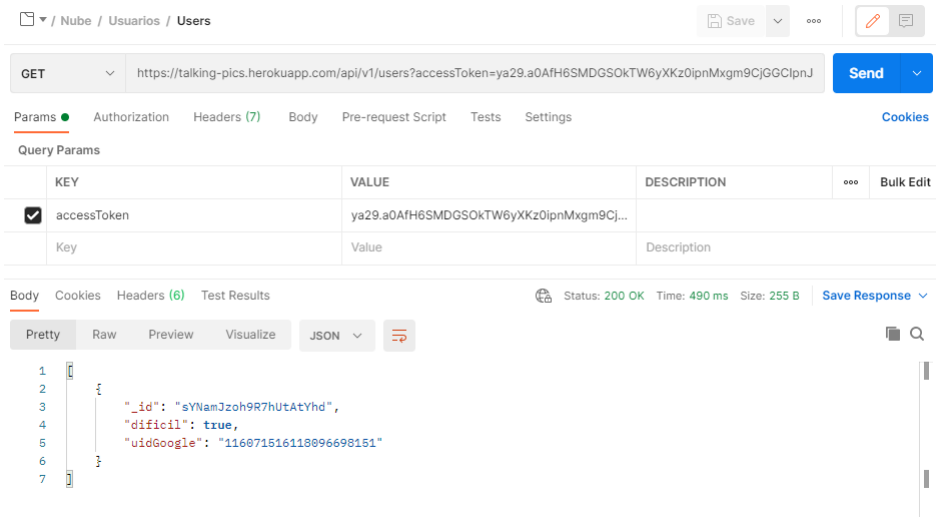


Figura 40: GET Users

Si comprobamos con la base de datos, el resultado concuerda con lo almacenado.

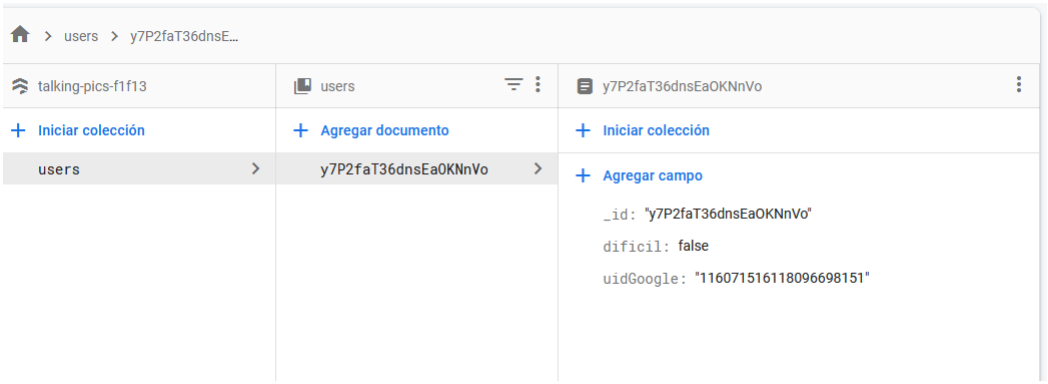


Figura 41: GET Users Firebase

Get User

Con esta request obtendremos la información de un solo usuario. Al ejecutarlo obtenemos el siguiente resultado

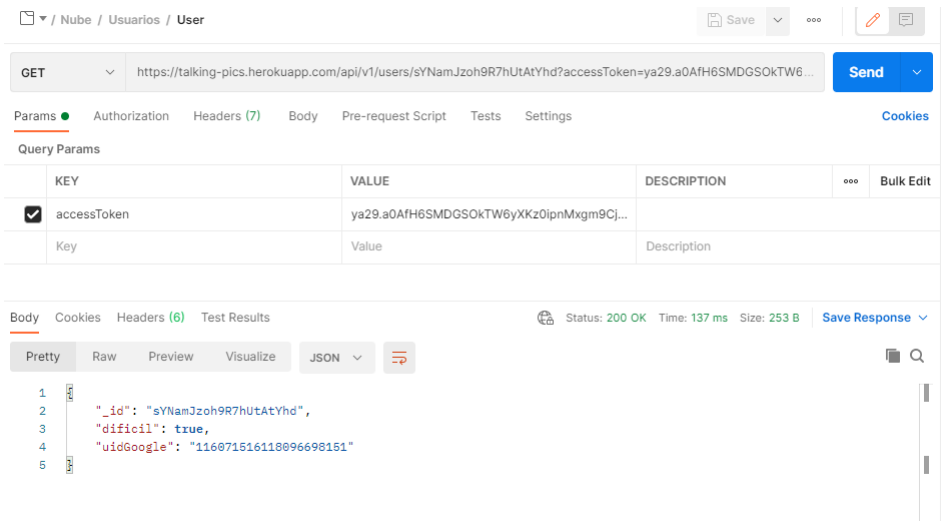


Figura 42: GET User

Y si lo comprobamos en la base de datos, el resultado es correcto.

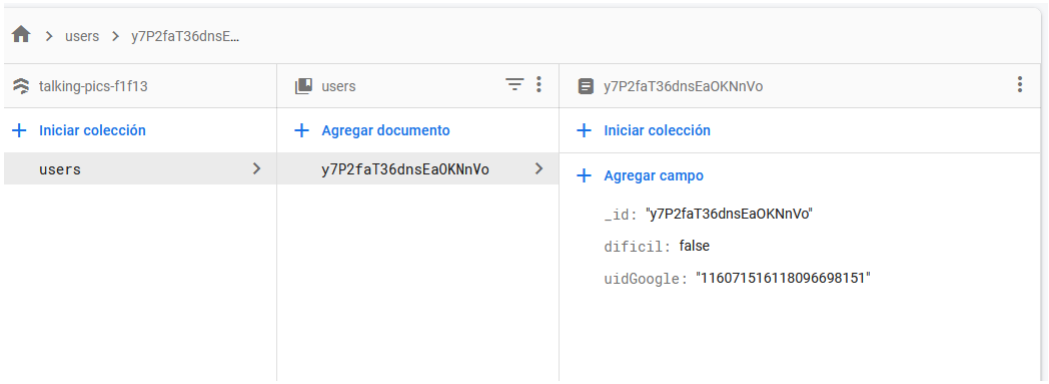


Figura 43: GET user Firebase

Get User By Uid

Con esta request obtendremos la información de un usuario a partir del UID proporcionado por Google. Al ejecutarlo obtenemos el siguiente resultado.

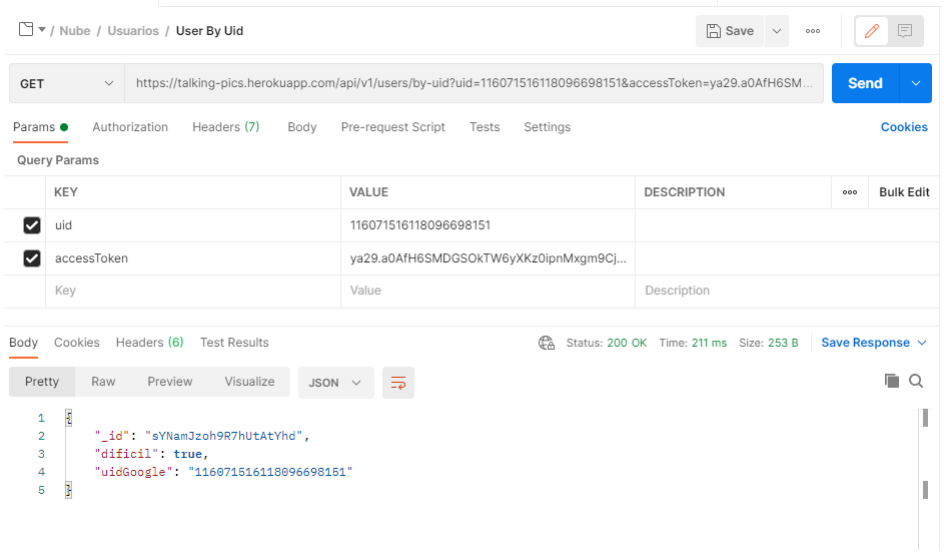


Figura 44: GET By Uid

Que concuerda con la base de datos.

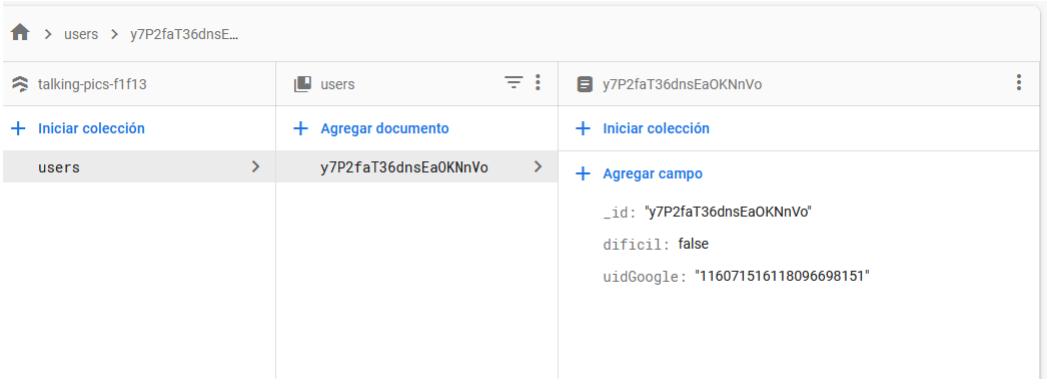


Figura 45: By Uid

Post User

Con este POST podremos añadir un nuevo usuario a nuestra base de datos. Al ejecutarlo la API nos devuelve el siguiente resultado.

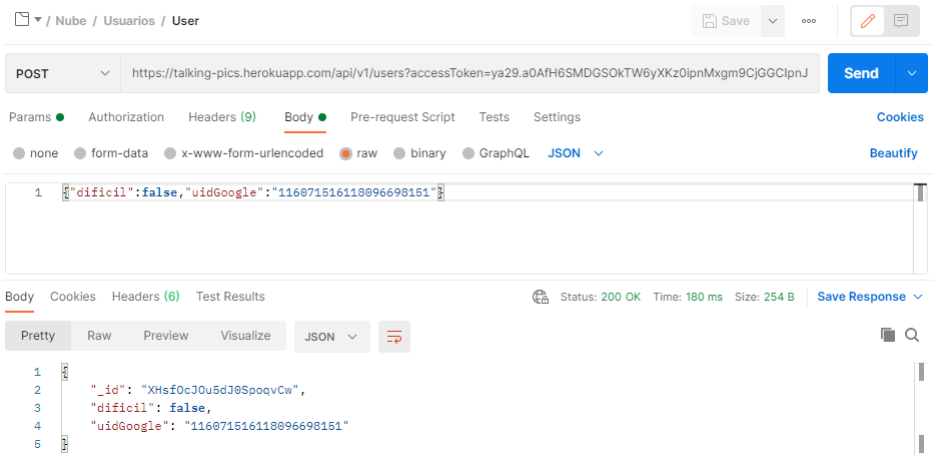


Figura 46: POST User

Y en la base de datos podemos comprobar que se ha añadido correctamente.

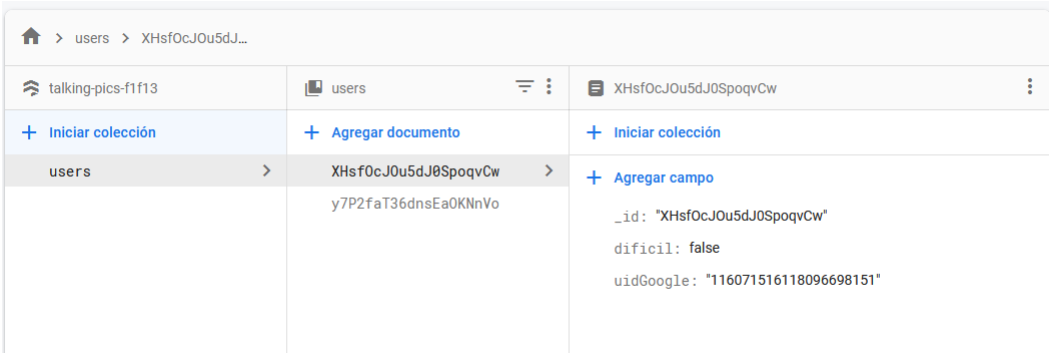


Figura 47: Firebase POST

Put User

Con este PUT podremos actualizar un usuario. Actualizaremos la información del usuario añadido en POST User.

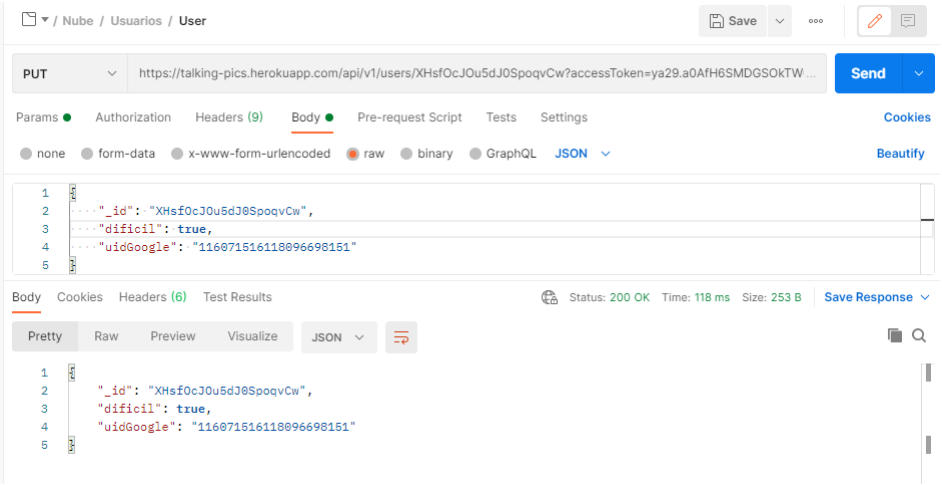


Figura 48: PUT User

Al comprobar en la base de datos, podemos ver que efectivamente se ha actualizado.

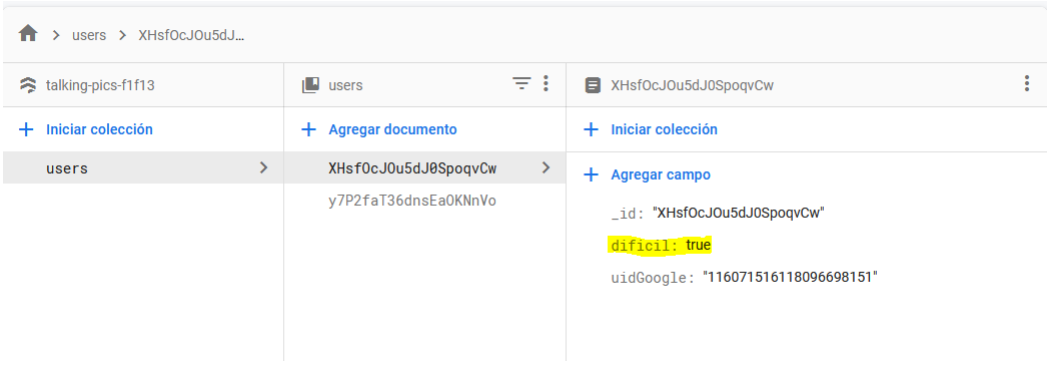


Figura 49: Paquete usuarios

Delete User

Finalmente, eliminaremos el usuario añadido anteriormente.

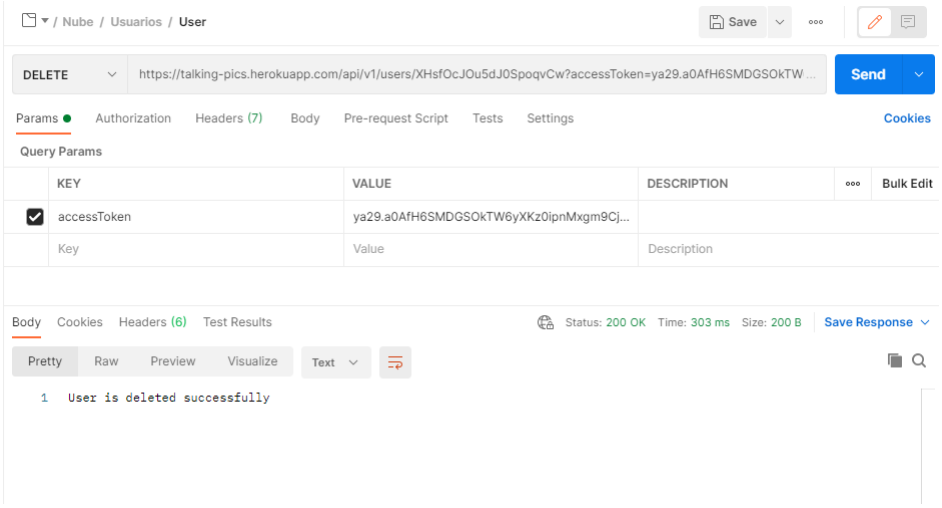


Figura 50: Paquete usuarios

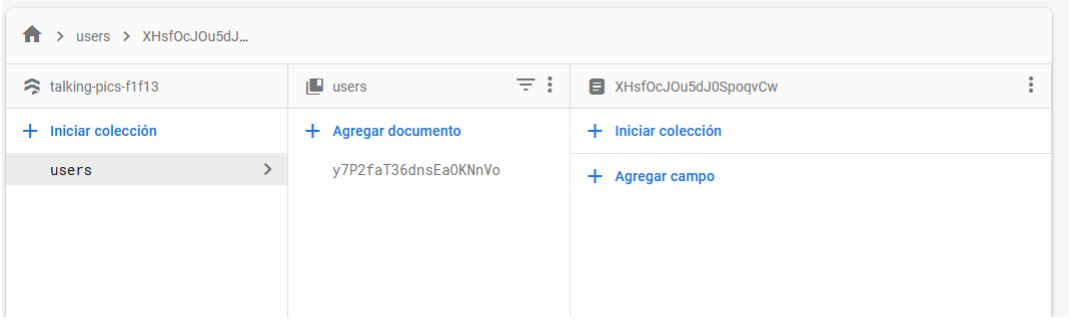


Figura 51: Paquete usuarios

5

Conclusiones y Trabajos Futuros

5.1. Conclusiones

En este TFG se ha desarrollado una aplicación que permite a las personas con problemas de comunicación obtener herramientas para tener conversaciones más sencillas con personas de su alrededor. Para ello, Talking Pics incluye las características de construir una frase de imágenes, para poder formular frases más complejas como si estuviéramos usando una tabla de pictogramas, y permite traducir frases a imágenes, de forma que sea mas entendible para el usuario.

Para desarrollar esta aplicación se han usado múltiples tecnologías (2.2) tales como Java, Spring, Python o Flask para el backend, o React Native para el frontend. Esto ha añadido cierta complejidad al desarrollo ya que, debido al poco conocimiento que se poseía de React Native se ha tardado más de lo esperado en desarrollar la aplicación, aunque esta experiencia ha servido para dar un paso adelante en mis conocimientos sobre desarrollo de Software, especialmente en el desarrollo de frontend.

Además de esta dificultad, el poder entender las diferentes tecnologías que se han usado durante todo el desarrollo ha implicado una dedicación extra de tiempo que no se tenia en un principio planeado. Es verdad que este tiempo no ha sido perdido, ya que se han aprendido nuevas tecnologías que al inicio de este proyecto no se conocían, pero este detalle ha incrementado bastante la dificultad.

En resumen, el resultado es bastante fiel a lo que se imaginaba en un primer momento excepto por algunos detalles, sobre todo relacionados con la interfaz gráfica, ya que esta es demasiado simple.

Aunque es evidente de que no se ha podido seguir al pie de la letra como sería un desarrollo con Scrum, ha sido una gran experiencia que me servirá como base en un futuro no muy lejano.

Espero que próximamente pueda mejorar aún más mis habilidades para poder aumentar la calidad de la interfaz gráfica y la aplicación en general.

5.2. Líneas futuras

En un futuro, la aplicación podría mejorarse aún más. Puesto que cada vez más personas estamos en contacto con extranjeros, este proyecto podría desarrollarse de forma de que permitiera a personas de diferentes idiomas usar la aplicación, según sus necesidades.

Otra posibilidad sería, a raíz de lo anteriormente dicho, que se pudieran hacer predicciones de otros idiomas. Además, como extensión de este TFG, se podría desarrollar una Inteligencia Artificial propia, en vez de ser una externa, de forma que pudiéramos nosotros elegir el idioma y limpiar aún más el *DataSet*, ya que actualmente este modelo incluye signos como el punto, el paréntesis, la coma, etc, los cuales pueden no aportar nada al significado de la frase.

Finalmente, una mejora que facilitaría el uso de la aplicación a los usuarios es el que se puedan buscar imágenes con verbos conjugados, ya que la API de Arasaac no tiene todos los verbos conjugados y esto puede ser un problema.

Referencias

[1] *Spring*. (s. f.). Spring. Recuperado 20 de abril de 2021, de https://spring.io/

[2] *Home.* (s. f.). Scrum.Org. Recuperado 20 de junio de 2021, de https://www.scrum.org/

[3] *Spring initializr.* (s. f.). Spring initializr. Recuperado 21 de mayo de 2021, de https://start.spring.io/

[4] *Firebase.* (s. f.). Firebase. Recuperado 25 de abril de 2021, de https://firebase.google.com/?hl=es

[5] Google. (s. f.-a). *Cloud Vision documentation | Cloud Vision API |*. Google Cloud. Recuperado 25 de abril de 2021, de https://cloud.google.com/vision/docs

[6] *ARASAAC.* (s. f.). Arasaac. Recuperado 30 de abril de 2021, de https://arasaac.org/

[7] Departamento de Ciencias de la Computación Universidad de Chile. (s. f.). *dccuchile/bert-base-spanish-wwm-uncased · Hugging Face.* BETO: Spanish BERT. Recuperado 30 de abril de 2021, de https://huggingface.co/dccuchile/bert-base-spanish-wwm-uncased

[8] Uresti, I. (2019, 7 abril). *Spring boot + oAuth2 + firebase - Nearsoft Jobs.* Medium. https://blog.nearsoftjobs.com/spring-boot-oauth2-firebase-d8a4bf37ce15

[9] Ligade, M. (2020, 28 noviembre). *Spring Boot Firebase CRUD - techwasti.* Medium. Recuperado 30 de abril de 2021, de https://medium.com/techwasti/spring-boot-firebase-crud-b0afab27b26e

[10] *Testing Spring Boot | Cloudflare.* (s. f.). Baeldung. Recuperado 10 de mayo de 2021, de https://www.baeldung.com/spring-boot-testing

[11] *Spring Cloud GCP Reference Documentation.* (s. f.). Spring Cloud GCP Reference Documentation. Recuperado 10 de mayo de 2021, de https://docs.spring.io/spring-cloud-gcp/docs/1.0.0.RELEASE/reference/htmlsingle/

[12] S. (s. f.). *spring-cloud/spring-cloud-gcp.* GitHub. Recuperado 1 de mayo de 2021, de https://github.com/spring-cloud/spring-cloud-gcp/tree/main/spring-cloud-gcp-samples/spring-cloud-gcp-vision-api-sample

[13] *Open API 3 Library.* (s. f.). OpenAPI. Recuperado 10 de mayo de 2021, de https://springdoc.org/

[14] Mor, R. (2021, 8 abril). *REST API Testing Strategy: What Exactly Should You Test?* Sisense. https://www.sisense.com/blog/rest-api-testing-strategy-what-exactly-should-you-test/

[15] *Curso de React Native Desde Cero.* (s. f.). YouTube. Recuperado 15 de mayo de 2021, de https://www.youtube.com/playlist?list=PLUlw6638d2QZFbQfC3sB5mKWJUWr2HEkc

[16] *React Native Elements | React Native Elements.* (s. f.). React Native Elements. Recuperado 20 de mayo de 2021, de https://reactnativeelements.com/

[17] *React Navigation | React Navigation.* (s. f.). React Native Navigation. Recuperado 22 de mayo de 2021, de https://reactnavigation.org/

[18] Agrawal, S. (2020, 21 noviembre). *Example of Grid Image Gallery in React Native.* About React. https://aboutreact.com/grid-image-gallery/

[19] Ezekiel, E. (2020, 22 junio). *Combining Stack, Tab & Drawer Navigations in React Native With React Navigation 5.* DEV Community. https://dev.to/easybuoy/combining-stack-tab-drawer-navigations-in-react-native-with-react-navigation-5-da

[20] *React Native · Learn once, write anywhere.* (s. f.). React Native. Recuperado 20 de mayo de 2021, de https://reactnative.dev/

[21] *Pagination | Cloudflare.* (s. f.). Baeldung. Recuperado 20 de mayo de 2021, de https://www.baeldung.com/rest-api-pagination-in-spring

[22] *Introduction to Expo.* (s. f.). Expo Documentation. Recuperado 15 de junio de 2021, de https://docs.expo.io/

[23] Google. (s. f.). *Firebase.* Firebase. Recuperado 30 de abril de 2021, de https://firebase.google.com/docs/build?hl=es

[24] Romero, I. G. (2021, 14 noviembre). *14 Nov Qué es la discapacidad. Concepto y evolución histórica.* El Blog de la Fundación Adecco. https://fundacionadecco.org/blog/que-es-la-discapacidad-evolucion-historica/

[25] OMS. (2020, 1 diciembre). *Discapacidad y salud.* Discapacidad y salud. https://www.who.int/es/news-room/fact-sheets/detail/disability-and-health

Apéndice A
Manual de Instalación

API de predicciones:

Requisitos:

Tener instalado Python 3.9.0 en la máquina.

Despliegue:

Abrir una terminal en la carpeta donde se encuentra el proyecto. Tras esto ejecutar el siguiente comando:

```
pip install -r requirements.txt
```

Listing 8: Instalar requirements

Tras haber instalado todos los paquetes, el proyecto se podrá ejecutar con:

```
flask run
```

Listing 9: Ejecutar con flask

API principal:

Requisitos:

- Tener instalada la version 11 del jdk de Java.
- Tener Apache Maven instalado.
- (Opcional) Instalar un IDE compatible. En este caso para desarrollar el proyecto se uso IntelliJ.

Despliegue: Ejecutar con IntelliJ:

1. Extraer el proyecto en una carpeta de la máquina.
2. Abrir Intellij y seleccionar ‘Open project‘.

3. Buscar en el explorador de archivos la carpeta donde hemos almacenado el proyecto.
4. El proyecto se abrirá y se compilará
5. Ejecutar el proyecto
6. Se podrá acceder a la api desde la dirección localhost:8080

Ejecutar con Maven:

1. Extraer el proyecto en una carpeta de la máquina.
2. Ejecutar la linea de comandos
3. Ejecutar el comando *mvn clean install*
4. El jar se habrá creado en una carpeta target. Copiar todo el contenido de la carpeta target y moverlo a la raiz del proyecto.
5. Ejecutar con *java -jar <artifact-name>*
6. Se podrá acceder a la api desde la dirección localhost:8080

Proyecto React Native:

Requisitos:

- Tener instalado Visual Studio Code
- Instalar la ultima version de Node
- Instalar React Native con el comando

```
npm install -g create-react-native-app
```

Listing 10: Instalar react native

- Instalar Expo CLI con el comando

```
npm install --global expo-cli
```

Listing 11: Instalar expo

Despliegue:

Tras haber instalado todos los paquetes, hay que seguir los siguientes pasos:

1. Extraer el proyecto en una carpeta
2. Abrir la carpeta con Visual Studio Code
3. Compilar el proyecto con el comando

```
expo build:android -t apk
```

Como uno de los paquetes usados para desarrollar el proyecto ha sido Expo Google Sign In y no se puede usar en proceso de desarrollo, la única forma de probar el proyecto es obteniendo la APK e instalándola en un móvil Android.

Apéndice B
Manual de Usuario

Inicio de Sesión

Desde esta ventana, al pulsar en el botón que aparece podrás iniciar sesión con las cuentas de Google que tengas almacenadas en tu proyecto.

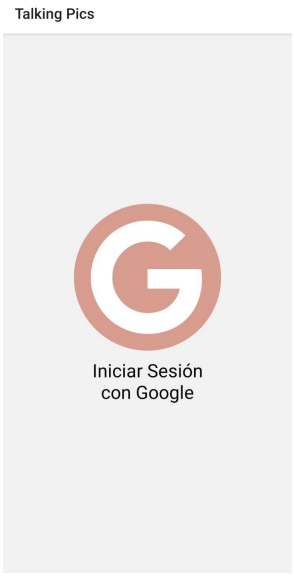


Figura 52: Inicio de sesión

Página de inicio

Desde la página de inicio podrás moverte a las dos funcionalidades más importantes de la aplicación, Construir una frase de imágenes y Traducir una frase.

Figura 53: Inicio fácil

Además, si se hace el gesto de arrastrar hacia la derecha podrá ver el menú que permitirá acceder al resto de funcionalidades de la aplicación.

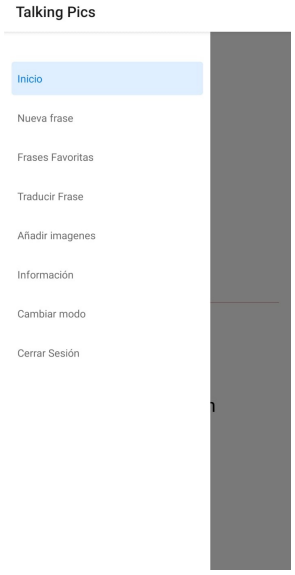


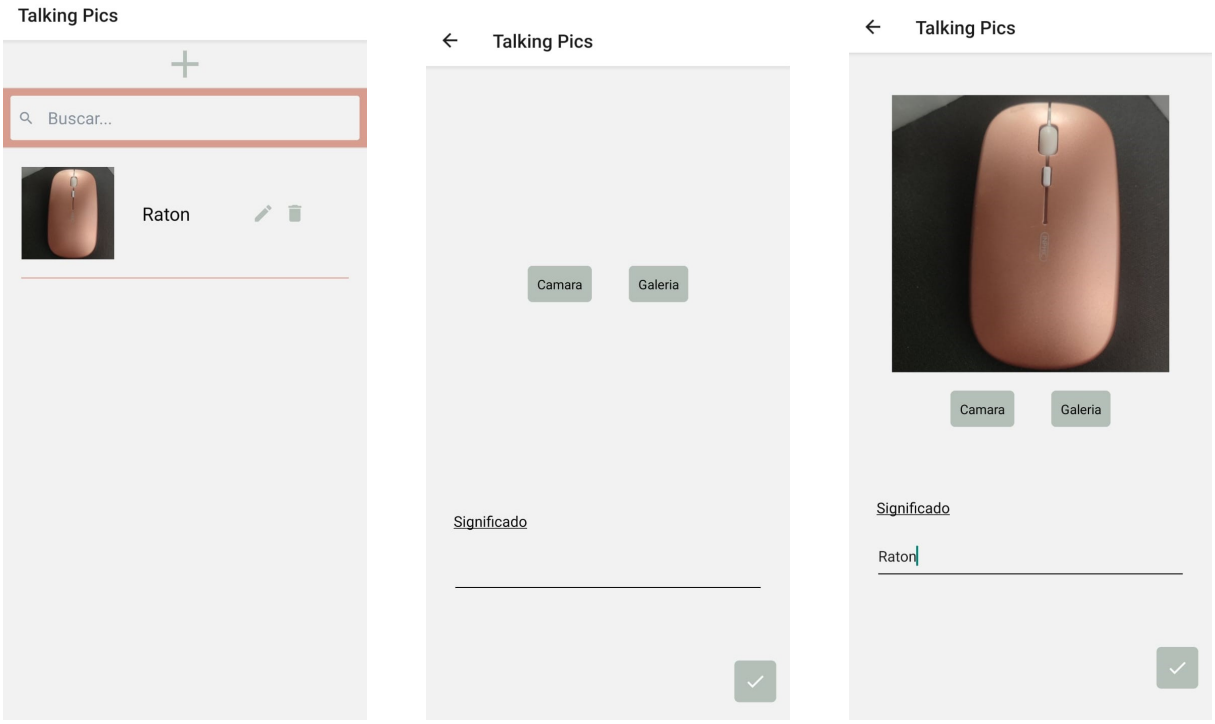
Figura 54: Menú

Para esta ventana también existe una versión difícil con más texto en vez de iconos.

Figura 55: Inicio difícil

Almacena tus propias imágenes

Desde esta ventana podrás visualizar tus propias imágenes, además de añadir otras nuevas. Para añadir nuevas imágenes solo hay que pulsar el botón + para acceder a la pantalla de añadir imágenes. Esta pantalla también servirá para cuando se quiera editar una imagen que ya esta añadida, pulsando el botón que tiene forma de lápiz.



(a) Imágenes almacenadas (b) Añadir nueva imagen (c) Editar imagen

Figura 56: Almacenar tus propias imágenes

Construye tu frase de imágenes

En esta pantalla podremos construir una frase de imágenes. Al entrar se nos mostrarán varias imágenes, además de nuestros propios pictogramas almacenados. Al elegir uno de estos pictogramas, las imágenes se recargarán y se mostrarán al principio del todo imágenes que la aplicación nos recomienda añadir a esta frase.

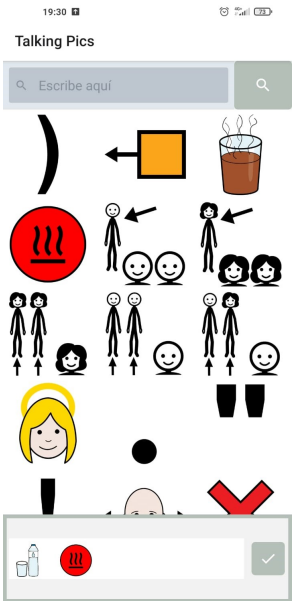


Figura 57: Construir

También se podrán buscar imágenes con un significado concreto.

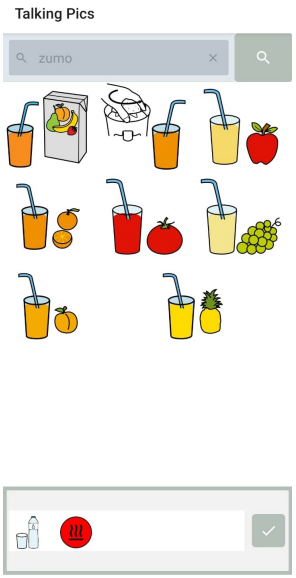


Figura 58: Búsqueda

Tras haber terminado la frase y pulsar el botón de “Ok” se nos mostrará una nueva ventana desde la que podremos visualizar nuestra nueva frase de imágenes, su significado y nos da la opción de almacenarla en favoritos. Finalmente pulsaremos el botón de ‘Ok’

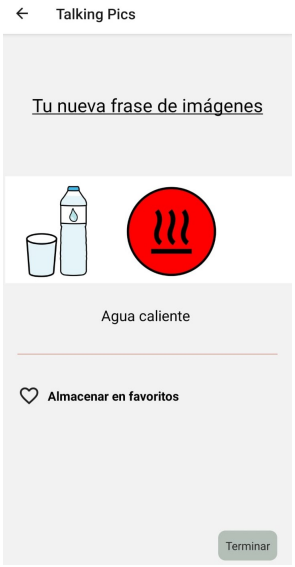


Figura 59: Frase construida

Visualiza tus frases favoritas

Desde esta pantalla se podrán visualizar todas las frases que hemos creado y que hemos seleccionado como favoritas. Estas frases también podrán ser eliminadas pulsando el botón de borrar.

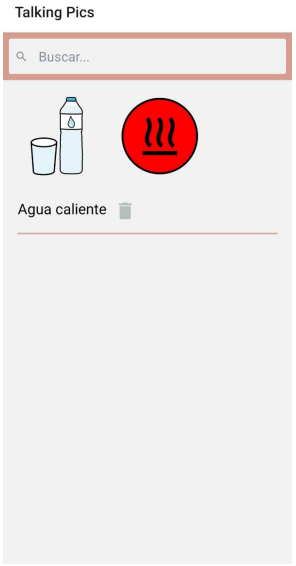


Figura 60: Frases favoritas

Traduce una frase

Desde esta pantalla se podrá elegir una imagen que contenga texto a través de la cámara o galería de nuestro dispositivo. Tras haber seleccionado la imagen se procederá a traducir cada una de las palabras que estén en la imagen por pictogramas. Esta frase de imágenes nos aparecerá justo debajo.

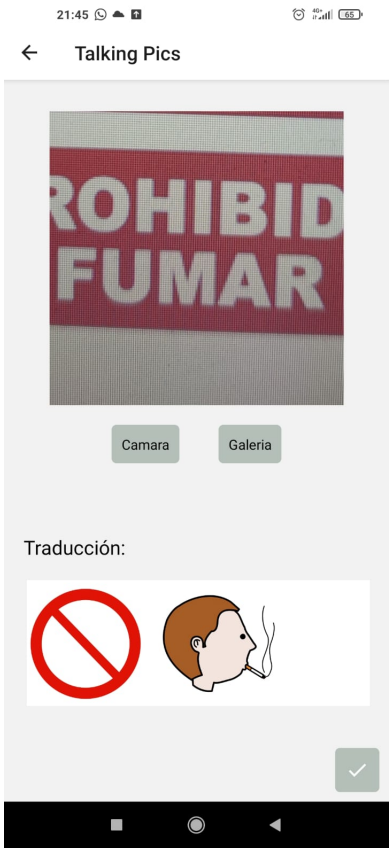
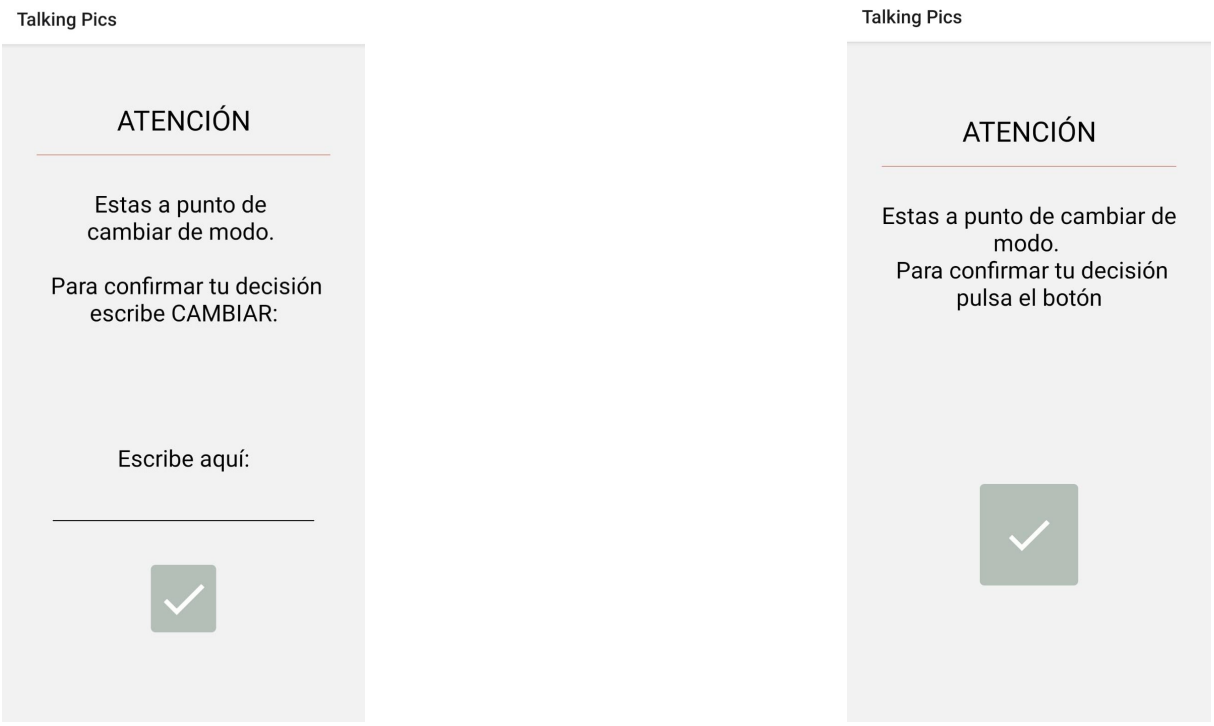


Figura 61: Traducir imagen

Cambia de modo

Desde estas ventanas se podrán cambiar de modo, de forma que la pantalla de inicio será diferente según en que modo estemos. Si queremos pasar de fácil a difícil nos hará falta confirmar nuestra acción escribiendo la palabra 'CAMBIAR' y pulsando el botón de aceptar. Por el contrario, si queremos cambiar de modo difícil a fácil solo nos hará falta pulsar un botón para confirmar.



(a) Fácil a difícil

(b) Difícil a fácil

Figura 62: Cambiar modo

Cierre de sesión

Al pulsar el botón “Cerrar sesión” del menú, se nos redirigirá a la página de inicio de sesión y habremos cerrado nuestra sesión.

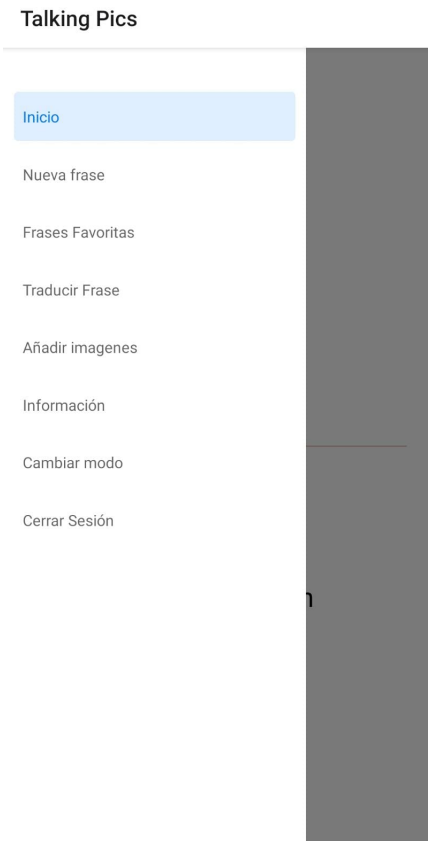


Figura 63: Cerrar sesión

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA