Aplicación web de desafíos de programación

Introducción

Este trabajo de fin de grado (en adelante TFG) trata del desarrollo de una aplicación web de desafíos de programación en Javascript. Los retos consistirán en pruebas que puedan validarse de forma automática, como preguntas tipo test o el desarrollo de un código fuente que cumpla una serie de aserciones predefinidas. Agrupando varios desafíos en módulos podrán crearse diferentes tutoriales con fin educativo.

Objetivo principal

La idea para este TFG surge de conversaciones con compañeros de trabajo, sobre cómo ayudar a los recién llegados a mejorar y afianzar sus conocimientos de JavaScript (en adelante JS).

La popularidad alcanzada en los últimos años y su omnipresencia en el desarrollo web hace que muchos profesionales tengan que hacer frente a tareas relacionadas con JS. Su principal ventaja, la versatilidad para hacer cualquier cosa, es a la vez su mayor debilidad (Gómez, 2019). Si le sumamos una relativamente rápida curva de aprendizaje para realizar ciertas funcionalidades, tenemos un caldo de cultivo perfecto para dar por sentado muchos conocimientos y caer en malas prácticas al afrontar tareas más complicadas.

De ahí que el principal objetivo sea el disponer de una herramienta práctica y sencilla con la que poder reafirmar conocimientos sin abandonar a los desarrolladores frente a una lista de libros o enlaces a sitios web.

Subobjetivos

- Desarrollar el TFG siguiendo una metodología TDD[1] y, como comenta Martin Fowler, esperar alcanzar una cobertura de al menos el 80% (Fowler, 2012). La motivación para ello reside en conocer empresas donde las pruebas automáticas se consideran un coste adicional y se dejan fuera del flujo de desarrollo. De hecho la

1 *Test-driven development*

tónica habitual es que los planes de pruebas manuales se realicen al final de la entrega de un proyecto y, durante el mantenimiento o en nuevas iteraciones, se pruebe sólo la parte afectada, provocando en ocasiones errores debidos a efectos colaterales. En este proyecto quiero comprobar si el coste adicional de TDD se compensa con calidad y el ahorro en la solución de errores.

- Montar un entorno de desarrollo que disponga de las herramientas necesarias para una implementación ágil del proyecto: linters, testing automatizado, servidor de desarrollo con live-reload…
- Conseguir una experiencia de usuario (en adelante UX[1]) satisfactoria, para lo cual, además de un diseño de interacción intuitivo, será importante un rendimiento web aceptable, tanto en tests tipo Lighthouse[2] como en rendimiento percibido.
- Conseguir que la UX del editor de código online sea lo más parecida posible a uno de escritorio. Es importante cuidar la accesibilidad para que la aplicación web sea usable mediante teclado, ya que el usuario objetivo (desarrolladores) está acostumbrado a manejarse únicamente con teclado.
- La idea inicial era que el sitio web fuese estático, pero con el objetivo de poner a prueba los conocimientos adquiridos a lo largo del Grado, se implementará un gestor web para crear los diferentes retos y colecciones.
- Preparar una imagen de docker con el backend necesario (Linux, Nginx, Node.js, BBDD). De esta manera podremos tener un entorno de trabajo limpio y el producto final será posible probarlo en cualquier máquina o desplegarlo a un proveedor como DigitalOcean.

Tecnologías que se van a utilizar

Cliente:

- Vue.js
- ES2019
- CSS moderno, incluyendo,
  - *Candidate Recommendations*
  - *Working Drafts*
  - y posiblemente algún *Editor Draft* que evite utilizar Sass.
- Axios, como cliente http.

Servidor:

- Node.js

---

1 *User experience*

2 *Lighthouse scoring guide <https://developers.google.com/web/tools/lighthouse/v3/scoring#perf-color-coding>.*

- Express o Fastify, como servidor de aplicaciones.
- Nginx, como servidor web. Aunque pueda pensarse que un servidor Node.js es *production-ready*, instalar por delante un *reverse proxy* proporciona ciertas ventajas. En nuestro caso nos interesa por mejoras en el rendimiento para servir estáticos, encriptación SSL, compresión... (Hunter, 2019)
- BBDD:
  - Postgres, en caso de decidirnos por un modelo relacional, o,
  - MongoDB o Couchbase, si nos decantamos por una documental.

Stack tecnológico para desarrollo:

- VSCodium[1], como IDE,
- Git, como control de versiones,
- Webpack o Parcel, como bundler.
- Babel, como transpilador[2] de código ES2019 al adecuado para los navegadores soportados.
- Postcss, para transpilar el código CSS.

Los navegadores soportados serán aquellos que dan soporte casi completo a ES2015[3], de forma que su motor javascript nos permita ofrecer algunos ejercicios en dicho lenguaje sin tener que hacer transpilación en tiempo de ejecución. En formato browserlist el soporte se resumiría en:

```
last 2 Chrome major versions
last 2 Firefox major versions
last 2 Safari major versions
last 2 Edge major versions
last 2 ChromeAndroid major versions
last 2 iOS major versions
```

Planificación temporal

Como bien se pregunta Jeffries sobre las estimaciones, ¿es prever mejor que dirigir? (Jeffries, 2015). Seguramente él opinaría que en un proyecto de este calado lo ideal sería desarrollar siguiendo una metodología ágil e ir sumando valor al producto en cada sprint. Siguiendo un desarrollo *feature-by-feature* el producto podría evolucionar e ir adaptándose

1 *Proyecto libre, basado en Microsoft VSCode, cuyo binario es publicado bajo licencia MIT y está libre de la telemetría y tracking de Microsoft. <https://vscodium.com/>*

2 *Un transpilador de código es un tipo especial de compilador que traduce un código fuente a otro de un nivel similar de abstracción. (Wik01)*

3 *ES6 compatibility table <https://kangax.github.io/compat-table/es6>*

en base a la experiencia adquirida durante el desarrollo y la satisfacción de los usuarios finales.

De todas formas, dado que este proyecto se realizará bajo un marco académico en el que hay designados cuatro hitos de entrega, se utilizará un modelo en cascada. Así, el proyecto se dividirá en cuatro etapas:

1. Plan de trabajo
2. Análisis y diseño
3. Desarrollo
4. Documentación

Tarea	Fase	Fecha inicio	Fecha fin	Dias	Horas	Descripción
PEC1		20/feb	06/mar		39h	Plan de trabajo
PT1	Plan de trabajo	20/feb	24/feb	5	15h	Elección del proyecto y objetivos
PT2	Plan de trabajo	25/feb	25/feb	1	3h	Preparación de plantilla de documento
PT3	Plan de trabajo	25/feb	26/feb	2	7h	Redacción de documento y elección de tecnologías
PT4	Plan de trabajo	28/feb	29/feb	2	6h	Captura inicial de requisitos (historias de usuario)
PT5	Plan de trabajo	01/mar	02/mar	2	8h	Planificación y riesgos
PEC2		09/mar	10/abr		96h	Primer hito
ANA1	Análisis	09/mar	09/mar	1	1h	Describir stakeholders identificados
ANA2	Análisis	09/mar	10/mar	2	7h	Casos de uso UML a partir de historias
ANA3	Análisis	15/mar	16/mar	2	8h	Diagrama de casos de uso UML y mapa navegacional
ANA4	Análisis	17/mar	18/mar	2	6h	Modelo de dominio
DIS1	Diseño	19/mar	24/mar	6	24h	Diseño de la arquitectura de la aplicación
DIS2	Diseño	28/mar	30/mar	3	12h	Wireframes
DIS3	Diseño	31/mar	01/abr	2	8h	Selección final de tecnologías en base al análisis y tras estudiar alternativas: BBDD, ORM o ODM, bundler, servidor de aplicaciones...
DEV1	Desarrollo	02/abr	07/abr	6	30h	Documentarse y preparar entorno de desarrollo: linters (estilo de código, accesibilidad...), test runner, test automatico lighthouse, transpilacion de es2019 y >css3, bundler, servidor de desarrollo con hot reloading...
PEC3		13/abr	29/may		153h	Segundo hito
DEV2	Desarrollo	13/abr	20/abr	8	32h	Documentarse y preparar docker para backend local
DEV3	Desarrollo	21/abr	21/abr	1	3h	Preparar datos de ejemplo para la base de datos y mocks
DEV4	Desarrollo	22/abr	26/abr	5	20h	API backend
DEV4	Desarrollo	29/abr	04/may	6	24h	Editor online (editor, consola)
DEV5	Desarrollo	05/may	07/may	3	12h	Ejecutar código fuente de usuario y comprobar aserciones
DEV6	Desarrollo	08/may	10/may	3	12h	Gestión de desafíos tipo test
DEV7	Desarrollo	11/may	13/may	3	12h	Gestión de módulos
DEV8	Desarrollo	14/may	20/may	7	30h	Parte pública
DEV9	Desarrollo	21/may	22/may	2	8h	Registro y login
DEV10	Desarrollo	23/may	23/may	1	4h	Gestión de preguntas tipo test
DEV11	Desarrollo	24/may	25/may	2	8h	Plan de pruebas
PEC4		01/jun	12/jun		36h	Memoria final
MEM1	Documentación	01/jun	08/jun	8	32h	Redacción de memoria y correcciones
MEM2	Documentación	09/jun	09/jun	1	4h	Presentación
TFG					324h	

Tabla 1: Planificación

De la anterior tabla podemos extraer el siguiente diagrama de Gantt:

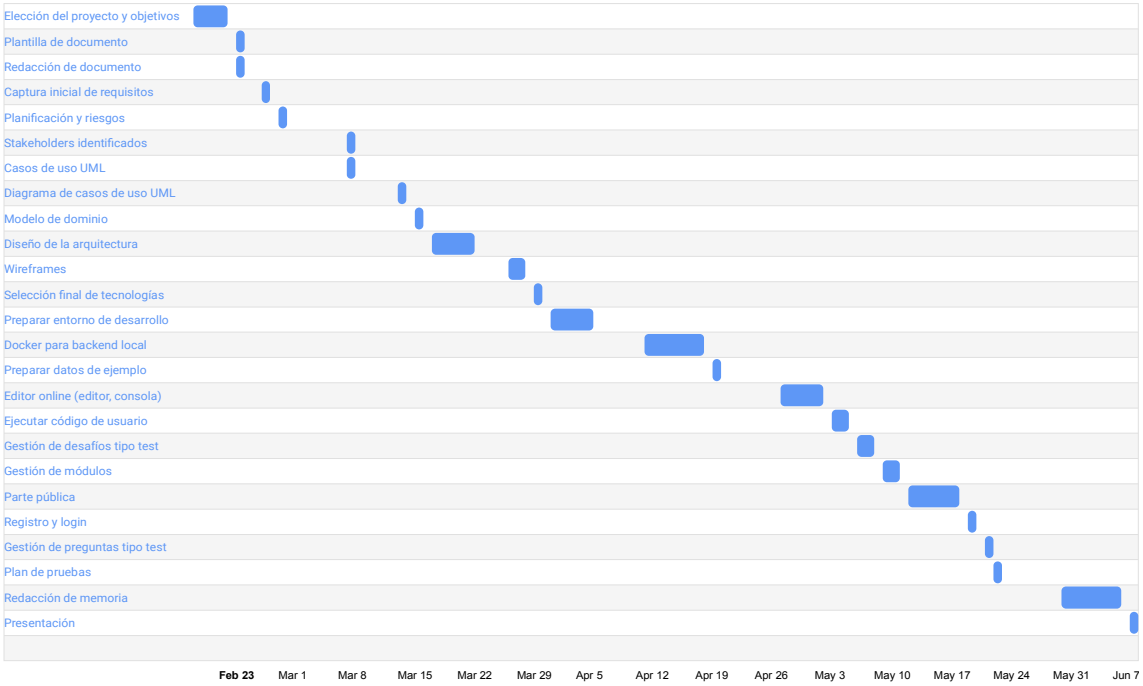

Figura 1: Diagrama de Gantt

Evaluación de riesgos

Se han identificado diversos riesgos que pueden afectar al proyecto. Entre los generales y con una probabilidad media alta se hayan los derivados de cargas familiares y carga laboral.

Riesgo	Impacto	Probabilidad	Medidas
Cargas familiares	Alto	Alta	Evitarlo adelantando trabajo los fines de semana o festivos de la madre. Mitigarlo eliminando funcionalidades, requisitos de menor valor o revisando alcance de tareas.
Cargas laborales	Medio	Alta	Mitigarlo eliminando funcionalidades, requisitos de menor valor o revisando alcance de tareas.
Mala planificación (riesgo negativo)	Alto	Medio	Mitigarlo eliminando funcionalidades, requisitos de menor valor o revisando alcance de tareas.
Mala planificación (riesgo positivo)	Medio	Medio	Explotarlo mejorando la calidad o añadiendo nuevos requisitos.
Requisitos excesivos	Medio	Bajo	Definir y reevaluar con precisión requisitos que hayan podido quedar poco detallados o con un alto grado de exigencia y cuyo impacto no sea crítico.
Problemas técnicos	Medio/Bajo	Medio	Documentación, búsqueda alternativas tecnológicas o redefinición del requisito dependiendo de su importancia.

Bibliografía

Colaboradores de Wikipedia. *Transpilador*. Wikipedia, La enciclopedia libre, 2019 [fecha de consulta: 27 de febrero del 2020]. Disponible en <https://es.wikipedia.org/w/index.php?title=Transpilador&oldid=121585326>

Fowler, Martin. *TestCoverage*. martinFowler.com, 2012 [fecha de consulta: 26 de febrero de 2020]. Disponible en <https://martinfowler.com/bliki/TestCoverage.html>

Gómez, Miguel A.. *Clean Code, SOLID y Testing aplicado a JavaScript*. 1ª edición. Software Crafters, 2019.

Hunter, Thomas. *Why should I use a Reverse Proxy if Node.js is Production-Ready?*. Medium: Intrinsic, 2019 [fecha de consulta: 27 de febrero de 2020]. Disponible en <https://medium.com/intrinsic/why-should-i-use-a-reverse-proxy-if-node-js-is-production-ready-5a079408b2ca>

Jeffries, Ron. *The Nature of Software Development: Keep It Simple, Make It Valuable, Build It Piece by Piece*. 1ª edición. The Pragmatic Bookshelf, 2015.