UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN
TECNOLOGÍAS DE LA INFORMACIÓN

"Desarrollo de un Marketplace para la colección NFT Mr. Crypto"

TRABAJO FIN DE GRADO

Mayo - 2024

AUTOR: Samuel Lajara Agulló
DIRECTOR: Jesús Javier Rodríguez Sala

RESUMEN

Este proyecto web es un marketplace dedicado a la compra y venta de una colección única de NFTs llamada "Mr. Crypto". La mayoría de los NFTs utilizan como contrato estándar el denominado ERC721. Mr. Crypto, utiliza un estándar adicional que introduce los NFT linkables a ERC721 (E7L). En otras palabras, cada uno de estos tokens no fungibles (NFT) únicos, puede tener múltiples NFTs E7L asociados, creando una relación de "padres e hijos", donde un Mr. Crypto puede tener varios E7L, pero un E7L solo puede estar vinculado a un Mr. Crypto específico.

Mr. Crypto es una comunidad de usuarios y aficionados al mundo de la blockchain y las criptomonedas creada por la empresa Racks Labs. Pretende ser un club exclusivo, y cada NFT de la colección Mr. Crypto es el avatar/identificador de cada uno de los miembros de dicho club. Esta comunidad organiza diversos tipos de eventos y también crea ciertas utilidades a los cuales se accede, bien directamente o bien con descuentos o facilidades para los poseedores de uno de estos tokens. Cuando alguno de estos eventos y/o utilidades produce otro NFT representativo de dicho evento o utilidad, este nuevo NFT, que será de tipo E7L, se vincula al NFT Mr. Crypto del usuario, de tal forma que esté “avatar/identificador” va almacenando una especie de historial de las operaciones realizadas por su propietario.

El marketplace ofrece a los usuarios la posibilidad de explorar, comprar y vender los Mr. Crypto, así como explorar sus E7L relacionados. Con el tiempo, se espera que un token Mr. Crypto vaya acumulando valor por la cantidad y/o calidad de tokens E7L vinculados a él. Este marketplace incorporará la funcionalidad de pujas, que permitirá a los coleccionistas competir por estas valiosas piezas.

AGRADECIMIENTOS

Quiero expresar mi agradecimiento a los compañeros de equipo que han colaborado en este proyecto y me han brindado la oportunidad de aprender sobre Web3 y Blockchain.

Agradezco al equipo de Racks Labs, la empresa detrás de la colección "Mr. Crypto", por brindarnos la oportunidad de trabajar en este proyecto.

Además, agradezco a mi familia por su apoyo durante mis años de estudio y graduación. Su respaldo ha sido fundamental para mi éxito.

Finalmente, agradezco a Jesús Javier Rodríguez, mi director del trabajo, por su apoyo y paciencia aprendiendo sobre una tecnología tan nueva.

Este proyecto no habría sido posible sin la colaboración y el apoyo de estas personas e instituciones. Les agradezco a todos por su contribución.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ALGORITMOS

Capítulo 1
Introducción

1.1.- DIFERENCIAS ENTRE WEB2 Y WEB3

La transición de la Web 2.0 a la Web 3.0 se caracteriza por una evolución gradual en lugar de cambios drásticos. Aunque a primera vista puede no ser fácilmente distinguible, es posible identificar tendencias y características que marcan la transformación. No solo estamos siendo testigos de avances tecnológicos, sino que también observamos cómo la relación entre los usuarios y la red digital avanza, reconfigurando conceptos como la participación del usuario en las aplicaciones en línea o la propiedad de sus datos en dichas aplicaciones. [1]

La Web 2.0 introdujo la interactividad a través de redes sociales y creación de contenido. En la Web 3.0, la participación del usuario se amplía, permitiendo decisiones descentralizadas y un papel más activo en la toma de decisiones en las diversas plataformas. Un ejemplo destacado son los sistemas de finanzas descentralizadas o DeFi

(abreviatura en inglés de "*Decentralized Finance*"), que pretenden convertir las estructuras de finanzas centralizadas que tenemos ahora en estructuras descentralizadas, sin terceros de confianza, ejecutadas sobre smart contracts, dentro de una blockchain donde quede reflejado de forma transparente un registro escrito e inalterable de cada transacción realizada. [2]

Mientras la Web 2.0 se centralizó en plataformas corporativas, la Web 3.0 abraza la descentralización, una ventaja de esto es que las publicaciones en Web3 no serían censurables precisamente porque el control es descentralizado. Blockchain y smart contracts eliminan intermediarios, proporcionando mayor seguridad, control sobre datos y transacciones. Un ejemplo concreto es el uso de smart contracts en NFTs o Tokens No Fungibles (Non-Fungible Token) [3], donde la propiedad y transferencia de activos digitales se realiza de forma automatizada y sin intermediarios (los diferentes tipos de tokens se explican con más detalle en el capítulo 2).

En la Web 3.0, los usuarios ganan control sobre sus datos. La descentralización basada en blockchain redefine la propiedad, permitiendo a los usuarios decidir cómo, cuándo y con quién comparten su información. [4]

1.1.1.- El reto del doble gasto

El concepto del doble gasto se refiere a un desafío en el mundo de las criptomonedas, la posibilidad de que una unidad de moneda digital se utilice más de una vez. Esta es una cuestión fundamental que se debe prevenir desde un punto de vista tecnológico. El doble gasto amenaza la integridad y la fiabilidad de una blockchain, una base de datos no sólo resistente a alteraciones sino que además registra de manera precisa cada transacción realizada en su red. La capacidad de realizar un doble gasto acabaría con la confianza depositada en criptomonedas como Bitcoin o en cualquier otra infraestructura basada en blockchain.

La solución al problema se encuentra en un protocolo basado en el uso de la tecnología blockchain junto con un sistema de consenso distribuido, donde múltiples nodos verifican y registran de manera independiente cada transacción en un libro contable digital compartido por todos los nodos, y público para cualquiera que quiera consultarlo. En el caso de Bitcoin, por ejemplo, se utiliza un algoritmo de prueba de trabajo (Proof of Work), que requiere que los mineros resuelvan problemas matemáticos computacionalmente complejos para validar las transacciones y añadirlas a la blockchain. Todo este proceso no solo garantiza la seguridad y la integridad de cada transacción, sino que también impide que la misma unidad de criptomoneda sea gastada más de una vez, asegurando así la fiabilidad y la confianza en todo el sistema. [5]

1.1.2.- El problema de los generales bizantinos

El problema de los generales bizantinos es una analogía utilizada para describir un dilema de confianza y coordinación en sistemas distribuidos, como lo es una blockchain. Este problema se imagina a un grupo de generales que deben acordar un plan de acción unificado, pero se encuentran separados y solo pueden comunicarse a través de mensajeros. La dificultad radica en asegurar que todos los generales lleguen al mismo plan de acción, incluso cuando algunos de ellos podrían ser traidores intentando sabotear la decisión.

Esta situación es similar a los desafíos a los cuales se enfrentan los sistemas Web3 por el hecho de ser descentralizados. En estos sistemas, diversos nodos (equivalentes a los generales) deben llegar a un consenso sobre el estado actual de la información contenida en la cadena de bloques, a pesar de la posible (y muy probable) presencia de nodos maliciosos o no confiables que puedan pretender alterar ilegítimamente la información para su propio beneficio. El problema se resuelve mediante la aplicación de protocolos de consenso, que garantizan que todos los nodos honestos y confiables lleguen a un acuerdo común, manteniendo la integridad y la seguridad de la red. [6]

1.1.3.- La solución de Satoshi Nakamoto

La solución de Satoshi Nakamoto al problema del doble gasto y los generales bizantinos revolucionó la tecnología blockchain, siendo clave en el desarrollo de Bitcoin. Nakamoto propuso un sistema de consenso basado en la prueba de trabajo (Proof of Work). En este sistema, los mineros compiten para resolver un problema matemático complejo y así validar transacciones y crear nuevos bloques, poniendo muy difícil y caro a los nodos maliciosos alterar la blockchain. Esto garantiza la seguridad y la integridad de la cadena de bloques, haciendo casi imposible el fraude o el doble gasto.

Esta innovación ha dado lugar a una red descentralizada y transparente donde todos los participantes tienen una copia exacta del libro contable. Este enfoque descentralizado no solo resuelve el problema de confianza en la red, sino que también establece un nuevo paradigma de seguridad y transparencia en el mundo digital. [7]

1.2.- ¿QUÉ ES UNA DAO?

Una DAO, o Decentralized Autonomous Organization (Organización Autónoma Descentralizada), es una entidad organizativa que opera de manera autónoma y descentralizada, utilizando smart contracts y la tecnología de blockchain para tomar

decisiones y ejecutar acciones sin la necesidad de una autoridad central [6]. Aquí hay tres aspectos clave para entender una DAO:

- Una DAO se caracteriza por su estructura descentralizada, lo que significa que no está controlada por una entidad central, como un gobierno o una empresa. En lugar de depender de una jerarquía tradicional, las decisiones en una DAO se toman de manera colectiva mediante la participación de los poseedores de tokens en la red blockchain.

- Las decisiones en una DAO se facilitan a través de smart contracts, estos contratos definen reglas y condiciones específicas para la toma de decisiones, votaciones y ejecución de acciones. Los participantes votan mediante sus tokens, y las decisiones se implementan automáticamente según las reglas establecidas en el smart contract.

- La propiedad y la participación en una DAO están vinculadas a la posesión de tokens específicos asociados con esa organización. Los titulares de tokens tienen derechos de voto proporcionales a la cantidad de tokens que poseen, lo que les otorga influencia en la toma de decisiones. Esta estructura fomenta la participación activa de los miembros y alinea los intereses de la comunidad con el éxito y la dirección de la DAO.[8]

1.2.1.- Smart Contract

Un smart contract, es una pieza clave en los sistemas Web3 descentralizados. Se trata de un acuerdo digital que se almacena en la blockchain y se ejecuta automáticamente cuando se cumplen ciertas condiciones predefinidas. A diferencia de los contratos tradicionales, los smart contracts no requieren intermediarios humanos ni notarios, lo que reduce costos y tiempos.[9]

Consideremos un ejemplo genérico en la industria de productos alimenticios. Imaginemos un sistema de trazabilidad para un producto como el vino o el queso. En vez de depender de métodos tradicionales para verificar la procedencia y calidad del producto, se podría utilizar un smart contract en toda la cadena de suministro. Este contrato se activaría automáticamente en cada etapa del proceso: desde la producción inicial, como la cosecha de uvas o la recolección de leche, hasta el procesamiento y la distribución final del producto. Por ejemplo, cuando se cosechan las uvas en una viña determinada, el smart contract registraría este evento. Conforme el vino se fermenta, se embotella y luego se distribuye, el contrato actualizará su estado en cada paso, asegurando un registro detallado y transparente de su origen y recorrido. De esta manera, cuando un consumidor compra una botella de vino, puede verificar fácilmente su trazabilidad, confirmando que el

producto es auténtico y proviene de la región específica indicada, garantizando así la calidad y procedencia del producto.

Los smart contracts están escritos en lenguajes de programación, se almacenan en una blockchain, y se ejecutan en los nodos que soportan dicha red blockchain. Son inmutables (como su blockchain), lo que significa que no se pueden cambiar una vez que se registran. Además, son transparentes, lo que permite a todas las partes involucradas auditar su código y verificar su funcionamiento. [9]

1.2.2.- Wallet

En el mundo Web3 de los smart contracts y las DAOs, las wallets son los puntos clave de acceso. En términos simples, una wallet es un software o dispositivo que permite almacenar, gestionar y realizar transacciones de forma segura. Funciona como un depósito digital que guarda las claves privadas y públicas necesarias para acceder y gestionar tokens y criptomonedas en la cadena de bloques. La clave privada desempeña un papel crítico, se utiliza para firmar transacciones y demostrar la propiedad de los activos digitales almacenados.

Existen dos categorías principales de wallets, una hot wallet está conectada a Internet y permite realizar transacciones en tiempo real. Un ejemplo destacado de hot wallet es MetaMask, una extensión de navegador que facilita la interacción con aplicaciones descentralizadas (dApps) en la red Ethereum. Por otro lado, las cold wallets, como Ledger, son dispositivos físicos que almacenan las claves privadas de manera offline, desconectados de Internet. Esta característica proporciona un nivel adicional de seguridad, ya que las transacciones se firman internamente en el dispositivo físico antes de ser transmitidas a la red blockchain. [10][11]

Cuando se crea una nueva wallet, se genera una frase semilla compuesta por al menos 12 palabras. Esta frase es esencial, ya que sirve como clave para iniciar o recuperar la wallet en caso de pérdida de acceso. La seguridad y la gestión adecuada de esta frase semilla son cruciales para garantizar la integridad y el acceso continuo a los activos digitales almacenados en la blockchain.

1.3.- TECNOLOGÍA Y COMUNIDAD

Antes de profundizar en la justificación y los objetivos de este proyecto, es esencial entender algunos conceptos clave que forman su base. Este apartado proporciona una visión general de tres elementos fundamentales: los marketplaces de NFTs, la comunidad

Mr. Crypto y la empresa Racks Labs. Cada uno de estos elementos desempeñan un papel vital en el contexto de este proyecto, y su comprensión nos permite apreciar mejor el alcance y la relevancia del proyecto.

1.3.1.- ¿Qué es un MarketPlace de NFTs?

Un marketplace de NFTs es una plataforma digital donde se compran, venden y subastan NFTs (Tokens No Fungibles). Estas plataformas facilitan las transacciones entre usuarios y proporcionan un espacio para que artistas y creadores presenten y moneticen sus obras digitales en forma de NFTs.

En estos marketplaces, se encuentran disponibles variadas colecciones de NFTs alojadas en diferentes blockchains. El uso de estas plataformas es sencillo: los usuarios sólo necesitan conectar su wallet, tener el saldo necesario y firmar la transacción para realizar la acción.

Algunos ejemplos conocidos de marketplaces de NFTs son OpenSea [12], MagicEden [13] y Blur [14]. Cada uno de estos ofrece distintas características y facilita el intercambio de activos digitales en un entorno seguro y transparente.

1.3.2.- La colección de Mr. Crypto

El uso de los NFTs (Tokens No Fungibles) abarca varias aplicaciones, siendo la creación de comunidades una de ellas. Aunque a primera vista no parezca tener una utilidad directa, los NFTs pueden fomentar la formación de grupos con intereses comunes. La tecnología de los NFTs se explicará con más detalle en el capítulo 2, enfocándonos en los NFTs tipo ERC-721.

La comunidad de Mr. Crypto es un ejemplo de cómo los NFTs pueden servir para unir a personas con intereses similares. En esta comunidad, ligada a una marca de ropa (detalle que se ampliará en el punto 1.3.3), los miembros que poseen NFTs obtienen beneficios como descuentos en productos y servicios. Esto incluye reducciones en tiendas asociadas, ofertas en apartamentos turísticos y entrenamientos personales proporcionados por otros miembros, así como acceso preferente y promociones en eventos. [15]

Personalmente, mi participación en la comunidad de Mr. Crypto ha sido beneficiosa. Poseer un NFT de esta colección me ha dado la oportunidad de asistir a eventos específicos y conocer a personas interesadas en desarrollar proyectos. Esta experiencia demuestra que el valor de los NFTs trasciende lo económico, contribuyendo también al enriquecimiento personal y profesional.

1.3.3.- Racks Labs

Racks es una startup que comenzó en el sector de la moda en España, específicamente bajo el nombre de Racks Mafia. Los fundadores, con un fuerte enfoque en el sector cripto y un amplio alcance en redes sociales, lanzaron la colección de NFT Mr. Crypto. Esta colección consistió en 10.000 NFTs y logró alcanzar el 'sold out' en tan solo 1 minuto.

Tras el éxito de su lanzamiento en el ámbito de los NFTs, Racks comenzó a recibir numerosos proyectos tecnológicos, lo que impulsó a la empresa a expandir su enfoque. Esta creciente demanda de sus habilidades en tecnología llevó a los fundadores a una nueva dirección. En 2023, dieron un paso adelante al lanzar oficialmente Racks Labs, una startup que se posiciona como consultora tecnológica. Esta iniciativa representó un cambio significativo para la empresa, que pasó de estar centrada en la moda a involucrarse plenamente en el desarrollo tecnológico, especialmente en soluciones basadas en blockchain. [16]

1.4.- JUSTIFICACIÓN DEL PROYECTO

Este proyecto es fruto de una motivación personal. Como miembro activo de la comunidad de NFTs Mr. Crypto y tras haber tenido la oportunidad de interactuar con los desarrolladores en diversos eventos, he adquirido un profundo interés en el mundo de las criptomonedas y la tecnología blockchain.

1.4.1.- Aprender desarrollo blockchain

Desde mi participación en la comunidad de Mr. Crypto y mis interacciones con los desarrolladores, mi interés por el mundo de las criptomonedas y la blockchain ha ido en constante crecimiento. Este proyecto es una oportunidad única para sumergirme en este apasionante campo, adquirir conocimientos prácticos y profundizar en la tecnología blockchain, que es una de las tendencias más importantes en la actualidad.

1.4.2.- Contribuir a la Comunidad de Mr. Crypto

Más allá de mi motivación personal, con este proyecto también se busca contribuir al crecimiento y desarrollo de la comunidad de Mr. Crypto. La creación de un marketplace específico para esta colección no solo beneficia a los propios coleccionistas, sino que también promueve la adopción de la tecnología E7L (ver capítulo 2) desarrollada por

Racks Labs (empresa desarrolladora de la colección de Mr. Crypto), lo cual es un aspecto importante para el mundo de los NFTs. Este proyecto no solo representa un aprendizaje y reto personal, sino también un aporte significativo a una comunidad en constante crecimiento.

1.5.- OBJETIVOS

Los objetivos de este proyecto son variados y abarcan tanto aspectos técnicos como personales. Se plantea como un desafío integral que busca combinar la adquisición de habilidades tecnológicas con la aplicación práctica de conocimientos y el trabajo en equipo.

1.5.1.- Objetivos principales

En el centro de estos objetivos se encuentra la creación de una aplicación web completa, aprovechando tecnologías de vanguardia como Next.js y Node.js. Además, se busca el aprendizaje de Solidity, el lenguaje esencial para desarrollar smart contracts en la blockchain. Paralelamente, se persigue adquirir experiencia en trabajo en equipo, siguiendo la metodología SCRUM. Esto engloba desde la planificación y asignación de tareas hasta el seguimiento del progreso y una colaboración eficiente con los compañeros de equipo para alcanzar los objetivos del proyecto de manera efectiva.

1.5.2.- Funcionalidad del Marketplace

En marketplaces como OpenSea, se encuentran muchas colecciones, incluida Mr. Crypto. El marketplace del proyecto está diseñado principalmente para comprar y vender NFTs de esta colección. Además, se destaca por dar visibilidad a los E7L, un tipo de NFT nuevo que no se encuentra en otros marketplaces comunes.

1.5.3.- Metas Personales

Junto a los objetivos principales, existen metas personales que hacen más valiosa mi participación en el proyecto Mr. Crypto. Uno de ellos es la oportunidad de adquirir experiencia en nuevas tecnologías, mejorando mis habilidades técnicas. Este proyecto proporciona la plataforma ideal para aplicar los conocimientos adquiridos durante mi grado universitario en un entorno real y práctico. Aunque no es el objetivo principal, existe la posibilidad de que este proyecto, una vez finalizado, pueda generar oportunidades de

ingresos, lo que sería un beneficio adicional. En resumen, estos objetivos combinan desarrollo técnico, aplicación de conocimientos académicos y exploración de oportunidades, ayudando tanto a mi crecimiento personal como profesional.

Capítulo 2 Antecedentes y estado de la cuestión

En este capítulo, se analizarán las transacciones y los estándares en la blockchain. Vamos a profundizar en cómo se realizan, verifican y registran las transacciones digitales en esta tecnología clave detrás de las criptomonedas y los smart contracts. También nos centraremos en estándares esenciales como ERC-20 y ERC-721, fundamentales para entender la interoperabilidad y la funcionalidad de los tokens en la red Ethereum. Se abordarán tanto los aspectos técnicos de las transacciones y los estándares como su relevancia práctica en el mundo de los NFTs y la Web3, buscando comprender mejor cómo estas tecnologías están marcando nuevos rumbos en el sector.

2.1.- LAS TRANSACCIONES EN BLOCKCHAIN

Una transacción implica transferir un valor específico de una criptomoneda o token de una wallet a otra. Se registra en la red correspondiente de dicha criptomoneda y se añade a la

blockchain, el libro contable digital que registra todas las transacciones de esa moneda. Para enviar criptomonedas, se necesita acceso a las claves privadas de la wallet del emisor y a las claves públicas de la wallet del destinatario. La clave pública actúa como la dirección de la wallet para recibir criptomonedas, mientras que la clave privada es secreta y se usa para firmar la transacción además de validar la propiedad de los fondos.

Una vez que una transacción es emitida, entra en la mempool, un área de espera donde las transacciones son validadas por nodos y esperan ser seleccionadas por los mineros para su confirmación. Este proceso es fundamental en muchas redes como Bitcoin y Ethereum. Los mineros seleccionan las transacciones de la mempool y las incluyen en bloques, utilizando el sistema de Prueba de Trabajo (Proof of Work, PoW) en el caso de Bitcoin, y un mecanismo denominado Prueba de Participación (Proof of Stake, PoS) en Ethereum. La confirmación de las transacciones y su inclusión en la blockchain depende de varios factores, incluyendo la comisión de la transacción y la congestión de la red, lo que puede afectar el tiempo de procesamiento. [17]

2.1.1.- Las transacciones en Bitcoin

La verificación de las transacciones se realiza a través de un proceso denominado "*minería*" que utiliza PoW, que no es más que la resolución de un problema matemático complejo para validar las transacciones y añadir nuevos bloques a la blockchain. Los mineros, que participan en la red con sistemas de computación de alto rendimiento, reciben recompensas en la criptomoneda correspondiente y las comisiones de las transacciones que se incluyen en el bloque. Las transacciones con comisiones más altas suelen ser priorizadas debido al límite de espacio en cada bloque. El tiempo de procesamiento de una transacción de Bitcoin puede variar dependiendo de la congestión de la red, el tamaño de la transacción y la comisión pagada. En promedio, una transacción de Bitcoin puede tardar unos 10 minutos en recibir la primera confirmación. [18]

Además del proceso de minería en las transacciones de Bitcoin, hay varios aspectos relevantes. Uno de los temas más discutidos en relación con Bitcoin es el consumo de energía asociado con el proceso de minería. Dado que la minería de Bitcoin requiere una cantidad significativa de poder computacional, el consumo energético para mantener la red es considerable. Este gasto energético ha generado debates sobre la sostenibilidad y el impacto ambiental de Bitcoin, especialmente en comparación con métodos más eficientes energéticamente, como la Prueba de Participación (PoS) que se utiliza en otras criptomonedas. [19]

Otra área de interés en las transacciones de Bitcoin es la seguridad y la robustez de la red. La red de Bitcoin, siendo la más antigua y una de las más probadas, ofrece un alto nivel de seguridad y resistencia a ataques, lo que ha sido un factor clave en su adopción y

confiabilidad. Sin embargo, esto también viene con ciertas limitaciones, como la velocidad de procesamiento de transacciones y la escalabilidad, que son aspectos en los que otras criptomonedas han intentado mejorar.

Finalmente, cabe mencionar la naturaleza descentralizada y la transparencia de las transacciones en Bitcoin. Al ser una red descentralizada, Bitcoin permite a los usuarios realizar transacciones sin la necesidad de intermediarios, ofreciendo un grado de autonomía y privacidad. A su vez, todas las transacciones son registradas en la blockchain, lo que proporciona un registro transparente y permanente.

2.1.2.- Transacciones en Ethereum

Las transacciones en Ethereum originalmente eran validadas mediante el sistema PoW. Sin embargo, el 15 de septiembre de 2022, Ethereum realizó una actualización significativa en su mecanismo de consenso, migrando al sistema PoS. Este cambio marcó un punto de inflexión en la forma en que se procesan las transacciones en la red.

La actualización, conocida como 'The Merge', fue un cambio significativo en su mecanismo de consenso previo. En este sistema, los nodos validadores (ya no son "mineros") juegan un papel crucial en la verificación de las transacciones y la adición de nuevos bloques a la blockchain. A diferencia de la minería que requiere poder computacional extensivo, la Prueba de Participación permite a los validadores participar en el proceso de consenso al bloquear una cierta cantidad de Ether (ETH), la criptomoneda nativa de Ethereum. [20]

En PoS, los validadores son seleccionados para crear un nuevo bloque, basándose en la cantidad de criptomonedas que han bloqueado y otros factores como el tiempo que han mantenido su bloqueo. Una vez seleccionados, validan las transacciones y las agregan a la blockchain. Por su trabajo, los validadores reciben recompensas en Ether, que incluyen las comisiones de las transacciones del bloque que han validado. [21]

Una característica distintiva de Ethereum es su capacidad para ejecutar smart contracts, que son programas que se ejecutan automáticamente en la propia red cuando se cumplen condiciones predefinidas. Cada transacción en Ethereum puede involucrar la ejecución de estos contratos, lo que requiere un cálculo denominado "*gas*". El gas es una medida del poder computacional necesario para ejecutar esas operaciones y contratos. Los usuarios deben pagar una tarifa de gas para que sus transacciones y smart contracts puedan ejecutarse en la red. [22]

El tiempo de procesamiento de una transacción en Ethereum puede variar, pero generalmente es más rápido que en sistemas basados en PoW, como el antiguo Ethereum o

Bitcoin. La implementación de PoS también busca abordar problemas de escalabilidad y eficiencia energética, haciendo a Ethereum más sostenible y capaz de manejar un mayor volumen de transacciones.

Las ventajas de las transacciones en Ethereum incluyen su naturaleza descentralizada, la capacidad de ejecutar smart contracts complejos y una mayor eficiencia energética con PoS. Sin embargo, también enfrenta desafíos, como costos de gas variables y la complejidad en la gestión y comprensión de contratos inteligentes.

2.1.3.- Ventajas e inconvenientes

Las transacciones en la blockchain tienen ventajas y desventajas en comparación con otros métodos de pago [18], algunas de las ventajas son:

- Son descentralizadas, lo que significa que no dependen de ninguna autoridad central o intermediario que pueda censurarlas, bloquearlas o manipularlas.

- Son globales, por lo que se pueden enviar y recibir desde cualquier parte del mundo, sin importar las fronteras o las restricciones geográficas.

- Son transparentes, es decir, pueden ver y auditar en la cadena de bloques, sin necesidad de confiar en terceros.

- Son seguras, están protegidas por la criptografía y por el consenso de la red, que impide que se puedan falsificar o duplicar.

- Son rápidas, por tanto, se pueden confirmar en cuestión de minutos, sin importar la distancia o el horario.

- Son baratas, tienen comisiones muy bajas o nulas, en comparación con otros sistemas de pago que cobran altas tasas o comisiones ocultas.

Algunas de las desventajas son:

- Son irreversibles, lo que significa que una vez que se envían, no se pueden cancelar ni devolver, lo que implica un riesgo de pérdida o fraude si se envían a una dirección equivocada o maliciosa.

- Son volátiles, su valor puede cambiar drásticamente en poco tiempo, esto implica un riesgo de pérdida o ganancia si se almacenan o se cambian por otras monedas.

- Son complejas, requieren de un cierto nivel de conocimiento técnico y de responsabilidad para usarlas correctamente, por tanto, existe un riesgo de pérdida o robo si no se manejan adecuadamente, por ejemplo, si se pierden o se comprometen las claves privadas o las wallets.

- Son limitadas, lo que significa que tienen una capacidad limitada para procesar transacciones, lo que implica un riesgo de retraso o exclusión si la red está saturada o si la comisión es insuficiente.

2.2.- ESTÁNDARES DE TOKENS EN BLOCKCHAIN

En esta sección, se explorarán los estándares de tokens más relevantes en la blockchain, proporcionando una comprensión clara de cómo estos estándares facilitan una amplia gama de aplicaciones y usos. Los estándares ERC-20 y ERC-721 son específicos de la red Ethereum. No se utilizan directamente en otras redes blockchain como Solana o Cardano. Cada una de estas redes tiene sus propios estándares, protocolos para tokens y NFTs.

2.2.1.- Implementación en la Testnet de Polygon

Para este proyecto, se ha tenido que clonar la colección de NFTs en la testnet de Polygon llamada Mumbai, que es básicamente un lugar de pruebas para desarrolladores, separado de la red principal de Ethereum. Polygon es lo que se llama una solución de capa 2 para Ethereum, lo que significa que añade una nueva capa sobre la blockchain existente para hacerla más rápida y eficiente. Al hacer las transacciones fuera de la cadena principal (off-chain), Polygon ayuda a que todo sea más barato y rápido, pero sin perder seguridad. Esto es importante para nosotros, ya que estamos trabajando con tokens ERC-721 y ERC-E7L. En los siguientes apartados 2.2.3 y 2.2.4, se hablará más sobre estos estándares y cómo se utilizan en el proyecto.

2.2.2.- ERC-20

El estándar ERC-20 se refiere a un conjunto de normas definidas para los tokens fungibles en la blockchain de Ethereum. Los tokens fungibles son aquellos que se pueden intercambiar uno por otro, siendo cada uno de igual valor y características, como es el caso de criptomonedas como ETH (Ethereum) o USDT (Tether). El propósito principal de este estándar es asegurar la interoperabilidad entre diferentes tokens y las plataformas que los soportan, incluyendo wallets y exchanges de criptomonedas. [23]

Para que un token sea considerado ERC-20, debe incorporar ciertas funciones y eventos en su smart contract. Esto incluye capacidades para consultar y modificar el saldo del token, su suministro total, símbolo y nombre. También debe permitir la transferencia de tokens entre cuentas y la autorización de tokens para ser utilizados por terceros. Estas especificaciones están detalladas en la Ethereum Improvement Proposal número 20 (EIP-20, propuesta de mejora nº 20), planteada en 2015 por Fabian Vogelsteller, uno de los desarrolladores clave en proyecto Ethereum. [24]

Los tokens ERC-20 han jugado un papel fundamental en el desarrollo del ecosistema de Ethereum. Fueron particularmente prominentes durante el fenómeno de las ofertas iniciales de monedas (ICO) en 2017, un método de recaudación de fondos que muchas startups y proyectos utilizaron para emitir tokens a cambio de inversiones. Entre los ejemplos más conocidos de tokens ERC-20 se encuentran Tether (USDT), Binance Coin (BNB), Chainlink (LINK) y Uniswap (UNI). Estos tokens tienen diversas aplicaciones, desde representar monedas estables hasta servir como tokens de gobernanza o de utilidad en diversas plataformas, aplicaciones y DAOs.[25]

Algoritmo 2.1: Ejemplo código el Solidity

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MockToken is ERC20 {

  constructor(string memory _name, string memory _symbol, uint256 supply)
              ERC20(_name, _symbol) {
    _mint(msg.sender, supply * 10 ** decimals());
  }

}
```

El algoritmo 2.1 muestra un ejemplo de código en Solidity que, al ser desplegado, permite configurar los parámetros iniciales del token. Durante el despliegue del contrato, se introducen los valores para el constructor. Por ejemplo, _name sería 'Ethereum', _symbol sería 'ETH', y supply sería una cantidad representativa del suministro total. Al desplegar este contrato, se crearán automáticamente esa cantidad de tokens y se enviarán a la wallet que realiza el despliegue (se profundizará en el funcionamiento Solidity en el capítulo 3).

2.2.2.- ERC-721

El estándar ERC-721 es un conjunto de reglas para la creación y manejo de tokens no fungibles (NFTs) en la blockchain de Ethereum. A diferencia de los tokens fungibles, los

NFTs son activos digitales únicos e irreemplazables, representando ítems como obras de arte, coleccionables o incluso boletos. Cada token ERC-721 posee un identificador único que lo distingue de otros tokens dentro del mismo smart contract, permitiendo así la diferenciación y singularidad de cada activo. [26]

Para interactuar con tokens ERC-721, se necesita un smart contract que implemente las funciones y eventos definidos en el estándar. Esto incluye capacidades para consultar el saldo de tokens en una cuenta, determinar el propietario de un token específico, transferir tokens de una cuenta a otra, y autorizar a terceros para que manejen tokens en nombre del propietario (las propiedades se verán en el capítulo 3). Estas funcionalidades son esenciales para el manejo seguro y eficiente de los NFTs en la red Ethereum.

Los tokens ERC-721 han sido fundamentales en el desarrollo y popularización de los mercados de NFTs. Permiten la creación de plataformas y aplicaciones descentralizadas (DAPPs) que pueden visualizar y utilizar estos tokens de maneras creativas, como mostrar imágenes, sonidos o animaciones asociadas a cada NFT. Ejemplos destacados de uso de tokens ERC-721 incluyen colecciones de arte digital, juegos blockchain y sistemas de propiedad digital [27]. Este estándar ha revolucionado la forma en que se perciben y comercializan los activos digitales, proporcionando un medio para la autenticación y propiedad digitales inmutables.

Algoritmo 2.2: Ejemplo contrato simplificado

```
 1 contract Ownable{
 2
 3   address public owner;
 4
 5   constructor(){
 6     owner = msg.sender;
 7   }
 8
 9   modifier onlyOwner(){
10     require(msg.sender == owner, "Not owner");
11     _;
12   }
13 }
```

El fragmento de código que se presenta en el algoritmo 2.2, es una versión simplificada de un smart contract de la copia de nuestra colección de NFTs, se trata de un contrato básico para asignar roles, que en este caso se centra en el rol de '*propietario*'. Lo importante aquí es el '*modificador*', una característica de Solidity que restringe el uso de ciertas funciones del contrato a sólo el '*modifier*'. Esto es una práctica común en los smart contracts para controlar el acceso y la ejecución de funciones críticas, se profundiza en el capítulo 3.

Se presenta un segundo contrato en el algoritmo 2.3, este es el de los NFTs ERC-721. En Solidity, que es un lenguaje orientado a objetos, este contrato heredará del contrato de roles

y del estándar '*ERC721Enumerable*'. A través del constructor, se establece el nombre y símbolo del token, además de definir un límite fijo para la cantidad total de NFTs y la ubicación de sus imágenes asociadas. El método '*mint*' (que significa '*acuñar*') se utiliza para crear nuevos NFTs. En este caso, cuando se despliega el contrato, no se generan todos los NFTs de golpe; en cambio, se van creando individualmente cuando el usuario los acuña, cada uno con su propio ID. Por ejemplo, si la colección es de 10.000 y solo se acuñan 7.800, ese sería el '*supply*' actual, mientras que 10.000 sería el '*total supply*'. El último método, '*withdraw*', que utiliza el modificador mencionado anteriormente, permite retirar las criptomonedas acumuladas en el contrato a la dirección del propietario.

Algoritmo 2.3: Ejemplo contrato ERC-721 para NFTs

```
contract MRCRYPTO is ERC721Enumerable, Ownable {

  uint16 public immutable MAX_SUPPLY;

  string public baseURI;

  constructor(string memory _name, string memory _symbol)
             ERC721(_name, _symbol) {
    MAX_SUPPLY = 10000;
    baseURI = "https://...";
  }

  function mint(uint amount) external payable {
    _mint(amount);
  }

  function _mint(uint amount) internal {
    uint totalsupply = totalSupply();
    for (uint i = 1; i <= amount; ) {
      _safeMint(msg.sender, totalsupply + i);
      unchecked { ++i; }
    }
  }

  function withdraw(uint amount) public onlyOwner {
    require(amount < address(this).balance,
            "There is not so much amount in smart contract");
    (bool successOwner, ) = owner.call{value: amount}("");
    require(successOwner == true, "Transaction failed");
  }

  receive() external payable {}
}
```

2.2.3.- ERC-E7L

El estándar E7L, también conocido como ERC-721Linkable, introduce una innovación en el mundo de los NFTs al permitir que estos sean enlazados entre sí. La idea detrás de E7L

es que un NFT, al implementar este estándar, no es transferible hasta que se vincula a otro NFT de un smart contract diferente, conocido como '*token padre*'. Una vez vinculado, el E7L solo puede transferirse al actual propietario de dicho token padre.

Esta capacidad de vincular NFTs abre un abanico de posibilidades para aportar valor a los poseedores de tokens y crear nuevas experiencias en línea. Por ejemplo, E7L permite desarrollar experiencias donde las personas pueden obtener un E7L y vincularlo a su NFT favorito de un proyecto, aumentando así el valor percibido de este último y permitiendo que un NFT adquiera una reputación en cadena.

Otro uso interesante de E7L es en los lanzamientos de múltiples NFTs. Por ejemplo, se puede lanzar una sudadera física (o cualquier otra prenda) que tenga un PFP (Picture for Profile) vinculado, una versión descentralizada de la sudadera y un renderizado en 3D del NFT para mostrar en plataformas como Oncyber [28]. La propiedad de todos estos NFTs será automáticamente del actual propietario del token PBT (Physically Backed Token) de la sudadera. Si este token se transfiere, todos los NFTs complementarios pueden ser reclamados por el nuevo propietario.

Con la implementación de E7L, el valor de los coleccionables no se diluye en múltiples subcolecciones. Cada NFT tiene enlazados todos los demás tokens adjuntos a él. En el caso de nuestra colección, asociada a una marca de ropa, cuando se lanzan prendas exclusivas, se emiten con cada una de ellas su token E7L asociado, el cual puede sincronizarse con el token Mr. Crypto ERC721 del comprador de la prenda. Esto es parte fundamental del marketplace que se está desarrollando, para permitir la visualización y gestión de estos NFTs vinculados.

Desde el punto de vista técnico, ERC-721Linkable es una extensión de ERC-721. Las principales adiciones técnicas incluyen la propiedad '*parentContract*', la estructura '*LinkableToken*', y los siguientes métodos:

- '*_linkToken*'.- Inicializa un ID de token acuñado, vinculándolo a un ID de token del contrato padre. Si un token no está inicializado, no puede transferirse; una vez inicializado, solo puede transferirse usando la función '*syncToken*'.

- '*syncToken*'.- Transferirá el NFT al actual propietario del ID de token padre.

- '*_unlinkToken*'.- Revierte ese token a su estado inicial, esta operación está pensada para utilizarse solo con un sistema de control de acceso a eventos.

En el algoritmo 2.4 se muestra un ejemplo de cómo utilizar el token E7L.

Algoritmo 2.4: Ejemplo contrato E7L

```
contract DreamBig is ERC721Linkable {

  IERC721 public immutable parentContract;

  address public owner;
  string private BASE_URI;

  constructor(string memory _name, string memory _symbol,
              IERC721 _parentContract) ERC721Linkable(_name, _symbol) {
    owner = tx.origin;
    parentContract = _parentContract;
    BASE_URI = https://mrcrypto-sources.s3.eu-central-1.amazonaws.com/...";
  }

  function mint(uint256 tokenId) public {
    _safeMint(msg.sender, tokenId);
  }

  function _baseURI() internal view override returns (string memory) {
    return BASE_URI;
  }

  function tokenURI(uint256 tokenId)
           public view override returns (string memory) {
    _requireMinted(tokenId);
    return string(abi.encodePacked(_baseURI(), ".json"));
  }

  function linkToken(uint256 tokenId, uint256 parentTokenId, IERC721)
           external {
    _linkToken(tokenId, parentTokenId, parentContract);
  }

  function unlinkToken(uint256 tokenId) external {
      _unlinkToken(tokenId);
  }

}
```

Capítulo 3
Hipótesis de trabajo

La presente aplicación se construye sobre el framework Next.js, utilizando JavaScript y React para desarrollar componentes dinámicos, y TypeScript para asegurar la integridad del código, especialmente en interacciones con Web3. Node.js es el fundamento backend del proyecto, mientras que Prisma, como ORM, fortalece la conexión entre Node.js y TypeScript, facilitando la gestión de la base de datos. Dos indexadores, uno propio y otro externo, optimizan la recopilación de datos de la blockchain, necesario para el manejo eficiente de tokens E7L.

Antes de enumerar las tecnologías para el desarrollo de esta aplicación, se va a plantear un ejemplo sencillo para tratar de explicar cómo funciona todo en un contexto no tecnológico. Imaginemos una marca de zapatos, denominada "*ZapatosX*", que ha lanzado 1.000 pares de zapatos, cada uno de ellos único y exclusivo, estos modelos únicos serían como una colección de NFTs, cada uno distinto. Estos zapatos se pueden vender en una plataforma de segunda mano, como por ejemplo Wallapop, donde también se venden zapatos de otras marcas, y también otros tipos de productos. En este ejemplo, Wallapop sería el equivalente

a un marketplace generalista como OpenSea en el mundo de los NFTs, donde se venden todo tipo de NFTs, de diferentes creadores y colecciones, algunos originales y exclusivos, y otros que son copias de ediciones limitadas.

Ahora, supongamos que ZapatosX decide abrir su propia tienda online exclusiva para los productos de su marca, algo así como su propio "*Wallapop*", pero exclusivo (no generalista). Llevada esta idea al mundo de los NFTs, equivale a tener un marketplace exclusivo donde comercializar únicamente los NFTs de una colección propia. En este punto surge un problema: si alguien pone a la venta un par de zapatos tanto en Wallapop como en nuestra tienda exclusiva, ¿cómo podemos asegurarnos de que, una vez vendidos en una plataforma, no se vendan de nuevo en la otra? (recuérdese que cada par es único). La solución consiste en desarrollar unas herramientas, llamadas "*indexadores*" cuyo trabajo consiste en monitorear constantemente la blockchain y anotar las ventas que se produzcan en otras plataformas para garantizar que un producto ya vendido deje de ofrecerse en la plataforma propia. Este seguimiento es posible gracias a la tecnología blockchain, que permite tener un registro de cada transacción.

3.1.- INDEXADORES

Los indexadores son herramientas esenciales que permiten acceder y manipular eficientemente los datos almacenados en la blockchain. En aplicaciones donde se manejan grandes volúmenes de información, como es el caso de las transacciones en criptomonedas o la interacción con smart contracts, los indexadores juegan un papel importante en la organización, búsqueda y recuperación de datos.

Los indexadores funcionan escaneando bloques de la blockchain y extrayendo información relevante, que luego se almacena en una base de datos más convencional y de fácil acceso. Esto es especialmente útil en aplicaciones donde los usuarios necesitan consultar información histórica o realizar búsquedas específicas, tareas que serían ineficientes si se hicieran directamente sobre la blockchain. [30]

3.1.1.- Indexador Mr. Crypto

El indexador de Mr. Crypto [31] representa una herramienta indispensable en el desarrollo de nuestro marketplace. Este indexador, desarrollado por un miembro de la comunidad de Mr. Crypto, cumple una función esencial: indexar y organizar los datos de la colección de Mr. Crypto para que podamos consultarlos fácilmente. Permite acceder a la información necesaria de manera rápida, facilitando las búsquedas y el manejo de datos en el marketplace.

En el algoritmo 3.1, se muestra una consulta (query) simple en la que, proporcionando el ID de un NFT Mr. Crypto, se obtiene información relevante como la dirección de la wallet donde se encuentra, el número de Mr. Crypto presentes en dicha wallet, y los E7L vinculados a este (ID). La capacidad de recuperar los E7L vinculados es clave, ya que sin el uso de este indexador, sería extremadamente complejo, si no imposible, realizar esta consulta directamente a la blockchain. Esto se debe a que los 'padres' (en este caso, los tokens Mr. Crypto) no tienen conocimiento directo de sus 'hijos' (los E7L vinculados), lo que resalta la importancia de tener un sistema de indexación eficiente para manejar estas relaciones en nuestras consultas.

Algoritmo 3.1: Ejemplo query MrCryptoIndexer

```
query Test{
  mrCryptoById(tokenId: 8141){
    Owner{
      address,
      numberOfMrCryptos
    }
    E7LTokens{
      E7L{
        name
      }
    }
  }
}
```


En la figura 3.1 se muestra una interfaz web para interactuar con el indexador, donde, mediante una petición query (izquierda), se obtiene el resultado deseado en forma de objeto JSON (derecha).

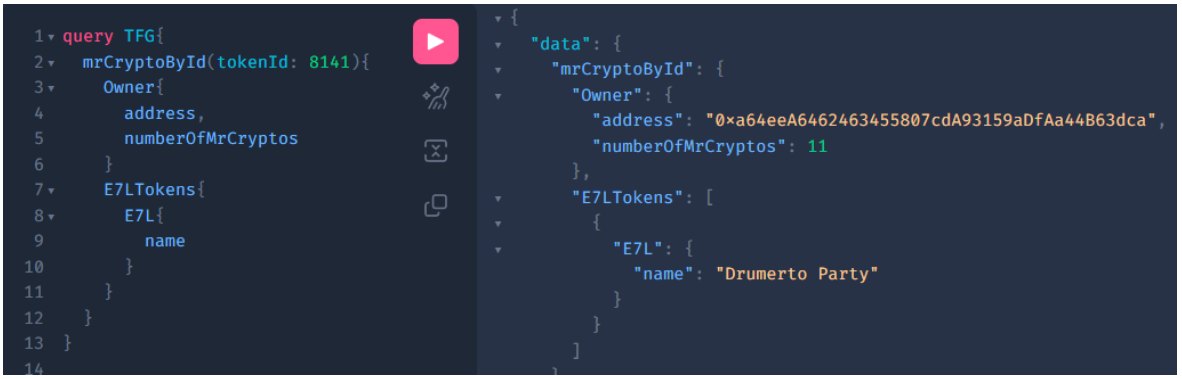

Figura 3.1: Petición query en el indexador de Mr. Crypto

Un aspecto de este indexador es su implementación con Docker. Esto significa que en una máquina local solo es necesario ejecutar el contenedor de Docker para que el indexador comience a procesar y organizar todos los datos relevantes. Esta funcionalidad es particularmente útil durante la fase de desarrollo y pruebas, ya que permite trabajar con los

datos de manera local sin tener que enviar múltiples peticiones a la versión del indexador alojada en la web. El uso de Docker aquí simplifica significativamente el proceso de desarrollo, asegurando que el entorno de trabajo sea consistente y que el indexador funcione de manera uniforme, tanto en máquinas locales como en el entorno de producción. Así, se pueden realizar pruebas y desarrollos, sin sobrecargar el servicio web con consultas excesivas mientras se afina y mejora el marketplace.

3.1.2.- Indexador Marketplace

El Indexador del Marketplace, desarrollado por nuestro equipo, es una herramienta dinámica que periódicamente, cada ciertos segundos, revisa la blockchain en busca de eventos específicos que se han definido previamente. Pero, ¿qué es exactamente un evento en este contexto? De manera sencilla, un evento en la blockchain es una señal o registro que se emite cuando ocurre una acción determinada, como una transacción o cualquier otra operación específica. Son como huellas digitales que dejan las operaciones en la blockchain, permitiendo rastrear actividades específicas. [32]

Una vez establecidos los eventos que se desea rastrear, el indexador se mantiene vigilante y, al detectar que estos eventos ocurren, captura y almacena la información relacionada en alguna base de datos. En el caso de nuestro marketplace se monitorizan eventos como los listados de NFTs (cuando un NFT se pone a la venta), las transacciones de compra-venta y las cancelaciones de listados (retirada de un NFT). Esta estrategia permite mantener un registro actualizado de las actividades del marketplace sin necesidad de consultar la blockchain constantemente.

El resultado es una base de datos accesible desde el frontend, donde se puede consultar la información relevante sobre el estado y actividad de los NFTs en el marketplace. Este enfoque reduce la carga en la blockchain y optimiza la experiencia del usuario al proporcionar acceso instantáneo a datos actualizados sobre listados, transacciones y cualquier cambio relevante en el estado de los NFTs.

3.2.- FRONTEND

Este apartado del proyecto se centra en las tecnologías para el desarrollo del frontend, una parte importante que define cómo los usuarios interactúan con la aplicación. El frontend es la cara visible del proyecto, donde se combina diseño, funcionalidad y experiencia de usuario. Aquí se discuten las tecnologías y herramientas empleadas para crear una interfaz atractiva y eficiente, asegurando que sea intuitiva y accesible para los usuarios.

3.2.1.- HTML, CSS y Tailwind: Diseño y Estilo

HTML, o Lenguaje de Marcas de Hipertexto (HyperText Markup Language), es el componente más básico de la Web. Actúa como el esqueleto de cualquier sitio web, proporcionando la estructura sobre la cual se construyen todos los demás aspectos visuales y funcionales. El término 'Hipertexto' en HTML se refiere a la capacidad del lenguaje para enlazar páginas web entre sí, lo cual es una característica de la experiencia en Internet. Estos enlaces permiten a los usuarios navegar de una página a otra, tanto dentro de un mismo sitio web como entre diferentes sitios.

HTML utiliza 'marcas' o 'etiquetas' para definir diferentes tipos de contenido, como texto, imágenes y enlaces. Estas etiquetas, como <head>, <title>, <body>, <header> y <footer>, permiten a los navegadores web interpretar y mostrar el contenido de la manera deseada. Cada elemento de HTML se distingue del texto regular mediante estas etiquetas, que se presentan con los signos '<' (menor) y '>' (mayor). Estas marcas no solo organizan el contenido de manera lógica y estructurada, sino que también ofrecen información sobre cada elemento de la página. [33]

En este proyecto, HTML ha sido la base sobre la que se construye toda la interfaz de usuario. Desde la organización del contenido hasta la inclusión de elementos multimedia y enlaces, HTML proporciona las herramientas necesarias para estructurar el sitio web de manera clara y efectiva. Su simplicidad y universalidad lo hacen indispensable en cualquier desarrollo web, permitiendo crear una base sólida y accesible para usuarios y navegadores por igual.

Las Hojas de Estilo en Cascada, o CSS, son el lenguaje que se utiliza para añadir estilo y diseño a una página HTML. Mientras que HTML proporciona la estructura, CSS se encarga de la presentación visual, permitiendo personalizar aspectos como colores, fuentes, disposición de elementos, entre muchas otras opciones. Esta capacidad para definir y controlar el aspecto visual es esencial para crear una experiencia de usuario atractiva y coherente. Con CSS, el código HTML básico se puede transformar en un diseño visualmente atractivo y profesional.

CSS es un lenguaje basado en reglas, donde cada regla define los estilos que se aplicarán a elementos específicos o grupos de elementos en la página web. Estas reglas se inician con un selector, que especifica a qué elemento o elementos de HTML se aplicará la regla. Seguido del selector, se encuentran las declaraciones, encerradas entre llaves '{}', que constan de pares de propiedades y valores. Cada par define una característica particular del elemento seleccionado, como su color, tamaño, margen, entre otros. [34]

En el presente trabajo, CSS se usa para dar vida y personalidad a la aplicación. Se ha utilizado CSS para garantizar que la interfaz de usuario, además de funcional, sea también visualmente atractiva y alineada con la identidad de la marca.

Tailwind es un framework CSS que se ha elegido por su enfoque en clases de utilidad y su flexibilidad para personalizar el diseño. Permite construir interfaces de forma rápida y con mayor control sobre el estilo, facilitando la creación de un diseño adaptativo (responsive). La instalación de Tailwind CSS es sencilla y directa. Utilizando el gestor de paquetes npm, común en proyectos basados en JavaScript, para instalar Tailwind como una dependencia de desarrollo se ejecuta el comando:

```
npm install -D tailwindcss
```

Luego, se genera el archivo de configuración con:

```
npx tailwindcss init
```

lo que crea el archivo '*tailwind.config.js*' en nuestro proyecto. Este archivo es donde se especifican los caminos a todos los archivos de plantillas, lo que permite a Tailwind aplicar los estilos. [35]

En nuestro código, la integración de Tailwind se puede observar de esta manera:

Algoritmo 3.2: Ejemplo código de Tailwind CSS

```
<button
  className={'btn-primary btn mx-auto flex items-center rounded-full
              border-0 bg-primary bg-opacity-80 bg-gradient-to-tr
              px-6 py-2 font-bold text-light hover:scale-105
              ${className}'}
  onClick={onClick}
  disabled={disabled}></button>
```

La clase *'btn'* indica una clase genérica para botones, *'mx-auto'* centra el botón horizontalmente, y las clases *'flex'* e *'items-center'* establecen un diseño flexible y centran el contenido verticalmente. Luego *'rounded-full'* hace que el botón tenga bordes completamente redondeados y *'border-0'* elimina cualquier borde. Para el texto, *'font-bold'* lo hace en negrita y *'text-light'* establece un color claro. Finalmente, *'hover:scale-105'* amplía ligeramente el botón al pasar el ratón sobre él, añadiendo un efecto interactivo sutil. Estas clases demuestran cómo Tailwind facilita un diseño rápido y modular.

Además de Tailwind CSS, se ha incorporado en el proyecto la herramienta DaisyUI. Es un plugin que se integra con Tailwind, proporcionando componentes de interfaz de usuario completamente personalizables y listos para usar.

Con DaisyUI, se han podido implementar rápidamente una amplia gama de elementos como botones y tarjetas, manteniendo la coherencia visual y aprovechando la facilidad de personalización que ofrece Tailwind. Este plugin ayuda a acelerar el proceso de diseño, manteniendo la flexibilidad y la eficiencia en el desarrollo de la interfaz de usuario. [36]

3.2.2.- Javascript y TypeScript: Programación Frontend

Javascript es un lenguaje de programación que aporta dinamismo y flexibilidad a las páginas web. Es el motor que impulsa las interacciones del usuario, las actualizaciones en tiempo real y la manipulación de elementos en la página, convirtiendo la interfaz de usuario en una experiencia interactiva y reactiva. A diferencia de otros lenguajes que requieren compilación previa, JavaScript es interpretado por el navegador en tiempo real. A pesar de ser un lenguaje interpretado, la mayoría de los navegadores modernos utilizan técnicas como la compilación en tiempo real para mejorar el rendimiento. Esto significa que el código JavaScript se compila en un formato binario mientras se ejecuta, permitiendo una ejecución rápida y eficiente.

El código JavaScript de lado del cliente se ejecuta en la máquina del usuario, cuando se carga una página web el código se descarga, se ejecuta y se muestran sus efectos o resultados directamente en el navegador. Esta ejecución del lado del cliente es lo que permite a JavaScript interactuar con el entorno del navegador y responder a las acciones del usuario en tiempo real.

En cuanto a la sincronía, JavaScript tradicionalmente ha sido un lenguaje de programación sincrónico y de un solo hilo. Sin embargo, con el tiempo, ha evolucionado para manejar operaciones asíncronas eficientemente, permitiendo que procesos más largos se ejecuten en el fondo, sin bloquear la interfaz de usuario. Esto se maneja a través de características como callbacks, promesas y async/await, que permiten a los desarrolladores escribir código que responde a eventos o datos que pueden no estar inmediatamente disponibles, estos mecanismos se han empleado a fondo para hacer llamadas a la blockchain. [37]

TypeScript es una extensión de JavaScript que aporta tipado estático al código, lo que resulta de mucha ayuda para detectar errores más rápidamente y a escribir un código más limpio y mantenible. Su uso permite desarrollar un frontend más robusto y con menos errores en tiempo de ejecución [38]. El uso de TypeScript en este tipo de proyectos no es casual, especialmente cuando se trabaja en el contexto de Web3. TypeScript añade una capa adicional de seguridad al código gracias a su sistema de tipos estáticos. Esta

característica es crucial en el desarrollo de aplicaciones Web3, donde la seguridad y la prevención de errores son de máxima importancia debido a la naturaleza financiera e irreversible de las transacciones en la blockchain. Al utilizar TypeScript se minimizan los posibles errores de codificación y se mejora la calidad general del código, lo que es esencial para construir aplicaciones confiables y robustas en el ecosistema Web3. [39]

3.2.3.- Next.js: Framework React

Next.js es un framework basado en React que ha servido para llevar nuestro desarrollo frontend al siguiente nivel. Ofrece características como la generación de páginas estáticas y el renderizado del lado del servidor, lo que mejora significativamente la velocidad de carga y el rendimiento de la aplicación. Next.js también facilita la implementación de rutas y la optimización SEO, lo que es útil para la visibilidad y accesibilidad de la web. Su integración con React ha permitido crear una interfaz de usuario rica y moderna, con componentes reutilizables y una arquitectura sólida que beneficia tanto el desarrollo como la experiencia del usuario final. [40]

Next.js también es el framework con mayor integración en el mundo de Web3. Ofrece una amplia gama de librerías y herramientas que facilitan la integración con tecnologías blockchain y smart contracts. Una de las librerías utilizadas en nuestro proyecto es 'wagmi', [41] específicamente diseñada para el desarrollo Web3 en React. 'wagmi' proporciona ganchos (hooks) y componentes para interactuar fácilmente con Ethereum, permitiendo conectar wallets, leer y escribir en la blockchain, y manejar estados de transacciones. La integración de 'wagmi' con Next.js ha permitido que nuestro proyecto aproveche las tecnologías blockchain de manera efectiva, mejorando la experiencia del usuario al interactuar con los smart contracts y la blockchain. Al elegir Next.js y 'wagmi', se logra una base sólida y flexible para el desarrollo frontend, ideal para enfrentar los requerimientos particulares del desarrollo en Web3.

3.3.- BACKEND

Este epígrafe se enfoca en el backend, el motor de la aplicación que maneja la lógica del servidor, la base de datos y la integración con otras aplicaciones y servicios. El backend es vital para el funcionamiento del producto, ya que gestiona las operaciones que no son visibles para el usuario pero son fundamentales para el rendimiento y la funcionalidad general del sistema. Aquí, se profundizará en las tecnologías clave utilizadas para construir un backend robusto y eficiente, destacando cómo cada una contribuye al funcionamiento global de la aplicación.

3.3.1.- Node.js: Eficiencia en el Servidor

Node.js es la piedra angular del backend del proyecto. Este entorno de ejecución de JavaScript en el servidor permite construir una aplicación escalable. Utilizar JavaScript tanto en el frontend como en el backend ofrece una uniformidad del lenguaje de programación, lo que facilita la integración y el mantenimiento del código.

Uno de los aspectos más destacados de Node.js es su rendimiento en operaciones de I/O (Input/Output o Entrada/Salida) no bloqueantes. Tradicionalmente, las operaciones de I/O en servidores son bloqueantes, lo que significa que el servidor debe esperar a que se complete una operación de I/O antes de poder continuar con la siguiente tarea. Esto puede crear cuellos de botella, especialmente en aplicaciones web con múltiples usuarios y solicitudes simultáneas, como puede ser la compra y venta de NFTs. Sin embargo, Node.js adopta un enfoque diferente al manejar estas operaciones de manera no bloqueante a través de la programación asíncrona. En lugar de esperar a que se complete una operación de I/O, Node.js inicia la operación y continúa con otras tareas. Cuando la operación de I/O se completa, Node.js recibe una notificación para procesar el resultado. Este modelo asíncrono y no bloqueante permite a Node.js manejar una gran cantidad de solicitudes de manera eficiente, mejorando el rendimiento y la capacidad de respuesta del servidor en situaciones con muchas operaciones de I/O. [42]

En el desarrollo de este proyecto con Node.js, se han utilizado dos gestores de paquetes fundamentales: npm (Node Package Manager) y pnpm (Performant Node Package Manager). npm es el gestor de paquetes por defecto para Node.js, que ha permitido instalar y administrar librerías y dependencias necesarias para el desarrollo. Por otro lado, pnpm es una alternativa más eficiente a npm que se ha utilizado para mejorar el rendimiento y la velocidad de la instalación de paquetes. pnpm se destaca por su eficiencia en el almacenamiento y el manejo de dependencias, utilizando un sistema de almacenamiento de enlace que reduce significativamente la duplicación de archivos en proyectos con múltiples paquetes. La combinación de npm y pnpm ha permitido aprovechar lo mejor de ambos sistemas, por un lado la amplia disponibilidad de paquetes de npm, y por otro el manejo más eficiente de recursos de pnpm, mejorando así la gestión de dependencias en nuestra aplicación. [43]

3.3.2.- Base de Datos con Prisma: Integración y Gestión

SQLite, Prisma, GraphQL, tRPC, y el concepto de endpoints son componentes clave en la arquitectura de aplicaciones modernas, cada uno desempeñando un papel específico que, en conjunto, facilita un flujo de trabajo para el manejo de datos y la comunicación entre el cliente y el servidor.

SQLite es una base de datos relacional que se caracteriza por su ligereza y autonomía, ya que se integra directamente en la aplicación. A diferencia de otros sistemas de gestión de bases de datos que requieren un servicio o proceso independiente, SQLite opera dentro de la misma aplicación, utilizando un archivo local para el almacenamiento de datos. Esto facilita una forma simple y directa de gestionar datos sin la complejidad de configurar un servidor de base de datos externo, haciéndolo especialmente útil para aplicaciones que necesitan una solución de almacenamiento de datos compacta y autónoma.[44]

Para la gestión de la base de datos se ha elegido Prisma, que se integra perfectamente con Node.js y TypeScript. Prisma actúa como un ORM (Object-Relational Mapping), proporcionando una capa de abstracción sobre la base de datos. Con Prisma, se puede interactuar con la base de datos mediante un API de alto nivel en lugar de escribir consultas SQL directas. Esto ayuda a la gestión de la base de datos y asegura que las interacciones con ella sean seguras. [45]

GraphQL, por su parte, es un lenguaje de consulta para APIs que permite a los clientes solicitar exactamente los datos que necesitan, lo que resulta en respuestas más específicas. En una aplicación, GraphQL define cómo se accede a los datos y qué parte de ellos se expone a los clientes. Funciona como una capa intermedia entre el cliente y el servidor, procesando las consultas y mutaciones enviadas por el cliente, comunicándose con la base de datos (a través de Prisma) para obtener los datos requeridos. [46]

tRPC agrega otra dimensión a esta arquitectura. Mientras que GraphQL se centra en proporcionar un punto de acceso flexible para varios tipos de datos y operaciones, tRPC permite definir procedimientos y consultas en TypeScript que se ejecutan en el servidor. tRPC aprovecha el sistema de tipos de TypeScript para garantizar la seguridad y corrección de las solicitudes y respuestas entre el cliente y el servidor. En lugar de usar endpoints de URL como en un API REST o GraphQL, tRPC permite invocar procedimientos del servidor directamente desde el cliente, como si fueran funciones locales, pero con la seguridad y validación que proporciona el tipado en TypeScript [47]. En el contexto de tRPC, los endpoints son estos procedimientos que proporciona el servidor. Aunque no son endpoints en el sentido tradicional de REST (URLs específicas), conceptualmente cumplen la misma función, es decir, son puntos de acceso de un API a través de los cuales el cliente interactúa con el servidor para realizar operaciones concretas. [48]

3.4.- SMART CONTRACT - SOLIDITY

Los smart contracts y Solidity son elementos esenciales en el mundo de las blockchain y especialmente en el desarrollo de aplicaciones descentralizadas. Los smart contracts son

programas *autoejecutables* almacenados en la blockchain, y Solidity es el lenguaje de programación más utilizado para escribirlos. En esta sección, se va a analizar cómo estos componentes forman la base de las funcionalidades avanzadas en Ethereum, y su importancia en el desarrollo de soluciones seguras, transparentes y descentralizadas.

3.4.1.- Solidity: Lenguaje de Contratos Inteligentes

Solidity es un lenguaje de programación orientado a objetos, creado específicamente para el desarrollo de smart contracts en la blockchain de Ethereum. Su diseño está influenciado por JavaScript y C, pero se distingue por implementar un tipado fuerte, lo que aumenta la seguridad en la definición de variables y argumentos. Esta característica es importante para asegurar la precisión y fiabilidad de los smart contracts. En este proyecto, Solidity es el lenguaje para desarrollar los smart contracts que manejan las transacciones e interacciones con los tokens, y otras lógicas de negocio.

Una de las capacidades destacadas de Solidity es su entorno de programación local, que permite a los desarrolladores escribir y probar smart contracts en sus propios equipos antes de desplegarlos en la red Ethereum. Una vez desplegados, estos contratos se alojan de manera descentralizada en toda la red de Ethereum, lo que garantiza que no estén bajo el control directo de ninguna entidad individual. Esta descentralización es clave para la naturaleza autónoma y segura de los smart contracts en Ethereum. [49]

Solidity no solo es aplicable a la red de Ethereum, sino que también es compatible con otras cadenas de bloques similares, como Polygon o BSC (Binance Smart Chain) [50]. Gracias a esta versatilidad, Solidity se ha convertido en un estándar en el desarrollo de dApps (aplicaciones descentralizadas) en varias plataformas blockchain.

El lenguaje es Turing Completo, lo que significa que puede representar cualquier algoritmo computacional. La Ethereum Virtual Machine (EVM) ejecuta el código Solidity, permitiendo una amplia gama de posibilidades en la programación de smart contracts. Esta capacidad completa de Turing, combinada con las características de la EVM, permite a Ethereum manejar estructuras de código complejas, como bucles y decisiones condicionales, que no están presentes en lenguajes como Bitcoin Script. [51]

Solidity puede ser tanto compilado como interpretado. En el modo de compilación, se genera un bytecode para ser ejecutado por la EVM, mientras que en el modo interpretado, un intérprete transforma las instrucciones en OP_CODES (códigos de operación) y bytecode dentro de un entorno de desarrollo. Además, Solidity ofrece funciones avanzadas para programación ensamblada, permitiendo el uso directo de OP_CODES de la EVM y la transformación inversa del bytecode en código Solidity. Esta característica es muy útil para la depuración y auditoría de smart contracts.[51]

Por último, resaltar que Solidity es un lenguaje orientado a objetos (POO, paradigma de modelado basado en el mundo real), lo que facilita la manipulación de elementos como los tokens.

3.4.2.- OP_CODES y Ethereum Virtual Machine (EVM)

Los OP_CODES (códigos de operación) y la Ethereum Virtual Machine (EVM) son conceptos en la arquitectura y funcionamiento de Ethereum, especialmente en la creación y ejecución de smart contracts. Los OP_CODES son instrucciones primitivas que permiten la programación de operaciones complejas dentro de blockchains como Ethereum. Actúan como las instrucciones básicas que la máquina virtual puede interpretar y ejecutar, lo que permite la implementación de lógicas de programación avanzadas en los smart contracts. Estos códigos de operación varían dependiendo de si se aplican al hardware, donde su forma está definida por la arquitectura del conjunto de instrucciones (ISA) del computador, o al software, donde los OP_CODES están diseñados para ser interpretados por una máquina virtual específica. [52]

La EVM es el componente clave que permite a Ethereum funcionar como un gran computador mundial descentralizado. La EVM es capaz de ejecutar el bytecode resultante de la compilación de smart contracts escritos en Solidity. El proceso comienza con la transformación del código Solidity en OP_CODES, que luego se convierte en bytecode. Este bytecode es ejecutado por la EVM, permitiendo realizar desde operaciones simples hasta las más complejas especificadas en los smart contracts. Esta capacidad de ejecución es lo que dota a Ethereum de su potencial para resolver casi cualquier problema computacional, todo dentro de su red descentralizada. [53]

Los smart contracts en Ethereum, escritos y desplegados en la blockchain, contienen este bytecode que la EVM puede ejecutar. Desde el momento en que estos contratos se incorporan a la blockchain, se activan y están listos para interactuar con usuarios o con otras aplicaciones descentralizadas (DApps). La belleza de este sistema es que cualquier persona con algo de ether, la criptomoneda de la red Ethereum, puede interactuar con estas DApps o crear sus propios smart contracts, aprovechando así el poder computacional que la EVM pone a disposición.

3.4.3.- Hardhat: Entorno de Desarrollo Ethereum

Hardhat es un entorno de desarrollo integral para Ethereum, que facilita la tarea de desarrollar, compilar, probar y desplegar smart contracts. Hardhat ha jugado un papel importante en este proyecto al proporcionar un marco de trabajo local para la ejecución de

smart contracts, lo que permite un desarrollo más rápido y cómodo. Una de las características destacadas de Hardhat es su capacidad para simular un entorno de blockchain completo, lo que permite realizar tests exhaustivos de los contratos antes de su despliegue en la red principal. Además, Hardhat se integra perfectamente con otras herramientas y librerías de Ethereum, como ethers.js, mejorando la capacidad para escribir, probar y desplegar código.

Un aspecto de Hardhat es su capacidad para ejecutar tests. Estas pruebas son importantes para asegurar que los smart contracts funcionen como se espera antes de desplegarlos en la red de Ethereum. Hardhat facilita la escritura y ejecución de tests automatizados. Además, Hardhat Network, su red de blockchain local, permite a los desarrolladores simular la ejecución de los contratos en un entorno controlado, simulando las condiciones reales de la red Ethereum.

Ejecutar y desplegar smart contracts con Hardhat es un proceso directo. El entorno de Hardhat proporciona una consola interactiva para interactuar con los contratos, y los scripts de despliegue se pueden personalizar para ajustarse a las necesidades del proyecto. La flexibilidad en la configuración y ejecución de tareas hace que Hardhat sea una herramienta adaptable a diferentes flujos de trabajo de desarrollo.[54]

Hardhat se integra sin problemas con librerías y frameworks populares en el ecosistema de Ethereum, como OpenZeppelin, que ofrece smart contracts seguros y probados que los desarrolladores pueden utilizar para construir aplicaciones en Ethereum. Esta integración permite a los equipos incorporar rápidamente prácticas y patrones de desarrollo, reduciendo el tiempo y el esfuerzo necesarios para crear smart contracts desde cero.[55]

En el Anexo II, se detalla el proceso de creación y configuración de un proyecto utilizando Hardhat incluyendo instalación del entorno, inicialización del proyecto, configuración de dependencias, compilación y ejecución de pruebas, y despliegue de smart contracts.

3.4.4.- Niveles de visibilidad en Solidity

En Solidity, los métodos y atributos de un smart contract pueden tener diferentes niveles de visibilidad, que definen cómo y desde dónde pueden ser accedidos. Estos niveles son: private, public, internal, y external [56]. A continuación se explican cada uno de ellos:

- Private: Los elementos marcados como private sólo son accesibles desde dentro del mismo smart contract. Ningún otro smart contract, incluso aquellos que heredan de éste, puede llamar a un método private. Es el nivel de visibilidad más restrictivo.

- Internal: Similar a private, pero con una flexibilidad adicional. Los elementos de tipo internal pueden ser accedidos no solo desde el smart contract que los define, sino también desde smart contracts que hereden de este. Es útil cuando se tiene una lógica que se necesita reutilizar en contratos derivados. Los atributos por defecto son internal.

- Public: Los métodos y atributos de tipo public son accesibles desde cualquier lugar, lo que significa que cualquier otro smart contract o wallet externa puede llamar a este método. Cuando no se especifica un nivel de visibilidad para una función en Solidity, por defecto es public.

- External: Los métodos external solo pueden ser llamados desde fuera del smart contract; no pueden ser llamados internamente, excepto a través de this.func(). Suelen ser más eficientes en términos de consumo de gas cuando se reciben grandes arrays de datos, ya que los datos se pasan directamente a la memoria y no se copian entre la memoria y el almacenamiento. Por otro lado, no se puede aplicar este nivel de visibilidad a los atributos (es solo para métodos).

3.4.5.- Smart Contracts: Ownable, ReentrancyGuard

En este proyecto se han integrado dos smart contracts a través de librerías, y apoyándonos en diagramas de clase para mostrar sus atributos y métodos se van a comentar para resaltar atributos de Solidity. Comenzaremos por el smart contract “Ownable”, ampliamente utilizado en muchos proyectos blockchain de la red de Ethereum. Este puede ser personalizado o importado directamente desde la librería OpenZeppelin.

El propósito principal de Ownable es gestionar la propiedad y los permisos administrativos dentro de un smart contract. Funciona asignando roles específicos, como el de propietario (Owner) y administrador (Admin). A través de este smart contract, se facilitan métodos para transferir la propiedad del smart contract y para añadir o eliminar administradores. Estos roles son importantes ya que permiten la ejecución restringida de ciertos métodos dentro del smart contract a través de modificadores.

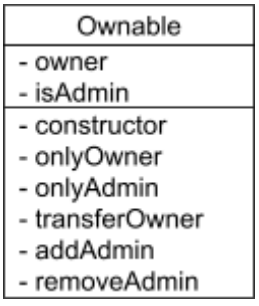

Figura 3.2: Diagrama de clase: Ownable.sol

Un modifier en Solidity es una forma de agregar requisitos previos a los métodos en un smart contract. Funciona como una cláusula contractual que debe cumplirse antes de que se pueda ejecutar el cuerpo del método. Por ejemplo, un modifier puede verificar si el llamante del método es el propietario del smart contract o si ha cumplido ciertas condiciones necesarias antes de permitirle proceder. Esto es especialmente útil para controlar el acceso y modificar el comportamiento de los métodos para mejorar la seguridad y gestionar el flujo de ejecución dentro de los smart contracts. Al incluir un modifier, se puede reutilizar el mismo código de verificación en múltiples funciones, lo que mejora la claridad y reduce el potencial de errores al implementar controles de acceso o validaciones repetitivas. [57]

Algoritmo 3.3: Ejemplo de código del modifier onlyOwner

```
1  modifier onlyOwner(){
2          require(msg.sender == owner, "Not owner");
3          _;
4  }
```

Por ejemplo, solo el propietario puede retirar el balance del smart contract utilizando el modifier onlyOwner. Aquí, el "balance" se refiere a la cantidad de tokens ERC20 que contiene el smart contract. Este método asegura que solo el propietario pueda controlar los activos significativos del smart contract.

Algoritmo 3.4: Ejemplo de código de uso del modifier onlyOwner

```
1 function withdraw(uint amount) public onlyOwner {
2     require(amount < address(this).balance,
3             "There is not so much amount in smart contract");
4
5     (bool successOwner, ) = owner.call{value: amount}("");
6     require(successOwner == true, "Transaction failed");
7 }
```

ReentrancyGuard de OpenZeppelin ayuda a prevenir ataques de reentrada en los smart contracts. Un ataque de reentrada puede ocurrir cuando un smart contract llama a otro smart contract externo y este último realiza una llamada de vuelta al smart contract original antes de que termine su ejecución. Esto puede resultar en efectos no deseados como extracciones no autorizadas de fondos.

ReentrancyGuard emplea un modifier llamado nonReentrant para bloquear llamadas recursivas a métodos críticos. Este modifier evita que una función se ejecute si ya está activa en la pila de llamadas, protegiendo contra modificaciones no deseadas del estado del smart contract durante una ejecución.

Por ejemplo, en operaciones de transferencia de tokens donde la seguridad es indispensable, aplicar nonReentrant previene manipulaciones y asegura que las transacciones se ejecuten de manera segura. El algoritmo 3.4 muestra un ejemplo de un método del marketplace que utiliza este modificador, ya que a la hora de comprar un NFT estamos ante una transferencia de tokens. [58]

Algoritmo 3.5: Ejemplo de uso del modifier nonReentrant en el método comprar

```
function buyItem(uint tokenId) external payable isListed(tokenId) nonReentrant{
    Listing memory list = listings[tokenId];
    uint collectionRoyaltyFee = (listing.price * MarketplaceFee) / 10000;

    if (listing.paytoken == address(0)) {
      if (msg.value < list.price) {
         revert NotEnoughAmount(tokenId, list.price);
      }
      Recipient.transfer(collectionRoyaltyFee);
      payable(list.seller).transfer(list.price - collectionRoyaltyFee);
    } else {
      // ERC20 token payment
      IERC20(list.paytoken).transferFrom(msg.sender, Recipient,
                                         collectionRoyaltyFee);

      IERC20(list.paytoken).transferFrom(msg.sender, list.seller,
                                         list.price - collectionRoyaltyFee);
    }

    delete (listings[tokenId]);

    //Trasfer nft
    IERC721(nftAddress).safeTransferFrom(list.seller, msg.sender, tokenId);

    emit ItemBought(msg.sender, tokenId, list.paytoken, list.price,
                    collectionRoyaltyFee);
}
```

3.5.- PAQUETES DE SOFTWARE ADICIONALES

En el desarrollo de aplicaciones descentralizadas, el uso de librerías y paquetes específicos juega un papel al aportar seguridad y funcionalidades avanzadas. A continuación, se detallan algunas de las librerías y paquetes de software adicionales utilizados en el proyecto y el propósito que cumplen:

- ethers.js.- Es una librería ligera y completa que permite la interacción con la blockchain de Ethereum y sus smart contracts. Facilita el envío de transacciones, la interacción con contratos inteligentes, y la gestión de wallets [59].

- Reportes gas.- Este paquete ofrece una forma detallada de visualizar y analizar el consumo de "*gas*", que es la tarifa que hay que pagar al realizar transacciones y operaciones con contratos inteligentes. Es útil para optimizar y reducir los costos de gas en el despliegue y ejecución de funciones [60].

- Coverage.- Herramienta utilizada para evaluar la cobertura de pruebas en los smart contracts. Ayuda a identificar secciones del código que no han sido testeadas [61].

- Dotenv.- Es un módulo que carga variables de entorno desde un archivo .env a process.env. Permite gestionar de forma conveniente las configuraciones, como claves API y direcciones de nodos, sin exponerlas directamente en el código [62].

- NextAuth.- Es una solución completa para la autenticación en aplicaciones Next.js, ayudando a la integración de inicio de sesión con proveedores externos, autenticación por email y más. Permite una gestión segura de sesiones, adaptándose fácilmente a diferentes flujos de autenticación y protegiendo rutas específicas [63].

3.6.- OTRAS HERRAMIENTAS

Estas son algunas herramientas adicionales que también se han utilizado para el desarrollo del proyecto. Aunque no están directamente relacionadas con la programación blockchain o el desarrollo frontend y backend, sí son de ayuda a la hora de desarrollar y para facilitar el trabajo en equipo.

3.6.1.- Github

Github es una plataforma para la colaboración y el control de versiones. Permite trabajar simultáneamente en diferentes partes del código sin riesgo de conflictos o pérdida de información. Mediante el uso de repositorios en Github, es posible mantener un historial claro de todos los cambios realizados, facilitando la revisión del código y la colaboración entre los miembros del equipo. Además, Github sirve como un espacio para almacenar de manera segura el código del proyecto, acceder a él desde cualquier lugar y compartirlo con otros colaboradores o interesados. [64]

3.6.2.- Visual Studio Code

Visual Studio Code (VS Code) es el editor de código que se ha empleado para codificar. Su interfaz amigable, la variedad de extensiones disponibles y su rendimiento lo convierten en

una herramienta ideal para desarrolladores. VS Code ofrece funcionalidades como resaltado de sintaxis, autocompletado de código, depuración y una terminal integrada, lo que agiliza significativamente nuestro flujo de trabajo. Además, su capacidad para integrarse con otros servicios como Github y su soporte para múltiples lenguajes y frameworks hacen de VS Code una herramienta versátil y potente para cualquier desarrollo. [65]

3.6.3.- Docker

Docker es una plataforma de contenedores que facilita la implementación, despliegue y ejecución de aplicaciones mediante el uso de contenedores. Los contenedores permiten empaquetar una aplicación con todas sus dependencias y bibliotecas en un solo paquete denominado 'contenedor', lo que asegura que la aplicación funcione de manera uniforme en cualquier entorno de computación. Esto es especialmente útil en el desarrollo de software, ya que elimina el clásico problema de '*en mi máquina sí funciona*', garantizando que si la aplicación se ejecuta correctamente en un contenedor Docker, se ejecutará de la misma manera en cualquier otro sistema que soporte Docker. [66]

En el contexto del presente proyecto, se utiliza Docker para arrancar en local el indexador de MrCrypto. Al utilizar Docker, podemos asegurar que el indexador se ejecute bajo las mismas condiciones cada vez, facilitando un desarrollo más coherente y predecible.

3.6.4.- Turbo Repo

Turbo Repo es una herramienta de gestión de mono repositorios que sirve para mantener la coherencia y eficiencia en todo el proyecto. Se ha utilizado para gestionar y coordinar el desarrollo de varios componentes interconectados.

La principal ventaja de usar Turbo Repo es su capacidad para optimizar los tiempos de compilación y testing. Al almacenar en caché los resultados de compilaciones y tests anteriores, se reduce significativamente el tiempo necesario para realizar estas operaciones recurrentes. Esto ha sido especialmente útil cuando trabajamos con múltiples subproyectos que comparten dependencias o necesitan ser probados en conjunto.

Otro aspecto ha sido la gestión de dependencias entre los diferentes paquetes del proyecto. Con Turbo Repo, se ha mantenido un control sobre cómo los cambios en un componente afectan a los demás, evitando así problemas de incompatibilidad o errores inesperados.

Además, Turbo Repo se ha integrado sin problemas con otras herramientas que se están utilizando, como Git para control de versiones y Node.js para el backend. Esta integración

ha permitido automatizar procesos que abarcan varios paquetes del proyecto, como despliegues y actualizaciones. [67]

3.6.5.- T3-Stack

T3 Stack no es una herramienta en sí misma, sino un enfoque o una arquitectura para construir aplicaciones web. Se trata de una pila de tecnologías recomendadas que trabajan bien juntas para crear aplicaciones full-stack. Las herramientas principales de este strack son Next.js y TypeScript, aunque Tailwind CSS también se suele incluir. Adicionalmente, para la parte backend también se incluyen tRPC, Prisma y NextAuth.js. Vemos su instalación en el Anexo I, puede parecer que son muchas piezas, pero no son todas imprescindibles siempre, sino que, según el proyecto que se quiera desarrollar pueden unas y descartar otras, también es posible añadir otros componentes, además de los ya citados, si fuera necesario. [68]

Capítulo 4
Metodología y resultados

El presente proyecto, tiene como objetivo la creación de una aplicación diseñada para satisfacer los requisitos de una empresa con la cual se han establecido colaboraciones continuas. A través de reuniones sucesivas se ha definido el conjunto de características de la aplicación para cumplir con las expectativas de la empresa.

4.1.- PLANIFICACIÓN DEL PROYECTO

En el desarrollo del marketplace de NFTs que se presenta en esta memoria, se han usado metodologías ágiles para guiar todo el proceso de desarrollo, depuración y mejora de la aplicación. Estas metodologías, que ponen el foco en la colaboración, la flexibilidad y la mejora continua, se han convertido en un punto importante para enfrentar los desafíos del desarrollo de software complejo. Al dividir el trabajo en sprints cortos e iterativos, el equipo ha sido capaz de adaptarse a cambios rápidamente, integrar retroalimentación de

manera efectiva y, en última instancia, mejora la productividad. Este enfoque iterativo ha permitido, no solo responder a las necesidades cambiantes de la empresa colaboradora, sino también evolucionar el producto de manera constante, asegurando que cada entrega sucesiva aporte valor. [69]

4.1.1.- Ciclo de vida

SCRUM se ha destacado como la metodología ágil elegida, dada su estructura y eficacia en el manejo de proyectos de desarrollo que requieren flexibilidad y entregas rápidas. Basada en la iteración, el ajuste continuo y una comunicación fluida, SCRUM permite organizar el trabajo en sprints definidos, durante los cuales se planifican, implementan y prueban funcionalidades clave del proyecto que se desea desarrollar. [70]

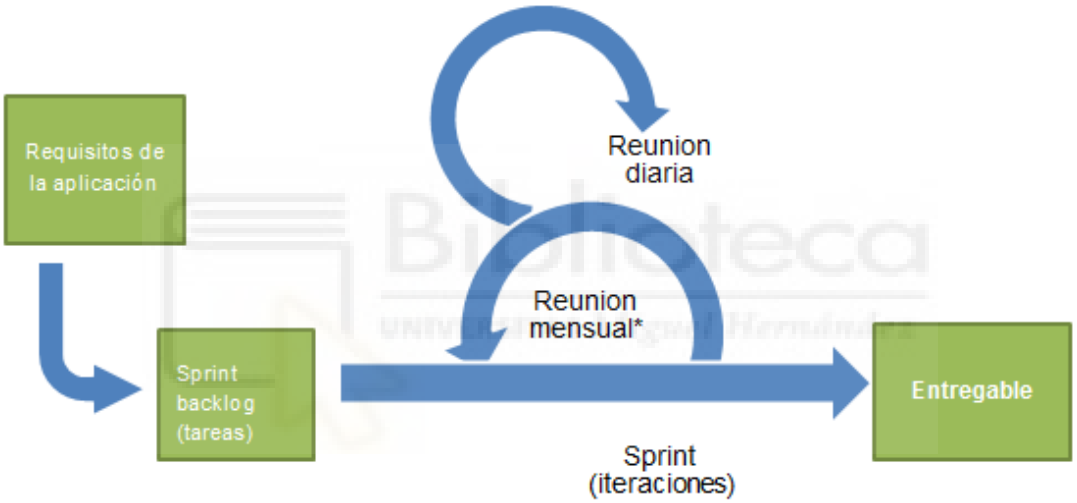

Figura 4.1: El proceso Scrum (wikipedia)

Esta metodología se basa en la cooperación de tres tipos de participantes esenciales para su correcta aplicación, cada uno de ellos desempeña el sigui:

- El Scrum Master: Es quien facilita el proceso y asegura que el equipo siga las prácticas de Scrum, es el responsable de resolver cualquier problema que surja para que el equipo pueda trabajar eficientemente.

- El Product Owner: Representa los intereses y necesidades del cliente, determina las características específicas y prioriza el trabajo. Hace de puente entre los desarrolladores y los usuarios finales de la aplicación.

- El Equipo de Desarrollo: Son los encargados de ejecutar las tareas y producir los entregables. Se espera que sepan autoorganizarse y coordinarse para implementar cada tarea de la mejor forma posible.

Los sprints, que son ciclos de desarrollo de duración fija, comienzan con una reunión de planificación donde se define el objetivo y se especifican las tareas del "*backlog*" del producto, es decir, se detalla una lista de funcionalidades priorizadas que se espera que tenga el producto final. Durante cada sprint, se realizan reuniones diarias (dailys) para evaluar el progreso y detectar posibles obstáculos. Al final de cada sprint, se lleva a cabo una revisión para demostrar el trabajo realizado a las partes interesadas y una retrospectiva que permite al equipo reflexionar sobre su desempeño y buscar formas de mejorar.

La adopción de SCRUM en este proyecto de desarrollo de un marketplace de NFTs no solo ha mejorado la coordinación del equipo sino que también ha garantizado una alineación constante con los objetivos del negocio y las expectativas del cliente. Esta metodología ha probado ser muy útil para mantener un ritmo de trabajo sostenible mientras se manejan las complejidades del desarrollo de aplicaciones descentralizadas, permitiéndonos realizar un producto innovador. [71]

En nuestro caso, la implementación de SCRUM se adaptó a las dinámicas de un equipo de desarrollo dedicado a la creación de un marketplace de NFTs. Además de encargarnos de las diversas tareas de desarrollo, asumimos un rol activo en la planificación y priorización del trabajo, evaluando el esfuerzo y tiempo necesario para cada tarea. Contábamos con un Product Owner asignado por la empresa colaboradora, quien se convirtió en un guía para la orientación del proyecto. A través de reuniones semanales durante los primeros dos meses, este Product Owner proporcionó información sobre la visión del producto, incluyendo aspectos del diseño, la estructura y las tecnologías aplicables.

4.1.2.- Diagrama de Gantt

El diagrama de Gantt es una herramienta en la gestión de proyectos que proporciona una representación visual del tiempo asignado o transcurrido para las tareas programadas. Se compone de dos secciones principales: una lista de tareas y un cronograma con barras que representan la duración de cada tarea. Este tipo de diagrama permite a los gestores de proyectos visualizar las fechas de inicio y finalización, los hitos importantes, las dependencias entre tareas, y las asignaciones de personal, facilitando así la planificación y seguimiento del proyecto. [72]

Para el presente proyecto, iniciado en junio de 2023 y aún en curso, se ha implementado un diagrama de Gantt para visualizar y gestionar el flujo de trabajo. Esto permite reflejar las tareas realizadas, su duración y los puntos clave del desarrollo. Hay que hacer ver que, si bien el proyecto comenzó en junio del año pasado, tanto yo, como el equipo de desarrollo hemos estado trabajando en otros proyectos diferentes, además de que este proyecto en concreto ha experimentado "parones" a causa del cliente. Adicionalmente hay que sumarle

el tiempo dedicado al estudio y búsqueda de información. El diagrama de Gantt que aquí se muestra no refleja dichas interrupciones, simplemente muestra la priorización y duración en semanas de cada una de las tareas realizadas.

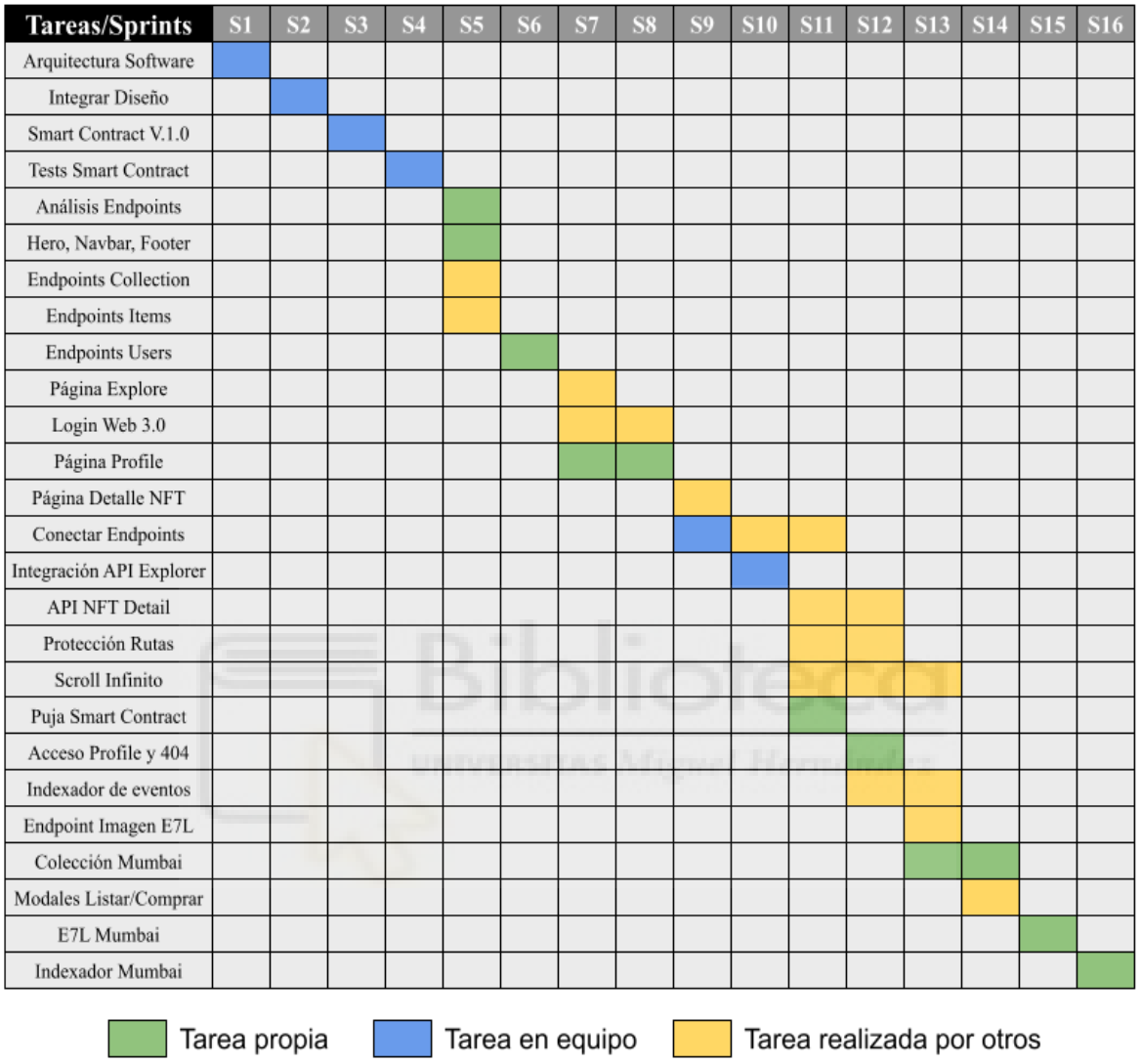

Figura 4.2: Planificación de tareas/sprints del proyecto por semanas

4.2.- TAREAS

Desde la fase inicial, donde se planificó el proyecto y se tomaron decisiones sobre tecnologías y diseño, hasta las etapas posteriores de ajuste y refinamiento, se ha permitido reorganizar semanalmente las tareas asignadas a los miembros del equipo, adaptándolas a los cambios y nuevos requerimientos. Aquí se explica de qué tratan y se entrará en más detalle en aquellas tareas en las que he tenido una participación directa. En definitiva, las tareas realizadas han sido las siguientes:

4.2.1.- Arquitectura Software

Lo primero que se hizo fue pensar en cómo queríamos que fuera la aplicación. Se eligió usar Create-T3 porque facilitaba mucho el trabajo inicial, ya que viene con Next.js para el frontend y Node.js para el backend. Además, se decidió que Hardhat sería la herramienta para manejar los smart contracts, porque proporciona muchas facilidades para desarrollar y testear. También, se eligieron una serie de librerías que ayudarían a lo largo del desarrollo, que ya se han explicado en capítulos anteriores.

4.2.2.- Integrar Diseño

Después de tener clara la arquitectura inicializamos todo para asegurarnos de que las herramientas funcionaban como esperábamos. Lanzamos la aplicación en local para ver que todo estaba en orden y empezamos a jugar con el smart contract de prueba que proporciona Hardhat. Esto nos ayudó a familiarizarnos con el entorno de desarrollo y sirvió de entrenamiento para lo que vendría después.

4.2.3.- Smart Contract V.1.0

El siguiente paso fue desarrollar nuestro propio smart contract. Queríamos que tuviera la mayoría de las funcionalidades necesarias para nuestra aplicación, así que dedicamos bastante tiempo a asegurarnos de que todo estuviera correcto. Este smart contract es el corazón de nuestra dApp y lo que permite que funcione de manera descentralizada (se puede leer el código en el explorador de bloques de la red de pruebas: https://sepolia.etherscan.io/address/0x8f7fc45ea5eff1572f4e3c307faf2706c671a377#code).

4.2.4.- Test Smart Contract

Por supuesto, no es nada recomendable lanzar un smart contract en una blockchain sin antes probarlo a fondo. Se escribieron tests para cada una de las opciones, probando que todo funcionara correctamente (ver código de pruebas en el Anexo III).

4.2.5.- Hero, Navbar, Footer

Esta tarea se centra en el diseño inicial de la página: el componente Hero, que actúa como la puerta de entrada visual a nuestra dApp, capturando la atención del usuario desde el

primer momento. El Navbar, básico para la navegación dentro de la aplicación, incluyendo el acceso al login integrado con Web3 para gestionar la autenticación de los usuarios. Y, por supuesto, no podía faltar el Footer, manteniendo las tradiciones del diseño web con información útil y enlaces de contacto.

Figura 4.3: Navbar

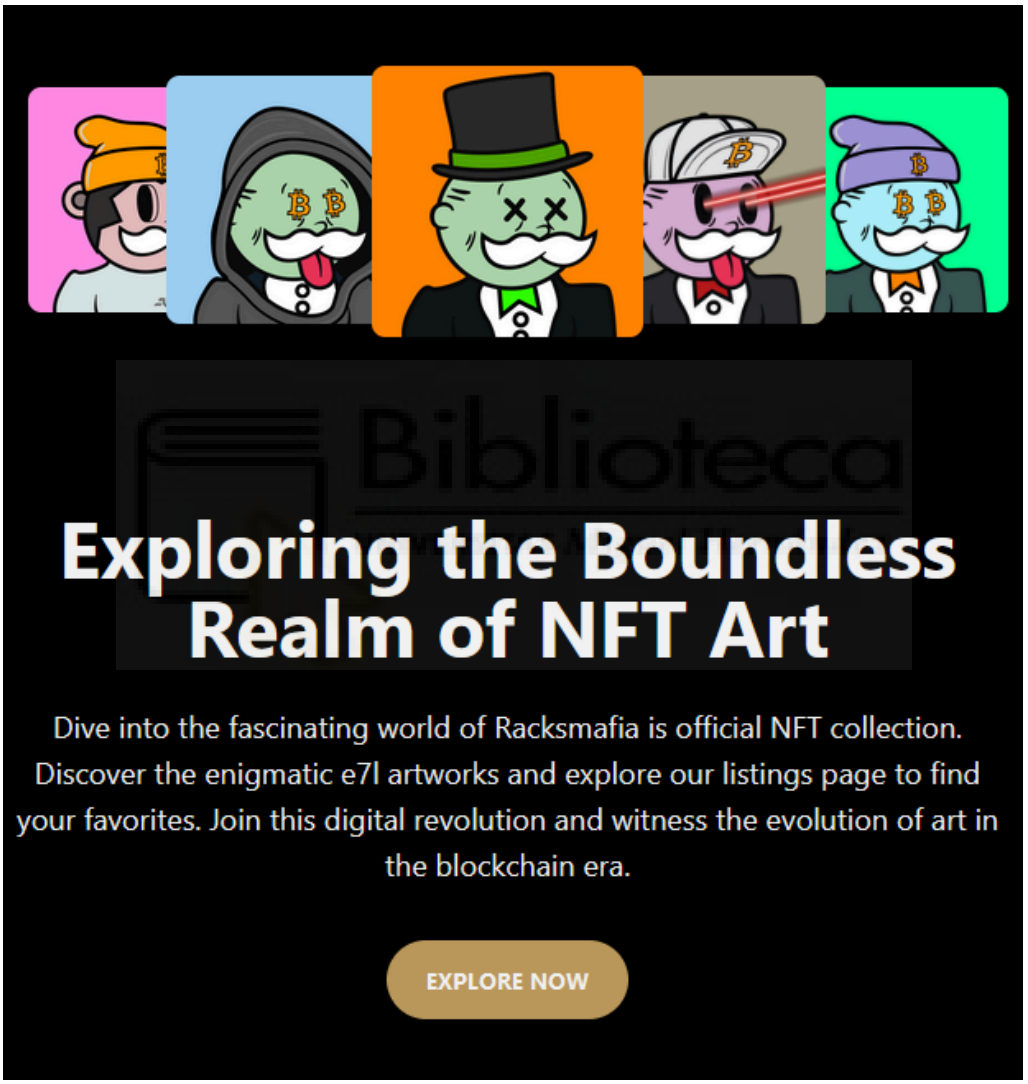

Figura 4.4: Página Hero
(accesible en: https://marketplace-mr-crypto.vercel.app)

Figura 4.5: Footer

4.2.6.- Análisis Endpoints

El desarrollo de la aplicación implica un detallado análisis para determinar los endpoints necesarios para su correcto funcionamiento. Este análisis ayuda a esclarecer las funcionalidades claves y cómo deberían ser accesibles desde el frontend. Tras identificar estos endpoints, se procedió a organizarlos en grupos según su propósito y funcionalidad, estos endpoints se pueden clasificar en tres tipos:

- Collections.- Contiene todos los endpoints para interactuar con las colecciones de NFTs. ERC721 y E7L recopilan información de la colección principal o de varias colecciones respectivamente, estos últimos, además proporcionan la dirección del padre de los NFTs E7L. Por su parte ERC721 NFT y E7L NFT se centran en obtener información específica de un NFT concreto dentro de la colección, el primero para los NFTs propios, y el segundo para NFTs otras colecciones.

- Ítems.- Estos endpoints facilitan la interacción directa con las funcionalidades del smart contract de nuestro marketplace, enfocándose especialmente en el seguimiento de los eventos en la blockchain:
  - Item Listed: Este endpoint capta los eventos emitidos cuando un NFT se lista en el marketplace. Es clave para actualizar en tiempo real la disponibilidad de NFTs para los usuarios.
  - Item Canceled: De forma similar, este endpoint recoge los eventos cuando el dueño de un NFT decide cancelar su listado en el marketplace, permitiendo así mantener actualizada la oferta de NFTs disponibles.
  - Items Bought: Este endpoint se encarga de captar los eventos generados al comprar un NFT. Se usa para reflejar las transacciones completadas.
  - Bid on NFT: Sirve para pujar por un NFT, captando los eventos relacionados con las pujas en la blockchain. Esta función introduce una dinámica de subasta, enriqueciendo la experiencia del usuario en el marketplace.

Tabla 4.1: Relación de endpoints de la aplicación

Collections		Items	Users
● Get ERC721 - Title - Description - Image - Address ● Get ERC721 NFT - ID - Name - Image - Collection - Price	● Get E7L - Title - Description - Image - Address collection parent ● Get E7L NFT - ID - Name - Image - ID Parent	● Item listed ● Item canceled ● Item Bought ● Item bid up	● Get users - Lista de usuarios - Address - Name - Number NFT - Info NFT

- Users.- Este es el único endpoint dedicado específicamente a los usuarios registrados de la plataforma, obteniendo información sobre los mismos. La acción de registro se completa cuando un usuario inicia sesión con su wallet, momento en el cual se captura su dirección y los NFTs que posee. Además, desde la página de perfil del usuario, se ofrecen opciones adicionales para interactuar con la plataforma (ver sección 4.2.9, Página Profile).

4.2.7.- Página Explore

En el desarrollo del proyecto, uno de los componentes es la página "Explorer", donde se despliegan los 10,000 NFTs de la colección MrCrypto, presentándolos en una interfaz de usuario amigable y accesible. Para optimizar la experiencia del usuario y manejar eficientemente la carga de datos, se implementa un sistema de scroll infinito, que carga gradualmente los NFTs en tarjetas visuales a medida que el usuario avanza por la página, evitando así la sobrecarga de la web con demasiadas peticiones simultáneas. En esta tarea, nos centramos exclusivamente en desarrollar la arquitectura y en implementar un diseño básico para esta sección de la aplicación.

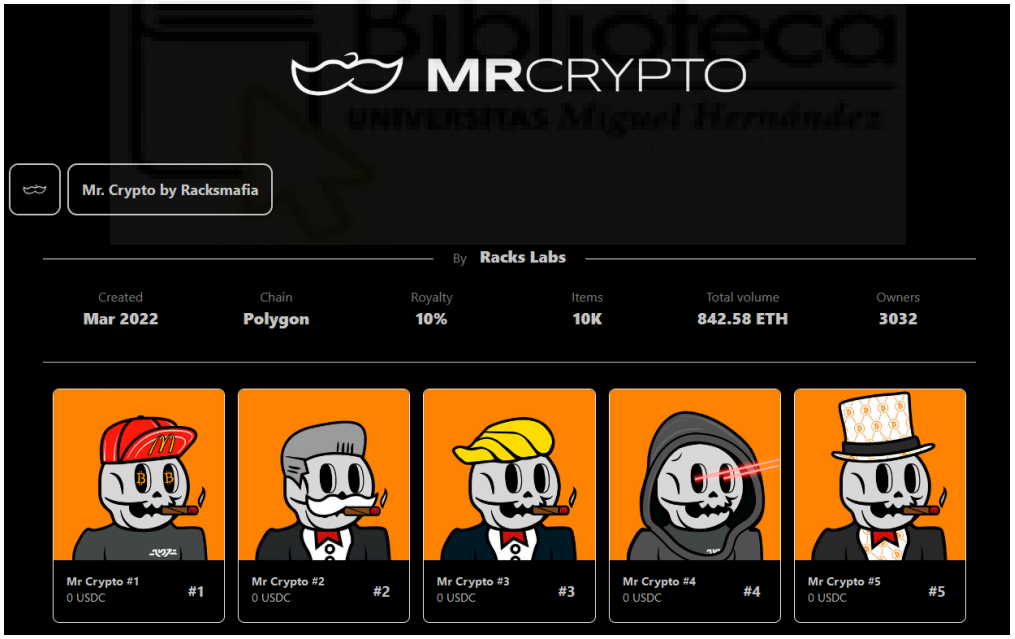

Figura 4.6: Página Explore

4.2.8.- Login Web 3.0

Para habilitar el inicio de sesión Web 3.0 en la aplicación hay que integrar la billetera MetaMask con Next.js utilizando NextAuth.js, adoptando el estándar "*Sign-In with Ethereum*" (SIWE). Este estándar proporciona una forma estructurada y segura de

autenticar usuarios mediante sus wallets de Ethereum. A continuación, se describen los pasos específicos del proceso:

- Solicitud de Conexión: Cuando el usuario elige iniciar sesión a través de nuestra interfaz, se le solicita que conecte su wallet MetaMask a la aplicación. Esto se inicia con una petición que incluye una solicitud de firma para un mensaje SIWE. Este mensaje, preparado según dicho estándar, incluye un nonce generado de manera segura para evitar ataques de repetición y asegurar la singularidad de cada sesión de autenticación.

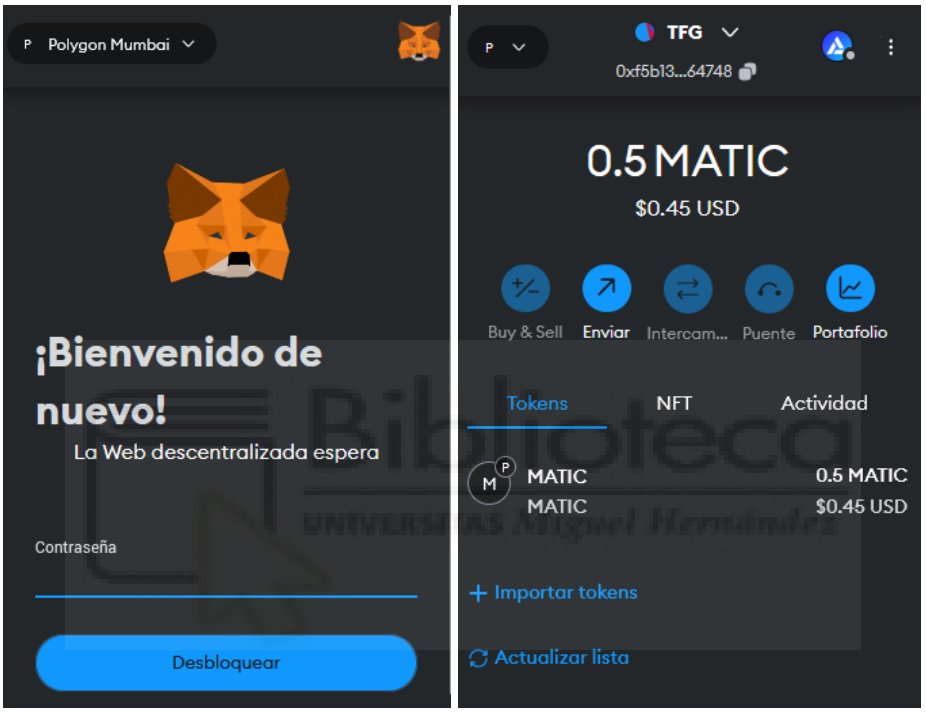

Figura 4.7: Extensión Metamask y Criptomonedas para operar

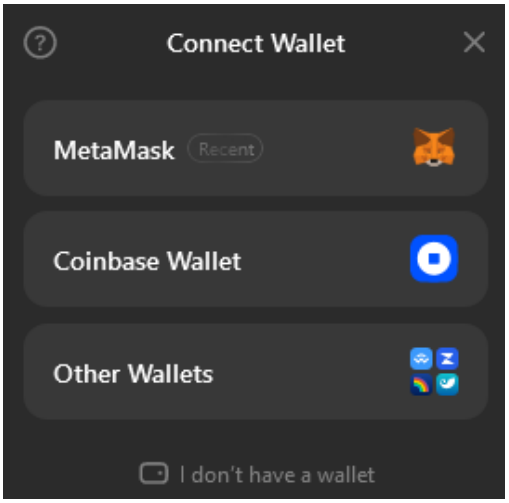

Figura 4.8: Conectar wallet

- Firma del Mensaje SIWE: El usuario firma el mensaje SIWE usando MetaMask. Este mensaje contiene información clara y estructurada, como la dirección del usuario, el dominio de la aplicación, y el nonce. Al firmar este mensaje, el usuario no solo verifica la posesión de su dirección de billetera sino que también da su consentimiento explícito para iniciar sesión en la aplicación, mejorando la transparencia y seguridad del proceso.

- Verificación y Autenticación: La firma digital y los detalles del mensaje SIWE son enviados al servidor. Utilizando las funcionalidades proporcionadas por NextAuth.js y el estándar SIWE, el servidor verifica la autenticidad de la firma y la correspondencia con la dirección de la billetera del usuario. Este paso asegura que la firma es válida y corresponde al mensaje SIWE original, autenticando efectivamente al usuario.

- Establecimiento de la Sesión: Tras la verificación exitosa de la firma SIWE, NextAuth.js procede a crear una sesión de usuario. Esto significa que la aplicación puede reconocer al usuario en sus visitas posteriores, ofreciendo una experiencia fluida y segura sin la necesidad de múltiples autenticaciones.

4.2.9.- Página Profile

La página de perfil es accesible únicamente tras iniciar sesión con una wallet, específicamente MetaMask.

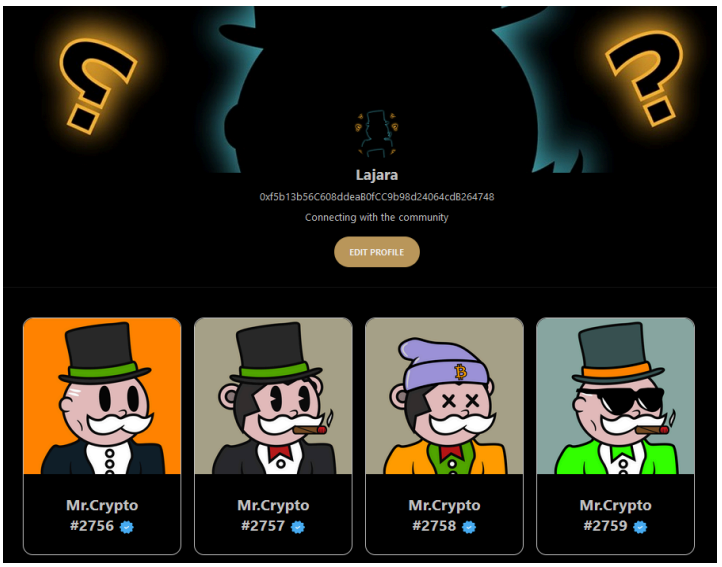

Figura 4.9: Página Profile

Esta página se compone de varios elementos dinámicos que muestran información relevante del usuario, como su imagen de perfil, banner, nombre, dirección de la wallet,

descripción y los NFTs asociados a esa dirección. Inicialmente, algunos de estos datos se presentan con valores predeterminados, excepto la dirección y los NFTs, que se actualizan automáticamente al conectar la wallet.

Esta sección facilita la gestión de los NFTs del usuario, permitiendo un acceso directo para visualizar, listar o cancelar la venta, mejorando así la interacción con la plataforma.

4.2.10.- Página Detalle NFT

Al hacer clic en un NFT, ya sea desde la página de exploración o desde el perfil del usuario, se redirige a una página detallada dedicada a ese NFT específico. Aquí, se ofrecen detalles como los E7L vinculados y la dirección del propietario. Dependiendo de si el NFT pertenece al usuario logueado, se presentan opciones para listar o cancelar su venta, de lo contrario, para NFTs de otros usuarios, se muestran las opciones de compra o puja.

En esta tarea, nos centramos exclusivamente en desarrollar la arquitectura y en implementar un diseño básico para esta sección de la aplicación.

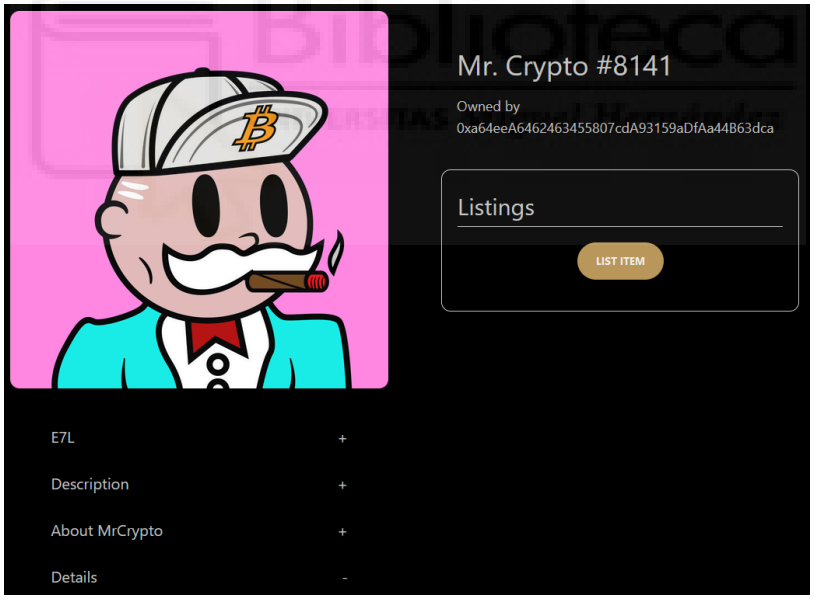

Figura 4.10: Página Detalles

4.2.11.- Conectar Endpoints

La tarea de conectar los endpoints con el frontend consiste en hacer que la página web pueda interactuar con los datos gestionados en el backend y en la blockchain, para lo cual se utiliza una serie de herramientas que permiten enviar y recibir información de manera eficaz. Por ejemplo, para mostrar información sobre nuestra colección de NFTs, como el

número de Owners, también llamados holders (direcciones que poseen uno o más NFTs de nuestra colección), primero se llama al endpoint específico que proporciona estos datos usando una función específica del frontend, la cual esta preparada para hacer consultas a nuestro servidor mediante tRPC. Una llamada para hacer un petición de datos de nuestra colección sería así:

```
api.queriesE7L.getCollectionInfo.useQuery().data;
```

Esta petición, una vez resuelta, devuelve los datos solicitados (como la cantidad de holders), que pueden ser almacenados en una variable, en este caso '*collectionInfo*'. Para mostrar estos datos en la página web basta con referenciar esta variable en nuestro componente. Por ejemplo, para mostrar la cantidad de holders de nuestra colección de NFTs, insertamos la información directamente utilizando la instrucción:

```
{collectionInfo?.collectionInfo.holders}
```

Este método permite mantener una interfaz de usuario reactiva y actualizada con los últimos datos disponibles, sin necesidad de realizar cargas de página completas o peticiones innecesarias.

Figura 4.11: Información de la colección

4.2.12.- Integración API Explorer

Tarea que consiste en desarrollar un sistema dinámico para consultar y mostrar los tokens no fungibles (NFTs) MrCrypto en la sección "Explorer" de nuestra aplicación descentralizada (dApp). Esta funcionalidad se realiza mediante un controlador de consulta que se comunica con nuestro indexador a través de GraphQL. Utilizando el endpoint específico "*getMrCryptoTokens*", este controlador permite recuperar información de cada NFT, incluyendo su imagen y el ID único. Esta información se muestra en la sección "Explorer", garantizando que los usuarios siempre tengan acceso a los datos más actualizados y relevantes sobre los NFTs.

Con el objetivo de presentar estos NFTs de una manera ordenada y coherente, se implementa una estrategia de ordenación inicial basada en el ID numérico de los tokens. Este enfoque permite mostrar los NFTs en un orden lógico y fácil de navegar.

4.2.13.- Integración API NFT Detalle

Esta tarea aborda cómo conectar la información específica de cada NFT con la página de detalle en nuestra aplicación. Se utilizan varios endpoints para recoger datos como la dirección a la que pertenece el NFT, la dirección de la colección, el ID del NFT, los E7L asociados, descripción, imagen, y si está listado, también su precio. Esta información se recoge y se presenta en una página dedicada, donde los usuarios pueden obtener una vista completa de cada NFT. Este enfoque permite mostrar los detalles del NFT, así como ofrecer a los usuarios la opción de comprar o pujar si el NFT les interesa.

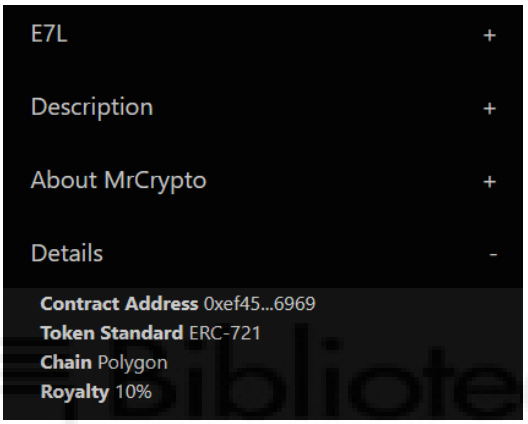

Figura 4.12: Detalles del Mr. Crypto

4.2.14.- Protección Rutas

La protección de rutas garantiza que solo los usuarios autenticados y autorizados puedan acceder a ciertas páginas de nuestra aplicación Web 3.0. Al integrar SIWE (Sign-In with Ethereum) y NextAuth.js, se establece un sistema de control de acceso basado en autenticación blockchain, aprovechando la identidad digital única de los usuarios a través de sus wallets de Ethereum.

Cuando un usuario intenta acceder a una ruta protegida en la aplicación, primero se verifica su sesión mediante NextAuth.js, esto implica comprobar si el usuario ha realizado previamente un proceso de inicio de sesión SIWE exitoso, estableciendo así una sesión válida. La sesión contiene información clave, como la dirección Ethereum del usuario que se utilizó durante la autenticación. Dado que el estado de la wallet de un usuario puede cambiar (por ejemplo, cambiando de cuenta o desconectándose), SIWE y NextAuth.js cierra automáticamente la sesión del usuario ante tales eventos. Esto garantiza que la sesión de un usuario refleje con precisión su estado actual de conexión y autorización, reforzando la seguridad y la relevancia de las sesiones de usuario.

4.2.15.- Scroll Infinito

La aplicación web implementa una técnica conocida como "scroll infinito" para explorar la colección extensa de NFTs, permitiendo a los usuarios descubrir gradualmente miles de NFTs únicos. Esta estrategia evita sobrecargar el dispositivo del usuario, asegurando una experiencia fluida y continua sin necesidad de cargar toda la colección de una vez.

Al avanzar en la exploración, la aplicación carga automáticamente más NFTs cuando el usuario se acerca al final de los que ya ha visualizado gracias al "observador de intersecciones". Este mecanismo actúa discretamente, anticipándose a las necesidades del usuario y presentando nuevos NFTs sin interrupciones.

4.2.16.- Puja Smart Contract

Se introduce un nuevo método en el smart contract para mejorar el proceso de compra-venta en la plataforma, permitiendo las pujas por NFTs. Esto significa que un usuario puede ofrecer un precio por un NFT específico. No solo eso, el propietario del NFT recibirá varias ofertas y podrá seleccionar la que más le convenga. Una vez aceptada una oferta, el smart contract automatiza el intercambio transfiriendo el NFT al comprador y el pago al vendedor, garantizando una transacción segura, directa y sin intermediarios.

Algoritmo 4.1: Código del método para pujar por un ERC721 (Mr. Crypto)

```
function makeBidUp(address paytoken,uint tokenId,uint bidPrice) external
                    payable isPaytokenValid(paytoken) {
    Bid memory bid = bids[tokenId];
    if (bid.bidder != address(0) && bid.paytoken != paytoken)
      revert PaytokenNotValid();
    if (bidPrice <= bid.priceBid)
      revert LowerThanCurrentBid(bid.priceBid);

    if (paytoken == address(0)) {
      if (msg.value <= bid.priceBid || msg.value != bidPrice)
        revert NotEnoughAmount(tokenId, msg.value);
      if (bid.bidder != address(0))
        payable(bid.bidder).transfer(bid.priceBid);
    } else {
      IERC20 token = IERC20(paytoken);
      if (token.allowance(msg.sender, address(this)) < bidPrice)
        revert PaymentNotApprovedForMarketplace();
    }

    bids[tokenId].bidder = msg.sender;
    bids[tokenId].paytoken = paytoken;
    bids[tokenId].priceBid = bidPrice;
    emit ItemBetted(msg.sender, tokenId, paytoken, bidPrice);
}
```

4.2.17.- Acceso Página Perfil y Página 404

Con la implementación de un sistema de autenticación que aprovecha las wallets Ethereum se facilita el acceso a los perfiles personales mediante botones específicos en el Navbar. Esta mejora permite a los usuarios acceder a su perfil de manera directa y sencilla tras la autenticación, ofreciendo un camino intuitivo hacia la gestión personal de sus NFTs y la personalización de su espacio en la plataforma.

Además, se ha desarrollado una página de error 404 personalizada para mejorar la experiencia del usuario en casos donde se accede a enlaces rotos o páginas no existentes. Esta página ofrece una interfaz amigable y guía a los usuarios de vuelta a secciones activas de la plataforma, asegurando que incluso los errores contribuyan a una navegación fluida y agradable. La combinación de estas dos tareas mejora significativamente la interacción del usuario con la aplicación, haciéndola más accesible y menos propensa a fricciones innecesarias.

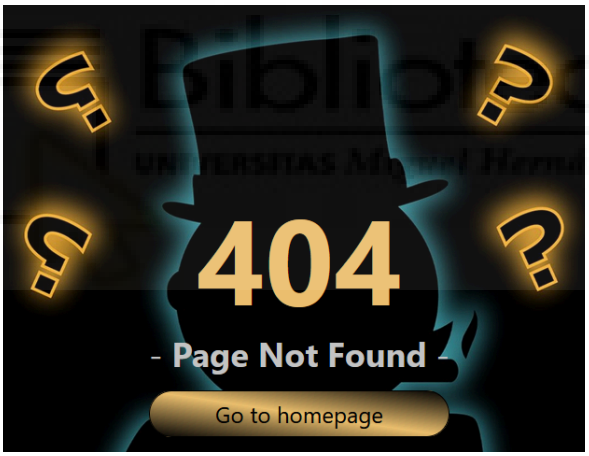

Figura 4.13: Página Error 404

4.2.18.- Indexador de Eventos

Esta fase del proyecto se enfoca en el desarrollo del indexador de eventos para el marketplace. Este indexador es una pieza que rastrea y registra los eventos emitidos por las transacciones en la blockchain, específicamente aquellos relacionados con la compra y venta de NFTs. Al capturar estos eventos, el indexador permite mantener actualizada la aplicación con la información más reciente sobre las actividades del mercado. Es una herramienta que asegura que los usuarios tengan acceso a datos en tiempo real sobre la disponibilidad y el movimiento de los NFTs dentro del marketplace.

4.2.19.- Endpoint Imagen E7L

Se ha creado un endpoint específico destinado a recuperar la imagen representativa de los E7L vinculados al NFT principal. Este desarrollo permite mostrar de manera dinámica y atractiva los E7L asociados a cada NFT, brindando a los usuarios una vista más completa y visual de las conexiones entre las piezas de la colección. La implementación de este endpoint facilita la integración de estas imágenes en el apartado de detalles del NFT, donde los usuarios pueden explorar de manera más profunda la relación entre los NFTs y sus E7L correspondientes.

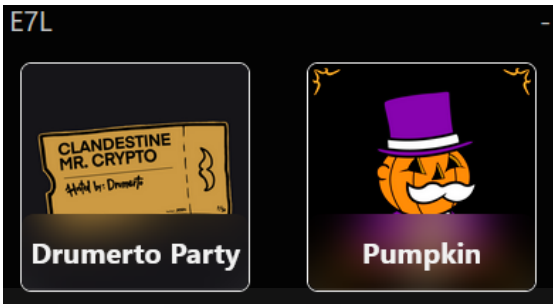

Figura 4.14: E7L Linkados

4.2.20.- Colección Mr. Crypto en la red de Mumbai

Se crea una versión de prueba de la colección de NFTs en la testnet de Ethereum para realizar testeos. Este paso incluye el despliegue de un smart contract y el desarrollo de un interfaz frontend sencillo que permite a los usuarios conectar su wallet y mintear NFTs.

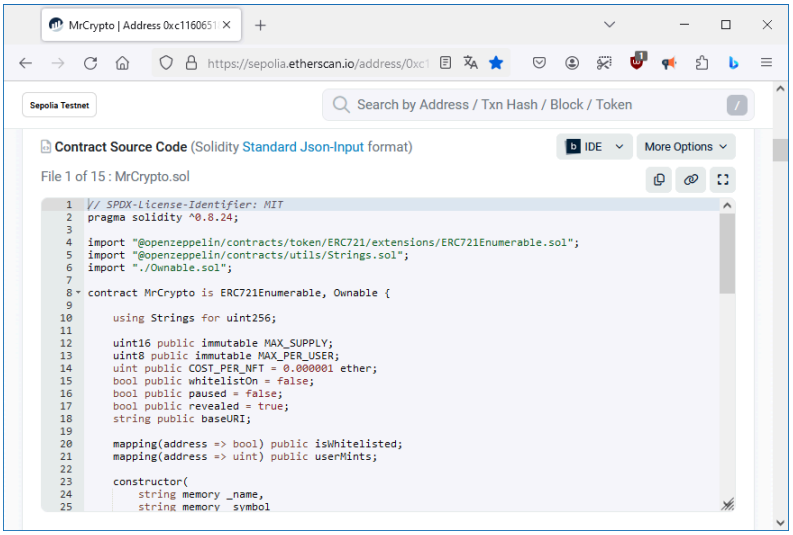

Figura 4.15: Contrato Mr. Crypto desplegado en la red de pruebas

URL: https://sepolia.etherscan.io/address/0xc11606518dbf7d2891797f01b7cb03149cfca899#code

La finalidad de esta acción es facilitar pruebas más intensas y realistas en el entorno del marketplace sin incurrir en los costos asociados con la mainnet. Gracias a esta colección de pruebas, se pueden simular transacciones, listar, comprar, y vender NFTs.

Figura 4.16: Mint NFT ERC721
(accesible en: https://collection-erc721-mrcrypto.vercel.app/)

4.2.21.- Modales Listar/Comprar

En esta parte del proyecto, se implementa un modal que adapta su contenido según el contexto en el que se usa. Si un usuario está viendo uno de sus propios NFTs en la página de detalles, el modal mostrará un botón que permite listar ese NFT para la venta. Por otro lado, si el NFT pertenece a otro usuario, el modal cambiará para mostrar un botón de compra. Esta funcionalidad facilita la interacción directa y específica con los NFTs, dependiendo de la propiedad y el estado del mismo.

4.2.22.- E7L Mumbai

Con el fin de ampliar las pruebas y experimentar con diferentes escenarios, se desplegaron varias colecciones E7L en la testnet de Mumbai, aprovechando el mismo frontend desarrollado para la colección principal. Este proceso incluye la implementación de la funcionalidad para enlazar y sincronizar NFTs E7L con sus contratos padres, permitiendo una mayor flexibilidad y experimentación con las dinámicas de la colección y las interacciones dentro del marketplace.

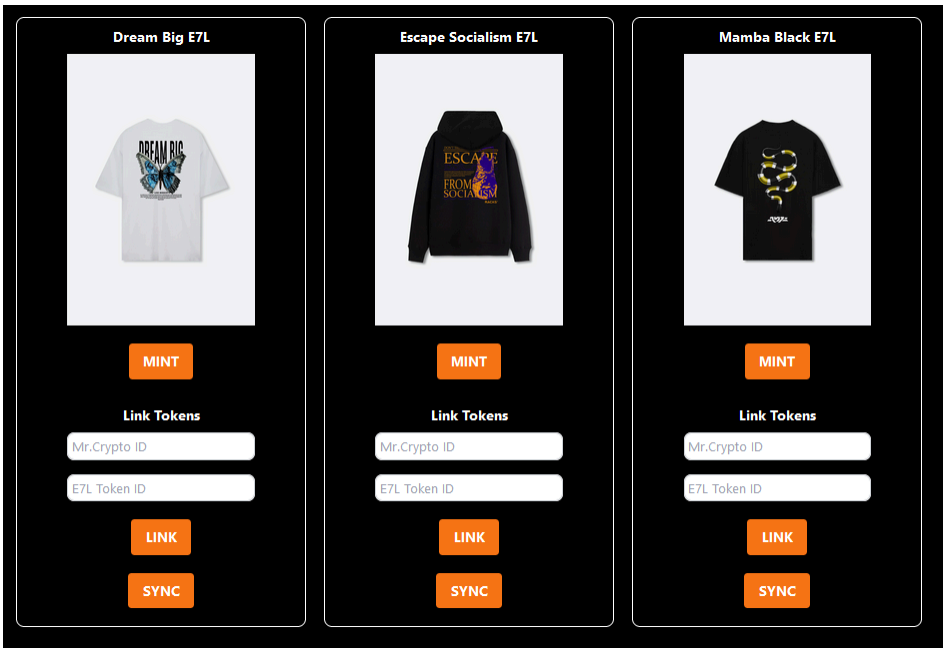

Figura 4.17: Mint NFT E7L

4.2.23.- Indexador Mumbai

Dado que el indexador original de MrCrypto está diseñado para operar en la mainnet, para realizar pruebas intensivas sin incurrir en costos de gas, se creó una versión modificada del indexador ajustada para trabajar con la testnet de Mumbai. Este ajuste permitió recopilar datos de las nuevas colecciones desplegadas en Mumbai, garantizando que nuestras pruebas en el entorno de desarrollo fueran lo más precisas y relevantes posible, reflejando de manera efectiva cómo se comportaría el indexador en un entorno de producción.

4.3.- CAPTURA DE REQUISITOS

La captura de requisitos en el desarrollo de una aplicación descentralizada (DApp) presenta un escenario ligeramente distinto al de las aplicaciones centralizadas tradicionales. En un entorno descentralizado, operando sobre tecnología blockchain, las funcionalidades y operaciones son dirigidas de manera autónoma a través de smart contracts. Esto significa que las transacciones y acciones ejecutadas por los usuarios no requieren la intervención directa de administradores, modificando la gestión y el control habituales.

En lugar del rol tradicional de administrador, que supervisa y gestiona las operaciones en aplicaciones centralizadas, en una DApp, esta responsabilidad recae en los smart contracts. Estos smart contracts están programados para ejecutar automáticamente las acciones correspondientes una vez que se cumplen ciertas condiciones.

En este contexto, la jerarquía de actores y los casos de uso se definen en función de las interacciones permitidas por los smart contracts. Los usuarios interactúan directamente con estos contratos para realizar transacciones, acceder a servicios o participar en decisiones comunitarias, sin necesidad de un intermediario.

4.3.1.- Tipos de usuario, actores

Para operar con esta aplicación se definen tres roles o tipos de usuarios:

- Usuario no identificado: Puede explorar listados de NFT sin operar con ellos.
- Usuario identificado: Habilitado para comprar, listar NFTs.
- Propietario del Smart contracts: Posee acceso a ciertas funcionalidades del smart contract.

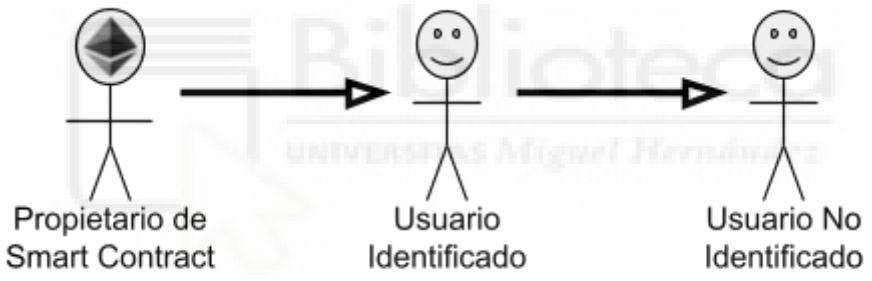

Figura 4.18: Usuarios del sistema

Tabla 4.2.- Rol "Usuario no identificado"

Actor	Usuario no identificado
Descripción	El usuario no identificado tiene un acceso limitado y puede explorar ciertas partes de la aplicación.
Casos de uso	CU1, CU2, CU3

Tabla 4.3.- Rol "Usuario identificado"

Actor	Usuario identificado
Descripción	El usuario identificado tiene acceso completo a todas las funcionalidades de la aplicación. Para ello, debe conectar la wallet para acceder a todas las funcionalidades de la plataforma.
Casos de uso	CU4, CU5, CU6, CU7, CU8, CU9, CU10, CU11, CU12, CU13

Tabla 4.4.- Rol “Propietario del Smart Contract”

Actor	Propietario del Smart Contract
Descripción	El propietario del Smart Contract puede ejecutar ciertas funciones a las cuales solo él tiene acceso.
Casos de uso	CU14, CU15, CU16

Los diagramas de casos de uso para cada usuario quedan como se muestra en las figuras 4.18, 4.19 y 4.20. En el Anexo IV se encuentran las tablas que describen los detalles de cada caso de uso.

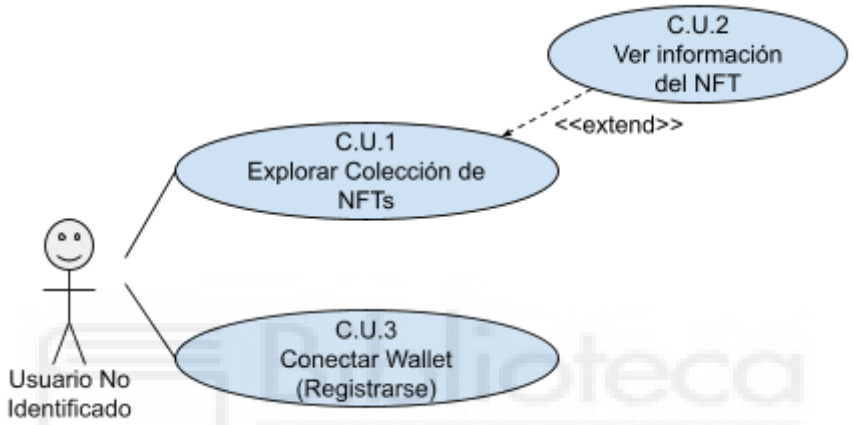

Figura 4.19: Casos de Uso: Usuario no identificado

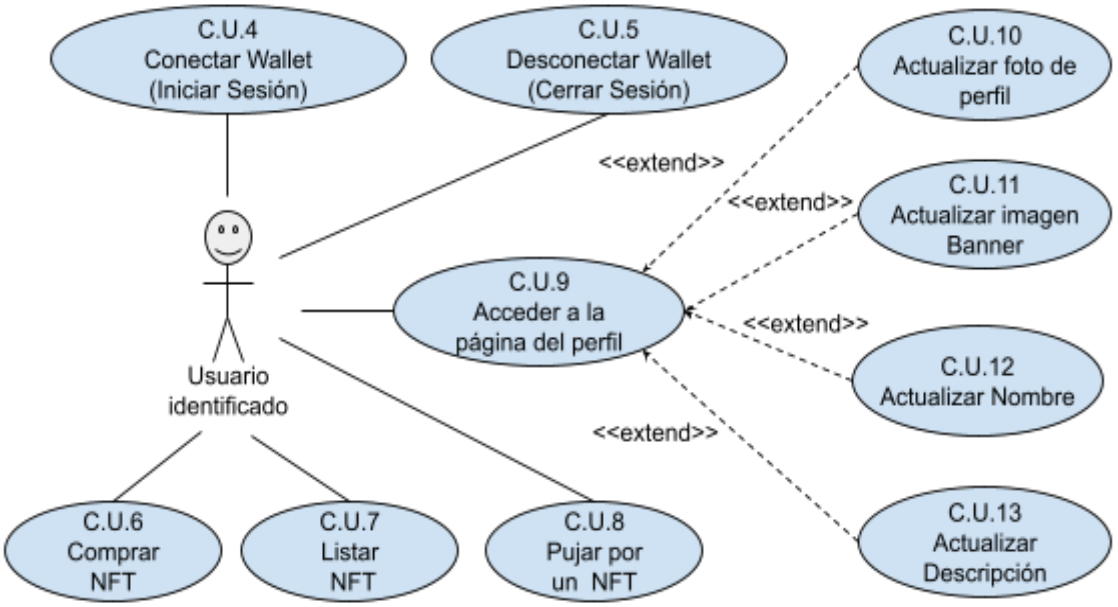

Figura 4.20: Casos de Uso: Usuario identificado

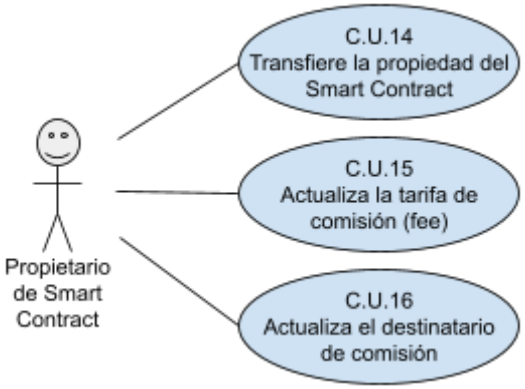

Figura 4.21: Casos de Uso: Propietario del smart contract

4.4.- DISEÑO

4.4.1.- La interfaz de usuario (mockups)

En el desarrollo de nuestra aplicación, el diseño de la interfaz de usuario se realizó inicialmente a través de bocetos creados con Excalidraw, una herramienta de dibujo online fácil de usar. Estos mockups sirvieron para esbozar la estructura visual básica y las características que esperábamos incluir en nuestro marketplace de NFTs. En los bocetos, buscamos reflejar una interfaz limpia y organizada, típica de los marketplaces modernos con la que estuviera familiarizado cualquier usuario de este tipo de aplicaciones, y donde los elementos visuales no sobrecarguen la interfaz y ofrezcan una navegación intuitiva y accesible. Se diseñaron varios croquis para diferentes secciones de la aplicación, mostrando desde la página de inicio hasta las vistas detalladas de cada NFT, con el objetivo de proporcionar una idea clara del aspecto final que se deseaba alcanzar.

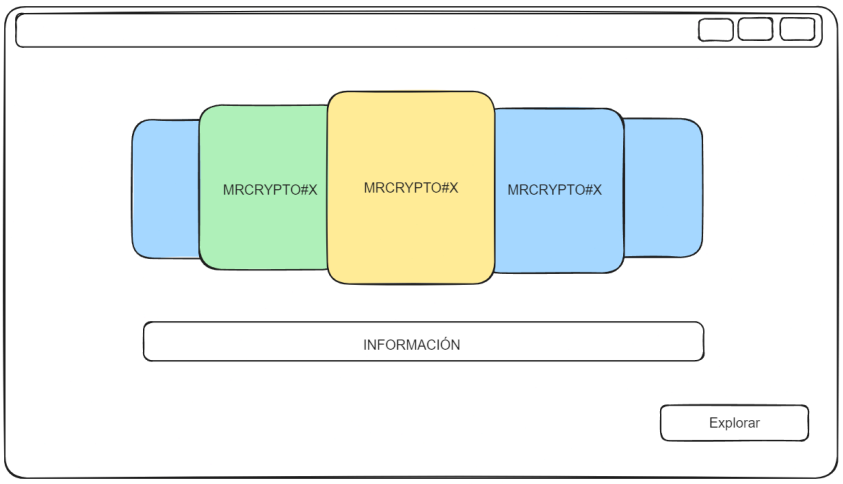

Figura 4.22: Página Hero

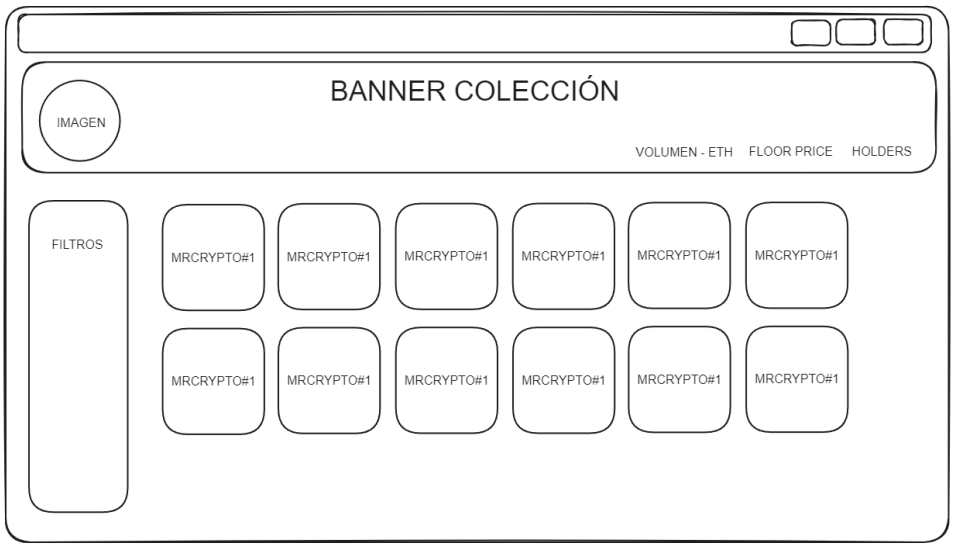

Figura 4.23: Página Explore

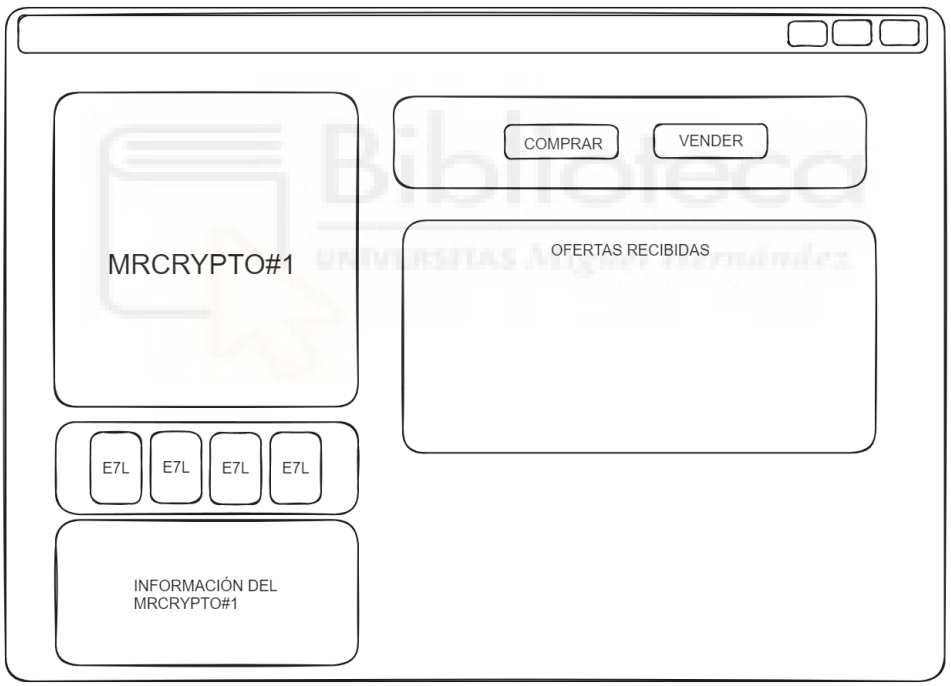

Figura 4.24: Página detalle del NFT

4.4.2.- La base de datos

El diagrama Entidad/Relación es una herramienta para diseñar y entender cómo funciona una base de datos, especialmente en aplicaciones como nuestro marketplace de NFTs. Este diagrama nos ayuda a ver claramente qué datos necesitamos guardar (como información de usuarios y NFTs) y cómo estos datos se relacionan entre sí.

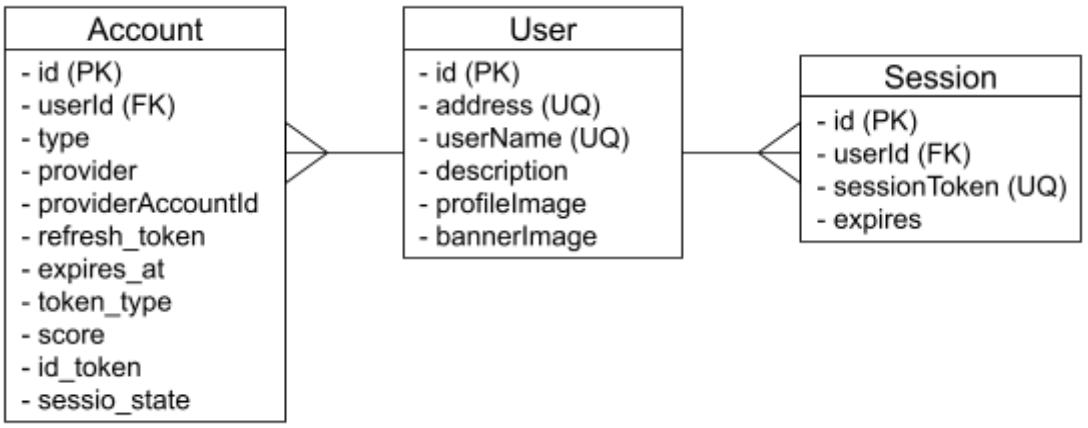

Figura 4.25: Tablas User, Account, Session

En la base de datos hay varias tablas importantes: Account, Session, User, ERC721, Listing, y Sale. Los Usuarios son el centro de la aplicación, y cada uno tiene una cuenta que guarda información como el address. Los Usuarios también pueden tener Sesiones activas. Cada Usuario puede interactuar con los NFTs, que están representados en la tabla ERC721. Estos NFTs se pueden listar para la venta en la tabla Listing o pueden haber sido vendidos, lo que se registra en la tabla Sale. Todo está interconectado, manteniendo un registro de todas las transacciones y actividades de los usuarios.

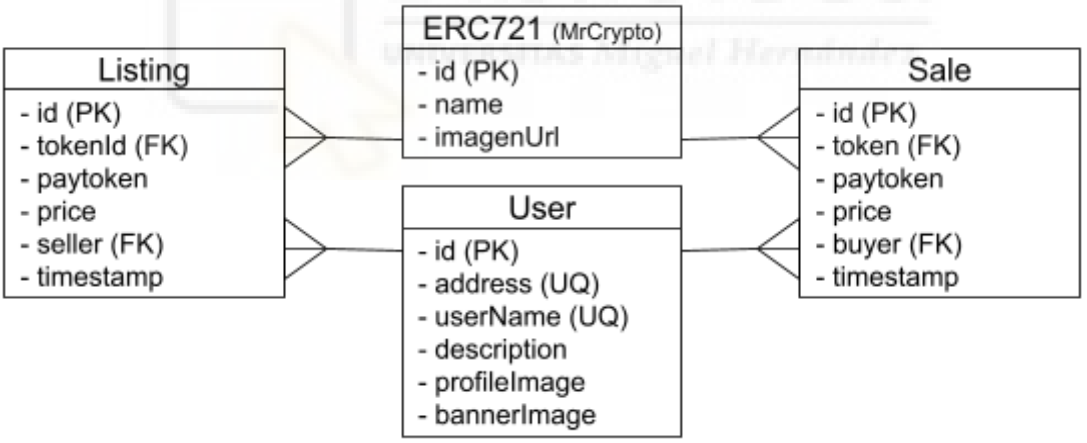

Figura 4.26: Tablas User, Listing, Sale, ERC721

La tabla User actúa como el núcleo central de la base de datos, donde cada usuario es identificado de manera única mediante un identificador generado por la función "*cuid()*" de Prisma. Esta función es preferible sobre los métodos de incremento automático tradicionales debido a que produce identificadores que son más difíciles de predecir, aumentando así la seguridad.

Cada usuario puede asociarse con múltiples Account y Session a través del campo userId, que actúa como clave ajena en estas tablas. La relación entre User y Account permite que

un mismo usuario se conecte mediante diferentes proveedores de wallets, no limitándose a una sola opción como MetaMask, por ejemplo.

Por otro lado, la relación entre User y Session, cada vez que un usuario se conecta, se genera una nueva sesión, lo que permite un seguimiento detallado y seguro de las interacciones del usuario con la aplicación a lo largo del tiempo.

Las relaciones entre las tablas User, Listing, Sale, y ERC721 tratan de la siguiente manera, cada Listing está asociado con un User a través del campo “seller”, estableciendo una relación de uno a muchos donde un usuario puede tener múltiples listados activos, cada uno representando un NFT que se ofrece para la venta. De manera similar, la tabla Sale vincula cada venta a un User utilizando el campo “buyer”, permitiendo que un usuario acumule múltiples compras a lo largo del tiempo.

La tabla ERC721 se integra estrechamente con Listing y Sale para trazar el ciclo de vida de cada NFT. Un ERC721 puede estar relacionado con varios Listing, mostrando que un NFT puede ponerse a la venta varias veces bajo diferentes condiciones. Igualmente, cada Sale está vinculada a un NFT del modelo ERC721, lo que indica que un NFT puede cambiar de manos repetidamente a través de ventas sucesivas.

4.4.3.- Diseño de los Smart Contracts

Para el diseño de los smart contracts del proyecto, se ha optado por utilizar un diagrama de clases (figura 4.26) que ayuda a visualizar la estructura y las relaciones entre los diferentes contratos. Este diagrama incluye detalles como atributos y métodos de los smart contracts implicados. Se han utilizado colores en el diagrama para plasmar algunas diferencias entre los smart contracts: el E7LMarketPlace, destacado en verde, es el principal y fue desarrollado específicamente para este proyecto. Este smart contract hereda características de ReentrancyGuard y Ownable, ambos provenientes de la reconocida librería de OpenZeppelin, de los cuales hemos hablado en el capítulo 3.

Además, en el diagrama aparecen dos smart contracts en amarillo, ERC721 (MrCrypto) y E7L (que incluye varios smart contracts), que representan los smart contract de prueba utilizados durante el desarrollo. Estos smart contract de prueba también heredan funcionalidades de smart contract estándares de OpenZeppelin, lo que asegura la robustez de la aplicación.

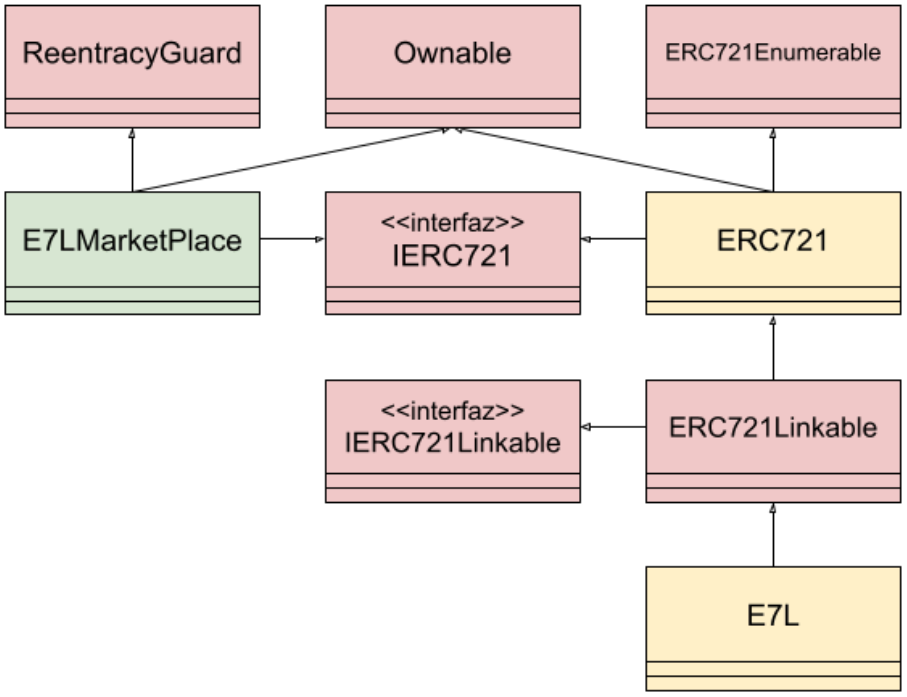

Figura 4.27: Diagrama de clases de todos los smart contracts

En la figura 4.28, se presentan los diagramas de clases de los tres principales smart contracts desarrollados. Estos diagramas detallan los atributos y métodos contenidos en cada uno de ellos, proporcionando una visualización clara de cómo están estructurados. A través de estos diagramas, podemos apreciar la funcionalidad específica que cada smart contract aporta al sistema.

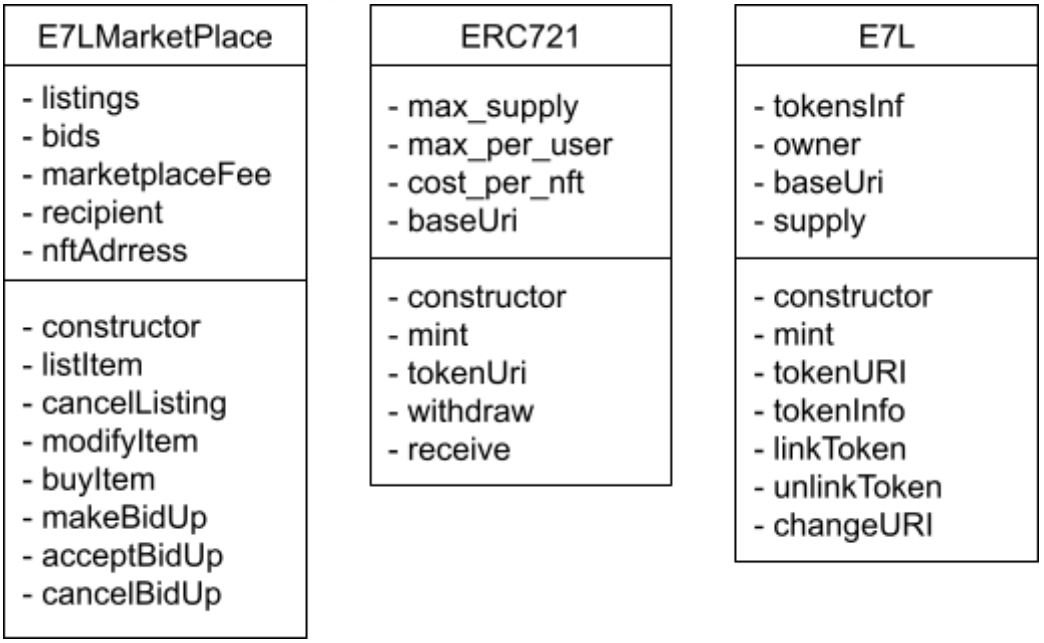

Figura 4.28: Diagrama de clases de los smart contract principales

4.5.- IMPLANTACIÓN Y DESPLIEGUE

Todo proyecto Web 3 en explotación está alojado en algún sistema blockchain, en nuestro caso, el presente proyecto será desplegado en la red de Polygon, que es una capa 2 (layer 2) construida sobre la red de Ethereum (layer 1). Durante toda la etapa de desarrollo es necesario emular de alguna manera este tipo de entornos en el equipo local, para ello se hace uso de herramientas como Hardhat y Node.js. En modo local, un contrato inteligente se puede desplegar sobre Hardhat, mientras que la web con la que interactúa el usuario final se construye sobre el servidor Node.js.

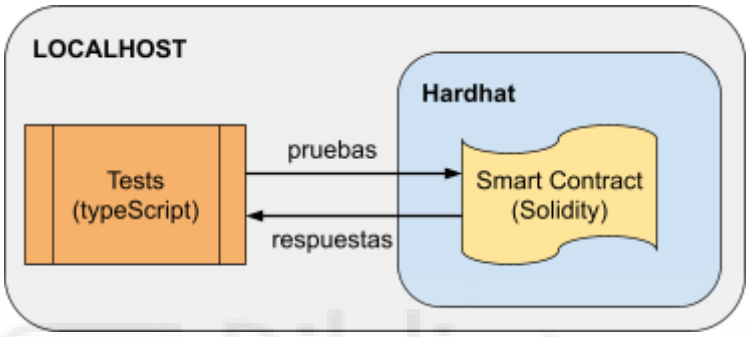

Figura 4.29: Entorno de desarrollo local

Cuando un smart contract está suficientemente validado, se despliega sobre una red de pruebas para su conexión con la aplicación web que hace de frontend para el usuario final, este entorno es mucho más parecido al entorno donde se realizará despliegue definitivo (producción), y en él se pueden hacer otras pruebas antes de dicho despliegue final.

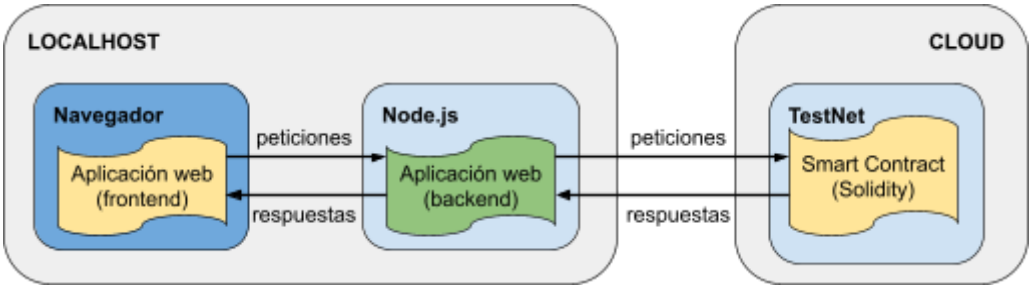

Figura 4.30: Entorno en red de pruebas

Los detalles acerca de cómo configurar Hardhat, cómo se han realizado los tests (en typescript) en localhost, y como se despliega un smart contract, y como verifica para que el código sea público, se pueden consultar en el Anexo II.

Capítulo 5
Conclusiones y trabajo futuro

5.1.- CONCLUSIONES

Durante el desarrollo de este proyecto, se ha profundizado en una variedad de tecnologías y explorado su integración en blockchain. La aplicación descentralizada desarrollada está en funcionamiento, sirviendo actualmente como un prototipo que ofrece una base sólida para futuras mejoras y desarrollos.

Aunque la funcionalidad principal se ha implementado con éxito, reconocemos que el diseño y la experiencia de usuario representan áreas de mejora continua. La estructura base está terminada y proporciona un marco sobre el cual seguir construyendo.

En cuanto al smart contract, funciona conforme a lo previsto, proporcionando un mecanismo funcional para las transacciones e interacciones dentro de la plataforma.

Recientemente, en abril de 2024, nos enfrentamos a un cambio cuando la plataforma Ethereum dejó de ofrecer soporte para la testnet Mumbai de Polygon, reemplazándola por una nueva testnet denominada Amoy [73]. Este cambio incluye actualizaciones y nuevas funcionalidades que requieren adaptaciones en nuestra aplicación [74].

El desafío no se limita solo a actualizar los datos de la testnet en los archivos de configuración de nuestro proyecto. También implica modificar varias de las bibliotecas que utilizamos, las cuales necesitan ser actualizadas para reconocer y funcionar correctamente con Amoy. Este proceso es complicado debido a que algunas de estas bibliotecas han modificado sus métodos, lo que requiere una revisión exhaustiva y adaptación del código existente para que nuestra aplicación continúe operando sin interrupciones en este nuevo entorno de pruebas.

5.2.- POSIBLES DESARROLLOS FUTUROS

Mirando hacia adelante, el camino para la evolución de esta dApp está claro y lleno de oportunidades para expandir y profundizar sus capacidades. El paso inmediato es realizar pruebas de las funcionalidades de listar y comprar NFTs utilizando nuestra colección de testeo para verificar y afinar las funcionalidades de listar y comprar NFTs.

Otra funcionalidad que se quiere añadir es el sistema de pujas en el frontend, permitiendo a los usuarios realizar ofertas por NFTs. Esto agrega una nueva dimensión al mercado, fomentando una mayor interacción y participación de la comunidad.

La seguridad es primordial en cualquier plataforma blockchain. Por ello, nos enfocaremos en fortalecer la seguridad de nuestro smart contract, implementando prácticas de codificación seguras y realizando auditorías exhaustivas para prevenir vulnerabilidades.

Uno de los pasos importantes que debemos tratar en el futuro es asegurar que el frontend sea completamente responsive. Esto implica ajustar el diseño de la página web para que se vea y funcione bien en una variedad de dispositivos, desde ordenadores de escritorio hasta smartphones. Hacer que el sitio web sea responsive no solo mejorará la accesibilidad y la experiencia del usuario, sino que también nos ayudará a alcanzar a una audiencia más amplia, facilitando el acceso y la interacción con nuestra dApp en diferentes plataformas.

Planificamos implementar filtros avanzados en la sección "Explorer" de nuestra dApp para mejorar la experiencia del usuario al buscar NFTs. Dado que los NFTs incluyen varios traits, un trait se refiere a las características únicas que define a un NFT individual, como son el color o la forma. con diferentes grados de rareza, como por ejemplo diferentes tipos de gorras de los cuales algunos estilos son más raros que otros, el filtrado permitirá a los

usuarios buscar eficazmente a través de estas características. Además, se incluirán opciones para ordenar los NFTs por precio, desde el más económico al más caro, y viceversa, así como por los más recientemente listados.

A medida que el diseño de la aplicación se finalice, ajustaremos el desarrollo para alinearnos con esta visión, asegurando que la interfaz de usuario no solo sea funcional sino también visualmente atractiva y acorde a las expectativas de nuestros usuarios.

Por último, el paso final necesario para que todo el desarrollo tenga sentido es desplegar toda la aplicación en la red principal de Polygon (mainnet), la capa 2 de la red de Ethereum más popular.

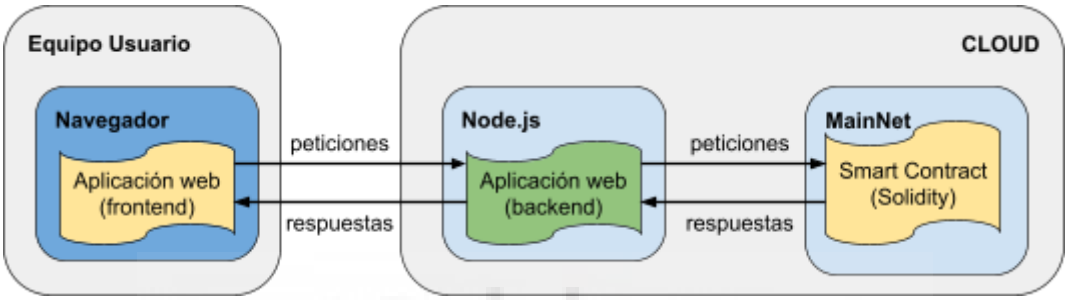

Figura 5.1: Despliegue final de la aplicación en MainNet

Bibliografía

[1] Beincrypto: Web2 y Web3: ¿Cuál es la diferencia entre ambos conceptos?
https://es.beincrypto.com/aprende/web2-web3/
08 Julio 2022

[2] Bit2me Academy: ¿Qué es DeFi o Finanzas Descentralizadas?
https://academy.bit2me.com/que-es-defi-o-finanzas-descentralizadas/
02 Marzo 2023

[3] Binance Academy: What Is An NFT?
https://academy.binance.com/es/articles/what-is-an-nft
11 Enero 2024

[4] Ethereum: Comparación entre web2 y web3
https://ethereum.org/es/developers/docs/web2-vs-web3
13 Noviembre 2023

[5] Bitpanda: ¿Qué es el doble gasto y por qué supone un problema?
https://www.bitpanda.com/academy/es/lecciones/que-es-el-doble-gasto-y-por-que-supone-un-problema/
Fecha Consulta: 21/12/2023

[6] Medium: Bitcoins y el Problema de los Generales Bizantinos
https://freddy-abadl.medium.com/bitcoins-y-el-problema-de-los-generales-bizantinos-54b067dca501
09 Noviembre 2019

[7] Cointelegraph: How does blockchain solve the Byzantine generals problem?
https://cointelegraph.com/learn/how-does-blockchain-solve-the-byzantine-generals-problem/
Fecha Consulta: 22/12/2023

[8] Bit2me Academy: ¿Qué es una DAO?
https://academy.bit2me.com/que-es-una-dao/
17 Febrero 2023

[9] Bit2me Academy: Smart contracts: ¿Qué son, cómo funcionan y qué aportan?
https://academy.bit2me.com/que-son-los-smart-contracts/
09 Febrero 2023

[10] Bit2me Academy: ¿Qué son las Cold Wallets?
https://academy.bit2me.com/que-son-cold-wallets/
24 Febrero 2023

[11] Bit2me Academy: ¿Qué son las Hot Wallets?
https://academy.bit2me.com/que-son-hot-wallets/
11 Febrero 2020

[12] OpenSea
https://opensea.io/
Fecha consulta: 29/12/2023

[13] Magic Eden
https://magiceden.io/
Fecha consulta: 29/12/2023

[14] Blur
https://blur.io/
Fecha consulta: 29/12/2023

[15] Mr.Crypto
https://www.mrcryptonft.com/
Fecha consulta: 30/12/2023

[16] Racks Labs
https://www.labs.racksmafia.com/
Fecha consulta: 30/12/2023

[17] Beincrypto: Guía básica ¿Cómo funcionan las transacciones con Bitcoin?
https://es.beincrypto.com/aprende/guia-basica-como-funcionan-transacciones-bitcoin-btc/
07 Mayo 2020

[18] Bit2me Academy: Transacciones Bitcoin, ¿cómo funcionan?
https://academy.bit2me.com/transacciones-bitcoin/
23 Febrero 2023

[19] Xataka
https://www.xataka.com/criptomonedas/bitcoin-como-desastre-medioambiental-que-sea-mayor-despilfarro-energetico-historia-depende-su-futuro/
13 Mayo 2021

[20] Cointelegraph: Ha ocurrido la fusión de Ethereum a proof-of-stake
https://es.cointelegraph.com/news/breaking-historic-day-for-crypto-as-ethereum-merge-to-proof-of-stake-occurs/
15 Septiembre 2022

[21] Binance Academy: ¿Qué es Proof of Stake (PoS)?
https://academy.binance.com/es/articles/proof-of-stake-explained/
09 Junio 2023

[22] Ethereum: Gas
https://ethereum.org/es/developers/docs/gas/
25 Septiembre 2023

[23] Criptoactualidad: El estándar ERC-20
https://criptoactualidad.net/estandar-erc-20/
Fecha consulta: 04/01/2024

[24] Fabian Vogelsteller & Vitalik Buterin
https://eips.ethereum.org/EIPS/eip-20
19 Noviembre 2015

[25] Criptonoticias: Datos curiosos que nos dejaron las ICO del 2017
https://www.criptonoticias.com/mercados/datos-curiosos-dejaron-ico-2017/
27/12/2017

[26] Bit2me Academy: ¿Qué es un token ERC 721?
https://academy.bit2me.com/que-es-token-erc-721/
28 Abril 2023

[27] Ethereum: Estándar de token no fungible ERC-721
https://ethereum.org/es/developers/docs/standards/tokens/erc-721/
19 Noviembre 2023

[28] Oncyber
https://oncyber.io/
Fecha consulta: 14 Enero 2023

[29] Github: Racks Labs
https://github.com/Racks-Labs/ERC721-Linkable
04 Enero 2023

[30] Medium: Indexadores en el protocolo de NEAR
https://medium.com/@owa_academy/indexadores-en-el-protocolo-de-near-66d2ca11941a
27 Abril 2023

[31] Racks Community
https://github.com/Racks-Community/MrCryptoIndexer.git/
30 Noviembre 2023

[32] Neoguias: Eventos en Solidity
https://www.neoguias.com/eventos-solidity/
13 Diciembre 2022

[33] Developer Mozilla: HTML Lenguaje de etiquetas de hipertexto
https://developer.mozilla.org/es/docs/Web/HTML
24 Julio 2023

[34] Developer Mozilla: ¿Qué es el CSS?
https://developer.mozilla.org/es/docs/Learn/CSS/First_steps/What_is_CSS
02 Agosto 2023

[35] Tailwindcss: Rapidly build modern websites without ever leaving your HTML.
https://tailwindcss.com/
Fecha consulta: 18 Enero 2024

[36] DaisyUi: The most popular component library for Tailwind CSS
https://daisyui.com/
Fecha consulta: 18 Enero 2024

[37] Developer Mozilla: ¿Qué es JavaScript?
https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript
02 Agosto 2023

[38] Profile: TypeScript: qué es diferencias con JavaScript y por qué aprenderlo
https://profile.es/blog/que-es-typescript-vs-javascript/
25 Octubre 2021

[39] Medium: Por qué deberías empezar ya con TypeScript
https://medium.com/@mario_gl/por-qu%C3%A9-deber%C3%ADas-empezar-ya-con-typescript-f0e4f5dfed9d
05 Septiembre 2018

[40] Next: The React Framework for the Web
https://nextjs.org/
Fecha consulta: 19 Enero 2024

[41] Wagmi: Reactivity for Ethereum apps
https://wagmi.sh/
Fecha consulta: 19 Enero 2024

[42] Node.js
https://nodejs.org/en/learn/getting-started/introduction-to-nodejs
Fecha consulta: 20 Enero 2024

[43] pnpm vs npm
https://pnpm.io/es/pnpm-vs-npm/ç
Fecha consulta: 20 Enero 2024

[44] Softzone: Conoce SQLite, el popular motor de bases de datos
https://www.softzone.es/programas/lenguajes/que-es-sqlite/
28 Noviembre 2021

[45] Prisma
https://www.prisma.io/
Fecha consulta: 21 Enero 2024

[46] Graphql
https://graphql.org/
Fecha consulta: 21 Enero 2024

[47] tRPC
https://trpc.io/
Fecha consulta: 21 Enero 2024

[48] ¿Qué es Endpoint? La guía completa de seguridad Endpoint.
https://admcloudservices.com/blog/que-es-endpoint-la-guia-completa-de-seguridad-de-endpoints-y-como-prevenir-filtraciones-de-datos.html/
Fecha consulta: 21 Enero 2024

[49] Xataka: Solidity, qué es y para qué sirve este lenguaje de programación.
https://www.xataka.com/basics/solidity-que-sirve-este-lenguaje-programacion/
12 Abril 2022

[50] Binance: Qué es Solidity: aprende cómo iniciarte en lenguaje de programación
https://www.binance.com/es/blog/ecosystem/qu%C3%A9-es-y-c%C3%B3mo-entender-la-programaci%C3%B3n-solidity-1234417533305856 87
12 Abril 2022

[51] Cointelegraph: Solidity, el lenguaje de programación de Ethereum.
https://es.cointelegraph.com/explained/solidity-the-programming-language-of-the-ethereum/
23 Mayo 2020

[52] Bit2me Academy: ¿Qué es un OP_CODE?
https://academy.bit2me.com/que-es-un-op-code/
26 Junio 2020

[53] Bit2me Academy: ¿Qué es la Ethereum Virtual Machine (EVM)?
https://academy.bit2me.com/que-es-ethereum-virtual-machine-evm/
04 Abril 2023

[54] Hardhat
https://hardhat.org/
Fecha consulta: 23 Enero 2024

[55] OpenZeppelin
https://www.openzeppelin.com/
Fecha consulta: 24 Enero 2024

[56] Basic Solidity
https://medium.com/@fuelusumar/basic-solidity-state-variables-types-constants-functions-visibility-and-getters-functions-6528c70e218/
23 Julio 2018

[57] ¿Qué son los modificadores en Solidity y cómo se usan?
https://keepcoding.io/blog/modificadores-en-solidity-guia-completa/#:~:text=Los%20modificadores%20en%20Solidity%20son,otras%20funciones%20en%20el%20contrato.
11 abril 2024

[58] Introducción a las vulnerabilidades más comunes y ataques en smart-contracts
https://spanish-marketing.medium.com/coinex-research-introducci%C3%B3n-a-las-vulnerabilidades-m%C3%A1s-comunes-y-ataques-en-smart-contracts-569deb0e9a5a
10 diciembre 2023

[59] Ethers.js
https://docs.ethers.org/v5/
Fecha consulta: 25 Enero 2024

[60] Reporter Gas
https://www.npmjs.com/package/hardhat-gas-reporter/
Fecha consulta: 25 Enero 2024

[61] Coverage
https://www.npmjs.com/package/solidity-coverage/
Fecha consulta: 25 Enero 2024

[62] Dotenv
https://www.npmjs.com/package/dotenv/
Fecha consulta: 25 Enero 2024

[63] NextAuth.js
https://next-auth.js.org/
Fecha consulta: 25 Enero 2024

[64] Github
https://github.com/
Fecha consulta: 26 Enero 2024

[65] Visual Studio Code
https://code.visualstudio.com/
Fecha consulta: 26 Enero 2024

[66] Docker
https://www.docker.com/
Fecha consulta: 26 Enero 2024

[67] Turbo
https://turbo.build/
Fecha consulta: 26 Enero 2024

[68] T3 Stack
https://create.t3.gg/
Fecha consulta: 26 Enero 2024

[69] Platzi: Qué son las metodologías ágiles de desarrollo de software
https://platzi.com/blog/amb-que-son-las-metodologias-agiles/
04 Enero 2023

[70] Platzi: Qué es la metodología SCRUM y sus roles
https://platzi.com/blog/que-es-scrum-y-los-roles-en-scrum/
04 Enero 2017

[71] Platzi: ¿Cómo funciona la metodología Scrum? Qué es y sus 5 fases
https://platzi.com/blog/metodologia-scrum-fases/
04 Enero 2015

[72] Atlassian: ¿Qué son los diagramas de Gantt?
https://www.atlassian.com/es/agile/project-management/gantt-chart
Fecha consulta: 28 Enero 2024

[73] Polygon Mumbai Support Ending April 13th - Migrate to Amoy
https://www.alchemy.com/blog/polygon-mumbai-testnet-deprecation
10 Abril 2024

[74] Introducing the Amoy Testnet for Polygon PoS
https://polygon.technology/blog/introducing-the-amoy-testnet-for-polygon-pos
10 Abril 2024

Anexo I
Creación de un proyecto con T3-Stack

Para preparar el entorno de desarrollo y comenzar a trabajar en nuestro proyecto, es necesario tener Node.js y npm (Node Package Manager) instalados en nuestro sistema. Node.js es un entorno de ejecución para JavaScript en el servidor, que permite ejecutar aplicaciones fuera del navegador. Por otro lado, npm es el sistema de gestión de paquetes de Node.js, indispensable para instalar y administrar librerías y herramientas necesarias en el desarrollo de aplicaciones.

Para instalar Node.js y npm, primero hay que acceder al sitio web oficial de Node.js (https://nodejs.org/) y navegar hasta la sección de descargas. Aquí encontraremos dos versiones principales para descargar: la versión LTS (Long Term Support), que es recomendada para la mayoría de los usuarios debido a su estabilidad y soporte a largo

plazo, y la versión Current, que contiene las últimas funcionalidades. Seleccionamos y descargamos el archivo de instalación de Windows Installer (.msi) apropiado para la arquitectura de nuestro sistema (32 bits o 64 bits), figura A.I.1. El proceso de instalación es sencillo: ejecutar el archivo .msi y seguir las instrucciones del asistente de instalación, seleccionando "Siguiente" en cada paso hasta completar la instalación.

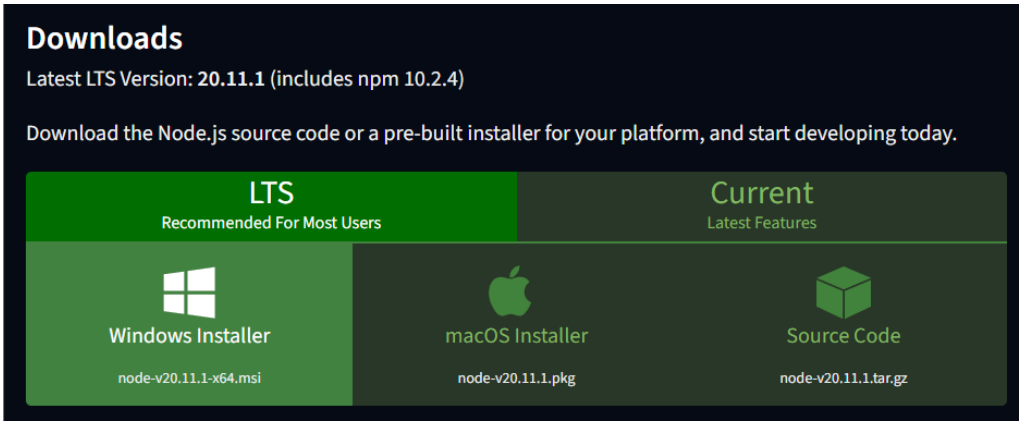

Figura A.I.1: Instalador de Node.js

Una vez finalizada la instalación, es importante verificar que Node.js y npm estén correctamente instalados en nuestro sistema. Para ello, abrimos una terminal o línea de comandos y escribimos:

```
node --version
```

Esto debería mostrar el número de versión de Node.js, indicando que está correctamente instalado. Para verificar la instalación de npm, escribimos:

```
npm --version
```

Debería verse el número de versión de npm, confirmando su instalación correcta, como se observa en la figura A.I.2.

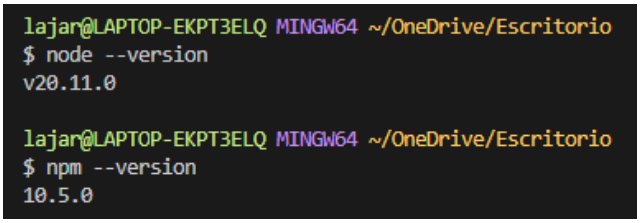

Figura A.I.2: Versiones de node y npm

Para instalar pnpm, se utiliza npm ejecutando:

```
npm install -g pnpm
```

Esta acción instala pnpm de forma global, permitiéndote acceder a él desde cualquier ruta en el sistema. Una vez finalizada la instalación, se puede confirmar que pnpm está listo para usar verificando su versión, como se muestra en la figura A.I.3.

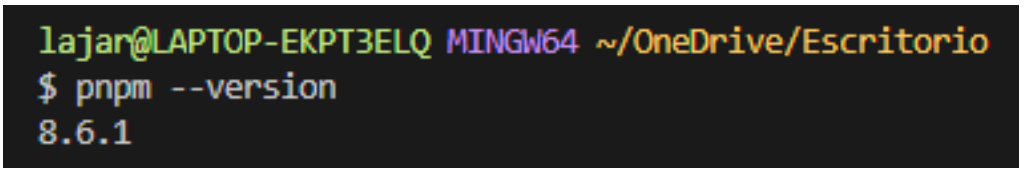

Figura A.I.3: Versión de pnpm

Para instalar npx globalmente, un ejecutor de paquetes que viene incluido con npm 5.2.0 y versiones superiores, simplemente hay que ejecutar:

```
npm install -g npx
```

La verificación de la instalación se puede realizar como se muestra en la figura A.I.4, confirmando que npx está listo para su uso.

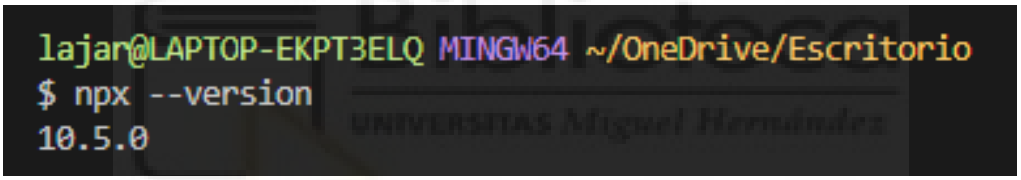

Figura A.I.4: Versión de npx

Una vez instalados todos los gestores de paquetes necesarios, se puede instalar el entorno de desarrollo.

T3 Stack es un marco de trabajo para el desarrollo de aplicaciones web que combina tecnologías como TypeScript, Tailwind CSS, tRPC, entre otras, enfocándose en la escalabilidad y el uso de tipado fuerte. Este enfoque crea un mono repositorio, es decir, agrupa tanto el backend como el frontend dentro de un mismo repositorio. Esto facilita la gestión del proyecto al centralizar el código, permitiendo una integración y colaboración más fluidas entre las diferentes partes de la aplicación, además de simplificar los procesos de prueba y despliegue.

Se utiliza el siguiente comando para iniciar la creación de nuestro proyecto. Como se muestra en la figura A.I.5, si la terminal no es interactiva, se producirá un error. Para evitar esto, utilizaremos la terminal integrada en Visual Studio Code, que soporta sesiones interactivas, figura A.I.6.

```
pnpm create t3-app@latest
```

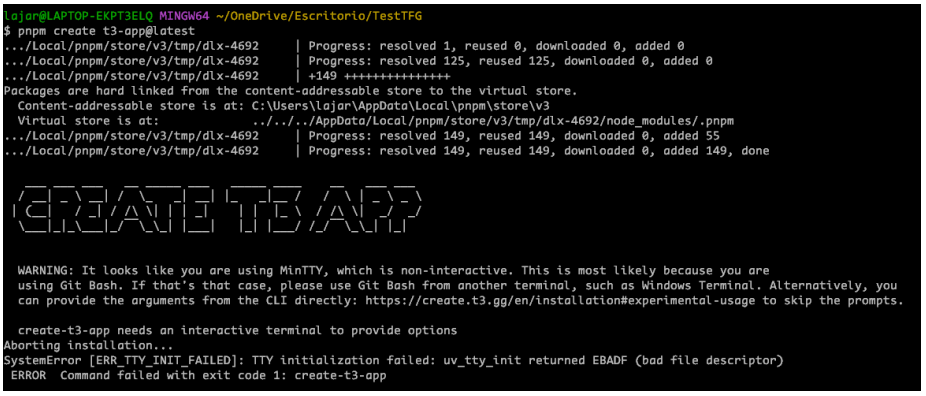

Figura A.I.5: Terminal no interactiva

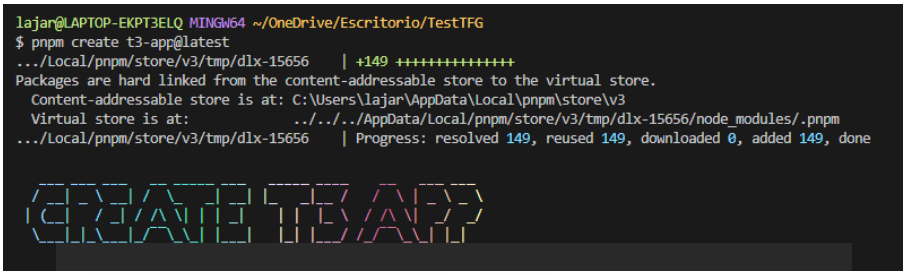

Figura A.I.6: Terminal interactiva

Una vez ejecutado el comando, iniciará un proceso interactivo como se observa en la figura A.I.7. Durante este proceso, se nos harán varias preguntas para configurar el proyecto según nuestras preferencias. En nuestro caso, seleccionamos TypeScript para el lenguaje de programación, Tailwind CSS para el diseño, tRPC para la comunicación entre el cliente y el servidor, Prisma como nuestro ORM, y SQLite como nuestra base de datos.

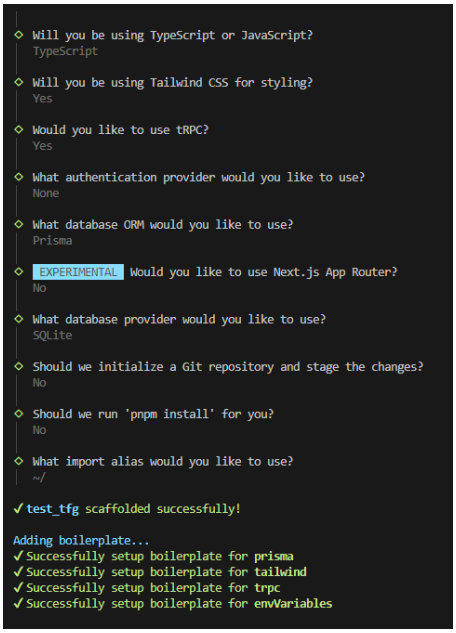

Figura A.I.7: Preguntas T3-Stack

Tras completar la configuración inicial, el siguiente paso es acceder al directorio del proyecto creado, usando el comando 'cd'. Una vez dentro, se ejecuta el siguiente comando para instalar todas las dependencias especificadas en el archivo "package.json":

```
pnpm install
```

Este paso asegura que todas las herramientas y librerías necesarias para el desarrollo del proyecto estén disponibles y listas para usar (ver figura A.I.8).

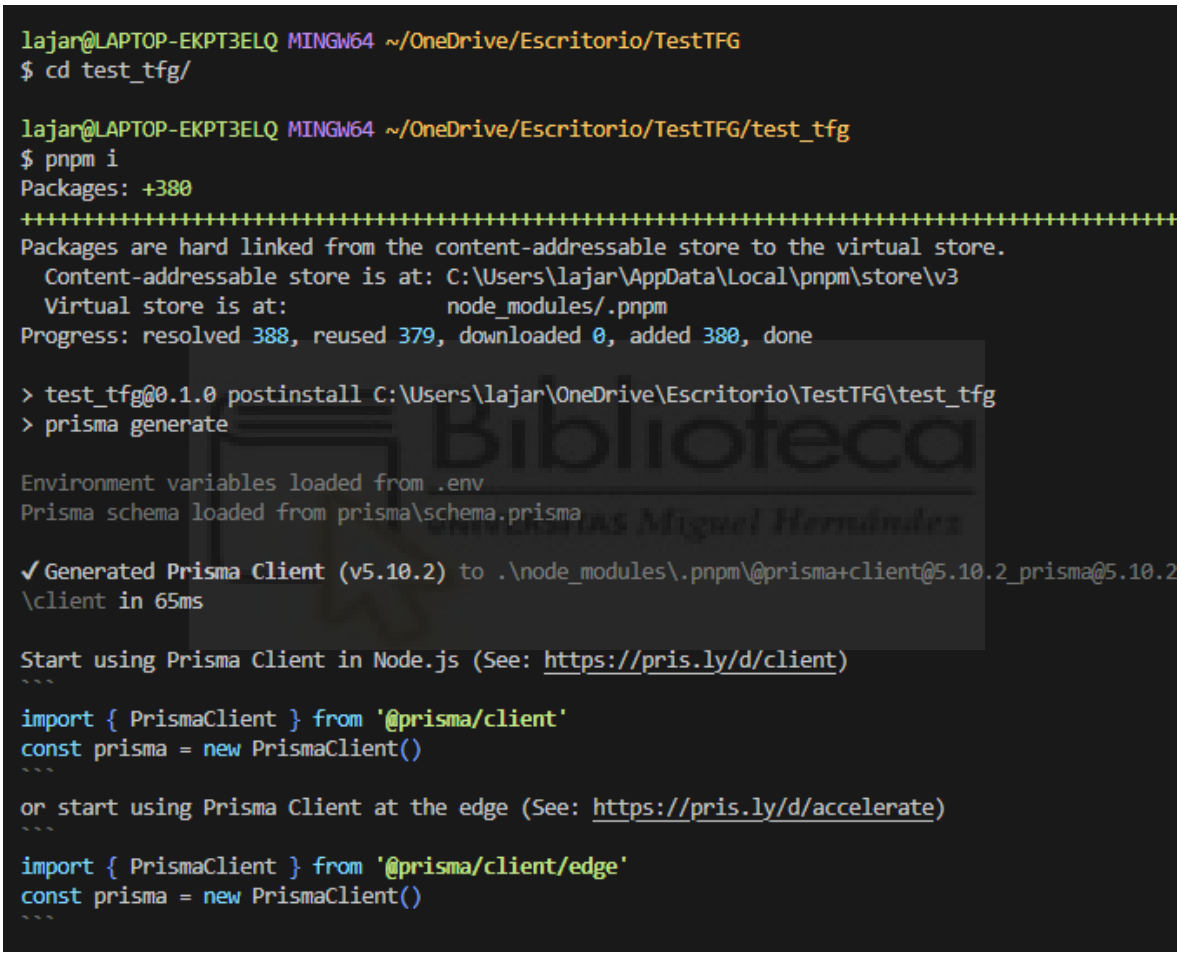

Figura A.I.8: Instalación de las librerías del package.json

A continuación de la información observada según la figura A.I.8, puede verse un listado completo de las dependencias instaladas junto con sus versiones específicas (figura A.I.9). Este paso asegura que el entorno de desarrollo está correctamente configurado con todas las herramientas y versiones necesarias para el proyecto.

```
dependencies:
+ @prisma/client 5.10.2
+ @t3-oss/env-nextjs 0.9.2
+ @tanstack/react-query 4.36.1 (5.24.8 is available)
+ @trpc/client 10.45.1
+ @trpc/next 10.45.1
+ @trpc/react-query 10.45.1
+ @trpc/server 10.45.1
+ next 14.1.0 (14.1.1 is available)
+ react 18.2.0
+ react-dom 18.2.0
+ superjson 2.2.1
+ zod 3.22.4

devDependencies:
+ @types/eslint 8.56.2 (8.56.5 is available)
+ @types/node 20.11.20 (20.11.24 is available)
+ @types/react 18.2.57 (18.2.61 is available)
+ @types/react-dom 18.2.19
+ @typescript-eslint/eslint-plugin 7.0.2 (7.1.0 is available)
+ @typescript-eslint/parser 7.0.2 (7.1.0 is available)
+ eslint 8.56.0 (8.57.0 is available)
+ eslint-config-next 14.1.0 (14.1.1 is available)
+ postcss 8.4.34 (8.4.35 is available)
+ prettier 3.2.5
+ prettier-plugin-tailwindcss 0.5.11
+ prisma 5.10.2
+ tailwindcss 3.4.1
+ typescript 5.3.3

Done in 12.3s
```

Figura A.I.9: Instalación de las librerías del package.json

Con la instalación completada, ahora se dispone de un repositorio preparado para el desarrollo web tradicional, tal como se visualiza en la estructura presentada en la figura A.I.10. Entre los archivos y carpetas importantes, encontramos el esquema de Prisma para la definición de la base de datos, la carpeta public destinada a almacenar imágenes y otros activos estáticos, y el archivo de configuración environment para gestionar variables de entorno esenciales para nuestra aplicación, entre otros archivo y/o carpetas.

Además, hemos añadido manualmente cuatro directorios clave para organizar nuestro proyecto:

- Components: Este directorio albergará los componentes React de nuestra aplicación, facilitando la reutilización y el mantenimiento del código de la interfaz de usuario.

- Constants: Tras desplegar un smart contract, obtenemos una dirección (address) y una interfaz de programación de aplicaciones (API). Esta carpeta contiene esa información, necesaria para interactuar con el smart contract desde la aplicación.

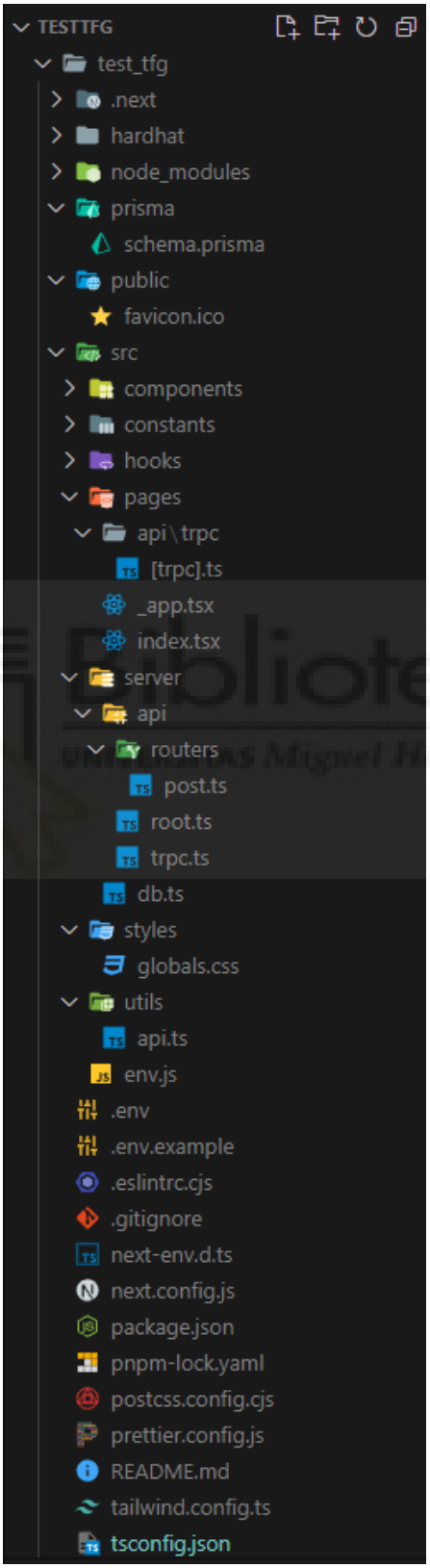

Figura A.I.10: Estructura del directorio

- Hooks: Es una función que permite integrar funcionalidades de Web3 dentro de componentes React. Estos hooks específicos de Web3 facilitan la interacción con la blockchain, como el envío de transacciones, la lectura de datos de smart contracts, y la gestión del estado de estas operaciones dentro de la interfaz de usuario de una aplicación descentralizada (DApp). Por ejemplo, en el desarrollo de DApps, se pueden utilizar hooks personalizados que aprovechan librerías como ethers.js para realizar operaciones como conectar a wallets de criptomonedas, leer el balance de tokens, escribir en smart contracts, o esperar por la confirmación de transacciones en la red blockchain.

- Hardhat: Este directorio, que configuraremos en detalle en el anexo II, es el entorno de desarrollo de smart contracts. Aunque su creación es manual en este paso, su presencia anticipada en la estructura del proyecto ayuda a localizar y organizar las tareas relacionadas con el desarrollo.

Para iniciar y visualizar la aplicación en un entorno local, se ejecuta el siguiente comando, que produce la salida que se muestra en la figura A.I.11:

```
pnpm dev
```

```
lajar@LAPTOP-EKPT3ELQ MINGW64 ~/OneDrive/Escritorio/TestTFG/test_tfg
$ pnpm dev

> test_tfg@0.1.0 dev C:\Users\lajar\OneDrive\Escritorio\TestTFG\test_tfg
> next dev

   ▲ Next.js 14.1.0
   - Local:        http://localhost:3000
   - Environments: .env

 ✓ Ready in 3s
 ○ Compiling / ...
 ✓ Compiled / in 3.2s (333 modules)
 ✓ Compiled /api/trpc/[trpc] in 224ms (113 modules)
```

Figura A.I.11: Arranque del proyecto

Este comando arranca el servidor de desarrollo, permitiéndonos acceder a la aplicación web desde el navegador mediante la dirección *http://localhost:3000/*. Al hacerlo, es posible navegar por la aplicación y ver su interfaz de usuario y funcionalidades en acción, como se muestra en la figura A.I.12. Este paso ofrece una manera rápida de visualizar los cambios en tiempo real.

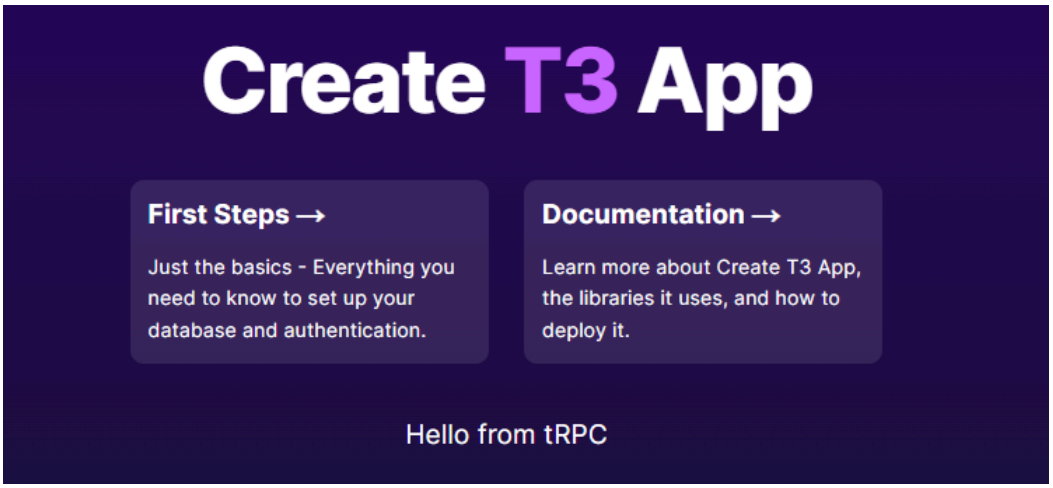

Figura A.I.12: Página base de T3 App

Anexo II Instalación y configuración de Hardhat

La instalación y configuración de Hardhat, una herramienta para el desarrollo de smart contracts en la máquina virtual de Ethereum, se integra sin problemas en el proyecto gracias al gestor de paquetes Node.js. A continuación, se detalla el proceso paso a paso.

La instalación de Hardhat en el propio ordenador es sencilla. Primero se ubica la terminal en la ruta específica del directorio del proyecto y se ejecuta el comando:

```
npm install --save-dev hardhat
```

Es importante situarnos en el directorio específico dentro del proyecto global dedicado al manejo de smart contracts antes de ejecutar el comando. Esto permite organizar mejor el

código y separar los paquetes de desarrollo del frontend y del manejo de smart contracts, lo cual implica que cada directorio contenga un conjunto específico de herramientas, bibliotecas y dependencias necesarias.

Para configurar un nuevo proyecto Hardhat el el equipo local, generalmente se utilizaría el comando "*npx hardhat init*". Sin embargo, en nuestro caso, para mantener la coherencia en el uso de herramientas dentro del proyecto, se usará el comando:

`pnpm dlx hardhat init`

La figura A.II.1 muestra el proceso de inicialización y en la figura A.II.2 las opciones que se presentan para configurar el proyecto Hardhat.

Figura A.II.1: Iniciar proyecto en Hardhat

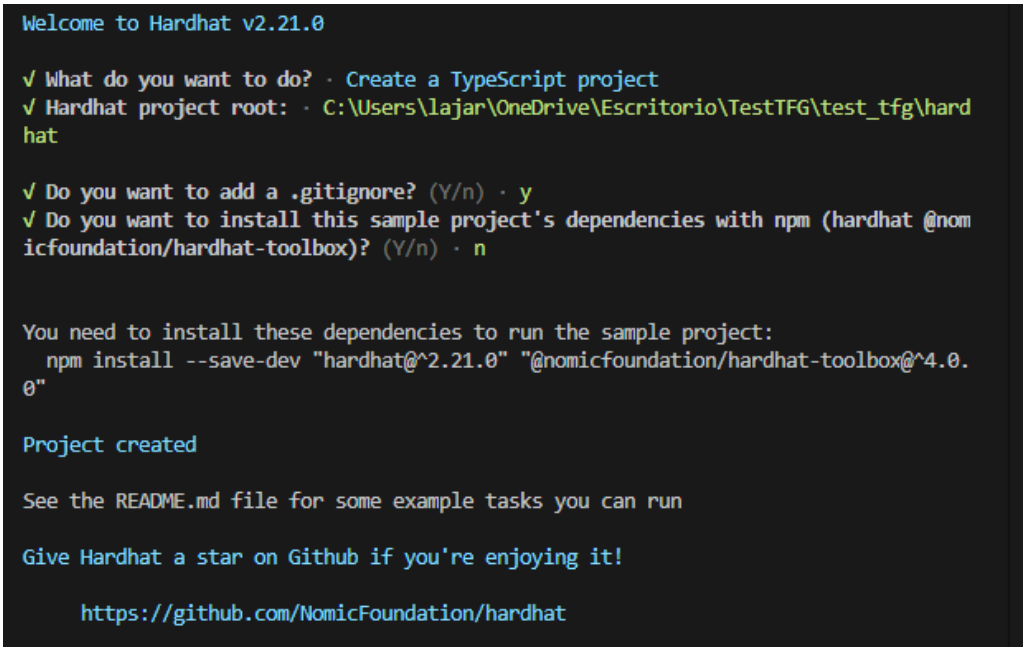

Figura A.II.2: Opciones al crear un nuevo proyecto en Hardhat

Durante la inicialización, se presentan opciones para instalar ciertas dependencias necesarias para trabajar con Hardhat y TypeScript. Si no se seleccionan estas opciones directamente en la terminal, siempre se pueden agregar manualmente ejecutando:

```
pnpm install --save-dev "hardhat@^2.21.0"
"@nomicfoundation/hardhat-toolbox@^4.0.0"
```

Este comando instala la versión específica de Hardhat y el conjunto de herramientas de @nomicfoundation/hardhat-toolbox, proporcionando un entorno de desarrollo para los smart contracts. La terminal mostrará este comando actualizado (figura A.II.3), asegurando el uso de las versiones más recientes y compatibles de las herramientas necesarias.

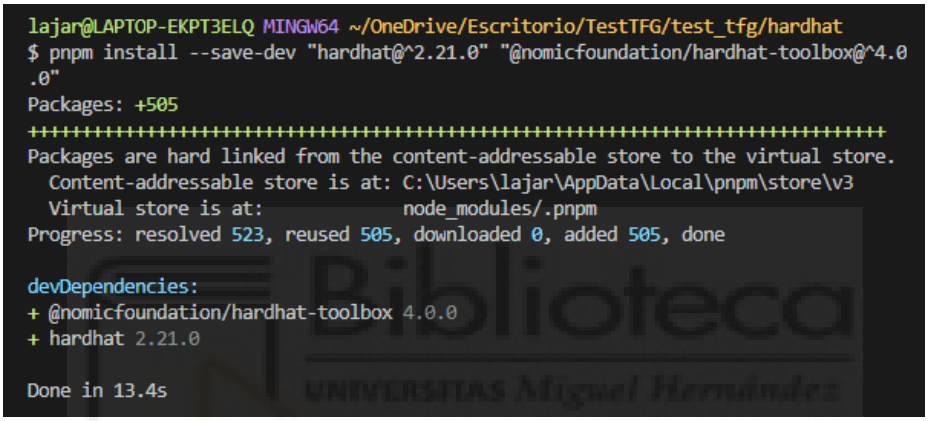

Figura A.II.3: Instalación de dependencias

Siguiendo estos pasos, configuramos eficazmente nuestro entorno de desarrollo para smart contracts con Hardhat como se muestra en la figura A.II.4.

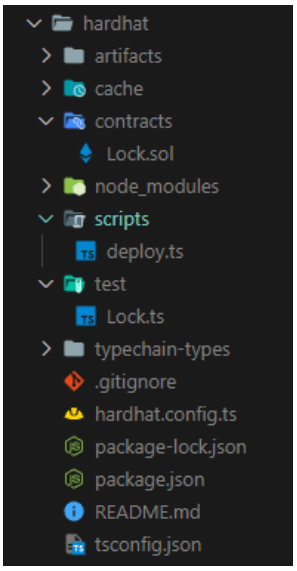

Figura A.II.4: Estructura del directorio Hardhat

A continuación, se añade la suite de smart contracts seguros de OpenZeppelin, ampliamente reconocidos por su robustez y seguridad; también se instala hardhat-verify para incorporar la herramienta de verificación de Hardhat que facilita la verificación de los contratos en redes públicas (como Ethereum) con los comandos:

```
npm install @openzeppelin/contracts
```

```
npm install @nomicfoundation/hardhat-verify --save-dev
```

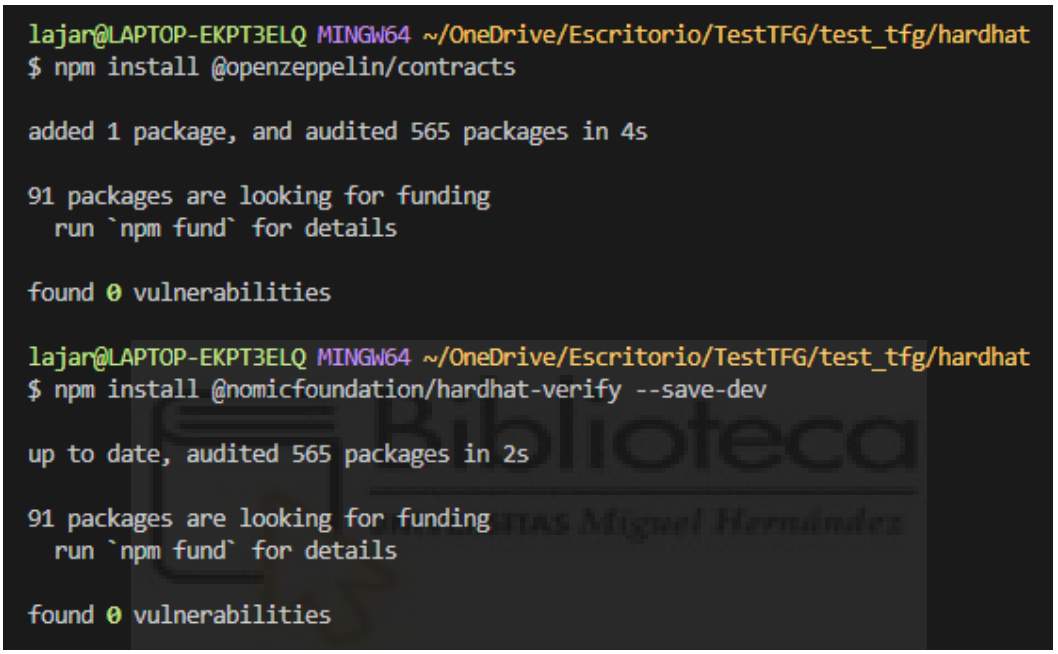

Figura A.II.5: Instalación de librerías adicionales

Con el entorno ya configurado, el siguiente paso es compilar los smart contracts para asegurar que estén libres de errores y listos para el despliegue con el comando:

```
npx hardhat compile
```

Este proceso verifica la sintaxis y la lógica de los contratos, señalando cualquier error encontrado para su corrección, como se muestra en la figura A.II.6.

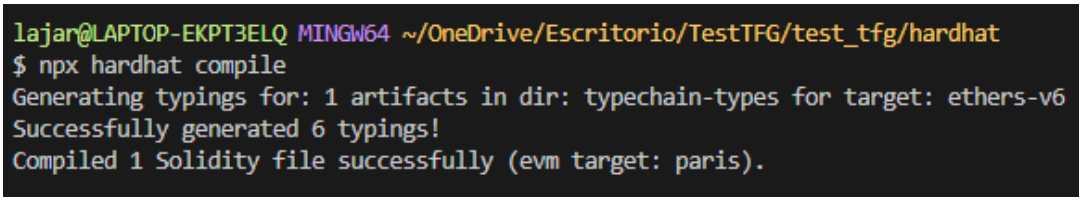

Figura A.II.6: Compilar smart contracts

Hardhat también facilita un entorno de pruebas para smart contracts. Dentro del directorio Hardhat, en el subdirectorio test hay un archivo de ejemplo proporcionado durante la instalación. Aquí, se crean los archivos de prueba específicos para los contratos, como "*test/Locks.ts*", para realizar tests que aseguren su correcto funcionamiento. Ejecutamos estas pruebas con el comando:

```
npx hardhat test test/Locks.ts
```

Todos estos pasos permiten simular la ejecución de contratos en un entorno controlado y verificar su comportamiento antes del despliegue en la red blockchain. Este paso es útil para identificar y corregir posibles fallos de manera anticipada, asegurando la calidad y la seguridad, como se muestra en en la figura A.II.7.

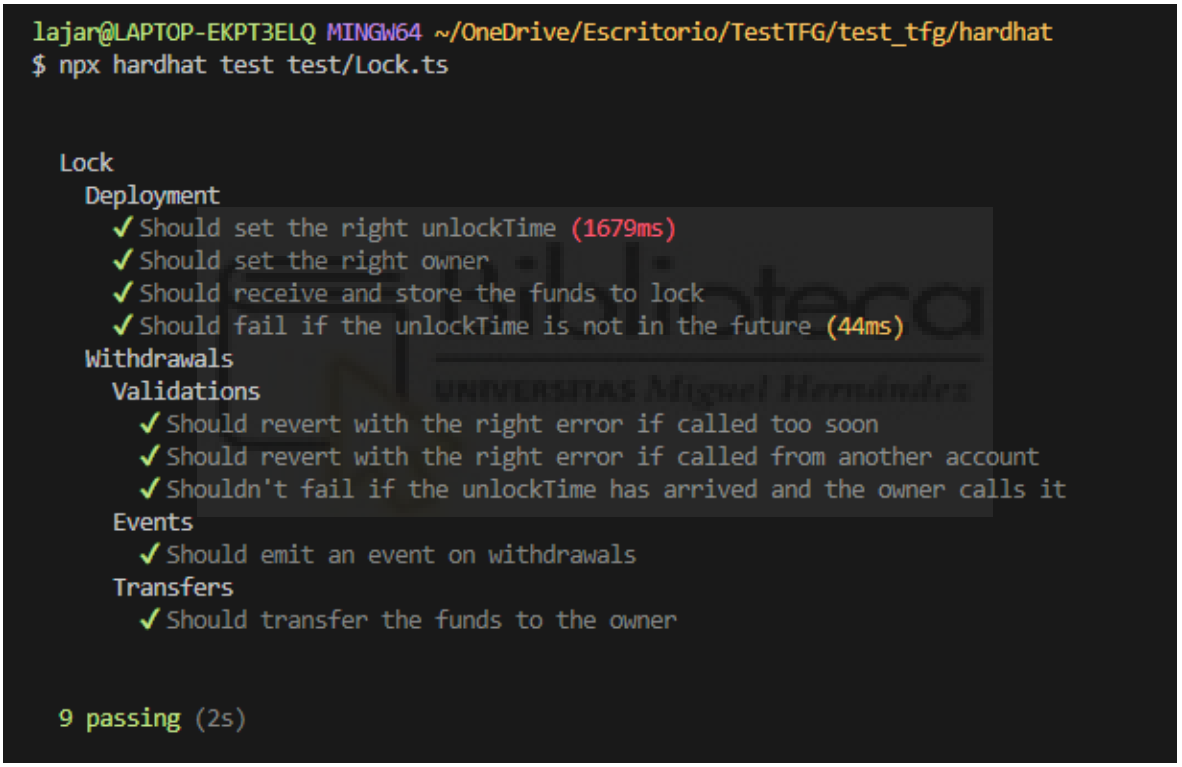

Figura A.II.7: Tests en Hardhat

Antes de proceder con el despliegue de un smart contract en la blockchain, hay que configurar el entorno de Hardhat adecuadamente. Para ello, creamos un archivo .env dentro del directorio de Hardhat, figura A.II.9, donde se especifican algunas variables de entorno para el proceso.

Antes de crear el archivo debemos instalar la biblioteca 'dotenv' como el siguiente comando (ver figura A.II.8):

```
pnpm install dotenv --save
```

```
lajar@LAPTOP-EKPT3ELQ MINGW64 ~/OneDrive/Escritorio/TestTFG/test_tfg/hardhat
$ pnpm install dotenv --save
Packages: +1
+
Progress: resolved 524, reused 506, downloaded 0, added 1, done

dependencies:
+ dotenv 16.4.5

Done in 7.2s
```

Figura A.II.8: Instalación de la biblioteca 'dotenv'

Las variables de entorno que vamos a necesitar son:

- PROVIDER_URL_MUMBAI: Este es el enlace al nodo de la testnet de Mumbai en Polygon proporcionado por servicios como Alchemy, tenemos que registrarnos en el servicio. Aquí colocamos nuestra clave privada de Alchemy para establecer una conexión segura con la red. La URL completa se verá algo así: PROVIDER_URL_MUMBAI=https://polygon-mumbai.g.alchemy.com/v2/tu_clave_privada_de_alchemy.

- PRIVATE_KEY: La clave privada de nuestra wallet, que utilizaremos para el despliegue del contrato. Es necesario que esta wallet contenga fondos, incluso en una testnet, para cubrir los costos de gas del despliegue. Los fondos de testnet se pueden obtener gratuitamente pero son limitados y se solicitan en el sitio web correspondiente de la testnet.

- POLYGONSCAN_API_KEY: Para la verificación de nuestro contrato en la blockchain y facilitar su interacción, necesitamos una clave API de PolygonScan, que se obtiene creando una cuenta en su plataforma.

PolygonScan es una herramienta que ofrece una interfaz de usuario para visualizar y buscar transacciones, bloques, y smart contracts en la blockchain de Polygon. Permite a desarrolladores y usuarios verificar las operaciones realizadas, consultar el código de los smart contracts y acceder a datos.

```
PROVIDER_URL_MUMBAI=
PRIVATE_KEY=
POLYGONSCAN_API_KEY=
```

Figura A.II.9: Tests en Hardhat

Para configurar correctamente nuestro proyecto y establecer la comunicación con la blockchain, es necesario ajustar el archivo hardhat.config.ts. Este archivo es el corazón de la configuración de Hardhat, donde se define cómo se comportará Hardhat durante el proceso de compilación, pruebas, y despliegue de nuestros smart contracts, las forma de configurarlo es como se muestra en la figura A.II.10.

```
const config: HardhatUserConfig = {
  solidity: "0.8.24",
  networks: {
    polygonMumbai: {
      url: process.env.PROVIDER_URL_MUMBAI!,
      accounts: [process.env.PRIVATE_KEY!],
    },
  },
  etherscan: {
    apiKey: {
      polygonMumbai: process.env.POLYGONSCAN!,
    },
  },
```

Figura A.II.10: Configuración archivo hardhat.config.ts

Antes de desplegar en redes principales como Ethereum o Polygon, donde las operaciones tienen un costo en gas, es práctico y económico realizar pruebas en redes de prueba (testnet) como Mumbai para Polygon o Sepolia para Ethereum. Esto permite simular el despliegue y la interacción con el contrato sin incurrir en costos reales.

Si se trabaja con un smart contract personalizado, es necesario ajustar el script de despliegue, comúnmente ubicado en 'scripts/deploy.ts' dentro del entorno de Hardhat. Este script define cómo y qué contrato será desplegado en la blockchain. Primero se comprueba que el script funciona correctamente lanzandolo en local (figura A.II.11):

npx hardhat run scripts/deploy.ts

```
lajar@LAPTOP-EKPT3ELQ MINGW64 ~/OneDrive/Escritorio/TestTFG/test_tfg/hardhat
$ npx hardhat run scripts/deploy.ts
Lock with 0.001ETH and unlock timestamp 1709552221 deployed to 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

Figura A.II.11: Test del script de deploy.ts

Para iniciar el despliegue en una testnet, se ejecuta el mismo comando, pero añadiendo la blockchain de destino para el contrato (el nombre tiene que ser el mismo que el configurado previamente en hardhat.config.ts):

npx hardhat run scripts/deploy.ts --network polygonMumbai

Este paso se visualiza en la figura A.II.12, donde el script de despliegue interactúa con la testnet de Mumbai para Polygon, simulando el proceso de despliegue real pero sin costos asociados.

```
lajar@LAPTOP-EKPT3ELQ MINGW64 ~/OneDrive/Escritorio/TestTFG/test_tfg/hardhat
$ npx hardhat run scripts/deploy.ts --network polygonMumbai
Compiled 1 Solidity file successfully (evm target: paris).
Lock with 0.001ETH and unlock timestamp 1709578363 deployed to 0x5366fC2b36A770B75
8ac0D0c3e7E9178Ebe0BC22
```

Figura A.II.12: Despliegue en Mumbai

La verificación de un smart contract es un paso extra tras su despliegue en la blockchain, ya que facilita la interacción con el contrato a través de exploradores como PolygonScan. Para verificar un smart contract y hacer que su código fuente sea accesible y legible en la plataforma, se ejecuta el comando:

`npx hardhat verify --network <blockchain> <address> <arg>`

En la figura A.II.13, se observa el despliegue de un smart contract, mostrando la dirección donde ha sido desplegado (esto hay que escribirlo manualmente en el script de deploy). Esta dirección, junto con los argumentos usados por el constructor del contrato (si los hubiera), serán necesarios para completar el proceso de verificación en PolygonScan.

```
lajar@LAPTOP-EKPT3ELQ MINGW64 ~/OneDrive/Escritorio/DISRUP3/Proyectos/Collection-ERC721-MrCrypto/MrCrypto-Smart-Contracts (main)
$ npx hardhat run scripts/deployMockERC20.ts --network polygonMumbai
Compiled 4 Solidity files successfully (evm target: paris).
Constants written
Address:  0xb3F48c1F4751Db0904a69f4fAA4b569Fe68d3a8F

lajar@LAPTOP-EKPT3ELQ MINGW64 ~/OneDrive/Escritorio/DISRUP3/Proyectos/Collection-ERC721-MrCrypto/MrCrypto-Smart-Contracts (main)
$ npx hardhat verify --network polygonMumbai 0xb3F48c1F4751Db0904a69f4fAA4b569Fe68d3a8F "TokenMRC" "TMRC"
Successfully submitted source code for contract
contracts/MrCrypto.sol:MrCrypto at 0xb3F48c1F4751Db0904a69f4fAA4b569Fe68d3a8F
for verification on the block explorer. Waiting for verification result...

Successfully verified contract MrCrypto on the block explorer.
https://mumbai.polygonscan.com/address/0xb3F48c1F4751Db0904a69f4fAA4b569Fe68d3a8F#code
```

Figura A.II.13: Verificación de un Smart Contract

Anexo III
Test de prueba

En este anexo, nos centraremos en los tests de prueba que se han realizado para validar el funcionamiento del smart contract desarrollado para el marketplace de NFTs del presente proyecto. El proceso de pruebas es una fase importante para garantizar que todas las funcionalidades del smart contract operen correctamente bajo diferentes condiciones y escenarios. Aquí se explican en detalle algunos de los tests más significativos y mencionaremos otros para dar una visión completa del alcance de las pruebas realizadas.

Además, se incluye la captura de pantalla de la figura A.III.1, que muestra el resultado exitoso de todos los tests realizados, confirmando que han pasado satisfactoriamente y que el smart contract está listo para ser implementado de manera segura en la producción.

Para iniciar los tests de nuestro smart contract, comenzamos con la simulación de un despliegue completo que incluye el smart contracts del marketplace, así como smart contracts de prueba como MockERC721 y un token MockERC20, "*mock*" quiere decir que son tokens simulados. Además, se generan varias direcciones blockchain para simular interacciones entre diferentes usuarios. Los detalles específicos de este proceso de configuración y simulación se encuentran descritos en el algoritmo A.III.1.

```
E7LMarketplace
  Listing actions
    ✓ List with price 0 => Pass if reverted with PriceMustBeAboveZero error (
    ✓ List item not approved by nft contract => Pass if reverted with NotAppro
    ✓ List item (46ms)
    ✓ List item already listed => Pass if reverted with AlreadyListed error (4
    ✓ List item with paytoken not in whitelist (38ms)
    ✓ List with address(0) => Pass if list success
    ✓ List item with tokenId = 0 => Pass if reverted with TokenIdNotValid erro
    ✓ List item not owner => Pass if reverted with NotOwner() error
  Cancel actions
    ✓ Cancel item (48ms)
    ✓ Cancel item not listed => Pass if reverted with NotListed() error
    ✓ Cancel item not owner => Pass if reverted with NotOwner() error (46ms)
  Modify actions
    ✓ Modify item => Pass if modify success and price equals 50 (59ms)
    ✓ Modify item with price = 0 => Pass if revert with PriceMustBeAboveZero e
    ✓ Modify item with payTokenAddress = address(0) => Pass if success (41ms)
    ✓ Modify item with tokenId = 0 => Pass if revert with NotListed error
    ✓ Modify item not owner => Pass if revert with NotOwner error
  Buy actions
    ✓ Buy item => Pass if emit ItemBought event and add1 erc720 = 50 and add1
    ✓ Buy item with less price send it => Pass if reverted with ERC20: insuffi
    ✓ Buy item with tokenId = notexists => Pass if reverted with NotListed err
    ✓ Buy item not listed => Pass if reverted
    ✓ Buy item listed with address(0) => Pass if revert with AddressZeroNotVal
    ✓ Buy item listed with address(0) and less value sended - native token=> P
  Fallback and Receive
    ✓ Receive (78ms)
    ✓ Fallback (67ms)
  Configuration actions
    ✓ Set recipient
    ✓ Set marketplace fee
  Bid ups actions
    ✓ Bidding for an item with an ERC20 token (52ms)
    ✓ Bidding for an item with Native token=> Pass if success
    ✓ Bidding for an item with Native token with less value => Pass if reverte
    ✓ Bidding for an item with Native token 2 times => Pass if reverted with I
    ✓ Accept bidding with ERC20 token (77ms)
    ✓ Accept bidding with Native token (63ms)
    ✓ Accept bidding of a NTF with any bids => Pass if revert with NotBidded e
    ✓ Bidding for an item not approved by nft contract => Pass if reverted wit
    ✓ Bidding for an item with a lower bid than the current bid => Pass if rev
    ✓ Bidding with a wrong ERC20 => Pass if reverted with PaytokenNotValid err
    ✓ Cancel bidding of a NTF => Pass if reverted with NotOwnerOrAdmin error
    ✓ Cancel bidding of a NTF => Pass if success


38 passing (5s)
```

Figura A.III.1: Tests pasados satisfactoriamente

Algoritmo A.III.1: Simulación de despliegues de smart contracts

```
describe("E7LMarketplace", function () {
  async function deployE7LMarketPlaceContract() {
    const nameERC20 = "Mr.Crypto";
    const symbolERC20 = "MRC";
    const nameERC721 = "Mr.Crypto";
    const symbolERC721 = "MRC";
    const supplyERC721 = 10000;

    const [owner, add1, add2, add3, add4] = await ethers.getSigners();

    const MockERC721 = await ethers.getContractFactory("MrCrypto");
    const mockERC721 = await MockERC721.deploy(nameERC20, symbolERC20);

    await mockERC721.connect(owner).mintNft();

    const E7LMarketplace =
                          await ethers.getContractFactory("E7LMarketplace");
    const e7LMarketplace =
                           await E7LMarketplace.deploy(mockERC721.address);

    const MockERC20 = await ethers.getContractFactory("MockERC20");
    const mockERC20 = await MockERC20.deploy(
                                  nameERC721, symbolERC721, supplyERC721, );
    const mockERC20_2 = await MockERC20.deploy(
                                  nameERC721, symbolERC721, supplyERC721, );

    await e7LMarketplace.setPaytokenWhitelist(mockERC20.address, true);

    await mockERC20.transfer(add1.address, 100);

    return {e7LMarketplace, mockERC20, mockERC721,
            owner, add1, add2, add3, add4, };
  }
}
```

Tras la configuración, se procede a la creación de tests, los cuales se organizan por categorías para mejorar la claridad. Estas categorías incluyen aspectos del funcionamiento del marketplace, como la creación de listados (listing), cancelación de listados (cancel), modificación de detalles (modify), compra de NFTs (buy), manejo de operaciones no anticipadas (fallback), configuraciones del smart contract (configurations), y el proceso de pujas (bid ups). Cada categoría trata diferentes funcionalidades y escenarios que el smart contract puede enfrentar. La estructura de estos tests se pueden visualizar en la figura del A.III.1, donde se muestra cómo se organizan.

Cada test individual se define con la estructura ***it("title"){};***, que describe la acción que el test va a verificar. Cuando se ejecutan los tests, todos los incluidos en el archivo se corren secuencialmente. Sin embargo, si se necesita omitir la ejecución de un test específico sin eliminarlo del código, se puede utilizar ***it.skip***, para pasarlo por alto.

AIII.1.- Categoría Listar

En el algoritmo A.III.2 se detalla cómo realizar un test en Hardhat para asegurar que un ítem ya listado en el marketplace de NFTs no pueda ser listado nuevamente.

Algoritmo A.III.2: Test categoría listar

```
it("List item already listed =>
          Pass if reverted with AlreadyListed error", async function () {

  const { e7LMarketplace, mockERC20, owner, mockERC721 } =
                        await loadFixture(deployE7LMarketPlaceContract);

  await mockERC721.approve(e7LMarketplace.address, 1);
  await e7LMarketplace.connect(owner).listItem(mockERC20.address, 1, 23);
  await expect(
       e7LMarketplace.connect(owner).listItem(mockERC20.address, 1, 22),
     ).to.be.revertedWithCustomError(e7LMarketplace, "AlreadyListed");
});
```

La función ***loadFixture(deployE7LMarketPlaceContract)*** permite reutilizar un entorno inicializado para múltiples pruebas. La función configura y devuelve instancias del contrato del marketplace (e7LMarketplace), un token ERC20 (mockERC20), el propietario del contrato (owner), y un token ERC721 (mockERC721).

Antes de listar un ítem en el marketplace, el contrato ERC721, que representa el NFT, debe autorizar al marketplace para manejar el token en nombre del propietario ejecutando el comando ***mockERC721.approve(e7LMarketplace.address, 1)***, donde 1 es el ID del NFT.

El método ***listItem*** del contrato e7LMarketplace lista el ítem por primera vez. El propietario realiza esta acción conectándose al contrato (connect(owner)), especificando la dirección del token ERC20 (mockERC20.address), el ID del NFT (1), y el precio establecido (23). Intentamos listar el mismo ítem nuevamente con un precio modificado (22), esperando que el contrato del marketplace impida esta acción por ya estar listado.

El test confirma que cualquier intento de listar nuevamente el mismo ítem sea efectivamente bloqueado y revertido, asegurando que la transacción sea revertida con el error específico ***AlreadyListed***.

Otros tests de la categoría Listar son:

- List with price 0: Verifica que el intento de listar un artículo a un precio de cero sea rechazado con un error específico ***PriceMustBeAboveZero***, asegurando que todos los listados tengan un precio válido y mayor que cero.

- List item not approved by nft contract: Comprueba que un ítem no pueda ser listado si el contrato del NFT no ha otorgado aprobación previa al marketplace. Espera que la operación sea revertida con el error ***NotApprovedForMarketplace***.

- List item: Simplemente verifica que un ítem puede ser listado correctamente en el marketplace y espera que este evento emita un ***ItemListed***, confirmado que el listado fue exitoso.

- List item with paytoken not in whitelist: Verifica que un intento de listado usando un token de pago que no está en la whitelist sea revertido con el error ***PayTokenAddressNotValid***, asegurando que solo se usen tokens aprobados.

- List with address(0): Examina si listar un ítem usando la dirección cero (que normalmente es un indicador de un error o dirección no válida en contratos de Ethereum) se ejecuta sin ser revertido, mostrando flexibilidad o un manejo especial para esta dirección.

- List item with tokenId = 0: Asegura que el intento de listar un ítem con un ID de token igual a cero sea rechazado con un error ***TokenIdNotValid***, manteniendo la integridad de los identificadores válidos en el sistema.

- List item not owner: Comprueba que un usuario que no es el propietario de un NFT no pueda listar ese NFT en el mercado, esperando que la transacción sea revertida con el error ***NotOwner***, reforzando la seguridad y la propiedad adecuada de los activos listados.

AIII.2.- Categoría Cancelar

El test del algoritmo A.III.3 se centra en validar que solo el propietario de un ítem listado en el mercado de NFTs puede cancelar el listado. El algoritmo muestra cómo se realiza esta verificación utilizando un smart contract de marketplace y un token ERC721 simulado.

Después de realizar los primeros pasos como el despliegue y el listado, el siguiente paso intenta cancelar el listado por parte de un usuario que no es el propietario (add1), lo cual no debería estar permitido. El método ***cancelListing(1)*** es llamado por add1, intentando cancelar el listado del ítem con ID 1.

El test espera que esta acción sea revertida con un error específico ***NotOwner***, que indica que la operación de cancelación fue intentada por alguien que no es el propietario del ítem.

Algoritmo A.III.3: Test categoría cancelar

```
it("Cancel item not owner =>
          Pass if reverted with NotOwner() error", async function () {
    const { e7LMarketplace, mockERC20, owner, mockERC721, add1 } =
                          await loadFixture(deployE7LMarketPlaceContract);

    await mockERC721.approve(e7LMarketplace.address, 1);
    await e7LMarketplace.connect(owner)
                                     .listItem(mockERC20.address, 1, 23);
    await expect(e7LMarketplace.connect(add1).cancelListing(1),)
       .to.be.revertedWithCustomError(e7LMarketplace, "NotOwner");
});
```

Otros tests de la categoría cancelar:

- Cancel item: Verifica que un ítem listado pueda ser cancelado exitosamente por el propietario, y espera que esta acción emita un evento ***ItemCanceled***, confirmando que la cancelación fue procesada correctamente.

- Cancel item not listed: Prueba que intentar cancelar un ítem que no está listado resulte en una falla. Espera que la transacción sea revertida, asegurando que solo ítems activamente listados puedan ser cancelados.

AIII.3.- Categoría Modificar

El test del algoritmo A.III.4 verifica el funcionamiento correcto del método para modificar el precio de un ítem listado.

Algoritmo A.III.4: Test categoría modificar

```
it("Modify item =>
          Pass if modify success and price equals 50", async function () {
    const { e7LMarketplace, mockERC20, owner, mockERC721 } =
                            await loadFixture(deployE7LMarketPlaceContract);

    const newPrice = 50;
    await mockERC721.approve(e7LMarketplace.address, 1);
    await e7LMarketplace.connect(owner).listItem(mockERC20.address, 1, 23);

    await expect(
      e7LMarketplace.connect(owner)
                    .modifyItem(mockERC20.address, 1, newPrice),
      ).to.emit(e7LMarketplace, "ItemListed");

    expect((await e7LMarketplace.listings(1)).price).to.be.equal(newPrice);
});
```

Tras listar un NFT, se modifica el precio del ítem listado utilizando el método ***modifyItem***. El propietario del ítem (owner) llama a ***modifyItem(mockERC20.address, 1, newPrice)***, donde newPrice es el nuevo precio establecido a 50.

Se espera que la modificación emita un evento ***ItemListed*** y se verifica que el precio del ítem listado haya sido actualizado a 50. La verificación se realiza consultando el precio directamente del contrato con ***await e7LMarketplace.listings(1)).price*** y comparándolo con ***newPrice*** para confirmar que el cambio ha sido efectivo.

Otros tests de la categoría modificar:

- Modify item with price = 0: Comprueba que el intento de modificar el precio de un ítem listado a cero resulte en una falla. El test espera que la transacción sea revertida con el error ***PriceMustBeAboveZero***, asegurando que todos los precios sean válidos y mayores que cero.

- Modify item with payTokenAddress = address(0): Verifica que el cambio de la dirección del token de pago a la dirección cero (utilizada para indicar pago en la criptomoneda nativa de la red, como ETH en Ethereum) sea procesado sin revertir. Esto permite que los usuarios opten por recibir pagos en la moneda nativa del blockchain.

- Modify item with tokenId = 0: Prueba que el intento de modificar un ítem con un ID de token igual a cero sea rechazado. Espera que la acción sea revertida con el error ***TokenIdNotValid***, manteniendo la integridad y la validez de los identificadores de tokens en el sistema.

- Modify item not owner: Verifica que la modificación de un listado por alguien que no es el propietario del ítem resulte en una falla. Espera que la transacción sea revertida, asegurando que solo el propietario pueda modificar los detalles del listado.

AIII.4.- Categoría Comprar

El test del algoritmo A.III.5 verifica que el smart contract del marketplace revierte una transacción de compra si el monto enviado es insuficiente respecto al precio listado del NFT.

Un segundo usuario (add1) intenta comprar el NFT aprobando un monto (***SENDED_AMOUNT***) que es menor que el precio listado. ***SENDED_AMOUNT*** se

establece en 25, que es insuficiente para cubrir el costo del NFT. El usuario aprueba este monto para el marketplace con el comando de las líneas 11 y 12 ***mockERC20.connect(add1).approve(e7LMarketplace.address, SENDED_AMOUNT)***.

Algoritmo A.III.5: Test categoría comprar

```
 1 it("Buy item with less price send it => Pass if reverted with ERC20:
 2          insufficient allowance: Insufficient amount", async function () {
 3       const { e7LMarketplace, mockERC20, owner, mockERC721, add1 } =
 4                              await loadFixture(deployE7LMarketPlaceContract);
 5
 6       const BUY_AMOUNT = 50;
 7       const SENDED_AMOUNT = 25;
 8       const tokenId = 1;
 9       await mockERC721.approve(e7LMarketplace.address, tokenId);
10       await e7LMarketplace.listItem(mockERC20.address,tokenId,BUY_AMOUNT);
11       await mockERC20
12             .connect(add1).approve(e7LMarketplace.address, SENDED_AMOUNT);
13       await expect(
14         e7LMarketplace.connect(add1).buyItem(tokenId, { value:
15                                                          SENDED_AMOUNT }),
16       ).to.be.revertedWith("ERC20: insufficient allowance");  });
```

Cuándo add1 intenta realizar la compra (***e7LMarketplace.connect(add1).buyItem(tokenId, { value: SENDED_AMOUNT })***), el test espera que la transacción sea revertida con el mensaje "***ERC20: insufficient allowance***", indicando que el monto enviado no es suficiente para completar la compra.

Otros tests de la categoría comprar:

- Buy item: Verifica que un comprador pueda comprar un ítem exitosamente. Primero, se aprueba el transfer del NFT y se lista en el marketplace. Luego, el comprador aprueba la cantidad necesaria de ERC20 y realiza la compra. El test espera que se emita el evento ***ItemBought*** y verifica que el saldo del token ERC20 del comprador sea de 50 y que posea 1 token ERC721.

- Buy item with tokenId = not exists: Verifica que intentar comprar un ítem con un ID de token que no existe sea revertido con el error ***NotListed***.

- Buy item not listed: Asegura que un intento de comprar un ítem que no ha sido listado resulte en una reversión con el error ***NotListed***, asegurando que solo ítems activamente listados estén disponibles para la compra.

- Buy item listed with address(0): Prueba que la compra de un ítem listado con la dirección cero como token de pago (indicando pago en criptomoneda nativa) no sea revertida, mostrando que el sistema permite esta forma de pago bajo ciertas condiciones.

- Buy item listed with address(0) and less value sent - native token: Verifica que intentar comprar un ítem pagando con criptomoneda nativa enviando menos del valor listado resulte en una reversión con el error ***NotEnoughAmount***, asegurando que se envíe la cantidad completa requerida para la compra.

AIII.5.- Categoría Fallback y configuraciones

El test del algoritmo A.III.6 evalúa el comportamiento del smart contract del marketplace al recibir fondos de manera directa, sin una llamada a método específico, lo que comúnmente se conoce como una transacción "fallback".

Algoritmo A.III.6: Test categoría fallback y configuraciones

```
it("Fallback", async function () {
    const { e7LMarketplace, mockERC20, owner, mockERC721, add1 } =
                          await loadFixture(deployE7LMarketPlaceContract);
     const balanceBefore=await ethers.provider.getBalance(owner.address);
     const tx = {
       to: e7LMarketplace.address,
       data: ethers.utils.formatBytes32String("Mr. Crypto"),
       value: ethers.utils.parseEther("1.0"),
     };

     await add1.sendTransaction(tx);
     const balanceAfter= await ethers.provider.getBalance(owner.address);
     expect(balanceAfter.sub(balanceBefore) ==
                                       ethers.utils.parseEther("1.0"))
});
```

El usuario add1 prepara una transacción que envía 1.0 Ether directamente a la dirección del smart contract e7LMarketplace (tx). La transacción incluye un pequeño dato (data) para emular un mensaje, aunque este dato no se utilizará debido a la naturaleza de la transacción fallback.

El usuario add1 realiza la transacción y se verifica que el balance de Ether del propietario (owner) del smart contract haya aumentado en exactamente 1.0 Ether. La verificación se realiza comparando el balance antes y después de la transacción, asegurando que la diferencia sea la cantidad enviada.

Otros tests de la categoría fallback y configuraciones:

- Receive: Verifica que el smart contract pueda recibir ETH correctamente y que el balance del propietario del smart contract se incremente en 1 ETH como resultado de la transacción directa sin datos adicionales.

- Set Recipient: Verifica la funcionalidad del smart contract para actualizar la dirección del destinatario de los fondos o pagos dentro del smart contract, comprobando que la nueva dirección se almacena correctamente.

- Set Marketplace Fee: Evalúa la capacidad para ajustar la tarifa del marketplace, estableciendo y verificando que la nueva tarifa se configure correctamente.

AIII.6.- Categoría Pujas

El test del algoritmo A.III.7 verifica la funcionalidad de aceptar una oferta (puja) en el marketplace utilizando un token ERC20 como moneda de pago.

Algoritmo A.III.7: Test categoría pujas

```
it("Accept bidding with ERC20 token", async function () {
   const { e7LMarketplace, mockERC20, owner, mockERC721, add1 } =
           await loadFixture(deployE7LMarketPlaceContract);

   const priceBidded = 50;
   const priceListed = 60;
   const tokenId = 1;

   await mockERC721.approve(e7LMarketplace.address, tokenId);
   await e7LMarketplace.connect(owner)
                       .listItem(mockERC20.address, tokenId, priceListed);
   await mockERC20.connect(add1)
                  .approve(e7LMarketplace.address, priceBidded);
   await e7LMarketplace.connect(add1)
                       .makeBidUp(mockERC20.address, tokenId, priceBidded);

   await expect(e7LMarketplace.connect(owner).acceptBidUp(tokenId))
                               .to.emit(e7LMarketplace, "ItemBetAccepted",);
   expect(await mockERC20.connect(add1).balanceOf(add1.address))
                                       .equal(50);
   expect(await mockERC721.connect(add1).balanceOf(add1.address))
                                        .equal(1);
});
```

El propietario aprueba que el marketplace maneje un NFT específico y luego lo lista en el por un precio (***priceListed = 60***).

El usuario add1 aprueba que el marketplace pueda usar una cantidad específica de tokens ERC20 (***priceBidded = 50***) y hace una puja por el NFT listado. Aunque la puja es menor que el precio listado, se procede a evaluar si el smart contract maneja correctamente esta situación.

El propietario acepta la puja hecha por add1. Este paso se verifica para asegurar que el evento ***ItemBetAccepted*** sea emitido, lo cual indica que la transacción fue procesada correctamente.

Finalmente, se verifica que el saldo de tokens ERC20 de add1 se haya reducido adecuadamente y que ahora posea el NFT, asegurando que la transferencia de la propiedad del token y del pago se realizó correctamente.

Otros tests de la categoría pujas:

- Bidding for an item with Native token - Success: Asegura que un usuario pueda ofertar exitosamente usando el token nativo de la red (como ETH). Verifica que la transacción emita el evento ***ItemBetted*** al completar la oferta.

- Bidding for an item with Native token with less value: Prueba que intentar hacer una oferta con un valor menor al necesario resulte en un error ***NotEnoughAmount***, asegurando que los usuarios envíen el monto completo requerido.

- Bidding for an item with Native token 2 times: Verifica que hacer dos ofertas seguidas con token nativo ajuste correctamente los balances y que la segunda oferta con mayor valor se procese adecuadamente sin emitir errores.

- Accept bidding with ERC20 token: Comprueba que el propietario de un NFT pueda aceptar una oferta realizada con un token ERC20, verificando que la transacción emita ***ItemBetAccepted*** y que los balances de tokens y la propiedad del NFT se actualicen correctamente.

- Accept bidding with Native token: Evalúa que una oferta hecha con el token nativo pueda ser aceptada por el propietario del NFT, verificando que se emita el evento ***ItemBetAccepted*** y que el balance del propietario aumente por el monto ofertado.

- Accept bidding of an NFT with no bids: Verifica que intentar aceptar una oferta en un NFT que no ha recibido ofertas previas resulte en un error ***NotBidded***, manteniendo la integridad del proceso de ofertas.

- Bidding for an item not approved by NFT contract: Asegura que una oferta sea rechazada si el NFT no ha sido previamente aprobado para transacciones en el marketplace, esperando el error ***PaymentNotApprovedForMarketplace***.

- Bidding for an item with a lower bid than the current bid: Confirma que una oferta menor a la actual será rechazada, esperando el error ***LowerThanCurrentBid***, para que solo se acepten ofertas que igualen o superen la oferta más alta actual.

- Bidding with a wrong ERC20 token: Verifica que usar un token ERC20 no válido o no aprobado en la whitelist resulte en un error ***PaytokenNotValid***, asegurando que solo se utilizan tokens autorizados en las ofertas.

- Cancel bidding of an NFT - Not Owner or Admin: Evalúa que intentar cancelar una oferta por alguien que no es el propietario ni administrador del NFT sea rechazado, esperando un error ***NotOwnerOrAdmin***, garantizando que solo personas autorizadas puedan cancelar ofertas.

- Cancel bidding of an NFT - Success: Verifica que un intento de cancelar una oferta por parte de alguien autorizado (como el dueño) resulte en la cancelación exitosa de la oferta, esperando que se emita ***ItemBetCanceled***.

Anexo IV
Casos de uso

AIV.1.- USUARIO NO IDENTIFICADO

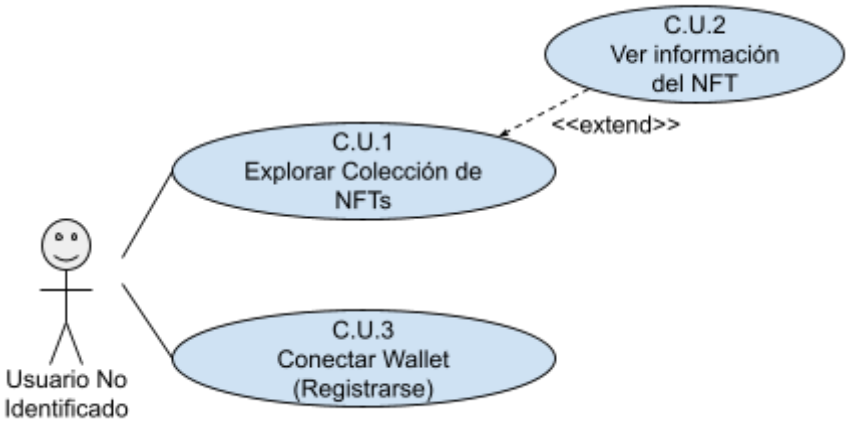

Figura AIV.1: Casos de Uso: Usuario no Identificado

Tabla AIV.1. - C.U. 1: Explorar Colección de NFT

C.U. 1	Explorar Colección de NFT
Actores	Usuario No Identificado
Descripción	El usuario explora la colección de NFT sin necesidad de registrarse.
Precondición	Acceso a la plataforma.
Secuencia Normal	El usuario accede a la página principal. El usuario navega por el explorador de NFTs.
Poscondición	El usuario obtiene información sobre los NFTs.
Excepciones	No aplicable, ya que no se requiere registro.
Comentarios	Este caso de uso solo permite visualizar la colección, sin interacción.

Tabla AIV.2. - C.U. 2: Ver Información del NFT

C.U. 2	Ver Información del NFT
Actores	Usuario No Identificado
Descripción	El usuario visualiza información detallada de un NFT específico, incluyendo detalles como el ID, propietario actual y E7L asociados, sin necesidad de estar identificado o conectado.
Precondición	Acceso a la plataforma y navegar hasta un NFT específico.
Secuencia Normal	El usuario realiza click sobre un NFT en la sección de exploración. Se muestra una página con información detallada del NFT seleccionado.
Poscondición	El usuario ha visualizado la información detallada del NFT sin modificar ningún estado en la blockchain o en la aplicación.
Excepciones	No aplicable, ya que no se requiere registro.
Comentarios	Este caso de uso permite visualizar la información específica de cada NFT, pero no permite interactuar con los smart contracts.

Tabla AIV.3. - C.U. 3: Conectar Wallet (Registrarse)

C.U. 3	Conectar Wallet (Registrarse)
Actores	Usuario No Identificado
Descripción	El usuario crea automáticamente su perfil al conectar su wallet.
Precondición	Tener una wallet.
Secuencia Normal	El usuario selecciona la opción "Conectar Wallet". El usuario elige su proveedor de wallet y concede permisos.
Poscondición	El usuario queda identificado y conectado en la plataforma.
Excepciones	Error en la conexión con la wallet.
Comentarios	Este paso es necesario para participar en la compra o listado de NFTs.

AIV.2.- USUARIO IDENTIFICADO

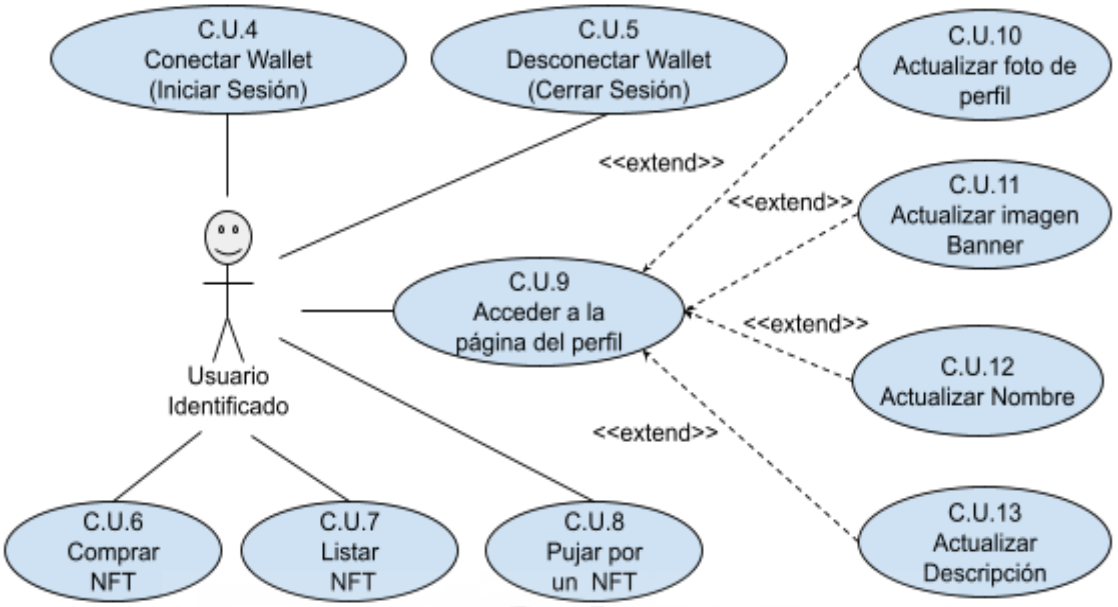

Figura AIV.2: Casos de Uso: Usuario Identificado

Tabla AIV.4. - C.U.4: Conectar Wallet (Iniciar Sesión)

C.U. 4	Conectar Wallet (Iniciar Sesión)
Actores	Usuario Identificado
Descripción	El usuario inicia sesión en la aplicación conectando su wallet, lo que le permite acceder a su perfil y a todas las funcionalidades disponibles para usuarios identificados.
Precondición	El usuario debe estar previamente identificado.
Secuencia Normal	El usuario selecciona la opción de "Conectar Wallet" en la plataforma. El usuario elige su proveedor de wallet y concede los permisos necesarios para la conexión. La aplicación verifica la wallet y establece la sesión del usuario.
Poscondición	El usuario queda conectado a la plataforma con acceso completo a su perfil y funcionalidades exclusivas para usuarios identificados.
Excepciones	Error en la conexión con la wallet debido a problemas de red. La wallet no es reconocida o no está soportada por la plataforma.
Comentarios	Este proceso es necesario para poder realizar los siguientes casos de uso.

Tabla AIV.5. - C.U.5: Desconectar Wallet (Cerrar Sesión)

C.U. 5	Desconectar Wallet (Cerrar Sesión)
Actores	Usuario Identificado
Descripción	El usuario cierra la sesión en la aplicación, desconectando su wallet.
Precondición	El usuario debe estar previamente conectado a la plataforma.
Secuencia Normal	El usuario selecciona la opción de "Cerrar Sesión" o "Desconectar Wallet" en la plataforma. La aplicación confirma la solicitud y desconecta la wallet del usuario.
Poscondición	El usuario queda desconectado de la plataforma, perdiendo acceso a su perfil y a las funcionalidades reservadas para usuarios identificados hasta que vuelva a iniciar sesión.
Excepciones	Error al cerrar sesión por problemas técnicos o de red.
Comentarios	Mantiene la seguridad de la cuenta del usuario, permitiéndole controlar cuándo y cómo finaliza su sesión en la plataforma.

Tabla AIV.6. - C.U.6: Comprar NFT

C.U. 6	Comprar NFT
Actores	Usuario Identificado
Descripción	El usuario compra un NFT de la colección utilizando su wallet.
Precondición	Wallet conectada y saldo suficiente.
Secuencia Normal	El usuario selecciona el NFT deseado. El usuario confirma la transacción en su wallet.
Poscondición	El NFT se transfiere al usuario.
Excepciones	Transacción fallida o saldo insuficiente.
Comentarios	El NFT adquirido aparece en el perfil del usuario.

Tabla AIV.7. - C.U.7: Listar NFT (Poner en venta)

C.U. 7	Listar NFT (Poner en venta)
Actores	Usuario Identificado
Descripción	El usuario pone a la venta un NFT de su propiedad.
Precondición	Poseer un NFT elegible para la venta.
Secuencia Normal	El usuario selecciona el NFT a vender. El usuario establece el precio y confirma el listado.
Poscondición	El NFT aparece listado en la plataforma.
Excepciones	Error al listar el NFT.
Comentarios	Los usuarios interesados pueden ver y comprar el NFT listado.

Tabla AIV.8 - C.U.8: Pujar por un NFT

C.U. 8	Pujar por un NFT
Actores	Usuario Identificado
Descripción	El usuario realiza una oferta económica (puja) por un NFT que está en la plataforma.
Precondición	El usuario debe estar identificado y haber iniciado sesión mediante la conexión de su wallet.
Secuencia Normal	El usuario navega por la colección de NFTs y selecciona. El usuario introduce el monto de su puja. El usuario confirma la puja, autorizando la transacción desde su wallet. La plataforma registra la puja del usuario y actualiza el estado de la oferta.
Poscondición	La puja del usuario queda registrada en la plataforma.
Excepciones	Problemas de conexión con la wallet o con la red blockchain que impiden registrar la puja. Saldo insuficiente.
Comentarios	

Tabla AIV.9. - C.U.9: Acceder al su página de perfil

C.U. 9	Acceder al su página de perfil
Actores	Usuario Identificado
Descripción	El usuario accede a su perfil, visualizando su información.
Precondición	El Usuario debe estar identificado y conectado a su cuenta.
Secuencia Normal	El usuario hace click en el botón para acceder a su perfil. El sistema redirige al usuario a su página de perfil. El usuario visualiza la información personal, así como los NFTs que posee.
Poscondición	El usuario ha accedido a su perfil y ha podido visualizar su información y NFTs.
Excepciones	El usuario no está Identificado o no ha iniciado sesión.
Comentarios	La página de perfil es un componente para visualizar la información personal de cada usuario.

Tabla AIV.10. - C.U.10: Actualizar foto de perfil

C.U. 10	Actualizar foto de perfil
Actores	Usuario Identificado
Descripción	El usuario actualiza la foto de su perfil en la plataforma.
Precondición	El usuario debe estar identificado y conectado a su cuenta.
Secuencia Normal	El usuario selecciona la opción para editar su perfil. El usuario elige una nueva foto de perfil y la carga. La plataforma actualiza el perfil del usuario con la nueva foto.
Poscondición	La foto de perfil del usuario ha sido actualizada.
Excepciones	Problemas al cargar la foto por formatos no compatibles o errores de red.
Comentarios	Opción para mantener el perfil de usuario actualizado y personalizado.

Tabla AIV.11. - C.U.11: Actualizar imagen banner del perfil

C.U. 11	Actualizar imagen banner del perfil
Actores	Usuario Identificado
Descripción	El usuario reemplaza el banner de su perfil por uno nuevo.
Precondición	El usuario debe estar identificado y conectado a su cuenta.
Secuencia Normal	En la opción de edición de perfil, el usuario selecciona cambiar banner. El usuario sube una nueva imagen para el banner y confirma. La plataforma actualiza el banner en el perfil del usuario.
Poscondición	El banner del perfil del usuario ha sido actualizado.
Excepciones	Error al subir el banner por formato incorrecto o fallo de conexión.
Comentarios	El cambio de banner es una forma de personalización del perfil que mejora la experiencia visual en la plataforma.

Tabla AIV.12. - C.U.12: Actualizar nombre de usuario

C.U. 12	Actualizar nombre de usuario
Actores	Usuario Identificado
Descripción	El usuario reemplaza el nombre de su perfil por uno nuevo.
Precondición	El usuario debe estar identificado y conectado a su cuenta.
Secuencia Normal	En la opción de edición de perfil, el usuario selecciona cambiar nombre. El usuario escribe un nuevo nombre y confirma. La plataforma actualiza el nombre en el perfil del usuario.
Poscondición	El nombre del perfil del usuario ha sido actualizado.
Excepciones	Error al guardar los cambios.
Comentarios	Opción para diferenciar perfiles de forma común (no mediante el address)

Tabla AIV.13. - C.U.13: Actualizar descripción del perfil

C.U. 13	Actualizar descripción del perfil
Actores	Usuario Identificado
Descripción	El usuario reemplaza la descripción de su perfil por uno nuevo.
Precondición	El usuario debe estar identificado y conectado a su cuenta.
Secuencia Normal	Desde la opción de edición de perfil, el usuario selecciona cambiar su descripción. El usuario escribe una nueva descripción y confirma. La plataforma actualiza la descripción en el perfil del usuario.
Poscondición	La descripción del perfil del usuario ha sido actualizado.
Excepciones	Error al guardar los cambios.
Comentarios	La descripción del usuario permite que el usuario añada algo que él considere importante a su perfil.

AIV.3.- PROPIETARIO DE SMART CONTRACT

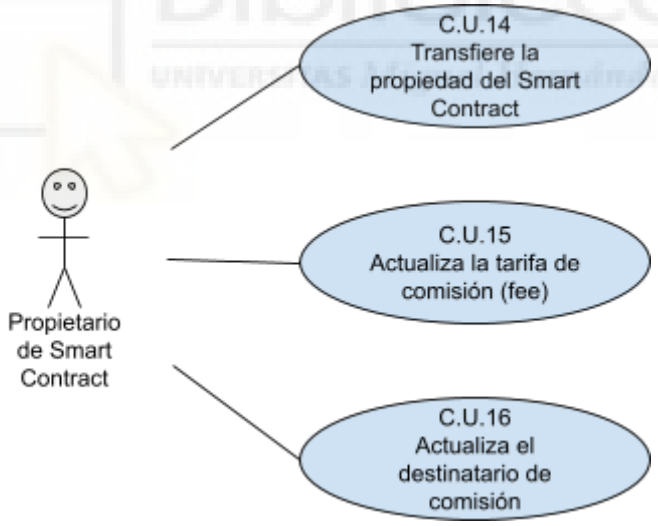

Figura AIV.3: Casos de Uso: Propietario del smart contract

Tabla AIV.14. - C.U.14: Transfiere la propiedad del Smart Contract

C.U. 14	Transfiere la propiedad del Smart Contract
Actores	Propietario Smart Contract
Descripción	El propietario actual del smart contract transfiere la propiedad a otra dirección (address) .
Precondición	El actor debe ser el actual propietario del smart contract y debe tener acceso a la función de transferencia de propiedad.
Secuencia Normal	Acceder al contrato con un explorador de bloques como Etherscan. Navega hasta la sección de 'Write Contract'. Se conecta utilizando su wallet que tiene control del contrato. Ingresar la dirección de la wallet del nuevo propietario en la función apropiada para transferir la propiedad. Ejecuta la transacción, que es enviada a la blockchain para procesarse.
Poscondición	La propiedad del smart contract ha sido transferida al nuevo propietario.
Excepciones	Error de transacción si la dirección de la wallet es incorrecta. Fallo al ejecutar si el usuario actual no tiene permisos de propietario.
Comentarios	Esta operación permite cambiar el control del contrato a otro usuario, asegurando la continuidad en la gestión y operación del mismo.

Tabla AIV.15. - C.U.15: Actualiza la tarifa de comisión (fee)

C.U. 15	Actualiza la tarifa de comisión (fee)
Actores	Propietario Smart Contract
Descripción	El propietario actualiza la tarifa de comisión aplicable a las transacciones realizadas a través del smart contract.
Precondición	El actor debe ser el actual propietario del smart contract y tener acceso a la función de modificación de tarifas en el contrato.
Secuencia Normal	Accede al contrato a través de un explorador de bloques como Etherscan. Navega hasta la sección de 'Write Contract'. Se conecta utilizando su wallet que tiene control del contrato. Ingresa el nuevo valor de la tarifa de comisión en la función correspondiente para actualizar la tarifa. Ejecuta la transacción, que es enviada a la blockchain para procesarse.
Poscondición	La tarifa de comisión del Smart Contract ha sido actualizada.
Excepciones	Error de transacción si el nuevo valor de la tarifa no cumple con las reglas del contrato. Fallo al ejecutar si el usuario actual no tiene permisos de propietario.
Comentarios	Este cambio asegura que las tarifas de transacción se mantengan alineadas con la política económica o estratégica del propietario del contrato.

Tabla AIV.16. - C.U.16: Actualiza el destinatario de comisión

C.U. 16	Actualiza el destinatario de comisión
Actores	Propietario Smart Contract
Descripción	El propietario del Smart Contract cambia la dirección del destinatario que recibe las comisiones generadas por las transacciones.
Precondición	El actor debe ser el actual propietario del smart contract y tener acceso a la función de modificación de tarifas en el contrato.
Secuencia Normal	El propietario accede al contrato a través de un explorador de bloques como Etherscan. Navega hasta la sección de 'Write Contract'. Se conecta utilizando su wallet que tiene control del contrato. Localiza la función específica para actualizar la dirección del destinatario y proporciona la nueva dirección de wallet. Ejecuta la transacción, que es enviada a la blockchain para procesarse.
Poscondición	La dirección del destinatario para recibir comisiones ha sido actualizada en el contrato
Excepciones	Error si la nueva dirección no es válida o no está en formato de dirección de wallet correcto. Error si la transacción falla por falta de gas o fondos.
Comentarios	Este cambio permite al propietario del contrato redirigir las comisiones a una nueva dirección según necesidades administrativas o de gestión.