Universidad Politécnica
de Madrid

Escuela Técnica Superior de

Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Desarrollo de una Aplicación Web para la Identificación de Neologismos

Autor: Roberto Montiel Rodríguez

Tutor(a): Raquel Cedazo León

Madrid, mayo de 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Desarrollo de una Aplicación Web para la Identificación de Neologismos
Mayo de 2022

Autor: Roberto Montiel Rodríguez

Tutor:
Raquel Cedazo león
Ingeniería Eléctrica, Electrónica Automática y Física Aplicada
ETSI Informáticos
Universidad Politécnica de Madrid

A mis padres, mi hermana y mis abuelos, que me han apoyado a lo largo de toda mi vida y especialmente en los momentos difíciles.

A mi novia, por su cariño y apoyo constante hacia mi trabajo y mis capacidades.

A mis amigos y compañeros, sin los que habrían sido cuatro años mucho más aburridos.

A mí, por seguir consiguiendo unos objetivos y sueños que seguirán creciendo.

"No son las habilidades lo que demuestra lo que somos, son nuestras decisiones" – Albus Dumbledore

Resumen

El vertiginoso desarrollo de la tecnología hace que constantemente aparezcan nuevos conceptos a los que hay que asignar un nombre. Nuestra condición de miembros de una universidad tecnológica como la Universidad Politécnica de Madrid nos ubica en una ubicación perfecta para observar este fenómeno de la creación de nuevos conceptos y sus denominaciones.

Por esto, se decidió comenzar un proyecto para el desarrollo de una aplicación web que permitiera al alumnado de las diferentes escuelas de la UPM aportar neologismos en el campo del Internet de las Cosas, pudiendo ser revisados por lingüistas. Para fomentar el uso de esta aplicación y, por lo tanto, del aprendizaje sobre estos nuevos términos y la curiosidad de buscar más, se utilizará el conocido método de la gamificación, explicando cómo se ha ido comprobando que da resultado en una gran variedad de situaciones.

En este documento se analizará todo el proceso de desarrollo de la aplicación, desde el diseño de la base de datos hasta la implementación de los métodos de la API REST.

Abstract

As the development of technology constantly produces new concepts which need a new name, we, as members of a technological university, are in a perfect position for observing this phenomenon.

That is why a new project started with the purpose of developing a web application that would allow students from different schools at the Polytechnic University of Madrid to contribute with neologisms in the field of Internet of Things, which can be reviewed by linguists. To promote the use of this application and thus the students' learning about these new terms and curiosity in looking for more, we will use gamification, explaining how it has been found to be successful in a variety of situations.

This document will analyze the development process of the application, from the design of the database to the implementation of the REST API.

Tabla de contenidos

1. Introducción

Desde el nacimiento de internet en 1983 hasta el día de hoy con tecnologías como la realidad virtual y aumentada, la ciencia de datos y Machine Learning, el internet de las cosas, etc., ha surgido una enorme cantidad de términos nuevos, o neologismos, que describen nuevas herramientas, situaciones o roles y que no se pueden especificar con palabras ya existentes. Más concretamente, y según la RAE [1], un neologismo es:
1. m. Ling. Vocablo, acepción o giro nuevo en una lengua.

Ante este cambio tan rápido del vocabulario es imposible formalizar toda o gran parte de la nueva terminología, por lo que tan sólo los términos más utilizados o que llevan más tiempo entre nosotros (e.g. “antivirus”) son aceptados por las organizaciones dedicadas a la formalización de las distintas lenguas como la Real Academia Española [1].

Es de este problema que nace la necesidad de llevar un registro de los neologismos que son utilizados en el día a día relacionados con las nuevas tecnologías o el ámbito científico. Con este objetivo y con la iniciativa de Ciencia Ciudadana [2] nació este proyecto, queriendo llevar un registro de los neologismos de uso común para más tarde analizarlos y estudiarlos. La idea es que personas de distintos entornos relacionados con la ciencia colaboren con propuestas de neologismos que más tarde serán revisados y aceptados o rechazados por un lingüista.

Para fomentar la actividad desinteresada de los usuarios del proyecto se decidió que, entre los distintos mecanismos ya existentes de motivación de la actividad de usuarios, el mejor y más adecuado para la ocasión era el de gamificación. El término gamificación se exploró por primera vez en 2008 por cincuenta expertos en el libro *The Gameful World* [3]. Más tarde se modeló la definición de gamificación como el uso de técnicas, elementos y dinámicas propias de los juegos y el ocio en actividades necesariamente recreativas con el fin de potenciar la motivación [4]. Inicialmente se relacionó con la educación, para más tarde extenderse al márquetin y otros ámbitos relacionados especialmente con el crecimiento digital.

Debido a que la idea inicial requería de una aplicación de considerable magnitud, se decidió repartir el diseño y desarrollo del proyecto en dos partes.

La primera es la parte correspondiente a la interfaz o Front-End, la parte del programa que se relaciona directamente con el usuario y que le muestra las estructura visualmente con diseños, colores, información… Esta sección también comprende el diseño de la gamificación de la aplicación y fue realizada en su mayor parte el alumno Javier Barragán Haro, aunque para una mejor comprensión del funcionamiento y estructura de la aplicación, se describirán algunas de las secciones de la interfaz, ya que están muy relacionadas con el resto y ha sido necesario comprenderlas y modificarlas.

La segunda parte del proyecto es el diseño e implementación del código de fondo o Back-End de la aplicación, la cual corresponde a este documento y se

describirá parte a parte. Este apartado es el que se encarga de la gestión de los datos de la aplicación, con la base de datos, la autenticación y creación de usuarios, etc., pero no se relaciona directamente con el usuario final, sino que lo hace con la interfaz.

Todo este proyecto no sería posible sin la ayuda de personal cualificado en la lengua española, como es el departamento de lingüística de la Universidad Politécnica de Madrid, y más concretamente la profesora María de la Nava Maroto García y la becaria Noelia Rodríguez Rodríguez, además por supuesto de la tutora Raquel Cedazo León.

1.1 Objetivos

A continuación, se describen los objetivos tanto generales como específicos de este proyecto.

1.1.1 Objetivos generales

- Diseño y desarrollo de una aplicación web de recolección de neologismos con mecanismos de gamificación integrados en ella.
- Realización y análisis de pruebas con usuarios reales.
- Conseguir una aplicación web fiable, segura, agradable para los usuarios y que se ajuste lo máximo posible a los requisitos.

1.1.2 Objetivos específicos

- Desarrollo de una memoria (este documento) que sirva de documentación del proceso de diseño y construcción de la parte trasera de la aplicación.
- Elección de tecnologías para el desarrollo del Back-End de la aplicación.
- Implementación del código de fondo de la aplicación.
- Mejora de la interfaz de la aplicación donde se requiera.
- Pruebas con usuarios reales.
- Despliegue de la aplicación.

2. Estado del arte

2.1 Gamificación

Uno de los principales objetivos de este proyecto es el de la introducción de la gamificación en una aplicación educativa y con carácter también científico. Es por ello por lo que se requiere dar un contexto del porqué de la gamificación, ya que es un concepto que ha sido estudiado a lo largo de décadas, especializándose en los últimos años en la aplicación en software dirigido al usuario medio.

Cómo ya se ha comentado antes, la gamificación es un conjunto de técnicas y dinámicas que ayudan a incrementar la motivación de un usuario para involucrarse, en nuestro caso, en la actividad de una aplicación y a su uso a largo plazo. Pero ¿por qué es tan importante la gamificación para el éxito de una aplicación?

2.1.1 La importancia de la gamificación

Ya en 1971 Piaget [5], y más tarde Malone y Lepper en 1980 [6] y 1987 [7] remarcaban que el uso del juego en la educación podía potenciarla en gran medida. Hay muchos más ejemplos de estudios y contemplaciones acerca del concepto de utilizar juegos para implicar a las personas en diversas tareas, pero es de especial interés el estudio de Brewer et al. en 2013 [8], que concluyó que, tras estudiar el uso de la gamificación en niños de 5 a 7 años, el porcentaje de tareas acabadas era significativamente mayor.

Aun así, crear las técnicas particulares que implementen la gamificación de una forma efectiva no es una tarea nada sencilla, por lo que también se han hecho estudios sobre cómo se debe hacer. Ejemplos de esto son el modelo de los profesores B.J Fogg en 2009 [9] o Mihaly Czikszentmihalyi en 1975 [10]. En el primero se postula que para que una mecánica de gamificación sea exitosa, se requiere de motivación, habilidad y causa, mientras que el segundo concreta que se debe buscar una zona entre lo aburrido y lo estresante en la que el usuario se pueda concentrar en la tarea que está realizando.

Si lo llevamos a la actualidad, aparte de la educación, hay muchas otras áreas en las que se está usando, pero una en la que la mayoría de las jóvenes de hoy en día se ve involucrada es la de las aplicaciones, ya sean aplicaciones web, aplicaciones para dispositivos móviles, etc.

Esto es lo que postulan Eduardo Herranz Sánchez y Ricardo Colomo-Palacios en su escrito de 2012 [11], en el que se habla de la gamificación como un agente de cambio en el ámbito de la mejora del proceso y la innovación en entornos software.

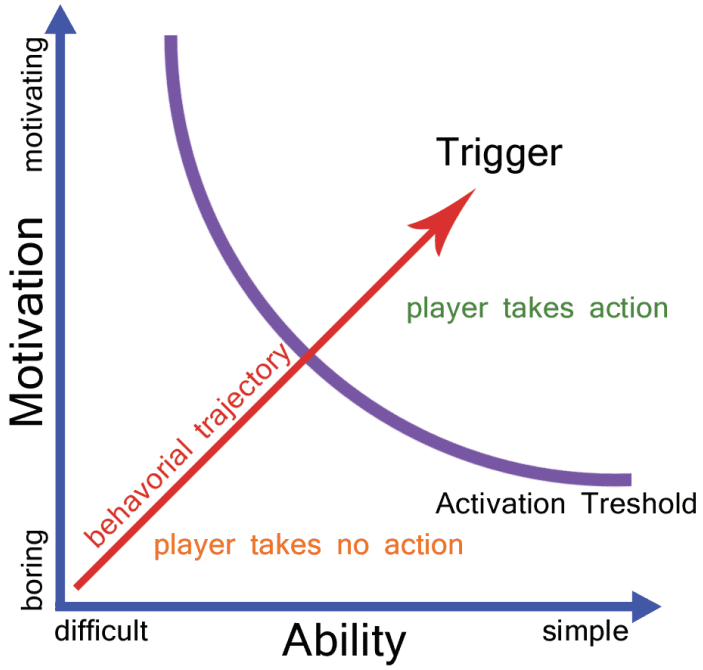


Ilustración 1. Modelo de Fogg [9]

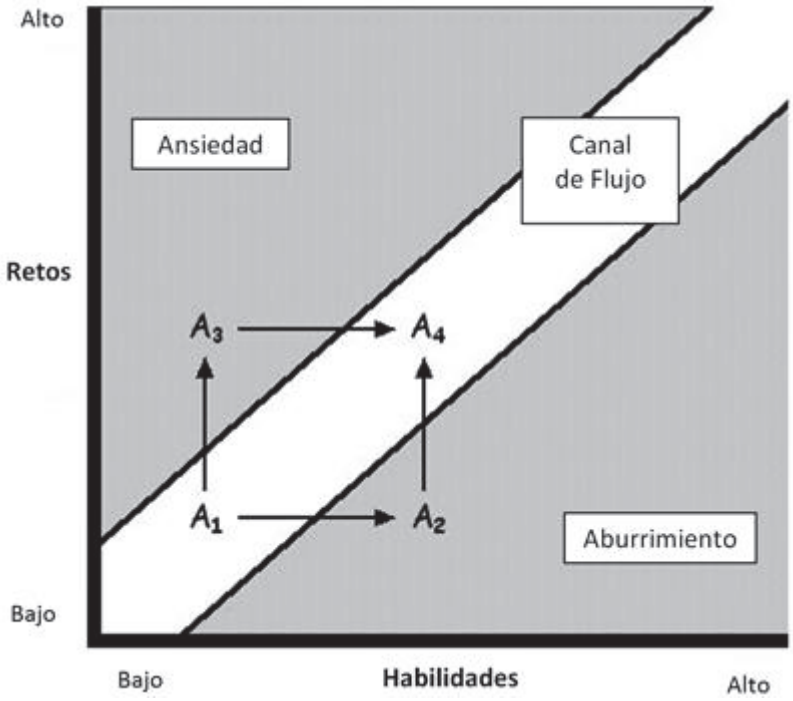


Ilustración 2. Modelo de Czikszentmihalyi [10]

Según un estudio de la empresa Appboy (ahora llamada Braze) [12], tan solo un 25 por ciento de los usuarios de las aplicaciones móviles regresan a la aplicación al día siguiente de usarla por primera vez, y este porcentaje no hace más que bajar, hasta un 4,1% en el día 90. Esto remarca lo difícil que es mantener a los usuarios motivados (o "retenerlos") en la actualidad. Por tanto, con la gamificación en las aplicaciones lo que se busca es que el usuario regrese a su uso lo más frecuentemente posible, pero ¿qué técnicas concretas son las que se utilizan?

2.1.2 Técnicas actuales de gamificación en aplicaciones

Algunas de las técnicas que se llevan usando unos años y que podremos encontrar en el futuro también son las siguientes, que destacan por su efectividad y sencillez:

- Beneficios y recompensas. Esta es la técnica más común y simple, ya que promueve el uso continuado del software y a medida que avance en su uso, dar algún tipo de premio.
- Metas y logros. Esta técnica es muy similar a la anterior, pero las recompensas se dan solo cuando llegue a ciertos hitos que pueden ser algo más difíciles de alcanzar.
- Competición. Este recurso se combina con el anterior, creando una clasificación de usuarios relacionada con los logros y metas que ha conseguido cada uno, fomentando la competición entre ellos.
- Estatus. También relacionada con las metas y logros, una de las recompensas que se le pueden brindar al usuario es la de ponerle una etiqueta, cuyas características mejoren según es más complicado conseguir el logro.

2.1.3 Las ventajas de la gamificación en aplicaciones

Según la emprendedora Vanessa Estorach Cavaller [13], hay ciertas ventajas del uso de la gamificación en las aplicaciones [14], como son las siguientes:

- Aumento de la retención y mejora de la fidelización. Como ya se ha comentado, es muy importante motivar al usuario al uso de la aplicación en cuestión, y consiguiendo esto también logramos que se elija esta y no otra aplicación ya que consigue marcar una diferencia.
- Enriquecimiento de la experiencia de usuario. Es a través del entretenimiento como el usuario cambia su percepción de una aplicación, pasando a ser un estímulo más allá de la satisfacción de una necesidad determinada.
- Guía para el usuario. A través de la gamificación se puede ir guiando al usuario para que conozca todas las funcionalidades de la aplicación y aprenda a utilizar cada una de ellas.

2.1.4 Ejemplos de gamificación en aplicaciones

- **Kahoot** [15]. Cuando la mayoría de los alumnos de instituto o universidad piensa en gamificación en las aulas, se imagina el juego de Kahoot, una aplicación tanto web como móvil que se centra en la mecánica de gamificación de competición, que ya hemos comentado antes. Es común que para implicar al alumnado en algún tema concreto los profesores creen un cuestionario de Kahoot, el cual se suele resolver en clase. El método que usan es el de clasificación en tiempo real, con bonificaciones por racha de aciertos, añadiendo un temporizador para dar inmediatez.

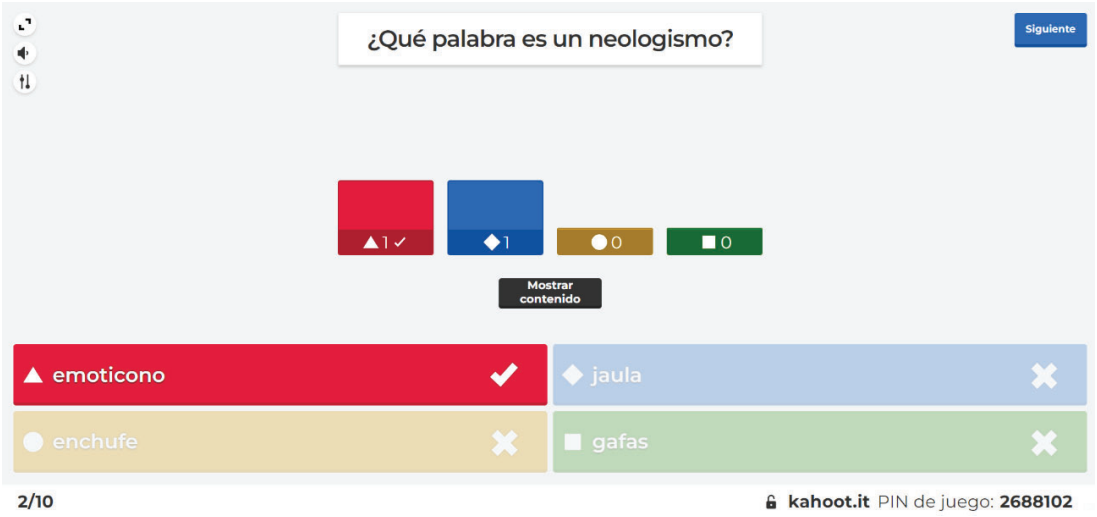


Ilustración 3. Pantalla de juego en Kahoot [15]

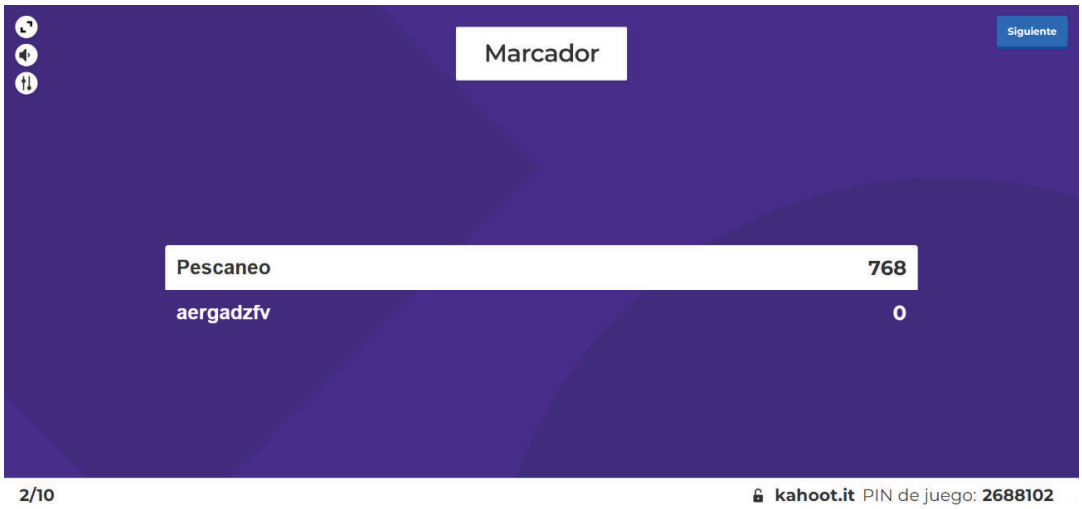


Ilustración 4. Clasificación en tiempo real de Kahoot [15]

- **Moodle**. Este es un caso que se debe ocupar en este documento por la relación que existe entre este proyecto con él, y es que el software de Moodle es el que se utiliza en la Universidad Politécnica de Madrid y otras universidades como aula virtual para la enseñanza y por lo tanto he estado en estrecho contacto con él durante varios años.

  Además, el proyecto en el que está enmarcado este TFG tiene una parte pensada exclusivamente para esta plataforma, por lo que con el fin de mejorar la perspectiva para el desarrollo de la aplicación, el desarrollador ha asistido a una sesión en la que se describió cómo hacen o pueden hacer los profesores para mejorar la inmersión de los alumnos en sus asignaturas mediante la gamificación que ya implementa Moodle. Esta se basa en la pirámide de los elementos de la gamificación (Ilustración 5), en la que se observan tres niveles: las dinámicas, que determinan el comportamiento de los participantes (estudiantes en este caso) y están relacionadas con la motivación; las mecánicas, que son los procedimientos básicos del juego, sus reglas, motor y funcionamiento; y los componentes, es decir, los recursos con los que se cuenta y las herramientas que se utilizan para el diseño de una actividad de gamificación en la práctica. En cuanto a este nivel más bajo, lo más común es hacer uso de la triada PBL (Puntos, insignias y clasificaciones, por sus siglas en inglés).

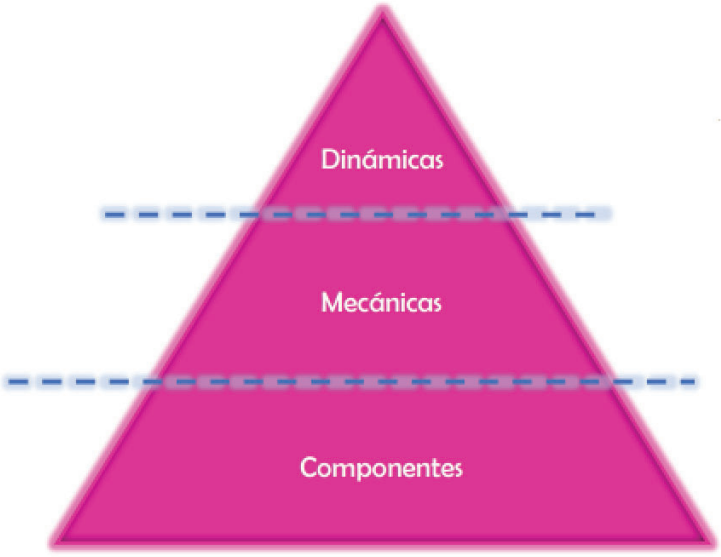


Ilustración 5. Pirámide de elementos de la gamificación

Moodle, creado en 2002, nació con la idea de la pedagogía constructivista, por lo que ha ido implementando a lo largo de los años, con la iniciativa *gaMoodle,* cada vez más técnicas de

gamificación hasta llegar a ser lo que es hoy. Entre estas opciones de gamificación para los profesores, como hemos comentado antes, se encuentran las insignias, puntos y clasificaciones, el progreso del curso, creación de animaciones y mucho más con H5P, plugins como Game, Stash o Level Up, o la organización del contenido en "losas".

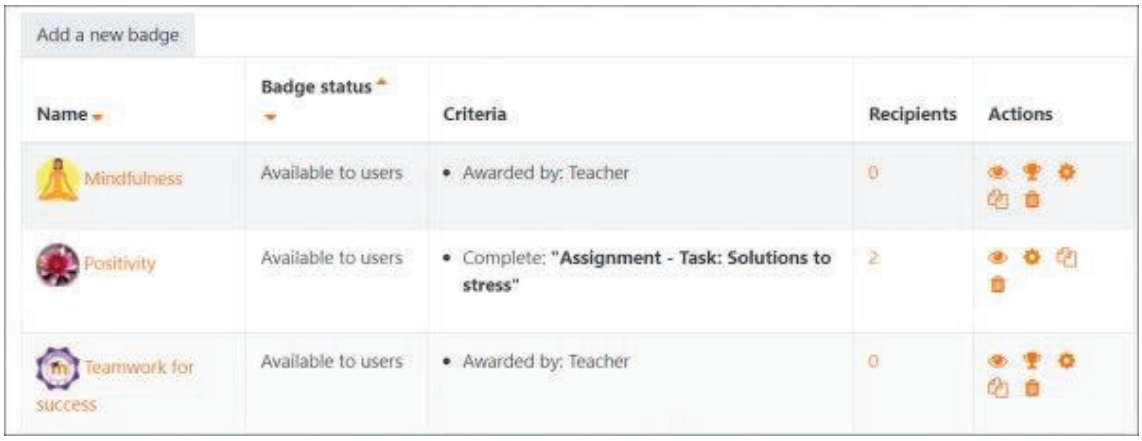


Ilustración 6. Insignias en Moodle

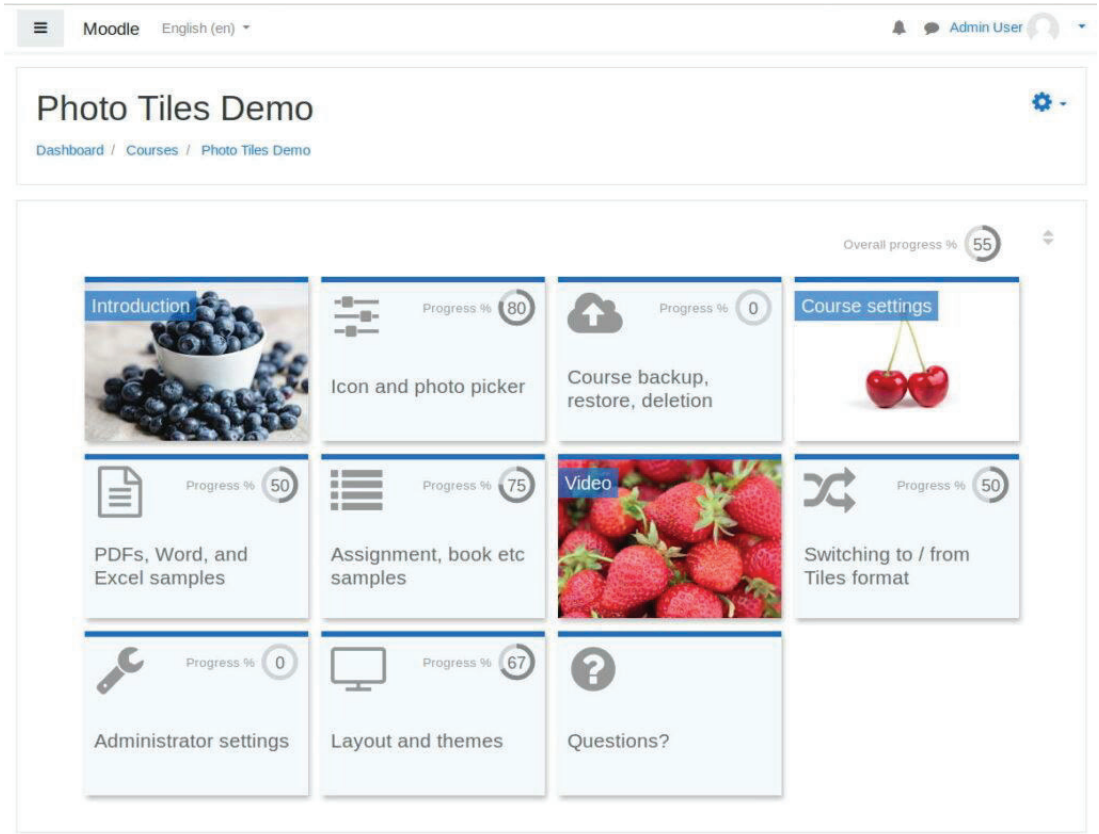


Ilustración 7. Organización de losas en Moodle

- **Nýyrðavefurinn** [16]. Esta es un proyecto considerablemente parecido al que nos ocupa aquí, dado que tiene el objetivo de recabar neologismos a partir de la actividad de sus usuarios, en este caso para el idioma islandés, y en una parte de su propio nombre se puede leer “Nuevas palabras” (Nýyrða).

Ilustración 8. Pantalla principal de Nýyrðavefurinn [16]

Esta aplicación web utiliza un ranking de neologismos y botones de “Me gusta” y “No me gusta”, además de la capacidad de comentar los neologismos. Además, similarmente a como se describirá más tarde en nuestra aplicación, los usuarios de la aplicación pueden crear propuestas de neologismos, que más tarde serán o no aprobadas para aparecer en la página principal. Estos neologismos de la página inicial se pueden reordenar y clasificar según ciertas categorías. También es posible ampliar la información de un neologismo en concreto.

- **Milmots** [17]. Esta es una propuesta parecida a la anterior, ya que cuenta con el mismo objetivo de crear un registro de neologismos con ayuda del usuario, pero en este caso se acerca más a nuestra propuesta al estar integrada con mecanismos de gamificación, lo que la hace un perfecto punto de partida para nuestra idea.

  La diferencia de esta aplicación web con la nuestra es que Milmots se centra más en la localización geográfica en la que se encuentran los neologismos que en el significado de estos, además de que tan solo abarca el catalán, sin incluir el castellano. También se puede observar que el procedimiento para proponer un neologismo es significativamente más sencillo, y que no se incluye ningún sistema de usuarios, como sí que hace nuestra aplicación.

En cambio, tiene muchas similitudes, como el uso de actividades interactivas y de rankings (incluyendo, además de por neologismos, por localización).

Ilustración 9. Página principal de Milmots [17]. Adición de neologismo.

Ilustración 10. Página principal de Milmots [17]. Lugares destacados.

2.2 Tecnologías

En este apartado se van a comentar las principales herramientas tecnológicas que se han empleado en la implementación del proyecto.

2.2.1 Interfaz

Con el objetivo de dar una perspectiva amplia del proyecto y dejar más cosas claras, lo primero que se va a describir en este apartado son las

tecnologías que se han usado para desarrollar la interfaz de la aplicación.

2.2.1.1 Vue

Vue [18] es un marco de trabajo para JavaScript de código abierto para la creación de interfaces de usuario que construye sobre los lenguajes estándar HTML, CSS y JavaScript. Provee de un modelo de programación declarativa y basada en componentes que ayuda a construir eficientemente interfaces de usuario, sean simples o complejas. A continuación, se muestra una imagen como ejemplo de un fichero típico de Vue.

```
<script>
export default {
 data () {
  return {
   count: 0
  }
 }
}
</script>

<template>
 <button @click="count++">Count is: {{ count
}}</button>
</template>

<style scoped>
button {
 font-weight: bold;
}
</style>
```

Ilustración 11. Fichero típico de Vue [19]

Aunque es muy simple, este ejemplo nos muestra la estructura básica que utiliza Vue, la cual se divide en tres fragmentos:

- **Código (Script)**. En este fragmento se definen todas las variables, métodos, funciones… Todo lo que se encuentre entre las etiquetas de <script> debe estar codificado en lenguaje JavaScript. **JavaScript** [19] es un lenguaje de programación orientado a objetos, dinámico, basado en prototipos y débilmente tipado. Nació a manos de Brendan Eich en 1995 con el nombre de Mocha, más tarde LiveScript, y finalmente JavaScript coincidiendo con la incorporación de la compatibilidad con Java en 1995. El uso más

extendido de este lenguaje el de trabajar en páginas HTML e interactuando con el *Document Object Model* de la página. En concreto, JavaScript es capaz de proporcionarnos, por ejemplo, animaciones para los elementos de la página, contenido interactivo como juegos y reproducción multimedia, validación de la entrada de formularios, carga de contenido sin necesidad de recargar la página y transmisión de información de hábitos de actividad sobre los usuarios.

- **Plantilla (template)**. Este apartado está reservado para el código HTML, que a diferencia de un código HTML cualquiera, tiene una serie de modificaciones que le permiten "comunicarse" con la parte de Script. Esto son las anotaciones como *@click*, que describe lo que se debe hacer cuando el usuario clique en el elemento en cuestión, y otros elementos.
  HTML (*Hypertext Markup Language*) [20] es el lenguaje de marcado para la creación de páginas web estándar a cargo del W3C (World Wide Web Consortium). Define una estructura básica y un código para la definición de contenido como imágenes, texto, etc. mediante etiquetas, consiguiendo una significativa adaptabilidad y facilidad de comprensión.

- **Estilo (Style)**. En esta zona es donde encaja el código correspondiente a la configuración del estilo para el código contenido en *template*. Debe utilizarse el lenguaje CSS.
  CSS (*Cascading Style Sheets*) [20] es un lenguaje de diseño gráfico para la definición y creación de la presentación de un documento estructurado que se haya escrito con lenguaje de marcado, como HTML. Es mantenido también por la W3C, y nos brinda muchas ventajas como la separación de contenido y presentación, la consistencia del sitio, el formateo de página y la accesibilidad.

2.2.1.2 Axios

Axios [21] es una de las librerías más populares para hacer llamadas HTTP asíncronas a EndPoints de APIs y llevar a cabo así operaciones CRUD, ya que simplifica mucho el proceso al no ser necesario procesar la respuesta JSON y otras reducciones de código. Por esto se ha decidido que es el método que se va a utilizar para llamar, desde la interfaz, al servidor del Back-End.

2.2.2 Back-*E*nd

A continuación, se describen las tecnologías a usar durante el desarrollo del código de fondo de la aplicación web.

2.2.2.1 Flask

Flask [22] es un micro marco de trabajo para web escrito en Python que agiliza el desarrollo de aplicaciones web brindando componentes de Back-End esenciales. Las principales ventajas de este *framework* es que es simple y ligero, destacando sobre los demás en estos aspectos, algo que fue vital en la elección ya que nuestra aplicación no va a ser pesada. Además, está diseñado para ser altamente extensible, dando oportunidades para el futuro. Otra ventaja es que Flask no te exige que utilices cierta base de datos o ciertas librerías de autenticación, por ejemplo, sino que lo deja a la elección del programador. Entre las compañías que hacen uso de esta tecnología se puede encontrar a Netflix, LinkedIn o Uber. Otro punto a favor que provocó la decantación por este software es que es muy popular, por lo que cualquier error que pudiera surgir en el desarrollo estaría solucionado en páginas como *stackoverflow.com* múltiples veces.

```python
from flask import Flask
app = Flask(__name__)

@app.route("/")
def holamundo():
    return "¡Hola Mundo!"

app.run(port=5000)
```

Ilustración 12. Código básico de Flask [22]

2.2.2.1.1 Python

Como hemos comentado antes, el micro *framework* Flask está construido sobre Python. Python [23] es el lenguaje del que más se ha hablado en los últimos años, y esto es por si gran legibilidad, mantenibilidad, compatibilidad, robustez, simplicidad y popularidad. Es un lenguaje de alto nivel interpretado multiparadigma que fue creado con la idea de la legibilidad, por lo que es mucho más "limpio" que otros lenguajes parecidos. Otra ventaja que tiene es que en el grado de Ingeniería Informática se utiliza en diversas asignaturas, por lo que el grado de conocimiento con el que se parte es medio. En este proyecto se ha usado también el intérprete de comandos de Python3, con el objetivo de hacer pruebas rápidas y manejar los ficheros *.py* ya creados.

2.2.2.1.2 Módulos de Flask

Ya se ha descrito el marco de trabajo Flask [22] y, como se ha explicado, deja elegir al programador las librerías adicionales de

bases de datos, autenticación, validación de formularios, etc. Aun así, también tienen sus propias implementaciones de estas librerías, la mayoría de las cuales hemos seleccionado para este proyecto. Estas son las siguientes:

- *Flask-login*. Este módulo brinda todas las herramientas necesarias para el desarrollo de un sistema de autenticación web, contando con el inicio y finalización de sesión, el mantenimiento de la sesión durante un tiempo a configurar y mediante cookies, etc. Tiene un funcionamiento simple y eso ha sido de gran ayuda.

- *Flask-session*. Con esta librería conseguimos un control total sobre las sesiones de usuario, y junto al módulo anterior se puede completar el servicio de autenticación de usuario de una manera rápida y sencilla. Aun así, en el proceso de implementación surgieron algunos errores costosos de arreglar debido a la insuficiente documentación de esta librería.

- *Flask-cors*. Esta extensión nos permite manejar el intercambio de recursos de origen cruzado, algo esencial cuando hay distintos servidores para el Front-End y el Back-End, como es el caso. Así, evitamos tener que configurar cabeceras permitidas, métodos, etc. El uso es muy sencillo y evita errores indeseados en las peticiones desde la interfaz.

- *Flask-mail*. Esta es una simple extensión que permite el envío de correos electrónicos con distintos servidores como el de Gmail. En el caso de este proyecto ha sido utilizado para la recuperación de la contraseña del usuario.

- *Flask-wtf*. Este módulo integra los WTForms, una librería de validación y renderización de formularios flexible. Con esto podemos trabajar con formularios rellenados por el usuario muy fácilmente. En este proyecto se ha implementado para la recuperación de la contraseña.

2.2.2.2 SQLite

SQLite [24] es una librería escrita en C que implementa un motor de bases de datos SQL pequeño, autocontenido, de alta fiabilidad y completamente equipado. Es el motor más usado en el mundo, con 10^{12} bases de datos activas. Destaca por su simplicidad de uso, ya que simplifica mucho el lenguaje SQL, además de que es de código abierto.

Aunque sería posible, en este proyecto se ha decidido no ejecutar directamente directrices SQLite, ya que sería más engorroso. En cambio, se utiliza **SQLAlchemy** [25], un set de herramientas para Python y SQL que también sirve para como mapeador relacional de objetos (ORM), dando mucha facilidad para el trabajo con objetos en el programa y la base de datos.

Pero este proyecto no se queda ahí, sino que añade una capa más de abstracción utilizando, en vez de SQLAlchemy, el módulo Flask-SQLAlchemy, el cual facilita la integración con los demás elementos de Flask.

2.2.2.3 Werkzeug Security

Werkzeug [26] es una librería para aplicaciones web del tipo de Interfaz de puerta de enlace para servidores web (WSGI). En este proyecto tan sólo se va a utilizar el módulo Security, el cual nos da utilidades para cifrar y descifrar las contraseñas de los usuarios, por lo que nunca almacenaremos las contraseñas en texto plano en la base de datos.

2.2.2.4 Itsdangerous

Itsdangerous [27] nos permite enviar datos a entornos no confiables y traerlos de vuelta sabiendo si han sido modificados. Para ello se utiliza una clave maestra que debe ser larga y complicada. En nuestro código se ha utilizado para la generación de tokens para la recuperación de contraseñas de usuario con una duración específica, inservibles después de ese tiempo.

2.2.3 Herramientas

A continuación, se describen las herramientas o programas que se han utilizado para el desarrollo del código del proyecto.

2.2.3.1 DB Browser for SQLite

DB Browser for SQLite [28] es un programa muy sencillo para monitorear el estado de una base de datos SQLite, y en nuestro caso ha servido para ver el contenido de nuestra base de datos, ya que al utilizar y manejar la base de datos desde SQLAlchemy no tenemos acceso visual, y a veces ayuda en ciertas tareas.

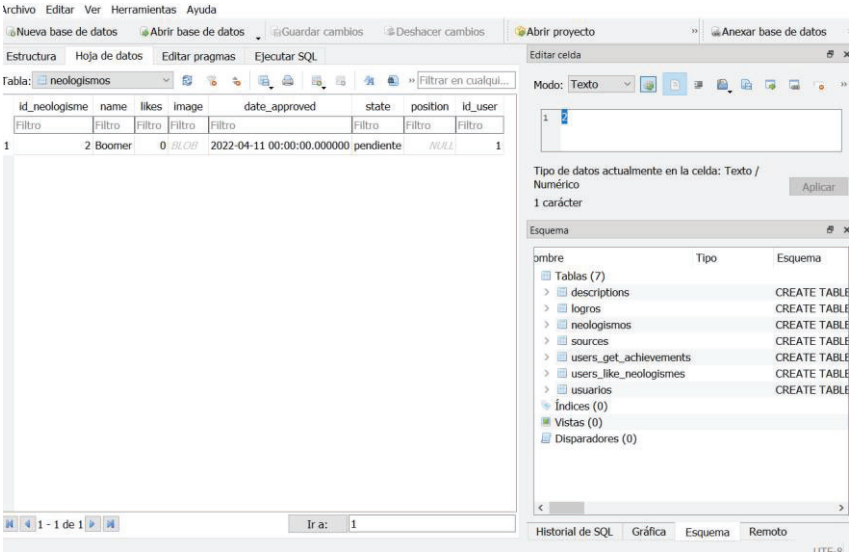


Ilustración 13. Panel principal de DB Browser for SQLite [28]

2.2.3.2 *Visual Studio Code*

VS Code [29] es un editor de código optimizado para la construcción y *debugging* de aplicaciones web y cloud modernas. Tiene numerosas ventajas, pero las que más destacan son que soporta cientos de lenguajes y, gracias a la posibilidad de instalar extensiones, es muy fácil personalizarlo con subrayado de sintaxis, coincidencia de paréntesis, fragmentos de código… Además, hace especial énfasis en el desarrollo web e integra el sistema de control de versiones Git, lo cual es perfecto para este proyecto.

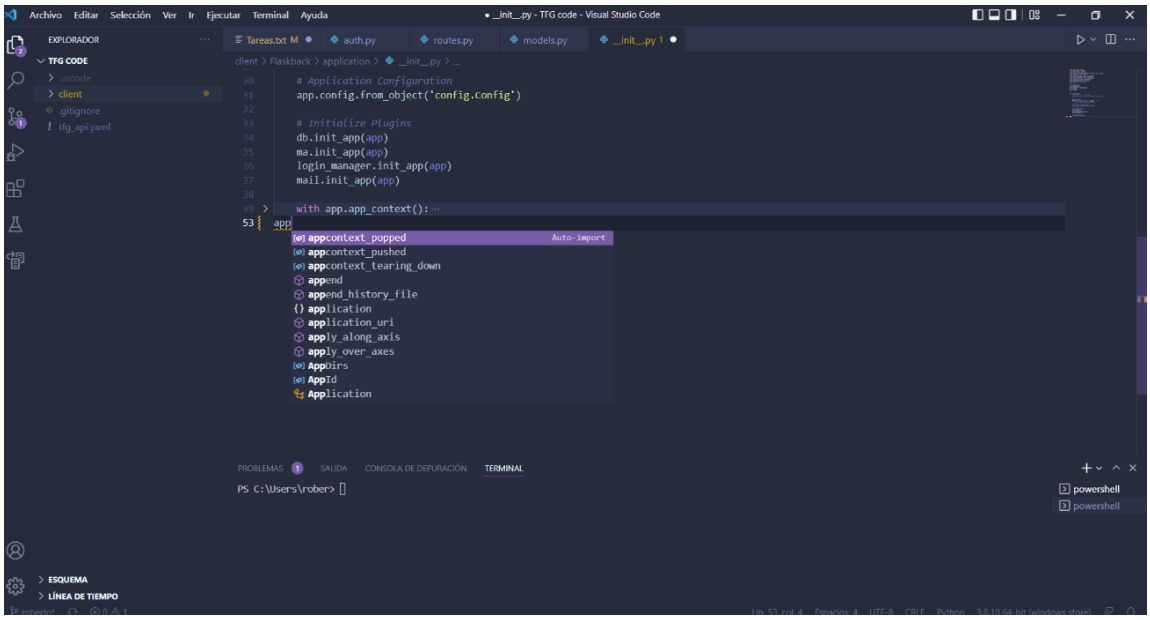


Ilustración 14. Ejemplo de pantalla de VS Code [29]

2.2.3.3 Git

Git [30] es un sistema de control de versiones de código abierto y gratuito diseñado por Linus Torvalds para pequeños y grandes proyectos y centrado en la eficiencia, confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones. Un ejemplo de uso de este sistema es Linux, controlado con Git. Se caracteriza por:

- Una gestión distribuida, por lo que cada programador tiene una copia del proyecto en su repositorio local, existiendo uno remoto del cual los cambios se importan como ramas adicionales y se propagan y fusionan entre estos repositorios.

- Desarrollo no lineal para una gestión rápida de ramas y mezclado de diferentes versiones.

- Gestión eficiente de proyectos grandes gracias a la rapidez de gestión de diferencias entre archivos.

- Compatibilidad con VS Code [29] y Github [31].

2.2.3.3.1 Github

Como hemos descrito en el anterior punto, en el sistema de control de versiones Git hay dos elementos principales, el repositorio local y el remoto. El local se encuentra en nuestro ordenador, mientras que el remoto tenemos que alojarlo en algún sitio. Aquí es donde entra GitHub [31], que nos sirve como servidor para almacenar nuestros repositorios remotos. La gran ventaja que tiene es que es compatible con muchas herramientas, por lo que su manejo es sencillo. El repositorio concreto en el que se encuentra alojado nuestro proyecto [32] es público de momento.

2.2.3.3.2 SourceTree

Algo que no se ha comentado todavía sobre el sistema Git es que el manejo más común es con un intérprete de comandos como el Command Prompt de Microsoft Windows, con comandos sencillos como *fetch*, *pull*, *checkout* o *push* pero, para más comodidad, se ha decidido utilizar SourceTree [33], una Interfaz Gráfica de Usuario para Git y desarrollada por Atlassian que ofrece una representación visual ideal para trabajar con los repositorios. Este software es compatible con GitHub, por lo que hace incluso más sencilla la tarea para la que está diseñado.

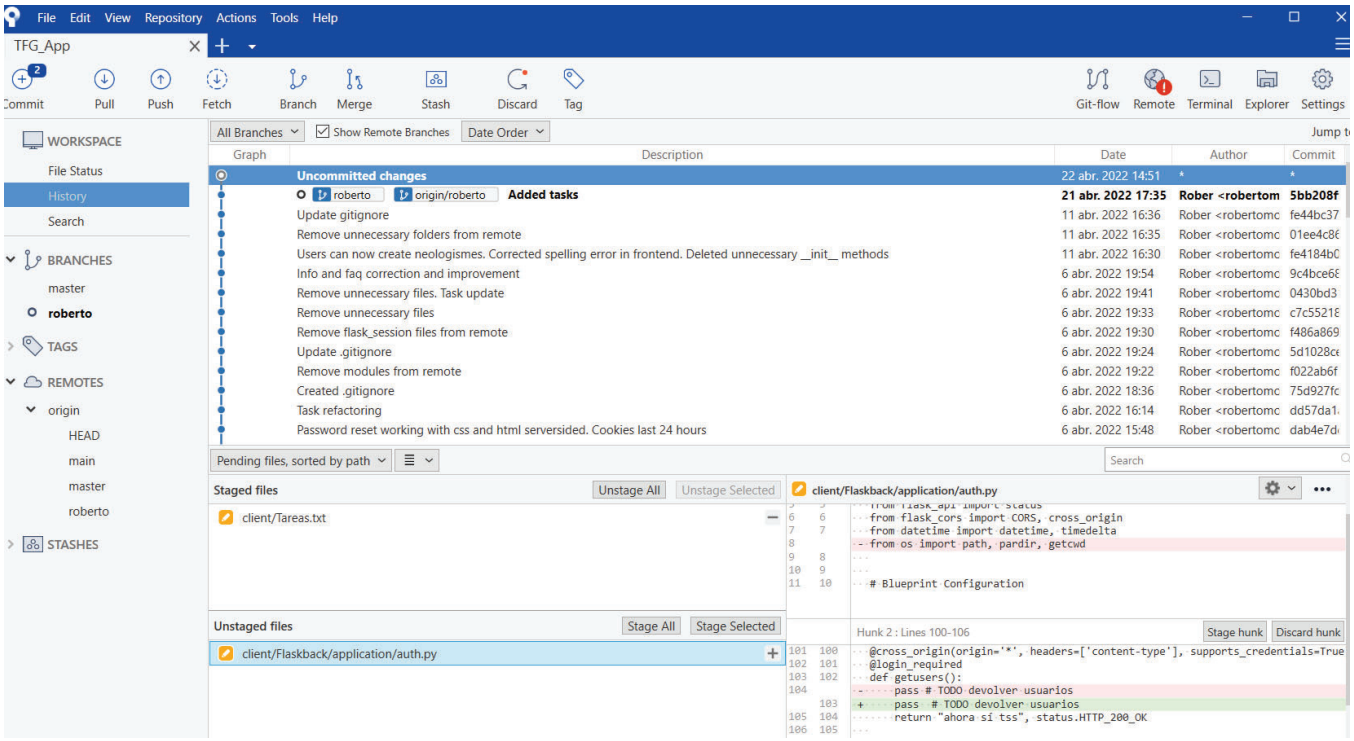


Ilustración 15. Panel de control de SourceTree [33]

3. Desarrollo

3.1 Diseño del Back-End

Antes de empezar a codificar nada, se debe diseñar hasta un nivel muy específico ciertos aspectos de la parte trasera de la aplicación. Para ellos se va a hacer uso de diagramas como el de entidad-relación o el de clases, para así comprender también visualmente los componentes con los que contará el Back-End.

3.1.1 Base de datos

A continuación, se describen los dos principales pasos para el diseño de la base de datos de la aplicación, previamente a la implementación de esta. La creación de los diagramas se ha llevado a cabo mediante las aplicaciones web diseñadas para tal efecto LucidChart [34] y draw.io [35].

3.1.1.1 Diagrama entidad-relación

Lo primero que se debe diseñar son las entidades con las que se va a trabajar, y para ello es muy útil diseñar y visualizar la base de datos. Para saber qué objetos se tienen que crear, hay que fijarse en los requisitos funcionales de la aplicación, enumerados por Javier Barragán en su TFG, y que son los siguientes:

- El administrador puede dar de alta a otros usuarios como administradores. Estos administradores tendrán todas las mismas características que el resto de los administradores.

- El administrador puede eliminar usuarios independientemente de su rol o función.

- Un Administrador podrá eliminar un Neologismo o información relacionada con él mismo.

- Un usuario con rol de Lingüista o de Administrador podrá aceptar o rechazar una propuesta de Neologismo. Las propuestas rechazadas no serán eliminadas del sistema, serán marcadas como rechazadas. El usuario creado de la propuesta recibirá una notificación con esta indicación y con la razón del rechazo. También se entienden como propuestas de Neologismo las propuestas de modificación.

- El usuario podrá descargar un documento CSV con las siguientes opciones.

  - Todos los Neologismos de la aplicación con toda la información, es decir, Neologismos validados y sin validar.
  - Propuestas de Neologismos sin validar.
  - Propuestas de Neologismos Validadas.

- Se podrá de alta a un usuario en la aplicación facilitando la siguiente información:
  - Alias.
  - Nombre y Apellidos.
  - Fecha de Nacimiento.
  - Lengua materna.
  - Escuela UPM al que pertenece el usuario.
  - Correo Electrónico.
  - Genero.
  - Ocupación.
  - Contraseña.
  - Imagen de Usuario (Opcional).

- El usuario podrá restablecer su contraseña mediante una confirmación por correo electrónico.

- El usuario podrá iniciar sesión en la aplicación siempre y cuando el nombre y la contraseña sean correctos. Para iniciar sesión se podrá utilizar el correo electrónico facilitado por el usuario o el nombre de usuario de este.

- El usuario podrá cerrar sesión.

- El usuario podrá proponer un nuevo neologismo, proporcionando la siguiente información:
  - Nombre del Neologismo.
  - Descripción (Mínimo 1).
  - Fuente de la que se ha sacado el neologismo (Mínimo 1).
  - Imagen de Caso de uso (opcional).

- El usuario podrá marcar un neologismo ya aceptado como Me gusta. Dicho neologismo aparece en la sección de palabras favoritas del usuario.

- El usuario podrá acceder a la información de un Neologismo que haya sido aceptado por un Lingüista o por un Administrador.

- El usuario podrá acceder a la información de su perfil.

- El usuario podrá acceder a las clasificaciones de jugadores y de popularidad de Neologismos.

- Se podrá acceder a la aplicación.

- Un usuario podrá notificar errores sobre la aplicación o sobre información errónea de un neologismo.

- El sistema calculará la puntuación total del usuario.

- El usuario podrá ganar logros.

- Un usuario podrá consultar la información de los logros conseguidos.

- El usuario podrá acceder a las actividades y realizarlas siempre que quiera.

- Se podrá proponer modificar la información de un Neologismo, creándose una propuesta de Modificación, en caso de que el Neologismo hubiera estado validado, o una modificación en una propuesta de Neologismo, en caso de que no hubiera estado validado. Un administrador o un Lingüista deberá validar el cambio antes de que la información sea pública.

Lo primero que se viene a la mente al leer estos requisitos es la entidad del **usuario**, que es de la que parten todas las acciones y entorno a la cual se diseña la aplicación. Para crear este usuario se requerirán una serie de atributos básicos que deberá proporcionar la persona que cree el usuario. Algunas de estas características son necesarias para el funcionamiento de la aplicación y por lo tanto es obligatorio que el usuario las proporcione. Estas son:

- El nombre de usuario o nickname, una cadena que servirá para identificar a cada usuario unívocamente.
- El nombre real y apellidos, formando dos cadenas.
- El correo electrónico, que se comprobará que tenga forma de email y servirá tanto para iniciar sesión como para recuperar la contraseña.
- La contraseña, una cadena que tendrá unas características determinadas para aumentar su seguridad (ver apartado de seguridad).

Además, se añadirán otras características que, aunque no son necesarias, pueden ser muy útiles para ciertas funciones. Estas son:

- La fecha de nacimiento, la cual se introducirá mediante un calendario para evitar errores de forma.
- El género, el cual se podrá elegir de entre cuatro opciones:

    - Masculino
    - Femenino
    - No binario
    - Prefiero no decirlo
- La escuela de la UPM a la que pertenecen, una cadena que podrán elegir de entre todas las escuelas con las que cuenta la Universidad Politécnica de Madrid:
    - ETSAM
    - ETSE
    - ETSIINF
    - ETSII
    - ETSIAE
    - INEF
    - ETSISI
    - ETSIST
    - ETSME
- La lengua materna, una cadena que podrán elegir de entre las siguientes:
    - Español
    - Portugués
    - Inglés
    - Chino
    - Alemán
    - Francés
- Una imagen de perfil opcional, que se podrá visualizar en las clasificaciones y en la página principal.

Por otro lado, hay ciertos atributos que no los proporciona el usuario, sino que tiene que gestionarlos la aplicación. Estos atributos gestionados por la aplicación son:

- Los privilegios de usuario, que podrán ser usuario normal y corriente, lingüista, administrador o usuario eliminado (de esto se hablará más adelante).
- Puntuación, un valor numérico de tipo entero que llevará la cuenta de los puntos que ha conseguido el usuario mediante logros.
- Identificador de usuario, un valor entero que se creará incremental y automáticamente.
- Además, inicialmente se pensó en un atributo de posición en las clasificaciones, aunque más tarde se hizo evidente que era un valor más fácil de calcular cada vez, debido a su gran variabilidad.

Ahora hay que pensar en otras entidades para la base de datos. La siguiente, que también es muy evidente, viene del mero objetivo de la aplicación, el registro de neologismos. Por ello, la próxima entidad será el **neologismo**.

Del neologismo, igual que con el usuario, hay información que se requiere del creador del neologismo. En este caso el principal atributo es el nombre del neologismo, es decir, el neologismo en sí. Además, hay varios atributos que no proporciona el usuario pero que son necesarios para la gestión interna de la aplicación. Estos son los siguientes:

- El número de "me gusta", es la cifra entera que describe el número de veces que un usuario ha dado "*like*" al neologismo.
- Identificador de neologismo, un número entero que sirve para identificar unívocamente a un neologismo.
- Fecha de admisión del neologismo, un atributo de tipo fecha y hora que servirá para la implementación de la clasificación de neologismos de la semana.
- Estado del neologismo, una cadena cuyos posibles valores son:
    - Pendiente. Cuando el neologismo ha sido propuesto o modificado y está a la espera de la revisión de un lingüista o administrador.
    - Aceptado. Este estado lo tienen los neologismos que han sido aceptados y que se pueden encontrar en las clasificaciones.
    - Rechazado. Los neologismos con este estado no cumplen con las características necesarias para ser aceptados, por lo que deberán ser modificados para que se vuelvan a considerar.

Los neologismos deben contar con otros dos atributos necesarios que deben ser proporcionados por los usuarios que los creen, pero tienen una característica que los hace especiales y que obliga a tratarlos de otra forma. Estos son las descripciones o contextos, y las fuentes de los neologismos. En principio se planteó utilizarlos como un atributo más, como lo puede ser el nombre, pero al poder haber más de uno de cada tipo esto es imposible. La solución que se utiliza en estos casos es la creación de una nueva entidad que tenga una única relación con la entidad de la cual es atributo. Esto es lo que se ha hecho para las descripciones y fuentes, teniendo a su vez cada una de ellas dos atributos. Empezando con los de los contextos:

- Identificador de descripción, un número entero único para identificar a cada contexto.
- Texto, que es la cadena de caracteres que contiene la descripción en sí.

Ahora, los atributos de las fuentes de los neologismos:

- Identificador de fuente, muy parecido al de las descripciones.
- Enlace, una cadena de caracteres que contiene el "*link*" a la página web de la cual se ha obtenido el neologismo.

Por último, hay una entidad más relacionada con los usuarios, y es la de **logro**. De vez en cuando, cuando un usuario realiza una acción en concreto o consigue acumular un cierto número de acciones, se le recompensa con un

logro, como se ha comentado en la sección de gamificación. Se podría pensar en los logros como un atributo del usuario, pero en cambio, si se profundiza un poco más sale a la luz que puede haber muchos usuarios con el mismo logro, mientras que un mismo usuario contará con más de un logro, haciendo imposible así la posibilidad de contarlo como atributo. Por lo tanto, se crea una nueva entidad que cuenta con unos atributos predefinidos por el administrador de la aplicación:

- Identificador de logro, un número entero cuyo objetivo es identificar unívocamente cada logro.
- Descripción del logro, una cadena de caracteres que describe muy brevemente y a modo de nombre las acciones necesarias para conseguir el logro.
- Acción, una cadena de caracteres que describe con mucho más detalle qué se debe hacer para obtener el logro en cuestión.
- Dificultad, un valor numérico entero que servirá para saber cuántos puntos dar al usuario que lo ha conseguido a la hora de obtenerlo. Concretamente, el valor de dificultad se multiplica por 10 para obtener el número de puntos a dar al usuario (ver por qué en sección de gamificación).
- Nombre, una cadena de caracteres que describe, mediante una forma general, varios aspectos del logro. Esta forma es la siguiente:
  Tipo_logro(de neologismo o diario).id.dificultad
- Medalla, que es una imagen que sirve de refuerzo visual para la obtención del logro.

Hasta aquí llegó el diseño inicial de la base de datos, pero en la fase de implementación se vio la necesidad de introducir otra entidad más, ya que era la única manera de gestionarlo. Esta entidad es la notificación de error por parte de los usuarios acerca del sistema, que no se llegaron a implementar completamente en el Front-End en el anterior TFG, pero que sí se ha hecho en este y requería de esta nueva entidad. Las entidades con las que cuenta son:

- Identificador de notificación de error, un valor numérico entero para identificar las notificaciones.
- Un título del error, cadena de caracteres para resumir el fallo encontrado.
- Una descripción más amplia del error, con un límite de caracteres de 500 unidades.

Ahora se debe profundizar en las relaciones entre estas entidades, ya que esto es algo de gran importancia en la creación de la base de datos.

Se va a partir del usuario, que se podría considerar como el centro de la arquitectura de entidades. Entre el usuario y el logro hay una relación de tipo de muchos a muchos ya que, como se ha explicado antes, un usuario puede

obtener varios logros y el mismo logro puede ser obtenido por distintos usuarios.

Hablando de usuarios, pero en relación con los neologismos, se hace evidente que existen dos relaciones, ya que un usuario puede crear neologismos, pero también los puede marcar como favoritos. En cuanto a la primera, será una relación de uno (y solo uno) a muchos (de cero a muchos), ya que un usuario puede crear de cero a muchos neologismos, pero un neologismo solo puede haber sido propuesto por un usuario, ni más ni menos. La segunda relación entre estas dos entidades deberá ser de muchos a muchos, ya que un usuario puede dar “me gusta” a varios neologismos y un neologismo puede ser marcado como favorito por múltiples usuarios.

La última relación del usuario será con las notificaciones de error, y será de tipo uno a muchos porque cada notificación es creada solo por un usuario, pero un usuario puede crear más de una notificación.

Ya para terminar con las relaciones, se va a describir las que hay entre los neologismos y sus “atributos”, los contextos y las fuentes, que deben ser iguales para ambos. Más concretamente, deberán ser del tipo uno a muchos, como la relación de creación de neologismos por los usuarios, ya que una descripción o fuente solo puede pertenecer a un solo neologismo, pero estos pueden tener numerosos contextos o fuentes.

Descrito todo el esquema de la base de datos con entidades y relaciones, se procede a visualizarlo de forma gráfica con el diagrama entidad-relación de la ilustración 16, en la que se observan todos los elementos anteriormente descritos, dispuestos de forma que sigan el estándar de los diagramas entidad-relación.

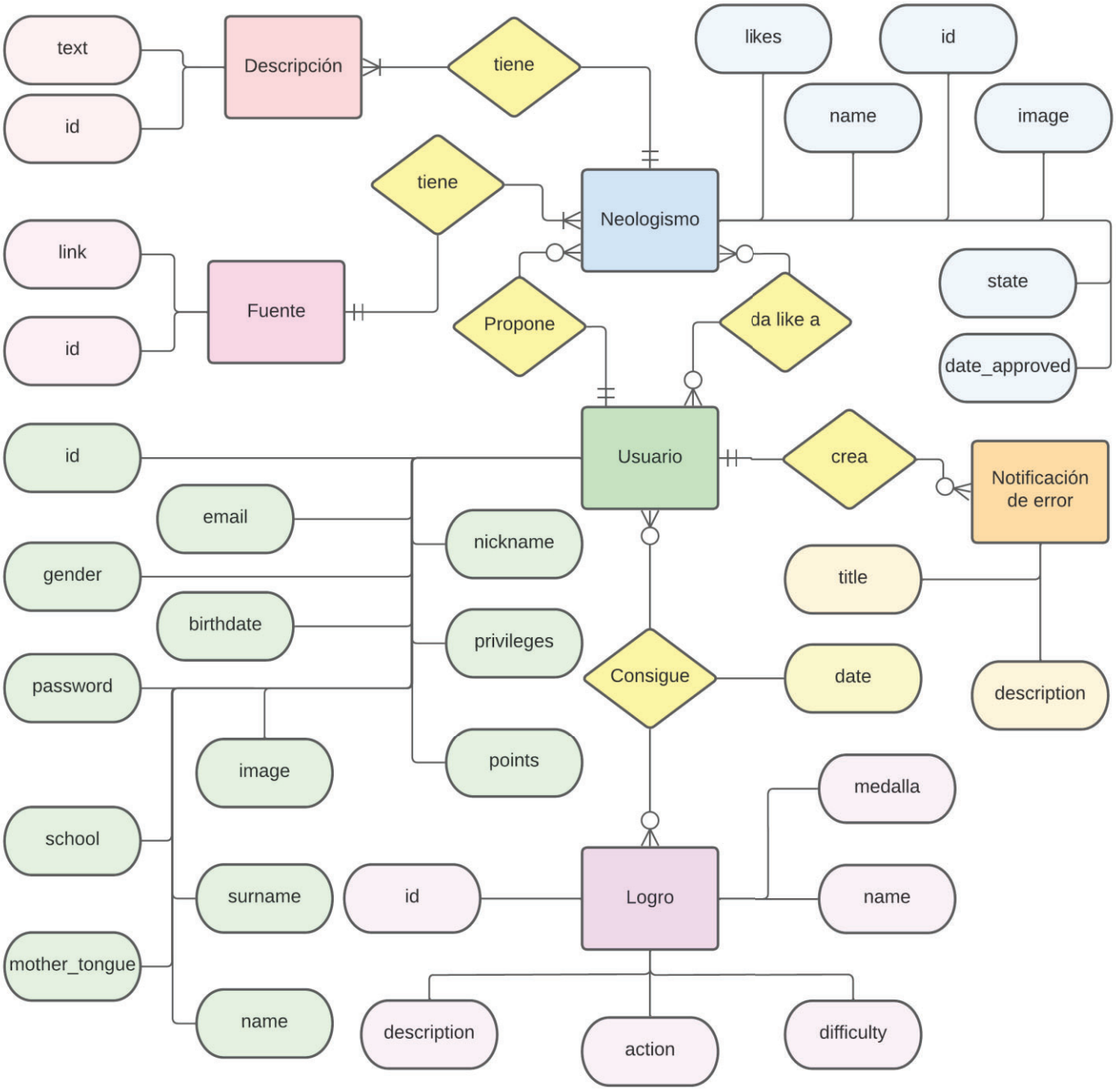


Ilustración 16. Diagrama de entidad-relación

Con esto, los rectángulos son las entidades, los rombos son las relaciones y los rectángulos más alargados y con esquinas redondeadas son los atributos de las entidades a las cuales están unidas con líneas. Los tipos de relación del diagrama se explican mediante una simbología básica indicada en la ilustración 17.

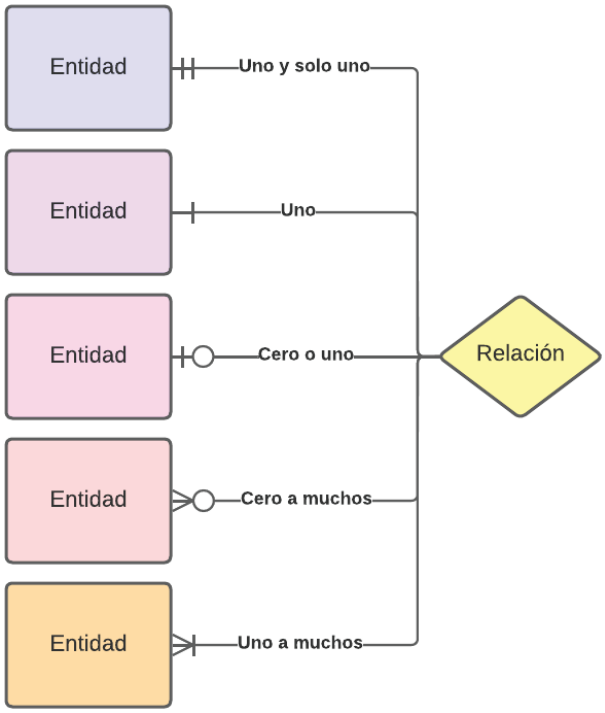


Ilustración 17. Leyenda de tipos de relación

3.1.1.2 Diagrama de tablas

El anterior es el diagrama entidad-relación o diagrama E/R, pero hasta llegar a la creación de la base de datos hay un paso intermedio, y este es el de la creación de las tablas de la base de datos, algo que se facilita con un diagrama de tablas. El diagrama entidad-relación está muy relacionado con el de tablas, ya que cada elemento de uno se traslada al otro de una manera concreta.

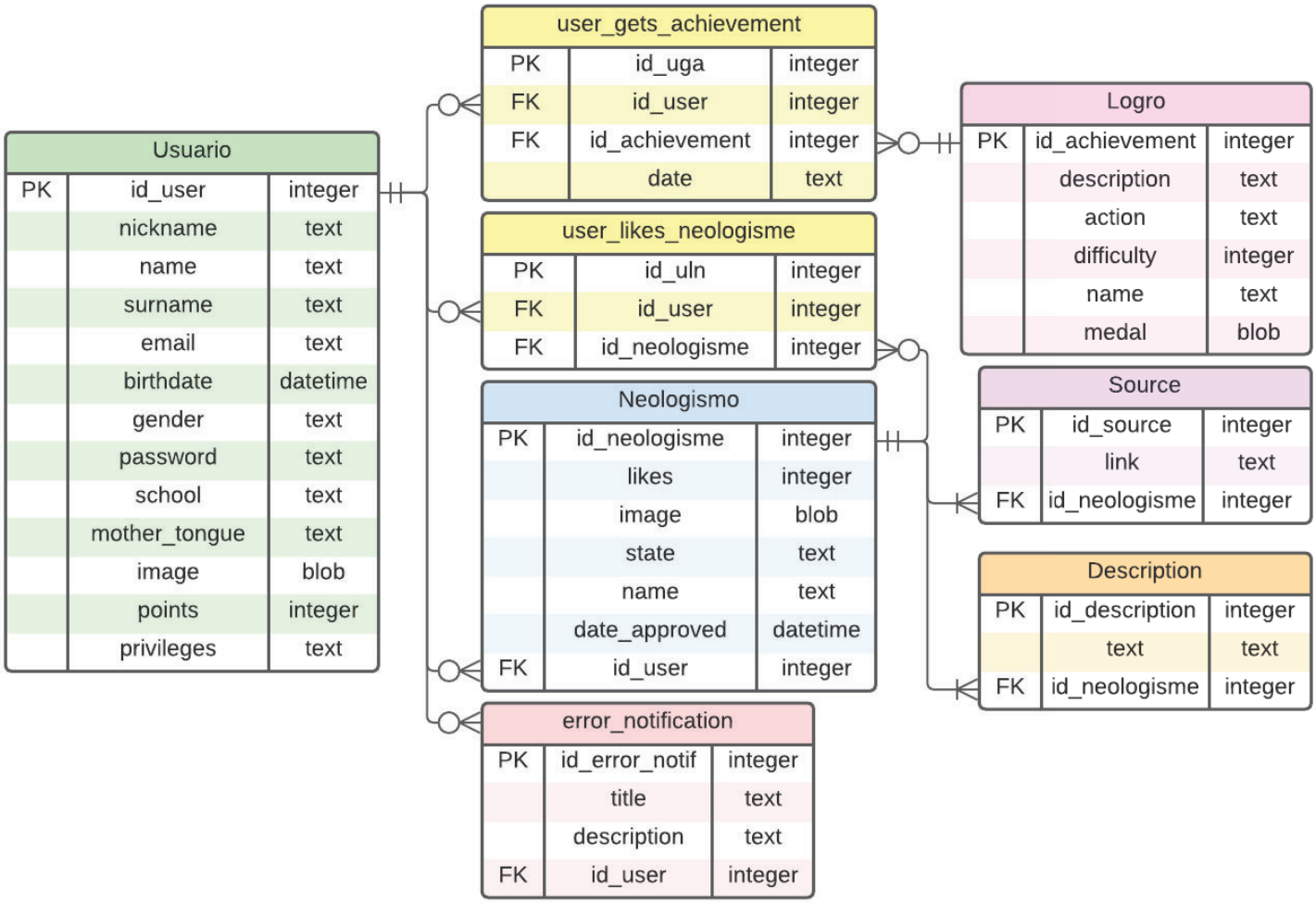
Cada entidad del E/R se transforma en una tabla del de tablas, cogiendo los atributos del primero y pasándolos como columnas del segundo, añadiendo siempre una columna de identificador, llamada clave primaria, que debe ser única, con valor numérico entero e incremental.

Las relaciones son algo más complicadas de trasladar, dado que depende del tipo que sean se hace de una manera u otra. Cuando son de muchos a muchos, se crea una nueva tabla que tenga como columnas una de identificación y luego otras dos, una por cada clave primaria de las otras. Cuando una clave primaria de una tabla se encuentra en otra para representar una relación su nombre cambia a clave foránea, por lo que la tabla de relación de muchos a muchos estará formada por una clave primaria y dos foráneas. Las relaciones de uno a muchos, a diferencia de las anteriores, no se transforman en una tabla, sino que se añade una columna en la entidad

con la parte de “uno” en forma de clave foránea que referencia a la clave primaria de la entidad con la parte de la relación de “muchos”.

Además de lo ya explicado, cada columna ha de tener un tipo de datos definido, entre los cuales se encuentran integer (número entero), texto, datetime (datos con formato de fecha y hora) y blob (datos binarios como, por ejemplo, imágenes).

Descrito el proceso de paso a tablas de un diagrama E/R, veamos ahora como queda el de nuestra base de datos.



Como dato curioso acerca de estos diagramas, las aplicaciones más avanzadas con las que se suelen crear permiten obtener el código SQL necesario para la creación de las tablas representadas directamente. En este caso esto no es útil, ya que no se van a crear de la forma tradicional mediante sentencias de SQL porque SQLAlchemy funciona de forma diferente, pero puede ser muy útil para alguien que lo haga de la forma “tradicional”.

3.1.2 API

Una vez diseñada la base de datos, hay que pensar en cómo se dará el servicio de proveedor de datos de la base de datos a las peticiones que lleguen al servidor de Back-End. Lo ideal siempre es hacer una API REST, pero vamos a ver el porqué.

REST (Representational State Transfer) es una interfaz entre sistemas que utiliza el protocolo HTTP para recuperar datos o crear operaciones sobre datos en todos los formatos posibles, como XML y JSON. Se trata de una alternativa cada vez más frecuente a otros protocolos estándar de intercambio de datos, como el SOAP.

Las características principales de REST y las cuales lo convierten en el estándar más utilizado en servicios web son las siguientes:

- Utiliza un protocolo cliente/servidor sin estado, es decir, cada petición HTTP contiene toda la información necesaria para llevar a cabo las acciones pertinentes, por lo que no es necesario que ninguna de las dos partes involucradas recuerde nada anterior mediante memoria o estado.
- Todo lo que está disponible es un recurso, y tiene asociada una o varias representaciones. Algunos ejemplos de representaciones estándar son text/html, application/pdf, image/gif o text/csv. Cada uno de estos recursos deben ser identificables unívocamente mediante, por ejemplo, un identificador numérico único y debe estar disponible en una URI concreta.
- Se utiliza solo un pequeño conjunto de métodos u operaciones bien definidas aplicables a todos los recursos. Las operaciones más importantes y las que usaremos en este proyecto son las siguientes:
    - GET. Operación de lectura de un recurso que recupera información específica del servidor. Es una operación idempotente (siempre arroja el mismo resultado) y segura (no cambia nada en el servidor).
    - PUT. Esta operación permite cargar la representación adjunta con la petición en el recurso asociado a la dirección de la petición. Se usa como método de actualización o modificación y es idempotente, ya que enviando la misma representación siempre dará el mismo resultado.
    - DELETE. Con esta operación eliminaremos el recurso solicitado. Es idempotente como las dos anteriores.
    - POST. Con este método creamos un nuevo recurso, siendo no idempotente y no seguro.
    - OPTIONS. Con esta operación se pide al servidor información acerca de sus capacidades, como por ejemplo para comprobar si un método se puede realizar en cierta URI.

Para diseñar la API de este proyecto, habrá que fijarse en los diseños de la base de datos, ya que el resultado no será muy distinto. Además, como base para todas las URIs, se propone la base *"http://www.pescaneo.es/api*", que no es la que se ha estado utilizando en el desarrollo debido a que la aplicación no está alojada en ningún servidor externo a la máquina de desarrollo, pero que si se llega a desplegar la aplicación puede servir.

A continuación, se van a enumerar todos los EndPoints de la API que vamos a construir, profundizando en cada uno de ellos, detallando la funcionalidad y códigos HTML que devolverán mediante las tablas estándar para la definición de APIs.

En cuanto a los usuarios, necesitaremos un EndPoint para actuar sobre el conjunto y otra para cada uno en concreto, que serán ***/users*** y ***/users/{user_id}***, respectivamente. También se añadirá otro EndPoint específico para los usuarios que conforman la clasificación de los cinco mejores usuarios, y será ***/users/ranking-five***. Además, es necesario otro EndPoint para los neologismos que ha propuesto el usuario concreto y para los neologismos que le han gustado, para los que usaremos ***/users/{user_id}/neologismes*** y ***/users/{user_id}/favs***, respectivamente.

Obtener lista de usuarios del sistema		
URI	http://www.pescaneo.es/api/users	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	200 - OK	Se devuelve un objeto de tipo text/JSON que contiene una lista de todos los usuarios que están dados de alta en el sistema. Según la sesión de la que provenga la petición, estos usuarios devueltos serán los que tengan unos privilegios u otros. De cada usuario se proporcionará el nombre de usuario ('nickname'), posición

		en el ranking general ('position'), los puntos que tiene ('points'), el id de usuario ('id') y los privilegios que tiene ('privileges').
	204 – No content	-

Añadir un usuario al sistema		
URI	http://www.pescaneo.es/api/users	
Método	POST	
Cuerpo de consulta	Privileges	Privilegios del usuario
	nickname	Nombre de usuario del usuario
	name	Nombre real del usuario
	surname	Apellidos del usuario
	email	Correo electrónico del usuario
	date	Fecha de nacimiento del usuario
	gender	Género del usuario
	password	Contraseña del usuario
	school	Escuela a la que pertenece el usuario
	mother_tongue	Lengua materna del usuario
	points	Puntos del usuario
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	201 - Created	-
	400 – Bad request	-

Obtener lista de los 5 mejores usuarios	
URI	http://www.pescaneo.es/api/users/ranking-five
Método	GET

Cuerpo de consulta	-	-
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	200 – OK	Se devuelve un objeto de tipo text/JSON que contiene una lista de los cinco usuarios con más puntos del sistema. De cada usuario se proporcionará el nombre de usuario ('nickname'), posición en el ranking general ('position'), los puntos que tiene ('points').
	204	-

Obtener la información de un usuario en particular		
URI	http://www.pescaneo.es/api/users/{user_id}	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	200 - OK	Se devolverá un objeto JSON con la información del usuario con el id igual a user_id. Se proporcionará el nombre de usuario ('nickname'), posición en el ranking general ('position'), los puntos que tiene ('points'), el id de usuario ('iduser'), el nombre ('name'), los apellidos ('surname'), el correo

		electrónico ('email'), la fecha de nacimiento ('date'), la lengua materna ('mother_tongue'), el género ('gender'), la escuela a la que pertenece ('school'), los privilegios ('privileges') y si hay un login activo en ese momento ('success').
	404 – Not found	-
	401 - Unauthorized	-

Actualizar la información de un usuario en particular		
URI	http://www.pescaneo.es/api/users/{user_id}	
Método	PUT	
Cuerpo de consulta	Privileges	Privilegios del usuario
	nickname	Nombre de usuario del usuario
	name	Nombre real del usuario
	surname	Apellidos del usuario
	email	Correo electrónico del usuario
	date	Fecha de nacimiento del usuario
	gender	Género del usuario
	password	Contraseña del usuario
	school	Escuela a la que pertenece el usuario
	mother_tongue	Lengua materna del usuario
	points	Puntos del usuario
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	201 - Created	-

Eliminar un usuario en particular		
URI	http://www.pescaneo.es/api/users/{user_id}	
Método	DELETE	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	204 – No content	-
	401 – Unauthorized	-
	404 – Not found	-

Obtener una lista de los neologismos que ha creado un usuario en particular		
URI	http://www.pescaneo.es/api/users/{user_id}/neologismes	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	200 - OK	Se devuelve un objeto de tipo text/JSON que contiene tres listas con los neologismos propuestos por el usuario con el id de usuario igual a user_id. En una lista se incluyen los neologismos propuestos, pero no aceptados ('proposed'), en otra los aceptados ('accepted') y en otra todos los neologismos que ha propuesto el usuario ('allneos'). De cada neologismo se proporciona el id de neologismo ('id'), el nombre ('neologisme'), el estado ('state') y el

		número de “me gustas” (‘likes’).
	401 - Unauthorized	-
	404 – Not found	-

Obtener una lista de los neologismos que le han gustado a un usuario en particular		
URI	http://www.pescaneo.es/api/users/{user_id}/favs	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	200 – OK	Se devuelve un objeto de tipo text/JSON que contiene una lista con los neologismos que le han gustado al usuario con el id de usuario igual a user_id. De cada neologismo se proporciona el id de neologismo (‘id’), el nombre (‘name’), el estado (‘state’) y el número de “me gustas” (‘likes’) y el nombre de usuario del usuario que lo creó (‘user’).
	404 – Not found	

Hablando ahora de los EndPoints de neologismos, se necesita, al igual que con los usuarios, uno para el conjunto y otro para un neologismo concreto, para lo que utilizaremos ***/neologismes*** y ***/neologismes/{neo_id}***. Adicionalmente, se creará otro para los neologismos de la semana, ***/neologismes/week-neologismes***, y otro más para los “me gusta” de cada neologismo, ***/neologismes/{neo_id}/likes***.

Obtener lista de neologismos del sistema	
URI	http://www.pescaneo.es/api/neologismes
Método	GET

Cuerpo de consulta	-	-
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	200 - OK	Se devuelve un objeto de tipo text/JSON que contiene una lista de todos los neologismos que están dados de alta en el sistema. De cada neologismo se proporciona el id de neologismo ('id'), el nombre ('neologismo'), el estado ('state') y el número de "me gustas" ('likes') y el nombre de usuario del usuario que lo creó ('user'), la posición que ocupa el neologismo en el ranking ('position'), la fecha de aprobación si corresponde ('date'), una lista con los textos de las descripciones ('descriptions') y otra lista con los enlaces de las fuentes ('sources').
	204 – No content	-

Añadir un neologismo al sistema		
URI	http://www.pescaneo.es/api/neologismes	
Método	POST	
Cuerpo de consulta	name	Nombre del neologismo
	likes	Número de "me gusta" del neologismo
	state	Estado actual del neologismo
	user	Nombre de usuario que ha creado el neologismo

	descriptions	Lista con las descripciones del neologismo
	sources	Lista con las fuentes del neologismo
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	201 - Created	-

Obtener lista de los 5 mejores neologismos de la semana		
URI	http://www.pescaneo.es/api/neologismes/week-neologismes	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	200 – OK	Se devuelve un objeto de tipo text/JSON que contiene una lista de los cinco neologismos con más “me gusta” del sistema. De cada neologismo se proporciona el id de neologismo (‘id’), el nombre (‘neologismo’), el número de “me gustas” (‘liked’), el nombre de usuario del usuario que lo creó (‘user’), la posición que ocupa el neologismo en el ranking (‘position’), la fecha de aprobación si corresponde (‘date’) y una lista con los textos de las descripciones (‘descriptions’).
	204	-

Obtener la información de un neologismo en particular		
URI	http://www.pescaneo.es/api/neologismes/{neo_id}	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	200 - OK	Se devolverá un objeto JSON con la información del neologismo con el id igual a neo_id. Se proporcionará el nombre ('neologisme'), el número de "me gustas" ('liked'), el estado del neologismo ('state'), el nombre de usuario del usuario que lo creó ('user'), la fecha de aprobación si corresponde ('date'), una lista con los textos de las descripciones ('descriptions') y otra con los enlaces de las fuentes del neologismo ('sources').
	404 – Not found	-

Actualizar la información de un neologismo en particular		
URI	http://www.pescaneo.es/api/neologismes/{neo_id}	
Método	PUT	
Cuerpo de consulta	do	Acción a realizar sobre el neologismo. Los posibles valores son 'accept' para aceptarlo, 'reject' para rechazarlo y 'modify' para modificarlo

	name	Nombre del neologismo. Solo para cuando 'do' es igual a 'modify'.
	numDescr	Número de descripciones. Solo para cuando 'do' es igual a 'modify'.
	numSourc	Número de fuentes. Solo para cuando 'do' es igual a 'modify'.
	description{i}	Descripción número i del neologismo. Solo para cuando 'do' es igual a 'modify'.
	source{i}	Fuente número i del neologismo. Solo para cuando 'do' es igual a 'modify'.
	message	Mensaje de explicación del rechazo del neologismo. Solo aplicable a las peticiones con el campo 'do' igual a 'reject'.
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	201 - Created	-

Eliminar un neologismo en particular		
URI	http://www.pescaneo.es/api/neologismes/{neo_id}	
Método	DELETE	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	204 – No content	-
	401 – Unauthorized	-
	404 – Not found	-

Obtener una lista de los usuarios a los que les ha gustado un neologismo en particular		
URI	http://www.pescaneo.es/api/neologismes/{neo_id}/likes	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	200 – OK	Se devuelve un objeto de tipo text/JSON que contiene una lista con los usuarios a los que les ha gustado el neologismo con el id igual a neo_id. De cada usuario se proporciona el nombre de usuario ('nickname').
	404 – Not found	-

Crear un "me gusta" en un neologismo en particular del usuario actual		
URI	http://www.pescaneo.es/api/neologismes/{neo_id}/likes	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	201 - Created	-
	404 – Not found	-
	401 - Unauthorized	-

Eliminar un "me gusta" en un neologismo en particular del usuario actual	
URI	http://www.pescaneo.es/api/neologismes/{neo_id}/likes
Método	DELETE

Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	204 – No content	-
	404 – Not found	-
	401 - Unauthorized	-

Para los logros se usará ***/badges*** y para las notificaciones de error ***/error***.

Obtener lista de los logros de un usuario		
URI	http://www.pescaneo.es/api/badges	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	200 - OK	Se devuelve un objeto de tipo text/JSON que contiene una lista de todos los logros que ha conseguido el usuario que tiene iniciada la sesión en el momento de la petición. De cada logro se proporcionará el nombre ('Nombre'), acción que hay que hacer para conseguirlo ('Acción'), los puntos que da al usuario que lo consigue ('Puntos') y la fecha actual ('Fecha').
	204 – No content	-
	401 – Unauthorized	-

Añadir un logro para un usuario	
URI	http://www.pescaneo.es/api/badges
Método	POST

Cuerpo de consulta	achiev	Tipo de logro que se quiere dar al usuario. Actualmente los posibles valores son '5' para el logro con id 5 y 'login' para los logros de inicio de sesión.
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	201 - Created	-
	204 – No content	-
	401 - Unauthorized	-

Obtener lista de las notificaciones de error del sistema		
URI	http://www.pescaneo.es/api/error	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	200 - OK	Se devuelve un objeto de tipo text/JSON que contiene una lista de las notificaciones de error que han dado de alta en el sistema los usuarios. De cada logro se proporcionará el nombre de usuario del usuario que ha creado la notificación ('user') y el texto que describe el error encontrado.
	204 – No content	-
	401 – Unauthorized	-

Añadir una notificación de error del sistema	
URI	http://www.pescaneo.es/api/error
Método	POST

Cuerpo de consulta	description	Descripción del error encontrado.
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	201 - Created	-
	401 - Unauthorized	-

También se necesitan URIs para los inicios de sesión, para los que se usará ***/login*** y ***/logout***.

Obtener la información del usuario que tiene sesión iniciada		
URI	http://www.pescaneo.es/api/login	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	200 - OK	Se devuelve un objeto de tipo text/JSON que contiene una lista con la información del usuario que ha iniciado sesión. Se incluye un atributo booleano de éxito ('success'), el nombre de usuario del usuario ('username'), el id de usuario ('id') y los privilegios del usuario ('privileges').
	401 – Unauthorized	Se devuelve un objeto de tipo text/JSON que contiene una lista con la información del usuario que ha iniciado sesión, pero como no existe, solo se incluye un atributo booleano de éxito ('success') igual a False.

Añadir una sesión de usuario		
URI	http://www.pescaneo.es/api/login	
Método	POST	
Cuerpo de consulta	username	Nombre de usuario del usuario del cual se pretende iniciar sesión.
	password	Contraseña del usuario del cual se pretende iniciar sesión.
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	200 - OK	-
	400 – Bad request	-

Eliminar (cerrar) la sesión de usuario actual		
URI	http://www.pescaneo.es/api/logout	
Método	GET, POST	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	Cookie	Session = id_session
Estado de respuesta-cuerpo de respuesta	204 – No content	-

Por último, queda comentar las URIs para la recuperación de la contraseña, que serán ***/reset-password*** para la petición de restablecer la contraseña y el envío del correo electrónico y ***/reset-password/{token}*** para el restablecimiento concreto de una contraseña.

Crear una solicitud de restablecimiento de contraseña		
URI	http://www.pescaneo.es/api/reset_password	
Método	POST	
Cuerpo de consulta	email	Correo electrónico asociado a la cuenta de la cual se quiere restablecer la contraseña.

Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta	200 - OK	-
	204 – No content	-

Restablecer la contraseña de un usuario con token		
URI	http://www.pescaneo.es/api/reset_password/{token}	
Método	GET	
Cuerpo de consulta	-	-
Cabecera de consulta - valor	-	-
Estado de respuesta-cuerpo de respuesta		
	200 - OK	Renderización de html para la reconfiguración de contraseña y redirección a página de inicio de sesión.
	404 – Not found	Redirección a página de reconfiguración de contraseña.

3.2 Arquitectura

Como se ha comentado anteriormente, este es un proyecto que debido a sus dimensiones se fraccionó en dos trabajos de fin de grado, uno dedicado al Front-End y otro al Back-End. En un inicio, se decidió la arquitectura básica que seguiría todo el proyecto.

Por la parte frontal, esta cuenta con Vue como principal componente, dada la simplicidad al centrarse en el modelo vista – controlador, su accesibilidad, versatilidad y ligereza, además de que permite escalar fácilmente. La principal ventaja de este Framework es que solo carga los componentes necesarios en cada momento, ganando mucho en eficiencia. Además, ofrece la posibilidad de utilizar librerías de terceros como Boostrap-Vue, que facilitan mucho la creación de nuevos componentes.

Para la parte trasera o del Back-End, en un inicio se decidió utilizar Flask, el micro Framework que ya se ha comentado y que viene con funcionalidades básicas, pero es muy extensible con los plugins que deseemos. Tras una investigación acerca de si realmente era la mejor opción para este proyecto, en

la que se valoraron otras opciones como Django, se decidió confirmar la elección de utilizar Flask, junto a librerías como las descritas en el apartado de tecnologías, Flask-login, Flask-forms, Flask-cors, Flask-session, Werkzeug Security… La API que crearemos en Flask hará a su vez llamadas a una base de datos, para lo cual se ha escogido el motor SQLite3 mediante la interfaz de SQLAlchemy adaptada a Flask con la librería Flask-SQLAlchemy.

Para conectar estas dos partes, frontal y trasera, es necesario hacer llamadas desde el Front-End hacia el Back-End utilizando la API creada con Flask. Para esta tarea lo ideal es utilizar Axios, que facilita las llamadas, permitiendo una gran configurabilidad, necesaria para, por ejemplo, elegir el uso que queremos hacer de las cookies.

Descritos todos los elementos del proyecto, la configuración general de estos se puede comprender mejor mediante la ilustración 18.

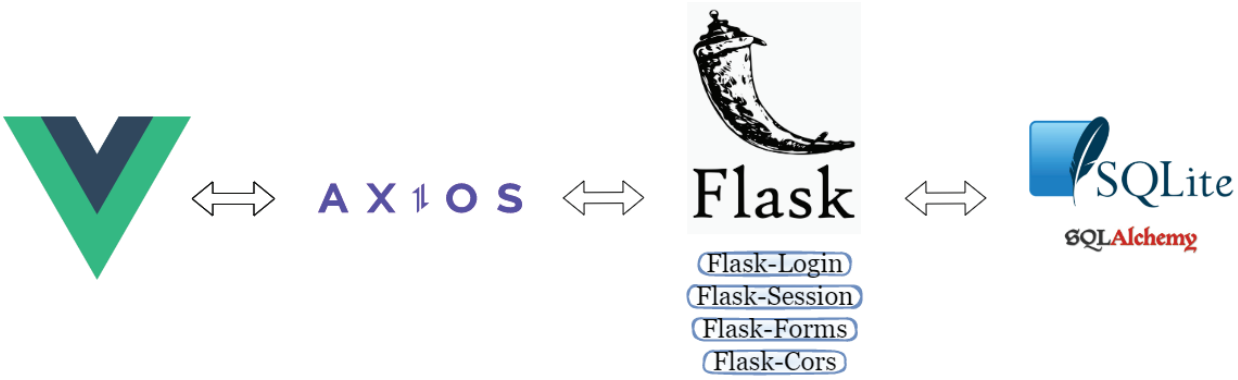


Ilustración 18. Arquitectura de la aplicación de neologismos

3.3 Implementación del Back-End

Esta sección se va a dedicar a la explicación del proceso de implementación o codificación del programa que sirve como servidor de Back-End, pasando por la creación de la base de datos y cada uno de los métodos que dan servicio a la API. Para ello se irán introduciendo también fragmentos del código de la aplicación en Python.

Como se ha descrito en la sección de tecnologías, la herramienta a utilizar durante el proceso como entorno de programación es VSCode. En este programa, la estructura general del proyecto se representa en la ilustración 19.

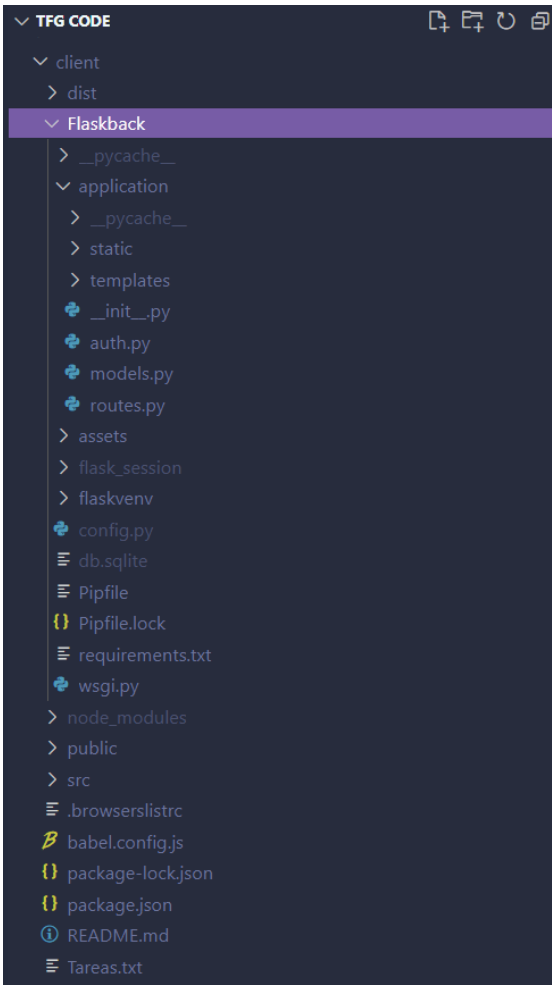


Ilustración 19. Estructura del proyecto en VSCode

La parte relacionada con el Back-End se encuentra alojada en la carpeta denominada *Flaskback*. El EntryPoint de la aplicación, que sirve para ejecutarla e iniciar el servidor en el puerto 5000 del localhost, es el fichero *wsgi.py*, por lo que para iniciar la aplicación se deberá ejecutar el siguiente comando:

```
➢  python wsgi.py
```

Es necesario encontrarse en la misma carpeta que el fichero para que funcione.

La base de datos se encontrará en la misma carpeta que el fichero anterior, denominada *db.sqlite*. Si todavía no se ha ejecutado el proyecto, se creará la primera vez que se haga. El siguiente fichero al que se debe prestar atención es *requirements.txt*. En él se encuentran todos los paquetes necesarios para el correcto funcionamiento de la aplicación de Flask y las versiones exactas con

las que no habrá ningún problema. Los requerimientos básicos y sus versiones de esta aplicación Flask se pueden observar en la figura 20.

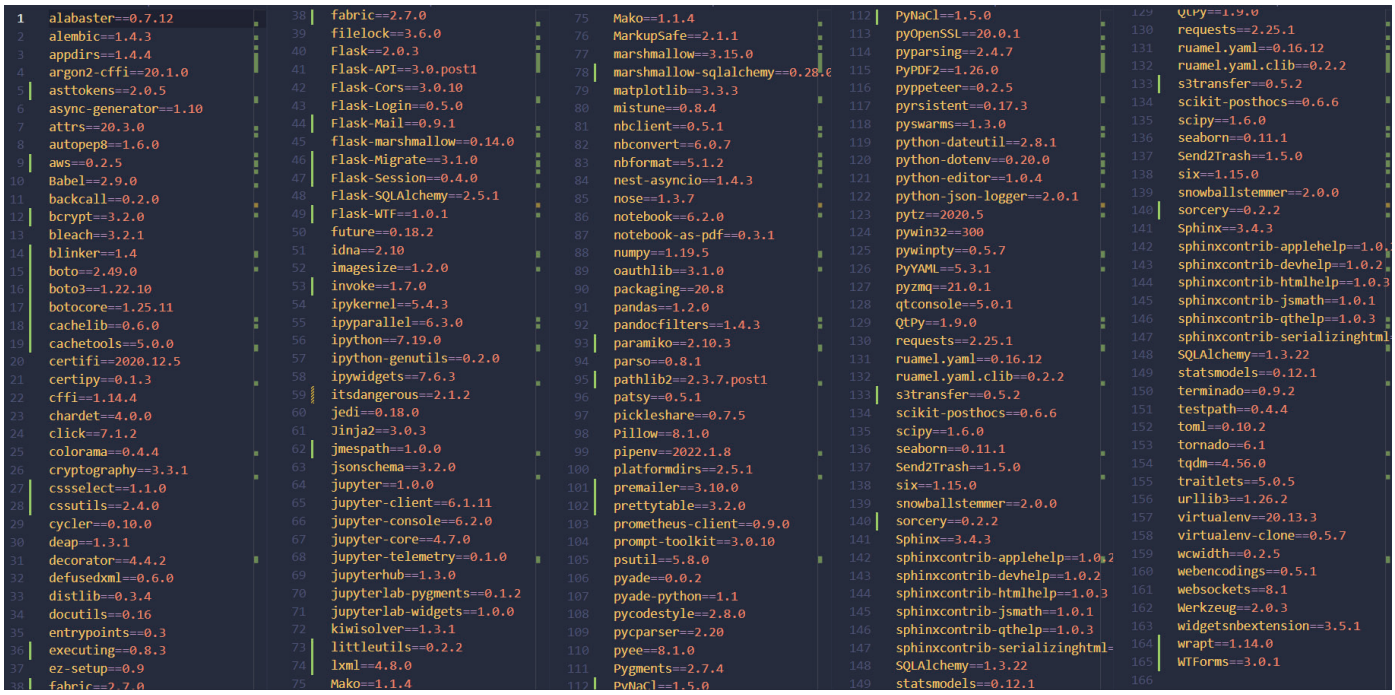


Ilustración 20. Requerimientos básicos de la aplicación

Flask también requiere de un fichero de configuración (también se puede utilizar la configuración mediante código, pero es más aparatoso y desordenado), que se encuentra en la misma carpeta con el nombre *config.py*.

Se pasa ahora a hablar de la carpeta que contiene gran parte del trabajo realizado en este proyecto, el de *application*. En ella se encuentran los ficheros que contienen los modelos de la base de datos y los métodos a ejecutar cuando se hagan peticiones a la API.

Pero antes de definir ninguna función, hay que crear la base de datos. Como se va a utilizar SQLAlchemy, no es necesario llevar a cabo la ejecución de las sentencias SQL directamente, sino que en el fichero *models.py* se definirá cada una de las clases de objetos que se necesitan.

Comenzando con la clase de usuario, cuyos atributos ya se han comentado en el apartado del diagrama de entidad-relación, el código para crear esta clase es el siguiente:

```
class Usuario(UserMixin, db.Model):
    __tablename__ = "usuarios"
    id = db.Column(db.Integer, db.Sequence('id_user'), primary_key=True)
    nickname = db.Column(db.String(20), unique=True, nullable=False)
    name = db.Column(db.String(30), nullable=False)
    surname = db.Column(db.String(100), nullable=False)
    email = db.Column(db.String(100), unique=True, nullable=False)
    birthdate = db.Column(db.DateTime)
    gender = db.Column(db.String(20))
    password = db.Column(db.String(200), nullable=False)
    school = db.Column(db.String(15), nullable=False)
    mother_tongue = db.Column(db.String(20))
    image = db.Column(db.LargeBinary)
    points = db.Column(db.Integer, nullable=False)
    privileges = db.Column(db.String(15), nullable=False)

    def set_password(self, password):
        """Create hashed password."""
        self.password = generate_password_hash(
            password,
            method='sha256'
        )

    def check_password(self, password):
        """Check hashed password."""
        return check_password_hash(self.password, password)

    def get_token(self):
        serial = TimestampSigner('fmVb@st^jCP7f$uM')
        return serial.sign(str(self.id)).decode('utf-8')

    @staticmethod
    def verify_token(token):
        serial = TimestampSigner('fmVb@st^jCP7f$uM')
        try:
            user_id = int(serial.unsign(token, max_age=600).decode('utf-
8'))
        except:
            return None
        return Usuario.query.get(user_id)

    def __repr__(self):
        return '<User {}>'.format(self.nickname)
```

En un inicio se definen todos los atributos de clase, cuyo equivalente en SQL es el siguiente:

```
CREATE TABLE usuarios (
      id INTEGER NOT NULL,
      nickname VARCHAR(20) NOT NULL,
      name VARCHAR(30) NOT NULL,
      surname VARCHAR(100) NOT NULL,
      email VARCHAR(100) NOT NULL,
      birthdate DATETIME,
      gender VARCHAR(20),
      password VARCHAR(200) NOT NULL,
      school VARCHAR(15) NOT NULL,
      mother_tongue VARCHAR(20),
      image BLOB,
      points INTEGER NOT NULL,
      privileges VARCHAR(15) NOT NULL,
      PRIMARY KEY (id),
      UNIQUE (nickname),
      UNIQUE (email)
)
```

A continuación, se definen dos métodos para la configuración y comprobación seguras de las contraseñas (más información en apartado de seguridad). Y, finalmente, se definen dos métodos más para la gestión de restablecimientos de contraseña (más información en apartado de seguridad). También se define una función para la representación de un objeto usuario, algo que ha sido útil en el proceso de codificación.

Para las demás clases se utiliza una fórmula mucho más simple, ya que solo se definen las columnas de la base de datos. Como ejemplo ilustrativo, a continuación, se muestra el código de la clase neologismo:

```
class Neologismo(db.Model):
    __tablename__ = "neologismos"
    id_neologisme = db.Column(db.Integer, db.Sequence(
        'id_neologismo'), primary_key=True)
    name = db.Column(db.String(100))
    likes = db.Column(db.Integer)
    image = db.Column(db.LargeBinary)
    date_approved = db.Column(db.DateTime)
    state = db.Column(db.String(20))
    id_user = db.Column(db.Integer, db.ForeignKey("usuarios.id"))
    user = db.relationship("Usuario", backref="neologismes")
```

En el fichero *__init__.py* está escrito el código que crea e inicializa todos los objetos necesarios para el funcionamiento de la aplicación. Este código es el siguiente:

```
db = SQLAlchemy()
ma = Marshmallow()
login_manager = LoginManager()
s = Session()
mail = Mail()
app = Flask(__name__, instance_relative_config=False)


def create_app():

    # CORS
    Cors = CORS(app)
    CORS(app, resources={r'/*': {'origins': '*'}},
         CORS_SUPPORTS_CREDENTIALS=True)

    # Application Configuration
    app.config.from_object('config.Config')

    # Initialize Plugins
    db.init_app(app)
    ma.init_app(app)
    login_manager.init_app(app)
    mail.init_app(app)

    with app.app_context():
        from . import routes
        from . import auth

        # Register Blueprints
        app.register_blueprint(routes.main_bp)
        app.register_blueprint(auth.auth_bp)

        # Create Database Models
        db.create_all()

        s.init_app(app)

        return app
```

Hay ciertos elementos cuyo objetivo no es obvio, como los blueprints o CORS. Los primeros sirven para separar dos partes de la API, la dedicada a las sesiones y a la creación de usuarios (lo más relacionado con la autenticación) y todo lo demás.

CORS en cambio, sirve para permitir las peticiones que no vengan desde el mismo servidor en el que está alojado el Back-End. Si esta aplicación llegase a desplegarse, se debería permitir tan solo al servidor de Front-End hacer peticiones, ya que de otra manera se podría crear una brecha de seguridad (esto se verá con más detalle en el capítulo de seguridad).

Ahora se pasa a observar el contenido del fichero auth.py. En él se encuentran los métodos asociados con la blueprint auth, relacionada con la autenticación.

Estos son el de login, para comprobar si hay un inicio de sesión (y devolver la información del usuario que la tiene), al hacer GET, y para hacer un intento de inicio de sesión, cuando se hace POST.

```
@auth_bp.route('/login', methods=['POST', 'GET'])
@cross_origin(origin='*', headers=['content-type'],
supports_credentials=True)
def login():
    if request.method == "POST":
        if current_user.is_authenticated:
            return "Ok", 200
        usuario = Usuario.query.filter(((Usuario.email == request.json[
            'username']) | (Usuario.nickname ==
            request.json['username']))).first()
        pwd = request.json['password']
        if usuario is None or not usuario.check_password(password=pwd):
            return "Usuario y/o contraseña incorrectos", 400
        else:
            session.permanent = True
            login_user(usuario)
            return "Ok", 200
    elif request.method == "GET":
        res_fields = {
            'success': False,
            'username': "",
            'image': "",
            'user_id': 0
        }
        if current_user.is_authenticated:
            res_fields['success'] = True
            res_fields['username'] = current_user.nickname
            res_fields['user_id'] = current_user.id
```

```
            res_fields['privileges'] = current_user.privileges
            return res_fields, status.HTTP_200_OK

        return res_fields, status.HTTP_401_UNAUTHORIZED
```

En caso de hacer POST, se comprueba primero si ya hay una sesión iniciada, y si la hay se retorna. Sino, se busca en la base de datos un usuario con ese nombre de usuario y si existe se comprueba que sea la contraseña correcta. Si no se encuentra o la contraseña es incorrecta se devuelve error. Si, en cambio, se hace GET, se devuelve la información del usuario que tiene iniciada sesión y, si no existe, información vacía.

La otra función que contiene *auth.py* es signup(), dedicada a crear usuarios. El código es el siguiente.

```
@auth_bp.route('/users', methods=['POST'])
@cross_origin(origin='*', headers=['content-type'],
supports_credentials=True)
def signup():
    username = request.form['nickname']
    passw = request.form['password']
    email = request.form['email']
    name = request.form['name']
    surname = request.form['surname']
    birthd = request.form['date']
    gender = request.form['gender']
    school = request.form['school']
    mother_tongue = request.form['mother_tongue']
    points = request.form['points']
    privileges = request.form['privileges']

    # Busqueda de usuario ya existente
    if Usuario.query.filter_by(nickname=username).count() > 0 \
            or Usuario.query.filter_by(email=username).count() > 0 \
            or Usuario.query.filter_by(email=email).count() > 0:
        return "Usuario ya existente", status.HTTP_400_BAD_REQUEST

    # Creacion y adicion del usuario
    new_user = Usuario(nickname=username,
                       email=email,
                       name=name,
                       surname=surname,
                       birthdate=datetime.strptime(birthd,'%Y-%m-%d'),
                       gender=gender,
                       school=school,
```

```
                        mother_tongue=mother_tongue,
                        points=points,
                        privileges=privileges)
    new_user.set_password(passw)
    db.session.add(new_user)
    try:
        db.session.commit()
    except:
        db.session.rollback()
    return "ok", status.HTTP_201_CREATED
```

Primero recoge todos los campos de la petición, después comprueba que no exista otro usuario con ese nombre de usuario o correo electrónico y después crea el usuario con los campos anteriormente recogidos y lo añade a la base de datos.

Por último, hay dos métodos más que son necesarios para que Flask-login pueda realizar sus funciones con normalidad. Estos son load_user() y unauthorized().

```
@login_manager.user_loader
def load_user(user_id):
    """Check if user is logged-in on every page load."""
    if user_id is not None:
        return Usuario.query.get(user_id)
    return None

@login_manager.unauthorized_handler
def unauthorized():
    return "You need to be logged in to enter this place.",
status.HTTP_401_UNAUTHORIZED
```

Ahora se va a observar el contenido del fichero *routes.py*, que contiene todas las demás rutas de la API que no se han repasado ya. Como se ha podido observar en los métodos anteriores y en los de routes.py, la gran mayoría llevan antes unas etiquetas. Las más usadas son `@login_required`, que sirve para obligar a quien envía la petición a tener sesión iniciada; `@{blueprint}.route('{route}', methods=[{methods}])` para configurar la ruta 'route' y los posibles métodos 'methods' de la blueprint 'blueprint' para ese método. La etiqueta `@cross_origin(origin='*', headers=['content-type'], supports_credentials=True)` permite el acceso al método desde una fuente diferente al mismo servidor en el que se ejecuta el Back-End y también el uso de cookies de sesión en la petición.

Los métodos que se encuentran dentro de *routes.py* responden a los métodos GET, POST, PUT y DELETE que se hacen a los EndPoints descritos en el apartado de API. Para hacerse una buena idea del funcionamiento de todos ellos no es necesario mostrarlos uno a uno, así que se va a mostrar un ejemplo de cada tipo de método para diferentes recursos.

Se empieza con la operación GET de */usuarios*, que tiene el siguiente código.

```
@main_bp.route('/users', methods=['GET'])
@cross_origin(origin='*', headers=['content-type'],
supports_credentials=True)
@login_required
def getallusers():
    if current_user.privileges == 'admin':
        res = Usuario.query.order_by(Usuario.points.desc()).filter(
            Usuario.privileges != 'removed').all()
    elif current_user.privileges == 'linguist':
        res = Usuario.query.order_by(Usuario.points.desc()).filter(
            Usuario.privileges != 'admin' and
             Usuario.privileges != 'removed').all()
    else:
        res = Usuario.query.order_by(
            Usuario.points.desc()).filter_by(privileges='user').all()
    users = []
    for i, user in enumerate(res):
        usuario = {}
        usuario['nickname'] = user.nickname
        usuario['position'] = i+1
        usuario['points'] = user.points
        usuario['id'] = user.id
        usuario['privileges'] = user.privileges
        users.append(usuario)
    return jsonify(users), status.HTTP_200_OK
```

En este caso se quiere hacer un filtrado según los privilegios del usuario que hace la petición por lo que, si es un administrador, se recogerán de la base de datos todos los usuarios, mientras que si es lingüista recogerá los usuarios con permisos de lingüista o usuario y si es un usuario solo recogerá los usuarios. Después, rellena una lista con diccionarios de cada usuario para devolverla como JSON, ya que esta es la manera más sencilla para trabajar en Flask, con listas de diccionarios, que luego se recorren muy fácilmente desde el Front-End de Vue.

A continuación, se va a observar el método PUT del EndPoint */neologismes/{neo_id}*.

```
@main_bp.route('/neologismes/<neoid>', methods=['GET', 'PUT', 'DELETE'])
@cross_origin(origin='*', headers=['content-type'], supports_credentials=True)
def neo(neoid):
    if request.method == 'GET':
        ...
        ...
    elif request.method == 'PUT':
        neo = Neologismo.query.get(neoid)
        try:
            method = request.form['do']
            if method == 'accept':
                neo.state = 'aceptado'
                neo.date_approved = datetime.date.today()
                uga = UserGetsAchievement(
                 id_user=neo.id_user, id_achievement=1, date=datetime.date.today())
                db.session.add(uga)
                user = Usuario.query.get(neo.id_user)
                user.points += 50
                try:
                    db.session.commit()
                except:
                    db.session.rollback()
                    return "Something went wrong while commiting",
                        status.HTTP_500_INTERNAL_SERVER_ERROR

                ugaq = UserGetsAchievement.query.filter(
                         (UserGetsAchievement.id_user==neo.id_user) &
                         (UserGetsAchievement.id_achievement==1)).all()
                if len(ugaq) == 3:
                    uga = UserGetsAchievement(
                        id_user=neo.id_user, id_achievement=18,
                        date=datetime.date.today())
                    db.session.add(uga)
                    user.points += 80
                elif len(ugaq) == 6:
                    uga = UserGetsAchievement(
                        id_user=neo.id_user, id_achievement=19,
                        date=datetime.date.today())
                    db.session.add(uga)
                    user.points += 100
            elif method == 'reject':
                neo.state = 'rechazado: ' + request.form['message']
            elif method == 'modify':
                neo.name = request.form['name']
```

```
                neo.state = 'pendiente'

                descriptions = []
                for i in range(int(request.form['numDescr'])):
                    descriptions.append(request.form['description'+str(i)])

                sources = []
                for i in range(int(request.form['numSourc'])):
                    sources.append(request.form['source'+str(i)])

                Description.query.filter_by(id_neologisme=neoid).delete()
                Source.query.filter_by(id_neologisme=neoid).delete()
                for i in range(len(sources)):
                    new_source = Source(
                        link=sources[i], id_neologisme=neoid)
                    db.session.add(new_source)

                for i in range(len(descriptions)):
                    new_description = Description(
                        text=descriptions[i], id_neologisme=neoid)
                    db.session.add(new_description)
            try:
                db.session.commit()
            except:
                db.session.rollback()
                return "Something went wrong while commiting",
                        status.HTTP_500_INTERNAL_SERVER_ERROR
        except Exception as e:
            print('Modifying: ', e)
            return "Something went wrong", status.HTTP_500_INTERNAL_SERVER_ERROR
        return "neologism succesfully modified", status.HTTP_201_CREATED

    elif request.method == 'DELETE':
        ...
        ...
```

Como se puede observar, en este método se encuentran las tres operaciones GET, PUT y DELETE, ya que basta con comprobar el método de la consulta para diferenciarlos.

Lo primero que se hace, si el método de la petición es PUT, es recoger el neologismo en cuestión de la base de datos, para después poder ir editando sus propiedades. Después se comprueba un campo del cuerpo de la petición,

concretamente '*do*', que corresponde con la acción a realizar sobre el neologismo.

Si la acción es aceptarlo, se cambia el estado del neologismo a 'aceptado', su fecha de aprobación a hoy, se le da al usuario que lo ha propuesto el logro de aprobación de un neologismo del usuario y se le añaden los puntos correspondientes. Después, se comprueba cuántos logros de este tipo tiene el usuario, para darle otro de acumulación si corresponde y añadirle los puntos.

Si, en cambio, la acción a realizar es rechazar el neologismo, el estado se cambia a rechazado, incluyendo el mensaje que viene en el campo 'message' para explicar al usuario el porqué.

Finalmente, si la acción es modificar el neologismo, se recopilan los campos necesarios del cuerpo de consulta, se modifican en el que se traído de la base de datos y se hace *commit* para guardarlo de nuevo. Es necesario resaltar que el método para la recogida de las fuentes y las descripciones es algo rudimentario, pero funciona excelentemente. Para guardar estos elementos en la base de datos hay que crear un objeto por cada uno de ellos que haya, ya que tienen sus propias tablas.

Vamos a pasar ahora con otro método y recurso, POST y */error*. Este es el código del método.

```
@main_bp.route('/error', methods=['GET', 'POST'])
@cross_origin(origin='*', headers=['content-type'],
supports_credentials=True)
@login_required
def error():
    if request.method == 'POST':
        text = request.form['description']
        error = ErrorNotification(id_user=current_user.id,
                  description=text)
        db.session.add(error)
        try:
            db.session.commit()
        except:
            db.session.rollback()
            return "Something went wrong while commiting",
                  status.HTTP_500_INTERNAL_SERVER_ERROR
        return "Error notification created", status.HTTP_201_CREATED
    elif request.method == 'GET':
            ...
```

Si el método es POST, se recoge el campo 'description' del cuerpo de consulta y se crea el objeto de notificación de error para después añadirlo a la base de

datos y cometer para guardarlo definitivamente. Este es un ejemplo muy simple, pero ilustra a la perfección cuál es el esqueleto básico de todos los métodos POST.

Por último, se va a observar un método DELETE, concretamente sobre el recurso */neologismes/<neoid>/likes*.

```
@main_bp.route('/neologismes/<neoid>/likes',
      methods=['GET', 'POST', 'DELETE'])
@cross_origin(origin='*', headers=['content-type'],
      supports_credentials=True)
@login_required
def neolikes(neoid):
    if request.method == 'GET':
            ...
    elif request.method == 'POST':
            ...
    else:  # DELETE
        UserlikesNeologisme.query.filter(
            (UserlikesNeologisme.id_neologisme == neoid) &
            (UserlikesNeologisme.id_user == current_user.id)).delete()
        neo = Neologismo.query.get(neoid)
        neo.likes -= 1
        try:
            db.session.commit()
        except:
            db.session.rollback()
            return "Something bad happened while commiting",
                    status.HTTP_500_INTERNAL_SERVER_ERROR
        return "Removed succesfully", status.HTTP_204_NO_CONTENT
```

Lo primero que se hace es hacer una búsqueda en la base de datos del “me gusta” en cuestión y del neologismo al que corresponde. El primero se elimina y al segundo se le resta una unidad de la cantidad de ‘likes’. Después, se comete a la base de datos y se devuelve un código de respuesta HTML 204 que implica falta de contenido, como se describe en el estándar.

Una vez mostrados los ejemplos de los métodos disponibles en la API, cabe hacer una mención especial al sistema diseñado para la recuperación de la contraseña, que es algo complicado debido a que cuenta con nivel considerable de seguridad, como se explicará con detalle en la sección de seguridad.

Con esto se ha dejado sin mostrar una gran cantidad de código, pero con lo explicado se espera que sea necesario para comprender lo demás que falta en este documento.

3.4 Cambios en el Front-End

Aunque este trabajo de fin de grado se centraba en desarrollar un Back-End funcional para el Front-End ya existente, en este último eran necesarias numerosas modificaciones, entre las que se encuentran las llamadas al servidor de Back-End y el manejo de los datos una vez recibidos. También se han hecho múltiples modificaciones visuales, la mayoría sutiles, para perfeccionar la interfaz. Además, en alguna ocasión se ha requerido de la creación de nuevos componentes por la ausencia de estos anteriormente. Por todo esto, no ha sido necesario solo aprender las tecnologías relacionadas con el Back-End, sino que era fundamental estar muy familiarizado con el código del Front-End. A continuación, se van a visitar algunas de estas modificaciones.

Para empezar, veamos lo que ha sido primordial realizar en todos y cada uno de los componentes, la realización de las llamadas al Back-End y el manejo de los datos al recibirlos. Es natural que Javier, la persona que realizó todas las tareas relacionadas con el Front-End y alguna más, no supiese cómo se iba a realizar exactamente toda la parte del Back-End, por lo que hizo lo que él creyó conveniente. Por desgracia, no acertó en muchas de las cosas, por lo que ha habido que retocar muchas partes de código ya existente.

En cuanto a las llamadas, el puerto del servidor era diferente, y era preciso añadir algunos parámetros para la configuración de las cookies, algo esencial para el trabajo con sesiones de Flask. Adicionalmente, en múltiples ocasiones se han dado errores de recogidas de datos tardías, resultando en el mal funcionamiento de la aplicación, por lo que se requería cierto orden de llegada de distintas llamadas y eso ha sido una tarea ardua con aparente fácil solución, pero, en la práctica, origen de muchos quebraderos de cabeza.

En relación con el manejo de datos, también ha sido necesario realizar modificaciones en la mayoría de los componentes, ya que se tratan de diferente forma a la que se había planeado. Esto, en su mayoría, se debe a que anteriormente la mayoría de las modificaciones y filtros se hacían tras recibir los datos y en el Front-End, mientras que, de la manera en la que se ha desarrollado el Back-End, no hace falta realizar tantas acciones en el Front-End. Es por esto por lo que todas y cada una de las funciones de los ficheros *.vue* han sido modificadas y reducidas para encajar con las funciones de Flask. Por ejemplo, si antes se realizaba un filtrado de fechas en los neologismos recibidos al hacer GET sobre */neologismes*, ahora se realiza en el Back-End devolviendo ya los resultados filtrados si el GET se hace sobre */neologismes/week-neologismes*.

También existía otro problema, y es que muchas veces cuando se le da a un botón en la interfaz, la función de este es cambiar valores o elementos de la interfaz, por lo que es necesario actualizar estos elementos y valores. Un ejemplo es el cierre de sesión, que limita ciertas funciones al usuario y cambia la interfaz. Para actualizar la interfaz, Vue ofrece unas pocas opciones, algunas que funcionan en algunos casos y otras en otros, por lo que ha habido que ir probando cuál era la más adecuada.

Habiendo visitado la corrección de errores y las adaptaciones, vamos ahora con los elementos que antes no existían y se han creado por distintas razones.

Anteriormente, se planearon numerosas funciones en la aplicación que no llegaron a materializarse de una forma práctica debido a la falta de un código de fondo que las respaldara. Ahora esas funciones, muchas de ellas esenciales, era vital que se añadiesen.

Una de estas funciones es un botón en la clasificación de neologismos, que se muestra si el usuario con sesión iniciada es un administrador o lingüista, y que sirve para revisar las propuestas de neologismos que no han sido aprobadas ni rechazadas todavía. El aspecto del botón se observa en la figura 21, en las dos variantes que se intercambian al pulsar en él.

Ilustración 21. Botón para la revisión de neologismos

También ha habido que añadir más botones, pero esta vez en la barra lateral, fruto de algunas pruebas realizadas con usuarios reales, que se describirán en el apartado de pruebas. Estos botones son el de contribuir, para poder crear una propuesta de neologismo directamente, aunque no se esté en la página principal, y el de neologismos favoritos, para acceder a los que le han gustado al usuario. Además, se añadió otro más de enlace a la página principal para cuando no se está en ella. También se reordenaron el resto de los botones y se añadieron distintos colores para cada botón, quedando como se representa en la imagen 22.

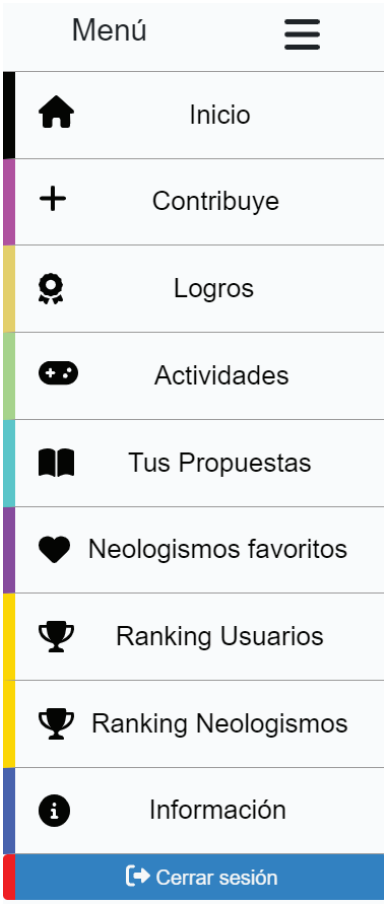


Ilustración 22. Barra lateral modificada

Adicionalmente, en la cinta de opciones de la vista de usuario se modificaron las opciones disponibles para quedar como se ve en la figura 23 para administradores y lingüistas y como se ve en la figura 24 para usuarios.

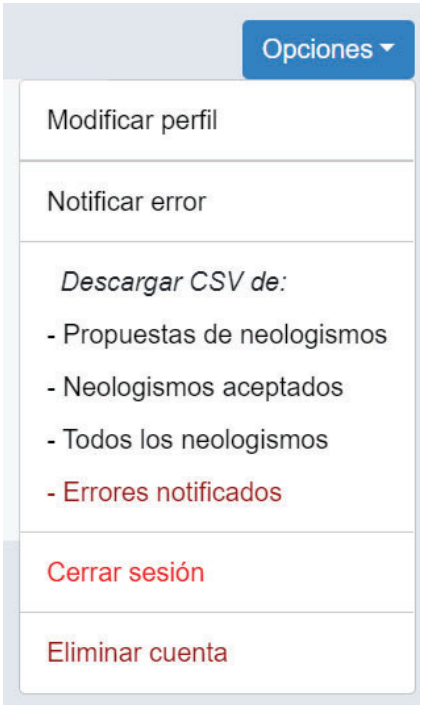


Ilustración 23. Cinta de opciones para el administrador

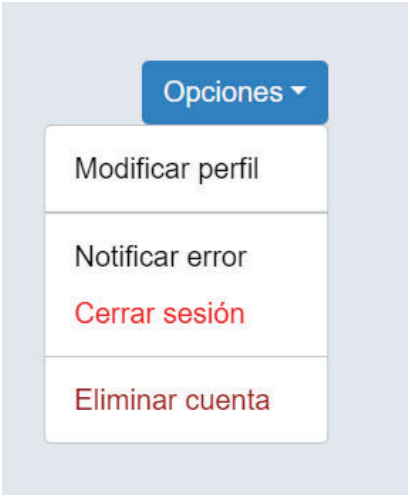


Ilustración 24. Cinta de opciones para usuarios

También se modificó el aspecto de la página principal para modernizarlo y dar una bienvenida más cálida a la aplicación. El resultado se muestra en la ilustración 25.

Ilustración 25. Aspecto modificado de la página principal

Aparte de lo ya descrito, en muchas ocasiones los usuarios realizaban acciones como completar una actividad, crear una propuesta de neologismo, iniciar sesión… que no eran lo suficientemente visualmente recompensadas, por lo que se decidió hacer uso de las notificaciones "*flash*", que informan al usuario de ciertas cosas, como que ha conseguido un nuevo logro, y le animan a continuar con el uso de la aplicación, recomendándole que vaya a la sección de logros, por ejemplo. Estas notificaciones generalmente son de tres tipos, de éxito, como el ejemplo anterior (ilustración 26); de advertencia, como cuando se les envía un correo electrónico para recuperar la contraseña y solo tienen 10 minutos para hacerlo (ilustración 27); y de error, como al marcar un neologismo como rechazado (ilustración 28).

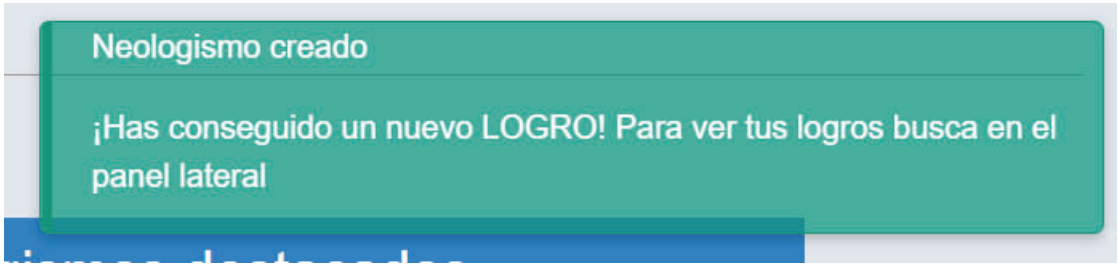


Ilustración 26. Notificación rápida de éxito

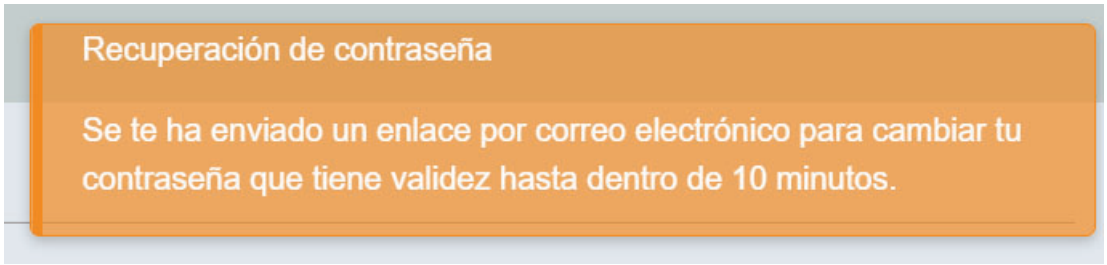


Ilustración 27. Notificación rápida de advertencia

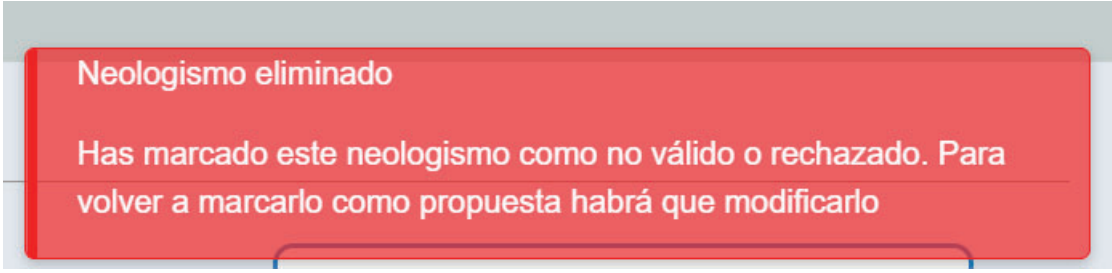


Ilustración 28. Notificación rápida de error

3.5 Seguridad

En este apartado se van a discutir algunas de las consideraciones que ha habido en el desarrollo de la aplicación con el objetivo de aumentar la seguridad.

La que primero surgió, ya que da problemas nada más empezar a enviar solicitudes al servidor de Flask, es **CORS**, el intercambio de recursos de origen cruzado, que es una característica del navegador para aumentar la seguridad y sirve para restringir las solicitudes HTTP.

Cuando un navegador recibe una solicitud HTTP que no es simple, el protocolo CORS requiere que el navegador envíe una solicitud inicial al servidor y espere la aprobación del servidor (o solicite un certificado) antes de enviar la solicitud real. Es por esto por lo que el servidor debe implementar un método OPTIONS para configurar estos accesos. Aquí es donde entra el plugin de Flask, Flask-cors, que facilita todo este proceso. Con ello, solo es necesario una etiqueta en cada método de recurso para configurar los orígenes que se aceptan, como se ha descrito en el apartado de desarrollo del Back-End. Por lo tanto, esta etiqueta de los métodos deberá ser cambiada si se llega a desplegar para el público, ya que, si no se hace, cualquiera sería capaz de acceder a todos los recursos y llegar a escalar privilegios en la aplicación.

Se pasa ahora a comentar los diferentes aspectos de seguridad del elemento más crítico de la mayoría de los softwares, las **contraseñas**. Las contraseñas son una forma de autenticación que utiliza información confidencial para restringir el acceso a una cuenta, datos, recursos, servicios... El uso de estos códigos se remonta a la antigüedad, así como las diferentes maneras de sobrepasar esta seguridad. En la actualidad cada persona cuenta con decenas de cuentas en diferentes servicios, cada uno con su propia contraseña. Por esto, es muy complicado mantener la seguridad de las contraseñas a la vez que se intenta recordar cada una de ellas, ya que, generalmente, cuanto más cómoda de recordar es una contraseña, más facilidad tiene un atacante de adivinarla.

La seguridad de la contraseña viene muy influenciada por la política de contraseñas particular de cada aplicación o página web, que van desde las que no ponen ninguna restricción, hasta las que piden tantos tipos de carácter que es muy difícil acordarse después. En el caso de esta aplicación se ha decidido utilizar un punto medio, con una longitud de entre 6 y 12 caracteres, incluyendo, como mínimo, una mayúscula y un carácter no alfabético. Además, se comprueba que el usuario es consciente de la contraseña que ha escrito mediante la necesidad de la repetición de esta.

Esta es la seguridad que se ha implementado en cuanto a la creación de las contraseñas, pero si la seguridad se quedase ahí habría muchas vulnerabilidades fáciles de explotar. Hay que tener en cuenta más factores, como el almacenamiento de las contraseñas. La opción más fácil y menos segura sería guardarlas en un archivo de texto y como texto en claro, pero esto es inadmisible en cualquier aplicación actual. La práctica más habitual es guardarlas encriptadas de alguna manera, protegiéndolas incluso si alguien ajeno ha conseguido acceso al sistema. En nuestro caso hemos utilizado la función criptográfica **SHA-256**, un algoritmo de hash seguro de 256 bits que forma parte del conjunto de funciones diseñadas por la NSA en 2001 como estándar federal de procesamiento de la información, y que generan hashes irreversibles y únicos para cada contraseña. Este algoritmo es, actualmente, bastante seguro, usándose en la mayoría de los proyectos derivados de Bitcoin. Más concretamente, en el siguiente fragmento de código se muestra el método que utiliza la función hash para crear la cadena que se almacena en la base de datos:

```
def set_password(self, password):
        """Create hashed password."""
        self.password = generate_password_hash(
            password,
            method='sha256'
        )
```

De ahora en adelante, cuando el servidor necesita comprobar si una contraseña es la correcta, lo único que debe hacer es pasarla por la función SHA-256 y comprobar si es la misma cadena que la de la que se encuentra en la base de datos. Esto es lo que se hace en la función del fragmento que se muestra a continuación, que se encuentra, al igual que el anterior, en la clase del modelo del usuario, por lo que tiene acceso directo a la cadena de hash:

```
def check_password(self, password):
        """Check hashed password."""
        return check_password_hash(self.password, password)
```

Ahora se va a visitar otro aspecto en el que se pueden encontrar vulnerabilidades, el restablecimiento de la contraseña en caso de pérdida por parte del usuario. Cuando el usuario prueba a iniciar sesión y no lo consigue tras varios intentos, es habitual que requiera de una función para volver a configurar una contraseña, normalmente por correo electrónico. Para este fin hay muchas opciones, como enviar un **enlace por correo electrónico** para que lo introduzca en su navegador y le lleve a una página de configuración de contraseña, o el envío por SMS de un código numérico que pueda introducir para que le lleve a dicha página. Como es natural, en este proceso se debe tener mucha cautela, ya que es fácil que aparezcan vulnerabilidades. Lo primero que se debe hacer es limitar el tiempo que ese código o enlace es válido. Después, hay que asegurarse de que no es posible reconfigurar la contraseña sin el elemento que se le ha dado al usuario, bien complicando el código o creando un enlace muy largo y aleatorio. En el caso de este proyecto, se ha elegido la opción del enlace, con una duración de 10 minutos (600 segundos), enviándolo por correo electrónico. El enlace es único y se genera a partir del usuario y de una **clave secreta**.

Esta clave se utiliza en distintos aspectos de la aplicación, y se encuentra en el fichero de configuración que se describió, anteriormente. Este fichero se ha eliminado del repositorio público para evitar futuras vulnerabilidades. Es un código con un grado de complejidad muy alto ya que, si se llega a adivinar por un método de fuerza bruta u otros, podría comprometer la aplicación entera.

El código que genera y comprueba los tokens para la recuperación de la contraseña de usuario es el siguiente:

```
    def get_token(self):
        serial = TimestampSigner(app.config.get('SECRET_KEY'))
        return serial.sign(str(self.id)).decode('utf-8')

    @staticmethod
    def verify_token(token):
        serial = TimestampSigner(app.config.get('SECRET_KEY'))
        try:
            user_id = int(serial.unsign(token, max_age=600)
                          .decode('utf-8'))
        except:
            return None
        return Usuario.query.get(user_id)
```

3.6 Pruebas

Todo sistema necesita ser probado. Ya sea software, hardware o de cualquier otro tipo, en el desarrollo siempre se van a pasar por alto ciertos fallos o posibles mejoras que, para el uso general, es esencial calibrar. Esta aplicación no es ninguna excepción, así que a medida que se iban desarrollando las funciones, se iban probando paralelamente. La mayoría de las mejoras fruto de estas pruebas se han ido describiendo a lo largo de este escrito, por lo que no es primordial volver a mostrarlas. Aun así, cuando el sistema ya tenía un grado de completitud lo suficientemente avanzado, se hicieron algunas pruebas con personas reales de entre 19 y 50 años. A estas se les dio el sistema para que lo usaran, tanto dándoles una serie de tareas que llevar a cabo, como sin pedirles nada en específico, simplemente que navegaran por él.

A continuación, se da una reducida lista de diez de las cuestiones que salieron a la luz gracias a estas pruebas para dar una idea general de lo que ayudaron en el perfeccionamiento de la versión final de la aplicación.

- Cuestión 1. Al crear un usuario, en uno de los campos se pedía un alias, algo que algunos usuarios no llegaban a entender del todo, por lo que se cambió a nombre de usuario, que es más convencional.

- Cuestión 2. Al iniciar sesión tras haber creado el usuario, casi todos los usuarios usaban el botón de 'enter' como sustituto a pulsar el botón de enviar, para lo que hay que cambiar la posición de la mano al ratón, pero esta función no estaba implementada, por lo que se añadió sin mucha dificultad gracias a Vue, ya que simplemente hay que añadir un campo *v-on*:

```
<b-form-input
  v-model="form.password"
  type="password"
  placeholder="Contraseña"
  :state="this.correct_login"
  v-on:keypress.enter="getUserInfo"
></b-form-input>
```

- Cuestión 3. En la creación del neologismo, para añadir una descripción, tras haberla escrito es necesario pulsar el botón de añadir, pero algunos usuarios no lo pulsaban al pensar que con escribirlo era suficiente. Esto es algo complicado de implementar porque es un componente más complejo, así que esta funcionalidad se pasará a líneas futuras debido a la falta de tiempo y a la menor prioridad que se le da, puesto que siguen pudiendo añadir estos campos en la modificación de neologismo.

- Cuestión 4. En la actividad, la descripción era demasiado larga y prácticamente ningún usuario se la leyó entera, equivocándose más tarde en el propósito de la propia actividad o sin saber qué hacer exactamente. Por ello, se simplificó mucho la descripción e instrucciones que se dan antes de comenzar y se añadieron colores para clarificar el objetivo, quedando como se ve en la ilustración 29.

Ilustración 29. Explicación modificada de la actividad

- Cuestión 5. La existencia del sistema de logros era desconocida para muchos usuarios incluso algunos minutos después de haber comenzado a usar la aplicación, por lo que se añadió un enlace en la barra lateral y se comenzó a notificar al usuario cada vez que conseguía un logro, siendo el primero nada más iniciar sesión por primera vez.

- Cuestión 6. De una manera parecida al apartado anterior, los usuarios no eran conscientes de que existía o no sabían como visualizar los neologismos que les habían gustado. La solución que se implementó fue la inclusión de este apartado en la barra lateral, aparte de en la página del perfil como ya estaba antes.

- Cuestión 7. También para visualizar más el hecho de que había una página específica para visitar el perfil, se añadió una pequeña imagen o icono al lado del nombre en el botón de enlace en la página principal, quedando finalmente como en la ilustración 30.

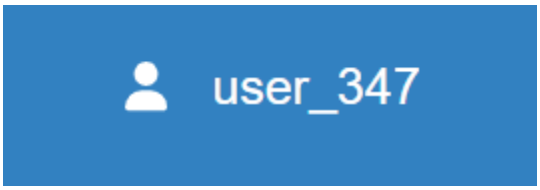


Ilustración 30. Botón de usuario modificado

- Cuestión 8. Una vez creado un neologismo, para comprobar si se lo habían aceptado, los usuarios acudían al apartado de sus propuestas. Aquí en un inicio se encontraban únicamente las propuestas que no habían sido aceptadas, por lo que, si sí que lo habían sido, el usuario estaba confuso al no verlo allí. Por esta razón, en la página de propuestas, se pueden ver todos los neologismos propuestos por el usuario.

- Cuestión 9. Algunos usuarios tardaban algo más en encontrar cómo cerrar sesión cuando se les pedía que lo hicieran, por lo que se añadió también en la barra lateral, mostrando mucha mejora en posteriores pruebas.

- Cuestión 10. Hasta ahora solo se han mostrado algunos de los errores de usabilidad, pero con las pruebas también se encontraron errores técnicos que deberían haberse encontrado en las pruebas anteriores. Algún ejemplo de estos errores es que el botón de iniciar sesión o ir a la página de perfil tan sólo funcionaba si se le clicaba justo en el texto, dejando inutilizado el resto del botón; o que en la actividad se mostraban todos los neologismos, no solo los que estaban aceptados.

4. Líneas futuras

Aunque el desarrollo de la aplicación ha sido un proceso largo y arduo, en el que se han superado muchas barreras y se ha aprendido mucho, han quedado muchas funciones en el plumero que habrían mejorado mucho la versión final. Con un tamaño considerable, esta aplicación ha conllevado muchos retos, y a la hora de priorizar se ha tenido más en cuenta el buen funcionamiento y usabilidad de unas pocas (muchas, en realidad) funciones, sobre más funciones con más fallos o menos calidad. Por esto, se muestran a continuación algunas de las utilidades que se pueden implementar en el futuro.

La primera y más importante de estas funcionalidades es la gestión de imágenes, ya que se comenzó el desarrollo, pero la acción de traerlas desde la base de datos y mostrarlas en el Front-End llevó un quebradero de cabeza que no dio lugar a ninguna solución válida. Es por ello por lo que gran parte del código necesario para esta función ya está escrito (todo comentado para que no de problemas), como la selección de la imagen por el usuario y la subida a la base de datos.

Otras funciones más relacionadas con el Front-End que han quedado por realizar son la elaboración de una página que cargue cuando se haga un GET a un recurso del Front-End y no exista, es decir, una página de error 404; y la eliminación de algunos errores que salen por la consola del navegador en ciertos casos; o también el hecho de que no haga falta pulsar en + para que se añadan las descripciones y fuentes del neologismo, como se ha comentado anteriormente.

Adicionalmente, hay otras funciones que necesitarían de creación de nuevos componentes y elementos en ambos cabos de la aplicación, como son la búsqueda de neologismos por el usuario, el cambio de idioma, la inclusión de comentarios en cada neologismo, más actividades y logros, y por último la posibilidad de visualizar los logros que están implementados en la aplicación y que pueden conseguir los usuarios.

Y, por último, cabe comentar funciones más complejas y amplias como la utilización de OAuth2 para la autenticación sin contraseñas; un sistema de mensajería entre usuarios o para contacto con los administradores; y más generalmente, mejoras visuales, recompensas visuales como marcos de avatar y temas de la aplicación, y mejoras de seguridad, un apartado muy importante cuando las aplicaciones van creciendo.

5. Conclusiones

Este ha sido un proyecto exigente en el que se han requerido de muchos conocimientos nuevos y habilidades que antes no se tenían. Se ha tratado siempre como un proyecto real en el que la fase de preparación y diseño anterior al desarrollo necesita de una importancia mucho mayor que la de cualquier otro proyecto realizado en el grado. Además, se ha ido documentando todos y cada uno de los cambios realizados en los archivos de código del proyecto, pudiéndose visitar en el repositorio público que se describe en el anexo. Siendo un proyecto de larga duración respecto a los anteriormente realizados, ha sido proporcionalmente, o incluso más, satisfactorio a su finalización.

En el comienzo hubo muchos problemas relacionados con la novedad de las tecnologías para el desarrollador, la falta de documentación de algunos aspectos, etc. Aun así, poco a poco se fueron resolviendo uno a uno para llegar a una dinámica de trabajo constante y gratificante que ha llevado al éxito del proyecto.

El equipo principal de desarrollo está formado únicamente por el autor de este documento, pero ha sido esencial la ayuda de colaboradores como el autor de la parte previa a este TFG, Javier Barragán Haro, que ha resuelto cualquier duda que existiera sobre su trabajo; la profesora María de la Nava por su conocimiento en la lingüística y su guía en lo que debía ser la aplicación; la tutora del trabajo de fin de grado, Raquel Cedazo, que ha ayudado en lo que se le ha pedido; y los anónimos usuarios que han probado la aplicación y a los que se les agradecen sus comentarios desinteresados.

A lo largo de este proyecto se ha invertido una cantidad considerable de horas, pero es agradable tener la certeza de que han sido bien invertidas.

6. Análisis de impacto

6.1 Impacto personal

Este ha sido un proyecto muy amplio, de varias maneras. Ha permitido al desarrollador ampliar en gran medida sus **capacidades técnicas**, llevando a áreas muy distintas de la informática y descubriendo algunas a las que antes no se les había dado importancia.

El **aprendizaje** no ha sido sólo técnico, sino que abarca también una serie de habilidades no técnicas como el trabajo en un proyecto de mayor envergadura en el que hay que cumplir una serie de plazos y en el que se necesita una seria planificación. Otra habilidad que se ha ampliado enormemente es la de **resolución de problemas**, sin la cual desde el segundo día de desarrollo no se habría logrado continuar.

Además del desarrollador del proyecto, hay más personas que se han beneficiado o se van a beneficiar del proyecto, como por ejemplo los usuarios, que aprenderán nuevos términos y se verán animados a participar en una comunidad que se espera que sea agradable e instructiva.

6.2 Impacto social

El impacto social que puede provocar este proyecto está relacionado con lo antes mencionado, y es que a muchas personas les servirá para aprender nuevos términos sobre, en este caso, el internet de las cosas, aunque más tarde está pensado que se adapte para recoger términos relacionados con la sostenibilidad. Por esto, el hecho de que un número considerable de personas conozcan más un ámbito tan importante como lo es la sostenibilidad actualmente, es muy importante en el aspecto social.

Adicionalmente, el tipo de gamificación pensado para este proyecto tiene que ver con dos de las dinámicas que se propusieron en 2012 por Eduardo Herranz Sánchez y Ricardo Colomo-Palacios [11], la progresión del participante, que siente que avanza y se ve estimulado para continuar jugando o participando; y la relación con los otros compañeros, incluyendo a los de la misma facultad, pero también a los de otras.

6.3 Impacto medioambiental

Este proyecto se enmarca en el Proyecto de Innovación Educativa de Estrategias de gamificación aplicadas a la adquisición colaborativa de vocabulario científico-técnico en inglés y español que elabora la UPM y que dirige María de la Nava Maroto García y, como se ha comentado en el anterior apartado, más adelante está pensado adaptar esta aplicación a la recogida de términos sobre la sostenibilidad (de hecho, ya están pensadas las modificaciones específicas y no tardarán en implementarse). Por eso, esta aplicación podría convertirse en algo muy importante dentro de la Universidad Politécnica de

Madrid para mejorar la sostenibilidad, instruyendo y animando al alumnado a participar en este ámbito tan importante.

6.4 Impacto empresarial

En el ámbito empresarial no se espera un gran impacto, aunque, hablando de la UPM como una empresa, se podría considerar que puede mejorar esta compañía en el aspecto de la sostenibilidad y transición energética.

6.5 Objetivos de desarrollo sostenible

Todos estos tipos de impacto tienen cierta relación con los objetivos de desarrollo sostenible de la Agenda 2030 [36]. Concretamente los objetivos con los que se vincula este proyecto son el "**Garantizar una educación inclusiva, equitativa y de calidad y promover oportunidades de aprendizaje durante toda la vida para todos**" (objetivo número 4, educación de calidad), "**Lograr que las ciudades sean más inclusivas, seguras, resilientes y sostenibles**" (Objetivo número 11, Ciudades y comunidades sostenibles), "**Adoptar medidas urgentes para combatir el cambio climático y sus efectos**" (objetivo número 13, Acción por el clima) y "**Gestionar sosteniblemente los bosques, luchar contra la desertificación, detener e invertir la degradación de las tierras, detener la pérdida de biodiversidad**" (objetivo número 15, Vida de ecosistemas terrestres). Con esta aplicación se podrá enseñar cientos de términos y definiciones nuevas sobre cada uno de estos ámbitos dentro de la sostenibilidad.

7. Bibliografía

[1] R. A. Española, «Diccionario de la lengua española,» 2022. [En línea]. Available: https://dle.rae.es/. [Último acceso: Abril 2022].

[2] FECYT, «Ciencia Ciudadana,» [En línea]. Available: https://ciencia-ciudadana.es/sobre-el-observatorio/. [Último acceso: Abril 2022].

[3] S. P. W. y. S. Deterding, The Gameful World, The MIT Press, 2014.

[4] «Ludificación,» Wikipedia, 15 Febrero 2022. [En línea]. Available: https://es.wikipedia.org/wiki/Ludificaci%C3%B3n. [Último acceso: Abril 2022].

[5] J. Piaget, Structuralism, Basic Books, 1971.

[6] T. W. Malone, «What makes things fun to learn? A study of intrinsically motivating computer games,» Xerox PARC, Palo Alto, 1980.

[7] T. W. y. L. M. R. Malone, «Making learning fun: A taxonomy of intrinsic motivations for learning,» 1987.

[8] H. y. B. C. Klar, «Successful leadership in high-needs schools: An examination of core leadership practices enacted in challenging contexts,» *Educational Administration Quarterly,* vol. 49, nº 5, 2013.

[9] B. J. Fogg, «Fogg Behavior Model,» [En línea]. Available: https://behaviormodel.org/. [Último acceso: Abril 2022].

[10] M. Csikszentmihalyi, Beyond boredom and anxiety, San Francisco: Jossey-Bass, 1975.

[11] E. Herranz Sánchez y R. Colomo-Palacios, «La Gamificación como agente de cambio en la Ingeniería del Software,» *Revista de procesos y métricas de las tecnologías de la información,* vol. 9, nº 2, pp. 30-56, 2012.

[12] T. Grennan, «Braze,» 19 Mayo 2016. [En línea]. Available: https://www.braze.com/resources/articles/app-customer-retention-spring-2016-report.

[13] V. Estorach Cavaller, «Vanessa Estorach,» [En línea]. Available: https://www.vanessaestorach.com/vanessa-estorach-cavaller/. [Último acceso: Abril 2022].

[14] V. E. Cavaller, «Por qué implementar gamificación a tu app,» [En línea]. Available: https://www.vanessaestorach.com/implementar-gamificacion-app. [Último acceso: Abril 2022].

[15] Kahoot, «Kahoot!,» [En línea]. Available: https://kahoot.it/.

[16] «Nýyrðavefur (Web de Nuevas Palabras),» Stofnun Árni Magnússonar í íslenskum fræði, [En línea]. Available: https://nyyrdi.arnastofnun.is/. [Último acceso: Abril 2022].

[17] U. d. València, «Milmots,» 2015. [En línea]. Available: https://milmots.eu. [Último acceso: Abril 2022].

[18] Vue, «Vue.js,» [En línea]. Available: https://vuejs.org/. [Último acceso: Abril 2022].

[19] «JavaScript,» Mozilla, [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. [Último acceso: Abril 2022].

[20] «w3: HTML & CSS,» W3C, [En línea]. Available: https://www.w3.org/standards/webdesign/htmlcss. [Último acceso: Abril 2022].

[21] @. e. al., «GitHub: Axios,» GitHub, [En línea]. Available: https://github.com/axios/axios. [Último acceso: Abril 2022].

[22] «Flask Documentation (2.1.x),» Pallets, [En línea]. Available: https://flask.palletsprojects.com/en/2.1.x/. [Último acceso: Abril 2022].

[23] «Welcome to Python.org,» Python Software Foundation, [En línea]. Available: https://www.python.org/. [Último acceso: Abril 2022].

[24] «SQLite Home Page,» SQLite Consortium, [En línea]. Available: https://www.sqlite.org/index.html. [Último acceso: Abril 2022].

[25] «SQLAlchemy - The Databse Toolkit for Python,» Michael Bayer, [En línea]. Available: https://www.sqlalchemy.org/. [Último acceso: Abril 2022].

[26] «Werkzeug - Werkzeug Documentation (2.1.x),» Pallets, [En línea]. Available: https://werkzeug.palletsprojects.com/en/2.1.x/. [Último acceso: Abril 2022].

[27] «ItsDangerous - ItsDangerous Documentation (2.1.x),» Pallets, [En línea]. Available: https://itsdangerous.palletsprojects.com/en/2.1.x/. [Último acceso: Abril 2022].

[28] «DB Browser for SQLite,» DigitalOcean, [En línea]. Available: https://sqlitebrowser.org/. [Último acceso: Abril 2022].

[29] «Visual Studio Code - Code editing. Redifined,» Microsoft, [En línea]. Available: https://code.visualstudio.com/. [Último acceso: Abril 2022].

[30] «Git,» Software Freedom Conservancy, [En línea]. Available: https://git-scm.com/. [Último acceso: Abril 2022].

[31] «GitHub,» Microsoft, [En línea]. Available: https://github.com/. [Último acceso: Abril 2022].

[32] R. M. R., «unLokillo/TFG_App,» GitHub, 2022. [En línea]. Available: https://github.com/unLokillo/TFG_App. [Último acceso: Abril 2022].

[33] «SourceTree | Free Git GUI for Mac and Windows,» Atlassian, [En línea]. Available: https://www.sourcetreeapp.com/. [Último acceso: Abril 2022].

[34] «LucidChart,» Lucid Software Inc., [En línea]. Available: https://www.lucidchart.com/. [Último acceso: Mayo 2022].

[35] «Draw.io,» Atlassian, [En línea]. Available: https://drawio-app.com/. [Último acceso: Mayo 2022].

[36] N. Unidas, «Objetivos de desarrollo sostenible,» [En línea]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/. [Último acceso: Mayo 2022].

8. Anexos

8.1 Repositorio de código

El código generado en el desarrollo de la aplicación web y descrito en el apartado de desarrollo del Back-End se encuentra disponible a todo el público en un repositorio de GitHub, en la rama ***/Roberto*** del repositorio https://github.com/unLokillo/TFG_App [32]. En esta rama se pueden observar todos los commits realizados desde el inicio del proyecto, es decir, qué cambios o desarrollos se hicieron a qué ficheros, y qué día y hora específicos.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Thu Jun 02 20:15:03 CEST 2022
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)