Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

ECodium: Plataforma gamificada para fomentar la creatividad entre programadores con recompensas basadas en la red Ethereum

---

Alumno: Samuel Encinas Plaza

Tutor/a/es: José Vicente Álvarez Bravo

ECodium: Plataforma gamificada para fomentar la creatividad entre programadores con recompensas basadas en la red Ethereum

Samuel Encinas Plaza

Índice general

Índice de figuras

Índice de cuadros

"La creatividad no se gasta. Cuánta más usas, más tienes"
- Maya Angelou

Agradecimientos

En primer lugar, gracias a mi tutor, D. José Vicente Álvarez Bravo, por apoyarme en mi idea de llevar a cabo este proyecto, y brindarme opiniones y consejos durante el desarrollo del mismo.

Gracias también al resto de profesorado que componen la Escuela de Ingeniería Informática de Segovia, por todo lo aprendido durante estos años que, habrá contenidos que me resulten más o menos interesantes, pero todos contribuyen a mi formación.

Finalmente agradecer a mi familia, amigos, y pareja por el ánimo y el apoyo recibidos a lo largo del desarrollo de este trabajo, así como los buenos momentos compartidos para que pudiera descansar mínimamente cuando el estrés se comenzaba a apoderar de mí.

Resumen

La creatividad es una de las mayores herramientas para dominar la complejidad, y por ello juega un papel cada vez más esencial en el desarrollo de software, un campo que, por definición se encuentra en constante evolución. La resolución innovadora y creativa de problemas complejos debería ser una de las mayores metas de todo aquel, profesional o estudiante, que busque su lugar en el mundo de la informática. Por ello, cada vez son más las entidades, tanto a nivel laboral como educativo, que buscan talento creativo o proponen nuevos enfoques a la hora de formar o contar con personas en el ámbito de las nuevas tecnologías.

ECodium se propone como plataforma gamificada para fomentar esa creatividad entre profesionales, estudiantes y aficionados a la informática, que busquen aprender conceptos nuevos, profundizar en los conocimientos que ya conocen o simplemente reinventarse a sí mismos. Además, a modo de alicientes, por un lado, permitiría a las empresas encontrar talento mediante la existencia de ojeadores dentro del marco de la aplicación y, por otro, resultaría atractivo para los jugadores al tener la posibilidad de generar *tokens* de la red Ethereum.

Este Trabajo Final de Grado busca documentar el Proceso de Desarrollo Software completo, desde la concepción hasta el despliegue, de esta aplicación web, construida con tecnologías punteras bajo un enfoque *full stack* y utilizando metodologías ágiles, todo ello desplegado en un entorno *cloud* y apoyándose en una base de datos no relacional.

Palabras clave: ECodium, gamificacion, Ethereum, creatividad, full stack, proceso de desarrollo software

Abstract

Creativity is one of the greatest tools for mastering complexity, and therefore plays an increasingly essential role in software development, a field that, by definition, is constantly evolving. Innovative and creative solving of complex problems should be one of the main goals of every professional or student who is looking for his place in the IT world. For this reason, there are more and more organisations, both in the workplace and in education, that are looking for creative talent or proposing new approaches when it comes to training or employing people in the scope of new technologies.

ECodium is proposed as a gamified platform to encourage this creativity among professionals, students and computer enthusiasts, who are looking to learn new concepts, deepen the knowledge they already know or simply reinvent themselves. Furthermore, as an incentive, on the one hand, would allow companies to find talent through the existence of scouts within the framework of the application and, on the other hand, it would be attractive for players to have the possibility of generating Ethereum network tokens.

This Final Degree Project seeks to document the complete Software Development Process, from conception to deployment, of this web application, built with cutting-edge technologies under a full stack approach and using agile methodologies, all deployed in a cloud environment and supported by a non-relational database.

Keywords: ECodium, gamification, Ethereum, creativity, full stack, software development

Capítulo 1

Introducción

1.1. Estructura del documento

Este presente documento se organiza en nueve **capítulos** diferenciados entre sí, de forma que cada uno de ellos agrupe contenidos:

- El primer capítulo, **Introducción**, en el que se encuentra este epígrafe, detalla aspectos iniciales como la motivación, el contexto y los objetivos sobre los que parte el proyecto.
- En el segundo capítulo, **Estudio del alcance y viabilidad**, se detalla el estudio que se hizo para determinar si el proyecto sería viable.
- El **Fundamento Teórico del Proyecto** constituye el tercer capítulo del documento, y asienta las bases teóricas en las que se apoya el proyecto, tanto teoría acerca de la *gamificación* como teoría técnica (sobre bases de datos no relacionales, *blockchain* o el *stack MEVN* - sobre el cual se construye la infraestructura del proyecto).
- En el cuarto capítulo, **Planificación y Gestión del Proyecto**, se detalla una de las partes sustanciales del Proceso de Desarrollo de Proyectos *Software*, y es la planificación, organización temporal y gestión del proyecto.
- El quinto capítulo corresponde con el **Análisis Funcional** del proyecto, y en él se explican los roles del sistema, así como la especificación de requisitos mediante Historias de Usuario.
- Posteriormente, en el sexto capítulo, se describe el **Modelado de Datos** a nivel conceptual y lógico del proyecto.
- En el séptimo capítulo, se detalla el **Diseño y Arquitectura** de la aplicación. El capítulo comienza con la exposición del Diseño de Experiencia de Usuario del proyecto y continúa con la Arquitectura del mismo, a distintos niveles.

- El octavo capítulo describe dos aspectos importantes: **Implementación y Despliegue** del proyecto. El enfoque que recibe este capítulo se centra en explicar qué cosas se han hecho, cómo y por qué.
- Por último, el documento finaliza con un noveno capítulo de **Conclusiones** y líneas de trabajo futuras, así como con conclusiones personales.
- En cuanto a **apéndices**, el documento incluye un único apéndice: el **Apéndice A: Manual del Usuario**, donde se explica y se detalla el funcionamiento de la aplicación a distintos roles

1.2. Motivación

De acuerdo con el estudio *The Dynamics of Creativity in Software Development* [6], la creatividad es una de las mayores herramientas para dominar sistemas complejos, ya que permite aportar soluciones innovadoras a los problemas que suelen acompañar a dichos sistemas complejos, siendo uno de los mayores ejemplos de sistemas complejos el campo del desarrollo de *software*.

Especialmente en el campo del desarrollo de *software*, la creatividad juega un papel crucial, ya que, por definición, se trata de un campo en constante evolución. Además, conceptos como la transformación digital se encuentran a la orden del día en la mayoría de empresas actuales, justificando aún más la necesidad de lograr innovar en el campo digital. Para ello, resulta muy útil la existencia de equipos creativos de desarrollo de *software* que permitan resolver problemas complejos de forma innovadora, pero manteniendo un alto rendimiento y nivel de calidad.

Con la mirada puesta en esta resolución innovadora de problemas complejos en el desarrollo de *software*, comienzan a surgir y a ponerse de moda eventos como el *hackathon*, que según *Digital Innovation: The Hackathon Phenomenon* [2], el *"fenómeno del hackathon"* se constituye como una aproximación efectiva a la innovación con tecnologías digitales de manera multidisciplinar, ya que habitualmente en los eventos de este tipo, se exige el uso de tecnologías digitales como medio para conseguir un fin, que poco o nada tiene que ver con la tecnología en sí, pero que gracias a ella, se puede lograr.

Así pues, un *hackathon* ya no solo permite hacer de la programación una auténtica competición, sino que además permite a sus participantes estimular *soft skills* realmente importantes, como la creatividad o la multidisciplinariedad. Y es precisamente por este tipo de cuestiones por las que eventos de este tipo comienzan a ganar popularidad entre los aficionados a la tecnología.

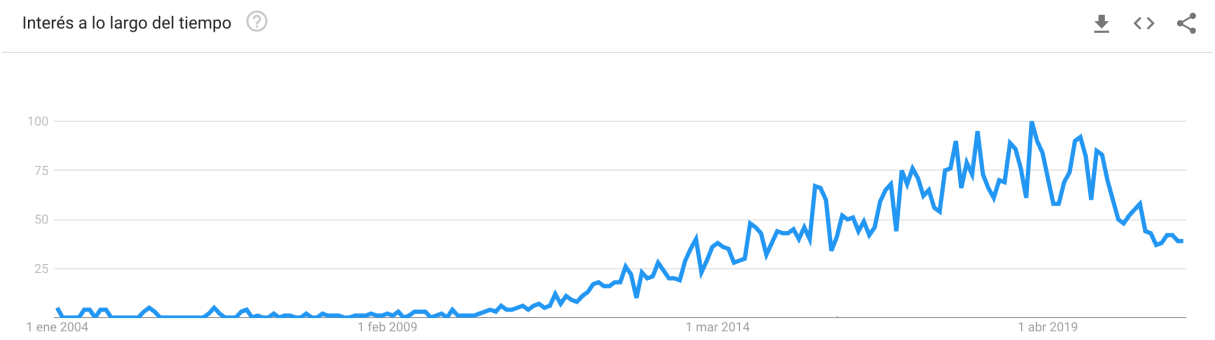


Figura 1.1: Evolución del interés en Google Trends de la palabra "Hackathon"

Tras la irrupción de la pandemia mundial por el COVID-19, y con la eliminación de eventos presenciales como una de sus mayores consecuencias, se comenzaron a poner de moda *hackathones online*, desarrollados de forma rápida, improvisada y con la propia pandemia por coronavirus como contexto y temática principal. Incluso organizaciones gubernamentales como la Comunidad de Madrid [1] o la propia Comisión Europea [2] se sumaron a la causa, ratificando aún más la idea de que, aún en condiciones excepcionales y de manera no presencial, **todo el mundo quiere *hackathon*.** Esta idea, de manera literal, sirvió como titular y premisa principal a la periodista Ana Torres Menárguez, que en su artículo [9], además de repasar la historia del *hackathon*, explica por qué las empresas necesitan innovación y, para ello, buscan talento incluso fuera de sus plantillas.

Por otra parte, el aprendizaje autodidacta es otro aspecto que puede resultar interesante para mejorar la creatividad y, a su vez, esta capacidad de resolución de problemas tan demandada por los programadores. Según un artículo del popular portal *Geeks for Geeks* [17], el aprendizaje autodidacta hace que se deje de confiar plenamente en lo ya aprendido, consiguiendo algo parecido a "entrenar la mente" para pensar de una manera lógica, analítica y programática. De este modo, la capacidad de resolución de problemas con rapidez se ve mejorada y optimizada.

Así pues, con el fin de mantener el ritmo de innovación que requieren los avances digitales de nuestros días, puede llegar a ser una necesidad fomentar estos factores tanto en la formación de nuevos desarrolladores como en la especialización de desarrolladores ya formados. Es por ello que para satisfacer esta nueva necesidad en creciente auge se busca desarrollar una plataforma que, de manera entretenida, permita a los usuarios desarrollar *soft skills* como la creatividad o la capacidad de resolución de problemas, además de organizar eventos de las características de los ya comentados *hackathones* y permitirles desarrollar un aprendizaje autodidacta de manera cómoda mediante talleres y retos que se sustente en la compartición de conocimiento.

La fórmula de la plataforma a desarrollar se completa con un sistema gamificado, que

[1] `https://www.compromisoempresarial.com/coronavirus/2020/04/la-comunidad-de-madrid-elige-15-proyectos-del-hackathon-madridvencealvirus/`

[2] `https://www.euvsvirus.org`

como se explicará más adelante, permite convertir el aprendizaje en un auténtico juego, así como un sistema de recompensas basadas en un criptoactivo de la red *Ethereum*. Tecnologías como *Blockchain* y la propia red *Ethereum* hoy en día constituyen el reclamo perfecto para incentivar la aparición de nuevos usuarios a cualquier plataforma de estas características.

1.3. Conceptualización y objetivos

La premisa principal de la que se partió para desarrollar este Trabajo Fin de Grado es llevar a cabo algo que recorra, desde un punto de vista global, todos los conocimientos adquiridos durante el Grado, así como conocimientos adicionales que pueden resultar interesantes y potencien los conocimientos del Grado. Por esta premisa, este Trabajo Fin de Grado consiste tanto en llevar a cabo un proceso de desarrollo software completo, de principio a fin, como en documentarlo.

El objetivo principal de este Trabajo Fin de Grado, es, por tanto, la documentación del proceso de desarrollo **full stack** de una plataforma online que permita tanto el aprendizaje a distintos niveles como la programación competitiva, desde un punto de vista gamificado y con recompensas basadas en la red Ethereum. De ahora en adelante, dicha plataforma será referida como ***ECodium***.

Este objetivo principal se puede desglosar en múltiples objetivos esquematizados:

- Llevar a cabo un **proceso de desarrollo full stack** de la plataforma web, es decir, que combine desarrollo front-end con desarrollo-backend. El proceso de desarrollo deberá ser completo, comenzando por una planificación y análisis, siguiendo por una fase de diseño e implementación y culminando con pruebas, despliegue en un entorno en la nube y documentación.

- El proceso de desarrollo implementa el marco de trabajo SCRUM, y por tanto, hace uso de conceptos como *Historias de Usuario* o *sprints* para su planificación.

- La aplicación deberá ser *responsive*, es decir: se deberá ver bien en múltiples dispositivos, tanto en entorno de escritorio como en entorno móvil

- El diseño de la plataforma será lo más claro y moderno posible, inspirado en Material Design [3], ya que es un diseño que actualmente se encuentra en auge en todo tipo de aplicaciones multiplataforma y resulta cómodo para gran parte de usuarios.

- Tanto la arquitectura de la plataforma como el modelado de los datos de los que haga uso la misma deberán garantizar que ECodium sea una plataforma multiplataforma y segura, que no comprometa la integridad de los datos que almacene, así como permita el acceso de diferentes roles de usuarios.

[3] `https://material.io`

- La plataforma se implementará en el stack seleccionado para ello, que como se explicará más adelante, es el stack MEVN, formado por MongoDB como Sistema Gestor de Bases de Datos - en este caso, de bases de datos no relacionales -, ExpressJS como marco de backend, Vue.js como marco de frontend y Node.js como entorno del servidor. El stack se complementa con la librería Quasar Framework sobre Vue.js, una capa gracias a la cual disponemos de distintos componentes web sobre los que construir la plataforma.
- La plataforma se desplegará en un entorno en la nube, emulando un *"paso a producción"* de la release final en el dominio registrado para tal fin (`https://ecodium.dev`).
- La documentación estará formada por un capítulo introductorio, una breve explicación del fundamento teórico del proyecto y una descripción exhaustiva del proceso de desarrollo en todas sus fases.

Capítulo 2

Estudio del alcance y viabilidad

En este capítulo se realiza un breve estudio del alcance y viabilidad del proyecto, comenzando por la descripción del alcance del mismo, siguiendo por un análisis de los antecedentes y *state of the art* de la situación actual, así como el análisis de resultados de una encuesta preliminar real llevada a cabo antes de comenzar a desarrollar la plataforma. El capítulo concluye con un análisis DAFO a modo de conclusión.

2.1. Descripción del alcance del proyecto

Así pues, el proyecto se constituye con un alcance determinado con el fin de satisfacer los objetivos anteriormente expuestos, y dejando las características que no sean del todo necesarias para versiones futuras. Un ejemplo muy notorio de esto es la **aplicación móvil**, que es una característica muy icónica para ser protagonista de una futura versión de la plataforma. No obstante, no se debe descuidar el mercado móvil, y por tanto la aplicación web a desarrollar deberá ser **multiplataforma** y contar con una interfaz *responsive*

El alcance, por tanto, del proyecto, se centra en el desarrollo full stack de la aplicación web multiplataforma de modo que cubra los **retos**, **eventos** y **talleres** de programación y tecnologías punteras, de cara a que la primera versión sea usable, concentre la esencia de la plataforma con aspectos como el *token* de la red Ethereum y tenga una arquitectura moderna y basada en la Nube.

Fuera del alcance de esta primera versión se encontrarán características futuras, mejora de características existentes y aspectos relevantes al mantenimiento de la plataforma.

2.2. Antecedentes y *state of the art*

Por lo expuesto en el capítulo anterior, existen a día de hoy diversas plataformas online para facilitar tanto el primer contacto con un lenguaje de programación como para desarrollar y ampliar conocimientos en un lenguaje ya conocido. De la misma manera, existen herramientas para organizar y participar en competiciones similares a un *hackathon* online, y en ocasiones herramientas para ambos fines - el aprendizaje y la competición. En

este capítulo se repasan brevemente el estado de las plataformas existentes más cercanas a la plataforma que se quiere a desarrollar.

Exercism: *Open source* y con tutores personales

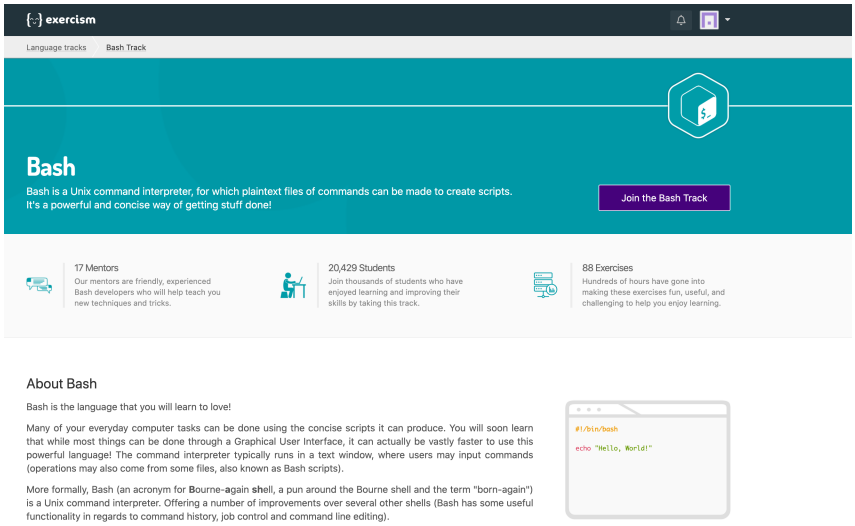


Figura 2.1: Página de inicio de Exercism

Exercism (`https://exercism.io`) es una plataforma *open source* que ofrece a sus usuarios aprender muchos lenguajes de programación a distintos niveles. Cuenta con un sistema de tutorías (llamada *mentoring*)

Hackerrank: Orientado a la programación competitiva y al mundo laboral

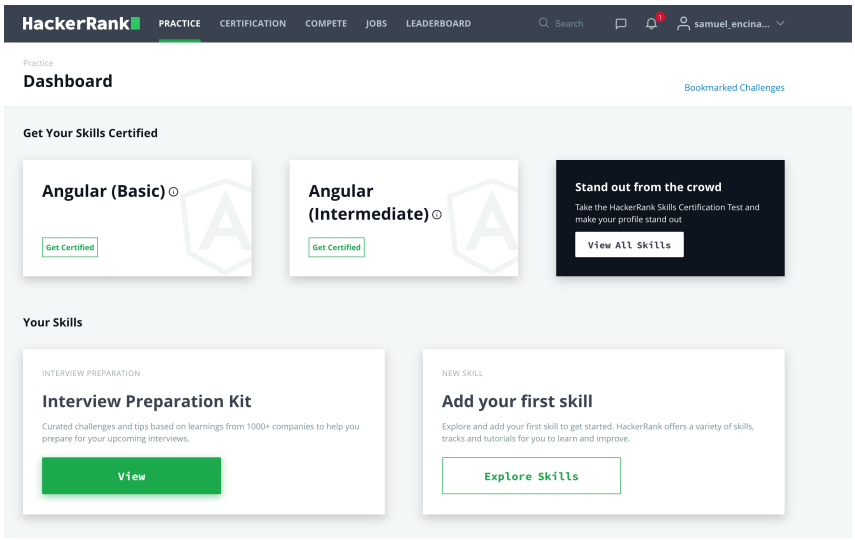


Figura 2.2: Página de inicio de Hackerrank

Hackerrank (`https://hackerrank.com`) permite a sus usuarios mejorar su habilidad en determinados conocimientos relativos al mundo de la programación y, mediante una serie de exámenes, poder llegar a obtener certificados, demostrando así que efectivamente poseen dichos conocimientos. De manera adicional, Hackerrank organiza concursos de programación competitiva e incluye un escaparate al mundo laboral.

CodingGame: Una plataforma gamificada y completa

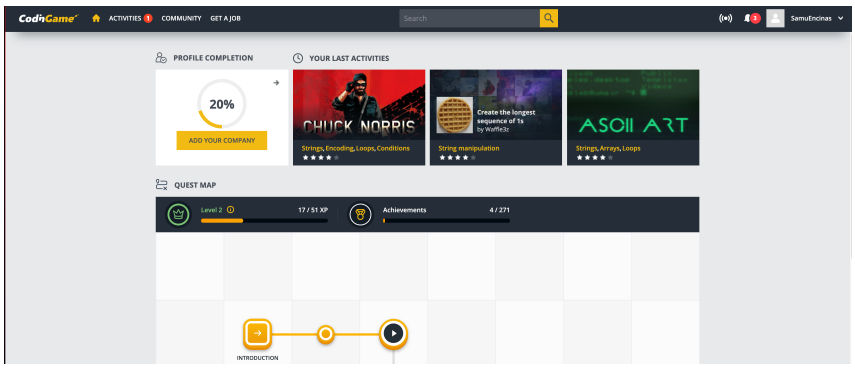


Figura 2.3: Página de inicio de CodingGame

CodingGame (`https://codinggame.com`) es una plataforma con enfoque gamificado, similar a la plataforma cuyo proceso de desarrollo se documenta en este Trabajo Fin de Grado. Así pues, permite a los usuarios aprender conocimientos de programación mediante la resolución de retos, así como competir en eventos y ganar logros. Aunque no sea la plataforma más utilizada, sí es de las plataformas más completas que existen, y de manera adicional ofrece características orientadas a la comunidad, como streamings en directo, y también un pequeño escaparate al mundo laboral, al igual que Hackerrank. CodingGame sirve de inspiración principal para la plataforma que se pretende desarrollar, si bien guarda algunos matices.

CodeWars (`https://codewars.com`) es otro ejemplo de plataforma de estas características con un enfoque gamificado, sin embargo, es menos completa que CodingGame.

BitDegree: Aprendizaje con recompensas basadas en Ethereum

En lo que respecta a recompensar a los usuarios con criptoactivos, apenas se pueden encontrar plataformas de estas características, no obstante, sí existe una aproximación. En 2018, el portal ZDNet documentaba en un artículo [4] la existencia de una *startup* que recompensaba a sus usuarios con criptomonedas por aprender Python. La plataforma a la que se refiere el artículo - **BitDegree** (`https://bitdegree.org`) existe, y tiene un *token* en la red Ethereum asociado a ella, de similares características al *token* que se crea para la plataforma descrita en este Trabajo Final de Grado. Los usuarios pueden optar a ganar el *token* de BitDegree completando cursos.

Conclusiones

En síntesis, existen numerosas plataformas que permiten aprendizaje de tecnologías digitales, así como organizar y participar programación competitiva. Algunas de ellas combinan ambos fines y algunas añaden un enfoque gamificado. Tan solo una plataforma combina el aprendizaje - en su caso, basado en cursillos - con las recompensas basadas en un criptoactivo de la red Ethereum. La plataforma cuyo proceso de desarrollo se refleja en este documento, así pues, no inventa nada nuevo pero sí es única, ya que combina los dos fines descritos, de una manera gamificada y con recompensas de Ethereum. Además, cuenta con el aliciente de la posibilidad de ser atractivo para las empresas a la hora de buscar talento mediante la plataforma.

2.3. Encuesta preliminar: resultados y análisis de los mismos

Con el fin de complementar el estudio de viabilidad del proyecto, se confeccionó una pequeña encuesta preliminar que se distribuyó antes de comenzar el proceso de desarrollo entre estudiantes de estudios relacionados con informática que no sean de primer año, así como trabajadores del sector con poca experiencia. En la encuesta, se preguntaban aspectos relevantes sobre la plataforma a desarrollar, para ver si a los potenciales usuarios les resultaba atractiva. La encuesta sigue estando disponible en `https://forms.gle/ZTa9aw1LCWfA3cFz5`

Estos han sido los resultados de dicha encuesta.

Conocimiento y experiencia de los encuestados

La mayoría de encuestados se sitúa en un 7 sobre 10 en cuanto a conocimiento de alguna disciplina informática, existiendo encuestados que aseguran dominar alguna disciplina informática y encuestados que se sitúan en niveles inferiores, pero en ningún caso por debajo de 5 sobre 10:

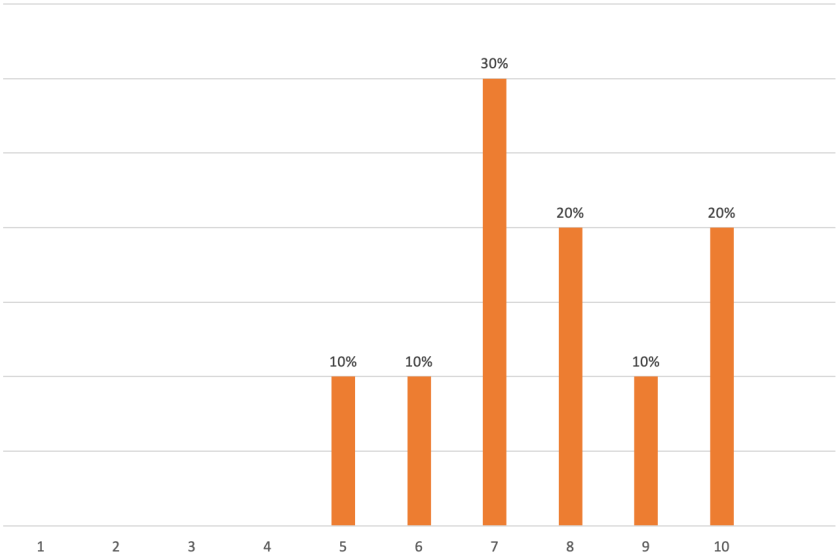


Figura 2.4: Nivel de conocimiento de los encuestados

En cuanto a experiencia, el 40 % de los encuestados cuenta con más de tres años de experiencia demostrable, existiendo gente con menos experiencia demostrable y gente sin experiencia demostrable.

Figura 2.5: Nivel de experiencia de los encuestados

Gamificación en el aprendizaje de lenguajes de programación

Todos los encuestados coinciden en que un enfoque gamificado es positivo para el aprendizaje de lenguajes de programación, si bien un 60 % de los encuestados cree que este enfoque dependería de qué lenguaje y cómo se quiere enseñar.

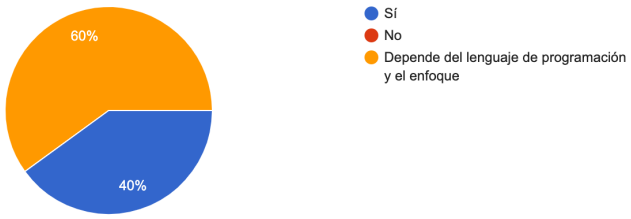


Figura 2.6: Opinión sobre la gamificación en el aprendizaje de lenguajes de programación

Hackathones online

El 80 % de los encuestados ve viable realizar *hackathones* y programación competitiva online, mientras que el 10 % opina que perdería la esencia, y el otro 10 % se muestra dubitativo, opinando que un *hackathon online* es viable en función del estilo del evento.

Figura 2.7: Opinión sobre la viabilidad de *hackathones* online

Aprendizaje autodidacta y gamificado de tecnologías punteras

Con el fin de no alargar demasiado esta sección, no se adjuntarán todas las respuestas recibidas y que resultan muy similares entre sí, pero sí alguna reseñable. La mayoría

son positivas, destacando que se necesita tiempo libre para llevarlo a cabo e incluyendo opiniones de encuestados que ya han participado en aprendizaje gamificado:

"Sí me gustaría aprender, la gamificación es una muy buena forma de aprender casi cualquier cosa"

"Estaría bien pero cuando me sobre algo de tiempo"

"Dentro de poco haré un curso online que utiliza la gamificación en el entorno de aprendizaje para programar un videojuego y su apartado gráfico (con suerte conseguiré hacerlo con colegas)"

"Seria especialmente util sobre todo para la gente que por trabajo o circustancias no puede dedicarse a estudiar ya que pueden aprender con dedicar una pequeña parte de su tiempo libre a ello."

2.4. Análisis DAFO y conclusiones

Para concluir este capítulo referente al alcance y viabilidad del proyecto, se adjunta un análisis DAFO del proyecto a crear. El análisis DAFO[1] supone una forma de concretar los retos a los que se enfrenta el proyecto de cara a elaborar una estrategia sobre el mismo.

Este es el diagrama DAFO elaborado sobre el proyecto con el fin de concluir el capítulo:

Figura 2.8: Diagrama de análisis DAFO (Debilidad, Amenaza, Fortaleza, Oportunidad)

Con este breve estudio, se puede concluir que el proyecto a desarrollar resulta viable, interesa al público objetivo y cuenta con un análisis estratégico llevado a cabo mediante

[1]Información extraída de la *"Guía fundamental del Análisis DAFO"*, accesible en `https://www.infoautonomos.com/plan-de-negocio/analisis-dafo/`

un análisis DAFO, en el que se pueden ver los retos y límites en los que se encuentra *ECodium.*

Capítulo 3

Fundamento Teórico del Proyecto

En este capítulo se pretende describir, desde un punto de vista global, los pilares teóricos sobre los que se asienta el proyecto que se detalla en este Trabajo Fin de Grado. El capítulo, por tanto, comienza describiendo a nivel conceptual las **bases de datos no relacionales**, el sistema de almacenamiento de información que constituye la base del modelado de datos de este proyecto.

Posteriormente, se describirá el *stack* tecnológico sobre el que se ha construido y que sienta las bases de la arquitectura del proyecto, el **stack MEVN** y la librería ***Quasar Framework***, así como fundamentos de la tecnología **blockchain** y la red **Ethereum**, que aportan a los usuarios del proyecto un aliciente para participar.

Finalmente, el capítulo concluirá con apuntes sobre la **gamificación** y el enfoque gamificado de plataformas como ésta, de modo que el usuario sienta que está jugando mientras aprende nuevos conocimientos o potencia conocimientos que ya posee.

3.1. Bases de datos no relacionales, NoSQL y MongoDB

3.1.1. Definición

Se conoce como **base de datos no relacional** o **base de datos NoSQL**, (del inglés *"Not Only SQL"*) a una clase de sistemas de gestión de bases de datos (SGBD) que se distingue en múltiples cuestiones de los sistemas relacionales, pudiendo administrar grandes volúmenes de datos no estructurados y altamente volátiles.

Mientras que los sistemas gestores de bases de datos basados en el modelo relacional son las aproximaciones más clásicas al manejo de bases de datos, los sistemas NoSQL han adquirido popularidad a pasos agigantados en los últimos años, debido a las exigencias modernas del desarrollo de software, que cada vez requiere almacenar cantidades más ingentes de datos que, a su vez, cada vez son más volátiles.

3.1.2. Diferencias con las bases de datos relacionales. Principios ACID y BASE y el teorema CAP

Según numerosos autores y publicaciones, entre las que se encuentra *."Experimental evaluation of NoSQL databases"* [1], la diferencia fundamental entre un sistema no relacional y uno relacional radica en el principio básico en el que se basa cada uno:

- Los **SGBD relacionales** se fundamentan en lo que se conoce como el **principio ACID** *(Atomic, Consistent, Isolated, and Durable)*, un principio que describe cuatro propiedades sobre las transacciones en este sistema:
  - **Atomicidad**, en el sentido en el que o la transacción se completa como una sola, o hace *rollback* anulando la transacción al completo.
  - **Consistencia** que garantice que solamente se guarden aquellos datos que sean válidos en el momento de la transacción.
  - **Aislamiento**, mediante el cual una transacción no afectará a otra.
  - **Durabilidad**, que garantice que los datos escritos no se pierdan.
- Los **SGBD no relacionales o NoSQL**, en cambio, toman como pilar el **principio BASE** *(Basically Available, Soft State, Eventually Consistent)*, un principio que define tres propiedades sobre las bases de datos:
  - **Disponibilidad básica**, es decir, que los datos parezcan estar disponibles constantemente.
  - **Estado flexible**, de tal forma que la consistencia no tiene que estar garantizada en todo momento.
  - **Consistencia eventual**, de modo que las bases de datos terminen exhibiendo, eventualmente, la consistencia de sus datos.

A priori, el principio ACID puede parecer el modelo que mejor describe cualquier proyecto que precise de una base de datos ya que la consistencia de los datos es el protagonista, sin embargo, el principio BASE sacrifica los exigentes requisitos para mantener una consistencia constante, en pos de ganar otros beneficios, como escalabilidad y resiliencia (tolerancia a fallos) del sistema.

Para comprender esto de una forma visual, es útil el **teorema CAP**, anunciado por Eric Brewer, de la Universidad de Berkeley. Este teorema establece que es imposible que un almacén de datos distribuido garantice estas tres propiedades

- **Consistencia** de los datos
- **Disponibilidad** de los datos
- **Tolerancia a particionado**, también llamado tolerancia a fallos, consiste en la capacidad del sistema de continuar funcionando a pesar de que distintos nodos de

la red intercambien o retrasen un número arbitrario de mensajes. Esta propiedad interfiere con el Aislamiento del principio ACID, por lo que no podrá satisfacerse por parte de modelos que se sustenten en este principio.

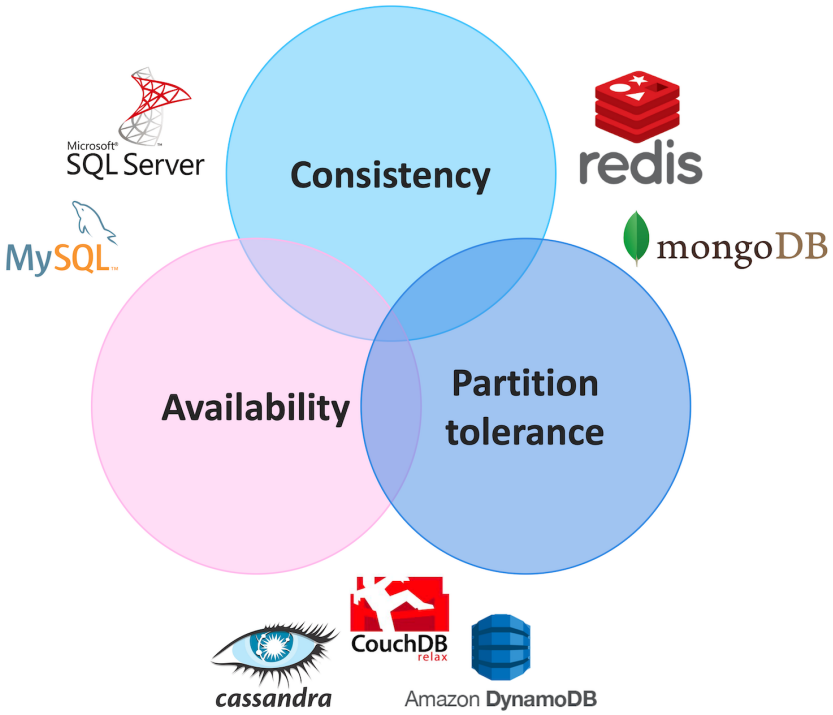


Figura 3.1: Comparativa de distintos sistemas gestores de bases de datos según el teorema CAP[1]

Como se puede ver en el gráfico y por lo explicado anteriormente, los sistemas relacionales garantizan Consistencia y Disponibilidad, pero no Tolerancia a particionado, mientras que los sistemas no relacionales se pueden dividir entre los que garantizan Disponibilidad y Tolerancia a particionado y los que garantizan Consistencia y Tolerancia a particionado. Dentro de este último grupo se encuentra MongoDB, el sistema gestor de bases de datos usado en el proyecto.

3.1.3. Tipos de SGBD no relacionales

Fundamentalmente existen cuatro tipos de sistemas gestores de bases de datos no relacionales:

- **Bases de datos clave-valor**, donde cada elemento está identificado por un par único clave-valor, lo que permite una rápida recuperación de los datos. Un ejemplo de sistema gestor de bases de datos no relacional de este tipo sería Cassandra[2]

[1] Imagen extraída de `https://www.itdo.com/blog/que-modelo-de-base-de-datos-se-adapta-a-mi-proyecto/`

[2] `https://cassandra.apache.org`

- **Bases de datos basadas en documentos**, donde la información se representa mediante documentos, que suelen seguir el formato JSON o XML. Cada registro, igual que en el esquema anterior, conlleva un par único clave-valor, lo cual conserva la rápida recuperación de datos, pero al estar basada en documentos, permite consultas más complejas sobre el contenido del documento. El ejemplo más importante de este tipo es MongoDB[3]

- **Bases de datos en grafo**, donde los datos se representan como nodos de un grafo, y las relaciones existentes entre los datos se representan como las aristas del grafo. La base de datos, por tanto, se recorre aplicando la Teoría de Grafos. Un sistema de este tipo es Neo4J [4]

- **Bases de datos orientadas a objetos**, que siguen el paradigma de la orientación a objetos. No son muy utilizadas y un sistema que ejemplifica este tipo es ObjectDB [5]

3.1.4. MongoDB

MongoDB es uno de los sistemas gestores de bases de datos por antonomasia hoy día, y son muchos los expertos que consideran MongoDB como un sistema interesante para que conozca cualquier desarrollador. Las razones los explica el desarrollador *full stack* Adrián Alonso en su artículo [15]:

- MongoDB ofrece un **rendimiento** mucho mayor al de los sistemas gestores de bases de datos tradicionales.

- **Replicación**, ya que MongoDB soporta el tipo de replicación maestro-esclavo.

- **Balanceo de carga**, permitiendo así la escalabilidad, una característica muy poco vista en sistemas gestores de bases de datos tradicionales.

- **Almacenamiento**, pudiendo utilizarse MongoDB con un sistema de archivos.

- **Uso adecuado**, debido a la cantidad de casos de éxito ya documentados que existen.

Las bases de datos construidas mediante este sistema gestor de bases de datos siguen una **arquitectura** determinada, congruente con la naturaleza basada en documentos de MongoDB:

[3] `https://mongodb.com`
[4] `https://neo4j.com`
[5] `https://www.objectdb.com`

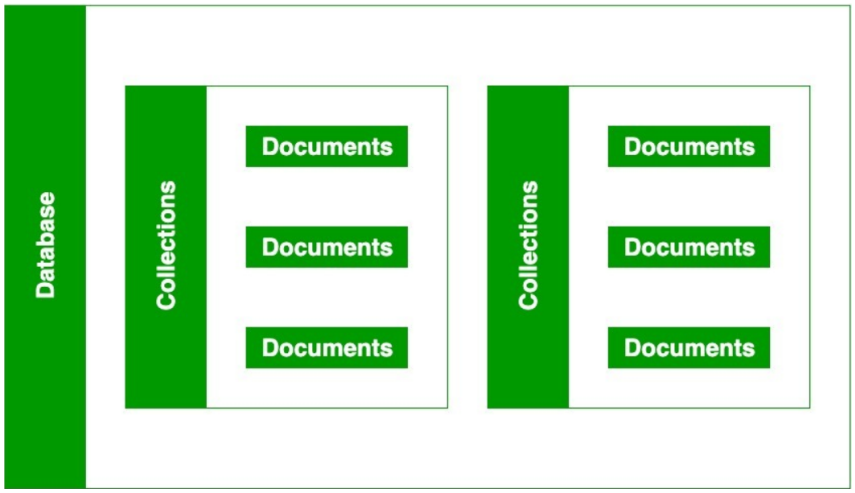


Figura 3.2: Diagrama simplificado de la arquitectura de una base de datos construida en MongoDB[6]

Como se puede ver en el diagrama, la base de datos se divide en **colecciones**, que serían el concepto análogo a las tablas en modelos tradicionales, y dentro de cada colección se encuentran los **documentos**, que estarían a su vez descritos en formato JSON, albergando un conjunto de propiedades descritas en el modelo clave-valor.

3.1.5. NoSQL en ECodium

A modo conclusivo, se exponen las razones por las que se ha escogido, para desarrollar el proyecto, un sistema NoSQL frente a uno relacional, y más específicamente, MongoDB:

- Se ha escogido un sistema gestor de bases de datos no relacional o NoSQL debido a características, que por definición y por el teorema CAP, aportan estos sistemas, como la **disponibilidad continua** como principal objetivo, la **resiliencia** o tolerancia a fallos que éstos aportan o la **escalabidad**, ya que disponer de un sistema escalable resulta realmente interesante en caso de lanzar este proyecto para usuarios finales en un futuro.

- Dentro de los sistemas NoSQL, se ha escogido **MongoDB** por la cantidad de casos de éxito documentados que existen, con la consecuente cantidad generosa de información disponible, además de por sus principales características, rendimiento y propiedades. Además, su sistema cloud para crear y desplegar bases de datos, MongoDB Atlas[7], ofrecía crédito gratuito para estudiantes disponible el programa GitHub Student Developer Pack[8], por lo que fue un aliciente muy interesante de cara a construir la base de datos del proyecto en este sistema.

[6]Imagen extraída de `https://www.geeksforgeeks.org/mongodb-database-collection-and-document/`

[7]`https://cloud.mongodb.com`

[8]`https://education.github.com/pack?sort=newest&tag=Cloud`

3.2. Stack MEVN y *Quasar Framework*

3.2.1. Introducción

Tradicionalmente en desarrollo web, un **stack** se podría definir como un conjunto de *software* especialmente configurado para implementar páginas y aplicaciones web. Estos *stacks* se componen, principalmente, de un **sistema operativo** sobre el que construir el stack, **servidor web** que sirve documentos mediante el protocolo HTTP, un **Sistema Gestor de Bases de Datos** que almacenen de forma persistente todos los datos necesarios para la correcta operativa de la aplicación web a implementar, y un **intérprete de scripts** que permita ejecutar código, de manera que se puedan conseguir experiencias web dinámicas tanto en el lado cliente - lo que se conoce como *frontend* - como en el lado servidor - lo que se conoce como *backend*.

El *stack* más clásico es, sin duda, **LAMP**. Un *stack* formado por Linux como sistema operativo, Apache como servidor web, MySQL como sistema gestor de bases de datos y PHP como intérprete de scripts.

3.2.2. Stacks web modernos: MEAN, MERN y MEVN

Debido en parte al mero avance de las tecnologías, al auge de la figura del ***full stack developer*** o a la evolución de las tendencias en diseño de experiencias de usuario, el concepto de **stack** ha cambiado ligeramente. Hoy en día, los tres *stacks* de desarrollo web más punteros son MEAN, MERN y MEVN, los cuales comparten tres de las cuatro letras:

- La **N** se corresponde con **node.js**, un entorno en tiempo de ejecución en el lado servidor basado en Javascript. Hoy en día, es uno de los entornos más usados para la implementación de *backends* en aplicaciones modernas, debido a su eficiencia.
- **E**, por su parte, se refiere a **Express.js**, un *framework* de *backend* de aplicaciones web para **node.js**. Express.js es el marco más utilizado para crear **APIs** en entornos basados en **node.js**
- **M** hace referencia a **MongoDB**, el sistema gestor de bases de datos NoSQL (de carácter **no relacional**) por antonomasia en el desarrollo web moderno.
- Por último, la letra que varía, **A, R o V**, se corresponde con Angular, React o Vue.js respectivamente, y hace referencia al *framework* de *frontend* que se utilice para desarrollar la parte del lado servidor de la aplicación web. Los tres están basados en JavaScript y comparten parte de filosofía y metodología común.

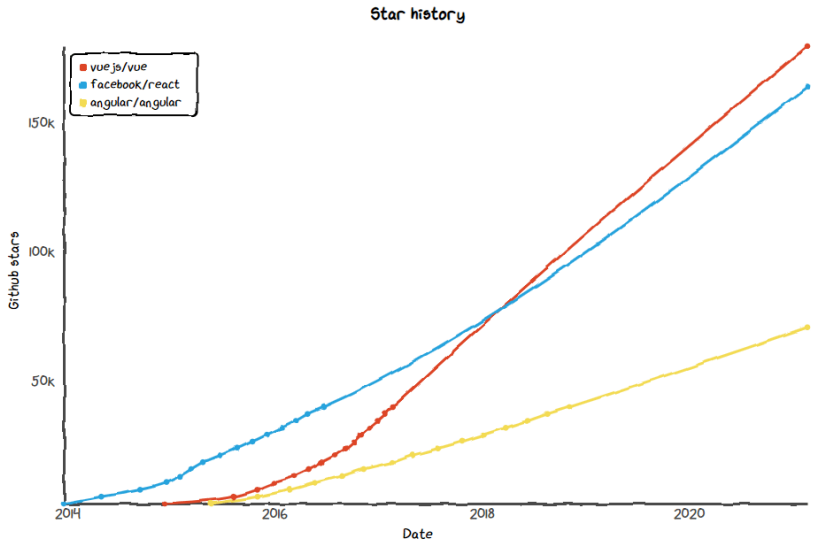


Figura 3.3: Evolución de la valoración de Angular, React y Vue por parte de desarrolladores

3.2.3. *Stack web* en ECodium: MEVN y *Quasar Framework*

En el caso del proceso de desarrollo de la aplicación ECodium, se hará uso de **Vue.js** como entorno de *frontend* por ser un entorno con una curva de aprendizaje sencilla, además de contar con una aceptación por parte de los desarrolladores que crece a notablemente más velocidad que la aceptación de sus homólogos.

Por tanto, el *stack* que se utilizará será el *stack* **MEVN**, apoyándose en MongoDB, Express.js, Vue.js y node.js como los cuatro pilares fundamentales de la arquitectura del proyecto.

Quasar Framework(`https://quasar.dev`) es una librería de *frontend* basado en Vue.js, que incorpora una serie de **componentes web** genéricos, así como una **estructura** de proyecto predeterminada. Permite generar con rapidez aplicaciones web visualmente atractivas apoyándose en la estética **Material Design** desarrollada por Google. Dicha librería solamente aporta componentes visuales, dejando en manos del desarrollador aspectos como la lógica de cada componente, el manejo de datos, la comunicación con el *backend* del servidor...

Quasar Framework, por tanto, es una capa de personalización de vue.js a muy alto nivel y, dentro de este proyecto, constituye la base de la experiencia e interfaz de usuario diseñada, que se detallará más adelante.

3.3. Gamificación

3.3.1. Definición y conceptos básicos

Una de las definiciones más pragmáticas y concretas que existen del concepto de gamificación es la definición de Dan Hunter y Kevin Werbach en 2012 [16]:

"Gamificación es el uso de elementos de juegos y técnicas de diseño de juegos en contextos que no son de juegos"

La gamificación **no** es, por tanto, jugar en horario de trabajo o restar importancia a asuntos que sí deberían tomarse en serio, sino que simplemente la gamificación consiste en coger aquellas características de los juegos e introducirlos en ámbitos sin naturaleza lúdica, con el fin de fomentar la participación del usuario.

3.3.2. Elementos de la gamificación y su uso en ECodium

Dan Hunter y Kevin Werbach [16] clasifican los distintos elementos que componen la gamificación en tres grandes grupos: las **mecánicas**, que hacen las veces de motor y reglas del juego, constituyendo una pieza clave en el funcionamiento del mismo. Estas mecánicas se llevan a cabo mediante las **dinámicas**, segundo tipo de elementos de la gamificación, que constituyen precisamente en el **cómo** se realizan las mecánicas. Por último, los **componentes** corresponden al conjunto de materiales, recursos y herramientas de los que disponemos como organizadores de lo que queremos gamificar.

En *"Técnicas de gamificación aplicadas en la docencia de Ingeniería Informática"* [5], su autora adapta esta aproximación de Hunter y Werbach a un diagrama piramidal, de manera que resulte más visual:

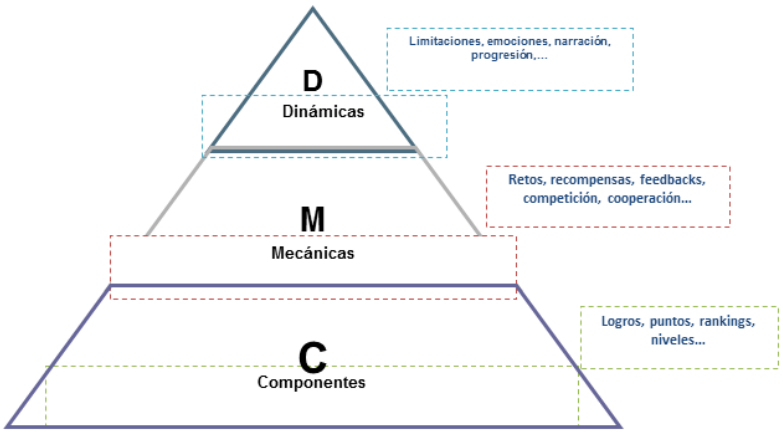


Figura 3.4: Pirámide de los elementos de gamificación[9]

[9]Imagen extraída del artículo de Carina Soledad González González [5], que a su vez adapta el contenido de Kevin Werbach [16]

A continuación, se resume cada elemento de gamificación con un **cuadro resumen** confeccionado para ofrecer de manera visual ejemplos de cada elemento, y cómo se aplican y se integran en ECodium:

Dinámicas

Dinámica	*Definición e integración en ECodium*
Progresión	Los jugadores pueden ascender o descender de categoría mediante su participación
Expresión	Los jugadores deberán documentar debidamente sus candidaturas a eventos o entregas en talleres.
Limitaciones	Los jugadores deberán respetar las limitaciones y restricciones asociadas a cada reto, taller o evento y buscar la forma de progresar, aún con estas restricciones, en la plataforma.

Cuadro 3.1: Dinámicas de gamificación en ECodium

Mecánicas

Mecánica	*Definición e integración en ECodium*
Competición	Los jugadores compiten por superar a otros jugadores en los **eventos** y **talleres**.
Desafíos	Los jugadores disponen de una serie de **retos** que quieren resolver.
Aprendizaje autodidacta	Los jugadores pueden aprender, de manera autodidacta, uno o más conceptos, o reforzar conocimientos que ya posean
Cooperación	Los jugadores pueden colaborar entre sí para alcanzar un objetivo difícil común, por ejemplo, formando un equipo para un **evento colectivo**.
Compartición	Los jugadores pueden **compartir** retos, talleres, eventos, logros desbloqueados... con otros jugadores o jugadores potenciales.
Recompensas	Los jugadores pueden obtener **recompensas** de diversa índole conforme progresen en la plataforma.

Cuadro 3.2: Mecánicas de gamificación en ECodium

Componentes

Componente	*Definición e integración en ECodium*
Puntos de Experiencia	Los puntos de experiencia simbolizan el progreso del jugador en la plataforma. Determinan el nivel del jugador, se pueden intercambiar por Pases de Acceso y, sobre todo, sirven para reclamar TALENTOS (el token ERC20 de la plataforma)
Sistema de Niveles	Existe un sistema de niveles en la plataforma que viene determinado por los puntos de experiencia acumulados del usuario. Este sistema de niveles determina el rango del jugador.
Pases de Acceso	Un pase de acceso es un objeto virtual que permite a su jugador acceder a talleres o a eventos. Se adquiere gastando Puntos de Experiencia, si bien dichos puntos podrán ser recuperados (con creces) en el caso de buen desarrollo del evento o taller por parte del usuario
Rangos	Cada jugador tiene un rango asociado, lo que le permite acceder a unos eventos u a otros.
Ranking y Comunidad	En la plataforma existe una sección dedicada a la Comunidad donde existe un Ranking de los distintos jugadores.

Cuadro 3.3: Componentes de gamificación en ECodium

3.3.3. Conclusiones

Existe mucha más teoría acerca de la gamificación y sus interesantes beneficios sobre el aprendizaje y afianzamiento de conceptos o metodologías, sin embargo, simplemente con estos conceptos fundamentales se puede definir ECodium como una plataforma gamificada, que alberga en su seno tanto dinámicas como mecánicas y componentes que así lo corroboran.

3.4. Blockchain, Ethereum y los tokens ERC20

3.4.1. Definiciones fundamentales

En esta sección se pretende definir, sin extenderse demasiado ya que no es el tema central de este Trabajo Fin de Grado, los conceptos fundamentales de Blockchain y la red Ethereum y cómo afectan al proyecto de ECodium, desde el más bajo al más alto nivel de abstracción:

- Podemos definir ***blockchain*** o cadena de bloques como una estructura de datos cuya información se agrupa en unos conjuntos llamados *bloques* que albergan de manera adicional metadatos y se encuentran conectados entre sí, formando una red descentralizada, distribuida y basada en la criptografía. Se puede entender, desde un nivel muy alto, como una *"libreta compartida"* que almacena todas las transacciones y está diseñada para ser incorruptible.[10]

  - A nivel conceptual, podemos representar los distintos conceptos que engloban blockchain con el siguiente diagrama extraído del artículo publicado por Antoni Olivé [13] :

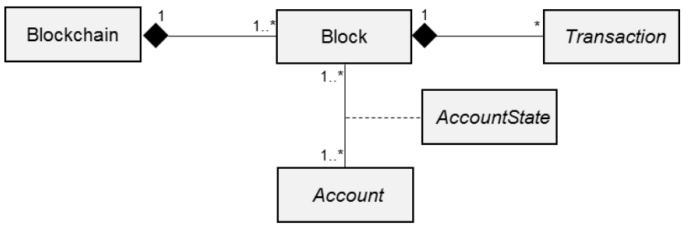


Figura 3.5: Diagrama conceptual del concepto de blockchain

  - Como se puede ver en el diagrama, dentro de una cadena de bloques existen bloques y dentro de los cuales existen Cuentas y Transacciones (entre cuentas). Además, en un instante concreto, una cuenta está en un Estado determinado.

- Si bien la primera cadena de bloques fue la de Bitcoin, lo cierto es que no es la única. En 2013, Vitalik Butelin lideró un grupo de desarrolladores con el que inició un proyecto denominado **Ethereum**[11]. *Ethereum* nace, por tanto, con la finalidad de extender las capacidades de la *blockchain* y convertirse en una plataforma para

[10] El divulgador de contenido científico *Quantum Fracture* explica en un vídeo, con animaciones y con lenguaje cercano, esta manera de entender el concepto de blockchain como una *"libreta compartida"*. Enlace al vídeo: `https://www.youtube.com/watch?v=pqEidVW9da0&t=50s` (Min. 15:39 - 21:58)

[11] `https://ethereum.org/en/`

crear aplicaciones descentralizadas y *tokens* digitales, tal y como relata María Nieves Pacheco [13] .

  - Es importante señalar que *Ethereum* no es una criptomoneda, sino más bien un protocolo, aunque usualmente se confunden entre sí.
  - La criptomoneda asociada a este protocolo se denomina ETH, y ha sido considerada tradicionalmente por los medios como la principal competidora de Bitcoin.[12]

- En la plataforma Ethereum, existe un concepto llamado **contrato inteligente**, siendo esta plataforma la pionera en incorporar dicho concepto. Un contrato inteligente, o *smart contract* en inglés, no es más que la adaptación de un contrato tradicional a la perspectiva digital de la *blockchain*, si bien existe una diferencia sustancial entre ambos contratos: los contratos inteligentes no precisan de intermediarios.

- Un **token** (*"ficha"* en inglés) se define, en este escenario, como *ünа unidad de valor que una organización [o entidad privada] crea para gobernar su modelo de negocio y dar más poder a sus usuarios para interactuar con sus productos, al tiempo que facilita la distribución y reparto de beneficios"*[13]

- En este sentido, un token actúa como un criptoactivo, en el sentido de que se puede comerciar con él, y a su vez actúa de manera determinada en la plataforma creadora, ya sea como aliciente para un fin determinado, como moneda de cambio... Las posibilidades son ilimitadas.

- Existen generalmente dos tipos sustanciales de tokens, los **tokens fungibles** y los **tokens no fungibles**. Los primeros pueden fraccionarse, funcionando de manera similar a como funciona una moneda, mientras que los segundos son piezas únicas, que no pueden dividirse ni cambiarse entre sí. Estos últimos se han popularizado últimamente con el arte digital basado en NFTs, y han estado servidos de polémica debido a su uso meramente especulativo.[14]

- En la red *Ethereum*, existen distintos modelos de token, siendo uno de los más populares el estándar **ERC20**. Consisten en contratos inteligentes, programados con un lenguaje de programación propietario llamado Solidity, que funcionan como un token fungible, dando a su vez flexibilidad a los desarrolladores para personalizar como deseen el token mediante la programación del contrato inteligente pertinente.

[12] `https://www.bolsamania.com/noticias/criptodivisas/ethereum-versus-bitcoin-profunda-division-sobre-quien-ganara-carrera-criptos--7899546.html`

[13] Definición extraída del libro de Mougayar [12] y traducida al castellano por María Nieves Pacheco [7]

[14] Comparativa sintetizada de este artículo de la web BlockchainServices: `https://www.blockchainservices.es/novedades/token-fungible-ft-vs-token-no-fungible-nft/`

[14] `https://soliditylang.org`

3.4.2. Token ERC20 de ECodium: TALENTO

Es precisamente un **token ERC20** el que se ha creado para complementar a la plataforma de este proyecto. El TALENTO (TAL) es un token ERC20 real, con su correspondiente *smart contract* programado en Solidity, el cual se mostrará y describirá en el capítulo 7.2 de este Trabajo Final de Grado.

No obstante, para ahorrar las tasas de GAS[15] de la red Ethereum, las cuales ahora mismo son elevadas por el ingente uso de la red, se ha hecho uso de una **red de pruebas**.

En este tipo de redes, se trabaja con una cantidad de ETH simulada, de manera que se pueden emular los costes de GAS y realizar cualquier tipo de operaciones con *smart contracts* que se desee probar. Si una funcionalidad de un *smart contract* funciona correctamente y como se espera en una red de prueba, solamente será cuestión de desplegarlo en la red principal y pagar el GAS respectivo - esta vez, con ETH reales - para hacerlo funcionar en la red real.

En el caso del proyecto, se ha trabajado con la **red de prueba Ropsten**[16]

El principal problema actual de las plataformas con tokens ERC20 es precisamente este, el alto coste de las tasas de GAS, ya que normalmente recae en los usuarios, los cuales normalmente pierden el interés por el criptoactivo o esperan a almacenar suficiente criptoactivo como para que sea rentable pagar los costes de GAS que requiere la transacción necesaria para recibirlo.

Por este motivo, han surgido alternativas a ERC20, que cuentan con los mismos principios y tienen tasas con un coste mucho menor. El mayor ejemplo es el estándar BEP-721, de la red Binance, de comportamiento similar a ERC20 pero en la blockchain propietaria de Binance.

Pese a lo anteriormente expuesto, se ha decidido continuar el desarrollo del proyecto en la red Ethereum ya que considero que resulta más interesante al ser una red de mayor envergadura, aunque si este proyecto se lanzara al público, probablemente resultara interesante trasladar el concepto del TALENTO (TAL) a un estándar con menores costes de GAS para hacerlo más atractivo al público.

[15]El GAS de Ethereum es una unidad que mide el costo computacional de una operación en la red Ethereum, ya sea una transacción o el despliegue de un *Smart Contract*. Más información sobre el GAS: `https://coinmotion.com/es/que-es-el-gas-ethereum/`

[16]Más información sobre Ropsten: `https://daiparaprincipiantes.com/red-ropsten-y-metamask/`

[16]`https://academy.binance.com/en/glossary/bep-721`

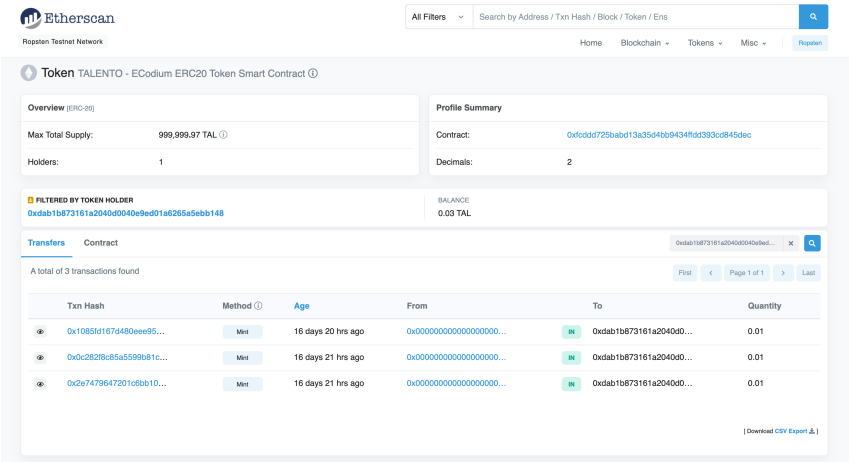


Figura 3.6: Captura de pantalla de Etherscan[17], desde donde se puede obtener información sobre el token ERC20 de ECodium: el TALENTO

[17] `https://etherscan.io`

Capítulo 4

Planificación y Gestión del Proyecto

En este capítulo se detalla en profundidad la metodología de gestión del proyecto llevada a cabo durante todo el proceso de desarrollo del mismo. Comenzará, por tanto, describiendo esta metodología para después mostrar como siguiendo esta metodología se ha llevado a cabo una estimación de tiempos del proyecto, así como una aproximación a la estimación de costes.

A continuación, se adjuntará y detallará un presupuesto estimado que comprenda los gastos para todas las herramientas necesarias para el desarrollo y despliegue de la aplicación y, finalmente, el capítulo concluirá con un balance de la planificación de proyecto, explicando cómo se ha seguido y si ha existido algún problema.

4.1. Metodología de gestión del proyecto

La metodología de gestión de este proyecto se fundamenta en las metodologías ágiles. Si bien es cierto que estas metodologías van orientadas a equipos de trabajo y quizá pueda parecer que no es la metodología más idónea a seguir para el desarrollo de un trabajo individual, realmente una metodología ágil puede adaptarse al trabajo individual para seguir obteniendo sus beneficios[1]

De este modo, la metodología de gestión de este proyecto se fundamenta en los siguientes puntos:

- Las semanas de trabajo se dividen en **sprints de trabajo** de **18 días** de duración cada uno, que concluyen con una **release** que se despliega en privado. Generalmente, en el desarrollo de metodologías ágiles se recomienda establecer sprints con una duración lo más mínima posible, sin embargo, la única norma es que no sea demasiado extenso, esto es, que sobrepase los 30 días de duración. Son muchos los medios

[1]En el blog de LucidChart dedican una entrada precisamente a esto: adaptar el marco de trabajo SCRUM (un marco de trabajo basado en metodologías ágiles) a un trabajo que se realiza de manera individual. Más información: `https://www.lucidchart.com/blog/scrum-for-one`

que señalan las 2 semanas como el tiempo idóneo de duración de sprint[2] y también existen medios que opinan que 3 semanas de duración es más favorable[3], por lo que me decanté en un punto medio: dos semanas y media, o lo que es lo mismo, 18 días.

- Siguiendo los principios de las metodologías ágiles, no deberían existir descansos ni paradas de ningún tipo entre sprints. Es importante señalar que si bien se ha intentado respetar al máximo,se ha tenido que repartir el tiempo entre las horas dedicadas a las prácticas en empresa, y adicionalmente, en febrero, tuve que dedicar tiempo también a un cursillo, por lo que en los sprints más tardíos se ha sido algo más riguroso.

- Se han considerado los días naturales en vez de los días laborables, ya que, al ser un proyecto individual compaginado con trabajo, cursos y demás, no todos los días se dedicará la misma cantidad de tiempo, pudiendo recuperar tiempo perdido entre semana en fines de semana. En este sentido, cada sprint dura 18 días naturales de lunes a domingo.

- La **planificación temporal** por tanto, se construye repartiendo los Puntos de Historia en los distintos sprints, de manera más o menos equilibrada, con el fin de que todos los sprints tengan la misma carga de trabajo.

- La **estimación temporal** se lleva a cabo mediante **Puntos de Historia**, una métrica de estimación idónea para proyectos construidos siguiendo metodologías ágiles. Como se verá en el capítulo 4, la especificación de requisitos del proyecto se lleva a cabo mediante Historias de Usuario, por tanto, la estimación del mismo se llevará a cabo estimando el esfuerzo de las mismas con Puntos de Historia.

- La **estimación de costes** resulta algo más compleja de estudiar por la naturaleza del proyecto y su metodología, si bien se aplicará una aproximación a la estimación de costes del proyecto en el respectivo epígrafe.

- Finalmente, el **balance de la planificación** se lleva a cabo *a posteriori* tras la conclusión del proceso de desarrollo, determinando si la gestión ha sido o no acertada y documentando si han existido retrasos o modificaciones en la estimación y planificación inicial.

[2]Más información en esta entrada del portal Hackernoon: `https://hackernoon.com/what-is-the-optimal-sprint-length-in-scrum-368e966f3243`

[3]Más información en: `https://www.linkedin.com/pulse/3-week-sprint-conner-birch`

4.2. Estimación de tiempos y costes

4.2.1. Introducción y planificación temporal inicial

El **proceso de desarrollo** del proyecto comienza, tras los exámenes del Primer Cuatrimestre, con una **fase conceptual** que, si bien sí forma parte del proceso en sí, no tiene Historias de Usuario ni tareas concretas asociadas, sino más bien consiste en idear, investigar, definir, planificar y describir de manera conceptual el proyecto, así como las tecnologías utilizadas, de cara a tener claro qué hacer durante los *sprints* de trabajo.

Sin embargo, para que no repercuta en la planificación por sprints de 18 días ni descoloque nada, en los primeros días, se decidió otorgar a esta fase una duración equivalente a dos *sprints* de trabajo: 36 días.

En esta fase también se analizan y elicitan los requisitos del proyecto, extrayendo un total de 21 Historias de Usuario, de mayor o menor complejidad, que se reparten en un total de 8 Épicas de Usuario. El desarrollo de estas ocho Épicas de Usuario se reparte en un total de **seis *sprints* de trabajo**. Todos estos conceptos sobre análisis y elicitación de requisitos, así como las decisiones tomadas acerca de los mismos, se detallarán en el epígrafe respectivo.

El resto de planificación temporal ya continúa por *sprints*, dividiendo el calendario en las ocho porciones de 18 días pertinentes - dos para la fase inicial, y una para cada uno de los seis *sprints* de trabajo:

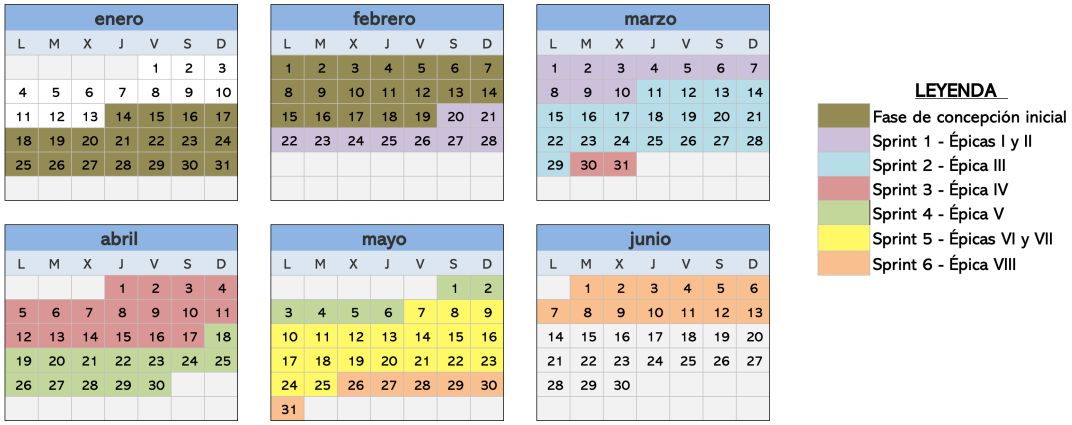


Figura 4.1: Calendario de la planificación temporal del proceso de desarrollo de ECodium

4.2.2. Estudio de la complejidad por Puntos de Historia

Dentro de los proyectos que siguen metodologías ágiles, existen diversas técnicas de estimación:

- El ***planning poker*** es una técnica de estimación para proyectos ágiles, basada en el consenso que se produce al jugar a un determinado juego que consiste en que

cada participante del equipo de desarrollo otorga, con unas cartas, unos puntos que siguen la serie **0, 1, 2, 3, 5, 8, 13, 21, 34...** a cada Historia de Usuario elicitada.[4]

- De la técnica anterior surge el concepto de **Puntos de Historia**, como esos *puntos que siguen la serie numérica de Fibonacci*. Es importante señalar que un Punto de Historia **no** equivale a horas de trabajo, ni tampoco es lo mismo que un Punto de Función, aunque a menudo sean conceptos que se confundan.

  Por definición, los **Puntos de Historia** (referidos en este documento en algunas secciones como P.H.) es una métrica que indica la complejidad relativa de un ítem, ya sea una tarea, una historia de usuario o una épica entera.[5]

- La técnica de **tallas de camiseta** se fundamenta en clasificar las distintas Historias de Usuario y sus tareas en *tallas* según su esfuerzo, siendo XS la talla para describir tareas muy sencillas y XXL la talla para describir tareas complejas. Resulta una técnica de estimación muy visual, pero poco precisa. [6]

- La **estimación por afinidad** consiste en una mezcla entre ambos conceptos. Se disponen una serie de tallas (usualmente de la XS a la XL) y se realiza un *mapping* entre cada talla y una cantidad de Puntos de Historia.[7]

Por tanto, la **estimación de la complejidad** que se lleva a cabo en este proceso de desarrollo se basa en los Puntos de Historia como métrica principal para las tareas, las historias de usuario y las épicas de usuario. No obstante, a modo de matiz, las Historias de Usuario se estiman siguiendo una técnica de estimación similar a la **estimación por afinidad**, estableciendo el siguiente *mapping*:

- **Complejidad MENOR:** Historias de Usuario de 2 Puntos de Historia
- **Complejidad MEDIA:** Historias de Usuario de 3 Puntos de Historia
- **Complejidad MAYOR:** Historias de Usuario de 5 Puntos de Historia
- **Complejidad CRÍTICA:** Historias de Usuario de 8 Puntos de Historia
- No existen Historias de Usuario de 1 Punto de Historia, pero sí tareas y, si alguna Historia de Usuario supera los 8 Puntos de Historia - es decir, que se estima que tiene 13 Puntos de Historia o incluso más - es convertida en una **Épica de Usuario** y dividida en Historias de Usuario con el fin de desgranar más las tareas a la hora de planificar.[8]

[4]Más información sobre el *planning poker* en: `https://www.mountaingoatsoftware.com/agile/planning-poker`

[5]Más información sobre los Puntos de Historia en: `https://www.javiergarzas.com/2015/06/puntos-historia-para-estimar-y-no-horas.html`

[6]Más información en: `http://getskillsblogs.com/agile-estimation-with-t-shirt-sizes/`

[7]Más información en: `https://www.techagilist.com/agile/scrum/affinity-estimation-agile-estimation-method/`

[8]Con la excepción de la Historia de Usuario 8.1, que se ha estimado en 13 Puntos de Historia y no es considerada Épica para no romper con la agrupación establecida.

4.2.3. Estimación de tiempos por sprint

La **estimación de tiempos** por sprint sí puede hacerse con rigor, y de múltiples formas distintas. En este caso que se documenta, dado que el proyecto se divide en épicas de usuario, las épicas en historias y las historias en tareas, y, a su vez, todos estos ítems tienen una serie de **Puntos de Historia** asociados como métrica del esfuerzo requerido para finalizarlos, se ha optado por listar todas las tareas del sprint y estimar los días necesarios para cada una.

Asimismo, el **alcance** del sprint viene determinado por el número total de Puntos de Historia a completar.

Las tablas se han diseñado en un programa externo, ya que la metodología de confección de tablas en LaTeX es bastante precario para tablas de una magnitud tan elevada como estas, por lo que se encuentran insertadas como imágenes, si bien son descritas como cuadros para que se listen correctamente en su índice:

Así pues, comenzamos a detallar, sprint por sprint, la estimación temporal mediante las tablas que se han confeccionado para tal fin:

Sprint 1

SPRINT 1 – Épicas I y II (20 de febrero – 10 de marzo)				
ALCANCE DEL SPRINT	26 Puntos de Historia			
TAREAS:				
Historia de Usuario	**Tarea**	**Puntos de Historia estimados**	**Inicio**	**Fin**
HdU 1.1. – Creación del proyecto y landing page (Complejidad MENOR – 2 Puntos de Historia)	Crear proyecto tanto a nivel cliente como a nivel servidor	1 Puntos de Historia	20 de febrero	20 de febrero
	Crear landing page	1 Puntos de Historia	20 de febrero	21 de febrero
HdU 1.2.- Inicio de sesión y registro (Complejidad CRÍTICA – 8 Puntos de Historia)	Diseñar widget de inicio de sesión y registro	1 Punto de Historia	21 de febrero	21 de febrero
	Crear formulario de inicio de sesión y registro tradicional	3 Puntos de Historia	21 de febrero	24 de febrero
	Integrar "Iniciar sesión con Google"	2 Puntos de Historia	24 de febrero	25 de febrero
	Integrar "Iniciar sesión con GitHub"	2 Puntos de Historia	25 de febrero	26 de febrero
HdU 1.3.- Control de Usuarios y Comunidad (Complejidad MEDIA – 3 Puntos de Historia)	Crear navegación	1 Punto de Historia	26 de febrero	26 de febrero
	Crear página de inicio	1 Punto de Historia	27 de febrero	28 de febrero
	Crear página de Comunidad y Perfil	1 Punto de Historia	28 de febrero	1 de marzo
HdU 2.1.- Sistema de Experiencia y Niveles (Complejidad CRÍTICA – 8 Puntos de Historia)	Diseñar un sistema de niveles	2 Puntos de Historia	1 de marzo	3 de marzo
	Implementar el sistema de experiencia y niveles en la aplicación a nivel servidor	3 Puntos de Historia	3 de marzo	5 de marzo
	Implementar el sistema de experiencia y niveles en la aplicación a nivel cliente	3 Puntos de Historia	5 de marzo	7 de marzo
HdU 2.2.- Banco de Logros (Complejidad MAYOR – 5 Puntos de Historia)	Implementar el sistema de reparto de logros en la aplicación	2 Puntos de Historia	7 de marzo	8 de marzo
	Crear la sección "Logros" de la aplicación	3 Puntos de Historia	8 de marzo	10 de marzo

Cuadro 4.1: Estimación temporal del sprint 1

Como se puede ver en la estimación, en este primer sprint se abarcan las dos primeras Épicas de Usuario, englobando un total de 27 Puntos de Historia en su alcance. Las tareas más complejas tienen 3 Puntos de Historia mientras que las más sencillas tienen solamente uno.

Sprint 2

SPRINT 2 – Épica III (11 de marzo – 29 de marzo)				
ALCANCE DEL SPRINT	21 Puntos de Historia			
TAREAS:				
Historia de Usuario	**Tarea**	**Puntos de Historia estimados**	**Inicio**	**Fin**
HdU 3.1.- Creación y Gestión de Retos (Complejidad CRÍTICA – 8 Puntos de Historia)	Modelar los retos como estructura de datos	2 Puntos de Historia	11 de marzo	12 de marzo
	Crear formulario de creación de retos	3 Puntos de Historia	12 de marzo	14 de marzo
	Validar formulario de creación de retos	3 Puntos de Historia	15 de marzo	17 de marzo
HdU 3.2.- Consulta y Acceso a Retos (Complejidad MAYOR – 5 Puntos de Historia)	Recibir listado de retos del servidor en el cliente	2 Puntos de Historia	17 de marzo	18 de marzo
	Mostrar listado de retos en el formato indicado (mosaico) sin romper la estética de la plataforma y de forma reactiva	3 Puntos de Historia	19 de marzo	21 de marzo
HdU 3.3.- Participación en Retos (Complejidad CRÍTICA – 8 Puntos de Historia)	Implementar la lógica de ejecución del código en modo sandbox en el servidor	3 Puntos de Historia	22 de marzo	24 de marzo
	Implementar la lógica de ejecución del código en modo sandbox en el cliente	3 Puntos de Historia	25 de marzo	27 de marzo
	Crear la pantalla de participación en retos, que emulará el comportamiento de un editor de código (IDE	2 Puntos de Historia	28 de marzo	29 de marzo

Cuadro 4.2: Estimación temporal del sprint 2

En este segundo sprint, se cubre una única Épica de Usuario, con un alcance total de 21 Puntos de Historia. Las tareas muestran una complejidad uniforme, entre 2 y 3 Puntos de Historia por tarea.

Sprint 3

SPRINT 3 – Épica IV (30 de marzo – 17 de abril)				
ALCANCE DEL SPRINT	21 Puntos de Historia			
TAREAS:				
Historia de Usuario	**Tarea**	**Puntos de Historia estimados**	**Inicio**	**Fin**
HdU 4.1.- Creación y despliegue del token (Complejidad CRÍTICA – 8 Puntos de Historia)	Crear token mediante Solidity	5 Puntos de Historia	30 de marzo	2 de abril
	Probar token mediante la herramienta Remix IDE	2 Puntos de Historia	3 de abril	4 de abril
	Desplegar token en la red de prueba Ropsten	1 Punto de Historia	5 de abril	5 de abril
HdU 4.2.- Gestión de Talentos (Complejidad CRÍTICA – 8 Puntos de Historia)	Integrar token en la aplicación mediante Web3	5 Puntos de Historia	6 de abril	10 de abril
	Permitir al usuario leer su balance y al administrador generar TAL	3 Puntos de Historia	11 de abril	13 de abril
HdU 4.3.- Banco (Complejidad MAYOR – 5 Puntos de Historia)	Crear formulario de solicitud de TAL	3 Puntos de Historia	14 de abril	16 de abril
	Crear botón de compra de Pases	1 Punto de Historia	16 de abril	16 de abril
	Crear panel de transacciones de TAL	1 Punto de Historia	17 de abril	17 de abril

Cuadro 4.3: Estimación temporal del sprint 3

En el tercer sprint, al igual que en el anterior, solamente se cubre una Épica de Usuario, con el mismo alcance de sprint: 21 Puntos de Historia. La complejidad de las tareas aquí es menos uniforme, existiendo dos tareas de complejidad mayor (5 Puntos de Historia) a la vez que existen tareas de complejidad trivial (1 Punto de Historia)

Sprint 4

SPRINT 4 – Épica V (18 de abril – 6 de mayo)				
ALCANCE DEL SPRINT	21 Puntos de Historia			
TAREAS:				
Historia de Usuario	**Tarea**	**Puntos de Historia estimados**	**Inicio**	**Fin**
HdU 5.1.- Creación y Gestión de Eventos (Complejidad CRÍTICA – 8 Puntos de Historia)	Modelar los eventos como estructura de datos	2 Puntos de Historia	18 de abril	19 de abril
	Crear formulario de creación de eventos	3 Puntos de Historia	19 de abril	21 de abril
	Validar formulario de creación de eventos	3 Puntos de Historia	22 de abril	24 de abril
HdU 5.2.- Consulta y Acceso a Eventos (Complejidad MAYOR – 5 Puntos de Historia)	Recibir listado de eventos del servidor en el cliente	2 Puntos de Historia	25 de abril	26 de abril
	Mostrar listado de eventos en el formato indicado (mosaico) sin romper la estética de la plataforma y de forma reactiva	3 Puntos de Historia	27 de abril	29 de abril
HdU 5.3.- Participación en Eventos (Complejidad CRÍTICA – 8 Puntos de Historia)	Implementar la lógica de candidaturas en el lado servidor	3 Puntos de Historia	30 de abril	2 de mayo
	Implementar la lógica de candidaturas en el lado cliente	3 Puntos de Historia	2 de mayo	4 de mayo
	Crear la pantalla de envío y recepción de candidaturas	2 Puntos de Historia	5 de mayo	6 de mayo

Cuadro 4.4: Estimación temporal del sprint 4

El cuarto sprint sigue la tónica similar a los dos anteriores, con una única Épica de Usuario cubierta y 21 Puntos de Historia como alcance de sprint. La complejidad de las tareas es uniforme al igual que en el segundo sprint, existiendo tareas de 2 y de 3 Puntos de Historia.

Sprint 5

SPRINT 5 – Épicas VI y VII (7 de mayo – 25 de mayo)				
ALCANCE DEL SPRINT	21 Puntos de Historia			
TAREAS:				
Historia de Usuario	**Tarea**	**Puntos de Historia estimados**	**Inicio**	**Fin**
HdU 6.1.- Creación y Gestión de Talleres (Complejidad MAYOR – 5 Puntos de Historia)	Modelar los talleres como estructura de datos	1 Punto de Historia	7 de mayo	7 de mayo
	Crear formulario de creación de talleres	2 Puntos de Historia	7 de mayo	8 de mayo
	Validar formulario de creación de talleres	2 Puntos de Historia	9 de mayo	10 de mayo
HdU 6.2.- Consulta y Acceso a Talleres (Complejidad MEDIA – 3 Puntos de Historia)	Recibir listado de talleres del servidor en el cliente	1 Punto de Historia	10 de mayo	10 de mayo
	Mostrar listado de talleres en el formato indicado (mosaico) sin romper la estética de la plataforma, y de forma reactivs	2 Puntos de Historia	11 de mayo	12 de mayo
HdU 7.1.- Interacción con el usuario (Complejidad MAYOR – 5 Puntos de Historia)	Crear un sistema de notificaciones en el cliente en función de eventos y con carácter reactivo	3 Puntos de Historia	13 de mayo	15 de mayo
	Mostrar las notificaciones mediante diversos componentes, como cuadros de diálogo, popups…	2 Puntos de Historia	16 de mayo	18 de mayo
HdU 7.2.- Versión móvil (Complejidad CRÍTICA – 8 Puntos de Historia)	Crear una versión móvil del cliente de la aplicación web	5 Puntos de Historia	19 de mayo	23 de mayo
	Implementar que el sistema determine qué versión servir al usuario en función del tamaño de su pantalla	3 Puntos de Historia	23 de mayo	25 de mayo

Cuadro 4.5: Estimación temporal del sprint 5

El quinto sprint vuelve a cubrir dos Épicas de Usuario y tiene una distribución más dispare de la complejidad de las tareas, existiendo un pico de esfuerzo en la primera tarea de la última tarea, con 5 Puntos de Historia a la vez que existen tareas de complejidad trivial.

Sprint 6

SPRINT 6 – Épica VIII(26 de mayo – 13 de junio)				
ALCANCE DEL SPRINT	21 Puntos de Historia			
TAREAS:				
Historia de Usuario	**Tarea**	**Puntos de Historia estimados**	**Inicio**	**Fin**
HdU 8.1.- Pruebas Unitarias (Complejidad SUPERIOR A CRÍTICA - 13 Puntos de Historia	Realizar pruebas generales de usabilidad	3 Puntos de Historia	26 de mayo	28 de mayo
	Realizar pruebas de Retos	3 Puntos de Historia	29 de mayo	31 de mayo
	Realizar pruebas de Eventos	3 Puntos de Historia	1 de junio	3 de junio
	Realizar pruebas del Banco	2 Puntos de Historia	4 de junio	5 de junio
	Realizar pruebas de Talleres	1 Punto de Historia	5 de junio	6 de junio
	Realizar pruebas de Logros	1 Punto de Historia	6 de junio	6 de junio
HdU 8.2.- Despliegue y paso a producción (Complejidad CRÍTICA - 8 Puntos de Historia)	Desplegar servidor a Heroku	3 Puntos de Historia	7 de junio	9 de junio
	Desplegar cliente a Netlify	3 Puntos de Historia	10 de junio	12 de junio
	Enlazar cliente y servidor con sus respectivos endpoints	2 Puntos de Historia	12 de junio	13 de junio

Cuadro 4.6: Estimación temporal del sprint 6

El último sprint planificado cubre la última Épica de Usuario elicitada, que alberga la única Historia de Usuario con complejidad superior a 8 Puntos de Historia, estimándose en 13 Puntos de Historia y conteniendo seis tareas que oscilan entre 1 y 3 Puntos de Historia cada una.

4.2.4. Aproximación a la estimación de costes

*(**Nota: w**Para la elaboración de todo este epígrafe, si bien explico la metodología que he seguido a nivel personal, para desarrollarla ha sido muy útil el apoyo en el artículo del blog de la empresa tecnológica G-Data [14], así como en el artículo publicado por los investigadores en Matemáticas Mitre-Hernández, Ortega-Martínez y Cuauhtémoc [10])*

Como he señalado anteriormente, la estimación de costes en proyectos que siguen las metodologías ágiles es compleja por la propia definición de la métrica de Puntos de Historia y su nula traducibilidad a horas de trabajo, existen algunas aproximaciones para estimar los costes. Es importante señalar unos objetivos que debe cumplir toda estimación de costes, incluyendo la estimación ágil[9]:

- Los costes deben tener un carácter transparente para que puedan conocerlos cualquier miembro del equipo de trabajo.
- Los costes deben ser calculados de forma frecuente y automática, por tanto es interesante obtener una fórmula a partir de la cual extraer los costes.
- El cálculo de costes debe ser comprensible.

[9]Objetivos extraídos del artículo del blog de G-Data[14] y traducidos al castellano

Partiendo de estos objetivos como premisa, se ha buscado una aproximación a la estimación de costes lo más precisa posible. En el ya citado artículo del blog de G-Data [10] plantean una aproximación basándose en las horas de los trabajadores por sprint, el coste por hora medio de un trabajador, la velocidad del equipo y el número de Puntos de Historia.

Esta es la fórmula de la función Coste:

$$C_{Sprint} = \frac{Horastotales_{Sprint} * Salario_{/hora}}{Velocidad_{media}} * PH_{Sprint} \tag{4.1}$$

Si bien es una aproximación que considero bastante precisa para estimar los costes del personal en el desarrollo de un proyecto ágil, lo cierto es que en este proyecto no existe ningún equipo, ya que es un proyecto individual, por lo que se realizará la aproximación para un único trabajador, lo cual es irreal en un proceso de desarrollo real.

Se calculará el coste por sprint teniendo en cuenta que la velocidad media del proyecto es la media entre los alcances de todos los sprints, ya que la velocidad no es más que el número de puntos de historia que se resuelven en ese sprint.

Además, se tomará una jornada laboral estándar (40 horas semanales) y un salario por hora medio de un ingeniero informático, 12.82€ por hora de trabajo.[10].

- Un único desarrollador trabaja 8 horas diarias. Suponiendo que en cada sprint de 18 días hay 4 días no laborables, el desarrollador trabaja **112 horas por *sprint***
- Existe un sprint con alcance de 26 Puntos de Historia y cinco sprints con alcance de 21 Puntos de Historia. Adicionalmente, existe una **fase conceptual** conformada por 36 días de los cuales 8 son laborables, por tanto 24 días laborables corresponderán a la fase conceptual que no cuenta con los Puntos de Historia como métrica.
- Se supone que en cada sprint se completará su alcance, pero después en el balance veremos si efectivamente se ha cumplido.

Así pues, por tanto:

[10]Fuente: `https://es.talent.com/salary?job=ingeniero+informático`

Sprint	*Velocidad esperada*	*Coste estimado*
Fase de concepción	192 horas laborables * 12,82 €/hora	2461,44 e
Sprint 1	26 Puntos de Historia	1707,44 €
Sprint 2	21 Puntos de Historia	1379,09 €
Sprint 3	21 Puntos de Historia	1379,09 €
Sprint 4	21 Puntos de Historia	1379,09 €
Sprint 5	21 Puntos de Historia	1379,09 €
Sprint 6	21 Puntos de Historia	1379,09 €
Coste total	**11064,33 €**	

Cuadro 4.7: Aproximación a la estimación de los costes por sprint de trabajo según los Puntos de Historia

4.3. Presupuesto

A los costes estimados anteriormente hay que sumar un **presupuesto** de toda la infraestructura y herramientas necesarias para trabajar y desplegar el proyecto. Antes de adjuntar el presupuesto, se incluyen unos matices con el fin de facilitar su comprensión:

- Al ser estudiante, he podido aprovechar planes de estudiante gratuitos para todas estas herramientas, por lo que realmente no me han supuesto un coste, pero es importante incluir estos gastos en el presupuesto del proyecto ya que, si lo desarrollara un equipo de profesionales, no podrían beneficiarse de las ventajas para estudiantes.

- Los gastos que conllevaría la presencialidad tales como alquiler de oficinas, luz o gas se han obviado, ya que en las condiciones de pandemia mundial en las que estamos actualmente, el teletrabajo es una realidad en los procesos de desarrollo de *software*. Sin embargo, sí se ha tenido en cuenta la cesión de un ordenador portátil ya que las empresas suelen proporcionarlos al contratar a un empleado en remoto.

- Las **herramientas gratuitas y *open source*** se obvian en el presupuesto, pero no por ello no han sido utilizadas. El caso principal es el de **Visual Studio Code** como editor, tanto a nivel código, como a nivel documentación, todo el stack **MEVN** utilizado, o **Netlify** como proveedor de infraestructura *cloud* para contenido estático, el cual es utilizado para desplegar el lado cliente de nuestra aplicación.

- Los gastos se adjuntan tanto como nos viene dados por el proveedor, como **prorrateados** para los seis meses que dura el proceso de desarrollo, es decir, que si el precio de una herramienta es mensual, se multiplicará por 6, mientras que si es anual, se dividirá por 2.

Este es el presupuesto calculado:

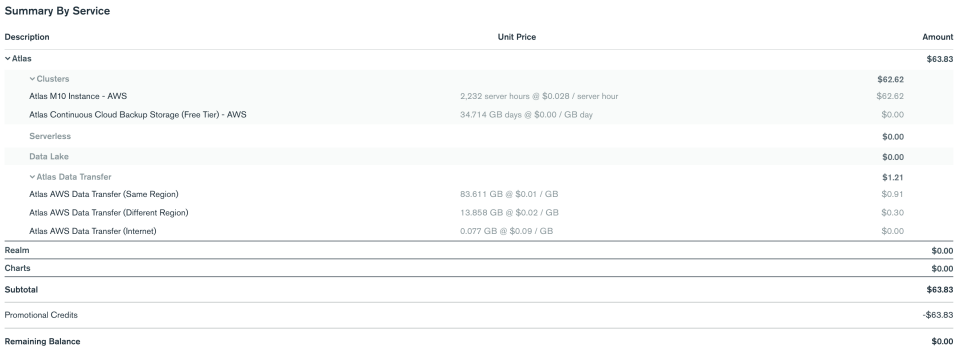
Categoría	Items		Subtotal
	Nombre	Precio y prorrateo	
Infraestructura cloud	Servidor *"Hobby Dyno"* de Heroku	$7/mes **$42** **35 €**	**375,16 €**
	Cluster *"Atlas M10"* de MongoDB	$63/mes **$378** **319,9 €**	
	Cuenta de GitHub Team para poder tener repositorios privados	$4/mes **$24** **20,26 €**	
Herramientas hardware y software utilizadas	Adobe Illustrator para el diseño de UX/UI	$240/año **$120** **101,30 €**	**1164,3 €**
	Microsoft 365 Empresa Estándar	10,50€/mes **63 €**	
	Cesión de ordenador portátil de trabajo	**1000€**	
Otros gastos	Compra del dominio https://ecodium.dev	$12 **10,13 €**	**10,13 €**
Coste del desarrollo del proyecto (estimado)	**11064,33 €**		
COSTE TOTAL	**12613,92 €**		

Cuadro 4.8: Presupuesto estimado del proyecto

Todos los precios han sido extraídos a fecha junio de 2021 de las webs corporativas de cada proveedor: Heroku[11], GitHub[12], Adobe[13], Microsoft[14] y Google Domains[15].

Los gastos de MongoDB se basan en las propias facturas que emite el proveedor, las cuales han sido pagadas por bonos de 200 dólares que otorga MongoDB a estudiantes. Un ejemplo de factura:

[11] `https://www.heroku.com/pricing`
[12] `https://github.com/pricing`
[13] `https://www.adobe.com/es/creativecloud/plans.html`
[14] `https://www.microsoft.com/es-es/microsoft-365/business/compare-all-microsoft-365-business-products`
[15] `https://domains.google.com`



Summary By Service

Description	Unit Price	Amount
Atlas		$63.83
Clusters		$62.62
Atlas M10 Instance - AWS	2,232 server hours @ $0.028 / server hour	$62.62
Atlas Continuous Cloud Backup Storage (Free Tier) - AWS	34.714 GB days @ $0.00 / GB day	$0.00
Serverless		$0.00
Data Lake		$0.00
Atlas Data Transfer		$1.21
Atlas AWS Data Transfer (Same Region)	83.611 GB @ $0.01 / GB	$0.91
Atlas AWS Data Transfer (Different Region)	13.858 GB @ $0.02 / GB	$0.30
Atlas AWS Data Transfer (Internet)	0.077 GB @ $0.09 / GB	$0.00
Realm		$0.00
Charts		$0.00
Subtotal		$63.83
Promotional Credits		-$63.83
Remaining Balance		$0.00

Figura 4.2: Ejemplo de factura real de MongoDB Atlas

4.4. Balance de planificación del proyecto

A nivel general, la planificación temporal ha resultado bastante acertada, si bien ha resultado realmente difícil compaginarlo con el trabajo diario en la empresa de prácticas, los cursos y la vida personal. No obstante, la existencia de **bugs** en el código, funcionalidades que no terminaban de realizarse de forma óptima y despliegues fallidos, creó la necesidad de añadir un ***sprint extra***, un séptimo sprint no planificado y por tanto, sin historias de usuario asociadas. Simplemente una ventana temporal, de 18 días igual que los sprints convencionales, dedicada al *bugfix* o resolución de bugs, a reintentar despliegues y a limpiar el código.

Además, aunque no forme parte del Proceso de Desarrollo en sí, se ha incluido el período de documentación, en el que se maqueta este documento en sí a partir de pequeños documentos que fui confeccionando en cada fase del desarrollo, y se les dota a todos del mismo formato y estilo. Por último, fue necesaria una **fase de estabilización** del entorno para cargar datos de prueba, eliminar datos basura de la base de datos que se habían generado con el desarrollo y a estabilizar el entorno *cloud* donde se despliega la plataforma.

El calendario temporal final, por tanto, dista aproximadamente 3 meses del planificaco inicialmente:

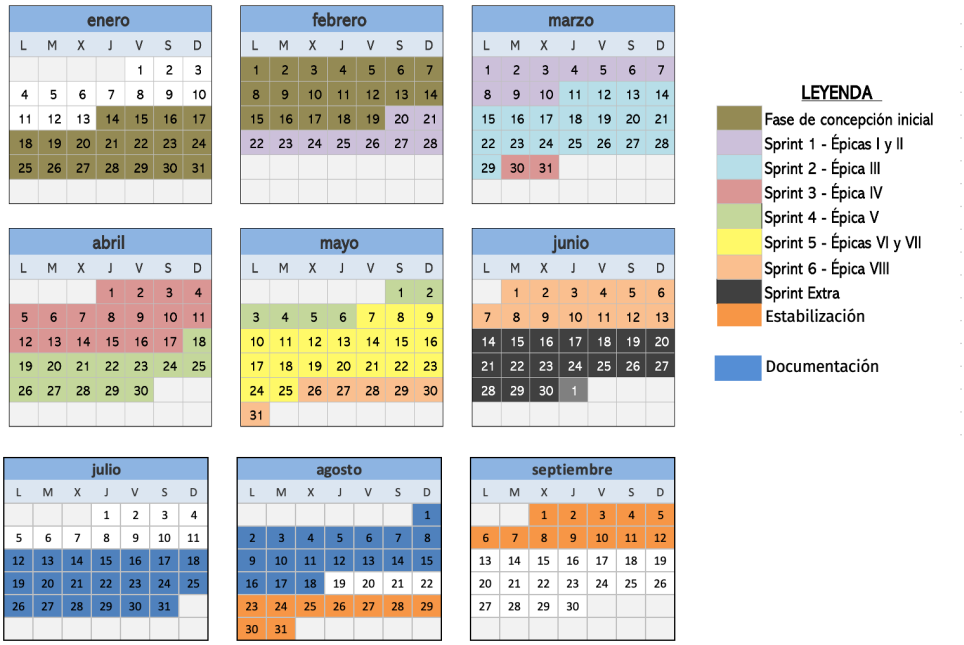


Figura 4.3: Calendario final de la organización temporal del proceso de desarrollo del proyecto

Capítulo 5

Análisis Funcional

En este capítulo se describirá y detallará todo lo relacionado con el **análisis funcional** de la plataforma, desde el punto de vista de las metodologías ágiles que sigue todo el proyecto. Para ello, se comenzará identificando los distintos **roles** que puede tener el usuario registrado en la plataforma. Posteriormente, partiendo de dichos roles, se llevará a cabo la **especificación de requisitos** mediante el modelo de **Épicas e Historias de Usuario**.

5.1. Identificación de Roles

En la fase de análisis y elicitación de requisitos de un proceso de desarrollo software siguiendo metodologías clásicas, lo primero que se define son los **actores**, que realizan una serie de **casos de uso** para abarcar los distintos flujos que componen el sistema a analizar.

En este proyecto, al regirse por metodologías ágiles, hablaremos de **roles**, que en última instancia se corresponden con los actores que hacen uso de la aplicación, pero no realizan casos de uso, sino que se encuentran en los enunciados de las historias de usuario planteadas.

Estos son, por tanto, los cuatro roles de usuario registrado que disponemos en ECodium:

- **Jugadores:** son el rol principal de la plataforma. Se dedican a resolver retos, participar en eventos, inscribirse a talleres y aspirar por ascender puestos en la Liga.
- **Organizadores:** se dedican a organizar eventos, talleres o retos. Son colaboradores de la plataforma.
- **Ojeadores:** pueden ver, sin pagar Pases de Acceso, cómo se desenvuelven los distintos eventos y talleres, así como poder contactar usuarios. Están pensados para utilizar ECodium como vía de encontrar talento, algo que puede ser útil a las empresas.

- **Administrador:** es el desarrollador de la plataforma. Tiene todos los permisos y, en principio, es único.

5.2. Especificación de Requisitos

5.2.1. Introducción y visión global

El proyecto se descompone en ocho Épicas de Usuario, y desde un punto de vista global, se resumen en la sigueinte tabla:

VISIÓN GLOBAL DEL ANÁLISIS FUNCIONAL DE ECODIUM			
Épica de Usuario	**Funcionalidad Asociada**	**Historias de Usuario Asociadas**	**Complejidad**
Épica I	Inicio de sesión, registro y Comunidad de usuarios	3	13 Puntos de Historia
Épica II	Gamificación, sistema de niveles, logros y recompensas	2	13 Puntos de Historia
Épica III	Retos: consulta, creación, edición y participación	3	21 Puntos de Historia
Épica IV	Integración con la red Ethereum y uso de un token ERC20	3	21 Puntos de Historia
Épica V	Eventos: consulta, creación, edición y participación	3	21 Puntos de Historia
Épica VI	Talleres: consulta, creación, edición y visualización	2	8 Puntos de Historia
Épica VII	Optimización de la usabilidad de la aplicación	2	13 Puntos de Historia
Épica VIII	Pruebas y despliegue	2	21 Puntos de Historia

Figura 5.1: Épicas de Usuario del proyecto desde una visión global

5.2.2. Épica I: Creación del proyecto y páginas asociadas

La primera épica a tratar se fundamenta en la creación del proyecto dentro de la arquitectura elegida (*stack MEVN* con la capa *Quasar Framework*), así como la puesta a punto de una serie de páginas iniciales que constituyen la base de las páginas funcionales que vendrán después, en especial referentes a la función social de la plataforma: la Comunidad.

HdU 1.1. Creación del proyecto y landing page

La primera Historia de Usuario hace referencia al proceso de creación a nivel estructural del proyecto y la materialización del mismo mediante una primera página: una landing page para usuarios invitados.

HDU 1.1.- CREACIÓN DEL PROYECTO Y LANDING PAGE			
DESCRIPCIÓN:	COMO usuario invitado QUIERO visualizar una landing page PARA conocer la aplicación sin necesidad de registrarme		
DEPENDENCIAS:	No tiene	**COMPLEJIDAD:**	**Mínima** (2 P.H)
CRITERIOS DE ACEPTACIÓN:			
- El proyecto deberá estar constituido en arquitectura, tanto en cliente como en servidor - La landing page deberá ser la página de inicio del usuario invitado			
TAREAS:			
- Crear proyecto tanto a nivel cliente como a nivel servidor (1 Puntos de Historia) - Crear landing page (1 Puntos de Historia)			

Figura 5.2: Historia de Usuario 1.1.-Creación del proyecto y landing page

Esta Historia de Usuario resulta la base del resto de Historias de Usuario de esta Épica, ya que sin el proyecto creado y funcionando correctamente, no se podría continuar el proceso de desarrollo del sistema. Por tanto, aunque su complejidad haya sido descrita como "Mínima"porque si bien no se trata de una tarea sustancialmente difícil, la prioridad de la tarea será la más alta posible ya que es dependencia del resto.

HdU 1.2. Inicio de sesión y registro social

Esta Historia de Usuario describe el proceso de implementación de un control de acceso de usuarios mediante el sistema de inicio de sesión, registro y almacenamiento de credenciales, ya vistos en otras muchas plataformas web.

HDU 1.2.- INICIO DE SESIÓN Y REGISTRO SOCIAL			
DESCRIPCIÓN:	COMO usuario invitado QUIERO registrarme o iniciar sesión tanto de forma tradicional como de forma social utilizando mi cuenta de Google o GitHub PARA formar parte de la aplicación		
DEPENDENCIAS:	HdU 1.1.	**COMPLEJIDAD**	**Crítica (8 P.H)**
CRITERIOS DE ACEPTACIÓN:			
- La landing page deberá mostrar un formulario de inicio de sesión y registro que cuente con todos los mecanismos disponibles en la operativa de la aplicación para tal fin - Dicho formulario deberá contar con una validación tanto en el lado cliente como en el lado servidor - En el modal, por tanto, deberán existir botones para el inicio de sesión con Google, con GitHub y sin necesidad de inicio social - Para llevar a cabo un inicio de sesión social, el usuario primero deberá haber sido registrado socialmente. - En caso de registro social, se tomará como mail el mail asociado a la cuenta social enlazada.			
TAREAS:			
- Diseñar widget de inicio de sesión y registro (1 Puntos de Historia) - Crear formulario de inicio de sesión y registro tradicional (3 Puntos de Historia) - Integrar "Iniciar sesión con Google" (2 Puntos de Historia) - Integrar "Iniciar sesión con GitHub" (2 Puntos de Historia)			

Figura 5.3: Historia de Usuario 1.2.- Inicio de sesión y registro social

Nuevamente, al igual que en la anterior, esta Historia de Usuario tiene una prioridad elevada debido a que de ella depende la siguiente. Sin embargo, la complejidad de esta Historia de Usuario ha sido etiquetada como Crítica ya que las tareas son bastante complejas y delicadas.

HdU 1.3. Control de Usuarios y Comunidad

Esta Historia de Usuario define el comportamiento de dos de las páginas principales del lado social de la plataforma: Perfil y Comunidad.

HDU 1.3.- CONTROL DE USUARIOS Y COMUNIDAD			
DESCRIPCIÓN:	COMO usuario registrado QUIERO acceder a un PARA conocer el rol de usuario asociado a mi perfil y acceder a la operativa de dicho rol		
DEPENDENCIAS:	HdU 1.2.	**COMPLEJIDAD**	**Media** (3 P.H)
CRITERIOS DE ACEPTACIÓN:			
- El sistema deberá distinguir cuatro roles: jugador, ojeador, organizador y administrador. Esta distinción se llevará a cabo tanto en el lado cliente como en el lado servidor. - Cualquier usuario podrá buscar a cualquier usuario no administrador y acceder a su perfil público, donde conocer además de su rol, las estadísticas de jugador si es jugador, el número de cursos si es mentor, el número de retos si es master. - Los ojeadores podrán añadir jugadores a favoritos para poder contactarles si lo desean. - Solamente el administrador podrá cambiar el rol de un usuario, así como eliminar perfiles.			
TAREAS:			
- Crear navegación (1 Puntos de Historia) - Crear página de inicio (1 Puntos de Historia) - Crear página de Comunidad y Perfil y sus respectivos modales (1 Puntos de Historia)			

Figura 5.4: Historia de Usuario 1.3.- ontrol de Usuarios y Comunidad

5.2.3. Épica II: Gamificación

HdU 2.1.- Sistema de experiencia y niveles

HDU 2.1.- SISTEMA DE EXPERIENCIA Y NIVELES			
DESCRIPCIÓN:	COMO jugador QUIERO un sistema de experiencia y niveles PARA poder progresar en ECodium y así tener un aliciente para seguir mejorando		
DEPENDENCIAS:	-	**COMPLEJIDAD**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- Por su carácter gamificado, la aplicación deberá contar en su operativa con un sistema de puntos de experiencia (XP) que permita al jugador tanto subir de nivel como conseguir TAL - Los XP se conseguirán consiguiendo logros, resolviendo retos, conquistándolos, participando en talleres, participando en eventos y ganando los mismos. - Los XP se podrán perder en algunos eventos. - El sistema de niveles deberá ser justo y proporcional, así como depender de los XP del usuario, basándose por tanto en una fórmula matemática			
TAREAS:			
- Diseñar un sistema de niveles (2 Puntos de Historia) - Implementar el sistema de experiencia y niveles en la aplicación a nivel servidor (3 Puntos de Historia) - Implementar el sistema de experiencia y niveles en la aplicación a nivel cliente (3 Puntos de Historia)			

Figura 5.5: Historia de Usuario 2.1.- Sistema de experiencia y niveles

HdU 2.2.- Banco de logros

HDU 2.2.- EL BANCO DE LOGROS			
DESCRIPCIÓN:	COMO administrador QUIERO un banco de logros PARA que los usuarios canjeen los logros ya desbloqueados por EXPERIENCIA		
DEPENDENCIAS:	-	**COMPLEJIDAD**	**Mayor** (5 P.H)
CRITERIOS DE ACEPTACIÓN:			
- Por su carácter gamificado, la aplicación deberá contar en su operativa con un banco de logros disponible para que los usuarios puedan desbloquearlos y así conseguir puntos de experiencia de manera adicional. - Una vez que un usuario obtiene un logro, no puede perderlo, ni caduca en ningún caso. - Los logros no pueden ser editados, pero sí borrados. - La plataforma contará con una sección dedicada a los Logros, donde el usuario Jugador pueda ver de una manera visual qué logros tiene bloqueados y cuáles desbloqueados. - Solamente los Administradores pueden crear logros. Los Mentores y Masters ni tienen logros ni pueden crearlos.			
TAREAS:			
- Implementar el sistema de reparto de logros en la aplicación (2 Puntos de Historia) - Crear la sección "Logros" de la aplicación (3 Puntos de Historia)			

Figura 5.6: Historia de Usuario 1.2.- Inicio de sesión y registro social

5.2.4. Épica III: Retos

Historia de Usuario 3.1.- Creación y gestión de Retos

HDU 3.1.- CREACIÓN Y GESTIÓN DE RETOS			
DESCRIPCIÓN:	COMO usuario no jugador QUIERO gestionar retos PARA añadirlos al catálogo de ECodium		
DEPENDENCIAS:	-	**COMPLEJIDAD**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- Los retos deberán contar con un título, un enunciado, una recompensa en EXP y un banco de pruebas donde cada prueba es descrita mediante el formato (input, output, type), donde input es la entrada de la prueba, output la salida esperada y type el tipo de prueba: Pública (el jugador conocerá tanto input como output), Oculta (el jugador conocerá input pero no output) y Privada (el jugador no conocerá la prueba, pero sí sabrá de su existencia) - Adicionalmente, podrá tener descripción a modo de pistas, y etiquetas para su filtrado. Asimismo, el conquistador de un reto será el primer jugador que logre resolverlo. - Se deberá validar tanto en el lado cliente como en el lado servidor el modal de creación de retos. - Un usuario puede editar y/o eliminar los retos que ha creado él mismo. La edición se llevará a cabo mediante el propio modal. - Cada reto será accesible mediante un permalink (enlace permanente) e identificable mediante un ID.			
TAREAS:			
- Modelar los retos como estructura de datos (2 Puntos de Historia) - Crear formulario de creación de retos (3 Puntos de Historia) - Validar formulario de creación de retos (3 Puntos de Historia)			

Figura 5.7: Historia de Usuario 3.1.- Creación y gestión de Retos

Historia de Usuario 3.2.- Consulta y acceso a Retos

HDU 3.2.- CONSULTA Y ACCESO A RETOS			
DESCRIPCIÓN:	COMO jugador QUIERO ver los retos disponibles PARA decidir si intento resolver alguno		
DEPENDENCIAS:	3.1.	**COMPLEJIDAD**	**Mayor** (5 P.H)
CRITERIOS DE ACEPTACIÓN:			
- El listado de retos será una página dedicada a ello, donde aparecerán los retos en formato mosaico, dividiéndose en páginas de máximo 9 retos. - El listado de retos podrá ser filtrable por etiquetas, por estado ('Resuelto' o 'Disponible') y por si han sido o no conquistados. Asimismo, el listado de retos contará con un buscador.			
TAREAS:			
- Recibir listado de retos del servidor en el cliente (2 Puntos de Historia) - Mostrar listado de retos en el formato indicado (mosaico) sin romper la estética de la plataforma, y de forma reactiva (3 Puntos de Historia)			

Figura 5.8: Historia de Usuario 3.2.- Consulta y acceso a Retos

Historia de Usuario 3.3.- Participación en Retos

HDU 3.3.- PARTICIPACIÓN EN RETOS			
DESCRIPCIÓN:	COMO jugador QUIERO participar en retos PARA mejorar como jugador		
DEPENDENCIAS:	3.2.	**COMPLEJIDAD**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- Los retos se resolverán desde el propio navegador, en el lenguaje JavaScript (en el alcance actual, no se admiten más lenguajes) - Para garantizar la seguridad, se utilizará un mecanismo de sandbox, con el fin de que el código que el usuario ejecute en los retos no interfiera con el código de la aplicación.			
TAREAS:			
- Implementar la lógica de ejecución de código en modo sandbox en el servidor (3 Puntos de Historia) - Implementar la lógica de ejecución de código en modo sandbox en el cliente (3 Puntos de Historia) - Crear la pantalla de participación en retos, que emulará el comportamiento de un editor de código (IDE) (2 Puntos de Historia)			

Figura 5.9: Historia de Usuario 3.3.- Participación en Retos

5.2.5. Épica IV: Integración con Ethereum

Historia de Usuario 4.1.- Creación y despliegue del token

HDU 4.1.- CREACIÓN Y DESPLIEGUE DEL TOKEN			
DESCRIPCIÓN:	COMO administrador QUIERO crear un token de Ethereum ERC20 PARA posteriormente integrarlo en la plataforma ECodium		
DEPENDENCIAS:	-	**COMPLEJIDAD**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- El token (TALENTO) deberá pertenecer a la red Ethereum y cumplir con el estándar ERC20 - El token deberá permitir la creación de nuevos tokens (minting) hasta un tope. - El token (TALENTO) deberá estar desplegado y existir en la red de prueba Ropsten, de modo que si un usuario dispone de una wallet digital compatible con Ethereum – por ejemplo, Metamask -, ésta reconozca el token.			
TAREAS:			
- Crear token mediante Solidity (5 Puntos de Historia) - Probarlo mediante la herramienta Remix IDE (2 Puntos de Historia) - Desplegar token en la red de prueba Ropsten (1 Puntos de Historia)			

Figura 5.10: Historia de Usuario 4.1.- Creación y despliegue del token

Historia de Usuario 4.2.- Gestión de Talentos

HDU 4.2- GESTIÓN DE TALENTOS			
DESCRIPCIÓN:	COMO administrador QUIERO integrar un token de Ethereum ERC20 PARA disponer de un criptoactivo como una de las recompensas de mi plataforma gamificada		
DEPENDENCIAS:	4.1.	**COMPLEJIDAD**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- El usuario NO podrá depositar TAL a través de la aplicación, sino que los deberá ir ganando y por tanto generando - El usuario se hará cargo del coste de transacción (GAS) - El administrador podrá generar o quitar TAL con dos botones en su panel			
TAREAS:			
- Integrar token en la aplicación mediante Web3 (5 Puntos de Historia) - Permitir al usuario leer su balance y al administrador repartir TAL (3 Puntos de Historia)			

Figura 5.11: Historia de Usuario 4.2.- Gestión de Talentos

5.2.6. Épica V: Eventos

Historia de Usuario 5.1.- Creación y gestión de eventos

HDU 5.1.- CREACIÓN Y GESTIÓN DE EVENTOS			
DESCRIPCIÓN:	COMO usuario no jugador QUIERO gestionar eventos PARA añadirlos al catálogo de ECodium		
DEPENDENCIAS:	-	**COMPLEJIDAD**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- Los retos deberán contar con un título, unas recompensas en EXP, un precio de acceso en pases y un jurado que reparta los premios. - Adicionalmente, podrá tener descripción a modo de pistas, una imagen, y un conjunto de etiquetas para su filtrado. - Se deberá validar tanto en el lado cliente como en el lado servidor el modal de creación de eventos. - Un usuario puede editar y/o eliminar los eventos que ha creado él mismo. La edición se llevará a cabo mediante el propio evento. - Cada evento será accesible mediante un permalink (enlace permanente) e identificable mediante un ID.			
TAREAS:			
- Modelar los eventos como estructura de datos (2 Puntos de Historia) - Crear formulario de creación de eventos (3 Puntos de Historia) - Validar formulario de creación de eventos (3 Puntos de Historia)			

Figura 5.12: Historia de Usuario 5.1.- Creación y gestión de eventos

Historia de Usuario 5.2.- Consulta y acceso a eventos

HDU 5.2.- CONSULTA Y ACCESO A EVENTOS			
DESCRIPCIÓN:	COMO jugador QUIERO ver los eventos disponibles PARA decidir si intento apuntarme a alguno		
DEPENDENCIAS:	5.1.	**COMPLEJIDAD**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- El listado de eventos será una página dedicada a ello, donde aparecerán los retos en formato mosaico, dividiéndose en páginas de máximo 9 retos. - El listado de eventos podrá ser filtrable por etiquetas, por estado ('Cerrado o 'Abierto) o por precio. Asimismo, el listado de eventos contará con un buscador.			
TAREAS:			
- Recibir listado de eventos del servidor en el cliente (2 Puntos de Historia) - Mostrar listado de eventos en el formato indicado (mosaico) sin romper la estética de la plataforma, y de forma reactiva (3 Puntos de Historia)			

Figura 5.13: Historia de Usuario 5.2.- Consulta y acceso a eventos

Historia de Usuario 5.3.- Participación en eventos

HDU 5.3.- PARTICIPACIÓN EN EVENTOS			
DESCRIPCIÓN:	COMO jugador QUIERO participar en eventos PARA mejorar como jugador		
DEPENDENCIAS:	5.2.	**COMPLEJIDAD**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- La participación en eventos se llevará a cabo mediante candidaturas. Los jugadores podrán presentar candidaturas, una por usuario o por grupo de usuarios. En caso de ser colectivo, solamente lo entregará un jugador. - El coste de presentar candidaturas viene medido en Pases de Acceso. - Todas las candidaturas contarán, como mínimo, con un título y un repositorio de GitHub que contenga obligatoriamente un readme o documentación y el proyecto presentado. - Tras la validación del jurado, el usuario recibirá una notificación en la plataforma indicando que su candidatura ha ganado.			
TAREAS:			
- Implementar la lógica de candidaturas en el lado servidor (3 Puntos de Historia) - Implementar la lógica de candidaturas en el lado cliente (3 Puntos de Historia) - Crear la pantalla de envío y recepción de candidaturas (2 Puntos de Historia)			

Figura 5.14: Historia de Usuario 5.3.- Participación en eventos

5.2.7. Épica VI: Talleres

Historia de Usuario 6.1.- Creación y gestión de talleres

HDU 6.1.- CREACIÓN Y GESTIÓN DE TALLERES			
DESCRIPCIÓN:	COMO usuario no jugador QUIERO gestionar talleres PARA añadirlos al catálogo de ECodium		
DEPENDENCIAS:	-	**COMPLEJIDAD**	**Mayor** (5 P.H)
CRITERIOS DE ACEPTACIÓN:			
- Los talleres deberán contar con un título, unas recompensas en EXP/TAL, un precio de acceso en pases y un profesor que imparta los talleres. - Adicionalmente, podrá tener un temario, una imagen, y un conjunto de etiquetas para su filtrado. - Se deberá validar tanto en el lado cliente como en el lado servidor el modal de creación de talleres. - Un usuario puede editar y/o eliminar los talleres que ha creado él mismo. La edición se llevará a cabo mediante el propio taller. - Cada taller será accesible mediante un permalink (enlace permanente) e identificable mediante un ID.			
TAREAS:			
- Modelar los talleres como estructura de datos (1 Puntos de Historia) - Crear formulario de creación de talleres (2 Puntos de Historia) - Validar formulario de creación de talleres (2 Puntos de Historia)			

Figura 5.15: Historia de Usuario 6.1.- Creación y gestión de talleres

Historia de Usuario 6.2.- Consulta y acceso a talleres

HDU 6.2.- CONSULTA Y ACCESO A TALLERES			
DESCRIPCIÓN:	COMO jugador QUIERO ver los talleres disponibles PARA decidir si intento apuntarme a alguno		
DEPENDENCIAS:	6.1	**COMPLEJIDAD**	**Media** (3 P.H)
CRITERIOS DE ACEPTACIÓN:			
- El listado de talleres será una página dedicada a ello, donde aparecerán los retos en formato mosaico, dividiéndose en páginas de máximo 9 retos. - El acceso a talleres conlleva un coste en Pases de Acceso para el usuario - El listado de talleres podrá ser filtrable por etiquetas, por estado ('Cerrado o 'Abierto) o por precio. Asimismo, el listado de talleres contará con un buscador.			
TAREAS:			
- Recibir listado de talleres del servidor en el cliente (1 Puntos de Historia) - Mostrar listado de talleres en el formato indicado (mosaico) sin romper la estética de la plataforma, y de forma reactiva (2 Puntos de Historia)			

Figura 5.16: Historia de Usuario 6.2.- Consulta y acceso a talleres

5.2.8. Épica VII: Optimización de la Usabilidad

Historia de Usuario 7.1.- Sistema de Notificaciones

HDU 7.1.- SISTEMA DE NOTIFICACIONES			
DESCRIPCIÓN:	COMO usuario QUIERO disponer de avisos PARA enterarme si ocurre algo nuevo en ECodium mientras estoy usándolo		
DEPENDENCIAS:		**COMPLEJIDAD**	**Mayor** (5 P.H)
CRITERIOS DE ACEPTACIÓN:			
- La aplicación deberá enviar notificaciones a todos los usuarios cuando ocurra algún evento reseñable: en retos, la existencia de un nuevo reto o la resolución de alguno, en eventos, la creación, la resolución o el plazo de cierre, en logros, el desbloqueo… - Asimismo, deberá existir un panel de notificaciones en la barra de navegación de la cabecera de la plataforma.			
TAREAS:			
- Crear un sistema de notificaciones en el cliente en función de eventos, y que sea reactivo (3 Puntos de Historia) - Mostrar las notificaciones mediante diversos componentes, como cuadros de diálogo, popups y notificaciones (2 Puntos de Historia)			

Figura 5.17: Historia de Usuario 7.1.- Sistema de Notificaciones

Historia de Usuario 7.2.- Acceso móvil

HDU 7.2.- ACCESO MÓVIL			
DESCRIPCIÓN:	COMO usuario QUIERO disponer de una versión móvil PARA poder acceder a ECodium desde cualquier lugar		
DEPENDENCIAS:		**COMPLEJIDAD**	**Crítica (8 P.H)**
CRITERIOS DE ACEPTACIÓN:			
- La versión móvil consistirá, fundamentalmente, en una versión adaptada a pequeños dispositivos de la aplicación web - En la versión móvil no estarán disponibles los modales creación ni edición. Asimismo, tampoco estará disponible la resolución de retos ni la presentación de candidaturas - En dispositivos intermedios, como tablets, se determinará si se sirve la versión móvil o la versión escritorio en función del tamaño de la pantalla del dispositivo, para garantizar la óptima experiencia de usuario.			
TAREAS:			
- Crear una versión móvil del cliente de la aplicación web (5 Puntos de Historia) - Implementar que el sistema determine qué versión servir al usuario en función del tamaño de su pantalla (3 Puntos de Historia)			

Figura 5.18: Historia de Usuario 7.2.- Acceso móvil

5.2.9. Épica VIII: Pruebas y Despliegue

Historia de Usuario 8.1.- Pruebas unitarias

HDU 8.1.- PRUEBAS UNITARIAS			
DESCRIPCIÓN:	COMO administrador QUIERO probar el sistema antes del paso a producción PARA asegurarme de que funciona correctamente		
DEPENDENCIAS:	**Todas las anteriores**	**COMPLEJIDAD**	**Superior a crítica (13 P.H)**
CRITERIOS DE ACEPTACIÓN:			
- Las pruebas unitarias deberán probar todas las entidades mínimas de funcionalidad que existen, en distintos flujos, así como documentarse. - La documentación de estas pruebas servirá como referencia para redactar el Manual del Usuario, apéndice de la documentación del proyecto.			
TAREAS:			
- Realizar pruebas generales de usabilidad (3 Puntos de Historia) - Realizar pruebas de RETOS (3 Puntos de Historia) - Realizar pruebas de EVENTOS (3 Puntos de Historia) - Realizar pruebas de BANCO (2 Puntos de Historia) - Realizar pruebas de TALLERES (1 Puntos de Historia) - Realizar pruebas de LOGROS (1 Puntos de Historia)			

Figura 5.19: Historia de Usuario 8.1.- Pruebas unitarias

Historia de Usuario 8.2.- Despliegue y paso a producción

HDU 8.2.- DESPLIEGUE Y PASO A PRODUCCIÓN			
DESCRIPCIÓN:	COMO administrador QUIERO desplegar el sistema en un entorno en la nube PARA que puedan acceder los usuarios		
DEPENDENCIAS:	**Todas las anteriores**	**PUNTOS DE HISTORIA**	**Crítica** (8 P.H)
CRITERIOS DE ACEPTACIÓN:			
- El sistema deberá desplegarse a un entorno web conformado por Heroku en el servidor y Netlify en el cliente. - Existirá un endpoint para el servidor, denominado https://api.ecodium.dev, y la raíz de ese mismo dominio será el punto de acceso al lado cliente (https://ecodium.dev)			
TAREAS:			
- Desplegar servidor a Heroku (3 Puntos de Historia) - Desplegar cliente a Netlify (3 Puntos de Historia) - Enlazar cliente y servidor con sus endpoints (2 Puntos de Historia)			

Figura 5.20: Historia de Usuario 8.2.- Despliegue y paso a producción

5.3. Conclusiones

A modo de conclusión, la elección del uso de Épicas e Historias de Usuario a la hora de llevar a cabo la elicitación de requisitos del proyecto *software* no ha sido casual. Al tratarse de una plataforma con diversas funcionalidades y características, resulta útil desgranar esas funcionalidades y características desde una visión global en Épicas, para posteriormente, desgranar cada una de las Épicas en sus correspondientes Historias de Usuario.

Capítulo 6

Modelado de Datos

En este capítulo se describirá y explicará el modelo de datos sobre el cual se apoya la plataforma.

6.1. Introducción y contextualización

El **modelado de datos** es una parte esencial de todo proceso de desarrollo de *software*, ya que, si no se hiciera, los datos estarían sin estructurar y por tanto, no se podrían implementar Sistemas Gestores de Bases de Datos.

Para facilitar este proceso de modelado de datos, se suelen hacer uso de diagramas con el fin de representar los datos a modelar. Una de las herramientas más conocidas para llevar a cabo esta representación es el **Diagrama Entidad/Relación**, que como su nombre indica, representa una serie de Entidades que se relacionan entre sí mediante Relaciones. Es un tipo de diagrama que funciona muy bien para representar datos que posteriormente se implementarán en Sistemas Gestores de Bases de Datos relacionales.

Sin embargo, **no podría usarse** para proyectos donde lo que se use sea un Sistema Gestor de Bases de Datos no relacional, ya que la principal característica de estos sistemas, como su propio nombre indica, es que carecen de relaciones.

Al ser el modelo no relacional un modelo emergente y no tener a su espalda tantos años como el relacional, no existe ninguna norma o convenio para representar los datos. Sin embargo, en lo que muchos autores se ponen de acuerdo es en la representación del modelo de datos mediante un diagrama, similar al Diagrama de Clases, que describa todas las **Colecciones** existentes en nuestra base de datos no relacional. [1]

[1] Para confeccionar el diagrama del modelo lógico de datos del proyecto, me he apoyado en las pautas de Talha Awan al escribir un artículo muy completo sobre ello en el blog TecHighness (`https://www.techighness.com/post/how-to-draw-no-sql-data-model-diagram/`)

6.2. Modelo Lógico de Datos

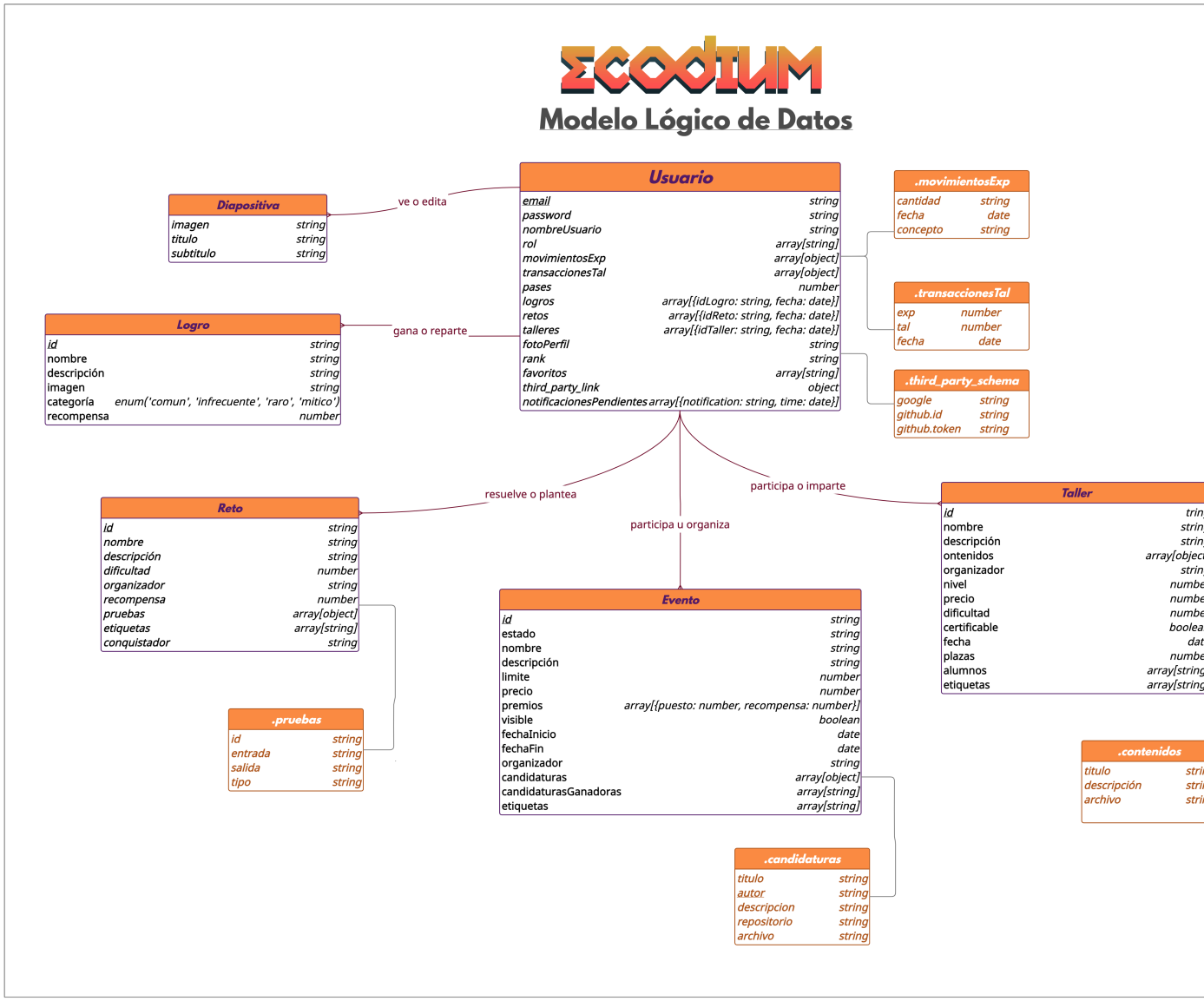


Figura 6.1: Diagrama para representar el Modelo Lógico de Datos de forma no relacional.

En el diagrama, se incluyen pequeñas notas a nivel conceptual para poder ver cómo se relacionarían los distintos tipos de datos entre sí. Así, podemos ver que el centro del modelo lógico de datos está el Usuario, con todas sus propiedades y tipos de las mismas. Estos Usuarios ven (o editan si son administradores) las diapositivas; con respecto a los Logros, los ganan si son jugadores y los reparten si son Organizadores, de la misma forma que se relacionan con Retos, Eventos y Talleres.

Siguiendo la similitud con el Diagrama de Clases, donde se representan las **clases** y cada *.ejemplar"* será una **instancia** de una clase, en algunos Sistemas Gestores de Bases

de Datos no relacionales, como MongoDB, los datos se modelan mediante **colecciones**, siendo cada ejemplar de una colección un **documento**, por lo que tiene bastante sentido que se representen de manera muy similar a las clases en el Paradigma Orientado a Objetos.

6.3. Seguridad de los datos modelados

6.3.1. La importancia de la seguridad en el modelado de datos

Un punto muy importante a la hora de implementar datos es la **seguridad**. Si hablamos de implementación mediantes bases de datos, uno de los tipos de vulnerabilidades más conocidos - y que pone más en riesgo una aplicación - es el conjunto de vulnerabilidades que se apoyan en un método de infiltración de código intruso llamado **inyección**.

Una **inyección** (en inglés, *injection*) consiste en aprovechar campos de entrada de datos de la aplicación para introducir **sentencias** que ejecuten consultas no autorizadas sobre la base de datos o, incluso, alteren datos. Por supuesto, la **inyección SQL** es muy popular debido a que los sistemas basados en SQL son populares, pero en este punto del Proceso de Desarrollo surgió la pregunta: ¿están los sistemas no relacionales (NoSQL) exentos de este tipo de ataques?

La respuesta es obvia: **no.** Las **inyecciones NoSQL** si bien son menos populares, pueden ser igual de problemáticas. Su mecanismo de funcionamiento es el mismo, cambiando aspectos relativos al lenguaje y arquitectura como la sintaxis. Sin embargo, evitarlas resulta algo más sencillo.

Centrándonos en el sistema que usa la plataforma que se está construyendo en este trabajo, MongoDB, existen dos vías principales de evitar las inyecciones: mongo-sanitize y hacer uso de Mongoose para, entre otras cosas, modelar los datos mediante **esquemas**[2]

6.3.2. La utilidad del modelado mediante esquemas

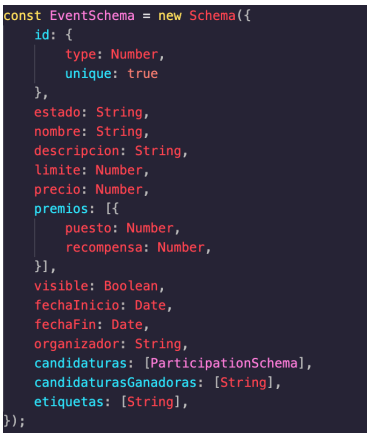
Mongoose es un *middleware* que se utilizará en la arquitectura de la plataforma, tal y como se describe en el siguiente capítulo. El motivo por el cual los **esquemas** evitan la llamada inyección de código es simple: un esquema (en inglés, *schema*), es la forma de modelar una Colección en Mongoose. En la construcción del esquema se debe describir la colección propiedad por propiedad, especificando para cada propiedad sus tipos de datos y restricciones si tiene.

De este modo, todas las acciones sobre la base de datos que impliquen una colección se ciñen a su esquema, **no permitiendo** la entrada de otros datos que no sean los especificados en el esquema.

Así pues, dado que la inyección NoSQL se lleva a cabo enviando objetos de tipo *Object* maliciosos (que comienzan por $ y albergan las sentencias), al usar esquemas, se traducirán

[2]Información extraída de un artículo del blog Zanon donde explican qué es una inyección y cómo evitarlas en MongoDB (`https://zanon.io/posts/nosql-injection-in-mongodb/`)

a **String** y no sucederá ninguna alteración que no queremos que suceda.



```
const EventSchema = new Schema({
    id: {
        type: Number,
        unique: true
    },
    estado: String,
    nombre: String,
    descripcion: String,
    limite: Number,
    precio: Number,
    premios: [{
        puesto: Number,
        recompensa: Number,
    }],
    visible: Boolean,
    fechaInicio: Date,
    fechaFin: Date,
    organizador: String,
    candidaturas: [ParticipationSchema],
    candidaturasGanadoras: [String],
    etiquetas: [String],
});
```

Figura 6.2: Captura de pantalla del esquema de Mongoose de la colección Eventos, donde se puede ver la especificación de sus propiedades y sus restricciones de tipo o unicidad. Además, podemos ver que se puede referenciar a otros esquemas.

Capítulo 7

Diseño y Arquitectura

En este capítulo se describirá el diseño y la arquitectura de la plataforma. Se comenzará detallando el **diseño** de la experiencia de usuario de la misma, para posteriormente describir su **arquitectura** a distintos niveles.

7.1. Introducción

Tanto el diseño de experiencia de usuario como la arquitectura del software, si bien son dos partes completamente diferenciadas, confluyen en un punto: constituir los pilares fundamentales de cara a la implementación, que solamente consistirá en hacer realidad estas pautas de diseño y arquitectura.

El capítulo se divide en varios epígrafes: el primero, en el diseño de UX, se especificará las características de la experiencia de usuario objetivo, así como los recursos visuales utilizados. Del segundo en adelante, se describirá la arquitectura de la plataforma a distintos niveles y se apoyará en diagramas para representarla.

7.2. Diseño de Experiencia de Usuario (UX)

7.2.1. Características de la Experiencia de Usuario

Se suele definir experiencia de usuario como todo el conjunto de patrones, elementos y factores de los que depende la interacción de un usuario con un determinado producto o servicio. Si bien tradicionalmente ha ido asociada a la interfaz, el diseño de experiencias de usuario cada vez más se enfoca como un *"todo"* que, si bien incluye el diseño de interfaces de usuario, no se limita a ellos.

La guía *.Experiencia de Usuario: Principios y Métodos"*[11], antes de realizar un recorrido sobre todos los principios de diseño y su correcta aplicación, define una serie de conceptos fundamentales sobre el diseño de experiencia de usuario y por qué son interesantes.

A continuación, se detallan cuáles de esos conceptos fundamentales se aplican y cómo se han aplicado a la hora de diseñar la experiencia de usuario de ECodium:

- **Usabilidad y *affordance***: La usabilidad es uno de los pilares básicos de toda aplicación, apoyándose en el *affordance* como complemento ideal. La interfaz de ECodium se ha pensado con la usabilidad como principal enfoque, contando con un menú muy simple y iconos lo más autoexplicativos posible.
- **Accesibilidad**: En el contexto del diseño de UX, la accesibilidad consiste en que la aplicación pueda ser usada por el mayor número de personas posible. ECodium no requiere de ningún conocimiento en específico para ser utilizada, por lo que es accesible.
- **Arquitectura de información** o cómo se organizan los datos. Tanto a alto nivel (mapa del sitio y distintas secciones que tiene la página) como a bajo nivel (modelado de datos), la información se encuentra organizada y clasificada.
- **Relación Esfuerzo-Beneficio**: Si bien la usabilidad es una parte importante, no es bueno conformarse con ofrecer una experiencia de usuario lo más fácil que podamos ofrecer, porque se podría perder la oportunidad de ofrecer una experiencia de usuario mucho más completa. Esta relación esfuerzo-beneficio resulta crucial para el usuario

7.2.2. UX e interfaz

Inspiración

Una de las características de todo buen *software* es precisamente que sea capaz de entrarle al usuario por los ojos. Con el fin de lograr una estética atractiva se buscó lograr una estética que tuviera identidad propia, fácilmente reconocible, agradable y sin caer en la redundancia o el ornamento excesivo.

Así pues, se comenzó tomando como base los patrones de diseño de Material Design de Google [1], un sistema de diseño muy de moda en nuestros días y que la librería de componentes utilizada *(Quasar Framewor)* toma como referencia principal.

El añadido diferenciador es una estética algo desenfadada, que busca rendir homenaje a los años 80 y a los inicios de la informática y la ciencia ficción, una estética atractiva para el público objetivo, que está interesado en una experiencia tecnológica gamificada.

[1] Más información sobre Material Design: `https://material.io`

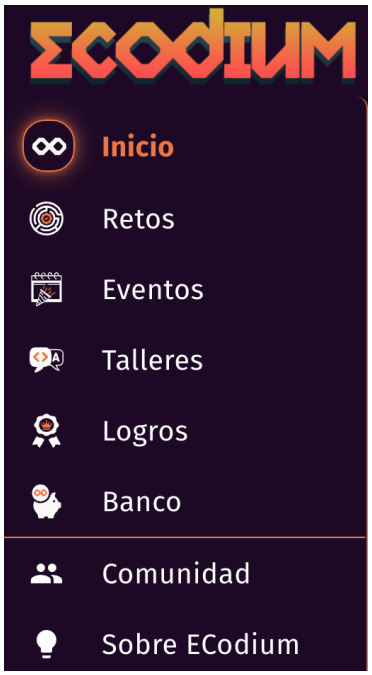


Figura 7.1: Menú de navegación de ECodium. Se pueden ver los dos colores de acento de la paleta de la plataforma, así como un ligero efecto de neón para resaltar la ubicación actual del usuario

Paleta de colores

Esta es la paleta de colores utilizada por la plataforma:

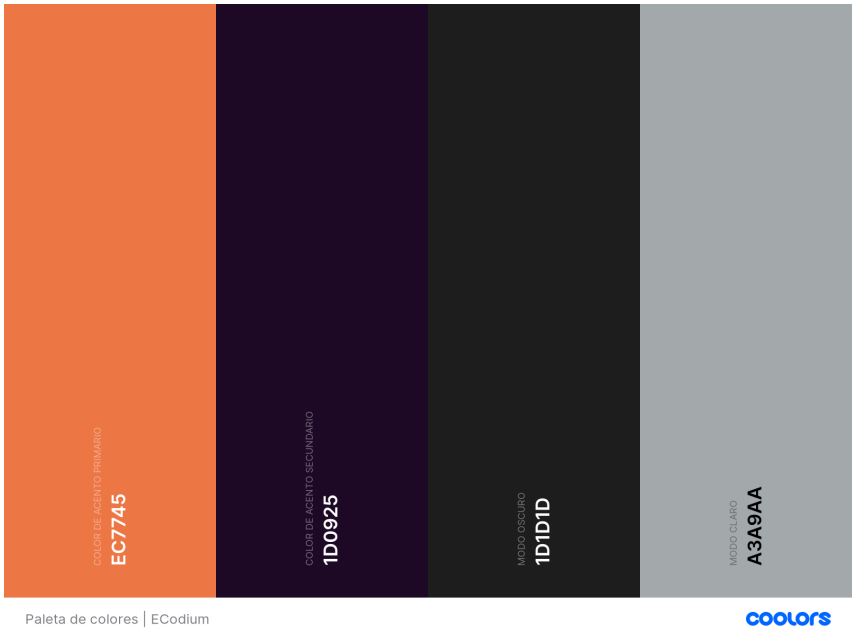


Figura 7.2: Paleta de colores de ECodium. De izquierda a derecha: color de acento primario, color de acento secundario, color identificativo del modo oscuro y color identificativo del modo claro

Tipografía

La tipografía de ECodium viene marcada por dos fuentes de letras, ambas creadas por Mozilla: **Fira Sans** como fuente principal y **Fira Mono** como fuente secundaria, para algún detalle que requiera cambiar de tipografía.[2]

Iconos

Los iconos usados en ECodium se dividen en **iconos propios,** diseñados en formato vectorial y solamente utilizados para esta plataforma, e **iconos externos,** de la librería de iconos de Material Design[3]

Figura 7.3: Iconos propios utilizados en ECodium, en su versión oscura (sobre fondo claro)

7.3. Arquitectura Lógica: Diagramas Multinivel

Para representar la **arquitectura lógica** de la aplicación, se ha tomado como referencia un modelo de **arquitectura multinivel**, que defiende la arquitectura software mediante **capas** diferenciadas y que interactúan entre sí, logrando simplificar el desarrollo de soluciones software.

El modelo más conocido de arquitectura multinivel es el llamado **modelo de 3 capas**, que define tres capas:

[2]Más información sobre ambas fuentes en la página que Mozilla habilitó para ello: `http://mozilla.github.io/Fira/`

[3]Librería de iconos de Material Design: `https://material.io/icons/`

- **Capa de Presentación**, que muestra los datos al usuario mediante una interfaz, proporcionando así la experiencia de usuario.
- **Capa de Lógica de Negocio**, que procesa los datos y realiza todo tipo de interacciones con los mismos.
- **Capa de Datos**, que proporciona el acceso a los datos que normalmente se encuentran en un sistema de bases de datos.

Se han realizado dos diagramas de tres capas para representar la arquitectura lógica: el primero, a más alto nivel, se ha definido como **Arquitectura Lógica Simplificada**, mientras que el segundo es más detallado.

7.3.1. Arquitectura Lógica Simplificada

Figura 7.4: Diagrama de tres capas a alto nivel, que representa la arquitectura lógica de manera simplificada

- En la **capa de presentación**, encontramos la lógica de la propia capa y la interfaz de usuario, que se divide en una serie de fragmentos:

    - El **layout** que describe la organización de la página
    - La **propia página** que puede albergar una serie de **componentes web reutilizables**

- En la **capa de lógica de negocio** describimos los **endpoints** por los que la API será accesible, que se apoyan en las distintas **rutas** del *backend* y en los **modelos** de datos. A su vez, esta capa también alberga la **lógica del *backend*** y sus dos aspectos más importantes: Seguridad e Integración de datos y Manejo de Middlewares.
- En la **capa de datos** simplemente se especifica la lógica de persistencia de datos y los mecanismos de acceso y orquestación de acciones sobre la base de datos.

7.3.2. Arquitectura Lógica Completa

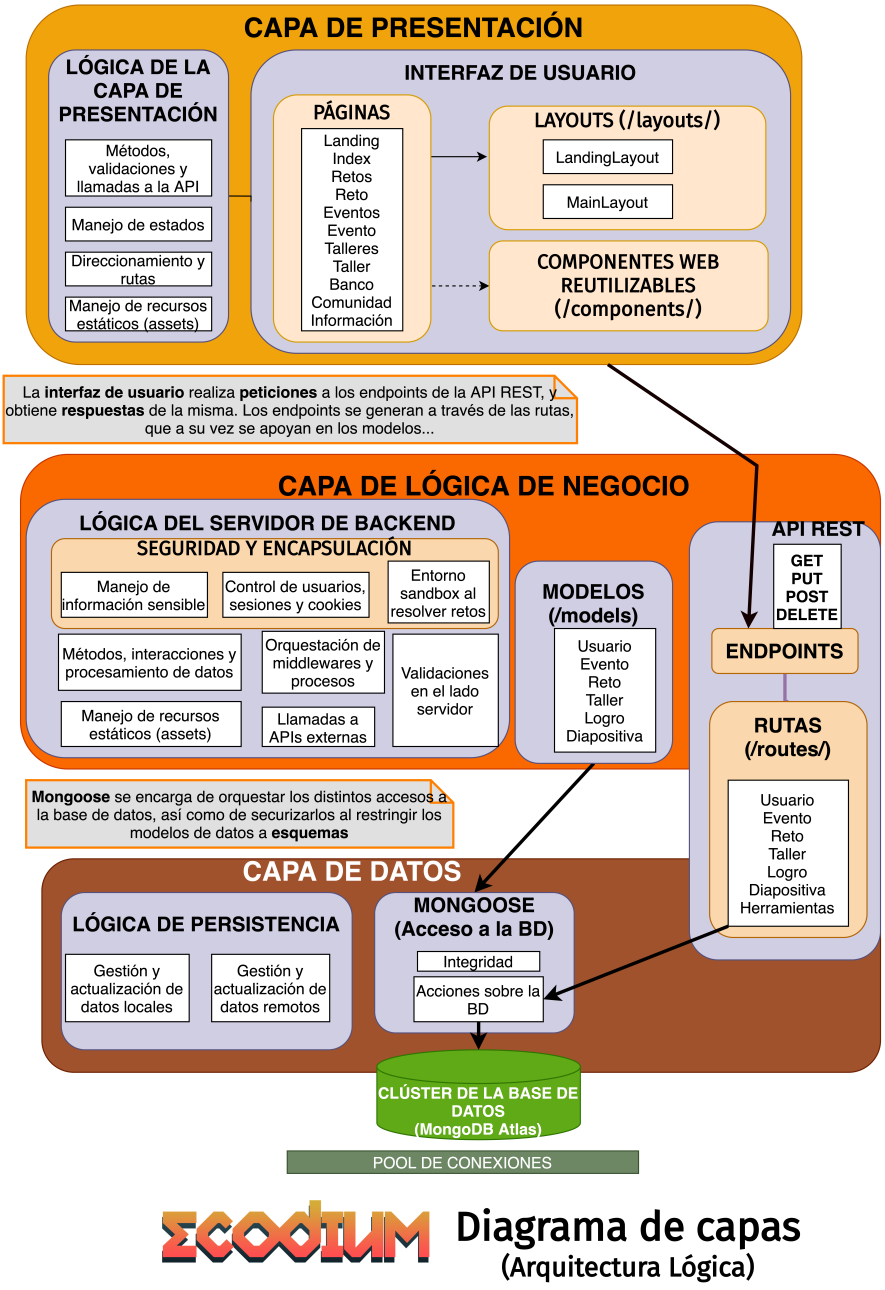


Figura 7.5: Diagrama de tres capas a alto nivel, que representa la arquitectura lógica con mayor nivel de detalle

Capa de Presentación

La capa de presentación se compone de:

- **Lógica de la Capa de Aplicación:** que alberga todos aquellos métodos, validaciones y llamadas a la API, así como el manejo de estados, direccionamiento de rutas e integración de recursos estáticos o *assets*
- **Interfaz de Usuario:** que como se ha explicado anteriormente se compone de Páginas, Layouts y Componentes Web Reutilizables.

Capa de Lógica de Negocio

La capa de lógica de negocio se compone de:

- Un importante componente de **lógica de *backend***, compuesto a su vez por:
  - Herramientas para favorecer la **seguridad y encapsulación** tales como el manejo de información sensible, el control de usuarios y sesiones y el entorno sandbox donde operan los jugadores que resuelven retos.
  - Los métodos, interacciones y el procesamiento de datos y su validación, así como el manejo de recursos estáticos o *assets* y las llamadas a APIs externas (en el caso que se describe, la API REST de GitHub [4] y el estándar OAuth2 para el inicio de sesión social mediante Google y GitHub[5])
  - La orquestación de todos los *middlewares* y procesos subyacentes al servidor.
- Los **modelos** de datos ceñidos a esquemas
- Toda la **API REST interna**, que hace las veces de "puente.[en]tre esta capa y la capa de datos, compuesta por *endpoints* apoyados en una serie de rutas.

Capa de Datos

La capa de datos se compone de:

- La **lógica de persistencia**, que principalmente se fundamenta en la gestión y actualización de datos tanto locales como remotos. Esto normalmente se traduce en la serialización (o guardado) y en la deserialización (o recuperación) de los mismos.
- El **acceso a la Base de Datos**, en este caso respaldado por Mongoose, que garantiza una serie de directrices de integridad de los datos, así como ejercer de *middleware* para las distintas acciones sobre la Base de Datos.
- El propio **clúster de la base de datos**, alojado y gestionado por MongoDB Atlas.

[4] API REST de GitHub: `https://docs.github.com/en/rest`
[5] Más información sobre el estándar OAuth 2.0: `https://oauth.net/2/`

7.4. Arquitectura Física: Diagrama de Despliegue

Toda arquitectura lógica debe apoyarse en una arquitectura física que sea capaz de permitir su implementación y por tanto hacerla realidad. La arquitectura física de ECodium se ha representado mediante un **diagrama de despliegue**:

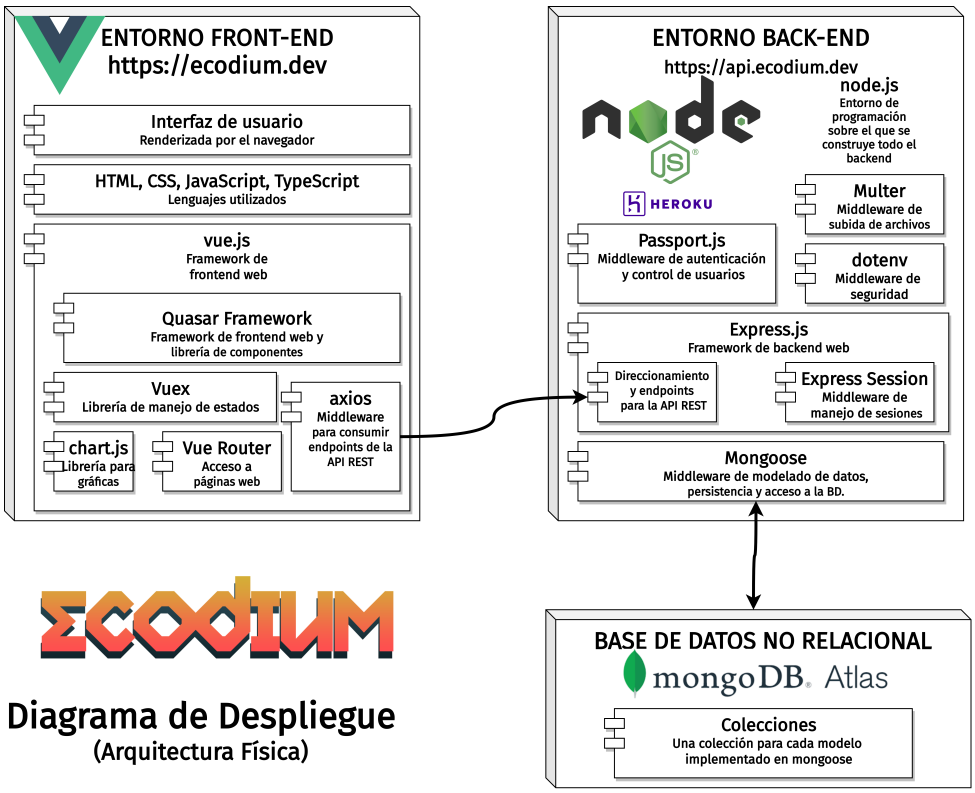


Figura 7.6: Diagrama de despliegue, que representa la arquitectura física del entorno de la aplicación

Con un rápido vistazo al diagrama de despliegue podemos ver tres bloques completamente diferenciados: el **entorno de *frontend***, el **entorno de *backend*** y la **base de datos no relacional**.

La base de datos, por su parte, se compone de Colecciones y estas a su vez de tantos documentos como .[ej]emplares.[ex]istan.

Entorno de *frontend*

En el lado del *frontend* destaca, de mayor a menor nivel de abstracción:

- La **interfaz de usuario** que renderice el navegador del usuario
- Los **lenguajes** sobre los que se construye dicha interfaz. En este caso, se han utilizado HTML, CSS, JavaScript y TypeScript.

- El **framework** de *frontend* utilizado, **vue.js**, que a su vez alberga una serie de componentes adicionales:
  - **Quasar Framework** como librería de componentes, que sirve de apoyo de cara a elaborar una experiencia de usuario visualmente atractiva
  - **Vuex** como librería de manejo de estados
  - **chart.js** como librería para generar gráficos a partir de datos
  - El **Vue Router** como mecanismo de direccionamiento de rutas y acceso a páginas web
  - **axios** como *middleware* para consumir los distintos *endpoints* de la API. Axios será el componente responsable de comunicarse con la API e interpretar la respuesta recibida.

Entorno de *backend*

Por su parte, el entorno de *backend* también se divide en fragmentos, todos ellos sobre **node.js** como infraestructura principal de *backend*:

- **Passport.js** como *middleware* de autenticación y gestión de usuarios. Se ocupa, entre otras cosas, de llevar a cabo la autenticación social mediante el estándar OAuth 2.0 descrito anteriormente.
- **Multer** como *middleware* de subida y recepción de archivos locales por parte del usuario. Por seguridad, se valida el tipo de archivo antes de procesarlo, permitiendo solamente imágenes y documentos ofimáticos, así como archivos con extensión .pdf
- **dotenv** como *middleware* de seguridad y encapsulación que protege datos sensibles mediante variables de entorno.
- **Express.js** como *framework de backend web.*, ocupándose tanto del direccionamiento de rutas y *endpoints* de la API como del control de sesiones mediante su complemento *Express Session*.
- **Mongoose** como *middleware* de modelado y acceso a datos, persistencia y comunicación con la base de datos.

7.5. Arquitectura de la API REST interna

Con el fin de ilustrar el funcionamiento de la API REST interna, se han confeccionado una serie de **mapas de *endpoints*** que ayudan a entender, de manera visual, cómo está pensada la API que sirve al proyecto:

Figura 7.7: Mapa de *endpoints* desde una visión global: muestra las rutas en las que se divide la API

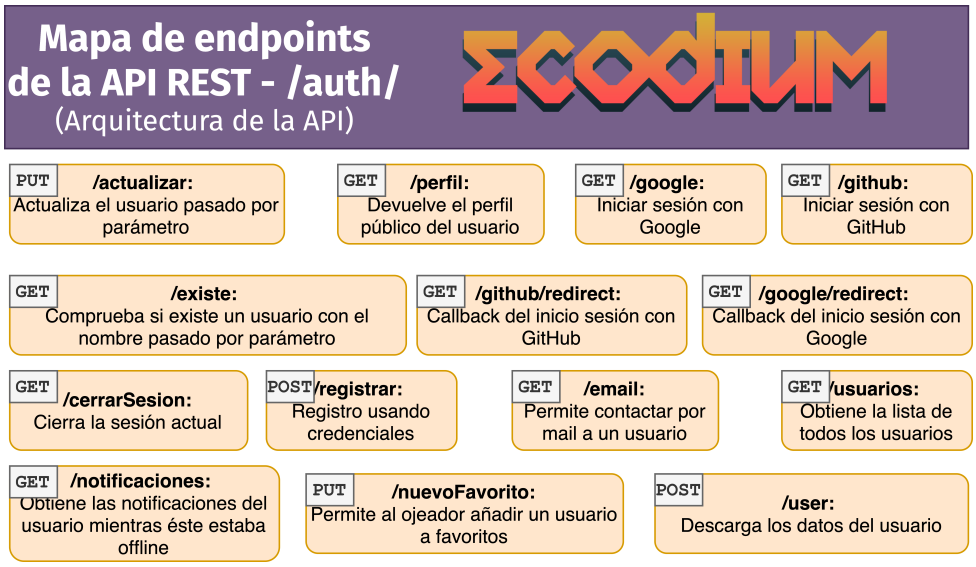


Figura 7.8: Mapa de *endpoints* de Usuarios: muestra los *endpoints* relativos a la gestión de usuarios

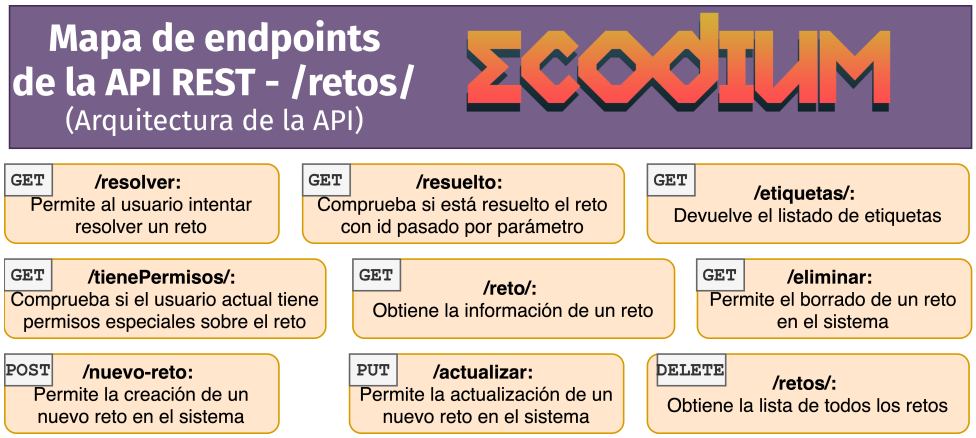


Figura 7.9: Mapa de *endpoints* de Retos: muestra los *endpoints* relativos a los Retos

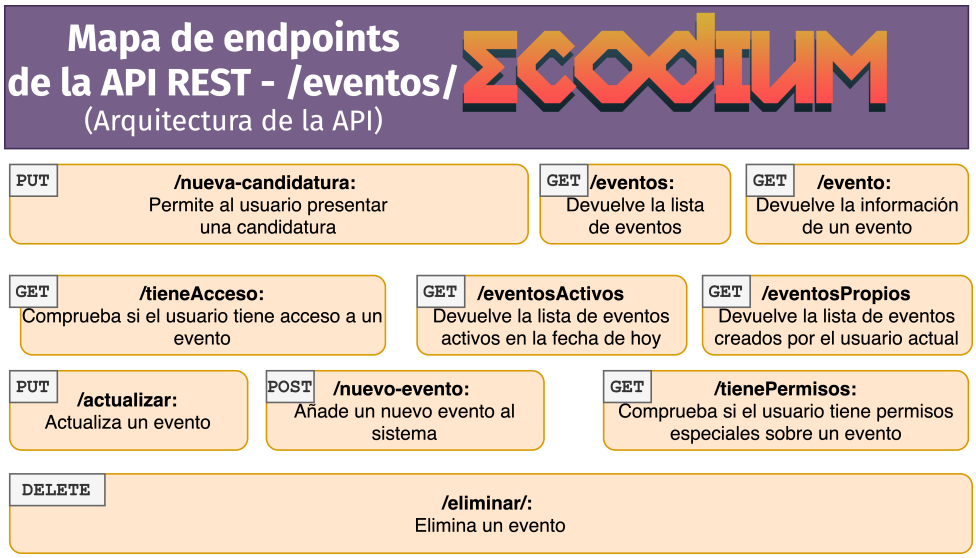


Figura 7.10: Mapa de *endpoints* de Eventos: muestra los *endpoints* relativos a los Eventos

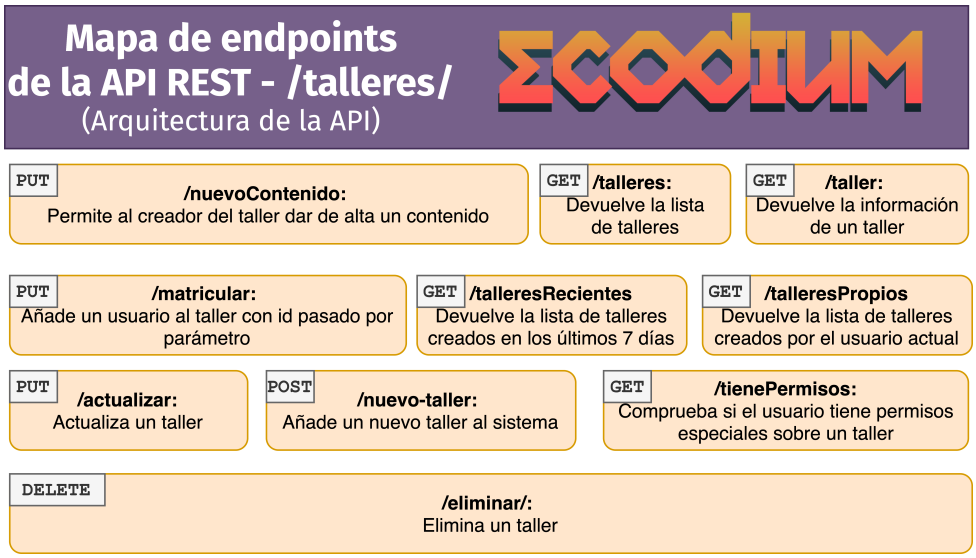


Figura 7.11: Mapa de *endpoints* de Talleres: muestra los *endpoints* relativos a los Talleres

Figura 7.12: Mapa de *endpoints* del resto de rutas: Herramientas y Logros.

7.6. Ejemplificación de un flujo de la aplicación: Diagrama de Secuencia

Por último, con el fin de ejemplificar cómo debería funcionar la plataforma con su diseño y arquitectura planteados, se ha elaborado un único **diagrama de secuencia** que muestre un flujo concreto: la **resolución de un reto** por parte de un jugador partiendo desde la página de inicio.

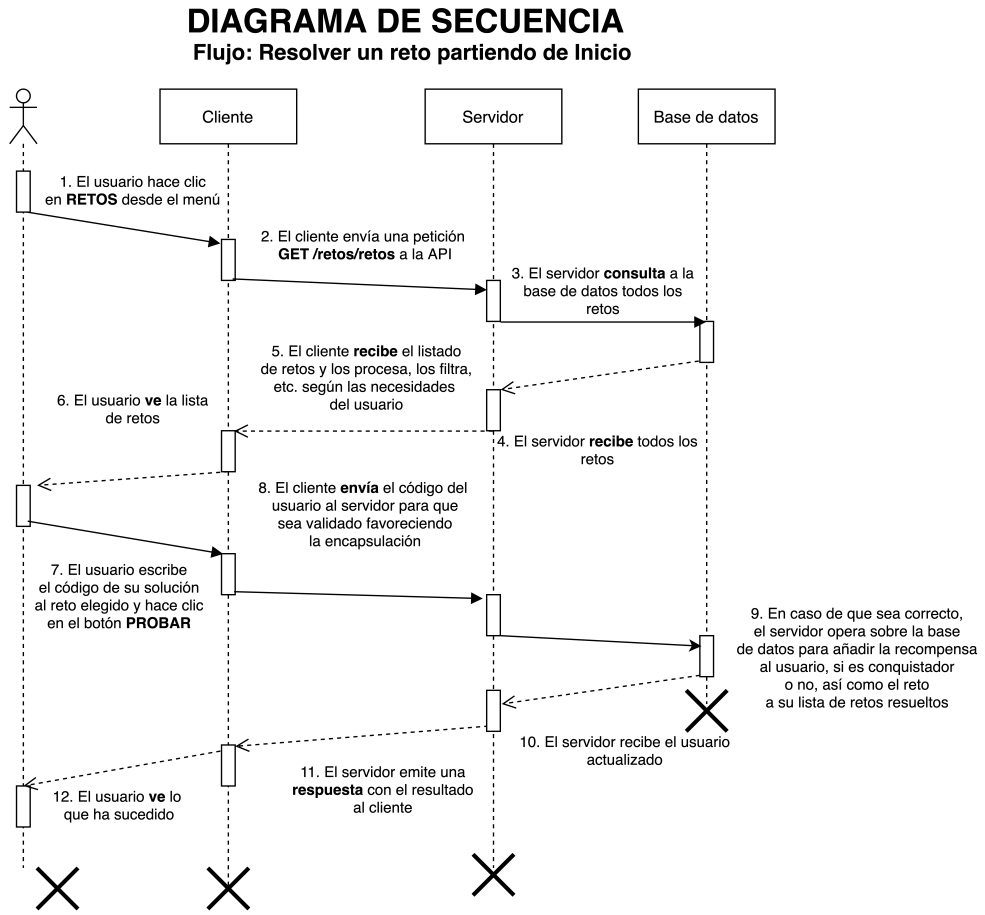


Figura 7.13: Diagrama de secuencia que ejemplifica un flujo determinado de la aplicación

Capítulo 8

Implementación y Despliegue

En este capítulo se explicará como se han implementado los aspectos esenciales de la plataforma, centrándose en tres aspectos: el *token de ERC20*, el *entorno backend* y el *entorno frontend*. Posteriormente se comentará como se lleva a cabo el despliegue a entorno *cloud*.

8.1. Implementación y funcionamiento del *token*

Como se describió con anterioridad en este documento, el TALENTO (TAL) es el **token ERC20** de la plataforma. Para desarrollarlo, se ha utilizado el lenguaje Solidity[1] y el IDE Remix[2].

Solidity es el lenguaje sobre el que se desarrollan los *smart contracts* de la red Ethereum. Es un lenguaje orientado a objetos y de alto nivel, y, según su propia documentación, está influenciado por lenguajes como C, Python o JavaScript.

Para desarrollar el token TAL, se ha seguido la guía de Juan Cruz Martínez[8]. Se ha comenzado escribiendo en Solidity un *smart contract* que sea *mintable*. Un *token* es *mintable* si permite que nuevos *tokens* se creen en cualquier momento (transfiere al usuario desde la dirección Genesis, una dirección especial). En nuestro caso, existe un cupo máximo, del cual se van descontando los *tokens*. Este cupo máximo hace que los TAL sean limitados. [3]

Por tanto, el *mint* es la función más principal del *smart contract* del TALENTO. El fichero completo del *smart contract* se encuentra en el repositorio adjunto, así que con el fin de no extender demasiado la documentación, solamente se incluye en esta documentación dicha función *mint*:

```
1 //
    ------------------------------------------------------------------------
```

[1]Documentación de Solidity: `https://docs.soliditylang.org/en/v0.8.7/`

[2]Más información sobre el IDE Remix, un IDE oficial de Ethereum: `https://remix.ethereum.org`

[3]Artículo explicativo sobre los *mintable tokens*: `https://tokenmint.io/blog/mintable-erc20-token-explained.html`

```
2  // Generar TALENTOS (ECodium --> BANCO --> Canjear)
3  //
       ------------------------------------------------------------------------

4  function mint(uint256 tokens) public returns (bool success) {
5      // Limitar el minting de tokens a 5 minutos para no sobrecargar la
           red
6      if (now - last_mint_time < 5 minutes) {
7          return false;
8      }
9      require(tokens <= _totalSupply);
10     balances[msg.sender] = safeAdd(balances[msg.sender], tokens); //
           Sumamos los tokens generados al saldo del usuario
11     _totalSupply = safeSub(_totalSupply, tokens); // Restamos los tokens
           generados del supply disponible
12     _givenSupply = safeAdd(_givenSupply, tokens); // Sumamos los tokens
           repartidos al supply ya repartido
13     last_mint_time = now; // Guardamos la fecha del minting actual
14     emit Transfer(address(0), msg.sender, tokens); // Emitimos un nuevo
           evento de transferencia de la direccion Genesis al usuario
15     return true;
16 }
```

El siguiente paso a seguir es, hacerse con una cartera de Ethereum en la Red de Prueba Ropsten, una red, que como su nombre indica, es de prueba, por lo que te permite añadir una cantidad ilimitada de *ethereum* para pagar GAS (todo simulado) y así simular operaciones reales en la red Ethereum.

Una vez hecho esto, podemos, mediante el Remix IDE, compilarlo y, si la compilación es exitosa, proceder a desplegar el *token* en dicha red de prueba Ropsten.

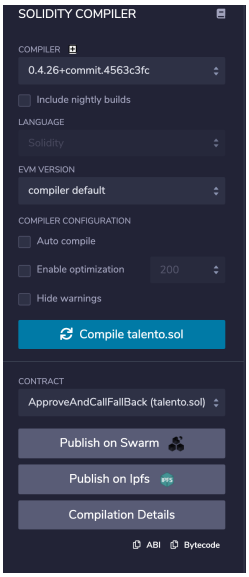


Figura 8.1: Compilación del token en el Remix IDE

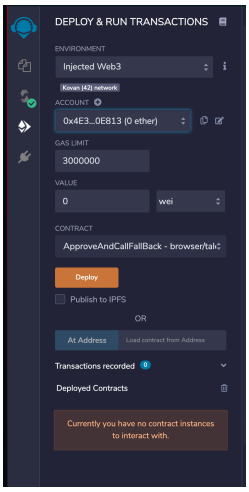


Figura 8.2: Despliegue del token en la red vía el Remix IDE

Por último, una vez desplegado el token en la red de prueba Ropsten, el token estaría listo para usarlo en la aplicación, en carteras de dicha red de prueba o en escáneres específicos como el ya mencionado Etherscan *(ver Figura 3.6)*.

8.2. Implementación del *backend*

La **implementación del *backend*** se divide en estas partes importantes:

- **server.js:** Es el archivo principal del servidor node.js. Carga y orquesta los *middlewares* del entorno y además importa el resto de archivos.
- **/models/:** Directorio donde se almacenan los modelos de datos.
- **/routes/:** Directorio donde se describen las distintas rutas, su lógica y se crean los *endpoints* de la base de datos.

8.2.1. Archivo principal del servidor

Dado que no es muy extenso y está comentado, se adjunta aquí el código íntegro del archivo server.js, omitiendo las tildes en los comentarios para no causar problemas con LaTeXy UTF-8:

```
1   /**
2   * ECODIUM - Servidor de Backend
3   */
4
5   // Variables de middleware y servicios usados
6
7   // Express - Middleware
8   const express = require('express');
9   const app = express();
```

```
10    const session = require("express-session");
11
12    // Mongoose - Middleware de persistencia en la base de datos
13    const MongoStore = require("connect-mongo");
14    const mongoose = require('mongoose');
15    const AutoIncrement = require('mongoose-sequence')(mongoose);
16
17    const bodyParser = require('body-parser');
18
19    // PassportJS - Middleware de autenticacion
20    const passport = require("./api/auth");
21    const cors = require("cors");
22
23    // DotENV - Middleware para la seguridad en produccion
24    const dotenv = require('dotenv');
25    dotenv.config();
26
27    // Manejo de secretos y datos sensibles
28    const secretMongoDB = process.env.MONGODB_URI;
29
30    // ObjectID - Validacion de datos
31    const Joi = require('@hapi/joi');
32    Joi.objectId = require('joi-objectid')(Joi);
33
34    // Otros middlewares menores
35    app.use(bodyParser.json()); // JSON
36    app.use(cors({credentials: true, origin: 'https://www.ecodium.dev'}))
          // CORS para los endpoints
37
38    // Configuracion de Express
39    app.use(express.urlencoded({extended: true}))
40    app.use(express.json());
41    app.use('/subidas', express.static('uploads'));
42
43    // Configuracion de Mongoose y Multer
44    mongoose.connect(secretMongoDB, {useNewUrlParser: true,
          useUnifiedTopology: true, useFindAndModify: false})
45    .then(() => console.log("Conectado a la base de datos"))
46    .catch(err => console.log('Error:' + err))
47    app.set('trust proxy', 1)
48    // Configuracion de Express Session
49    app.use(
50      session({
51          secret: "secretoDeLaCookie",
52          resave: true,
53          saveUninitialized: true,
54          store: MongoStore.create({ mongoUrl: secretMongoDB}),
55          cookie: {
56    httpOnly: true, secure: true, maxAge: 1000 * 60 * 60 * 48, sameSite:
          'none'
57              }
58      })
```

```
59    );
60
61    // Inicializacion de PassportJS
62    app.use(passport.initialize());
63    app.use(passport.session());
64
65    // Carga de las rutas y definicion de los paths de la API
66    const auth = require("./routes/usuarios");
67    app.use('/api/auth', auth);
68    const retos = require("./routes/retos");
69    app.use('/api/retos', retos);
70    const logros = require("./routes/logros");
71    app.use('/api/logros', logros);
72    const eventos = require("./routes/eventos");
73    app.use('/api/eventos', eventos);
74    const talleres = require("./routes/talleres");
75    app.use('/api/talleres', talleres);
76    const herramientas = require("./routes/herramientas");
77    app.use('/api/herramientas', herramientas);
78
79    // Configuracion de ejecucion de la aplicacion
80    app.set('puerto', process.env.PORT || 3000);
81    app.listen(app.get('puerto'), err => {
82      if(err) throw err;
83      console.log("Servidor ejecutandose");
84    });
```

8.2.2. Entorno sandbox

Para ejecutar el código introducido por el usuario al resolver un reto en un entorno seguro (lo que se conoce como *sandbox*), se hace uso de **NodeVM**, un complemento que hace uso del propio módulo interno de VM de node.js para encapsular el código y no dejar que aplique funcionalidad maliciosa al propio servidor. [4]

8.2.3. Mensajería

Por último, se ha implementado un **sistema de mensajería por email** que favorece la encapsulación utilizando *middlewares* y las herramientas **EmailJS** [5] y **Mailgun** [6] para proveer a nuestros usuarios de una comunicación interna segura y práctica.

Su funcionamiento es muy sencillo. Si un usuario envía a otro un mensaje en ECodium, lo hace mediante un formulario de contacto con seguridad aplicada mediante reCAPTCHA v3 de la API de Google [7]. Si el *captcha* concluye satisfactoriamente, al usuario destinatario le llegará un correo del *çartero de ECodium"*, una forma amigable de referirnos al

[4] Más información sobre NodeVM: `https://github.com/patriksimek/vm2`
[5] `https://emailjs.com`
[6] `https://mailgun.com`
[7] `https://developers.google.com/recaptcha/docs/v3`

middleware mensajero:

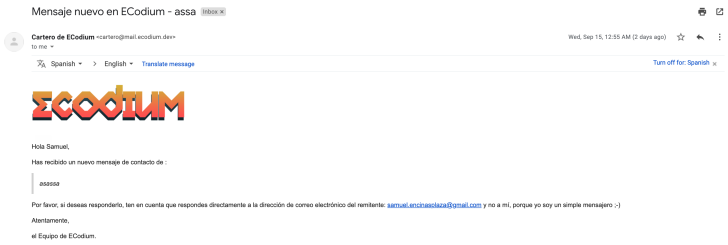


Figura 8.3: Mensaje recibido desde ECodium

8.3. Implementación del *frontend*

La premisa principal de implementación del *frontend* es construir un sistema que no sólo cumpla los patrones de diseño de interfaz de usuario descritos con anterioridad, sino que lo haga utilizando tecnologías punteras.

Es por eso que, además de implementar el *backend* con node.js, se ha implementado el *frontend* desde el punto de vista de los ***web components***: un conjunto de características que permiten el desarrollo web modular mediante widgets, documentos y componentes reutilizables. [8]

Se trata de una tecnología emergente, pero que ha ganado muchísima popularidad y se ha convertido en una auténtica revolución del desarrollo web con el auge de la web reactiva y de *frameworks* como Angular, React o el propio Vue.

El *frontend* de ECodium se compone de los siguientes archivos y directorios importantes:

- /**layouts**/: Sobre los cuales se construyen las páginas. En ECodium existen dos: MainLayout y LandingLayout
- /**pages**/: Directorio que contiene las distintas páginas.
- /**components**/: Directorio que contiene los distintos componentes, que pueden ser usados por una o más páginas o *layouts*.
- /**router**/: Directorio que contiene los archivos de configuración necesarios para el direccionamiento de rutas en el lado del *front*.
- /**store**/: Directorio que contiene los archivos de configuración necesarios para el manejo de estados mediante Vuex.
- **quasar.conf.js**: Archivo de configuración de la librería *Quasar Framework*

[8] Más información en este artículo: `https://desarrolloweb.com/manuales/web-components.html`

8.3.1. Layouts, Páginas y Componentes

Si bien los *layouts*, las páginas y los componentes difieren entre sí, ya que normalmente uno incluye al segundo y el segundo al tercero, lo cierto es que confluyen en la estructura. Podríamos decir que los *layouts* son componentes que pueden tener páginas y componentes, las páginas son componentes que pueden tener otros componentes e implementar layouts y los componentes simplemente pueden tener otros componentes. Estos archivos tienen la extensión .vue. De nuevo, se adjuntan todos los directorios, y para no hacer la memoria muy extensa, dado que estos archivos ocupan una cantidad considerable de líneas de código, se incluye a modo ilustrativo una captura de pantalla de la estructura de, en este caso, una página. Aunque como se expone anteriormente, los *layouts* y los componentes contienen la misma estructura en tres partes:

Figura 8.4: Estructura de un documento Vue

Un componente especial es el **editor de código** de los Retos, que se fundamenta en **codemirror**, un componente de editor de texto basado en JavaScript que permite la carga de sintaxis y palabras reservadas de varios lenguajes de programación. [9] No obstante, ha sido modificado ligeramente, por cuestiones de diseño e integración con el diseño de ECodium.

[9] Más información sobre CodeMirror en `https://codemirror.net`

8.3.2. Manejo de rutas y estados

Rutas

La declaración de rutas se realiza en el fichero **routes.ts**, escrito en TypeScript, y tiene esta estructura:

```
    const routes: RouteConfig[] = [

  {
    path: '/',
    component: () => import('layouts/LandingLayout.vue'),
    children: [
      { path: '', component: () => import('pages/Landing.vue') }
    ]
  },
  {
    path: '/inicio',
    component: () => import('layouts/MainLayout.vue'),
    children: [
      { path: '', component: () => import('pages/Index.vue') }
    ]
  },
  {
    path: '/reto/:id',
    component: () => import('layouts/MainLayout.vue'),
    children: [
      { path: '', component: () => import('pages/Reto.vue') }
    ]
  },...
```

Como se puede ver en la estructura, cada ruta de *frontend* tiene un **path** o ruta de acceso, un **componente de tipo Layout** que implementa, y como *"hijo"*, la **página** a la que hace referencia.

Estados

Los **estados** en Vuex se manejan mediante **propiedades, acciones y mutaciones**: Un estado cuenta con una serie de propiedades, que son alteradas mediante mutaciones. Las acciones, por su parte, también pueden pero no tienen por qué modificar ninguna propiedad.

```
export default {
  state: {
    user: null,
    objetivo: null,
  },
  mutations: {
    SET_USER(state, user) {…
    },
    SET_OBJETIVO(state, objetivo) {…
    },
    setAuthStatus(state, authStatus) {…
    }
  },
  getters: {
    sesionIniciada(state) {…
    }
  },
  actions: {
    //Registro
    async REGISTER({…
    },
    //Login
    async LOGIN({…
    },
    //Cerrar sesión
    async LOGOUT({…
    },
    //CheckUser
    async CHECK({…
    },
```

Figura 8.5: Estructura de un documento de manejo de estados de Vuex

8.4. Despliegue en la Nube

8.4.1. Despliegue del *backend* mediante Heroku

Heroku [10] es un servicio que permite el despliegue en la nube de servidores de todo tipo, desde node.js hasta Nginx pasando por Apache. Se constituye por distintos nodos (en Heroku los llaman *"dynos"*) que se pueden interconectar entre sí formando tu propio clúster. En el caso de ECodium, el *backend* ocupa un *dyno* de los primeros de pago, que son gratuitos si eres estudiante. Si quisiera desplegar el *frontend* aquí también, tendría que usar otro *dyno*. Podría usar un *dyno* gratuito, pero tienen un hándicap enorme: los gratuitos no están 24 horas activos, sino que se desconectan si no reciben peticiones, existiendo un tiempo de .[en]cendido.[en] cuanto llegue la primera petición.

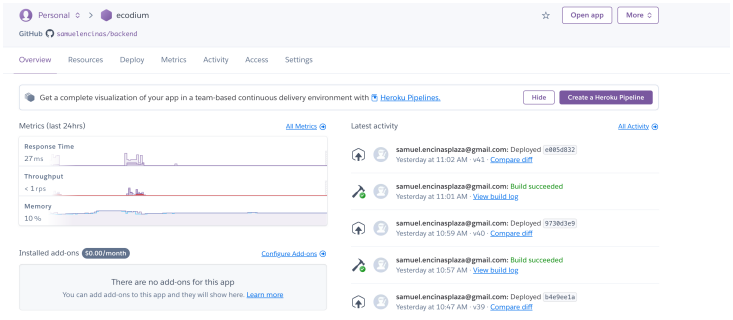


Figura 8.6: Panel de Control de ECodium en Heroku

[10] `https://heroku.com`

8.4.2. Despliegue del *frontend* mediante Netlify

Netlify [11] es una herramienta que permite el despliegue en la nube (tanto gratuito como *premium*) de contenidos web **estáticos**. La gracia de usar un *framework* como Vue radica precisamente en esto: se puede plantear como archivos estáticos, no requiere nada dinámico como sí lo hacen otros *frameworks y CMS* .

De este modo, ejecutando **quasar build** se compila el *frontend* en una cómoda carpeta repleta de todos los archivos estáticos. Dicha carpeta, puedes sincronizarla con un repositorio de Git a partir del cuál desplegar en Netlify tu aplicación web.

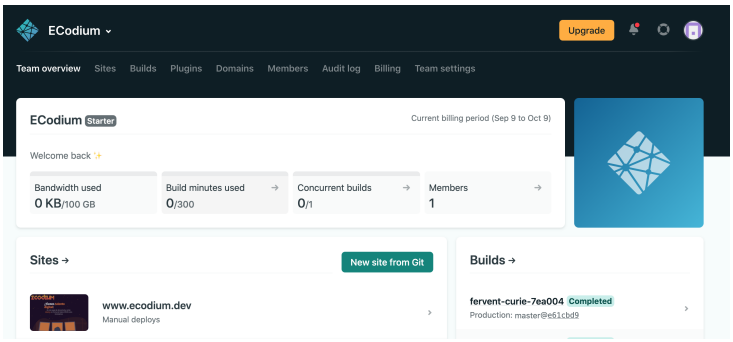


Figura 8.7: Panel de Control de ECodium en Netlify

8.5. Pruebas realizadas

En este último apartado, se describen algunas de las **pruebas** realizadas sobre distintas funcionalidades de la aplicación con el fin de verificar su correcto funcionamiento. Las pruebas realizadas son de **caja negra** y tienen un carácter **funcional**: se limitan a probar el comportamiento de las funcionalidades de la aplicación ante diversas casuísticas.

A continuación se adjuntan cinco pruebas de caja negra realizadas:

[11] `https://app.netlify.com`

PCN-02	**Filtrado de Retos**
Objetivo	Comprobar si el sistema permite al usuario filtrar los retos con unas reglas determinadas, así como si el filtrado es correcto.
Prerrequisitos	Debe existir mínimo un reto dado de alta en el sistema
Entrada	El usuario, con cualquier rol y con la sesión iniciada, marca los filtros correspondientes para obtener retos que estén marcados con la etiqueta *Matemáticas* y/o *Estructuras de Datos*, que su dificultad sea inferior a *Extremo* y que contengan *palabras* en el título
Salida Esperada	Mosaico de retos filtrado de forma que únicamente alberga un reto: *De números a palabras*
Salida Obtenida	Mosaico de retos filtrado de forma que únicamente alberga un reto: *De números a palabras*

Cuadro 8.2: PCN-02: Filtrado de Retos

PCN-01	**Acceso a un Taller sin pagar su coste en Pases de Acceso**
Objetivo	Comprobar si el sistema permite a un jugador acceder a un Taller sin que el mismo exista en la lista de Talleres accedidos del usuario.
Prerrequisitos	Debe existir mínimo un reto que no haya sido adquirido por el jugador.
Entrada	El usuario, de tipo Jugador y con la sesión iniciada, intenta acceder con su navegador al enlace permanente del Taller.
Salida Esperada	Aparición de un modal de error que indica al usuario que no tiene acceso al mismo.
Salida Obtenida	Aparición de un modal de error que indica al usuario que no tiene acceso al mismo.

Cuadro 8.1: PCN-01: Acceso a un Taller sin pagar su coste en Pases de Acceso

PCN-03	**Alta de Evento incorrecta**
Objetivo	Comprobar que el sistema no permite al usuario con rol Organizador dar de alta un evento sin introducir las fechas de inicio y fin
Prerrequisitos	No tiene
Entrada	El usuario, con rol de Organizador y con la sesión iniciada, abre el modal de alta de Evento para rellenar todos los campos, salvo la fecha de inicio y fin del mismo. Acto seguido, hace clic en el botón *Enviar*
Salida Esperada	Mensaje de error en los campos de fecha: *Campos obligatorios*
Salida Obtenida	Mensaje de error en los campos de fecha: *Campos obligatorios*

Cuadro 8.3: PCN-03: Alta de Evento incorrecta

PCN-04	**Contacto de usuarios favoritos**
Objetivo	Comprobar que el usuario con rol ojeador puede contactar a la persona correcta.
Prerrequisitos	La lista de favoritos del ojeador no debe estar vacía, es decir, debe estar poblada por mínimo un usuario.
Entrada	El usuario, de tipo Ojeador, hace clic en el botón de Contacto de un determinado usuario al que quiere contactar.
Salida Esperada	Aparición de un modal con un formulario de contacto, cuyo contenido tendrá como destinatario el email del usuario al que contacta.
Salida Obtenida	Aparición de un modal con un formulario de contacto, cuyo contenido tendrá como destinatario el email del usuario al que contacta.

Cuadro 8.4: PCN-04: Contacto de usuarios favoritos por parte de los ojeadores

PCN-05	Intento de fraude en la compra de Pases de Acceso
Objetivo	Comprobar que el sistema no permite al usuario con rol Jugador gastar más Puntos de Experiencia de los que podría comprar.
Prerrequisitos	El usuario debe contar con una cantidad de Puntos de Experiencia inferior a 600
Entrada	El usuario, con rol de Jugador y con la sesión iniciada, intenta adquirir 40 Pases de Acceso (con un precio de 600 Puntos de Experiencia) teniendo menos puntos de los necesarios. Para ello, introduce 600 en el campo de Puntos de Experiencia, esperando poder recibir los 40 Pases en su perfil.
Salida Esperada	El campo se reinicia a su total de Pases, no permitiendo en ningún caso al usuario introducir un número mayor de dicha cantidad.
Salida Obtenida	El campo se reinicia a su total de Pases, no permitiendo en ningún caso al usuario introducir un número mayor de dicha cantidad.

Cuadro 8.5: PCN-05: Intento de fraude en la compra de Pases de Acceso

Capítulo 9

Conclusiones

En este último capítulo se describen las posibles líneas de trabajo futuras que podría tener la plataforma y, finalmente, una serie de conclusiones personales.

9.1. Líneas de trabajo futuras

Son muchas las posibilidades que una plataforma como la planteada puede tener en un entorno de producción real, además de ser muchos los horizontes a los que se puede ampliar una plataforma así. Destaco cuatro posibles líneas futuras, que creo que serían interesantes de cara a incluir si se hiciera una versión futura de ECodium:

9.1.1. Compatibilidad con más lenguajes en los retos

La primera funcionalidad adicional es casi obvia. Permitir retos en lenguajes que no sean JavaScript, de modo que el *backend* sea capaz de detectar, interpretar y ejecutar en un entorno de sandbox seguro múltiples lenguajes.

9.1.2. Migración de ERC20 a otros estándares más asequibles

El estándar ERC20 de la red Ethereum es uno de los estándares del mundo de los tokens de Blockchain con más tasas (*fees*). Sin duda alguna, estándares como el BEP-721, un estándar emergente, facilitan mucho el desarrollo de aplicaciones con este tipo de *tokens*

9.1.3. Aplicación móvil

Si bien la aplicación ya es completamente *responsive* y compatible con el móvil, no deja de ser una aplicación web. Se podría llegar a desarrollar una aplicación móvil nativa que permitiera al usuario conocer estadísticas, notificaciones y el estado actual de su perfil dentro de ECodium.

9.1.4. Colaboración con empresas tecnológicas y universidades

Mediante la colaboración y patrocinio con empresas tecnológicas y/o universidades, se podría lograr sustentar la página, ya que contaría con una serie de organizadores dando de alta nuevos contenidos, ojeadores buscando talento y a su vez, un nicho de jugadores, quienes podrían tener un aliciente adicional para entrar en ECodium.

9.2. Conclusiones personales

A nivel personal, como en todo buen proceso de aprendizaje, he disfrutado (y he sufrido) mucho desarrollando ECodium. Ya no sólo a nivel tecnológico, que también, al utilizar tecnologías emergentes, sino a nivel introspectivo, ya que me ha ayudado a fomentar mis capacidades de reinventarme a mí mismo, análisis, investigación y comunicación escrita.

Considero este proyecto un buen colofón para mi carrera en el Grado en Ingeniería Informática de Servicios y Aplicaciones y creo que sin duda alguna ha cumplido con creces su objetivo, que era aglutinar de alguna forma la mayor cantidad de contenidos aprendidos durante el Grado en un solo trabajo. Y a la vez, me movía mi pasión por la informática presente desde muy joven y las ganas de querer hacer algo que fuera ligeramente distinto a lo que se suele hacer normalmente.

Ha sido un trabajo largo y tedioso en muchos momentos, pero es que no deja de ser un Proceso de Desarrollo Software más o menos completo, con todas sus fases y metodologías aplicadas de la manera más detallada posible.

Los tres meses de retraso con respecto a la planificación inicial han sido la principal consecuencia de desarrollar un proyecto de este calibre a la vez que realizaba prácticas curriculares en empresa, y después prácticas extracurriculares hasta que finalmente, estoy trabajando como contratado, teniendo así mi primera experiencia laboral, por lo que compaginarlo ha sido realmente complicado. Y con un verano de por medio.

Pero en definitiva, me quedo con lo bueno de este proyecto. Con lo disfrutado, con lo sufrido, pero sobre todo: con lo aprendido. Porque creo que realmente, el mayor objetivo de todos era ese: aprender.

Apéndice A

Manual del Usuario

En este apéndice se pretende recoger un pequeño manual para cada tipo de usuario. Si alguna información mostrada no es congruente entiéndase que las capturas de pantalla son meramente ilustrativas y han sido extraídas de una batería de pruebas, no perteneciendo a ningún usuario real de ECodium.

A.1. Manual del Jugador

Bienvenido a ECodium, jugador. Tu misión es, sin duda, subir niveles hasta convertirte en una Leyenda. ¿Cómo se suben niveles? ¡Ganando Puntos de Experiencia mediante retos de programación, eventos y talleres! Los Puntos de Experiencia, además de servir para determinar tu nivel, pueden canjearse por Pases de Acceso a eventos y talleres o por TAL, nuestro propio token de Ethereum.

Si has iniciado sesión correctamente, habrás llegado aquí. A la **página de Inicio**, que ofrece un *slider* de bienvenida personalizado con novedades, una serie de **estadísticas** numéricas, así como una **gráfica** de tu evolución anual en la plataforma

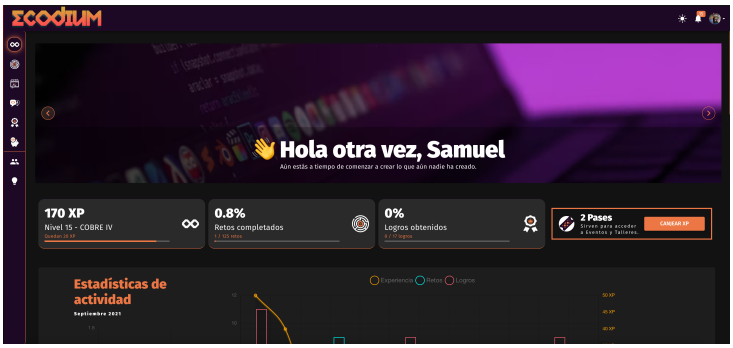


Figura A.1: Página de inicio de ECodium

Accediendo, en el menú a la página de **Retos**, llegas al **mosaico de retos**, donde puedes visualizar de una manera gráfica qué retos te ofrecen más o menos Puntos de Experiencia de recompensa:

Figura A.2: Mosaico de Retos

Asimismo, maximizando la ventana de "Filtrar retos", puedes filtrar el mosaico de retos según tus necesidades. Estas *"ventanas maximizables"* están presentes en gran parte de la plataforma:

Figura A.3: Filtrado de retos

Para resolver un reto, haz clic en Ïntentar resolver". Te llevará a nuestro Editor de JavaScript, donde podrás programar (y probar) tu solución hasta que vayas superando las pruebas del reto. ¡Recuerda que si eres el primero en resolverlo serás el **conquistador** de ese reto!

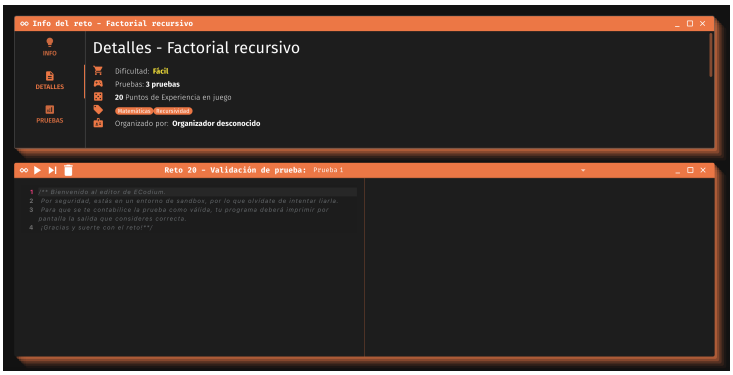


Figura A.4: Resolución de retos

En **Eventos** aparece un calendario, personalizable por meses, semanas, días o agenda. En este calendario aparecerán los eventos y, si cumples el nivel mínimo de acceso y pagas los Pases pertinentes, podrás añadir candidaturas sincronizando un repositorio de GitHub y subiendo tus propios archivos.

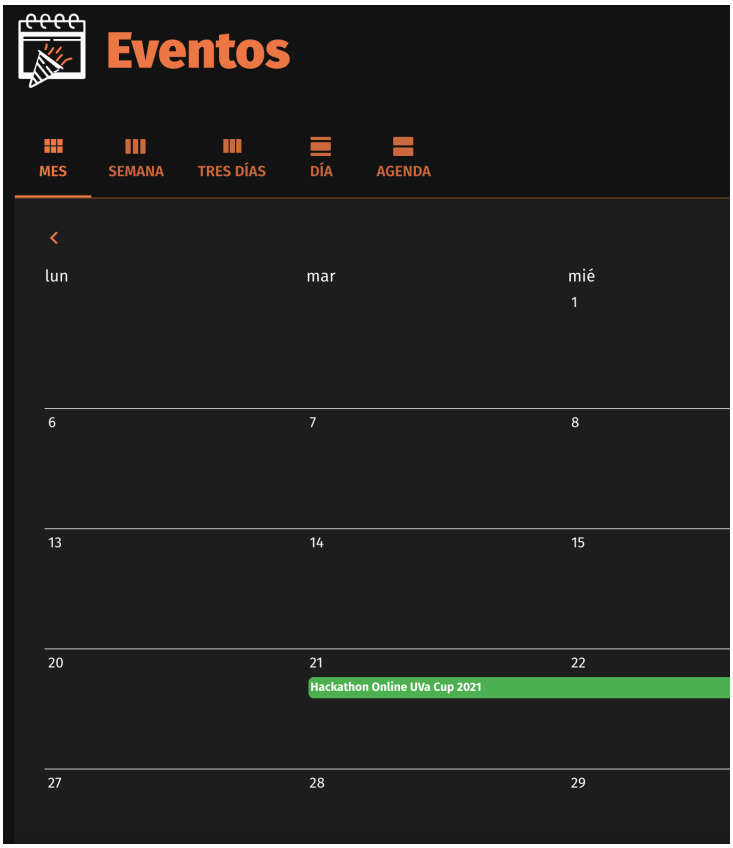


Figura A.5: Calendario de Eventos

Figura A.6: Info del Evento

Los **Talleres** funcionan de un modo similar: si tienes los Pases pertinentes y quedan plazas, puedes inscribirte en el taller y acceder a sus contenidos.

Figura A.7: Vista de un Taller en el listado

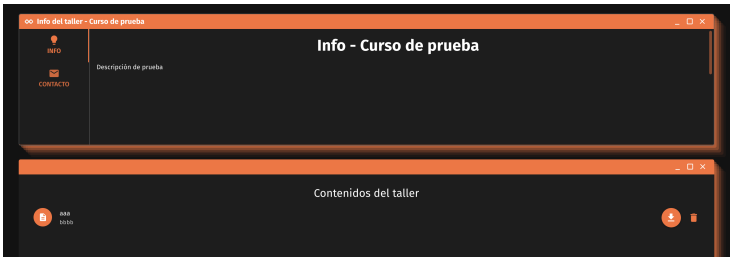


Figura A.8: Página del Taller y sus contenidos

Irás ganando **logros** conforme avances en la aplicación. Puedes ver tus logros (coloreados) o los logros pendientes (en blanco y negro) en la página de Logros

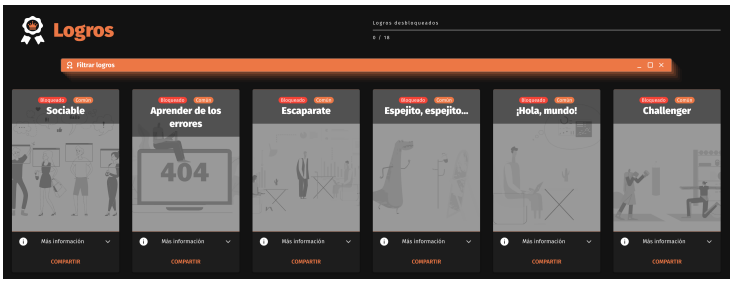


Figura A.9: Vista de Logros

En el **Banco** puedes intercambiar experiencia acumulada por TAL o experiencia actual por Pases, así como consultar tus transacciones. Es normal que si no usas una cartera de Ethereum aparezca este error indicando que Metamask (la cartera que aconsejamos) no está funcionando, para ello tienes que instalarlo y recargar la página y así operar con normalidad:

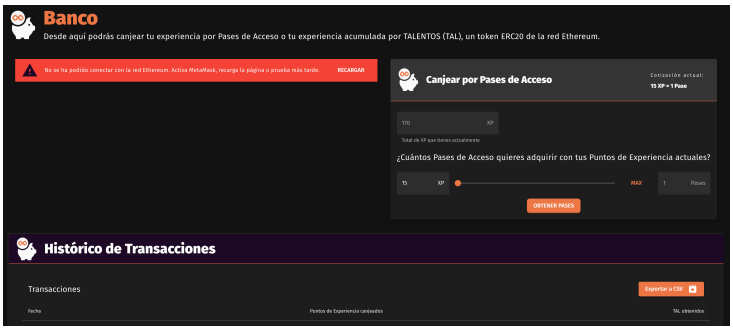


Figura A.10: Banco de ECodium - Error de Metamask

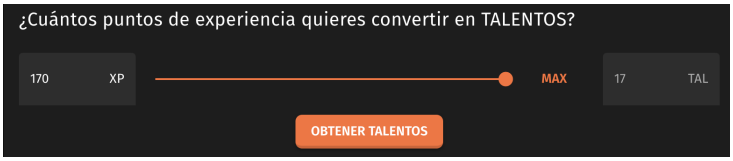


Figura A.11: Banco de ECodium - Selector de cambio

Sin duda, la **Comunidad** es una de las características fundamentales de la plataforma. Mediante esta página podrás consultar el ranking de mejores jugadores y consultar tu perfil propio:

Figura A.12: Comunidad de ECodium

En la última página, **Sobre ECodium**, tienes diversas herramientas informativas así como un formulario de contacto:

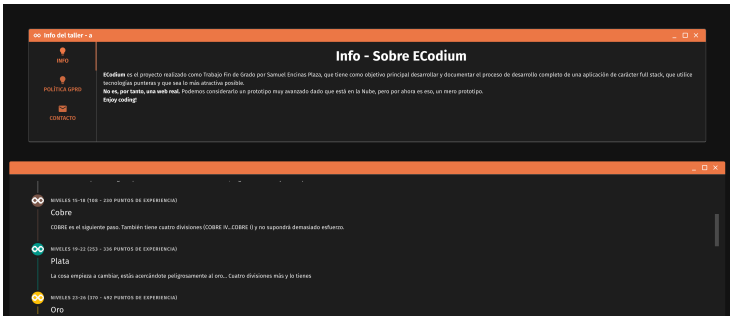


Figura A.13: Páginas informativas

Por último, nuestro sistema de notificaciones te avisa en tiempo real de los eventos que sucedan o hayan sucedido mientras no estabas conectado.

Figura A.14: Sistema de notificaciones de ECodium

A.2. Manual del Organizador

Como Organizador, tu objetivo es organizar retos, talleres y eventos que resulten atractivos a los jugadores. Serás el encargado de todo, y lo crearás mediante modales como estos:

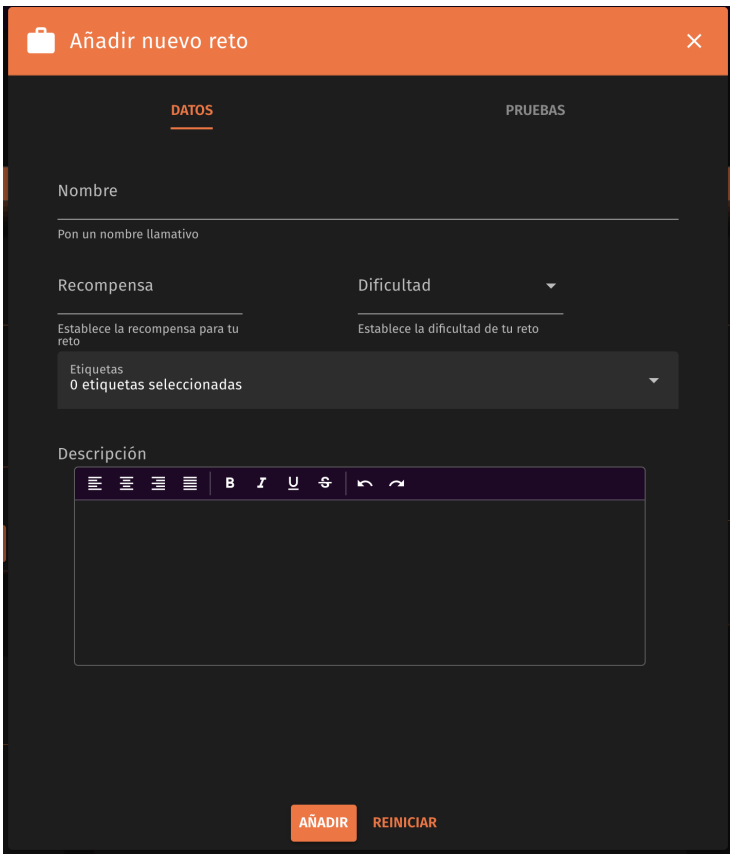


Figura A.15: Modal de creación de Retos

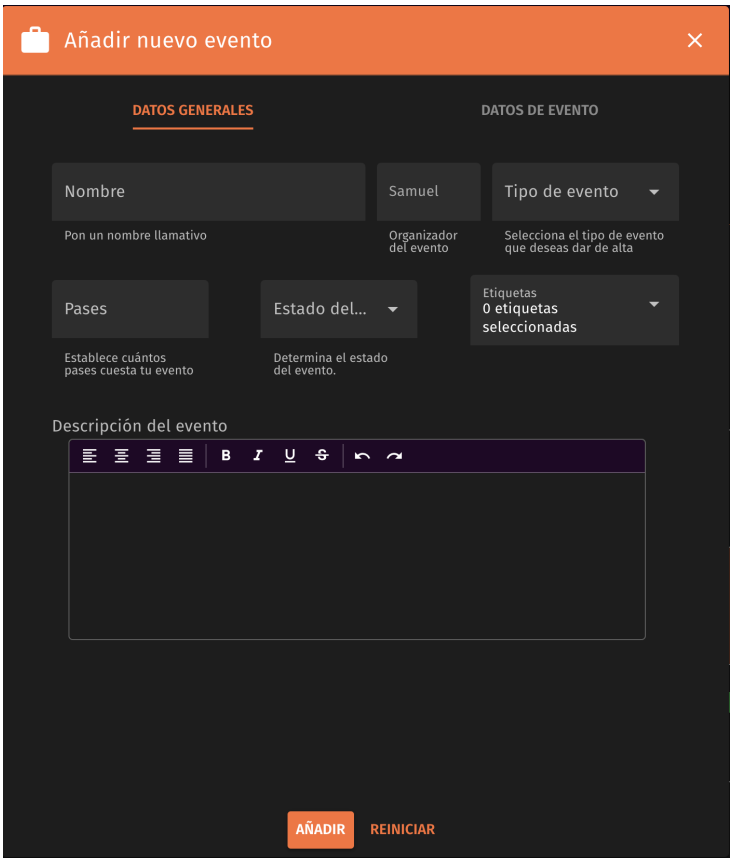


Figura A.16: Modal de creación de Eventos

Asimismo, tendrás que seleccionar las candidaturas ganadoras para así premiar a sus respectivos autores. En el caso de que el evento lo vote un jurado, serás tú el encargado de subir los votos del jurado mediante el modal pertinente:

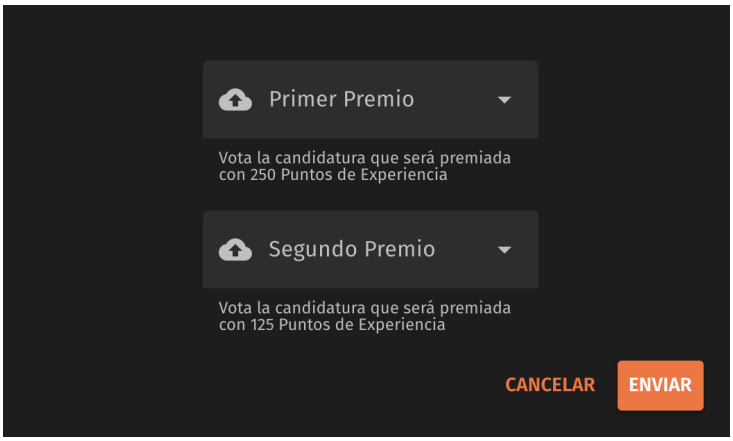


Figura A.17: Modal de Cierre de Eventos

A.3. Manual del Ojeador

Como ojeador, tienes acceso a todo sin pagar Pases de Acceso y así poder ver cómo se comportan nuestros jugadores. De manera adicional, puedes agregar a Favoritos a cualquier jugador para poder contactarle vía mensajería interna más rápidamente:

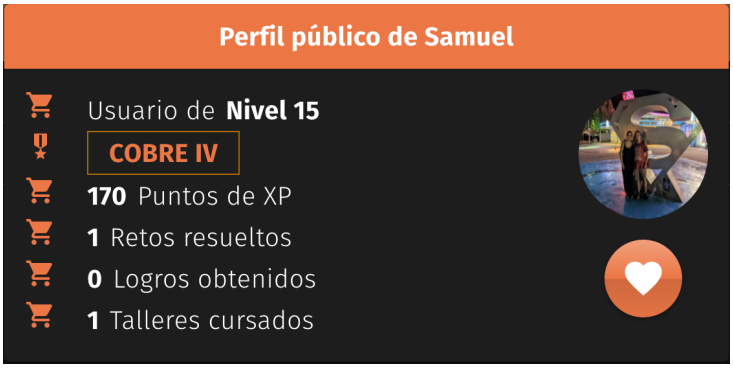


Figura A.18: Añadir a favoritos

A.4. Manual del Administrador

Por último, como administrador, tienes todos los permisos con el fin de probar y desarrollar nueva funcionalidad. El rol de administrador debe ser único y además conocerá como se instala y despliega la aplicación en local, así que ese es su manual:

Como requisitos, imprescindible tener instalado node.js, y sus gestores de paquetes npm y yarn. Además, si quieres ejecutarlo en local, deberás **cambiar los *endpoints*** de *api.ecodium.dev* a *localhost:3000* y de *ecodium.dev* a *localhost:8080*

Para instalar y ejecutar el *backend*: Lo primero es ejecutar **npm install** para instalar todos los paquetes de las dependencias. Este proceso tardará unos cuantos minutos, y ya después puedes ejecutar **nodemon server.js** y ejecutar el entorno de *backend* en local.

Para instalar y ejecutar el *frontend*: De manera análoga, se procede a instalar los paquetes de dependencias del *frontend*, en este caso con el otro gestor de paquetes: **yarn install**. Si el proceso concluye de forma satisfactoria, podrás ejecutar **quasar dev** para arrancar el *frontend* en entorno de desarrollo o **quasar build** para compilar una versión estática del *frontend*

Bibliografía

[1] Verónika Abramova, Jorge Arnardino y Pedro Furtado. *Experimental evaluation of NoSQL databases*. Jun. de 2014.

[2] Dr. Gerard Briscoe. *Digital Innovation: The Hackathon Phenomenon*. 2014.

[3] Alberto Hernández Chillón y col. *Visualización de Esquemas en Bases de Datos NoSQL basadas en documentos*. 2017. URL: https://biblioteca.sistedes.es/submissions/uploaded-files/JISBD_2017_paper_71.pdf.

[4] Andrada Fiscutean. *Want to learn programming? This startup pays you cryptocurrency to study Python*. 1 de abr. de 2018. URL: https://www.zdnet.com/article/want-to-learn-programming-this-startup-pays-you-cryptocurrency-to-study-python/.

[5] Carina Soledad González González. *Técnicas de gamificación aplicadas en la docencia de Ingeniería Informática*. Ene. de 2015. URL: https://www.researchgate.net/publication/299169806_Tecnicas_de_gamificacion_aplicadas_en_la_docencia_de_Ingenieria_Informatica.

[6] Daniel Graziotin. *The Dynamics of Creativity in Software Development*. URL: https://arxiv.org/pdf/1305.6045.pdf.

[7] María Nieves Pacheco Jiménez. *De la tecnología blockchain a la economía del token*. Feb. de 2019.

[8] Juan Cruz Martínez. "Create and Deploy Your Own ERC-20 Token on the Ethereum Network". En: *BetterProgramming* (2020). URL: https://betterprogramming.pub/create-and-deploy-your-own-erc-20-token-on-the-ethereum-network-87931fe4db20.

[9] Ana Torres Menárguez. "Por qué todo el mundo quiere hackathon". En: *El País* (2 de nov. de 2016). URL: https://elpais.com/elpais/2016/10/13/talento_digital/1476355605_011048.html.

[10] Hugo A. Mitre-Hernández, Lemus-Olalde Cuauhtémoc y Edgar Ortega-Martínez. *Estimación and control de costos en métodos ágiles para desarrollo de software: un caso de estudio*. Sep. de 2014. URL: https://www.researchgate.net/publication/270005877_Estimacion_y_control_de_costos_en_metodos_agiles_para_desarrollo_de_software_un_caso_de_estudio.

[11] Yusef Hassan Montero. *Experiencia de Usuario: Principios y métodos*. 2015. URL: https://yusef.es/Experiencia_de_Usuario.pdf.

[12] William Mougayar. *The Business Blockchain: Promise, Practice, and Application of the Next Internet Technology*. Jun. de 2016.

[13] Antoni Olivé. *The Conceptual Schema of Ethereum and of the ERC–20 Token Standard*.

[14] *Story Point based Cost Estimation*. 9 de abr. de 2018. URL: https://www.gdatasoftware.com/blog/story-point-based-cost-estimation.

[15] Adrián Alonso Vega. "5 razones para probar MongoDB". En: (16 de ago. de 2014). URL: https://adrianalonso.es/desarrollo-web/nosql/5-razones-para-probar-mongodb/.

[16] Kevin Werbach y Dan Hunter. *For the Win: How Game Thinking Can Revolutionize Your Business*. 2012.

[17] *Why You Should Become a Self-Taught Programmer?* 15 de mayo de 2021. URL: https://www.geeksforgeeks.org/why-you-should-become-a-self-taught-programmer/.