Tecnologías y herramientas: Wordle+.

Alumno: David Correa Rodríguez
Tutor: Juan Manuel Fernández Luna

Diseño e implementación del proyecto

Frontend

Una vez planteadas las HUs, su división en tareas, y la planificación de los Sprints, es necesario realizar un análisis de las tecnologías y herramientas candidatas para realizar este proyecto.

Antes de profundizar más en la cuestión, me gustaría aclarar que uno de los grandes objetivos de este proyecto es el aprendizaje y la adquisición de nuevo conocimiento, al igual que todo mi transcurso por la carrera, por lo que le daré mayor peso a aquellas herramientas con las que no tengo experiencia y/o demasiado conocimiento. A priori, sería más sencillo utilizar herramientas ya conocidas, pero estoy más interesado en ampliar mi perfil académico e invertir cierto tiempo en aprender ámbitos nuevos.

Como ya se ha mencionado varias veces, Wordle+ consiste en realizar un proyecto *fullstack,* es decir, tiene componentes tanto *backend* como *frontend*. Además, uno de los principales objetivos del proyecto es conseguir una aplicación web multiplataforma, por lo que tecnologías dedicadas exclusivamente para desarrollo de aplicaciones para móviles (como Android Studio, Flutter, etc) quedarán descartadas. Con esto, el dominio queda reducido a tecnologías web, por lo que las alternativas principales son:

- **Pila LAMP**: consiste en una infraestructura que utiliza Linux (Linux s.f.), Apache (Apache s.f.), MySQL/MariaDB (MySQL s.f.) y PHP (PHP s.f.), también utilizando JavaScript como lenguaje de *scripting*. Es una infraestructura ya ampliamente utilizada en otros proyectos de la carrera, por lo que quedan descartados. Además, puede resultar complicado diseñar completamente el proyecto de forma adaptativa o *responsive.* (IBM s.f.)

- **Pila MERN/MEAN:** similar a la pila LAMP, pero o se utiliza MongoDB (MongoDB s.f.) como gestor de base de datos; Express (Express s.f.) como gestor de peticiones HTTP; React o Angular (React s.f.) como herramienta principal para el *frontend;* y Node.js (NodeJS s.f.) como plataforma de *backend*. El principal lenguaje a utilizar es JavaScript (JavaScript s.f.). Esta alternativa resulta interesante y se presenta como una renovación de la pila LAMP. Sin embargo, personalmente ya he utilizado esta infraestructura (MERN) en un proyecto de la asignatura "Dirección y Gestión de Proyectos". En enlace al repositorio público es: https://github.com/davidcr01/Class4All. Además, presenta el mismo problema del diseño *responsive* que presentaba la pila LAMP.

Con lo anterior expuesto, se podría cambiar el foco a frameworks que permitan el desarrollo de aplicaciones multiplataforma o híbridas. A pesar de que hay una extensa lista de alternativas, se comentarán las principales de ellas:

- **Flutter**: si bien es cierto que ha sido mencionado anteriormente, Flutter es uno de los sistemas más utilizados para el desarrollo de este tipo de aplicaciones. Sin embargo, se suele utilizar en mayor medida para el desarrollo de aplicaciones móviles, además de utilizar el lenguaje de programación Dart (Dart s.f.), por lo que se aleja del uso de tecnologías web, que es uno de los objetivos de este proyecto. (Flutter s.f.)

- **React Native:** se presenta como otra alternativa interesante, en donde el lenguaje principal es JavaScript, con posibilidad de escribir módulos en Objective-C (Wikipedia s.f.), Swift (Apple s.f.) o Java (Oracle s.f.).Sin embargo, como ya he comentado anteriormente, he realizado un proyecto utilizando React, por lo que el aprendizaje que me aportaría esta alternativa se vería reducido. (Native s.f.)

- **Ionic Framework:** al igual que las demás, permite crear aplicaciones híbridas, pero en este caso utilizando HTML, CSS y JavaScript, con otra notación distinta. Este framework permite un desarrollo ágil para diseñar aplicaciones híbridas, pudiendo utilizar elementos web ya construidos. Cabe destacar que es uno de los framework más utilizado actualmente, además de tener una gran comunidad y documentación. (Ionic s.f.)

Finalmente he escogido de **Ionic Framework**, por lo comentado anteriormente. Sigue la línea de las tecnologías web, que es uno de los objetivos de este proyecto, permitiendo desarrollar Wordle+ tanto en web como en móvil de forma **simultánea.** Además, esta alternativa permite utilizar tres de los grandes lenguajes de programación *frontend* actual a elegir, es decir, solamente se puede utilizar una de ellas: React, Angular (Angular s.f.) y Vue (Vue s.f.). En mi caso, descarto la primera por haberla utilizado en otros proyectos, y respecto a

las dos restantes, ambas son similares, pero **Angular** tiene una comunidad y tasa de mercado mucho más significativa.

Por tanto, y a modo de resumen, se utilizará el framework Ionic utilizando el lenguaje de programación Angular.

Backend

El groso de las tecnologías comentadas anteriormente son herramientas de desarrollo de *frontend*. Respecto al *backend*, podemos categorizar las alternativas siguiendo diferentes criterios:

- Por **cómo** se gestiona la información: las principales alternativas en este campo son modelos **relacionales** y modelos **no relacionales**. Personalmente he utilizado ambos exhaustivamente, con herramientas como MySQL y MongoDB.
  Sin embargo, tengo cierto interés en utilizar **PostgreSQL** (PostgreSQL s.f.), uno de los sistemas de gestión de bases de datos relacional más utilizado actualmente. En este caso, tengo preferencia en utilizar modelos relacionales, ya que gran parte de la información que va a alojar la base de datos tiene bastante conexión mediante claves externas, y tendría poco sentido utilizar modelos no relacionales.

- Por **dónde** se ubica la información: en este caso, las principales alternativas son utilizar servicios en la **nube** o alojar la base de datos de forma **local**. La nube es interesante ya que los datos están almacenados en internet, y hay poca probabilidad de pérdida. Sin embargo, se requiere de un proveedor externo, de un ancho de banda considerable para obtener la información, y evidentemente de conexión a internet. Por otro lado, la alternativa local no presenta las desventajas descritas anteriormente, pero la información solamente estaría alojada en el sistema local. Finalmente, he preferido escoger la alternativa **local,** aunque para paliar sus desventajas me gustaría realizar un despliegue del backend en un contenedor Docker (Docker s.f.), punto que se tratará más adelante.

Este proyecto presenta una arquitectura del tipo Modelo-Vista-Controlador. Ya se han mencionado los aspectos del “Modelo” y de la “Vista”. Respecto al controlador, se propone desarrollar una API REST, que permite la estandarización de intercambio de datos entre los servicios web, abstrayéndose del lenguaje de programación y de los sistemas operativos.

Utilizar una API REST tiene las siguientes ventajas:

- Escalabilidad: se optimizan las interacciones entre el cliente y el servidor.
- Flexibilidad: existe un desacoplo entre los componentes de manera que éstos pueden evolucionar de forma independiente
- Independencia: no importa la tecnología utilizada, no afecta al diseño de la API.

En cuanto a tecnologías, existen dos alternativas ampliamente conocidas:

- **Node.js**: junto con la herramienta Express, es uno de los métodos más utilizados para crear APIs. Sin embargo, esta alternativa ya ha sido utilizada en el proyecto anteriormente mencionado.
- **Python** (Python s.f.): uno de los lenguajes multiparadigma más utilizados en el panorama actual, permite construir APIs de forma sencilla mediante el framework **Django REST framework** (Framework s.f.). Esta opción me parece interesante para incorporar al proyecto una potente tecnología como es Python, y hacer uso de otro de los frameworks más utilizados actualmente, como es el citado anteriormente.

Abstracción del sistema

Actualmente, la tecnología está evolucionando continuamente y a una gran velocidad. Ya se han mencionado anteriormente aspectos como la integración y distribución continuas (CICD) que aportan nuevo software rápidamente, con actualizaciones muy frecuentes.

Ante este panorama, no es recomendable hacer un despliegue en un sistema en concreto, utilizando un sistema operativo y un hardware específico. Esta metodología haría el proyecto muy rígido, permitiendo no poder migrarlo en caso de que fuera necesario. Ante esto, surgen varios proyectos como **Docker** que permiten realizar un despliegue de aplicaciones dentro de contenedores software. Esto, a su vez, abstrae y automatiza la virtualización de aplicaciones en distintos sistemas operativos. (Docker s.f.)

Por ello, lo ideal sería hacer el despliegue de todo el proyecto utilizando la herramienta *docker-compose* que permite orquestar varios contenedores y realizar una comunicación interna entre ellos. Si esto no es posible, se pretende como mínimo desplegar el *backend* en contenedores, para así

abstraer toda la información e instalación de esta parte del proyecto del sistema sobre el que se construya.

Control de versiones y seguimiento del Sprint

GitHub es sin duda alguna el sitio web en donde más desarrolladores alojan sus proyectos software. Gracias a su integración con la herramienta *Git* para el control de cambios, y sus características como las *Issues, Pull Request, Milestones*, la convierte en una herramienta idónea para este tipo de proyectos. Por tanto, el proyecto se alojará en un repositorio de GitHub, haciendo uso de Git para el control de cambios.

Respecto al seguimiento de los Sprints, existen varias conocidas alternativas como Trello o Jira, en donde plasmar todo el Product Backlog, planificación de los Sprints y su seguimiento. Sin embargo, esta tarea se puede también realizar el GitHub de formas distintas.

El sistema propuesto es crear varios hitos o *milestones* en GitHub, que representen las funcionalidades del proyecto, como por ejemplo “registro de usuarios”, “login de usuarios”, “Wordle clásico”, y a dichos hitos asociarles las respectivas Issues y Pull Request. De esta manera, se permite un seguimiento mucho más óptimo y una documentación más extensa del progreso del proyecto, teniendo en un mismo lugar los cambios realizados en el proyecto y su seguimiento asociado. Esta información ha sido extraída de: https://docs.github.com/es/issues/using-labels-and-milestones-to-track-work/about-milestones

Bibliografía

Angular. s.f. https://angular.io/.

Apache. s.f. https://httpd.apache.org/.

Apple. s.f. https://www.apple.com/es/swift/.

Dart. s.f. https://dart.dev/.

Docker. *https://www.docker.com/.* s.f.

Express. s.f. https://expressjs.com/.

Flutter. *https://flutter.dev/.* s.f.

Framework, Django Rest. s.f. https://www.django-rest-framework.org/.

IBM. *www.ibm.com.* s.f. https://www.ibm.com/es-es/cloud/learn/lamp-stack-explained.

Ionic. *https://ionicframework.com/.* s.f.

JavaScript. s.f. https://www.javascript.com/.

Linux. s.f. https://www.linux.org/.

MongoDB. s.f. https://www.mongodb.com/mern-stack#:~:text=MERN%20stands%20for%20MongoDB%2C%20Express,MongoDB%20%E2%80%94%20document%20database.

MySQL. s.f. https://www.mysql.com/.

Native, React. *https://reactnative.dev/.* s.f.

NodeJS. s.f. https://nodejs.org/en/.

Oracle. s.f. https://www.java.com/es/.

PHP. s.f. https://www.php.net/.

PostgreSQL. s.f. https://www.postgresql.org/.

Python. s.f. https://www.python.org/.

React. s.f. https://es.reactjs.org/.

Vue. s.f. https://vuejs.org/.

Wikipedia. s.f. https://es.wikipedia.org/wiki/Objective-C.