Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin De Grado

Grado en Ingeniería Informática

CryptoCards: Creación de un juego de cartas basado en blockchain, Ethereum y en los NFT

Alumno:
Javier Perrote Gómez

Tutor Académico:
César Pablo Gutiérrez Martínez

Tutor de Empresa:
Julia Zuara Jiménez

Agradecimientos

Muchas gracias a César y Julia por la ayuda y los consejos para realizar el trabajo.

Muchas gracias a mis amigos por aguantarme y escucharme.

Muchas gracias a mi familia por estar siempre conmigo.

Muchas gracias papa, estoy seguro que estas muy orgulloso de tu hijo.

Resumen

El proyecto consiste en el desarrollo de una aplicación descentralizada sobre un juego de cartas de rol basado en la tecnología *blockchain* Ethereum y en los *NFT*, en colaboración con el Observatorio Tecnológico HP.

Una misma carta puede tener un gran número de unidades, cada una de estas constituye un *NFT* (Not Fungible Token) es decir, un *token* único que no se va a poder modificar y quedara registrado en la *blockchain* a través de un certificado digital que recoge sus propiedades y asegura su autenticidad, lo que se conoce como contrato inteligente o *smart contract*.

Los *smart contracts* desarrollados quedarán almacenados en la *blockchain* y se ejecutarán de manera automática cuando se cumplan los acuerdos escritos en ellos, su cometido en nuestra aplicación no solo quedará limitada a la creación de las cartas, si no que será necesario su uso para proporcionar cada uno de los servicios del juego, por ejemplo, el intercambio de cartas entre usuarios o la consulta por parte de uno de ellos de su colección. La ejecución de uno de estos contratos garantizará un mecanismo seguro al usuario, para hacer uso de una funcionalidad concreta del juego, todo esto, sin la necesidad de un intermediario, tal como un servidor central, que en aplicaciones centralizadas suele ser el encargado de la toma de decisiones.

En referencia a la estructura de la aplicación, al no permitir a los usuarios interactuar directamente con la *blockchain*, es necesario desarrollar, por un lado, un *frontend* encargado de interactuar con el usuario y por otro, un *backend* que será el encargado de ejecutar los *smart contracts* residentes en la red.

Abstract

The project consists of the development of a decentralized application on role-playing card game based on Ethereum blockchain technology and NFTs, in collaboration with the HP Technology Observatory.

A single card can have a large number of units, each of which constitutes an NFT (non-fungible token), i.e. a unique token that cannot be modified and that will be registered in the blockchain through a digital certificate that collects its properties and ensures its authenticity, which is known as a smart contract.

The smart contracts developed will be stored in the blockchain and will be automatically executed when the agreements written in them are fulfilled, its role in our application will not be limited only to the creation of the tokens, but it will be necessary to use it to provide each of the services of the game, for example, the exchange of tokens between users or the consultation by one of them of their collection. The execution of one of these contracts will ensure a secure mechanism for the user to make use of a particular functionality of the game, all this, without the need for an intermediary, such as a central server, which in centralized applications is usually the decision maker.

In reference to the structure of the application, as it does not allow users to interact directly with the blockchain, it is necessary to develop, on the one hand, a frontend in charge of interacting with the user and, on the other hand, a backend that will be responsible for executing the smart contracts residing in the network.

Índice General

Contenido

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Introducción

En este proyecto se explicarán las bases para el desarrollo de una aplicación en la red Ethereum y uso de los *NFT*.

Con la información de este documento, se tendrá un conocimiento sobre los principios, características y funcionamiento de la *blockchain*, incluyendo de manera más detallada, como se producen las transacciones entre usuarios en Ethereum y con los *smart contracts*, por ende, se explicará que son estos, su funcionamiento, desarrollo e implementación, incluyendo una breve explicación de Solidity, que será el lenguaje de programación utilizado para su creación. Además, se expondrá el concepto, propiedades y desarrollo de un *NFT* en Ethereum.

1.2 Motivación

En el último año se ha producido la explosión y consolidación del mundo de las criptomonedas y los *NFT*, el número de usuarios se ha incrementado exponencialmente, se ha convertido en tema de constante discusión, dónde unos ven la posibilidad de realizar la mejor inversión de sus vidas, mientras que otros consideran que se trata de un mercado de gran especulación y que personas influentes no deberían promocionar su uso.

Hace no tanto, eran unas pocas las criptomonedas que existían en el mercado y los *NFT* estaban estrechamente vinculados al mundo del arte [1], ahora la colección de criptomonedas es enorme, incluso hay empresas que han desarrollado su propia criptomoneda, dando incluso la posibilidad de pagar por sus servicios a través de su moneda, en cuanto a los *NFT*, la variedad de ramas que ahora abarcan es enorme, videos, tweets se han convertido en *NFT* y se han venido por cantidades enormes de dinero.

El principal motivo de este proyecto es dar a conocer la otra parte que la gente no conoce, el mundo de la blockchain en general no es solo una posibilidad de inversión, si no que proporciona una forma totalmente nueva para realizar transacciones, garantizando seguridad, rapidez y transparencia en otras cosas, todo ello sin la necesidad de un intermediario.

La elección de desarrollar un juego de cartas y específicamente un juego de cartas de rol es debido a que es posible incorporar todas las ventajas que ofrece el uso de la *blockchain* y más específicamente el de los *NFT.*

En este tipo de juegos es común la existencia de coleccionistas que compran y venden cartas para hacerse con toda la colección o con una carta en concreto. Además, en ocasiones existen cartas únicas que adquieren un coste muy alto en el mercado y necesitan de un mecanismo que acredite su autenticidad, para evitar que circulen copias falsas de las mismas. Además, existe la posibilidad de que se produzca un robo al ser la misma un objeto de valor.

Es aquí donde el uso de los *NFT* cobra sentido, proporcionando un mecanismo que asegura la autenticidad de la carta y además de un sistema seguro que evita su robo, ya que es imposible que una persona distinta al propio dueño tenga acceso a la misma.

Por último, los juegos de rol ofrecen por lo general otra manera de conseguir cartas, que consiste en comprar sobres. La diferencia con lo anterior es que no se conoce el contenido del sobre, es decir, no se sabe que carta o cartas están dentro de este.

Esta idea permite incorporar la creación de un NFT. Se puede simular la abertura de un sobre con la creación al azar de una de las cartas de la colección del juego.

1.3 Objetivos

El objetivo principal es desarrollar de una aplicación descentralizada sobre un juego de cartas basado en la tecnología *blockchain* Ethereum, para ello se va a implementar una aplicación descentralizada en Ethereum que permita realizar transacciones en la red, más concretamente la compra o venta de *NFT.*

Otros objetivos son:

- Para poder realizar estas transacciones en la blockchain, cada usuario deberá disponer de una billetera con la que podrá gastar sus Ether (moneda de Ethereum) y será capaz de firmar digitalmente la transacción para garantizar la veracidad de esta.

- Además de la posibilidad de comprar o vender *NFT*, el usuario puede crear sus propios *NFT* pagando una cantidad de Ether.

- Es necesario que el usuario pueda visualizar su colección de *NFT* y tenga un mecanismo para poder poner en venta aquellos que quiera. A su vez, debe poder visualizar los *NFT* que están en venta para poder comprarlos.

- Al tratarse de un juego, otro de los objetivos es proporcionar jugabilidad al juego de manera que este sea sencillo e interactivo, proporcionando una interfaz fácil de usar.

- Debido a que cualquier transacción incluyendo la implementación de los contratos en la red, suponen un coste de Ether, es necesario implementar una red local de pruebas que simule la red de Ethereum, creando una serie de cuentas que corresponden a distintos usuarios, cada una de ellas con Ether ficticios.

Capítulo 2 Estado del Arte

En este capítulo se detallarán los conceptos más importantes que se van a tratar, así como las tecnologías y lenguajes de programación utilizados.

2.1 Blockchain

El término de blockchain o también cadena de bloques se remonta a 2008 cuando Satoshi Nakamoto publica *"Bitcoin: A Peer-to-Peer Electronic Cash System"*. En este documento se relata un sistema de pago entre dos partes sin la necesidad de un tercero que en la mayoría de los casos se trata de un banco o servidor central, pero garantizando la seguridad en la transacción apoyándose en un mecanismo de criptografía. Sin embargo, no es hasta 2013 cuando empiezan a surgir otras ideas sobre el uso de la blockchain aparte del de Bitcoin y otras criptomonedas que se desarrollaron durante ese tiempo.

La blockchain consiste en una base de datos formada por una cadena de paquetes denominados bloques, que almacenan un conjunto de transacciones. Cada vez que se realiza una transacción esta genera unos datos que quedan almacenados en un bloque, y una vez está completo se acopla a la cadena ya existente de manera que la blockchain contiene un registro completo de todo el historial de transacciones, conocido como libro de cuentas.

Como antes dijimos la blockchain permite tener un libro de contabilidad actualizado y seguro, pero esta no reside en un servidor central, si no que se distribuye a través de una red de computadoras privadas que almacenan datos y ejecutan cálculos. Cada una de estas computadoras representa un nodo de la red y contiene una copia del libro de cuentas actualizado. Esta explicación hace referencia al concepto de sistema descentralizado, que no es más que un tipo de red donde los nodos no depende de un nodo maestro, si no que el control está distribuido entre todos los nodos y además estos pueden comunicarse entre sí sin la necesidad de un intermediario funcionando como una red P2P (peer to peer) [2].

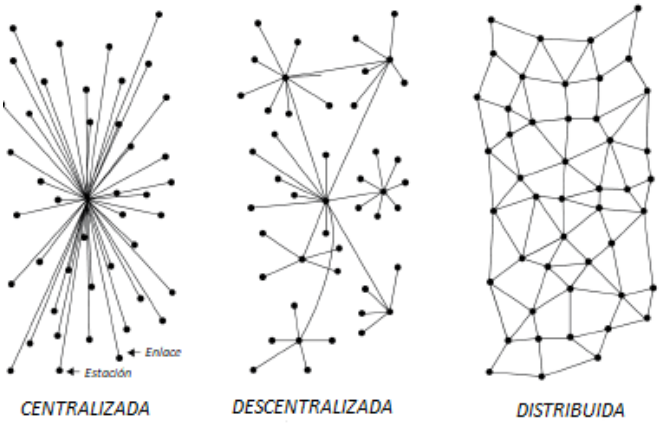


Figura 1 Tipos de Redes. Adaptado de [2]

A partir de esto nace el concepto de consenso descentralizado, cuyo significado no es más que un mecanismo que permite a un usuario acordar algo a través de un algoritmo sin la necesidad de un tercero o intermediario, el método más comúnmente utilizado se conoce como Prueba de Trabajo [2].

Cuando se quiere añadir un bloque a la cadena, se aplica la prueba de trabajo. Esta verifica que trascurrido un tiempo el bloque formara parte de la cadena y además una vez se haya actualizado el libro de cuentas, estas transacciones no se pueden modificar.

La prueba de trabajo consiste en un problema matemático extremadamente difícil que se pregunta a los nodos de la red, que necesita ser resuelto para poder añadir el bloque a la cadena, parte del problema forma parte del hash del bloque anterior, de forma que cuantos más bloques se añaden a la cadena, para poder modificar una transacción de un bloque anterior primero hay que resolver el problema de los bloques posteriores, por lo que transacciones más antiguas se consideran más seguras. El problema está configurado para resolverse entorno a los 10 minutos, de manera que una transacción realizada hace una hora se considera irreversible [3].

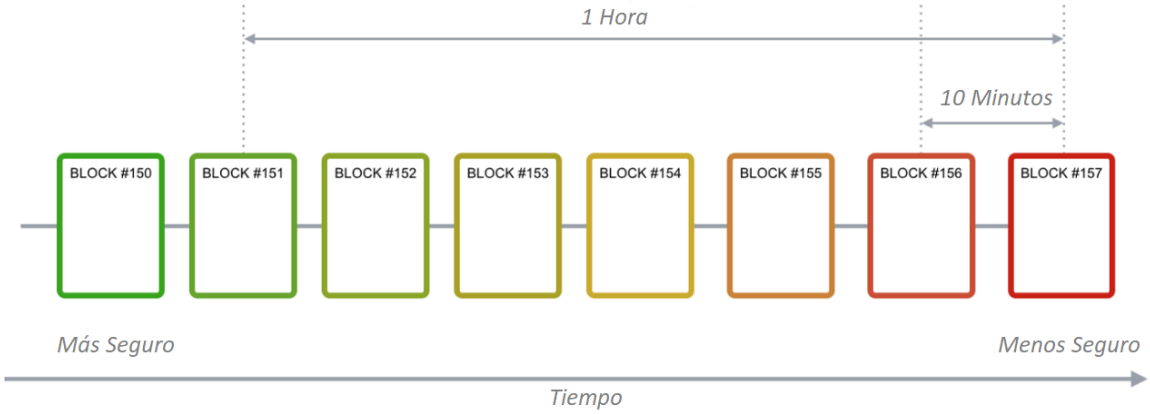


Figura 2 Ejemplo seguridad en la blockchain. Adaptado de [3]

A la tarea de resolver el problema se le denomina minería y con el fin de fomentar la competencia entre los nodos, existe una recompensa que va decrementando con el tiempo y que se da al primer nodo que calcula la solución.

Debido al enorme gasto energético y de cómputo que conlleva, los nodos suelen dividirse el trabajo y las ganancias formando los llamados pools de minería. Estos grupos suelen limitar el número de nodos con el objetivo de garantizar la seguridad, ya que si un grupo tiene una gran capacidad de cómputo tiene mayor control sobre la red y puede insertar bloques maliciosos. Una vez que el minero añade el bloque a la cadena, este se propaga al resto de la red, de manera que todos los nodos restantes agregan el nuevo bloque al final de su cadena.

En resumen, la blockchain consiste en una base de datos descentralizada que funciona como una P2P. Es como un libro de cuentas que mantiene un registro de todas las transacciones realizadas entre los participantes, ordenadas de manera cronológica y que se replica y sincroniza entre todos los nodos.

Para garantizar que todos los nodos contienen la misma versión de los datos, la transacción tiene que ser validada por los nodos de la red siguiendo un acuerdo. Este acuerdo sigue reglas criptográficas de manera que una vez que se valida una transacción esta no se puede remover ni alterar. Este mecanismo evita la necesidad de un servidor central que contiene toda la información y es el único con autoridad para controlar el sistema, proporcionando mayor seguridad ante la perdida de información.

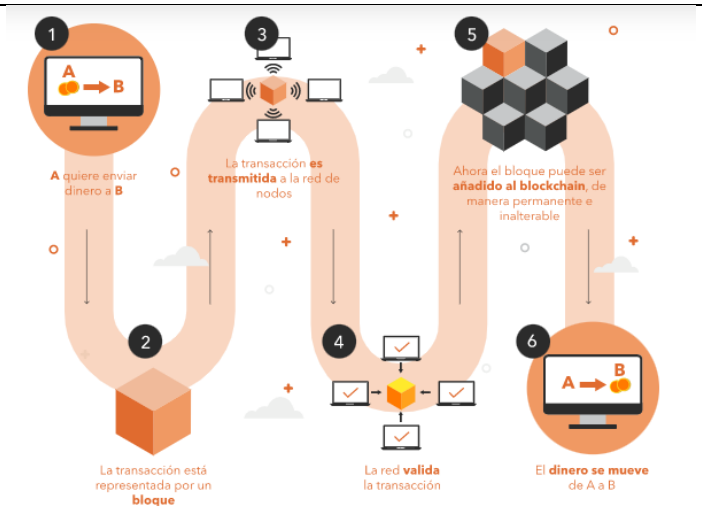


Figura 3 Fases de una transacción. Extraído de [4]

El principio de funcionamiento básico de la blockchain se muestra en la Figura 4. Un usuario A quiere enviar dinero o cualquier otro tipo de transferencia a un usuario B, esta transacción se almacena en un bloque, este se distribuye a todo el resto de los nodos de la red, donde se verifica y aprueba. Una vez aprobado se agrega el bloque a la cadena y se produce el pago de A a B.

2.2 Estructura de la blockchain

La blockchain como su nombre indica es una cadena de bloques, esto indica que de alguna manera los bloques deben estar conectados. Ya se explicó el proceso mediante el cual un bloque se añade a la cadena, pero no se especificó como es que estos bloques se relacionan entre sí. En la Figura 5 se puede ver un ejemplo de cadena. Cada uno de los bloques tiene una referencia al bloque anterior de manera que quedan vinculados, menos el primer bloque de la cadena que se denomina bloque génesis y no dispone de esta referencia.

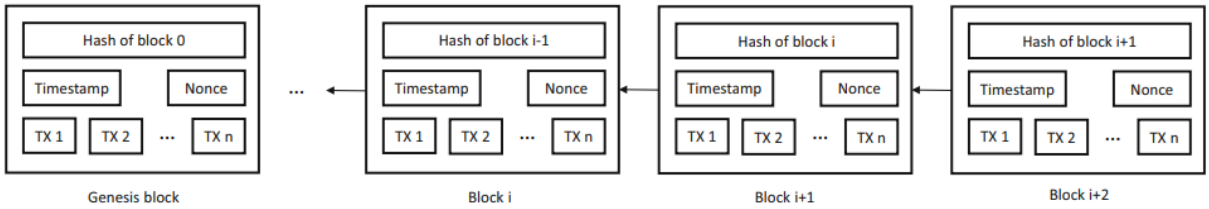


Figura 4 Ejemplo de blockchain. Extraído de [5]

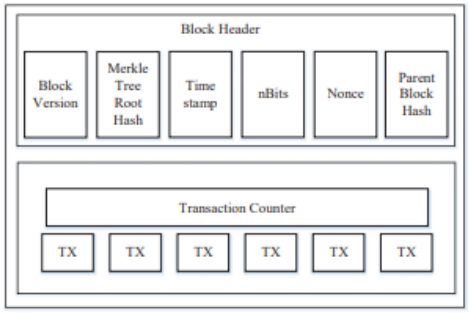


Figura 5 Componentes de un bloque. Extraído de [6]

Una vez comentado la estructura general de la cadena y como los bloques se relacionan entre ellos, se detallará los componentes existentes en cada uno de los bloques. Un bloque se divide en cabeza y cuerpo, teniendo este primero los siguientes campos [6] :

- **Versión del bloque:** El número de versión del software no es importante en la mayoría de los casos, pero indica el conjunto de reglas de validación que soporta.

- **Hash:** Se trata de un número único que se forma a partir de la información del bloque, de manera que este valor cambia si se detectan cambios en el bloque o en anteriores.

- **Timestamp:** Informa sobre que todas las transacciones almacenadas en ese bloque se han producido en el instante en que se minó el bloque, medido en la hora en la que sucedió en segundos desde el 1 de enero de 1970.

- **nBits:** aporta información sobre la dificultad de encontrar el nuevo hash para reclamar su validez, indica el número de ceros al principio del hash, a mayor número de ceros el hash es más pequeño y por lo tanto será más fácil encontrar uno que coincida.

- **Nonce:** número aleatorio que sirve para verificar el hash.

- **Hash del bloque padre:** un valor hash de 256 bits que apunta al hash del bloque y forma en la que se enlaza con este.

En cuanto al cuerpo este se compone de un contador y un conjunto de transacciones, la cantidad máxima de transacciones que puede tener cada bloque depende tanto del tamaño del propio bloque como del de las transacciones.

2.3 Ethereum

Ethereum es un tipo de *blockchain* pública, es decir es una red accesible por cualquiera por internet, y la red más importante y conocida junto a Bitcoin. Generalmente este tipo de redes comparten muchas similitudes y suelen estar formadas por [7] :

- Una red P2P que conecta a los participantes y transmite transacciones verificadas mediante un protocolo estandarizado.

- Una serie de reglas que definen lo que es una transferencia y cuando se considera que es válida.

- Una máquina de estados que procesa las transacciones según las reglas.

- Una cadena de bloques criptográficamente seguros que registran todas las transacciones verificadas.

- Un algoritmo de consenso que, descentralizada el control, forzando a los usuarios a comunicarse siguiendo el consenso establecido.

- Una Prueba de trabajo que incentiva al que la resuelva, con el propósito de garantizar la seguridad de la red.

Existen otra serie de características propias que la diferencian de otras redes públicas como es el caso de Bitcoin. La principal es que es una red que no está diseñada principalmente para ser una red de pagos. Es más a diferencia de Bitcoin posee un lenguaje muy completo que permite desarrollar aplicaciones descentralizadas de alta disponibilidad y transparencia entre otras cualidad, mediante los llamados contratos inteligentes o *smart contracts* [7].

Al igual que sucede con otras redes, Ethereum tiene su propia llamada Ether o ETH, cuyo precio en los meses desde Abril hasta Junio del año 2022 ha ido variando desde un máximo de 3500 dólares hasta unos 950 en el último mes [8].

Al igual el euro que tiene sus céntimos, como cualquier otra moneda, el Ether tiene sus propias subdivisiones.

$$1\ Ether = 10^{18}\ Wei$$
$$1\ Ether = 10^{15}\ Kwei$$
$$1\ Ether = 10^{12}\ Mwei$$
$$1\ Ether = 10^{9}\ Gwei$$
$$1\ Ether = 10^{6}\ Szabo$$
$$1\ Ether = 10^{3}\ Finney$$

Es imprescindible para utilizar Ethereum ya cualquier acción que se realice sobre Ethereum conlleva un gasto que se paga en esta moneda, a este gasto se le denomina gas.

El gas es la unidad que mide la cantidad de esfuerzo computacional requerida para llevar a cabo una transacción y es una comisión necesaria para que la transacción sea exitosa. Como se menciona antes la comisión se paga en Ether, más concretamente los precios del gas están en Gwei.

Existe una formula implementada desde el 5 de agosto de 2021 para calcular el coste total de la transacción [9]:

$$Límite\ de\ gas\ *\ (Comisión\ Base\ +\ Propina)$$

El límite de gas se calcula en función de los requisitos de espacio del bloque.
La comisión base se calcula en función de los bloques anteriores
La propina la recibiría el minero al resolver la Prueba de Trabajo.

2.4 Cuenta Ethereum

Cuando se quiere realizar una transferencia es necesario que tanto el ordenante como el beneficiario tengan cuentas bancarias. Lo mismo sucede en Ethereum, pero es este caso el saldo son ETH. A este tipo de cuenta se le denomina cuenta de propiedad externa.

La creación de una cuenta de propiedad externa no supone ningún coste y este tipo de cuenta permite iniciar transacciones, aunque las transacciones entre distintas cuentas deben de ser con ETH.

Existen otro tipo de cuentas llamadas contratos inteligentes o *smart contracts,* que se controlan mediante código escrito normalmente en un lenguaje denominado Solidity [10]. En este caso crear un contrato supone un coste, ya que este se almacena en la red y solo se pueden enviar transacciones si se ha recibido otra transacción previamente. En este caso, una transacción a un contrato ejecuta su código, que a su vez puede hacer muchas otras cosas.

En resumen, no son más que programas almacenados en la blockchain que se ejecutan cuando se cumplen ciertas condiciones y permiten automatizar un acuerdo entre participantes asegurando la seguridad del intercambio o automatizando un proceso al activar el siguiente paso [11].

Todas las cuentas tienen los siguientes campos [11]:

- **Nonce:** Contador que indica el número de transacciones enviadas desde la cuenta si esta es de propiedad externa o el número de contratos creados por la cuenta si es una cuenta de contrato.

- **Saldo:** Indica el número de Wei que tiene la dirección. Los Ether que posee una cuenta residen en la propia blockchain y se calculan viendo el historial de transacciones para esa misma dirección.

- **CodeHash:** Hash que hace referencia al fragmento de código del contrato inteligente que esta almacenando en la propia blockchain para poder recuperarlo. Si la cuenta es de propiedad externa, no existe código, por lo tanto, el codeHash hace referencia a un hash de cadena vacía.

- **Hash de almacenamiento**: Hash que codifica el contenido almacenando en la cuenta y esta vació por defecto.

Además de lo mencionado, las cuentas están formadas por un conjunto de claves criptográficas una privada y otra pública.

La clave privada proporciona un mecanismo de verificación que asegura que la transacción esta realizada por el dueño de la cuenta y se utiliza para firmar transacciones impidiendo que se difundan transacciones falsas.

La clave publica se genera a través de la privada mediante un algoritmo y permite a otros autentificar al que genero el mensaje.

2.5 Cartera Ethereum

Para facilitar el uso de una cuenta existen las llamadas carteras o *wallet* que no son más que una aplicación que permite gestionar la cuenta, facilitando la consulta del saldo, él envió de transacciones o la conexión a aplicaciones que residen en la red.

Existen diferentes tipos de carteras y entre ellas una de las más conocidas es Metamask.

2.6 NFT

NFT (*Non-Fungible Token*) se puede definir como un artículo digital que puede representar cualquier cosa y que se caracteriza por tener propiedades únicas, por lo que no existen 2 NFT iguales. Esto hace que no sean intercambiables ya que no tienen un valor definido.

Además, solo pueden tener un propietario al mismo tiempo y este puede ser conocido por todo el mundo, ya que la información es pública.

Otra de las características es que no se pueden modificar, puesto que están asegurados por la propia de blockchain.

Un token no fungible puede ser cualquier cosa compatible con Ethereum desde una obra de arte, hasta una entrada de un concierto.

2.6.1 ERC 721

En Ethereum se desarrollan estándares ERC (*Ethereum Request for Comments*). Un ERC son normas para definir y especificar estándares, de forma que los tokens que estos definen tengan propiedades comunes y sean compatibles, haciendo posible su reutilización en muchas aplicaciones distintas.

Entre estos entandares está el ERC-721 que es un estándar para los NFT. Este introduce una norma de forma que garantiza que el token es único y que además puede tener un valor distinto que otro token del mismo contrato.

Todos los NFT tienen un identificador, que no es más que un número único distinto para cada NFT almacenado en una variable denominada *tokenId*.

Capítulo 3 Planificación

Una vez identificado que se quiere hacer y explicado cómo se quiere hacer. En este capítulo se detalla que metodología se ha llevado a cabo, para desarrollar el proyecto.

3.1 Metodología Usada

3.1.1 Metodología usada en desarrollo con Observatorio HP

El trabajo se ha realizado según una programación con el Observatorio HP, coordinado por la tutora de empresa del TFG.

- Reuniones para entender cómo se trabaja.
- Información (antes de navidades) necesaria sobre *blockchain*, *NFT* …
- Reuniones bisemanales para seguimiento (al principio eran semanales) y planificación de las siguientes semanas.

Entre reuniones el alumno elabora el proyecto y va subiendo las modificaciones al repositorio de GitLab: https://gitlab.com/HP-SCDS/Observatorio/2021-2022/cryptocards/uva-cryptocards

3.1.2 Metodología con la UVA

En cuanto a cómo trabajaremos el tutor y el estudiante será de la siguiente manera:

Hemos tenido una reunión de arranque del proyecto, en la que hemos establecido cuál va a ser el trabajo para realizar, los objetivos a conseguir y el procedimiento o modo de trabajar.

Metodología ágil

Usaremos una herramienta ágil para trabajar: el tablero Kanban.

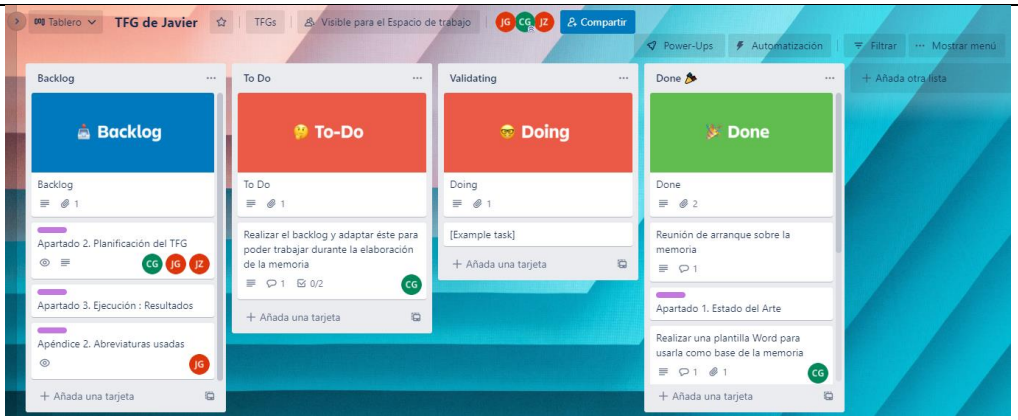


Figura 6 Captura Tablero Kanban del proyecto

En cada una de las tarjetas del Kanban se pondrá toda la información relevante: quienes hacen o participan en la tarea, que hacen, tiempos y tendrán ficheros anexados (versiones de documentos), es importante recordar que el uso de una metodología y una herramienta ágiles permiten al alumno autogestionar el trabajo y al tutor controlar que este trabajo va bien a tiempos y que le es entregado incrementales periódicamente.

Definiremos sprints, que serán de 2 semanas de duración, para ello en el Trello estableceremos por cada uno de ellos un sprint backlog en el que pondremos las historias que entran. Dichas historias las iremos retirando del Product Backlog (en nuestro caso la lista de tareas).

Desde el inicio (6 meses antes de la presentación del TFG) trabajaremos en la memoria y en la presentación PPT. Así iremos haciendo incrementales en los sprints, en Julio de 2022 por tanto sólo quedará el último incremental y entregable de estas (memoria y presentación).

Por tanto; la metodología a usar seguirá un marco ágil; en nuestro caso SCRUM; cómo se ve en la figura siguiente:

Figura 7 Diagrama de Scrum. Extraído de [12]

Las comunicaciones entre el tutor (César Pablo Gutiérrez) y el estudiante (Javier Perrote) serán a través del Teams de la Escuela.

Se trabajará con las herramientas (nube OneDrive) de Microsoft.

3.2 Presupuesto y Calendario del proyecto

3.2.1 Presupuesto en horas y calendario

	Fecha inicio	Fecha final	Participantes	Horas Estimadas	Horas realizadas	Coste
Matrículación e inicio de Cuatrimestre	jun-21	jun-21	Javier Perrote	0	0	0,00 €
Reunión de Inicio con HP	16/02/2022	16/02/2022	Javier Perrote, tutor/es	2	2	29,24 €
Reuniones de seguimiento semanales	feb-22	abr-22	Javier Perrote, tutor/es	26	26	380,12 €
Reuniones de seguimiento semanales	abr-22	jul-22	Javier Perrote, tutor/es	24	24	350,88 €
Trabajo inicial de estudio y realización de Intruducción y Estado del arte	ene-22	feb-22	Javier Perrote	40	44	643,28 €
Análisis de requisitos	feb-22	mar-22	Javier Perrote	20	20	292,40 €
Estudio de lenguajes de programación y uso de las herramientas	mar-22	abr-22	Javier Perrote	40	40	584,80 €
Montar el proyecto	mar-22	mar-22	Javier Perrote	20	25	365,50 €
Desarrollo del proyecto	abr-22	jun-22	Javier Perrote	160	170	2.485,40 €
Elaboración de la memoria	jun-22	jul-22	Javier Perrote	40	40	584,80 €
Elaboración de la presentación PPT	jun-22	jul-22	Javier Perrote	10	10	146,20 €
				372	391	5.716,42 €

Tabla 1 Calendario y presupuesto del proyecto

En la Tabla 1 se puede ver el calendario y presupuesto del proyecto. Como se observa tanto la estimación como las horas realizadas finalmente exceden por bastante las 300 horas requeridas para la realización de este TFG.

3.2.2 Estimación y monitorización de los recursos a utilizar

Si bien los recursos proporcionados por Observatorio HP y los otros por el alumno, este sería su coste en el mercado.

Recurso a utilizar (entorno de trabajo)	Fecha inicio uso	Fecha final uso	Descripción	Precio (si aplica)	Precio final (si aplica)
Ordenador pesonal	feb-22	jul-22	Ordenador Portátil	400,00 €	400,00 €
Microsoft Office Estudiantes 2019	feb-22	jul-22	Software programación	149,00 €	149,00 €
Software Visual Studio CODE	mar-22	jul-22	Software programación	0,00 €	0,00 €
Truffle	mar-22	jul-22	Software para desarollar los contratos inteligentes de Blockchain	0,00 €	0,00 €
Ganache	mar-22	jul-22	Software para desarrollar un simulado una red Ethereum	0,00 €	0,00 €
Metamask	mar-22	jul-22	Extensión de google chrome necesaria para	0,00 €	0,00 €
Axure RP PRO	may-22	jun-22	Software para realizar bocetos	25,00 €	25,00 €
Astah	may-22	jun-22	Software para realizar diagramas	0,00 €	0,00 €
				574,00 €	574,00 €

Tabla 2 Estimación y monitorización de los recursos

3.2.3 Presupuesto total del proyecto

Totales	Unidades / Horas	Precio Unitario	Precio total
Trabajo del proyectante	391,00	14,62 €	5.716,42 €
Recursos	1,00	0,00 €	574,00 €
			6.290,42 €

Tabla 3 Presupuesto final del proyecto

En la Tabla 3 se visualiza el presupuesto final del proyecto. El total es la suma de los recursos, más el sueldo por las horas del trabajo realizado por el estudiante. Se ha considerado que el salario que vendría cobrando una persona para la realización de este proyecto es de 14,62 € / hora (Salario medio de un Ingeniero Informático en España).

3.3 Riesgos del proyecto

Riesgos y oportunidades que presenta el proyecto son los siguientes:

Id de	categoria del riesgo	Descripción	Probabilidad	Impacto	Acciones de Mitigación	Responsable/s del Rie	Acciones de Contingencia	Estado	Proximidad	Alcance
R001	Calendario	Debido a otras responsabilidades del proyectante o del tutor, que no de tiempo a realizar las tareas a realizar y se tenga que posponer la entrega del Trabajo más allá de Julio	baja	alto	1. Usar metodología ágil para tener un sistema de entregables (incrementos) y control y mejor adaptación al cambio 2.- Tener reuniones semanales y diarías (si hiciera falta) para poder tener un control cercano 3.- Empezar a trabajar con tiempo (6 meses)	Javier Perrote César Pablo Gutiérrez	1. Modificar la fecha de entrega y retrasarla 2. Reducir la funcionalidad total a entregar	mitigandose	En cada sprint	Entrega TFG
R002	Funcionalidad	El proyecto entre en un punto muerto, al ser un estudio puede que nos encontremos ante la posibilidad de que estemos en un punto muerto sin poder concluir el estudio	baja	alto	1. Estructurar el proyecto como todo método científico con una investigación previa y/o la elaboración de unas hipotesis. 2. Desarrollar un DAFO para estar seguros de la viabilidad del proyecto 3. Realizar una planificación exhaustiva 4. Contrastar y discutir las hipotesis antes de proceder a la elaboración del estudio	Javier Perrote César Pablo Gutiérrez	1. Cambiar el objeto y objetivos del estudio a realizar.	mitigandose	A principio, despues de planificación del TFG y antes de su ejecucción.	Fases iniciales del TFG
O001	Empresa	El proyectante ha realizado el TFG bajo el Observatario de HP, esto le proporciona recursos y formación adiccional a usar en el TFG	media	bajo	1. Investigar posibles sinergias de lo que realiza en la empresa que pueda ayudarle a establecer nuevos conceptos o desarrollar los que ha plantaeado en el proyecto 2. Mencionar y aplicar (considerando las restricciones de confidencialidad) conocimientos en el TFG	Javier Perrote Julia Zuara	1. Buscar otras sinergias o aprovechamientos quw ayuden al alumno en su cometido	aprovechando	A principio, despues de planificación del TFG y antes de su ejecucción.	Hasta la entrega del TFG
R004	Calendario	Es que debido al trabajo y tener asignaturas no se pueda dedicar el proyectante a realizar la memoria del trabajo	baja	alto	1. Gestionar por metodología ágil la memoria 2. Dedicar en los fines de semana tiempo a la realización	Javier Perrote César Pablo Gutiérrez	1. Posponer la fecha de entrega del TFG	cerrado	En Julio	Fases finales del TFG

Tabla 4 Riesgos del proyecto

Para determinar si se debe mitigar el riesgo o aprovechar la oportunidad, se sigue la tabla de impacto y oportunidad siguiente:

NIVEL DE RIESGO/ OPORTUNIDAD		IMPACTO		
		baja	medio	alta
PROBABILIDAD	bajo	bajo	bajo	medio
	medio	bajo	medio	medio
	alto	medio	medio	alta

bajo	No se mitiga, no hace falta hacer nada
medio	Si estamos en nivel de riesgo medio, tenemos que aplicar la mitigación
alta	Si estamos nivel de riesgo alto: mitigar lo que se pueda y preparar la contingencia

Tabla 5 Matriz Impacto/Probabilidad

Capítulo 4 Análisis

Una vez establecido los objetivos, lo siguiente es realizar un análisis del proyecto, con el fin de identificar los requisitos que este debe cumplir. Para ello hay que seguir una serie de pasos previos.

El primer paso del análisis consiste en la búsqueda de información con el objetivo de comprender las necesidades mínimas que debe cumplir el proyecto. Para ello se estudia otras aplicaciones similares donde se puedan comprar y vender NFT, como es el caso de OpenSea una de las más conocidas.

Con esta primera búsqueda se decide que el proyecto sea una página web, ya que observa que la gran mayoría de aplicaciones que desarrollan NFT son páginas web.

En un segundo paso se identifica los distintos roles que interactúan con la página web. Esto es bastante sencillo ya que se trata de una aplicación descentralizada, por lo que no existe un servidor y son los usuarios que visitan la página web los únicos que interaccionan con la página.

Una vez identificado los distintos roles, se estudia los requisitos que debe tener la página teniendo en cuenta que se trata de una aplicación que va a residir en la blockchain, por lo que la seguridad en las transacciones en un requisito fundamental.

Se realiza un análisis de requisitos divididos en funcionales y no funcionales donde se recoge también los requisitos de seguridad.

4.1 Especificación de requisitos

4.1.1 Requisitos funcionales

Código	Nombre	Descripción
RF01	Confirmación de transacción	El sistema debe permitir a los usuarios confirmar las transacciones mediante su cartera de Ethereum.
RF02	Creación	El sistema debe permitir a los usuarios crear una carta al azar de entre todo el repertorio.
RF03	Colección cartas	El sistema debe permitir a los usuarios poder ver su colección de cartas.
RF04	Colección cartas en venta	El sistema debe permitir a los usuarios poder ver la colección de cartas en venta disponibles.
RF05	Mejora	El sistema debe permitir al usuario dueño de una carta mejorar la carta.
RF06	Venta	El sistema debe permitir a los usuarios poner en venta una carta al precio que estos consideren.

RF07	Compra	El sistema debe permitir a los usuarios comprar una carta que otro usuario haya puesto en venta.
RF08	Colección cartas otros usuarios	El sistema debe permitir a los usuarios poder el resto de las cartas que existen.
RF9	Juego	El sistema debe permitir a los usuarios jugar con otros usuarios.

Tabla 6 Tabla Análisis de Requisitos Funcionales

4.1.2 Requisitos de información

Nombre	Descripción
Funcionalidad	Toda la funcionalidad que proporciona el sistema debe residir en la blockchain mediante contratos inteligentes.
Cartas	Todas las cartas existentes deben estar almacenados en la blockchain. Esto incluye toda la información que va a tener una carta: • Imagen • Identificador • Nombre • Tipo • Ataque • Defensa • Vida • Nivel
Dueños	La información sobre el dueño de cada una de las cartas debe estar almacenada en la blockchain.
Cartas en Venta	La información sobre las cartas en venta debe estar almacenadas en la blockchain. Esto incluye tanto el identificador de la carta como el precio de la carta.
Cuentas	Las cuentas con el saldo de cada uno de los usuarios deben estar almacenada en la blockchain.

Tabla 7 Tabla Análisis de Requisitos de Información

4.1.3 Requisitos no funcionales

Nombre	Descripción
Backend	El sistema debe disponer de un backend que consista en una serie de contratos inteligentes que residan en la red de Ethereum y que proporcionen todas las funcionalidades del sistema.
Frontend	El sistema debe disponer de un frontend que consiste en una página web con la que interactúa el usuario.

Conexión backend-frontend	El frontend tiene que poder hacer llamadas al backend, para hacer uso de las distintas funcionalidades del sistema.
Red Local	El sistema debe desarrollarse sobre una red local de pruebas que proporcione cuentas con Ether ficticios.
Metamask	El sistema debe poder hacer uso de las cuentas proporcionadas por la red de pruebas a través de Metamask.
Dueño único	Una carta solo puede tener un único dueño al mismo tiempo.
Seguridad	El dueño de la carta es el único que puede interactuar con la misma.
Seguridad en las transacciones	El sistema debe proporcionar un mecanismo seguro para la compra, venta, creación y modificación de una carta.
Descentralización	Se debe proporcionar todos los servicios de la aplicación a través de contratos inteligentes almacenados en la blockchain.
Privacidad	La información sobre los dueños de cada una de las cartas debe ser accesible por todos.

Tabla 8 Tabla Análisis de Requisitos no Funcionales

4.2 Diagrama de caso de uso

Haciendo uno de la herramienta Astah se representa de manera visual las distintas funcionalidades que proporciona el sistema y los actores que la utilizan.

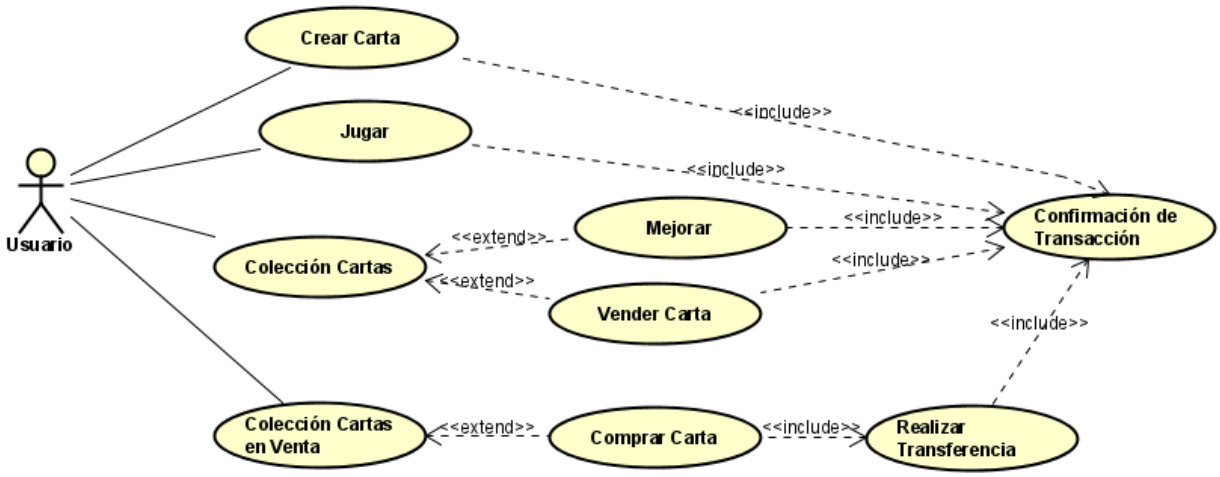


Figura 8 Diagrama de clases

4.3 Descripción de los casos de uso

Los casos de uso corresponden sólo a los requisitos funcionales.

UC01	**Confirmación de Transacción**
Descripción	Metamask permite confirmar una transacción

Versión	1.2
Actor	Usuario.
Dependencias	RF01 Confirmación de transacción
Precondición	El usuario ha accedido a la página web. La cartera del usuario está conectada a la página por Metamask. El sistema ha recibido una petición de transacción por parte del usuario.
Flujo básico	1. El sistema muestra una notificación de Metamask, preguntando si se quiere confirmar la transacción. Además del precio y la cuenta que va a recibir el ETH. 2. El usuario presiona el botón de confirmar de la transacción. 3. El sistema ejecuta la función del contrato inteligente que implementa el servicio. 4. El sistema muestra un mensaje confirmando el éxito de la transacción.
Postcondición	La transacción ha sido exitosa
Flujo alternativo	2a. El usuario presiona el botón de rechazar de la notificación. 1. El sistema informa que la transacción ha sido rechazada. 4b. La transacción ha fallado. 1. El sistema muestra un mensaje indicando que la transacción ha fallado.

Tabla 9 Tabla UC01

UC02	**Crear Carta**
Descripción	El usuario crea una carta al azar de entre todo el repertorio disponible.
Versión	1.2
Actor	Usuario.
Dependencias	RF02 Creación UC01 Confirmación de transacción
Precondición	El usuario ha accedido a la página web. La cartera del usuario está conectada a la página por Metamask.
Flujo básico	1. El usuario presiona el botón de Crear Carta. 2. El sistema muestra un símbolo de cargando, indicando que se está gestionando la petición.

	3. Se ejecuta UC01 Confirmación de Transacción. 4. El sistema informa que la carta ha sido creada de manera exitosa.
Postcondición	El usuario ha añadido una nueva carta al azar a su colección de entre todo el repertorio de cartas. La nueva carta tiene nivel 1. La nueva carta tiene un Id único.
Flujo alternativo	4a No se ha podido crear la carta. 1. El sistema informa que no ha podido crearse la carta.

Tabla 10 Tabla UC02

UC03	**Colección Cartas**
Descripción	El usuario desea ver su colección de cartas.
Versión	1.2
Actor	Usuario.
Dependencias	RF03 Colección cartas UC5 Mejorar UC6 Vender Carta
Precondición	El usuario ha accedido a la página web. La cartera del usuario está conectada a la página por Metamask.
Flujo básico	1. El usuario presiona el botón de Mostrar Carta. 2. El sistema recupera de la red las cartas que posee el usuario. 3. El sistema muestra todas las cartas del usuario.
Postcondición	No existe.
Flujo alternativo	3a. El usuario no posee cartas. 1. El sistema informa que el usuario no tiene ninguna carta.

Tabla 11 Tabla UC03

UC04	**Colección Cartas en Venta**
Descripción	El usuario desea ver la colección de cartas en venta.
Versión	1.2
Actor	Usuario.
Dependencias	RF04 Colección cartas en venta UC7 Comprar
Precondición	El usuario ha accedido a la página web.

	La cartera del usuario está conectada a la página por Metamask.
Flujo básico	1. El usuario presiona el botón de Comprar Carta. 2. El sistema recupera de la red las cartas que están en venta. 3. El sistema muestra las cartas que están en venta.
Postcondición	No existe.
Flujo alternativo	3a. No existen cartas en venta 2. El sistema informa que no existe ninguna carta en venta.

Tabla 12 Tabla UC04

UC05	**Mejorar Carta**
Descripción	El usuario desea mejorar una carta.
Versión	1.2
Actor	Usuario.
Dependencias	RF05 Mejora UC01 Confirmación de transacción UC02 Colección Cartas
Precondición	El usuario ha accedido a la página web. La cartera del usuario está conectada a la página por Metamask. El usuario está visualizando su colección de cartas.
Flujo básico	1. El usuario presiona el botón de mejorar de la carta que quiere mejorar. 2. El sistema informa del coste de la mejora y pregunta que atributo se quiere mejorar. 3. El usuario presiona uno de los botones indicando que quiere mejorar ese atributo. 4. Se ejecuta UC01 Confirmación de Transacción. 5. El sistema informa que la carta ha sido mejorada de manera exitosa.
Postcondición	El usuario ha mejorado una carta.
Flujo alternativo	3a. El usuario presiona el botón de Cancel. 1. El sistema informa que se ha cancelado la operación.

Tabla 13 Tabla UC05

UC06	Vender Carta
Descripción	El usuario quiere vender una carta.
Versión	1.2
Actor	Usuario.
Dependencias	RF06 Venta UC01 Confirmación de transacción UC02 Colección Cartas
Precondición	El usuario ha accedido a la página web. La cartera del usuario está conectada a la página por Metamask. El usuario está visualizando su colección de cartas.
Flujo básico	1. El usuario presiona el botón de vender de la carta que quiere poner en venta. 2. El sistema pregunta por cuanto quiere vender la carta. 3. El usuario introduce un número en el input y presiona el botón Vender o pulsa la tecla Enter. 4. Se ejecuta UC01 Confirmación de Transacción. 5. El sistema informa que la carta ha sido puesta en venta de manera exitosa.
Postcondición	El usuario ha puesto en venta una carta.
Flujo alternativo	4a. La carta ya estaba en venta. 1. El sistema informa que se ha producido un error en la transacción. 2. El sistema informa que la carta ya estaba en venta.

Tabla 14 Tabla UC06

UC07	Comprar Carta
Descripción	El usuario quiere vender una carta.
Versión	1.2
Actor	Usuario.
Dependencias	RF07 Compra UC01 Confirmación de transacción UC03 Colección Cartas en Venta
Precondición	El usuario ha accedido a la página web. La cartera del usuario está conectada a la página por Metamask. El usuario está visualizando la colección de cartas en venta.

Flujo básico	1. El usuario presiona el botón comprar de la carta que quiere comprar. 2. El sistema informa sobre el precio de la carta y pregunta si se quiere comprar. 3. El usuario presiona el botón OK. 4. Se ejecuta UC01 Confirmación de Transacción. 5. El sistema informa que la carta se ha comprado de manera exitosa.
Postcondición	El anterior dueño recibe una transferencia con la cantidad establecida en el precio de la carta por parte del usuario. El usuario añade la carta a su colección.
Flujo alternativo	3a. El usuario presiona el botón de Cancel. 1. El sistema informa que se ha cancelado la operación.

Tabla 15 UC07

UC08	**Jugar**
Descripción	El usuario desea jugar al juego.
Versión	1.2
Actor	Usuario.
Dependencias	RF08 Colección cartas de otros usuarios RF09 Juego UC01 Confirmación de transacción
Precondición	El usuario ha accedido a la página web. La cartera del usuario está conectada a la página por Metamask.
Flujo básico	1. El usuario presiona el botón de Jugar. 2. El sistema recupera de la red las cartas que el usuario posee. 3. El sistema recupera de la red el resto de las cartas que existen. 4. El sistema muestra una interfaz con las cartas que el usuario posee y con el resto de las cartas. 5. El usuario selecciona la carta con la que quiere luchar y contra la que se quiere enfrentar. 6. El usuario presiona el botón de LUCHAR. 7. Se ejecuta UC01 Confirmación de Transacción 8. El sistema muestra el resultado de la lucha.
Postcondición	No existe.

Flujo alternativo	2a. El usuario no tiene cartas. 1. El sistema informa que el usuario no tiene cartas para jugar. 3ª. El resto de los usuarios no tienen cartas. 1. El sistema informa que el resto de los usuarios no tienen cartas

Tabla 16 Tabla UC08

4.4 Diagrama de despliegue

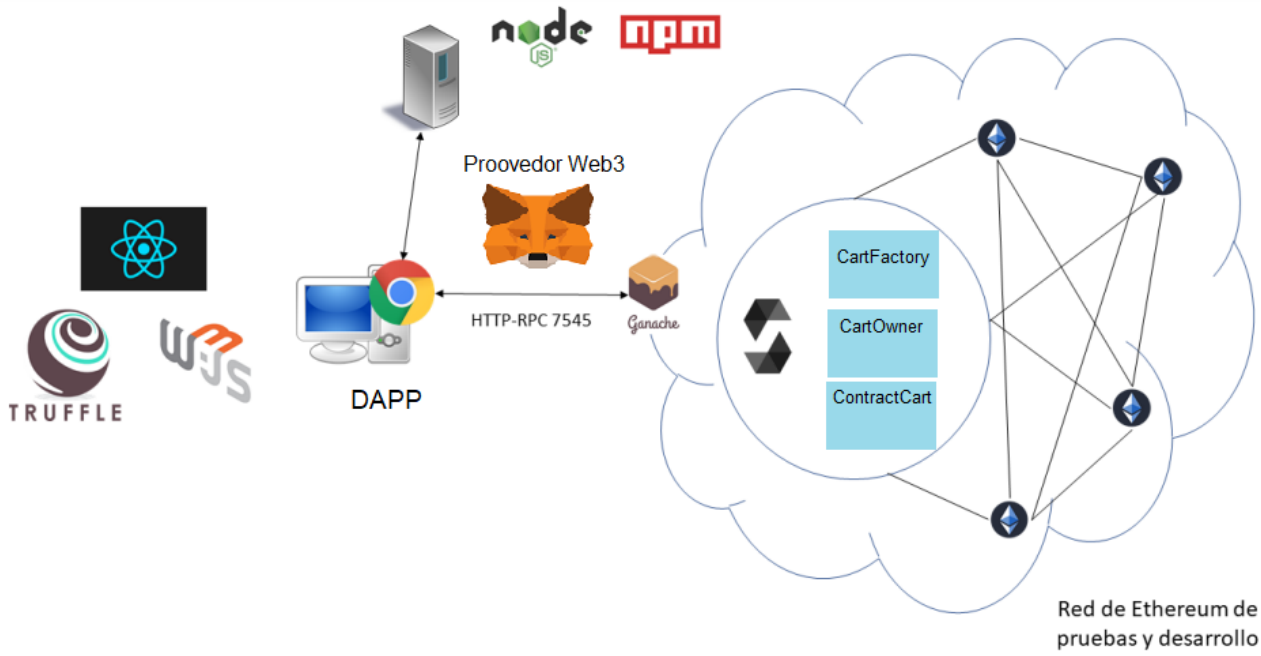


Figura 9 Diagrama de Despliegue. Adaptado de [13]

La aplicación descentralizada se desarrolla usando Truffle, React y Web3.js. Librerías disponibles en node.js que se agregan al proyecto gracias al instalador npm.

Se utiliza Ganache para generar una red de pruebas de Ethereum.

A través de una serie de comandos proporcionados por Truffle, se almacenan los contratos inteligentes desarrollados utilizando el lenguaje Solidity en la blockchain.

Se utiliza un proveedor de Web, en este caso Metamask, para conectar la aplicación con la red Ethereum y poder ejecutar las funciones de los contratos, vía protocolo HTTP en la dirección 7545 (dirección de la red).

La aplicación se ejecuta en Google Chrome debido a la facilidad de utilizar Metamask en este navegador.
En el siguiente capitulo se hará una explicación más detallada de cada una de las herramientas mencionadas.

Capítulo 5 Diseño

Una vez identificado los requisitos del proyecto y analizado las herramientas necesarias para desarrollarlo. La siguiente etapa consiste en describir de manera detallada el diseño de la página web, junto con el de las cartas y el propio juego.

También, se añade más información sobre cada una de las herramientas utilizadas y su finalidad, para identificar de manera más clara que papel desempeñan y porque son necesarias para el desarrollo del propio proyecto.

5.1 Arquitectura del sistema

El diseño del sistema está basado en el patrón arquitectónico *Model-View-ViewModel* (MVVM).
El patrón MVVM ayuda a separar la lógica de negocios y presentación, de la interfaz de usuario.
Se distinguen tres componentes principales cada uno con un propósito diferente [14]:

- **Modelo:** Representa el modelo de dominio de la aplicación que suele incluir un modelo de datos y la lógica del negocio.

- **Vista:** Responsable de definir el diseño y apariencia de lo que ve el usuario por pantalla.

- **Modelo de vista:** Enlaza la vista con el modelo y notifica a la vista de los cambios del modelo. Define la funcionalidad que ofrece la interfaz del usuario y la vista es la encargada de determinar como se muestra esa funcionalidad.

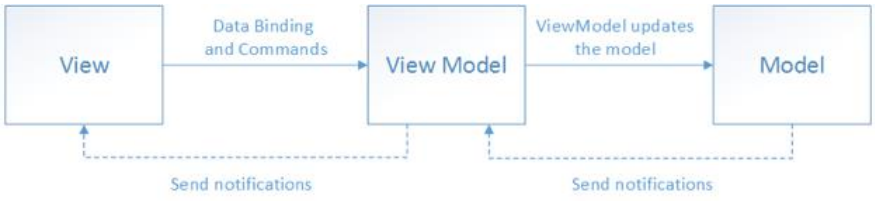


Figura 10 Diagrama del patrón MVVM. Extraído de [14]

La vista conoce el modelo de vista y el modelo de vista conoce el modelo, pero el modelo no conoce el modelo de vista y el modelo de vista no conoce la vista. En otras palabras, el modelo de vista a la vista del modelo y permite realizar cambios en el modelo sin afectar a la vista.

5.1.1 MVVM en el proyecto

- **Vista:** Se utiliza React para crear la vista en su totalidad.

- **Modelo de vista** [15]**:** A partir de React 16.8 se añaden una cosa llamada Hooks. En concreto, existen dos Hooks que proporcionan una manera de usar un estado en un componente sin necesidad de escribir una

clase: *useState()*, que se va a usar para definir un estado y una función para poder actualizarlo. De este modo, cuando se renderice un componente será siempre el valor más reciente devuelto por *useState().* Por otro lado, se tiene *useEffect(),* que permite observar los cambios de los componentes y actualizar automáticamente la vista. De esta manera se consigue separar la vista del modelo.

- **Modelo:** Serán los datos almacenados en la blockchain a través de la ejecución de los smart contracts.

5.2 Atom Web Design

Metodología que surge a principios del 2013 por Brad Frost, que consiste en eliminar las inconsistencias sin sentido en los diseños de las interfaces de usuario.

Brad Frost explica cómo se puede diseñar una interfaz web descomponiendo en unidades más pequeñas denominadas átomos y reutilizando estas para generar elementos más grandes a través de la composición.

Se distinguen 5 tipos de componentes:

- **Átomos.** Unidad más básica indivisible y con sentido por sí misma. Equivalen a los elementos más básicos como botones títulos o iconos.

- **Moléculas.** Agrupaciones de átomos que funcionan como una unidad y adquieren nuevas características. Equivalen a un menú de navegación o una caja de búsqueda.

- **Organismos**. Conjunto de moléculas y átomos. Constituyen unidades más complejas como cabeceras.

- **Plantillas.** Grupos de organismos colocados de manera conjunta para formar páginas. Conocidos como *wireframes* que dan contexto a las unidades más pequeñas.

- **Páginas.** Diseño final combinando un conjunto de plantillas dando como resultado el proyecto completo.

5.2.1 Atom Web Design en el proyecto

Para la realización del proyecto se han generado numerosos componentes. Cada uno de estos equivale a un programa JavaScript.

- **App.** Componente padre que agrupa todos los componentes, es decir, es el proyecto completo.

- **AlbumCartas.** Componente que se encarga de mostrar la colección de cartas del usuario.

- **AlbumCartasVenta.** Componente que se encarga de mostrar la colección de cartas en venta.

- **Botones**. Componente que muestra los botones de la carta. Distingue entre si la carta es del propio usuario o es una carta para comprar.

- **Carrusel.** Componte que permite seleccionar y visualizar las cartas tanto del usuario como del enemigo para el juego.

- **Carta.** Componente que muestra una carta. Formado por la imagen y cada uno de los atributos de la carta, incluido el Id que es único para cada una. Además de los botones para poder mejorar, vender o comprar la carta.

- **Footer.** Componente que añade el icono de retorno en la esquina inferior derecha de la página.

- **Ganador.** Componente que muestra el resultado de la lucha en el juego cuando el usuario gana.

- **Header.** Componente que contiene información del proyecto incluida la dirección de descarga del código y datos del desarrollador.

- **Inicio.** Componente que muestra la vista de inicio de la página.

- **Item.** Componente que muestra la carta en el Carrusel. El Carrusel por lo tanto estará formado por una colección de ítems, donde cada ítem es una carta distinta.

- **Jugar.** Componente que muestra la vista para el juego de cartas.

- **Loading.** Componente que muestra un símbolo de cargando cuando se está creando una carta.

- **Main.** Componente que agrupa a los componentes AlbumCartas, AlbumCartasVenta, Inicio y Jugar, es decir todas las vistas de la aplicación.

- **Perdedor.** Componente que muestra el resultado de la lucha en el juego cuando el usuario pierde.

Todos ellos se generan con la ayuda de la librería reactstrap como ya se mencionó cuando se describió la vista. Esta librería incorpora muchos tipos distintos de componentes elaborados con Bootstrap.

5.2.2 Diagrama de Componentes

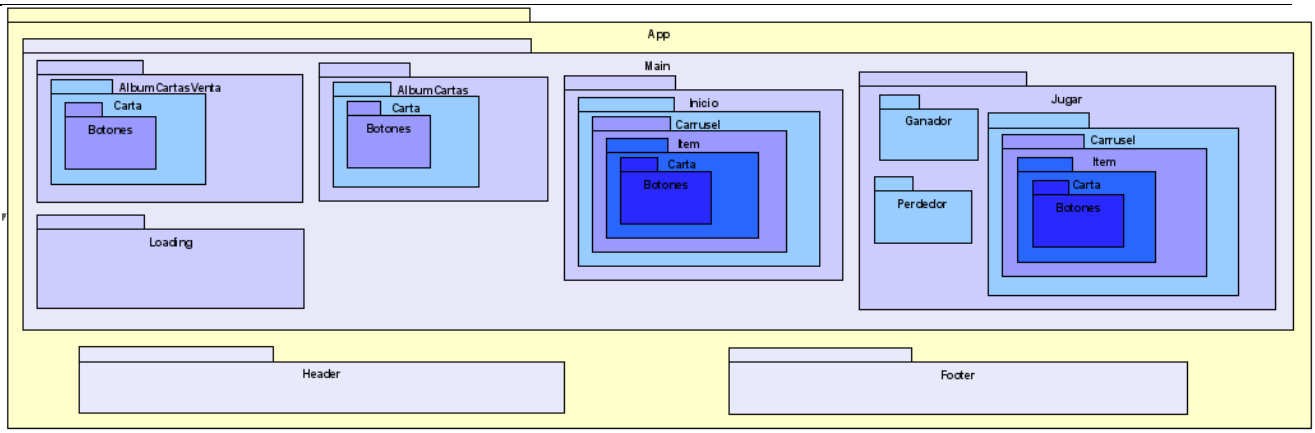


Figura 11 Diagrama de Componentes

En la Figura 10 se muestra como se ha utilizado el patrón de diseño basado en componentes para realizar la página web. El uso de componentes como se menciona anteriormente facilita la reutilización de los mismos para generar componentes más complejos, hasta generar la propia página

5.3 Diseño de la interfaz de usuario

El diseño de la interfaz de usuario se llevó a cabo teniendo en cuenta otras aplicaciones web. En la inmensa mayoría de aplicaciones que trabajan con NFT, entre ellas OpenSea una de las más conocidas muestra estos en forma de álbum como se puede ver en la Figura.

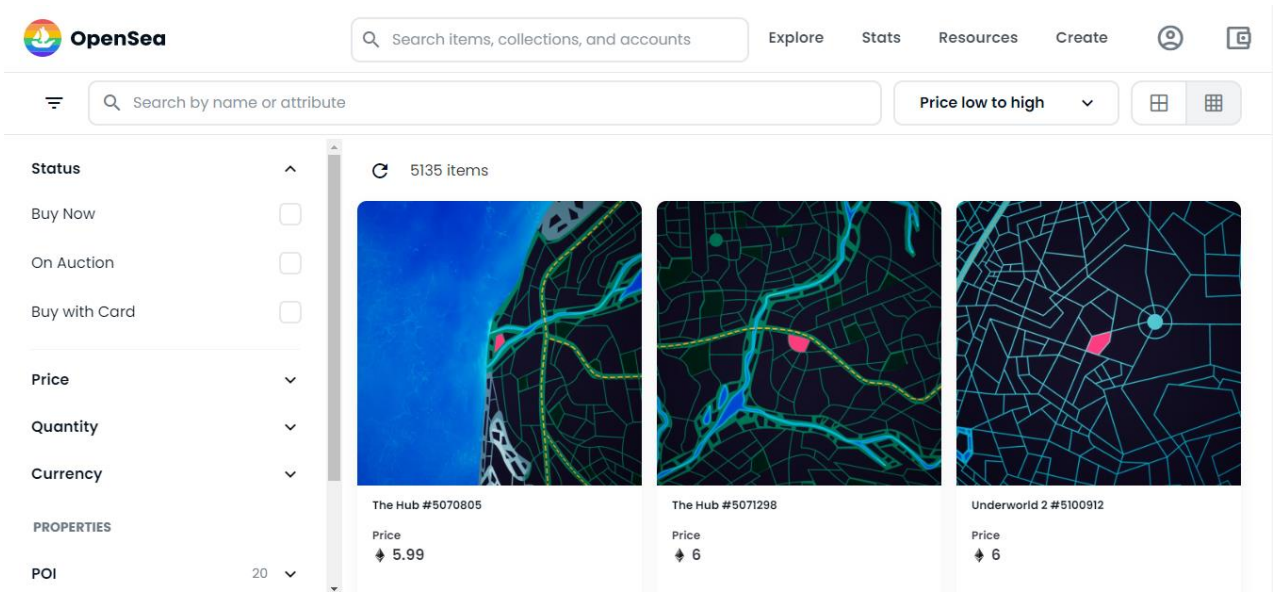


Figura 12 Captura de la página OpenSea

Teniendo en cuenta como muestran los NFT este tipo de páginas. Se elabora un diseño que sea intuitivo y fácil de usar a través de la realización de bocetos con la herramienta Axure RP 10.

5.3.1 Bocetos

Para la interfaz de inicio se ha decidido mostrar las imágenes de toda la colección de cartas posibles, junto a una frase que resuma brevemente la funcionalidad que ofrece la aplicación.

En cuanto a los elementos comunes que van a compartirse con otras vistas, se encuentra el encabezado donde se añade información sobre el proyecto y el estudiante al pulsar sobre el icono.

También, en la parte superior se ubican una serie de botones que, al pulsar sobre cada uno de ellos, se muestra una vista que implementa una funcionalidad distinta. Además, en la esquina inferior derecha se añade un icono para volver al principio.

Figura 13 Boceto Inicio

Para la interfaz que muestra la colección de cartas de un usuario, se ha decidido por mostrar estas en forma de álbum al pulsar sobre el botón de Mostrar Carta.

En cuanto a la propia carta, se ha decidido separar el identificador del resto de atributos para poder ver de manera clara que cada una de las cartas posee un identificador único al ser cada una de ellas un NFT.

Igualmente, todas las cartas tienen unos botones para poder vender o mejorar alguno de los atributos.

Figura 14 Boceto colección de cartas

Para mostrar la colección de cartas en venta, se ha seguido el mismo diseño, incluyendo la funcionalidad que en este caso corresponde a la compra de la carta también a través de botones.

Figura 15 Boceto colección cartas en venta

Para la interfaz del juego se ha decidido representar las colecciones tanto del usuario como del resto en forma de carrusel, permitiendo así identificar mejor que dos cartas son las que van a enfrentarse. Además, se ha dejado el espacio central para poder visualizar el resultado.

Figura 16 Boceto Juego

5.4 Diseño de las cartas y dinámica de juego

El juego consiste en un juego de cartas de rol. Todos estos juegos tienen en común, que las cartas tienen una serie de atributos para medir su fuerza y una imagen representativa que tiene relación con el nombre de la carta.

Es necesario que todas las cartas tengan un patrón común y las imágenes también tengan concordancia, para poder identificar que todas forman parte de una misma colección.

Para ello se seleccionaron una serie de imágenes del artista Gordon Johnson en la página de Pixabay [16] . Esta consiste en un sitio web donde los artistas suben imágenes libres de cualquier derecho de autor.

Este artista tiene una serie de diseños que comparten un estilo de dibujo en blanco y negro. Además, dentro de este conjunto se consigue diferenciar según el personaje que se dibuja. Gracias a esto se consigue dividir las cartas en varios tipos:

- Animal
- Guerrero
- Esqueleto
- Mitológica
- Dios

Cada uno de los tipos empezando desde Animal hasta Dios va teniendo menor probabilidad de aparecer cuando se crea una carta. Con esto se fomenta que los usuarios compren más cartas para ver si tienen suerte y consiguen una carta rara, o que compren cartas de este tipo a otros usuarios. Además, cada tipo de carta comparte características y cuanto más rara, mejores atributos base va a tener.

Una vez seleccionada las imágenes se identificaron una serie de atributos en cada una de las cartas para poder añadirle funcionalidad al juego:

- Nombre
- Tipo
- Ataque
- Defensa
- Vida

También, se añadieron dos atributos más; el primero denominado Id que permite identificar a una carta y asegurarse que es única (todas las cartas tienen un id distinto).

Por último, se añadió un nuevo atributo denominado nivel, que permite visualizar cuando se ha mejorado alguno de los atributos, indicando cuantas mejoras se han realizado en la carta. La idea consiste en que cuando se crea una carta esta tiene nivel 1 y se puede ir mejorando.

Una vez identificado los campos y basándose en modelos de otras cartas, se decide el siguiente formato:

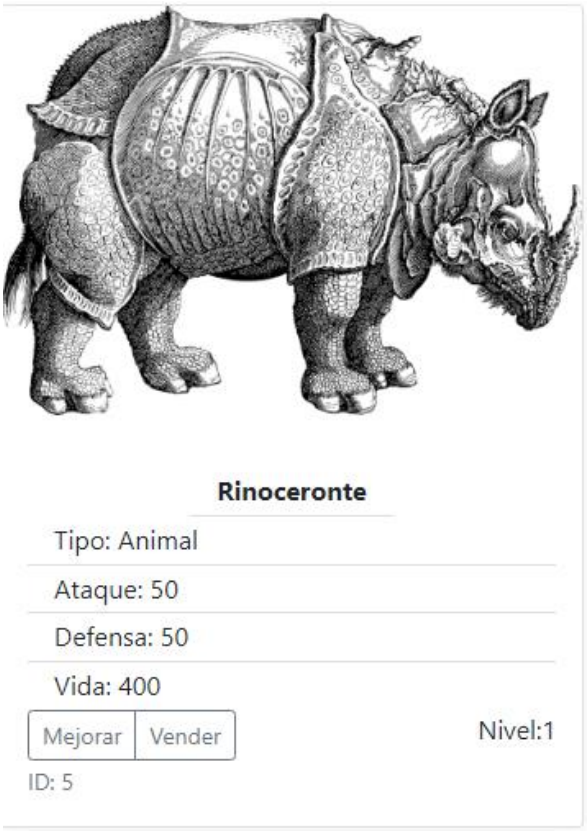


Figura 17 Diseño Carta

La imagen en grande junto al nombre para identificar de manera rápida la carta, junto a esto una lista donde se muestran el resto de los atributos. Para añadir la funcionalidad correspondiente se añade una serie de botones que facilita la selección. Por último, se decide mostrar algo separado tanto el nivel como el id.

Una vez selecciona las cartas y los atributos, se decide cual va a ser la dinámica del juego. Como el objetivo no es establecer una serie de reglas para crear un juego complejo, se desarrolla una idea sencilla.

La idea básica sobre la que se crean todos los juegos de rol consiste en enfrentar una carta contra otra y gana la que mejores atributos posee. A partir de esta idea, los mejores juegos añaden otros muchos condicionantes que hacen el juego más complejo y entretenido.

En este caso únicamente se ha introducido una probabilidad de victoria y derrota, que varía según si la suma de los atributos ataque, defensa y vida es mayor que la misma suma de la otra carta. En el caso de que la carta sea mejor la probabilidad de victoria es de un 70%, mientras que, si esta es peor, la probabilidad de victoria se reduce a un 30%.

5.5 Herramientas Utilizadas

5.5.1 Visual Studio Code

Visual Studio Code [17] es un editor de código muy potente, pero de uso muy sencillo. Permite trabajar con Node.js, Javascript y TypeScript, pero puede soportar otros muchos lenguajes como C++, Java y Python entre todos. También proporciona muchas otras ayudas como autocompletado e identificación de errores de programación. Aparte es necesaria para poder trabajar con Truffle que es el sistema de desarrollo elegido para desarrollar los contratos.

5.5.2 Node.js

Node.js [18] es un entorno en tiempo de ejecución de JavaScript que incluye todo lo necesario para ejecutar un programa escrito en JavaScript.

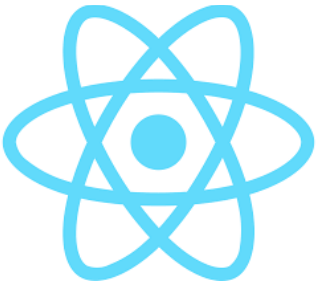
El código en Node.js se encuentra estructurado por módulos. Que hay que ir agregando a medida que se trabaja con ellos, para ello sirve NPM.

NPM es un gestor de paquetes que permite agregar dependencias de forma simple a un proyecto, obteniendo cualquier librería con una línea de código.

5.5.3 NVM

Software que permite instalar y gestionar diversas versiones de Node.js en un mismo ordenador.

5.5.4 React



React [19] es una librería de Javascript que ayuda a crear interfaces de usuario de manera más ordenada y con menos código que si solo se utiliza JavaScript puro o alguna librería centrada en la manipulación del DOM(Document Object Model) como jQuery.

Siguiendo el patrón de programación basada en componentes, se consigue crear una interfaz a partir de un componente que encapsula su funcionamiento y estructura. Este componente se basa a su vez en otros para ofrecer otras funcionalidades. Esto consigue separar la interfaz en partes independientes, reutilizables y que se pueden analizar por separado.

Para crear estos componentes con React es aconsejable utilizar JSX. JSX es una extensión de la sintaxis de JavaScript que recuerda a HTML, pero con todo el poder de JavaScript y permite crear elementos de React.

La ventaja frente a otras librerías es que React actualiza la vista de una manera mucho más eficiente gracias al llamado Virtual Dom, que en resumen es es una representación del DOM en memoria.

Actualizar el Virtual Dom es mucho más rápido que el DOM real y permite actualiza solo las partes que se necesiten.

Esto se consigue ya que cada componente de React tiene su propio estado. Cuando el estado cambia, el componente se actualiza. En el capítulo siguiente se detalla como React, actualiza el estado de un componente.

5.5.5 Web3.js

Web3.js [20] es una colección de librerías de JavaScript que permiten interactuar con un nodo local o remoto de Ethereum usando HTTP, IPC o WebSocket. El nombre hace referencia al termino de Web3.0 que describe un nuevo modelo de web basado en la tecnología blockchain.

Web3.js implementa las especificaciones genéricas de RPC (Remote Procedure Call) en formato JSON.

Para que una aplicación funcione en Ethereum es necesario utilizar el objeto **web3** proporcionado por la librería. La librería a su vez se comunica con un nodo local a través de llamas PRC.

En concreto, Web3 contiene un objeto **eth** (web3.eth) para interacciones específicas de cadenas de bloques de Ethereum.

5.5.6 Truffle

Truffle [21] es un marco de desarrollo para Ethereum que facilita el desarrollo de aplicaciones descentralizadas, ya que ayuda al despliegue de los contratos inteligentes, vinculación de bibliotecas y proporciona mecanismos para probar los contratos incluyendo una consola con la que hacer llamadas a estos mismos.

5.5.7 Ganache

Ganache [22] permite activar una *blockchain* de Ethereum de manera local ofreciendo todas las características necesarias para desarrollar una aplicación en Ethereum y visualizar todas las transacciones, contratos y cuentas, incluyendo el saldo la dirección y la clave privada.

5.5.8 Metamask

Metamask [23] es una extensión del navegador (Google Chrome, Firefox o Opera entre otros) que genera contraseñas y claves en el propio dispositivo proporcionando un acceso privado a la cuenta y los datos.

5.5.9 Ganache

Solidity [10] es un lenguaje de programación de alto nivel, con una sintaxis similar a JavaScript, orientado a desarrollar contratos inteligentes sobre la red de Ethereum.

5.5.10 Axure RP 10

Axure RP 10 [24] es una aplicación para desarrollar prototipos muy sin la necesidad de tener que programar.

5.5.11 GitLab

GitLab [25] es un repositorio para el control de versiones y ciclo de vida usando Git. Es una aplicación nativa en la nube que facilita el flujo de trabajo y las revisiones del código. Además, vincular un proyecto de GitLab con Visual Studio Code es muy fácil, permitiendo trabajar desde diferentes dispositivos.

5.5.12 astah

astah [26] es un software utilizado para el desarrollo del software que en entre otras cosas permite crear diagramas UML o diagramas de flujos. Es un software de modelado utilizado en distintas asignaturas del grado y cuenta con una licencia de uso para estudiantes de este.

Capítulo 6 Implementación

Desde el diseño y con las herramientas técnicas usadas se ha realizado la siguiente implementación.

6.1 Implementación Ethereum

Hay que tener en cuenta que se trata de una aplicación descentralizada. Esto implica que todos los datos y funcionalidades deben estar almacenados en la blockchain y se puede acceder a ellos a través de la ejecución de contratos inteligentes.

En la especificación de requisitos, ya se menciona tanto la información como la funcionalidad que debe tener la aplicación, pero a continuación se hace un resumen como aclaración.

- Información que hay que almacenar:
- Todas las cartas que se han creado.
- Dueño de cada una de las cartas.
- Número de cartas que posee cada usuario.
- Número de cartas en venta que tiene un usuario.
- Todas las cartas que están en venta y el precio de cada una de estas.
- Precio para mejorar una carta.
- Precio para comprar una carta.

Funcionalidad que debe tener la aplicación:

- Consultar el dueño de una carta.
- Consultar el número de cartas de un usuario.
- Consultar el número de cartas en venta que tiene un usuario.
- Función para extraer las características de una carta.
- Función para devolver todas las cartas que posee un usuario.

- Función para devolver todas las cartas que puede comprar un usuario.
- Función para devolver todas las cartas que existen y no pertenecen a un usuario.
- Función para realizar la transferencia de una carta de un usuario a otro.
- Función para realizar la transferencia del precio de la carta de un usuario a otro.
- Función para poder poner en venta una carta.
- Función para poder comprar una carta.
- Función para poder mejorar una carta.
- Función para implementar el juego de cartas.

La manera de implementar todas las funcionalidades y de almacenar la información es a través de los *smart contracts*. Además, hay que asegurarse que las transacciones son seguras, es por ello por lo que hay que incluir mecanismos de seguridad. En concreto para el desarrollo de este proyecto se han elaborado 3 contratos distintos.

6.1.1 CartFactory

Este contrato incluye la creación de las cartas y permite cambiar el precio que cuesta, pero solo al dueño que desplego el contrato en la red, como mecanismo de seguridad.

Además, incluye un mapping que relaciona cada carta con su dueño y cada dueño con el número de cartas que posee.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "../node_modules/@openzeppelin/contracts/access/Ownable.sol";

contract CartFactory is Ownable {

// Definicion de una carta

  struct Cart {
    uint id;
    string imagen;
    string name;
    string tipo;
    uint16 nivel;
    uint16 ataque;
    uint16 defensa;
```

```
    uint16 vida;
  }


  uint cartPrice = 1 ether; // Precio que cuesta crear una carta

  Cart[] public carts; // Array que almacena todas las cartas de la aplicación

  mapping (uint => address)  cartToOwner;  // Mapeo que relaciona la carta con su
dueño
  mapping (address => uint) ownerCartCount; // Mapeo que relaciona un usuario con
el numero de cartas que posee


// Funcion para cambiar el precio de la carta
  function changePrince(uint _price) internal onlyOwner {
    cartPrice = _price;
  }

// Funcion para crear una carta, si se paga el precio que cuesta crear una
(cartPrice)
  function createRandomCart(string memory _imagen, string memory _name, string
memory _tipo,uint16 _nivel,uint16 _ataque,uint16 _defensa, uint16 _vida) public
payable {
     require(msg.value == cartPrice);  // se obliga a que se pague el precio de la
carta
    uint id = carts.length ; // se asegura que el nuevo id es unico para la carta.
    carts.push(Cart(id,_imagen, _name,_tipo,_nivel, _ataque, _defensa, _vida));
//se agrega la carta al array que las contiene
    cartToOwner[id] = msg.sender; // se identifica al usuario quien ha llamado a la
funcion como el dueño de la carta.
    ownerCartCount[msg.sender]++; // se aumenta el numero de cartas que posee el
dueño

  }


}
```

6.1.2 CartOwner

Este contrato implementa toda la funcionalidad que proporciona la aplicación a excepción de la transferencia de ETH de una cuenta a otra y de la funcionalidad del propio juego que por motivos de simplicidad no se consideró necesario almacenar en la blockchain.

Además, incluye el estándar ERC721 de la biblioteca de @openzeppeling que incluye el estándar de NFT y las funciones que debe incluir uno.

Por último, se incluyen las siguientes restricciones para asegurar el control de las cartas:

- Solo el dueño de una carta puede mejorar o poner en venta la misma
- Un usuario solo puede comprar cartas que no sean suyas.
- Solo el dueño de una carta puede trasferir está a otro usuario.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "../node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "./CartFactory.sol";
import "./ContractCart.sol";

// necesitas añadir el constructor de ERC721 para añadir todas las propiedades y
funcionalidades de un NFT
contract CartOwner is ERC721("CartaNFT", "NFT"),ContractCart, CartFactory {

  mapping (uint => bool) cartApprovals;   // Mapeo que indica que cartas estan en
venta
  mapping (address => uint) ownerCartSold;  //Mapeo que indica el numero de cartas
que tiene en venta cada usuario.
  mapping(uint => uint) cartPrices;    // Mapeo que indica el precio de cada una de
las cartas.

  modifier noOwner(uint _cartId) {  // Restriccion para que el usuario que compra
la carta no pueda ser el dueño
  require(msg.sender != cartToOwner[_cartId]);
    _;

}
   modifier onlyOwnerOf(uint _cartId) { // Restriccion para que el usuario deba ser
el dueño de la carta
    require(msg.sender == cartToOwner[_cartId]);
    _;
  }

  uint levelPrice = 0.01 ether;  // se establece el precio que cuesta mejorar una
carta
  uint numeroCartasEnVenta = 0;  // Inicialmente el numero de cartas en venta es 0

//funcion para devolver el ID de todas las cartas que pertenecen a un usuario.

  function getCardsByOwner(address _owner) external view returns(uint[] memory ) {
    uint[] memory cartas = new uint[](ownerCartCount[_owner]);
    uint ncartas = 0;
    for (uint i = 0; i < carts.length; i++) {
      if (cartToOwner[i] == _owner) {
        cartas[ncartas] = i;
```

```
        ncartas++;
      }
    }
    return cartas;
  }


  // funcion para devolver todas las cartas de otros usuarios.

  function getEnemyCartsByOwner(address _owner) external view returns(uint[] memory
) {
    require(carts.length - ownerCartCount[_owner] > 0);
    uint[] memory cartas = new uint[](carts.length - ownerCartCount[_owner]);
    uint ncartas = 0;
    for (uint i = 0; i < carts.length; i++) {
      if (cartToOwner[i] != _owner) {
        cartas[ncartas] = i;
        ncartas++;
      }
    }
    return cartas;
  }

   // funcion para devolver el numero de cartas que tienen el resto de usuarios

  function balanceEnemyCarts(address _owner) external view returns(uint ) {
   return(carts.length - ownerCartCount[_owner]);

  }


  //funcion para devolver las cartas en venta

  function getCardsInSold(address _owner) external view returns(uint[] memory ) {
    uint[] memory cartas = new uint[](numeroCartasEnVenta -
ownerCartSold[_owner]);

    uint ncartas = 0;
    for (uint i = 0; i < carts.length; i++) {
      if (cartApprovals[i] && cartToOwner[i] != _owner ) {
        cartas[ncartas] = i;
        ncartas++;
        }
    }
    return cartas;
  }

  // funcion para devolver el numero de cartas que hay en venta

  function balanceSoldCarts(address _owner) external view returns(uint ) {
   return(numeroCartasEnVenta - ownerCartSold[_owner]);
```

```
  }

// funcion para devolver el numero de cartas que tiene un usuario

  function balanceOf(address _owner) public override view returns (uint256
_balance) {
    return ownerCartCount[_owner];
  }

// funcion para devolver el dueño de una carta.

  function ownerOf(uint256 _tokenId) public override  view returns (address _owner)
{

    return cartToOwner[_tokenId];
  }

// funcion para devolver el precio que cuesta una carta

  function priceOf(uint256 _tokenId) public view returns (uint256 _cartPrice){
    return cartPrices[_tokenId];
  }

// funcion para transferir una carta de un usuario a otro

  function _transfer(address _from, address _to, uint256
_tokenId)  internal  override {
    ownerCartCount[_to]++;
    ownerCartCount[_from]--;
    cartToOwner[_tokenId] = _to;
    cartApprovals[_tokenId] = false; // la carta ya no esta en venta
    ownerCartSold[_from]--; // el dueño de la carta tiene una carta menos en venta
    numeroCartasEnVenta--; // hay una carta menos en venta

  }


  function transfer(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    _transfer(msg.sender, _to, _tokenId);
  }

// funcion para poner en venta una carta al precio que se quiera

  function approve(uint256 _tokenId,uint _price) public onlyOwnerOf(_tokenId) {
    require(!cartApprovals[_tokenId]); //restriccion que asegura que la carta no
tiene que estar en venta
    cartApprovals[_tokenId] = true;
    cartPrices[_tokenId] = _price;
    numeroCartasEnVenta++;
    ownerCartSold[msg.sender]++;
```

```
  }


// Funcion para transferir la carta y el precio de la carta al antiguo dueño

  function takeOwnership(uint256 _tokenId,address _to) public payable
noOwner(_tokenId) {    // pongo que sea payable  el que compra la carta no puede
ser el dueño de la misma
   require(cartApprovals[_tokenId] ); // la carta tiene que estar en venta
   require(msg.value >= cartPrices[_tokenId]);  //si pagas el precio de la carta
    address owner = ownerOf(_tokenId);
    _transfer(owner, _to, _tokenId);
    // realizamos la tranferencia de los ETH que vale la carta al dueño de esta
    address payable addr4 = payable(owner);
    send_ETH(addr4,msg.value);


  }


// funcion para mejorar el atributo que se quiera de una carta si se paga lo que
cuesta mejorar.
function levelUp(uint _cartId, string memory _caracteristica) external payable {
    require(msg.value == levelPrice);  // restriccion para asegurar que se paga el
precio
   carts[_cartId].nivel++;  //el nivel de la carta aumenta

   // Se comprueba que atributo se quiere mejorar
   if (keccak256(bytes(_caracteristica)) == keccak256(bytes("ataque"))){
    carts[_cartId].ataque  = carts[_cartId].ataque +10 ;
   }
   else if (keccak256(bytes(_caracteristica)) == keccak256(bytes("defensa"))){
    carts[_cartId].defensa  = carts[_cartId].defensa +10 ;
   }
     else if (keccak256(bytes(_caracteristica)) == keccak256(bytes("vida"))){
    carts[_cartId].vida  = carts[_cartId].vida +10 ;
   }
  }

// funcion que devuelve el precio que cuesta mejorar una carta

  function getLevelPrice() external view returns(uint _price) {
    return levelPrice;
  }
}
```

6.1.3 ContractCart

Implementa la transferencia de ETH entre dos cuentas.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0;

// Contrato para realizar una transferencia de ETH entre dos cuentas
contract ContractCart {
    address payable[] recipients;
    mapping(address => uint256) public balanceAccount;
    address public cuenta;

    constructor() {
        cuenta = msg.sender;
    }

    function send_ETH(address payable recipient,uint _amount) payable public {

        fund(recipient, _amount);
    }

   // funcion para transferir a una cuenta el ETH que se indique

    function fund(address payable recipient, uint _amount) internal {

        recipient.transfer( _amount);
    }
}
```

6.2 Implementación del proyecto

A continuación, se detallan los pasos llevados a cabo para implementar el proyecto.

1. Se crea una nueva carpeta donde estará almacenado todos los archivos y carpetas del proyecto.

2. En esa misma carpeta ejecutar los siguientes comandos:

$$npm\ install\ -g\ truffle$$
$$truffle\ unbox\ react$$

Esto crea todos los directorios y archivos de configuración necesarios para trabajar con Truffle y poder interactuar con ellos a través de una aplicación de React.

A partir de la estructura general del proyecto, se desarrolla la aplicación final. La estructura final del proyecto es la siguiente:

- **client:** Directorio que contiene toda la aplicación de React

    - **nodes_modules:** Almacena todas las librerías de Node.js que se instalan para realizar la aplicación de React.

    - **public:** Contiene el archivo .html que genera la página web.

    - **Src**

        - **contracts:** Contendrá todos los contratos en formato. json para poder trabajar con web3.

        - **componentes:** Carpeta creada por el estudiante para almacenar todos los componentes creados por React.

        - **css:** Carpeta que almacena todos los archivos con extensión .css.

        - **imagenes:** Carpeta que contiene las imágenes de las distintas cartas.

        - Esta Carpeta también contiene otros archivos como App.js (componente principal) y getWeb3.js que permite la conexión con la red de Ganache por HTTP.

- **contracts:** Contiene los contratos escritos en lenguaje Solidity.

- **migrations**: Contiene los archivos de JavaScript que permiten desplegar los contratos en la red de Ganache.

- **node_modules:** Contiene los estándares ERC necesarios para desarrollar una aplicación que permita la creación de NFT entre ellos ERC721. Instalados de la librería de OpenZeppellin.

Hay que mencionar que las imágenes y los atributos de las cartas están almacenados de manera local. La aplicación no puede almacenar este tipo de información de manera directa. La solución para mantener la descentralización sería utilizar una base de datos donde guardar esta información y acceder a ella a través de la ejecución de un contrato.

6.2.1 Librerías utilizadas en el proyecto

- **Bootstrap:** Framework que combina CSS y JavaScript para estilizar los elementos de una página HTML. Ofrece una serie de componentes como menús de navegación y barras de progreso, que facilitan la comunicación con el usuario.

- **Reactstrap:** Librería de React compatible con Bootstrap, que contiene una serie de componentes ya creados que facilitan la construcción de una interfaz.

- **React-elastic-carousel:** Librería de React que implementa un componente de un carrusel flexible.

- **React-moving-text**: Librería de React para animar texto construida con styled-components

- **Styled-components:** Librería para mejorar CSS que facilita diseñar componentes en React.

- **SweetAlert:** Librería para crear mensajes emergentes sencillos y bonitos, que permiten interactuar con ellos.

6.2.2 OpenZepelling

OpenZepelling proporciona estándares construir, automatizar y operar de manera segura aplicaciones descentralizadas en la blockchain.

Contiene entre otras cosas el conjunto de interfaces, contratos y utilidades relacionados con el estándar ERC721 que permite implementar los NFT.

Capítulo 7 Pruebas

Una vez desarrollado el proyecto es necesario comprobar si se cumplen con los requisitos establecidos.

El objetivo de este capítulo consiste concretamente en realizar una serie de pruebas. Esto no es más,que definir una actividad y ejecutarla bajo circunstancias previamente especificadas y registrar su resultado. Con esto se puede evaluar los resultados obtenidos comparándolos con los resultados esperados para localizar fallos en el proyecto.

Se van a distinguir dos tipos distintos de pruebas:

- **Pruebas unitarias:** Consisten en examinar unidades individuales. Suelen ser métodos o funciones que se prueban de manera aislada sin tener en cuenta sus dependencias y se comprueban si cumplen lo que deben hacer.

- **Pruebas de aceptación:** Tras realizar las pruebas unitarias, se examina el comportamiento conjunto de varios componentes o usuarios. Se comprueba que la interacción entre los componentes es la esperada.

7.1 Pruebas Unitarias

Prueba 1	**Crear Carta**
Resultado Esperado	El usuario presiona el botón Crear Carta. Metamask pide confirmación de la transacción. El usuario añade una nueva carta al azar a su colección de entre todo el repertorio de cartas. La nueva carta tiene nivel 1. La nueva carta tiene un Id único.
Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC02 Crear Carta

Tabla 17 Prueba 1

Prueba 2	**Mostrar Carta**
Resultado Esperado	El usuario tiene cartas para mostrar. Se muestran todas las cartas que tiene el usuario.
Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC03 Colección de Cartas

Tabla 18 Prueba 2

Prueba 3	**Mostrar Cartas en Venta**
Resultado Esperado	Existen cartas a la venta para un usuario. Se muestran todas las cartas en venta.

Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC04 Colección de Cartas en Venta

Tabla 19 Prueba 3

Prueba 4	**Mejorar Carta**
Resultado Esperado	El usuario quiere mejorar un atributo de una carta Metamask pide confirmación de la transacción. El usuario mejora una carta. Se muestra el nuevo valor del atributo de la carta.
Resultado Obtenido	Fallo 26/06/2022. Solo se muestra el nuevo valor del atributo cuando se refresca la página. Fallo 04/07/2022 Solo se muestra el nuevo valor del atributo cuando se refresca la página o se cambia de vista.
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC03 Colección de Cartas UC05 Mejorar Carta

Tabla 20 Prueba 4

Prueba 5	**Vender Carta**
Resultado Esperado	El usuario quiere vender una carta Metamask pide confirmación de la transacción. El usuario pone en venta una carta.
Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC03 Colección de Cartas UC06 Vender Carta

Tabla 21 Prueba 5

Prueba 6	**Comprar Carta**
Resultado Esperado	El usuario quiere comprar una carta. Metamask pide confirmación de la transacción. El usuario compra una carta
Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC04 Colección de Cartas en Venta UC07 Comprar Carta

Tabla 22 Prueba 6

Prueba 7	**Jugar**
Resultado Esperado	El usuario quiere jugar y presiona el botón Jugar. Se muestran las cartas del usuario y las del resto.

	El usuario elige con qué y contra que carta quiere luchar. Presiona el botón de luchar. Metamask pide confirmación de la transacción. Se muestra el resultado de la batalla.
Resultado Obtenido	Fallo 26/06/2022. La función que ejecuta el juego no reside en la blockchain por tanto no hay transacción. Éxito 04/07/2022. Se decide no incluir la función del juego en un contrato inteligente debido a la sencillez de la misma.
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC08 Jugar

Tabla 23 Prueba 7

7.2 Pruebas de Aceptación

Prueba 8	**Crear Carta 2**
Resultado Esperado	El usuario 1 crea una carta. El usuario 1 visualiza la carta en su colección. La carta forma parte del repertorio disponible de cartas. La nueva carta tiene todos los atributos sin ninguna mejora. La nueva carta tiene un Id único.La carta forma parte del catálogo de cartas disponibles. La carta tiene todos los atributos sin ninguna mejora.
Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC02 Crear Carta UC03 Colección de Cartas

Tabla 24 Prueba 9

Prueba 9	**Vender Carta 2**
Resultado Esperado	El usuario 1 pone en venta una carta. El usuario 2 puede comprar esa carta.
Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC04 Colección de Cartas en Venta UC06 Vender Carta UC07 Comprar Carta

Tabla 25 Prueba 9

Prueba 10	**Comprar Carta 2**
Resultado Esperado	El usuario 1 compra una carta. El usuario 1 visualiza la carta en su colección.
Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC03 Colección de Cartas UC07 Comprar Carta

Tabla 26 Prueba 10

Prueba 11	**Comprar Carta 3**
Resultado Esperado	El usuario 1 compra una carta. Al usuario 1 ya no le aparece la carta para comprar.
Resultado Obtenido	Fallo 26/06/2022. Solo no aparece la carta cuando se recarga la aplicación. En caso contrario, aparece la carta, pero si se intenta comprar, la transacción provocará un error. Fallo 04/07/2022. Solo no aparece la carta cuando se recarga la aplicación o se cambia de vista. En caso contrario, aparece la carta, pero si se intenta comprar, la transacción provocará un error.
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC04 Colección de Cartas en Venta UC07 Comprar Carta

Tabla 27 Prueba 11

Prueba 12	**Comprar Carta 4**
Resultado Esperado	El usuario 1 compra una carta del usuario 2. El usuario 1 transfiere el precio de la carta al usuario 2. El usuario 2 recibe la transferencia con la cantidad establecida en el precio de la carta.
Resultado Obtenido	Fallo 26/06/2022. El usuario 2 recibe una cantidad distinta al precio de la carta. Éxito 26/06/2022. Se modifica la función del contrato inteligente, para garantizar que el usuario 2 recibe la cantidad establecida por el precio de la carta.
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC07 Comprar Carta

Tabla 28 Prueba 12

Prueba 13	Comprar Carta 5
Resultado Esperado	El usuario 1 compra una carta al usuario 2. El usuario 2 ya no tiene la carta en la colección.
Resultado Obtenido	Éxito 26/06/2022
Caso/s de Uso/s relacionados	UC01 Confirmación de transacción UC03 Colección de Cartas UC07 Comprar Carta

Tabla 29 Prueba 13

Capítulo 8: Conclusiones y trabajo futuro.

8.1 Conclusiones

Una vez finalizado el proyecto, se considera todo el trabajo realizado y si se ha cumplido con los objetivos establecidos en el capítulo 1.

En primer lugar, se ha desarrollado de manera exitosa una aplicación descentralizada en la red Ethereum que permite jugar un juego de cartas donde cada carta es un NFT, a excepción de la información con la que se generan estas mismas que están almacenadas de manera local.

Para poder realizar esta aplicación ha sido necesario adquirir conocimiento sobre desarrollo de aplicaciones en la blockchain. Esto a su vez ha conllevado aprender un nuevo lenguaje de programación como es Solidity para la creación de contratos inteligentes.

A su vez se ha aprendido los mecanismos de verificación de transacciones y como utilizar una cartera Ethereum para facilitar todas estas gestiones.

En resumen, se ha aprendido mucho sobre el mundo de la blockchain y los NFT y las posibilidades que esto permite más allá del mercado de compra y venta de estos activos.

También se ha adquirido conocimiento sobre el desarrollo de aplicaciones web y en concreto mediante React.

Por último, el proyecto ha servido para mejorar la capacidad de gestionar un proyecto. Haciendo uso de distintas herramientas que permiten organizar el trabajo y estableciendo pequeños objetivos que permiten ir dividiendo el trabajo y controlar el progreso y los tiempos.

El resultado del proyecto esta subido y es descargable en un repositorio de GitLab, se puede descargar como se explica en el apéndice II.

8.1 Trabajo Futuro

Como posibles ampliaciones al trabajo se pueden mencionar las siguientes:

- Añadir un filtro de búsqueda para facilitar la visualización de la colección de cartas.
- Añadir un filtro de búsqueda para facilitar la compra de cartas.
- Desarrollar una mecánica de juego mucho más atractiva.
- Mejorar la funcionalidad del juego incluyendo animaciones y otros mecanismos.

- Implementar otros tipos de NFT con características distintas.

- Implementar la aplicación en la red Ethereum real.

- Almacenar las imágenes y atributos de las cartas que se crean en una base de datos y acceder a esta información a través de un contrato inteligente.

- Implementar la funcionalidad del juego una vez mejorado, a través de un smart contract para hacer la aplicación totalmente descentralizada.

Bibliografía

[1] J. F. Franch, "¿Son arte los NFT que se venden como tal?," *The Conversation*. https://theconversation.com/son-arte-los-nft-que-se-venden-como-tal-175953

[2] I. Bashir, *Mastering Blockchain*. Packt Publishing, 2017. [Online]. Available: https://books.google.es/books?id=urkrDwAAQBAJ

[3] M. D'Aliessi, "How Does the Blockchain Work?," *OneZero*. https://onezero.medium.com/how-does-the-blockchain-work-98c8cd01d2ae

[4] Si. Solera, "¿Qué es el blockchain?," *Occam Agencia Digital*. https://www.occamagenciadigital.com/blog/blockchain-que-es-como-funciona#:~:text=En blockchain%2C la transparencia se,del producto desde el origen.

[5] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, no. 3, pp. 183–187, 2017, doi: 10.1007/s12599-017-0467-3.

[6] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," *2017 IEEE Int. Congr. Big Data (BigData Congr.*, pp. 557–564, 2017.

[7] A. M. Antonopoulos and G. Wood, *Mastering ethereum: building smart contracts and dapps*. O'reilly Media, 2018.

[8] CointMarketCap, "Ethereum", [Online]. Available: https://coinmarketcap.com/es/currencies/ethereum/

[9] @corwintines, "GAS Y TARIFAS," *Ethereum.org*. https://ethereum.org/es/developers/docs/gas/

[10] Solidity-es.readthedocs.io, "Solidity."

[11] @corwintines, "CUENTAS DE ETHEREUM," *Ethereum.org*. https://ethereum.org/es/developers/docs/accounts/

[12] European Scrum, "Certificación Scrum Foundations." https://www.europeanscrum.org/certificacion-scrum-foundations.html

[13] J. de Frutos Cerezo, "Desarrollo de un servicio de seguimiento de mercancías basado en la cadena de bloques" Ethereum"," 2019.

[14] J. Davidbritch, nschonni, conceptdev, "El patrón Model-View-ViewModel," *Microsoft docs*. https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[15] "Usando el Hook de estado," *React*. https://es.reactjs.org/docs/hooks-state.html

[16] G. Johnson, "GDJ," *Pixabay*. https://pixabay.com/es/users/gdj-1086657/

[17] Visual Studio Code, "VISUAL." https://code.visualstudio.com/

[18] "Node.js." https://nodejs.org/es/

[19] "React." https://es.reactjs.org/

[20] "web3.js - Ethereum JavaScript API." https://web3js.readthedocs.io/en/v1.7.4/index.html

[21] "Truffle." https://trufflesuite.com/truffle/

[22] "Ganache", [Online]. Available: https://trufflesuite.com/ganache/

[23] "Metamask." https://metamask.io/

[24] “Axure RP 10”, [Online]. Available: https://www.axure.com/

[25] “GitLab.” https://about.gitlab.com/

[26] “astah.” https://astah.net/

[27] Wikipedia, “Front end y back end --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Front_end_y_back_end&oldid=143937851

[28] Wikipedia, “Cadena de bloques --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Cadena_de_bloques&oldid=143884981

[29] Wikipedia, “Criptografía --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Criptografía&oldid=141544304

[30] Wikipedia, “Ethereum --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Ethereum&oldid=144270746

[31] Wikipedia, “Función hash --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Función_hash&oldid=141090957

[32] Wikipedia, “Biblioteca (informática) --- Wikipedia{,} La enciclopedia libre.” 2021. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Biblioteca_(informática)&oldid=138652726

[33] Wikipedia, “Máquina de estados --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Máquina_de_estados&oldid=142633118

[34] Wikipedia, “Desarrollo ágil de software --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Desarrollo_ágil_de_software&oldid=144262212

[35] Wikipedia, “Token no fungible --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Token_no_fungible&oldid=144359456

[36] Wikipedia, “Contrato inteligente --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Contrato_inteligente&oldid=143889429

[37] Wikipedia, “Peer-to-peer --- Wikipedia{,} La enciclopedia libre.” 2022. [Online]. Available: https://es.wikipedia.org/w/index.php?title=Peer-to-peer&oldid=143754562

ÁPENDICE I. GLOSARIO

Backend

Backend es la parte del software que procesa la entrada desde el frontend [27].

Blockchain (Cadena de Bloques)

Es una etiqueta que a través de una estructura de datos cuya información se agrupa en conjuntos (bloques) a los que se le añade metainformaciones relativas a otro bloque de la cadena anterior en una línea temporal para hacer un seguimiento seguro a través de grandes cálculos criptográficos. De esta forma, gracias a técnicas criptográficas, la información contenida en un bloque solo puede ser repudiada o editada modificando todos los bloques anteriores. Esta propiedad permite su aplicación en un entorno distribuido de manera que la estructura de datos blockchain puede ejercer de base de datos pública no relacional que contenga un histórico irrefutable de información [28].

Criptografía

La criptografía se ha definido, tradicionalmente, como el ámbito de la criptología que se ocupa de las técnicas de cifrado o codificado destinadas a alterar las representaciones lingüísticas de ciertos mensajes con el fin de hacerlos ininteligibles a receptores no autorizados. Estas técnicas se utilizan tanto en el arte como en la ciencia y en la tecnología [29].

Ethereum

Ethereum es una plataforma de código abierto, que sirve para ejecutar contratos inteligentes. La plataforma tiene un alto grado de descentralización, a diferencia de otras cadenas de bloques. Es programable, lo que significa que los desarrolladores pueden usarlo en la creación de aplicaciones descentralizadas [30].

Frontend

Frontend es la parte del *software* que interactúa con los usuarios [27].

Hash

Una **función resumen** en inglés hash *function*, también conocida con el híbrido **función *hash*,** convierte uno o varios elementos de entrada a una función en otro elemento. También se las conoce como **función extracto**, del inglés *digest function*, **función de extractado** y por el híbrido **función *digest*** [31].

Librería

En informática, una **biblioteca** o, llamada por vicio del lenguaje, **librería** (del inglés *library*) es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca [32].

Máquina de estados

Se denomina **máquina de estados** a un modelo de comportamiento de un sistema con entradas y salidas en donde las salidas dependen no solo de las señales de entradas actuales, sino también de las anteriores [33].

Las máquinas de estados se definen como un conjunto de estados que sirven de intermediarios en esta relación de entradas y salidas, haciendo que el historial de señales de entrada determine, para cada instante, un estado para la máquina de forma tal que la salida depende únicamente del estado y las entradas actuales.

Metodología Ágil

El **desarrollo ágil de software** envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Así el trabajo es realizado mediante la colaboración de equipos autoorganizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo [34].

NFT (Non-Fungible Token)

Un **token no fungible** (en inglés, *non-fungible token*), o **NFT** por sus siglas en inglés, es un activo digital encriptado, un tipo especial de token criptográfico que representa algo único. Los tókenes no fungibles no son, por tanto, intercambiables de forma idéntica. Esto contrasta con criptomonedas como el bitcoin, y muchos tókenes de red o de utilidad que son fungibles por naturaleza. Las cuatro principales características de los NFT es que son únicos, indivisibles, transferibles y con la capacidad de demostrar su escasez [35].

Smart contract

Un **contrato inteligente** (en inglés *smart contract*) es un programa informático que facilita, asegura, hace cumplir y ejecuta acuerdos registrados entre dos o más partes (por ejemplo, personas u organizaciones). Como tales ellos les ayudarían en la negociación y definición de tales acuerdos que causarán que ciertas acciones sucedan como resultado de que se cumplan una serie de condiciones específicas [36].

Sprint

Se define como cada uno de los requisitos que constituyen un proyecto que utiliza metodología ágil.

P2P (Peer to Peer)

Una **red *peer-to-peer***, **red de pares**, **red entre iguales** o **red entre pares** (***P2P***, por sus siglas en inglés) es una red de ordenadores en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es más, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados [37].

Apéndice II Manual de Instalación

En este apartado se explicará los pasos a seguir para poder ejecutar la aplicación.

Descarga del proyecto

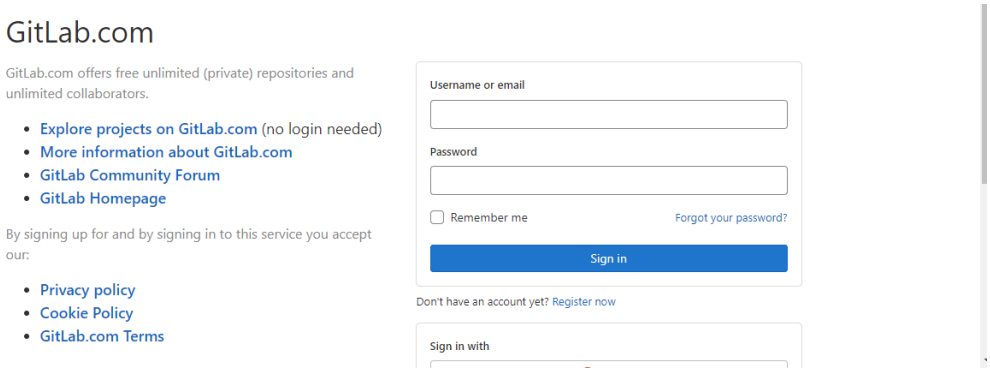
Lo primero es descargar el código del proyecto subido en la plataforma de GitLab. Existen dos formas de realizar este primer paso:

La primera es tener una cuenta en GitLab y acceder directamente al siguiente enlace: https://gitlab.com/HP-SCDS/Observatorio/2021-2022/cryptocards/uva-cryptocards .

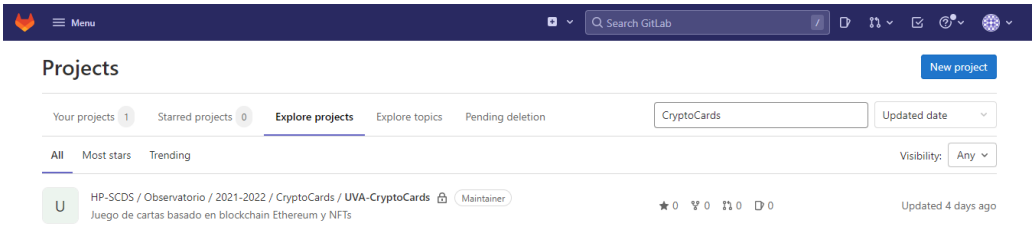
La segunda no requiere que inicies sesión y costa de los siguientes pasos:

1. Acceder a la página de inicio de sesión de GitLab: https://gitlab.com/users/sign_in

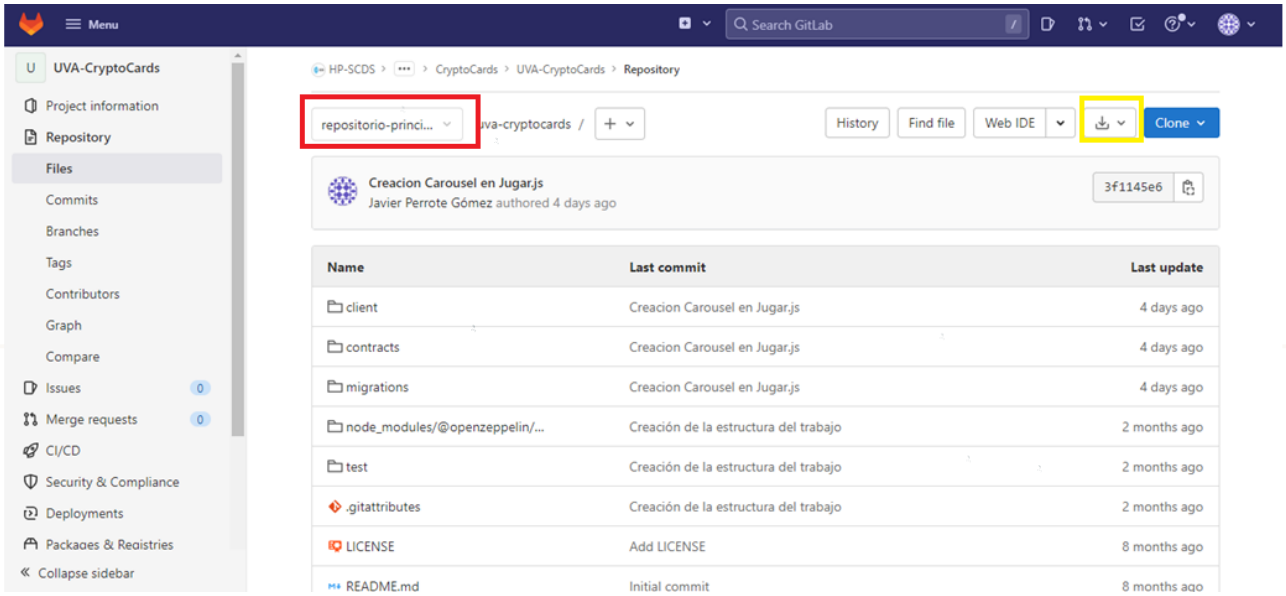
2. Pulsar sobre Explore proyects on GitLab.com.



3. Pulsar sobre All para ver todos los proyectos, e introducir "*CryptoCards"* en el buscador, para mostrar el proyecto en los resultados.



4. Una vez en el proyecto seleccionar la rama repositorio principal (rojo) y pulsar sobre el botón de descarga (amarillo).



5. Guardar y descomprimir el archivo .zip generado

Una vez descargado el código del proyecto se procede a realizar la instalación de diferentes programas necesarios para poder ejecutarlo y probarlo.

Instalación de NVM

Este proceso se compone de los siguientes pasos:

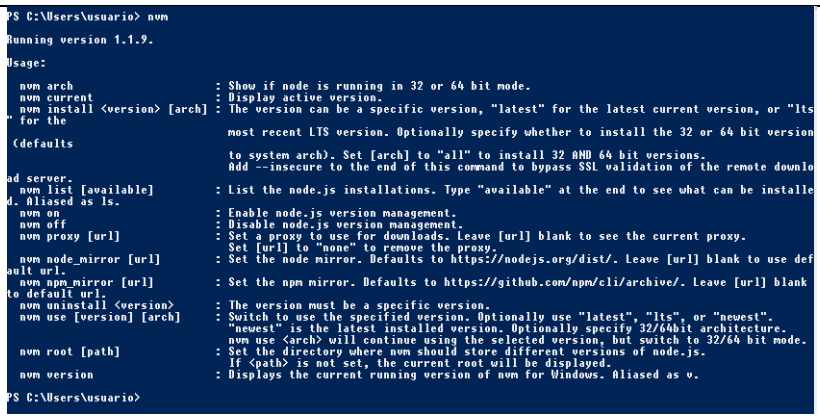
1. Acceder al enlace https://github.com/nvm-sh/nvm#install-script.

2. Instalar NVM siguiendo las instrucciones según el sistema operativo con que se trabaje. En el caso de Windows de nuevo existe un instalador.

3. Comprobar que se ha instalado de manera correcta. Para esto, abrir un terminal y ejecutar el comando:

nvm

Salida:



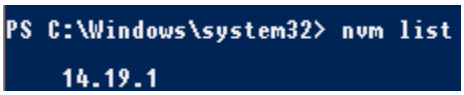
4. Instalar una versión de Node.js utilizando NVM, esto a su vez instala una versión de NPM compatible. Para este proyecto se ha utilizado la versión **14.19.1 de Node.js.** Para ello, es necesario abrir un terminal como administrador y ejecutar el comando:

nvm install 14.19.1

5. Comprobar que se ha instalado correctamente ejecutando el comando:

nvm list

Salida:



6. Hay que especificar que se quiere usar esa versión con el comando

nvm use 14.19.1

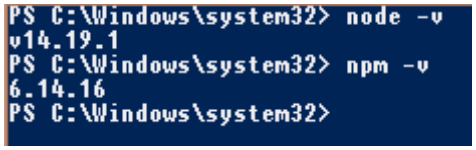
Salida:

7. Comprobar las versiones de Node.js y NPM instaladas ejecutando los comandos:

node –v

npm -v

Salida:



Se han instalado las versiones 14.19.1 de Node.js y la 6.14.16 de NPM. Ahora puedes utilizar el comando npm para instalar el resto de las librerías.

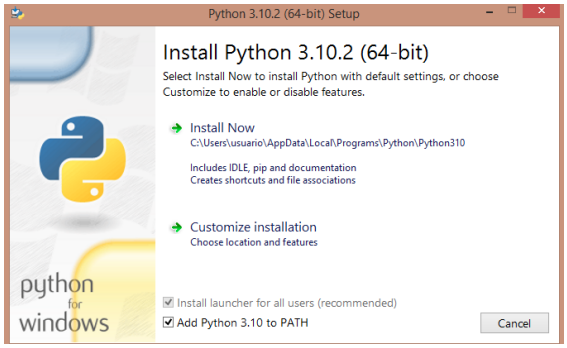
Instalación de Python

Es necesario instalar previamente Python, para poder descargar Truffle. Para este proyecto se ha descargado la versión **3.10.2**.

1. Acceder al enlace: https://www.python.org/downloads/release/python-3102/

2. Realizar la instalación según el sistema operativo utilizado.

3. En el caso de Windows. Seleccionar la opción Add Python 3.10 to PATH en la ventana del instalador.



4. Comprobar que la instalación se ha hecho correctamente con el comando:

python –version

Salida:

```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. Todos los derechos reservados.

PS C:\Windows\system32> python --version
Python 3.10.2
PS C:\Windows\system32>
```

Instalación de Truffle

Ahora si se puede proceder a la instalación de Truffle, para poder desplegar los contratos en la *blockchain*.

1. Instalar Truffle con el comando:

npm install -g truffle

Salida:

```
PS C:\Users\Propietario> npm install -g truffle
```

2. Esta instalación puede tomar un tiempo y puede producir un error. Si se produce un error ejecuta el siguiente comando:

Set – ExecutionPolicy – Scope CurrentUser – ExecutionPolicy Bypass -Force

3. Vuelve a probar instalar Truffle.

Es importante instalar las versiones que se especifican, ya que Truffle tiene incompatibilidades con muchas de ellas. También, se pueden producir otros muchos errores, para lo que habría que instalar diferentes versiones de Node.js y NPM, hasta encontrar una que no produzca error en la instalación de Truffle.

4. Comprobar que se ha instalado correctamente.

Salida:

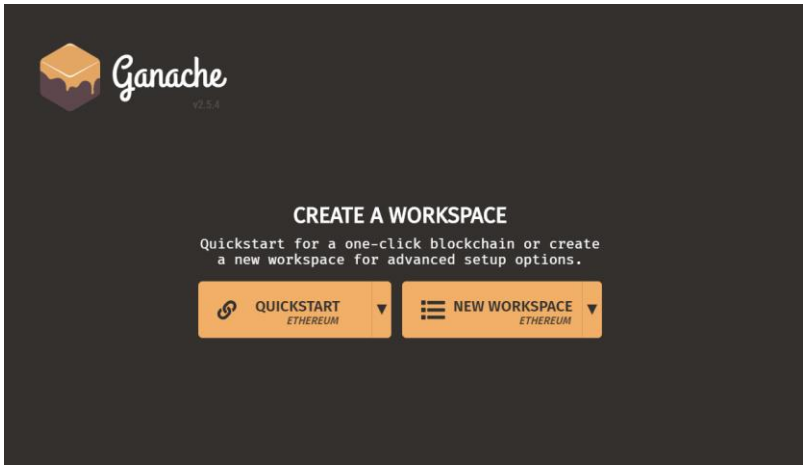
```
PS C:\Users\Propietario> truffle
Truffle v5.5.6 - a development framework for Ethereum

Usage: truffle <command> [options]

Commands:
  build       Execute build pipeline (if configuration present)
  compile     Compile contract source files
  config      Set user-level configuration options
  console     Run a console with contract abstractions and commands available
  create      Helper to create new contracts, migrations and tests
  dashboard   Start Truffle Dashboard to sign development transactions using
              browser wallet
  db          Database interface commands
  debug       Interactively debug any transaction on the blockchain
  deploy      (alias for migrate)
  develop     Open a console with a local development blockchain
  exec        Execute a JS module within this Truffle environment
  help        List all commands or provide information about a specific command
  init        Initialize new and empty Ethereum project
  install     Install a package from the Ethereum Package Registry
  migrate     Run migrations to deploy contracts
  networks    Show addresses for deployed contracts on each network
  obtain      Fetch and cache a specified compiler
  opcode      Print the compiled opcodes for a given contract
  preserve    Save data to decentralized storage platforms like IPFS and Filecoin
  publish     Publish a package to the Ethereum Package Registry
  run         Run a third-party command
  test        Run JavaScript and Solidity tests
  unbox       Download a Truffle Box, a pre-built Truffle project
  version     Show version number and exit
  watch       Watch filesystem for changes and rebuild the project automatically
```

Instalación y configuración de Ganache

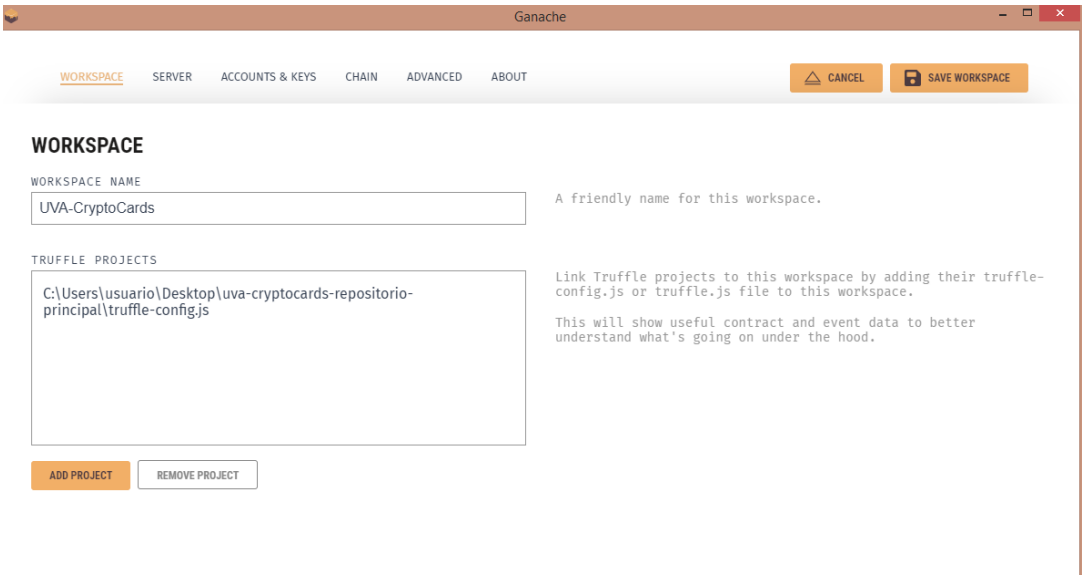
Lo siguiente consiste en instalar Ganache y crear la *blockchain* local.

1. Instalar ganache desde el enlace: https://github.com/trufflesuite/ganache-ui/releases.

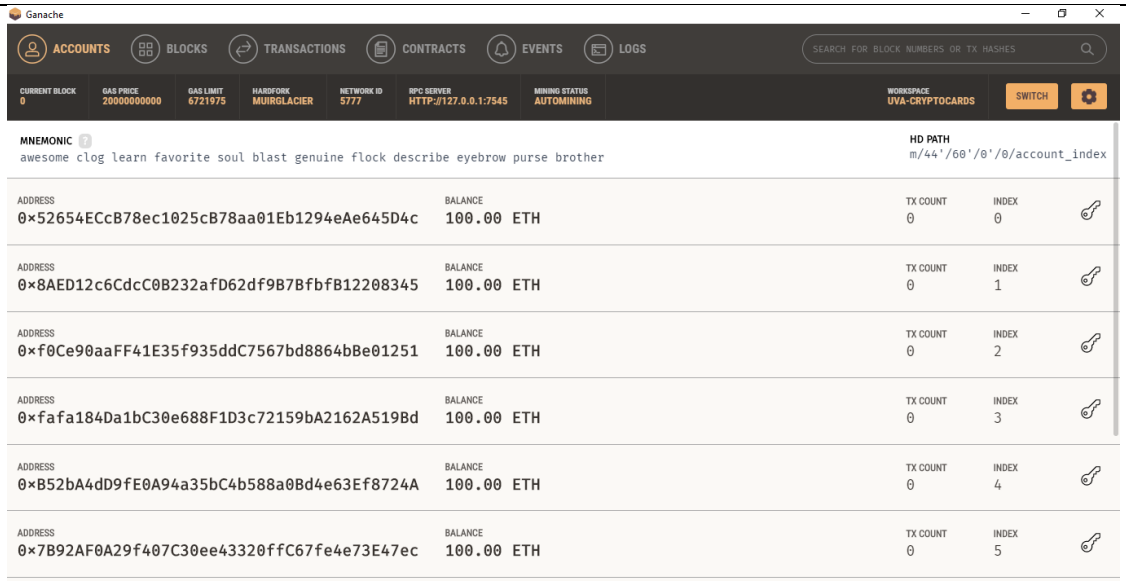
2. Crear una nueva red. Para esto pulsar sobre el botón NEW WORKSPACE.



3. Introducir el nombre del proyecto y añadir el archivo truffle-config.js del proyecto pulsando sobre el botón ADD PROYECT.



4. Pulsar sobre el botón SAVE WORKSPACE para guardar los cambios. Con esto se genera la red de Ethereum y se crean 10 cuentas distintas cada una con su propia clave privada y 100 ETH ficticios para poder utilizar.



Despliegue de los contratos en la red

El siguiente paso consiste en desplegar los contratos en la red.

1. Abrir un terminal.

2. Cambiar al directorio del proyecto.

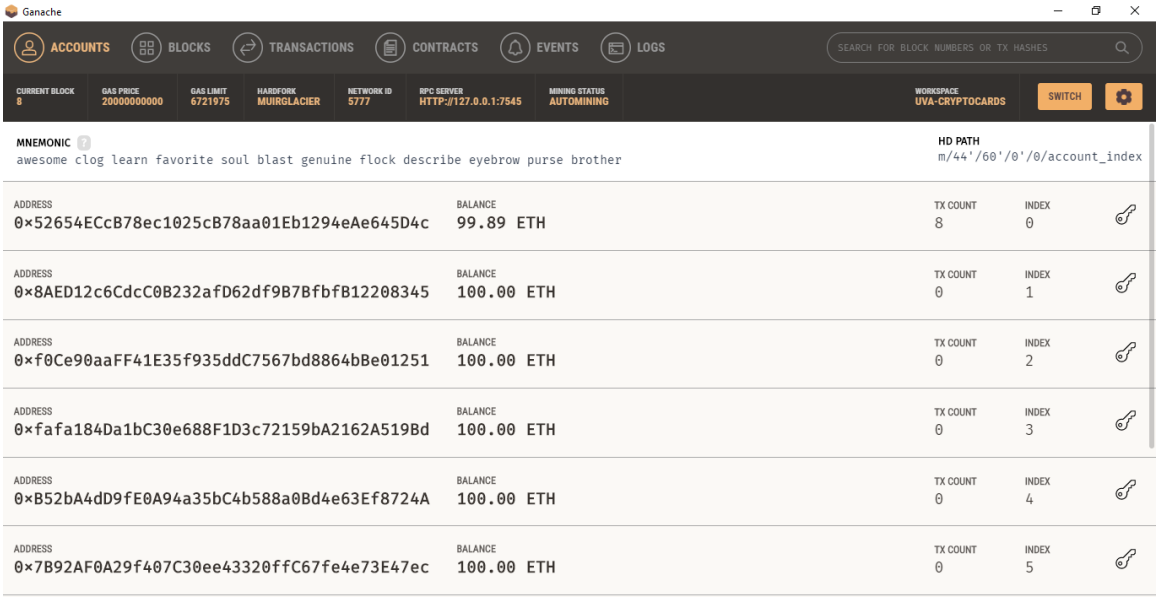
3. Comprobar que Ganache está funcionando y desplegar los contratos en la red con el comando:

truffle migrate

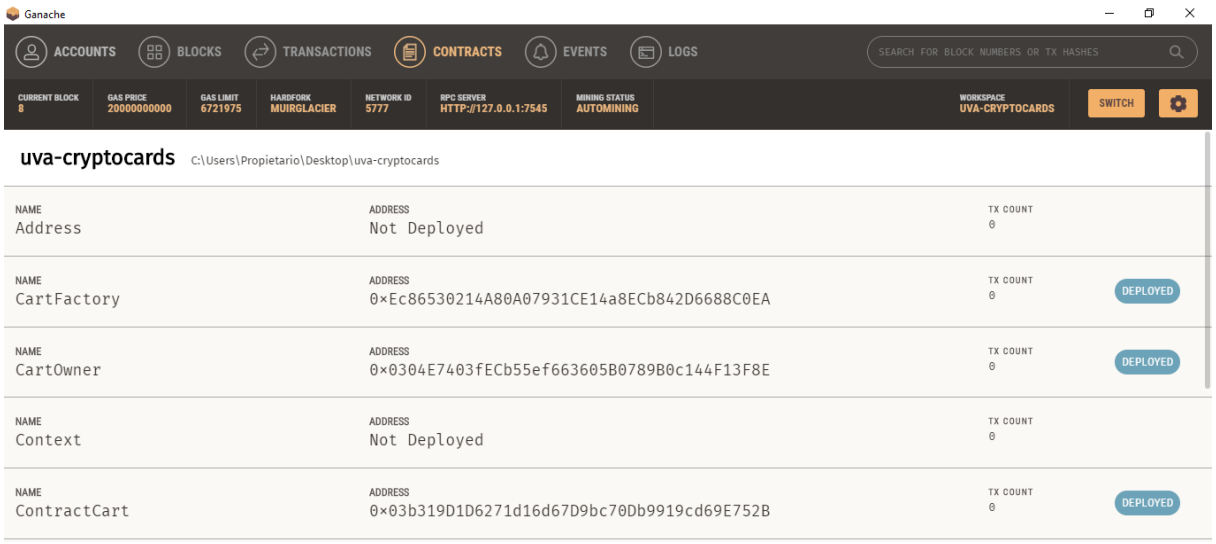
Salida:

```
6_ContractCart.js
=================

   Replacing 'ContractCart'
   ------------------------
   > transaction hash:    0xd2e938ede5eb37f9a0261e3a4f3006efd3c723997fe738284b883dd0a25d6a01
   > Blocks: 0            Seconds: 0
   > contract address:    0x03b319D1D6271d16d67D9bc70Db9919cd69E752B
   > block number:        7
   > block timestamp:     1657305936
   > account:             0x52654ECcB78ec1025cB78aa01Eb1294eAe645D4c
   > balance:             99.89537422
   > gas used:            248162 (0x3c962)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00496324 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00496324 ETH

Summary
=======
> Total deployments:   4
> Final cost:          0.102675 ETH
```

Desplegar los contratos tiene un gas asociado que se descuenta de la primera cuenta de Ganache.



Ahora los contratos ya están almacenados en la blockchain.

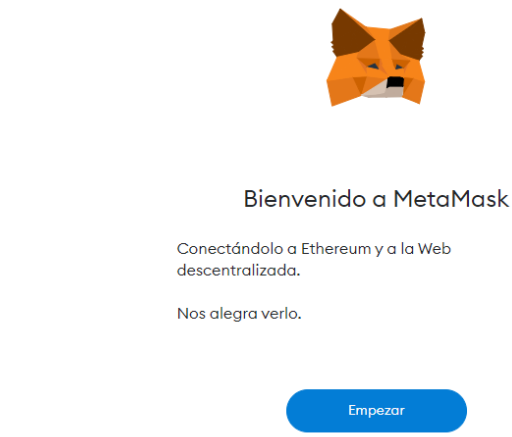


Instalación y configuración de Metamask

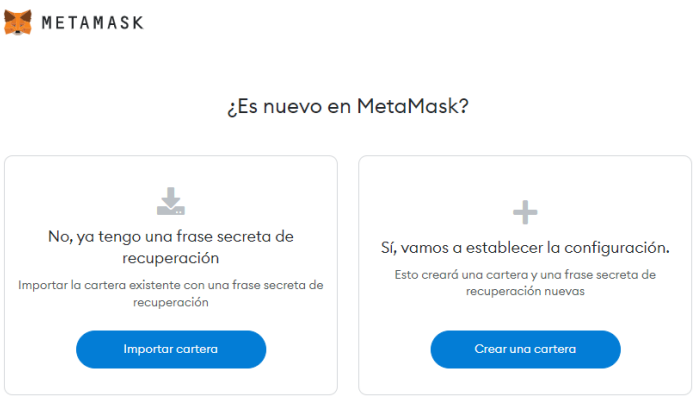
Una vez has desplegado los contratos lo siguiente es configurar la cartera Metamask.

1. Añadir la extensión Metamask de nuestro navegador.

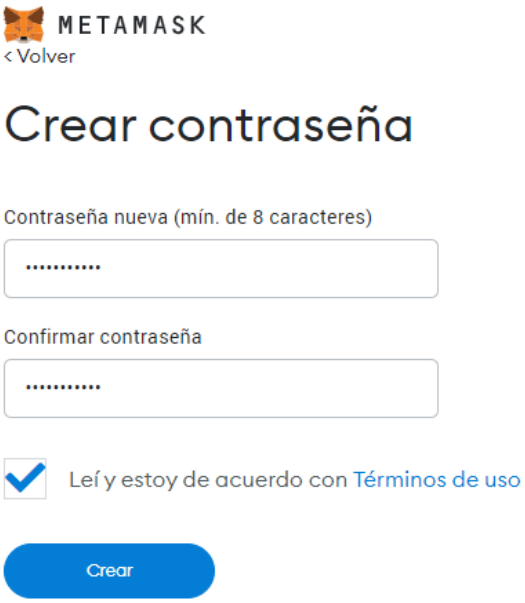
2. Pulsar sobre Empezar.



3. Pulsar sobre Crear una Cartera y en la siguiente ventana hacer clic sobre “Acepto”.

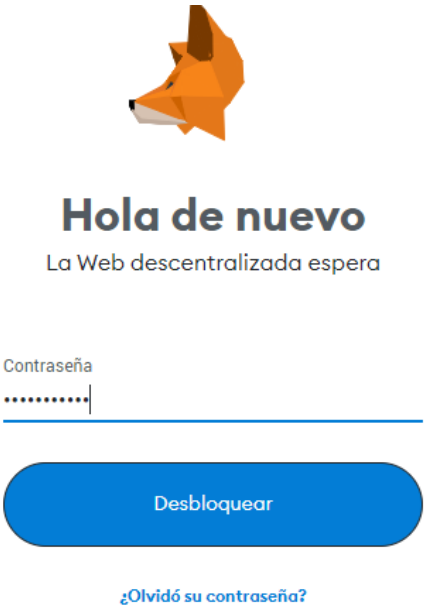


4. Introducir una contraseña y aceptar los términos.



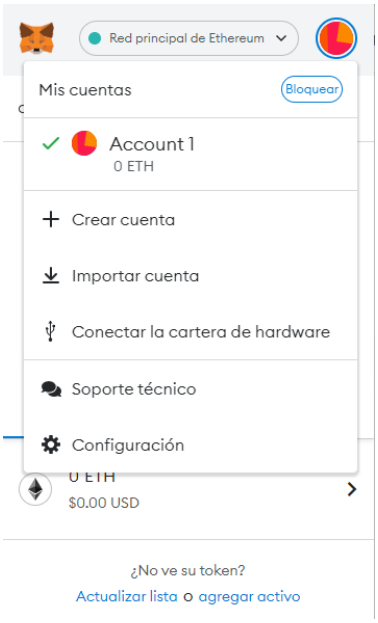
5. Fijar la extensión en el navegador para facilitar el uso. (Recomendable).

6. Pulsar sobre el icono y acceder a Metamask introduciendo la contraseña.

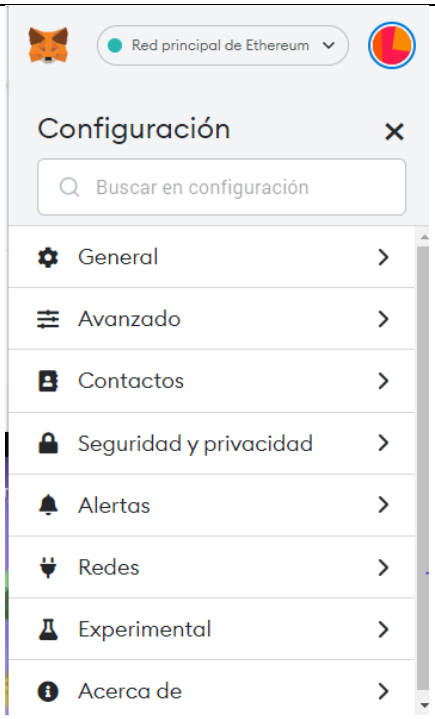


7. En las siguientes ventanas pulsar sobre “Siguiente” y sobre “Recordármelo más tarde”.

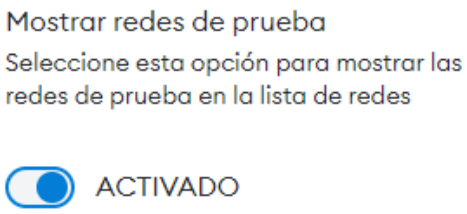
8. Pulsar sobre “Configuración”.



9. Pulsar sobre Avanzado.

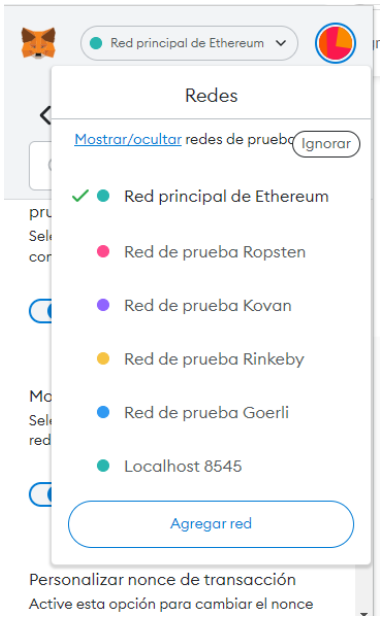


10. Activar Mostrar redes de Prueba



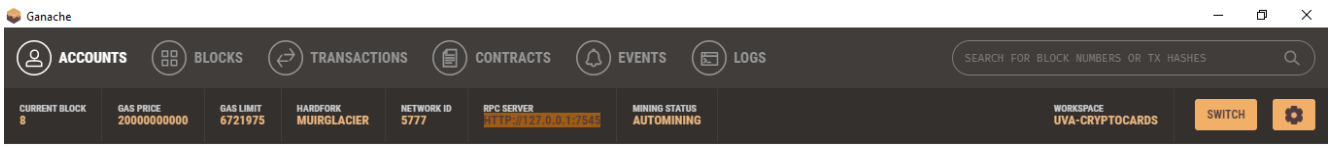
Ahora te aparecen todas las redes de Prueba y puedes agregar la nueva red.

11. Pulsar sobre Agregar red



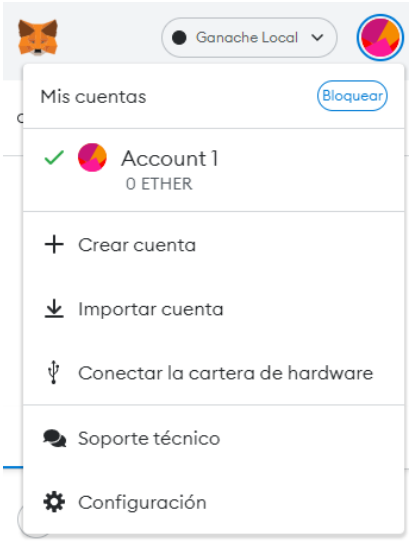
12. Introducir la información de la red y pulsar sobre el botón Guardar.

El campo Nueva dirección URL se obtiene a través de Ganache.

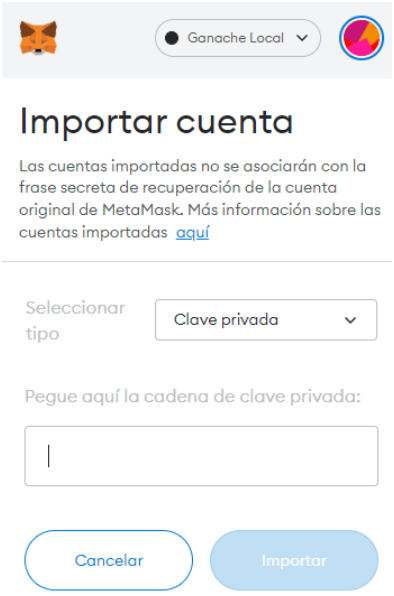


Una vez hecho esto ya tienes la red de Ganache conectada con Metamask y puedes añadir alguna de las cuentas que esta proporciona.

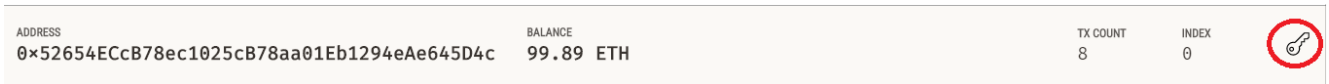
13. Pulsar sobre Importar Cuenta

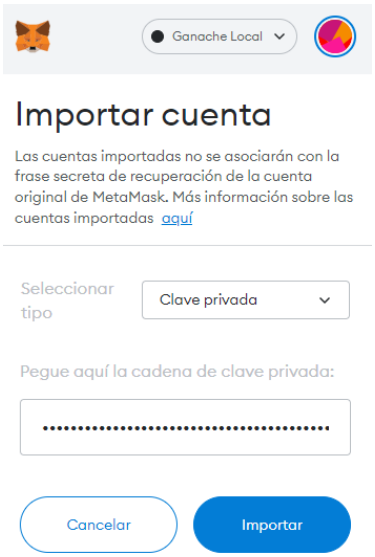


14. Introducir la clave privada y pulsar sobre Importar.

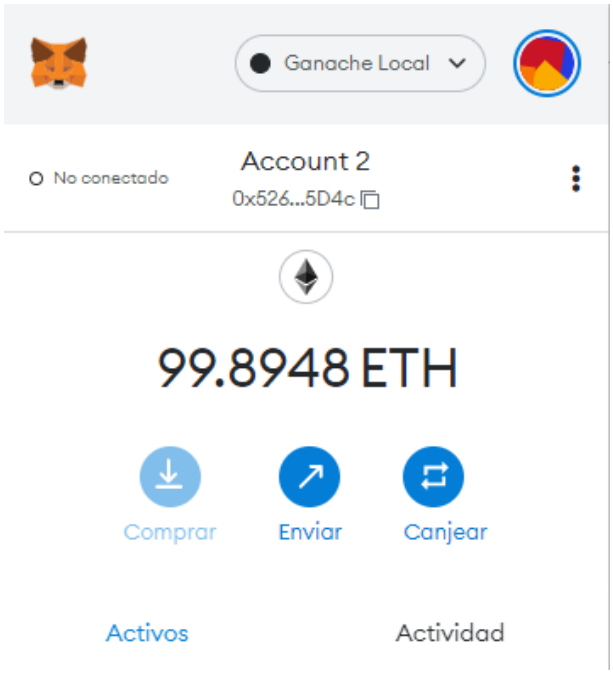


Esta se obtiene pulsando sobre la llave de cualquiera de las cuentas de Ganache.





15. Comprobar que aparece la cuenta con su saldo.



Conectar cuenta y ejecución del programa

Ya se tiene una cuenta que se puede utilizar para el proyecto, pero aún no está conectada a la aplicación para ello primero se debe ejecutar el programa.

1. Descargar las dependencias de package.js del proyecto. Para esto ir a la carpeta

Para ejecutar el programa primero debes descargar las dependencias del package js para ello ejecutas
Te vas a la carpeta client del proyecto y ejecutar el comando:

npm install

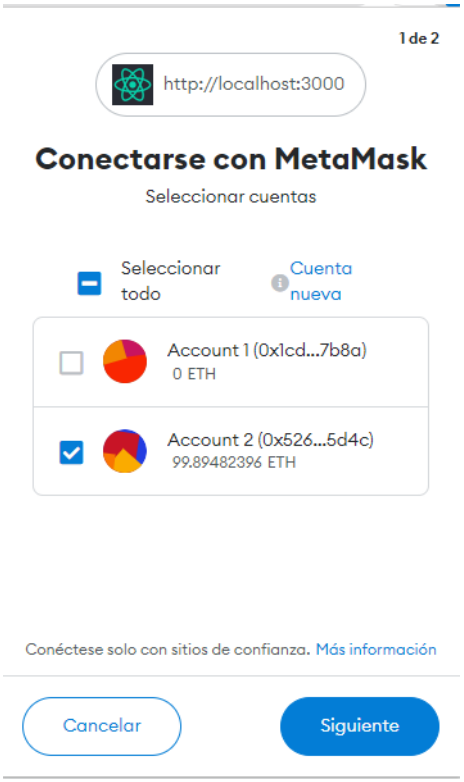
Este paso de instalar las dependencias puede tomar un poco de tiempo.

2. En la misma carpeta ejecutar el comando:

npm run start

Pasado un tiempo se abrirá una ventana en el navegador.

3. Es necesario conecta la cuenta a la aplicación, para ello hay que seleccionar la cuenta que se quiere conectar y pulsar sobre Siguiente. Después pulsar sobre Conectar.



Ahora la cuenta ya está conectada a la aplicación vía Metamask y puede hacer llamadas a los contratos utilizando la interfaz.

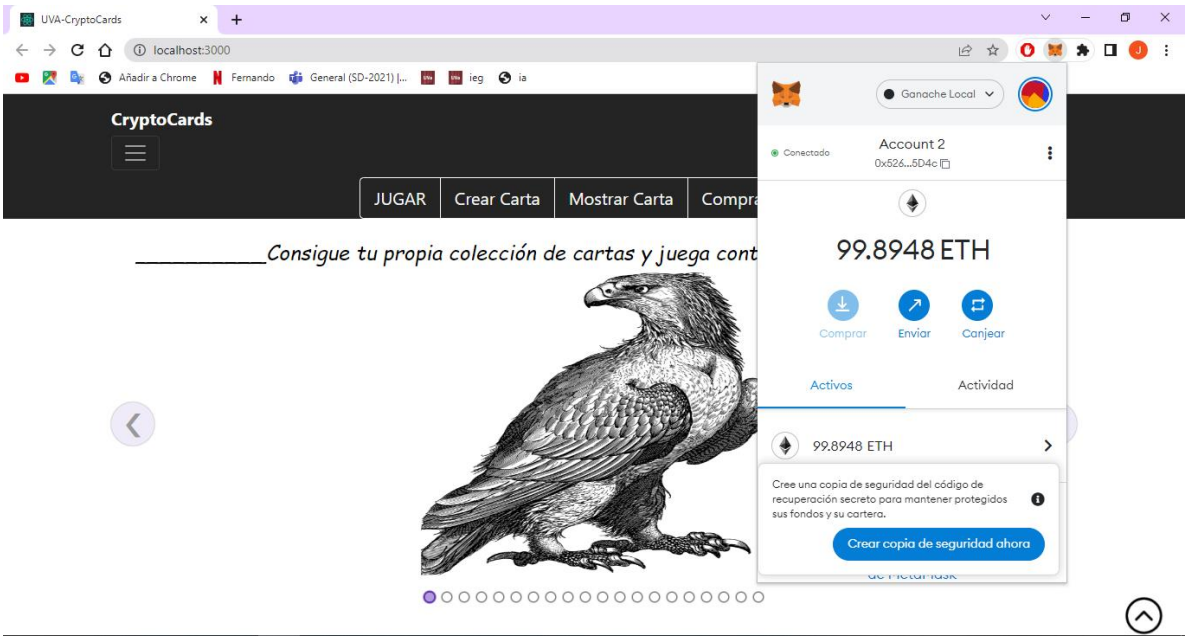
Finalmente se puede agregar más cuentas siguiendo los pasos previos para simular que distintos usuarios utilizan la aplicación.

Apéndice III

Guía de Usuario

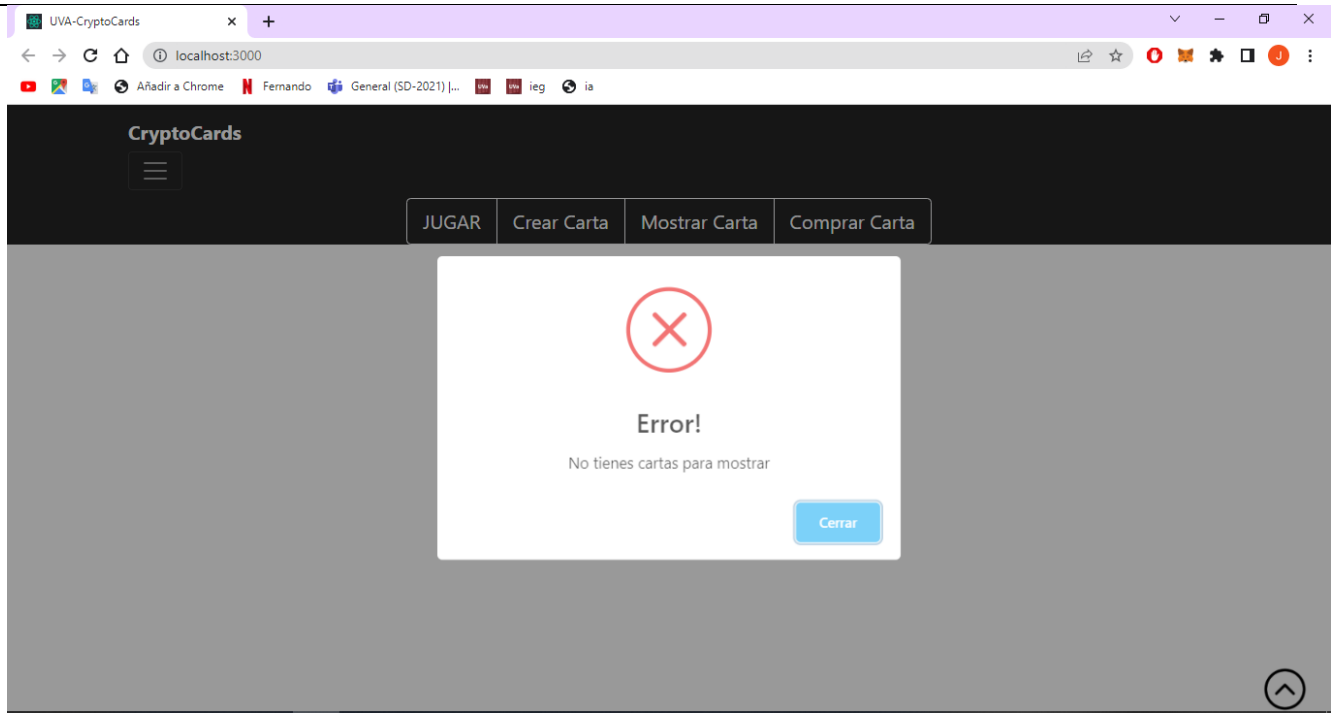
Se simula el comportamiento que tendría la aplicación si accedieran distintos usuarios agregando distintas cuentas a Metamask de las que proporciona Ganache.

Según se inicia la aplicación se muestra la página de inicio y se puede ver si la cuenta esta conectada a la aplicación.

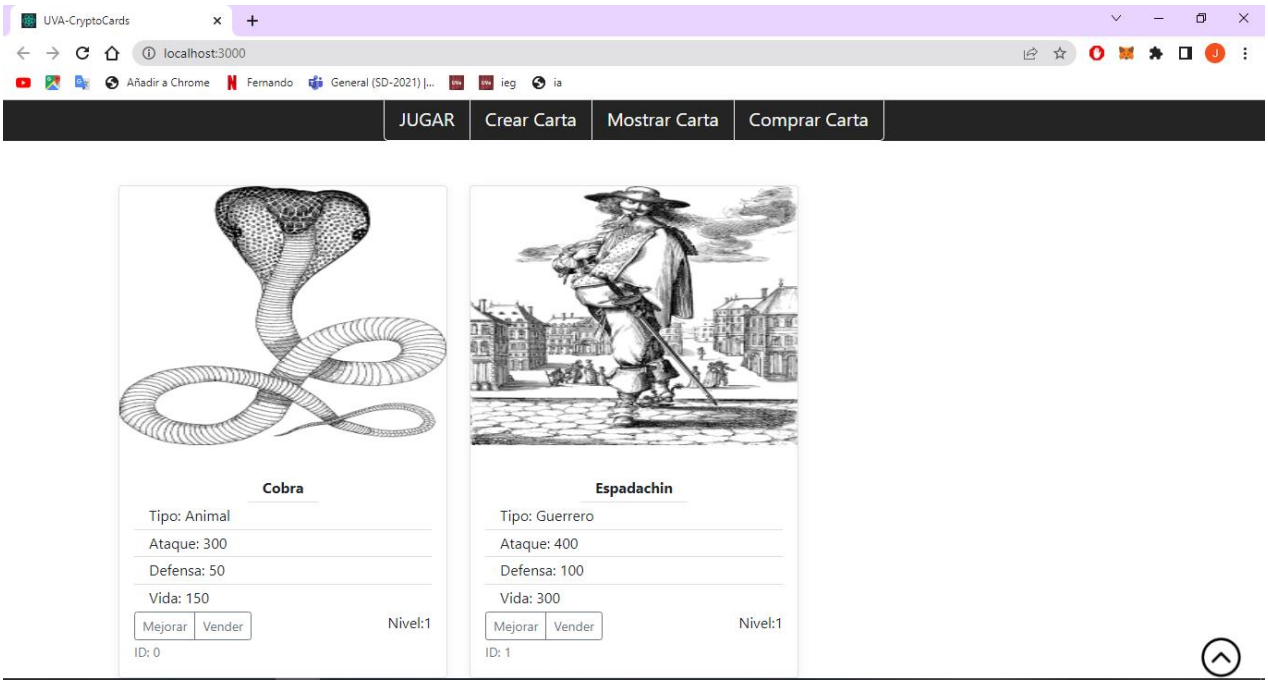


Mostrar Colección Cartas

Para mostrar la colección de cartas de un usuario es necesario tener previamente alguna carta, si este no es el caso saltara una alerta indicando que no existen cartas.



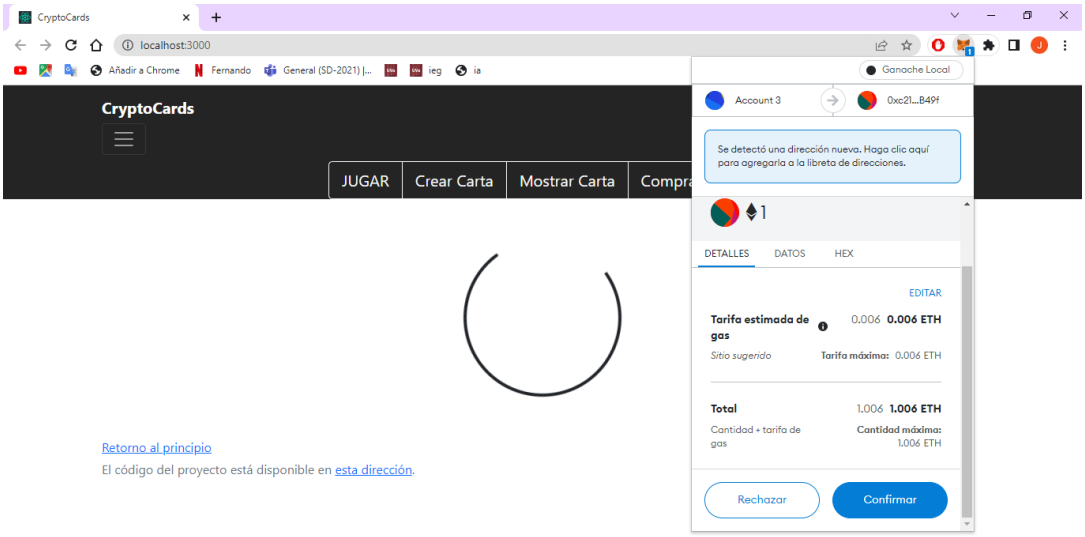
En el caso de que el usuario sí que disponga de cartas estas se muestran en forma de álbum.



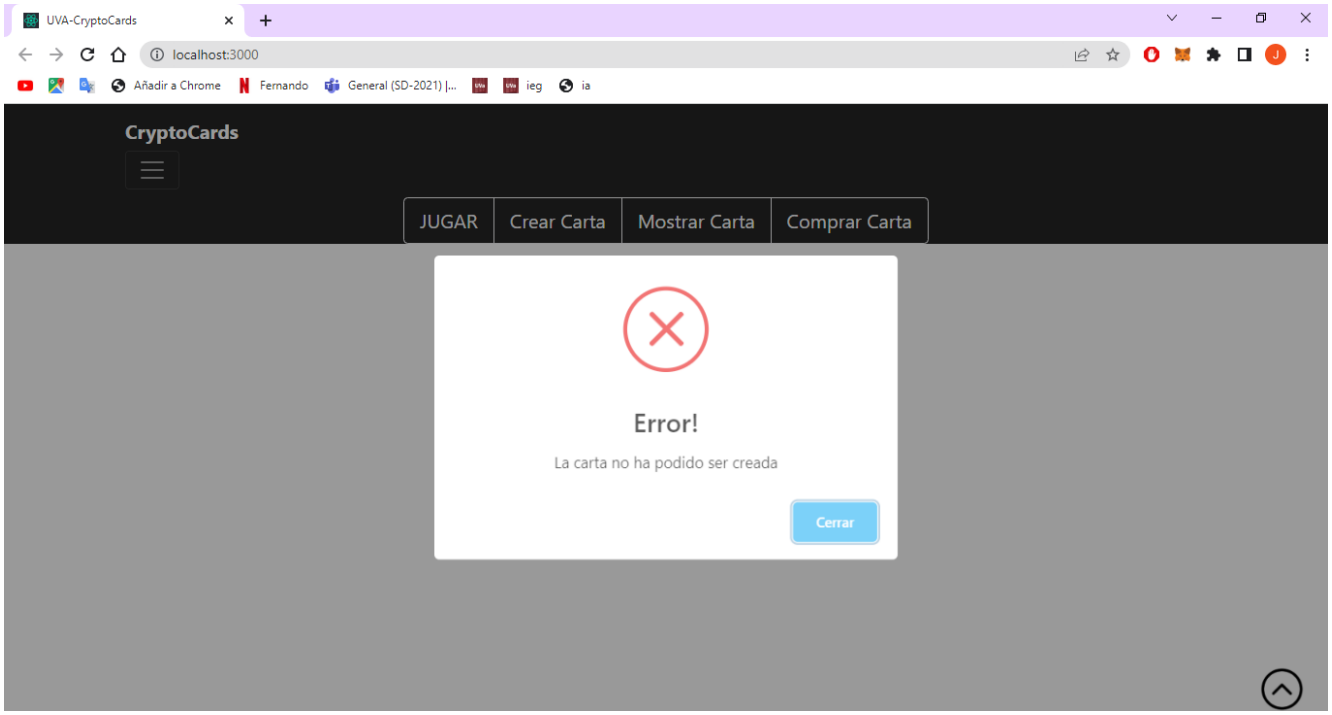
Crear Carta

Para crear una nueva carta hay que presionar sobre el botón Crear Carta.

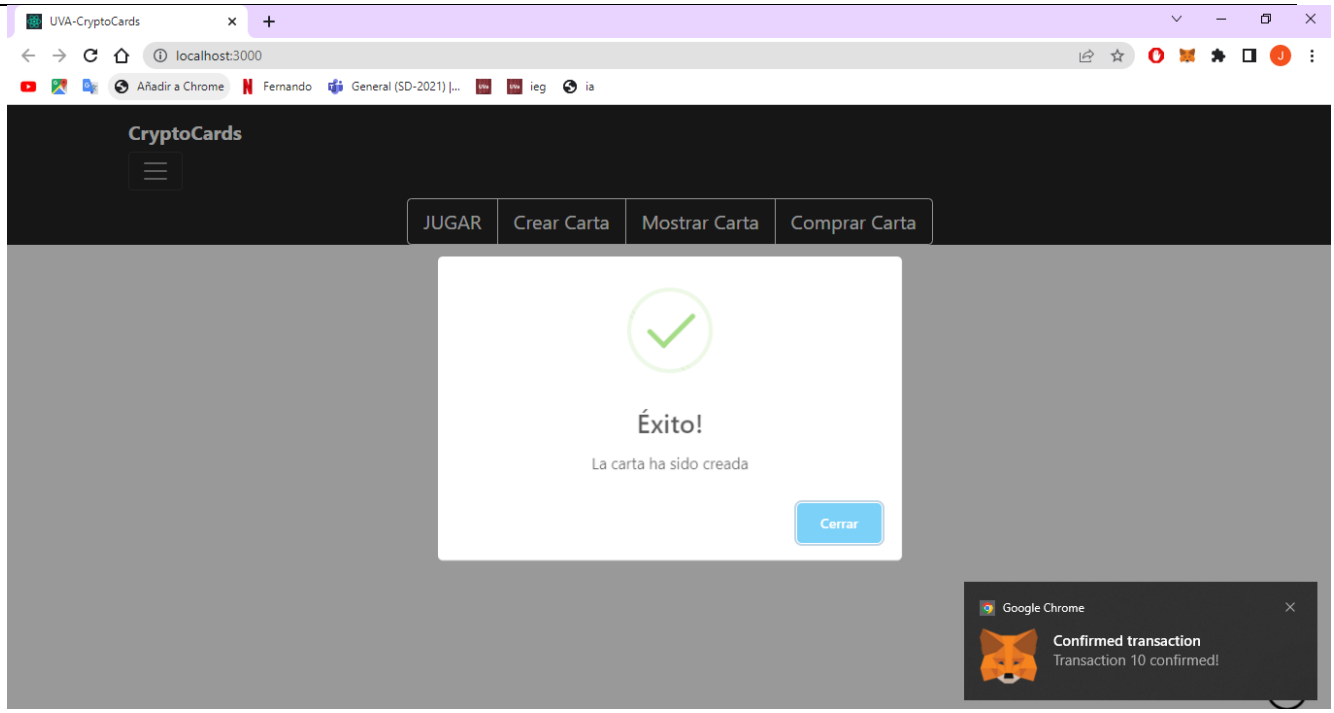
Tras esto saltara una notificación de Metamask para confirmar la transacción, indicando el coste en ETH que esta supone. El precio por crear una carta es de 1 ETH, pero a esto hay que sumarle el gas lo que el coste final es algo mayor.



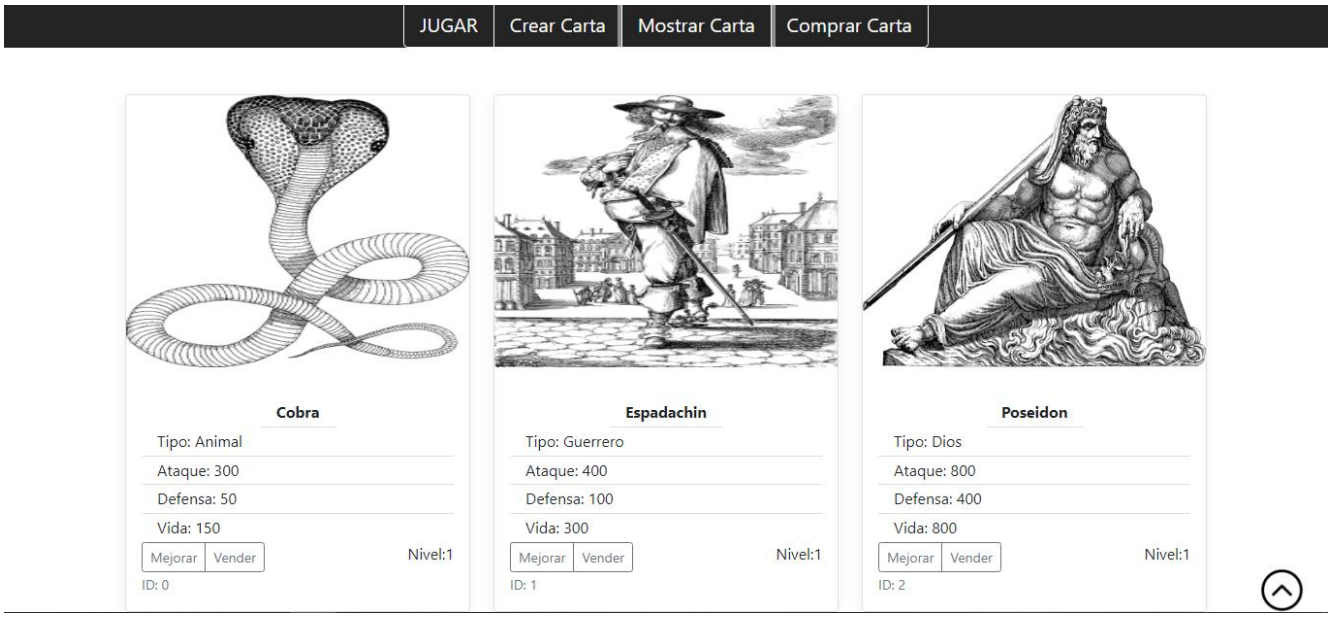
Si durante la transacción se produce un error o simplemente se rechaza, se muestra un mensaje indicando que la carta no ha podido ser creada.



En caso contrario se indica que la carta ha sido creada de manera exitosa.



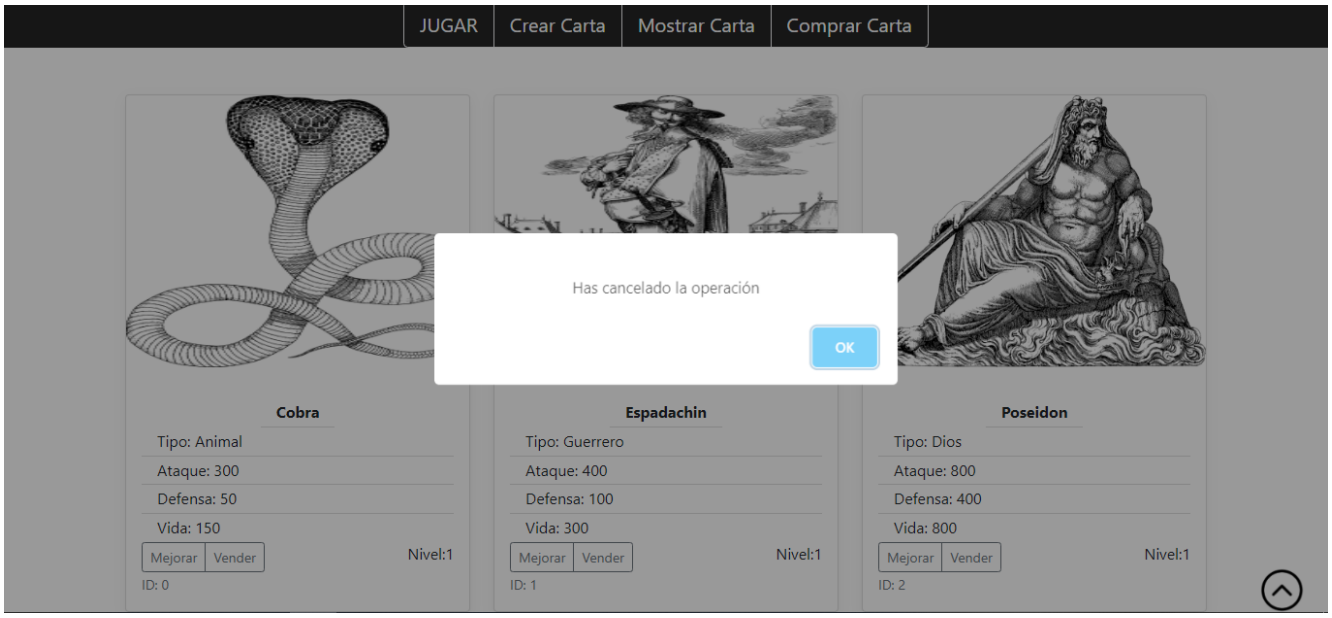
Ahora una nueva carta ha sido añadida a la colección.



Mejorar Carta

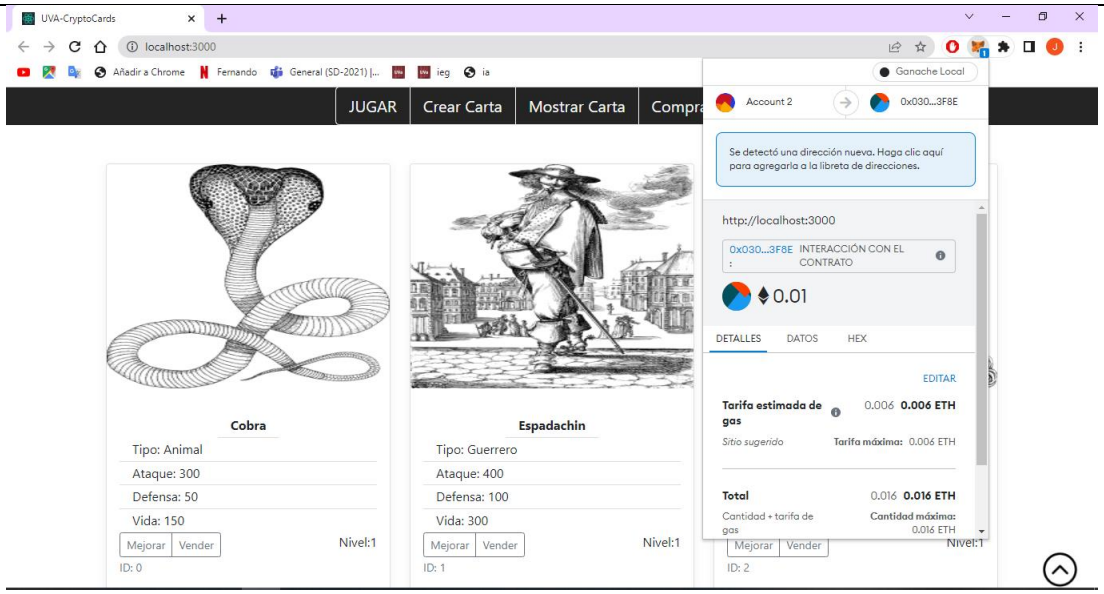
Con la colección de cartas a la vista seleccionas la carta que deseas mejorar. En este caso la cobra.

En caso de que se quiera cancelar la operación:

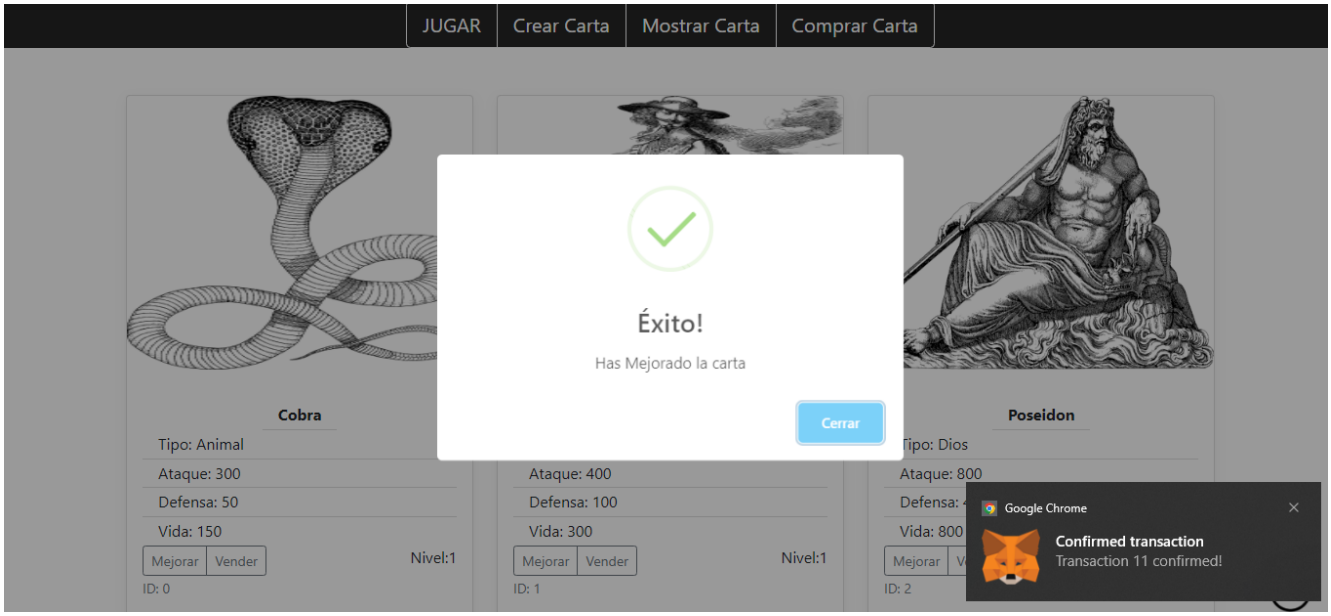


En caso contrario, el usuario decide mejorar el atributo de ataque de la carta.

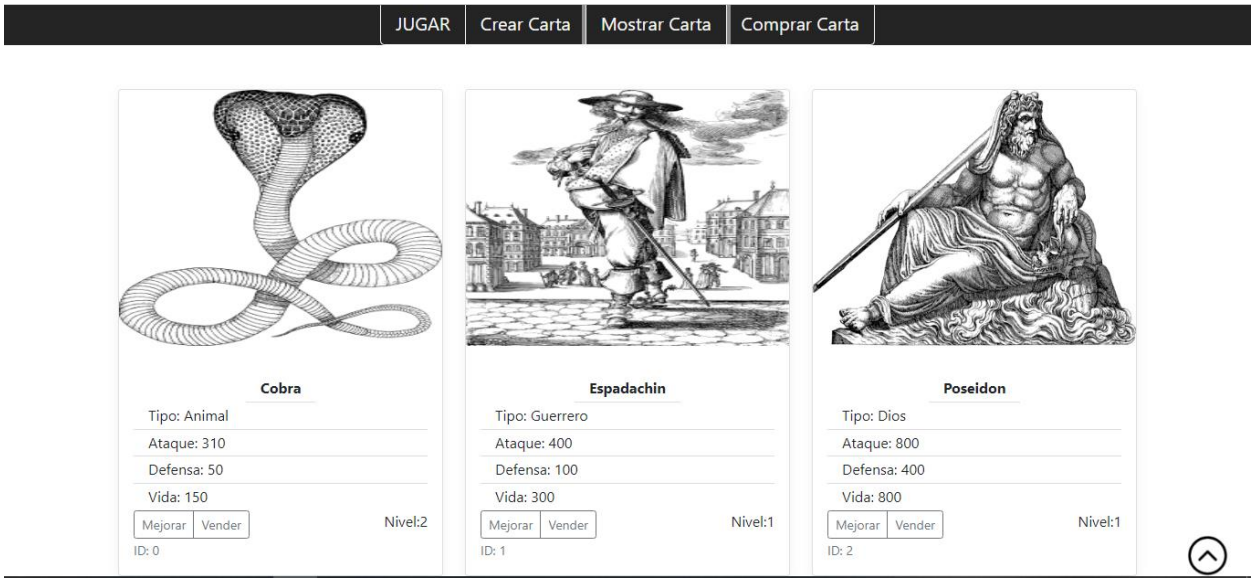
Salta una notificación de Metamask indicando el coste de la operación, coste de mejorar la carta establecido en 0.01 ETH + gas.



Si la transacción se completa de manera exitosa, se informa a través de un mensaje.

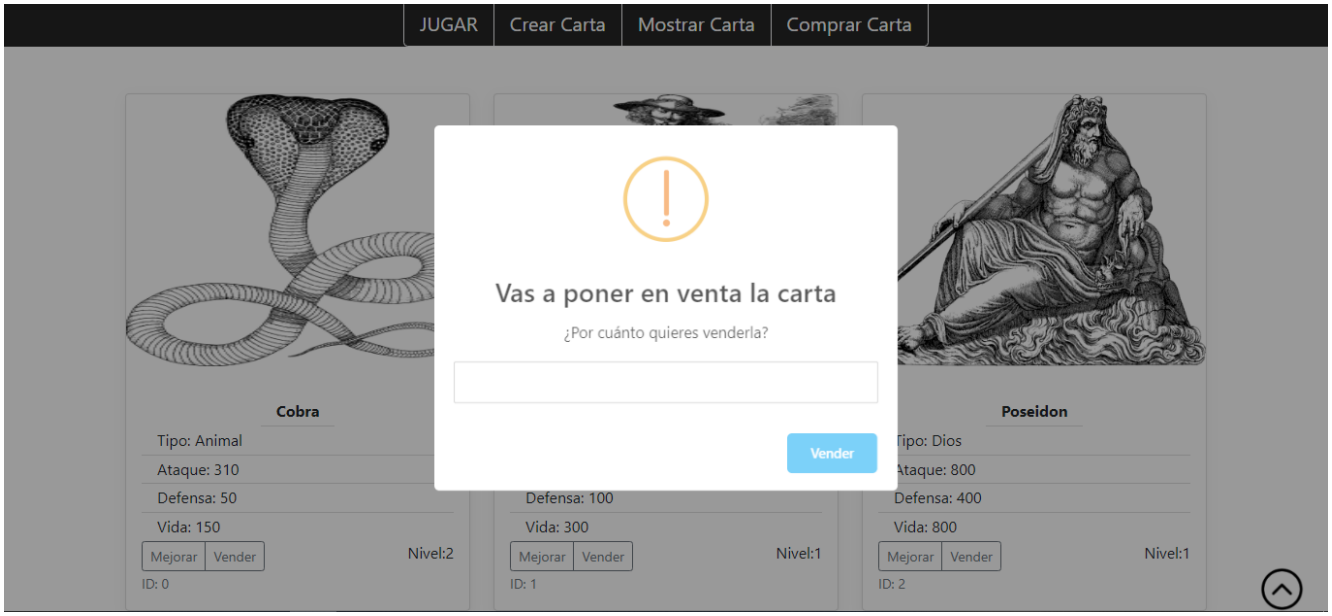


Si vuelves a recargar la aplicación o cambiar de vista se puede visualizar como el atributo ha mejorado.

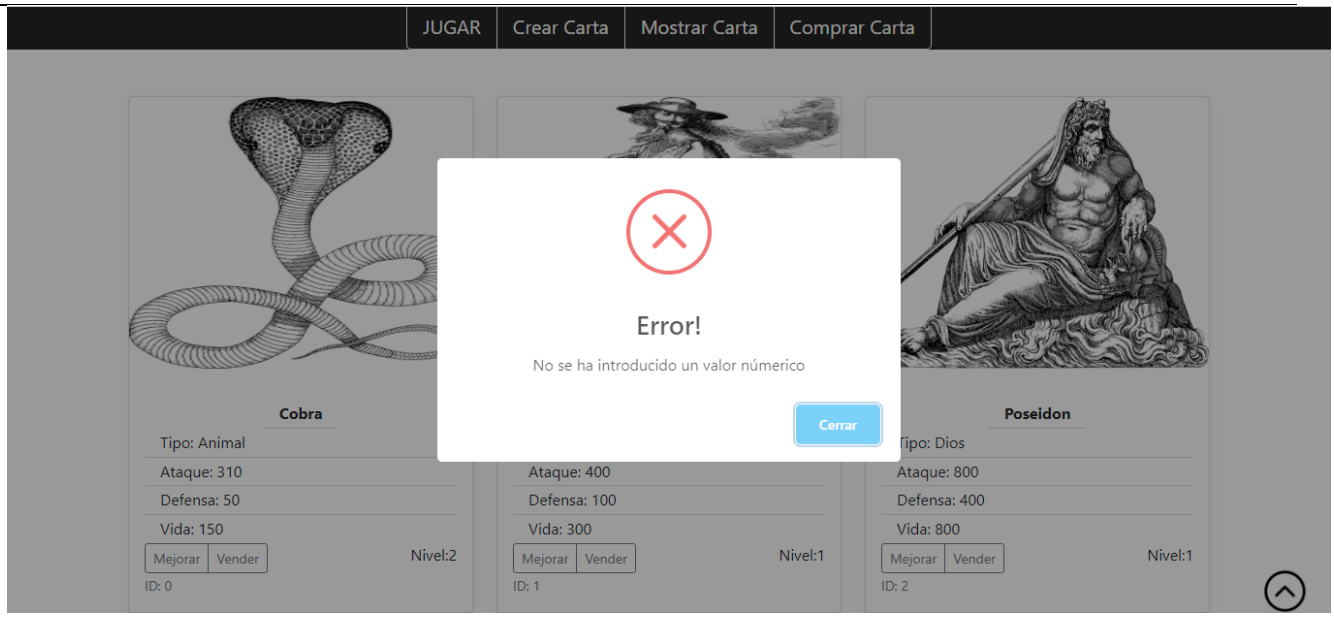


Vender Carta

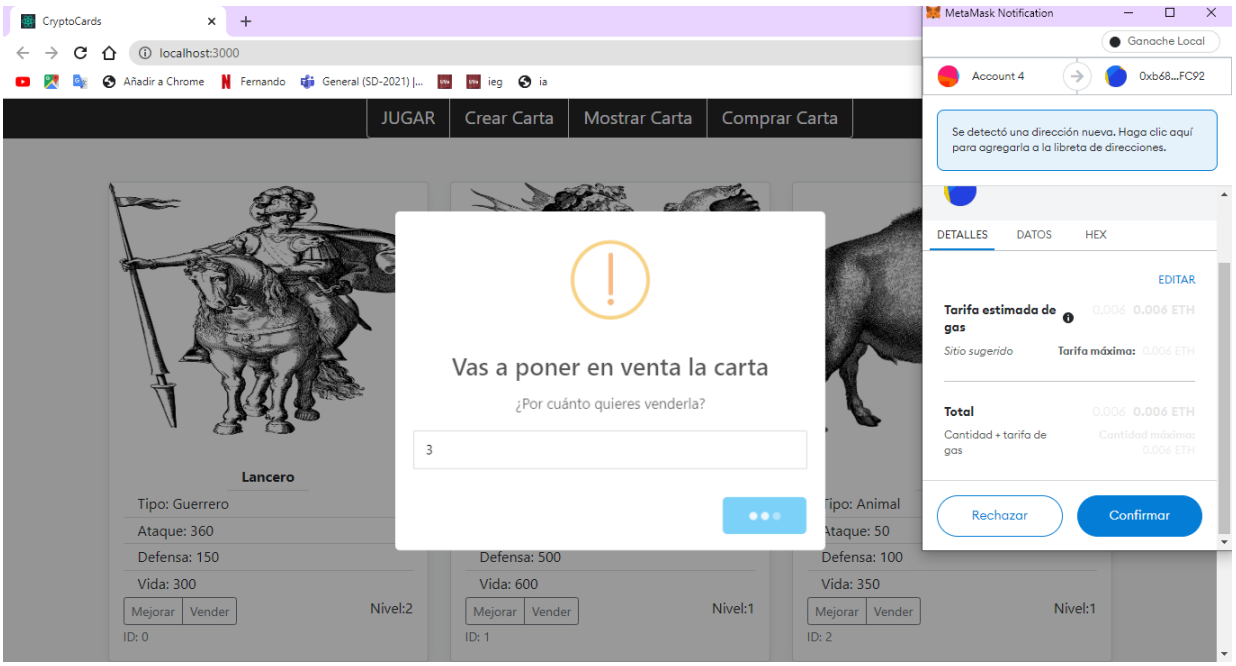
Pon que ahora quieres vender la carta de Poseidon, lo primero es pulsar sobre Vender.



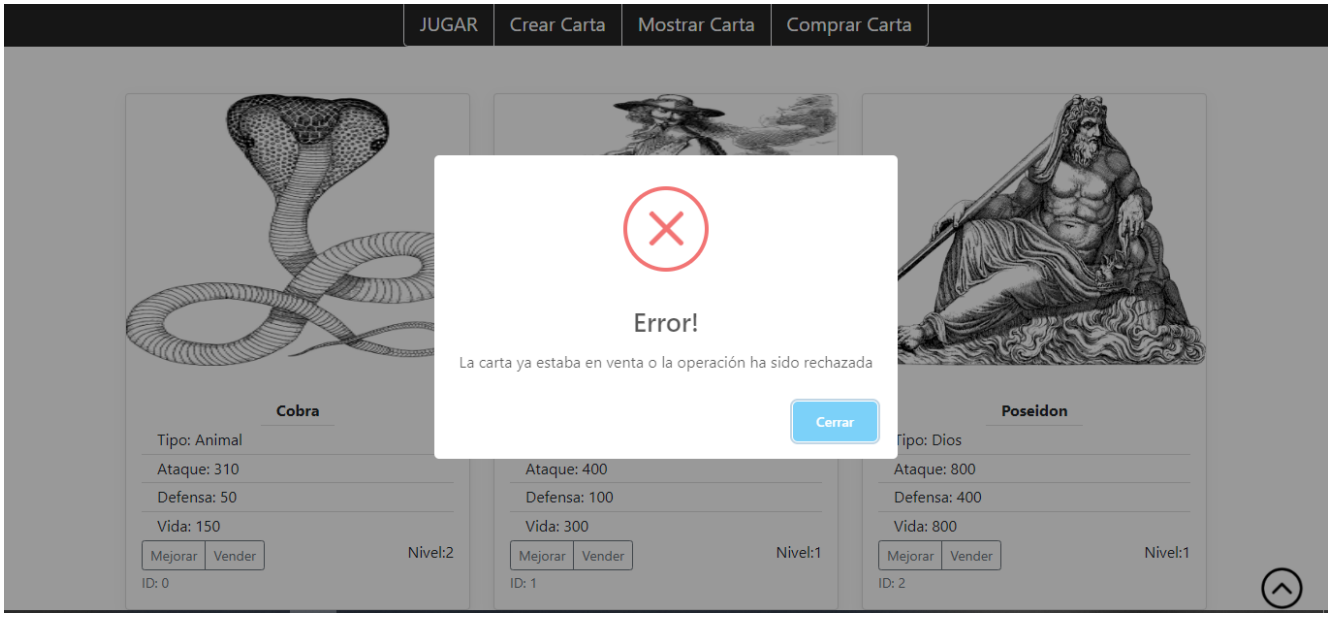
En el campo de texto se debe introducir un valor numérico para indicar el precio al que quieres poner la carta en venta. En caso de que no se introduzca un número se avisa mediante un mensaje de error.



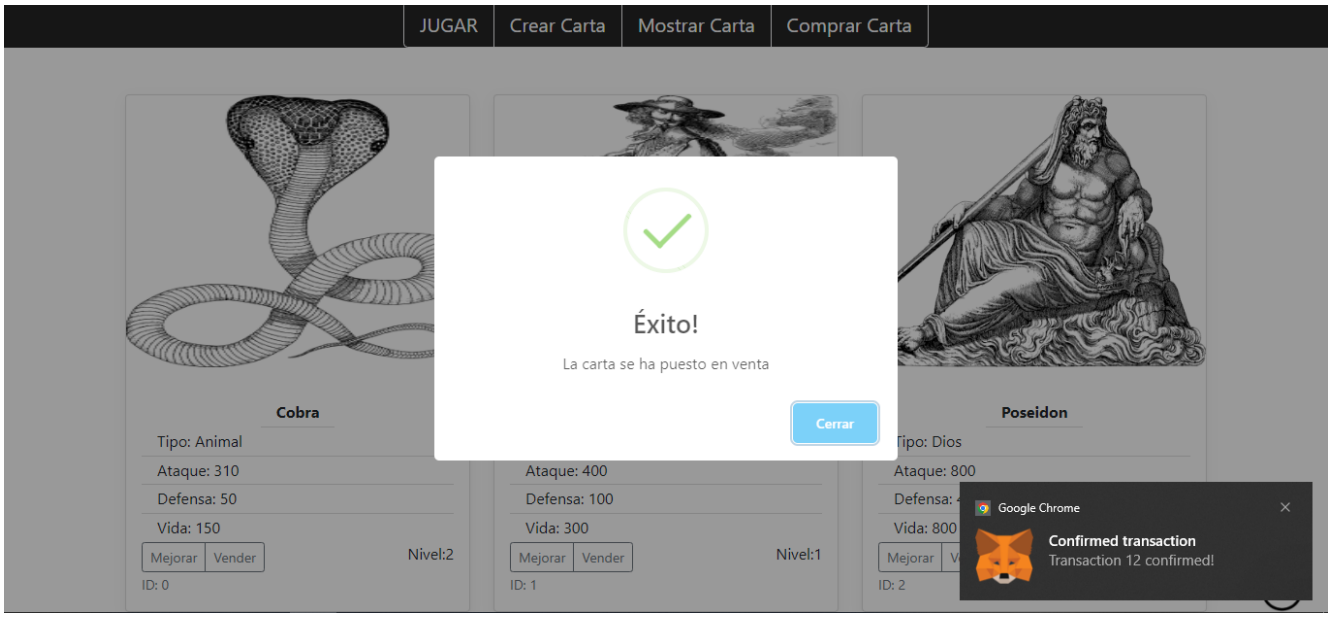
El usuario pretende vender la carta por 3 ETH.



Puede suceder que la carta ya estuviera en venta o que se produzca un error en la transacción.

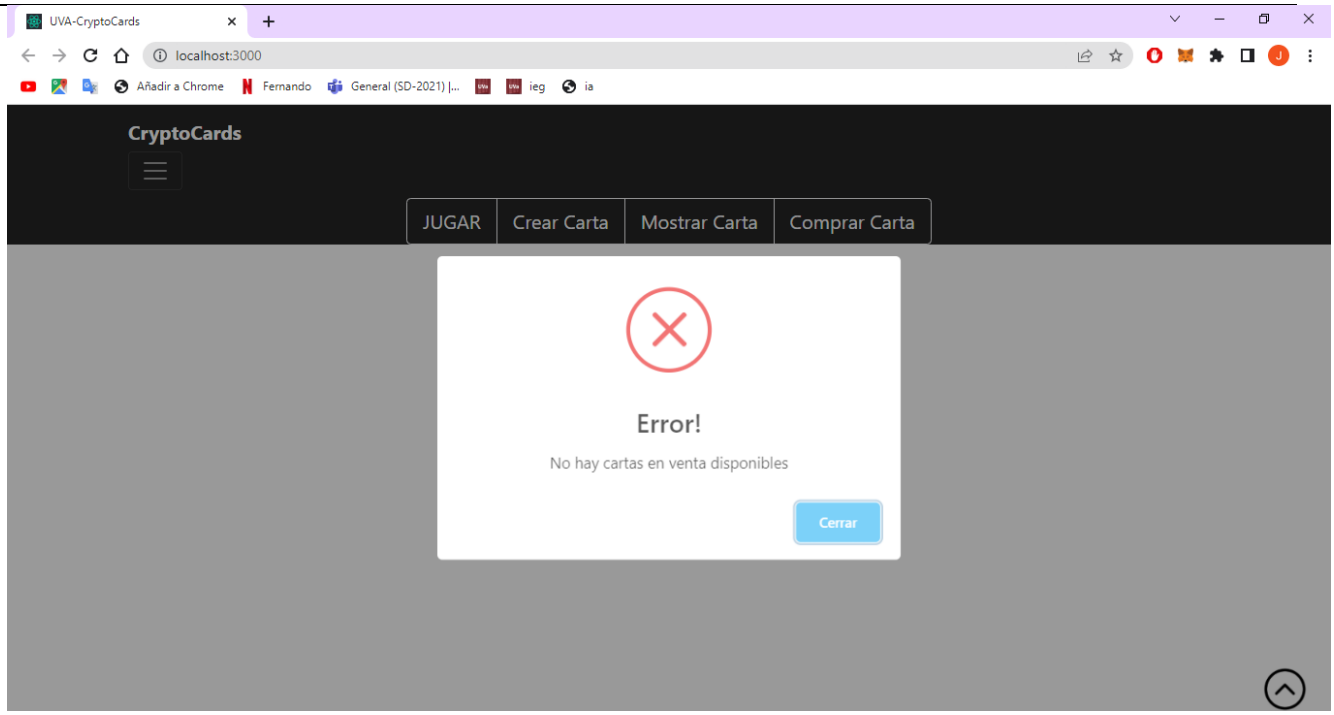


En caso contrario la carta se pone en venta.

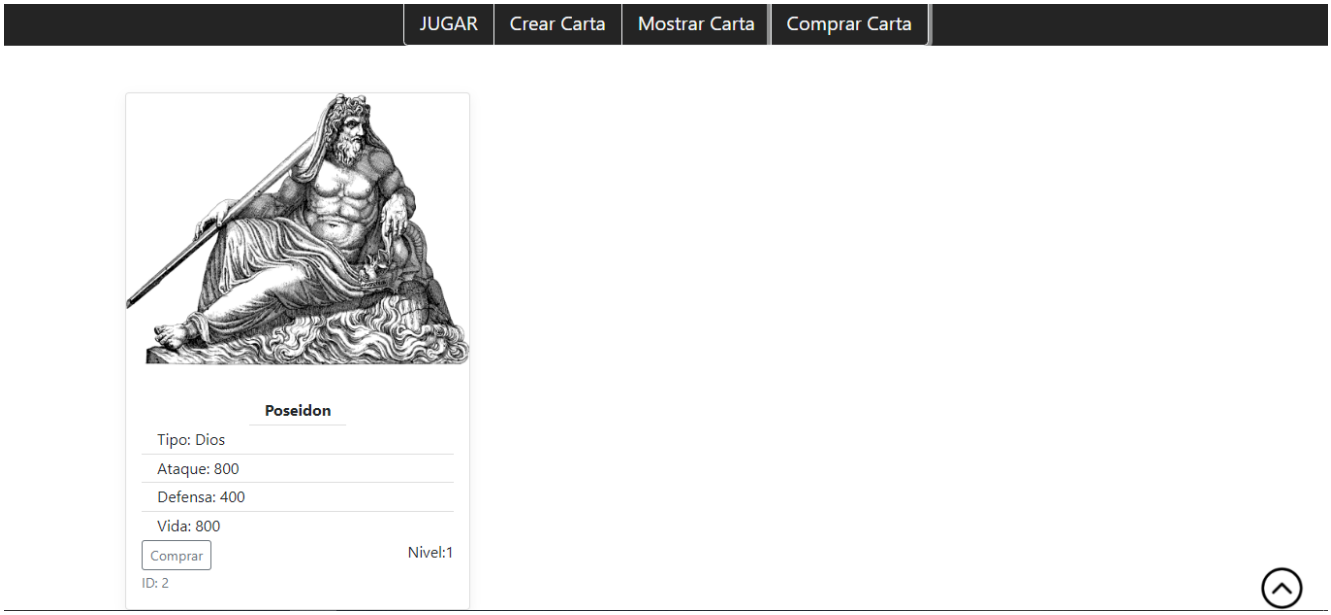


Comprar carta

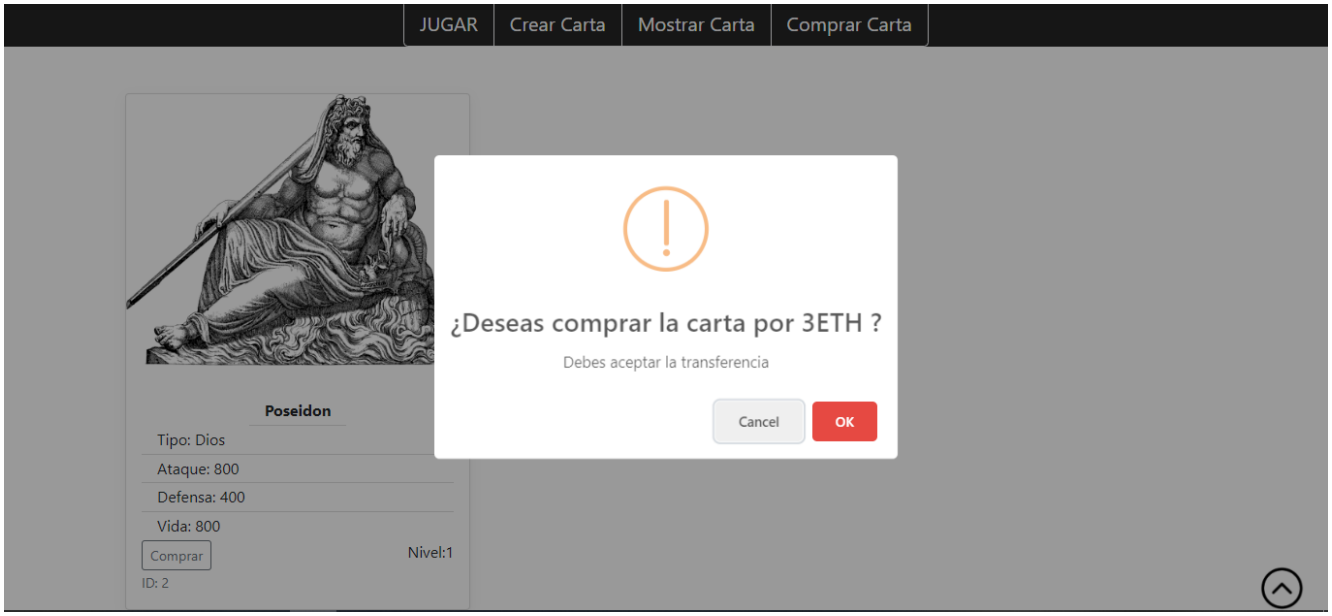
Este mismo usuario quiere comprar ahora una carta, pero ningún otro usuario ha puesto en venta alguna de sus cartas.



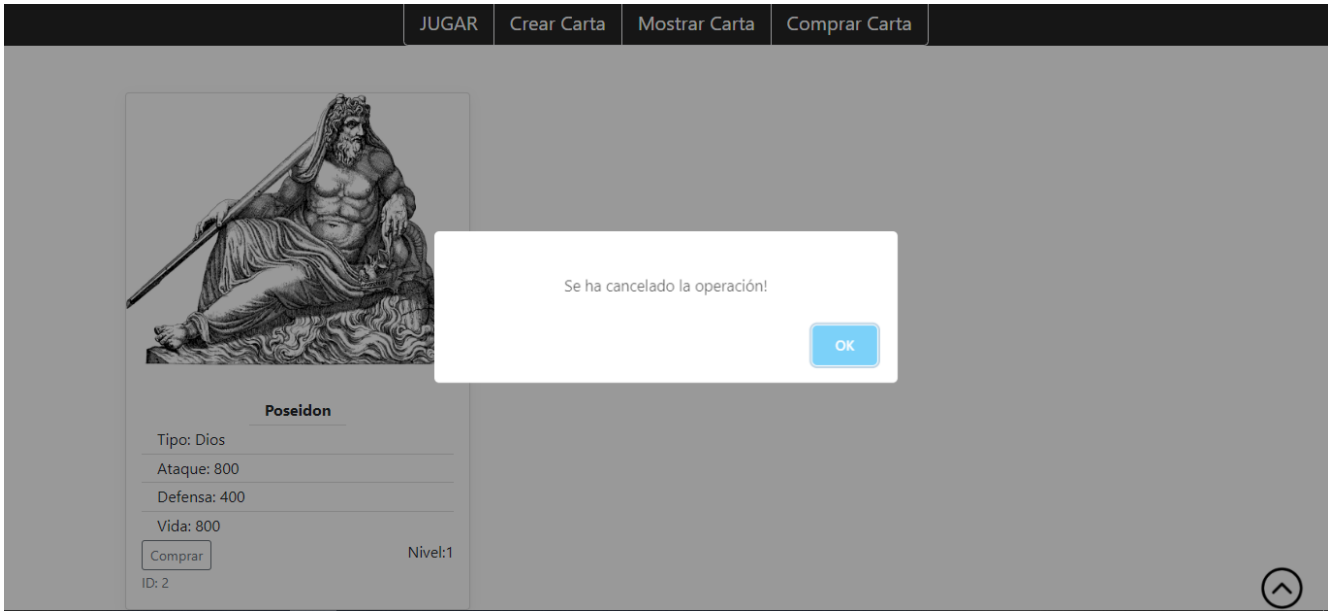
Ahora un segundo usuario desea comprar una carta. Para ello, pulsa sobre el botón Comprar Carta. En este caso se muestra la carta de Poseidon que el primer usuario puso en venta.



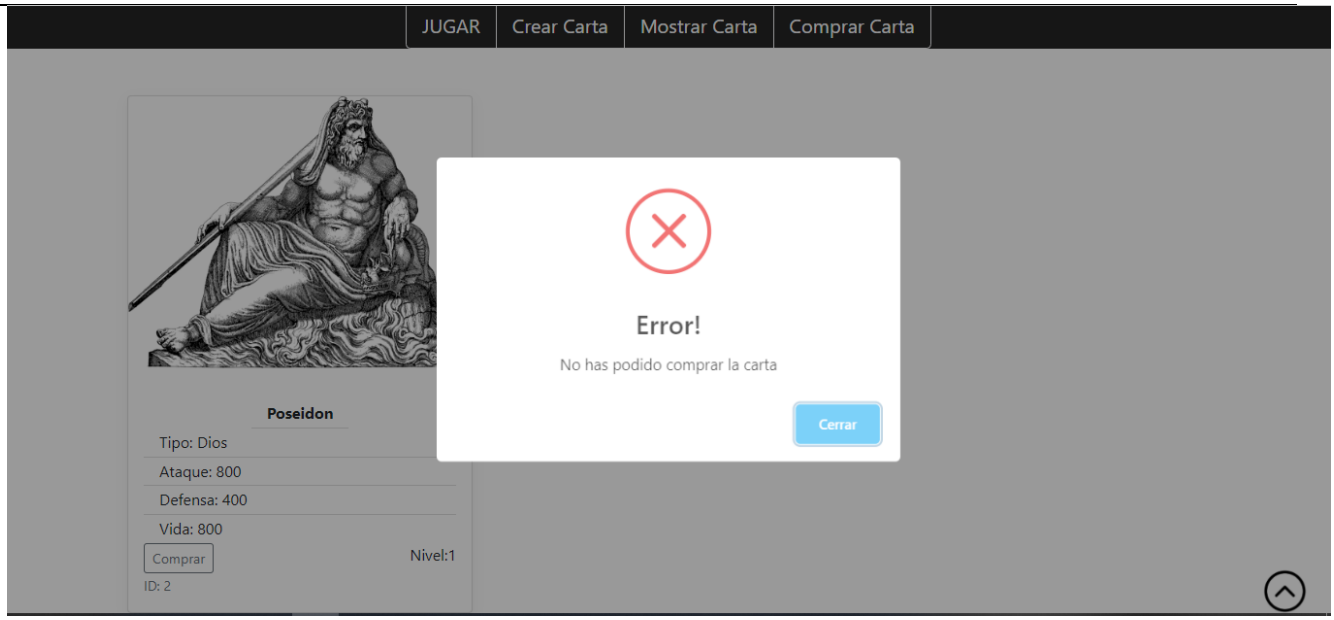
Pulsa sobre el botón de Comprar. Salta un mensaje indicando si se quiere comprar la carta y el precio fijado por su dueño que en este caso fue de 3 ETH.



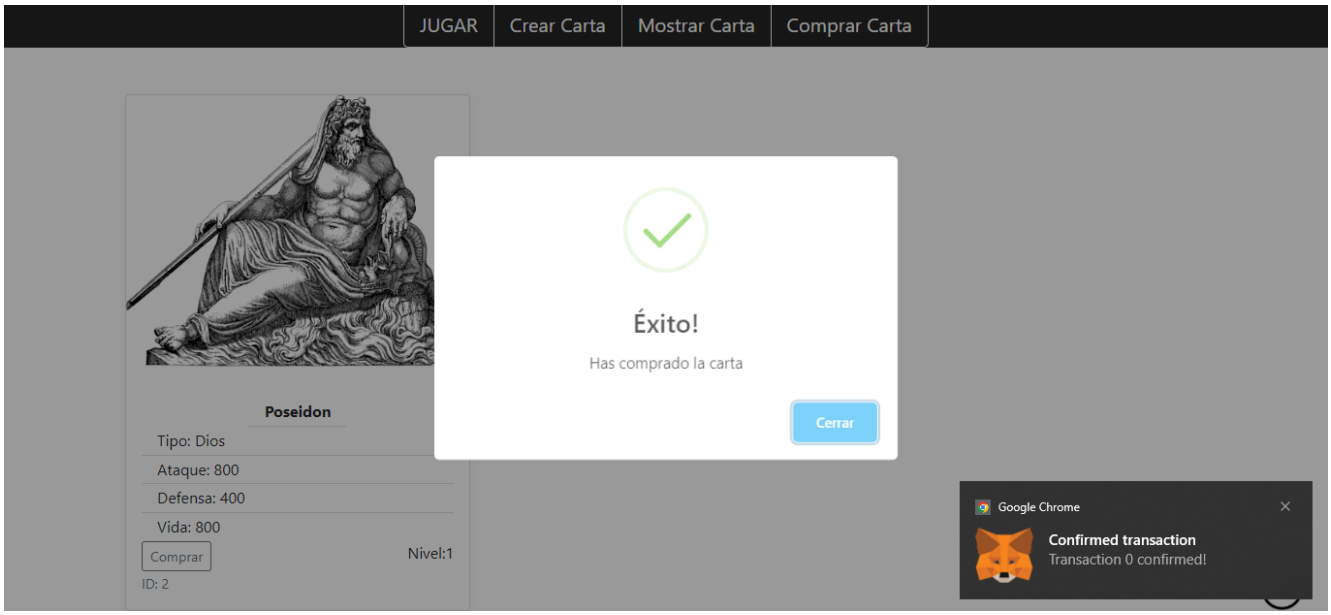
El usuario puede cancelar la operación.



También puede suceder que el usuario desea comprar la carta, pero la transacción es rechazada.

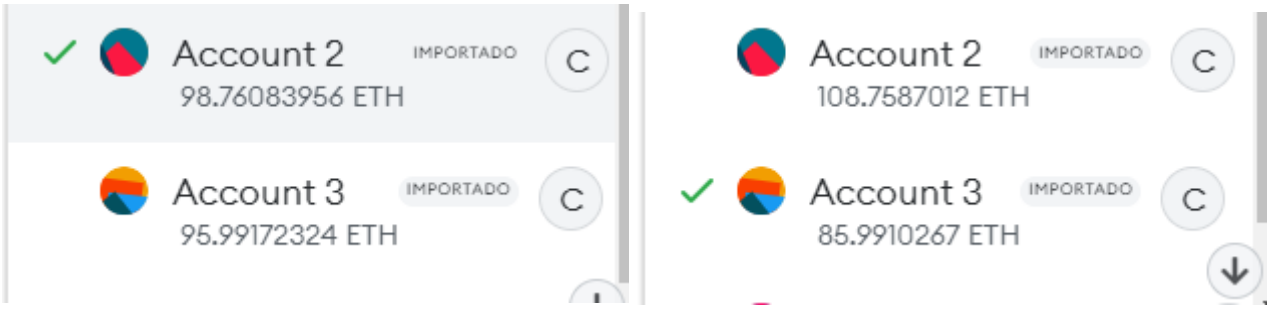


Por otro lado, la operación puede ser exitosa y el usuario añade la nueva carta a su colección.



En este caso se puede comprobar si se realizado la transferencia al antiguo dueño de manera correcta. Esto se puede observar tanto desde Ganache como desde Metamask.

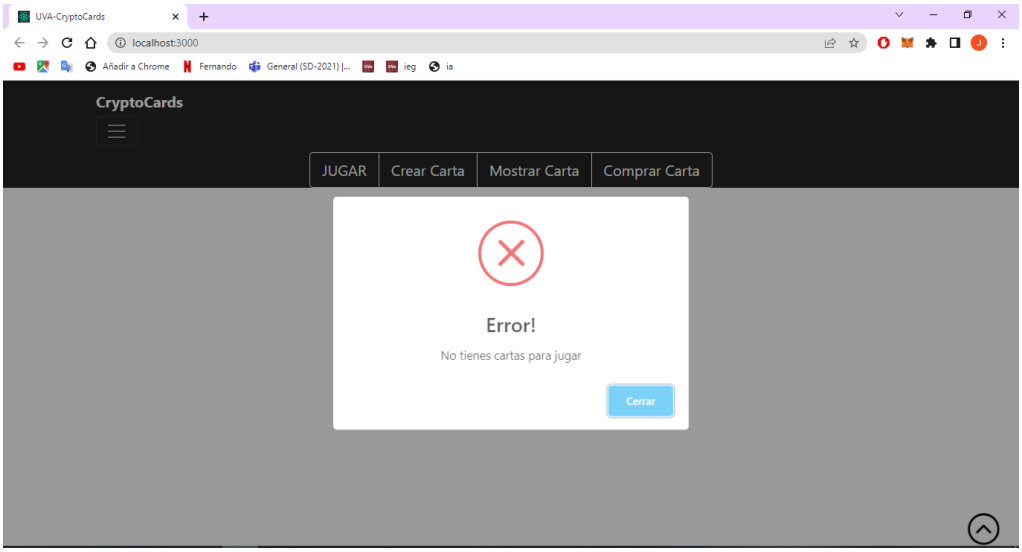
Ejemplo: La cuenta 3 compro la carta a la cuenta 2 por un total de 10 ETH.



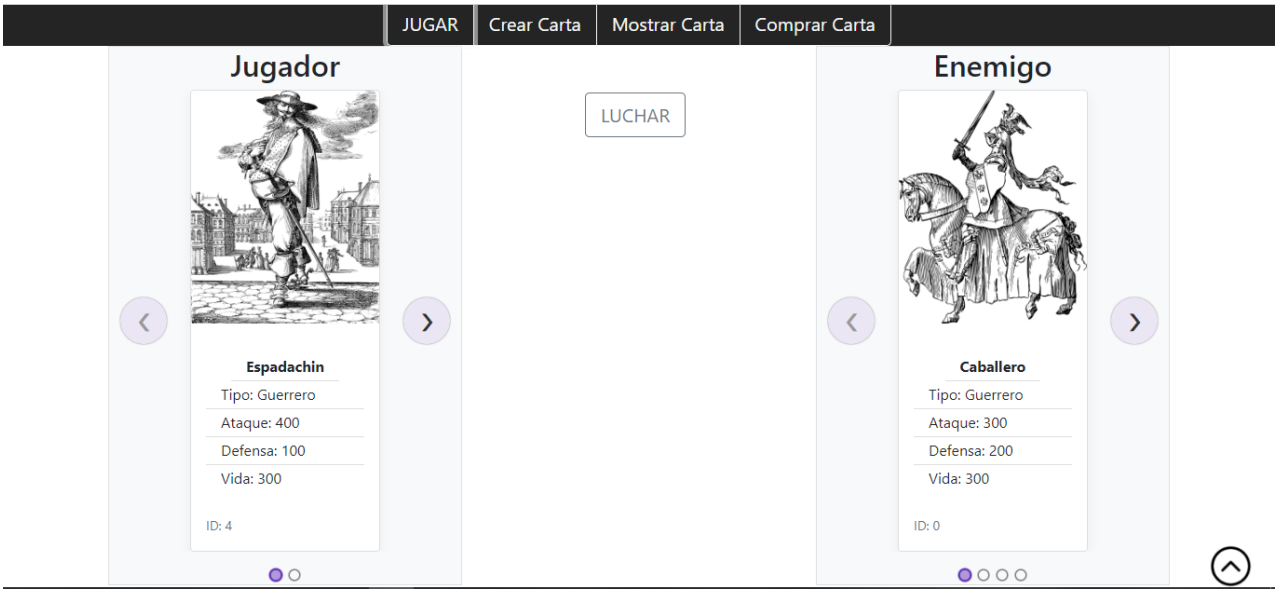
Además, se puede ver como se cobra el gas de las transacciones. Por un lado, poner en venta la carta por parte de la cuenta 2. Por otro lado, comprar la carta por parte de la cuenta 3.

Jugar

Para jugar es necesario que tanto el usuario, como otros usuarios de la aplicación tengan cartas para poder enfrentarse.



En caso de que se cumplan ambos supuestos. Se muestra la vista del juego y ahora el usuario puede elegir entre algunas de sus cartas y también seleccionar la carta del enemigo con la que se quiere enfrentar.



En este caso el usuario posee dos cartas, mientras que el resto de los usuarios hacen un total de 4 cartas. El usuario puede seleccionar cualquiera de sus cartas y las de otros usuarios para enfrentarlas entre ellas.

En este caso decide seleccionar las cartas que se muestran en la anterior imagen y pulsa sobre el botón de Luchar para ver el resultado.

En este caso el usuario ha resultado ganador.