Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Plataforma de valor para clientes y organizaciones, basada en Blockchain.

Sebastián Daniel Tamayo Guzmán

La Laguna, 13 de septiembre de 2022

D. Cándido Caballero Gil, con DNI 42201070A profesor Titular de la Universidad de La Laguna, como tutor.
D. Sergio Díaz Martín, con DNI 78617234Z, Jefe de Datos & DevOps en Atlantis Technology, como cotutor.

CERTIFICAN
Que la presente memoria titulada: "*Plataforma de valor para clientes y organizaciones, basada en Blockchain*", ha sido realizada bajo su dirección por **D. Sebastián Daniel Tamayo Guzmán** con DNI 43848827V.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 13 de septiembre de 2022.

Cualquier mensaje con respecto a este trabajo puede ser enviado al siguiente correo: alu0101131108@ull.edu.es.

Agradecimientos a mi familia y amistades por apoyarme a lo largo de toda mi etapa educativa y universitaria. También al tutor y cotutor de este proyecto por orientarme durante su realización.

Resumen

El objetivo de este trabajo ha sido tener una primera toma de contacto con el desarrollo de sistemas basados en blockchain. De esta manera, se aborda tanto el estudio de las bases teóricas que respaldan esta tecnología como el diseño y desarrollo de un sistema basado en blockchain. Se ha propuesto una plataforma de gestión de reglas de negocio, concretamente una plataforma de valor donde cualquier organización puede crear, de manera individual o colaborativa, planes de recompensa personalizados para sus clientes. El sistema está destinado para ser usado en la red de Ethereum, aunque sea de fácil portabilidad a otras redes compatibles. Con el propósito de demostrar un ejemplo completo de uso, también se incluye un cliente web que permite interactuar con la plataforma.

Palabras clave: Blockchain, Ethereum, Web 3.0, Contratos Inteligentes, Aplicaciones descentralizadas, plataforma de valor.

Abstract

The aim of this project has been to have a first point of contact with the development of blockchain-based systems. In this way, both the theoretical basis that supports this technology and the development of a blockchain-based system are addressed. A business rule management platform has been proposed, specifically a value platform where any organization can create, individually or collaboratively, customized reward plans for their customers. The system is intended to be used on the Ethereum network, although it is easily portable to other compatible networks. In order to demonstrate a complete example of use, a web client that allows interaction with the platform is also included.

Key words: Blockchain, Ethereum, Web 3.0, Smart Contracts, Decentralized Apps, Value Platform.

Índice General

Índice de Figuras y Tablas.

Capítulo 0 – Introducción

Blockchain

Las redes distribuidas basadas en blockchain son un tipo de tecnología emergente que está evolucionando mucho en los últimos años. Dicha tecnología es de elevado interés para la comunidad investigadora y para los desarrolladores, dado que permite crear una gran variedad de nuevas soluciones y mejoras descentralizadas para usuarios y empresas. De hecho, se ha registrado un crecimiento exponencial en su adopción a lo largo de los últimos años.

Estos sistemas actúan como bases de datos cuya información es segura e inmutable, pues está protegida mediante técnicas de encriptado y además, se encuentra replicada en todos los nodos que conforman la red. Esta permite realizar transacciones de valor entre sus usuarios que quedan registradas en ella. Estas transacciones pueden ser monetarias (criptomonedas) o de otra naturaleza (bienes, información, servicios, etc).

Un protocolo de consenso asegura la consistencia de los datos, al orquestar las comunicaciones entre sus participantes de manera que se eviten conflictos y sea posible evitar acciones con intención fraudulenta. De esta manera, se consigue una fuente de verdad única y confiable que no está sujeta, en gran medida, a una autoridad central con la capacidad de alterar su contenido.

Adicionalmente, la aparición de los contratos inteligentes extiende las capacidades de esta tecnología de manera significativa. Se trata de una lógica computacional con un fin específico, representada mediante códigos que se almacenan y ejecutan dentro de una cadena de bloques y permiten la interacción con ella. De esta manera se pueden crear servicios basados en estos contratos inteligentes, cuya funcionalidad gozará de las propiedades intrínsecas de blockchain: inmutabilidad, seguridad, transparencia y descentralización.

Propuesta

En este trabajo se plantea el análisis, diseño y desarrollo de un sistema respaldado por una serie de contratos inteligentes que se alojen en una red de bloques. Este actuará como un gestor de reglas de negocio para una plataforma de valor entre clientes y organizaciones. Además, deberá disponer de una capa de servicios mediante la cual se generen los eventos que quedarán registrados en la base de datos distribuida. Por último, se propone construir un cliente web que demuestre de manera interactiva el funcionamiento de la plataforma.

Como caso de uso se tomará como referencia un programa de fidelización a clientes. A diferencia del planteamiento convencional para estos programas, este sistema no será exclusivo de ninguna de las organizaciones que participen. A cambio, actuará como un intermediario entre las empresas y el conjunto formado por los clientes de todas ellas. De este modo, cualquier negocio podrá integrar su plan de recompensas disponiendo de la posibilidad de establecer criterios conjuntos con cualquier otra organización adscrita al programa.

Por otro lado, las recompensas serán dadas en forma de tokens digitales que podrán ser canjeados por productos o divisas digitales. De este modo, se aprovecharán las ventajas de los contratos inteligentes automatizando estas tareas de manera segura y transparente.

Justificación

La fidelización de clientes es una técnica que usan las empresas y que se viene produciendo desde hace muchos años. En la actualidad cada vez es más frecuente encontrarnos con diferentes programas de fidelización. De hecho, la gran diversidad de programas suele abrumar a los clientes consiguiendo que muchos no participen en dichos planes. Por otro lado, estos sistemas suelen suponer costes más altos de mantenimiento para las empresas y obtienen un resultado menos efectivo, a parte de poder estar sujeto a errores y/o ser objeto de ataques maliciosos.

Con el planteamiento propuesto, los clientes solo tendrán que formar parte de un único programa para beneficiarse de la fidelización de varias empresas, viéndose así más motivados a participar en él.

Por el lado de las empresas, al quedar todas las transacciones registradas en la red, podrán realizar fácilmente un seguimiento de los datos de sus consumidores y así tomar decisiones de negocio más acertadas basadas en un análisis de estos datos. Adicionalmente, en el caso de empresas que pretendan ampliar sus programas de fidelidad con recompensas colaborativas, solo tendrán que realizar una única integración con la plataforma de valor, en vez de una por cada organización con la que lleguen a acuerdos. De todas formas, no será necesario disponer previamente de un programa de fidelización para poder suscribirse a la plataforma.

Por estos motivos, se considera que utilizar un plataforma basada en blockchain para el caso de uso propuesto es muy acertado y ofrece una serie de ventajas frente a los modelos tradicionales que rigen este tipo de plataformas. De hecho, existen artículos que describen proyectos similares y también exponen las ventajas de utilizar esta tecnología en el área de los programas de fidelización, ver entrada [14] de la bibliografía.

Plan de trabajo

Este proyecto será dividido en tres fases principales: análisis, diseño y desarrollo. Durante el transcurso de cada una de ellas se irá generando una memoria que documente el proceso a medida que se avance. Además, se propone usar una metodología Agile con el fin de llegar a los objetivos planteados de forma iterativa, teniendo seguimiento y control por cada una de las partes que conforman el equipo de proyecto.

En la primera fase, se realizará un estudio de los temas principales que aborda el proyecto con el fin de familiarizarse con ellos. Principalmente se investigará acerca de la tecnología blockchain, sus orígenes, los conceptos teóricos que garantizan sus características, sus aplicaciones y algunas de las diferentes cadenas de bloques que están en funcionamiento en el presente. Además, será necesario recabar información respecto a la construcción de aplicaciones web para facilitar una interfaz mediante la cual se permita la interacción con el sistema.

La segunda fase cubrirá inicialmente, el diseño del aspecto funcional del sistema blockchain sobre el cual se basará la plataforma. Esto incluye la identificación de los distintos participantes del sistema, el formato de las reglas de negocio, el formato de los planes de recompensa y de su ciclo de vida, así como el mecanismo para generarlos y mantener un registro para su supervisión. Además, se irán formulando los tests que pondrán a prueba los contratos.

A continuación, se diseñará una aplicación web que facilite la interacción con los contratos inteligentes. Por un lado, tendrá que permitir a las empresas la creación individual o colaborativa de planes para recompensar a sus consumidores, así como su puesta en marcha y el seguimiento de los datos asociados a estos planes y sus clientes. Por otro lado, permitirá a los clientes la supervisión de sus recompensas en forma de tokens para cada uno de los planes a los que estén suscritos. Para concluir esta fase, habrá que seleccionar el conjunto de herramientas de desarrollo (red distribuida, lenguajes, frameworks, librerías) que se consideren adecuadas para la implementación en la siguiente etapa.

Para finalizar, en la tercera fase se implementarán los diseños generados en el ciclo previo con las herramientas de desarrollo elegidas. Cabe resaltar que se trata de un prototipo, y por tanto algunos elementos serán simulados, por ejemplo la cadena de bloques a utilizar probablemente será una red local o de prueba cuyo coste es ficticio pero simula el de la red real.

Capítulo 1 - Antecedentes

Sistemas blockchain

Historia

Los fundamentos teóricos nacen de los investigadores Stuart Haber y W. Scott Stornetta en 1991, al publicar un artículo que describe un método para asegurar la inmutabilidad de las marcas temporales en registros digitales. Este diseño no es lo que actualmente se conoce como blockchain, pero fueron las ideas iniciales que más tarde evolucionaron en esta tecnología.

El primer punto de inflexión tiene lugar en 2008, cuando se publica un artículo en internet bajo el pseudónimo *Satoshi Nakamoto*. En él se introduce una nueva moneda digital, el Bitcoin. Además, se describe el funcionamiento de sus componentes principales, justificando sus ventajas principales: seguridad, rapidez, privacidad y descentralización.

Tras nacer la primera aplicación que utiliza una red distribuida de cadenas de bloques, continuaron apareciendo distintas implementaciones con diversos propósitos y una gran variedad de aplicaciones que utilizan estas redes aprovechando sus características.

Conceptos teóricos

- Idea general

Una blockchain o cadena de bloques, es una base de datos que crece continuamente y además actúa como una lista de registros llamados bloques. El primer bloque de la cadena se conoce como génesis y el resto se van enlazando formando una cadena. Este enlace está respaldado criptográficamente, pues utilizan un identificador que se genera a partir de su contenido y cada bloque almacena el identificador del predecesor en la cadena. Véase la Figura 1.

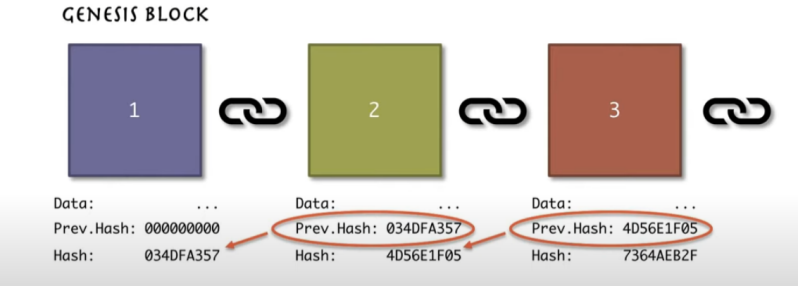

Figura 1: Idea general de una cadena de bloques

- Hashing

El mecanismo criptográfico encargado de generar los identificadores se conoce como función hash. Estas reciben uno o varios datos de entrada y los convierte en un rango de salida finito. Normalmente tanto la entrada como la salida de estas funciones son cadenas de texto y la salida es de un tamaño preestablecido.

Existen varios algoritmos de hashing pero todos comparten una serie de características. En primer lugar son unidireccionales, es decir que partiendo de un hash no se puede obtener el archivo de entrada mientras que de un archivo siempre se podrá computar su identificador. También son deterministas, esto asegura que ante una misma entrada, el algoritmo siempre devolverá la misma salida. Están sujetos al efecto avalancha, esto se produce si al realizar cualquier modificación por muy pequeña que sea a la entrada del algoritmo, el resultado será completamente diferente e independiente del anterior. Además, su tiempo de cómputo ha de ser rápido y tienen que ser resistentes ante colisiones, es decir que dos entradas produzcan la misma salida.

En concreto se ha estudiado el algoritmo SHA256, pues su uso está ampliamente extendido en el ámbito del blockchain. El nombre proviene de *Secure Hash Algorithm* y 256 es el número de bits que ocupa su resultado. Este acepta como entrada cualquier archivo digital y devuelve un hash, es decir una cadena de 64 caracteres que identifica el archivo de entrada. La probabilidad de que ocurra una colisión de forma accidental es de 4.3 * 10E-16.

- Registros inmutables

Como se menciona anteriormente, una de las características principales del blockchain es que los datos almacenados en sus bloques son inmutables. Esto se debe a que si el contenido de un bloque cambiase, ya sea por error o de manera intencionada, cambiaría su hash y habría que modificarlo en el apartado correspondiente del siguiente bloque, evento que repercutirá en un cambio de su hash, desencadenando la necesidad de modificar todos los bloques de la cadena posteriores al que se pretendía modificar inicialmente.

- Redes distribuidas P2P

Los sistemas blockchain son redes distribuidas de muchos clientes o nodos, que actúan como iguales, repartidos por el mundo. Estos clientes son programas que cualquier persona puede correr y su tarea principal es la validación de nuevos bloques que soliciten ser añadidos a la cadena.

Cada nodo tiene una copia actualizada de la cadena de bloques y verifica constantemente su validez. Esto quiere decir que para conseguir un ataque

malicioso no solo haría falta modificar el bloque objetivo y todos sus sucesores, sino que habría que conseguir esa modificación en todos los nodos de la red al mismo tiempo. Por ello, se afirma que un ataque solo podrá tener lugar si quien lo realiza posee el control de más de la mitad del poder de validación total de la red.

Además, cuando un nodo válida un nuevo bloque a ser añadido, lo notifica a sus nodos vecinos, quienes también lo verificarán y en caso positivo remitirán la modificación hasta que toda la red se actualice. En caso de que un cliente tenga información inválida, su registro de la cadena será sobreescrito por una copia de sus nodos vecinos. Estas entre otras características, forman parte del protocolo de consenso.

- Protocolos de consenso

Una blockchain es una base de datos distribuida, lo que implica que ya no existe un servidor que garantice la integridad de los datos. En cualquier blockchain existe un protocolo de consenso que es el que rige las interacciones entre los nodos para asegurar la consistencia de los registros almacenados en la cadena. En este protocolo los nodos se ponen de acuerdo sobre el estado de los bloques y una vez que un bloque es aceptado por el consenso es añadido a la cadena y se considera que es inmutable. Además debe garantizar la integridad de la cadena a pesar de la falla de uno o más nodos.

Este problema se aborda de forma distinta en las diversas implementaciones de redes blockchain. Por ejemplo, algunas implementaciones utilizan un consenso centralizado mientras que otras implementan un mecanismo para detectar y tratar los desacuerdos. Los dos modelos de protocolo de consenso distribuido más utilizados son Proof of Work (PoW) y Proof of Stake (PoS). La diferencia principal entre estos dos protocolos reside en cómo se elige el nodo que validará el siguiente bloque, y por consiguiente en el método para incentivar a las personas a participar en la red como nodos.

En el caso del modelo PoW, se requiere solucionar un problema criptográfico para añadir un bloque nuevo en la cadena. De esta manera, el primer nodo que consiga resolver dicho problema será elegido para validar el bloque en cuestión. Esta competición entre clientes es lo que se conoce comúnmente como minería, pues la recompensa es una tasa económica que aporta quien solicite la adición del nuevo bloque. Este planteamiento genera un grave problema, pues todos los nodos de la red están constantemente explotando su potencia computacional al máximo para ganar dicha competición, lo cual se traduce en un gasto energético inmenso.

En los sistemas PoS el individuo necesario para validar el siguiente conjunto de bloques será elegido "por sorteo" mediante un algoritmo con un componente de aleatoriedad. Además cada candidato tendrá que poner temporalmente una cantidad de tokens a disposición de la red, en estado de *stake*. Estos servirán como respaldo del buen comportamiento del validador. De esta manera, el algoritmo elegirá más frecuentemente aquellos candidatos que dispongan de un mayor respaldo y por tanto, los beneficios serán directamente proporcionales a esta cantidad. En contraparte y a fin de proteger la red, cuando se detecta algún comportamiento sospechoso o fraudulento por parte de un nodo, se penaliza eliminando una fracción del capital que haya puesto en estado de *stake*. Al no existir una competencia computacional entre los nodos, se solventa el problema del gasto energético presente en el modelo PoW.

Aplicaciones

Su principal aplicación ha sido sin duda en el campo de la economía, inicialmente con la aparición de las criptomonedas. Con ellas se consigue agilizar el procesamiento de cualquier transacción a nivel mundial, evitando su paso por entidades financieras. Además, han ido surgiendo una serie de herramientas financieras que dan lugar al ecosistema del *Decentralized Finance* (DeFi). Quienes apuestan por estas herramientas afirman que ofrecen un mayor nivel de seguridad, privacidad, transparencia, control y rapidez a menor costo que las finanzas tradicionales y por tanto son capaces de reemplazar por completo a las instituciones financieras actuales.

Por otro lado, existen muchas más formas de aplicar esta tecnología en prácticamente cualquier sector. La garantía de consistencia y fiabilidad que ofrecen estas bases de datos las hacen ideales para registros de la administración pública, como son los de propiedad o autoría. De manera similar, existe la posibilidad de mejorar el sistema de identidad digital con el uso de esta tecnología.

Adicionalmente, cabe citar el sector logístico (trazabilidad y gestión de mercancías), el energético (mejora del sistema minimizando intereses particulares), el sanitario y farmacéutico (historiales, gestión médica, trazabilidad de los medicamentos), la industria audiovisual (gestión de autoría y propiedad), el turismo (gestión de reservas, tarifas, contrataciones) y la industria 4.0 (registro seguro en tiempo real de dispositivos IoT).

Principales redes blockchain de propósito general

Ethereum

Se trata de una plataforma de código abierto, pionera en la implementación de contratos inteligentes y por tanto del desarrollo de aplicaciones descentralizadas. Debido a su aparición temprana, cuenta con el mayor ecosistema de herramientas de desarrollo y aplicaciones. Según la fundación de Ethereum, la red cuenta con un total de 7.027 nodos participando en el procesamiento de transacciones en fecha de redacción. Ver entrada [19] de la bibliografía.

Ethereum funciona de manera descentralizada a través de la *Ethereum Virtual Machine* (EVM). Se trata del entorno de ejecución de las transacciones de la red, que además permite la gestión y funcionamiento de una serie de funcionalidades añadidas como son los contratos inteligentes. Con el fin de facilitar la interacción con la EVM se creó *Solidity*, un lenguaje de programación Turing completo, específico de la plataforma.

El término *gas* está asociado al coste de procesamiento e inclusión en la cadena de una transacción. Esta tasa dependerá principalmente del coste computacional que conlleva dicha transacción, la prioridad que establezca el remitente y la demanda de procesamiento en la red.

Recientemente, se ha identificado un problema de escalabilidad grave, pues en ciertos periodos de tiempo en los que ha habido un aumento significativo de las transacciones y por tanto de su demanda de procesamiento, se han disparado estos costes transaccionales resultando en muchos casos en la no rentabilidad de uso de esta red.

Por otra parte, inicialmente Ethereum se planteó con un protocolo de consenso de tipo *Proof of Work*, aunque tras analizar el problema del gasto energético y la aparición de distintos modelos, se decidió iniciar la transición a un planteamiento *Proof of Stake*.

Este cambio forma parte de una gran actualización cuyo resultado sería inicialmente apodado Ethereum 2.0, aunque poco después en respuesta a una serie de problemas causados por este apodo, se decidió cambiar la convención de nombres, dividiendo esta gran actualización en tres fases: The Beacon Chain, The Merge y Sharding. El objetivo fundamental de estos cambios es solventar el problema energético y optimizar la red de manera que se puedan procesar muchas más transacciones por unidad de tiempo, manteniendo los costos transaccionales dentro de unos límites razonables.

Solana

Solana es una red de blockchain de código abierto, de alta velocidad y bajo costo, que tiene como objetivo mejorar la escalabilidad y la eficiencia de la cadena de bloques. A fecha de redacción, consta de 1,967 nodos y aproximadamente 10.000 millones de transacciones totales. A diferencia de Ethereum, Solana se enfoca en la rapidez y escalabilidad mediante la introducción de una serie de mejoras en la arquitectura, el protocolo y la infraestructura.

En cuanto a la arquitectura, Solana introduce un elemento en el protocolo de comunicación, la "Gossip Network" que permite que los nodos de la red intercambien información entre sí de forma constante. Además, Solana utiliza un protocolo de consenso llamado "Proof of History" que permite que los bloques se firmen y verifiquen de forma simultánea, lo que acelera el proceso de validación y confirmación de las transacciones. Como consecuencia, Solana procesa una media de 2.000 transacciones por segundo, a un coste medio de 0.00025 $.

En cuanto a la complejidad de desarrollo, parece haber un consenso general que concluye que ésta es bastante alta en la red de Solana y puede resultar abrumador si no se ha trabajado con herramientas similares anteriormente. Como consecuencia existen algunas herramientas que abstraen una gran porción de la lógica de bajo nivel inherente a Solana, para facilitar el desarrollo de los contratos inteligentes. Un buen ejemplo es el framework Anchor, pues ofrece una interfaz amigable y cargada de herramientas para el desarrollo, testing, debugging y despliegue de contratos inteligentes, mediante el lenguaje de propósito general Rust.

Cardano

Cardano es una cadena de bloques de código abierto, cuyo objetivo es crear una red más segura y escalable que las anteriores. Una de sus principales diferencias características es que el token ADA que alimenta la red está incluido de manera nativa en el núcleo del sistema de Cardano, a diferencia de otras redes como la de Ethereum que implementan los tokens mediante la capa de contratos inteligentes. Además, destaca por su enfoque en la investigación rigurosa y un respaldo teórico robusto.

Cardano cuenta con cuatro capas funcionales: Settlement Layer, que actúa como núcleo del sistema sobre el cual se integran el resto de componentes. Consensus Layer, donde tiene lugar el protocolo de consenso. Networking Layer, que se encarga de ejecutar el protocolo de comunicaciones entre los nodos. Y por último, Scripting Layer, una capa que proporciona un

lenguaje de bajo nivel para dotar a la red de contratos inteligentes. La familia de protocolos que conforman la red de Cardano se apoda Ouroboros y tienen un planteamiento Proof of Stake.

Cardano proporciona un abanico bastante grande de herramientas propias. Entre las más importantes está un entorno de desarrollo llamado Plutus, orientado a la creación de aplicaciones descentralizadas que interactúen con la red de Cardano a través de contratos inteligentes. Paralelamente, ofrece Marlowee, un lenguaje de dominio específico orientado a los contratos financieros.

Polkadot

Polkadot se diferencia de las redes anteriores principalmente en que su mayor objetivo es poder conectar múltiples cadenas de bloques entre sí que den lugar al entorno de Polkadot. Esta red sigue un esquema derivado de Proof of Stake, que se denomina Nominated Proof of Stake. Los desarrolladores de esta red afirman que este modelo es mucho más eficiente que PoW y a la vez más seguro que PoS.

Se trata de una plataforma de código abierto también dividida en capas. Esta arquitectura permite una mayor flexibilidad al permitir la interacción entre redes blockchain de distintos tipos.

La primera de ellas, Relay Chain, es la capa principal. Se trata de una cadena de bloques que gestiona la interacción entre las otras, ofreciendo funcionalidades de comunicación, transferencias y puentes entre ellas. La segunda capa está formada por Parachains. Se trata de cadenas de bloques que se asocian a una Relay Chain y por tanto, heredan sus características y funcionalidades. La tercera capa está formada por Parathreads. Se trata de cadenas de bloques no asociadas a una Relay Chain, pero que se puede establecer conexiones mediante un puente. Por último, la cuarta y última capa está formada por la Plataforma de Aplicaciones Web. Se disponen de varias herramientas para la construcción de aplicaciones descentralizadas. Por ejemplo Substrate, que permite la interacción con los nodos de la red, desarrollada principalmente en Rust.

Cabe destacar que solo algunas de las parachains existentes soportan el uso de contratos inteligentes. Aunque de manera adicional, se pueden aprovechar las funcionalidades de puente para interactuar con contratos que habiten en cadenas que si soporten los contratos inteligentes.

Capítulo 2 - Análisis y diseño

Diseño de la plataforma

Identificación de los participantes

El primer paso en el diseño de la plataforma consiste en identificar aquellas entidades que van a dar uso o desempeñar algún papel dentro del sistema. Tendremos dos grupos de participantes bien marcados.

Por un lado están los clientes: estos son los usuarios finales que se beneficiarán del ecosistema de planes de recompensa que existan en la plataforma. Una vez registrados en la plataforma, solo tienen que realizar las actividades recompensables asociadas a cada plan, para recibir las recompensas.

Por otro lado estarán las entidades: formadas por fundadores y notificadores. Los fundadores serán los encargados de la construcción y financiación de los planes de recompensa, pues tendrán que aportar una cantidad inicial de divisa digital al fondo común que será utilizado para las recompensas. Mientras tanto, los notificadores tendrán la responsabilidad de registrar en la plataforma las acciones recompensables de los clientes asociados al plan. El esquema de la Figura 2 muestra estas relaciones.

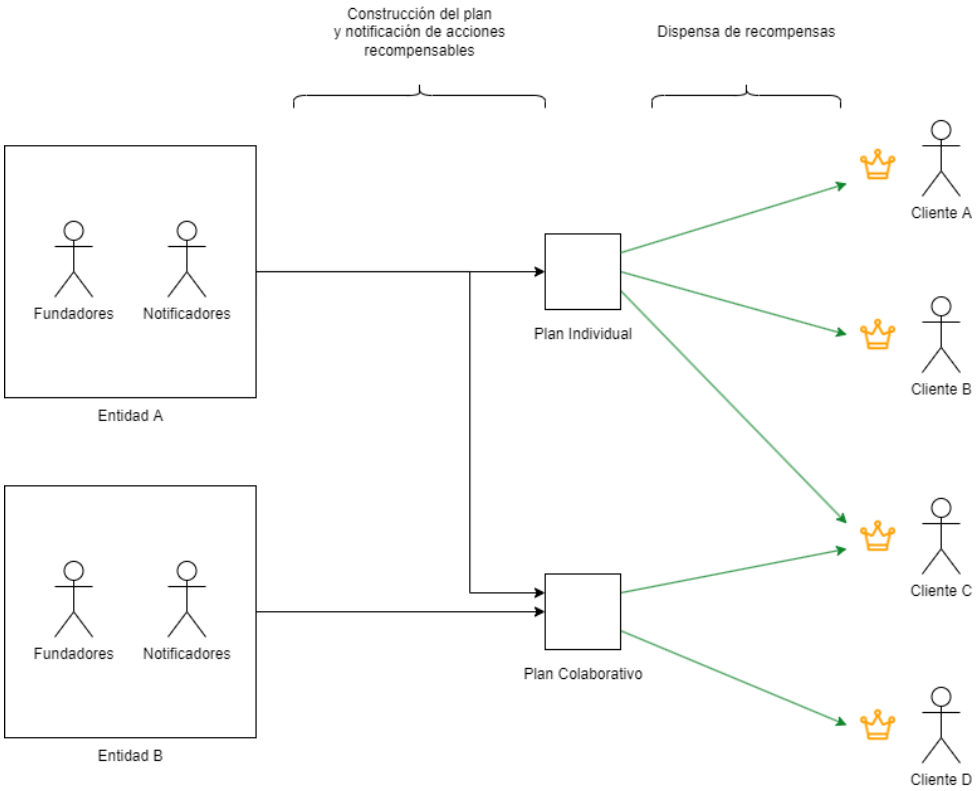

Figura 2: Esquema de los participantes de la plataforma.

Reward Center

El sistema gira en torno a los planes de recompensa, pero éstos necesitarán de algún elemento central que facilite su creación y disponga de un mecanismo para realizar un seguimiento global de los planes existentes y de los participantes del sistema. Se hará referencia a este elemento central como Reward Center. De esta manera, tendrá un registro de los planes, clientes y entidades existentes, siendo capaz de asociar un conjunto de planes a cada cliente y a cada entidad. Véase la Figura 3, que presenta un esquema de la información que gestiona el centro de recompensas.

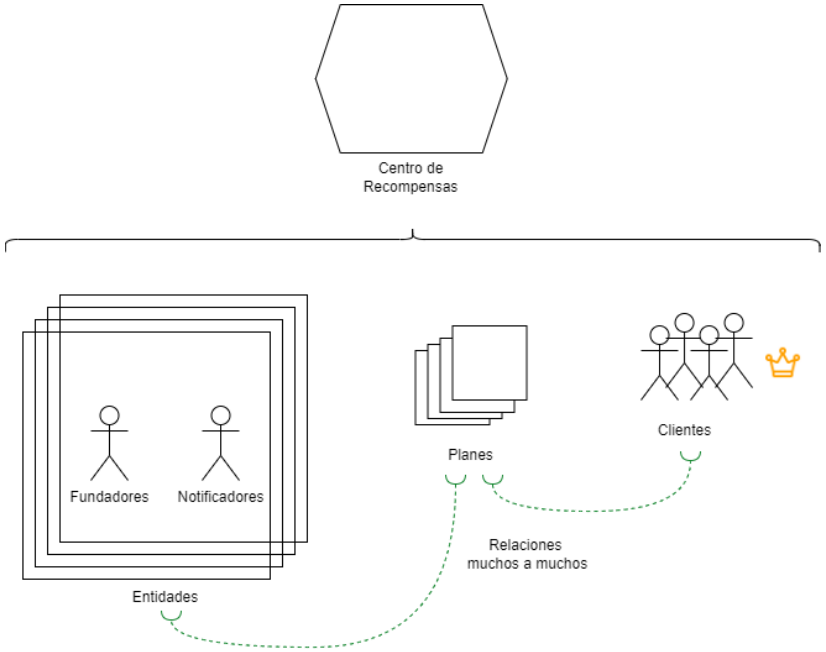

Figura 3: Registros del centro de recompensas.

Reward Plan

A través de Reward Center, una entidad puede crear un nuevo plan de recompensa, en el que además de fundador, desempeñará el rol de creador. Cada Reward Plan tendrá un ciclo de vida marcado por cuatro etapas con un propósito concreto. Véase la Figura 4.

La fase inicial es la de “Construction”, en la que se establecerán sus atributos principales, es decir el resto de fundadores, notificadores y reglas de recompensa. La siguiente etapa es la “Signing”, en la que tras haber construido y dado forma al plan, los fundadores lo firman aportando la cantidad de dinero asociada a cada uno. En caso de disconformidad por parte de algún fundador, se dotará de un mecanismo para que el resto pueda recuperar el aporte económico que pueda haber realizado y se pueda volver a iniciar la fase de construcción.

Una vez todos los fundadores hayan firmado, se transiciona a la fase “Active”, en el que el plan ya está listo para dispensar las recompensas en función de los registros generados por los notificadores. El plan se mantiene en este estado hasta que su fondo monetario se agota, momento en el que transiciona al estado de “Sleeping”. Se podrá reactivar el plan, aportando más fondos, o podrá ser reiniciado por su creador volviendo al mismo estado que en el momento de su creación.

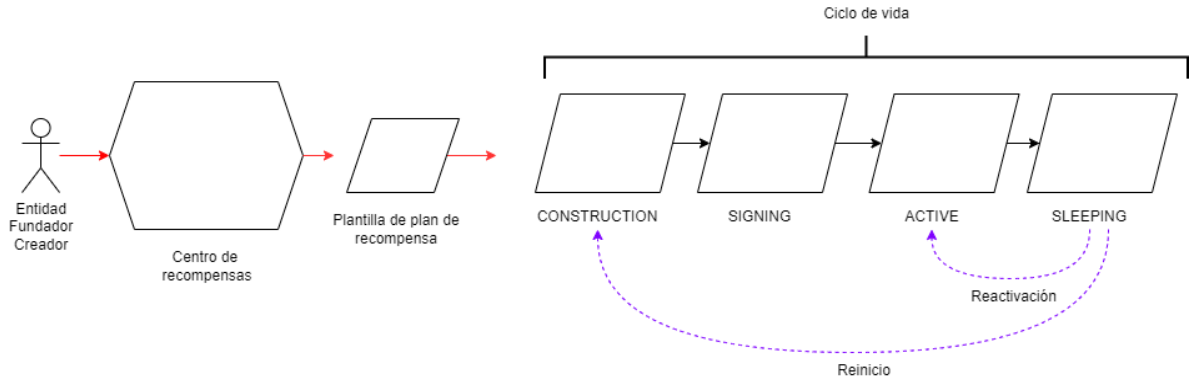

Figura 4: Ciclo de vida de un plan de recompensas.

Es importante aclarar que las reglas de recompensa podrían definirse de múltiples maneras, aunque por cuestiones de simplificación en este trabajo se considerará sólo una. Esta se basa en que cada cliente tiene asociado para cada plan, un número de puntos que va acumulando al realizar acciones recompensables. Entonces las reglas a considerar, relacionan una cantidad de puntos acumulados con una cantidad de divisa digital como recompensa. Cabe mencionar que son los notificadores los que comunican a la plataforma la cantidad de puntos que se le debe sumar a un cliente.

Adicionalmente, cualquier entidad miembro de un plan ya sea como fundador o notificador, puede abandonar el plan en cualquier momento. Esto será útil puesto que las entidades tendrán un límite de membresías a planes.

Funcionamiento conjunto del sistema

En la Figura 5 se unen todos los elementos de la plataforma, mostrándose un escenario con dos entidades que a través del centro de recompensas, han creado una serie de planes. La entidad A tiene un plan individual y la entidad B tiene un plan colaborativo con la entidad A. Estos a su vez reciben eventos de los notificadores para aplicar la lógica de las reglas de recompensa sobre los clientes asociados a cada plan.

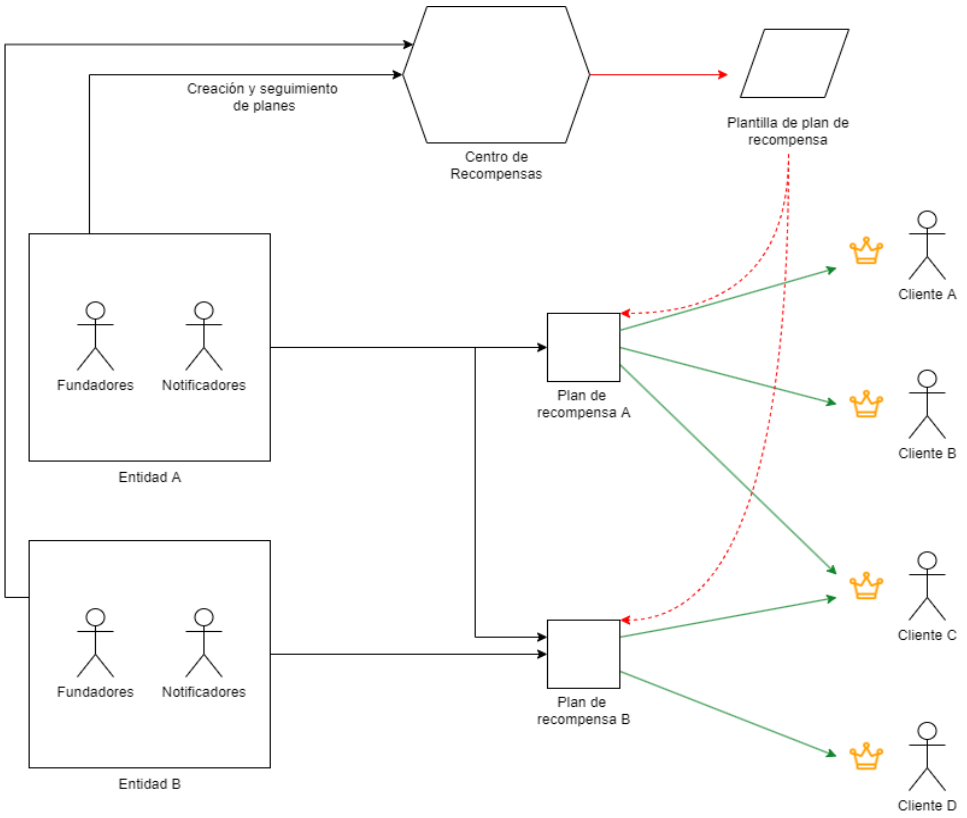

Figura 5: Esquema general del funcionamiento de la plataforma.

Un supuesto práctico podría ser que cada entidad esté asociada a un supermercado. Estos supermercados asignan una cuenta de la red blockchain a un trabajador, con fondos digitales suficientes para desempeñar la función de fundador del plan. Supongamos que durante la construcción se establecen las siguientes dos reglas.

- Por cada 10 puntos, se recompensa lo equivalente a 5€ en divisa digital.
- Por cada 100 puntos, se recompensa lo equivalente a 100€ en divisa digital.

Entonces, los supermercados dotarán a cada una de sus cajas con una cuenta asociada al plan como notificador. Además, cada supermercado seleccionará, por ejemplo, una serie de productos cuya venta rápida sea de interés, y notificará a los clientes que por cada unidad de estos productos, se obtendrá 1 punto. De esta manera, cuando los clientes pasen su compra por la caja, el notificador preguntará por el identificador del cliente en la plataforma para poder asignarle los puntos que le correspondan. Los clientes podrán realizar un seguimiento de los puntos que hayan acumulado a través del cliente web y recibirán las recompensas establecidas una vez lleguen a las metas de puntos.

Elección de red blockchain

Pese a existir otras redes más adecuadas en términos de rapidez y coste, para el proyecto que se propone, se ha elegido la red de Ethereum para el desarrollo de la plataforma. Esto se debe a una serie de motivos que se enumeran a continuación.

En primer lugar, esta red cuenta con una comunidad muy grande y activa, lo que facilita el acceso a recursos y herramientas de desarrollo, así como el acceso a la experiencia de otros desarrolladores. De hecho, es la red con mayor ecosistema de aplicaciones, pues es de las redes blockchain más antiguas y la primera en implementar los contratos inteligentes. De esta manera, la curva de aprendizaje resulta significativamente menor que la de sus competidores.

Por otro lado, Solidity, el lenguaje nativo para los contratos inteligentes de la red de Ethereum destaca por su sencillez, a comparación de otras redes como Solana o Cardano, y además se asemeja más a los lenguajes de programación tradicionales orientados a objetos, convirtiéndolo en una opción ideal para la iniciación en el mundo de los contratos inteligentes.

Por último, Solidity cuenta con un abanico de cadenas de bloques compatibles. Estas pueden ser lo que se conoce como cadenas de bloques de segunda capa, que funcionan de manera interconectada con la red principal de Ethereum adoptando sus cualidades y extendiendo su funcionalidad. O también se da el caso de redes totalmente independientes que implementan mecanismos de compatibilidad debido a la popularidad del lenguaje.

Framework y otras herramientas de desarrollo

La fundación Ethereum proporciona un entorno de desarrollo online llamado Remix en el que se puede iniciar la creación de contratos inteligentes e interactuar con ellos tras desplegarlos de manera muy fácil e intuitiva. Está orientado a ser una plataforma de inicio y educativa sobre los contratos inteligentes en Solidity. Con esta herramienta se realizará los primeros bocetos.

Una vez se tenga una idea más estructurada de los contratos, se podrá pasar a un entorno de trabajo más robusto y habilitado para la puesta en producción de contratos reales. Se trata del framework Hardhat, un módulo de Javascript que se construye encima de la librería EthersJS. Hardhat facilita el desarrollo, testing, debugging y despliegue de smart contracts en cualquier red ethereum compatible.

Hardhat incluye una serie de funcionalidades que consiguen abstraer gran parte de la lógica asociada a la blockchain, facilitando la familiarización del desarrollador con el entorno. Estas funcionalidades son por ejemplo, la

compilación de contratos, la administración de cuentas, la administración de redes o la gestión de los meta datos necesarios para desplegar y comunicarse con los contratos. Además provee de una máquina virtual local que simula la red de ethereum, en la que ejecutar cualquier secuencia de transacciones para analizar los resultados.

Por último, Hardhat cuenta con un amplio repertorio de paquetes de terceros que extienden sus funcionalidades. Por ejemplo, se utilizará hardhat-gas-reporter para analizar el costo de las transacciones, o hardhat-coverage para realizar el estudio de cubrimiento sobre los tests.

Diseño del cliente web

Requerimientos funcionales

La interfaz web tendrá que habilitar la interacción entre los distintos participantes del sistema y la red de bloques. Es decir, tendrá que permitir la interacción con el centro de recompensas y con cada plan de recompensa, ya sea para lanzar transacciones que alteren el estado de la blockchain o transacciones de lectura.

Tendrá que existir un elemento que permita interactuar con el centro de recompensas, de manera que cualquier usuario pueda darse de alta como entidad al crear un plan, convirtiéndose en su creador y fundador. Además, este mostrará la información asociada a la cuenta de la sesión actual, desde el punto de vista del centro de recompensas. Es decir, para las entidades el número de planes asociados, y para los clientes su identificador en la plataforma y el número de recompensas totales que haya recibido a lo largo de su participación en la plataforma.

Por otro lado, tendrá que existir un mecanismo para poder interactuar con todos los planes de recompensas asociados de manera individual. La información que se muestra y las funcionalidades disponibles, tendrán que depender del estado del plan además del rol que desempeñe la cuenta asociada a la sesión actual.

Por último, será importante disponer de un mecanismo resistente y amigable para manejar y mostrar al usuario cualquier error que pudiera surgir a raíz de la interacción con los contratos. En la Figura 5, se pueden ver todos los requisitos mencionados en conjunto.

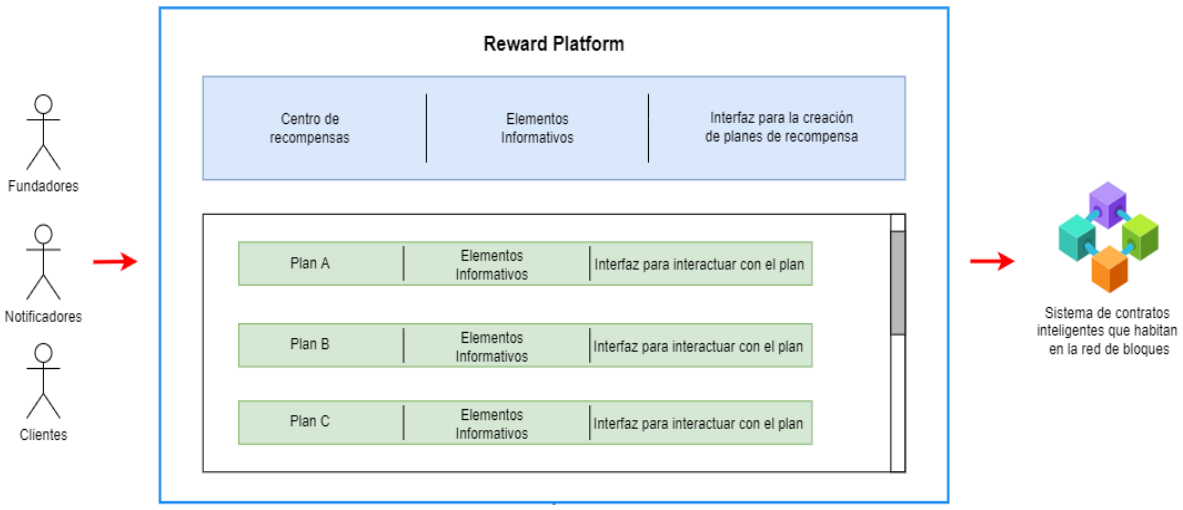

Figura 6: Boceto del cliente web, de acuerdo a los requisitos.

Elección del framework

En cuanto al framework principal para el desarrollo de la aplicación web, se ha decidido utilizar NextJS, un entorno de desarrollo construido encima de React que se autodenomina una mejor solución que su ascendencia para la puesta en producción de aplicaciones rápidas y escalables. Se utilizan archivos JSX que pueden contener toda la lógica de la aplicación, incluyendo la renderización de HTML y el manejo de CSS.

La principal ventaja de esta elección es que se trata de un entorno con una gran cantidad de funcionalidades y herramientas de desarrollo, que permiten minimizar el código, maximizando la calidad del resultado.

Para la implementación de elementos visuales con un estilo predefinido se utilizará el paquete react-bootstrap que es compatible con el framework elegido y habilita el uso de Bootstrap 5 en el proyecto.

Además, se utiliza EthersJS para la creación de transacciones que serán enviadas a la cadena de bloques y también la extensión de navegador Metamask para la autenticación de los usuarios a través de sus cuentas.

Capítulo 3 - Desarrollo

Primeros pasos

El primer paso dado para el desarrollo ha sido el seguimiento de dos vídeos tutoriales que en conjunto suman 48 horas de contenido. Los ofrece freeCodeCamp.org de manera gratuita a través de la plataforma Youtube, y concretamente los imparte Patrick Collins. Ambos están focalizados en Blockchain, Solidity y desarrollo de contratos inteligentes.

Se diferencian principalmente en que el primero utiliza el lenguaje Python así como herramientas como Truffle para el desarrollo, mientras que el segundo utiliza Javascript y añade una sección en la que se codifica una aplicación web que interactúa con los contratos desarrollados en las secciones anteriores. Las Figuras 7 y 8 muestran la portada de ambos tutoriales.

Figura 7: Portada del video tutorial de desarrollo blockchain con Python.

Figura 8: Portada del video tutorial de desarrollo blockchain con Javascript.

Como ya se comentó anteriormente, los contratos han sido codificados inicialmente en Remix IDE. En primer lugar, se codificaron una serie de contratos simples para la familiarización con el lenguaje. Por ejemplo “Simple Storage” (Figura 9), que muestra un ejemplo muy sencillo de cómo se ve un contrato con la capacidad de almacenar información que reciba a través de transacciones.

SimpleStorage.sol

```
pragma solidity ^0.8.0;
/**
 * @title SimpleStorage
 * @dev Store & retrieve value in a variable
 */
contract Storage {
    struct People {
        uint256 dni;
        string name;
    }

    People[] public people;
    mapping(string => uint256) public nameToDni;

    function addPerson(string memory _name, uint256 _dni) public {
        people.push(People(_dni, _name));
        nameToDni[_name] = _dni;
    }
}
```

Figura 9: Código de contrato inteligente - SimpleStorage.sol.

Se puede ver que los contratos se crean de manera similar a como se crea una clase en un lenguaje tradicional orientado a objetos. Se define una estructura de datos para representar a una persona, formada por un número entero sin signo que ocupará 256 bits de memoria y una cadena de caracteres que representarán respectivamente el DNI y el nombre del individuo.

A continuación, se definen dos atributos asociados al contrato. Un array de elementos que siguen la estructura definida para People, y un mapeo que permitirá vincular nombres (texto) a sus DNIs (entero). Para último, se define la función addPerson, que permite añadir elementos al array people y al mapeo nameToDni.

Es importante destacar que se especifica un tipo de visibilidad a la hora de declarar una variable o una función. Cuando es pública, el acceso se permite sin restricciones. Si se tratase de una visibilidad interna, solo los componentes del contrato y contratos que hereden de él tendrán acceso. En el tercer caso, con una una visibilidad privada solo los componentes del contrato pueden acceder a los datos. Esto puede resultar confuso porque pese a existir estos tipos de visibilidad, sólo rigen cómo interactúan los contratos entre sí de manera directa, pues la capa donde se almacenan los datos de la cadena de bloques es pública de cara al exterior de la red.

Adicionalmente, se puede especificar el tipo de memoria donde se quiere almacenar las variables, que puede ser Storage, Memory o Calldata. El primer tipo es donde se almacenan los datos que han de persistir a lo largo del tiempo, por ejemplo los atributos de un contrato. El segundo se utiliza para almacenar cualquier tipo de información de manera temporal, durante la ejecución de alguna función. Y el último funciona como una pila de datos, donde se almacenan relativamente pocos datos de manera temporal. De hecho es utilizada por la propia Ethereum Virtual Machine para la ejecución de llamadas. Esta elección afecta principalmente al coste transaccional, siendo Storage el almacenamiento más costoso y Calldata el más económico.

Se ha conseguido un contrato capaz de almacenar datos de manera estructurada, que además se pueden leer de sus atributos. Para comprender además cómo es que un contrato puede crear otros contratos y comunicarse con ellos, se codificó un segundo contrato que siguiera el patrón conocido como Contract Factory, cuyo código se puede ver en la Figura 10.

SimpleStorage.sol

```
pragma solidity ^0.8.0;

import "./SimpleStorage.sol";

contract StorageFactory {
    Storage[] public storageArray;

    function createStorageContract() public {
        Storage newStorage = new Storage();
        storageArray.push(newStorage);
    }

    function sfAddPerson(
        uint256 index, uint256 dni, string memory name
    ) public {
        Storage current = Storage(address(storageArray[index]));
        current.addPerson(dni, name);
    }

    function sfNameToDni(uint256 index, string memory name) public {
        Storage current = Storage(address(storageArray[index]));
    }
}
```

Figura 10: Código de contrato inteligente - StorageFactory.sol.

Este contrato es capaz de desplegar instancias del contrato Storage anteriormente descrito, guardandolas en un array y posibilitando la interacción con ellos mediante los métodos sfAddPerson y sfNameToDni.

Desarrollo de la plataforma

Configuración del entorno de trabajo

En primer lugar, mencionar que se ha elegido Visual Studio Code como IDE. Utilizando Yarn como gestor de paquetes, se ha instalado Node Version Manager 0.38.0 mediante el cual a su vez se ha instalado Node v16.16.0. A continuación, se ha inicializado un "package.json" y añadido el paquete de Hardhat. Adicionalmente, se utiliza también el paquete Prettier Plugin Solidity que se encarga de formatear el código de manera que siga la guía de estilo descrita en su repositorio de Github. Ver la entrada [32] en la bibliografía.

Al iniciar Hardhat, aparece una interfaz en la consola a través de la cual se guía la inicialización del proyecto, creando la estructura de archivos adecuada e instalando las dependencias necesarias. Véase en la Figura 11.

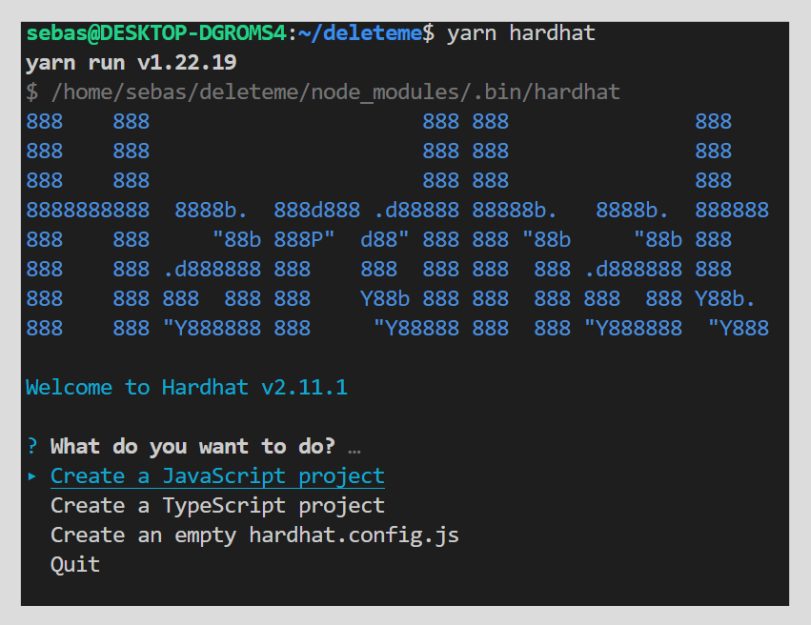

Figura 11: Interfaz de consola de Hardhat.

La configuración más específica del proyecto se especifica en el archivo hardhat.config.js, donde se indican una serie de parámetros como por ejemplo la versión de Solidity que se pretende utilizar.

También se especifica la configuración del optimizador, un elemento muy importante que se encarga de minimizar el tamaño de los contratos antes de su compilación, pues estos tienen un límite de tamaño de 24577 bytes para el almacenamiento de su parte lógica. De hecho sin el uso de esta herramienta, los contratos confeccionados para este trabajo exceden el límite mencionado.

Por último, existe una lista de especificaciones para redes a las que se pretenda conectar e interactuar, en mi caso solo fue necesario añadir la especificación para el nodo local que ofrece Hardhat.

Adicionalmente, se han aprovechado dos extensiones de Visual Studio Code: Solidity de Nomic Foundation, que está enfocada en dotar al IDE de funcionalidades que faciliten el desarrollo de proyectos de Hardhat, y Github Copilot, una IA que genera sugerencias de auto compleción de código.

Estructuras de datos definidas

El archivo "DataStructures.sol" contiene una serie de estructuras de datos definidas para poder representar los diversos elementos y participantes del sistema. Estas se pueden separar en dos grupos dependiendo de si son relevantes para el centro de recompensas o para si lo son para los planes.

DataStructures.sol

```
struct EntityProfile {
  bool active;
  address addr;
  uint8 runningPlans; // 256 max
}

struct ClientProfile {
  bool active;
  address addr;
  uint256 rewards;
}

struct PlanProfile {
  bool active;
  address creatorAddr;
  uint256 totalRewarded;
  string name;
}
```

Figura 12: Estructuras de datos I.

DataStructures.sol

```
struct Founder {
  address addr;
  uint256 collaborationAmount;
  bool signed;
}

struct Notifier {
  bool active;
  address addedBy;
}

struct ClientPoints {
  bool active;
  uint256 points;
}

struct RewardPointRule {
  uint256 points;
  uint256 reward;
  address creator;
}
```

Figura 13: Estructuras de datos II.

La Figura 12 corresponde al primer grupo, que permiten al centro de recompensas llevar un registro de los participantes y de los planes que se hayan creado. Mientras que el segundo grupo serán los structs de la Figura 13 y serán utilizados dentro de cada plan.

Por otro lado, se han definido dos contratos: Reward Center y Reward Plan. Ambos están compuestos por una serie de atributos, eventos, funciones, y modificadores. Los atributos son todas las variables asociadas de manera persistente al contrato, los eventos podrán ser emitidos en el código para generar un registro en el recibo de la transacción, las funciones permiten interactuar con el contrato y los modificadores permiten restringir el acceso de las funciones.

Reward Center

- Idea general

  Se pretende desplegar una única instancia de este contrato que será el pilar central del sistema. Su cometido principal será la creación de planes de recompensa y el seguimiento de estos últimos junto al de los participantes de la plataforma. Por ello, contendrá una serie de registros globales respecto a los planes, participantes y las relaciones que existan entre ellos.

- Atributos

  Consisten en seis variables públicas de tipo mapping. "ClientAddressToId" enlaza direcciones de clientes a sus identificadores numéricos. "ClientRegistry" asocia identificadores a perfiles de clientes. "EntityRegistry" vincula direcciones de entidades a sus perfiles. "PlanRegistry" relaciona direcciones de planes de recompensa a sus perfiles. "EntityRelatedPlans" conecta direcciones de entidades a un array de las direcciones de sus planes asociados. Por último, "ClientRelatedPlans" enlaza direcciones de clientes a un array de las direcciones de sus planes asociados. El código resulta bastante auto explicativo, como se puede ver en la Figura 14.

RewardCenter.sol

```
mapping(address => uint256) public clientAddressToId;
mapping(uint256 => ClientProfile) public clientRegistry;
mapping(address => EntityProfile) public entityRegistry;
mapping(address => PlanProfile) public planRegistry;
mapping(address => address[]) public entityRelatedPlans;
mapping(address => address[]) public clientRelatedPlans;
```

Figura 14: Atributos de RewardCenter.sol.

- Eventos

  Solo hay un evento en Reward Center y se emitirá cada vez que un plan sea creado. Véase la Figura 15.

RewardCenter.sol

```
event RewardPlanCreated(address planAddress, address creator);
```

Figura 15: Eventos de RewardCenter.sol.

- Modificadores

  Existe un único modificador, "onlyPlans", que se encarga de comprobar que la dirección del remitente coincide con la de algún plan de recompensa registrado en Reward Center. Véase la Figura 16.

RewardCenter.sol

```
modifier onlyPlans() {
    require(
        planRegistry[msg.sender].active,
        "Only active plans authorized"
    );
    _;
}
```

Figura 16: Modificadores de RewardCenter.sol.

- Capa de servicios

  Las funciones del contrato pueden agruparse según su visibilidad y tipo. De esta manera, encontramos funciones privadas, externas y externas de solo lectura. A su vez, las externas pueden ser de libre acceso o estar destinadas exclusivamente a los contratos asociados a los planes de recompensa mediante el modificador previamente descrito.

  La función principal es "createRewardPlan" (Figura 17), que junto a las funciones externas de lectura (Figura 18), pueden ser invocadas por cualquier persona o contrato. Estas conforman la capa de servicios del contrato y serán utilizadas en la interfaz web.

RewardCenter.sol

```
function createRewardPlan (
  uint256 refundNotAllowedDuration,
  string calldata name,
  uint256 collaborationAmount
) external
{...}
```

Figura 17: Función principal de RewardCenter.sol.

Esta función se encarga de crear y desplegar una instancia del contrato Reward Plan. Para ello primero registra al remitente como entidad si aún no figura como tal y en caso contrario se comprueba que no haya llegado al límite de 256 planes asociables. Tras esto, se crea una instancia del contrato Reward Plan, cuyo constructor recibe la dirección del creador, el tiempo en segundos que han de pasar para que el contrato pueda revertirse y la cantidad que aportará el creador al fondo del plan para firmar el plan. Por último, almacena su información en el registro de planes y añade la dirección al array de planes asociados a su creador. Una vez esto se ejecuta satisfactoriamente, se emite el evento de creación de plan.

Las funciones externas de solo lectura no contienen una lógica muy compleja, sino que facilitan el acceso a sus atributos o a los de sus planes.

RewardCenter.sol

```
function checkRolesInPlan (address planAddress)
  external view returns (
    bool isClient, bool isFounder, bool isNotifier
  )
{...}

function getSelfRelatedPlans ()
  external view returns (address[] memory)
{...}

function getClientAddress (uint256 clientId)
  external view returns (address)
{...}

function getPlanStage (address planAddress)
  external view returns (Stages)
{...}
```

Figura 18: Funciones externas de solo lectura de RewardCenter.sol.

Por último, las funciones externas que serán llamadas desde los planes de recompensa (Figura 19) tienen como objetivo complementar la lógica de funciones de Reward Plan que si serán iniciadas por los fundadores o notificadores. Por ello, indirectamente forman parte de la capa de servicios del segundo contrato.

RewardCenter.sol

```
function notifyMemberAddedToPlan (address memberAddress)
  external onlyPlans
{...}

function unlinkPlanToMember (address member)
  external onlyPlans
{...}

function signUpClient (uint256 clientId, address clientAddress)
  external onlyPlans
{...}

function notifyRewardGranted (uint256 clientId, uint256 amount)
  external onlyPlans
{...}
```

Figura 19: Funciones externas de RewardCenter.sol.

Cabe mencionar que las funciones privadas forman parte de las ya mencionadas, aunque se han separado por claridad y modularización.

Reward Plan

- Idea general

  Como se ha comentado previamente, se desplegarán varias instancias de este contrato a través del centro de recompensas. Cada una de ellas representará un plan de recompensa en la plataforma y tendrá como responsabilidad principal gestionar su propia construcción, financiación, y el reparto de las recompensas.

  Para la correcta gestión del ciclo de vida de cada uno de los planes, se sigue el patrón de diseño llamado “State Machine”. Ver entrada [23] en la bibliografía. Consiste en definir un estado en el contrato y un mecanismo para su transición. Cada estado representará una etapa de su ciclo de vida y determinará las funciones accesibles en cada etapa. Se utiliza una enumeración para listar los estados, como se ve en la Figura 20.

```
enum Stages { CONSTRUCTION, SIGNING, ACTIVE, SLEEPING }
```

Figura 20: Enumeración de las etapas del ciclo de vida de un plan de recompensa.

- Atributos

  Como puede verse en la Figura 21, cada plan tendrá una referencia al centro de recompensas, un elemento de la enumeración mostrada que representará el estado del plan, la dirección del creador, el tiempo en segundos que ha de pasar para poder revertir el contrato y un registro para fundadores, notificadores, reglas de recompensa y la puntuación de los clientes. Cabe mencionar que el array de reglas se mantendrá ordenado de menor a mayor puntos a marcar para reducir el costo computacional de la comprobación de reglas.

RewardPlan.sol

```
RewardCenter public rewardCenter;
Stages public stage;
address public creator;
uint256 public nonRefundableDuration;
uint256 public allowRefundTimestamp;
Founder[] public founders;
address[] public notifierAddresses;
mapping(address => Notifier) public notifiers;
RewardPointRule[] public rewardPointsRules;
mapping(uint256 => ClientPoints) public rewardPointsRegistry;
```

Figura 21: Atributos de RewardPlan.sol.

- Eventos

  Se define un evento para cada uno de los sucesos que puedan desencadenar los participantes, concretamente los fundadores y notificadores. Véase en la Figura 22.

RewardPlan.sol

```
event FounderAdded(address founderAddress, uint256 collaborationAmount);
event RewardRuleAdded(address founderAddress, uint256 points, uint256 reward);
event RewardRuleRemoved(address founderAddress, uint256 index);
event NotifierAdded(address notifierAddress, address addedBy);
event SigningStageBegun(address caller);
event FounderSigned(address signer, bool allSigned);
event ClientSignedUp(uint256 clientId, address clientAddress);
event PointsScored(uint256 clientId, uint256 amount, uint256 grantedReward);
event PlanAwaken(address caller, uint256 collaboration);
```

Figura 22: Eventos de RewardPlan.sol.

- Modificadores

  Primero será necesario un modificador que permita o deniegue el acceso a las funciones en función del estado del plan. A continuación, dos modificadores para gestionar el acceso en función del rol que desempeñe el remitente de la transacción. Por último, otro que permita el acceso si el tiempo establecido durante el cual el contrato no puede revertirse, ha concluido. Véase la Figura 23.

RewardPlan.sol

```
modifier atStage(Stages allowedStage) {...}
modifier onlyFounders() {...}
modifier onlyNotifiers() {...}
modifier refundIsAllowed() {...}
```

Figura 23: Modificadores de RewardPlan.sol.

- Capa de servicios

  Las funciones más importantes de la capa de servicio de los planes pueden agruparse según cual sea la etapa en la que pueden ser invocadas. De esta manera, en la fase de construcción solo se permite el uso de aquellas funciones que se encarguen de editar los atributos que definirán el comportamiento del plan (Figura 24). Estas sólo estarán disponibles para los fundadores.

RewardPlan.sol

```
function addFounder(address founderAddress, uint256 collaborationAmount)
  external atStage(Stages.CONSTRUCTION) onlyFounders
{...}

function addNotifier(address notifierAddress)
  external atStage(Stages.CONSTRUCTION) onlyFounders
{...}

function addRewardRule(uint256 scoredPoints, uint256 rewardAmount)
  external atStage(Stages.CONSTRUCTION) onlyFounders
{...}

function removeRewardRule(uint256 ruleIndex)
  external atStage(Stages.CONSTRUCTION) onlyFounders
{...}

function beginSigningStage()
  external atStage(Stages.CONSTRUCTION) onlyFounders
{...}
```

Figura 24: Funciones de la etapa de construcción en RewardPlan.sol.

En la etapa de firma serán necesarias dos funciones. Como se ve en la Figura 25, la primera será a través de la cual un fundador puede firmar el plan aportando la cantidad que le haya sido asignada para colaborar con su financiación. La segunda sólo será necesaria en caso de disconformidad o desacuerdo entre los fundadores, y una vez haya pasado el tiempo preliminar de construcción, les permite reiniciar el plan por completo devolviendo el dinero que pueda haber aportado cada fundador. Ambas solo podrán ser invocadas por los fundadores del plan.

RewardPlan.sol

```
function sign()
  external payable atStage(Stages.SIGNING) onlyFounders
{...}

function refundAndReset()
  external atStage(Stages.SIGNING) onlyFounders refundIsAllowed
{...}
```

Figura 25: Funciones de la etapa de firma en RewardPlan.sol.

Durante la fase de actividad del plan, serán los notificadores quienes puedan invocar sus funciones. La primera será para asociar un cliente al plan, dándole de alta en la plataforma si fuera necesario. La segunda será a través de la cual se notifiquen los puntos que generen los clientes como consecuencia de realizar actividades recompensables. Cada vez que se notifique una cantidad de puntos marcados por un cliente, se realizará la comprobación de las reglas del plan, de manera que se dispensen las recompensas correspondientes. Véase la Figura 26.

RewardPlan.sol

```
function signUpClient(uint256 clientId, address addr)
  external atStage(Stages.ACTIVE) onlyNotifiers
{...}

function notifyPointsScored(uint256 clientId, uint256 amount)
  external atStage(Stages.ACTIVE) onlyNotifiers
{...}
```

Figura 26: Funciones de la etapa de actividad en RewardPlan.sol.

Una vez el contrato agote sus fondos entrará en el estado "Sleeping". En este punto, sus fundadores podrán reactivar el plan aportando nuevos fondos con los que realizar las recompensas (Figura 27). Además, el creador tiene la opción de reiniciar por completo el plan, con lo que volvería a ser igual que inmediatamente después de ser desplegado.

RewardPlan.sol

```
function awakePlan(bool reset)
  external payable atStage(Stages.SLEEPING) onlyFounders
{...}
```

Figura 27: Funciones de la etapa de inactividad en RewardPlan.sol.

Adicionalmente, se incluye la función "leavePlan" que puede invocarse en cualquier fase y permitirá abandonar el plan a cualquier miembro siempre y cuando no se trate del creador.

Por otro lado, al igual que en el contrato anterior las funciones externas de solo lectura no contienen ninguna lógica especialmente compleja. Su objetivo es facilitar el acceso y análisis del valor de sus atributos. Se pueden ver sus cabeceras en la Figura 28.

RewardPlan.sol

```
function getFounderAddresses()
  external view returns (address[] memory)
{...}

function getNotifierAddresses()
  external view returns (address[] memory)
{...}

function isClient(uint256 targetId)
  external view returns (bool)
{...}

function isNotifier(address targetAddr)
  external view returns (bool)
{...}

function isFounder(address targetAddr)
  external view returns (bool)
{...}

function getRewardRules()
  external view returns (RewardPointRule[] memory)
{...}

function isRefundable()
  external view returns (bool)
{...}
```

Figura 28: Funciones de solo lectura de RewardPlan.sol.

Testing con Mocha y Chai en Hardhat

Se ha codificado una suite de tests con los paquetes Mocha y Chai. Ambos vienen incluidos en la instalación de Hardhat y el entorno está preconfigurado para facilitar el desarrollo de los tests. Para la comunicación con los contratos se utiliza el módulo EthersJS, incluido también con la instalación de Hardhat.

Se han construido de manera que abarquen todo el proceso de uso del sistema de principio a fin. Este proceso se inicia con el despliegue de Reward Center. A continuación, se da de alta a una entidad al crear un plan de recompensas. Esta añade un fundador dándole de alta como entidad y construyen el plan añadiendo una serie de reglas y notificadores. A continuación ambos firman el plan aportando los fondos correspondientes y el contrato pasa a estado activo. Se reparten una serie de recompensas a dos clientes tras realizar varias acciones recompensables. Una vez se agotan los fondos del contrato, entra en estado de sueño y una de las entidades decide despertarlo aportando fondos extra.

La Figura 29 y 30 muestran el resultado de su ejecución y las descripciones de cada comprobación realizada.

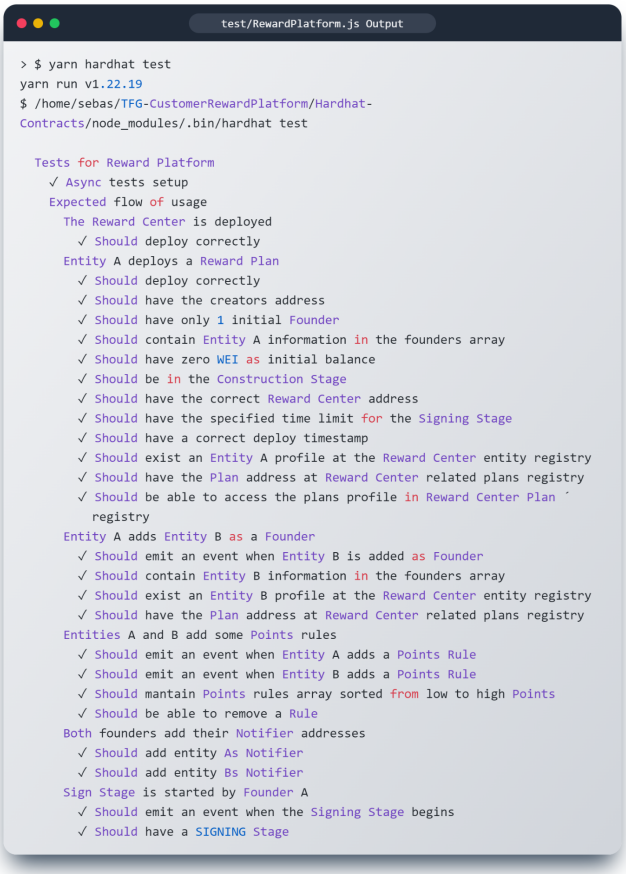

Figura 29: Tests de la plataforma I.

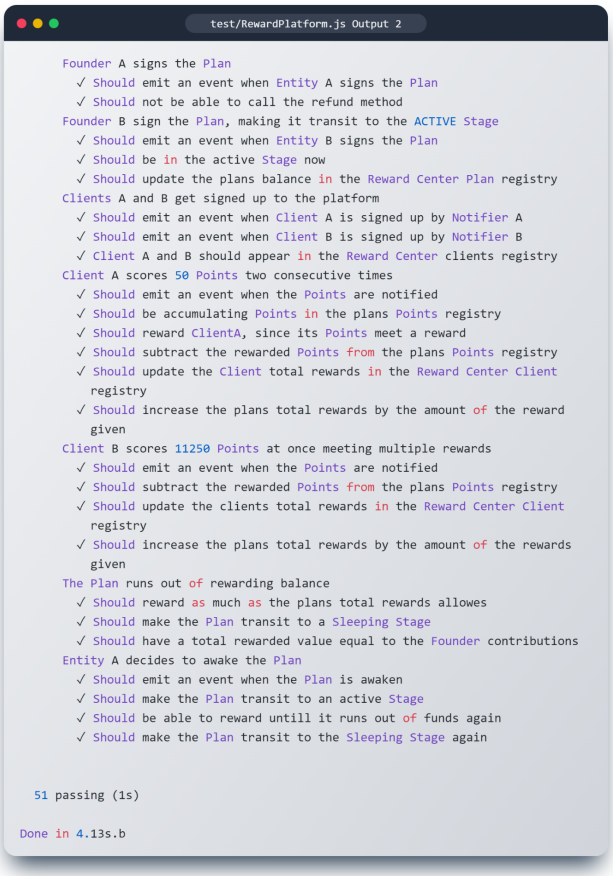

Figura 30: Tests de la plataforma II.

Cliente Web

Entorno y configuración inicial

El paso previo al desarrollo del cliente web ha sido codificar dos módulos de JavaScript, “contract-execution” y “contract-view”. Ambos proporcionan una serie de funciones que contienen la lógica de interacción específicamente para la capa de servicios de los contratos de la plataforma.

A continuación, se ha añadido el paquete NextJS y se ha ejecutado el comando *yarn create next-app* que genera una estructura de archivos plantilla para un proyecto e instala las dependencias necesarias. Este framework está construido sobre React y por tanto hereda en parte su planteamiento y características. Adicionalmente, se añade el módulo react-bootstrap que integra Bootstrap 5.0 en el proyecto, para tener acceso a plantillas de elementos como listas o formularios con un estilo predefinido.

Este framework provee de una carpeta pages en la que cada archivo JSX representa una página asociada a la ruta con mismo nombre. Por tanto, ya que el cliente que se ha desarrollado es una Single Page Application (SPA), el archivo principal será index.js, que asigna automáticamente a la ruta raíz.

Adicionalmente, existe el archivo _app.js donde se encapsula la aplicación entera en un solo componente y se pueden realizar cambios a nivel global. Como se ve en la Figura 31, en este caso se añade uno de las opciones de estilo CSS que proporciona Bootstrap, además de añadirse la componente Web3 React Provider que facilitará la comunicación con el proveedor del servicio MetaMask.

_app.js

```
import '../styles/globals.css';
import { Web3ReactProvider } from '@web3-react/core';
import { Web3Provider } from '@ethersproject/providers';

const getLibrary = (provider) => {
  return new Web3Provider(provider);
}

function MyApp({ Component, pageProps }) {
  return (
    <Web3ReactProvider getLibrary={getLibrary}>
      <link
        rel="stylesheet"
        type="text/css"
        href="
            https://cdn.jsdelivr.net/npm/bootstrap@5.2.0beta1/
            dist/css/bootstrap.min.css
        "
        integrity="
          sha3840evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjf
          HxaWutUpFmBp4vmVor
        "
        crossOrigin="anonymous"
      />
      <Component {...pageProps} />
    </Web3ReactProvider>
  );
}

export default MyApp
```

Figura 31: Código del cliente - _app.js.

Por último, es importante mencionar que para el desarrollo del cliente se ha establecido una conexión con el nodo local que proporciona Hardhat, donde se ha desplegado el centro de recompensas. Este actúa como simulador de la red de Ethereum que mientras esté activo muestra un histórico de las transacciones realizadas. Además, provee de 20 cuentas distintas con 10000 ETH para realizar las pruebas pertinentes. Para utilizarlas, es necesario tener una cartera donde se almacenen las credenciales de las cuentas que se vayan a utilizar. MetaMask desempeña esta tarea y es de las opciones más utilizadas en cuanto se refiere a la red de ethereum.

Reward Platform SDK

Los módulos contract-view y contract-execution mencionados anteriormente, pueden verse como el Software Development Kit de la plataforma. Cualquier desarrollador puede utilizarlo a la hora de implementar una interfaz funcional de manera independiente a la plataforma. Ambos archivos utilizan la librería Ether.JS para crear las transacciones que serán enviadas a los contratos de la plataforma. El de ejecución contiene aquellas funciones que suponen una transacción que cambia el estado de la red mediante operaciones de escritura y el archivo de vista, solo funcionalidades que no alteren el estado de la red. Las Figuras 32 y 33 muestran las cabeceras de sus funciones.

contract-execute.js

```
// Reward Center.
async function createRewardPlan(provider) {...}

// Reward Plan - Notifier.
async function signUpClient(provider, target, contractIndex) {...}
async function notifyPointsScored(provider, target, contractIndex) {...}

// Reward Plan - Founder.
async function leavePlan(provider, target, contractIndex) {...}
async function addFounder(provider, target, contractIndex) {...}
async function addNotifier(provider, target, contractIndex) {...}
async function addRewardRule(provider, target, contractIndex) {...}
async function removeRewardRule(provider, target, contractIndex) {...}
async function beginSigningStage(provider, target, contractIndex) {...}
async function sign(provider, target, contractIndex) {...}
async function refundAndReset(provider, target, contractIndex) {...}
async function awakePlan(provider, target, contractIndex) {...}
```

Figura 32: SDK contract-execute.js.

contract-view.js

```
async function getRewardCenterData(provider) {...}
async function getRelatedPlansBasics(provider) {...}
async function getPlanHeaderData(provider, target) {...}
async function getRolesInPlan(provider, target) {...}
async function getClientScoredPoints(provider, target) {...}
async function getContractRules(provider, target) {...}
async function getFounderRelatedData(provider, target) {...}
async function getPlanStage(provider, target) {...}
```

Figura 33: SDK contract-view.js.

Desarrollo de la interfaz

Los elementos visuales de la página se llaman componentes y cada uno contiene un estilo y una lógica de interacción asignada. Los componentes pueden contenerse entre sí para formar otros de mayor nivel de abstracción. Cada componente se define en una función que NextJS se encarga de transformar a una marca válida de HTML, de manera que puede añadirse a la jerarquía de elementos utilizando <*MiComponente*>.

Las componentes que se han utilizado se enumeran en la Figura 34 y están ordenadas de menor a mayor nivel de abstracción, siendo la última la correspondiente a la página completa.

index.js

```
import Router from "next/router";
import { useWeb3React } from '@web3-react/core';
import { useState, useEffect } from 'react';
import { InjectedConnector } from '@web3-react/injected-connector';
const injector = new InjectedConnector();

import Button from 'react-bootstrap/Button';
import Card from 'react-bootstrap/Card';
import Accordion from 'react-bootstrap/Accordion';
import Tab from 'react-bootstrap/Tab';
import Tabs from 'react-bootstrap/Tabs';
import ListGroup from 'react-bootstrap/ListGroup';
import Form from 'react-bootstrap/Form';
import Nav from 'react-bootstrap/Nav';
import Alert from 'react-bootstrap/Alert';
import Badge from 'react-bootstrap/Badge';

// Components.
export function ExecutionInterface({ target, elements, contractIndex }) {...}
export function ClientInterface({ target }) {...}
export function FounderInterface({ target, contractIndex }) {...}
export function NotifierInterface({ target, contractIndex }) {...}
export function RewardCenterInterface() {...}
export function ContractHeader({ target }) {...}
export function ContractCard({ contract, contractIndex }) {...}
export function ContractAccordion() {...}
export function RewardCenterCard() {...}

// Main component.
export default function Home() {...}
```

Figura 34: Componentes definidas en index.js.

NextJS también dispone de un comando *next dev* mediante el cual se inicia un servidor local donde se sirven las páginas del proyecto. La Figura 35 muestra el resultado visual de la página principal, adoptando el estilo de los componentes de bootstrap.

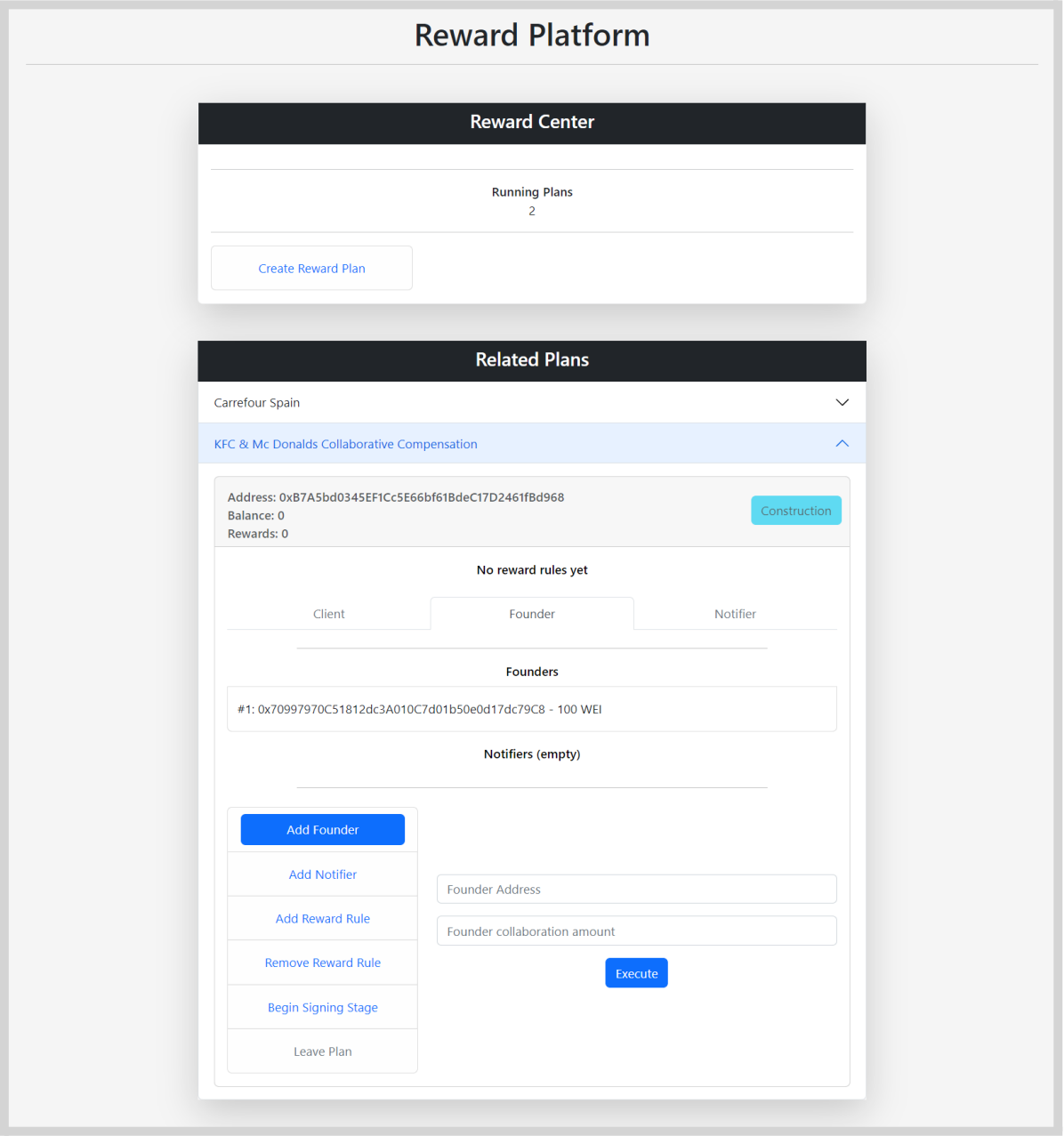

Figura 35: Resultado visual de index.js.

Como venía en la especificación hecha en la etapa de diseño, hay dos zonas diferenciadas, una para el centro de recompensas y la otra para todos los planes asociados a la cuenta actual. La Figura 35 muestra la vista de una entidad que desempeña el rol de fundador en el plan "*KFC & Mc Donalds Collaborative Compensation*", además de estar vinculado al plan "*Carrefour Spain*".

Capítulo 4 - Resultado

Plataforma e interfaz en funcionamiento

Una vez desplegado el centro de recompensas, la interfaz web está lista para ser utilizada. Cualquier participante puede desplegar un plan de recompensa e iniciar todo el proceso asociado a la dispensa de recompensas para su plan concreto. Como ejemplo representado en la Figura 36, se puede tomar una iniciativa a nivel nacional en la que se añade un miembro representante de cada universidad pública como fundadores del plan *Education Incentive Rewards*.

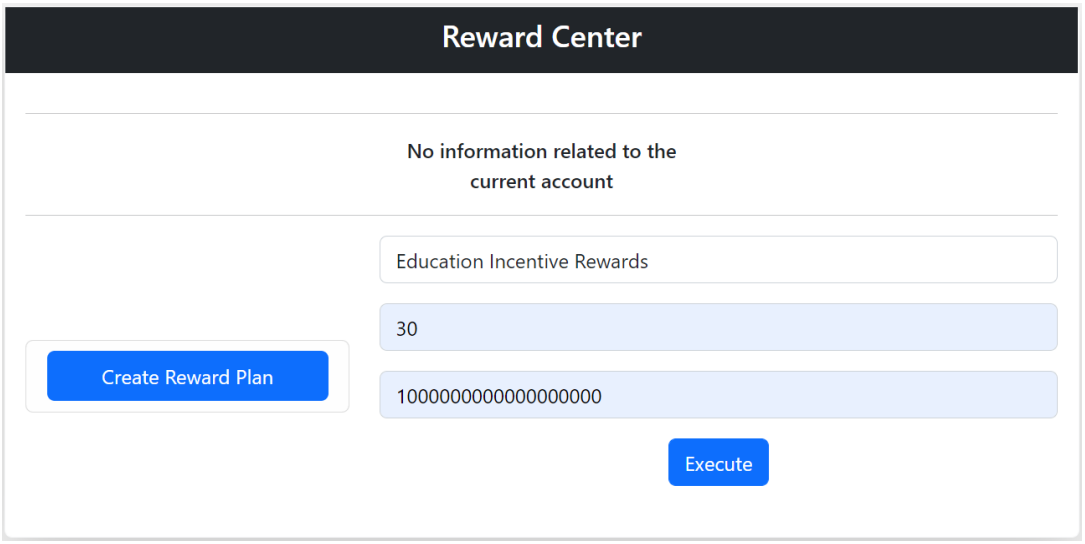

Figura 36: Creación de un plan a través del cliente web.

Las acciones recompensables de los estudiantes será la realización de pruebas calificables, y las notas (0 - 10) actuarán como puntos de recompensa. De esta manera se pueden establecer una serie de reglas cuya recompensa crezca exponencialmente según las notas se acercan al máximo. Además, cada universidad podrá añadir a un notificador (idealmente automatizado) que se encargue de cargar las notas en las plataformas.

Cuando el plan entre en la fase de firma, los fundadores valorarán la estructura del plan, para firmar y darle comienzo a la iniciativa, viendo algo similar a lo que se muestra en la Figura 37.

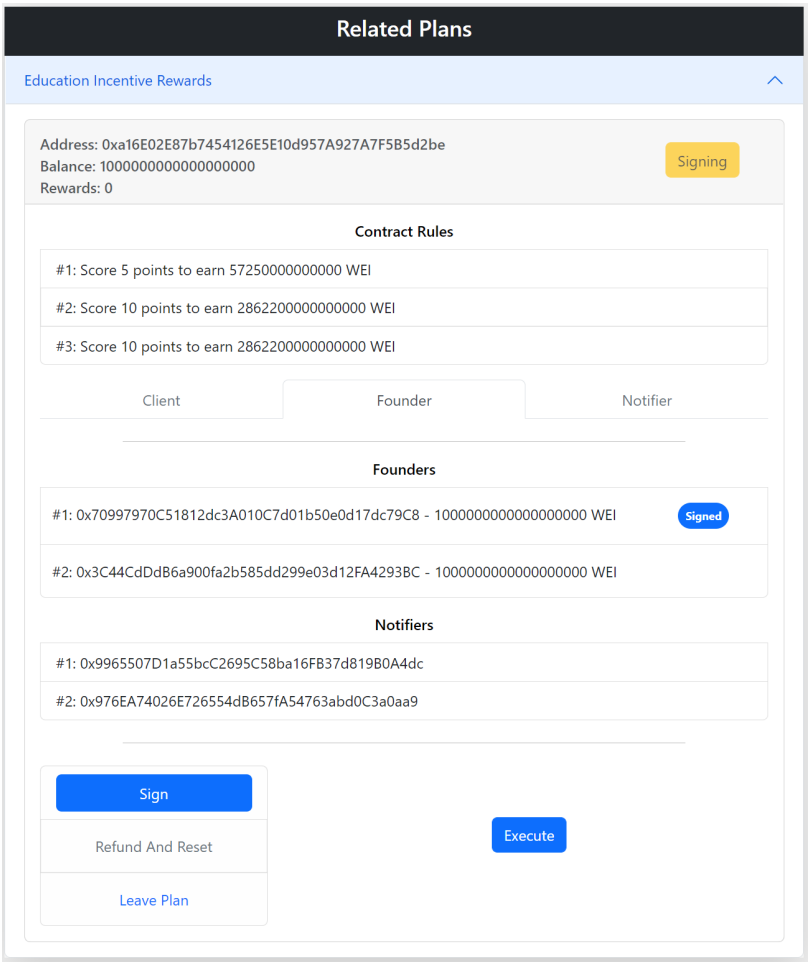

Figura 37: Firma de un plan a través del cliente web.

Al firmar todos los fundadores del plan, este pasa a estado activo y los notificadores pueden empezar a dar de alta a los alumnos, por ejemplo asignando la parte numérica del correo institucional como identificador único. Al finalizar una etapa de exámenes, todas las notas serán notificadas a la plataforma y las recompensas serán repartidas entre los alumnos.

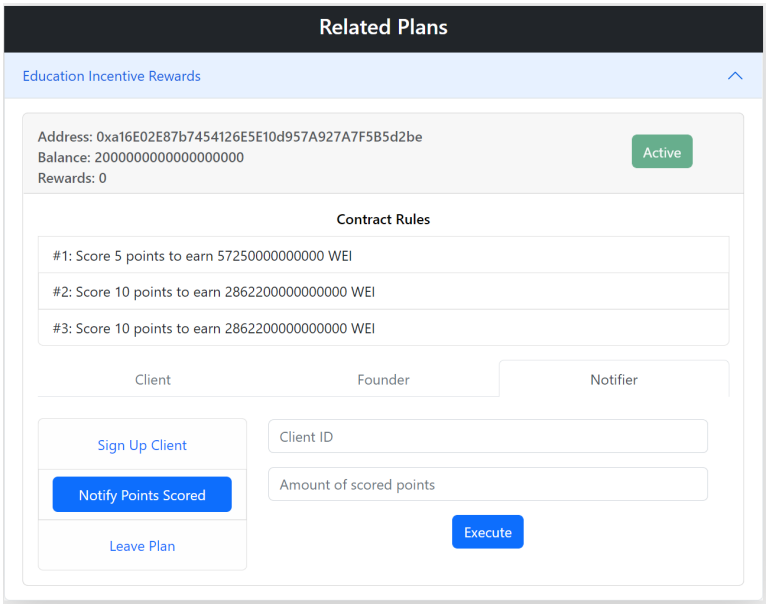

Figura 38: Fase activa de un plan a través del cliente web.

Costos transaccionales

Se ha utilizado la herramienta Hardhat Gas Reporter que extiende la funcionalidad de Hardhat de manera que se pueda generar una tabla con información relativa al coste transaccional.

Gas Reporter se encarga de analizar el costo de cada transacción iniciada a lo largo de la suite de tests. Una vez los test han terminado de ejecutarse, crea o sobreescribe un archivo con una tabla que contiene principalmente, el gas medio utilizado por cada transacción.

Además, se ha generado una clave de API para los servicios que ofrece CoinMarketCap, una analizador de precios de las varias criptomonedas existentes. Haciendo uso de esta API, Gas Reporter es capaz de averiguar el precio actual del gas en la red de Ethereum y traducir ese precio en ETH a una divisa tradicional, como el euro.

Una vez se ha configurado correctamente, cada vez que se ejecuten los tests se volcará en un archivo, una tabla de texto como la de la Figura 39 .

Solc version: 0.8.9	Optimizer enabled: true	Runs: 600	Block limit: 30000000 gas		
Methods		14 gwei/gas		1629.92 eur/eth	
Contract	Method	Min	Max	Avg	# calls	eur (avg)
RewardCenter	createRewardPlan	-	-	3098832	2	70.71
RewardPlan	addFounder	-	-	157303	2	3.59
RewardPlan	addNotifier	158851	173349	166100	4	3.79
RewardPlan	addRewardRule	107828	122415	113346	7	2.59
RewardPlan	awakePlan	-	-	33690	2	0.77
RewardPlan	beginSigningStage	-	-	33117	2	0.76
RewardPlan	notifyPointsScored	63580	154437	110606	15	2.52
RewardPlan	removeRewardRule	-	-	43241	1	0.99
RewardPlan	sign	58959	65384	62172	4	1.42
RewardPlan	signUpClient	159962	159974	159968	4	3.65
Deployments				% of limit		
RewardCenter		-	-	4433756	14.8 %	101.17

Figura 39: Salida de la extensión gas-reporter al ejecutar los tests de la plataforma.

Se puede ver que aunque los precios sean mucho menores que en la reciente temporada de máximos históricos en los precios del gas, siguen resultando lo suficientemente altos para dificultar en exceso la implementación del sistema en un caso real.

Conclusiones

A lo largo del proceso se han ido recabando conocimientos y habilidades que son necesarios para el desarrollo de un sistema basado en blockchain. Principalmente, las bases teóricas que respaldan esta tecnología, la configuración del entorno de trabajo, el desarrollo de contratos inteligentes, sus mecanismos de interacción y el desarrollo de una interfaz.

Aunque en este trabajo se ha intentado abarcar la mayor cantidad de contenido respecto a la tecnología blockchain, sin duda queda mucho por investigar. Por ejemplo, existe toda un área de estudio enfocada en la seguridad de los contratos inteligentes, debido a la presencia de activos económicos reales. También existen varios mecanismos que dotan a un sistema basado en blockchain la capacidad de realizar mejoras y actualizaciones, manteniendo el aspecto de descentralización ya que éstas se deciden a través un sistema de votos donde participan todos o determinados miembros de esa comunidad. Además, existen muchísimas redes de este tipo con características particulares pensadas para diversos ámbitos de aplicación.

Una vez se ha terminado el proyecto, es fácil señalar una serie de mejoras que aportarían mucho beneficio a la plataforma. Quizás la más importante es la migración de la plataforma a una red de bloques de menor costo transaccional como Solana, ya que de lo contrario resultaría inviable poner esta clase de sistema en funcionamiento en un contexto real.

Respecto al diseño de la plataforma, se podrían incluir más tipos de reglas de recompensa o mejor aún, reglas de negocio genéricas con un lenguaje de dominio específico para poder aplicar cualquier lógica relativa al negocio, como son las reglas de recompensa a clientes.

Por el lado del desarrollo, sería un gran aporte la mejora y ampliación de los tests, así como su realización para el cliente web. Otra mejora considerable sería la creación de una capa de servicios que actuase mediante peticiones HTTP, y contuviese la lógica de interacción con los contratos. Por último, se podría sofisticar el proceso de *User Acceptance Testing* desplegando la plataforma en una testnet de Ethereum, en vez del nodo local de Hardhat, y el despliegue de la interfaz, por ejemplo a través de Github Pages.

En cualquier caso, el objetivo planteado se ha cumplido satisfactoriamente. Aunque el resultado final no esté optimizado ni se pueda considerar adecuado para su puesta en producción, ha sido posible establecer una primera toma de contacto con este mundo. Además, se ha comprobado que se trata de una tecnología muy versátil y con un gran potencial, representando un escalón más del progreso hacia una nueva revolución informática.

Conclusions

During the process of this project, knowledge and skills that are necessary for the development of a blockchain-based system have been gathered. Mainly, the theoretical bases that support this technology, the configuration of the work environment, the development of smart contracts, their mechanisms of interaction and the development of an interface.

Although in this work we have tried to cover the largest amount of content regarding blockchain technology, there is undoubtedly much to investigate. For example, there is a whole area of study focused on the security of smart contracts, due to the presence of real economic assets. There are also several mechanisms that give a blockchain-based system the ability to make improvements and updates, while maintaining the aspect of decentralization since these are decided through a voting system where all or certain members of that community participate. Also, there are many networks of this type with particular characteristics designed for various areas of application.

Once the project is finished, it is easy to point out a series of improvements that would bring great benefit to the platform. Perhaps the most important is the migration of the platform to a lower cost transactional block chain network such as Solana, as otherwise it would be unfeasible to put this type of system into operation in a real context.

In terms of platform design, more types of reward rules or better yet, generic business rules with a specific domain language could be included to apply any logic related to the business, such as customer reward rules.

On the development side, it would be a great contribution to improve and expand the tests, as well as their extension for the web client. Another considerable improvement would be the creation of a service layer that acts through HTTP requests, and contains the interaction logic with the contracts. Finally, the User Acceptance Testing process could be sophisticated by deploying the platform on a Ethereum testnet, instead of the Hardhat local node, and the deployment of the web client as a static page, for example through Github Pages.

In any case, the proposed goal has been satisfactorily met. Although the final result is not optimized or can be considered suitable for production, it has been possible to establish a first contact with this world. In addition, it has been verified that this is a very versatile technology with great potential, representing a further step in the progress towards a new computer revolution.

Capítulo 5

Presupuesto

Todas las herramientas de desarrollo han sido de uso libre de cargos, pues la mayoría siguen un planteamiento de software libre. Esto es a excepción de los servicios de Github Copilot que sí tienen un coste de aproximadamente 10 € por mes. Cabe mencionar que Github ofrece planes para estudiantes con los que se puede utilizar esta herramienta de manera gratuita.

Por otro lado, hay que tener en cuenta las horas destinadas al proyecto. Este tiempo ronda unas 300 horas distribuidas a lo largo de nueve meses. Si la retribución para el desarrollador es de 30 € cada hora, y se suma el plan de Github Copilot, el presupuesto total del proyecto asciende a 9.364,65 €. En la Tabla 1 se desglosan todos los costes del proyecto, separados por fase y concepto.

Fase	**Concepto**	**Unidades (horas)**	**Precio por unidad (€/hora)**	**Costo (€)**
Análisis	Mano de obra	75	30,00	2.250,00
Diseño	Mano de obra	75	30,00	2.250,00
	Plan Github Copilot	75	0,33	24,75
Desarrollo	Mano de obra	100	30,00	3.000,00
	Plan Github Copilot	100	0,33	330,00
Testing	Mano de obra	30	30,00	900,00
	Plan Github Copilot	30	0,33	9,90
Documentación	Mano de obra	20	30,00	600,00
Total	N/A	300	31,22	9.364,65

Tabla 1: Presupuesto del proyecto desglosado por periodos y conceptos.

Bibliografía

[1] Shermin Voshmgir. (2020). How the Web3 reinvents the Internet. Recuperado de https://github.com/Token-Economy-Book/EnglishOriginal/wiki.

[2] Javier Pastor. (2017). Qué es blockchain. Recuperado de https://www.xataka.com/especiales/que-es-blockchain-la-explicacion-definitiva-para-la-tecnologia-mas-de-moda.

[3] DevsWiki. (2021). Blockchain & Cryptocurrency A-Z Complete Masterclass | Learn How To Build Your First Blockchain. Recuperado de https://www.youtube.com/watch?v=dn1QsirJ8gk&t=5159s.

[4] Wikipedia. Blockchain.(2022). Recuperado de https://en.wikipedia.org/wiki/Blockchain.

[5] Gorka Ramirez. (2015). MD5: The broken algorithm. Recuperado de https://www.avira.com/en/blog/md5-the-broken-algorithm#:~:text=The%20probability%20of%20just%20two,%3A%201.47*10%2D29.&text=SHA256%3A%20The%20slowest%2C%20usually%2060,%3A%204.3*10%2D60.

[6] Vitalik Buterin. (2017). The Meaning of Decentralization. Recuperado de https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274.

[7] Cardano. (2022). Cardano Official Documentation. https://docs.cardano.org/.

[8] Satoshi Nakamoto. (2008). Bitcoin whitepaper. https://bitcoin.org/bitcoin.pdf.

[9] Axel Nieto. (2021). La historia y la evolución de la Blockchain. Recuperado de https://medium.com/swissborg/la-historia-y-la-evoluci%C3%B3n-de-la-cadena-de-bloques-blockchain-6c2495dbe391.

[10] Ana Castaño. (2021). 7 aplicaciones de blockchain que revolucionan la economía y la sociedad. Recuperado de https://www.gextor.es/7-aplicaciones-de-blockchain-que-revolucionan-la-economia-y-la-sociedad/.

[11] Miguel Muñoz. (2021). ¿Qué es y cómo identificar un Rug Pull?. Recuperado de https://blog.bitnovo.com/que-es-y-como-identificar-un-rug-pull/.

[12] Adrián Rodriguez. (2019).¿Cuál es el problema de la escalabilidad de blockchain? Recuperado de https://adr-rod87.medium.com/qu%C3%A9-pasa-con-la-escalabilidad-de-blockchain-47b7a5a91013.

[13] Ethereum Org. (2022). Main Page. Recuperado de https://ethereum.org/es/.

[14] Şeref Bülbül, Gökhan İnce. (2018). Blockchain-based Framework for Customer Loyalty Program. Recuperado de https://ieeexplore.ieee.org/abstract/document/8566642/authors#authors.

[15] IBM. (2022). What is blockchain?. Recuperado de https://www.ibm.com/es-es/topics/what-is-blockchain.

[16] Hong-Ning Dai, Zibin Zheng, Yan Zhang. (2020). Blockchain for Internet of Things: A Survey. Recuperado de https://arxiv.org/abs/1906.00245.

[17] Alejandro Alija. (2018). El papel de blockchain en la seguridad y privacidad de los datos. Recuperado de https://datos.gob.es/es/blog/el-papel-de-blockchain-en-la-seguridad-y-privacidad-de-los-datos.

[18] Javier Alonso Lecuit. (2019). La seguridad y la privacidad del blockchain, más allá de la tecnología y las criptomonedas. Recuperado de

https://www.realinstitutoelcano.org/analisis/la-seguridad-y-la-privacidad-del-blockchain-mas-alla-de-la-tecnologia-y-las-criptomonedas/

[19] Ethereum Org. (2022). Ethereum Mainnet Statistics. Recuperado de https://ethernodes.org/.

[20] Bit2me Academy. (2022). ¿Qué es la Ethereum Virtual Machine (EVM)?. Recuperado de https://academy.bit2me.com/que-es-ethereum-virtual-machine-evm/.

[21] Solidity Org. (2022). Solidity Official Documentation. Recuperado de https://docs.soliditylang.org/en/v0.8.12/

[22] Manoj Pramesh. (2020). Solidity Cheat Sheet and Best practices. Recuperado de https://github.com/manojpramesh/solidity-cheatsheet#struct.
Rob Hitchens. (2017). Blog: Simple Storage Patterns in Solidity. Recuperado de https://ethereum.meta.stackexchange.com/questions/443/blog-simple-storage-patterns-in-solidity.

[23] Franz Volland. (2018). Solidity Patterns - State Machine. Recuperado de https://github.com/fravoll/solidity-patterns/blob/master/docs/state_machine.md.

[24] Open Zeppelin. (2022). Open Zeppelin Documentation. Recuperado de https://docs.openzeppelin.com/contracts/4.x/.

[25] A.Ufano. (2021). Solidity fundamentals: working with dates and time. Recuperado de https://soliditytips.com/articles/solidity-dates-time-operations/.

[26] Jake Frankenfield. (2021). Wei. Recuperado de https://www.investopedia.com/terms/w/wei.asp.

[27] Elena Gesheva. (2020). Solidity 0.6.x Features: Fallback and Receive Functions. Recuperado de https://betterprogramming.pub/solidity-0-6-x-features-fallback-and-receive-functions-69895e3ffe.

[28] Ethers JS. (2022). Ether JS Official Documentation. Recuperado de https://docs.ethers.io/v5/.

[29] Hardhat JS. (2022). Hardhat Official Documentation. Recuperado de https://hardhat.org/docs.

[30] Mocha. (2022). Asynchronous Code Testing. https://mochajs.org/#asynchronous-code.

[31] Patrick Collins. (2022). How to Connect your Smart Contracts to Metamask | Full Stack Web3. Recuperado de https://www.youtube.com/watch?v=pdsYCkUWrgQ&t=3307s.

[32] Klaus Hott Vidal. (2019). Prettier for Solidity - Style Guide. Recuperado de https://github.com/prettier-solidity/prettier-plugin-solidity/blob/main/STYLEGUIDE.md

[33] Sebastián Tamayo Guzmán. (2022). Repositorio con el código asociado al trabajo final de grado. Recuperado de https://github.com/alu0101131108/TFG-CustomerRewardPlatform