Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Desarrollo de una aplicación educativa multiplataforma para el entrenamiento de la pronunciación de idiomas

Autor:
Antonio Gamazo Ferrero

Universidad de **Valladolid**

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Software

Desarrollo de una aplicación educativa multiplataforma para el entrenamiento de la pronunciación de idiomas

Autor:
Antonio Gamazo Ferrero

Tutores:
Cristian Tejedor García
Mario Corrales Astorgano

A mi familia que me ha apoyado siempre.

Agradecimientos

A mis tutores, Cristian Tejedor García y Mario Corrales Astorgano, por su ayuda y dedicación durante todo el desarrollo del proyecto.

A mis compañeros de carrera, con los que he compartido tantas horas de estudio y trabajos, sin ellos no habría sido lo mismo.

Por último, a mi familia, que me ha apoyado en este trayecto y en todos.

Gracias.

Resumen

En la última década hemos visto como los teléfonos móviles han pasado a ser el centro de nuestras vidas. Las aplicaciones móviles nos facilitan todo tipo utilidades en nuestros dispositivos, como las aplicaciones educativas orientadas al aprendizaje de diferentes habilidades. También, en los últimos años se ha hecho popular el uso de sistemas de sistemas de entrenamiento de la pronunciación (computer-assisted pronunciation training, CAPT), con herramientas que incluyen síntesis de voz (text-to-speech, TTS) o reconocimiento del habla (automatic speech recognition, ASR). Gracias a la gran evolución en materia de inteligencia artificial y aprendizaje automático, estos sistemas son cada vez más precisos. Este proyecto, es una versión evolucionada de dos proyectos anteriores, TipTopTalk! y Clash of Pronunciations, dos aplicaciones móviles para el entrenamiento de la pronunciación extranjera. Éstas se basan en el uso de pares mínimos en ejercicios de entrenamiento diseñados por expertos en fonética. El objetivo principal de este proyecto es unir los conceptos y métodos de entrenamiento de los dos proyectos anteriores en una aplicación multiplataforma (móvil, web y escritorio) educativa, que permita expandir su funcionalidad de manera sencilla, independientemente de las particularidades de cada plataforma.

Abstract

In the last decade, smartphones have become the centre of our lives. Mobile applications allow us the use of many kinds of utilities in our devices, including educational applications focused on training of different skills. Furthermore, in recent years, the use of Computer-Assisted Pronunciation Teaching systems (CAPT) has become popular, including voice synthesis (text-to-speech, TTS) and automatic speech recognition (ASR) tools. The great evolution in terms of artificial intelligence and machine learning techniques has increased the performace of these tools. This project is an evolution of two previous projects, TipTopTalk! and Clash of Pronunciations, which are two applications intended for training foreign pronunciation. Both applications are based on the use of minimal pairs to develop training exercises designed by experts on phonetics. The main aim of this project is gathering the concepts and the training techniques of both previous projects in a educational multiplatform (mobile, web and desktop) application, which can be scaled up easily without reliying on the specific particularities of each platform.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

En un mundo en el que cada vez los dispositivos móviles han pasado a estar presentes en prácticamente todos los ámbitos de nuestro día a día, parece razonable, que cualquier tipo de aplicación que se quiera desarrollar tenga presencia en este ecosistema. Concretamente, en el apartado de juegos, las plataformas móviles ya superan en facturación a otras como PC o consolas [1]. Además, con las nuevas generaciones inmersas en una nueva era digital, las aplicaciones educativas para dispositivos móviles también están en auge [2].

Este proyecto surge de la propuesta de Trabajo Fin de Grado (TFG) publicada en la web de ofertas de TFG [3] de la Escuela de Ingeniería Informática de Valladolid por Cristian Tejedor García, "Desarrollo de aplicación multiplataforma para juego serio multijugador". El objetivo de ésta es construir una aplicación multiplataforma con la que se pueda practicar la pronunciación a la hora de aprender un idioma extranjero.

Este trabajo es una propuesta evolucionada de la aplicación TipTopTalk! del Trabajo Fin de Máster (TFM) de Cristian Tejedor García [4] y de Clash of Pronunciations del TFG de Rafael Sillero Navajas [5]. Ambas siguen la premisa de ser, mediante el uso de sistemas de reconocimiento automático del habla (Automatic Speech Recognition, ASR) y sintetizado de la voz (Text-To-Speech, TTS), un juego serio que, gracias a la gamificación [6] y a los sistemas mencionados anteriormente, faciliten la tarea de aprender y practicar la pronunciación de un idioma extranjero.

1.2. Motivación

Los cambios constantes en la tecnología de desarrollo de software de aplicaciones móviles implican un mantenimiento constante de las aplicaciones desarrolladas. En el caso de las aplicaciones TipTopTalk! y Clash of Pronunciations, las restricciones y cambios de Google Play Games durante los últimos cinco años han implicado una reducción de la funcionalidad original implementada por falta de recursos

y mantenimiento.

El principal motivo por el que se propone este proyecto es aprovechar las ventajas de un desarrollo multiplataforma para fabricar una aplicación, que con el mismo código, sea utilizable en distintos tipos de plataformas, móvil, web y escritorio. Para equipos de desarrollo pequeños es un problema el tener que desarrollar de forma nativa para diferentes entornos, dado que supone mucho esfuerzo tener un desarrollador especializado en cada ámbito. También resulta complicado encontrar alguien con conocimientos suficientes para que se encargue de todos los desarrollos, y si fuera el caso, le llevaría demasiado tiempo. Lo que se busca con el desarrollo multiplataforma es ahorrar tiempo y costes, de manera que con un desarrollador con conocimientos en JavaScript, HTML y CSS, se pueda abarcar un desarrollar para, al menos, las plataformas más utilizadas a día de hoy.

Por otra parte, se busca estudiar las posibilidades que ofrece el desarrollo multiplataforma, ya que, si bien es verdad que es una herramienta muy útil, es sabido que tiene ciertas limitaciones. Por lo tanto, otra motivación para ejecutar este proyecto es descubrir estas limitaciones o carencias que pueda tener el desarrollo de aplicaciones con tecnologías multiplataforma.

1.3. Estado de la cuestión

1.3.1. TipTopTalk!

Como se ha comentado en la introducción, esta aplicación fue desarrollada por Cristian Tejedor García [4] durante los años 2015 y 2016, y tiene el objetivo de facilitar y hacer más interesante la práctica de pronunciación de un idioma extranjero.

Es una aplicación Android para dispositivos móviles y educativa que consiste en un juego serio para la mejora de la pronunciación de un idioma extranjero (inglés, español y chino, entre otros). Integra actividades de entrenamiento en un ciclo de exposición, discriminación y pronunciación. Cuando los usuarios realizan dichas actividades, obtienen puntos y trofeos, y se ven reflejados en rankings.

Utilizando sistemas conversión de texto a voz y de reconocimiento de voz, se consigue que un usuario pueda escuchar cómo se pronuncia correctamente una palabra y posteriormente pronunciarla en voz alta para que el smartphone la interprete y pueda comprobar si esta es correcta.

El sistema de aprendizaje consiste en la comparación mediante pares mínimos de los fonemas de dos palabras [7]. Esto es, dos palabras que sólo difieren en un fonema, que es el que se quiere practicar, por ejemplo, *cede* y *sede*. Mediante la escucha, la grabación y la repetición se consigue llegar a diferenciarlos.

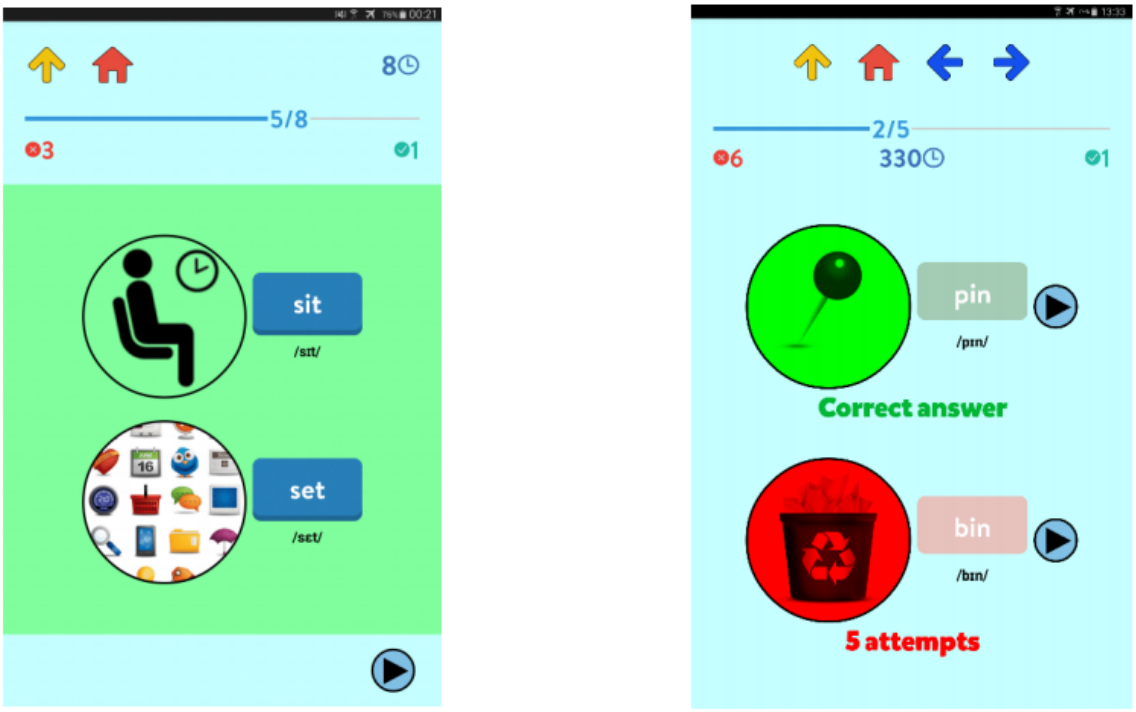

Figura 1.1: Apariencia de la aplicación TipTopTalk!

1.3.2. Clash of Pronunciations

Aplicación móvil Android desarrollada por Rafael Sillero Navajas como TFG [5], con el objetivo de añadir la funcionalidad multijugador y otros aspectos sociales mediante el uso de la plataforma Google Play Services a TipTopTalk!. Cuando esta plataforma dejó de dar servicio de tipo multijugador se encontró la necesidad de crear una backend que la sustituyera. Éste corrió a cargo de Andrés de la Maza Valles en su TFG en verano del 2020 [8].

Como se puede ver en la figura 1.2, hay disponibles cuatro modos de práctica, los cuales también son incluidos en este proyecto, junto a nuevos modos.

1.3.3. Duolingo

La aplicación de práctica y aprendizaje de idiomas más usada, con más de 100 millones de descargas en Google Play [9].Su funcionalidad básica es gratuita, siendo posible ser ampliada mediante subscripción de pago. Usa elementos de gamificación para hacerla más atractiva. Por ejemplo, se pierden vidas cuando hay una respuesta incorrecta y también se utiliza un sistema de niveles, que a parte de marcar el nivel del jugador, sirven para aumentar las ganas de autosuperación del mismo. Como se puede apreciar en la figura 1.3, consta de una interfaz muy minimalista y accesible para todo tipo de públicos.

Esta se centra más en el aprendizaje de un idioma a modo general, que en la pronunciación en particular. No dispone de demasiados ejercicios para practicar la pronunciación y que den una retroalimentación sobre esta.

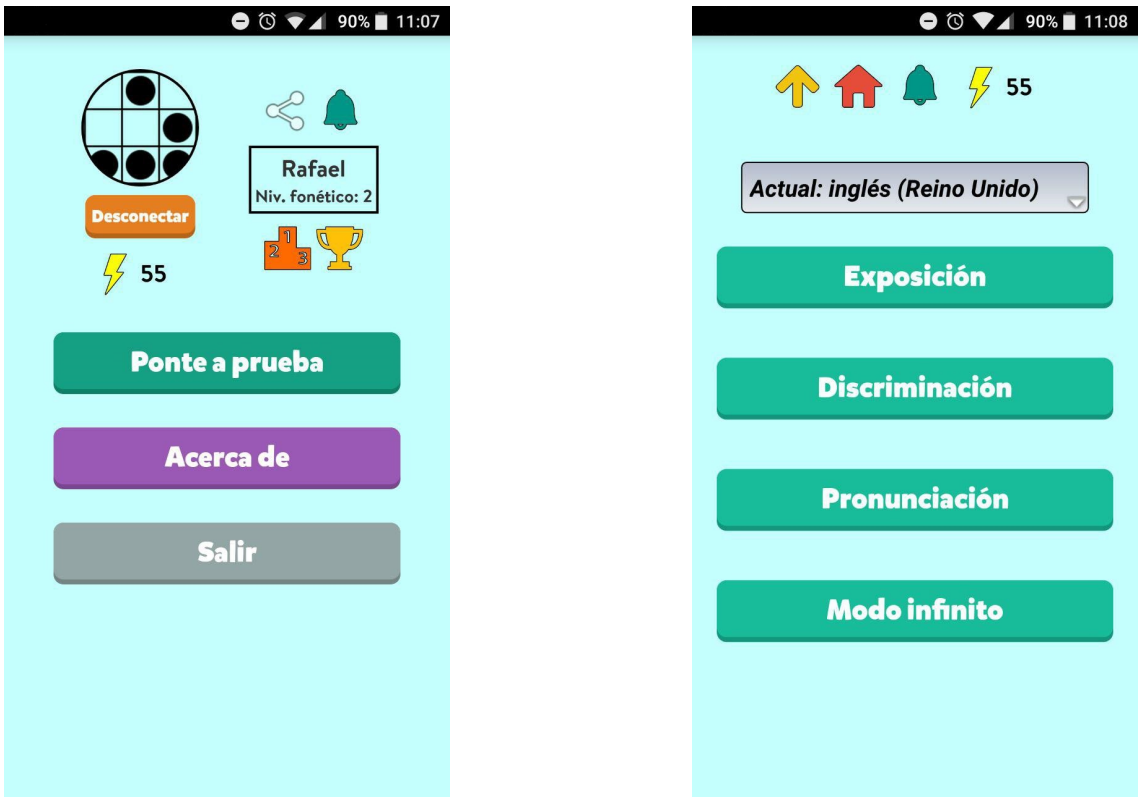

Figura 1.2: Apariencia de la aplicación Clash of Pronunciations.

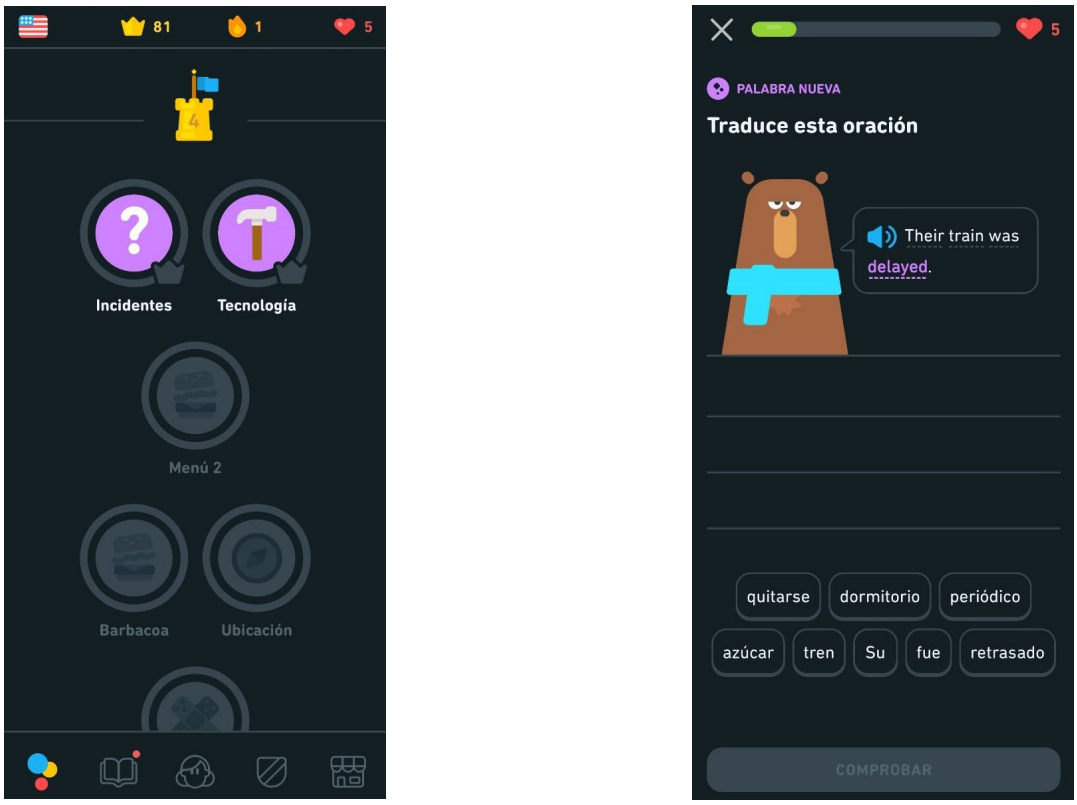

Figura 1.3: Apariencia de la aplicación Duolingo.

1.3.4. Babbel

Otra de las aplicaciones más usadas, con más de 10 millones de descargas en Google Play [10]. Ésta no dispone de planes gratuitos, si no que tiene varios planes de pago por meses o por año. Babbel no tiene tantos elementos de gamificación si no que se centra más en el contenido de los cursos ofreciendo centrados en temas concretos. A su vez, hace uso de sistemas TTS y ASR para la práctica de la pronunciación, aunque el uso de estas se reduce a un modo en el que se reproduce un audio y el usuario lo repite, con la única retroalimentación de si es correcto o no.

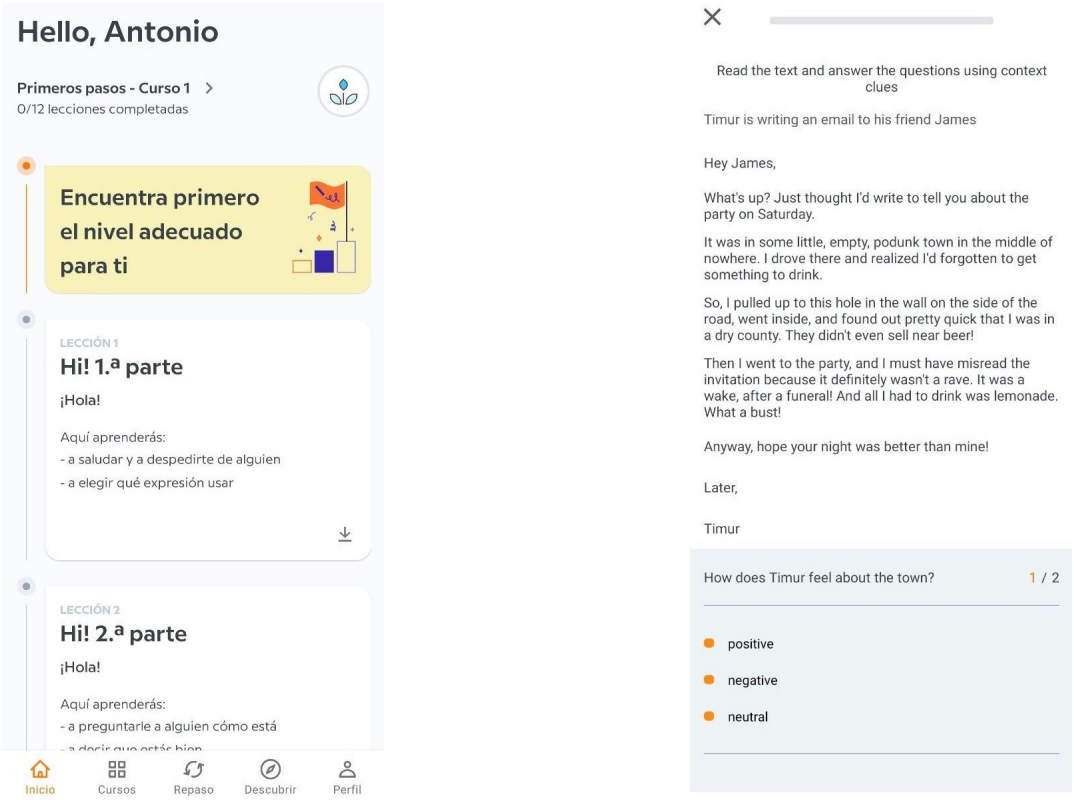

Figura 1.4: Apariencia de la aplicación Babbel.

1.3.5. Entorno y herramientas tecnológicas

En esta sección se exponen las tecnologías utilizadas en este TFG y las características de cada una de ellas. Para poner en contexto, en el backend, se ha utilizado: Node.js, para alojar el servidor; Express.js, para hacer la API; MongoDB, para la base de datos; y Google STT, para el sistema de ASR. Por otra parte, en el frontend: se ha utilizado Node.js como entorno de desarrollo; React Native, para crear la interfaz de la aplicación; Redux, para gestionar el estado global de la aplicación; y Google TTS, como sintetizador de voz. Como apoyo, se han utilizado otras tecnologías como Swagger, para documentar la API y, Postman, para probar las llamadas a la API durante el desarrollo.

1.3.5.1. Node.js

Node.js [11] es un entorno de ejecución de JavaScript orientado a eventos asíncronos. Por cada conexión, se activa la devolución de llamada, pero si no hay trabajo que hacer, Node.js se dormirá. Además, esta orientado al desarrollo de sistemas escalables debido a que casi ninguna de sus funciones utiliza operaciones de entrada/salida directamente, lo que evita bloqueos en el proceso.

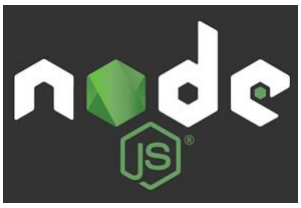

Figura 1.5: Logo de Node.js.

Construido en base a versiones modernas de V8 [12], un motor de alto rendimiento de código abierto de JavaScript y WebAssembly de Google, también utilizado en Chrome. Al usar las ultimas versiones de este motor se garantizan las características de la especificación ECMA-262 [13] de JavaScript y mejoras continuas en cuanto a rendimiento y estabilidad.

Para este proyecto se ha utilizado de dos formas diferentes. Por un lado, en el backend, la API está alojada en un servidor Node.js con el apoyo del framework Express.js. Por otro lado, en el frontend, se utiliza en entorno de desarrollo para ejecutar la aplicación React Native, de manera que se pueden probar los cambios que se van realizando si necesidad de generar los paquetes cada vez.

1.3.5.2. NPM

Figura 1.6: Logo de NPM.

NPM [14] (node package manager) es un gestor de paquetes para Node.js. Gracias a este podemos descargar y actualizar multitud de librerías. También sirve como herramienta para automatizar tareas, ya que permite scripting, por ejemplo, arrancar un proyecto, pasar los test o lanzar cualquier tipo de comando que se necesite.

1.3.5.3. Express.js

Figura 1.7: Logo de Express.js.

Express.js [15] es una infraestructura rápida, minimalista y flexible para Node.js. Proporciona un conjunto de características básicas para aplicaciones web y móvil. Entre las características más destacadas se encuentran:

- **Gestionar peticiones HTTP:** dispone de múltiples utilidades para gestionar las comunicaciones HTTP. Es el receptor y encargado de responder a estas peticiones.
- **Actuar de Middleware:** una de las utilidades más interesantes, permite desestructurar las peticiones, clasificarlas según el método HTTP detectado y manejar los posibles errores en la llamada.

1.3.5.4. React Native

Figura 1.8: Logo de React Native.

React Native [16] es una tecnología que, mediante el uso únicamente de código JavaScript, permite desarrollar aplicaciones nativas para Android e iOS.

Por otro lado, React [17] es una librería de JavaScript, publicada por Facebook en 2013, para construir interfaces de usuario. Utiliza vistas declarativas, que hacen que el código sea predecible y fácil de depurar. Está basado en componentes, cada uno de estos es capaz de gestionar su propio estado, lo que permite crear interfaces complejas. A parte, estos componentes facilitan la reutilización de código.

React Native, publicado en 2015, también por Facebook, combina lo anterior con la capacidad de desarrollar aplicaciones nativas. Dependerá del desarrollador y del caso de uso, en qué medida se usa

este. Dado que se puede combinar con código nativo de Android o de iOS, se pueden realizar proyectos enteramente con JavaScript (por ejemplo, este proyecto) o simplemente utilizarlo para ciertos aspectos puntuales de una aplicación nativa.

A diferencia de otras tecnologías similares como Phonegap [18], Cordova [19] o Ionic [20], React Native no se renderiza en el navegador, sino que renderiza una interfaz de usuario nativa, y esto es lo que le da gran ventaja sobre sus competidores. Como vemos en la figura 1.9, el código JavaScript se procesa con un compilador específico para cada plataforma. Aunque principalmente se usa para el desarrollo de aplicaciones para iOS y Android, también vale para hacer aplicaciones de escritorio, de realidad virtual y muchos otras plataformas [21].

El funcionamiento en tiempo de ejecución se separa en dos partes, unidas por un *puente* (bridge concept [22]). Uno de las partes se encarga de capturar los gestos del usuario y de pintar los elementos de la interfaz gráfica, esta parte es la que corre de forma nativa. La otra parte, es la de React, que se encarga de ejecutar el código JavaScript. El *puente*, desarrollado en C/C++, proporciona una comunicación bidireccional y asíncrona entre ambas partes. Este implementa el framework de Apple, JavaScriptCore [23], el cual expone una API para acceder a la máquina virtual de JavaScript, de manera que se puede inyectar código JavaScript en un programa C/C++. Esto es la comunicación desde la parte React hacia el *puente*. Por otro lado, de la parte nativa al puente, la comunicación depende de la plataforma de destino, es decir, en iOS, es simple ya que Objetive-C es una extensión de C, pero en el caso de Android, se necesita utilizar *Java Native Interface* [24] para que se pueda producir el dialogo con el *puente*.

Figura 1.9: Proceso de compilación de una aplicación React Native.

1.3.5.5. Redux

Figura 1.10: Logo de Redux.

En React, se puede gestionar el estado de cada componente individual sin necesidad de añadir nada. Redux [25] es una librería JavaScript que surge de la necesidad de gestionar el estado global de las aplicaciones React. Su funcionamiento está basado en la arquitectura FLUX (figura 1.11) [26].

El estado global de la aplicación esta contenido en el *Store*. Este es único, y su contenido es solo de lectura. Nunca se modifica, si no que se realiza una copia cada vez que se quiere cambiar algo. FLUX se divide en cuatro partes, que son:

- **Actions:** son objetos simples que definen como son las interacciones dentro de la aplicación. Este objeto siempre tiene un atributo *type*, que indica que hace esa acción, y va acompañado de los datos a actualizar.

- **Store:** los datos en el *store* solo pueden cambiar respondiendo a una *action*, no se pueden escribir directamente, solo leer. Cada vez que un dato del *store* cambia, debe emitir un evento de cambio. Puede, haber varios *stores* en una misma aplicación.

- **Dispatcher:** este se encarga de enviar las *actions* a todos los *stores* que tengan registrado este *dispatcher*.

- **Views:** la interfaz. Cuando desde una *view* se utiliza un dato del *store*, se debe suscribir a los cambios de este. Luego, cuando el *store* emite nuevos datos, la *view* se puede re-renderizar.

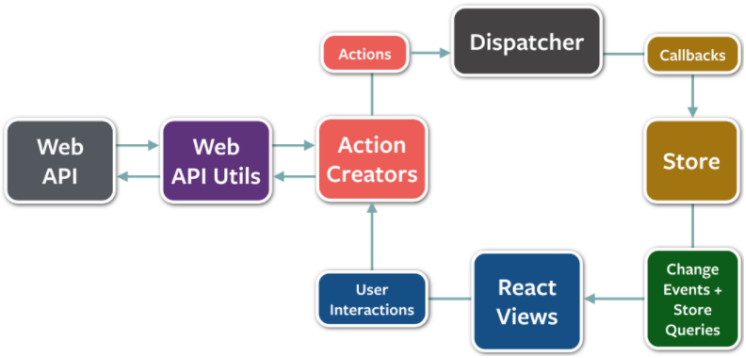

Figura 1.11: Arquitectura FLUX [26]

Para simplificar, el flujo que sigue FLUX es el siguiente: desde las *views* se envían las *actions*; el *dispatcher* manda la *action* a todos los *stores*; y, para cerrar el ciclo, los *stores* envían los datos nuevos a las *views*. Estos dos últimos pasos se realizan mediante unas funciones llamadas *reducers*, que actualiza y devuelve el *store* correspondiente a la *action* y datos recibidos.

En este proyecto, se ha utilizado, entre otras cosas, para guardar los datos del usuario logueado, para guardar las palabras que forman parte de una partida y para ir almacenando el log de actividad.

1.3.5.6. Expo

Figura 1.12: Logo de Expo.

Expo [27] es un framework que añade una capa sobre las API de React Native para facilitar el desarrollo de aplicaciones nativas, proporcionando métodos para acceder a características nativas del sistema operativo objetivo, sin necesidad de utilizar código nativo. Esto lo hace a través del Expo SDK, y nos permite características propias de aplicaciones nativas sin necesidad de utilizar Android Studio o Xcode.

Por otro lado, Expo también proporciona una forma de generar los paquetes de la aplicación desde la nube para poder subirlos directamente a las tiendas correspondientes.

1.3.5.7. MongoDB

Figura 1.13: Logo de MongoDB.

Mongo [28] es una base de datos NoSQL, orientada a documentos. Esto significa que a diferencia de las bases de datos convencionales, la información no se almacena en tablas, sino que se guardan en estructuras BSON. Esta es una especificación similar a JSON, lo cual la hace muy interesante combinada con un servidor escrito en JavaScript.

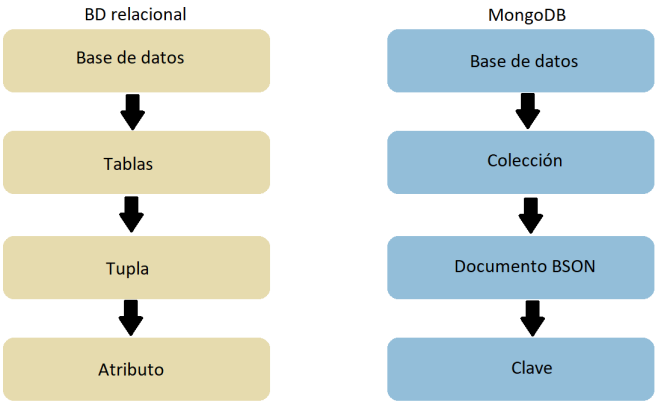

Figura 1.14: Equivalencia de términos de una base de datos SQL y MongoDB

La principal ventaja de utilizar este tipo de bases de datos es que son semi estructuradas, lo que aporta cierta libertad sobre los datos que se van a guardar. En contra tienen el rendimiento, que siendo bueno, no llega a alcanzar a las bases de datos SQL en determinados tipos de consultas, por ejemplo,

las que usan uniones, ya que esta operación no se puede hacer directamente y hay que ingeniárselas para poder llevarla a cabo.

1.3.5.8. Swagger

Figura 1.15: Logo de Swagger.

Swagger [29] es una herramienta que se utiliza para documentar las llamadas a la API, exponiendo su descripción, parámetros de entrada y estructura de las respuestas esperadas. También permite probar las llamadas directamente desde el propio Swagger, algo que resulta muy útil cuando se trabaja en equipos. Otra característica interesante es, que si el Swagger no tuviera conexión con el servidor, por ejemplo durante un despliegue, este es capaz de generar mocks para falsear las respuestas. En el apéndice C está disponible el Swagger de este proyecto.

1.3.5.9. Postman

Figura 1.16: Logo de Postman.

Postman [30] es una herramienta muy útil para probar las llamadas a la API. Esto es una gran ventaja durante el desarrollo ya que no hace falta tener un cliente enviando peticiones al servidor, sino que el propio Postman actúa como cliente. Permite enviar peticiones con sus cabeceras, cuerpo y otros parámetros, mediante una interfaz muy sencilla e intuitiva. En este proyecto solo se ha utilizado para probar las llamadas durante su desarrollo, pero también es capaz de hacer otras tareas como automatizar pruebas y monitorizar el rendimiento de la API. También permite tener varios espacios de trabajo para diferentes proyectos.

1.3.5.10. Google TTS y STT

Google Text-to-speech [31]: la síntesis de voz consiste en que un dispositivo reproduzca un texto de forma hablada. Gracias a tecnología de IA (inteligencia artificial) de Google y a su API, que expone esta utilidad, se consiguen reproducir textos con una voz muy natural. Este servicio está disponible en más de 40 idiomas.

Google Speech-to-Text [32]: este, se podría decir que es el caso contrario. Esta API permite el reconocimiento de voz para ser procesado y convertido a texto. Gracias a la IA de Google y las técnicas de machine learning, este sistema cada vez es más preciso. Este servicio no solo devuelve la palabra que ha entendido, sino que devuelve una lista de posibles palabras reconocidas (tantas como se elija en la petición) acompañadas de un número de 0 a 1, que representa la probabilidad de que esa palabra sea la que se ha pronunciado.

La combinación de ambas es la base para que un usuario pueda escuchar cómo se pronuncia una palabra, tratar de repetirla y recibir una retroalimentación de forma prácticamente instantánea.

1.4. Objetivos

El objetivo general de el proyecto es, mediante un frontend y un backend, desarrollar una aplicación multiplataforma y una API para implementar un juego de mejora de la pronunciación de un idioma extranjero. Este objetivo principal se puede dividir en los siguientes objetivos particulares:

- **Objetivo 1:** diseñar la interfaz multiplataforma de las diferentes pantallas.
- **Objetivo 2:** desarrollar las funcionalidades necesarias para los modos de juego de la aplicación, integrando sistemas ASR y TTS.
- **Objetivo 3:** desarrollar una API con vistas a ser ampliada en el futuro. Al ser independientes cliente y servidor, se puede modificar uno sin afectar al otro.

1.5. Metodología

Por la naturaleza del proyecto y el poco conocimiento de la tecnología se ha decidido aplicar un modelo iterativo e incremental [33], que brinda cierta flexibilidad en cuanto a cambios en los requisitos. En este caso, aplicar SCRUM [34], no parece demasiado realista al disponer de un solo desarrollador. Entre las ventajas de este modelo se encuentran las siguientes: el cliente final puede ver en cada iteración de una funcionalidad acabada, de esta manera se puede ver si el proyecto va por el camino deseado o hay que realizar algún cambio; menos gold-plating (requisitos innecesarios que piden los clientes); y, por último, este modelo permite que el proyecto sea abandonado temporalmente, lo cual, teniendo en cuenta la situación del estudiante, es una gran ventaja [33].

En la figura 1.17 se puede ver un esquema de cuáles son los pasos a seguir durante un desarrollo iterativo e incremental. En el capítulo 3 se habla en detalle de las acciones llevadas a cabo en cada iteración de este proyecto.

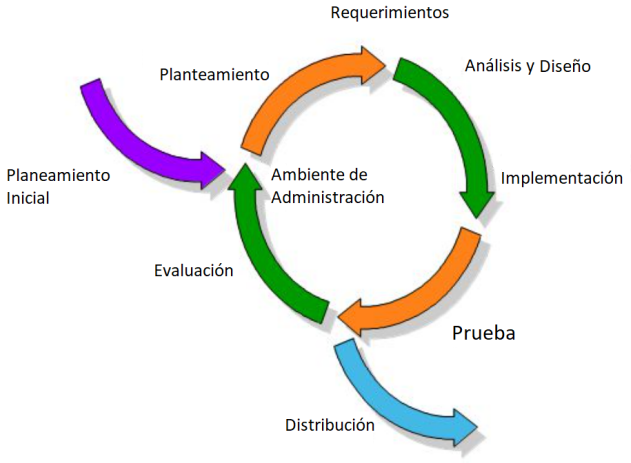

Figura 1.17: Esquema de un desarrollo iterativo e incremental [35]

1.6. Estructura de la memoria

- **Introducción:** se expone el proyecto desde una perspectiva general, hablando del contexto, metodología empleada, objetivos a conseguir, tecnologías implicadas y la estructura de la memoria.
- **Planificación:** descripción y estimación de las iteraciones con sus tiempos de entrega, estimación del coste del proyecto y descripción de riesgos y planes de acción.
- **Descripción de las iteraciones:** descripción del trabajo realizado en cada una de las iteraciones.
- **Estado final de la aplicación:** diagramas de análisis y diseño con sus correspondientes aclaraciones.
- **Pruebas:** descripción de las pruebas realizadas. Test de integración, unitarios y de usabilidad.
- **Conclusiones y trabajo futuro:** conclusiones sacadas del proceso y posibles mejoras de cara al futuro.
- **Apéndices:** distintos documentos adicionales como el manual de instalación y despliegue, la estructura de la API REST (Swagger), los acrónimos y el contenido del fichero adjunto.
- **Bibliografía:** referencias a páginas web, libros o proyectos consultados a lo largo del desarrollo.

Capítulo 2

Planificación

2.1. Planificación inicial

Para poder compatibilizar el horario con la situación del alumno de este TFG, con éste finalizando sus estudios y realizando las prácticas en empresa, se estableció al comienzo del proyecto un trabajo de 15 horas semanales desde principios de octubre hasta mediados de marzo, lo que sumaría un total de 360 horas, superando de esta forma las 300 horas mínimas que se especifica en la normativa vigente sobre el tiempo que se debe dedicar a un trabajo de fin de grado.

Durante estas 24 semanas de trabajo estimadas, se dedicarán 3 horas diarias, y se planificaron 7 iteraciones, repartidas de las siguiente forma:

- **1ª Iteración (1 semana - 15 horas)**
  En esta primera etapa, se realizará una reunión entre cliente, los tutores del proyecto, y desarrollador para definir la funcionalidad del producto y tomar nota de los requisitos. También se definirán las tecnologías y herramientas parra llevar a cabo el desarrollo.

- **2ª Iteración (1 semana - 15 horas)**
  En esta iteración se dedicará ha hacer pruebas con las tecnologías seleccionadas para el desarrollo. Los primeros pasos con React Native y con Express.

- **3ª Iteración (4 semanas - 60 horas)**
  Durante este periodo se va a realizar el análisis y diseño inicial del proyecto, con sus correspondientes diagramas. También, se desarrollará un prototipo funcional simple de la aplicación. Paralelamente, se trabajará en el diseño de las pantallas de esta, mediante una herramienta de maquetación. Se pedirá retroalimentación sobre ambas cuestiones.

- **4ª Iteración (4 semanas - 60 horas)**
  Implementación de la parte servidor del sistema. Consistirá en una serie de APIs, desarrolladas con Node.js, y una base de datos, no relacional, implementada con Mongo. Se llevarán a cabo pruebas de las APIs mediante Postman.

- **5ª Iteración (8 semanas - 120 horas)**
  Implementación de la aplicación cliente multiplataforma. Utilizando React-Native y Expo, se realizará la app, comenzando por las pantallas de login y registro y siguiendo por cada uno de los modos de juego.

- **6ª Iteración (2 semanas - 30 horas)**
  Una vez implementados todo y revisado por el cliente, se procede a realizar los cambios necesarios en la aplicación.

- **7ª Iteración (4 semanas - 60 horas)**
  Comienza la redacción de la memoria. Se desplegará el servidor en una máquina del grupo ECASIMM. También se realizarán las pruebas de integración y unitarias.

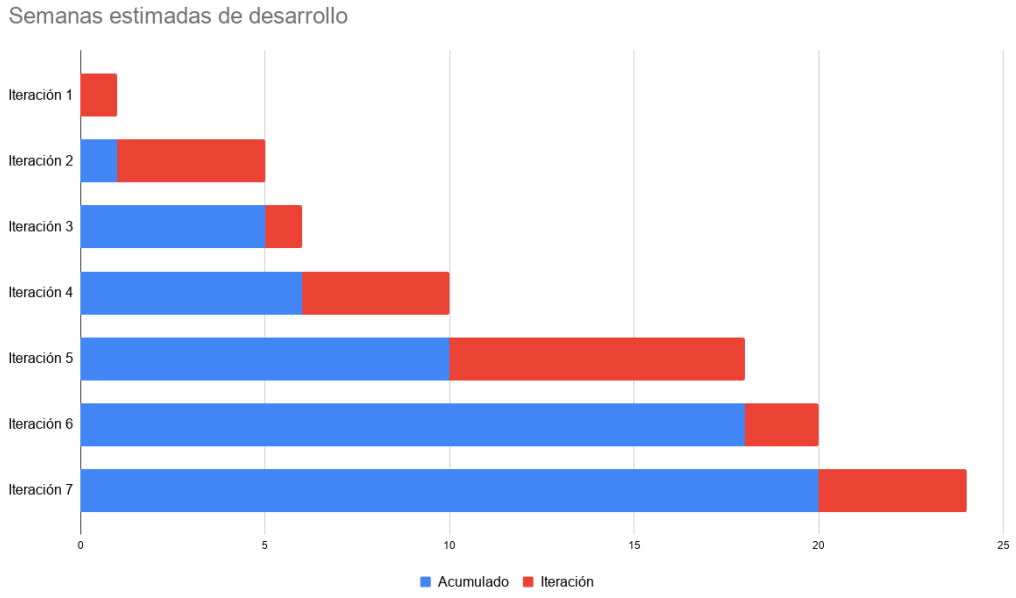

Figura 2.1: Distribución de las iteraciones según sus semanas de desarrollo.

2.2. Entorno tecnológico

En esta sección se enumeran las herramientas utilizadas para el desarrollo, tanto el software como los equipos.

2.2.1. Herramientas utilizadas

A continuación, se listan los programas y utilidades empleadas:

- **5kPlayer [36]**: Visualización de pantalla iPad en Windows.

- **Astah [37]**: Creación de diagramas UML.
- **Bitbucket [38]**: Almacenamiento de repositorios del proyecto en la nube.
- **Expo client para Android [39]**: Ejecutar aplicación en Android en modo desarrollo.
- **Expo client para iOS [40]**:Ejecutar aplicación en iOS en modo desarrollo.
- **Git for Windows [41]**: Control de versiones del código del proyecto.
- **Google Drive [42]**: Almacenamiento de ficheros en la nube relacionados con la memoria.
- **Google STT API [32]**: API para procesamiento de voz a texto (ASR).
- **Marvel [43]**: Creación de bocetos y prototipos.
- **Mongo DB Compass Community [44]**: Interfaz gráfica para MongoDB.
- **Nginx [45]**: Servidor para aplicación Node.
- **Node.js [11]**: Herramienta JavaScript para desarrollo tanto de servidor como de cliente. Unifica el lenguaje de programación (JavaScript) y el formato de los datos (JSON).
- **Overleaf [46]**: Editor LaTeX en la nube.
- **Oracle Virtual Box [47]**: Virtualización de máquinas.
- **Postman [30]**: Pruebas contra las APIs del proyecto.
- **React Native [16]**: framework que permite desarrollar con JavaScript y se compila a código nativo.
- **Swagger [29]**: Herramienta para la documentación de las API.
- **Skype [48]**: Aplicación de videollamadas para la comunicación con los tutores del proyecto.
- **Telegram [49]**: Aplicación de mensajería instantánea para la comunicación con los tutores del proyecto.
- **Vysor [50]**: Visualización de pantalla Android en Windows
- **Visual Studio Code [51]**: IDE para desarrollo tanto del backend como del frontend.

Módulos relevantes npm utilizado en el *backend*:

- **bcrypt [52]**: librería para codificar contraseñas.
- **express [15]**: infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones de Node.js.
- **jest [53]**: marco de trabajo para pruebas de aplicaciones JavaScript.
- **jsonwebtoken [54]**: generación de JsonWebToken (JWT).

- **mongoose [55]**: modelado de objetos mongodb para Node.js.
- **nodemon [56]**: ayuda para el desarrollo de aplicaciones Node, reiniciando el proyecto cada vez que se realizan cambios.
- **supertest [57]**: herramienta de pruebas para peticiones HTTP.

Módulos relevantes npm utilizado en el frontend:

- **Axios [58]**: cliente HTTP basado en promesas.
- **Expo [27]**: conjunto de herramientas y servicios construidos con React Native para facilitar el desarrollo.
- **i18n-js [59]**: internacionalización de la aplicación.
- **ramda [60]**: librería de funciones JavaScript.

2.2.2. Entorno de trabajo

En las siguientes tablas se describen los equipos utilizados, tanto para el desarrollo, como para las pruebas de la aplicación.

Ordenador portátil	
Asus X555L	
Hardware	
Procesador	Intel Core i5 5200u a 2.2GHz
RAM	8GB DDR3
SSD	256GB
HDD	1TB
GPU	NVIDIA GeForce 920M con 2GB DDR3 VRAM
Software	
Sistema operativo	Windows 10 Education
Arquitectura	64 bits

Tabla 2.1: Ordenador utilizado en desarrollo.

Monitor	
LG 27GL850	
Resolución	2560 x 1440
Tamaño	27”
Ratio de aspecto	16/9

Tabla 2.2: Monitor utilizado en desarrollo.

Smartphone	
Oneplus 7T	
Hardware	
Procesador	Qualcomm Snapdragon 855 Plus
RAM	8GB LPDDR4X
Almacenamiento	128GB UFS 3.0
GPU	Adreno 640
Pantalla	6,55”
Software	
Sistema operativo	OxygenOS basado en Android 10
Arquitectura	64 bits

Tabla 2.3: Smartphone utilizado en desarrollo.

Tablet	
iPad 2019	
Hardware	
Procesador	Apple A10 Fusion
RAM	3 GB
Almacenamiento	32GB
Pantalla	10,2”
Software	
Sistema operativo	iOS 13
Arquitectura	64 bits

Tabla 2.4: iPad utilizado en desarrollo.

Servidor	
Maquina virtual	
Hardware	
Procesador	Intel KVM proccessor a 2.2GHz
RAM	2GB
HDD	10GB
Software	
Sistema operativo	Ubuntu 20.04.1 LTS
Arquitectura	64 bits

Tabla 2.5: Servidor utilizado para despliegue.

2.3. Estimación de riesgos

En esta sección se procede a describir y enumerar los posibles riesgos que pueden afectar al desarrollo normal del proyecto. Además de la descripción se incluye la probabilidad de que ocurra y el retraso en días que supondría. En la tabla 2.6 se pueden ver estos riesgos representados mediante un identificador, nombre, descripción, probabilidad de 0 a 1 y retraso en días.

Por otro lado, en la tabla 2.7 se reflejan los planes de acción a llevar a cabo en caso de que alguno de los riesgos ocurra. También se menciona la exposición al riesgo, que representa el promedio de tiempo que se retrasaría el proyecto debido a cada uno de los riesgos, calculado como la multiplicación de la probabilidad de que el riesgo suceda y el retraso en días estimado.

ID	Riesgo	Probabilidad	Descripción	Retraso
R01	Suspender alguna asignatura en convocatoria ordinaria	0,5	El desarrollador suspende alguna asignatura y tiene que presentarse en convocatoria extraordinaria	14
R02	Suspender alguna asignatura en convocatoria extraordinaria	0,5	El desarrollador suspende alguna asignatura y no puede presentar el proyecto hasta el año siguiente	270
R03	Enfermedad	0,3	El desarrollador cae enfermo	3
R04	Falta de experiencia con los lenguajes de desarrollo	0,4	Dificultada a la hora de desarrollar debido al desconocimiento de las tecnologías utilizadas	14
R05	Falta de experiencia con la API de audio de Google	0,6	Dificultada a la hora de desarrollar debido al desconocimiento del uso de la API STT de Google	7
R06	Planificación inadecuada	0,7	Error al estimar riesgos y/o disponibilidad del desarrollador	20
R07	Problemas con los equipos informáticos	0,2	Fallo del algún dispositivo durante el desarrollo	2
R08	Modificación de los requisitos	0,3	Modificación en los requisitos que implique cambios en lo ya hecho o ampliación de funcionalidad	7

Tabla 2.6: Tabla de riesgos.

ID	Riesgo	Exposición al riesgo	Plan de acción
R01	Suspender alguna asignatura en convocatoria ordinaria	7	Hacer más horas después de la convocatoria extraordinaria
R02	Suspender alguna asignatura en convocatoria extraordinaria	135	Posponer la presentación al año siguiente
R03	Enfermedad	0,9	Recuperar horas una vez pasada la enfermedad
R04	Falta de experiencia con los lenguajes de desarrollo	5,6	Leer documentación y dedicar horas necesarias en los días sucesivos
R05	Falta de experiencia con la API de audio de Google	4,2	Leer documentación de Google y dedicar horas necearías en los días sucesivos
R06	Planificación inadecuada	14	Replanificar y hacer horas extra necesarias
R07	Problemas con los equipos informáticos	0,4	Sustituir el equipo
R08	Modificación de los requisitos	2,1	Hacer el código flexible y escalable

Tabla 2.7: Tabla de planes de acción para los riesgos.

2.4. Estimación de costes

En esta sección se procede a detallar la estimación de los costes del proyecto. Desde el salario bruto del desarrollador hasta los costes de los equipos, licencias y servicios. Todas estas cantidades se contabilizan con IVA.

2.4.1. Salario del trabajador

Ya que este proyecto se ha realizado para la asignatura de TFG, se da un coste estimado del salario de un trabajador. Según ”Hack a Boss” [61], un bootcamp de la Xunta de Galicia, el salario anual de un programador junior en 2020 ronda entre los 16.000 y 22.000€, siendo la media unos 18.000€. Se tomará este último valor para hacer los cálculos en este caso. Teniendo en cuenta una jornada anual de 1.800 horas el salario bruto es de 10€/hora. Con un trabajo estimado de 360h, el coste total del salario bruto del desarrollador es de 3600€.

2.4.2. Costes indirectos aplicados al espacio de trabajo

El proyecto se ha realizado enteramente en el domicilio personal del trabajador, entre otras cosas, por la pandemia de la COVID-19. Teniendo esto en cuenta, se calcularán los gastos estimando el coste de los servicios de abastecimiento de la vivienda y el alquiler mensual de esta. El alquiler corresponde al de una piso de 70m^2 en la zona centro de Valladolid.

Servicio	**Coste mensual**	**Nº de meses**	**Jornada**	**Coste total**
Luz y gas	200€	6	3 h/día	150€
Agua	7€	6	3 h/día	5,25€
Internet	58€	6	3 h/día	43,5€
Alquiler	500€	6	3 h/día	375€
Total				**948,75€**

Tabla 2.8: Tabla de estimación de costes aplicados al espacio de trabajo.

2.4.3. Costes del hardware

Para estimar los costes de los equipos utilizado se tendrá en cuenta el coste en el momento de compra, las horas de uso desde entonces y las horas de uso dedicadas al proyecto. Para hacer el cálculo se comparan las horas totales con las horas de uso durante el proyecto.

Equipo	Coste en el momento de compra	Nº de horas del equipo	Nº de horas de uso	Coste total
Portátil	499€	43.000	360	4,17€
Pantalla	400€	8.760	360	16,43€
Smartphone	599€	13.140	360	16,4€
Tablet	379€	4.380	180	15,57€
Total				**52,57€**

Tabla 2.9: Tabla de estimación de costes del hardware.

2.4.4. Costes del software y servicios

Se ha hecho uso de planes gratuitos de algunas herramientas como Marvel y Bitbucket, ya que para un proyecto es suficiente. Sin embargo, para otras se ha tenido que utilizar una licencia de pago. A continuación se dan los detalles para cada herramienta.

2.4.4.1. Licencia desarrollador Android

El coste para poder tener una cuenta de desarrollador para publicar una app en Google Play es de 25$ [62], unos 21€ a día 26/12/2020. Este coste se paga una vez, y es válido para siempre.

2.4.4.2. Licencia desarrollador iOS

En el caso de Apple, funciona diferente. La licencia de desarrollador se paga anualmente, y el coste de esta es de 99$ [63], unos 81,20€ a día 26/12/2020. Para la suma total sólo se tendrá en cuenta un año.

2.4.4.3. Google Speech to text

Para esta API de Google se paga por uso, concretamente 0,006$ [64] por cada bloque de 15 segundos de audio, siendo los primeros 60 minutos de cada mes gratuitos. Estos 60 min. equivalen a unas 240 palabras, por lo que en desarrollo no se ha llegado a superar ese número.

2.4.4.4. Astah

El coste de Astah profesional para una persona es de 7,5€ al mes. Como sólo se usó durante el mes de la tercera iteración solo se sumara un mes a la suma total.

Software	Coste
Licencia Android	21€
Licencia iOS	81,20€
Licencia Astah	7,5€
Total	**109,70€**

Tabla 2.10: Tabla de estimación de costes de software y servicios.

2.4.5. Costes totales

En la tabla 2.11 se puede ver un resumen, con el total de todos los costes detallados anteriormente.

Software	Coste
Salario desarrollador	3600€
Espacio de trabajo	948,75€
Hardware	52,57€
Software y servicios	109,70€
Total	**4711,02€**

Tabla 2.11: Tabla de estimación de costes totales.

2.5. Planificación final

El proyecto no ha cumplido los tiempos estimados inicialmente. Principalmente, debido a las consecuencias asociadas a los riesgos R01 y R02, es decir, suspender dos asignaturas en convocatoria extraordinaria. Por lo tanto, se llevó a cabo el plan de acción preparado para estas, que consistió en retrasar el proyecto al curso siguiente.

Antes de que esto ocurriera, ya se habían producido otros retrasos, uno de ellos debido al riesgo R05. A la hora de crear el prototipo inicial, se tardó más de lo esperado en conseguir una configuración correcta para hacer funcionar la API "Speech to text"de Google. Esto se tradujo en un retraso de una semana, en la tercera iteración, hasta poder presentar un prototipo con las características deseadas.

Otro de los problemas que aparecieron durante el desarrollo tuvo que ver con el riesgo R04. La falta de experiencia con la tecnología de desarrollo hizo que en ciertos momentos el avance fuera más lento de lo esperado, teniendo que leer documentación y mirar diferentes tutoriales para poder solventar el problema. Si bien el lenguaje utilizado en ambas partes de la aplicación es JavaScript, más que problemas con el lenguaje en sí, las dificultades surgieron con las particularidades para, por un lado, hacer la API con Express y MongoDB, y por otro, hacer la aplicación con React Native y Expo. Debido

a esto, se vió afectada la duración de las iteraciones 4 y 5, con 2 y 1 semanas respectivamente. Cabe comentar que realmente este retraso no afectó directamente a la duración del proyecto, ya que, como se ha explicado anteriormente, la fecha de entrega se retrasaba hasta el curso siguiente.

Además, cabe mencionar el periodo de confinamiento debido a la pandemia de la COVID-19. Esta situación, sumada a que el proyecto no se podría presentar, hizo que durante unos meses el tiempo dedicado al proyecto fuera en decremento, afectando a las iteraciones 4 y 5. En ningún momento se llegó a parar el desarrollo por completo pero si se dedicaban menos horas de las 15 semanales propuestas en un principio.

Teniendo en cuenta todo lo comentado anteriormente, en octubre de 2020, con la vuelta a la rutina, se retoma el horario de 15 horas semanales. En este punto, el proyecto se encuentra a mitad de la iteración 5. El final de esta iteración coincide con el periodo de exámenes de convocatoria extraordinaria de fin de carrera, dos semanas en las que se para el proyecto para que el desarrollador pueda dedicarse enteramente a preparar los exámenes.

Una vez el desarrollador acabó los exámenes, se retoma la iteración 6, en la que también se produce un retraso de 2 semanas debido a los diferentes bugs encontrados a la hora de probar la aplicación por parte del cliente. A parte de los bugs, el cliente también se percató de que sería útil implementar ciertas funcionalidades que no se habían tenido en cuenta, como por ejemplo la autenticación con un ”Refresh Token” o la pantalla de historial del jugador. A partir de aquí, la iteración 7 se desarrolló con normalidad.

Analizando el desarrollo del proyecto, haciendo referencia al riesgo R06, se puede deducir que en un principio se hizo una planificación demasiado optimista (ver figura 2.2), subestimando el desconocimiento de las tecnologías que se iban a utilizar. Finalmente, la duración real del proyecto, en semanas de trabajo dedicadas (sin tener en cuenta el tener que presentarlo un año más tarde), ha sido de 30 semanas frente a las 24 estimadas, que se traducen en 450 horas frente a 360 estimadas. La distribución por iteraciones quedaría de la siguiente manera:

- **1ª Iteración: 1 semana - 15 horas**
- **2ª Iteración: 1 semana - 15 horas**
- **3ª Iteración: 5 semanas - 75 horas**
- **4ª Iteración: 6 semanas - 90 horas**
- **5ª Iteración: 9 semanas - 135 horas**
- **6ª Iteración: 4 semanas - 60 horas**
- **7ª Iteración: 4 semanas - 60 horas**

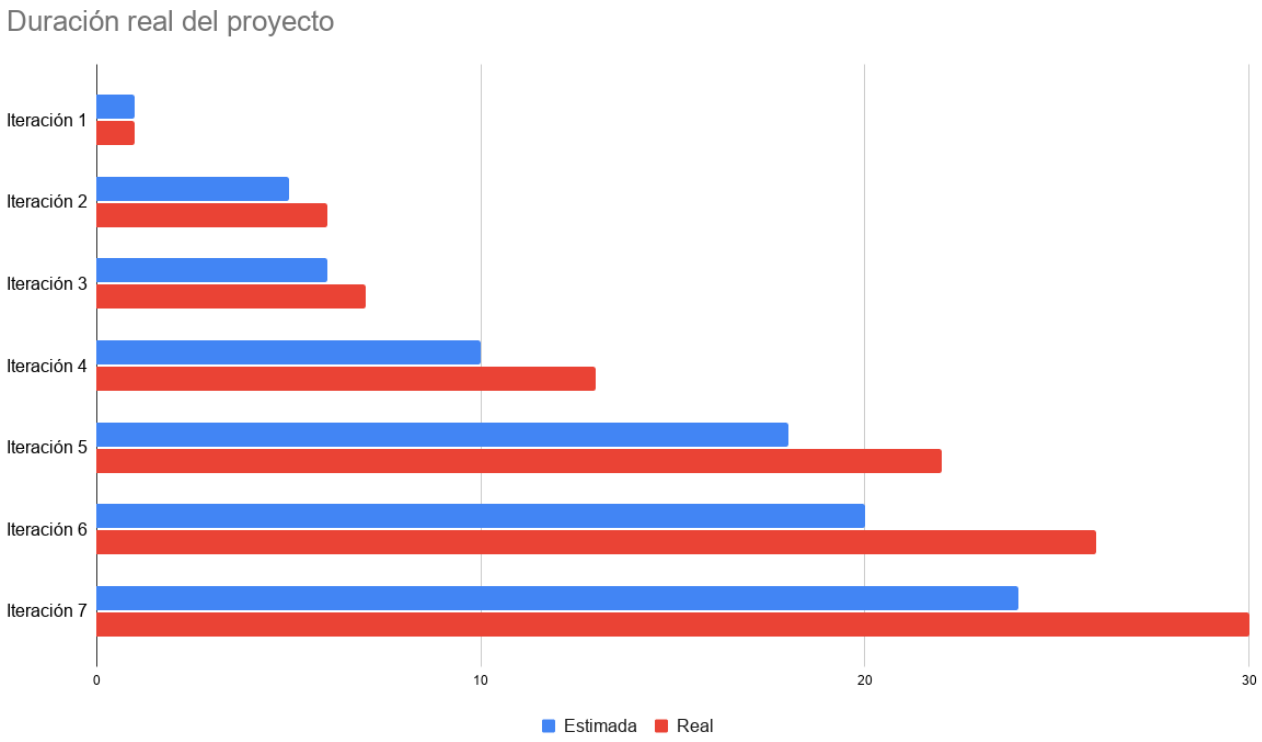

Figura 2.2: Comparación de la estimación original con la duración real.

Capítulo 3

Descripción de las iteraciones

En este capítulo se va a exponer el trabajo realizado durante las diferentes iteraciones a lo largo del desarrollo del proyecto. Entre otras cosas, se van a comentar los requisitos del proyecto, de las pruebas realizadas con las tecnologías, el prototipo implementado, el diseño de todas las pantallas de la parte *frontend* y, finalmente, las implementaciones primero del *backend* y luego del *frontend*.

3.1. Iteración 1

Para iniciar el proyecto, se produce una reunión del desarrollador con el cliente, en este caso, los tutores del TFG. En esta primera reunión, el cliente expone las características del proyecto de manera que se puedan identificar los diferentes requisitos y las correspondientes historias de usuario.

Una vez se tiene un contexto general de la dimensión y funcionalidades del proyecto se procede a realizar una estimación de los tiempos de desarrollo para cada parte. Esta planificación se puede ver detalladamente en la sección 2.1 de la memoria.

3.1.1. Descripción de requisitos

3.1.1.1. Requisitos funcionales

A continuación se describen los diferentes requisitos funcionales del proyecto. Estos requisitos describen detalladamente los servicios que se deben implementar. En la tabla 3.1 se se muestran estos requisitos con su descripción y prioridad, en una escala de ”Alta”, ”Media”, y ”Baja”.

ID	Nombre	Descripción	Prioridad
RF01	Registro de un usuario	El sistema permitirá crear nuevos usuarios	Alta
RF02	Identificación de un usuario	El sistema permitirá que los usuarios existentes se identifiquen	Alta
RF03	Cerrar sesión	El sistema permitirá que los usuarios cierren la sesión	Alta
RF04	Crear juego	El sistema permitirá crear un juego nuevo	Alta
RF05	Crear grabación	El sistema permitirá crear una grabación (para almacenarla en base de datos)	Alta
RF06	Crear log de partida	El sistema permitirá crear un log con los datos de una partida	Alta
RF07	Mostrar historial	El sistema permitirá mostrar el historial de un jugador	Media
RF08	Elegir modo de juego	El sistema permitirá elegir qué modo de juego se va a jugar	Alta
RF09	Mostrar resumen	El sistema permitirá mostrar el resumen final de una partida	Media
RF10	Grabar audio	El sistema permitirá al usuario grabar su voz	Alta
RF11	Reproducción de audio	El sistema permitirá al usuario reproducir un audio	Alta
RF12	Reproducción de vídeo	El sistema permitirá al usuario reproducir un vídeo	Media
RF13	Mostrar idiomas	El sistema mostrará al usuario que idiomas puede elegir para practicar	Alta
RF14	Mostrar fonema	El sistema mostrará al usuario que fonemas puede elegir para practicar	Alta

Tabla 3.1: Tabla de requisitos funcionales.

3.1.1.2. Requisitos no funcionales

En este apartado, se procede a describir los requisitos no funcionales. Estos definen propiedades emergentes del sistema o necesidades técnicas de éste. En la tabla 3.2 se muestran con una estructura similar a la utilizada para los funcionales solo que en este caso en lugar de utilizar prioridad se utiliza relevancia con una escala de ”Critica”, ”Deseable” o ”Baja”.

ID	Nombre	Descripción	Relevancia
RNF01	Multiplataforma	El sistema se podrá utilizar al menos en iOS y Android	Crítica
RNF02	Almacenamiento en base de datos	El sistema guardará los datos de usuarios, partidas y sonidos en una base de datos MongoDB	Crítica
RNF03	Formato de palabras	El sistema utilizará un formato JSON para la gestión de las palabras	Crítico
RNF04	Formato de logs	El sistema utilizará un formato JSON para guardar los logs	Deseable
RNF05	Almacenamiento de listas de palabras	Las listas de palabras se almacenarán como un archivo JSON en el servidor	Crítico
RNF06	Escalabilidad de listas de palabras	El sistema permitirá añadir nuevas listas de palabras manualmente	Crítico
RNF07	Formato audio	El sistema guardará los audios en un formato reproducible, .awb en Android y .wav en iOS	Crítica
RNF08	Notificación de errores	El sistema informará al usuario en caso de que ocurra algún error	Crítica
RNF09	Facilidad de instalación	Se debe poder instalar el sistema siguiendo el manual de instalación	Deseable
RNF10	Protección de datos sensibles	El sistema debe encriptar las contraseñas de los usuarios	Crítica
RNF11	Identificación mediante token	El sistema utilizará el estándar JWT para las comunicaciones con la API	Crítica

Tabla 3.2: Tabla de requisitos no funcionales.

3.1.1.3. Requisitos de información

Por último, en la tabla 3.3 se detallan los requisitos de información. Estos indican el tipo de información que se necesita guardar en el sistema.

ID	Nombre	Descripción
RI01	Datos de usuario	Los campos a guardar sobre los usuarios son: ■ Identificador único ■ Nombre ■ Contraseña ■ Correo electrónico ■ País de origen ■ Idioma materno ■ Lista de tokens
RI02	Datos de partida	Los campos a guardar sobre las partidas son: ■ Identificador único ■ Modo de juego ■ Fecha de la partida ■ Puntuación ■ Idioma que se está practicando ■ Fonema que se está practicando
RI03	Datos de las grabaciones	Los campos a guardar sobre las grabaciones son: ■ Identificador único ■ Nombre del archivo de audio en el servidor ■ Código de lenguaje con región ■ Sistema operativo en el que se ha grabado ■ Texto de la palabra que se ha grabado
RI04	Datos de las listas de palabras	Las listas deben ser archivos JSON, ver *más abajo* la estructura y con el siguiente formato de nombre: ■ **tipoLista_codigoldioma_codigoRegion**.json ■ Ej.: consonant_en_us.json, para un sonido consonante, de habla inglesa de la región de EEUU.

Tabla 3.3: Tabla de requisitos de información.

Ejemplo de formato de una lista de palabras:

```
[
  {
    "id": string,           // Identificador de la lista
    "name": string,         // Nombre descriptivo de la lista
    "short_name": string,   // Nombre corto de la lista
    "pairs": [{             // Pares de palabras de la lista
        "words": [{         // Array con las primeras palabra de cada par
            "word": string, // Primera palabra del par
            "trans": string // Transcripcion fonetica de la palabra
        }],
        {
        "words": [{         // Array con las segundas palabras de cada par
            "word": string, // Segunda palabra del par
            "trans": string // Transcripcion fonetica de la palabra
        }]
    }]
  }
]
```

Estas listas de palabras estarán almacenadas en una carpeta del servidor a la que se podrán añadir nuevas listas de forma manual, y éstas serán reconocidas automáticamente por el sistema. Como cabe la posibilidad de que hayan varias listas de la misma región, de diferente tipo, de cara al usuario se tratarán como una única lista presentándole una lista unificada para cada región.

Por otro lado, se guardarán los archivos JSON con los logs de las partidas. En el servidor habrá una carpeta de usuarios, en la que para cada usuario existe otra carpeta de logs. Dentro de esta se almacenarán los archivos de log por día, el cual contiene todas las partidas jugadas por el usuario ese día en concreto. La estructura de estos archivos es un array de objetos JSON. Cada elemento de este array contiene la información de cada partida, incluyendo los datos del usuario y los eventos que han ocurrido durante la partida. Su estructura se puede ver en detalle en el apartado 4.4.4.

3.2. Iteración 2

En esta segunda iteración se comienza la familiarización con las tecnologías elegidas para realizar del proyecto. Comenzando por hacer pequeños proyectos del estilo ”Hola mundo” tanto con React Native como con Express. El objetivo de estas pruebas es adquirir los conocimientos básicos para arrancar el proyecto.

3.2.1. Implementación de código básica

Por un lado, para la primera toma de contacto con React Native, siguiendo los consejos de los propios desarrolladores, se ha hecho uso de la herramienta Expo, recomendada para desarrolladores sin experiencia. Siguiendo el tutorial proporcionado en la misma página de Expo [65], se puede ver de una manera simple cómo iniciar un proyecto y hacer el primer ”Hola mundo”.

Se ha elegido React Native como tecnología de desarrollo frente a Ionic, que era otra de las propuestas, debido a la forma de compilar el código de forma nativa en contraposición a Ionic que sería más parecido a una Progressive Web Application (PWA). Además, echando un vistazo a la página npmtrends [66] se puede ver la tendencia de uso de los últimos años de ambas tecnologías:

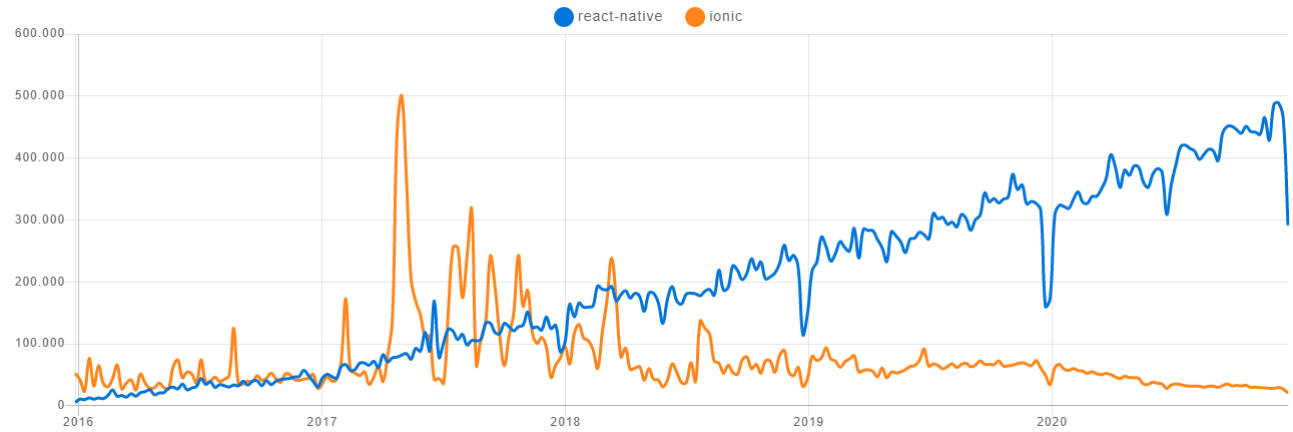

Figura 3.1: React Native VS Ionic.

El hecho de que React Native tenga una mayor comunidad será de gran ayuda a la hora de resolver los diferentes problemas que vayan surgiendo a lo largo del desarrollo.

Por otra parte, para el lado del servidor se ha seguido otro tutorial [67] en el que se explica desde la estructura de un proyecto Express, hasta como utilizar la autenticación con JWT, pasando por la conexión con una base de datos Mongo utilizando la librería Mongoose.

Para la base de datos se ha elegido Mongo para aprovechar las ventajas que nos brinda el modelo conocido como ”MERN stack” [68], que, en resumen, proporciona compatibilidad entre las estructuras de datos ya que usa un formato similar a JSON, el cual coincide con la estructura de los objetos en JavaScript.

3.2.2. Despliegue de la tecnología: estado de la cuestión de herramientas actuales

Primero, para el desarrollo de ambas partes de la aplicación se ha elegido como IDE el Visual Studio Code, perfecto para desarrollo con JavaScript gracias a sus múltiples extensiones. Una vez configurado, por una parte se crea una aplicación de React Native con el Command Line Interface (CLI) de Expo, este nos proporciona un esqueleto básico para una aplicación. Por otro lado, se crea un proyecto Express. Para la estructura de directorios de este se usa una plantilla, ya que no se dispone de un CLI que lo haga automáticamente. También se instalan otras herramientas secundarias que se utilizarán durante el desarrollo como MongoDB Compass, Postman o el cliente de Expo en los dispositivos móviles de prueba.

En esta etapa también se crean los repositorio Git, uno para el servidor y otro para la aplicación. Como servicio de alojamiento se ha elegido Bitbucket, ya que es con el que el desarrollador está más

familiarizado.

3.2.3. Pruebas de concepto

El objetivo de estas pruebas ha sido realizar pequeños desarrollo en ambos proyectos para probar a nivel conceptual el uso de las diferentes técnicas que se prevé que harán falta en un futuro, por ejemplo, trabajar con el formato JSON y los objetos JavaScript.

A parte de esto, se ha hecho alguna prueba contra la API Speech-to-text de Google, no solo para comprobar el funcionamiento, sino que también ha servido para ver el formato de la respuesta que genera.

3.3. Iteración 3

En esta iteración se realiza el análisis y diseño inicial de la aplicación. Para representarlo se preparan los siguientes diagramas: de casos de uso, del modelo de dominio, de actividad, modelo de datos, de paquetes y de despliegue. También, se detallan en tablas las historias de usuario. Todo esto, se puede ver con mayor detalle en el capítulo 4.

Además, se ha desarrollado un prototipo que incluye la funcionalidad básica de la aplicación. Este prototipo consiste en una aplicación móvil con la que se puede grabar un audio y después por pantalla se mostrará el texto correspondiente. Entrando un poco más en detalle, el flujo del prototipo es el siguiente: primero se pulsa un botón que activa la grabación, esta se sube al servidor y se almacena. Desde el servidor se hace una llamada a la API de Google y una vez tiene la respuesta, se devuelve el valor a la aplicación móvil.

El mayor problema encontrado al desarrollar este prototipo fue dar con una configuración correcta para que la API STT reconociera correctamente el audio que se mandaba. Finalmente, con la ayuda de la documentación oficial [69] y algún ejemplo encontrado en la web [70], se ha dado con una configuración compatible con iOS y Android.

También, durante esta fase se han ido diseñando las pantallas con la ayuda de la herramienta de maquetado Marvel [43]. Estas pantallas están basadas en las descripciones de los modos de juego dadas por el cliente. A continuación se pueden ver las pantallas y sus descripciones.

- **Login**: esta pantalla debe incluir un formulario para el login, selección de idioma para la aplicación y botón para creación de cuenta nueva (ver Figura 3.2).

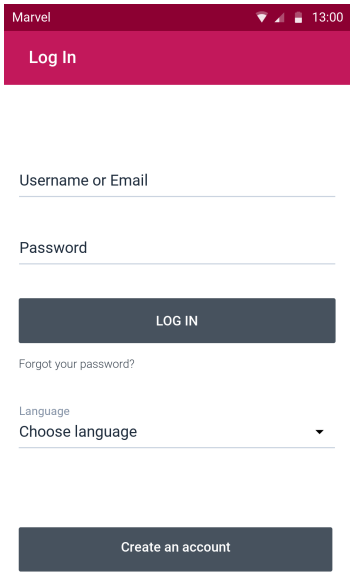

Figura 3.2: Pantalla de inicio de sesión.

- **Selección de idioma**: en esta pantalla se debe poder el elegir el idioma que se desea practicar (ver Figura 3.3).

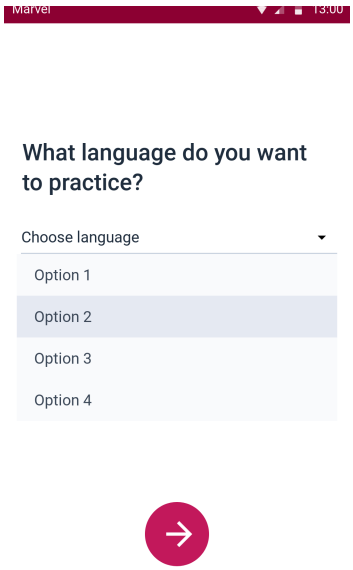

Figura 3.3: Pantalla de selección de lenguaje.

- **Selección de sonido**: en esta pantalla se debe poder el elegir el sonido que se desea practicar (ver Figura 3.4).

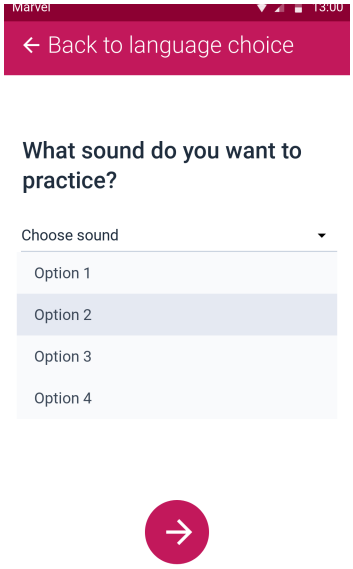

Figura 3.4: Pantalla de selección de sonido.

- **Menú principal**: en esta pantalla se podrá elegir entre los diferentes modos de juego, acceder a las puntuaciones o al menú de opciones (ver Figura 3.5).

Figura 3.5: Pantalla de menú principal.

- **Exposición**: aparecen dos textos y la app lo tiene que reproducir 5 veces seguidas, y una vez hecho esto el usuario tiene que grabar ambas palabras. Puede haber n rondas, en principio 3. Sin tiempo por ronda (ver Figura 3.6).

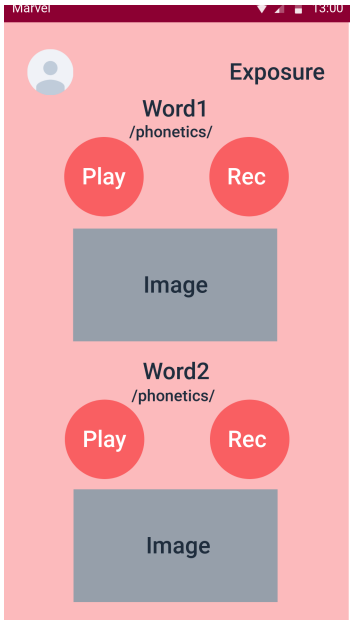

Figura 3.6: Pantalla de modo de juego exposición.

- **Percepción**: el sistema reproduce una palabra y el usuario elige entre dos en la interfaz, 10 rondas y 10 segundo por ronda. Añadir un botón de escuchar otra vez (ver Figura 3.7).

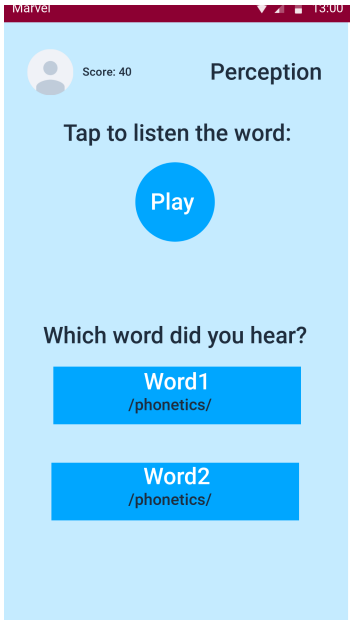

Figura 3.7: Pantalla de modo de juego Percepción.

- **Pronunciación**: aparecen dos palabras, el usuario las graba y se puntúa. Limitación de tres intentos si falla, si acierta se bloquea. Cuando falla el sistema dice lo que ha entendido y muestra consejo. Se muestra un botón de reproducir palabra por el TTS, que este visible siempre. Cinco rondas de 60 segundos cada una. Debajo de las palabras se debe mostrar la transcripción fonética (ver Figura 3.8).

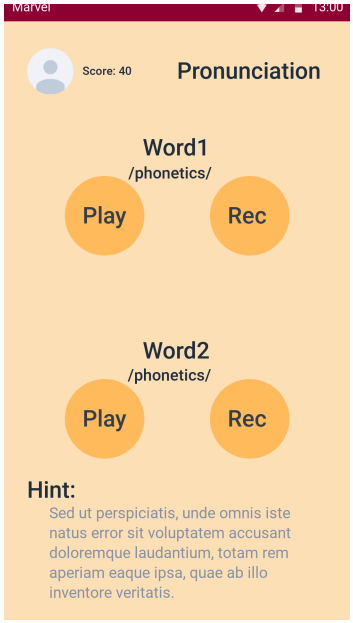

Figura 3.8: Pantalla de modo de juego pronunciación.

- **Mixto**: este modo alterna una ronda de percepción con una de pronunciación. Diez rondas con los tiempos de los modos originales (ver Figura 3.9).

Figura 3.9: Pantalla de modo de juego mixto.

- **Memoria**: se muestra un botón de escucha, al pulsar el sistema reproduce 3 palabras (de 3 pares). En la pantalla aparecen las 6 palabras de los 3 pares, sin transcripción fonética. Aparece el mensaje "selecciona en el orden de escucha". Tres rondas de 30 segundos (ver Figura 3.10).

Figura 3.10: Pantalla de modo de juego Memoria.

- **Puzzle**: Aparece en la pantalla el símbolo fonético en cuestión (uno del par). También aparecen por pantalla 6 palabras de las cuales de 0 a 6, aleatoriamente, tendrán ese sonido. No aparece la transcripción fonética. El usuario deberá seleccionar las palabras que contengan ese sonido. Tres rondas de 30 segundos (ver Figura 3.11).

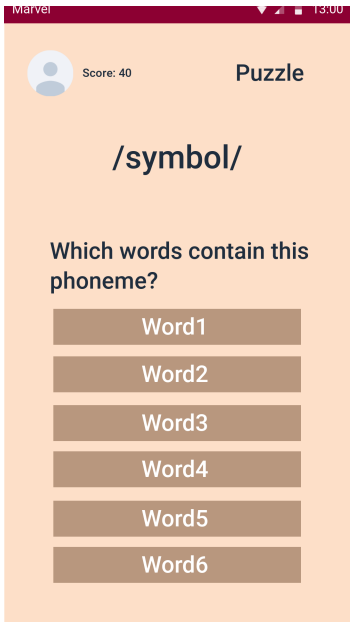

Figura 3.11: Pantalla de modo de juego Puzzle.

- **Teoría**: aparece un vídeo. El usuario reproduce el vídeo hasta el final, no hay puntuación (ver Figura 3.12).

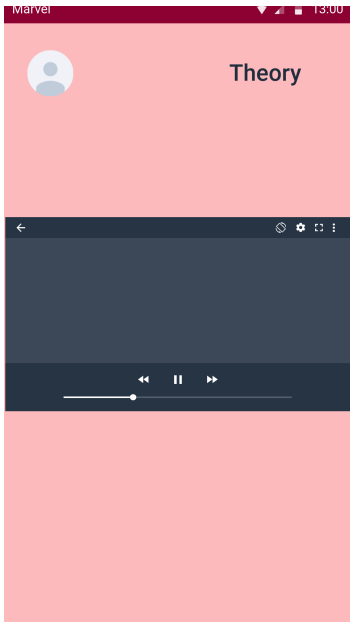

Figura 3.12: Pantalla de modo de juego Teoría.

- **Puntuación final**: en esta pantalla se debe mostrar la puntuación final obtenida al terminar una partida (ver Figura 3.13).

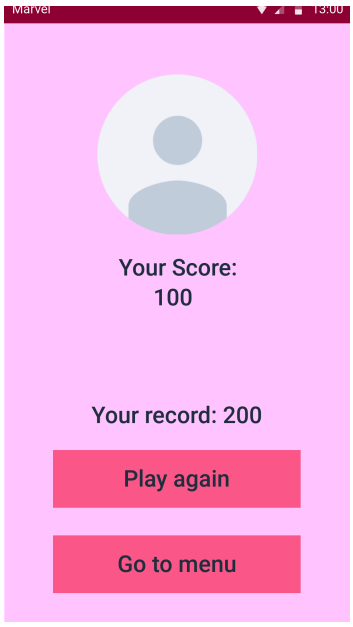

Figura 3.13: Pantalla de puntuación final de una partida.

- **Historial de puntuaciones**: en esta pantalla se debe mostrar las ultimas puntuaciones del usuario para el sonido elegido (ver Figura 3.14).

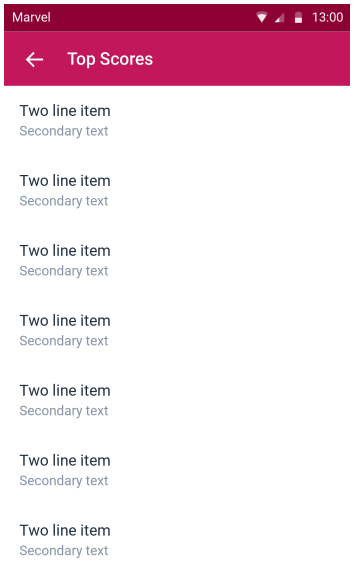

Figura 3.14: Pantalla de historial de puntuaciones.

- **Opciones**: en esta pantalla se debe mostrar las opciones de configuración de la aplicación (ver Figura 3.15).

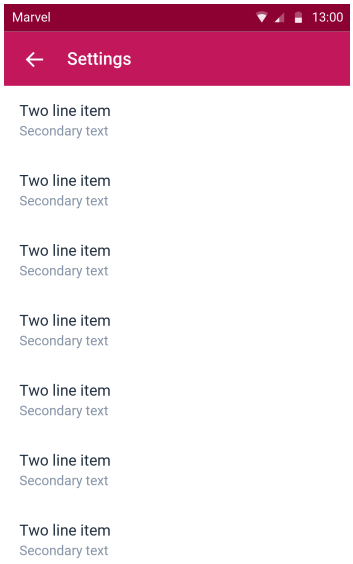

Figura 3.15: Pantalla de opciones.

- **Registro**: en esta pantalla debe mostrar un formulario con los campos necesarios para el registro de un usuario. Los campos que se incluyen son: username, email, password, confirmación de password, imagen. El email y username no se puede cambiar (ver Figura 3.16).

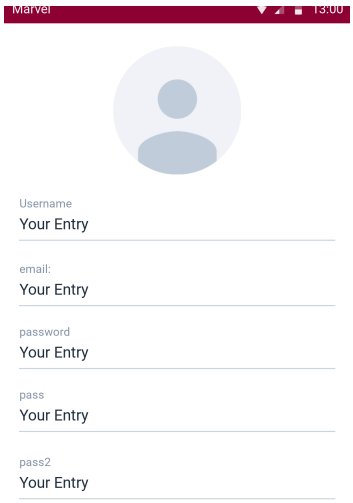

Figura 3.16: Pantalla de registro.

- **Selección de dificultad**: en esta pantalla se debe poder elegir una dificultad (ver Figura 3.17).

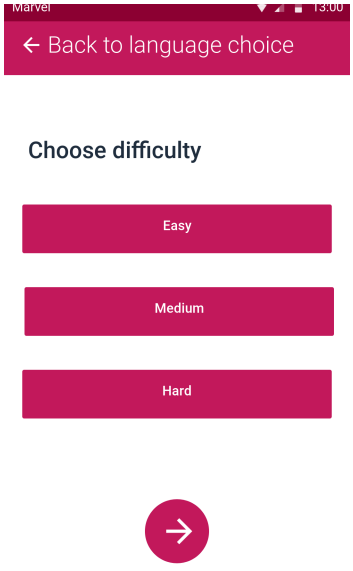

Figura 3.17: Pantalla de selección de dificultad.

3.4. Iteración 4

Tras presentar las pantallas al cliente, se detectaron diferentes ausencias en la interfaz, como el temporizador o el número de ronda. Estos cambios quedan anotados y se añadirán en la implementación final de la aplicación. Una vez queda aprobada la interfaz se comienza con el desarrollo de la parte de servidor del proyecto. Esta consiste en una serie de llamadas para gestionar la base de datos y la conexión a la API STT de Google.

3.4.1. Desarrollo Backend

Primero, se desarrolla el código necesario para las llamadas relacionadas con la gestión del usuario, la creación, la identificación, el inicio de sesión y cierre. La identificación de usuarios contra la API se hará mediante tokens, con el estándar JWT. Cada vez que un usuario se registra se le proporciona un token con una caducidad y un refresh token, que servirá para identificarse una vez el anterior caduque. Si ambos han caducado entonces será necesario que el usuario se vuelva a identificar.

De ahora en adelante, todas las llamadas se implementan con seguridad, de manera que no se puedan utilizar sin un token válido. Las siguientes llamadas que se implementan son las que tienen que ver con la subida de grabaciones. Estas se guardan en una carpeta del servidor asociadas al usuario que las ha grabado, y en la base de datos se guarda un registro con la ruta de acceso y otros datos. También, en este momento se realiza la conexión a la API STT para una de las llamadas, la cual requiere de credenciales para Google Cloud Platform. Simplemente, se crea una cuenta y se genera una clave en formato JSON que se guarda en el servidor y se envía en dicha llamada.

A continuación, se crean las llamadas para generar las listas de palabras para cada modo de juego. Partiendo de las listas de palabras en formato JSON proporcionadas por el cliente, se seleccionan aleatoriamente las palabras necesarias. También se crea la llamada para poder guardar las partidas y para ver el historial.

Tras una reunión con los tutores se detectan errores en alguna de las llamadas, como que no se estaba pasando el idioma de la grabación, o, el más relevante, que la respuesta del STT a veces devuelve palabras con alguna letra en mayúscula, lo que produce un error al comparar estas con las de la lista. La solución que se toma para esto es que se devuelvan todas las palabras, tanto las que devuelve el STT, como las que se extraen de las listas, con todas sus letras en minúscula. Además, se pidió una modificación para guardar el log de partidas de cada usuario por días, en lugar de un log por cada partida como se estaba haciendo hasta el momento.

3.4.2. Pruebas funcionales

Durante el desarrollo de las llamadas, estas se han ido probando mediante la aplicación Postman, para tratar de minimizar todos los posibles fallos antes de la integración con el frontend. Estas pruebas consisten es realizar la llamada y comprobar que devuelve el resultado esperado.

3.5. Iteración 5

Esta es la iteración con más carga de trabajo de todas. El desarrollo de las pantallas ha seguido el orden equivalente a las llamadas a la API, empezando por el registro y login, y siguiendo por los diferentes modos de juego y otras pantallas, además de implementar la internacionalización para todas ellas, en español y en inglés. Esta parte además de la más larga es en la que más problemas han surgido.

Los primeros problemas que surgieron en esta iteración estaban relacionados con la instalación de las librerías de npm. Al actualizar manualmente una de ellas el proyecto dejó de compilar, impidiendo la continuación del desarrollo hasta encontrar la solución. Es un problema común con npm, las librerías que se guardan en la caché, imposibilitan la instalación del paquete que interesa incluso creando el proyecto desde cero. La solución, una vez conocido el problema, es simple, limpiar la caché de npm y reinstalar los paquetes.

Otro problema recurrente durante el desarrollo ha sido la asincronía de las funciones JavaScript, afectando sobretodo al componte que implementa el temporizador para los juegos, dando un comportamiento errático. Cierto es, que según se gana experiencia con el lenguaje, hace más fácil el control del flujo de las llamadas. En el siguiente fragmento de código se pueden ver los manejadores implementados para controlar el temporizador con precisión. Todas las funciones de este código que empiezan por ”set” son hooks [71] de estado de React.

```
useImperativeHandle(ref, () => ({
    reset: () => {
      setIsActive(false);
      setRound(round => round - 1);
    },
    pause: () => {
      setIsActive(false);
    },
    unpause: () => {
      setIsActive(true);
      setCounter(counter - 1)
    },
    start: () => {
      setIsActive(true);
      setRound(round => round - 1);
    },
    restart: (time) => {
      setIsActive(true);
      setCounter(time)
    },
  }));
```

Mientras se iban desarrollando las pantallas y los juegos, se han ido integrando las correspondientes llamadas a la API. El primer juego implementado por completo es el de pronunciación, ya que está formado por componentes que servirán para casi todos los otros juegos. Una vez afinados, los componentes comunes como el temporizador o la grabadora, se continua con el desarrollo del resto de

juegos.

Al finalizar esta etapa, se presenta el producto para su prueba por parte de los tutores y de esta manera puedan sugerir cambios o encontrar fallos.

3.6. Iteración 6

Esta iteración se dedica a implementar los cambios sugeridos tras las pruebas. La mayoría suponen cambios estéticos, corrección de textos o avisos de que algo ha ido mal. También se encuentran algunos fallos funcionales, sobre todo relacionados con la grabación y reproducción de audios, que en ciertas situaciones no contempladas hacían que la aplicación se detuviera.

También se añaden ciertos cambios en la parte de servidor para gestionar varias listas de palabras del mismo idioma y, también, para parametrizar algunas opciones, como las rutas de guardado de archivos, la dificultad del ASR o el número de rondas para cada modo de juego.

3.7. Iteración 7

En esta última iteración, se procede al despliegue, realización de pruebas y redacción de la memoria.

3.7.1. Pruebas

Se añaden algunos test unitarios en la parte de la aplicación y también algunos test de integración en el servidor. No se ha cubierto todo el código por falta de tiempo, pero quedan hechos como ejemplo para el futuro. También se llevan a cabo pruebas de usabilidad con usuario reales y se recoge la retroalimentación para posibles mejoras. Para ver en detalle las pruebas ir al capítulo **5**.

3.7.2. Despliegue

Se despliega el servidor en una máquina virtual de la UVa siguiendo el manual del apéndice **B.1.2**. También se genera la APK de la aplicación para Android, siguiendo el manual del apéndice **B.2.2**.

3.7.3. Redacción de la memoria

Se comienza la redacción formal de la memoria, en Overleaf [72] utilizando LaTeX, recopilando las notas tomadas durante el desarrollo del proyecto. Estas notas se han ido anotando en un documento word y en comentarios dentro del código. Además, también ha sido de ayuda para la redacción de

este el historial de commits de Bitbucket. También se añaden los distintos diagramas creados con la herramienta Astah.

Capítulo 4

Estado final de la aplicación

4.1. Diagramas de análisis

En este capítulo se van a tratar los temas referentes al diseño de la aplicación abordando temas como las historias de usuario, diagrama de dominio, diagramas de actividad, estructura del backend, estructura del frontend, patrones de diseño y arquitectónicos, modelo de datos, diagrama de paquetes, diagrama de despliegue y estructura del log de actividad.

4.1.1. Historias de usuario

En esta sección de da una descripción detallada de todas las historias de usuario del proyecto. Todas son independientes unas de otras, si bien cabe destacar que todas, excepto ”Registro” y ”Login”, tienen como prerrequisito estar identificado en el sistema. En las siguientes tablas se pueden ver los detalles de las historias de usuario con los siguientes campos:

- **ID**: identificador de la historia de usuario.
- **Nombre**: nombre descriptivo de la historia de usuario.
- **Prioridad**: nivel de prioridad en el desarrollo de la funcionalidad correspondiente a la historia de usuario. Clasificación: baja, media y alta.
- **Riesgo**: impacto sobre el proyecto si se produce un fallo en la historia de usuario. Clasificación: bajo, medio y alto.
- **Descripción**: texto descriptivo de la funcionalidad que ha de implementar la historia de usuario. Con el formato *”Como usuario, quiero...”*
- **Validación**: condiciones ha cumplir para poder considerar la historia de usuario válida.

ID	HU01
Nombre	Registro de un usuario (*Registrar*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder registrarme en el sistema
Validación	– Quiero poder registrarme en el sistema con un nombre de usuario, contraseña, email, lenguaje materno, país de origen. – Quiero que el usuario, contraseña y email sean únicos.

Tabla 4.1: Historia de usuario HU01.

ID	HU02
Nombre	Identificación de un usuario (*Identificar*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder identificarme con mis credenciales en el sistema
Validación	– Quiero poder identificarme en el sistema con un nombre de usuario y una contraseña. – Quiero que mis datos se utilicen para futuras operaciones.

Tabla 4.2: Historia de usuario HU02.

ID	HU03
Nombre	Modificar datos de usuario (*Modificar*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder modificar mis datos en el sistema
Validación	– Quiero cambiar mis datos personales.

Tabla 4.3: Historia de usuario HU03.

ID	HU04
Nombre	Seleccionar idioma de la aplicación (*SeleccionarIdiomaAplicacion*)
Prioridad	Media
Riesgo	Bajo
Descripción	Como usuario, quiero poder utilizar la aplicación en diferentes idiomas
Validación	– Quiero seleccionar el idioma de la interfaz.

Tabla 4.4: Historia de usuario HU04.

ID	HU05
Nombre	Seleccionar idioma de la partida (*SeleccionarIdiomaPartida*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder seleccionar el idioma a practicar en las partidas
Validación	– Quiero seleccionar el idioma de la partida.

Tabla 4.5: Historia de usuario HU05.

ID	HU06
Nombre	Selección de fonemas a practicar (*SeleccionarFonemas*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder elegir el fonema de la lista de palabras que quiero practicar
Validación	– Quiero poder ver la lista de listas de palabras disponibles del idioma seleccionado.

Tabla 4.6: Historia de usuario HU06.

ID	HU07
Nombre	Reproducción de audios (*ReproducirAudios*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder reproducir audios que previamente he grabado
Validación	– Si hay un audio disponible, quiero reproducirlo.

Tabla 4.7: Historia de usuario HU07.

ID	HU08
Nombre	Síntesis de voz (*SintetizarVoz*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero que las palabras disponibles sean leídas en voz alta
Validación	– Si hay una palabra reproducible, quiero oírla.

Tabla 4.8: Historia de usuario HU08.

ID	HU09
Nombre	Reconocimiento de voz (*ReconocerVoz*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero que mi voz sea entendida
Validación	– Si digo una palabra, quiero que sea entendida. – Si digo una palabra y no se entiende, quiero que se me informe de lo que se ha entendido.

Tabla 4.9: Historia de usuario HU09.

ID	HU10
Nombre	Reproducción de vídeo (*ReproducirVídeos*)
Prioridad	Media
Riesgo	Medio
Descripción	Como usuario, quiero poder reproducir un vídeo
Validación	– Si hay un vídeo disponible, quiero reproducirlo. – Si hay un vídeo disponible, quiero pausarlo. – Si hay un vídeo disponible, quiero rebobinarlo o adelantarlo. – Si no hay vídeo disponible, quiero que se me informe.

Tabla 4.10: Historia de usuario HU10.

ID	HU11
Nombre	Visualización del historial de partidas (*VerHistorial*)
Prioridad	Media
Riesgo	Medio
Descripción	Como usuario, quiero ver el historial de mis partidas para la lista seleccionada
Validación	– Si existen partidas, quiero verlas en una lista con detalles. – Si no hay partidas, quiero que se me informe de ello.

Tabla 4.11: Historia de usuario HU11.

ID	HU12
Nombre	Grabación de audios (*GrabarAudios*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder grabar palabras con mi voz
Validación	– Quiero grabar mi voz.

Tabla 4.12: Historia de usuario HU012.

ID	HU13
Nombre	Jugar una partida (*JugarPartida*)
Prioridad	Alta
Riesgo	Alto
Descripción	Como usuario, quiero poder jugar una partida
Validación	– Quiero elegir el modo de juego de la partida. – Quiero iniciar una partida. – Quiero finalizar una partida.

Tabla 4.13: Historia de usuario HU13.

4.1.2. Modelo de dominio

En la figura 4.1 se pueden ver las entidades que componen el proyecto, sus relaciones y multiplicidades, representadas en un diagrama de dominio.

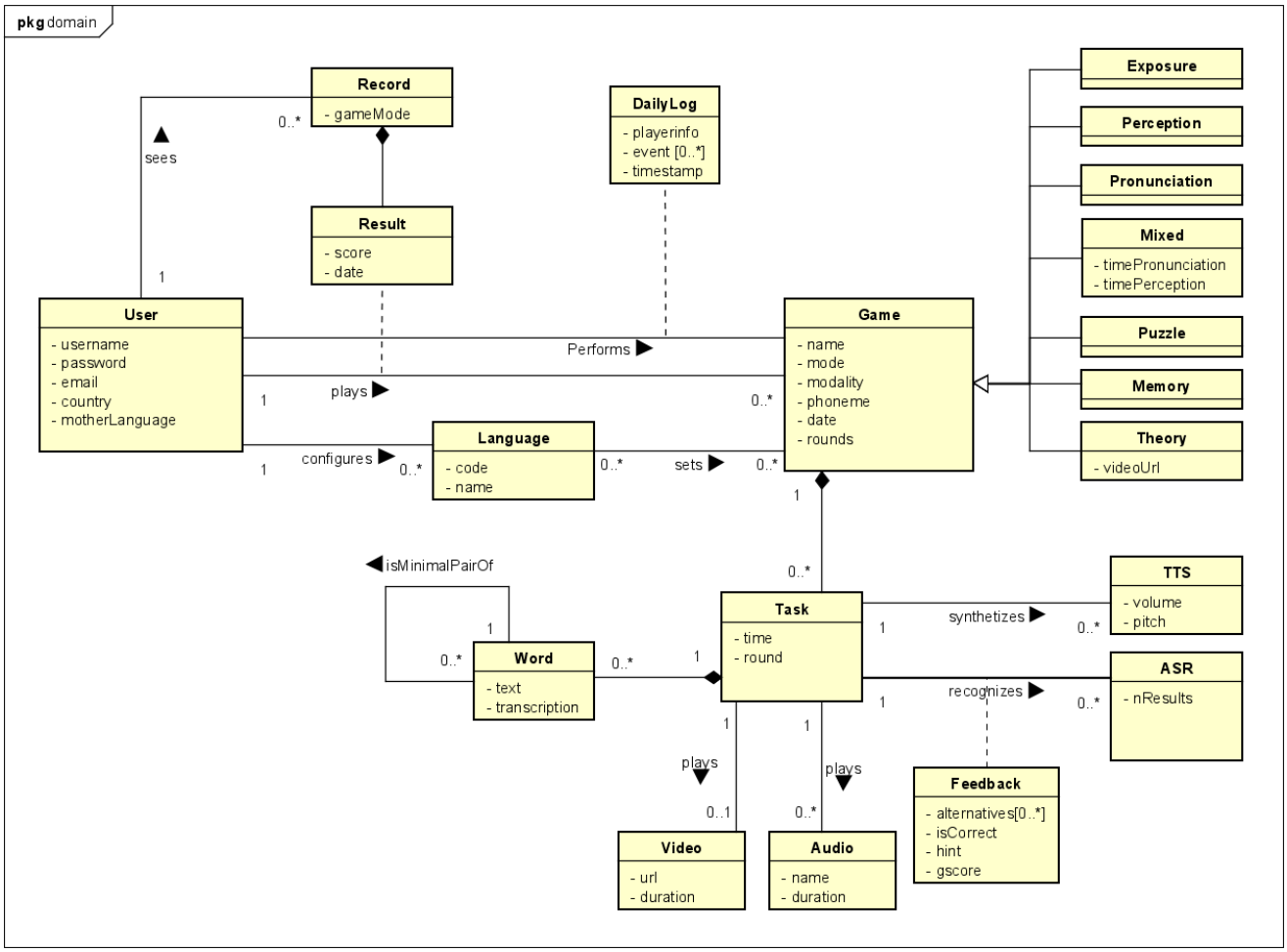

Figura 4.1: Diagrama de dominio.

En las siguientes tablas se describen estas entidades, con su descripción, responsabilidades y atributos:

Word	
Descripción	Modela una palabra para las partidas.
Responsabilidades	Modelar los detalles de una palabra para una partida.
Atributos	– **text**: texto de la palabra. – **transcription**: transcripción fonética de la palabra.

Tabla 4.14: Entidad Word.

User	
Descripción	Entidad que modela los atributos de un usuario del sistema.
Responsabilidades	Modelar un usuario con sus datos de registro para que éste pueda ser identificado en la distintas operaciones dentro del sistema.
Atributos	– **username**: nombre con el que el usuario se identifica en la aplicación. – **password**: contraseña cifrada del usuario. – **email**: correo electrónico del usuario. – **motherLanguage**: lenguaje materno del usuario. – **country**: país de origen del usuario.

Tabla 4.15: Entidad User.

Game	
Descripción	Modela los atributos de una partida.
Responsabilidades	Modelar una partida jugada por un usuario, con los datos de interés.
Atributos	– **type**: modo de juego correspondiente a la partida. – **language**: lenguaje que se está practicando en la partida. – **sound**: fonema o tipo de sonido que se está practicando. – **completed**: señala si se ha llegado al final de la partida. – **score**: puntuación final de la partida. – **date**: fecha y hora en la que se jugó la partida.

Tabla 4.16: Entidad Game.

Record	
Descripción	Modela los atributos del historial de partidas .
Responsabilidades	Modelar el historial de partidas, agrupando los resultados obtenidos por un usuario.
Atributos	– **gameMode**: modo de juego de la partida.

Tabla 4.17: Entidad Record.

Result	
Descripción	Modela los atributos del resultado de una partida.
Responsabilidades	Modelar el resultado obtenido tras jugar una partida.
Atributos	– **score**: puntos obtenidos en la partida. – **date**: fecha y hora de la partida.

Tabla 4.18: Entidad Result.

DailyLog	
Descripción	Modela los atributos del log diario de las partidas.
Responsabilidades	Modelar el log de las partidas jugadas en un mismo día.
Atributos	– **playerInfo**: información relevante del usuario. – **event**: array con los eventos transcurridos en una partida. – **timestamp**: fecha y hora del log.

Tabla 4.19: Entidad DailyLog.

Language	
Descripción	Modela los atributos de un idioma.
Responsabilidades	Modelar los idiomas para que sean seleccionables.
Atributos	– **code**: código del idioma siguiendo la normativa BCP 47 (de RFC 4646) [73]. – **name**: nombre del idioma.

Tabla 4.20: Entidad Language.

Task	
Descripción	Modela las tareas de una partida.
Responsabilidades	Modelar lo que sucede durante las rondas de una partida.
Atributos	– **time**: tiempo de duración de una ronda. – **round**: número de ronda.

Tabla 4.21: Entidad Task.

TTS	
Descripción	Modela el sintetizador de voz.
Responsabilidades	Sintetizar los diferentes textos.
Atributos	– **volume**: volumen de reproducción. – **peach**: velocidad de reproducción.

Tabla 4.22: Entidad TTS.

ASR	
Descripción	Modela el reconocedor de voz.
Responsabilidades	Modelar la configuración para el reconocimiento de voz.
Atributos	– **nResults**: número de resultados máximos esperados del reconocimiento de voz.

Tabla 4.23: Entidad ASR.

Feedback	
Descripción	Modela los atributos de la retroalimentación.
Responsabilidades	Ofrecer al usuario la retroalimentación necesaria tras reconocer una palabra.
Atributos	– **alternatives**: palabras que ha entendido el reconocedor. – **isCorrect**: si es correcta o no la palabra que se ha reconocido. – **hint**: consejo para pronunciar correctamente. – **gscore**: puntuación dada por el reconocedor de Google, que indica la probabilidad de que la palabra reconocida sea la palabra grabada.

Tabla 4.24: Entidad Feedback.

Exposure	
Descripción	Modela el modo de juego *Exposición* .
Responsabilidades	Establecer el modo de juego *Exposición* como dinámica actual de la partida.

Tabla 4.25: Entidad Exposure.

Perception	
Descripción	Modela el modo de juego *Percepción*.
Responsabilidades	Establecer el modo de juego *Percepción* como dinámica actual de la partida.

Tabla 4.26: Entidad Perception.

Pronunciation	
Descripción	Modela el modo de juego *Pronunciación*.
Responsabilidades	Establecer el modo de juego *Pronunciación* como dinámica actual de la partida.

Tabla 4.27: Entidad Pronunciation.

Mixed	
Descripción	Modela el modo de juego *Mixto*.
Responsabilidades	Establecer el modo de juego *Mixto* como dinámica actual de la partida.
Atributos	– **timePronunciation**: tiempo de las rondas que corresponden al modo de juego *Pronunciación*. – **timePerception**: tiempo de las rondas que corresponden al modo de juego *Percepción*.

Tabla 4.28: Entidad Mixed.

Puzzle	
Descripción	Modela el modo de juego *Puzzle*.
Responsabilidades	Establecer el modo de juego *Puzzle* como dinámica actual de la partida.

Tabla 4.29: Entidad Puzzle.

Memory	
Descripción	Modela el modo de juego *Memoria*.
Responsabilidades	Establecer el modo de juego *Memoria* como dinámica actual de la partida.

Tabla 4.30: Entidad Memory.

Theory	
Descripción	Modela el modo de juego *Teoría*.
Responsabilidades	Establecer el modo de juego *Teoría* como dinámica actual de la partida.
Atributos	– **videoUrl**: URL del vídeo correspondiente a la lista de palabras.

Tabla 4.31: Entidad Theory.

Audio	
Descripción	Modela un audio grabado *Teoría*.
Responsabilidades	Modelar los audios grabados por un usuario.
Atributos	– **name**: nombre del archivo. – **duration**: Duración del audio.

Tabla 4.32: Entidad Audio.

Video	
Descripción	Modela un vídeo *Teoría*.
Responsabilidades	Modelar los vídeos utilizado en la aplicación.
Atributos	– **url**: URL del vídeo correspondiente. – **duration**: Duración del vídeo.

Tabla 4.33: Entidad Video.

4.1.2.1. Relaciones entre entidades

De la entidad **User**, salen cuatro asociaciones. Con **Records**, la relación **sees**, hace referencia a un usuario que mira el historial de partidas. La realción **performs** con **Game**, simboliza la actividad de un usuario durante una partida, esta será guardada en el log **DailyLog**. También con **Game**, está la relación **pays**, que significa que un usuario juega una partida y obtiene un resultado **Result**. La última relación de **User**, es con la entidad **Language**, para configurar un idioma en las partidas. A su vez, **Language**, establece el idioma del **Game**.

Los modos de juego **Exposure, Perception, Pronunciation, Mixed, Puzzle, Memory y Theory**, heredan de la entidad **Game**, que está compuesta por entidades **Task**.

La entidad **Task**, se relaciona con otra cinco entidades. Con **TTS**, para representar la síntesis de palabras. Con **ASR**, para el reconocimiento de palabras, que da una retroalimentación, **Feedback**. Con **Audio**, para la reproducción de audios grabado, y con **Video**, también, para reproducir vídeos. **Task**, es una composición de **Word**, las palabras para practicar, y **Word**, forma parte de una par con otras **Word**.

4.1.3. Diagramas de actividad

En esta sección se presentan los diagramas de actividad para las historias de usuario descritas en el apartado 4.1.1. Se detallan las acciones realizadas en el frontend y las consiguientes operaciones llevadas a cabo por el backend. Se muestra tanto la secuencia normal como la secuencia de lo que sucede si se comete algún error en algún caso.

4.1.3.1. HU01 - Registar

La figura 4.2 muestra el diagrama de actividad de la historia de usuario HU01. En el formulario de registro se pide que se introduzca dos veces la contraseña, una comprobación habitual para asegurar que el usuario sabe qué contraseña escribe. Por esto, se comprueba en el frontend si las dos contraseñas introducidas coinciden, si es así se envían los datos al backend, si no, se le informa al usuario de que no coinciden para que pueda rectificar. Una vez llega el formulario, se comprueba todo lo demás, como que el usuario y el correo no existan ya en el sistema y también que el correo tenga un formato válido. Si estas comprobaciones no fallan, se envía un mensaje de éxito para que el frontend se lo muestre al usuario, en caso de error, se haría lo mismo con un mensaje de error especificando cual es el dato incorrecto.

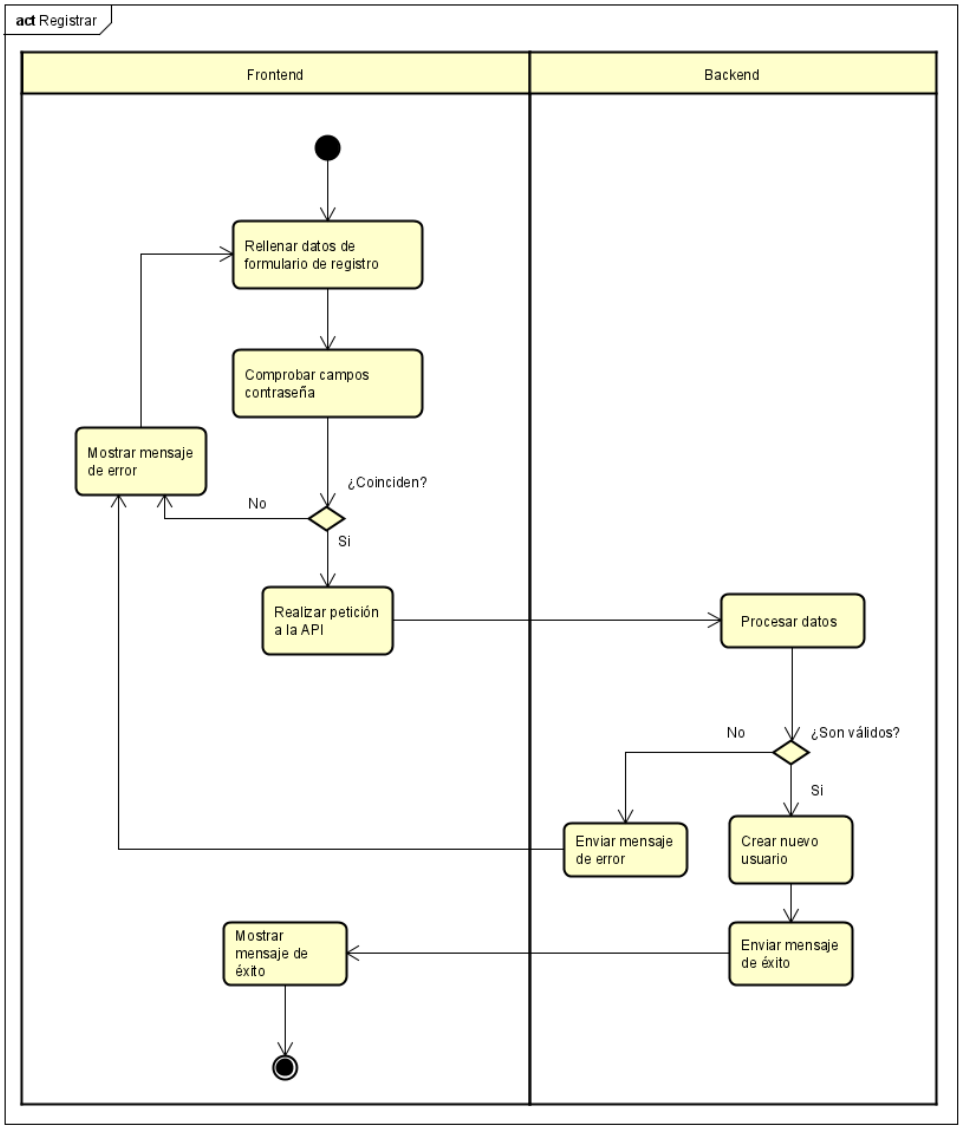

Figura 4.2: Diagrama de actividad de HU01.

4.1.3.2. HU02 - Identificar

La figura 4.3 muestra el diagrama de actividad de la historia de usuario HU02. Una vez el usuario introduce su nombre de usuario y contraseña, estos datos se mandan al backend para realizar la búsqueda en la base de datos. Si los datos son correctos, se guarda toda la información necesaria en el frontend y pasará a la siguiente pantalla, si no, se mostrará un mensaje de error neutro, sin información de lo que ha fallado, esto se hace por motivos de seguridad

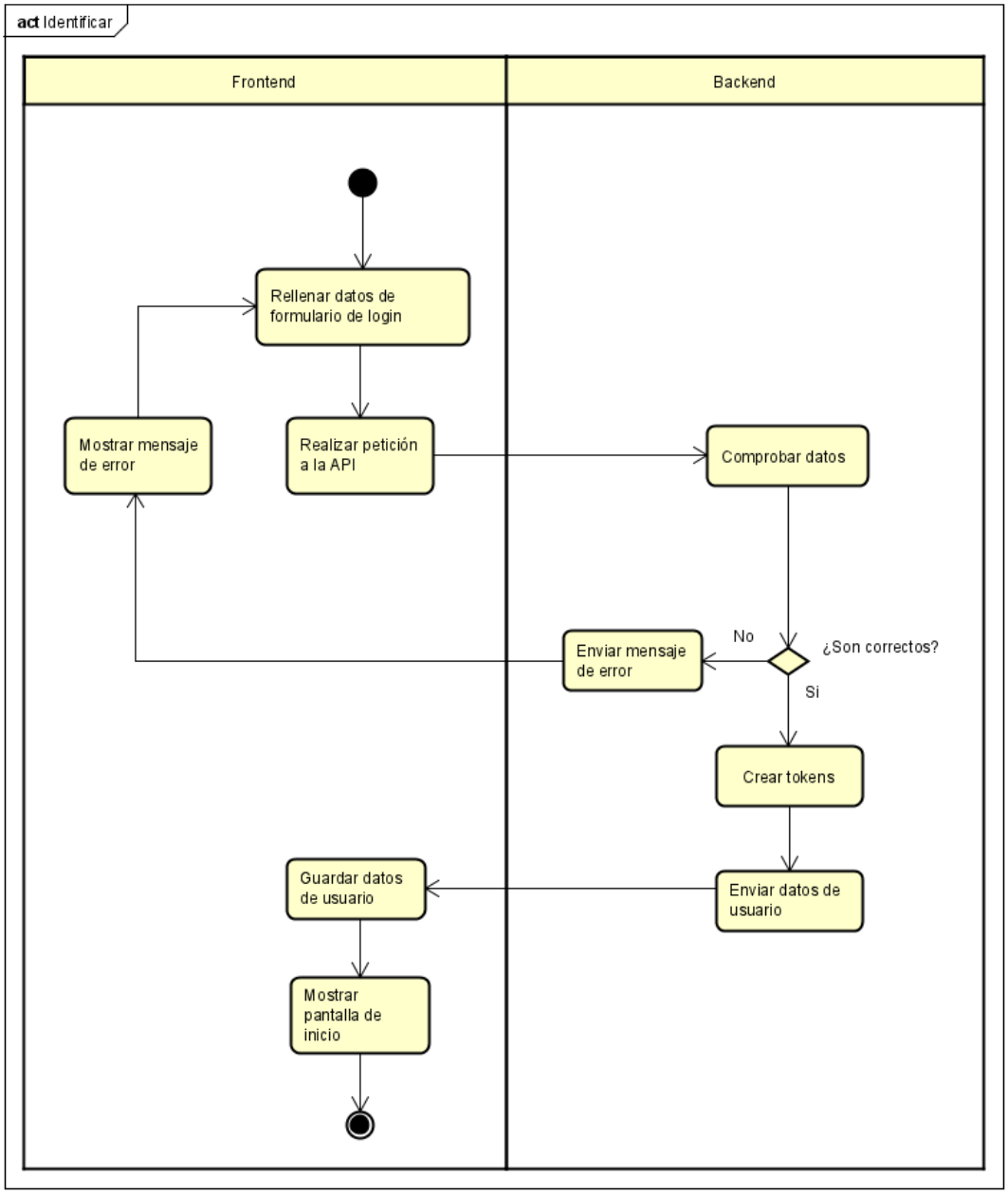

Figura 4.3: Diagrama de actividad de HU02.

4.1.3.3. HU03 - Modificar

La figura 4.4 muestra el diagrama de actividad de la historia de usuario HU03. Este caso es similar al de *Registrar*, lo único que esta vez las comprobaciones solo se realizan con los campos habilitados para edición, que son todos menos usuario y email.

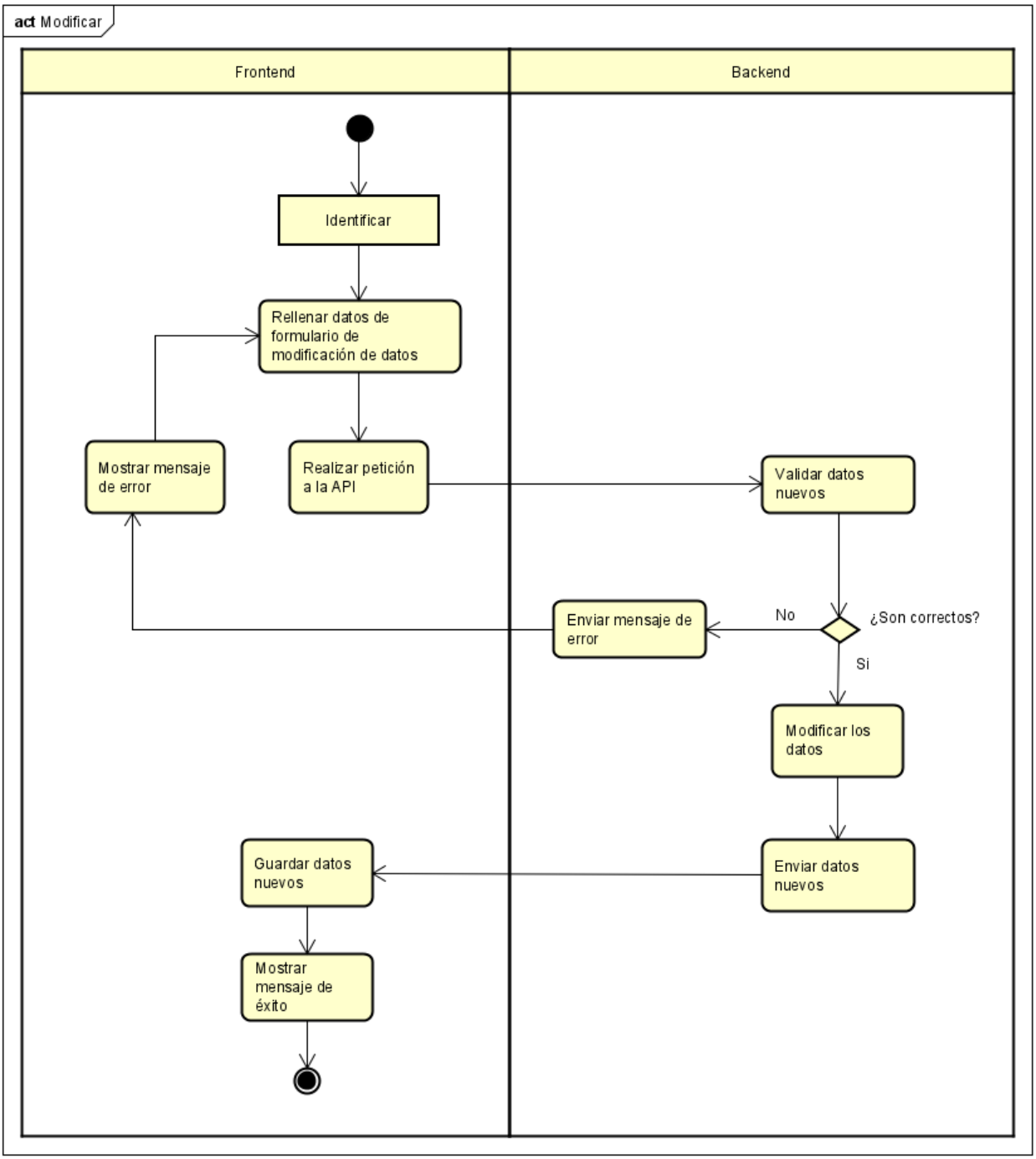

Figura 4.4: Diagrama de actividad de HU03.

4.1.3.4. HU04 - SeleccionarIdiomaAplicación

La figura 4.5 muestra el diagrama de actividad de la historia de usuario HU04. El usuario elige un idioma de entre los disponibles y al seleccionarlo este se queda guardado para que en futuras ocasiones que se acceda esté preseleccionado el que ha elegido.

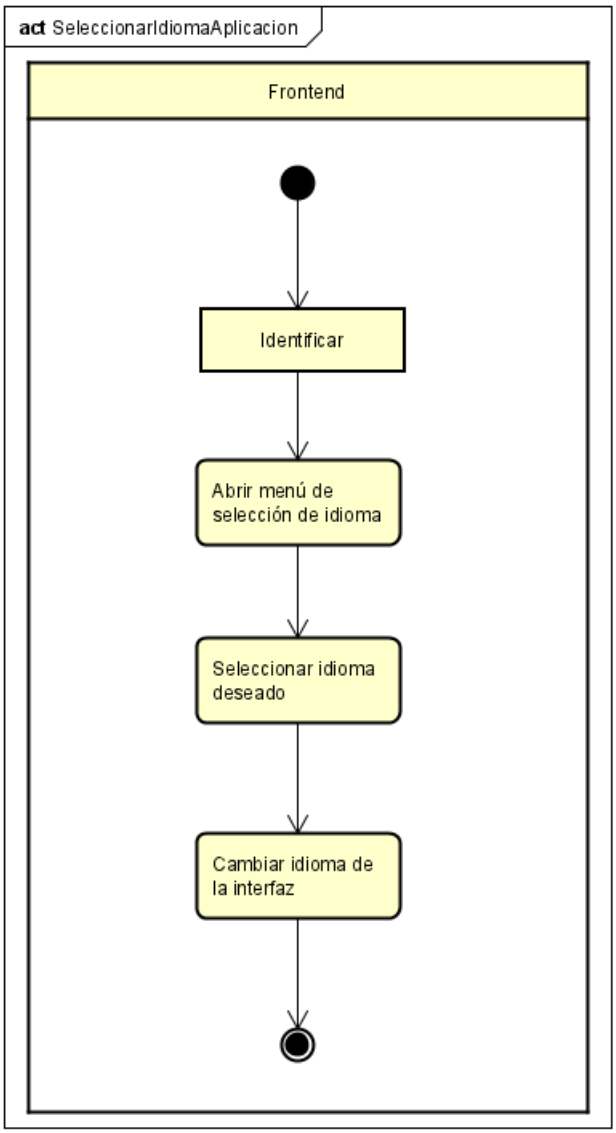

Figura 4.5: Diagrama de actividad de HU04.

4.1.3.5. HU05 - SeleccionarIdiomaPartidas

La figura 4.6 muestra el diagrama de actividad de la historia de usuario HU05. Una vez el usuario está identificado en el sistema se le muestra la pantalla de selección de idioma que quiere practicar. Al seleccionar el idioma deseado se le pide al backend la lista de palabras disponibles para este.

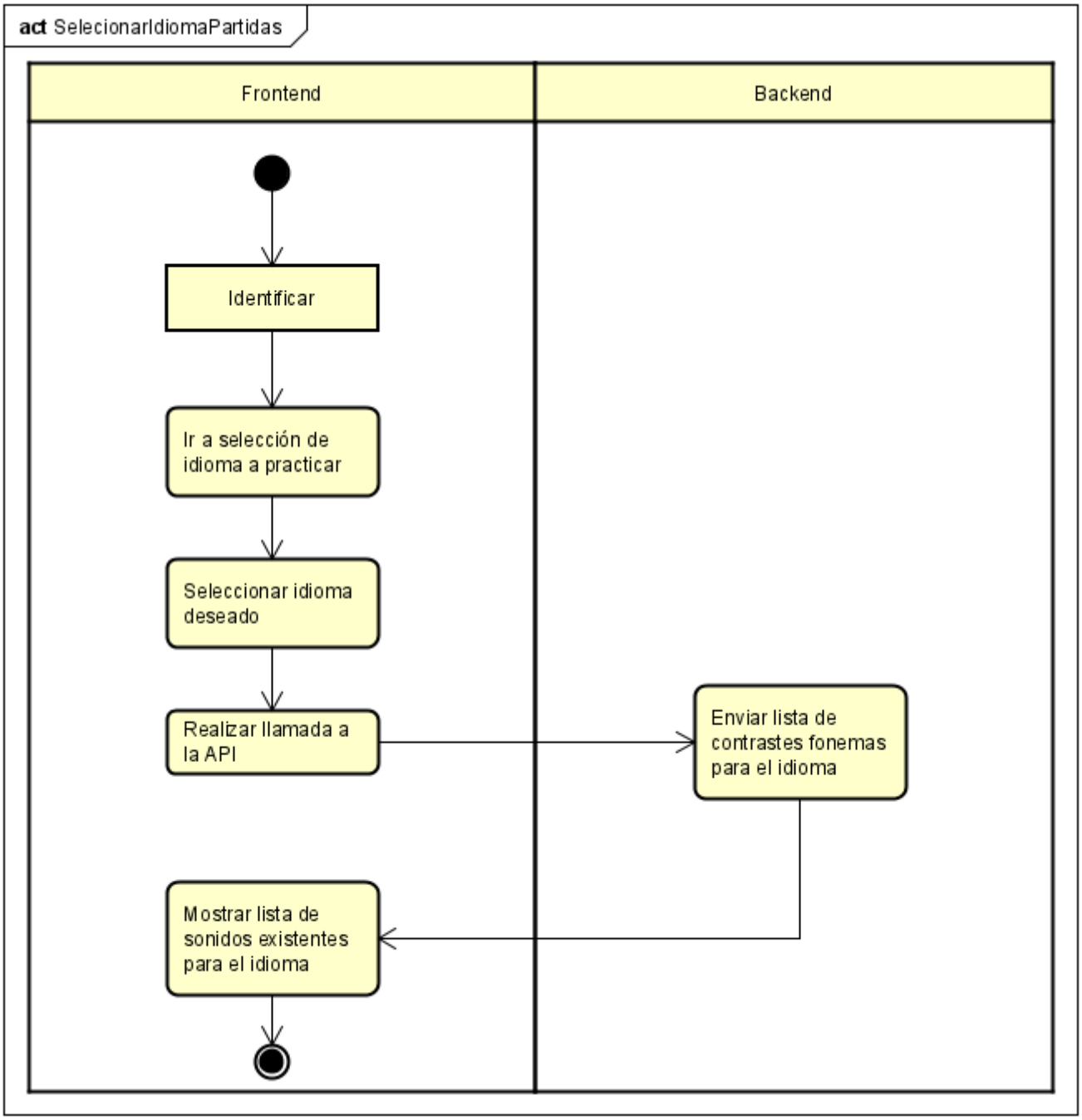

Figura 4.6: Diagrama de actividad de HU05.

4.1.3.6. HU06 - SeleccionarFonemas

La figura 4.7 muestra el diagrama de actividad de la historia de usuario HU06. Tras elegir el idioma el usuario selecciona un fonema de la lista, una vez confirma se guardan en el frontend tanto el idioma como la lista elegida.

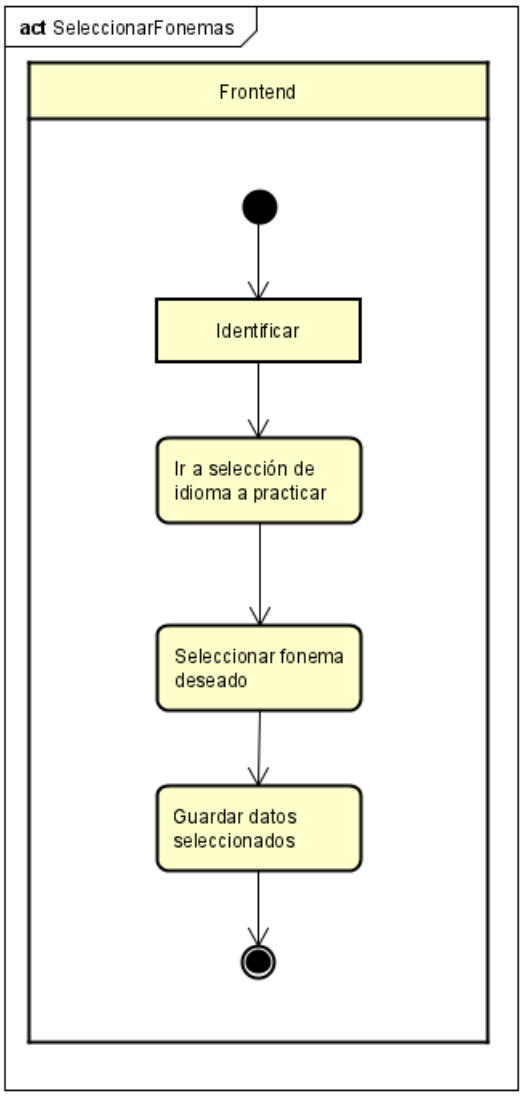

Figura 4.7: Diagrama de actividad de HU06.

4.1.3.7. HU07 - ReproducirAudios

La figura 4.8 muestra el diagrama de actividad de la historia de usuario HU07. Antes de poder reproducir un audio tiene que haberse grabado, si es así, basta con pedir al sistema que lo reproduzca.

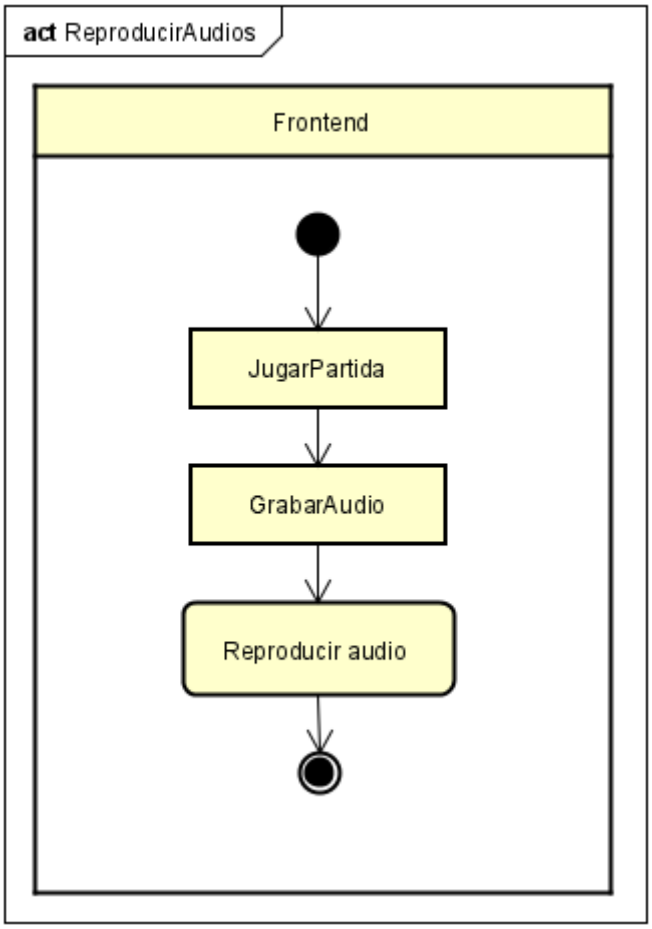

Figura 4.8: Diagrama de actividad de HU07.

4.1.3.8. HU08 - SintetizarVoz

La figura 4.9 muestra el diagrama de actividad de la historia de usuario HU08. Se le pasa una palabra al sintetizador y este la reproduce.

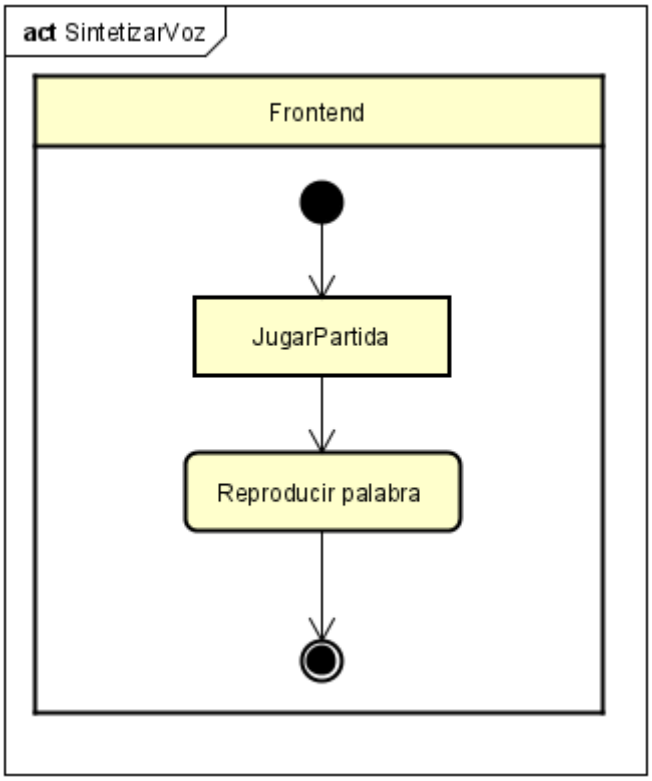

Figura 4.9: Diagrama de actividad de HU08.

4.1.3.9. HU09 - ReconocerVoz

La figura 4.10 muestra el diagrama de actividad de la historia de usuario HU09. Previamente se tiene que haber grabado un audio y enviado al backend. Lo primero se envía el audio para que lo procese la API STT de Google y esta devuelva los datos correspondientes. De estos datos, se extraen los interesantes para la operación en cuestión y se envían al frontend para mostrárselos al usuario.

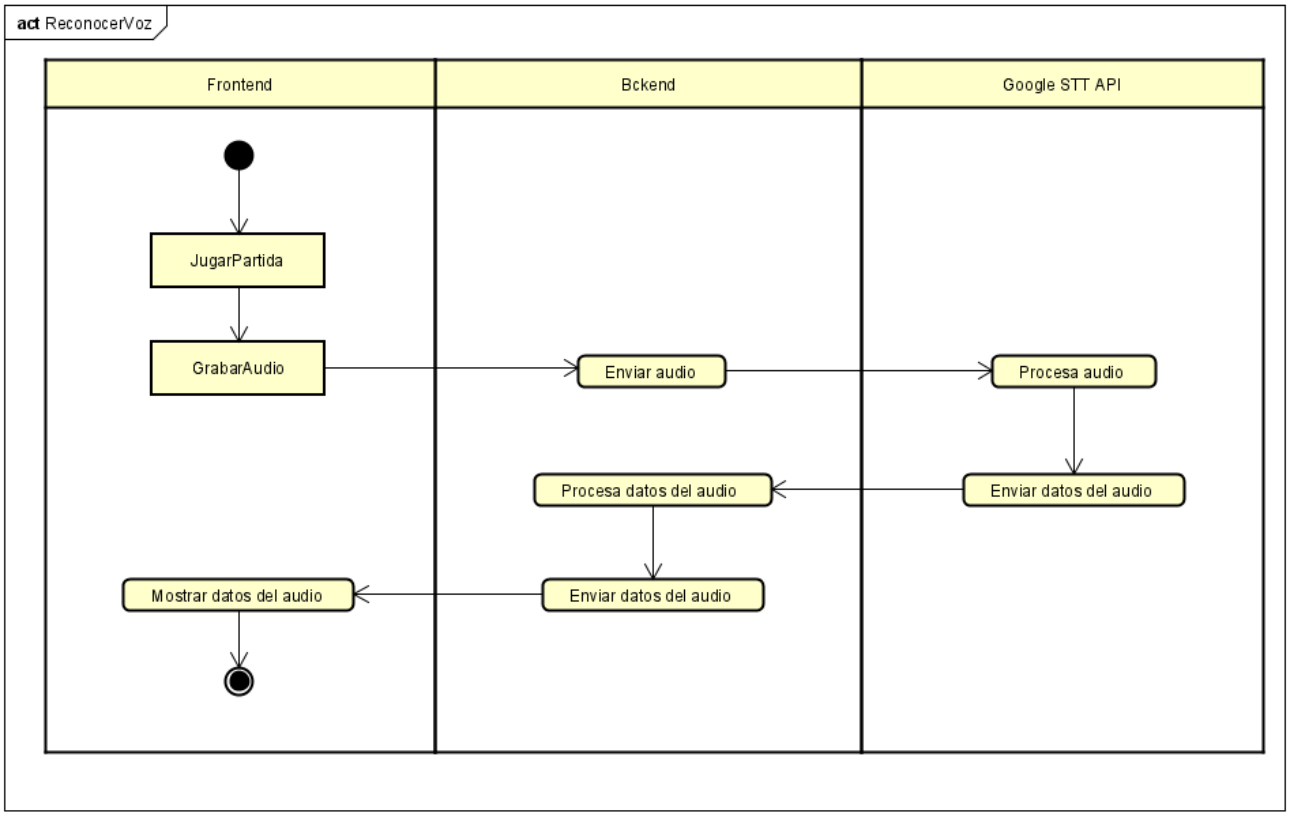

Figura 4.10: Diagrama de actividad de HU09.

4.1.3.10. HU10 - ReproducirVideos

La figura 4.11 muestra el diagrama de actividad de la historia de usuario HU10. Se solicita al backend la URL del vídeo que se va a cargar, si la lista que se está practicando tiene un vídeo asociado, se devuelve la URL, y al recibirla se carga y se comienza a reproducir el vídeo. Si no existe dicha URL, se envía un mensaje de error para poder mostrárselo al usuario.

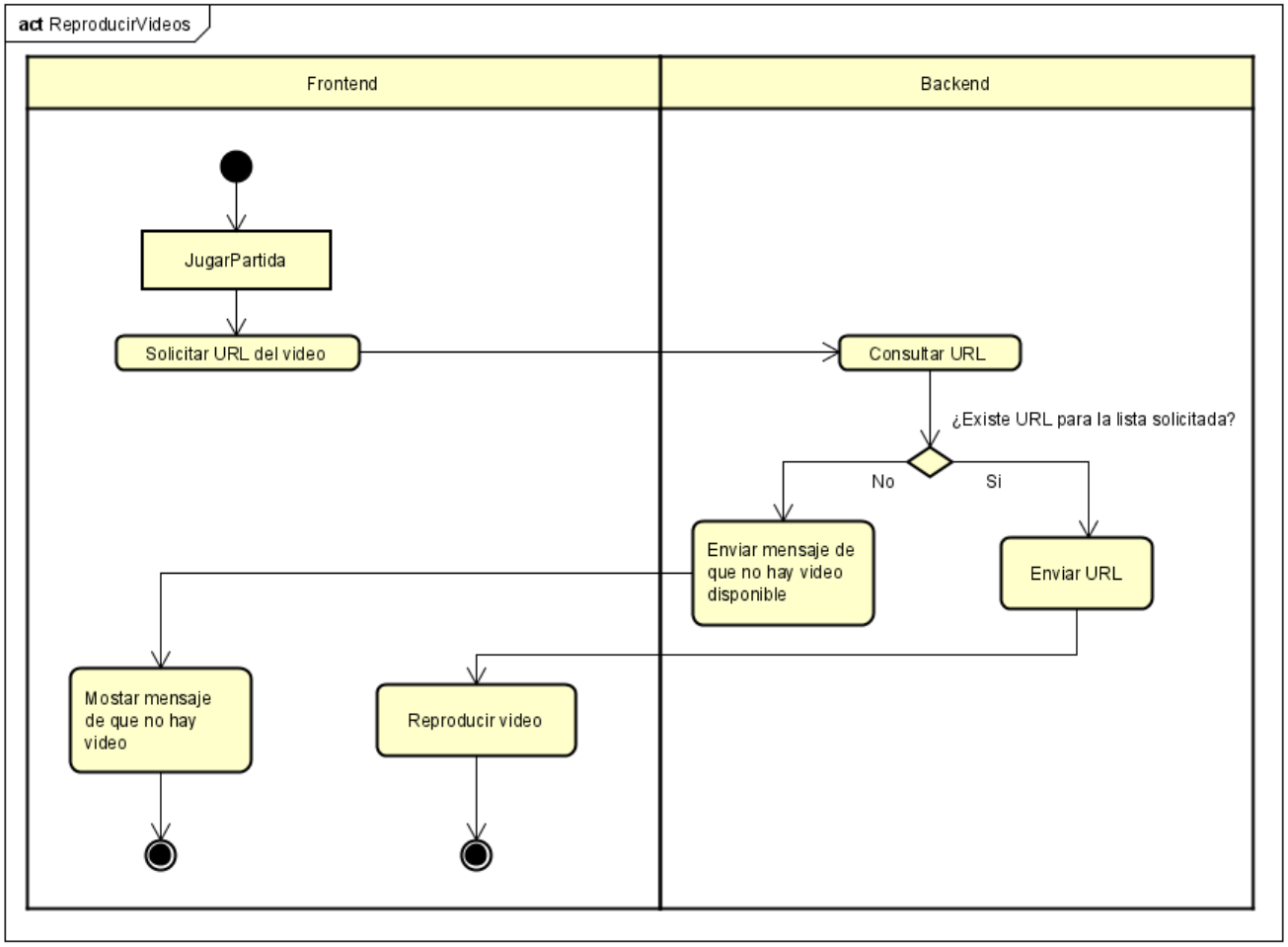

Figura 4.11: Diagrama de actividad de HU10.

4.1.3.11. HU11 - VerHistorial

La figura 4.12 muestra el diagrama de actividad de la historia de usuario HU11. Se le pide al sistema que muestre el historial, este comprueba qué lista se está practicando y si existen partidas guardadas devuelve la información de cada una de ellas. Si no hay partidas, se envía un mensaje de error para informar al usuario.

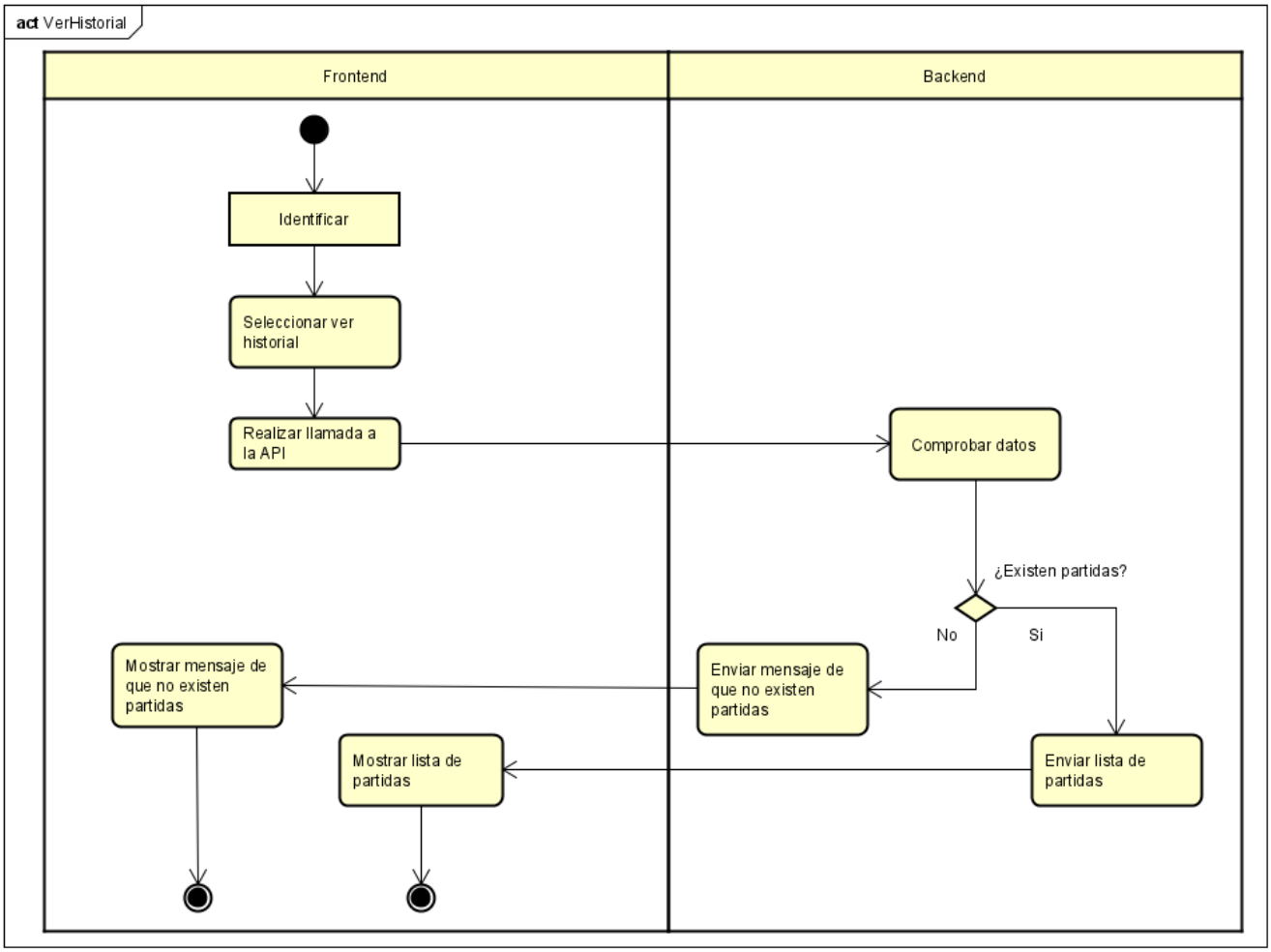

Figura 4.12: Diagrama de actividad de HU11.

4.1.3.12. HU12 - GrabarAudios

La figura 4.13 muestra el diagrama de actividad de la historia de usuario HU12. El usuario graba un audio, este se guarda temporalmente en el frontend y también se envía al backend donde se guardará de forma permanente. Después de esto en algún caso puede darse que se apliquen o la historia de usuario 4.8 o 4.10.

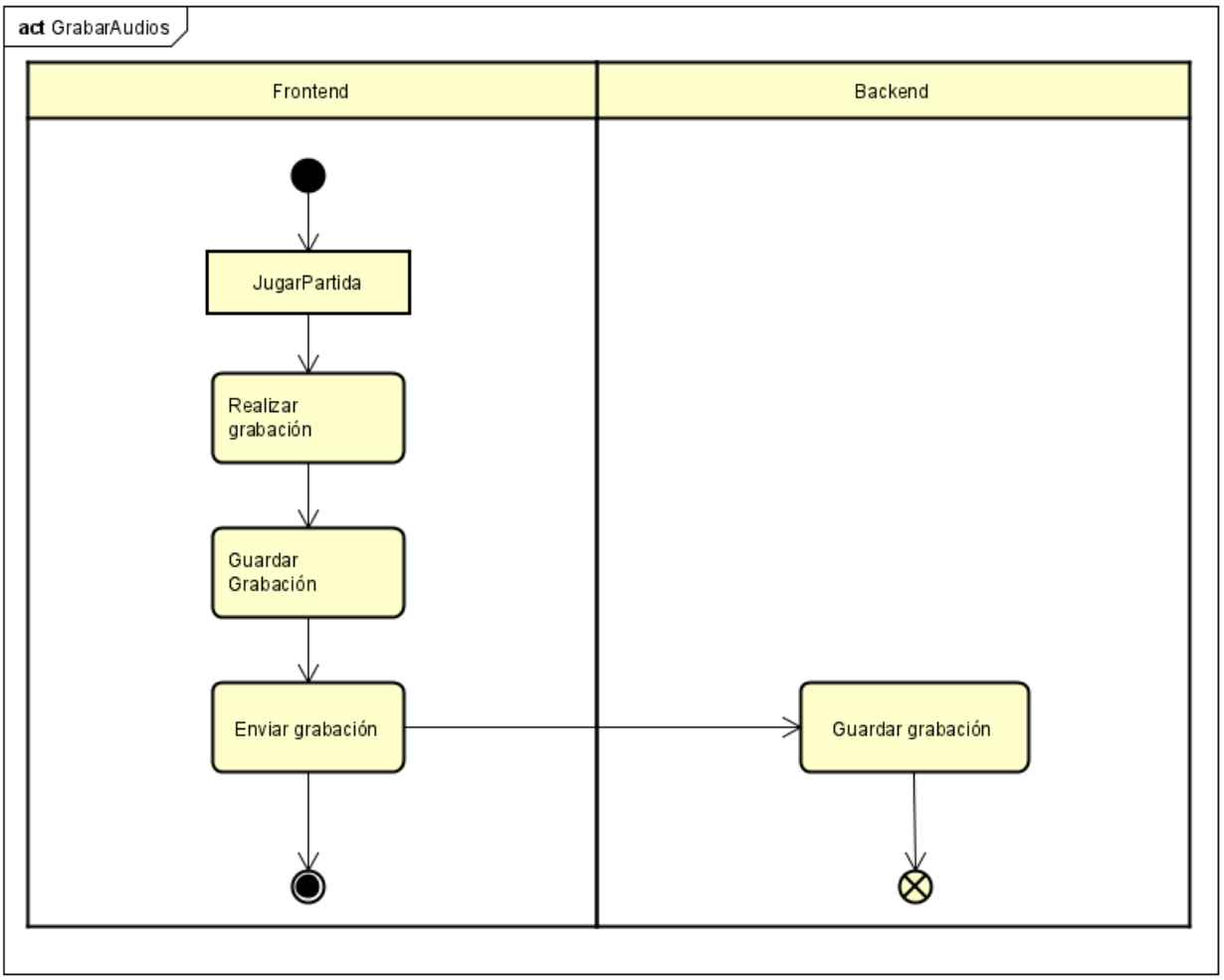

Figura 4.13: Diagrama de actividad de HU12.

4.1.3.13. HU13 - JugarPartida

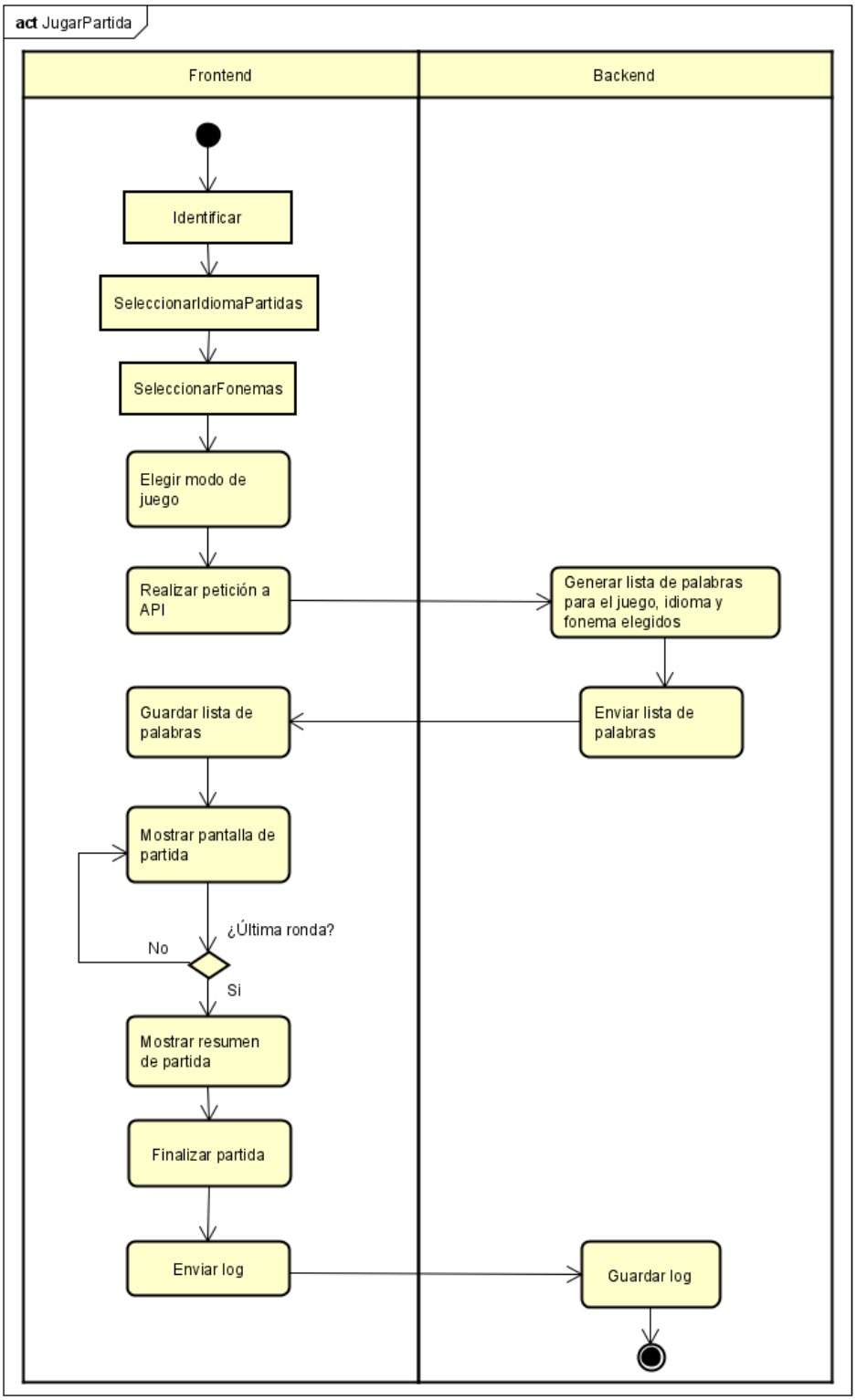

Figura 4.14: Diagrama de actividad de HU13.

La figura 4.14 muestra el diagrama de actividad de la historia de usuario HU13. Después de que el usuario haya elegido un idioma y un fonema a practicar, debe elegir el modo de juego para hacerlo.

Cuando selecciona el modo de juego, se envía la petición al backend para que genere la lista de pares de palabras correspondiente y la devuelva. Luego, estas palabras se irán cargando en cada pantalla de la partida hasta que no queden más rondas. Una vez acabada la partida, se le muestra al usuario un resumen de su actuación en ésta. Cuando el usuario ha acabado de ver el resumen y procede a finalizar la partida, se envía un log de la actividad y otros parámetros al backend.

4.2. Modelado de la interacción de los modos de juego

En esta sección se muestra mediante diagramas de actividad, las interacciones del usuario y el sistema para cada modo de juego. Para ver el diseño de las pantallas mirar la sección 3.3.

4.2.1. Exposición

En este modo el sintetizador empieza reproduciendo las palabras cinco veces alternativamente. El usuario puede volver a escuchar las palabras o grabarlas con su voz. Una vez grabadas puede reproducir los audios grabados. Esto puede hacerlo varias veces, hasta que el usuario quiera, luego puede pasar de ronda, en caso de que queden rondas, si no, se muestra la pantalla de fin de partida. Este modo es de práctica, no tiene puntuación.

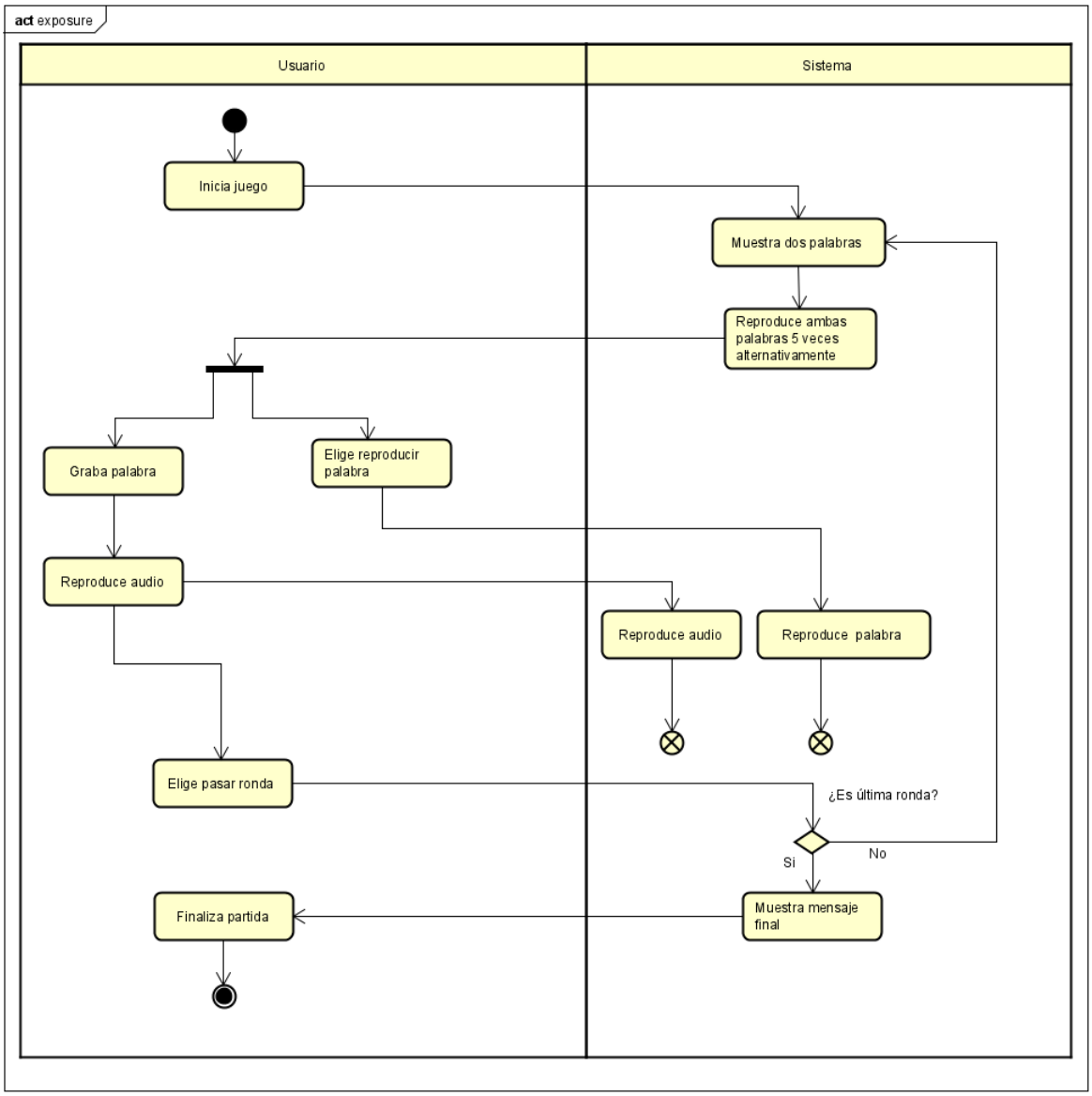

Figura 4.15: Diagrama de actividad modo de juego Exposición.

4.2.2. Percepción

Este modo consiste en que el sistema muestra dos palabras y reproduce una de ellas. El usuario tiene que elegir la palabra correcta, y puede volver a escucharla pulsando el botón de reproducir. Si la palabra elegida es correcta, sumara puntos y pasa de pantalla, si no, el sistema muestra que la palabra elegida es incorrecta y pasa a la siguiente pantalla. Si es la ronda final pasará a la pantalla de fin de partida.

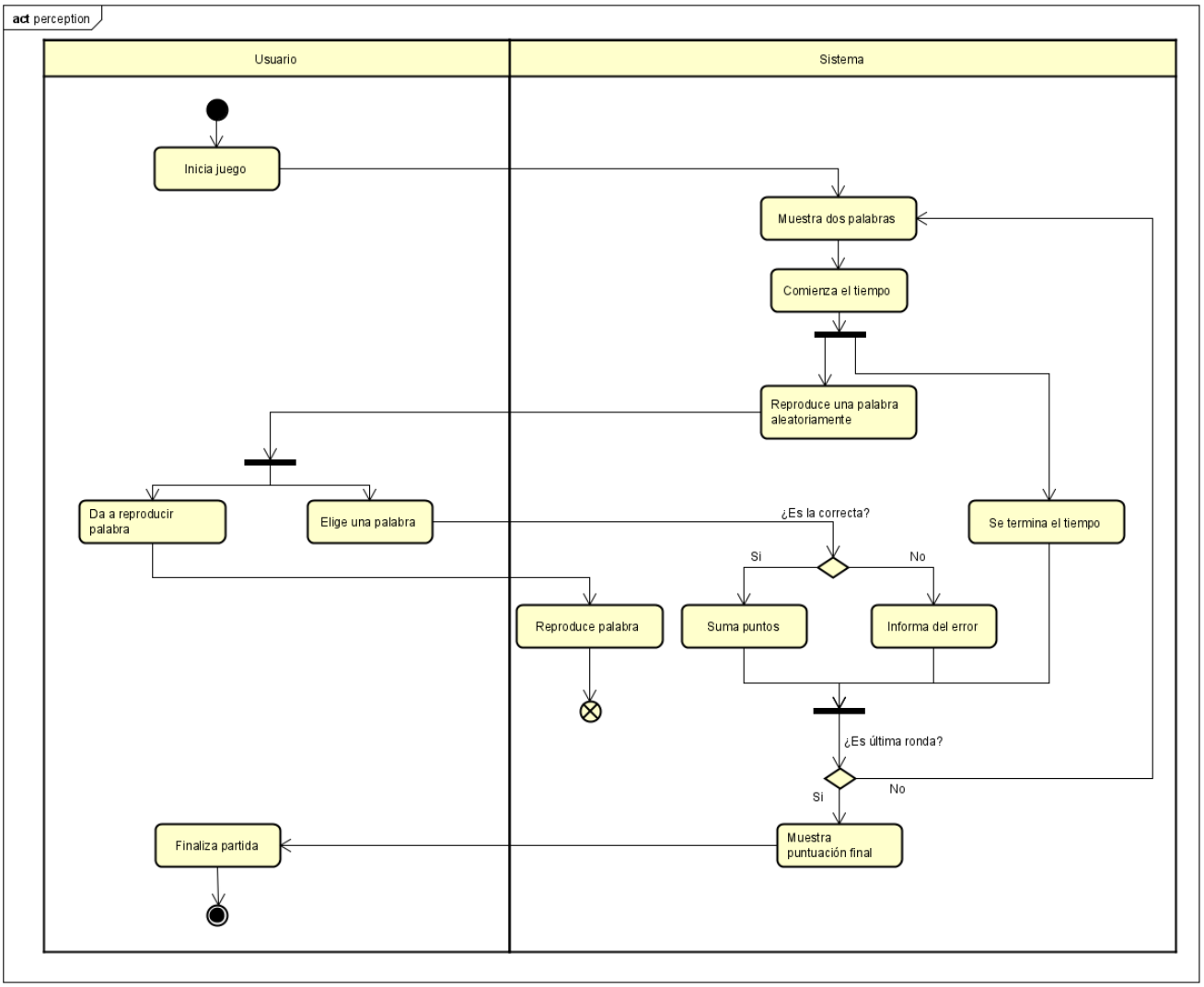

Figura 4.16: Diagrama de actividad modo de juego Percepción.

4.2.3. Pronunciación

En este modo de juego se muestran dos palabras, el usuario puede reproducirlas. Cuando graba una de ella si es correcta se bloquea, si es incorrecta puede seguir probando hasta que se le acaben los intentos, una vez se queda sin intentos se bloquea. Lo mismo para la otra palabra. Una vez las dos palabras están bloqueadas o se acaba el tiempo pasa a la siguiente ronda.

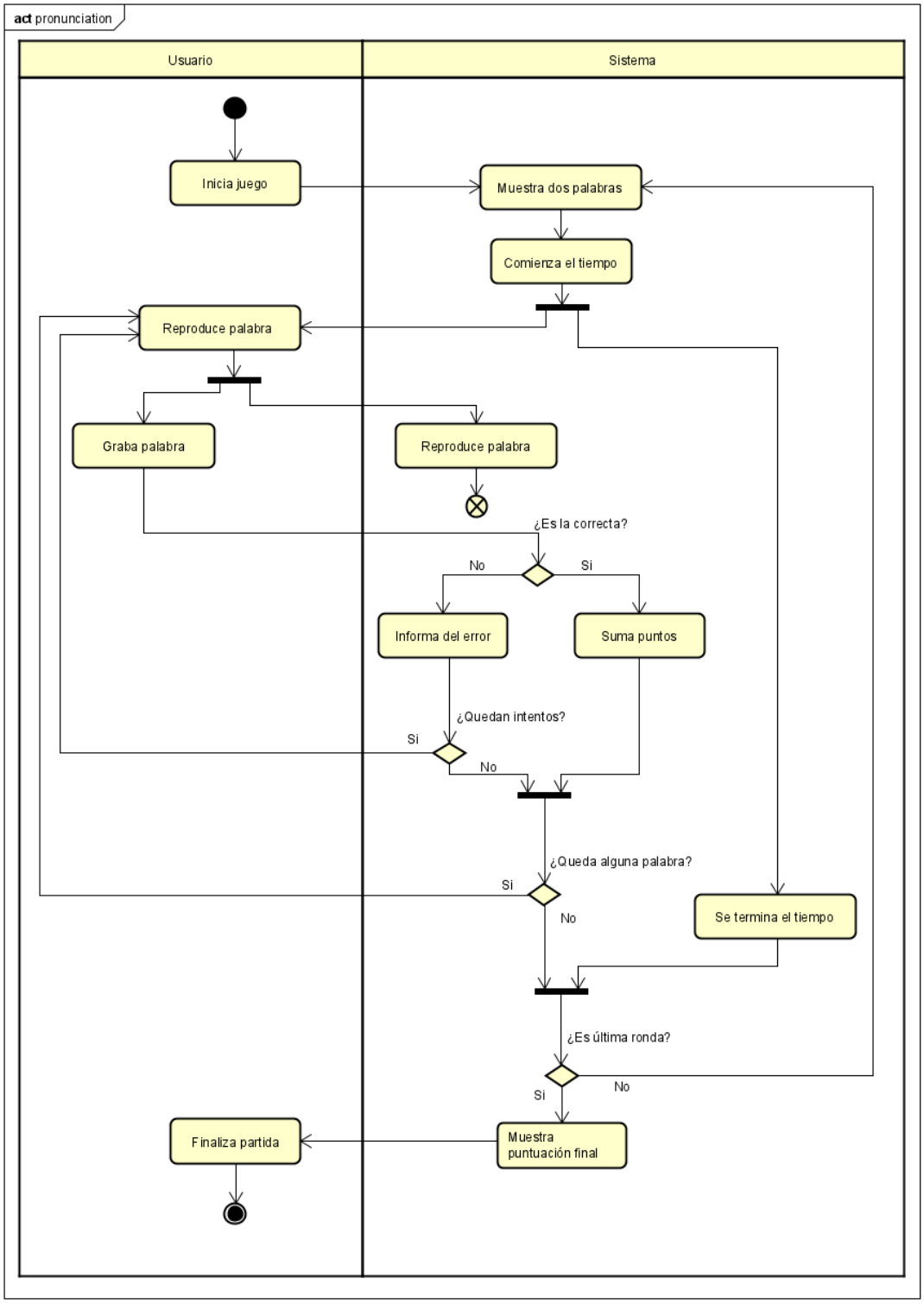

Figura 4.17: Diagrama de actividad modo de juego Pronunciación.

4.2.4. Mixto

Este modo de juego alterna una pantalla del modo Pronunciación (apartado 4.2.3) y una del modo Percepción (apartado 4.2.2).

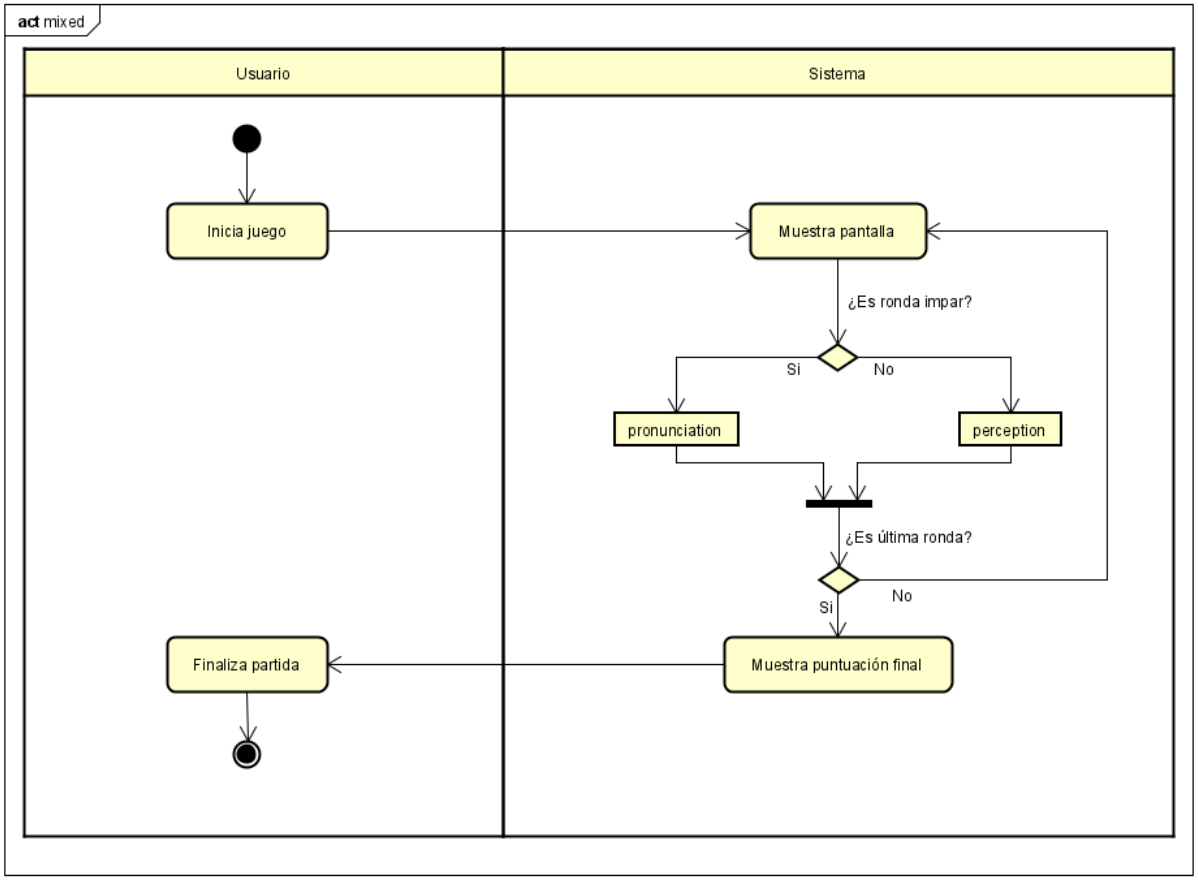

Figura 4.18: Diagrama de actividad modo de juego Mixto.

4.2.5. Memoria

En este modo se presentan seis palabras y el sistema reproduce tres de ellas en un orden concreto. El usuario debe seleccionarlas en orden. También puede elegir volver a reproducirlas. Una vez se acaba el tiempo o elige las tres palabras, el sistema le comunica cuales son correctas y cuales no, y muestra el botón para pasar de ronda. Al final de cada ronda se suman puntos por cada palabra correcta.

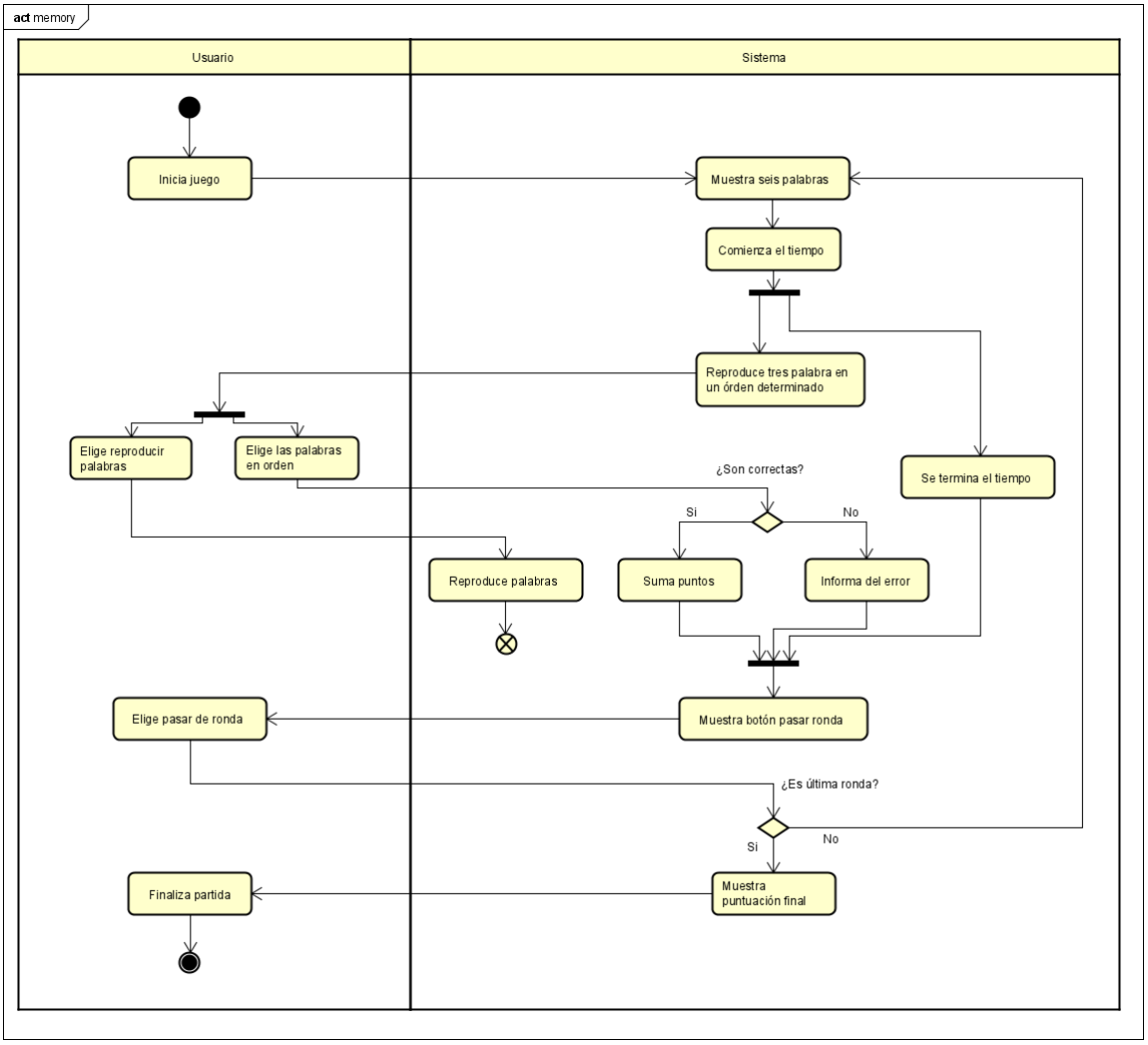

Figura 4.19: Diagrama de actividad modo de juego Memoria.

4.2.6. Puzzle

En este modo se muestran seis palabras y un fonema. El usuario debe seleccionar las palabras que contienen este fonema. Una vez se acaba el tiempo o elige corregir, el sistema le comunica cuales son correctas y cuales no, y muestra el botón para pasar de ronda. Al final de cada ronda se suman puntos por cada palabra correcta.

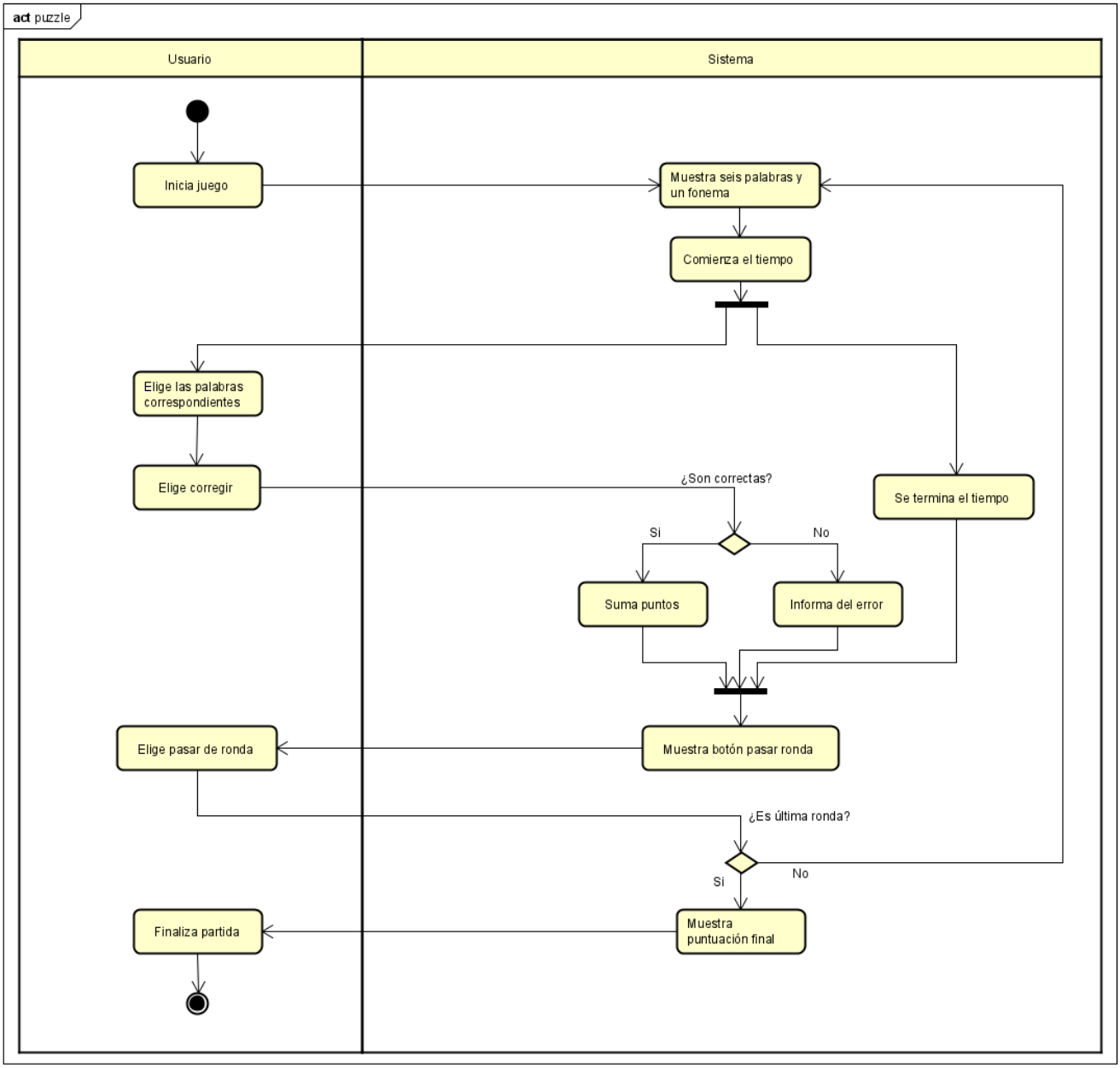

Figura 4.20: Diagrama de actividad modo de juego Puzzle.

4.2.7. Teoría

El modo teoría consiste en un vídeo en el que se explica como pronunciar el par de fonemas que se este practicando.

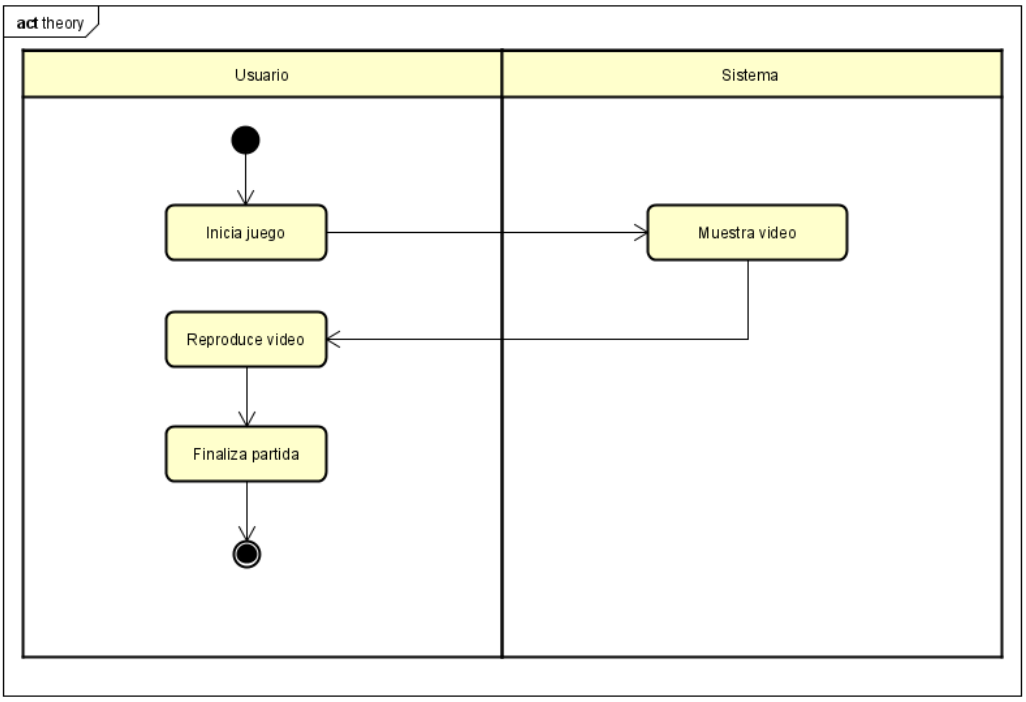

Figura 4.21: Diagrama de actividad modo de juego Teoría.

4.3. Estructura del proyecto

4.3.1. Estructura de *backend*

En la figura 4.22 se puede ver la estructura que se ha utilizado en este proyecto. Cabe señalar, que los proyectos Node con Express.js no tienen una arquitectura definida, éstos se puede estructurar libremente. En este caso, la solución es una adaptación de uno de los muchos ejemplos que existen [74].

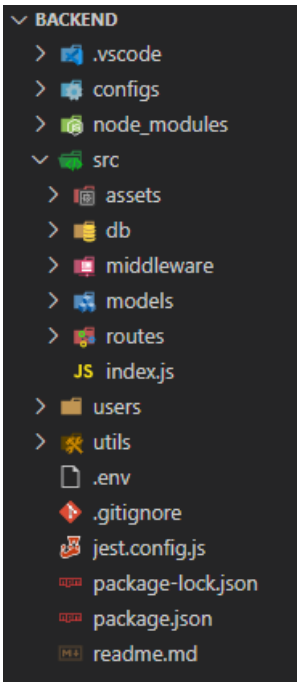

Figura 4.22: Estructura un proyecto Node.

A continuación, se explica la función que desempeñan cada uno de los directorios y su contenido.

- **.vscode**: esta carpeta no pertenece a la estructura del proyecto Node.js en sí, si no que es donde se guarda la configuración del IDE Visual Studio Code.
- **configs**: directorio donde se guarda la configuración por defecto de la aplicación. Entre otras cosas se pueden configurar la duración de las partidas, la dificultad, número de rondas...
- **node_modules**: en esta carpeta se encuentran todos los paquetes npm instalados. Se genera al ejecutar el comando *npm i*.
- **src**: contiene la lógica de la aplicación y esta compuesta por otras carpetas:
  - **assets**: aquí se guardan las listas de palabras para los juegos. Para añadir una lista de palabras basta con meterla en esta carpeta.
  - **db**: la configuración de la base de datos MongoDB. Desde aquí, entre otras cosas se puede cambiar la ruta a la que apuntan las consultas.

  - **middleware**: código necesario para comprobar que las llamadas a la API llegan con un token válido de identificación.
  - **models**: modelos de las entidades para la base de datos. Para modelar éstos se ha utilizado la herramienta Mongoose [55].
  - **routes**: contiene la lógica de las diferentes llamadas a la API que se llamarán desde la parte frontend.
  - **index.js**: punto de entrada de la aplicación. Es el archivo que debemos ejecutar para que el servidor arranque. Aquí se importan varios módulos necesario, pero lo más importante es que es donde se configura el puerto de acceso al servidor.

- **users**: en este directorio se crea una carpeta por usuario registrado. En estas carpetas se guarda, para cada usuario, los logs de sus partidas y lo audios subidos.

- **utils**: diferentes archivos utilizados dentro de la aplicación, los más importantes: *videos_ref.json*, que contiene las URLs de los vídeos para el modo de juego *teoría*; y, el *key.json* que contiene las credenciales para acceder a la API de Google.

- **.env**: contiene las variables de entorno, entre ellas, el puerto por defecto y la ruta de la base de datos.

- **.gitignore**: archivo de git con la lista de los archivos que no se quiere que se tengan en cuenta para el control de versiones.

- **jest.config.js**: configuración para los test de integración.

- **package.json**: contiene la lista de los paquetes npm utilizados en el proyecto y los scripts de arranque, parada, despliegue... Este archivo es el que indica a npm que módulos son necesarios a la hora de instalar el proyecto.

- **package-lock.json**: también contiene la información de los paquetes npm pero más detallada. Este se genera automáticamente una vez instalamos el proyecto.

- **readme.md**: instrucciones de instalación del proyecto en un entorno de desarrollo.

4.3.2. Estructura del *frontend*

Al igual que pasa con el servidor, las aplicaciones desarrolladas con React-Native y Expo, no tienen una arquitectura definida. Para este proyecto, se ha utilizado una estructura similar a la que se puede encontrar en el Trabajo de Fin de Máster de Blanca Lendoiro Valle [75]. En la figura 4.23 se puede ver esta estructura:

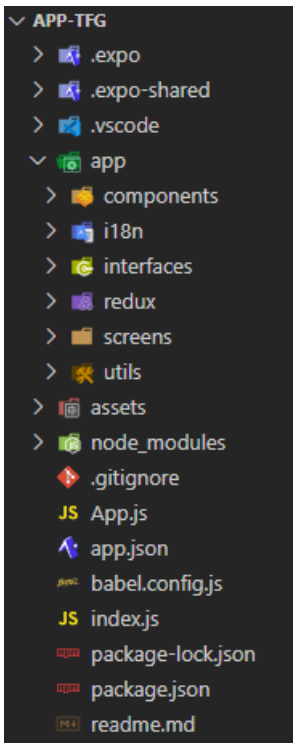

Figura 4.23: Estructura un proyecto Expo React-Native.

Como se puede ver, hay varios archivos iguales a los del servidor, esto se debe a qué ambas partes del proyecto están desarrolladas con Node. Estos archivos comunes no se describirán de nuevo, a no ser que haya algo que destacar:

- **.expo**: archivos internos de configuración de Expo.
- **.exposhared**: más archivos internos de Expo.
- **app**: contiene la lógica de la aplicación y se compone de los siguientes subdirectorios:
  - **components**: componentes que se usan en las pantallas de la aplicación. Estos componentes son genéricos para su uso en distintos puntos. Por ejemplo, aquí se encuentra el temporizador o el botón para grabar audios.
  - **i18n**: los archivos que permiten internacionalización. En este caso están disponibles los idiomas español e inglés, y aquí se encuentran las cadenas de texto y los formatos de fecha para cada idioma.
  - **redux**: en esta carpeta se encuentran los *actions* y los *reducers* para manejar el estado interno de la aplicación. Pero lo más importante, contiene las llamadas a la API. Muchas de las respuestas que se obtienen de la API se guardan directamente en redux, para poder acceder a la información en cualquier momento y desde cualquier punto de la aplicación.
  - **screens**: código de las distintas pantallas de la aplicación. Aquí se encuentra casi toda la lógica de la aplicación.
  - **utils**: archivos con variables de configuración y funciones que se usan en distintos puntos de la aplicación.

- **assets**: archivos de imagen y sonido utilizados.
- **App.js**: punto de entrada de la aplicación. En este archivo se definen entre otras cosas, los idiomas disponibles para la internacionalización y la pantalla inicial de la aplicación.
- **app.json**: configuración para generar las build de la aplicación. Aquí podemos establecer distintos parámetros, los más interesantes son: el nombre de la app, la versión y el icono.
- **babel.config.js**: configuración de *Babel* [76], en principio no debemos tocar nada de este archivo.

4.3.3. Arquitectura y patrones de diseño

En este apartado se comentaran los patrones utilizados para el desarrollo del proyecto tanto en la parte del frontend como en el backend.

4.3.3.1. Arquitectura de capas

Este patrón arquitectónico tiene el objetivo de simplificar sistemas complejos mediante la separación de sus principales componentes: lógica de negocio, interfaz de usuario y persistencia de datos. En este caso, de la capa de presentación se encarga el frontend con React-Native; la capa de negocio, el backend, con Express.js; y, la de persistencia, los modelos de MongoDB. En el apartado 4.4.2 se puede ver claramente la separación por capas.

4.3.3.2. Módulo

Este es un patrón de diseño creacional, que se utiliza cada vez que se declara un objeto JavaScript. Este aplica tanto en backend como en frontend, y se usa en la mayor parte del código ya que los objetos, son la forma más sencilla de manejar los datos en JavaScript. Por ejemplo, en *frontend* uno de los usos, es para declarar los estilos CSS de los componentes. Y en *backend*, para encapsular los datos obtenidos de las consultas a la base de datos para luego ser enviados en las llamadas.

4.3.3.3. Singleton

Este patrón de diseño permite asegurar que se crea una única instancia de una clase, siendo esta el único punto de acceso a sus operaciones. En JavaScript este patrón se aplica a la hora de crear cualquier objeto de configuración, en el caso concreto que nos ocupa, los ficheros de configuración del *frontend* (de audio o de idiomas) y los del *backend* (opciones de los modos de juego), para asegurar que se utiliza siempre la misma configuración. También, se aplica el patrón singleton en las llamadas a la API mediante la librería Axios [58], esta ofrece una única instancia para realizar las diferentes peticiones HTTP.

4.3.3.4. Observer

Este es otro patrón de diseño, muy utilizado en JavaScript, ya que es en lo que se basa el funcionamiento de las promesas. Dada la asincronía de el lenguaje, estas promesas son muy útiles cuando se necesita esperar por una respuesta de una función para poder utilizar sus datos. En este proyecto se ha usado en varios puntos, desde las llamadas al backend o a la API de Google, hasta las grabaciones de audio. También se utiliza este patrón a la hora de capturar las interacciones con los elementos de la interfaz gráfica.

4.3.3.5. Flux

Este es un patrón arquitectónico creado por Facebook [26], exclusivamente para React. Según Facebook, el patrón MVC, hacía muy complicado depurar aplicaciones grandes, y por ello, se inventaron su propio flujo. Este patrón, es en el que se basa el funcionamiento de Redux, que se encarga de gestionar el estado global de la aplicación *frontend*, del que ya se ha hablado en detalle en el apartado 1.3.5.5.

4.4. Diagramas de diseño

4.4.1. Modelo de datos

Mongo es una base de datos no relacional y existen varias maneras de modelar los datos [77]. En el caso de este proyecto se ha elegido establecer relaciones entre documentos, por una sencilla razón, es la más parecida a una base de datos relacional convencional con la que el desarrollador está familiarizado.

En el diagrama de la figura 4.24 se pueden observar las entidades que conforman la base de datos con sus atributos y relaciones.

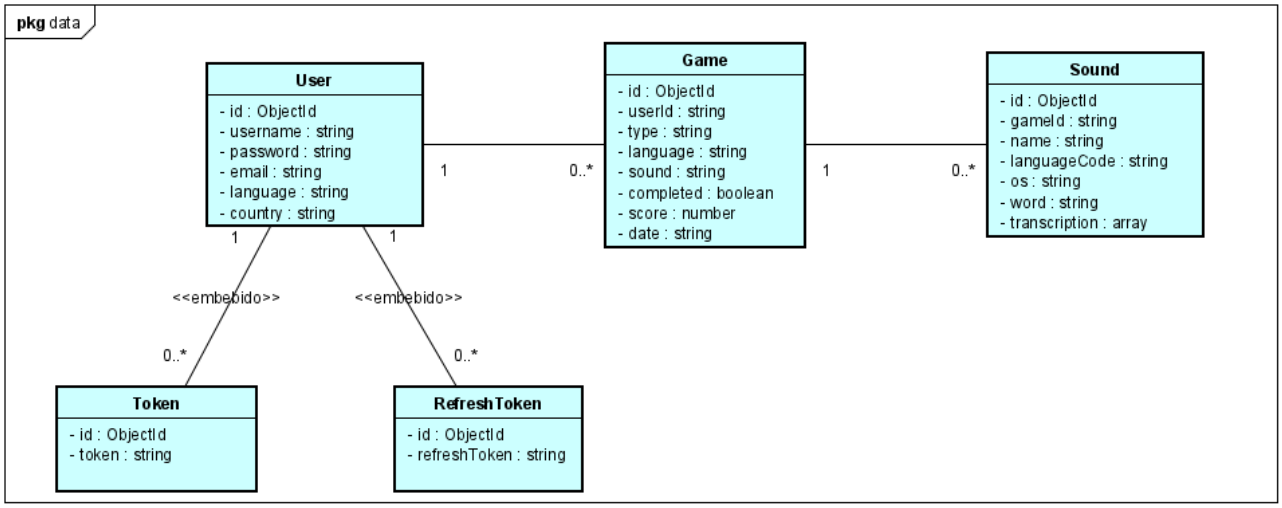

Figura 4.24: Diagrama de modelos de datos.

Como se puede apreciar en el diagrama anterior, el documento **User**, que contiene los datos de los usuarios, tiene una relación uno a mucho tanto con **Token**, que representa un *access token* de JWT, como con **RefreshToken**. Un usuario tendrá uno de cada por sesión tenga iniciada en diferentes dispositivos. También, **User** tiene una relación uno a muchos con **Game**, un usuario tendrá asociadas todas sus partidas. Y, por último, cada partida tiene asociados todos los audios, (siempre que en ese modo de juego se puedan grabar audios), **Sound**, que se han grabado durante ésta. Importante destacar que no son los archivos de audio, los cuales se almacenan en el servidor, sino que es la información de éstos.

4.4.2. Diagrama de paquetes

A continuación, en la figura 4.25 se puede ver una representación de los paquetes que componen la aplicación. En rojo, los paquetes correspondientes al *backend* y en azul el de *frontend*.

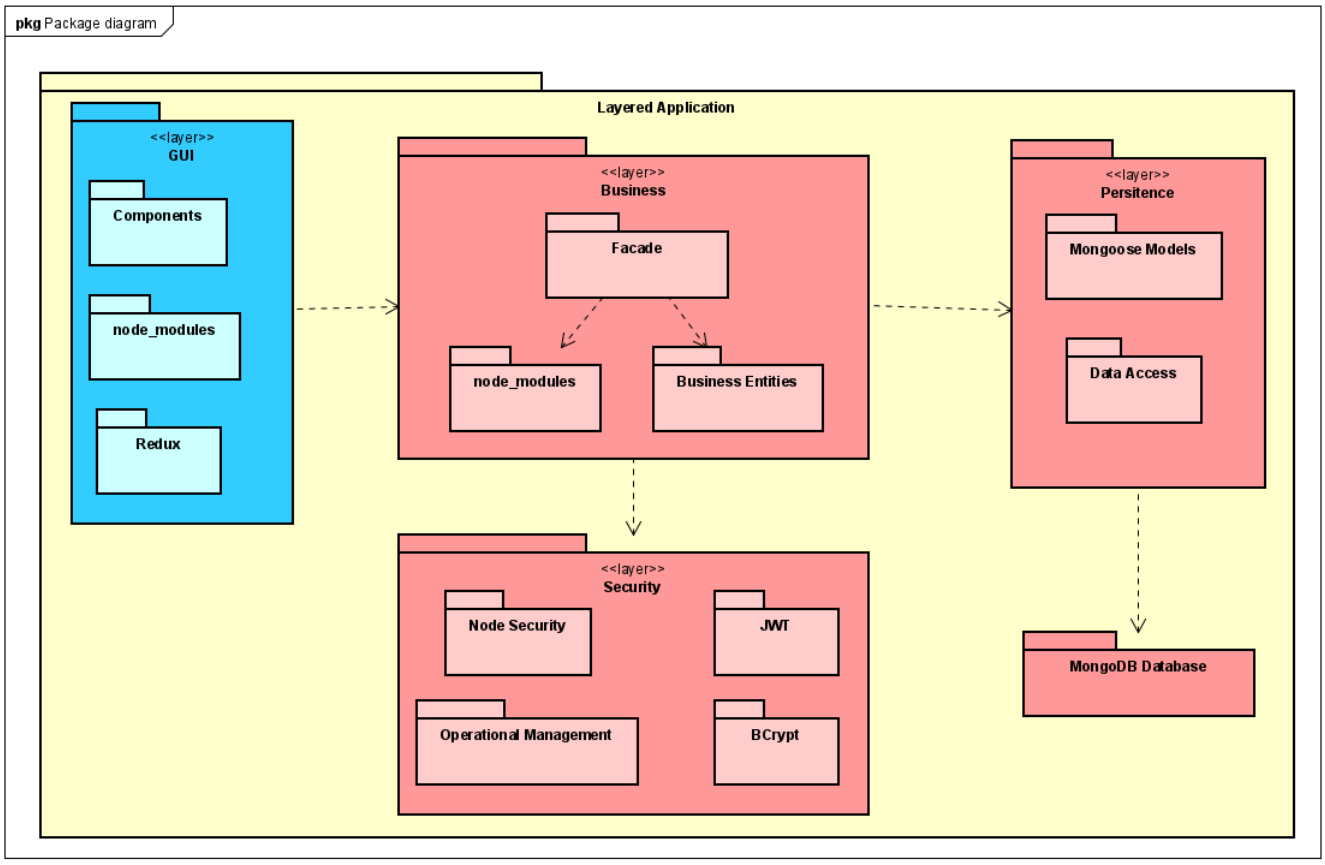

Figura 4.25: Diagrama de paquetes.

El proyecto se puede dividir en las siguientes capas:

- **GUI layer:** capa de interfaz gráfica de usuario. Esta es la capa con la que interactúa directamente el usuario y desde la que se envían las peticiones a la API.
    - **Components**: componentes necesarios para presentar las diferentes pantallas o componentes más pequeños de la aplicación móvil.

  - **Redux**: desde aquí se maneja el estado interno de la aplicación móvil y se envían las peticiones a la API.
  - **Node_modules:** paquetes de Node.js con funcionalidades utilizadas en el proyecto que facilitan el desarrollo de ciertas tareas.

- **Business layer:** capa de negocio o controlador. En esta capa se reciben las peticiones a la API y se realizan las operaciones necesarias para el correcto funcionamiento de la aplicación.
  - **Application facade:** la fachada que recibe las peticiones del cliente y envía los datos recibidos a las entidades para procesarlos.
  - **Business entities:** las entidades que operan con los datos recibidos en la fachada y devuelven una respuesta.
  - **Node_modules:** al igual que en *GUI layer*, son paquetes de Node.js.
- **Persitence layer:** capa de acceso datos. Esta capa hace de intermediario entre la capa de negocio y la base de datos, y es la que lanza las solicitudes necesarias para las transacciones.
  - **Data access**: parámetros de acceso a la base de datos, para poder realizar la conexión, como ruta, usuario y contraseña.
  - **Mongoose Models**: modelos del framework Mongoose, que definen los objetos y las operaciones de transacción con la base de datos.
- **Security layer:** capa de seguridad. Aquí se encuentran los paquetes necesarios para garantizar la seguridad de la aplicación, tanto en las transacciones como en el almacenamiento.
  - **Node security**: la propia seguridad que tiene Node.js por defecto.
  - **Operational management**: herramienta de Node.js que se encarga de gestionar las peticiones llegadas al servidor por diferentes usuarios, formando una cola con ellas y respondiendo en orden de llegada.
  - **JWT**: estándar que define una forma segura de transmitir información entre dos partes mediante la firma digital de los datos.
  - **BCrypt:** paquete de Node.js utilizado para codificar datos, en el caso de este proyecto, las contraseñas.
- **MongoDB database:** base de datos en al que se almacenan los datos necesarios para la aplicación.

4.4.3. Diagrama de despliegue

Como se puede observar en el diagrama de despliegue de la figura 4.26, un usuario se conectará desde una aplicación móvil, Android o iOS, y en el futuro podría ser posible vía web. Esta conexión con el servidor web Nginx se hace mediante peticiones HTTP. El servidor Nginx que actúa de servidor web y balanceador de carga, envía las peticiones al servidor Node.js. A su vez la aplicación Express.js realiza consultas a la base de datos MongoDB, conectándose mediante MongoDB driver, que en este

caso, ambas están en la misma máquina, pero perfectamente podrían estar en máquinas diferentes, simplemente habría que indicar en la aplicación Express.js la URL a la base de datos. Ver el apéndice B.1.2 para el manual de despliegue del sistema.

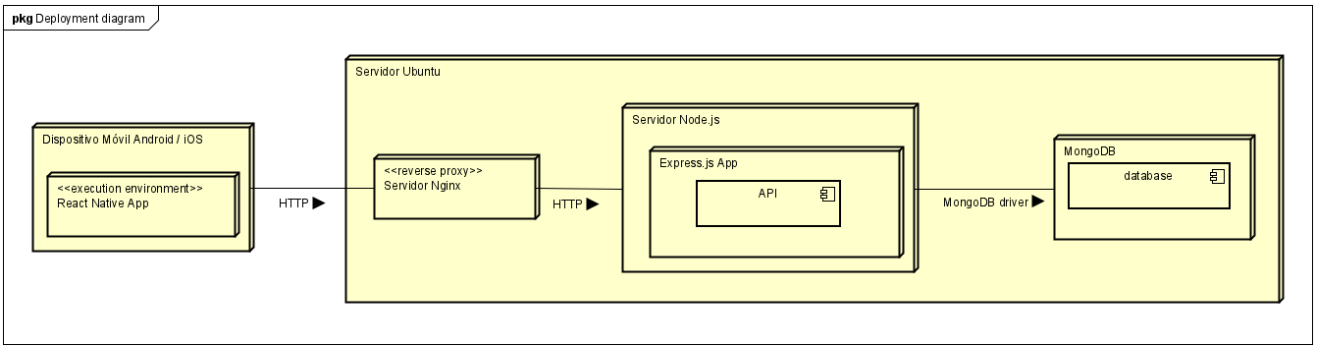

Figura 4.26: Diagrama de despliegue.

4.4.4. Estructura del log de actividad

Por día se genera un archivo JSON para cada jugador, almacenado en la ruta del servidor */users /nombre_usuario/logs*, que contiene todas las partidas de ese día. Éste está formado por un array de objetos JSON, cada objeto representa una partida jugada. Por lo tanto, en el array de un fichero habrá tantos objetos como partidas se hayan jugado ese día.

```
[
  {
    "player": {
      "username": string,
      "maternLanguage": string,
      "gameLanguage": string,
      "difficulty": string,
      "modality": {
        "modality": number,
        "mode": string
      },
      "challenge": {
        "nameList": string,
        "totalTimer": number,
        "totalPoints": number
      }
    },
    "events": [
      {
        "date": string,
        "eventName": {
          "name": string,
          "words": string[],
          "phoneme": string,
          "chosen": string
        }
      }
    ]
```

```
  }
]
```

Las propiedades del objeto son los siguientes:

- **player:** información relativa al jugador y detalles de la partida.
  - **username:** nombre del usuario que ha jugado la partida.
  - **motherLanguage:** idioma materno del usuario en formato BCP-47 [78].
  - **gameLanguage:** idioma que se ha practicado en formato BCP-47.
  - **difficulty:** dificultad de la partida. De momento el parámetro siempre es NA, hasta que haya niveles de dificultad implementados.
  - **modality:** modalidad elegida.
    - **modality:** juego individual (1) o multijugador (2).
    - **mode:** modo de juego de la partida. Posibles valores: *exposure, perception, pronunciation, mixed, memory, puzzle* y *theory*
  - **challenge:** resultado final de la partida.
    - **nameList:** lista que se ha practicado.
    - **totalTimer:** tiempo total que ha durado.
    - **totalPoints:** puntos totales obtenidos.
- **events:** array que contiene la información de los eventos transcurridos en la partida.
  - **date:** fecha y hora en formato ISO (ej.: 2011-10-05T14:48:00.000Z).
  - **event:** detalle del evento. Solo el atributo name es obligatorio, el resto depende de el tipo de evento que sea.
    - **name:** nombre del evento. Este puede ser: *exposure, perception, prounciation, mixed, puzzle, memory o theory*, cuando se inicia un modo de juego; *listen y record* para cuando se reproduce o graba un audio; y, *prompt_out* cuando se sale de la partida.
    - **result:** *true* si ha acertado, *false* si no.
    - **wordsList:** par de palabras involucradas.
    - **correctWord (modo perception):** palabra correcta.
    - **words: (modo puzzle y memory)** array de palabras.
    - **phoneme (modo puzzle):** fonema que hay que buscar.
    - **chosen (modo puzzle):** array con palabras elegidas.
    - **solution (modo memory):** array de las palabras correctas con las posiciones correspondientes al array de words.
    - **correct (modo memory):** array de booleanos. *True*, si la palabra es correcta, *false* si no.
    - **word (reproducrir y grabar sonido):** palabra que se ha grabado o reproducido.
    - **audioFile (grabar sonido):** nombre del archivo de audio.
    - **alternatives (grabar sonido):** alternativas proporcionadas por el ASR.

Capítulo 5

Pruebas

Las pruebas tanto de integración en el servidor, como las unitarias en la app, se han automatizado mediante Jest [53]. En ambos entornos, para pasar los test basta con ponerse en la carpeta raíz del proyecto y ejecutar el comando:

```
npm run test
```

5.1. Test en Backend

Para el backend se han realizado test de integración, que consisten en probar la interacción entre diferentes partes del proyecto. En este caso, se simula un cliente para realizar peticiones a la API y así poder comprobar las respuestas que ésta proporciona en las diferentes situaciones posible.

Antes de empezar a ejecutar los tests, automáticamente, se crea un usuario de prueba. Y antes de ejecutar cada test, se hace un login para poder obtener un token con el que identificarse en las llamadas que lo requieren. También se mockean algunos documentos a los que se hacen consultas para poder controlar las salidas.

CP01	Registro de usuario
Descripción	Registrar un usuario en el sistema
API REST	/api/users
Entrada	{ username: ”test_user_2”, password: ”1234”, email: ”test2@test.com”, country: ”SPAIN”, language: ”es-ES” }
Resultado esperado	{ user: { username: ”test_user_2”, email: ”test2@test.com”, country: ”SPAIN”, language: ”es-ES” }, token: * }

Tabla 5.1: Prueba CP01.

CP02	Registro de usuario ya existente
Descripción	Registrar un usuario que ya existe en el sistema
API REST	/api/users
Entrada	{ username: ”test_user”, password: ”1234”, email: ”test@test.com”, country: ”SPAIN”, language: ”es-ES” }
Resultado esperado	”duplicateError”

Tabla 5.2: Prueba CP02.

CP03	Registro de usuario: formato email incorrecto
Descripción	Registrar un usuario en el sistema con un email no válido
API REST	/api/users
Entrada	{ username: ”test_user”, password: ”1234”, email: ”test.com”, country: ”SPAIN”, language: ”es-ES” }
Resultado esperado	”validationError”

Tabla 5.3: Prueba CP03.

CP04	Cerrar sesión de un usuario
Descripción	Cerrar la sesión de un usuario identificado en el sistema
API REST	/api/users/me/logout
Entrada	Bearer token
Resultado esperado	”Logged out.”

Tabla 5.4: Prueba CP04.

CP05	Cerrar sesión de un usuario, sin identificación
Descripción	Cerrar sesión de un usuario si enviar el token de identificación
API REST	/api/users/me/logout
Entrada	
Resultado esperado	{ error: Unauthorized access. }

Tabla 5.5: Prueba CP05.

CP06	Listado de lenguajes
Descripción	Listar los lenguajes disponibles para creación de un usuario
API REST	/api/masters/languages
Entrada	
Resultado esperado	[{ ”code”: ”af”, ”name”: ”Afrikaans” }, { ”code”: ”en-GB”, ”name”: ”English (United Kingdom)” }, { ”code”: ”es”, ”name”: ”Spanish” }

Tabla 5.6: Prueba CP06.

CP07	Listado de lenguajes erroneo
Descripción	Los lenguajes no se ordenan por orden alfabético de nombre, se devolverían en el orden que estén en el JSON
API REST	/api/masters/languages
Entrada	
Resultado esperado	No Igual a [{ ”code”: ”af”, ”name”: ”Afrikaans” }, { ”code”: ”es”, ”name”: ”Spanish” }, { ”code”: ”en-GB”, ”name”: ”English (United Kingdom)” }

Tabla 5.7: Prueba CP07.

CP08	Listado de paises
Descripción	Listar los países disponibles para la creación de un usuario
API REST	/api/users
Entrada	
Resultado esperado	[{ ”name”: ”AFGHANISTAN”, ”code”: ”AF” }, { ”name”: ”ALBANIA”, ”code”: ”AL” }, { ”name”: ”ALGERIA”, ”code”: ”DZ” },]

Tabla 5.8: Prueba CP08.

5.2. Test en Frontend

En el frontend se han hecho test unitarios. Un test unitario, se puede definir como una prueba que comprueba el correcto funcionamiento de una unidad de código [79]. En este proyecto se han realizado para probar el renderizado correcto de los componentes de la aplicación. Además de probar los componentes también se ha utilizado para probar que el estado de la aplicación sigue un flujo correcto, es decir, se han hecho pruebas con los dispatchers de Redux.

Estos cuatro primeros test prueban la correcta renderización del componente "Box", el cual, se encarga de mostrar las palabras al usuario y renderizar los botones para grabar. El componente que se utiliza como **entrada** para esta suite tiene las siguientes propiedades:

```
<Box
   word={'palabra'}
   phonetic={'transcripcion'}
   color={'#fafafa'}
   isCorrect={correct => jest.fn(correct)}
   showHint={(alternatives) => jest.fn(alternatives)}
   setTimer={(recording) => jest.fn(recording)} />
```

Como se puede ver en el código anterior, en lugar de funciones reales, aparece *jest.fn()*. Este es el método [80] que tiene Jest para crear funciones mock, para casos en los que no se dispone de la función real. Para comprobar la estructura del componente renderizado, se ha utilizado la librería *react-test-renderer* [81], que proporciona diferentes funciones con este fin, en este caso se ha utilizado la función *toJSON*, que convierte el componente en un objeto JSON de manera que se puede acceder al contenido de éste y al de todos sus hijos.

CP09	Renderizado del componente Box
Descripción	Se comprueba que el componente se renderiza
Componente	Box
Resultado esperado	El componente se renderiza

Tabla 5.9: Prueba CP09.

CP10	Mostrar palabra correcta
Descripción	Se comprueba que la palabra mostrada al usuario es la que se ha establecido en la creación del componente (propiedad *word* en el componente)
Componente	Box
Resultado esperado	'palabra'

Tabla 5.10: Prueba CP10.

CP11	Mostrar transcripción fonética correcta
Descripción	Se comprueba que la transcripción mostrada al usuario es la que se ha establecido en la creación del componente (propiedad *phonetic* en el componente)
Componente	Box
Resultado esperado	'transcripcion'

Tabla 5.11: Prueba CP11.

CP12	Mostrar número de intentos
Descripción	Se comprueba que se muestra en número de intentos (por defecto 3)
Componente	Box
Resultado esperado	'Remaining attempts: 3'

Tabla 5.12: Prueba C12.

CP13	Colorear botones con el color correspondiente
Descripción	Se comprueba que los botones cogen el color que se ha establecido en la creación del componente (propiedad *color* en el componente)
Componente	Box
Resultado esperado	'#fafafa'

Tabla 5.13: Prueba CP13.

En los siguientes test se hacen pruebas sobre Redux, que gestiona el estado global de la aplicación. Para hacer estas comprobaciones se han realizado pruebas sobre las funciones que actúan como reducers (en el apartado 1.3.5.5 se habla más en detalle de estos conceptos). La entrada es un objeto conformado por el *type* y el *payload* de la acción que se ejecuta. Primero se prueba el reducer del usuario.

CP14	Retorno del estado inicial
Descripción	El reducer devuelve el estado inicial
Reducer	User
Entrada	'undefined'
Resultado esperado	{}

Tabla 5.14: Prueba CP14.

CP15	Estado correcto tras login
Descripción	Devuelve la información del usuario después de un login exitoso
Reducer	User
Entrada	{ type: LOGIN, payload: {user: {username: 'antonio', email: 'antonio@gmail.com', country: 'SPAIN', language: 'es-ES'}, token: 'aaaaaaaaaaaaaaa', refreshToken: 'bbbbbbbbbbb' }}
Resultado esperado	{user: {username: 'antonio', email: 'antonio@gmail.com', country: 'SPAIN', language: 'es-ES'}, token: 'aaaaaaaaaaaaaaa', refreshToken: 'bbbbbbbbbbb' }

Tabla 5.15: Prueba CP15.

CP16	Estado del usuario logueado
Descripción	Devuelve la información del usuario logueado
Reducer	User
Entrada	{ type: SAVE_USER, payload: {user: {username: 'antonio', email: 'antonio@gmail.com', country: 'SPAIN', language: 'es-ES'}}
Resultado esperado	{user: {username: 'antonio', email: 'antonio@gmail.com', country: 'SPAIN', language: 'es-ES'} }

Tabla 5.16: Prueba CP16.

De aquí en adelante se prueba el reducer de las partidas. Para este caso se usa un estado inicial simulando que el objeto ya se ha inicializado y el contenido del estado es el siguiente:

```
{
    gameLanguage: 'en-UK',
    gameId: '134'
}
```

CP17	Retorno del estado inicial
Descripción	El reducer devuelve el estado inicial
Reducer	Game
Entrada	'undefined'
Resultado esperado	{ gameLanguage: 'en-UK', gameId: '134' }

Tabla 5.17: Prueba CP17.

CP18	Actualiza el idioma de la partida
Descripción	Actualiza y devuelve el idioma de la partida actual
Reducer	Game
Entrada	{ type: SET_GAME_LANGUAGE, payload: 'es-ES' }
Resultado esperado	{ gameLanguage: 'es-ES', gameId: '134' }

Tabla 5.18: Prueba CP18.

CP19	Actualiza el ID de la partida
Descripción	Actualiza y devuelve el ID de la partida actual
Reducer	Game
Entrada	{ type: SET_GAME_ID, payload: '135' }
Resultado esperado	{ gameLanguage: 'en-UK', gameId: '135' }

Tabla 5.19: Prueba CP19.

5.3. Test de usabilidad

Estas pruebas se llevan a cabo con usuarios reales y consisten en probar la aplicación como si fuera un caso real, observado en primera persona el comportamiento del usuario durante el proceso. El hecho de que haga falta estar presente mientras el usuario completa la prueba, ha sido un problema, ya que durante la realización de estas (principios del año 2021), existían varias restricciones en cuanto a reuniones sociales y movilidad debido a la pandemia de la COVID-19. Por esto, sólo se han realizado pruebas con dos usuarios reales.

En estas pruebas se plantean dos escenarios. El primero, consiste en crear una cuenta y, posteriormente, identificarse en el sistema. El segundo, trata de jugar una partida completa al modo de juego *Mixto*. Al final de cada escenario, se le hacen varias preguntas al usuario para completar la información. Para evaluar las acciones de los usuarios se ha utilizado una plantilla para cada paso que requiere ser observado. La plantilla utilizada es la siguiente:

Nº	Descripción	¿Completado?	Valoración
Métrica			
Observaciones			

Tabla 5.20: Plantilla utilizada para los test de usabilidad.

- **Nº:** número de la acción.
- **Descripción:** breve descripción de lo que tiene que hacer el usuario.
- **¿Completado?:** si consigue completar o no la acción (sí o no).
- **Valoración:** puntuación de cómo se ha llevado a cabo la acción (1 a 5).
- **Métrica:** explicación de lo que hay que tener en cuenta para evaluar cómo se ha llevado a cabo la acción.
- **Observaciones:** anotaciones que puedan ser de interés sobre lo que hace el usuario.

Perfiles de usuario:

Nº	**Edad**	**Habilidad con la tecnología (0 a 5)**	**Experiencia en videojuegos (0 a 5)**
1	10	4	3
2	58	2	0

Tabla 5.21: Perfiles de usuario.

Las pantallas correspondientes al primer escenario son las siguientes:

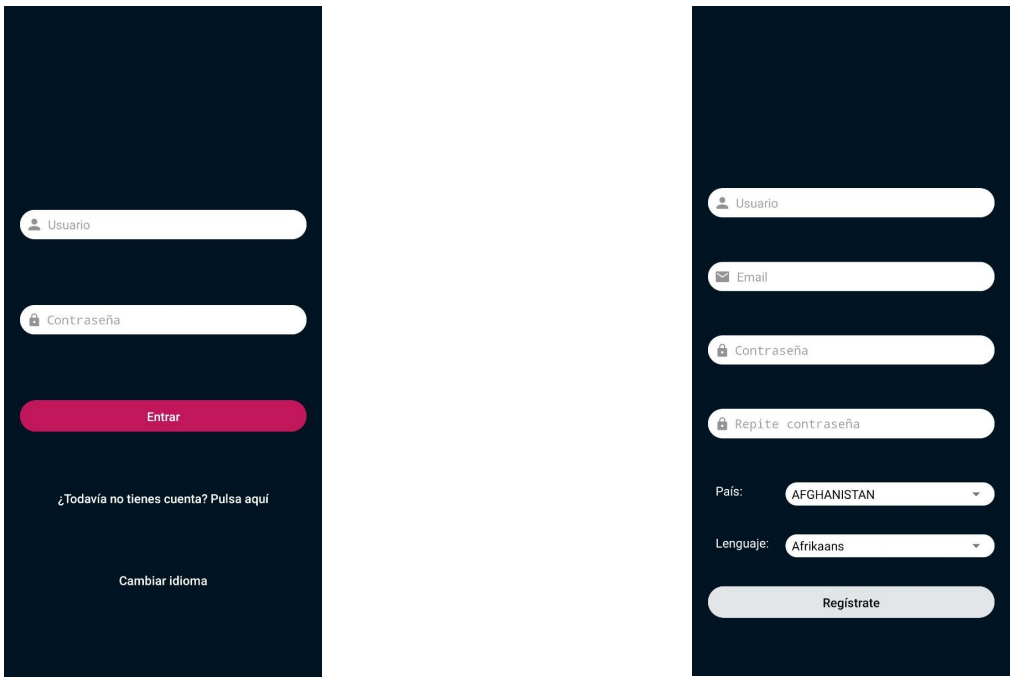

Figura 5.1: Pantallas de login y registro.

Y, las pantallas correspondientes al segundo escenario son las estas:

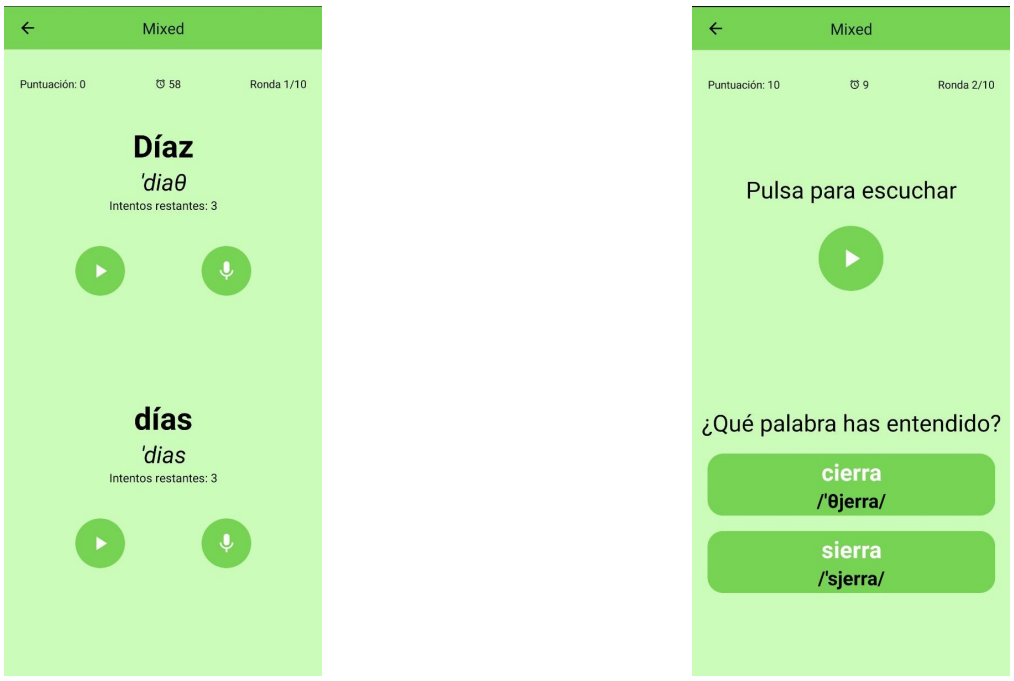

Figura 5.2: Pantallas del modo de juego mixto.

A continuación, se encuentran los documentos completados con las anotaciones pertinentes tomadas por el desarrollador. Estas anotaciones aparecen en letra negrita.

Usuario 1 - Tarea 1

Descripción: el usuario deberá completar un registro, hacer login y elige el idioma inglés de UK y un sonido cualquiera. Se parte desde la pantalla de login y se termina el menú de modos de juego.

1.	El usuario encuentra llamativo y agradable el diseño de la aplicación.	¿Completado? SI **NO**	Valoración **1** 2 3 4 5
Métrica: Hace algún comentario al respecto			
Observaciones: **No dice nada al respecto**			
2.	El usuario sabe dónde pulsar para crear una nueva cuenta	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si tarda menos de 5 segundos en seleccionar la opción correcta. Tiempo: **3 seg** Número de toques (mínimo 1): **1** Errores: **0**			
Observaciones: **Ninguna**			
3.	El usuario es capaz de realizar el registro.	¿Completado? **SI** NO	Valoración 1 2 3 **4** 5
Métrica: Satisfactorio si no se atasca con ningún paso Tiempo: **45 seg** Número de toques (mínimo 7 sin contar la escritura con el teclado): **7** Errores: **0**			
Observaciones: **En lenguaje materno, ha elegido idioma español genérico en lugar de español de España**			
4.	El usuario es capaz de hacer login en la app.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si consigue loguearse Tiempo: **10 seg** Número de toques (mínimo 3): **3** Errores: **0**			
Observaciones: **Ninguna**			

5.	El usuario entiende que tiene que hacer en la selección de idioma y sonido.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si elige la opción deseada sin confusión y en menos de 20 segundos. Tiempo: **8 seg** Número de toques (mínimo 4): **4** Errores: **0**			
Observaciones: **Ninguna**			

Preguntas:

¿Le ha parecido intuitiva la interfaz? **Si**

¿Ha habido algún momento en el que no supiera que hacer? **No**

¿El tamaño de la letra le parece correcto? **Si**

¿El tamaño de los botones es adecuado? **Si**

¿Algún comentario que quiera hacer? **No**

Usuario 1 - Tarea 2

Descripción: el usuario deberá completar un juego de la modalidad "mixto". Se parte desde el menú de modos de juego y se termina el mismo.

1.	El usuario encuentra llamativo y agradable el diseño de la aplicación.	¿Completado? SI **NO**	Valoración **1** 2 3 4 5
Métrica: Hace algún comentario al respecto			
Observaciones: **No dice nada al respecto**			
2.	El usuario elige el modo mixto.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si elige la opción correcta en menos de 10 segundos Tiempo: **4 seg** Número de toques (mínimo 1): **1** Errores: **0**			
Observaciones: **Ninguna**			
3.	El usuario reproduce la palabra del modo pronunciación.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si el usuario da al play en menos de 5 segundos Tiempo: **5 seg** Número de toques (mínimo 1): **1** Errores: **0**			
Observaciones: **Ninguna**			
4.	El usuario graba con su voz la palabra	¿Completado? **SI** NO	Valoración 1 **2** 3 4 5
Métrica: Satisfactorio si el usuario graba la palabra en menos de 5 segundos Tiempo: **30 seg** Número de toques (mínimo 1): **7** Errores: **6**			
Observaciones: **El usuario no se da cuenta que para grabar hay que dejar pulsado el botón hasta después de varios intentos pulsando y soltando**			

5.	El usuario una vez acaba con la primera palabra repite el proceso con la segunda	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si se da cuenta que tiene que contestar a la segunda palabra Tiempo: **10 seg** Número de toques (mínimo 2): **2** Errores: **0**			
Observaciones: **Una vez sabe como grabar lo hace sin problema**			
6.	El usuario reproduce manualmente la palabra del modo percepción.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si pulsa el botón de play a la primera Tiempo: **2 seg** Número de toques: **1** Errores: **0**			
Observaciones: **Ninguna**			
7.	El usuario elige una palabra del modo percepción.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si sabe cómo elegir la palabra deseada Tiempo: **3 seg** Número de toques: **1** Errores: **0**			
Observaciones: **Ninguna**			
8.	El usuario elige finalizar la partida.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si elige el botón volver al menú en menos de 10 segundos Tiempo: **4 seg** Número de toques: **1** Errores: **0**			
Observaciones: **Ninguna**			

Preguntas:

¿Le ha resultado agradable la interfaz? **Si**

¿Ha habido algún momento en el que no supiera que hacer? **No**

¿El tamaño de la letra le parece correcto? **Si**

¿El tamaño de los botones es adecuado? **Si**

¿Algún comentario que quiera hacer? **No**

Usuario 2 - Tarea 1

Descripción: el usuario deberá completar un registro, hacer login y elige el idioma inglés de UK y un sonido cualquiera. Se parte desde la pantalla de login y se termina el menú de modos de juego.

1.	El usuario encuentra llamativo y agradable el diseño de la aplicación.	¿Completado? SI **NO**	Valoración **1** 2 3 4 5
Métrica: Hace algún comentario al respecto			
Observaciones: **No dice nada al respecto**			
2.	El usuario sabe dónde pulsar para crear una nueva cuenta	¿Completado? **SI** NO	Valoración 1 **2** 3 4 5
Métrica: Satisfactorio si tarda menos de 5 segundos en seleccionar la opción correcta. Tiempo: **8 seg** Número de toques (mínimo 1): **1** Errores: **0**			
Observaciones: **Ninguna**			
3.	El usuario es capaz de realizar el registro.	¿Completado? **SI** NO	Valoración 1 2 3 **4** 5
Métrica: Satisfactorio si no se atasca con ningún paso Tiempo: **120 seg** Número de toques (mínimo 7 sin contar la escritura con el teclado): **7** Errores: **0**			
Observaciones: **Ha buscado el país (España) por la letra E en lugar de por la letra S (Spain)**			
4.	El usuario es capaz de hacer login en la app.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si consigue loguearse Tiempo: **15 seg** Número de toques (mínimo 3): **3** Errores: **0**			
Observaciones: **Ninguna**			

5.	El usuario entiende que tiene que hacer en la selección de idioma y sonido.	¿Completado? SI / NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si elige la opción deseada sin confusión y en menos de 20 segundos. Tiempo: **10 seg** Número de toques (mínimo 4): **4** Errores: **0**			
Observaciones: **Ninguna**			

Preguntas:

¿Le ha parecido intuitiva la interfaz? **Si**

¿Ha habido algún momento en el que no supiera que hacer? **No**

¿El tamaño de la letra le parece correcto? **Si**

¿El tamaño de los botones es adecuado? **Si**

¿Algún comentario que quiera hacer? **Poner los países e idiomas en castellano.**

Usuario 1 - Tarea 2

Descripción: el usuario deberá completar un juego de la modalidad "mixto". Se parte desde el menú de modos de juego y se termina el mismo.

1.	El usuario encuentra llamativo y agradable el diseño de la aplicación.	¿Completado? SI **NO**	Valoración **1** 2 3 4 5
Métrica: Hace algún comentario al respecto			
Observaciones: **No dice nada al respecto**			
2.	El usuario elige el modo mixto.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si elige la opción correcta en menos de 10 segundos Tiempo: **10 seg** Número de toques (mínimo 1): **1** Errores: **0**			
Observaciones: **Ninguna**			
3.	El usuario reproduce la palabra del modo pronunciación.	¿Completado? **SI** NO	Valoración 1 **2** 3 4 **5**
Métrica: Satisfactorio si el usuario da al play en menos de 5 segundos Tiempo: **20 seg** Número de toques (mínimo 1): **1** Errores: **0**			
Observaciones: **Parece que no sabe cómo actuar ante la primera pantalla, pero al final si que da al botón.**			
4.	El usuario graba con su voz la palabra	¿Completado? SI **NO**	Valoración **1** 2 3 4 5
Métrica: Satisfactorio si el usuario graba la palabra en menos de 5 segundos Tiempo: **60 seg** Número de toques (mínimo 1): **5** Errores: **5**			
Observaciones: **El usuario no se da cuenta que para grabar hay que dejar pulsado el botón**			

5.	El usuario una vez acaba con la primera palabra repite el proceso con la segunda	¿Completado? SI **NO**	Valoración **1** 2 3 4 5
Métrica: Satisfactorio si se da cuenta que tiene que contestar a la segunda palabra Tiempo: - Número de toques (mínimo 2): - Errores: -			
Observaciones: **No le da tiempo a contestar la segunda palabra**			
6.	El usuario reproduce manualmente la palabra del modo percepción.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si pulsa el botón de play a la primera Tiempo: **3 seg** Número de toques: **1** Errores: **0**			
Observaciones: **Ninguna**			
7.	El usuario elige una palabra del modo percepción.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si sabe cómo elegir la palabra deseada Tiempo: **2 seg** Número de toques: **1** Errores: **0**			
Observaciones: **Ninguna**			
8.	El usuario elige finalizar la partida.	¿Completado? **SI** NO	Valoración 1 2 3 4 **5**
Métrica: Satisfactorio si elige el botón volver al menú en menos de 10 segundos Tiempo: **7 seg** Número de toques: **1** Errores: **0**			
Observaciones: **Ninguna**			

Preguntas:

¿Le ha resultado agradable la interfaz? **Si**

¿Ha habido algún momento en el que no supiera que hacer? **Si. Cuando fallaba la grabación de palabras.**

¿El tamaño de la letra le parece correcto? **Si**

¿El tamaño de los botones es adecuado? **Si**

¿Algún comentario que quiera hacer? **Falta una explicación de lo que hay que hacer en el juego.**

5.3.1. Conclusiones de los test de usabilidad

Como era de esperar, el usuario con más manejo de la tecnología ha hecho prácticamente todas las tareas en menos tiempo que el otro. Aún así, el usuario con menos manejo, tampoco ha encontrado grandes problemas para moverse por la aplicación y jugar la partida.

Tras acabar las pruebas y analizar los datos obtenidos, se pueden destacar varios aspectos de la aplicación a mejorar. A continuación se hace una lista de estos y sus posibles soluciones.

- Elección de idiomas en la creación de usuario. En el desplegable donde se puede establecer el idioma del usuario, aparecen idiomas sin región, lo que en ciertos casos (ej.: español), puede causar problemas con el ASR. Cómo solución, habría que extraer de la lista de selección los idiomas sin región.
- Las listas de idiomas y regiones en la creación de usuario sólo aparecen en inglés. Hay que añadir estas listas en más idiomas.
- Confusión a la hora de grabar audios. En lugar de dejar pulsado el botón, el usuario pulsa una vez. La solución es añadir algún tipo de aviso informando del uso del botón. Esto ya estaba contemplado pero no se había implementado en el momento de las pruebas.
- El usuario no sabe de qué va el juego. Se podría añadir sección de instrucciones, explicando brevemente el objetivo, las mecánicas y el sistema de puntuación de cada juego.

Capítulo 6

Conclusiones

En este capítulo se exponen las diferentes ideas que se han ido recopilando a lo largo de la realización del proyecto. Además se arroja luz a posibles implementaciones y mejoras futuras del mismo.

En este TFG se ha desarrollado una aplicación multiplataforma en la que, mediante elementos de juego y actividades predefinidas, se permite entrenar y mejorar la pronunciación de un idioma extranjero. Mediante la implementación de diferentes modos de juegos, se consigue que una actividad que puede resultar tediosa, sea divertida, incrementando de esta forma la motivación del usuario por aprender y practicar. Esta aplicación consta de una parte de servidor o *backend*, desarrollado con Node.js, Express.js y MongoDB, y otra parte, cliente o *frontend*, en la que se ha utilizado Node.js y React Native apoyado en el framework Expo. Además, las tecnologías de reconocimiento y sintetizado de voz que provee Google en *Google cloud platform* [82] han sido una parte clave del sistema final. Se han realizado pruebas unitarias y de usabilidad que han permitido obtener retroalimentación del sistema.

Este proyecto implementa todos los modos de juego de los proyectos anteriores y añade dos modos de juego nuevos y un modo teórico. También deja preparada la base para añadir nuevos modos y funcionalidad multijugador. En el *frontend*, gracias a la fácil reutilización de los componentes básicos para la funcionalidad principal de la aplicación, entre otros, el de síntesis y el de reconocimiento de voz, ampliar la funcionalidad de la aplicación resulta muy sencillo y rápido. En cuanto a la parte *backend*, dependerá de la necesidad, pero la llamada para generar listas de pares es fácilmente modificable, como demuestran las adaptaciones del modo 'Puzzle' y 'Memoria'. Además, gracias en parte a los componentes y a la estructura del proyecto, aislando el código por pantallas en el *frontend* y por modelos en el *backend*, hace que sea sencillo localizar el origen de posibles fallos o funcionalidades que necesiten una modificación, haciendo que el código sea fácil de mantener. A esto, hay que añadir que solo hay que mantener un código para todas las plataformas disponibles. En particular, las plataformas compatibles con este proyecto son Android e iOS. El código es fácilmente adaptable a web y escritorio, pero por falta de tiempo en este TFG no se ha llevado a cabo dicha adaptación.

Durante el desarrollo del proyecto, han surgido muchas situaciones nuevas y se han descubierto nuevas tecnologías y herramientas. Para empezar, se ha hecho palpable la dificultad de hacer una estimación correcta durante la planificación, y más cuando no se conocen las tecnologías de desarrollo,

por ello es bueno tener un plan de actuación frente a los riesgos. También, la importancia de elegir una buena metodología para la situación concreta, en este caso, iterativa e incremental, ha sido muy importante porque ha permitido la adaptación de los requisitos y adaptar el ritmo de trabajo a los requisitos. Por otro lado, el realizar un despliegue real, con todo lo que ello conlleva en cuanto a configuración y otros detalles, también ha sido muy provechoso. Por último, tanto la automatización de pruebas para el código, como, sobretodo, las pruebas con usuarios reales han resultado ser de mucha ayuda para mejorar la aplicación, ya que desde el punto de vista del desarrollador, al conocerla perfectamente, hay ciertos escenarios que no se llegan a probar.

A pesar de todo, se han quedado algunas características en el camino. Por ejemplo, no se ha implementado la selección de dificultad desde el *frontend*, si bien, el backend está preparado para ello. Algo que podría ser mejorado es la componentización de la parte frontend, después de haber adquirido cierta experiencia con React Native y con el propio proyecto, se ve claro que hay partes que deberían estar menos acopladas. Otra mejora podría ser migrar el código de JavaScript a TypeScript [83], ya que el tipado hace que el código se más entendible y a la larga hace que sea más fácil mantenerlo. Por último, hubiera sido interesante implementar integración continua en el backend, y puede ser interesante hacerlo en un futuro.

En lo personal, el proyecto ha sido muy enriquecedor de principio a fin. Desde aplicar lo aprendido en la carrera en cuanto a planificación y estimaciones, hasta el aprender a usar herramientas nuevas como Expo o Swagger. Es cierto que en algunas partes se encontró mayor dificultad de la que se podía esperar, sobre todo por el desconocimiento de las tecnologías utilizadas. En definitiva, todo esto ha resultado en una experiencia muy interesante, similar a lo que podría ser el mundo laboral, y muy gratificante al ver el resultado final.

6.1. Trabajo futuro

- **Seguridad:** añadir elementos para aumentar la seguridad, como por ejemplo: enviar un email al realizar el registro, requisitos de caracteres para la contraseña, comunicación por HTTPS y otros que puedan surgir.

- **Nuevos casos de juego:** creando las pantallas correspondientes y ampliando la API REST con las llamadas necesarias. Por ejemplo, un modo de juego de entrenamiento, en el que el usuario pueda elegir las palabras a practicar en lugar de elegirse de forma aleatoria.

- **Multijugador:** implementar desafíos por turnos o tiempo real contra otros jugadores.

- **Modo web**: adaptar el código para que pueda compilar a web [84]. También habría que adaptar el diseño de la interfaz a escritorio [85].

- **Diseño especifico para tablet:** el juego corre perfectamente en tablets, pero el diseño es el mismo que en móvil. Una pantalla más grande se puede aprovechar mejor. Quizás pueda valer el mismo diseño que para escritorio.

- **Accesibilidad:** meter la opción de adaptar la interfaz para personas con algún tipo de discapacidad visual, auditiva o de otro tipo.

- **Internacionalizar servicios:** por ejemplo, en la creación de usuario, los idiomas y países están sólo en inglés. Estaría bien que coincidieran con el idioma de la aplicación.

- **Añadir un apartado de instrucciones:** dentro de la aplicación añadir una explicación breve de cómo jugar a cada modo de juego.

- **Solución de bugs:** arreglar posibles bugs que vayan surgiendo. Sobre todo en el *frontend*, ya que el *backend* no tiene demasiada lógica más allá de las consultas a la base de datos. Lo más crítico durante el desarrollo en la parte del *frontend* ha sido el temporizador y la grabación de audios, así que, ante alguna acción no contemplada es probable que aparezca algún bug.

- **Elementos de gamificación:** añadir elementos como rankings más específicos y sistemas de logros o trofeos.

Apéndices

Apéndice A

Acrónimos

- **API**: Application Programming Interface.
- **ASR**: Automatic speech recognition.
- **CLI**: Command line interface.
- **CPU**: Central processing unit.
- **ECA-SIMM**: Grupo de investigación de entornos de computación avanzada y sistemas de interacción multimodal.
- **HTTP**: Hypertext Transfer Protocol.
- **HTTPS**: Hypertext Transfer Protocol Secure.
- **IDE**: Integrated Development Environment.
- **IVA**: Impuesto sobre el Valor Añadido.
- **JSON**: JavaScript Object Notation.
- **JWT**: JSON web token.
- **MERN**: MongoDB, Express, React, Node.
- **PWA**: Progressive web app.
- **RAM**: Random access memory.
- **STT**: Speech-to-text.
- **TFG**: Trabajo de fin de grado.
- **TTS**: Text-to-speech.

Apéndice B

Manual de instalación y despliegue

En esta sección se explican los pasos a seguir para la instalación tanto del entorno de desarrollo como del de despliegue de ambas partes del proyecto. Primero se explica la parte del servidor (*backend*) y a continuación la parte de la aplicación cliente (*frontend*).

A no ser que se diga lo contrario, los comandos se pueden ejecutar desde cualquier ruta.

B.1. Instalación del servidor

Para empezar, se debe descargar el proyecto desde el repositorio correspondiente. Todos los comandos que se indican a continuación se ejecutan desde consola.

B.1.1. Entorno desarrollo

B.1.1.1. Prerrequisitos

- MongoDB en versión 4.2.1.
- Node en versión 12.3.0.

B.1.1.2. Instalación

Nos movemos al directorio donde hemos descargado el proyecto y ejecutamos:

```
npm i
```

El proceso puede demorarse unos minutos. Una vez acabe ejecutamos:

```
npm run start
```

Una vez veamos por consola el mensaje "Server running on port 3000", significará que el servidor ya está levantado. Mientras el servidor este arrancado, cada vez que se realice y guarde algún cambio en el código, el servidor se reinicia automáticamente para aplicarlo.

B.1.2. Entorno despliegue

B.1.2.1. Preparando el entorno

Aquí se describe cómo desplegar el servidor en una máquina con sistema operativo Ubuntu 20.04.1 LTS. Para ello, se usa Nginx, para configurar el servidor proxy inverso, y PM2, el cual se encargará de mantener el servidor Node levantado.

1. Instalar un firewall básico [86]:

   ```
   # apt update
   # apt install ufw
   ```

   Permitimos el acceso vía ssh y habilitamos el firewall

   ```
   # ufw allow OpenSSH
   # ufw enable
   ```

   Una vez hecho esto, se recomienda usar un usuario con privilegios de administrador para seguir con el proceso, no se aconseja usar la cuenta de root por motivos de seguridad.

2. Instalar Nginx [45]:

   ```
   $ sudo apt update
   $ sudo apt install nginx
   ```

   Configuramos el firewall para que permita conexiones con Nginx

   ```
   $ sudo ufw allow 'Nginx HTTP'
   $ sudo ufw status
   ```

   Se puede comprobar que Nginx está activo con el siguiente comando

   ```
   $ systemctl status nginx
   ```

   Además si se accede a http://localhost, ahora aparece la página por defecto de Nginx.

3. Configurar Nginx:
   Primero, hay que crear un archivo de configuración. Substituir name_domain por el nombre de dominio correspondiente.

```
$ sudo nano /etc/nginx/sites-available/name_domain
```

El contenido de este debe ser el siguiente:

```
upstream api {
    server 127.0.0.1:3000;
}
server {
    listen 80 name_domain;
    location / {
            proxy_pass http://localhost:3000;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
    }
    location /api {
        proxy_pass http://localhost:3000/api;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-Ip $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_redirect off;
    }
}
```

Lo siguiente es crear un link simbólico para activar el sitio:

```
$ sudo ln -s /etc/nginx/sites-available/name_domain /etc/nginx/sites-enabled/
```

Comprobar que el archivo de configuración es correcto:

```
$ sudo nginx -t
```

Por último, si la configuración es correcta, solo queda reiniciar Nginx:

```
$ sudo systemctl restart nginx
```

4. Instalar Node.js [87]:

```
$ sudo apt update
$ sudo apt install curl
$ cd ~
$ curl -sL https://deb.nodesource.com/setup_12.x | bash -
$ sudo apt install nodejs
```

Si todo ha ido correctamente, se debe poder comprobar la versión de node:

```
$ nodejs -v
```

Así como la versión de npm:

```
$ npm -v
```

Ahora necesitamos instalar el paquete *build-essential* para poder hacer funcionar ciertos paquetes de npm:

```
$ sudo apt install build-essential
```

5. Instalar PM2 [88]:

   PM2 hace posible ejecutar en segundo plano aplicaciones para que puedan funcionar en segundo plano como un servicio. Se instala mediante npm:

   ```
   $ sudo npm install pm2 -g
   ```

   La opción -g hace que se instale de manera global y sea accesible desde todo el sistema. Es interesante hacer que PM2 se ejecute en el inicio del servidor, para esto escribir lo siguiente:

   ```
   $ sudo pm2 startup
   ```

   Y, por último, iniciar el servicio:

   ```
   $ sudo systemctl start pm2-root.service
   ```

6. Instalar MongoDB [28]:

   Primero hay que importar la clave GPG:

   ```
   $ wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
   ```

   Crear el archivo lista.

   ```
   // Para Debian 10 "Buster"
   $ echo "deb http://repo.mongodb.org/apt/debian buster/mongodb-org/4.4 main" | sudo tee /
       etc/apt/sources.list.d/mongodb-org-4.4.list
   // Para Ubuntu 20 "Focal"
   $ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org
       /4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
   ```

   Instalar los paquetes de MongoDB:

   ```
   $ sudo apt-get update
   $ sudo apt-get install -y mongodb-org=4.4.2 mongodb-org-server=4.4.2 mongodb-org-shell
       =4.4.2 mongodb-org-mongos=4.4.2 mongodb-org-tools=4.4.2
   ```

   Iniciar MongoDB:

   ```
   $ sudo systemctl start mongod
   ```

Habilitamos MongoDB para que arranque de inicio:

```
$ sudo systemctl enable mongod
```

Para consultar los valores de la base de datos ejecutamos en el terminal:

```
$ mongo
> use ezlinguaDB
```

Ahora se pueden ejecutar las consultas que se necesiten, por ejemplo, la siguiente devuelve los documentos de la colección *games* que son de tipo *Puzzle*:

```
> show collections
games
sounds
users
> db.games.find({"type": "Puzzle"})
```

Después de llevar acabo estos pasos, el servidor ya está configurado. Lo siguiente será desplegar la aplicación.

B.1.2.2. Desplegando la aplicación

Nos movemos al directorio donde hemos descargado el proyecto y ejecutamos:

```
$ npm i
$ npm run deploy
```

Si queremos parar el servidor, por ejemplo, para aplicar nuevos cambios, debemos ejecutar lo siguiente:

```
$ npm run kill
```

Después hay que volver a levantarlo.

B.2. Aplicación cliente

B.2.1. Entorno desarrollo

Lo primero instalamos el CLI de Expo (versión 3.17.24 para asegurar compatibilidad con node 12.3.0):

```
npm install -g expo-cli@3.17.24
```

Descargamos el código fuente y nos movemos al directorio que hayamos elegido y ejecutamos:

```
npm i
```

El proceso puede demorarse unos minutos. Una vez acabe ejecutamos:

```
npm start
```

Por defecto, la app apunta a la IP 192.168.1.133:3000 para conectar con el servidor. La opción recomendada es cambiar la IP del pc donde se esté ejecutando el servidor; otra opción es cambiar la IP desde el código de la app en el archivo /app/redux/api/utils.api.js. Si se apunta contra el servidor de desarrollo, todos los dispositivos deben estar conectados a la misma red para que haya comunicación.

Una vez hecho esto veremos un codigo QR tanto en la consola como en el navegador, el cual tenemos que escanear con el dispositivo en el que queremos ejecutar la aplicación. Para poder ejecutar el código durante el desarrollo y así ver los cambios en tiempo real, se necesita instalar una aplicación, que se explica a continuación.

B.2.1.1. Cliente para desarrollo en Android

Se necesita descargar la aplicación Expo Go desde Google play [39]. Una vez descargada, desde la propia aplicación hay que escanear el código QR.

B.2.1.2. Cliente para desarrollo en iOS

Para iOS el proceso cambia un poco. Primero se descarga la aplicación Expo Go desde la App Store [40]. Una vez descargada, hay que escanear el código QR desde la aplicación de Cámara del dispositivo.

B.2.2. Generando build

Antes de proceder a generar los paquetes para los distintos sistemas operativos, hay que tener Expo CLI instalado, como se explica en la sección anterior, y los parámetros de configuración deseados en el archivo /App.json [89]. Además hace falta una cuenta de Expo.io, la cual se puede crear desde la consola una vez ejecutamos el comando *build* o directamente desde la página de Expo [27]. Desde esta página se puede ver el proceso de compilación y además guarda los paquetes generados durante treinta días.

B.2.2.1. Generando APK

Desde el directorio donde tengamos descargado el proyecto, ejecutar:

```
expo build:android
```

La primera vez que se hace una build se muestra el siguiente mensaje:

```
[exp] No currently active or previous builds for this project.

Would you like to upload a keystore or have us generate one for you?
If you don't know what this means, let us handle it! :)

  1) Let Expo handle the process!
  2) I want to upload my own keystore!
```

Si no se tiene keystore, existe la posibilidad de que Expo la genere automáticamente. Si se elige la opción 1, después hay que ejecutar el siguiente comando para poder ver las credenciales y hacer una copia de seguridad de estas.

```
expo fetch:android:keystore
```

Estas credenciales son importantes, ya que sin ellas no podremos actualizar la app en Google Play.

Cuando acabe el proceso se mostará por consola el siguiente mensaje, con el link desde el que hay que descargar el .apk.

```
Build finished.
Successfully built standalone app: [link]
```

B.2.2.2. Generando iOS

Ejecutar desde el directorio del proyecto:

```
expo build:ios
```

En este caso se necesita una cuenta de desarrollador de Apple para poder continuar con el proceso. Al igual que al generar el .apk, también se puede dejar que Expo gestione las credenciales, a elección de cada uno.

B.2.2.3. Generando estáticos para web

Ejecutar desde el directorio del proyecto:

```
expo build:web
```

Este comando genera los estáticos en la carpeta web-build. Sin embargo, para conseguir que la versión web funcione se requieren algunas modificaciones en el código original, ya que hay ciertas incompatibilidades que hacen que no llegue a arrancar.

B.3. Parámetros de configuración

En esta sección se va a hablar de las posibles configuraciones de la aplicación, tanto del backend como del frontend.

B.3.1. Backend

En el Backend hay, por un lado, el fichero */.env*, que establece las siguientes variables de entorno:

- **MONGODB_URL:** URL de la base de datos Mongo.
- **JWT_KEY:** la clave publica que usa la API para la autenticación con JWT.
- **PORT:** puerto en el que se despliega el servidor.
- **API_KEY_PATH:** ruta al fichero JSON con los datos de autenticación de Google Cloud Platform. Ver apartado B.4 para el turorial de cómo generarlo.
- **API_PROYECT_ID:** id del proyecto de Google Cloud Platform.
- **CONFIG_PATH:** ruta al fichero de configuración que se quiere utilizar. Gracias a esta variable de entorno se pueden tener varios ficheros de configuración, y elegir aquí cual aplicar. De este fichero se habla en detalle en el siguiente apartado.

El fichero *.env* no se puede cambiar en caliente, es decir, para que se apliquen los cambios hay que parar y desplegar el servidor. Por otro lado, esta el fichero de configuración */configs/config.default.js*, que si se puede cambiar en caliente, y contiene un objeto JavaScript con los siguientes atributos:

- **ASR:** número de palabras que se quiere que devuelva el ASR.
- **usersPath:** ruta donde se guardará la información de los usuarios (logs y audios).
- **filesPath:** ruta donde están almacenados los ficheros JSON con las listas de palabras, con formato de nombre *nombreLista_códigoPaís_códigoRegión.json*. Los códigos con el estándar BCP-47 [78].

También en este archivo se puede configurar para cada modo de juego lo siguiente:

- **puntuation:** puntuación por cada acierto en las partidas de ese modo.
- **time:** tiempo de cada ronda en las partidas de ese modo.
- **rounds:** número de rondas en las partidas de ese modo.

Hay dos excepciones, el modo *Exposure*, que solo necesita el parámetro *rounds*. Y el modo *Mixed*, que en lugar del atributo *time*, tiene los atributos *timeProunciation* y *timePerception*, que son, respectivamente, los tiempos para las rondas de pronunciación y percepción en el modo Mixto.

B.3.2. Frontend

En el frontend sólo hay un parámetro de interés en cuanto a configuración se refiere. Está en /app /redux/api/utils.api.js y es la URL a la que apunta para las conexiones con el backend. La variable se llama *baseURL* y cada vez que se cambie hay que generar los paquetes de nuevo y actualizarlos en la tienda correspondiente.

B.4. Generar key para la API STT de Google

En esta sección se explica, paso por paso, como obtener una key para las APIs de Google Cloud Platform, y como activar, en este caso, la API Speech-to-Text. Antes de empezar, lo primero que hay que hacer es crear una cuenta de Gmail, si no se tiene una. Una vez se dispone de la cuenta hay que acceder a la consola de Google Cloud Platform [82]. En este caso se va a crear una cuenta de prueba, que permite 3 meses de uso con un crédito de 300$. Una vez se ha creado la cuenta en Google Cloud Platform, hay que acceder a la consola [90].

1. Entrar en la sección **APIs y servicios**. La barra de búsqueda es de gran ayuda para moverse por la consola, ya que tiene multitud de secciones, y de primeras, puede resultar muy compleja.

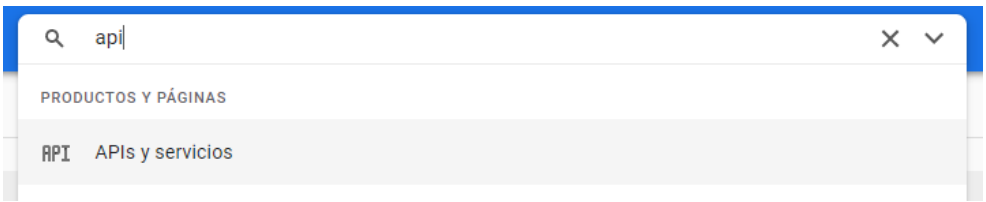

Figura B.1: Paso 1.

2. Entrar en el apartado de **Credenciales** y pulsar **Crear credenciales**.

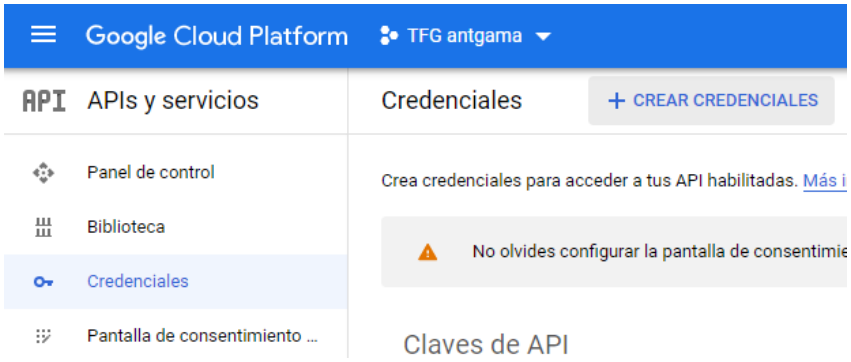

Figura B.2: Paso 2.

3. Elegir **Cuenta de servicio**.

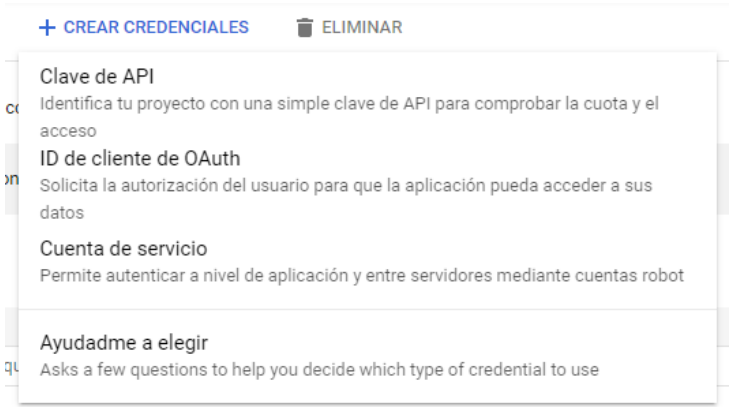

Figura B.3: Paso 3.

4. Escribir un nombre y una descripción y dar a **Crear**. El ID se genera automáticamente. En esta pantalla no hace falta rellenar nada más.

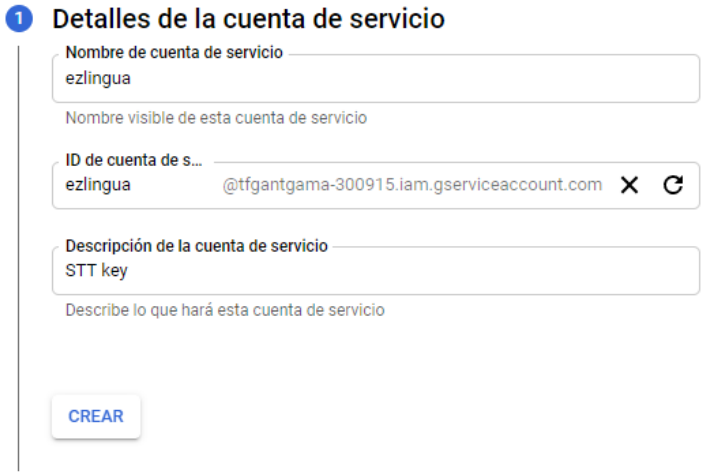

Figura B.4: Paso 4.

5. Volver al apartado de **Credenciales** y buscar, en **Cuentas de servicio**, el que se ha creado. Pulsar encima para entrar a los detalles.

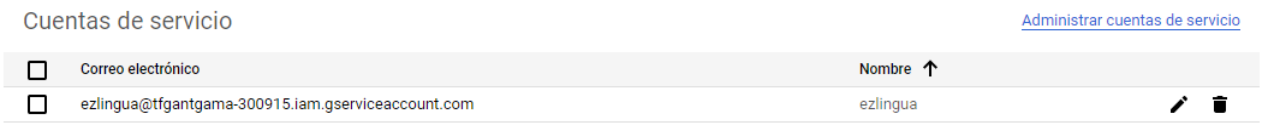

Figura B.5: Paso 5.

6. Donde pone **Claves**, hay que pulsar **Añadir clave** y, posteriormente, **Crear clave**.

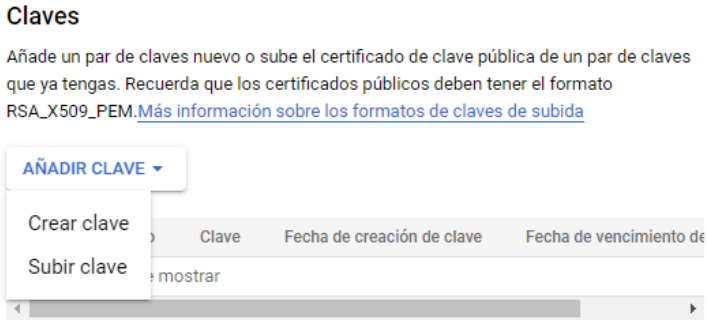

Figura B.6: Paso 6.

7. Para este caso, se necesita la clave en formato **JSON**. Seleccionar éste y dar a crear, después, automáticamente se descargara la clave en nuestro ordenador en un fichero JSON.

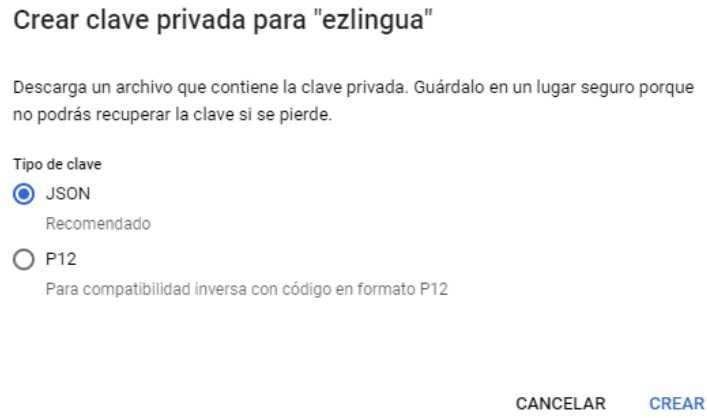

Figura B.7: Paso 7.

8. Ahora, antes de usar la clave, hay que activar la API Speech-to-Text. Para ello, haciendo uso del buscador, abrir en el **marketplace**, **Cloud Speech-to-Text API**.

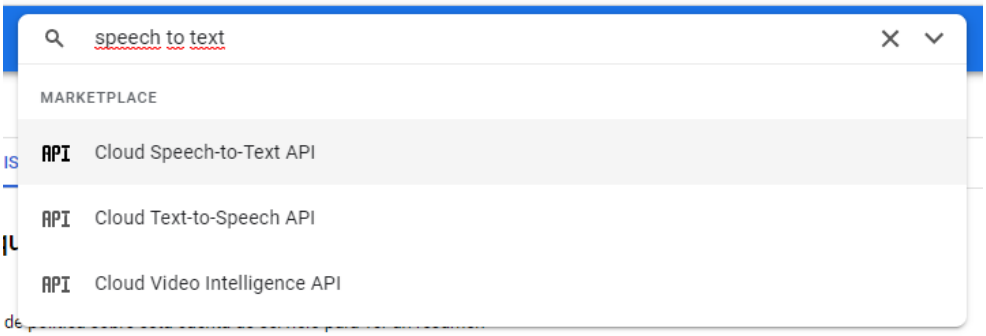

Figura B.8: Paso 8.

9. Por último, elegimos la opción **Habilitar**

Figura B.9: Paso 9.

Una vez se hayan completado todos los pasos anteriores, ya se puede utilizar la clave en el proyecto. Ir al apartado B.3, de este mismo apéndice, para ver la ubicación del archivo.

Apéndice C

Estructura de las API REST

En esta sección se expone la estructura de las diferentes llamadas a la API del backend. Estas pueden ser accedidas desde cualquier cliente.

Para documentarlas se ha utilizado la herramienta Swagger [29], que muestra de una forma clara y atractiva tanto los campos a incluir en la peticiones, como las respuestas esperadas. Las llamadas se ha dividido según el ámbito para el que se utilizan en *masters*, *users*, *languages* y *sounds*.

La mayoría de las llamadas requieren de estar identificado en el sistema, en el Swagger esto se representa mediante un candado que aparece en dichas llamadas.

C.1. Masters

Estas llamadas son genéricas, se podrían usar para distintas aplicaciones sin ningún problema.

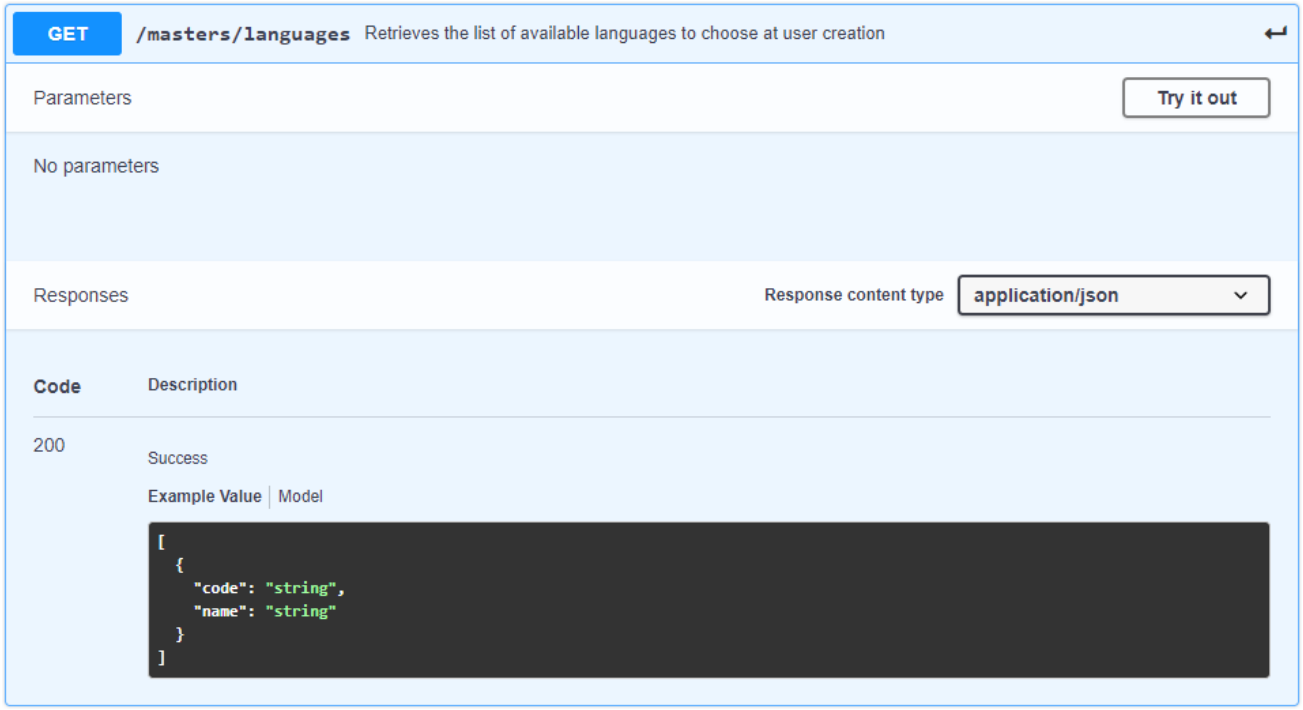

Figura C.1: Estructura API */masters/languages*.

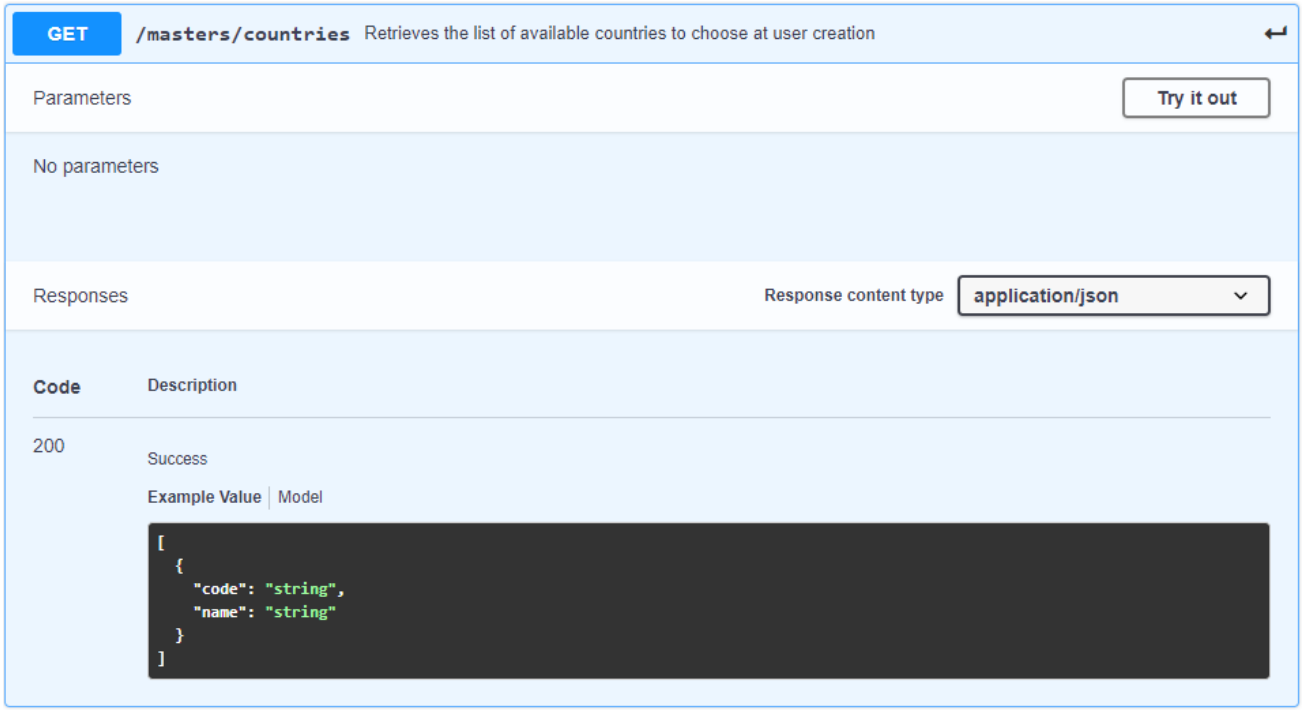

Figura C.2: Estructura API */masters/countries*.

C.2. Users

Llamadas relativas a la gestión de los usuarios.

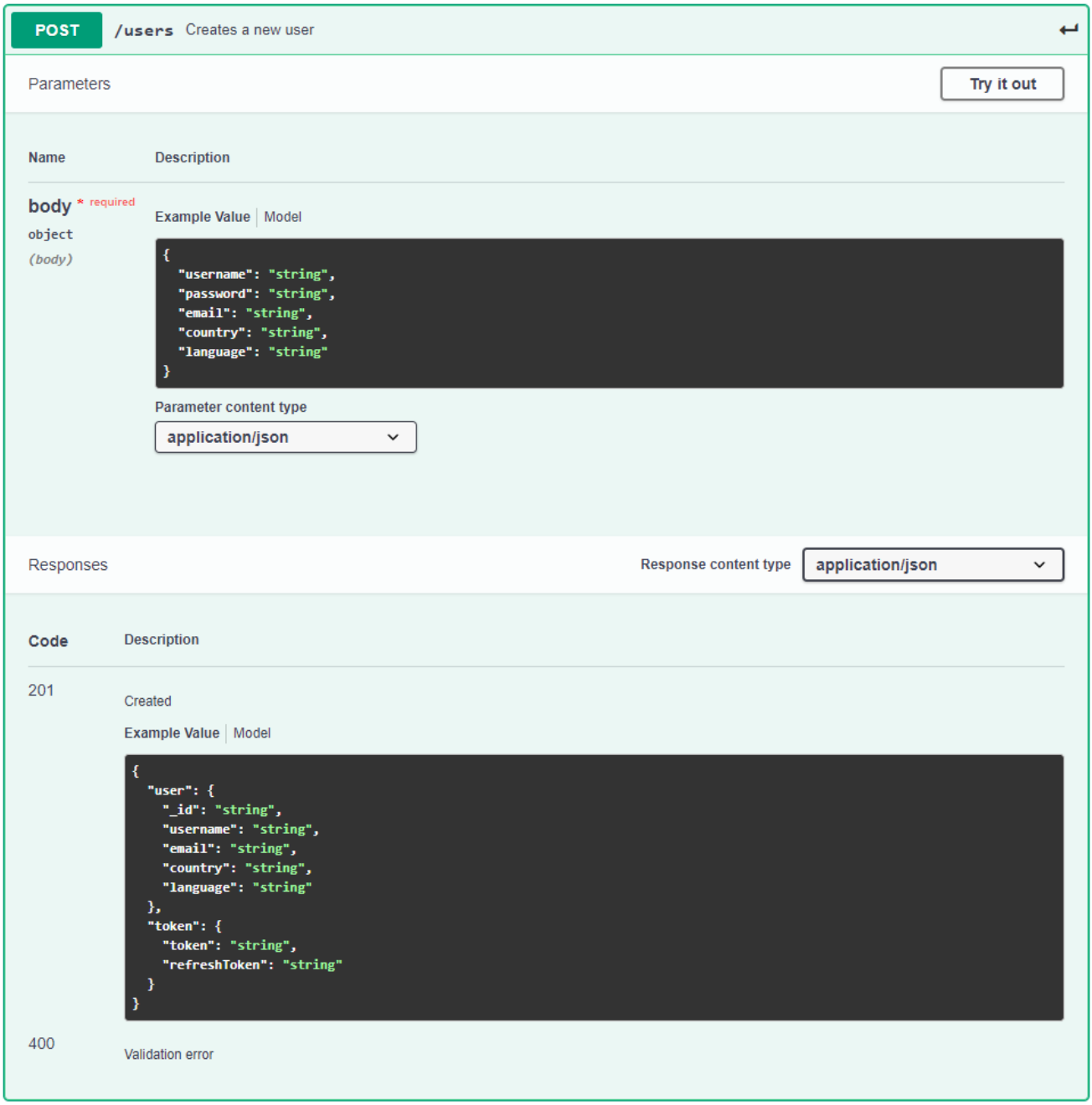

Figura C.3: Estructura API */users*.

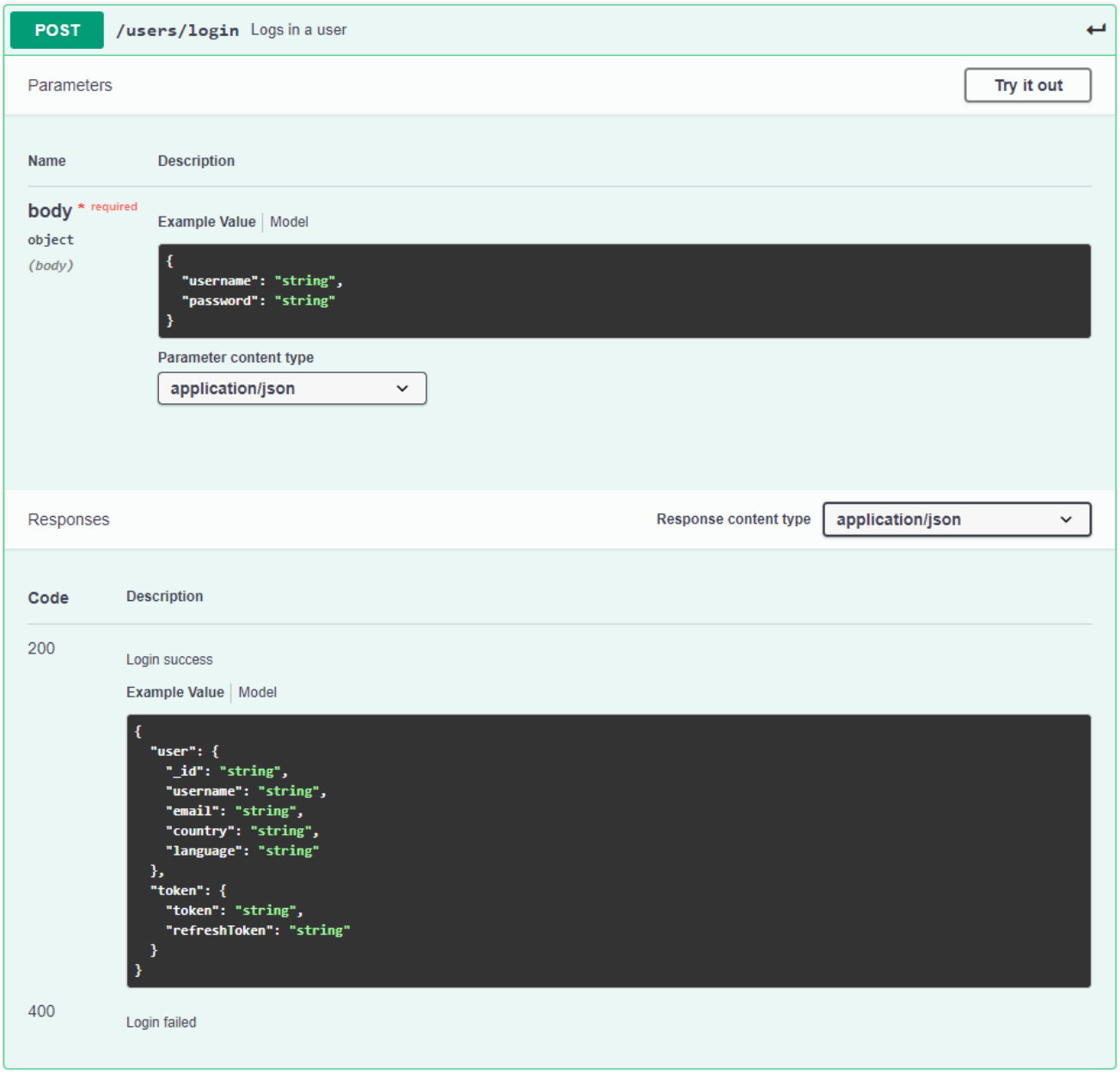

Figura C.4: Estructura API */users/login*.

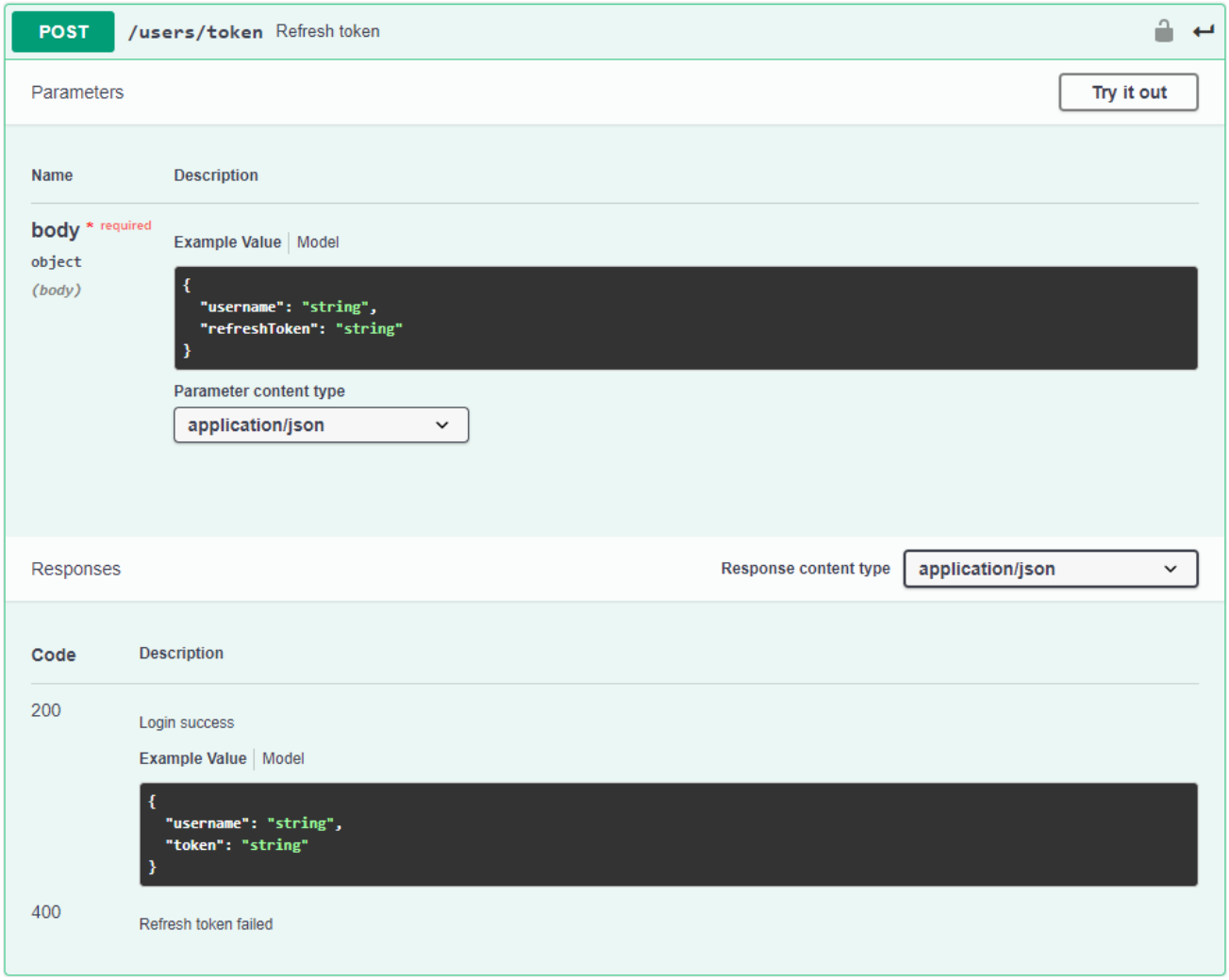

Figura C.5: Estructura API */users/token*.

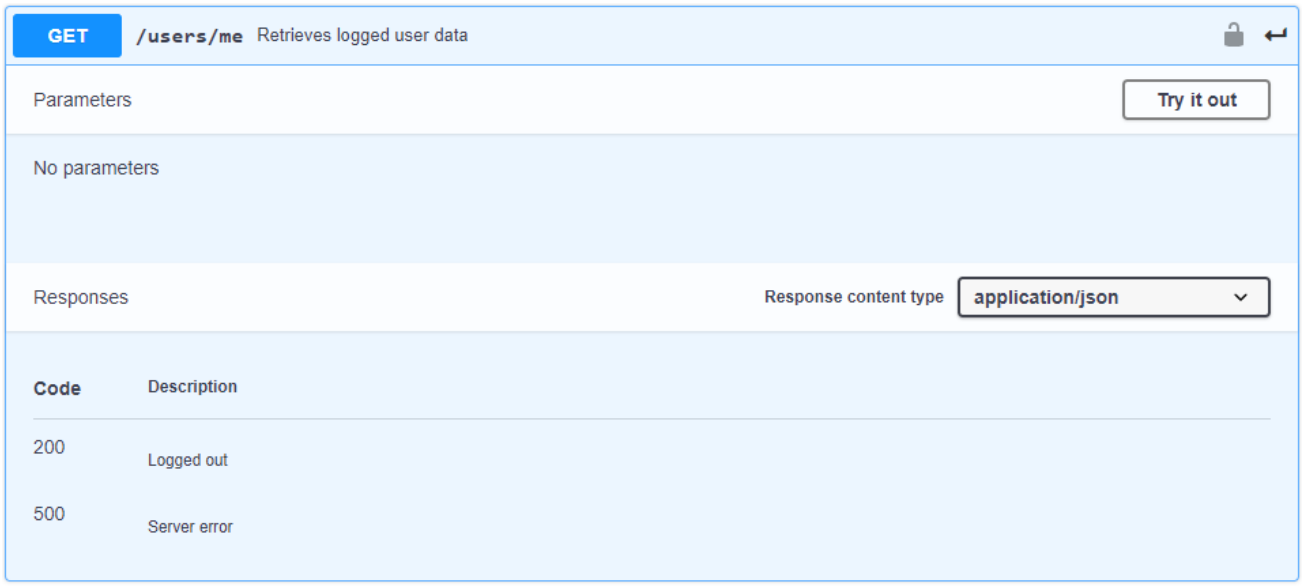

Figura C.6: Estructura API */users/me*.

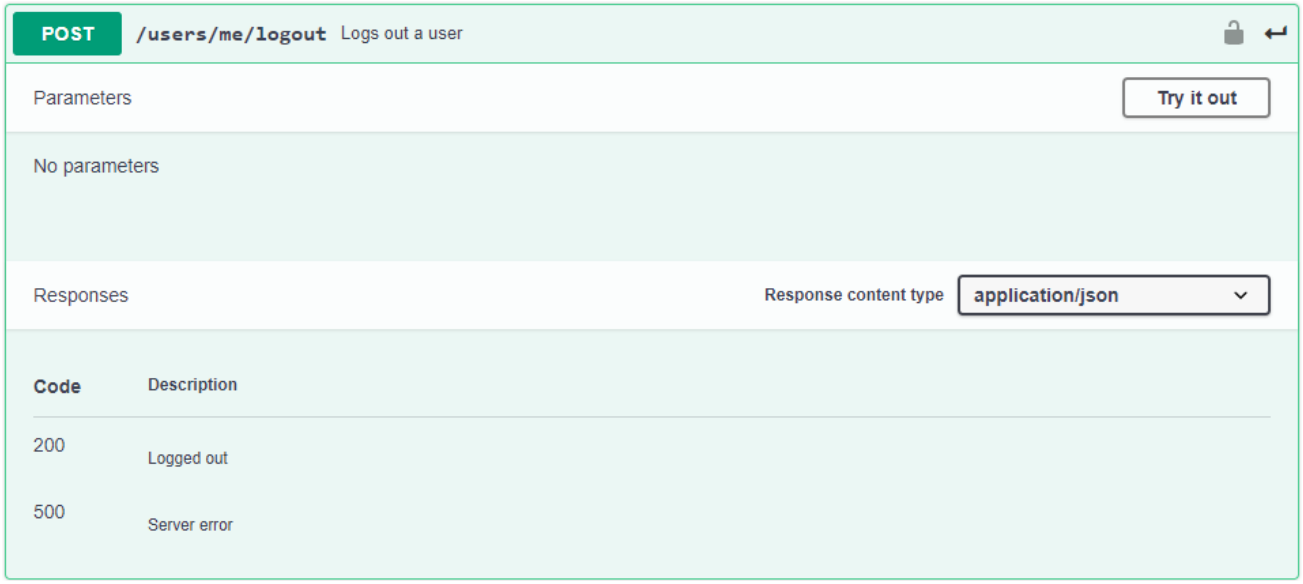

Figura C.7: Estructura API */users/logout*.

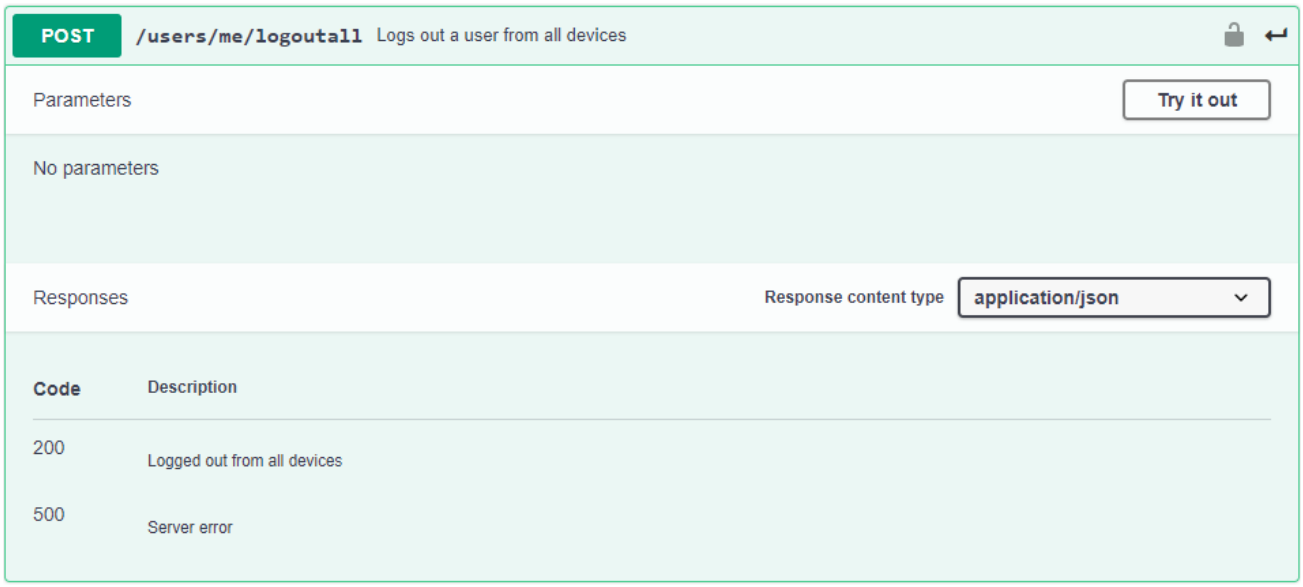

Figura C.8: Estructura API */users/logoutall*.

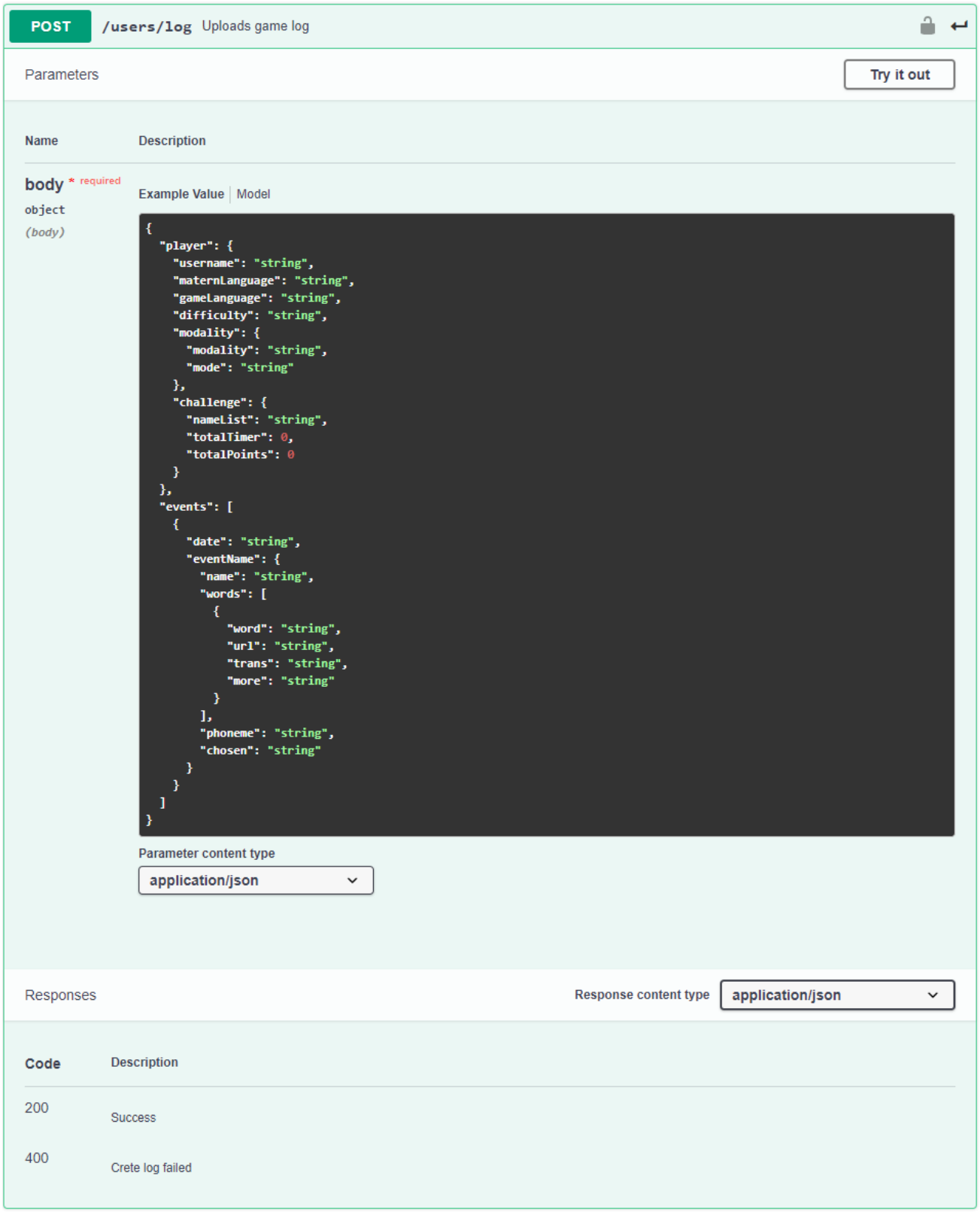

Figura C.9: Estructura API */users/log*.

C.3. Languages

Llamadas para gestionar las listas de palabras.

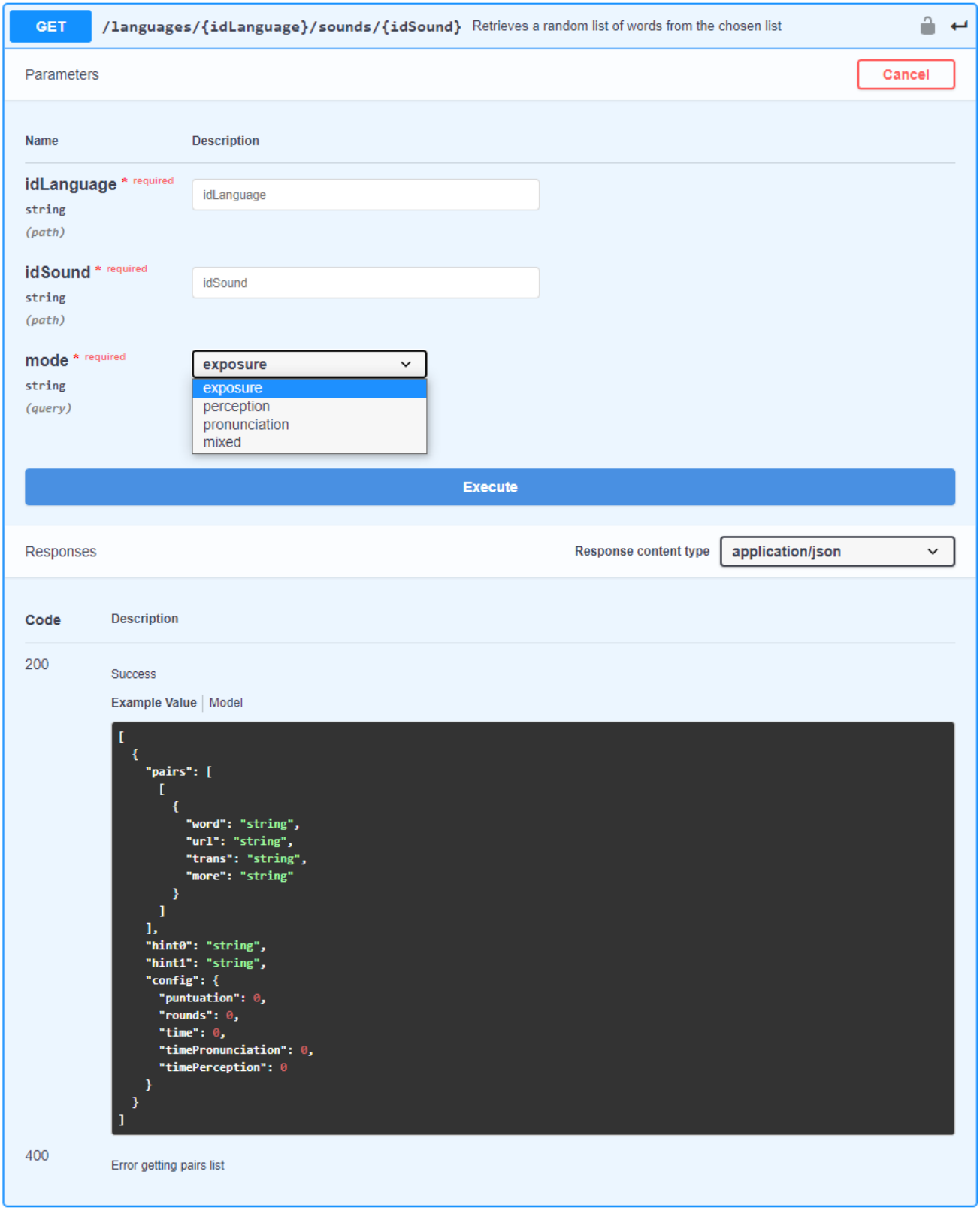

Figura C.10: Estructura API */languages/{idLanguges}/sounds/{idSound}*.

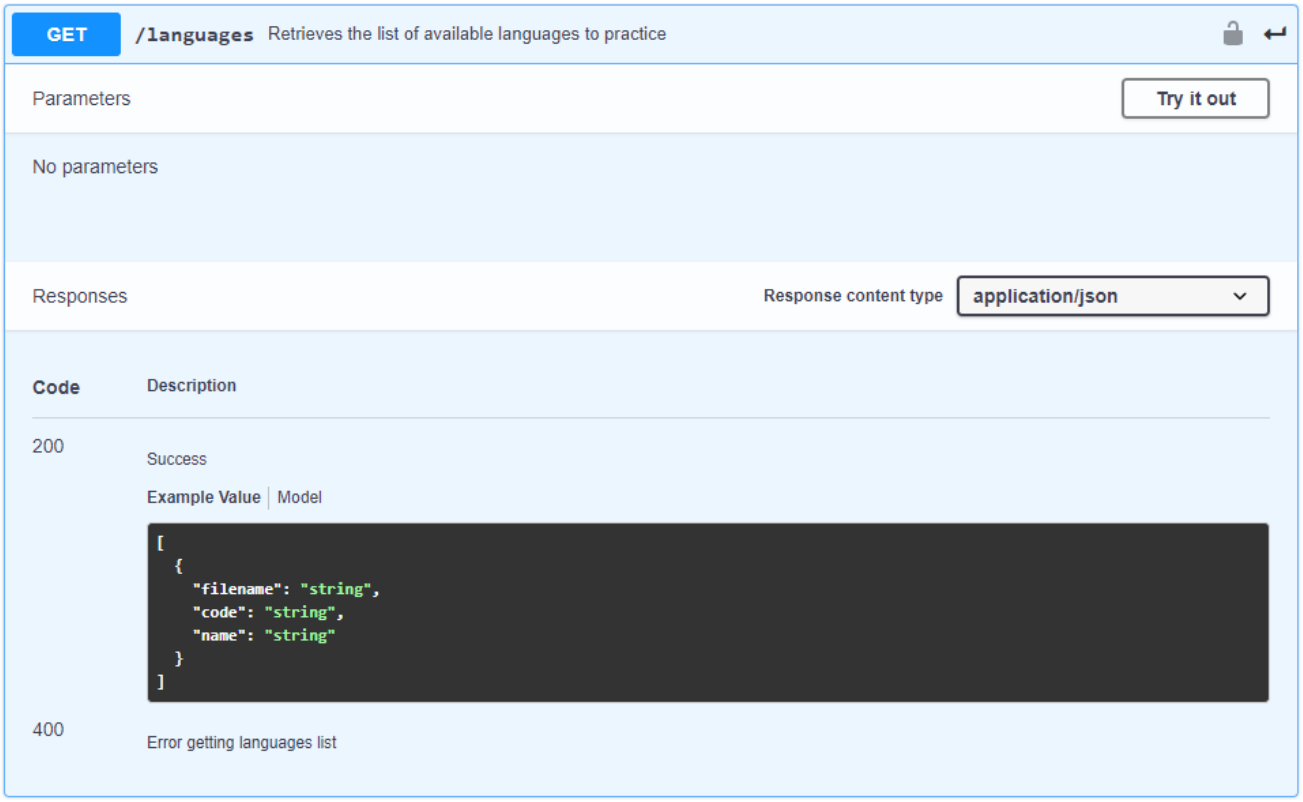

Figura C.11: Estructura API */languages*.

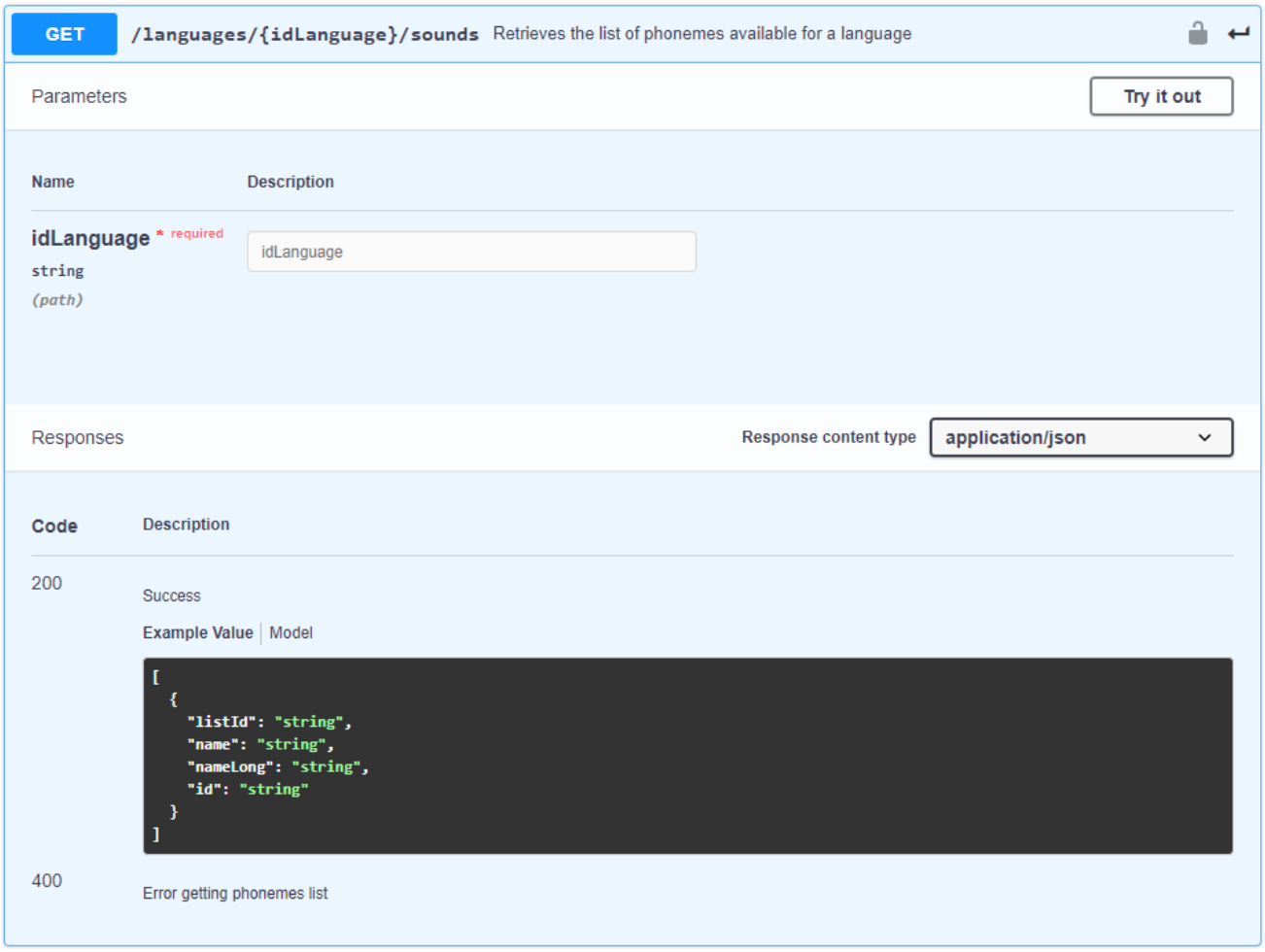

Figura C.12: Estructura API */languages/{idLanguges}/sounds*.

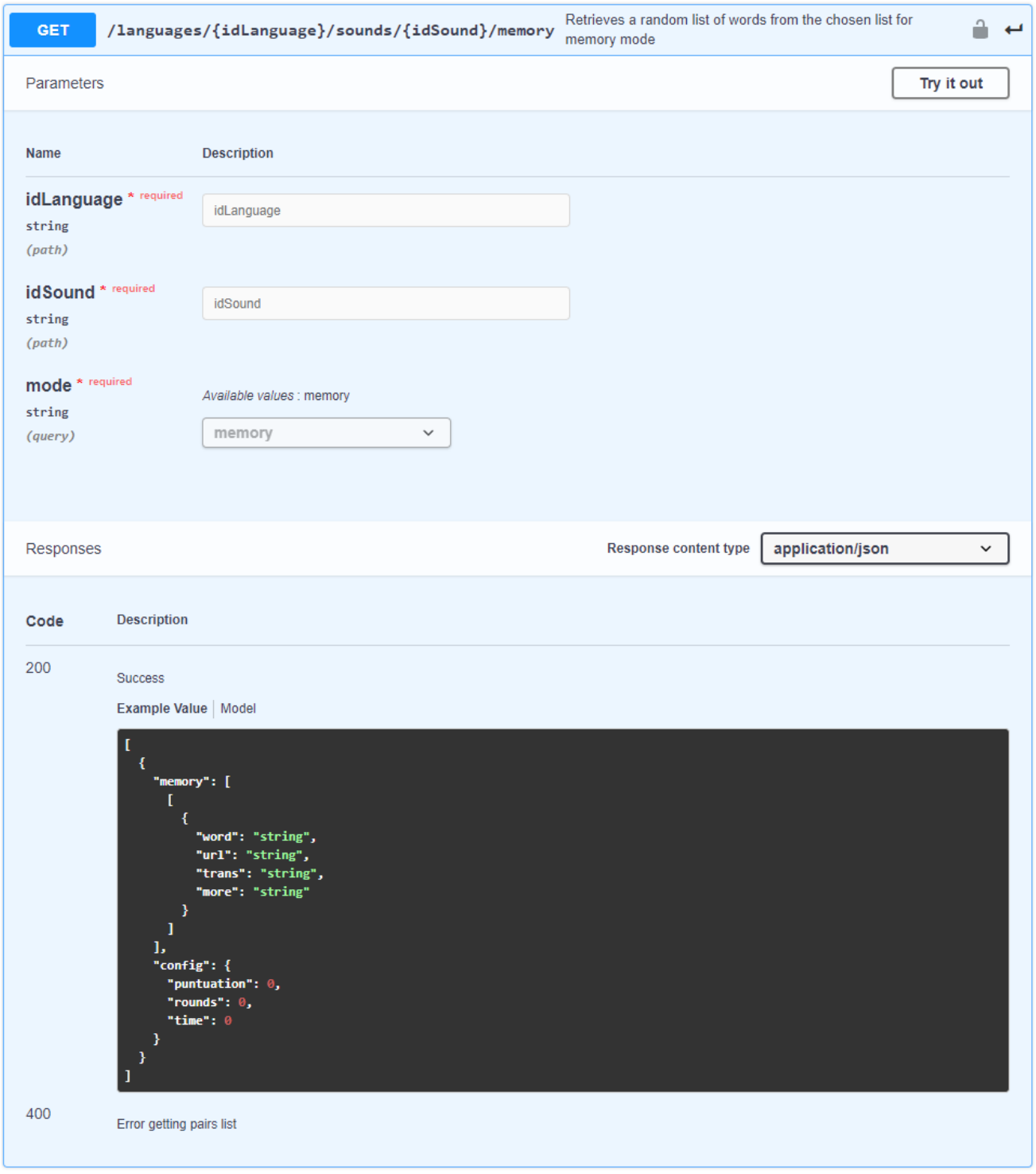

Figura C.13: Estructura API */languages/{idLanguges}/sounds/{idSound}/memory*.

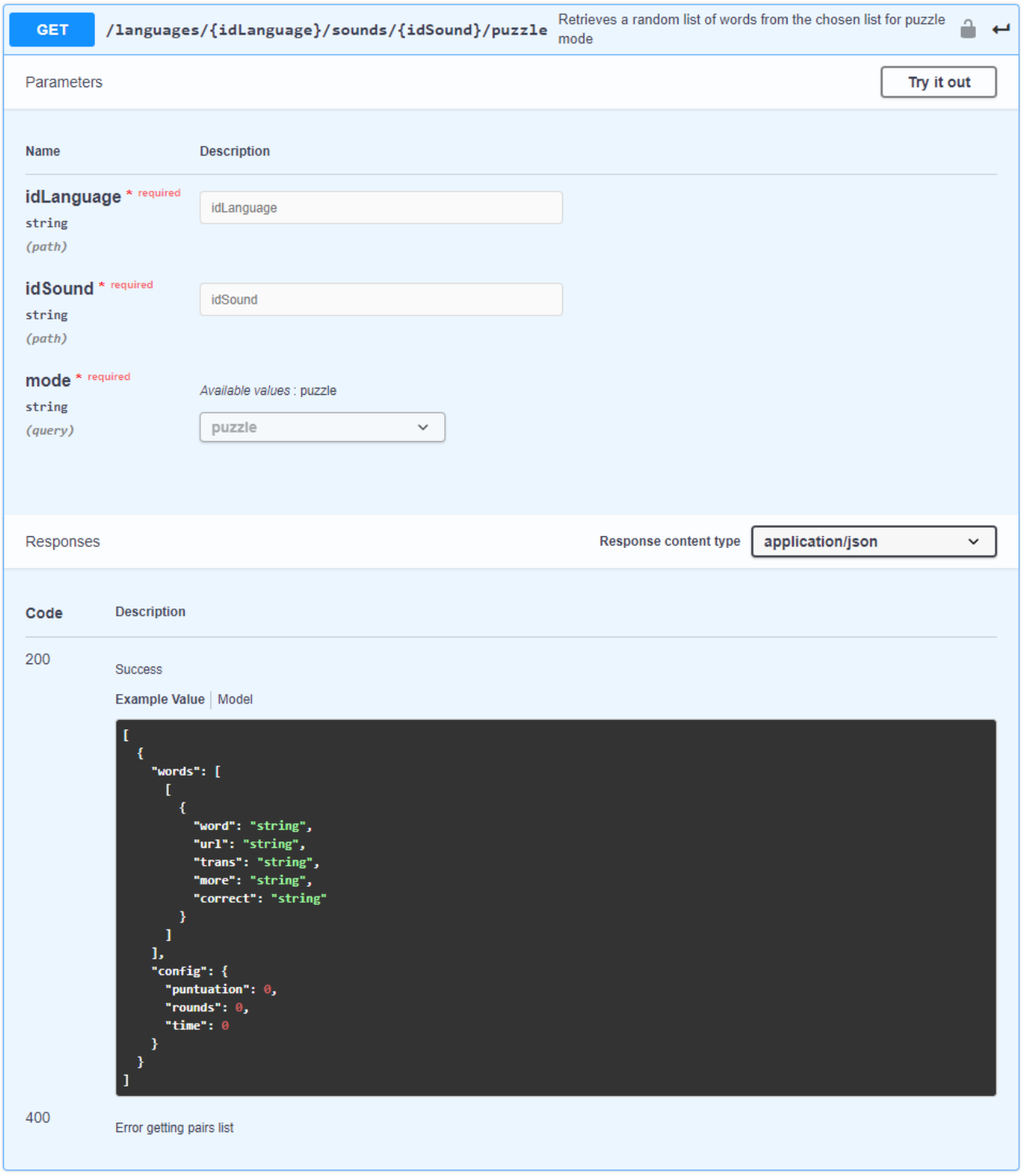

Figura C.14: Estructura API */languages/{idLanguges}/sounds/{idSound}/puzzle*.

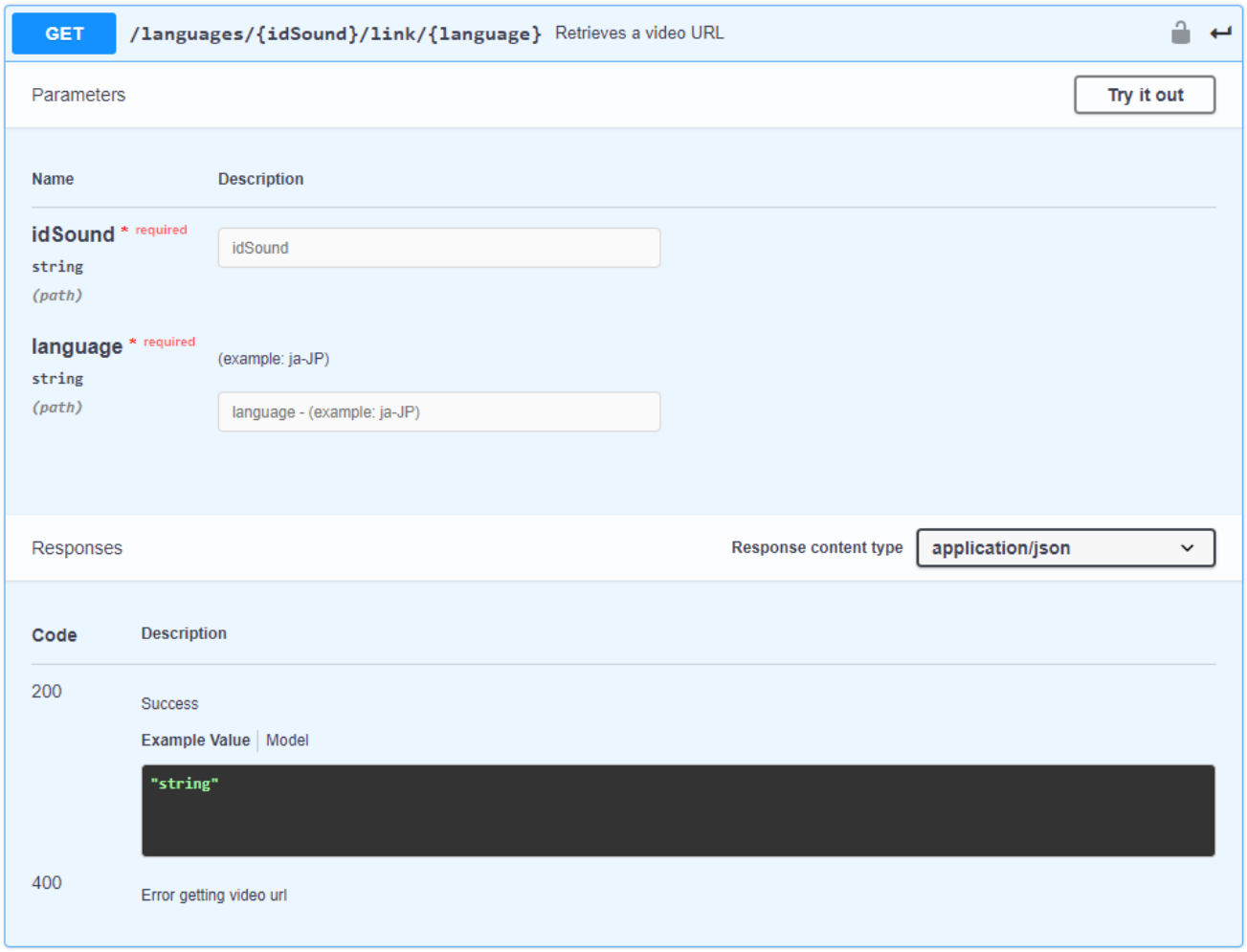

Figura C.15: Estructura API */languages/{idSound}/link/{languages}*.

C.4. Sounds

Llamada para subir y mandar audios a la API de Google.

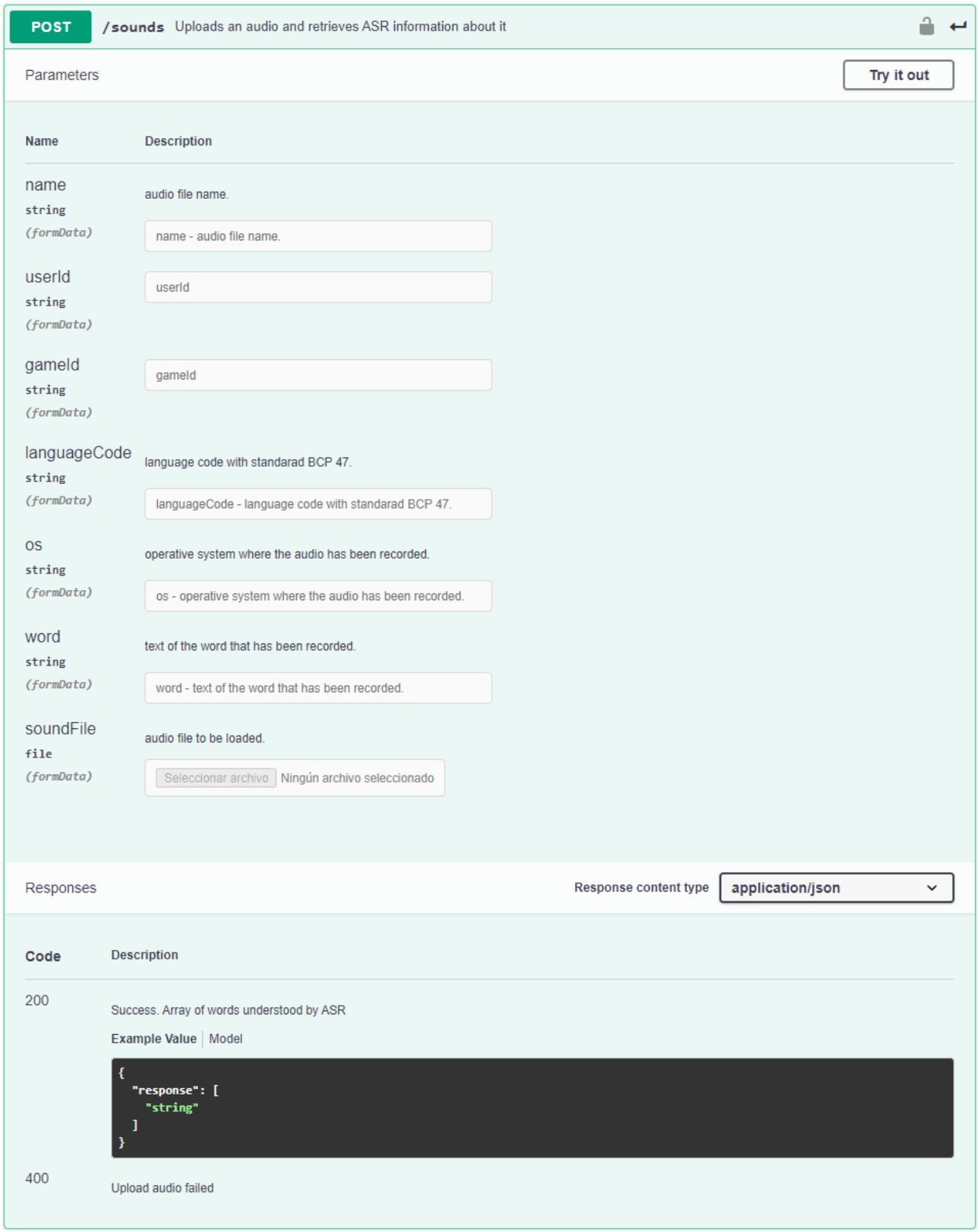

Figura C.16: Estructura API */sounds*.

Apéndice D

Contenido del archivo adjunto

El contenido del archivo adjunto está dividido en varios directorios:

Directorio raíz del .zip

memoria.pdf: Documento con la memoria del proyecto en formato PDF.

app/: Incluye el paquete (*ezlingua.apk*) para ser instalado en un dispositivo Android.

backend/: Incluye el código del *backend* desarrollado durante el proyecto descrito en esta memoria.

frontend/: Incluye el código del *frontend* desarrollado durante el proyecto descrito en esta memoria.

swagger/: Incluye la documentación de la API REST, en formato JSON de Swagger.

Bibliografía

[1] Artículo de facturación de industria de videojuegos. Última visita: 2021-20-01. [Online]. Available: https://gamedustria.com/los-juegos-para-movil-ingresaron-casi-90-000-millones-de-dolares-en-2019/

[2] Aplicaciones móviles educativas. Última visita: 2020-11-19. [Online]. Available: https://gruposolutia.com/aplicaciones-moviles-algunas-claves-para-entender-su-auge-en-los-ultimos-anos/

[3] Ofertas TFG Escuela de Ingeniería Informática de Valladolid. Última visita: 2021-20-01. [Online]. Available: https://www.fi.uva.es/tfg/ofertas.php

[4] Cristian Tejedor García. Aplicación móvil para la mejora de pronunciación multilingüe mediante pares mínimos y gamificación. Abril 2016. Trabajo Fin de Máster, Departamento de Informática, Universidad de Valladolid. Última visita: 2020-08-03. [Online]. Available: http://uvadoc.uva.es/handle/10324/17469

[5] Rafael Sillero Navajas. Desarrollo de la modalidad multijugador para la aplicación Clash of Pronunciation. 2017. Trabajo Fin de Grado, Departamento de Informática, Universidad de Valladolid. Última visita: 2020-08-03. [Online]. Available: http://uvadoc.uva.es/handle/10324/27645

[6] V. Gaitán. Gamificación: el aprendizaje divertido. Última visita: 2020-11-19. [Online]. Available: https://www.educativa.com/blog-articulos/gamificacion-el-aprendizaje-divertido/

[7] E. Cámara-Arenas, *Native Cardinality: on teaching American English vowels to Spanish students*. S. de Publicaciones de la Universidad de Valladolid, 2012.

[8] Andrés de la Maza Valles. API REST para la gestión de partidas multijugador de un juego serio. 2020. Trabajo Fin de Grado, Departamento de Informática, Universidad de Valladolid. Última visita: 2020-08-03. [Online]. Available: http://uvadoc.uva.es/handle/10324/27645

[9] Duolingo. Última visita: 2021-22-01. [Online]. Available: https://play.google.com/store/apps/details?id=com.duolingo

[10] Babbel. Última visita: 2021-22-01. [Online]. Available: https://play.google.com/store/apps/details?id=com.babbel.mobile.android.en

[11] Node.js. Última visita: 2020-12-15. [Online]. Available: https://nodejs.org/es/

[12] V8. Última visita: 2020-12-15. [Online]. Available: https://v8.dev/

[13] Ecma-262. Última visita: 2020-12-15. [Online]. Available: https://www.ecma-international.org/publications-and-standards/standards/ecma-262/

[14] Npm. Última visita: 2019-12-21. [Online]. Available: https://www.npmjs.com/

[15] Express. Última visita: 2020-12-19. [Online]. Available: https://expressjs.com/es/

[16] React native. Última visita: 2020-12-15. [Online]. Available: https://reactnative.dev/

[17] React. Última visita: 2020-12-15. [Online]. Available: https://es.reactjs.org/

[18] Npm. Última visita: 2021-02-02. [Online]. Available: https://phonegap.com/

[19] Apache cordova. Última visita: 2021-02-02. [Online]. Available: https://cordova.apache.org/

[20] Ionic. Última visita: 2021-02-02. [Online]. Available: https://ionicframework.com/

[21] Not just mobile apps. Última visita: 2021-02-02. [Online]. Available: https://news.ycombinator.com/item?id=16198843

[22] Understanding the react native bridge concept. Última visita: 2021-02-03. [Online]. Available: https://dev.to/mfrachet/understanding-the-react-native-bridge-concept-1k90

[23] JavaScriptCore. Última visita: 2021-02-03. [Online]. Available: https://developer.apple.com/documentation/javascriptcore

[24] Java native interface. Última visita: 2021-02-03. [Online]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jni/

[25] Redux. Última visita: 2021-02-04. [Online]. Available: https://redux.js.org/

[26] Flux. Última visita: 2021-01-11. [Online]. Available: https://github.com/facebook/flux

[27] Expo. Última visita: 2020-12-17. [Online]. Available: https://expo.io/

[28] Instalación mongodb en debian. Última visita: 2020-11-20. [Online]. Available: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-debian/

[29] Swagger. Última visita: 2021-01-11. [Online]. Available: https://swagger.io/

[30] Postman. Última visita: 2020-12-15. [Online]. Available: https://www.postman.com/

[31] Google text to speech. Última visita: 2020-12-24. [Online]. Available: https://cloud.google.com/text-to-speech/?hl=es

[32] Google speech to text. Última visita: 2020-12-24. [Online]. Available: https://cloud.google.com/speech-to-text/docs

[33] M. Cotterell and B. Hughes, *Software Project Management 5Th Edition.* McGraw-Hill Education, 2009, ch. 2.

[34] ¿Qué es SCRUM? Última visita: 2021-02-02. [Online]. Available: https://proyectosagiles.org/que-es-scrum/

[35] Esquema desarrollo iterativo e incremental. Última visita: 2021-22-01. [Online]. Available: http://aprendiendocositasdelsoftware.blogspot.com/p/desarrollo-iterativo-y-creciente.html

[36] 5kplayer. Última visita: 2020-12-22. [Online]. Available: https://www.5kplayer.com/index-es.htm

[37] Astah. Última visita: 2019-12-20. [Online]. Available: https://astah.net/

[38] Bitbucket. Última visita: 2020-12-22. [Online]. Available: https://bitbucket.org/

[39] Aplicación cliente expo para android. Última visita: 2020-12-22. [Online]. Available: https://play.google.com/store/apps/details?id=host.exp.exponent&hl=es

[40] Aplicación cliente expo para ios. Última visita: 2020-12-23. [Online]. Available: https://apps.apple.com/es/app/expo-client/id982107779

[41] Cliente git para windows. Última visita: 2020-12-23. [Online]. Available: https://gitforwindows.org/

[42] Google drive. Última visita: 2020-12-23. [Online]. Available: https://drive.google.com/

[43] Marvel. Última visita: 2020-12-24. [Online]. Available: https://marvelapp.com/signin?next=/dashboard/

[44] Interfaz gráfica de usuario para mongo db. Última visita: 2020-12-24. [Online]. Available: https://www.mongodb.com/try/download/compass

[45] Configuración nginx para debian. Última visita: 2020-11-20. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-debian-10

[46] Overleaf. Última visita: 2020-12-24. [Online]. Available: https://es.overleaf.com/

[47] Virtual box. Última visita: 2020-12-16. [Online]. Available: https://www.virtualbox.org/

[48] Skype. Última visita: 2020-12-15. [Online]. Available: https://www.skype.com/es/

[49] Telegram. Última visita: 2020-12-15. [Online]. Available: https://web.telegram.org/

[50] Vysor. Última visita: 2020-12-16. [Online]. Available: https://www.vysor.io/

[51] Visual studio code. Última visita: 2020-12-16. [Online]. Available: https://code.visualstudio.com/

[52] Bcrypt. Última visita: 2020-12-16. [Online]. Available: https://www.npmjs.com/package/bcrypt

[53] Jest. Última visita: 2020-12-19. [Online]. Available: https://jestjs.io/

[54] Json web token. Última visita: 2020-12-19. [Online]. Available: https://www.npmjs.com/package/jsonwebtoken

[55] Mongoose. Última visita: 2020-12-19. [Online]. Available: https://mongoosejs.com/

[56] Nodemon. Última visita: 2020-12-18. [Online]. Available: https://www.npmjs.com/package/nodemon

[57] Supertest. Última visita: 2020-12-18. [Online]. Available: https://www.npmjs.com/package/supertest

[58] Axios. Última visita: 2020-12-17. [Online]. Available: https://www.npmjs.com/package/axios

[59] i18n-js. Última visita: 2020-12-24. [Online]. Available: https://www.npmjs.com/package/i18n-js

[60] Ramda. Última visita: 2020-12-24. [Online]. Available: https://ramdajs.com/

[61] Overleaf. Última visita: 2020-12-24. [Online]. Available: https://bit.ly/3sNajtU

[62] Licencia desarrollador android. Última visita: 2020-12-26. [Online]. Available: https://play.google.com/console/signup

[63] Licencia desarrollador ios. Última visita: 2020-12-26. [Online]. Available: https://developer.apple.com/es/support/compare-memberships/

[64] Precios api stt. Última visita: 2021-02-15. [Online]. Available: https://cloud.google.com/speech-to-text/pricing?hl=es

[65] Tutorial expo. Última visita: 2019-10-20. [Online]. Available: https://docs.expo.io/get-started/installation/

[66] React native vs ionic. Última visita: 2020-12-28. [Online]. Available: https://www.npmtrends.com/react-native-vs-ionic

[67] Tutorial node, mongo, mongoose, jwt. Última visita: 2019-10-21. [Online]. Available: https://medium.com/swlh/jwt-authentication-authorization-in-nodejs-express-mongodb-rest-apis-2019-ad14ec818122

[68] Mern stack. Última visita: 2020-12-28. [Online]. Available: https://www.mongodb.com/mern-stack

[69] Primeros pasos TTS. Última visita: 2019-10-30. [Online]. Available: https://cloud.google.com/speech-to-text/docs/?hl=es

[70] Configuración para formato de grabaciones. Última visita: 2019-10-30. [Online]. Available: https://fostermade.co/blog/making-speech-to-text-work-with-react-native-and-expo

[71] React hooks. Última visita: 2021-02-16. [Online]. Available: https://es.reactjs.org/docs/hooks-state.html

[72] Overleaf. Última visita: 2021-02-15. [Online]. Available: https://www.overleaf.com/

[73] rfc4646. Última visita: 2020-10-15. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4646.txt

[74] Estructura proyecto node. Última visita: 2021-17-01. [Online]. Available: https://medium.com/williambastidasblog/estructura-de-una-api-rest-con-nodejs-express-y-mongodb-cdd97637b18b

[75] Blanca Lendoiro Valle. Celifinder: diseño y desarrollo de una aplicación de localización de restaurantes aptos para celíacos. Enero 2019. Trabajo Fin de Máster, Universitat Oberta de Catalunya. Última visita: 2021-17-01. [Online]. Available: http://hdl.handle.net/10609/88125

[76] Documentación de babel. Última visita: 2020-12-19. [Online]. Available: https://babeljs.io/docs/en/

[77] Modelado de datos en una base de datos mongo. Última visita: 2021-18-01. [Online]. Available: https://docs.mongodb.com/manual/applications/data-models/

[78] Estándar bcp-47. Última visita: 2021-02-07. [Online]. Available: https://tools.ietf.org/html/bcp47

[79] Test unitarios. Última visita: 2021-02-07. [Online]. Available: https://medium.com/@wc.testing.qa/test-unitarios-cobertura-y-una-t%C3%A9cnica-muy-chula-e8ac574e9c34

[80] Jest, mock funtions. Última visita: 2021-02-06. [Online]. Available: https://jestjs.io/docs/en/mock-functions

[81] React test renderer. Última visita: 2021-02-06. [Online]. Available: https://es.reactjs.org/docs/test-renderer.html

[82] Google Cloud Platform. Última visita: 2021-02-07. [Online]. Available: https://cloud.google.com/

[83] Typescript. Última visita: 2021-02-07. [Online]. Available: https://www.typescriptlang.org/

[84] React native for web. Última visita: 2021-02-15. [Online]. Available: https://docs.expo.io/workflow/web/

[85] React native for windows 10. Última visita: 2021-02-15. [Online]. Available: https://github.com/microsoft/react-native-windows

[86] Configuración servidor debian. Última visita: 2020-11-20. [Online]. Available: https://www.digitalocean.com/community/tutorials/initial-server-setup-with-debian-10

[87] Configuración aplicación node en debian. Última visita: 2020-11-20. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-debian-10

[88] Lanzar comandos npm desde pm2. Última visita: 2020-11-20. [Online]. Available: https://stackoverflow.com/questions/31579509/can-pm2-run-an-npm-start-script

[89] Contruyendo aplicaciones con expo. Última visita: 2020-11-20. [Online]. Available: https://docs.expo.io/distribution/building-standalone-apps/

[90] Consola de Google Cloud Platform. Última visita: 2021-02-07. [Online]. Available: https://console.cloud.google.com/