COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

ESTUDIO Y DESPLIEGUE DE UNA PLATAFORMA DESCENTRALIZADA DE INTERCAMBIO DE ACTIVOS DIGITALES: NFT TICKETING

Autor: Antonio Capilla Paredes

Director: David Contreras Bárcena

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Estudio y despliegue de una aplicación descentralizada de intercambio de activos digitales: NFT Ticketing

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2021/2022 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Antonio Capilla Paredes Fecha: 20/ 08/ 2022

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: David Contreras Bárcena Fecha: ……/ ……/ ……

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

ESTUDIO Y DESPLIEGUE DE UNA PLATAFORMA DESCENTRALIZADA DE INTERCAMBIO DE ACTIVOS DIGITALES: NFT TICKETING

Autor: Antonio Capilla Paredes

Director: David Contreras Bárcena

Madrid

ESTUDIO Y DESPLIEGUE DE UNA PLATAFORMA DESCENTRALIZADA DE INTERCAMBIO DE ACTIVOS DIGITALES: NFT TICKETING

Autor: Capilla Paredes, Antonio.

Director: Contreras Bárcena, David.

Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

En este Trabajo de Fin de Grado, se ha estudiado y desarrollado una aplicación web para la compra, venta e intercambio de tickets para eventos en formato de criptoactivos digitales. El funcionamiento de la aplicación es totalmente gestionado por una serie de contratos inteligentes, y solo es necesario el despliegue de uno de contratos, en la red blockchain, para que la aplicación comience a funcionar de manera autónoma y descentralizada.

Palabras clave: Blockchain, Contratos Inteligentes, Web 3.0, Ethereum, NFT Ticketing

1. Introducción

Desde el comienzo de la revolución informática, se ha podido observar como distintas industrias han adoptado tecnologías digitales para conseguir mejoras en sus servicios o productos. Este proceso de actualización va acorde con las innovaciones tecnológicas.

Con el auge de las redes Blockchain y los contratos inteligentes, muchas industrias deben realizar un estudio de adaptación para lograr implantar las ventajas que ofrecen estas tecnologías. Una industria que tras la primera adaptación digital vio una lista considerable de ventajas, es la del Ticketing de eventos. Pero con el paso del tiempo y el avance tecnológico que estamos vivimos hoy en día, se esta comenzando a quedar atrás. Es el momento de que las industrias comiencen a actualizarse al ritmo que lo hacen las tecnologías.

2. Definición del Proyecto

Con un motivo resolutor, este proyecto pretende cuestionar el funcionamiento de la gestión de eventos en los tiempos modernos. A través de medidas y herramientas basadas en Web 3.0, la cadenas de bloques y los contratos inteligentes, en este trabajo se han estudiado los defectos de la industria para, así, tomar medidas al respecto y tartar de resolverlos.

Por ello, se ha diseñado e implementado una aplicación web totalmente funcional con las características necesarias para suplir las necesidades del mercado de los de eventos y que hace uso de las tecnologías más innovadoras del mercado. El sistema, además, incorpora

soluciones novedosas, como un mercado secundario regulado para el intercambio de tickets entre usuarios, la inclusión de un archivo genérico único para cada evento en una forma de criptoactivo, etc.

Además, desde el comienzo hasta la finalización el proceso de desarrollo, se tendrá en cuenta al usuario y se estudiará la manera de que el sistema le satisfaga en una mayor medida. Esto verá reflejado en una interfaz de usuario sencilla y estéticamente cuidada, con comunicación eficiente con los sistemas más internos de la aplicación para no perder funcionalidad.

3. Descripción del sistema

El sistema se describirá en dos bloques de componentes. El primero lo conforman una serie de contratos inteligentes desplegados en una red blockchain. Estos contratos serán testeados usando la herramienta de Truffle, e implementan la lógica necesaria para gestionar los eventos: la creación de los mismos, la venta de tickets y el intercambio de tickets entre usuarios. Para interactuar con los contratos que se ejecutan en la blockchain, hace falta de un intermediador que funcione como proveedor de servicios, para eso usaremos la cartera o monedero digital Metamask y las librerías *ethers,js*, *web3modal* y *openzeppelin.* Estas librerías proporcionarán las funciones necesarias para realizar todas las interacciones entre el servidor y los contratos inteligentes.

El segundo bloque lo conforma la llamada Aplicación, que se encarga de gestionar la recepción y donación de información por parte de los servicios de la blockchain. Esta construida sobre Next.js, que es un framework de desarrollo basado en Node,js, que proporciona una serie de funcionalidades de React y permite que se generen páginas dinámicas a través de una técnica llamada Server Side Rendering. Next permite realizar una división en las funciones del servidor, existiendo dos partes: el File Server y el API Server. Estos dos elementos de la aplicación se unirán para crear la página con la que interactuará el usuario, conformando así el Frontend. Además, este Frontend estará estilado con Tailwind.

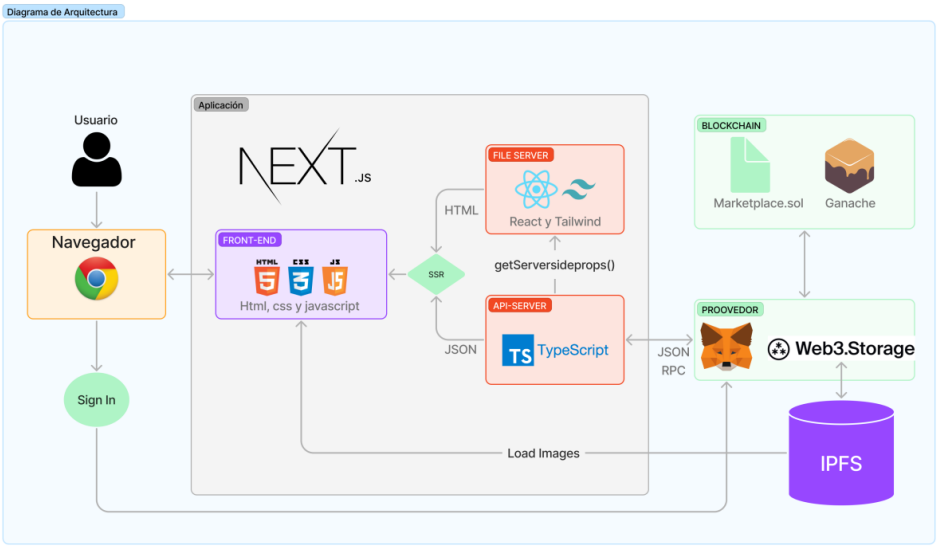

Figura 1. Esquema de la arquitectura del proyecto

4. Resultados

Gracias a un estudio previo y a un proceso de diseño e implementación, se ha logrado elaborar un sistema completamente funcional. Los contratos inteligentes se testearon y realizan correctamente su labor, la aplicación se conecta satisfactoriamente con la red y con los contratos, y muestra correctamente la información que se almacena en dichos componentes.

En la siguiente figura se puede ver la página de "Explorar", donde aparecen los eventos disponibles en el mercado.

Figura 2. Página de Explorar

Si accedemos, desde la barra de navegación, a la página de "Crear Evento", se desplegará un formulario para rellenar con la información necesaria.

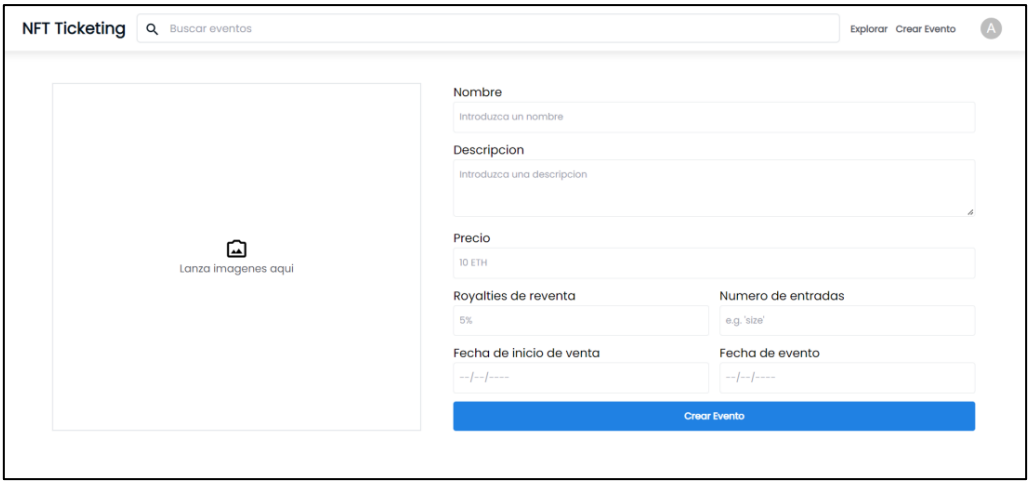

Figura 3. Página de Crear Evento

Y en las siguientes figuras, podemos oberservar el procedimiento para la compra, a tavés del minteo, de un ticket para un evento.

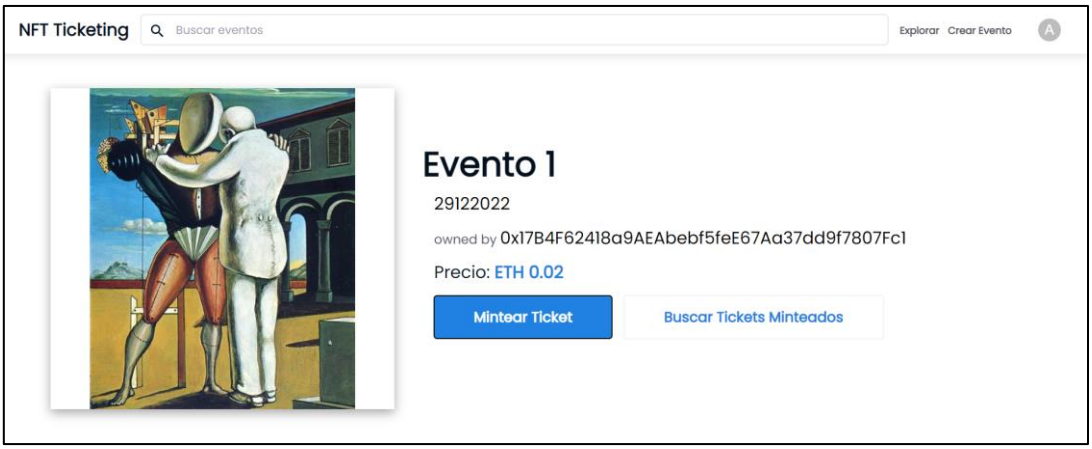

Figura 4. Página de información de un evento

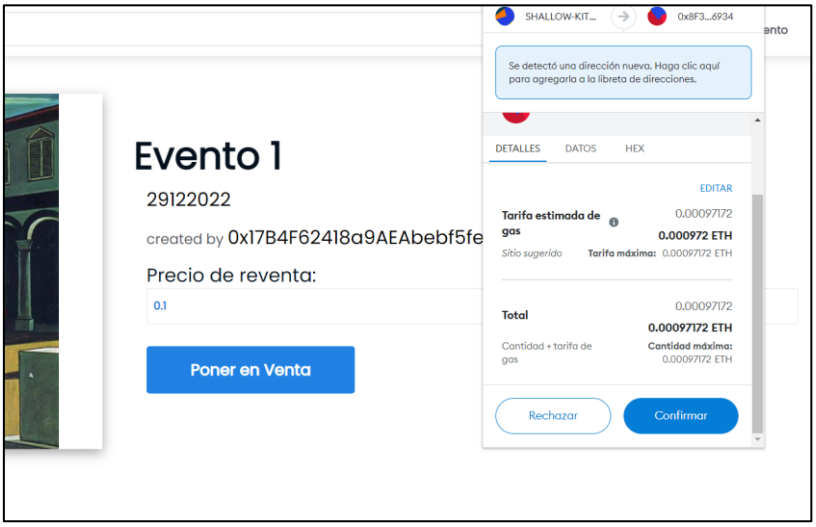

Figura 5. Transferencia correspondiente a la compra de un ticket

Conclusiones

El trabajo realizado cumple con todos los objetivos que se han propuesto y como se ha expuesto en el apartado anterior, con muy buenos resultados. Los contratos diseñados solucionan muchos de los problemas encontrados a la hora de gestionar tickets de un evento. De esta forma, se ha conseguido demostrar, de manera pragmática, que la tecnología blockchain y los contratos inteligentes son útiles para solucionar una serie de problemas que presenta las actuales infraestructuras de industrias digitales.

No obstante, durante el desarrollo de este proyecto se ha identificado una sobre-exaltación por parte de la comunidad sobre dichas tecnologías, desembocando en especulaciones y en otros asuntos que no benefician al verdadero estudio de las mismas. Dicho esto, la tecnología debe seguir siendo investigada y mejorada para que surjan más soluciones como la presentada en este trabajo.

STUDY AND DEPLOYMENT OF A DECENTRALIZED PLATFORM FOR THE EXCHANGE OF DIGITAL ASSETS: NFT TICKETING

Author: Capilla Paredes, Antonio.

Supervisor: Contreras Bárcena, David

Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

In this Final Degree Project, we have studied and developed a web application for the purchase, sale and exchange of tickets for events in the form of digital crypto-assets. The operation of the application is fully managed by a series of smart contracts, and it is only necessary to deploy one of the contracts on the blockchain network for the application to start working autonomously and decentralized.

Keywords: Blockchain, Smart Contracts, Web 3.0, Ethereum, NFT Ticketing

1. Introduction

Since the beginning of the IT revolution, it has been observed how different industries have adopted digital technologies to improve their services or products. This process of upgrading goes hand in hand with technological innovations. With the rise of Blockchain networks and smart contracts, many industries have to carry out an adaptation study in order to implement the advantages offered by these technologies. One industry that saw a considerable list of advantages after the first digital adaptation is the event ticketing industry. But with the passage of time and the technological advances we are experiencing today, it is starting to fall behind. It's time for industries to start catching up with the pace of technology.

2. Proyect Definition

With a resolution motive, this project aims to question the functioning of event management in modern times. Through measures and tools based on Web 3.0, blockchain and smart contracts, this work has studied the shortcomings of the industry in order to take action and try to solve them.

Therefore, a fully functional web application has been designed and implemented with the necessary features to meet the needs of the event market and which makes use of the most

innovative technologies on the market. The system also incorporates innovative solutions, such as a regulated secondary market for the exchange of tickets between users, the inclusion of a unique generic file for each event in a form of crypto-asset, etc.

In addition, from the beginning to the end of the development process, the user will be taken into account and ways of making the system more user-friendly will be considered. This will be reflected in a simple and aesthetically pleasing user interface, with efficient communication with the more internal systems of the application so as not to lose functionality.

3. System Description

The system will be described in two component blocks. The first is a series of smart contracts deployed on a blockchain network. These contracts will be tested using the Truffle tool, and implement the logic necessary to manage events: the creation of events, the sale of tickets and the exchange of tickets between users. To interact with the contracts that are executed on the blockchain, we need an intermediary that functions as a service provider, for which we will use the Metamask digital wallet and the ethers,js, web3modal and openzeppelin libraries. These libraries will provide the necessary functions to perform all the interactions between the server and the smart contracts.

The second block is the so-called Application, which is responsible for managing the reception and donation of information by the blockchain services. It is built on Next.js, which is a development framework based on Node.js, which provides a series of React functionalities and allows dynamic pages to be generated through a technique called Server Side Rendering. Next allows a division to be made in the server functions, with two parts: the File Server and the API Server. These two elements of the application will be joined to create the page with which the user will interact, thus forming the Frontend. In addition, this Frontend will be styled with Tailwind..

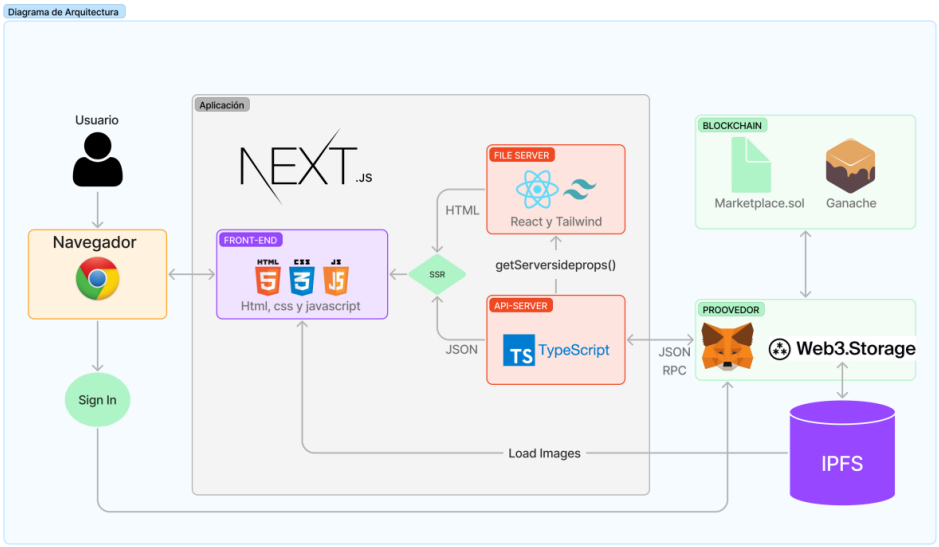

Figure 6. Outline of the project architecture

4. Results

Thanks to a previous study, design and implementation process, a fully functional system has been developed. The smart contracts were tested and performed correctly, the application connects successfully with the network and with the contracts, and correctly displays the information stored in these components.

The following figure shows the "Explore" page, where the events available on the market are displayed.

Figure 7. Page of Explore

If we access, from the navigation bar, the "Create Event" page, a form to fill in with the necessary information will be displayed.

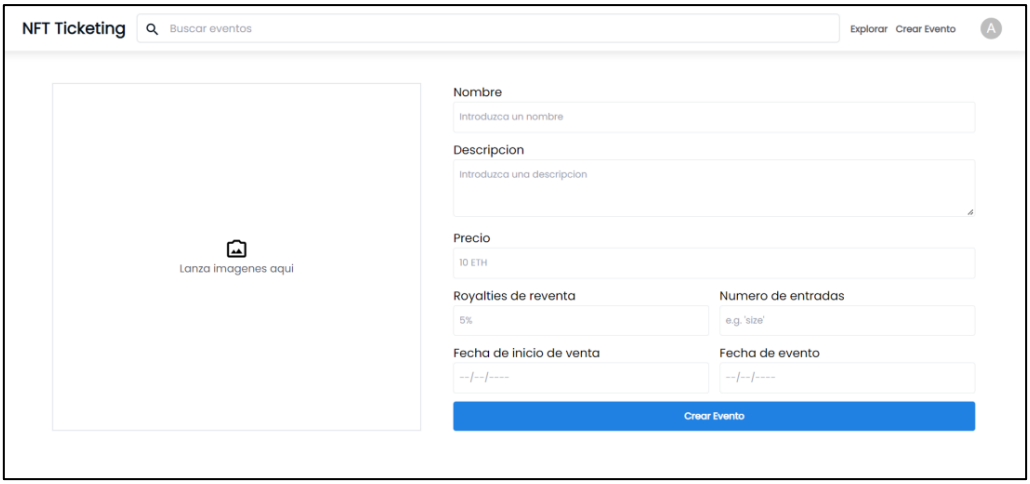

Figure 8. Page of Create Event

And in the following figures, we can see the procedure for the purchase of a ticket for an event through ticket-mining.

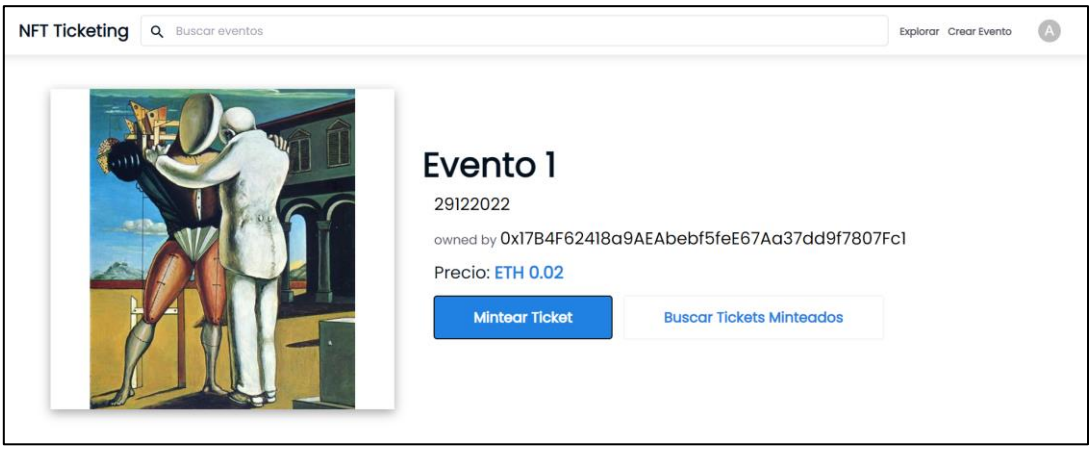

Figure 9. Page of information for an event

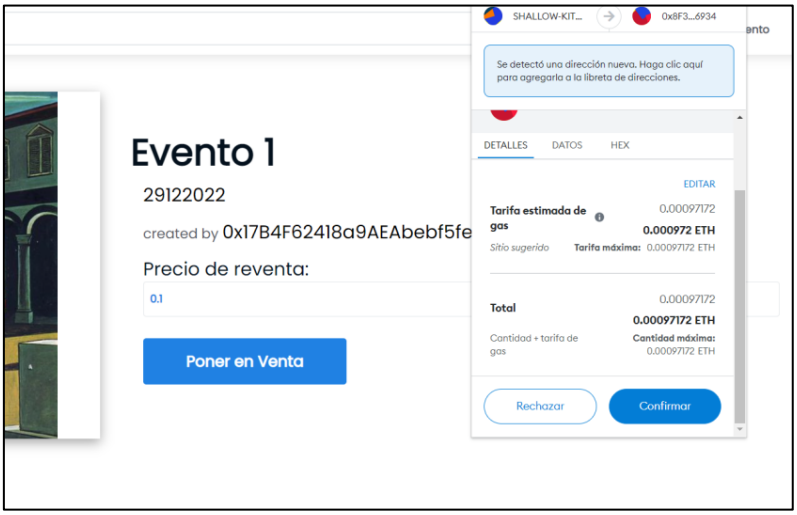

Figure 10. Transfer corresponding to the purchase of a ticket

5. Conclutions

The work carried out, fulfils all the objectives that have been proposed and, as stated in the previous section, with very good results. The contracts designed solve many of the problems encountered when managing tickets for an event. In this way, it has been possible to demonstrate, in a pragmatic way, that blockchain technology and smart contracts are useful for solving a series of problems presented by current digital industry infrastructures.

However, during the development of this project, an over-exaltation of these technologies by the community has been identified, leading to speculation and other issues that do not benefit the real study of these technologies. That said, the technology must continue to be researched and improved in order for more solutions like the one presented in this paper to emerge.

Índice de la memoria

Índice de figuras

1. INTRODUCCIÓN

Cuando Internet comenzó a dar sus primeros pasos, los pocos usuarios que hacían uso de ella se dedicaban a leer texto en bruto que era alojado en servidores ubicados a kilometros de distancia y se accedían a ellos, en unos pocos segundos, a través de un simple *link*. Entonces, se abrió un abanico de posibilidades y un sin fin de aplicaciones a una tecnología que se desconocia por completo hasta donde podía llegar. Con la llegada del *html*, se adoptó una disposición más interesante de los componentes de las páginas, aunque todavía no hubiese una interación muy compleja entre usuario y ordenador. No fue hasta los principios del siglo XXI, que se acuñó el termino Web 2.0, refiriéndose a la web como un escenario donde los usuarios debían interactuar con sus máquinas de forma colaborativa y, además, dando a entender que se trataba de un proceso que debía actualizarse. Por aquel entonces, ya existían las primeras redes sociales, donde la web comenzaba a realizar funciones de conexión entre usuarios, algo que se pensaba imposible. A fecha en la que se realiza este trabajo, la web sigue en una versión 2.0, pero con el auge de la Inteligencia Artificial, las redes Blockchain y todas las distintas tecnologías prometedoras, sería un error no avanzar y actualizarse.

Este trabajo, simplemente trata de poner en marcha esta actualización apostando por la tecnología blockchain y los contratos inteligentes. En base a su estudio, se intentará solucionar una serie de problemas que comienzan a identicarse en la industria de la venta de tickets de eventos.

¿Por qué los tickets de eventos? Los tickets de eventos, desde su comienzo han aportado un valor sentimental a través de la materialización del recuerdo que suponía el evento, pero este valor se perdió con el paso a la era digital, cuando se intercambió el recuerdo por un simple y vacio código QR. Este intercambio trajo mejoras como la eliminación de la impresión a

papel o la venta de tickets de manera remota. Y a raíz del tiempo, se encontraron aún más huecos como las ventas especulativas, los tickets falsos, los altos costes debido a los intermediarios o la simple inexistencia de mercados secundarios regulados.

No hace falta comprender en gran medida las tecnologías blockchain, para ubicar en ellas una posible solución a muchas de las carencias que trajo consigo el traslado de los tickets al mundo digital. Con este motivo, se comienza un estudio y despliegue de dichas tecnologías en el ámbito de la industria de los eventos. Para ello, se dividirá el trabajo en dos partes igual de claves. La primera, se dedicará al estudio y análisis de la cuestiones recién introducidas, y la segunda, a un acercamiento práctico y tangible de las medidas resolutorias.

2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

A continuación, revisaremos una selección de conceptos técnicos que se abordan en el proyecto. El objetivo de este capítulo es asentar unas bases para que el trabajo resulte más comprensible.

2.1 BLOCKCHAIN

Blockchain del inglés, *cadena de bloques*, es una estructura de datos que almacena información agrupada en los llamados bloques. Los bloques aparte de almacenar su propia información, almacenan información del bloque anterior y así se conectan todos entre ellos como una cadena. De esta manera, la información que se almacena en un bloque solo puede ser modificada si alteramos todos los bloques anteriores. Esto permite que se pueda aplicar en entornos descentralizados de manera que la *blockchain*, ejerza de base de datos pública.

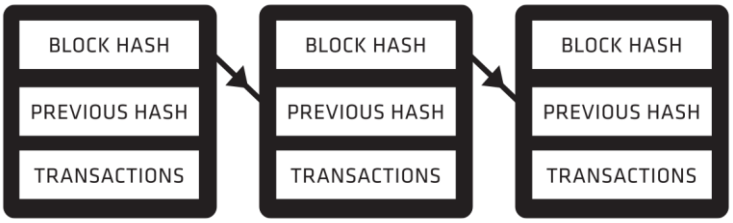

Figura 11. Esquema de una cadena de bloques. [1]

FUNCIONAMIENTO

Blockchain es una tecnología que combina tres tecnologías principales: claves criptográficas, una red peer-to-peer y un libro de contabilidad digital. En criptografía asimétrica, existen dos tipos de claves: claves privadas y claves públicas. Ambas claves son almacenadas por cada individuo o nodo y se utilizan para generar una firma digital. La característica más significativa de la tecnología blockchain es esta firma digital, que es forma de identificación digital única y muy segura. Cada transacción es aprobada por la firma digital del propietario.

Por otro lado, una red Blockchain funciona como una red *peer-to-peer*, y permite una transacción mediante una verificación común, un gran conjunto de personas operando para establecer un acuerdo sobre las transacciones. El libro mayor digital es un sistema que almacena todas estas transacciones. Podríamos decir, que funciona de manera similar a una hoja de cálculo, almacena todos los nodos de la red y realiza un seguimiento de todas las transacciones realizadas por cada nodo. La información en el libro mayor digital es muy segura y la firma digital la protege de la manipulación. El aspecto más interesante de este libro mayor es que cualquiera puede ver los datos, pero nadie puede corromperlos.

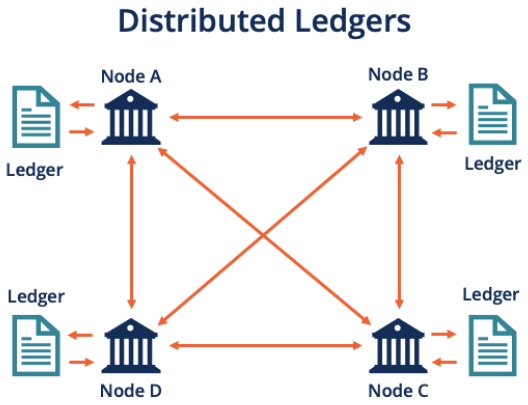

Figura 12. Tecnología de Contabilidad Distribuida [2]

https://corporatefinanceinstitute.com/resources/knowledge/other/distributed-ledgers/

FUNCIONES HASH Y LA CADENA DE BLOQUES

Una función hash es una operación criptográfica que convierte una entrada variable en una cadena de caracteres fija que funcionan como identificadores únicos. Funcionan gracias a complicados procedimientos matemáticos, y dos de las funciones has más usadas hoy en día son: MD5 y SHA-256.

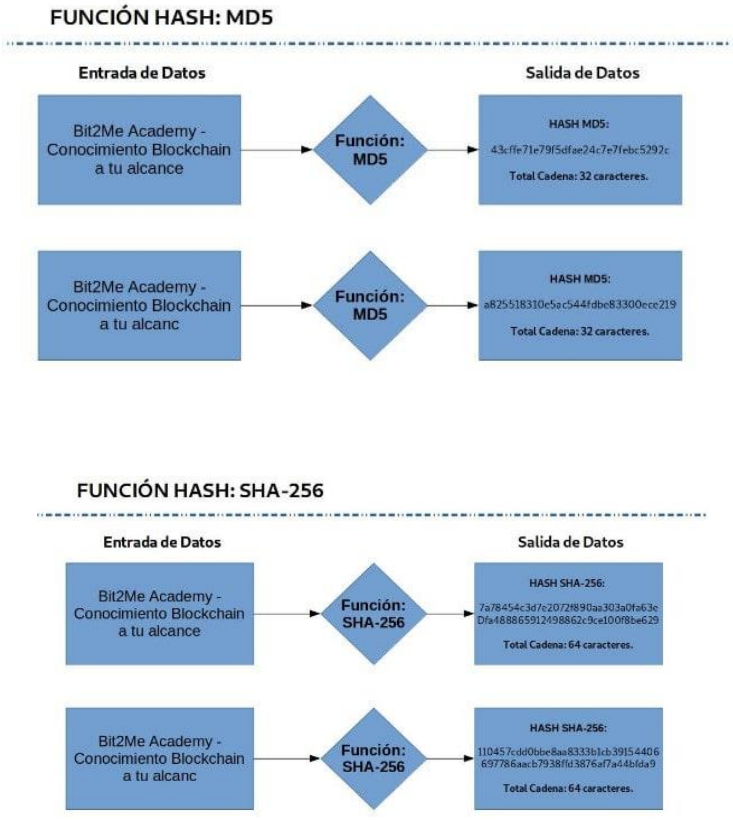

Figura 13. Funcionamiento de las funciones hash [2]

Las funciones hash son usadas en las redes blockchain debido a :

- Un bajo consumo de computo,
- No importa el tamaño de la entrada ya que siempre devuelve la misma longitud de salida.
- Cualquier cambio en la entrada altera drásticamente la salida.
- No puede haber dos entradas con el mismo hash de salida.
- Son funciones con una sola dirección, no se puede calcular una entrada a través de una salida.

En las cadenas de bloques, cada bloque se introduce a través de una función hash y se almacena en el siguiente bloque. De esta manera si el bloque es alterado, su hash se verá afectado y creará una cascada de errores debido a que es una cadena.

2.2 CRIPTOMONEDAS

Las criptomonedas son un tipo de moneda digital, que basan su seguridad en la criptografía. El control de la moneda se realiza a través de bases de datos descentralizadas, como la *Blockchain*. Haciendo uso de funciones asimétricas de encriptación y hashing o mensajería, es posible crear un libro mayor distribuido, almacenado en la cadena de bloques, que respalda y garantiza la seguridad de la moneda digital.

MINEROS Y PROTOCOLOS DE MINADO

La minería es la base de todas las criptomonedas. Se usa el término “minar”, ya que en esencia las criptomonedas y su manera de obtenerlas son similares a los minerales. Los mineros se encargan de procesar bloques compuestos de distintas transacciones y de esa manera confirmarlos y añadirlos a la blockchain. Si se quiere cerrar un bloque y añadirlo a la cadena, cada nodo de la red que participa “compite” con los demás para cerra el bloque él

y así llevarse él la recompensa. La "competición" en este caso es resolver una especie de puzle matemático.

Lo interesante de todo es que este proceso de minado es la raíz principal de muchas de las criptomonedas, porque permite que la red este funcionamiento a tiempo completo sin necesidad de servidores centralizados y sin pasar por intermediaros. Además, el trabajo de los mineros hace que sea muy robusta la seguridad de las transacciones debido a que solo se registran las transacciones que han sido validadas (los mineros auditan su legitimidad).

Existen distintos protocolos de minado:

- Proof of Work (PoW): es el protocolo mas usado. Todos los nodos son iguales y anónimos, compiten para ver quien cierra el bloque antes que el resto y conseguir criptomoneda a cambio. Para cerrar el bloque resuelven un algoritmo complejo que necesita un alto nivel de procesamiento (coste energético).
- Proof of Stake (PoS): el propósito de este protocolo es mejorar las características del Proof of Work (sobre todo el consumo de energía). Funciona como la lotería más o menos, los miembros que deseen cerrar el bloque (validadores) deben invertir una cantidad de criptomoneda que servirá como fondo. El algoritmo de Proof of Stake elegirá de manera aleatoria el miembro que deberá cerrar el próximo bloque, de entre los miembros que depositaron fondos. Cuanto más se invierta más probabilidades hay que de ser elegido. Beneficios de PoS: velocidad de procesamiento de transacciones, eficiencia energética, menos requerimiento de hardware. Por otro lado, es más vulnerable que el PoW ya solo hace falta invertir más dinero, en cambio en PoW además hay que invertir tiempo. Además, este protocolo favorece una concentración de la riqueza, los ricos tenderán a ser más elegidos y por lo tanto ganar más criptomonedas.
    - Delegated Proof-of-Stake (DPoS)

- Leased Proof-of-Stake (LPoS)

- Proof of Authority (PoA): en este protocolo participan solo los nodos autorizados que deben identificarse. No hay competencia para ver quien cierra el bloque antes y por lo tanto no suele haber criptomoneda asociada a ese trabajo. Además, no se consume tanta energía debido a la falta de competencia.

CARTERAS DIGITALES O MONEDEROS

Las carteras digitales son aplicaciones externas a las blockchains y a las criptomonedas, que principalmente se encargan de guardar las claves privadas y públicas. Cada blockchain suele tener una serie de carteras digitales asociadas, aunque los monederos más famosos trabajan con varias redes blockchain distintas. Al almacenar las claves de las cuentas de los usuarios son capaces de mostrar el saldo correspondiente a cada una. Todas implementan funcionalidades básicas como la transferencia de criptomonedas a otras cuentas, la confirmación de pagos, etc.… También, son a través de ellas, que los usuarios consiguen crearse las cuentas o direcciones en las blockchains.

2.3 ETHEREUM

Es una plataforma basada en blockchain. Su funcionamiento gira entorno a la Ethereum Virtual Machine o EVM. La EVM es una entidad única cuyo estado es acordado por los nodos participantes de la red. Todos ellos pueden hacerle peticiones de realización de algún cálculo y el resto de participantes de la red deben verificar, validar y ejecutar dicho cálculo. Esta petición de cáculo, provoca un cambio en el estado de la EVM que se envía a toda la red. Estas peticiones se denominan solicitudes de transacción. Un resgistro de todas ellas se almacena junto con el estado de la EVM en la blockchain, y esta a su vez en todos los nodos.

ETHER

Ether es la criptomoneda nativa de Ethereum y su función es crear un incentivo económico para que los participantes de la red puedan realizar y comprobar las solicitudes de transacción. Si un participante desea realizar una petición de transacción, debe aportar una cantidad de Ether determinada (en función del gasto computacional del cálculo). De esta manera, si un participante desea ejecutar un código que consume muchos recursos o que contiene bucles infinitos, deberá pagar constantemente.

2.4 CONTRATOS INTELIGENTES

De manera simplificada, los contratos inteligentes son unas líneas de código que se almacenan en una blockchain y se ejecutan cuando se cumplen una serie de condiciones. A menudo se utilizan para automatizar acuerdos digitales para que todas las partes involucradas puedan confiar en que el acuerdo se cumpla. También pueden automatizar un proceso de modo que cuando se cumplan las circunstancias, se ejecute la siguiente acción, como ocurre en cualquier otro programa informático. Todas estas características permiten que las partes involucradas en un contrato inteligente no tengan que confiar entre ellas o en una tercera parte, solo deben confiar en la tecnología de la blockchain. Esto permite que en los casos donde se puedan adoptar este tipo de tecnologías, los costes de gestión se reduzcan drásticamente, ya que puede no ser necesaria la implicación de un intermediario.

Algunos de los beneficios de los contraos inteligetes son:

- Ejecución instantánea: al ser código, esta digitalizado y automatizado, una vez que esten descritos los requisitos se ejecuta inmediatamente.
- Seguridad: todas las transaciones de la blockchain se resgitran y se encriptan para que no puedan existir datos malintencionados.

- Reducción del coste: al no haber intermediarios, se reducen los costes de operación y gestión.
- Confianza y transparencia: al no deberse a la confianza de un intermediario para que un acuerdo se cumpla, ya que la misma red se encarga de dicha tarea, no hay un problema de alteración de archivos.

Figura 14. Casos de Uso de los Contratos Inteligentes [4]

2.5 IPFS

IPFS proviene del las siglas en inglés *InterPlanetary File System.* Es un sistema de almacenamiento de archivos de manera descentralizada o sistema distribuido. Sus principales objetivos son: preservar la seguridad, garantizar la privacidad y eliminar la censura de la información. Internet, en un principio se ideo para que fuese una red distribuida, pero en la práctica a terminado funcionando en un modelo Cliente-Servidor. IFPS trata de resolver muchos problemas que acarrean los protocolos vigentes de internet como el *http.*

De manera simplificada, en el sistema IFPS, los usuarios no acceden a un servidor para que les proporcione los datos que desea, si no que se pone en contacto con los nodos más cercanos a él para que estos les proporcionen la ruta hacia la ubicación de su infromación. De esta manera la estructura de datos se encuentra totalmente distribuida.

2.6 HERRAMIENTAS

A continuación, se expondrán con una breve descripción las herramientas que se usarán durante el proyecto.

VISUAL STUDIO CODE

Es un editor de código creado por Microsoft que cuenta con un marketplace especifico ofreciendo miles de extensiones para hacer más comoda la escritura de código. Una extensión destacable que se ha usado durante el proyecto es la de Copilot, que facilita la escritura de código a través de un modelo de lenguaje entrenado por Github.

Figura 15. Logo de Visual Studio Code

TRUFFLE

Es un espacio de desarrollo que sirva para el testo, compilación y despliegue de contratos inteligentes. Esta especializado en la red blockchain de Ethereum, y proporciona numerosas funcionalidades que facilitan el desarrollo de contratos para dicha plataforma.

Figura 16. Logo de Truffle

GANACHE

Ganache es una blockchain de área local que se suele utilizar en paralelo con Truffle. Es ideal para ámbitos de testeo ya que proporciona una interfaz de usuario muy intuitiva y funcional y una gran cantidad de direcciones de cuentas con claves privadas para realizar transacciones de prueba con los contratos inteligentes que se hayan desplegado en dicha red.

Figura 17. Logo de Ganache

SOLIDITY

Solidity es un lenguaje de programación orientado a objetos que sirve para desarrollar contratos inteligentes. Es aceptado en numerosas redes blockchain conocidas, entre ellas Ethereum. Es un lenguaje que hereda características de muchos otros como *Javascript.*

Figura 18. Logo de Solidity

METAMASK

Es una plataforma que funciona como cartera digital de Ethereum. Es necesario descargarse su formato de extensión para navegadores como Chorme o Firefox. Tiene unas serie de características que la hacen más útil para el testing, por ejemplo, puede añadir redes locales e importar las cuentas creadas en dichas redes. Su uso es totalmente gratuito.

Figura 19. Logo de Metamask

NODE JS

Es un entorno de ejecución basado en *Javascript* que funciona a través de eventos asíncronos. Su principal uso es el de la creación de aplicaciones web dinámicas y eficientes.

Figura 20. Logo de Node.js

REACT

Es una biblioteca de código abierto de Javascript que facilita la creación de interfaces de usuario interactivas con el objetivo de realizar aplicaciones de una sola página. Su funcionamiento parte del diseño de vistas simplificadas para cada uno de los estados de la aplicación.

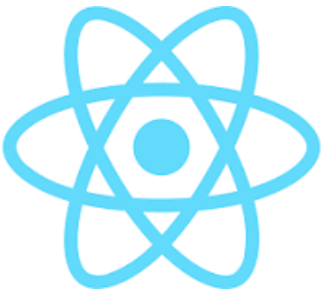

Figura 21. Logo de React

TAILWIND

Es un framework de código abierto basado en css que sirva para el desaroolo frontend. Se usa principalmente para darle un estilo a las páginas de la interfaz de usuario de una aplicación, a través de un listado de clases muy personalizadas.

Figura 22. Logo de Tailwind

NEXT JS

Next js es un marco de desarrollo que corre por encima de Node js y permite funcionalidades basadas en React. Una de ellas es el Server-Side-Rendering, que permite generar páginas web estáticas personalizables para cada usuario.

Figura 23. Logo de Next.js

3. ESTADO DE LA CUESTIÓN

3.1 NFT: ESTÁNDAR ERC-721

A día de hoy, Ethereum cuenta con un elevado número de ventajas que dan un notable valor al uso de su red blockchain frente al reste de redes disponibles La capacidad de implementar contratos inteligentes es una de ellas. Dichos contratos tienen numerosas aplicaciones que ya se están implementando en el mercado, y una de ellas son los NFT´s.

¿QUE SON LOS NFT'S?

El acrónimo de NFT es Token No Fungible, y una manera simplificada de definirlo es como un certificado de autenticidad, de un archivo, el cuál es almacenado en un red blockchain capaz de implementar contratos inteligentes, como es el caso de Ethereum. Se le denomina *token*, ya que es un tipo especial de criptoactivo, y le siguen *no fungible*, ya que su intercambio no se puede realizar de forma idéntica. Las aplicaciones de los NFT´s son numerosas, entre ellas: el mundo del arte, los videojuegos, los mundos virtuales, la música, el cine, etc.…

Las características principales que tienen los NFT´s es que son:

- Únicos
- Indivisibles
- Transferibles y coleccionables
- Capaces de demostrar su escasez

Los NFT's, son fruto de uno de los muchos estándares de tokens. En la comunidad de Ethereum, existen las llamadas *EIP*, que son unas propuestas de mejora que se someten a debates en la comunidad por medio de un *proceso estándar*. Los estándares tienen como objetivo tratar de lograr que los contratos inteligentes y las aplicaciones descentralizadas o *dapps* sigan siendo compuestas. En concreto, los NFT´s surgieron a raíz del token estándar ERC-721.

ERC-721

El ERC-721 es un estándar de token no fungible que implementa una API para tokens dentro de los contratos inteligentes. Le permite transferir tokens de una cuenta a otra, obtener el saldo actual de tokens de una cuenta y ver el suministro total de tokens disponibles en la red. También tiene otras características, como permitir que una cuenta de terceros gaste una cierta cantidad del token de una cuenta.

Si un contrato inteligentes implementa los siguientes métodos y eventos, se denomina contrato de token ERC-721 y será responsable de realizar un seguimiento de los tokens creados en Ethereum una vez implementados.

```
event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);
event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);
```

```
function balanceOf(address _owner) external view returns (uint256);
function ownerOf(uint256 _tokenId) external view returns (address);
function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;
function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;
function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
function approve(address _approved, uint256 _tokenId) external payable;
function setApprovalForAll(address _operator, bool _approved) external;
function getApproved(uint256 _tokenId) external view returns (address);
function isApprovedForAll(address _owner, address _operator) external view returns (bool);
```

Figura 24. Métodos ha implementar por un token ERC-721 [5]

A pesar de tener sus raíces en Ethereum, el estándar no está ligado exclusivamente a esta red. De hecho, los contratos ERC721 se pueden usar en cualquiera de las redes blockchain que usen la *Ethereum Virtual Machine*.

OPENZEPPELIN TOKEN ERC-721

Por lo tanto, si un contrato inteligente implementa las funciones y eventos que se acaban de describir, se convierte automáticamente en un token ERC-721. Un token ya implementado y muy usado para la creación de contratos inteligentes tipo NFT, es el que realiza Openzeppelin. Openzeppelin es una librería que proporciona una gran cantidad de utilidades relacionadas con los contratos inteligentes, siendo una de ellas es el token ERC-721. Para explicar el funcionamiento de los contratos de Openzeppelin, se va analizar el código de un NFT implementado a través de dicha librería.

```
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";
import "@openzeppelin/contracts/utils/Counters.sol";

contract NFT is ERC721URIStorage {
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    constructor() ERC721("GameItem", "ITM") {}

    function mintNFT(string memory tokenURI)
        public
        returns (uint256)
    {
        uint256 newItemId = _tokenIds.current();
        _mint(newItemId);
        _setTokenURI(newItemId, tokenURI);

        _tokenIds.increment();
        return newItemId;
    }
}
```

Figura 25. Contrato inteligente de un NFT [6]

En este ejemplo se ha usado una extensión que permite, en vez de establecer una URI base para todos los tokens, establecer una URI para cada token que se cree. Así, se ha descrito la función *mintNFT*, que recibe un tokenURI y lo almacena asociándolo con la dirección del que crea el token. Este proceso se conoce como *mint* o mintear un token y es referido a la creación de un nuevo identificador en un contrato que implemente el estándar 721.

3.2 NFT TICKETING

Según las estadísticas proporcionadas por *DappRadar*.com, el mercado de NFT alcanzó un volumen comercial de 30 mil millones de dólares como resultado de la creciente popularidad de las NFT. Si bien las NFT están transformando muchas otras industrias, la industria de eventos es pionera en la utilización de NFT para vincular entradas físicas y digitales. Antes del desarrollo de las entradas digitales, las personas coleccionaban entradas tradicionales. Ya fuera una ópera en el teatro de la Scala en Milán, de la final de la Champions en París o un concierto de los Rolling Stones en Londres, los tickets tenían un valor .

El objetivo del NFT Ticketing es volver a darle el valor a los tickets sin perder las ventajas de su transformación digital. Dado que la digitalización ha hecho que la emisión de entradas de eventos sea más eficiente, ahora todos pueden usar los sistemas de gestión de tickets con seguridad. Sin embargo, el uso de un código QR sencillo sin explicación detrás parece aburrido. Los tickets digitales no pueden servir como reliquias que las personas desean guardar y recordar.

¿QUÉ PLATAFORMAS OFERTAN NFT TICKETING?

A fecha en que se escribe esta memoria, en España no se ha popularizado el uso de las NFT´s como entradas para eventos. Existe una empresa, llamada *Get*, que oferta una infraestructura de gestión de NFT´s Ticketing para eventos que quieran dar el paso a esta nueva tecnología.

Por otra parte, y con una popularidad más pronunciada, existe Opensea, que es el Marketplace más usado a nivel global. En él, se ofertan la gran mayoría de NFT´s del mercado y se realizan las operaciones con mayor volumen pero, ¿puede estar preparado Opensea para el uso de NFT Ticketing?

3.3 MARKETPLACES Y OPENSEA

Con el motivo de encontrar una solución viable para ofertar tickets en formato de NFT en el mercado, se realizará un estudio de Opensea y cómo de útil resultaría usar dicha plataforma para la función que se exponía en el punto anterior. También, se anotará el funcionamiento de la plataforma para un posible futuro proyecto.

OPENSEA

OpenSea es una plataforma digital descentralizada que se dedica a la compra y venta de NFT's. Fue fundada en 2017 por Devin Finzer y Alex Atallah con la idea principal de convertirse en el "nuevo Amazon" de este sector. OpenSea cuenta ahora mismo con una cantidad de unos 20 millones de activos coleccionables, unos 2 millones de usuarios y alrededor de 200 categorías de NFT's. Funciona más o menos como todos los Marketplace tradicionales, solo que los productos que se ofertan son NFT's. Al comercializar NFT's, OpenSea necesita de una red blockchain para funcionar y esta utiliza la red de Ethereum.

Para empezar a realizar transacciones en OpenSea lo primero que es necesario es tener una cartera o monedero digital. Hay un montón de carteras digitales pero la propia plataforma te recomienda que uses Metamask. Una vez registrado en Metamask debes enlazar tu cartera con tu cuenta de OpenSea que servirá como un proceso de identificación biométrica que encontramos en otras plataformas. Una vez finalizado esta serie de pasos el usuario ya puede comenzar a comprar NFT's, o a subir los suyos propios y venderlos en la plataforma.

Cabe destacar que aunque OpenSea este corriendo en la blockchain de Ethereum, actualmente permite de manera transversal usar otras blockchains como las de Polygon y Klatyn. De esta manera OpenSea cuenta con casi 150 tokens distintas entre las que destacan WETH y USDC.

OFERTAR UN NFT EN LA PLATAFORMA

1-Configurar la billetera

Se he decidido usar la cartera digital de Metamask, ya que es la más famosa. Es sencillo creársela, piden una contraseña y se recibe una clave formada por una serie de palabras aleatorias. Si se usa el navegador Chrome, se tendrá que añadir la aplicación de Metamask en formato extensión en la banda superior del buscador.

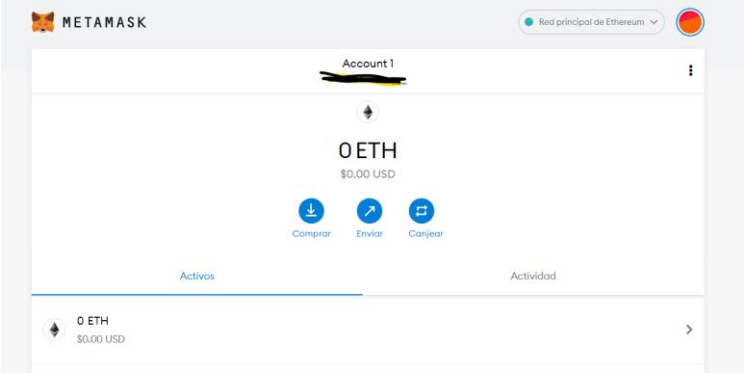

Figura 26. Vista de una cuenta desde Metamask

2- Crear cuenta en OpenSea y enlazarla a una cartera

Se entra en OpenSea y se verá una opción de elegir nuestra cartera, se selecciona la cartera de Metamask. La plataforma redireccionará nuestro navegador a otra pestaña y pedirá que

se confirme la conexión. Ahora, si se vuelve a la extensión de Chrome de Metamask, aparecerán las páginas web que tenemos conectadas con nuestra cartera, entre ellas la recién conectada OpenSea. Además, pide firmar los términos y condiciones de OpenSea.

3- Crear una colección

Habrá una opción de *create a collection* y aparecerá una página para rellenar un formulario. Los campos obligatorios son una imagen para el logo de la colección y un nombre para nuestra colección. Además, se podrá añadir una descripción, una serie de imágenes para embellecer la estética de la colección y una categoría para nuestra colección. Por último, pedirá seleccionar la blockchain en la que queremos que nuestra colección se cree.

4- Añadir un NFT a nuestra colección

Dentro de la pestaña de nuestra colección, hay un botón de *Add Item*, si se pulsa se redireccionará a otro formulario con distintos campos. El primer campo es el propio archivo que acabará siendo nuestro archivo tokenizado.

Los siguientes campos los enumerare con una breve descripción de cada uno:

- Nombre: no hace falta que sea único, solo sirve para identificar tu NFT.
- Enlace Externo: es un campo opcional, sirve para conectar tu NFT con una página con más información sobre este o cualquier página que quieras.
- Descripción: otro campo opcional para añadir una descripción. Admite Markdown syntax (un tipo de sintaxis personalizada).
- Propiedades: es opcional, sirve para añadir una serie de atributos para intentar diferenciar tu NFT (deberás decir su tipo y su nombre).
- Niveles: es muy parecido al campo anterior, pero en este caso son atributos cuantitativos (valores numéricos).
- Estadísticas: sirve para añadir estadísticas a tus NFT. Ejemplo: en los CriptoKitties
- Contenido Desbloqueable: es un campo de texto que solo el dueño del NFT será capaz de ver.

- Contenido Explicito y Sensible: sirve para marcar tu NFT como contenido sensible o no apto.
- Suministro: el numero de ítems que se pueden minar (numero de NFT), en OpenSea solo te permite crear 1 de momento. Planean añadir la opción de crear múltiples. Esto no significa que se creen copias del original, si no varios originales.
- Blockchain: elegir la red blockchain en la que se creará tu NFT.
- Metadatos congelados: los metadatos son atributos que no se van a guardar en la blockchain. Al estar fuera, son susceptible de ser modificados haciendo que pierda sus características originales. Por lo tanto esta opción te permite congelar y hacer invariable los metadatos.

TARIFAS DE OPENSEA

Hay tres tipos de formas que tiene OpenSea de cobrarte tarifas:

- Tarifa de servicio
- Derechos de autor
- Tarifas de gas

Tarifas de servicio

Las tarifas de servicio en OpenSea, a fecha en la que se escribe esta memoria, son de un 2,5%, lo que significa que OpenSea se lleva un 2,5% del precio total de una transacción cada vez que se vende un NFT. Esta tarifa es siempre pagada por el vendedor. Un competidor directo de OpenSea es LookRare, que no es más que otra plataforma de compra y venta de NFT, y en LooRare solo aplican un 2% de tarifa de servicio.

Derechos de autor

Los derechos de autor o *royalties* son una tarifa que de nuevo pagará el vendedor. Su valor lo decide el creador del NFT, en OpenSea el valor máximo es de 10%. Esta de hecho es una

de las grandes diferencias en la venta de arte por medio del Web3, los artistas/creadores son capaces de cobrar con cada reventa de su obra.

Tarifas de gas

Cuando hablamos de tarifas de gas en OpenSea nos referimos a las tarifas que pagan los usuarios a los mineros por asegurar las transacciones en la blockchain de Ethereum (en la que corre OpenSea). Siempre que tratemos con protocolos basados en PoW tendremos tarifas de gas, es una manera de pagar el gasto de procesamiento. Este gasto depende directamente de la oferta y la demanda del mercado y de la tasa de hash (Hash Rate y la complejidad de la red). Todo esto hace que la precio a pagar fluctúe a menudo. Hay dos tipos de tarifas de gas:

- Tarifas únicas
- Tarifas periódicas

4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

A lo largo del capítulo anterior, se ha tratado de acercar los temas principales de este trabajo, con el objetivo de dar respuesta a las preguntas que surgen sobre el asunto, y recapitulando, se ha explicado: el funcionamiento general de Ethereum, en qué consiste la Web 3.0, cómo funcionan los estándares de Ethereum, qué es un NFT, qué tecnología hay detrás, el significado de NFT Ticketing y qué soluciones hay en el mercado. Así, se ha expuesto el potencial de dichas tecnologías y, como se ha podido comprobar en los últimos apartados, se puede concluir que las plataformas más usadas a nivel global no facilitan la oferta de tickets NFT´s para eventos a través de las mismas. Si, de todos modos, se quisiera ofertar las entradas a través de la plataforma estudiada sería muy costoso debido al funcionamiento de las tasas o *fees* que se dan en dicha plataforma y, además, muy laboriosos debido a que la plataforma no se especializa en el asunto.

Con estas premisas, se ha decidido realizar un proyecto que satisfaga las necesidades del mercado y de los usuarios del Ticketing. La inclusión de los NFT´s en el Ticketing es una propuesta que trata de añadir valor a una industria que en su historia ya ha realizado grandes cambios. La razón más importante y por la que se ha realizado este proyecto, es porque se cree firmemente que con la llegada de la web 3.0, esta industria debe adoptar una serie de medidas para traer mejoras y así beneficiar a todos sus usuarios.

PROBLEMAS DEL TICKETING TRADICIONAL

- **Las ventas especulativas o *Scalping***

El *Scalping* es la técnica de comprar entradas y revenderlas para obtener una ganancia rápida usando distintas plataformas de venta no autorizadas para la distribución de las entradas. Una de las principales causas de que esto ocurra, es que los que realizan el *Scalping* cuentan con ejércitos de *bots*, perjudicando gravemente a los usuarios que realmente quiere acceder al evento.

- **No existen mercados de reventa**

La industria de las entradas ahora se basa en una gran cantidad de plataformas web independientes. Debido a que no existen protocolos de intercambio entre las plataformas, es muy difícil verificar la autenticidad de un ticket que se quiere comprar. Esto provoca que suba el riesgo de comprar una entrada falsa. Además, la detección de billetes falsos es más complicada, ya que cada vez más plataformas los ofrecen. No existe un sistema que examine la legitimidad de todos las entradas al mismo tiempo.

- **Entradas Falsas**

Cuanto más importante sea el evento, más probable es que se compre un ticket falso. Como comentábamos antes, los usuarios que de verdad quieran ir acaban obteniendo el ticket del evento a través de distribuidores no autorizados.

- **Coste del Servicio**

Normalmente, cuando se compra un ticket a través de una plataformas tradicional, no se tiene en cuenta que en el precio de la entrada no solo está incluido el precio por poder asistir al evento, si no que en torno a un 20% o 25% del precio va destinado a los gastos de servicio. Estos gastos de servicio varían mucho dependiendo de la plataforma y del evento, pero pueden incluir: los gastos de la gestión, los beneficios de la plataforma, los gastos de la entrega, etc. …

MEJORAS CON EL NFT TICKETING

Acercar la tecnología blockchain y los NFT al mercado de la venta de entradas, beneficiaría tanto a los organizadores como a los asistentes de los eventos. Mejorará las infraestructuras del mercado, lo que resultará en un aumento de las ventas. Los siguientes puntos son algunos de las ventajas y soluciones que propone dichas tecnologías en el sector de la organización de eventos.

- **Eliminar los Tickets Falsos**

La tecnología blockchain nos brindan un único punto de verificación que tanto los organizadores de eventos como los asistentes pueden utilizar para validar tickets y evitar esquemas fraudulentos. Esto es factible porque la acuñación y transmisión de entradas NFT´s se registran en la cadena de bloques, lo que permite que todas las personas involucradas verifiquen su legitimidad.

- **Contratos Inteligentes**

Los tokens no fungibles son contratos inteligentes, así que funcionan igual que cualquier código escrito en un programa, siempre se va a cumplir lo que este escrito en ellos. Esto implica que las entradas pueden incluir código que gestiona las ventas, las transferencias, las particiones de ganancias, las ventas primarias y secundarias, etc.… También, en un fututo podrían representar un acuerdo legal entre dos partes, promotor y artista, y que sus condiciones esten descritas en el contrato inteligente.

- **Eliminar los costes y tiempos**

El uso de las tecnologías blockchain significa una reducción en los costes en comparación con las infraestructuras tradicionales. La distribución de los tickets costaría muy poca cantidad ya que solo habría que pagar los gastos de despliegue del contrato NFT y los costes

del gas de la red, que en muchos casos son extremadamente bajos. Y ya si se comparasen los tiempos de producción de los tickets, que en los casos de tickets impresos pueden llegar a tardar una semana, en infraestructuras blockchain pueden crearse miles o millones de tickets en minutos.

- **Seguridad y Autenticidad**

Adoptando estas tecnologías, también se verá incrementada la seguridad de la infraestructura. Al estar distribuyendo las entradas en redes de tipo blockchain, no se pueden perder los tickets, ya que se almacenan en el registro descentralizado. Además, no se podrá suplantar la identidad ya que cada ticket estará conectado a su dueño dentro del código del contrato.

- **Mercados de Reventa**

Una de las principales ventajas que se entregan en esta actualización, son la eliminación de los mercados no autorizados. Se suprime la posibilidad de comprar una entrada falsa, añadiendo un mercado de ventas secundarias en la plataforma asegurados por la red blockchain.

- **Devolver el Valor Artístico**

Antes de la transformación digital del mercado de los tickets de eventos, las entradas se almacenaban como piezas de arte únicas. Se le atribuía un gran valor a los diseños y a el conservarlas. Gracias a la tecnología que se pretende adoptar esto puede volver a ser así, ya que la blockchain es capaz confirmar la veracidad de un archivo, ya sea: una imagen, un video, un GIF, etc. De esta manera no se almacenará un simple código QR.

4.2 OBJETIVOS

Una vez expuesta la justificación del proyecto, los objetivos, serán las medidas que se realizarán para tratar de conseguir las mejoras enunciadas en el punto anterior. El objetivo principal del trabajo es el de desplegar una plataforma descentralizada que gestione la compra y venta de eventos mediante la tecnología de contratos inteligentes y sus estándares. Para lograrlo, debemos tratar de dividirlo en objetivos más pequeños. Estos objetivos servirán de mapa durante todo el desarrollo del proyecto.

Primero especificaremos los elementos que se esperan obtener:

- Una interfaz para que el usuario sea capaz de comunicarse con el resto de sistemas. Debe de ser estéticamente atractiva, de funcionamiento intuitivo y totalmente funcional.
- Un sistema formado por contratos inteligentes que implementen los estándares correspondientes. Deben de definir la lógica del mercado al completo en su código.
- Un servidor compuesto por unas páginas dinámicas y una API, capaces de controlar todas las funcionalidades que ofrece el sistema de contratos digitales. Este sistema deberá, además, poder interactuar con la interfaz de la manera eficiente.

Además, el orden cronológico y funcional del proceso de desarrollo será:

1. Descripción de las especificaciones de la aplicación a desarrollar.
2. Diseño de la aplicación al completo usando las especificaciones descritas.
3. Implementación de la aplicación usando los diseños realizados.

4.3 METODOLOGÍA

Para este proyecto, se ha decidido usar una metodología en cascada ya que el proyecto debe entregarse en un tiempo limitado y una metodología secuencial permite tener algo que entregar al final de la planificación.

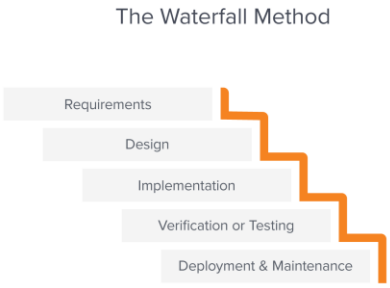

Figura 27. Esquema de un modelo en cascada [7]

Como se ve en la figura, hay cinco escalones en el desarrollo software a través de un modelo en cascada. En el caso de este proyecto, el apartado de Implementación englobará a el de Verificación y Testeo y al de Despliegue. Cabe destacar que en realidad se aplicarán técnicas de metodología agile durante el proceso de Requisitos, como el Scrum y las Historias de Usuario, y durante la implementación.

4.4 PLANIFICACIÓN

Una vez definido la metodología que se va a usar, se expondrá la planificación. Al ser una metodología en cascada se ha decidido usar un diagrama de Gantt.

	Noviembre	Diciembre	Enero	Febrero	Marzo	Abril	Mayo	Junio
Estudio de la Tecnología								
Analisis de la Cuestión								
Descripción de las Especificaciones								
Diseño de la Aplicación								
Implementación								

Figura 28. Diagrama de Gantt del proyecto

Como se puede observar en la figura, se va a dedicar gran parte del tiempo al estudio y análisis de las tecnologías y de la cuestión. Este paso es clave en el desarrollo de cualquier tipo de proyecto, ya que, para poder buscar soluciones a problemas concretos, se debe entender muy a fondo el tema que se está tratando. Después, y como ya se adelantó en el apartado de Objetivos, se realizará una descripción de las características del sistema a desarrollar, se hará un diseño completo de la aplicación y de la interacción entre cada subsistema y, por último, se realizará una implementación en base a tal diseño.

5. SISTEMA DESARROLLADO

Siguiendo con lo descrito en la definición del trabajo, el objetivo principal del trabajo consiste en el desarrollo de un sistema capaz de gestionar la compra y venta de entradas de eventos en forma de NFT o tokens no fungibles. El sistema estará formado por contratos inteligentes con sus respectivos test, desplegados en una red blockchain local, con funciones de creación, gestión y autorregulación. Además, se ha desarrollado una interfaz web para interactuar directamente con dichos contratos y una API REST para gestionar las solicitudes y necesidades de la aplicación, y el procedimiento de pagos se ha configurado para que sean posibles mediante una cartera digital externa conectada a la Blockchain local.

En este capitulo se explicará en profundidad el proceso completo del desarrollo. Se dividirá en tres bloques, Especificaciones, donde se expondrán las características del sistema a diseñar, Diseño, que, recopilando las especificaciones del bloque anterior, se evaluarán y decidirán las soluciones más optimas en cuanto al diseño, y por último, Implementación, que recogerá las materializaciones, con sus respectivos análisis, de los diseños expuestos en el bloque anterior.

5.1 ESPECIFICACIONES

Como ya se ha explicado durante la definición del trabajo, el sistema es una plataforma descentralizada para que permite crear NFT's (tokens no fungibles) y que también sea capaz de gestionar los minados de los contratos y el intercambio de los tokens ya minados. Además, como el proyecto pretende establecer un modelo de mercado de *Ticketing* de eventos, los tokens estarán especializados en ello, a través de las distintas propiedades intrínsecas de los tokens ERC721 (visto en el Capítulo 3).

En la aplicación existen tres elementos clave: el Marketplace, los Eventos y los Tickets. Los introduciremos conceptualmente para facilitar la comprensión de los mismos y poder proceder a definirlos de manera práctica.

El primero y más general, el Marketplace, es el enlace entre usuarios, eventos y tickets. Está compuesto por varios sistemas, entre ellos, un contrato inteligente, una API y una interfaz gráfica. El Marketplace es el único que se comunica directamente con los usuarios y el único capaz de gestionar los otros elementos. Además, es el encargado de cumplir con las tareas que involucren a usuarios con eventos y tickets.

El segundo elemento es los Eventos, que serán las representaciones computacionales de los eventos reales. Los Eventos se forman a través de contratos inteligentes y de estructuras de datos, por lo que solo existirán dentro de la Blockchain. En concreto, y como veremos en el apartado de diseño, cada Evento tendrá una dirección asociada y esta se guardará para que el Marketplace pueda interactuar con ellos. Cada Evento almacenará una serie de datos: nombre, descripción, precio, royalties, fecha del evento, fecha de inicio de venta, etc.

Nota: Al no ser el objetivo de este trabajo, no se incluirá en el desarrollo un sistema de verificación de eventos, los eventos se darán por autentificados al crearse.

El tercer y último elemento clave son los Tickets. Los Tickets serán muy parecidos a los eventos, ya que almacenan muchos datos idénticos. Estos también se forman a través de los contratos inteligentes, pero en vez de tener direcciones de memoria de la Blockchain individuales, se enumeran usando un contador almacenado en cada Evento.

Una vez introducidos los elementos clave, el siguiente paso es identificar las características que tiene el sistema como el que se pretende implementar, a nivel de usuario. Compartirá ciertas características comunes a los Marketplaces convencional, por lo que debe de ser capaz de:

1. Identificar a los distintos usuarios de manera única
2. Crear y gestionar nuevos eventos con características distintas.
3. Vender y gestionar entradas para los eventos disponibles.
4. Gestionar los pagos de las entradas.

Además, la aplicación, en vista de implementar las mejoras que se idearon en capítulos anteriores, y aprovechando las ventajas de la blockchain y la web 3.0, deberá ser capaz de:

1. Enlazar cada Evento a un contrato ERC721 (o implementaciones del mismo), para crear eventos con tickets *tokenizables*.
2. Disponer a cada Evento de un mercado de reventas único (explicado más adelante).
3. Sellar la información de los Eventos creados, sus cualidades, su dueño y los dueños de los tokens o tickets ya *minteados* del evento.
4. Guardar un registro de los balances asociado a cada dirección.
5. Permitir el acceso a la información sobre los eventos y los tickets, de manera púbica

En relación con la compra y venta de entradas, los eventos no dispondrán de un sistema de devoluciones, no obstante, se creará un mercado de reventas enlazado a cada evento. Este mercado será complementario de los Eventos y solo estará disponible si existen tickets en venta. Se darán siempre dos opciones para obtener un ticket. La primera será *minteándolo*, donde el usuario a través del Marketplace se comunicará con el contrato del evento y *tokenizará* una entrada y esta será enlazada con su dirección. La segunda opción será comprar un ticket que ya haya sido *minteado* por otro usuario, y que este halla decidido poner en venta con un nuevo precio. El usuario comprador transferirá la cantidad establecida como nuevo precio, y el Marketplace se encargará de transferir la entrada *tokenizada*, así como de enlazarla a la dirección del nuevo dueño.

Nota: Cada Evento tiene configurado un valor de *royalties*, un valor del 0 al 1, que indica el porcentaje que se lleva el organizador del evento cada vez que se cierra una venta del mercado de reventas.

HISTORIAS DE USUARIOS

Como ya se comentó en el apartado de Metodología, el proceso previo al diseño e implementación es realizar las llamadas *Historias de Usuario*. EL objetivo es representar en lenguaje natural, desde el punto de vista del usuario, los requisitos que debe incorporar la aplicación.

Antes de comenzar a exponer las historias, se procederá a localizar los distintos actores que forman parte en el sistema:

- Usuario: es cualquier persona que usa la aplicación, identificándose con una dirección única.
- Owner: es un usuario que posee algo, ya sea una o varias entradas o uno o varios eventos.

Una vez localizado a los actores, se procede a exponer las historias de los usuarios:

- Como usuario quiero poder buscar un evento para ver si hay alguno que sea de mi interés asistir.
- Como usuario quiero poder crear un evento para comenzar a vender entradas de manera segura y sin intermediarios.
- Como usuario quiero poder mintear un evento para recibir bonificaciones por parte del evento
- Como usuario quiero poder comprar una entrada que haya sido *minteada* para así poder acceder al evento aunque sea con un precio superior.

- Como owner quiero poder ver los eventos que he creado para poder cancelar o editar alguno de ellos.
- Como owner quiero poder ver los tickets que poseo para ver su información o ponerlos en venta con un precio que decida yo.

CASOS DE USO

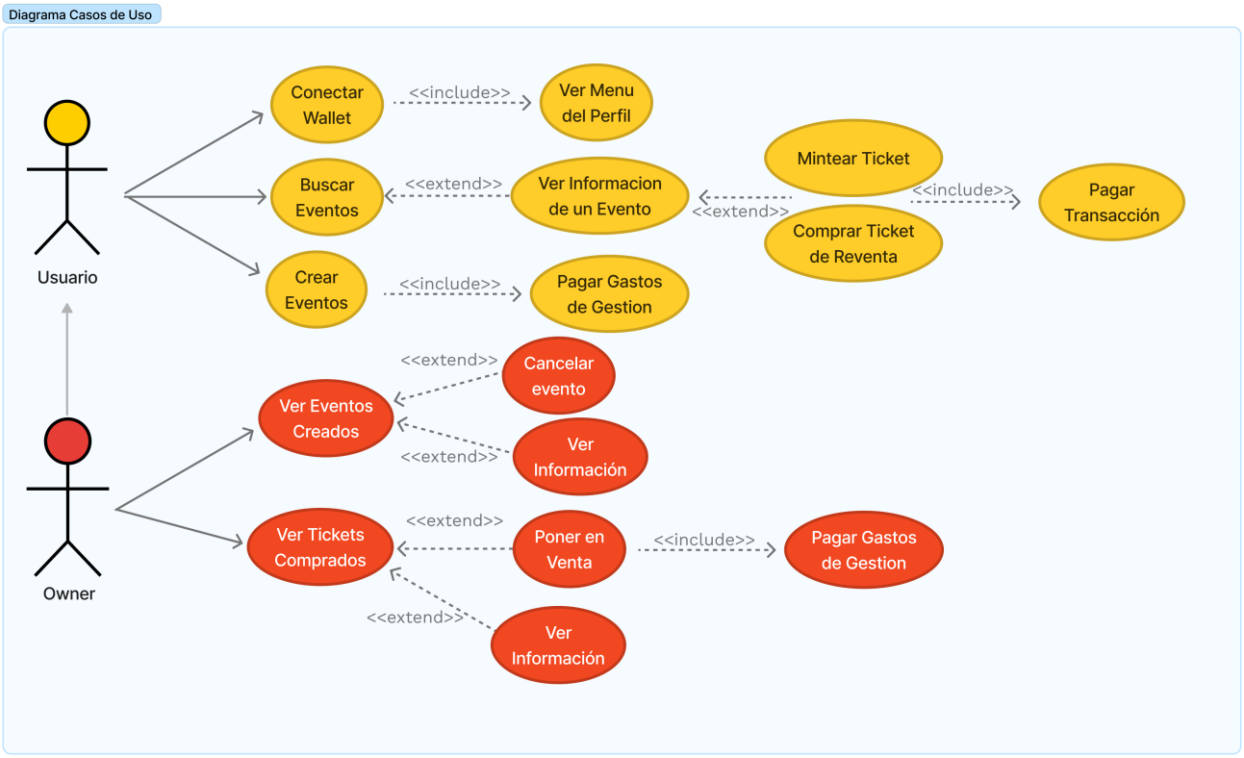

Figura 29. Diagrama de Casos de Uso

5.2 DISEÑO

En el mundo del desarrollo de software, existen diversas formas de desarrollar una misma idea, y una de las tareas más complicadas e importantes que hay es decidir de que manera vamos a enfocar el problema a resolver. En este apartado se explicarán las decisiones de diseño y porqué se han tomado.

ARQUITECTURA

Con motivo de realizar un análisis más comprensible, se ha decidido dividir la arquitectura en los distintos conjuntos que la conforman y sus principales componentes:

- Aplicación :
  - Front-end
  - File Server
  - API Server
- Proveedor
  - Metamask
  - Web3 Storage
- Blockchain
  - Contratos Inteligentes
  - Ganache

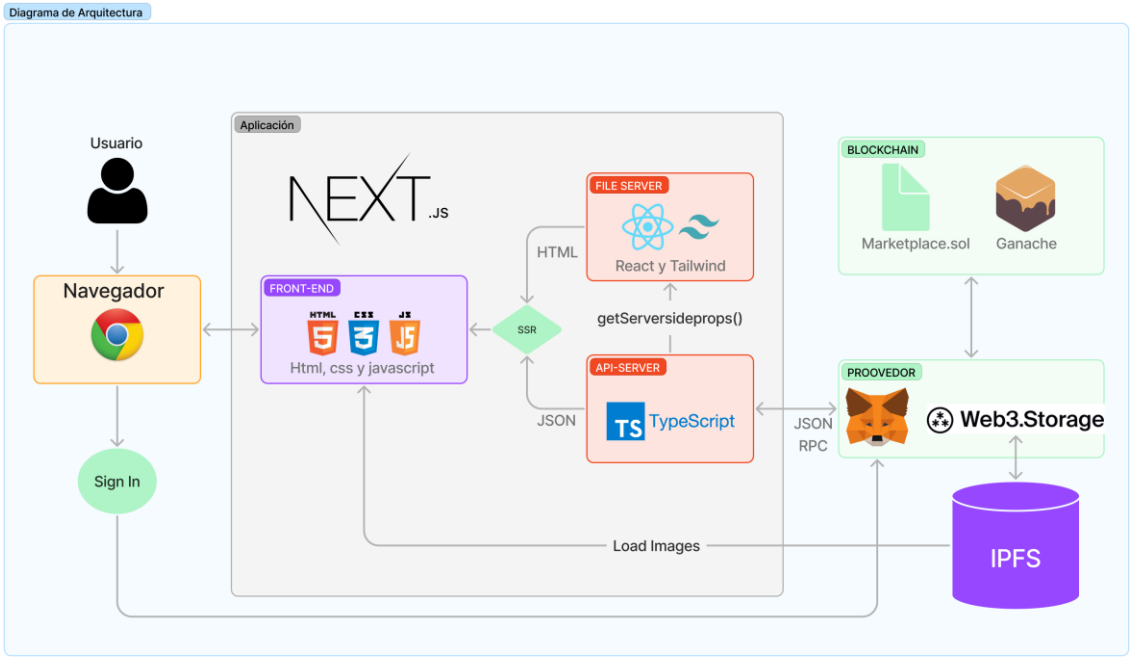

Figura 30. Diagrama de la arquitectura de la aplicación

El conjunto de Aplicación se ha realizado mediante Next.js. Next.js es un framework de desarrollo que permite funcionalidades de React. Una de estas funcionalidades es el Server-Side Rendering (SSR), que es una técnica usada para producir una respuesta personalizada para cada usuario mediante una separación de funciones.

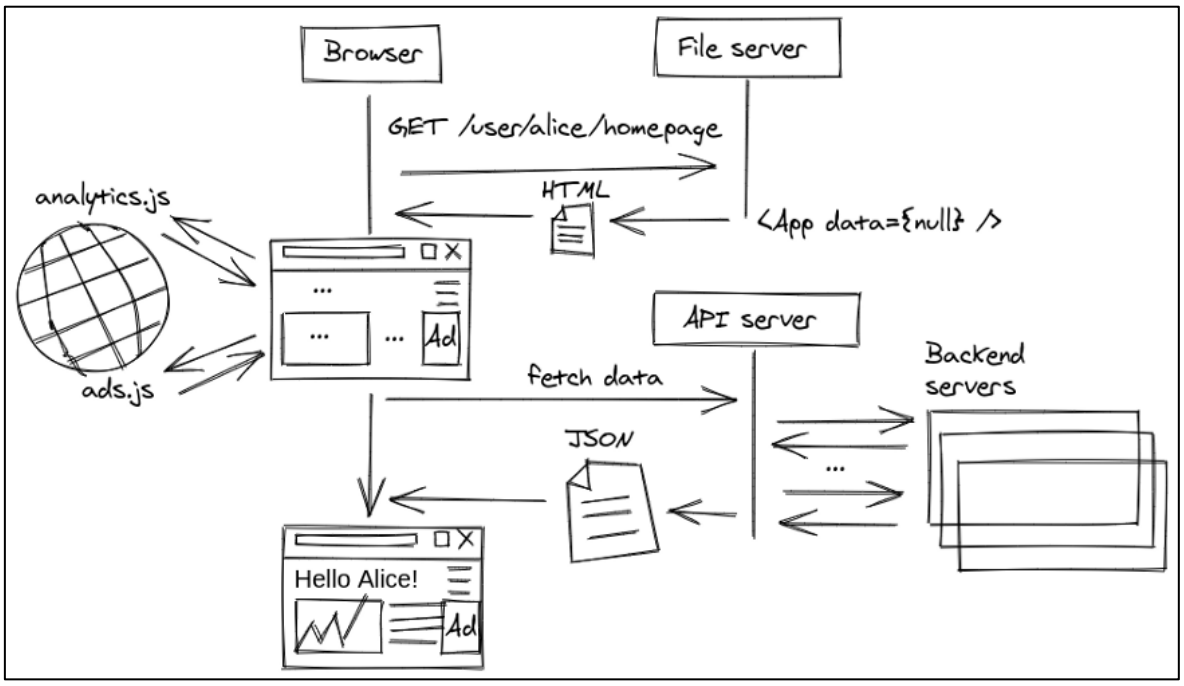

Figura 31. Esquema del funcionamiento del Server Side Rendering [8]

Por una parte esta el **File Server**, que se encarga de recibir peticiones de componentes y de páginas, y por otra parte se encuentra el **API Server**, que recibe peticiones de información y devuelve los datos en formato .JSON. Estos dos componentes se comunican a través de una función que implementa el propio Next.js que se llama **getServerSideProperties.** De esta manera, y aprovechando las funcionalidades que ofrece dicha división, se conforma el **front-end**, que a su vez se comunica con el navegador a través de HTML, CSS y Javascript. De esta manera, es posible que cada usuario reciba un archivo HTML personalizado consiguiendo una comunicación más dinámica y rápida.

Después, nos encontramos con el Proveedor. Este conjunto es más o menos un intermediador. Se encuentra en el medio de cualquier comunicación que este relacionado con web 3.0. Lo componen principalmente **Metamask** y **Web3 Storage**. **Metamask** es una

cartera digital y es capaz de conectarse a redes Blockchain, tanto locales como públicas. También, almacena direcciones de memoria de las Blockchain con sus claves privadas asociadas para dar autentificación y realiza tareas como: llamar a una funciones de contratos, realizar los pagos, autentificar cuentas, etc. Por otro lado, **Web3 Storage** es una plataforma que permite almacenar datos de manera descentralizada agregándoles una dirección. Trabaja con IFPS y Filecoin, y en el sistema se encarga de almacenar los archivos que se *tokenizan* y devolver el código hash asociado al archivo, que se almacena en el contrato inteligente.

Por último, se encuentra el conjunto de Blockchain. Representa esencialmente la cadena de bloques y los contratos que han sido desplegados en ella. Se podría etiquetar como el bloque más importante, ya que es el motor de la aplicación. Realiza funciones muy diversas e implementa la lógica del Marketplace, los Eventos y los Tickets. La blockchain en la que se ha decidido desplegar los contratos se llama **Ganache**. **Ganache** crea una blockchain a nivel local, proporcionando, además, un elevado número de cuentas con un saldo ficticio para poder hacer transacciones y testear los contratos. En cuanto a los **contratos**, están escritos en Solidity y se despliegan usando Truffle, otra herramienta complementaria a Ganache. Los **contratos** serán desmembrados más adelante, en el apartado de Implementación.

REQUISITOS FUNCIONALES

A continuación, se expondrán los Requisitos Funcionales más destacables del sistema desarrollado. Estos, tienen como objetivo definir las funciones de la aplicación como un conjunto de entradas, salidas y comportamientos. Fijándonos en los Casos de Uso descritos en el apartado anterior, se procede a desarrollar los siguientes requisitos:

- RF-01: Conectar Cartera Digital

RF- 01	Conectar Cartera Digital	
Versión	1.0	
Autores	Antonio Capilla	
Objetivos asociados	Identificación en la aplicación	
Descripción	El usuario quiere identificar una de dirección a una cuenta en el sistema a través de una cartera digital.	
Precondición	El usuario debe de: tener instalada la extensión de Metamask, haber introducido la red Blockchain y tener una cuenta en la plataforma con las claves privadas de las direcciones a usar.	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario entra en la aplicación y se le direcciona a la página principal.
	2	El usuario pulsa en el botón encontrado en la barra de navegación de “Conectar Cartera”.
	3	En la barra de navegación ya no aparece el botón de “Conectar Cartera” y en su lugar se encuentra un icono de perfil con diferentes apartados.
Postcondición		
Excepciones	**Paso**	**Acción**
	2.1	Metamask lanza una ventana flotante con un listado de las cuentas enlazadas.
	2.2	El usuario selecciona la cuenta que quiere enlazar.
Importancia	Alta	
Urgencia	Alta	

Figura 32. Requisito funcional: Conectar Cartera Digital

- RF-02: Crear un Evento

RF- 02	Crear un Evento	
Versión	1.0	
Autores	Antonio Capilla	
Objetivos asociados	Crear un evento	
Descripción	El usuario quiere crear un evento	
Precondición	El usuario debe de haber conectado su cartera digital con la aplicación (RF-01) y tener saldo en la cuenta seleccionada.	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario pulsa en el botón ubicado en la barra de navegación "Crear Evento"
	2	Se le dirige a una pantalla con un formulario con diferentes campos.
	3	El usuario rellena los campos correctamente y pulsa en el botón de "Crear Evento" ubicado al final del formulario.
	4	Se dirige al usuario a la página de "Mis Eventos", donde podrá interactuar con el evento recién creado.
Postcondición		
Excepciones	**Paso**	**Acción**
	1.1	Si el usuario no ha conectado su cartera digital la página no le cargará.
	3.1	Metamask acciona una ventana flotante con la información correspondiente a la transacción.
	3.2	El usuario acepta las operaciones, confirmando el pago de las tasas de creación de eventos y de los gastos de realizar la gestión.
Importancia	Alta	
Urgencia	Alta	

Figura 33. Requisito funcional: Crear un Evento

- RF-03: Mintear entrada

RF- 03	Mintear un Ticket	
Versión	1.0	
Autores	Antonio Capilla	
Objetivos asociados	Mintear una entrada	
Descripción	El usuario quiere tokenizar la entrada de un evento.	
Precondición	El usuario debe de haber conectado su cartera digital (RF-01) y tener saldo en la cuenta conectada. Debe de haber eventos disponibles en la plataforma.	
Secuencia Normal	**Paso**	**Acción**
	1	El usuario busca en la sección de explorar el evento del que le interesa comprar una entrada.
	2	El usuario selecciona un evento y se le direcciona a una página con la información de ese evento.
	3	En la página de información el usuario pulsa en el botón de "Mintear Ticket".
	4	El usuario es dirigido a la página de Mis Tickets, donde podrá interactuar con el ticket recién minteado.
Postcondición		
Excepciones	**Paso**	**Acción**
	3.1	Metamask acciona una ventana flotante con la información correspondiente a la transacción.
	3.2	El usuario acepta las operaciones, confirmando el pago del precio de minteo y de los gastos de realizar la gestión.
Importancia	Alta	
Urgencia	Alta	

Figura 34. Requisito funcional: Mintear un Ticket

INTERFAZ DE USUARIO

Cuando se realiza un proyecto de desarrollo de software, hay que tener en cuenta quién va a hacer uso de dicho software. Basándose en las corrientes estéticas y funcionales que nos encontramos en la gran mayoría de interfaces de aplicaciones del mercado, se ha diseñar una interfaz de usuario centrada en la usabilidad y la escalabilidad.

5.2.1.1 Diagrama de Navegabilidad

Antes de comenzar a construir las páginas y los componentes, es necesario hacer un estudio del funcionamiento de la aplicación a nivel de navegabilidad. Desde un primer momento, se ha decidido mantener un flujo de navegación sencillo, para que la usabilidad no se vea afectada y al usuario no pierda tiempo en averiguar el funcionamiento.

Además, se ha realizado un diseñado con el objetivo de implementar el menor número de páginas. Esto implicará tener que, de alguna manera, realizar páginas que sean dinámicas y que puedan usarse en distintos escenarios, tema que se tratará con más extensión en el apartado de Implementación.

Teniendo en cuenta estas consideraciones, se ha realizado el siguiente diagrama de navegación del aplicación:

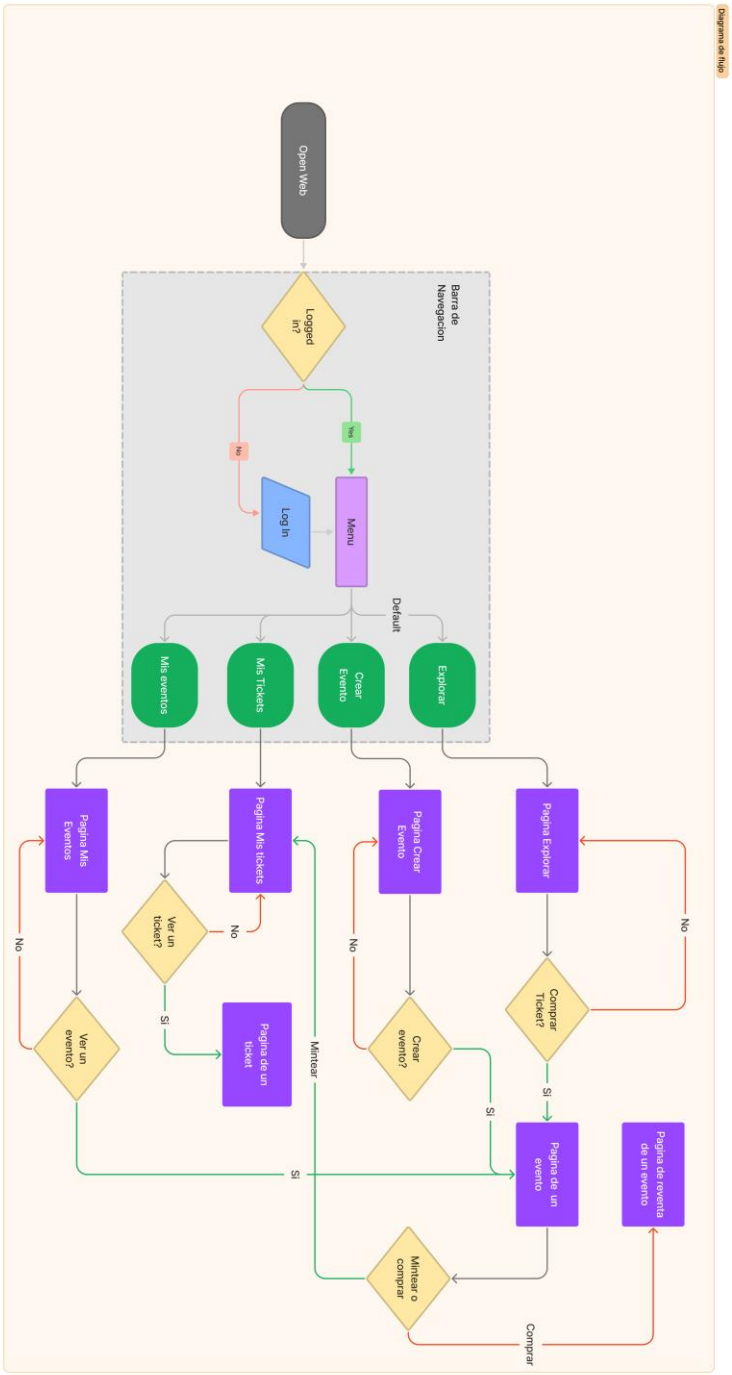

Figura 35. Diagrama de Navegabilidad

En la figura se observa el diagrama la completo. Si nos fijamos en la derecha del esquema se encuentra el grupo que conforma la barra de navegación y desde el que es posible acceder a todas las funcionalidades de la aplicación. Además, implementa todo el flujo relacionado con el sistema de identificación de los usuarios. También, cabe destacar que solo se podrá acceder a las páginas de "Mis Eventos" y de "Mis Tickets" una vez se haya identificado el usuario.

5.2.1.2 Esquema de las Páginas

Usando de referencia el diagrama de navegación del apartado anterior, recopilaremos el conjunto de páginas a diseñar. Se realizará un esquema de cada página en donde se describirá los componentes que la componen y su disposición en la misma. Las páginas diseñadas son:

- Ppágina "Explorar"
  - Barra de navegación
  - Titulo de la página
  - Cuadricula con las tarjetas de todos los eventos disponibles
- Página "Crear Evento"
  - Barra de navegación
  - Formulario para la creación de un evento
- Página "Mis Eventos"
  - Barra de navegación
  - Título de la página
  - Cuadricula con las tarjetas de todos los eventos que ha creado una cuenta
- Página "Mis Tickets"
  - Barra de navegación
  - Titulo de la página
  - Cuadricula con las tarjetas de todos los tickets que posee una cuenta

5.3 IMPLEMENTACIÓN

En este apartado se detallarán las soluciones que se han adoptado para lograr implementar los diseños ideados en el apartado anterior. Las principales herramientas que se han usado en el proyecto, detalladas en el Capitulo 1, son las siguientes:

- Visual Studio Code: editor de código
- Next: framework de la aplicación
- Tailwind: *style* de los componentes
- Truffle: compilar, testear y desplegar los contratos en la red
- Ganache: red blockchain local

Estas herramientas conforman el ambiente donde se ha desarrollado el sistema, y se configurarán más adelante.

Para finalizar la introducción a este apartado, se mostrará un diagrama de las carpetas que componen el proyecto, a modo de estructuración de este capítulo:

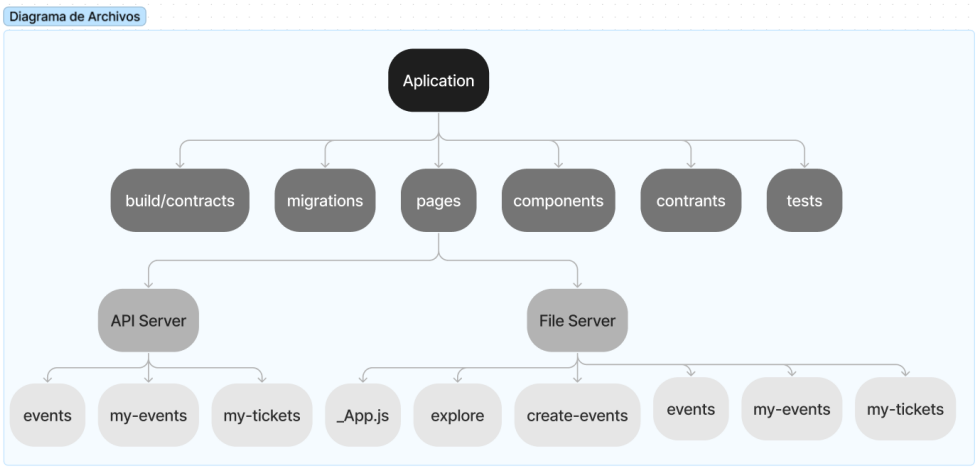

Figura 36. Diagrama de archivos del proyecto

Como se puede ver en la figura, encontramos numerosas carpetas dentro del sistema. Las carpetas de *build/contracts, migrations, contracts y tests* se analizaran en el apartado de Contratos Inteligentes y el resto de carpetas se verán en el apartado de Aplicación.

LIBRERÍAS

En este proyecto se han usado numerosas librerías. Al igual que Next.js, todas estas las librerías se manejan a través de *npm*, el cual es un sistema de gestión de archivos de Node.js. Este sistema archivará en una carpeta que el mismo creará, denominada *node_modules*, toda la biblioteca de archivos descargados que se quieran usar en el proyecto. Además, el gestionador inicializará unas dependecias en el archivo del proyecto *pachake-lock.json.* De esta manera si queremos traspasar el proyecto, no tendremos que incluir la carpeta *node_modules*, que almacena gran cantidad de información, ya que se incluirá automáticamente una vez introducido el comando *npm install.*

Ahora, se listaran las librerías más relevantes seguidas de una breve descripción del uso que se les ha dado durante el desarrollo del sistema:

- ethers
- web3modal
- @openzeppelin/contracts
- web3.storage
- axios
- @thirdweb-dev/react
- tailwindcss

CONFIGURACIÓN DEL PROYECTO

A continuación, se enumerarán los pasos que se realizaron para la configuración completa del proyecto:

1. Comenzamos creando el proyecto usando Next.js y el sistema de gestión de paquetes yarn, usando el comando:

yarn create next-app event-marketplace

2. Configuramos las dependencias y las librerías que vamos a usar en el proyecto, *yarn add*.
   a. Ethers
   b. Ethereum-waffle
   c. Web3modal
   d. @Openzeppelin/contracts
   e. Web3.Storage
   f. Axios
   g. @Thirdweb-dev
   h. Tailwindcss
   i. Postcss
   j. Autoprefixer
   k. @nomiclabs
   l. Hardhat

Comando completo:

yarn add ethers hardhat @nomiclabs/hardhat-waffle @nomiclabs/hardhat-ethers ethereum-waffle chai web3modal @openzeppelin/contracts ipfs-http-client@50.1.2 axios @thirdweb-dev/react @thirdweb-dev/sdk

3. Inicializamos Tailwindcss:

npx tailwindcss init -p y

4. Editamos el archivo *tailwind.config.js* añadiendo la ruta donde escribiremos el código .js y .jsx, y así podamos usar los estilos de Tailwind.

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    "./pages/**/*.{js,ts,jsx,tsx}",
    "./components/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        main: "#04111D",
        primary: "#576068",
        blue: "#2081E3",
      },
    },
  },
  plugins: [],
};
```

Figura 37.Configuración de tailwind.config.js

5. Importamos los estilos de Tailwind en el archivo global.css, dentro de la carpeta *styles*. Esta carpeta la crea por defecto Next.js.

```
/* ./styles/globals.css */
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Figura 38. Configuración de /styles/globals.css

6. Inicializamos Truffle en el proyecto para compilar, testear y desplegar los contratos. Introducimos el comando:

truffle init

7. Nos dirigimos al archivo de *truffle-config.js*, creado por el comando anterior, e introducimos la red que deseamos usar por defecto como red *development*:

```
networks: {

  development: {
   host: "127.0.0.1",     // Localhost (default: none)
   port: 9545,            // Standard Ethereum port (default: none)
   network_id: "*",       // Any network (default: none)
  },
```

Figura 39. Configuración de la red blockchain local en Truffle

Esta red que estamos añadiendo es la correspondiente a nuestra blockchain creada en Ganache. El puerto puede variar dependiendo de la instancia que se haya creado en Ganache. Si nos dirigimos a la interfaz:

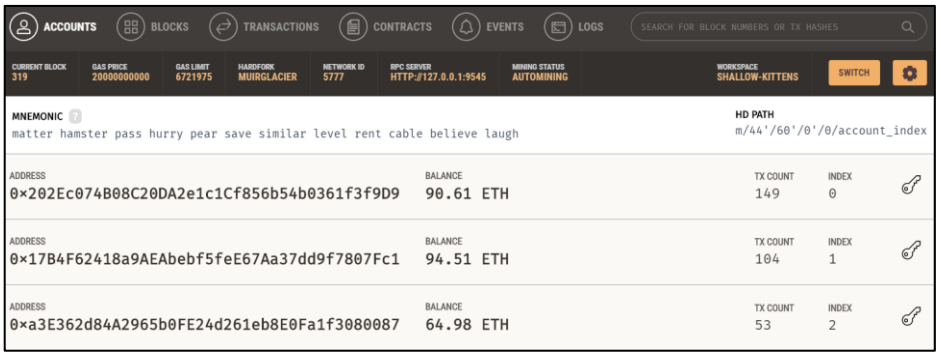

Figura 40. Vista de la interfaz de Ganache

Si nos fijamos en la figura, podremos ubicar en la barra superior un apartado denominado "RPC SERVER", a través de ese puerto nos comunicaremos.

Nota: En esta configuración se ha usado el sistema de gestión *yarn* y *npx*, pero funcionan de la misma manera que *npm*.

Ahora se incluirán los comandos asociados a esta configuración:

- Desplegar la aplicación en la dirección https://localhost300:

- *npm run dev*

- Compilar los contratos ubicados en la carpeta */contracts*:
  - *truffle compile*
- Correr los archivos dedicados al testing de los contratos:
  - *truffle test*
- Desplegar los contratos de la carpeta */contracts*:
  - *truffle migrate*
- Acabar con una actividad en un puerto
  - *npx kill-port 3000*

CONTRATOS INTELIGENTES

Los contratos inteligentes, como se estableció en la Definición del Trabajo, corresponden seguramente al engranaje más importante de todo el proyecto. Su correcto funcionamiento es de vital importancia para que el resto de módulos puedan operar.

Comenzaremos a explicar este apartado mostrando en la figura, un diagrama UML de las relaciones entre dichos contratos. Para entenderlo, hay que saber que se van a desarrollar dos contratos: *Event.sol* y *Marketplace.sol*.

Siguiendo la linea que se comenzó en el capitulo de Estado de la Cuestión, la idea es usar el estándar ERC721, o comúnmente conocido como NFT o token no fungible, para convertirlo en un contrato en donde se almacenen los datos específicos de cada evento y sea posible minar entradas nuevas en él. Este contrato que hereda directamente del estándar nombrado anteriormente, será el llamado *Evento.sol*. Un poco más adelante analizaremos su código.

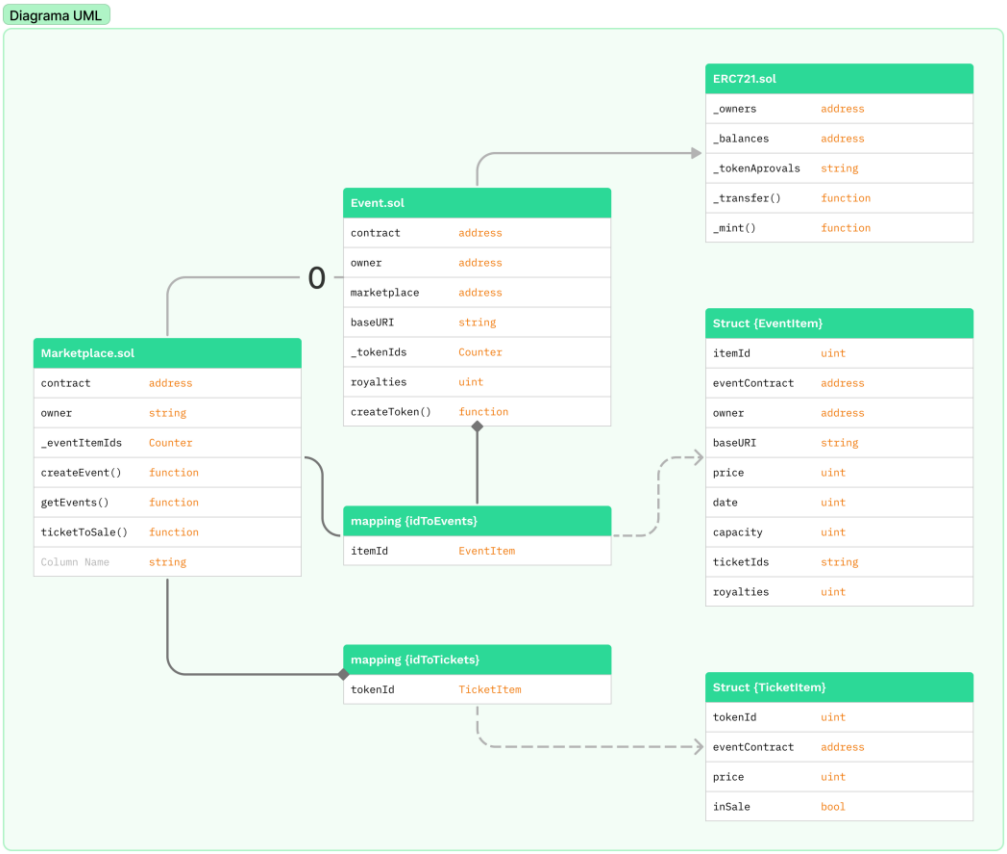

Figura 41. Diagrama UML de los contratos inteligentes

Por otro lado, debe de existir una forma de automatizar la creación de estos contratos Evento y la gestión de los mismos. Para ello, se crea el contrato *Marketplace.sol*. Las funciones que este contrato debe poder realizar son las siguientes:

- Construir contratos *Evento.sol* con las características que usuario introduzca y desplegar dichos contratos en la red.

- Almacenar los eventos creados con su correspondiente dirección de contrato.
- Llevar cuenta del número de eventos creados, el número de tickets minteados y el número de tickets que posee cada usuario.
- Ser capaz de proporcionar toda la información que almacena para que la Aplicación pueda mostrarla.
- Debe de poder transferir un ticket ya *minteado* de una dirección a otra, interactuando con el contrato del evento.

Una vez explicada la lógica de los contratos procedemos a analizarlos.

5.3.1.1 análisis del Código

Los contratos están escritos en *Solidity*, que es un lenguaje de programación orientado a objeto. Empezaremos analizando el contrato *Evento.sol*.

Evento.sol

Como ya explicamos en la introducción de este apartado, este contrato hereda directamente del ERC721. A su vez, el contrato ERC721 hereda del ERC165, e implementa las interfaces IERC721 y IERC721Metadata. Esto proporciona a nuestro contrato una serie de argumentos y funciones que no serán de gran utilidad. Todos estos contratos los obtendremos de la librería de *@openzeppelin/contracts*, y ya analizamos el ERC721 en el Capítulo 2. Para este contrato nos servirá con repasar el funcionamiento de la función _mint:

```
function _mint(address to, uint256 tokenId) internal virtual {
    require(to != address(0), "ERC721: mint to the zero address");
    require(!_exists(tokenId), "ERC721: token already minted");

    _beforeTokenTransfer(address(0), to, tokenId);

    _balances[to] += 1;
    _owners[tokenId] = to;

    emit Transfer(address(0), to, tokenId);

    _afterTokenTransfer(address(0), to, tokenId);
}
```

Figura 42. Función _mint del token ERC-721

Como se observa, es una función de ámbito *internal* que equivale "más o menos" a *private*[1], por lo que solo se podrá llamar desde el contrato actual o los contratos que hereden de este. Los argumentos de la función son una dirección y un valor de tipo *uint* denominado *tokenId*. Es importante anotar que ninguno de estos argumentos están definidos o provienen del propio contrato, por lo que tendremos que definirlos en nuestro contrato al llamar a esta función.

La función comienza comprobando que la dirección que se ha introducido como argumento no sea la dirección 0x0 y que el tokenId no este siendo usado. Después, procede a sumar un 1 en la posición del valor de la dirección en la variable de tipo *mapping* llamada *_balances.* Realiza la acción inversa en la variable *_owners* que también es un *mapping*, pero en este caso guarda la dirección de entrada en la posición del valor de tokenId. Finaliza la función emitiendo un evento heredado de la IERC721 que se llama Transfer, y que indica que un token ha sido traspasado en este caso de la dirección 0x0, ya que se esta creando, a la dirección que se introduzco a la función.

Retomando el contrato *Evento.sol*, podemos ver en la figura el constructor, que recibe como argumentos: la dirección del *Marketplace.sol* desplegado, la baseURI de la imagen del *flyer* del evento, el precio de minteo, la fecha del evento y el número máximo de entradas.

```
//Constructor del Evento
constructor(
    address _eventMarketplace,
    string memory baseURI,
    uint _price,
    uint _date,
    uint _capacity
) ERC721("EventPrueba", "EP") {
    eventMarketplace = _eventMarketplace;   // Guardamos la direccion del contrato EventMarketplace
    owner = msg.sender; // Guardamos el propietario del evento
    price = _price; // Guardamos el precio de la entrada
    _setBaseURI(baseURI);   // Configuramos la URI base para todos los tokens
    date = _date; // Guardamos la fecha del evento
    capacity = _capacity; // Guardamos la capacidad del evento
}
```

Figura 43. Constructor del contrato Evento.sol

Una de las funciones que implementa el contrato es la función *createNewToken*, que gestiona el *mineto* de los eventos:

```
function createNewToken(address to) public returns (uint256){
    _tokenIds.increment();  //Incrementamos el indice para añadir un nuevo token
    uint256 newTokenId = _tokenIds.current();   //Obtenemos el id del nuevo token
    _mint(to, newTokenId);  // Minteamos el token

    return newTokenId;
}
```

Figura 44. Función de createNewToken

La función recibe como argumento la dirección que se desea asociar al token, no se puede usar *msg.sender* porque el usuario no va a llamar directamente a esta función, el usuario llamará a una función de *Marketplace.sol* y este llamará a la función *createNewToken*. Además, aparece una variable llamada *_tokenIds* que se trata de un contador creado de manera global, de tipo Counter, de la librería de @openzeppelin-sdk/contracts/utils.

Marketplace.sol

Podemos comenzar el análisis de este contrato, describiendo las estructuras que sean tenido que crear dentro del contrato. Hay dos estructuras: EventItem y TicketItem. El objetivo de estas estructuras es tener unos objetos con los que poder trabajar dentro del Marketplace, de manera que se pueda almacenar más información que la que reside en el contrato de *Evento.sol*.

```
struct TicketItem {
    uint itemId;    // Id del item
    uint eventId; // Id del evento
    address payable eventContract;  // Direccion del contrato del Evento
    address payable owner;  //Organizador del evento
    uint price; // Precio del minteo del ticket, el precio del ticket sera otro si es de reventa
    bool inSale; // Si el ticket se ha vendido
}
```

```
// Estructuras y eventos
struct EventItem {
    uint itemId;    // Id del item
    address payable eventContract;  // Direccion del contrato del Evento
    address payable owner;  //Organizador del evento
    string name; // Nombre del evento
    string baseURI; // URI base para los tokens
    uint price; // Precio del minteo del ticket, el precio del ticket sera otro si es de reventa
    uint date; // Fecha del evento
    uint dateIni; // Fecha de inicio de venta
    uint capacity; // Capacidad del evento
    uint numTokens; // Numero de tokens del evento
    string description; // Descripcion del evento
}
```

Figura 45. Estructuras descritas en el contrato Marketplace.sol

El constructor del contrato, es sencillo, y solo inicializa el owner del contrato que se establece como *payable* para poder recibir pagos. Y por último, antes de analizar las

funciones del contrato debemos inicializar unos *arrays* dinámicos para gestionar los EventItems, los TicketItems y los balances de los dos con respecto a las direcciones de los usuarios.

```
// Los arrays dinamicos
mapping(uint => EventItem) private idToEventItem;   // Array de eventos
mapping(uint => mapping(uint => TicketItem)) private idToTicketItem;    // Array de tickets vendidos de cada evento
mapping(address => uint) private balancesTickets; // Array de balances de tickets de cada usuario
mapping(address => uint) private balancesEvents; // Array de balances de eventos de cada usuario
```

Figura 46. Arrays dinámicos del contrato Marketplace.sol

Ahora, se explicará el código de las funciones más importantes del contrato:

- createEvent:

```
function createEvent(
    string memory name,
    string memory baseURI,
    uint price
) public payable returns (EventItem memory) {
    require(price > 0, "El precio del evento debe ser mayor que 0");
    require(msg.value == createFee, "Valor enviado insuficiente");

    _eventItemIds.increment();
    uint256 itemId = _eventItemIds.current();
    uint numTokens = 0;
    uint capacity = 100;

    uint date = 29122022;
    uint dateIni = 29102022;
    // string memory name = "Evento ";
    string memory description = "Descripcion del evento ";

    balancesEvents[msg.sender] += 1;

    // Creamos el contrato del evento y guardamos su direccion
    address eventContract = address(new Event(address(this),baseURI,price,date,capacity));

    // Creamos el EventItem enlazado al evento
    // Lo indexamos en el mapping de idToEventItem
    idToEventItem[itemId] = EventItem(itemId,payable(eventContract),payable(msg.sender),name

    // Devolvemos el id del item del evento para trabajar con el en el front
    return idToEventItem[itemId];
}
```

Figura 47. Función de createEvent

Es una función que como su nombre indica, sirve para crear un evento. Despliega el contrato correspondiente de *Event.sol*, inicializando correctamente el constructor. Tambíen, añade a la variable *idToEventItem* el nuevo EventItem creado.

- mintTicket:

```
function mintTicket(
    uint eventItemId
) public payable {
    uint price = idToEventItem[eventItemId].price;

    require(msg.value == price, "Cantidad de gas incorrecta");

    address payable eventContract = idToEventItem[eventItemId].eventContract;
    //emit Message(eventContract);
    uint newTokenId = Event(eventContract).createNewToken(msg.sender);

    //eventContract.transfer(msg.value);

    //Incrementamos el numero de tokens minteados
    idToEventItem[eventItemId].numTokens = newTokenId;

    balancesTickets[msg.sender] += 1;

    //Añadimos el item al mapping
    idToTicketItem[eventItemId][newTokenId] = TicketItem(
        newTokenId,
        eventItemId,
        payable(eventContract),
        payable(msg.sender),
        price,
        false
    );
}
```

Figura 48. Función de mintTicket

Esta función sirve para mintear una entrada de un evento, solo le es necesario el id del EventItem para localizar la dirección del contrato al que esta enlazado y mintear una entrada. Al llamar a la función *createNewToken* hay que introducir de argumento la dirección del que ha llamado a la función *mintTicket*, para que quede asociada su dirección al token en el contrato *Event.sol*.

- getTicketsOfAccount:

Esta función, se encarga de recorrer los arrays dinámicos de EventItems y TicketItems, almacenando los TicketItems que correspondan a la dirección que se introduce como argumento.

```
function getTicketsOfAccount(address user) public view returns(TicketItem[] memory){
    uint eventItemCount = _eventItemIds.current();
    uint index = 0;

    uint itemCount = balancesTickets[user];

    TicketItem[] memory tickets = new TicketItem[](itemCount);

    for(uint i = 0; i < eventItemCount; i++){
        uint ticketItemCount = idToEventItem[i+1].numTokens;
        for(uint j = 0; j < ticketItemCount; j++){
            if(idToTicketItem[i+1][j+1].owner == user){
                uint currentId = idToTicketItem[i+1][j+1].itemId;
                TicketItem storage currentTicket = idToTicketItem[i+1][currentId];
                tickets[index] = currentTicket;
                index += 1;
            }
        }
    }

    return tickets;
}
```

Figura 49. Función de getTicketsOfAccount

- ticketToSale:

```
function ticketToSale(
    uint eventItemId,
    uint ticketItemId,
    uint price
) public {
    require(idToTicketItem[eventItemId][ticketItemId].inSale == false, "El ticket ya esta en venta");
    require(idToTicketItem[eventItemId][ticketItemId].owner == msg.sender, "No eres el dueño del ticket");

    //Cambiamos el estado del ticket
    idToTicketItem[eventItemId][ticketItemId].inSale = true;
    idToTicketItem[eventItemId][ticketItemId].price = price;
}
```

Figura 50. Función de ticketToSale

Esta función recibe como argumentos los identificadores del evento del ticket con respecto al evento y el nuevo precio para el mercado de reventas, y realizando una previa comprobación de que el remitente es el dueño del ticket, cambia el estado del TicketItem a en venta a través de la variable *inSale.*

- transferTicket:

```
function transferTicket(
    uint eventItemId,
    uint itemId
) public payable nonReentrant{
    uint price = idToTicketItem[eventItemId][itemId].price;
    uint tokenId = idToTicketItem[eventItemId][itemId].tokenId;
    address eventContract = idToTicketItem[eventItemId][itemId].eventContract;
    address payable seller = idToTicketItem[eventItemId][itemId].owner;

    require(msg.value == price, "El precio debe de ser igual al precio del ticket");

    payable(seller).transfer(msg.value); //Enviamos el dinero al vendedor
    IERC721(eventContract).transferFrom(
        seller,
        payable (msg.sender),
        tokenId
    );
    idToTicketItem[eventItemId][itemId].inSale = true;
    _ticketItemSold.increment();
}
```

Figura 51. Función de transferTicket

Esta función tiene el objetivo de transferir un ticket que se encuentra en venta, *inSale = true*, en el mercado de reventas de un evento. Primero, se comprueba que el remitente ha enviado con la transacción el precio del ticket correspondiente. Entonces, se procede a enviar dicha cantidad al antiguo propietario del ticket y se llama a la función *transferFrom*, correspondiente al ERC721, para transferir el token al nuevo dueño.

5.3.1.2 Test

Si cuando se desarrolla una aplicación de uso corriente, es esencial dedicarle tiempo a los test, cuando se trata del desarrollo de aplicaciones basadas en contratos inteligentes es aún más importante. Los contratos inteligentes son difíciles de probar a menos de que se realicen

test escritos. Estos test se pueden escribir de distintas maneras, en este desarrollo se ha decidido implementarlo usando JavaScript, en línea con el proyecto.

Cuando Truffle se inicializó, se creó una carpeta llamada */tests*. Para comenzar a interactuar con los contratos primero hay que crear un archivo .js para el test en dicha carpeta y compilar los contratos usando el comando *truffle compile*. Después, a través de la función *artifacts.require* obtendremos el .json fruto de la compilación. Truffle facilita la implementación de un test gracias a una la función *contract*, que le indica a Truffle que se está refiriendo a un test y aporta un vector de cuentas para realizar transacciones.

```
const Marketplace = artifacts.require('Marketplace');

contract("Event Marketplace",  (accounts) => {
```

Figura 52. Inicialización de un test en Truffle

Una vez iniciado el archivo usando la función *contract*, dividiremos el test del contrato en distintas actividades a través de las sentencias *it*. Ahora se mostrarán unos ejemplos de test que se han escrito:

- Crear un evento:

```
it("Deberia crear los eventos", async function () {

    const marketplace = await Marketplace.deployed()
    // Obtenemos las cuentas
    const owner = accounts[0]
    const organizer1 = accounts[1]

    let createFee = await marketplace.getCreateFee({from: owner})
    createFee = createFee.toString()

    const eventPrice = ethers.utils.parseUnits('1', 'ether')
    const eventName = "Evento 1"
    const eventDescription = "Descripcion del evento 1"
    const eventDate = "2020-01-01"
    const eventDateIni = "2020-01-01"

    const imgUrl = "https://bafybeib7aubgpsl3hr5xudx4gjmkc5ks7qc6nw6xemb7
    const event1 = await marketplace.createEvent(
        eventName,
        imgUrl,
        eventPrice,
        {
            value: createFee,
            from: organizer1
        })
});
```

Figura 53. Test para crear un evento

- Listar los eventos disponibles:

```
it("Deberia listear los eventos disponibles", async function () {

    const marketplace = await Marketplace.deployed()

    let events = await marketplace.getEvents()
    events = await Promise.all(events.map(async i => {
        return {
            name : i.name,
            price: i.price.toString(),
            address: i.eventContract,
            itemId: i.itemId.toString(),
            owner: i.owner,
            date: i.date,
            dateIni: i.dateIni,
            description: i.description,
            capacity: i.capacity.toString(),
            baseURI: i.baseURI,
            numTokens: i.numTokens.toString()
        }
    }))

    console.log('Eventos disponibles en el Marketplace: ', events)

});
```

Figura 54. Test para listar los eventos disponibles

- Mintear un ticket:

```
it("Deberia mintear un ticket del evento disponible", async function () {
    const marketplace = await Marketplace.deployed()
    //let IMarketplace = new ethers.utils.Interface(Marketplace.abi)

    // Obtenemos las cuentas
    const owner = accounts[0]
    const organizer1 = accounts[1]
    const buyer = accounts[2]

    let createFee = await marketplace.getCreateFee({from: owner})
    createFee = createFee.toString()

    const eventPrice = ethers.utils.parseUnits('1', 'ether')
    const eventPriceString = eventPrice.toString()

    let ticket1 = await marketplace.mintTicket(1, { value: eventPriceString, from: buyer })
});
```

Figura 55. Test para mintear un ticket

Una vez escrito los test, haciendo uso del comando *truffle test*, se ejecutarán para comprobar que los contratos funcionan correctamente. En la siguiente figura se podrán observar los resultados en la consola, de los test que se han mostrado anteriormente.

```
C:\Users\Anto\OneDrive - Universidad Pontificia Comillas\ICAI\Cuarto\TFG\Desarrollo\event-marketplace>truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Compiling .\contracts\Event.sol
> Compiling .\contracts\Event.sol
> Compiling .\contracts\Marketplace.sol
> Compiling .\contracts\Marketplace.sol
> Artifacts written to C:\Users\Anto\AppData\Local\Temp\test--5960-qZa6kovXsv3B
> Compiled successfully using:
   - solc: 0.8.14+commit.80d49f37.Emscripten.clang
Marketplace desplegado en:  0xC7407b341e4A0d5e364Cd5f5Ed2f7D9024cb73ce


  Contract: Event Marketplace
    ✓ Deberia crear los eventos (5006ms)
Eventos disponibles en el Marketplace:  [
  {
    name: 'Evento 1',
    price: '1000000000000000000',
    address: '0x68827524Fc4B4BdB72C3F97c874efdd443b99679',
    itemId: '1',
    owner: '0x17B4F62418a9AEAbebf5feE67Aa37dd9f7807Fc1',
    date: '29122022',
    dateIni: '29102022',
    description: 'Descripcion del evento ',
    capacity: '100',
    baseURI: 'https://bafybeib7aubgpsl3hr5xudx4gjmkc5ks7qc6nw6xemb7qmwix2hw67hsfu.ipfs.dweb.link/the-prodigal-son-19
    numTokens: '0'
  }
]
    ✓ Deberia listear los eventos disponibles (598ms)
    ✓ Deberia mintear un ticket del evento disponible (2095ms)
    ✓ Deberia listar los tickets del evento 1 (293ms)
Tickets del comprador 1:  [
  {
    itemId: '1',
    eventContract: '0x68827524Fc4B4BdB72C3F97c874efdd443b99679',
    owner: '0xa3E362d84A2965b0FE24d261eb8E0Fa1f3080087',
    price: '1000000000000000000',
    inSale: false
  }
]
    ✓ Deberia listear los tickets del comprador (377ms)
```

Figura 56. Resultados de los tests en la consola

5.3.1.3 Despliegue

Una vez implementado el diseño de los contratos inteligentes, solo falta desplegarlos a la red. Como ya se ha comentado, de momento, la red Blockchain que alimentará a la aplicación está provista por Ganache. De los contratos que se han escrito, solo se va a desplegar el de *Marketplace.sol*, ya que el resto de despliegues han sido automatizados dentro del contrato.

Al inicializar Truffle, se creo también una carpeta llamada */migrations*, donde se deben escribir las distintas migraciones que necesite el proyecto. En el caso del sistema desarrollado, solo hará falta desplegar un contrato.

```
const Marketplace = artifacts.require("Marketplace");
const fs = require('fs');

module.exports = function (deployer) {
  deployer.deploy(Marketplace)
  .then(() => Marketplace.deployed())
  .then(() => console.log("Marketplace desplegado en: ", Marketplace.address))
  .then(() => fs.writeFileSync('./config.js', `
  export const marketAddress = "${Marketplace.address}"
  `))
};
```

Figura 57. Función para el despliegue del contrato Marketplace.sol

Como se observa en la figura, se vuelve a hacer uso de la función *require* de *artifacts*. Además, guardaremos en un archivo denominado *config.js* la dirección del contrato, ya que se usan en distintos archivos de la aplicación y de esta manera queda automatizado.

Una vez escrito el código de la migración introducimos el comando *truffle migrate* en la consola y se ejecutará el código. Ahora si nos dirigimos a la interfaz de Ganache comprobaremos, en el apartado de contratos, que el contrato se encuentra desplegado:

Figura 58. Demostración despliegue del contrato Marketplace.sol

APLICACIÓN

Si se recuerda, en el esquema de arquitectura expuesto en el apartado de Diseño, se definió un conjunto llamado Aplicación del que es extraían tres partes: el Frontend, el File Server y

la API Server. Comenzaremos tratando los *endpoints* implementados en la API del sistema, y después, trataremos los Componentes y el File Server de los cuales se formará el denominado Frontend.

5.3.1.4 API Server

Una de las grandes ventajas de usar Next, es que esta muy preparado para establecer un API. Cuando creas el proyecto, por defecto en la carpeta de Pages, se crea un directorio */api*. Next permite usar distintos lenguajes para escribir los archivos de la API, en este proyectos se ha decidido usar TypeScript. El funcionamiento es a través de rutas, como si se tratase de una página más de la aplicación. Para que se convierta en una ruta válida simplemente debe de ir dentro de la carpeta */api* e implementar como argumentos el *NextApiRequest y* el *NextApiResponse* de la librería de Next.

```
export default async (req: NextApiRequest, res: NextApiResponse) => {

    const eventID = req.query.eventID
    const ticketID = req.query.ticketID
```

Figura 59. Ejemplo de una función de ruta en TypeScript

Ahora se realizará un listado de las rutas de la aplicación de manera que, además, sirva como esquema de la organización de los archivos:

- GET
    - **/api/events** → devuelve un array con los eventos disponibles
    - **/api/events/[eventID]** → devuelve el evento asociado a un eventID
    - **/api/events/[eventID]/tickets** → retorna los tickets que se han minteado de de un evento asociado a un eventID

- **/api/events/[eventID]/tickets/[ticketID]** → devuelve un ticket concreto a través de sus identificadores de evento y de ticket
- **/api/my-events/[account]** → retorna los eventos que ha creado una determinada dirección
- **/api/my-tickets/[account]** → devuelve los tickets que posee una dirección concreta

- POST
  - **/api/create-event** → crea un evento
  - **/api/mint-ticket** → mintea un ticket
  - **/api/transfer-ticket** → transfiere un ticket de una cuenta a otra
  - **/api/ticket-to-sale** → establece el precio de un ticket y activa su variable *inSale*

Ahora, analizaremos la ruta */api/events/[eventID]/tickets/[ticketID]*, para explicar la interacción de la aplicación con la red blockchain.

```
export default async (req: NextApiRequest, res: NextApiResponse) => {

    const eventID = req.query.eventID
    const ticketID = req.query.ticketID

    //const web3 = new Web3(window.web3.currentProvider)
    const provider = new ethers.providers.JsonRpcProvider(url)
    const marketplaceContract = new ethers.Contract(marketAddress, Marketplace.abi, provider)

    let data = await marketplaceContract.getTickets(eventID)
    data = await Promise.all(data.map(async i => {
      //const baseURI = await axios.get(i.baseURI)
      const ethValue = ethers.utils.formatEther(i.price);
        return {
            id: i.itemId,
            eventContract: i.eventContract,
            eventId: i.eventId,
            owner: i.owner,
            price: i.price.toString(),
            inSale: i.inSale
        }
    }))

    const ticket = data.find((e) => {
        return e.id == ticketID;
      });

    res.send(ticket);
}
```

Figura 60. Función de ruta para /api/events/[eventID]/tickets/[ticketID]

Como se observa en la figura, primero obtenemos ,a través argumento *query* de la variable de la solicitud *req*, el identificador del evento y del ticket. Después, obtenemos el proveedor para realizar la conexión con la red desplegada en *http://localhost:9545* gracias al método *JsonRpcProvider* de la librería *ethers*. Se crea, una vez iniciada la conexión, una instancia del contrato usando el constructor *Contract*, también de la librería *ethers*. Por último, se llama a la función del contrato *getTickets* pasándole como argumentos los identificadores definidos al comienzo de la función. Se *parsea* el array devuelto por el contrato al formato de la aplicación, y se devuelve usando el método *send* de la variable de respuesta *res*.

5.3.1.5 Componentes

Como ya se ha comentado anteriormente, una de las características por las que se decidió usar Next.js, es que está basado en React. React es una biblioteca de código abierto que facilita el desarrollo de aplicaciones en una sola página. Se suele utilizar en aplicaciones donde los datos cambian regularmente gracias al uso de funciones nativas como *useState*, *useEffect* o *useContext*, que sirven para la almacenar el llamado *Estado* de un componente.

Los componentes son de gran utilidad para la generación de estructuras repetitivas, reduciendo la cantidad de código de una aplicación. De manera simplificada, los componentes se pueden definir como funciones que se almacenan en un archivo externo y que tienen como objetivo devolver un código HTML. También, pueden realizar todo tipo de tareas como cualquier otra función, implementándolas dentro del archivo. Las entradas de los componentes se denominan propiedades y se introducen al inicializar un componente.

Fijándose en el diseño que se hizo de las páginas, se han decidido usar los siguientes componentes:

- Link: es un componente de la librería de Next y en la aplicación se ha usado para toda la gestión de enlaces. Recibe como argumento principal la href a la dirección de la página al que dirigirá si se pincha en los boques que envuelve.

```
<Link href={"/events/" + event.id}>
  <p className="text-5xl font-bold text-main capitalize">
    {event.name}
  </p>
</Link>
```

Figura 61. Componente Link

- Image: es otro componente de Next, como su nombre indica es el componente que se ha usado para mostrar todas las imágenes. Aunque Image haga mejor tratado de

imagen que <img/>, también se ha decidido usar porque este último no funciona correctamente en aplicaciones que usan Next.js.

```
<div className="relative w-full h-[450px]">
  <Image
    loader={() => event.image}
    src={event.image}
    layout="fill"
    objectFit="contain"
    alt={event.name}
  />
</div>
```

Figura 62. Componente Image

- Barra de navegación: es un componente de elaboración propia, se inicializará una sola vez en el archivo de _app.js y se encontrará en la parte superior de cada página. Contiene otros componentes como:
    - SignIn: lleva la lógica de la identificación del usuario, usando de la librería *@thirdweb-dev/react* las funciones *useMetamask()*, *useAddress()* y *useDisconnect()*. Para que funcione, necesita ser envolvido con el componente *<ThirdwebProvider/>*, el cual inicializaremos en _App.js
        - ProfileIcon: este componente se encuentra dentro de SigIn y será activado cuando se haya identificado al usuario. Esconde el menu donde se encontraron los botones hacia las páginas de "Mis Eventos" y de "Mis Tickets".
    - SearchIcon: es un componente de la librería *@mui/icons-material/Search*
- ThirdwebProvider: es un componente de la librería de @thirdweb-dev/react, que permite la conexión con el actual proveedor de web3 y sin él no se podría usar las funciones de esta librería en las distintas páginas.
- Tarjeta de un evento: este componente es personalizado para esta aplicación, y se usará siempre para mostrar eventos en una cuadrícula a modo de tarjetas donde aparecerá la información más relevante.

```
<div className="grid grid-cols-4 gap-4 w-[90%] m-auto my-12">
  {events.map((event, i) => (
    <EventCard
      key={i}
      id={event.itemId}
      name={event.name}
      image={event.image}
      price={event.price}
      description={event.description}
    />
  ))}
</div>
```

Figura 63. Componente EventCard

5.3.1.6 File Server

Para finalizar con este capítulo, se explicará el bloque de File Server. En realidad, el File Server no es más que el conjunto de las páginas que conforman el proyecto, pero están implementadas de manera dinámica. Se encuentran en la carpeta de */pages*, junto con la API, y se acceden a ellas de la misma manera, mediante rutas. Por lo tanto, los archivos deberán estar ordenados con el mismo criterio que se tomó con la API.

Si se recuerda, en el apartado de Diseño se realizó un esquema de las páginas. A continuación se describirán tres tipos de páginas teniendo en cuenta dicho diseño. El objetivo es crear páginas tipo para simplificar la implementación.

- Tipo I:

  Son páginas que tienen como objetivo mostrar eventos o tickets en forma de tarjetas distribuidas en cuadrícula. Para ello se hará uso del componente de tipo *Card* del objeto a mostrar: *EventCard, TicketCard, etc...* Además, existen distintas tarjetas dentro de un mismo tipo de componente, en función de la información que se quiera mostrar: *EventCard, EventCardEdit, TicketCard, TicketCardEdit, etc..*

```
return (
  <div>
    <div className="grid grid-cols-4 gap-4 w-[90%] m-auto my-12">
      {tickets.map((ticket, i) => {
        const event = events.find(event => event.itemId == ticket.eventId);
        return (
            <TicketCardEdit
            key={i}
            id={ticket.id}
            eventId = {ticket.eventId}
            image={event.image}
            price={ticket.price}
            inSales={ticket.inSale}
            account={ticket.owner}
          />);
      }
      )}
    </div>
  </div>
);
```

Figura 64. Ejemplo de una página Tipo I

Como vemos en la figura, este tipo de páginas retorna, a través de un *map*, las tarjetas del objeto que se quiere mostrar.

Por otra lado, siempre harán uso de un array de objetos, del que se quiere mostrar, obtenidos a través de la función *getServerSideProps.* Dentro dicha función, se hará un *fetch* a la ruta de la API Server que tenga la misma dirección.

Como vemos en la figura, si nos encontramos en la página de "Mis Tickets", con una ruta */pages/my-tickets/[account]*, el fetch se deberá realizar apuntando a */api/my-tickets/[account]*.

```
export async function getServerSideProps(ctx) {

  const res1 = await axios.get(`http://localhost:3000/api/my-tickets/${ctx.params.account}`)
  const data1 = res1.data

  const res2 = await axios.get(`http://localhost:3000/api/events`)
  const data2 = res2.data


  //console.log(event);

  return {
    props:
    {
      tickets: data1,
      events: data2
    },
  };
}
```

Figura 65. Ejemplo del uso de la función getServerProps

Los parámetros que se necesiten usar en esta función, se deberán de incluir en la ruta y obtener usando el *context* o *ctx*, y los argumentos *params.*

- Tipo II:

Estas páginas tienen el objetivo de mostrar la información de un solo ticket o evento. Usan el componente Image, para mostrar la imagen, del *baseUri* del evento, en grande y con la máxima defnición. Además, usa el componente de Link para enlazar distintos elementos. El uso que hace de *getServerSideProps* es idéntico al del tipo anterior.

A parte de mostrar información, la página siempre implementar una función asíncrona asociada al botón o botones inferiores. En el ejemplo de la figura, nos encontramos en la página de ruta */pages/my-tickets/[account]/[eventID]/[ticketID]*, el botón de esta página acciona una función *submit* que realiza un POST a la API.

```
function Ticket({ ticket, event }) {
  const [price, setPrice] = useState('');

  async function submit(){

    fetch('/api/ticket-to-sale', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: {
        eventId: event.id,
        ticketId: ticket.id,
        price: price,

      },
    })
  }
```

Figura 66. Ejemplo de una página de Tipo II

- Tipo III:

Este tipo de página, en realidad, se corresponde solo a la pagina de crear un evento. La estructura es sencilla, se trata de un formulario con diferentes inputs para introducir la información del evento que se va a crear y una función asíncrona que lanza el método POST con dicha información en formato JSON. Además, antes de crear el evento se debe subir la imagen asociada al evento a la IFPS y almacenar el hash de dicha imagen en el contrato inteligente del evento, en la variable *baseURI.*

```
const client = new Web3Storage({token: apiToken})

async function saveFileInIPFS() {
  try {
    const cid = await client.put(file)
    console.log('CID: ', cid)
    return `https://dweb.link/ipfs/${cid}/${file.name}`;
    /* despues de que el archivo se suba a la IFPS hay qeu retornar la url */
  } catch (error) {
    console.log('Error al subir el evento: ', error)
  }
}
```

Figura 67. ejemplo de una página de Tipo III

A continuación, y de la misma manera que se expuso en el apartado de API Server, se hará un listado de las rutas hacia las páginas etiquetando el tipo de página al que corresponde:

- **/pages/events** → página de explorar eventos. Tipo I
- **/pages/events/[eventID]** → página de información de un evento de explorar. Tipo I
- **/pages/events/[eventID]/tickets** → mercado de reventas de un evento. Tipo II
- **/pages/events/[eventID]/tickets/[ticketID]** → página de información de un ticket del mercado de reventas. Tipo II
- **/pages/my-events/[account]** → página de los eventos de un usuario. Tipo I
- **/pages/my-tickets/[account]** → página de los tickets de un usuario. Tipo I
- **/api/create-event** → página para la creación de un evento. Tipo III

6. ANÁLISIS DE RESULTADOS

Antes de comenzar a analizar los resultados, es conveniente comentar que se han cumplido todos los objetivos planteados en la Definición del Trabajo.

Para el análisis de los resultados se mostrará la interfaz del usuario al completo y todas las opciones que ofrece. Como ya se vio en el capítulo de Sistema Desarrollado, la interfaz se diseñó para que fuese simple y usable, por lo que no dispone de una gran cantidad de opciones. Este punto es importante, ya que se valoraron distintas opciones en las que la interfaz no resultaba más sencilla de implementar, y finalmente se decidió por construirla de manera que cualquier usuario intuitivamente pudiese disfrutar de la funcionalidad que ofrece la aplicación.

Para comenzar, nos localizaremos en la pantalla de inicio de la página web. El sistema se ejecutará en un ordenador personal usando el comando *npm run dev*. De esta manera, se desplegará en la dirección *localhost,* en el puerto número *3000.* También, añadiremos a nuestra cartera digital la red de Ganache desplegada en el puerto *9545.*

```
C:\Users\Anto\OneDrive - Universidad Pontificia Comillas\ICAI\Cuarto\TFG\Desarrollo\event-marketplace>npm run dev

> event-marketplace@0.1.0 dev
> next dev

ready - started server on 0.0.0.0:3000, url: http://localhost:3000
info  - SWC minify release candidate enabled. https://nextjs.link/swcmin
info  - Disabled SWC as replacement for Babel because of custom Babel configuration ".babelrc" https://nextjs.org/docs/messages/swc-disabled
event - compiled client and server successfully in 9.3s (1713 modules)
wait  - compiling /_error (client and server)...
event - compiled client and server successfully in 3s (1714 modules)
```

Figura 68. Vista de la consola al desplegar la aplicación

Si accedemos a dicha dirección se observa lo siguiente:

Figura 69. Vista de la página de Explorar sin eventos creados

Como vemos en la figura, no hay ningún evento creado, por lo que en la página de explorar no encontraremos eventos. Para ello debemos crear un evento primero. Pulsamos en el botón de "Conectar Wallet" y nos identificamos en el sistema.

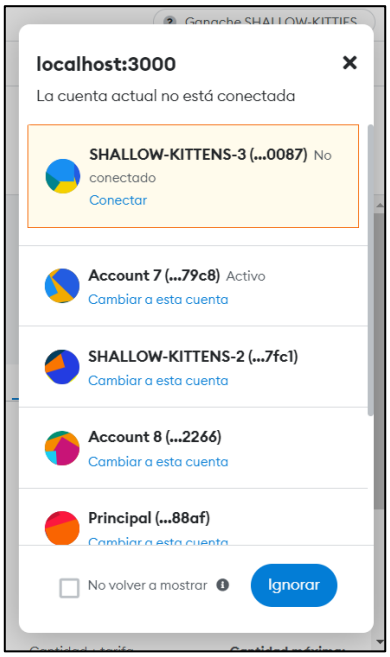

Figura 70. Ventana emergente para seleccionar una cuenta

Metamask nos lanzará una pestaña para seleccionar la cuenta con la que queremos conectarnos. Con motivo de la demostración se ha iniciado un Marketplace totalmente vacío de información y se ha conectado una de las cuentas que se disponen de Ganache. Sabremos si estamos conectados, si nos aparece el icono de perfil con un menú desplegable a la derecha, en la barrad e navegación.

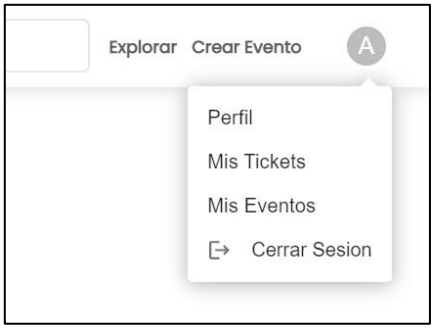

Figura 71. Vista del menú desplegable de la barra de navegación

Ahora crearemos un evento dirigiéndonos a la página principal y pulsando en el botón de "Crear Evento". Nos llevará a una página con un formulario.

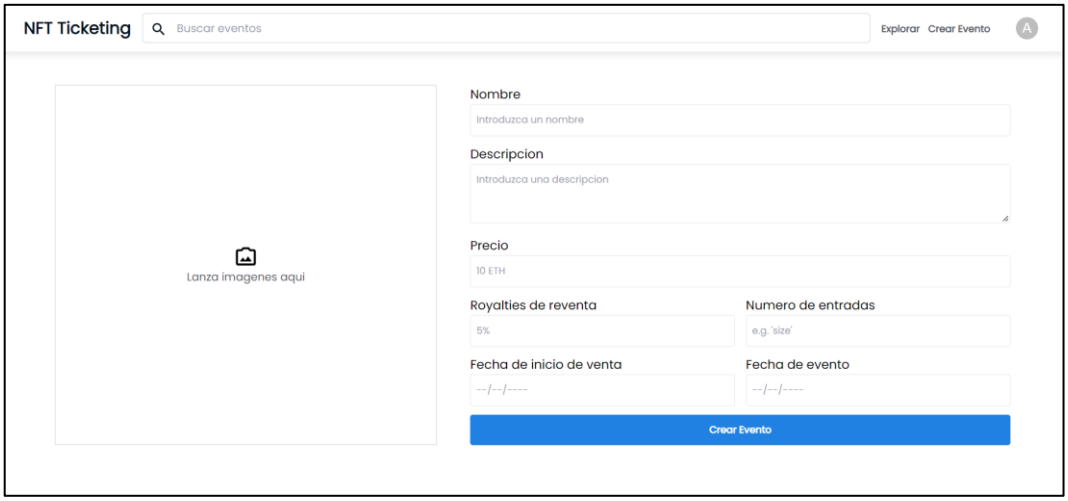

Figura 72. Vista de la página Crear Evento

Introducimos la información correspondiente al evento que deseamos crear. Para la demostración, crearemos un evento llamado *Evento 1* con un precio de minteo de entrada de *0.02 ETH*, que equivale a unos 30 euros.

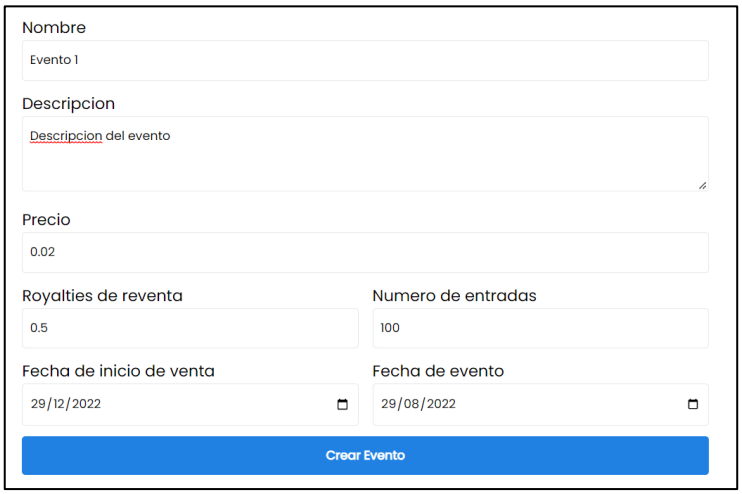

Figura 73. Formulario de la creación de un evento rellenado

Seleccionamos la imagen que deseamos establecer como portada del evento y como *baseURI* en el contrato de *Evento.sol*.

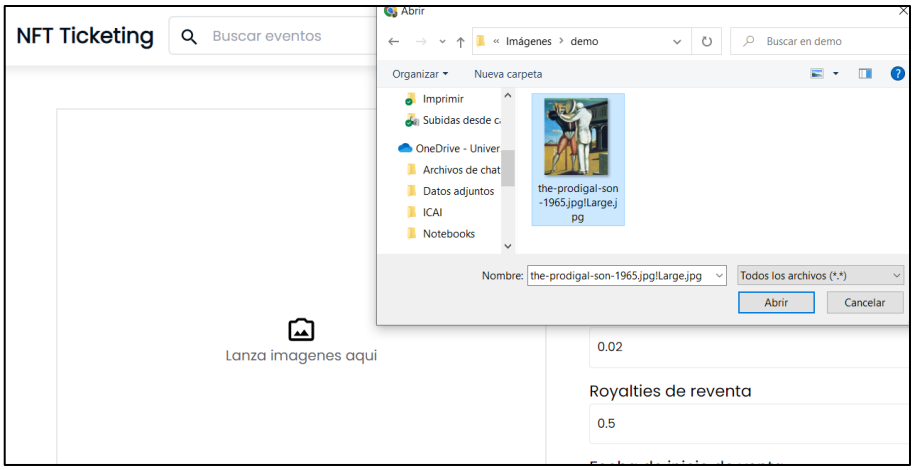

Figura 74. selección de una imagen para la portada del evento

Para la creación de eventos, existe en el código del contrato *Marketplace.sol* una constante llamada *createFee*, que se corresponde con las tasas correspondientes al Marketplace. Esta cantidad es arbitraria, no es objeto de estudio de este trabajo el calcular un valor razonable. También, se pedirán el coste de la tarifa de gas usada en la transacción.

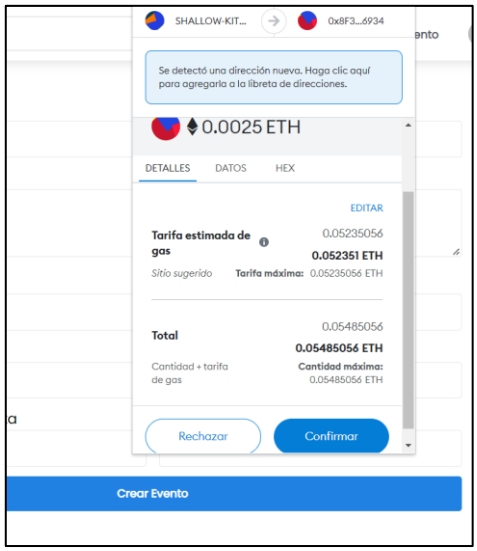

Figura 75. transacción para la creación de un evento

Ahora si nos dirigimos al menú desplegable del icono de perfil, y seleccionamos la opción de “Mis Eventos”, nos dirigiremos a la página de eventos de la cuenta conectada

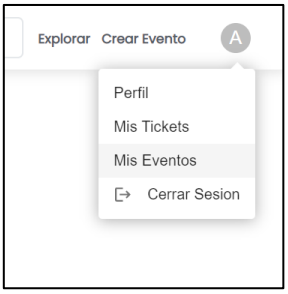

Figura 76. Localización del botón mis eventos

En dicha página nos encontraremos los eventos creados por la dirección.

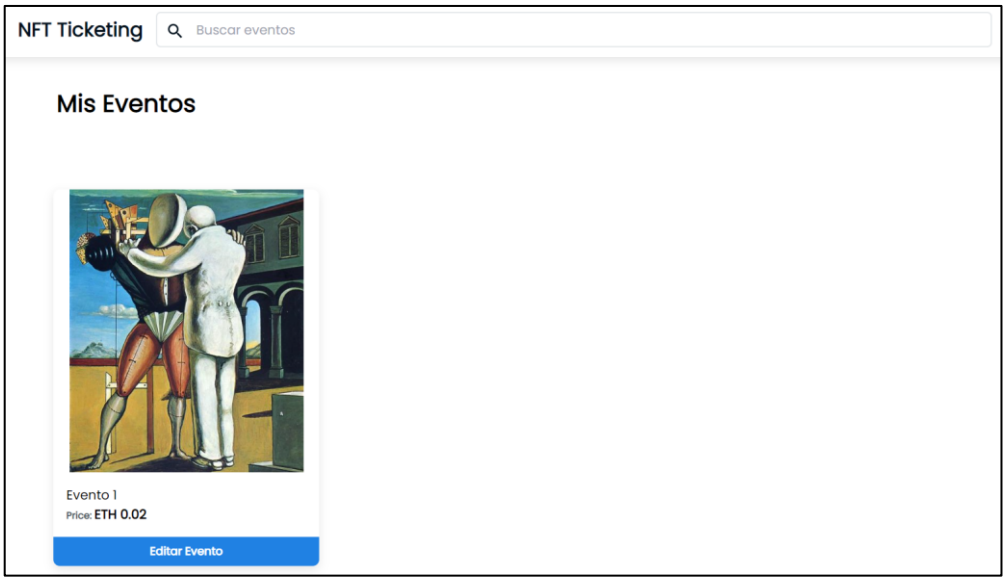

Figura 77. Vista de la pagina Mis Eventos

Como se puede observar en la figura, aparece un evento correspondiente al que recién se ha creado. Si clicamos en la tarjeta, nos direccionará a la información de dicho evento.

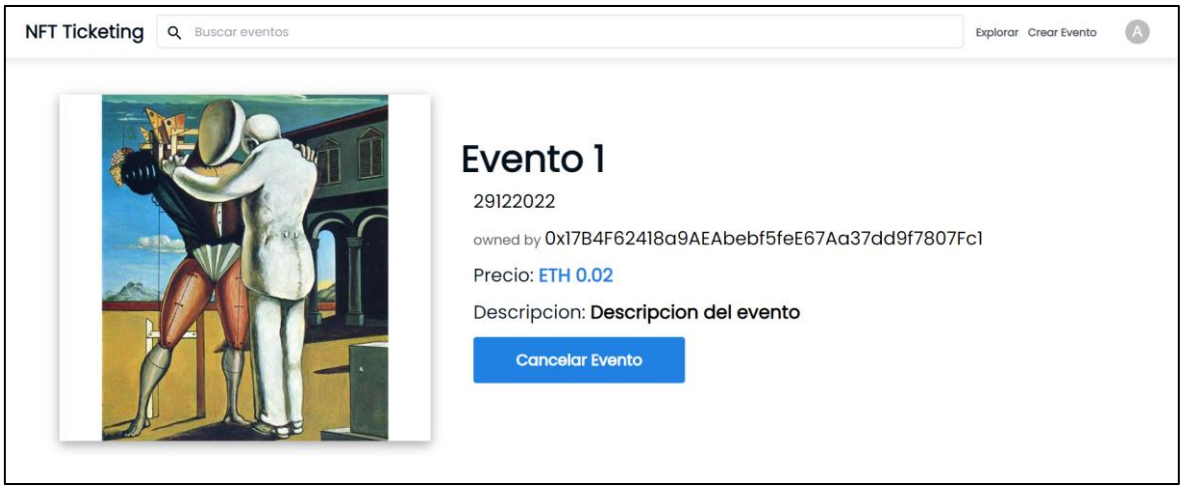

Figura 78. Vista de la información de un evento creado

Como se puede ver en la figura, nos aparece solo la opción de cancelar evento. Esto se debe a que se ha accedido al evento a través de la página de "Mis Eventos". Con motivo de la demostración, nos dirigiremos a la página de explorar, y pulsaremos en la tarjeta del evento que nos saldrá como disponible para comprar.

Figura 79. Vista de la pagina Explorar con un evento creado

Si pulsamos en el evento nos llevará a su información pero esta vez desde la vista de un posible comprador.

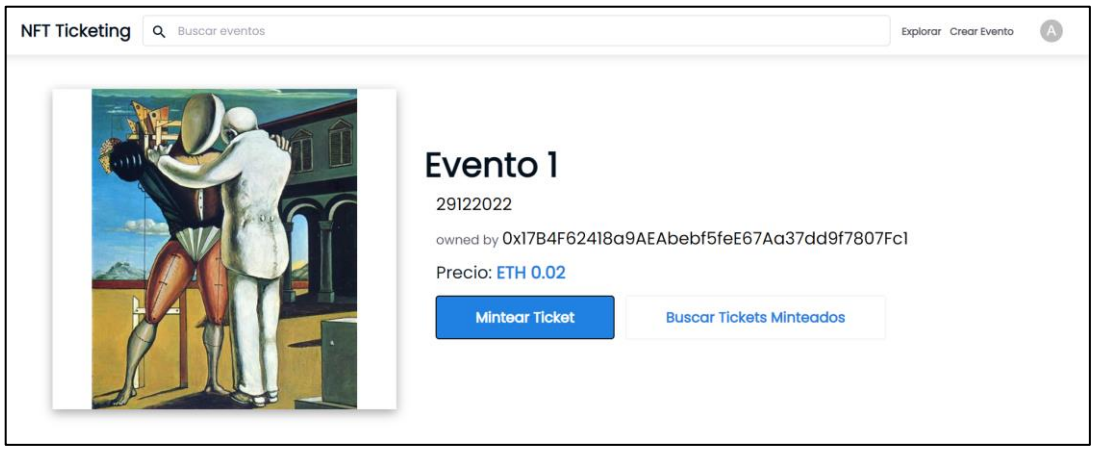

Figura 80. Vista de la información de un evento

Como observamos en la figura, ahora tenemos dos opciones: "Mintear Ticket" y "Buscar Tickets Minteados". Como ya se explicó, el mintear un ticket solo está disponible si la capacidad no se ha alcanzado. Y la opción de buscar un ticket ya minteado solo estará disponible si algún propietario de un ticket ha decidido ponerlo en venta. Con motivo de la demostración, *minearemos* un ticket original pulsando en el botón correspondiente.

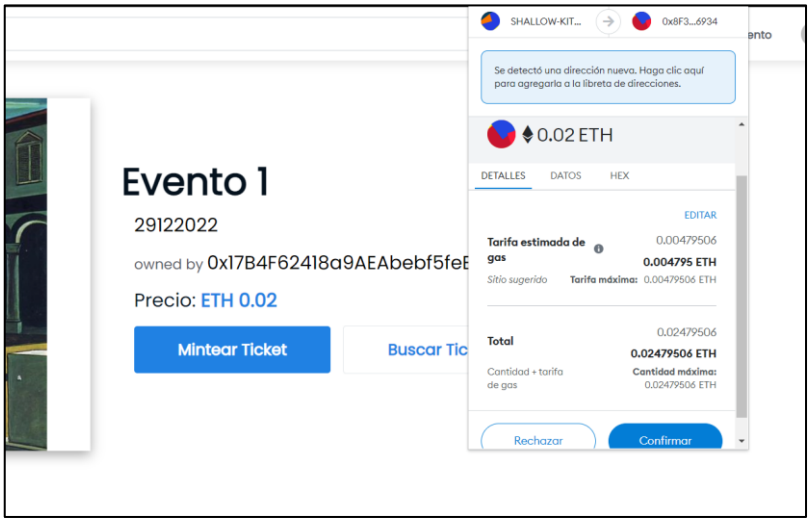

Figura 81. Transacción del minteo de un ticket

Se puede ver en la figura una pestaña similar a la que nos surgió al crear un evento. Esta vez, Metamask nos está pidiendo que confirmemos la transferencia con valor del precio de minteo, en este caso de *0.02 ETH*. También, como vimos anteriormente, nos solicita pagar el coste de la tarifa de gas de la tarea a realizar. Una vez confirmada la transacción, nos dirigiremos a la página de "Mis Tickets", para comprobar si se ha realizado la compra correctamente.

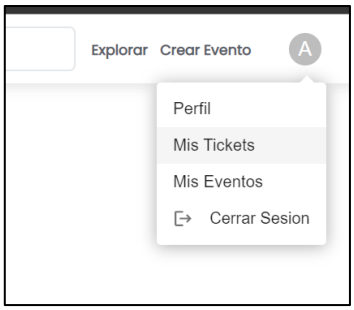

Figura 82. Localización del botón de Mis Tickets

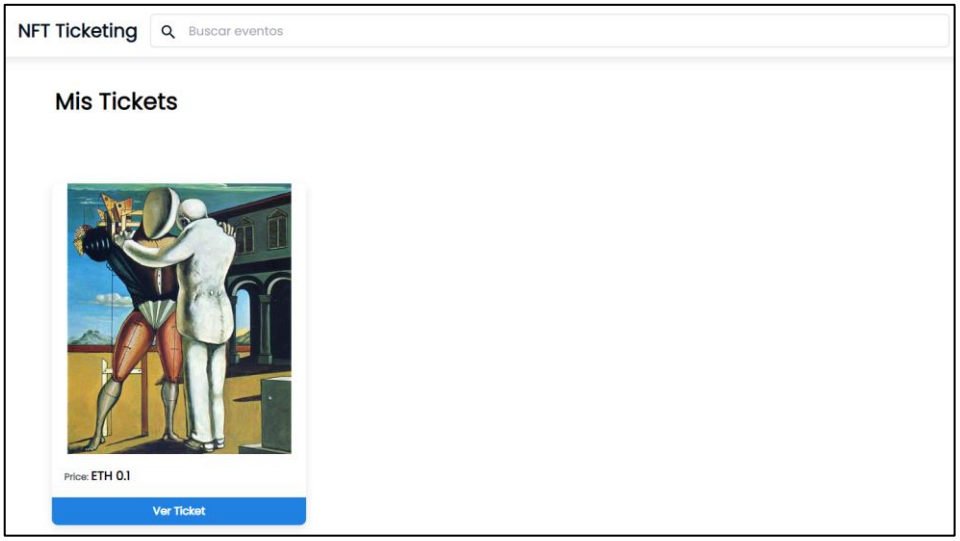

Figura 83. Vista de la página de Mis Tickets

Como se puede observar en la figura, la cuenta con la que se realiza la demostración es dueña de un ticket para el *Evento 1*.

Para seguir probando las funcionalidades de la aplicación, se tratará de poner en venta el ticket que posee la cuenta de prueba. Para ello, pulsaremos en la información del ticket, sin salirnos de la página de "Mis Tickets", y nos llevará a una página con la información de nuestro ticket.

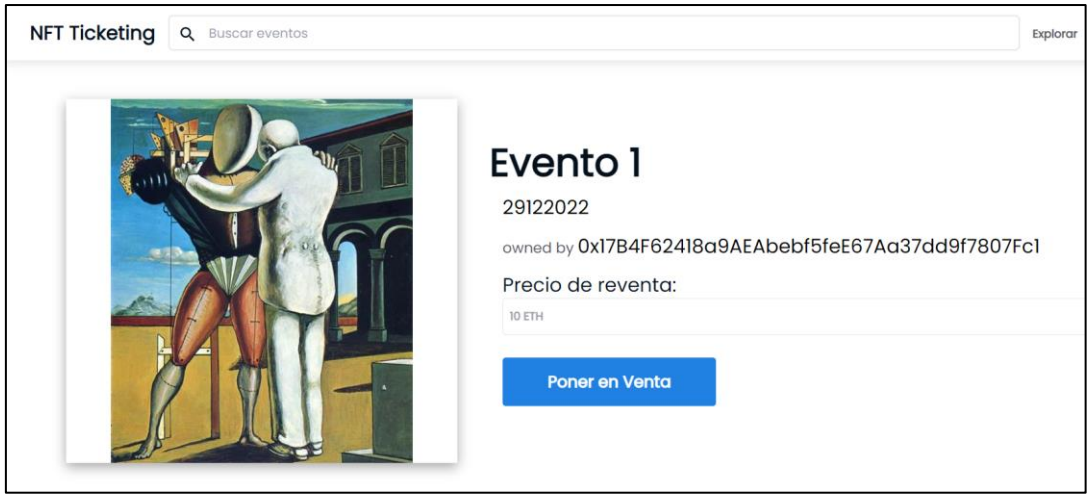

Figura 84. Vista de un evento concreto desde la página de Mis Tickets

Como se observa en la figura, existe la opción de introducir un nuevo precio seguido por un botón de "Poner en Venta". Además, se puede observar la dirección de la cuenta que ha creado el evento.

Si ahora, se quiere poner el ticket en venta, se introducirá una cantidad para el nuevo precio de mercado, y se pulsará en el correspondiente botón.

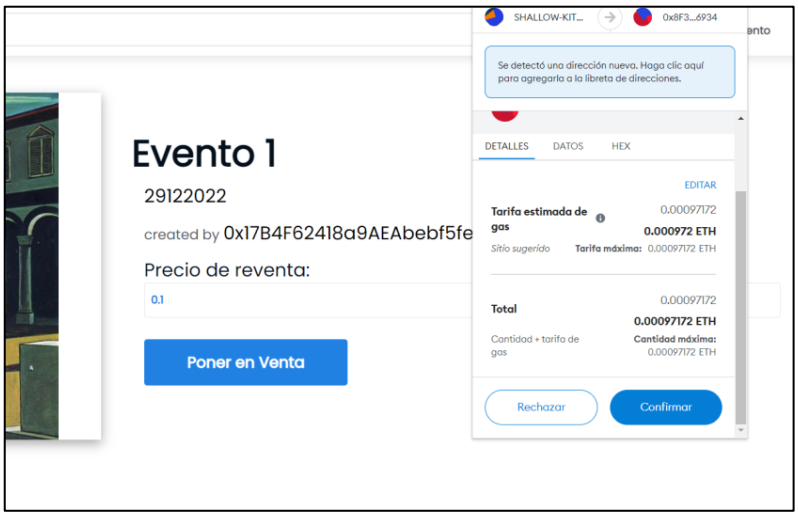

Figura 85. Transacción de establecer ticket en venta

Una vez pulsado el botón, Metamask nos pedirá que firmemos la transacción y emitamos el pago de la tarifa de gas correspondiente al gasto computacional de la tarea. Para confirmar si la operación se ha ejecutado correctamente, nos dirigiremos a la página de explorar y, entrando en la información del evento, pulsaremos en la opción “Buscar Tickets Minteados”.

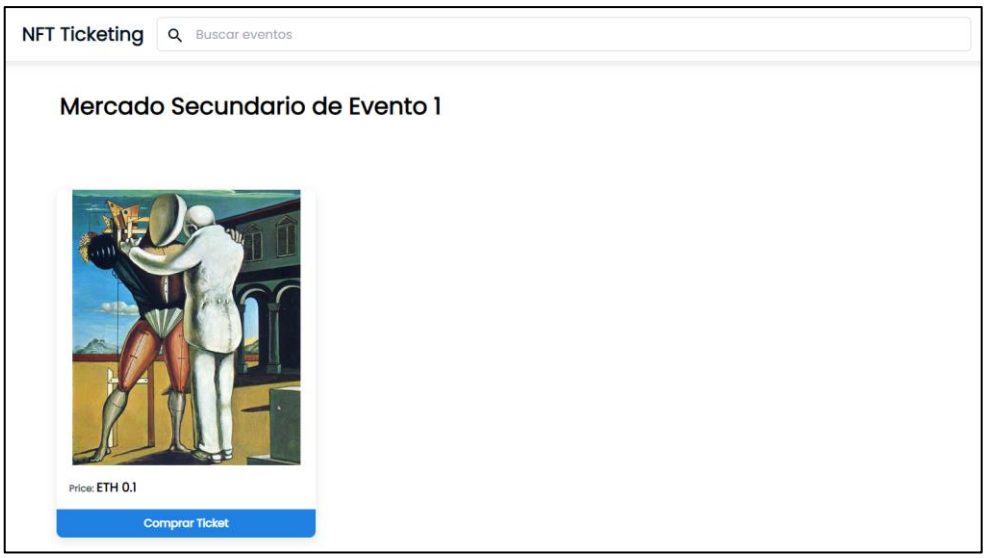

Figura 86. Mercado secundario del evento creado

Como hemos podido observar, los resultados muestran una respuesta acorde con lo esperado. Todas las funcionalidades que se diseñaron han sido materializadas. Hay que destacar que en el proceso de implementación se ha realizado a través siempre de una realimentación por parte de los resultados que se iban obteniendo.

7. CONCLUSIONES Y TRABAJOS FUTUROS

Es evidente, que la tecnología blockchain comenzará a formar parte de nuestro día a dia en un futuro no muy lejano. Estamos muy acostumbrados a tratar estos temas como si fueran simplemente especulaciones, que también las hay, pero hay un número enorme de aplicaciones en las estas tecnologías si tienen sentido. Como se explicó en la Justificación, este proyecto se ha realizado con el objetivo de solucionar una serie de problemas que se habían detectado en la industria de la gestión de tickets para eventos. Durante todo el trabajo, simplemente, se ha tratado de comprobar si a esta industria es posible acoplarle infraestructuras basadas en web 3.0, contratos inteligentes y redes blockchain. Y como se ha demostrado con la realización del sistema desarrollado, las medidas para resolver dichos problemas tienen sentido teórico y, sobre todo, son posibles de ejecutar en un marco práctico. Teniendo en cuenta los objetivos expuestos en la Definición del Trabajo, el sistema desarrollado supera todos ellos con creces. Además, el trabajo ha servido de acercamiento a unas tecnologías que están en auge y que necesitan la atención del sector para seguir desarrollándose.

Por otro lado, es lógico que debido a la limitación en el tiempo de desarrollo haya carencias que este proyecto no ha podido satisfacer. Muchas de ellas fueron surgiendo a raíz de una compresión más específica de los conceptos, otras simplemente dificultaban y alejaban al proyecto de los objetivos principales A continuación, se realizará una lista de todos los posibles futuros trabajos:

- Un sistema para que los organizadores puedan verificar los tickets que se presenten. Seria simplemente recibir la dirección de la cuenta que aporte el cliente y el sistema comprobaría si está dentro de la lista de los dueños de tickets.

- Una solución para que la aplicación permita pagos a través de fuentes que no sean criptomonedas. Haría falta crear una especie de intercambiador de divisas e incorporarlo al sistema. También se podría realizar a través de un agente externo.
- Unos contratos inteligentes aún más dinámicos, con más opciones para ofrecer a los creadores de eventos.
- Que el sistema sirva como un lugar de encuentro que para promotores y artistas establezcan las condiciones del evento, y estas queden escritas en el contrato del evento.
- Un sistema de autentificado para comprobar que los eventos han ocurrido y que no ha habido ningún fraude. Este punto se podría realizar mediante un sistema de recompensas donde los asistentes evalúan de forma común el evento. Se podría implementar usando el estándar ERC-20 y creando una token fungible asociada a la aplicación. De esta manera, los organizadores solo recibirán las ganancias del evento, si se corrobora que el evento ha cumplido con lo acordado en el contrato.
- Una mejora en la escalabilidad del proyecto mediante una separación más pronunciada de los subsistemas que lo conforman.
- Que se puedan agregar distintos tipos de entradas dentro de un mismo evento.
- Ampliación total del objetivo del proyecto. No solo mintear entradas de eventos, si no trasladar cualquier tipo de interacción relacionada con el ocio al mundo de las NFT’s : entradas de cine, promociones para restaurantes, etc.

8. BIBLIOGRAFÍA

[1] AMD, "What is a block," [Online]. Available: https://www.amd.com/system/files/2019-01/202964-what-is-a-block-1260x500.png.

[2] Wikipedia, "Cadena de bloques," [Online]. Available: https://es.wikipedia.org/wiki/Cadena_de_bloques.

[3] Wikipedia, "Token No fungible," [Online]. Available: https://es.wikipedia.org/wiki/Token_no_fungible.

[4] Ethereum, "Standards," [Online]. Available: https://ethereum.org/es/developers/docs/standards/.

[5] Ethereum, "ERC-721," [Online]. Available: https://ethereum.org/es/developers/docs/standards/tokens/erc-721/.

[6] IBM, "Smart Contracts," [Online]. Available: https://www.ibm.com/topics/smart-contracts.

[7] K. Parry, "Phys," 2017. [Online]. Available: https://phys.org/news/2017-11-ticketing-technology-scalping.html.

[8] Leewayhertz, "How NFT Ticketing Works," [Online]. Available: https://www.leewayhertz.com/how-nft-ticketing-works/.

[9] Y. Musienko, "Merehead," 2022. [Online]. Available: https://merehead.com/blog/nft-in-ticketing-industry/.

[10] Thirdweb, "Documentation," [Online]. Available: https://portal.thirdweb.com/auth.

[11] K. Lifanova, "Creating a contract with a smart contract," 2019. [Online]. Available: https://medium.com/upstate-interactive/creating-a-contract-with-a-smart-contract-bdb67c5c8595.

[12] CryptoMarketPool, "Gas in Oslidity Smart Contracts," [Online]. Available: https://cryptomarketpool.com/gas-in-solidity-smart-contracts/.

[13] GeeksforGeeks, "Creating Dapps using the Truffle Framework," 2022. [Online]. Available: https://www.geeksforgeeks.org/creating-dapps-using-the-truffle-framework/.

[14] Github, "Web3.js," [Online]. Available: https://github.com/ChainSafe/web3.js.

[15] L. Esparragoza, "Como programar y desplegar contratos inteligentes," 2020. [Online]. Available: https://www.criptonoticias.com/tecnologia/como-programar-desplegar-contratos-inteligentes-blockchain/.

[16] Coin Telegraph, "What is the Ethereum Virtual Machine," 2020. [Online]. Available: https://es.cointelegraph.com/explained/what-is-the-ethereum-virtual-machine-evm-the-ethereum-virtual-machine.

[17] Ethereum, "Dapps," [Online]. Available: https://ethereum.org/es/developers/docs/dapps/.

[18] BFA, "Protocolos de consenso," [Online]. Available: https://bfa.ar/blockchain/protocolos-de-consenso.

[19] Medium, "Entendiendo los protocolos de consenso," [Online]. Available: https://medium.com/astec/entendiendo-los-protocolos-de-consenso-de-blockchain-4858c71722d2.

[20] Criptonoticias, "Como crear vender nft ethereum opensea," [Online]. Available: https://www.criptonoticias.com/tutoriales-guias/como-crear-vender-nft-ethereum-opensea/.

[21] Tokenized, "Opensea Fees," 2022. [Online]. Available: https://tokenizedhq.com/opensea-

fees/#:~:text=OpenSea's%20service%20fee%20rate%20is,the%20current%20Ethereum%20gas%20price.

[22] Criptonoticias, "Mineria de Bitcoin," [Online]. Available: https://www.criptonoticias.com/criptopedia/que-mineria-bitcoins-criptomonedas/#:~:text=La%20miner%C3%ADa%20es%20la%20piedra,cadenas%20de%20una%20red%20blockchain.

[23] Openzeppelin, "Token ERC-721," [Online]. Available: https://docs.openzeppelin.com/contracts/2.x/api/token/erc721.

[24] Web 3 Storage, "Documentation," [Online]. Available: https://web3.storage/docs/reference/js-client-library/.

[25] Ethers, "Documentation," [Online]. Available: https://docs.ethers.io/v5/.

[26] Truffle, "Testing your contracts," [Online]. Available: https://trufflesuite.com/docs/truffle/testing/testing-your-contracts/.

[27] Truffle , "Ganache," [Online]. Available: https://trufflesuite.com/docs/ganache/.

[28] Medium, "Guide to Next js," [Online]. Available: https://medium.com/swlh/the-hitchhikers-guide-to-next-js-fd7aa14ae8d0.

[29] Workfront, "Waterfall method," [Online]. Available: https://www.workfront.com/sites/default/files/inline-images/waterfall-method-process.png.

[30] Academy Bit2me, "Que es un hash," [Online]. Available: https://academy.bit2me.com/que-es-hash/#:~:text=Un%20hash%20es%20el%20resultado,y%20tiene%20una%20amplia%20utilidad..

[31] Moesif, "Building an Ethereum Dapp," 2022. [Online]. Available: https://www.moesif.com/blog/blockchain/ethereum/Tutorial-for-building-Ethereum-Dapp-with-Integrated-Error-Monitoring/.

[32] Medium, "Blockchain development resources," [Online]. Available: https://medium.com/blockcentric/blockchain-development-resources-b44b752f3248.

ANEXO I: ALINEACIÓN DEL PROYECTO CON LOS ODS

Los ODS o Objetivos de Desarrollo Sostenible, son 17 objetivos a nivel mundial que estan diseñados tratar de encauzar a la sociedad hacia la dirección de la sostenibilidad y la mejora común. Los estableció la Asamblea General de las Naciones Unidas en 2015.

Figura 87. Objetivos de desarrollo sostenible

9 - Industria, Innovación y Infraestructura

El proyecto se alinea perfectamente con el objetivo número 9. Este objetivo, trata de la innovación, y el proyecto presentado se apoya en tecnologías que no están todavía adoptadas en las infraestructuras digitales. A través de un proceso de investigación e innovación, se ha

propuesto en este trabajo solucionar una serie de inconvenientes que se han encontrado en una industria en concreto, el mercado de las entradas para eventos.