Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN INGENIERÍA DE COMPUTADORES

Aplicación de iOS para el aprendizaje de vocabulario de inglés

Estudante: Pablo Maroto Ruiz
Dirección: Carlos Vázquez Regueira

A Coruña, febreiro de 2021.

A mi familia, por estar siempre ahí.

Agradecimientos

A mi tutor, Carlos, por guiarme en el proyecto y darme sus consejos.

Resumen

El objetivo del presente trabajo ha sido la creación de una aplicación móvil iOS para el aprendizaje de vocabulario de inglés de una manera divertida y eficiente. La aplicación se basa en un sistema para realizar sesiones de estudio diarias con *flashcards* y cuestionarios. Además se ha diseñado un algoritmo automático para planificar los repasos de las palabras, basado en técnicas de repetición espaciada y el agrupamiento de palabras en cajas (sistema de Leitner). A mayores, se han generado diversas estadísticas y animaciones para que el usuario pueda ver su progreso y aumente su motivación.

Se ha creado un base de datos inicial para nuestra aplicación. Para ello se ha obtenido un vocabulario con las palabras más importantes y frecuentes del inglés a partir de un script propio. Con este script se integra la información de múltiples fuentes de Wordnet (traducciones, definiciones, sinónimos y ejemplos de uso) para enriquecer dicha base de datos inicial.

Además, la aplicación se puede conectar con APIs de terceros para buscar imágenes, tanto estáticas como animadas (GIFs), y asociarlas con el vocabulario. Para que desde la aplicación se puedan buscar nuevas palabras, se ha implementado y desplegado un servicio web en la plataforma Heroku que puede ser accedido mediante una API propia.

Se han seguido las guías de estilo de la plataforma iOS para el diseño de las interfaces de la aplicación. También se ha creado una arquitectura robusta con buenas prácticas en el diseño de *software*.

Abstract

The aim of the present work has been the creation of an iOS mobile application for learning English vocabulary in a fun and efficient way. The application is based on a system for daily study sessions with flashcards and quizzes. In addition, an automatic algorithm has been designed to plan word reviews, based on spaced repetition techniques and the grouping of words in boxes (Leitner system). In addition, various statistics and animations have been generated so that users can see their progress and increase their motivation.

An initial database has been created for our application. For this purpose, a vocabulary with the most important and frequent English words has been obtained from our own script. This script integrates information from multiple Wordnet sources (translations, definitions, synonyms and usage examples) to enrich the initial database.

In addition, the application can be connected to third-party APIs to search for images, both static and animated (GIFs), and associate them with the vocabulary. In order to allow

the application to search for new words, a web service has been implemented and deployed on the Heroku platform that can be accessed via a proprietary API.

The style guides of the iOS platform have been followed for the design of the application interfaces. A robust architecture has also been created with good practices in software design.

Palabras clave:

- Aplicación iOS
- Vocabulario
- Repetición espaciada
- Gamificación
- WordNet
- Arquitectura Clean

Keywords:

- iOS application
- Vocabulary
- Spaced repetition
- Gamification
- WordNet
- Clean architecture

Índice general

Capítulo 1

Introducción

En este primer capítulo veremos una introducción de lo que va a consistir el proyecto, los objetivos que se pretenden alcanzar, la propuesta que definimos para afrontar los objetivos y por último veremos como se ha estructurado la memoria.

1.1 Motivación

Hoy en día aprender idiomas es muy importante, ya que vivimos en un mundo globalizado dónde es muy útil comunicarse en diferentes lenguas, ya sea para trabajar o para ocio. El sector de empresas que se dedica a la enseñanza de idiomas es amplio y muy competitivo. Hay muchas academias, cursos, libros o aplicaciones para facilitar el aprendizaje, pues no es una tarea sencilla ya que requiere mucho tiempo y dedicación.

Aprender un idioma no es simplemente estudiar un libro de gramática, son necesarias diversas habilidades cómo leer, escribir, escuchar y hablar. Una cosa en común que tienen estas habilidades son el vocabulario, una pata muy importante del aprendizaje de idiomas. En este proyecto nos vamos a centrar en la enseñanza de vocabulario a través de una aplicación móvil en la que los usuarios puedan aprender palabras de forma eficaz y amena. Por simplificar, se ha escogido uno de los idiomas más hablados y que más se aprende en todo el mundo, el inglés.

El vocabulario es fundamental cuándo se aprende inglés, ya que sin su suficiente conocimiento uno no puede entender a los demás o expresar sus propias ideas. Si bien no podríamos decir que aprendiendo vocabulario consigamos comunicarnos debidamente, su aprendizaje es vital para dominar y llegar a la fluidez.

Suele ocurrir que las palabras que se aprenden se acaban olvidando con el tiempo si no se repasan, por eso hoy en día existen sistemas que aplican técnicas de repetición espaciada. Estas técnicas se basan en algoritmos de estudio eficiente, de forma que el repaso es progresivo y las palabras se muestran en el momento oportuno antes de que se olviden.

El estudio de largos listados de palabras puede llegar a ser monótono y poco motivador

para el estudiante, por lo que es importante conseguir que el estudio sea lo más interactivo posible. Esto se puede conseguir a través de gamificación, existen muchas aplicaciones en el mercado con el propósito de enseñar jugando. Además para recordar mejor las palabras es útil asociarlas con imágenes de la vida real, escuchar cómo se pronuncian y también como se emplean en frases.

Otro punto negativo en la enseñanza de vocabulario es agrupar las palabras por temáticas. Es decir, no todas las palabras de un idioma son iguales, algunas salen mucho más que otras. Por esta razón es mucho más interesante poner el foco primero en las más frecuentes. Para la mayoría de los idiomas sabiendo las 1000 palabras más frecuentes se consigue cubrir casi el 80% de la mayoría de las conversaciones.

De esta forma, este proyecto trata de mejorar los aspectos negativos en el aprendizaje de vocabulario, creando una aplicación con la que el usuario pueda aprender y divertirse al mismo tiempo, todo esto de una forma eficiente gracias a la repetición espaciada.

1.2 Objetivos

Los principales objetivos que se esperan conseguir en este proyecto son los siguientes:

- Aprendizaje de vocabulario de inglés a través de una aplicación móvil.
- El usuario aprenderá las palabras más frecuentes del inglés.
- Gamificación del estudio.
- Planificación de las lecciones mediante un algoritmo inteligente.
- Una aplicación iOS intuitiva, fácil de usar y coherente con el resto de aplicaciones de la plataforma.

1.3 Propuesta

Para conseguir realizar los objetivos marcados anteriormente, presentaremos a continuación los principales elementos la propuesta inicial:

- Obtendremos un conjunto grande de las palabras más frecuentes del idioma que se quiera aprender.
- Podremos conseguir más información sobre ese conjunto inicial de palabras a partir de una librería que nos será útil llamada *Wordnet*. Recopilaremos por ejemplo traducciones, definiciones, ejemplos de uso, ...

- Una vez que procesemos los datos con la librería, crearemos una base de datos y la añadiremos directamente a una aplicación iOS. Este proceso sólo es necesario realizarlo una vez por cada idioma.
- En la aplicación, el modo de aprendizaje será mediante flashcards y cuestionarios.
- En la aplicación también se aplicarán técnicas de repetición espaciada para planificar los repasos de las palabras y así conseguir que el estudio sea progresivo y minimice los olvidos.
- Para complementar la información que tenemos de las palabras, haremos uso de varias APIs para conseguir imágenes y GIFs e incorporarlas a la aplicación iOS.
- Por último crearemos un servicio web para que el usuario pueda buscar más palabras como si fuese un diccionario y pueda añadirlas a su lista de estudio.

A la aplicación iOS la hemos bautizado con el nombre de ***Vocabulary***. Por lo que a lo largo de la memoria haremos referencia a la aplicación con este nombre.

1.4 Estructura de la memoria

El desarrollo de este proyecto se seguirá a través de 9 capítulos detallados a continuación.

1. **Introducción**: Se plantea el proyecto con las motivaciones y la propuesta inicial que se va a resolver.
2. **Estado del arte**: Se analizan otras alternativas parecidas a la solución propuesta.
3. **Fundamentos**: Se hace una introducción teórica a los conceptos o técnicas usados a lo largo del proyecto.
4. **Tecnologías**: Se listan las herramientas que han sido necesarias para implementar el proyecto.
5. **Análisis**: Se analiza el dominio del problema para obtener los requisitos e historias de usuario que implementaremos.
6. **Metodología y Gestión**: Se expone como se ha seleccionado y aplicado la metodología Scrum para la gestión del proyecto, junto a la planificación, el seguimiento y los costes finales.
7. **Diseño**: Se muestran las decisiones de diseño que se han tomado para construir los componentes del sistema y su arquitectura.

8. **Implementación**: Se documentan los detalles técnicos más importantes de la elaboración del proyecto.

9. **Conclusiones**: Se cierra la memoria con las conclusiones, lo que se ha aprendido, y el trabajo futuro para seguir evolucionando el proyecto.

Capítulo 2

Estado del arte

Existen muchas aplicaciones móviles en el ámbito del aprendizaje de idiomas. En este capítulo analizaremos algunas de las aplicaciones más relevantes, *Duolingo* y *Bright*, las cuales han sido útiles a la hora de buscar ideas en este proyecto.

2.1 Duolingo

Duolingo es una aplicación móvil para el aprendizaje de idiomas en las plataformas iOS y Android [1]. Hay una versión gratuita que permite un uso normal de la aplicación con anuncios y con algunas funcionalidades bloqueadas, que sólo están disponibles en la versión de pago.

Con Duolingo es posible aprender distintos idiomas como inglés, alemán, francés y portugués. Su propuesta se centra en mostrar un contenido atractivo y divertido para el aprendizaje. Dispone de diferentes tipos de juegos para aprender gramática, vocabulario, pronunciación e incluso *listening*.

La forma en que se organiza el contenido y las lecciones de estudio funciona por temáticas de menor a mayor nivel de dificultad. La aplicación te informará cuando tengas que hacer repasos y así no olvidar las lecciones aprendidas. En la figura 2.1 se pueden ver varias de las pantallas de la aplicación.

Duolingo es una aplicación muy completa y que funciona muy bien, aún así creemos que no es para cualquier tipo de usuario. Puede ser que haya usuarios que sólo les interese aprender o mejorar su vocabulario y no el resto de características. También la agrupación del contenido por temáticas puede ser más atractiva pero menos eficiente que aprender primero el vocabulario más frecuente. Además, no ofrece la posibilidad de buscar y añadir nuevo vocabulario.

Hemos investigado un poco los resultados obtenidos por Duolingo en cuanto a descargas y beneficios en el último mes. Para ello se ha usado la web SensorTower que permite obtener

Figura 2.1: Diversas capturas de pantalla de la aplicación *Duolingo*.

datos estadísticos estimados de las apps [2]. En la figura 2.2 se puede ver un análisis comparativo entre las versiones de Android (*GooglePlay*) e iOS (*iTunes*) para el mes de Enero del 2021.

Se puede apreciar claramente en este estudio que, aunque la plataforma Android obtiene muchas más descargas, los beneficios de iOS son mucho mayores. Esto es así porque aunque Android tiene una cuota de mercado más alta y por tanto más usuarios, los usuarios de iOS son más propensos a gastar en su tienda. Un análisis similar se daría en la mayoría de aplicaciones que se encuentran en ambas plataformas. Por lo tanto, se obtiene un mayor retorno de la inversión realizada.

El mejor rendimiento de la inversión es uno de los motivos por el cual consideramos que es más interesante desarrollar en iOS que en Android. Aunque el número de usuarios sea, apriori, más reducido.

2.2 Bright

Esta otra aplicación llamada en español "*Bright - Aprender inglés*" propone un enfoque diferente que Duolingo.

La aplicación Bright está disponible tanto para iOS como Android. Se especializa sólo en vocabulario y propone aprender 4.000 de las palabras más importantes del inglés usando su sistema inteligente [3]. Este tipo de aplicación se aproxima más a la idea de este proyecto.

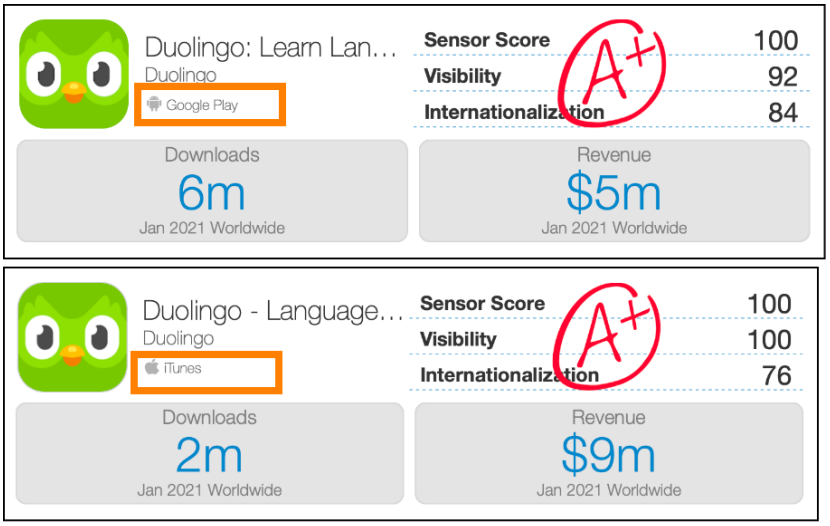


Figura 2.2: Resultados obtenidos con SensorTower de las descargas de Duolingo para Android e iOS

Dispone de diferentes juegos para aprender las palabras y agrupa el vocabulario por temas. Además sólo permite aprender vocabulario en inglés. La única pega que tiene es que se trata de una aplicación de suscripción y la prueba gratuita es de 7 días. La tarifa más barata sería de 30$ cada 2 meses.

Las descargas que ha recibido la aplicación Bright en el último mes según los datos de *Sensor Tower*, han sido de 100.000 descargas. Estos números nos hacen ver que este tipo de aplicaciones son muy demandadas y populares.

Capítulo 3

Fundamentos

En este capítulo hablaremos sobre las bases que sostienen este proyecto. Se ha dividido en tres secciones: Repetición espaciada, WordNet y la Arquitectura VIPER.

3.1 Repetición espaciada

La repetición espaciada es una técnica de memorización utilizada desde hace mucho tiempo. Surgió a partir de unos estudios en 1885 de un psicólogo alemán llamado Hermann Ebbinghaus [4]. Este psicólogo fue pionero en el estudio experimental de la memoria, identificando por primera vez el fenómeno del efecto del espaciamiento en su libro *"Sobre la memoria"* publicado en 1885. Junto con su investigación sobre la repetición espaciada, Ebbinghaus también descubrió la curva del olvido [5], una hipótesis que observaba la disminución de la retención de la memoria a lo largo del tiempo.

La curva del olvido (ver figura 3.1) muestra cómo la información o el conocimiento almacenado en el cerebro se pierde con el tiempo si el individuo no hace ningún intento de retenerlo. Con la información o los conocimientos recién adquiridos, la curva muestra que los seres humanos tienden a reducir a la mitad su memoria en cuestión de días o semanas, a menos que revisen conscientemente el material aprendido. La velocidad de olvido depende también de una serie de factores, como la dificultad del material aprendido, su significado, su representación y factores fisiológicos como el estrés y el sueño.

Para vencer la curva del olvido y aumentar la fortaleza de la memoria existen dos métodos principales, las técnicas mnemotécnicas y la repetición espaciada:

- **Técnicas mnemotécnicas.** Se basan en una mejor representación de la memoria. Crear una canción es quizás la técnica mnemotécnica más utilizada. Un ejemplo de esto es cómo los niños recuerdan el abecedario. Sin embargo, otras técnicas incluyen imágenes, conexiones, modelos, ...

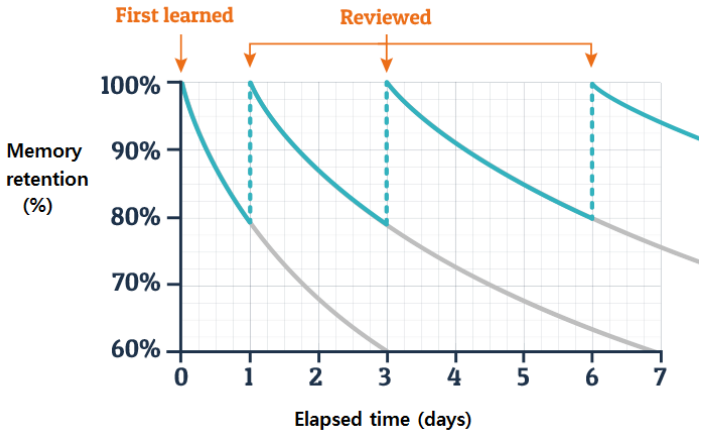


Figura 3.1: Representación de la *Curva del olvido* según los momentos de repaso en el estudio.

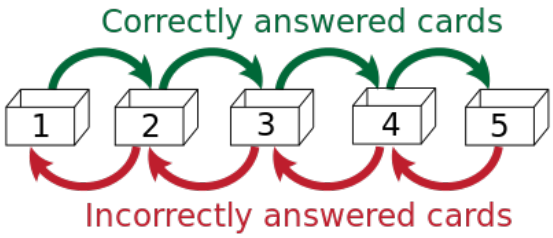


Figura 3.2: Diagrama del proceso de aprendizaje del *Sistema de Leitner*.

- **Repetición espaciada.** Es la repetición basada en la memoria activa. Al revisar la información aprendida separando las sesiones en el tiempo, hace que los elementos sean más fáciles de recordar. Esto se logra mediante una presentación espaciada en lugar de repetir el estudio en un período corto de tiempo, también conocido como presentación masiva.

Una de las implementaciones más conocidas de la repetición espaciada es el *Sistema de Leitner* [6], el cuál se creó en la década de 1970. Éste consiste en la utilización de cinco cajas en las que se van a almacenar las tarjetas o fichas que contienen la información de estudio (ver figura 3.2). Cada caja tiene una planificación distinta: las tarjetas de la caja uno se tendrán que estudiar todos los días, las tarjetas de la segunda caja cada dos días y así progresivamente con las siguientes cajas. Las tarjetas se mueven hacia una caja superior a medida que se responden correctamente y vuelven a la primera caja si se responden incorrectamente.

El *Sistema de Leitner* utiliza tarjetas físicas (conocidas cómo *flashcards*), pero también existen aplicaciones específicas de tarjetas digitales que se basan en el mismo principio, como Anki [7] o SuperMemo [8]. Estas herramientas son ampliamente utilizadas en el aprendiza-

je de cualquier materia, ya que automatizan el proceso de estudio utilizando algoritmos de planificación más complejos, obteniendo así tiempos de repaso más óptimos.

La única problemática de estas herramientas es que se centran en la opinión del usuario cuando se responde a una pregunta, es decir, es el propio usuario el que tiene que marcar el nivel de dificultad del ejercicio dependiendo de lo que le haya costado recordar la respuesta. Esto hace que el aprendizaje sea más lento, ya que el usuario tiene que pensar y hacer su propia valoración, lo que puede llevar también a engañarse a uno mismo calificándose de una manera poco honesta.

En este proyecto se intenta paliar estas dificultades creando un sistema de repetición espaciada sin necesidad de una valoración personal por parte del usuario y que a mayores incluye técnicas mnemotécnicas, como imágenes, para mejorar la representación de las palabras en la memoria.

3.2 WordNet

WordNet es una base de datos de palabras. Fue creada y es mantenida por el Cognitive Science Laboratory de la Universidad de Princeton [9]. El proyecto comenzó en 1985 y ha continuado creciendo debido a su uso en aplicaciones de inteligencia artificial, especialmente en el campo del procesamiento del lenguaje natural.

Originalmente se desarrolló para el inglés pero han ido surgiendo otros proyectos relacionados para más idiomas. Su principal uso es para desambiguar el significado de las palabras, es decir, asignar el concepto más apropiado (conocido cómo *"synset"*) a un término en un determinado contexto. WordNet está constituido de una red de palabras conectadas por relaciones semánticas, dónde los sinónimos son agrupados en *synsets* con definiciones y ejemplos de uso. Al final se comporta como un diccionario al uso pero con mayores capacidades que veremos a continuación.

Por ejemplo, para las siguientes frases en inglés:

1. "He went home and had pasta."
2. "Then he cleaned the kitchen and sat on the sofa."
3. "A little while later, he got up from the couch."

En procesamiento de lenguaje natural, se intenta encontrar el significado de las frases. Y gracias a WordNet se podrá identificar de las frases anteriores lo siguiente:

1. "*pasta*" es un tipo de comida.
2. "*kitchen*" es una parte de "*home*".

3. "*sofa*" es lo mismo que "*couch*".

La base de datos de WordNet cuenta con 117.000 *synsets* y está compuesta de sustantivos, verbos, adjetivos y adverbios. Dispone de un buscador online [10] donde se pueden buscar y visualizar sus datos. Además, puede ser usado tanto para proyectos de investigación como para proyectos comerciales, ya que tiene una licencia libre. Para que los desarrolladores puedan utilizar Wordnet existen varias librerías para varios de los lenguajes de programación más utilizados como C, java, python, etc...

Por tanto, WordNet será muy útil en el desarrollo de este proyecto, ya que nos va a permitir acceder a una fuente de datos de palabras con una muy buena jerarquización y con contenido de buena calidad. En contraposición existen otras alternativas diferentes para propositos parecidos como son APIs o servicios de diccionario. Hay varías reconocidas en este campo como la API de Oxford [11] o de Merriam-Webster [12], pero éstas cuentan con las siguientes desventajas:

- Estas APIs son de pago. WordNet es gratis.
- WordNet puede descargarse y usarse *offline* o también mediante librerías, en cambio las APIs sólo son accesibles con conexión a *internet*
- WordNet tiene mejores capacidades que un diccionario convencional como hemos comentado anteriormente.

En la sección de diseño explicaremos más a fondo como hemos integrado y utilizado WordNet en nuestro proyecto.

3.3 Arquitectura VIPER

Vamos a ver en esta sección una introducción a los principios de la arquitectura VIPER, la cual se utiliza en el desarrollo de aplicaciones iOS y es en la que nos hemos basado para construir *Vocabulary*.

La arquitectura VIPER sigue el principio de responsabilidad única de SOLID, este principio dice que cada componente tiene que tener un único fin en el sistema. Gracias a esto se conseguirá tener un código mucho más fácil de leer, mantener y modificar en el futuro. Al final VIPER es básicamente una implementación en iOS de *Clean Architecture* [13], término acuñado por Robert C. Martin sobre una arquitectura que divide la lógica del software en distintas capas.

En la figura 3.3 se puede ver el esquema general de la arquitectura **VIPER** con sus diferentes componentes que explicaremos a continuación:

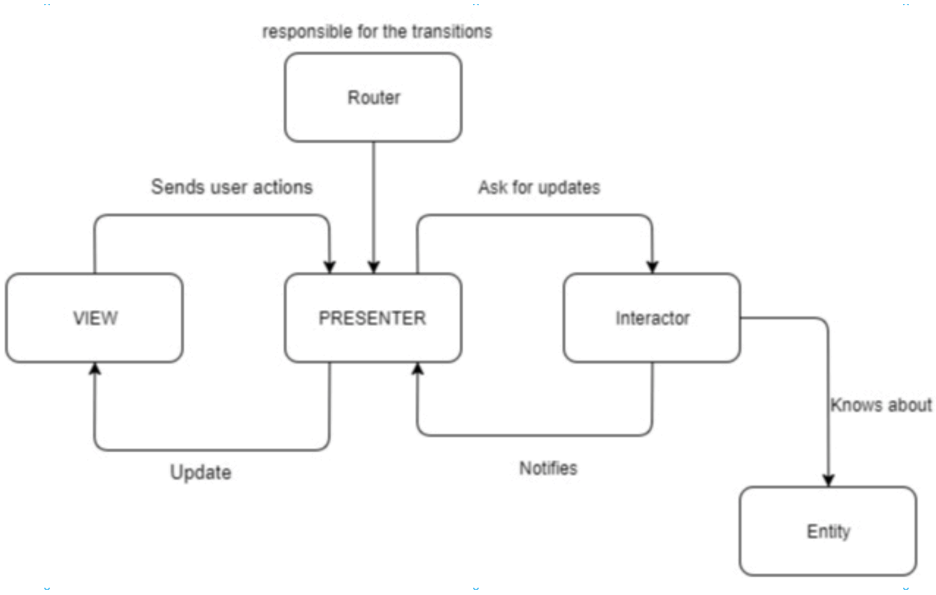


Figura 3.3: Diagrama de una arquitectura VIPER

- **V**iew: En esta capa irá todo lo relacionado con la interfaz de usuario. Será responsable de las vistas, animaciones y de recoger las pulsaciones del usuario para comunicárselo al *Presenter*. Esta capa es totalmente pasiva y no debe tener ningún tipo de lógica.
- **I**nteractor: Contiene la lógica de negocio de los casos de uso. Se harán tareas de preparación y transformación de datos, y será totalmente independiente a la interfaz gráfica. El *Interactor* recibirá los datos de la base de datos o de los servicios web.
- **P**resenter: Es el núcleo de un módulo VIPER, ya que se comunica con casi todos los componentes. Su función es tomar decisiones basadas en las acciones del usuario como hacer una navegación o pedir datos, y también de preparar los datos en el formato que requiere la vista.
- **E**ntity: Son los modelos que serán usados normalmente por los *Interactors*, representan los objetos del dominio de la aplicación que guardarán los datos.
- **R**outer: En esta capa irá toda la lógica de navegación de pantallas.

Algunos de los beneficios de usar la arquitectura VIPER son los siguientes:

- *Claridad*: Responsabilidades únicas
- *Reutilización*: Componentes desacoplados.
- *Menos conflictos*: Archivos más pequeños, lo que producirá menos conflictos en el control de versiones.

- *Testeabilidad*: Facilidad de testeo, al separar la UI de la lógica de negocio, por lo que se podrá conseguir una mayor cobertura de código.
- *Corrección de bugs*: Localizar más fácilmente los bugs y los problemas.
- *Extensión*: Será más fácil añadir nuevas funcionalidades.
- *Coherencia*: La estructura de todo el código será similar, por lo que será más fácil de leer y entender por otros.

Capítulo 4

Tecnologías

En este capítulo nombraremos las tecnologías o herramientas más destacables que han sido necesarias para desarrollar este proyecto.

4.1 Lenguajes

Este proyecto se ha desarrollado en dos lenguajes de programación diferentes: Swift para la aplicación iOSd y Python para la implementación de un *script* de procesamiento de palabras.

4.1.1 Swift

Lenguaje de programación utilizado para desarrollar aplicaciones en los dispositivos Apple de todas sus plataformas: iOS, macOS, watchOS y tvOS. Fue creado en 2014 para sustituir a su antecesor Objective-C. Swift [14] destaca por ser un lenguaje seguro y rápido, además de ser Open source.

4.1.2 Python

El lenguaje de programación Python [15] es de propósito general, lo que significa que se utiliza para desarrollar software en diferentes ámbitos. Especialmente para desarrollos del lado del servidor, aplicaciones de escritorio y *scripting*. También es muy popular para tareas de procesamiento de datos e inteligencia artificial, ya que hoy en día existen un abanico muy amplio de librerías para estos propósitos.

4.2 Librerías y frameworks

Además de los módulos estándar de los lenguajes de programación, se han empleado librerías y frameworks adicionales para simplificar la implementación de ciertas funcionalidades.

4.2.1 UIKit

Framework creado por Apple para la construcción y manejo de interfaces en dispositivos iOS. UIKit [16] permite acceder a todos los componentes visuales y de navegación.

4.2.2 Realm

Realm [17] es una base de datos orientada principalmente para móviles y está disponible en los lenguajes Swift, Objective-C, Java, Kotlin y JavaScript. Algunas de sus características más interesantes son las siguientes:

- **Orientada a objetos**: Realm es orientada a objetos, lo que permite abstraer más fácil el modelo de datos en clases y aumentar la velocidad de desarrollo por su simplicidad.
- **Arquitectura reactiva**: Es posible con Realm tener instancias vivas de los objetos de datos, de esta forma los datos estarán actualizados automáticamente en cualquier parte del código y se podrá refrescar la interfaz de usuario rápidamente.
- **Velocidad**: Se trata de una base de datos con la capacidad de hacer consultas de forma muy rápida. En comparación con otras alternativas de base de datos en la plataforma iOS, Realm resulta la ganadora, como se demuestra en el gráfico de la figura 4.1.
- **Multiplataforma**: La base de datos se puede compartir entre distintas plataformas, como por ejemplo Android, Escritorio o Web. Cada plataforma tiene su propio SDK para gestionar la base de datos y los datos en si estarán almacenados en un fichero Realm, fácilmente exportable e importable.

4.2.3 Charts

Charts [18] es un librería *open source* para iOS que sirve para añadir gráficos estadísticos a la aplicaciones. Ofrece distintos tipos de gráficos altamente configurables.

4.2.4 SwiftConfettiView

SwiftConfettiView [19] es un librería *open source* para iOS que sirve para añadir el efecto visual de confeti en la aplicaciones.

4.2.5 Flask

Flask [20] es un módulo de Python que permite desarrollar aplicaciones web fácilmente. Tiene muchas características interesantes como el enrutamiento de direcciones urls y motor de plantillas.

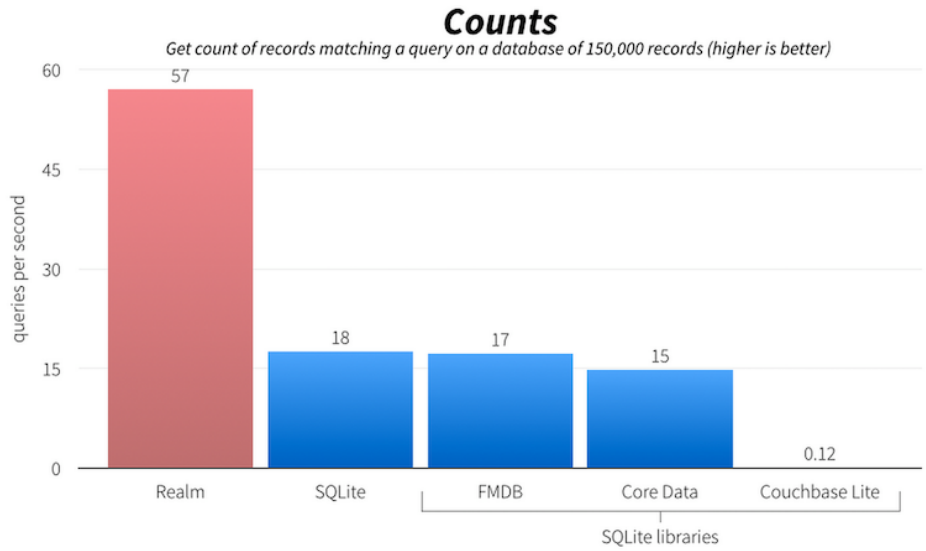


Figura 4.1: Velocidad de lectura entre distintas alternativas de bases de datos en iOS.

4.2.6 Nltk

La librería Nltk [21] de Python da acceso a un conjunto de herramientas para desarrollos relacionados con el procesamiento del lenguaje natural. Ha sido utilizada para acceder a WordNet y extraer información de la lista de vocabulario.

4.2.7 Pandas

Pandas [22] es una librería de Python que sirve para la manipulación de datos y hacer análisis sobre ellos. Se ha utilizado para poder leer y manipular datos en formato *csv*.

4.2.8 Json

Json [23] se trata de una librería de Python que permite trabajar con datos en formato JSON. Ha sido necesario para convertir un tipo de datos al formato JSON.

4.2.9 GoogleTrans

GoogleTrans [24] es una librería de Python que permite obtener traducciones en todos los idiomas a través del propio traductor de Google. En principio obtendremos las traducciones de las palabras al español mediante WordNet, pero hay una pequeña proporción de palabras que no cuentan con traducción. Debido a esto se ha recurrido a la librería GoogleTrans para obtener las traducciones en estos casos.

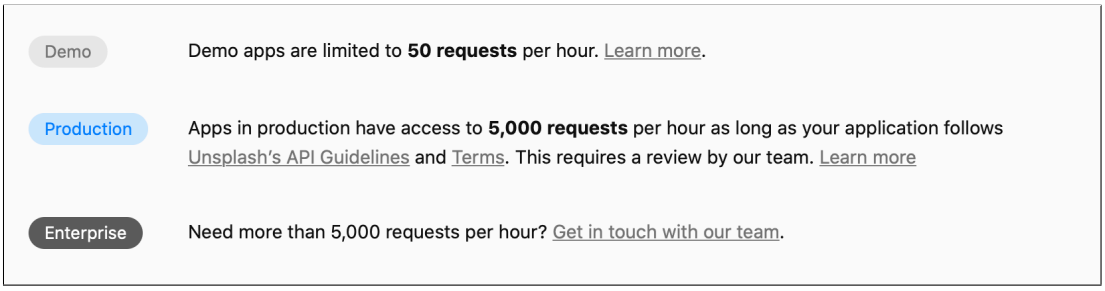


Figura 4.2: Tipos de cuenta de desarrollador de la API de Unsplash.

4.3 APIs

En este proyecto se han empleado dos APIs específicas para buscar imágenes.

4.3.1 Unsplash API

La API Unsplash [25] permite acceder a un banco de 2,4 M imágenes de alta calidad subidas por fotógrafos. La fotografías están libres de derechos de autor por lo que se pueden usar libremente para cualquier uso. Se pueden encontrar imágenes de cualquier temática y están bien categorizadas, lo que permite buscar y filtrar por muchos criterios. Actualmente esta API la integran conocidas empresas de tecnología en sus aplicaciones como pueden ser Trello, Google Slides, Adobe, etc...

Se utilizará una cuenta *Demo* de la API, se puede ver en la figura 4.2 los tipos de cuenta existentes. Este tipo de cuenta permite hacer hasta 50 peticiones por hora. En el caso que se quiera utilizar la API en producción sería conveniente utilizar otro planes superiores con menos restricciones. Destacar que esta API es gratuita para todas sus tipos de cuenta.

4.3.2 Giphy API

La API Giphy [26] da acceso a un montón de vídeos que se repiten en bucle llamados GIFs. La API permite buscar los GIFs más populares en ese momento, buscar los GIFs relacionados a partir de una palabra y obtener GIFs aleatorios de cualquier temática. Es comúnmente usado en las aplicaciones de mensajería para mandar GIFs y animar las conversaciones.

Usaremos la API en su versión inicial, la cual está limitada a 100 búsquedas al día. En el momento que se quiera utilizar la API en un entorno real, es necesario solicitarlo explícitamente para así obtener las claves de producción y no tener limitaciones. Esta API también tiene un uso gratuito.

4.4 Herramientas de desarrollo

En este proyecto se han empleado dos entornos de desarrollos y un sistema de control de versiones.

4.4.1 Xcode

Xcode [27] es un entorno de desarrollo o IDE para el sistema operativo macOS. Contiene las herramientas necesarias para crear y depurar aplicaciones en iOS.

4.4.2 Spyder

Spyder [28] es un entorno de desarrollo o IDE multiplataforma para el desarrollo de software con Python.

4.4.3 Git

Git [29] es un sistema de control de versiones. Ayuda a mantener un registro de los cambios hechos en el código.

4.5 Otros servicios utilizados

Por último, también se han empleado otros servicios adicionales para desplegar el servicio creado, gestionar las dependencias del código iOS, herramientas colaborativas y herramientas de edición para realizar esta memoria.

4.5.1 Heroku Platform

Heroku [30] es una plataforma como servicio (PaaS) de computación en la nube que soporta distintos lenguajes de programación. Nos va a permitir desplegar una API REST en un contenedor virtual para que ésta pueda ser usada desde cualquier cliente.

4.5.2 Cocoapods

Cocoapods [31] es un gestor de dependencias para proyectos iOS. Permite integrar en nuestro proyecto librerías externas (como Realm, Chart y SwiftConfettiView) y ayudar a mantener sus versiones al día.

4.5.3 Trello

Trello [32] es una herramienta de colaboración para organizar proyectos. Se usan tableros y tarjetas virtuales para la gestión de tareas.

4.5.4 LaTeX

Sistema de composición de textos que facilita la escritura de documentos. Se ha utilizado para redactar esta memoria dado su potencial y versatilidad.

Para la ayuda de la edición con LaTeX se uso el editor colaborativo Overleaf [33], el cual permite compartir en tiempo real un documento LaTeX y de tener un visor para ver el documento compilado.

4.5.5 Draw.io

Draw.io [34] es una aplicación web que sirve para crear todo tipo de diagramas y diseños visuales. Se ha utilizado en esta memoria para la edición de imágenes y la creación de diagramas.

Capítulo 5

Análisis

En este capítulo se exponen los requisitos, tanto funcionales como no funcionales, de la solución propuesta en el proyecto. También se analizan todas las necesidades de los usuarios a los que va dirigido la aplicación *Vocabulary*. Las historias de usuario recopiladas servirán de punto de partida para diseñar e implementar las funcionalidades en las próximas etapas de desarrollo.

5.1 Requisitos funcionales

En el apartado 1.2 se vieron los objetivos principales que se quieren alcanzar en este proyecto. Este tipo de objetivos son a alto nivel, por lo que ahora es necesario definir más profundamente como se tiene que comportar exactamente la aplicación que vamos a construir. Para este propósito están los requisitos funcionales.

Para capturar los requisitos funcionales vamos a usar **historias de usuario**, las cuales consisten en breves descripciones de las funcionalidades que se desean, siempre contadas desde la perspectiva del usuario. La aplicación no contará con diferentes roles de usuario, sólo existirá un único rol, que será simplemente el usuario que quiere aprender vocabulario de inglés.

Las funcionalidades que debería tener *Vocabulary* en forma de historias de usuario son las siguientes:

US01 - Visualizar listado de palabras

El usuario podrá ver un listado de todas las palabras que puede aprender. La palabras disponibles serán una selección amplia de las más frecuentes del inglés (entre 3.000 y 5.000 preferiblemente). A este conjunto de palabras que el usuario puede y quiere aprender le llamaremos *deck*.

US02 - Buscar palabras

El usuario podrá buscar palabras mediante un buscador para poder filtrar entre todas las palabras disponibles.

US03 - Visualizar detalle de palabras

El usuario podrá acceder a la ficha de una palabra para ver todas sus características. Se tendrán que mostrar traducciones, sinónimos, definiciones y ejemplos de uso.

US04 - Añadir imágenes

El usuario dentro de la ficha de una palabra, podrá buscar y escoger una imagen para asociarla con la palabra y facilitar el aprendizaje.

US05 - Añadir GIFs

El usuario dentro de la ficha de una palabra, podrá buscar y escoger una GIF para asociarlo con la palabra y facilitar el aprendizaje. Un GIF es un formato gráfico que permite imágenes animadas.

US06 - Escuchar pronunciación

El usuario podrá escuchar como se pronuncia una palabra pulsando un botón de reproducir dentro de su ficha.

US07 - Cuestionario de palabras

El usuario podrá examinarse de las palabras disponibles a partir de un cuestionario. Se irán mostrando por cada palabra un test con varias respuestas de las que sólo una será la correcta. La pregunta de los cuestionarios será preferiblemente una imagen o GIF, o también podría ser la palabra en español. Si se selecciona una respuesta incorrecta se le indicará al usuario la opción correcta y además, un botón para ir a la ficha por si se quiere revisar la palabra.

US08 - Estudiar lección

El usuario podrá empezar un cuestionario con un número limitado de preguntas. Se podrá ver una barra de progreso para saber el estado de la lección, cuantas preguntas se han contestado y cuantas faltan. También habrá un botón para salir de la lección si se desea.

US09 - Visualizar resultados de la lección

El usuario podrá ver un resumen del resultado de la lección con las palabras que ha respondido correctamente e incorrectamente.

US10 - Estudio programado

El usuario podrá hacer una lección cada día. Las lecciones serán construidas de forma inteligente mediante repetición espaciada. Cada lección estará formada tanto por palabras nuevas, como por palabras de repaso si estuviesen programadas en ese día. Si el usuario responde bien el ejercicio de una palabra, la palabra subirá de nivel y no volverá a repetirse hasta pasado un tiempo. En cambio si se responde incorrectamente el ejercicio, este se volverá a repetir y la palabra bajará de nivel para tener un periodo más corto de repaso.

US11 - Seleccionar nuevas palabras

El usuario podrá seleccionar las palabras que quiere aprender en la lección. Estas palabras serán seleccionadas aleatoriamente de entre las palabras disponibles para aprender. Se establecerá un límite fijo de palabras nuevas por lección. Si al usuario no le interesa aprender alguna palabra se descartarán y no volverán a aparecer. Además se dará la opción de asociar la palabra con una imagen.

US12 - Visualizar pantalla principal

El usuario podrá acceder a una pantalla principal. Esta pantalla se mostrará de inicio en la aplicación y en ella el usuario podrá seleccionar el empezar una nueva lección. Se le informará al usuario cuantas palabras tiene programadas para la lección de ese día.

US13 - Visualizar estadísticas

El usuario podrá ver estadísticas de su aprendizaje. Se dividirá en dos secciones:

- Un resumen general del estado de las palabras. Se agruparán por “no vistas”, “en progreso”, “aprendidas” y “descartadas”.
- Un gráfico circular para la representación de los distintos estados de las palabras que están en progreso.

US14 - Buscar en diccionario

El usuario podrá buscar cualquier palabra en inglés. Si la búsqueda tiene resultados se abrirá su ficha con toda su información.

US15 - Aprender palabras buscadas

Si la palabra buscada en el diccionario no está en el mazo de estudio del usuario, se le dará la opción de añadir la palabra para aprenderla.

US16 - Visualizar búsquedas recientes

El usuario podrá ver las palabras que ha buscado recientemente y seleccionarlas para acceder a su ficha.

US17 - Configurar recordatorio lección

El usuario desde la pantalla de ajustes podrá activar o desactivar una notificación diaria de aviso para realizar la lección. Además se podrá configurar la hora del día en la que desea ser notificado.

US18 - Configurar nuevas palabras

El usuario desde la pantalla de ajustes podrá seleccionar el número de palabras que quiere aprender diariamente.

US19 - Configurar pronunciación

El usuario desde la pantalla de ajustes podrá elegir la pronunciación que le interesa escuchar entre varias opciones:

- Pronunciación inglesa
- Pronunciación americana
- Pronunciación australiana

5.2 Requisitos no funcionales

Por último faltarían los requisitos no funcionales o también conocidos como requerimientos del sistema. Son las cualidades de calidad que se esperan tener del producto. Se han planteado los siguientes requisitos:

- **Usabilidad**: La interfaz debe ser fácil de usar y el usuario tiene que entender fácilmente que hace cada componente.
- **Fiabilidad**: El sistema tiene que funcionar sin fallos y estar siempre disponible.

- **Rendimiento**: Las tareas de acceso a base de datos o de red tienen que ser rápidas, no pueden existir excesivos tiempos de espera.
- **Escalabilidad**: Si aumenta la carga de trabajo por el aumento de usuarios, el sistema tiene que seguir funcionando correctamente.
- **Seguridad**: El sistema tiene ser capaz de evitar en la medida de lo posible amenazas maliciosas externas.

Capítulo 6

Metodología y Gestión

En este capítulo veremos la metodología que hemos aplicado en el proyecto y como la hemos adaptado. Posteriormente mostraremos como ha sido la planificación y el seguimiento, y finalmente analizaremos los costes finales.

6.1 Scrum

Se ha escogido la metodología Scrum como guía de trabajo en este proyecto. La metodología Scrum se basa en la gestión ágil (*"Agile Project Management"*), la cual permite adoptar en los proyectos una planificación y organización flexible para mejorar los resultados de gestión. Varios de los mayores beneficios de Scrum y de otras metodologías que siguen los mismos principios son los siguientes [35]:

- **Mejora la calidad del producto**: El enfoque proactivo de todo el equipo hace que se busque la excelencia del proyecto entre todos. Además, la revisión y mejora continúa de las características del producto, mejoran notoriamente el resultado final.
- **Mayor satisfacción del cliente**: El cliente esta más involucrado a lo largo del proceso de desarrollo gracias a demostraciones y entregas periódicas.
- **Mayor motivación de los trabajadores**: Se promueve una organización horizontal sin burocracia, lo que provoca una mayor productividad del equipo por una mayor autonomía y auto-organización.
- **Mayor flexibilidad**: Las características del proyecto se dividen en unidades pequeñas para permitir una mayor flexibilidad a los cambios. Según la evolución del producto y las necesidades del cliente se podrán integrar las tareas por prioridades en menos tiempo.

- **Mayor predicción de tiempos**: Gracias a la división de tareas se puede estimar mejor su duración, y también ir conociendo la velocidad real y el rendimiento del equipo.
- **Reducción de riesgos**: Los errores se van identificando a lo largo del desarrollo, y gracias la flexibilidad y predicción de tiempos es posible ir cumpliendo las expectativas del cliente.

6.1.1 Roles

Para la puesta en marcha de Scrum es necesario establecer ciertos roles a los diferentes miembros del proyecto, ya que cada rol tendrá responsabilidades diferentes. Los distintos perfiles son los siguientes:

- El **Product Owner** será la persona que represente los deseos del cliente y de los usuarios. Se encarga de fijar los objetivos del proyecto y de priorizar las tareas más importantes. Su objetivo principal es que se completen primero las tareas que den el máximo rendimiento a la inversión. La métrica usada para este aspecto es el ROI (Retorno sobre la inversión), así cuanto antes esté en manos del usuario el valor añadido, antes se producirá la retroalimentación y el ROI.
- El **Scrum Master** es el facilitador o *coach* del equipo de desarrollo. Se encarga de asegurar que el equipo está trabajando de acuerdo a las reglas y procesos de Scrum.
- El **Equipo de desarrollo** es el encargado de realizar el trabajo principal del proyecto. Sus integrantes poseen competencias transversales y son responsable de producir un código completo, probado y documentado.

La metodología Scrum normalmente se usa en equipos numerosos, pero **en nuestro caso**, al contar el proyecto con sólo dos miembros, el tutor y el alumno, ha sido necesario compartir varios de los roles existentes. Las tareas de Product Owner han sido divididas al 50% entre ambos, estas tareas han consistido en analizar bien el problema que intenta resolver la aplicación y decidir sus requisitos. El alumno se ha ocupado de hacer de Scrum Master, intentando en todo momento que se cumpliese Scrum y organizando las reuniones pertinentes. Y por último, también ha sido el alumno el encargado de hacer las tareas de desarrollo del proyecto.

6.1.2 Artefactos

Para entender el trabajo que hay que hacer y cómo hay que organizarse, Scrum está compuesto de varias herramientas o componentes presentes en todo el proceso. Estos artefactos los explicamos a continuación:

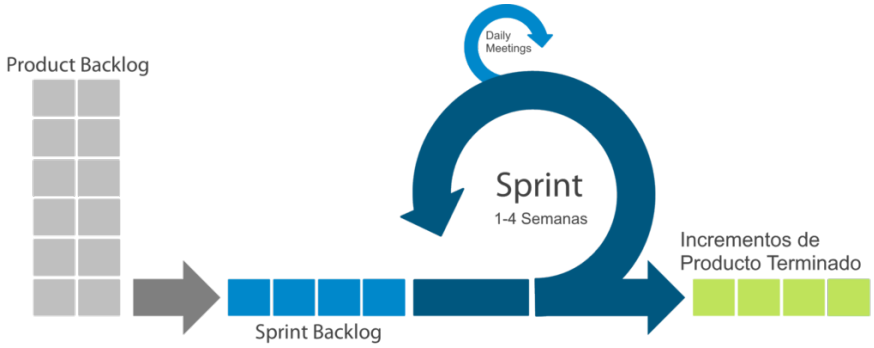


Figura 6.1: Artefactos de Scrum.

- **Product Backlog**: Es una lista ordenada de todo lo que puede ser necesario en el desarrollo del producto, y será la única fuente de requisitos para cualquier cambio que se haga. El Product Backlog es dinámico y puede que no esté completo, sólo expone los requisitos conocidos y a lo largo del proyecto sería posible añadir nuevas tareas que fuesen útiles. Está compuesto tanto de requisitos, como de mejoras y correcciones.
- **Sprint Backlog**: El proyecto estará dividido en *sprints*, espacios de tiempo periódicos, normalmente de entre 1 semana y un mes. En cada sprint el equipo de desarrollo se compromete a realizar complemente las tareas que formen parte del Sprint Backlog. Las tareas del Sprint Backlog son un subconjunto del Product Backlog.
- **Incremento**: El incremento del producto es el resultado del sprint. Constituye una versión del producto utilizable y potencialmente preparada para liberar en producción.

En la figura 6.1 se puede ver el flujo habitual de Scrum y las interacciones entre sus componentes. A partir del Product Backlog se creará un Sprint Backlog por cada sprint y progresivamente se harán avances del producto en forma de incrementos. Para poner en práctica estas convenciones de Scrum se ha utilizado la herramienta **Trello**. Hemos creado un tablero con varias columnas para ir guardando las tareas, e ir desplazándolos según su estado. En la figura 6.2 podemos ver una captura del tablero del proyecto.

6.1.3 Eventos

Según Scrum van a existir cinco tipos de reuniones o ceremonias, las cuales se realizarán durante los sprints para poner en uso los valores de la metodología, como son la colaboración y eficiencia. Este tipo de reuniones ayudan al equipo a mejorar y seguir avanzando en la dirección correcta. Son las siguientes:

- **Sprint planning**: Esta reunión debe hacerse al comienzo de cada sprint. Sirve para planificar las tareas del Backlog que se van a realizar en el próximo sprint. Hay que

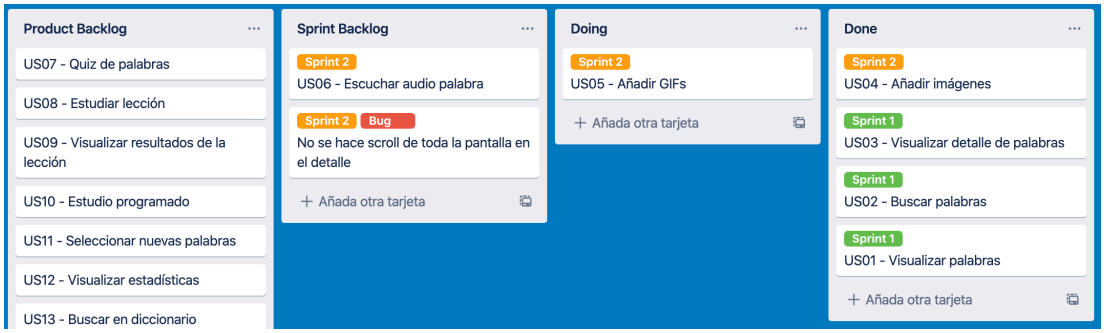


Figura 6.2: Muestra del tablero Scrum del presente proyecto hecho con Trello

tener en cuenta la capacidad o disponibilidad del equipo para poder añadir una cantidad realista de tareas. Antes de este evento es necesario que el equipo haya estimado las tareas del Backlog y también una previa priorización por parte del Product Owner.

- **Daily scrum**: Es una reunión de no más de 15 minutos que debería hacerse con una frecuencia diaria. El equipo se reúne todos los días para que cada miembro informe en lo que ha estado trabajando el día anterior, y que es lo que hará ese mismo día. Además de dar visibilidad de todo lo que sucede en el sprint, también sirve para intentar desbloquear algún impedimento que haya sucedido.

- **Refinement**: Es una reunión que hay que hacer al menos una vez a la semana. En ella se refina el Product Backlog, lo que quiere decir que se intenta dejar completamente preparadas las tareas para su implementación. Es necesario que cada tarea cuente con una descripción detallada de lo que hay que hacer y sus criterios de aceptación. Además, entre todos los miembros del equipo decidirán una estimación de la complejidad de las tareas y debatirán posibles dudas.

- **Sprint review**: Al final de cada sprint se celebrará esta reunión. Consiste en la demostración de las funcionalidades construidas durante el sprint. También se expone qué tareas han podido cerrarse y cuáles no.

- **Sprint retrospective**: Al final del sprint se hará esta reunión, de la misma forma que la Sprint review. Su objetivo es exponer abiertamente lo que ha ido bien y lo que no durante el sprint, así el equipo encontrará conjuntamente mejores formas de alcanzar los objetivos del proyecto.

La duración de los *sprints* para este proyecto ha sido aproximadamente de entre **2 y 3 semanas**. Tiempo variable porque no se ha dedicado un tiempo fijo de trabajo cada semana. Al ser un equipo reducido, el director y el alumno, se han adaptado las diversas reuniones que se suelen tener en única reunión de forma periódica, para así optimizar mejor el tiempo. Estas

reuniones han sido importantes sobre todo para debatir las funcionalidades de la aplicación y decidir cómo afrontarlas.

6.2 Planificación y costes

En esta sección veremos la planificación del proyecto en los diferentes *sprints*, los problemas surgidos durante su implementación y la duración final de los mismos. Por último, evaluaremos los recursos utilizados durante el proyectos y los costes asociados.

6.2.1 Sprints

Como comentamos anteriormente, la duración de los *sprints* ha sido flexible, ya que hemos preferido agrupar en cada *sprint* funcionalidades relacionadas con alguna parte de la aplicación. Esto nos ha permitido ir escalando poco a poco la aplicación y hacer iterativamente las fases de diseño, implementación, pruebas y documentación.

A continuación describiremos el trabajo realizado en cada *sprint*, junto con todas sus historias de usuario que se han podido cerrar. El detalle de las historias de usuario se puede ver en la sección 5.1.

Sprint 0: Investigación y formación

En este sprint se analizaron las funcionalidades de las que estaría formada la aplicación *Vocabulary*. Se creó el *Backlog* con las requisitos más importantes partiendo de los objetivos principales. También hubo que dedicar tiempo a formarse en las tecnologías desconocidas por el alumno, en especial *Wordnet*.

Sprint 1: Preparación datos

El propósito de este sprint fue crear la base de datos con toda la información de las palabras seleccionadas. No se resolvió ninguna historia de usuario porque lo que se hizo fue un trabajo previo a la aplicación.

Sprint 2: Integración datos

En este sprint se inició el proyecto de la aplicación iOS y se integró en ella la base de datos creada en el sprint anterior. Hubo que construir las bases del proyecto para cumplir con la arquitectura de la aplicación. Las historias de usuario que se desarrollaron fueron las relacionadas con la visualización de las palabras cargadas.

- US01 - Visualizar listado de palabras

- US02 - Buscar palabras
- US03 - Visualizar detalle de palabras

Sprint 3: Contenido multimedia

Se hizo posible poder buscar y asociar imágenes o GIFs con las palabras. También se implementó la reproducción del audio de las palabras.

- US04 - Añadir imágenes
- US05 - Añadir GIFs
- US06 - Escuchar pronunciación

Sprint 4: Flashcards

En este sprint se creó el subsistema de la aplicación que soporta todo lo relacionado con los cuestionarios de palabras de una lección. Se hizo un desarrollo modular para así poder soportar otro tipo de juegos o preguntas en el futuro.

- US07 - Cuestionario de palabras
- US08 - Estudiar lección
- US09 - Visualizar resultados de la lección

Sprint 5: Repetición espaciada

En este sprint se llevó a cabo la implementación del algoritmo de repetición espaciada para planificar los estudios. También se implementaron las funcionalidades necesarias para el comienzo de una lección, como por ejemplo la elección de las palabras a estudiar.

- US10 - Estudio programado
- US11 - Seleccionar nuevas palabras
- US12 - Visualizar pantalla principal

Sprint 6: Visualizar estadísticas

Esta iteración consistió en integrar la librería de gráficos estadísticos para mostrar el progreso del usuario.

- US13 - Visualizar estadísticas

Sprint 7: Diccionario

En este sprint se elaboró una API para que desde la aplicación se pueda buscar cualquier palabra como si fuese un diccionario. Además, se desarrollaron las tareas necesarias para acoplarlo con el resto de funcionalidades.

- US14 - Buscar en diccionario
- US15 - Aprender palabras buscadas
- US16 - Visualizar búsquedas recientes

Sprint 8: Preferencias

En este último sprint se desarrollaron todas las tareas relacionadas con la pantalla de ajustes de la aplicación.

- US17 - Configurar recordatorio lección
- US18 - Configurar nuevas palabras
- US19 - Configurar pronunciación

6.2.2 Seguimiento

En este apartado resaltaremos los progresos y problemas más importantes que han surgido a lo largo de los diferentes sprints. También mostraremos la duración final del proyecto.

Las tareas relacionadas con tecnologías iOS fueron fluyeron bastante bien durante todo el proyecto, ya que ya se tenía algo de experiencia previa en el desarrollo de este tipo de aplicaciones. Lo que más esfuerzo llevó, sobre todo al principio, fue instaurar la arquitectura de la aplicación con todos sus componentes. Según fue avanzando el proyecto se automatizaron procesos mediante la creación de plantillas en el entorno de desarrollo. De esta manera la creación de los componentes de la arquitectura se pudo acelerar en gran medida.

En algunos momentos se detectaron *bugs* de tareas ya resueltas. Estos errores se fueron resolviendo en el sprint en el que se encontraron. La mayoría de estos *bugs* no fueron críticos, ya que en la mayoría de las veces se trataba simplemente de fallos visuales, como por ejemplo de adaptabilidad a distintos tamaños de pantalla.

Al final de cada sprint, el progreso de la aplicación se dejó probar a varias personas cercanas para que nos devolviesen su *feedback*. Gracias a esto conseguimos encontrar algunos fallos de usabilidad que se fueron corrigiendo.

En el sprint 7 de creación del diccionario, tuvimos problemas que nos hicieron retrasarnos. Se construyó inicialmente la funcionalidad usando una tecnología de servidor, pero desafortunadamente se tuvo que desechar por otra alternativa porque no se cumplía con el rendimiento

Tabla 6.1: Tiempo dedicado al desarrollo de cada sprint del proyecto.

Sprint	Nombre	Inicio	Fin	Duración (sem)
0	Investigación y formación	07/09/2020	20/9/2020	2
1	Preparación datos	21/09/2020	04/10/2020	2
2	Integración datos	05/10/2020	25/10/2020	3
3	Contenido multimedia	26/10/2020	08/11/2020	2
4	Flashcards	09/11/2020	22/11/2020	2
5	Repetición espaciada	23/11/2020	13/12/2020	3
6	Visualizar estadísticas	14/12/2020	20/12/2020	1
7	Diccionario	21/12/2020	10/01/2021	3
8	Preferencias	11/01/2021	24/01/2021	2
	Total	07/09/2020	24/01/2021	**20**

esperado. Las peticiones de búsqueda en el diccionario tardaban demasiado tiempo, pero con los cambios que se hicieron se consiguió resolver el problema.

6.2.3 Duración

Por último, vamos a ver la duración final de cada uno de los sprints y también la duración total del proyecto. Debido a las circunstancias laborales del alumno, no se ha podido hacer un horario fijo de trabajo. Podemos establecer que cada semana se ha dedicado una media de 21 horas al proyecto, lo que equivaldrían a unas **3h diarias**. En la tabla 6.1 se puede observar las duraciones (en semanas) de cada sprint, junto con sus fechas de inicio y fin.

6.2.4 Costes

En esta sección se analiza el coste total del proyecto, considerando tanto los costes de recursos humanos como los de hardware y software.

Recursos humanos

En este proyecto se consideraron cuatro perfiles diferentes en cuanto a recursos humanos. En la tabla 6.2 se puede ver el trabajo realizado por cada uno, la dedicación en número de horas y el coste total asociado.

El trabajo realizado por el alumno durante los sprints se reparte en diferente proporción entre estos 3 roles: Jefe de proyecto (10%), Analista (30%) y Programador (60%).

El tutor del proyecto se ha encargado de hacer las labores de un Asesor, y el tiempo que ha consumido ha sido de 3h de media por sprint.

Tabla 6.2: Desglose del coste de los recursos humanos del proyecto.

Recurso	**Salario (€/h)**	**Trabajo (h)**	**Coste (€)**
Jefe de proyecto	40	42	1.680
Analista	35	126	4.410
Programador	25	252	6.300
Asesor	40	24	960
		Total	**13.350**
		Total con IVA (21%)	**16.154**

Tabla 6.3: Desglose del coste de los recursos hardware del proyecto.

Recurso	**Coste (€)**	**Vida útil (años)**	**Tiempo de uso (meses)**	**Total (€)**
Ordenador	1.680	4	5	175
Móvil	710	3	5	99
			Total	274

Recursos hardware

Para la realización de este proyecto se utilizaron materiales de los que ya disponía el alumno, pero aún así imputaremos su coste en el proyecto. Los materiales utilizados han sido los siguientes:

- Ordenador portátil con sistema operativo macOS, el cual es imprescindible para el desarrollo de aplicaciones en iOS.
- Dispositivo móvil con sistema operativo iOS. En concreto un iPhone 8 que se usó para ir haciendo pruebas en él a lo largo del desarrollo.

El coste final imputado de estos recursos físicos se pueden observar en la tabla 6.3.

Recursos software

Todas las tecnologías que se han usado han sido gratuitas. La mayoría no son de pago, salvo *Heroku* y *Trello* de las cuales hemos usado sus capas gratuitas.

Es necesario recalcar que existirían unos costes asociados al despliegue en producción y mantenimiento de la aplicación. Este tipo de costes no los hemos imputado en el coste total, pero hay que tenerlos en cuenta en un posible desarrollo comercial. Serían los siguientes:

- Para poder subir cualquier aplicación a la tienda de aplicaciones de Apple (*App Store*), es necesario disponer de una cuenta de pago de desarrollador, para así pertener al **Apple**

Tabla 6.4: Desglose del coste total del proyecto.

Tipo de recurso	Coste (€)
Humano	16.154
Hardware	274
Software	0
Total	**16.428**

Developer Program. Su coste es de 99$ al año y la aplicación siempre estará disponible mientras se esté subscrito al programa.

- La capa gratuita de **Heroku** es un poco limitada. Para el desarrollo de nuestro servicio web ha sido suficiente, pero en el momento que se despliegue en un entorno con usuarios reales va ser necesario una cuenta de pago para tener disponibilidad 24 h y que no haya problemas de rendimiento ni escalabilidad. Los precios pueden rondar entre los 7-50 mensuales.

Coste total

El costo total se puede deducir de los costes humanos y materiales, ver tabla 6.4.

Capítulo 7

Diseño

En este capítulo detallaremos las decisiones de diseño que se han tomado junto a la arquitectura del sistema, el modelo de datos y la arquitectura de la aplicación iOS.

7.1 Arquitectura del sistema

La arquitectura global del sistema que se ha diseñado se puede dividir en tres partes: un *script* de procesamiento de datos, la aplicación iOS conocida como *Vocabulary* y el *backend*. En la figura 7.1 se muestra el esquema de esta arquitectura con los distintos componentes que pueden interactuar y como se agrupan. A continuación veremos en que consiste el sistema:

- El ***script*** es un programa implementado con python, con la función de procesar una lista de palabras del vocabulario más frecuente y extraer su información mediante la librería Wordnet. Con este *script* se conseguirá crear una base de datos para añadir al cliente iOS.

- El **cliente iOS** es la aplicación móvil con la que interactuará el usuario. Casi toda la carga de trabajo va a recaer en la aplicación iOS, ya que tanto la lógica de negocio como la base de datos estará localmente en el dispositivo sin necesidad de ninguna conexión externa. La aplicación iOS dispondrá inicialmente de una base de datos precompilada con toda la información sobre las palabras que se ha recopilado con antelación. A lo largo de la ejecución de la misma, la base de datos se irá actualizando e irá creciendo con nuevos datos.

- La aplicación móvil va a necesitar comunicarse con distintos servicios externos para diversas funcionalidades. El **backend** va a consistir en una parte de servicios o APIs de terceros, y en una API que se ha implementado en este proyecto para un propósito específico. Las APIs de terceros son Giphy y Unsplash las cuales se han usado para

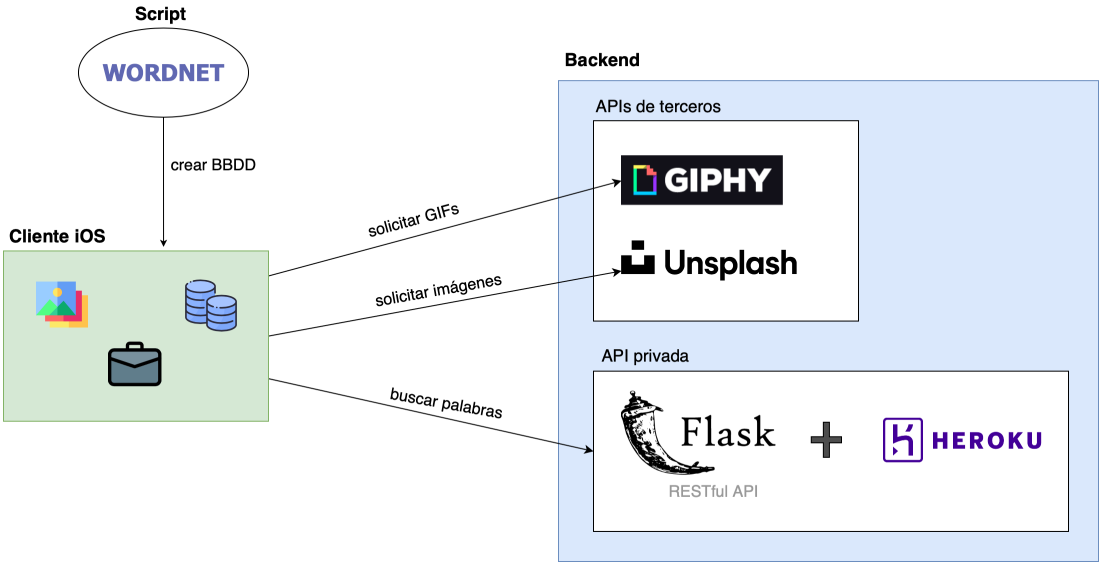


Figura 7.1: Arquitectura global del sistema propuesto.

conseguir GIFs e imágenes, respectivamente. Este tipo de contenido multimedia lo utilizará la aplicación para mostrarlo al usuario y también será posible almacenarlo en la base de datos del dispositivo si fuese necesario. El otro servicio que falta se trata de una API RESTful que se ha implementado mediante Flask para buscar información en inglés sobre nuevas palabras, de la misma forma que funcionaría un diccionario común. Esta API va a estar alojada en Heroku para que pueda ser accesible públicamente desde cualquier dispositivo.

En las próximas secciones veremos cada componente de la arquitectura del sistema.

7.2 Creación de la base de datos

Algo crucial para la creación de *Vocabulary* fue conseguir un conjunto de las palabras más frecuentes en inglés para que posteriormente se carguen en la aplicación y se puedan aprender. En esta sección veremos como se ha conseguido la base de palabras y también como se han preparado mediante *Wordnet*.

7.2.1 Lista de palabras

La idea principal era recopilar un grupo grande de vocabulario de por lo menos 3.000 palabras. Dada la magnitud del número de palabras que nos interesaba, la idea de ir eligiendo una a una cada palabra quedó descartada rápidamente. Entonces lo que se hizo fue buscar un listado en internet que se pudiese descargar y que cumpliese unos mínimos de calidad en cuanto

Tabla 7.1: Muestra de las primeras filas de las palabra más frecuente del corpus COCA

Palabra	Tipo de palabra	Frecuencia
the	a	22.038.615
be	v	12.545.825
and	c	10.741.073
of	i	10.343.885
a	a	10.144.200

la selección del vocabulario más importantes. Finalmente la lista que se obtuvo se consiguió de la web *www.wordfrequency.info* [36], y consiste en una selección de las 5.000 palabras más frecuentes de COCA, un corpus de varios millones de palabras, que se han extraído de textos públicos como pueden ser subtítulos de películas, libros, periódicos, etc...

Encontramos un inconveniente en las palabras, y es que muchas de ellas se tratan de artículos, preposiciones y conjunciones, las cuales aparecen en el listado porque están dentro de los 5.000 vocablos que más se repiten del corpus. Como buscamos otro tipo de vocabulario para la aplicación, hubo que hacer un filtrado y descartar estas palabras y así quedarnos sólo con los sustantivos, verbos y adjetivos.

En la tabla 7.1 se puede ver una muestra de los primeros 5 elementos del listado, que como comentamos son todas estas palabras que no interesan aprender. El listado está ordenado por la columna *Frecuencia* de mayor a menor. Esta columna indica el número de veces que se repite la palabra dentro del corpus, por eso el artículo 'the' por ejemplo, que aparece en la mayoría de las frases en cualquier contexto, será la palabra con más frecuencia. En la columna llamada *Tipo de palabra* se encuentra la categoría gramatical representada por una letra, por ejemplo la letra ´a' es para los artículos.

Hemos sacado una gráfica para ver visualmente la cantidad de palabras por categoría dentro del listado (ver figura 7.2). Se puede ver la distribución de las categorías gramaticales y apreciar que los verbos ('v'), sustantivos ('n') y adjetivos ('j') son los que más se repiten en el *Top 5.000* de COCA. Finalmente con la lista de palabras filtrada sin las palabras que no necesitamos, haremos una selección final de las 3.000 primeras.

7.2.2 Extracción de información con WordNet

Una vez tenemos las 3.000 palabras más frecuentes en inglés, es necesario obtener más datos sobre ellas, ya que sólo disponemos del nombre de estas. En este apartado explicaremos como extraer información sobre las palabras mediante WordNet.

En la sección 3.2, nos detuvimos a dar una explicación más teórica de que es WordNet y en qué consiste. Con la librería WordNet es posible obtener para una palabra su ficha de infor-

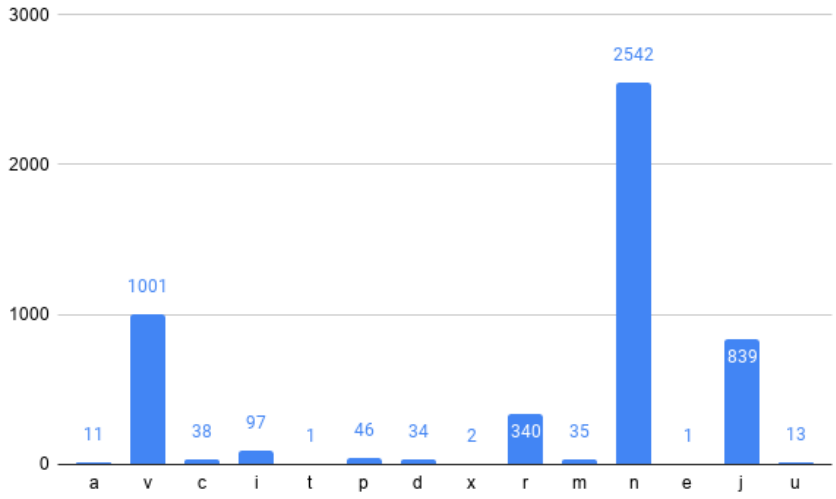


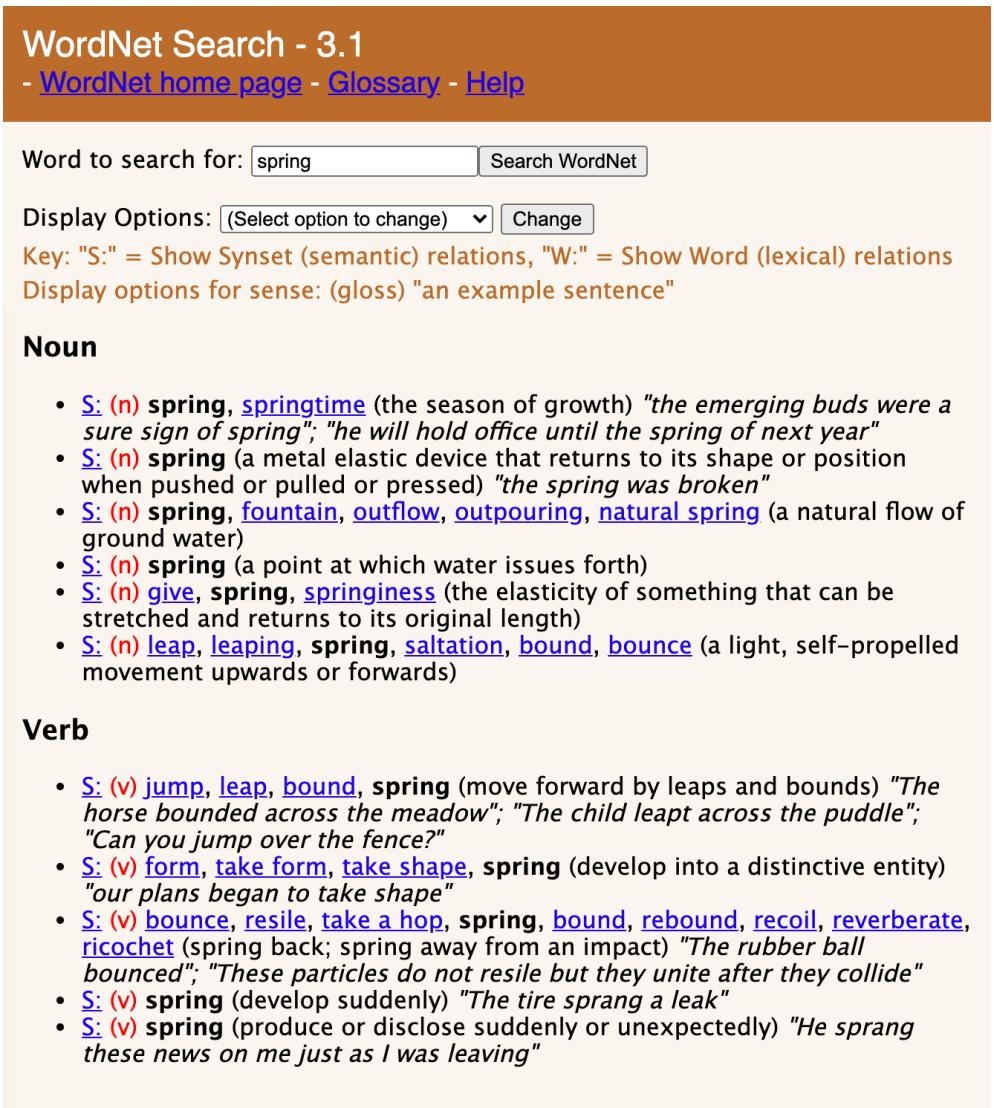
Figura 7.2: Recuento de las categorías de palabras del Top 5000 del corpus COCA.

mación, con su definición, sinónimos, ejemplos de uso e incluso una traducción. Esta librería puede ser utilizada desde el lenguaje de programación *python* para sacar todo su provecho.

También se puede acceder a WordNet y probar a hacer consultas en sus datos a través de un buscador *online* en [10]. Vamos a ver con un ejemplo, cómo son los datos que nos devuelve WordNet y cómo los vamos a utilizar. En la figura 7.3 se pueden ver los resultados para la búsqueda de la palabra "*spring*", esta palabra se puede catalogar como sustantivo o verbo y además, dentro de los sustantivos significará distintas cosas dependiendo en el contexto que salga. Cada significado de una palabra WordNet lo llama *synset*, en este caso para el sustantivo *spring*, los 3 primeros synsets tienen el significado de *primavera*, *muelle* y *manantial*. Para cada *synset* pueden aparecer palabras relacionadas porque significan lo mismo y que serán por tanto sus sinónimos.

En este proyecto como lo que nos interesa son los significados más importantes, lo que se ha hecho ha sido a partir de cada palabra de la lista con su categoría gramatical, coger su primer *synset* o significado que será el clasificado por WordNet como el más frecuente y nos aportará toda la información que queremos. La palabra 'spring' da el caso de que se encuentra en el Top 3000 de nuestras palabras y por tanto el significado que se aprenderá será el de *primavera*. Hay que saber que WorNet también dispone de traducciones en varios idiomas de los significados.

Una vez conocido como se sacarán los datos de WordNet, podemos ver el flujo completo de transformación de las palabras en la figura 7.4. El *script* que hemos implementado con WordNet extraerá la información pertinente y la convertirá en un fichero JSON. Este fichero se utilizará para crear finalmente la base de datos que usará la aplicación *Vocabulary*.



WordNet Search - 3.1

- WordNet home page - Glossary - Help

Word to search for: spring Search WordNet

Display Options: (Select option to change) Change

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations

Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) **spring**, springtime (the season of growth) *"the emerging buds were a sure sign of spring"; "he will hold office until the spring of next year"*
- S: (n) **spring** (a metal elastic device that returns to its shape or position when pushed or pulled or pressed) *"the spring was broken"*
- S: (n) **spring**, fountain, outflow, outpouring, natural spring (a natural flow of ground water)
- S: (n) **spring** (a point at which water issues forth)
- S: (n) give, **spring**, springiness (the elasticity of something that can be stretched and returns to its original length)
- S: (n) leap, leaping, **spring**, saltation, bound, bounce (a light, self-propelled movement upwards or forwards)

Verb

- S: (v) jump, leap, bound, **spring** (move forward by leaps and bounds) *"The horse bounded across the meadow"; "The child leapt across the puddle"; "Can you jump over the fence?"*
- S: (v) form, take form, take shape, **spring** (develop into a distinctive entity) *"our plans began to take shape"*
- S: (v) bounce, resile, take a hop, **spring**, bound, rebound, recoil, reverberate, ricochet (spring back; spring away from an impact) *"The rubber ball bounced"; "These particles do not resile but they unite after they collide"*
- S: (v) **spring** (develop suddenly) *"The tire sprang a leak"*
- S: (v) **spring** (produce or disclose suddenly or unexpectedly) *"He sprang these news on me just as I was leaving"*

Figura 7.3: Búsqueda de la palabra "*spring*" en WordNet Search [10].

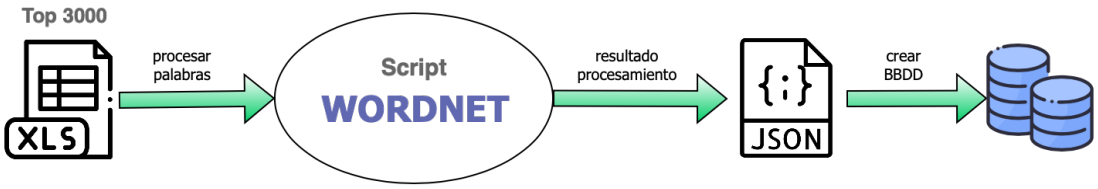


Figura 7.4: Flujo de transformación de datos de la solución propuesta.

7.3 Diseño del diccionario

Hemos visto en la sección anterior como obtener las palabras que se usarán inicialmente dentro de la aplicación, pero *Vocabulary* requiere una funcionalidad para buscar información sobre otras palabras que no estén en el Top 3.000 preseleccionado. En esta sección veremos como se ha diseñado esta solución.

Como no existe ningún tipo de servicio *online* para acceder a WordNet, la solución que se ha propuesto ha consistido en reutilizar el *script* creado en el apartado 7.2.2 y adaptarlo un poco para crear un API REST accesible desde cualquier dispositivo.

Se ha utilizado la librería Flask para crear el servicio web. La API creada va a consistir solamente en el *endpoint* **GET /dictionary**, el cual va a aceptar como parámetro la palabra que se desea. El resultado de esta petición será la ficha del primer *synset* (significado) de la palabra. Desde la aplicación se controlará si la palabra que se busca ya la tenemos en la base de datos, para así no hacer peticiones innecesarias a la API.

Para que la aplicación web construida con Flask sea visible públicamente para todo el mundo, se ha desplegado en la plataforma Heroku. La plataforma Heroku ejecutará la aplicación Flask en contenedores llamados *dynos* y también permitirá escalar los recursos necesarios bajo demanda. Además es posible monitorizar el servicio para conocer los fallos producidos, tiempos de respuesta, supervisar la CPU y más métricas interesantes.

7.4 Contenido multimedia

Según los requisitos planteados en la etapa de análisis de este proyecto, van a ser necesarios varios casos de uso para obtener contenido multimedia. Uno de esos casos de uso será escuchar la pronunciación de las palabras, y el otro será poder elegir una imagen que ayude en la representación de una palabra. En los siguientes apartados expondremos como se han resuelto estos casos de uso.

7.4.1 Audio

Para escuchar la pronunciación de las palabras vamos a hacer uso de una herramienta TTS (*text-to-speech*). Este tipo de herramientas convierten textos de palabras en oraciones audibles, lo que lo hace muy útil por ejemplo para personas ciegas que necesitan lectores de pantalla para poder usar algún dispositivo.

Existen varias posibilidades para poder utilizar un TTS en *Vocabulary*, desde APIs (por ejemplo Amazon Polly [37] y Google Cloud Text-to-Speech [38]) cómo la opción nativa de Apple para convertir texto a voz conocida como Speech Synthesis [39]. Al final la decisión fue utilizar la opción nativa, ya que nos va a permitir hacer lo que queremos a la perfección

y así, evitaremos depender de un servicio remoto de terceros. Además, la solución de Apple es gratuita no como los otros servicios.

7.4.2 Imágenes

Dentro de las imágenes van a existir de dos tipos diferentes, imágenes estáticas e imágenes animadas (GIFs). Esta división del tipo de imágenes se ha hecho porque se ha visto que muchas veces no es posible simbolizar una palabra mediante una imagen estática y por eso entran en juego los GIFs. Las imágenes estáticas se van a utilizar para representar sólo sustantivos, ya que los sustantivos al ser cosas concretas materiales o inmateriales, es más sencillo relacionarlos con este tipo de imágenes. Los GIFs los utilizaremos para los verbos y adjetivos que hemos comprobado que se relacionan mejor. Aún así habrá muchas veces que el usuario no podrá asociar una palabra con una imagen, porque hay palabras que no se pueden representar bien.

Utilizaremos las APIs de Unsplash y de Giphy, explicadas en el apartado 4.3, para buscar imágenes estáticas y GIFs respectivamente. Tanto Unsplash como Giphy cuentan con una librería para incorporar directamente un seleccionador de contenido sin necesidad de implementar nosotros las peticiones [40][41]. De todos modos, hemos preferido implementar nosotros mismos nuestro seleccionador de imágenes para así tener un componente genérico y no dos distintos.

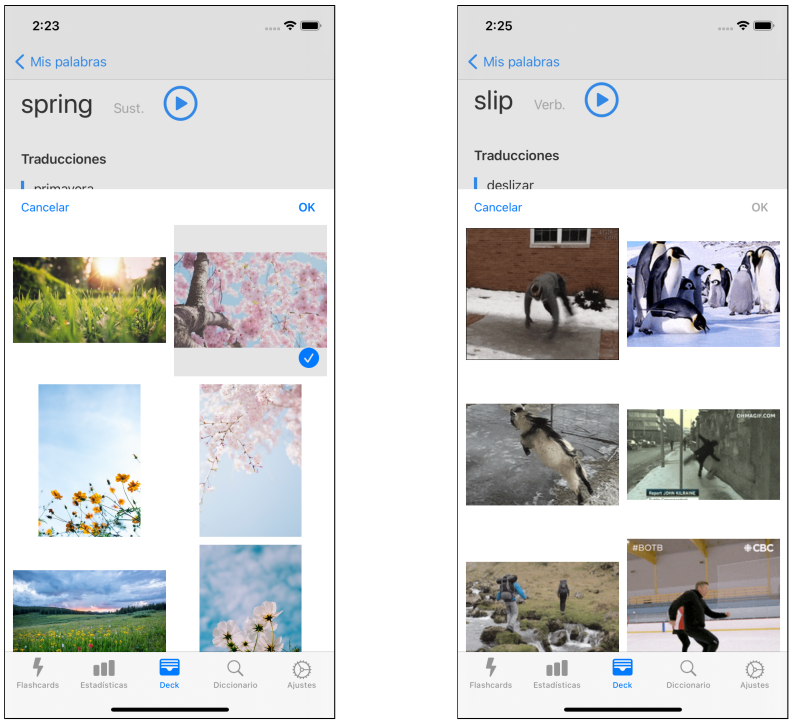
El *endpoint* que hemos usado de Unsplash es **GET /search/photos**, que permite obtener un listado de urls con las imágenes a partir del nomber de una palabra [42]. Para realizar la misma acción con los GIFs se ha usado el llamado **Search Endpoint** de Giphy [43]. Estas APIs proporcionan más servicios útiles para otras funcionalidades, pero no han hecho falta para cubrir nuestros requisitos.

A la hora de buscar imágenes se decidirá que *endpoint* usar según la categoría gramatical de la palabra, como se puede ver en las figuras 7.5a y 7.5b del selector desarrollado. Para la palabra *spring* al ser un sustantivo se mostrarán imágenes fijas y para el verbo *slip* se mostrarán curiosos GIFs relacionados.

7.5 Diseño del algoritmo de repetición espaciada

En esta sección explicaremos el diseño del algoritmo que planifica los repasos de las palabras, basado en las técnicas de repetición espaciada vistas en la sección de fundamentos 3.1.

Hemos modelado el algoritmo como una máquina de estados, su representación puede verse en la figura 7.6. Cada día en la lección diaria tendremos nuevas palabras para aprender de nuestro *deck*. Esta lección será una mezcla de palabras nuevas y de otras palabras que



(a) Imágenes estáticas

(b) Imágenes animadas (GIFs)

Figura 7.5: Selector de imágenes de *Vocabulary*.

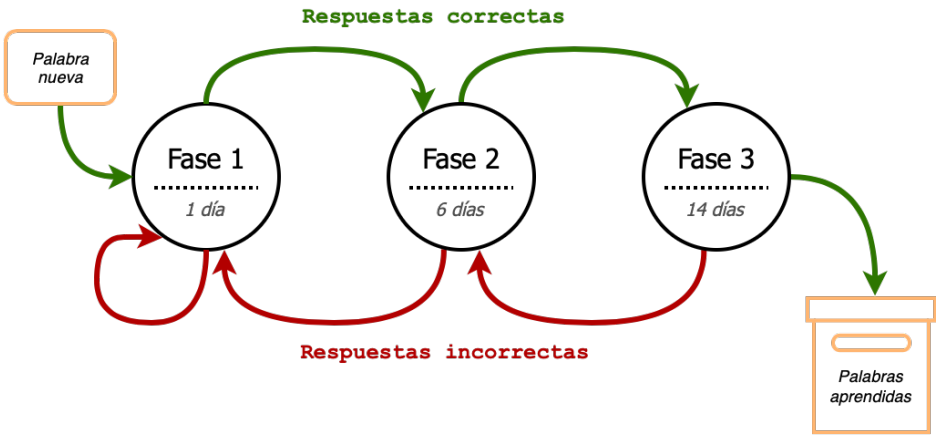


Figura 7.6: Diagrama de estados de la implementación de repetición espaciada en *Vocabulary*.

toca repasar de lecciones de distintos días. Para que termine una lección es obligatorio que se responda bien todos los ejercicios, esto quiere decir, que si una palabra se responde mal volverá a salir repetidamente hasta que se responda correctamente.

Una vez se termine la lección se planificará cada palabra para un día específico dependiendo en el estado que esté. En el diagrama podemos ver que existirán tres estados diferentes: *Fase 1*, *Fase 2* y *Fase 3*, y cada uno tendrá unos días establecidos para saber los días de espaciamiento de cada estado (1 día, 6 días y 14 días, respectivamente)

Vamos ver como funcionaría el algoritmo secuencialmente:

1. Sale por primera vez una palabra y se responde correctamente. Esta palabra entrará en el estado *Fase 1* y se planificará para el día siguiente.

2. La palabra en cuestión sale el siguiente día de nuevo. Si el usuario la recuerda y responde bien se moverá al estado *Fase 2*, si responde mal quedará en el estado *Fase 1*. En el caso de que se haya movido a *Fase 2*, la palabra no volverá a mostrarse hasta pasados 6 días.

3. Si la palabra se encuentra en el estado *Fase 2* y se responde bien pasa al estado *Fase 3*.

4. Y por último si se avanza desde el estado *Fase 3* se consideraría la palabra finalmente como aprendida y no volvería a salir en ninguna lección.

5. En los estados *Fase 2* y *Fase 3* si se responde incorrectamente se retrodecerá a un estado anterior, para que así, se vuelva a repetir la palabra en un tiempo más cercano, ya que hay que volver a recordarla.

Hemos considerado que tres estados son más que suficientes, si se añadiese más estados podría provocar que el usuario tuviese que hacer demasiados repasos y se le acumulase el

estudio. Lo ideal sería que el usuario haga los repasos en el mismo día que toque, pero en el escenario que el usuario se retrase, estos repasos se tendrán en cuenta para el día que se vuelva a hacer una lección. Podría darse el caso que hubiesen muchas palabras de repaso acumuladas y que no se quiera estar tanto tiempo con la lección. Una idea para solventar este problema sería poder configurar un límite de palabras de repaso diarias.

7.6 Almacenamiento

En esta sección explicaremos los distintos tipos de almacenamiento y también expondremos el modelo de datos final de la aplicación *Vocabulary*.

Para guardar los datos en el dispositivo hemos elegido la tecnología Realm de bases de datos, que se detalló en el apartado 4.2.2. Realm nos permite importar a la aplicación una base de datos construida previamente con WordNet, como vimos en la sección 7.2.2.

Los datos que guardaremos en la base de datos consistirán en la información de todas las palabras y el progreso del usuario en el aprendizaje. Existen otro tipo de datos a guardar relacionados con ajustes internos de la aplicación o de preferencias de usuario, los cuales no almacenaremos en la base de datos debido a su forma y comportamiento. La persistencia elegida para este tipo de datos la veremos más adelante en el apartado 7.6.2.

Hemos decidido tener la base de datos de forma local en el dispositivo en vez de tenerla en un servidor externo. La razón es porque así tendremos acceso a los datos sin necesitar una conexión a la red, y porque dadas las características de *Vocabulary* no se requiere compartir datos con otros usuarios, ni obtener información de manera dinámica, por lo que es mucho más eficiente la solución aplicada.

Hay un caso de uso no contemplado en este proyecto pero que sería interesante, es el de tener un *backup* del progreso del usuario para no perder su información por si cambia o pierde el móvil. Una posible solución para este caso sería hacer una copia de la base de datos Realm a un servidor externo si el usuario lo requiriese. Incluso se podría usar la solución de Apple de su nube iCloud, la cual requeriría un pequeño desarollo para su integración, además de ofrecer la ventaja de que todos los usuarios de iOS por defecto ya están autenticados en su espacio iCloud [44]. Hay que tener en cuenta que para poder integrar iCloud, hace falta una cuenta de pago de desarrollador de Apple. De todos modos, esta posible funcionalidad quedará como trabajo de mejora a futuro.

7.6.1 Modelo de base de datos

En función de las decisiones de diseño vistas hasta ahora, se ha diseñado el modelo de la base de datos que mostraremos en esta sección.

Realm es una base de datos orientada objetos, lo que quiere decir que las entidades de

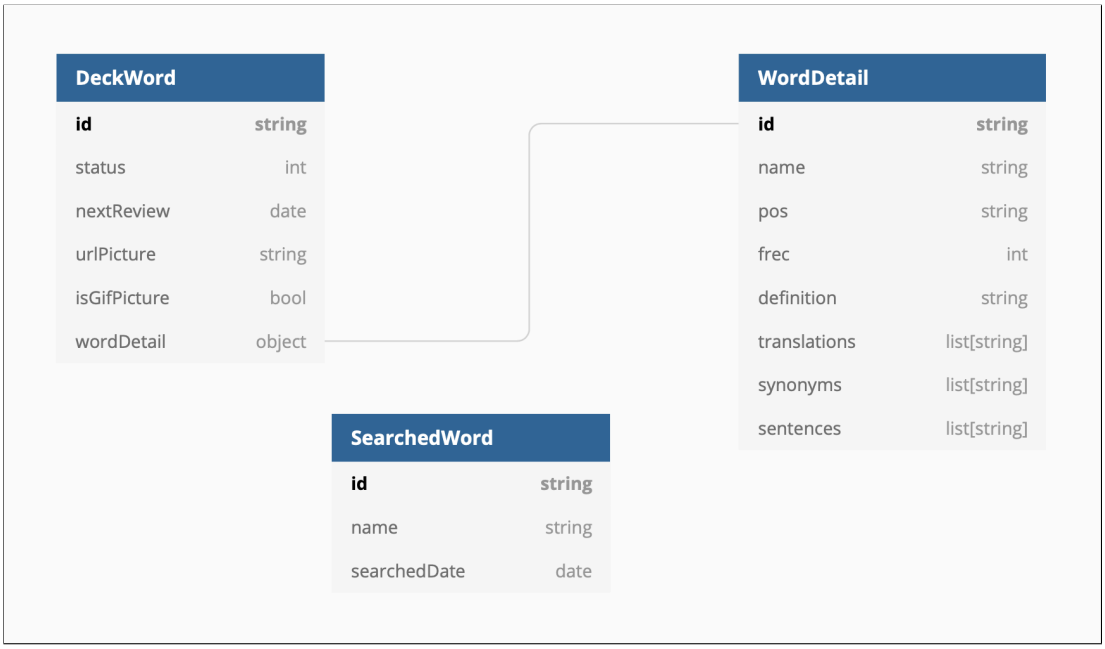


Figura 7.7: Modelo relacional de la base de datos propuesta en *Vocabulary*.

datos se representan con objetos y sus relaciones con punteros a otros objetos [45]. Difiere un poco de las bases de datos relacionales clásicas, ya que no se registran los datos en un sistema típico de tablas.

En la figura 7.7 podemos ver el modelo relacional final de la aplicación con las entidades *DeckWord*, *WordDetail* y *SearchedWord* que detallaremos a continuación.

Entidad *DeckWord*

Esta entidad es utilizada para guardar toda la información referente al estudio de las palabras. Sus atributos son los siguientes:

- **id**: Identificador único del objeto.
- **status**: Identificador numérico que representa el estado de la palabra en el *deck*. Los estados posibles se pueden ver en la tabla 7.2. Con este valor se podrá saber si la palabra todavía no se ha visto, si se está aprendiendo, si se ha aprendido o en cambio si se ha descartado. Por tanto el valor inicial para este campo será el *status 0*.
- **nextReview**: La fecha calculada del siguiente repaso de la palabra. Inicialmente nulo.
- **urlPicture**: La *url* de la imagen o GIF asociado a la palabra. Inicialmente nulo
- **isGifPicture**: Indica si la *url* asociada es de un GIF o no. Necesaria para controlar como presentar el contenido de la *url*. Por defecto a *false*.

Tabla 7.2: Valores posibles del atributo *status* de la entidad *DeckWord*.

Status	Estado
0	Para aprender
1	Fase 1 aprendizaje
2	Fase 2 aprendizaje
3	Fase 3 aprendizaje
4	Aprendida
5	Descartada

- **wordDetail**: Puntero a la entidad *WordDetail* con la información completa de la palabra.

Entidad *WordDetail*

Esta entidad representa la información del las palabras, con sus datos léxicos y semánticos. Sus atributos son los siguientes:

- **id**: Identificador único del objeto.
- **name**: El nombre de la palabra.
- **pos**: La categoría gramatical de la palabra. Los valores posibles son 'n' (sustantivo), 'v' (verbo) y 'j' (adjetivo).
- **frec**: Valor que representa la frecuencia de uso de la palabra, nos dice la importancia de la palabra dentro del corpus. Un valor más alto indica una mayor frecuencia.
- **definition**: Una definición del significado de la palabra.
- **translations**: Lista de traducciones posibles.
- **synonyms**: Lista de sinónimos asociados.
- **sentences**: Lista de frases de ejemplo donde se usa la palabra.

Entidad *SearchedWord*

Esta entidad sirve para guardar el historial de palabras que se han buscado en la funcionalidad del diccionario. Sus atributos son los siguientes:

- **id**: Indentificador único del objeto.
- **name**: El nombre de la palabra buscada.
- **searchedDate**: La fecha de la búsqueda.

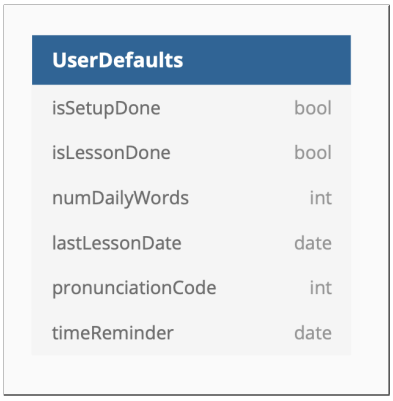


Figura 7.8: Atributos establecidos para la persistencia de *Vocabulary* con UserDefaults.

7.6.2 Ajustes y preferencias de usuario

Tendremos una persistencia distinta a la base de datos anterior para ajustes y preferencias. Esta clase de datos tiene la peculiaridad de que no tienen relaciones entre ellos y son pequeños. La solución que ofrece Apple para estos datos es usar *UserDefaults*, el cual forma parte del kit nativo de desarollo.

Con UserDefaults los datos se guardarán en el disco en un fichero XML, que tendrá entradas en forma de clave-valor. Este almacenamiento hace fácil el acceso a los datos de cualquier tipo, ya sean números, *strings*, etc... Lo único que hay que tener en cuenta es que hay que guardar datos ligeros, porque la clase *UserDefaults* se mantiene en memoria durante la ejecución de la aplicación y podría provocar problemas de rendimiento.

Los campos que hemos necesitado guardar en UserDefaults son los que aparecen en el modelo de la figura 7.8. A continuación explicaremos para que hemos usado cada uno:

- **isSetupDone**: Se usa para saber si es la primera ejecución de la aplicación y, si es así, hacer las inicializaciones necesarias de la base de datos y el resto de ajustes de la aplicación.
- **isLessonDone**: Indica si la lección del día se ha realizado.
- **numDailyWords**: Campo para guardar la preferencia de usuario de cuantas palabras nuevas quiere aprender al día.
- **lastLessonDate**: Fecha de la última lección realizada para poder controlar que sólo se prepare una lección al día.
- **pronunciationCode**: Campo numérico para guardar el tipo de pronunciación que configura el usuario, los valores posibles son: '0' (británica), '1' (americana) y '2' (australiana).

- **timeReminder**: Campo en el que se guardará la hora en la que se tiene que emitir la notificación diaria para realizar la lección. En el caso de que no este activada la notificación, este valor será nulo.

7.7 Arquitectura de la aplicación iOS

En esta sección vamos a ver la arquitectura de la aplicación iOS de *Vocabulary*, y cómo interactúan sus componentes.

La arquitectura propuesta de *Vocabulary* se puede ver en la figura 7.9. Se trata de una arquitectura basada en VIPER, de la cual vimos sus principios y fundamentos en la sección 3.3. La arquitectura se divide en varias capas cada una con sus propios componentes. Esta división permitirá tener una separación de conceptos con la que conseguiremos tener el código desacoplado y favorecer la reutilización.

Ahora explicaremos en que consiste cada una de las capas o divisiones principales:

- **Modules**: En esta capa van a estar todos los módulos de las pantallas existentes. Hay que saber que por cada pantalla que haya en la aplicación habrá un módulo. Dentro de los módulos estará todo lo relacionado con las vistas, la lógica de presentación y la lógica de navegación. En total tendremos 12 módulos en *Vocabulary*, como por ejemplo los módulos: Home, Detail, Lesson, etc...

- **Core**: Este grupo será la parte central de la arquitectura y comunicará la capa de *Modules* con la de *Data Sources*. Aquí entraría toda la lógica de negocio con los casos de uso (*Interactors*) y los modelos de dominio que representan los datos (*Models*). Se harán tareas de preparación y transformación de datos, como también solicitar datos a la siguiente capa.

- **Data Sources**: Esta capa se va a dividir en los controladores de acceso a las APIs y en los controladores de acceso a los datos locales (*Storage*). Para los primeros existirán las peticiones implementadas pertinentes, los modelos de parseo para convertir los datos, y los clientes que realizarán las peticiones de cada servicio web. En el grupo *Storage* dispondremos del *manager* de la base de datos junto a sus modelos específicos de Realm, además del otro *manager* de acceso a *UserDefaults*.

- **Common**: Esta última capa es de acceso común desde cualquier otra capa y su objetivo es agrupar recursos de la aplicación, como imágenes, colores, y también de guardar cosas comunes de utilidad que se puedan reutilizar.

Para entender de que estarían compuestos los módulos de las pantallas vamos a verlo con un ejemplo de la aplicación. Se puede ver en la figura 7.10 el módulo *Detail* el cual se

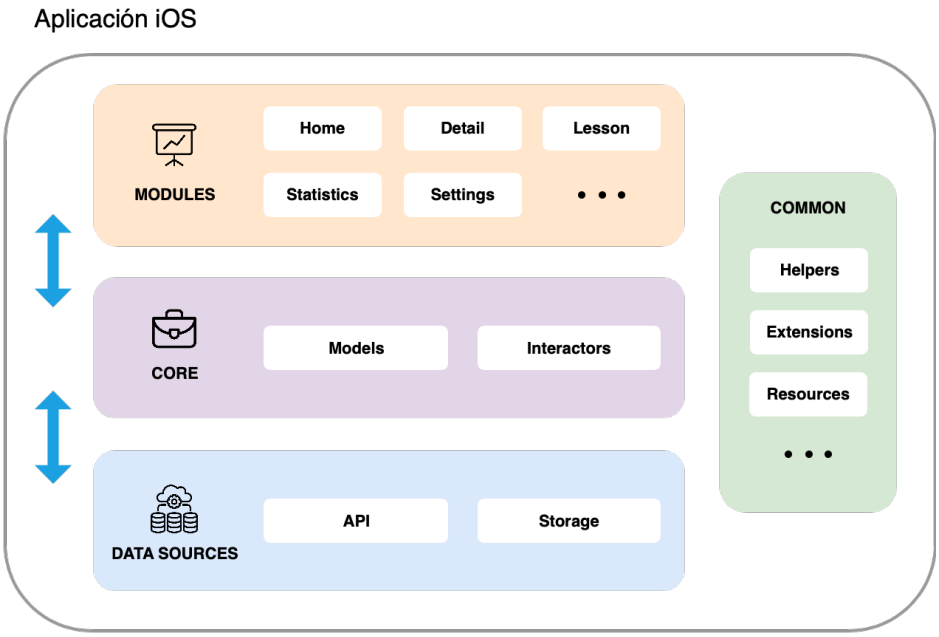


Figura 7.9: Arquitectura de la aplicación *Vocabulary*.

utiliza para ver la información en detalle de las palabras. En la imagen 7.10a se visualiza la interfaz que tendría este módulo, y en la otra imagen 7.10b la estructura en el proyecto con los **componentes típicos**: *View*, *Presenter*, *Router*, *Models*, *Builder*.

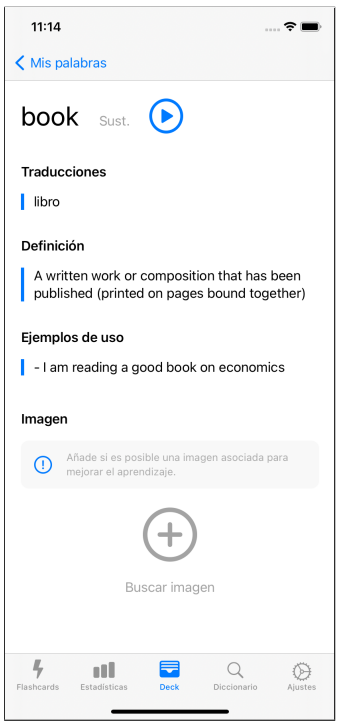
Vamos a ver cual sería el trabajo que tendría que hacer cada componente de este módulo:

- El componente ***DetailBuilder*** se encargará de inicializar el módulo, configurando el resto de componentes del módulo y pasando los parámetros de otros módulos si hubiese. En este caso se recibirán como parámetros la palabra que se quiere visualizar y el tipo de detalle que hay que mostrar, ya que esta pantalla se reutiliza en distintos lugares y tendrá aspectos visuales configurables.

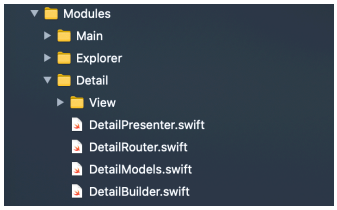
- Todo lo relacionado con la vista se encontrará en la carpeta **View**. En este componente estarán todos los archivos necesarios para componer la interfaz. Los únicos objetivos de la vista son mandar las acciones del usuario al *Presenter* y mostrar los datos que recibe del *Presenter*.

- En el fichero ***DetailModels*** se encontrará el modelo que se pasará a la vista con la información para pintar.

- El componente ***DetailPresenter*** se ocupará de tomar las acciones oportunas según el ciclo de vida de la pantalla o las interacciones del usuario. Por ejemplo si el usuario pulsa el botón “Buscar Imagen” tomará la acción de abrir otra pantalla y en este caso se



(a) Interfaz gráfica



(b) Estructura del módulo

Figura 7.10: Módulo *Detail* de la aplicación *Vocabulary*.

recibirá de vuelta una imagen que se utilizará para llamar al *WordsInteractor* (de Core) para que guarde la imagen seleccionada en la base de datos.

- El componente ***DetailRouter*** se encargará de la navegación, por ejemplo de abrir el seleccionador de imágenes pasándole el nombre de la palabra y haciendo la configuración para conseguir que la nueva pantalla se muestre sobre la pantalla actual.

Todo lo que no sea referente a la pantalla irá fuera del módulo, ya que por ejemplo los casos de uso que se encuentran en la capa *Core* serán comunes y reutilizables entre varias pantallas.

Los componentes *Builder* como se utilizan para inicializar los módulos nos servirán también para crear los módulos con *mocks* y poder hacer *Unit testing* fácilmente.

La interfaz de usuario suele ser el elemento más expuesto al cambio, con esta arquitectura conseguiremos que cualquier cambio en la vista no afecte a otras partes de la aplicación y sean así totalmente independientes.

Capítulo 8

Implementación

En esta sección detallaremos los puntos más importantes de la implementación del sistema. Dedicaremos especial atención a la creación de la base de datos, la creación de la API REST del diccionario y por último a la aplicación iOS.

8.1 Base de datos

En este apartado expondremos el desarrollo final para crear la base de datos de las palabras, basándose en las decisiones de diseño tomadas en la sección 7.2.

Se eligió ***python*** como lenguaje de programación para implementar el algoritmo de transformación de datos, ya que permite un fácil manejo de la librería *WordNet*. El listado inicial de palabras en formato *xls* fue convertido a un formato *csv* para que pudiese ser adecuadamente utilizado como fuente de entrada de nuestro *script*. Por tanto el *script* implementado recibirá como entrada un archivo *csv* y devolverá un JSON con los datos finales listos para añadir a la aplicación.

Las librerías de *python* que han sido necesarias para la elaboración del *script* son: *nltk* (4.2.6), *pandas* (4.2.7), *json* (4.2.8) y *googletrans* (4.2.9).

Se topó con un problema relacionado con la obtención de las traducciones para una minoría de nuestra lista de palabras. Este consistía en que WordNet no devolvía traducciones y nos dejaba ese campo en blanco. Para suplir este problema para las pocas palabras que no tenían traducción, el *script* hará uso de la librería *googletrans* para poder conseguir una traducción posible.

El inconveniente de esta solución, dónde no usaríamos una traducción de WordNet, es que la traducción obtenida con *googletrans* podría no coincidir con el significado extraído con WordNet, lo que podría provocar que la traducción y el resto de atributos como la definición o ejemplos de uso no correspondan. De todos modos de esta forma conseguiremos una traducción factible y no dejaremos el atributo vacío. Esta lógica que se ha comentado para

recuperar las traducciones se puede ver en el código 8.1. En él podemos ver que si no existen traducciones con WordNet nos quedaremos con la traducción del traductor Google.

```
def getTranslations(synset, word):
    wnTranslations = synset.lemma_names('spa')
    count = len(wnTranslations)
    if count == 0:
        translator = Translator()
        googleTranslation = translator.translate(word, src='en',
    dest='es')
        return [googleTranslation.text]
    else:
        return wnTranslations
```

Código 8.1: Método para conseguir traducciones de una palabra mediante WordNet y *googletrans*.

Un fragmento del código principal del *script* que se encarga de leer la lista de palabras inicial y procesarla se puede ver en el código 8.2. El *script* irá iterativamente palabra por palabra sacando su información mediante Wordnet y creando el objeto *Word*. Una vez se haya recorrido todas las palabras el resultado se encontrará en la variable de tipo lista llamada *words*.

Con los datos procesados en formato JSON, tuvimos que crear un proyecto temporal en el IDE XCode para crear la base de datos Realm que vimos en la sección 7.6. La única forma de añadir a una aplicación iOS una base de datos Realm ya poblada, es ir insertando en este proyecto temporal todos los registros para así obtener un fichero Realm. Posteriormente importaremos este fichero Realm al proyecto de la aplicación *Vocabulary* [46].

Los pasos iterativos que tiene que hacer el proyecto temporal por la razón que acabamos de comentar son:

1. Leer una entrada del JSON.

2. A partir de la entrada leída construir los objetos *DeckWord* y *WordDetail* definidos en el modelo de datos (7.6.1).

3. Insertar los dos objetos en la base de datos.

Esta base de datos obtenida tendrá toda la información que necesita la aplicación con el Top 3000 de palabras seleccionado. El espacio que usará la base de datos dentro de la aplicación será de 2.4 MB, un valor aceptable dentro de los límites para que no ocupe mucho almacenamiento en un dispositivo.

```
class Word(object):
    def __init__(self, synset, name, pos, frec):
        self.name = name
        self.pos = pos
        self.frec = frec
        self.translations = getTranslations(synset, name)
        self.synonym = getSynonyms(synset, name)
        self.definition = synset.definition()
        self.sentences = synset.examples()


columns = ['name', 'pos', 'frec']
wordsFrame = pd.read_table('dataSets/allWords.csv',
    engine='python', sep=',', header=None, names=columns, dtype=str)
wordRecords = wordsFrame.to_records()

words = list()
for x in wordRecords:
    synset = getFirstSynset(x.name, x.pos)
    if synset is not None:
        obj = Word(synset, x.name, x.pos, x.frec)
        words.append(obj)
```

Código 8.2: Fragmento de código del *script* que sirve para procesar las palabras.

8.2 API REST del diccionario

En esta sección veremos cómo se ha construido la API REST para que los usuarios puedan buscar nuevas palabras y como se ha desplegado el servicio. El diseño de esta solución se mostró en la sección 7.3.

8.2.1 Construcción

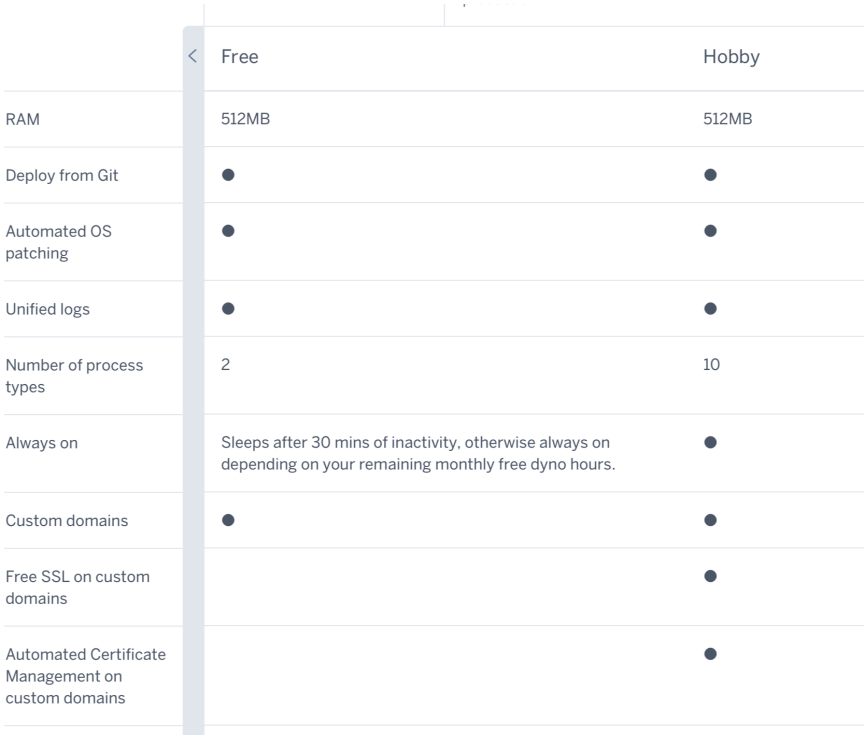
Mediante el framework **Flask** se creó el *endpoint* que necesitamos exponer en esta API. En el código 8.3 podemos ver un fragmento de su implementación, donde se inicializa la aplicación Flask y se define la ruta del endpoint (*/dictionary/<word>*) con el parámetro de entrada *word*.

```
app = Flask(__name__)

@app.route('/dictionary/<word>')
def searchWord(word):
    return getWord(word)
```

Código 8.3: Definición del endpoint GET /diccionary de la API REST del diccionario.

El método *getWord* se encarga de todo, obtendrá la información de la palabra mediante Wordnet, de la misma forma que se hizo en el *script* inicial de la sección 8.1. Finalmente se



	Free	Hobby
RAM	512MB	512MB
Deploy from Git	●	●
Automated OS patching	●	●
Unified logs	●	●
Number of process types	2	10
Always on	Sleeps after 30 mins of inactivity, otherwise always on depending on your remaining monthly free dyno hours.	●
Custom domains	●	●
Free SSL on custom domains		●
Automated Certificate Management on custom domains		●

Figura 8.1: Características de los tipos de cuenta *Free* y *Hobby* de Heroku.

creará la respuesta con la información de la palabra final, y en el caso de que no se encuentre ningún resultado se devolverá un error controlado.

8.2.2 Despliegue

Para desplegar la API en producción y que pueda ser accesible se ha utilizado la plataforma Heroku.

Heroku dispone de distintos tipos de cuenta con diferentes características. Para este proyecto se ha utlizado la cuenta *Free* que tiene alguna limitación. En la figura 8.1 se puede ver una comparación entre las características de una cuenta *Free* y una *Hobby* (ya de pago). Se puede ver que el rendimiento de la cuenta gratuita es menor y que la aplicación caerá en reposo después de 30 minutos de inactividad.

Si el servidor de Heroku se encuentra en reposo, notaremos un pequeño letargo en las peticiones a la API de no más de 3 segundos.

Para poder inicializar la aplicación Flask en Heroku es necesario crear un fichero *requirements.tx* con las dependencias de la aplicación. Por tanto en este fichero estarán por ejemplo las librerías *flask* y *nltk* (para WordNet).

Cuando se quiera subir la aplicación, simplemente habrá hacer *commit* de los cambios y hacer *push* al repositorio de nuestra aplicación en Heroku.

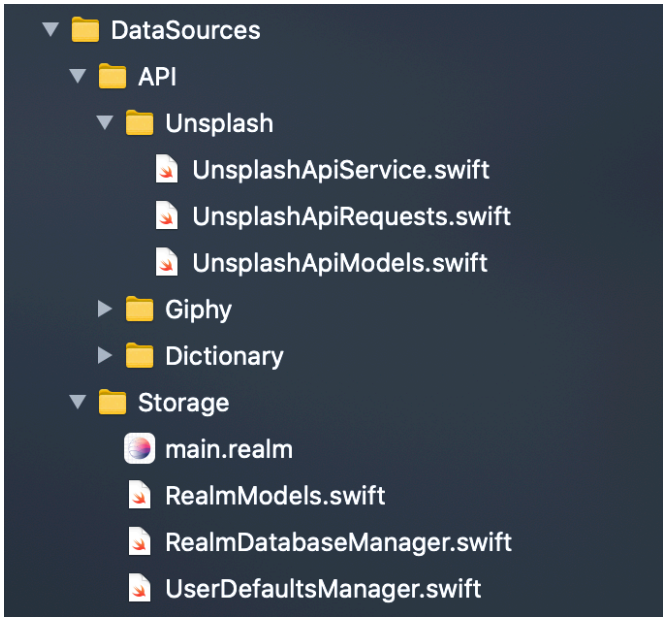


Figura 8.2: Jerarquía en la capa de datos del proyecto iOS de *Vocabulary*.

8.3 Capa de datos de la aplicación iOS

Esta sección hace referencia a los detalles de implementación de la capa de acceso a datos (*Data Sources*), comentada en la arquitectura de la aplicación iOS de la sección 7.7.

Esta es la capa donde residen los datos y la encargada de acceder a los mismos. Se ha dividido en dos partes distintas: el apartado *API* para los clientes de las APIs y *Storage* para el acceso al almacenamiento local. En la figura 8.2 se muestra esta estructura en el proyecto iOS. En las próximas subsecciones veremos cada una de ellas.

8.3.1 APIs

Van a existir 3 clientes de APIs en el proyecto: Unsplash, Giphy y Dictionary.

Cada cliente tendrá una estructura similar a la que se puede ver para Unplash en la figura 8.2. Los clientes tendrán su propio modelo de datos para poder parsear la respuesta de las peticiones. En el caso de que recibamos un código de respuesta HTTP del tipo “4xx” o “5xx”, o exista un error en el parseo, se manejará el error informándole al usuario con una alerta. A continuación veremos más a fondo los detalles de configuración e implementación de las APIs de terceros, Unsplash y Giphy.

Unsplash API

Como vimos en la sección 7.4.2, esta API se usará para recuperar las imágenes. Para poder usarla fue necesario crearse una cuenta como desarrollador en Unsplash.

El endpoint que usaremos de esta API se trata de **GET /search/photos**, construida su petición en el código 8.4. Para esta petición serán suficiente sólo dos parámetros: el *id* de acceso a la API (parámetro *client_id*) y el texto que se quiere buscar (campo *query*). Hay otros parámetros posibles que no han sido necesarios mandar en la petición, ya que utilizamos sus valores por defecto: el número de imágenes por página (10) y el orden de los resultados (*relevant*, las imágenes más relevantes).

En la respuesta de esta petición tendremos las imágenes recibidas con las urls de las imágenes en distintos tamaños. Nosotros sólo utilizaremos la url del atributo *small* de la respuesta, el cual contiene la imagen con un ancho de 400 píxeles.

```
struct GetPicturesApiRequest: ApiRequest {
    let query: String

    var urlRequest: URLRequest {
        var urlComponents = URLComponents(string:
    "https://api.unsplash.com/search/photos/")!

        let clientIdValue = "XXXXXXXXXXXXX"
        urlComponents.queryItems = [
            URLQueryItem(name: "client_id", value: clientIdValue),
            URLQueryItem(name: "query", value: query)
        ]

        let url: URL! = URL(string:
    urlComponents.url!.absoluteString)

        var request = URLRequest(url: url)
        request.httpMethod = "GET"

        return request
    }
}
```

Código 8.4: Petición implementada del endpoint *GET /search/photos* en la aplicación iOS.

Giphy API

Esta será la API que utilicemos para conseguir GIFs, también como se explicó en la sección 7.4.2. De la misma forma que con Unsplash, hubo que crear una cuenta de desarrollador para poder conseguir un *API Key* de pruebas.

Se usará el servicio llamado **Search Endpoint** y la implementación que se ha hecho de esta petición se puede ver en el código 8.5. Aparte de la *key* y del texto de consulta como

parámetros, es necesario pasar el número de GIFs que se quieren obtener (10 en nuestro caso) y el idioma Inglés para mejorar los resultados de la búsqueda.

Los GIFs que se obtienen en la respuesta se encuentran en diferentes formatos y resoluciones. Los formatos existentes son GIF, MP4 y WEBP. Pero según la guía de buenas prácticas de la API se recomienda usar los formatos MP4 o WEBP para mejoras en calidad y rendimiento [47]. Por esa razón hemos elegido soportar los GIFs en el formato MP4, que tendrán un ancho fijo de 200 píxeles.

```
struct GetGifsApiRequest: ApiRequest {
    let query: String

    var urlRequest: URLRequest {
        var urlComponents = URLComponents(string:
    "https://api.giphy.com/v1/gifs/search")!

        let clientIdValue = "XXXXXXXXXXXXX"
        let numResults = "10"
        let lang = "en"
        urlComponents.queryItems = [
            URLQueryItem(name: "api_key", value: clientIdValue),
            URLQueryItem(name: "q", value: query),
            URLQueryItem(name: "limit", value: numResults),
            URLQueryItem(name: "lang", value: lang)
        ]

        let url: URL! = URL(string:
    urlComponents.url!.absoluteString)

        var request = URLRequest(url: url)
        request.httpMethod = "GET"

        return request
    }
}
```

Código 8.5: Implementación del *Search Endpoint* de la API de Giphy en la aplicación iOS.

8.3.2 Almacenamiento local

En este apartado veremos la implementación del gestor de la base de datos. El controlador que manejará toda las consultas y escrituras a la base de datos será *RealmDatabaseManager* (ver en figura 8.2).

La primera vez que la aplicación *Vocabulary* se inicie va a ser necesario que se haga un *setup* del lugar donde se guarde el fichero de base de datos. Esto se debe a que inicialmente la base de datos se encuentra en el *Bundle*, el cual es un lugar sólo de lectura. Entonces lo que haremos en este *setup*, será mover la base de datos al directorio de la aplicación *Document*, que permite tanto lectura como escritura.

```swift
protocol DatabaseProtocol: class {
    func fetchDeckWords(completion: @escaping ([DeckWord]) -> Void)
    func fetchDeckWords(ids: [String], completion: @escaping ([DeckWord]) -> Void)
    func setPictureForDeckWord(withId id: String, picture: WordPicture, completion: (Bool) -> Void)
    func filterDeckWords(text: String, completion: @escaping ([DeckWord]) -> Void)
    func fetchAllWordDetails(completion: @escaping ([WordDetail]) -> Void)
    func updateDeckWordStatus(_ word: DeckWord)
    func fetchWordsToReviewToday(currentDate: Date, completion: @escaping ([DeckWord]) -> Void)
    func fetchWordsByStatus(status: WordStatusParameter, completion: @escaping ([DeckWord]) -> Void)
    func addDeckWord(_ word: Word, picture: WordPicture?, completion: (Bool) -> Void)
    func addSearchedWord(nameWord: String)
    func fetchSearchedWords(completion: @escaping ([SearchedWord]) -> Void)
    func removeSearchedWords(completion: (Bool) -> Void)
}
```

Figura 8.3: Interfaz *DatabaseProtocol* para la gestión de la base de datos en *Vocabulary*.

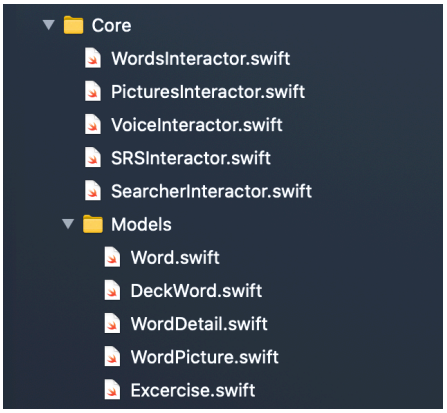


Figura 8.4: Directorio de la capa *Core* del proyecto iOS.

En la imagen 8.3 se puede ver la interfaz del protocolo que define el comportamiento de la base de datos. El controlador *RealmDatabaseManager* va a implementar este protocolo para controlar las operaciones de lectura y escritura. Si en el futuro se quiere dejar de usar Realm por otra tecnología o se quiere tener la base de datos en un servidor, entonces el nuevo controlador deberá implementar el contrato de *DatabaseProtocol*. De esta forma conseguiremos que el resto del código no se vea afectado por el cambio.

No vamos a entrar en detalle de los métodos de la interfaz, ya que lo que hacen son simplemente operaciones de lectura/escritura contra la base de datos.

8.4 Capa *core* de la aplicación iOS

En este apartado vamos a ver la capa core que es el núcleo de la aplicación tal como se vio en la arquitectura de la aplicación en la sección 7.7.

Esta capa contiene las reglas de negocio y los modelos del dominio que encapsulan los datos y los comportamientos necesarios. Las reglas de negocio o casos de uso van a residir en los componentes *Interactors*, los cuales se pueden ver en la figura 8.4, estos son: *WordsInterac-*

```
protocol WordsInteractorProtocol: class {
    func getAllWords(completion: @escaping ([DeckWord]) -> Void)
    func getWords(ids: [String], completion: @escaping ([DeckWord]) -> Void)
    func getRandomWordsToLearn(count: Int, skipIds: [String], completion: @escaping ([DeckWord]) -> Void)
    func savePictureForWord(_ word: Word, picture: WordPicture, completion: ((Bool) -> Void)?)
    func searchWords(text: String, completion: @escaping ([DeckWord]) -> Void)
    func getRandomResponses(count: Int, except wordId: String, completion: @escaping ([WordDetail]) -> Void)
    func getGeneralSummary(completion: @escaping (GeneralSummary) -> Void)
    func getLeaningSummary(completion: @escaping (LearningSummary) -> Void)
    func notStudyWord(_ word: DeckWord)
    func addNewWordToDeck(_ word: Word, picture: WordPicture?, completion: @escaping (Bool) -> Void)
}
```

Figura 8.5: Interfaz *WordsInteractor* de la aplicación iOS.

tor, *PicturesInteractor*, *VoiceInteractor*, *SRSInteractor*, *SearcherInteractor*. Cada uno tendrá unos casos de uso relacionados por alguna temática.

En los próximos apartados daremos los apuntes más importantes de estos *interactors*.

8.4.1 *WordsInteractor*

Contiene los casos de uso que se encargan de pedir las palabras a la capa de acceso de datos y hacer manipulaciones con ellas (ver figura 8.5). Los casos de uso son los siguientes:

- **getAllWords**: Su función es obtener todas las palabras existentes del *deck*.
- **getWords**: Se utiliza para obtener palabras concretas del *deck* a partir de su *id*.
- **getRandomWordsToLearn**: Se encarga de conseguir una serie de palabras del *deck* que sean aptas para aprender. Estas palabras se piden previamente a una nueva lección y deben cumplir varios requisitos: que estén en el estado 1 (palabras nuevas), y que no pertenezcan a la lista *skipIds* de entrada de la función.
- **savePictureForWord**: Su función es asociar un *WordPicture* a una palabra. El objeto *WordPicture* tendrá la información del tipo de representación (imagen o GIF) y su URL.
- **searchWords**: Se usa para buscar palabras del *deck* a partir de las letras que va introduciendo el usuario en un buscador.
- **getRandomResponses**: Se encarga de conseguir una serie de palabras necesarias para cuando se está examinando al usuario y hay que mostrar soluciones alternativas. Es necesario excluir de estas posibles soluciones la palabra correcta del ejercicio, así no saldrán opciones repetidas.
- **getGeneralSummary**: Obtiene el número de palabras que hay en cada estado del *deck* para información estadística.

```
protocol PicturesInteractorProcol: class {
    func getPictures(for word: Word, completion: @escaping ([WordPicture]) -> Void)
}
```

Figura 8.6: Interfaz *PicturesInteractorProtocol* de la aplicación iOS.

```
protocol VoiceInteractorProtocol: class {
    func playText(_ text: String)
}
```

Figura 8.7: Interfaz *VoiceInteractorProtocol* de la aplicación iOS.

- **getLearningSummary**: Obtiene el número de palabras que hay en cada fase de aprendizaje para información estadística.
- **notStudyWord**: Su propósito es actualizar una palabra del *deck* al estado 5 (palabras descartadas). Se utiliza antes de empezar una lección, cuando se proponen posibles palabras para aprender y el usuario descarta alguna de ellas.
- **addNewWordToDeck**: Para añadir una nueva palabra con su posible imagen al *deck*. Se utiliza desde la funcionalidad de diccionario.

8.4.2 *PicturesInteractor*

En la figura 8.6 se puede ver el caso de uso **getPictures**. Con este caso de uso a partir de una palabra se obtendrá una lista de posibles representaciones visuales. Se encarga de decidir el API que hay que usar (Unsplash o Giphy) según la categoría gramatical.

8.4.3 *VoiceInteractor*

En la figura 8.7 se encuentra el caso de uso **playText** de la interfaz *VoiceInteractorProtocol*. Esta interfaz permitirá convertir texto a voz. En el método *playText* se configura la pronunciación del habla y su velocidad.

```
protocol SRSInteractorProtocol: class {
    func getReviewWords(completion: @escaping ([DeckWord]) -> Void)
    func generateLessonExercises(wordsToStudy: [DeckWord]) -> [Exercise]
    func rightExercise()
    func wrongExercise(_ exercise: Exercise)
    func lessonCompleted() -> LessonSummaryValues
}
```

Figura 8.8: Interfaz *SRSInteractorProtocol* de la aplicación iOS.

8.4.4 *SRSInteractor*

La lógica del sistema de repetición espaciada (o *Spaced Repetition System*) se encontrará en este *Interactor* cuya interfaz se puede ver en la figura 8.8. Las claves del diseño de este sistema se vieron en el apartado 7.5. Los casos de uso son los siguientes:

- **getReviewWords**: Se encarga de obtener todas las palabras del *deck* cuya fecha de repaso es el día actual o una fecha pasada. Puede ser que haya palabras pendientes de otros días por repasar. Se solicitan para preparar la lección del día.

- **generateLessonExercises**: A partir de las palabras de la lección, tanto las nuevas como las de repaso, este método se encarga de obtener una lista de ejercicios desordenados. El objeto *Exercise* guarda información sobre la palabra a estudiar y del tipo de ejercicio qué visualizar. De momento sólo habrá un tipo de ejercicio que es el de cuestionario, pero en el futuro se podría ampliar con otros tipos de ejercicios.

- **rightExercise**: Cada vez que un ejercicio se responde bien se llamaría a este método. Lleva la cuenta de los ejercicios correctos de la lección.

- **wrongExercise**: Cada vez que un ejercicio se responde mal se llamaría a este método. Lleva la cuenta de los ejercicios incorrectos de la lección. Además, marcaría la palabra como fallida y así cuando termine la lección esta palabra se penalizará moviéndose a un estado anterior.

- **lessonCompleted**: Este método se lanza cuando termina una lección y devolverá los resultados finales de esta. También promociona o degrada el nivel de la palabra según la respuesta calculando la nueva fecha de repaso. En el código 8.6 podemos ver como se haría una promoción de una palabra que ha sido respondida correctamente a la primera.

```
1 private func promoteWord(_ word: DeckWord) {
2         word.status = word.status.promote()
3 
4         let currentDate = Date()
5         var dateComponent = DateComponents()
6         dateComponent.day = word.status.waitingDays()
7         let futureDate = Calendar.current.date(byAdding:
      dateComponent, to: currentDate)
8         word.nextReview = futureDate
9     }
```

Código 8.6: Método *promoteWord* para subir de nivel una palabra y calcular su próxima fecha de repaso.

```
protocol SearcherInteractorProtocol: class {
    func searchWord(_ word: String, completion: @escaping (Word?, Error?) -> Void)
    func saveRecentSearches(wordName: String)
    func getRecentSearches(completion: @escaping ([SearchedWord]) -> Void)
    func clearRecentSearches(completion: @escaping (Bool) -> Void)
}
```

Figura 8.9: Interfaz *SearcherInteractorProtocol* de la aplicación iOS.

8.4.5 *SearcherInteractor*

Este *interactor* contará con los casos de uso relacionados con el diccionario, se puede ver su interfaz en la figura 8.9.

El método **searchWord** se llamará cuando se requiere buscar una palabra en el diccionario, y su función será la de solicitar la palabra al cliente de la API del diccionario y transformar la respuesta a un modelo válido. El resto de métodos se usarán para tareas relacionadas con las búsquedas recientes.

8.5 Interfaz de usuario

En esta sección mostraremos las pantallas más importantes que se han construido y la navegación entre ellas.

8.5.1 Guías de diseño

El diseño de las interfaces se ha creado basándonos en las recomendaciones que propone Apple en la guía *Human Interface Guidelines* [48]. Esta guía informa de como deberían ser las interfaces de la plataforma iOS, para que todas las aplicaciones tengan un estilo similar. De esta forma se conseguirá que los usuario estén más cómodos y entiendan fácilmente las interfaces.

Los componentes visuales reutilizables que se pueden usar son accesibles desde la librería nativa *UIKit*, con ella se podrán añadir por ejemplo botones, tablas, barras de navegación, alertas, etc... [16]. Son altamente adaptables, por lo que será fácil adaptar los componentes a nuestro gusto y ofrecer versatilidad.

Para mantener una coherencia en todos los textos, se ha usado una fuente común para ellos, la fuente *San Francisco*. Es una de las más usadas en iOS ya que proporciona muchos estilos diferentes. En la figura 8.10 podemos ver estos estilos, con su peso y tamaño sacados de la guía de diseño de Apple.

La interfaz de la aplicación *Vocabulary* es adaptable a distintos tamaños de pantalla, para que así se visualice bien en móviles de 5 pulgadas y hasta 8 pulgadas. Para conseguirlo se han

Large (Default)

Style	Weight	Size (points)	Leading (points)
Large Title	Regular	34	41
Title 1	Regular	28	34
Title 2	Regular	22	28
Title 3	Regular	20	25
Headline	Semi-Bold	17	22
Body	Regular	17	22
Callout	Regular	16	21
Subhead	Regular	15	20
Footnote	Regular	13	18
Caption 1	Regular	12	16
Caption 2	Regular	11	13

Figura 8.10: Estilos diferentes de la tipografía *San Francisco*.

definido reglas (o *constraints*) en las vistas que determinan las restricciones de tamaños, de posiciones y de separación de los elementos.

Se ha decidido restringir la aplicación sólo para dispositivos iPhone, ya que el diseño para iPad implica otros desarrollos. En iPad al tener una pantalla mucho más grande, la posición de los componentes debería ser distinta, como también la navegación entre pantallas. Si se quisiese adaptar la aplicación también para iPad, sólo sería necesario añadir las vistas que se desean tener diferentes.

8.5.2 Pantallas y navegación

En este apartado vamos a ver los flujos principales de la aplicación. Estos serían el flujo *Flashcards* y el flujo *Diccionario*. Además mostraremos otras pantallas secundarias.

Flujo *Flashcards*

En este flujo el usuario podrá acceder a una lección y empezar a repasar o aprender palabras con las *flashcards* que se presenten. Se dividiría entre 4 pantallas distintas como se puede ver en la figura 8.11 y se detallan a continuación:

1. Es la pantalla inicial, en ella se van a mostrar el número de las palabras del día, divididas entre programadas (de repaso) y nuevas. El usuario podrá empezar la lección desde aquí.

2. Este tipo de pantalla mostrará palabras aleatorias que nunca se han visto y se quieren aprender. Si alguna de estas palabras al usuario no le interesa las puede descartar. Hay que seleccionar tantas palabras como palabras nuevas por lección se haya configurado. Además de ver información de las palabras, el usuario podrá elegir una imagen de la palabra.

3. Una vez el usuario haya seleccionado las palabras nuevas, comenzará la lección. El usuario tendrá que ir respondiendo preguntas secuencialmente y si falla alguna le volverá a salir. Existen dos modos o vistas diferentes según haya una imagen o no.

   (a) Como el usuario tiene asociado a la palabra una fotografía o GIF, se le mostrará en la pantalla a modo de pregunta.

   (b) Si no hay imagen asociada, se mostrará como pregunta la traducción de la palabra en español.

4. Por último, cuando terminen todas las preguntas de la lección, se mostrará la pantalla de resultados con el total de respuestas correctas e incorrectas. Habrá una animación de confeti realizada con la librería *SwiftConfettiView* (4.2.4).

Flujo *Diccionario*

En el caso de que el usuario quiera buscar palabras desconocidas, podrá utilizar la funcionalidad de diccionario. En la figura 8.12a se muestra la pantalla donde es posible introducir texto para hacer una búsqueda. También se encuentra el historial de búsquedas recientes.

Cuando se encuentre un resultado de lo que el usuario busca, se podrá ver la ficha de la palabra encontrada, que corresponderá con el resultado más frecuente. Se puede ver en la figura 8.12b. Si el resultado de la palabra no está ya en el *deck* del usuario, se mostrará una opción para poder añadirlo.

Otras pantallas

Existen otras pantallas secundarias dentro de *Vocabulary*, por ejemplo una para ver estadísticas y otra de ajustes.

En la figura 8.13a podemos ver la pantalla dónde se muestra información estadística del *deck*. En el apartado *Resumen* se contabilizan el número de palabras en cada estado posible (no vistas, en progreso, aprendidas y descartadas). Y en el apartado *Estado de palabras en progreso*, podemos ver un gráfico circular para conocer la proporción de las palabras que están en cada fase del sistema de repetición espaciada. El gráfico circular se ha implementado con la librería *Charts* (4.2.3).

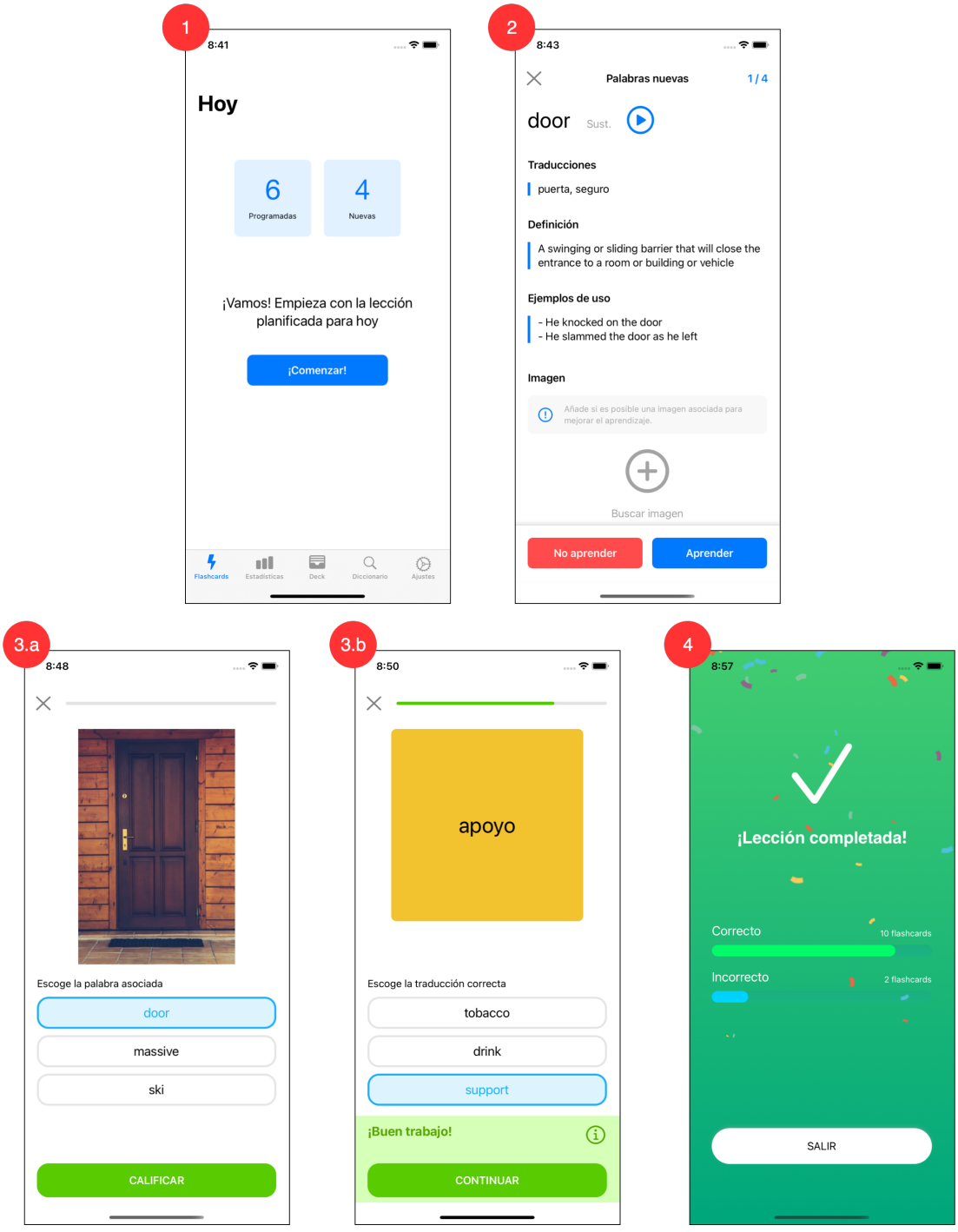
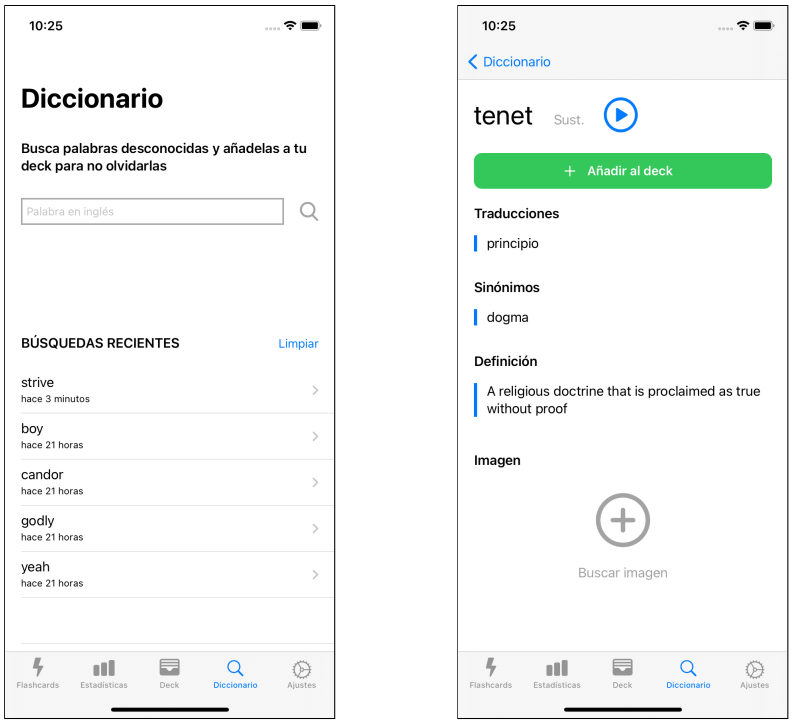
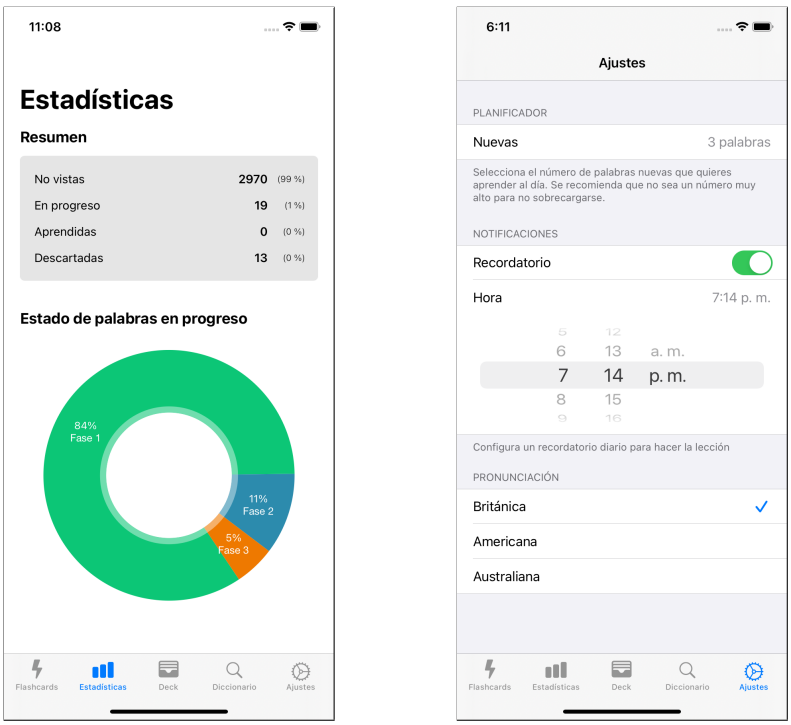


Figura 8.11: Secuencia de pantallas del flujo *Flashcards* de *Vocabulary*.



(a) Pantalla de búsqueda (b) Pantalla de resultados

Figura 8.12: Secuencia de pantallas del flujo *Diccionario* de *Vocabulary*.



(a) Pantalla de estadísticas (b) Pantalla de ajustes

Figura 8.13: Pantallas secundarias de *Vocabulary*.

Para poder configurar algunos parámetros, el usuario dispone de la pantalla de ajustes que se puede ver en la figura 8.13b. En ella se podrán seleccionar el número de palabras nuevas que se quieren aprender al día, la posibilidad de establecer un recordatorio diario para realizar la lección, y también un selector para elegir el acento de las palabras (británico, americano o australiano).

Capítulo 9

Conclusiones

En este último capítulo expondremos una valoración final del proyecto, las lecciones aprendidas y el posible trabajo futuro a realizar.

9.1 Valoración final

Finalmente consideramos que se han logrado cumplir con éxito todos los objetivos establecidos inicialmente y resolver adecuadamente tanto los requisitos funcionales como no funcionales.

Se ha conseguido elaborar una base de datos del vocabulario más frecuente en inglés mediante la librería *WordNet*, para posteriormente incorporar esta base de datos a la aplicación iOS. De esta forma se dispondrá de un conjunto de palabras con la información que nos interesa, sin necesidad de utilizar APIs externas ni internet.

La aplicación iOS desarrollada puede ser de mucha utilidad para personas que estén aprendiendo inglés y les interese aprender vocabulario. El modo de estudio que ofrece la aplicación es mediante lecciones interactivas con cuestionarios y flashcards, que intentan mediante la gamificación, que el aprendizaje resulte entretenido.

Para que las lecciones de estudio sean eficientes y se minimicen los olvidos, se ha diseñado un sistema de repetición espaciada, en el cual, el vocabulario tiene que superar tres etapas o estados diferentes para considerarse como aprendido. De esta forma el aprendizaje será progresivo y se conseguirá que las palabras aprendidas se recuerden de una manera más efectiva a largo plazo.

Para que el usuario conozca sus avances en su aprendizaje, se ha creado un apartado de estadísticas para poder ver el progreso y el estado de todo el vocabulario de la aplicación.

Se ha creado una API REST que pueda ser consumida desde nuestra aplicación iOS, con el propósito de crear un diccionario para buscar nuevas palabras. La API creada se ha desplegado en un servidor de la plataforma Heroku, de tal manera que pueda ser accedida desde cualquier

dispositivo con conexión a internet.

La aplicación iOS permite escuchar la pronunciación del vocabulario a través de un sistema 'text-to-speech'. Además, también se permite buscar imágenes en los servicios de terceros, Unsplash y Giphy, a través de sus APIs. Las imágenes obtenidas serán de utilidad para asociarlas con el vocabulario de la aplicación y facilitar su aprendizaje.

Se le permitirá al usuario hacer configuraciones en la aplicación a través de la pantalla de ajustes. En ella podrá configurar el número de palabras nuevas que quiere aprender al día, el acento que le interesa escuchar, y establecer un recordatorio periódico en forma de notificación.

La arquitectura VIPER que se ha adoptado en la aplicación móvil, hace que se formalice un *software* bien dividido y que ayuda a seguir los principios de buenas prácticas de desarrollo. Esta arquitectura favorecerá la extensión con funcionalidades futuras y ayudará en su mantenimiento.

En definitiva la aplicación *Vocabulary* cumple su cometido y además, tiene un diseño sencillo e intuitivo que encaja bien en el ecosistema de aplicaciones iOS.

9.2 Lecciones aprendidas

En este proyecto se ha aprendido a gestionar el trabajo con una metodología ágil como es *Scrum*. Gracias a esta metodología se han ido sacando versiones incrementales del producto final, y también ha permitido una organización flexible a lo largo del transcurso del proyecto.

Se han conocido librerías muy útiles de python como *WordNet* y *Pandas*, que permiten crear *scripts* centrados en la manipulación y extracción de datos.

Se ha profundizado en el desarrollo de aplicaciones para el entorno iOS. Previamente ya se tenía un conocimiento de las tecnologías para desarrollar este tipo de aplicaciones, pero se han conjugado con la arquitectura VIPER para conseguir una aplicación robusta y bien implementada.

Se ha aprendido como deben diseñarse las aplicaciones iOS para cumplir con las guías de diseño que propone Apple. Cada componente visual es necesario saber que tiene un sitio oportuno para usarse dentro de las aplicaciones, para así tener una coherencia con el resto de las aplicaciones de la plataforma iOS.

9.3 Trabajo futuro

Consideramos que la solución creada es un MVP o producto mínimo viable, ya que contiene las funcionalidades básicas que un usuario pueda necesitar de una aplicación de este tipo. El siguiente paso sería subir la aplicación a la *App Store*, para conocer la opinión de usuarios

reales y saber qué enfoque tomar para seguir mejorándola. Para ello sería muy útil añadir analíticas e ir recopilando las interacciones de los usuarios y sacar conclusiones.

Sería interesante seguir añadiendo nuevas funcionalidades para evolucionar la aplicación. Algunas de las ideas que se tienen en mente son las siguientes:

- La posibilidad de poder hacer un *backup* de los datos, para que así no se pierda el progreso si se cambia de dispositivo. Se podría usar la funcionalidad nativa de iCloud que ofrece iOS, una vez subida al *App store*.
- Actualmente sólo es posible aprender vocabulario de inglés desde el castellano. Por eso estaría bien soportar que se puedan obtener traducciones en otros idiomas y así, conseguir ampliar la audiencia.
- Una nueva funcionalidad de buscar palabras desde otras aplicaciones del móvil o incluso desde la extensión de un navegador. Así por ejemplo, si un usuario está leyendo una noticia en inglés y hay una palabra que desconoce, podría buscarla y añadirla directamente a su *deck* con esta funcionalidad.
- Incluir más tipos de juegos en las lecciones. Podrían ser juegos con el objetivo de mejorar más aspectos de las palabras como la pronunciación, la escucha y la escritura. También se podrían usar las definiciones, ejemplos de uso y sinónimos, para aumentar el contenido de los juegos.
- Una nueva característica con la que el usuario pueda subir sus propias imágenes a la aplicación, y que esta sea capaz de *taggear* las imágenes con los elementos que se visualicen. Para conseguirlo existen varias APIs como la de Clarifai con la que se pueden sacar palabras a partir de imágenes [49]. Sería una nueva forma de conocer y buscar vocabulario.
- Incluir más técnicas de gamificación, como por ejemplo: sistema de puntos para subir de nivel, recompensas, etc... También conseguir que la aplicación sea social, con la posibilidad de jugar entre usuarios y que existan *rankings*.

Índice de figuras

Índice de tablas

Índice de códigos

Bibliografía

[1] “Ficha de la aplicación duolingo en la app store.” [En línea]. Disponible en: https://apps.apple.com/app/id570060128

[2] “Análisis de aplicaciones.” [En línea]. Disponible en: https://sensortower.com

[3] “Aplicación bright.” [En línea]. Disponible en: https://engbright.com

[4] “Hermann ebinghaus.” [En línea]. Disponible en: https://psicologiaymente.com/biografias/hermann-ebbinghaus

[5] “Curva del olvido.” [En línea]. Disponible en: https://psicologiaymente.com/psicologia/curva-del-olvido

[6] “Sistema de leitner,” método de repetición espaciada. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Leitner_system

[7] “Anki,” software de repetición espaciada. [En línea]. Disponible en: https://apps.ankiweb.net/

[8] “Supermemo,” software de repetición espaciada. [En línea]. Disponible en: https://www.supermemo.com/

[9] “Wordnet.” [En línea]. Disponible en: https://wordnet.princeton.edu/

[10] “Buscador wordnet,” acceso web a WordNet. [En línea]. Disponible en: http://wordnetweb.princeton.edu/perl/webwn/

[11] “Oxford api.” [En línea]. Disponible en: https://developer.oxforddictionaries.com

[12] “Merriam-webster api.” [En línea]. Disponible en: https://dictionaryapi.com

[13] “Viper and clean architecture,” análisis de la arquitectura VIPER. [En línea]. Disponible en: https://medium.com/build-and-run/clean-architecture-en-ios-viper-893c8c3a75a4

[14] "Lenguaje swift." [En línea]. Disponible en: https://swift.org

[15] "Lenguaje python." [En línea]. Disponible en: https://www.python.org

[16] "Uikit framework." [En línea]. Disponible en: https://developer.apple.com/documentation/uikit

[17] "Realm mobile database." [En línea]. Disponible en: https://realm.io

[18] "Librería charts." [En línea]. Disponible en: https://github.com/danielgindi/Charts

[19] "Librería swiftconfettiview." [En línea]. Disponible en: https://github.com/ugurethemaydin/SwiftConfettiView

[20] "Flask framework." [En línea]. Disponible en: https://flask.palletsprojects.com/en/1.1.x/

[21] "Librería de python nltk." [En línea]. Disponible en: https://www.nltk.org/

[22] "Librería de python pandas." [En línea]. Disponible en: https://pandas.pydata.org/

[23] "Librería de python json." [En línea]. Disponible en: https://docs.python.org/3/library/json.html

[24] "Librería de python googletrans." [En línea]. Disponible en: https://pypi.org/project/googletrans/

[25] "Unsplash api," aPI para obtener imágenes. [En línea]. Disponible en: https://unsplash.com/developers

[26] "Giphy api," aPI para obtener GIFs. [En línea]. Disponible en: https://developers.giphy.com

[27] "Xcode ide." [En línea]. Disponible en: https://developer.apple.com/xcode/

[28] "Spyder ide." [En línea]. Disponible en: https://www.spyder-ide.org

[29] "Git." [En línea]. Disponible en: https://cocoapods.org

[30] "Heroku platform." [En línea]. Disponible en: https://www.heroku.com/platform

[31] "Cocoapods." [En línea]. Disponible en: https://cocoapods.org

[32] "Trello." [En línea]. Disponible en: https://trello.com

[33] "Overleaf." [En línea]. Disponible en: https://es.overleaf.com

[34] "Draw.io." [En línea]. Disponible en: https://es.overleaf.com

[35] "Scrum." [En línea]. Disponible en: https://www.atlassian.com/es/agile/scrum

[36] "Wordfrecuency," página web con corpus de palabras en inglés. [En línea]. Disponible en: https://www.wordfrequency.info/intro.asp

[37] "Amazon polly," aPI de Amazon para TTS. [En línea]. Disponible en: https://aws.amazon.com/es/polly/

[38] "Google cloud text-to-speech," aPI de Google para TTS. [En línea]. Disponible en: https://cloud.google.com/text-to-speech/?hl=es

[39] "Speech synthesis by apple," framework de Apple para TTS. [En línea]. Disponible en: https://developer.apple.com/documentation/avfoundation/speech_synthesis

[40] "Unsplash photo picker para ios," librería nativa de Unsplash. [En línea]. Disponible en: https://github.com/unsplash/unsplash-photopicker-ios

[41] "Giphy sdk," sDK para integrar GIPHY en iOS. [En línea]. Disponible en: https://github.com/Giphy/giphy-ios-sdk-ui-example

[42] "Documentación unsplash api," documentación del API para desarrolladores. [En línea]. Disponible en: https://unsplash.com/documentation#photos

[43] "Search endpoint giphy," endpoint para buscar GIFs. [En línea]. Disponible en: https://developers.giphy.com/docs/api/endpoint#search

[44] "Cloudkit framework." [En línea]. Disponible en: https://developer.apple.com/icloud/cloudkit/

[45] "Object database," información sobre las bases de datos de objetos. [En línea]. Disponible en: https://en.wikipedia.org/wiki/Object_database

[46] "Información sobre como precargar una base de datos realm." [En línea]. Disponible en: https://fluffy.es/preload-database-in-app-using-realm/

[47] "Rendition guide giphy," mejores prácticas Giphy para visualización de GIFs. [En línea]. Disponible en: https://developers.giphy.com/docs/optional-settings/#rendition-guide

[48] "Human interface guidelines," guías de diseño de Apple para iOS. [En línea]. Disponible en: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/

[49] "Clarifai api," aPI para tagear imágenes. [En línea]. Disponible en: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/