Universidad de Valladolid

Escuela de Ingeniería Informática de Segovia

Grado en Ingeniería Informática de Servicios y Aplicaciones

SCiBEth: Notario virtual en la cadena de bloques Ethereum

Alumno: Luis Mateos Fernández

Tutores: José Vicente Álvarez Bravo

Agradecimientos

"A mis padres, a mis hermanos y a toda mi familia, gracias a quienes soy quien soy y hacia quienes sólo puedo expresar mi más sincero agradecimiento por apoyarme durante la etapa académica que hoy culmina, gracias a aquellos compañeros y amigos que me han apoyado y han confiado en mi y gracias a mi profesor que me ha apoyado en este proyecto personal en el cuál me embarque y hoy os puedo presentar. "

Índice general

Índice de figuras

Índice de tablas

.

Capítulo 1

Introducción

En este capítulo se va a presentar una introducción y se expondrán las razones que han motivado la elección y realización del proyecto.

1.1. Introducción

La creación de Internet ha transformado la vida para gran parte del mundo, permitiendo la reducción de costes en investigación gracias a la enorme cantidad de información gratuita que ofrece, facilitando la comunicación entre personas gracias al correo electrónico y las redes sociales y ofreciendo gran cantidad de posibilidad de entretenimiento para los usuarios. Pese a esta gran revolución, Internet presenta ciertas limitaciones.

Muchos de los servicios que usamos a diario en Internet requieren que el usuario se conecte a un servidor para acceder a una aplicación en concreto. Por ejemplo, para acceder a páginas como Google o Facebook hay que pasar por alguno de los servidores en donde están alojados. Esto supone que muchas de las aplicaciones y servicios online que usamos cada día están centralizados.

La organización de la red según esta estructura conlleva varios problemas. El primer problema es el de la seguridad, si un servidor sufre un ataque, generalmente suele poner en peligro a todos sus usuarios. El segundo problema es el de la confianza, los usuarios no saben qué ocurre detrás de la aplicación una vez ha mandado sus peticiones. En una sociedad donde cada vez se confía más información sensible a estos sistemas, estos problemas pueden suponer grandes pérdidas para los usuarios.

Esta realidad ha forzado a que los usuarios que desean realizar transacciones por Internet necesiten de una tercera parte que garantice la seguridad de la transacción, siendo representado este rol normalmente por un banco. Estos sistemas que actúan como intermediarios almacenan la información de los usuarios, invadiendo así su privacidad y además suelen cobrar comisiones.

De este contexto surgió la tecnología de la cadena de bloques o Blockchain, ya que permite que los usuarios lleguen a un consenso de manera distribuida y segura sin necesidad de un servidor central. Años después, Ethereum supo aprovechar esta tecnología para permitir el desarrollo de aplicaciones distribuidas a todo tipo de usuarios con conocimientos técnicos.

Por lo tanto, podemos definir Blockchain como un tipo de base de datos distribuida que funciona como un libro de cuentas que va almacenando cualquier tipo de transacción realizada y que será verificada a través de consenso entre la mayoría de los participantes de la red. Es importante señalar que cuando hablamos de transacción no nos referimos a algo monetario, sino a un intercambio de información.

Aunque existen muchas plataformas que trabajan dentro del paradigma Blockchain (Bitcoin, Eris, Ethereum, Hyperdeledger. . . .), nosotros hemos elegido **Ethereum**.

Ethereum es una plataforma Blockchain, en la que aquellas aplicaciones basadas en transacciones pueden ser implementadas. Además, no sólo provee la metodología para realizar dichas aplicaciones, sino que también ofrece programas de desarrollo para que cualquier persona pueda llevarlas a cabo.

Uno de los principales problemas a la hora de llevar a cabo una transacción es la desconfianza que puede surgir entre los usuarios implicados. Esta desconfianza se debe a muchos factores: desde una separación geográfica, cambio de moneda o incluso, situación financiera del país de alguno de los intervinientes. Ethereum nace para poner fin a estos problemas, creando un sistema de transacciones pseudo-anónimas, seguras y con un lenguaje completo, no ambiguo y accesible para cualquier persona.

Ethereum cuenta con su propia criptomoneda, el Ether. Este tipo de moneda, no se utilizada únicamente para realizar transacciones de valor monetario, sino que también es utilizada a la hora de desplegar contratos inteligentes.

Está tecnología cuenta con una gran cantidad de aplicaciones, pero en este trabajo nos centraremos en su uso para la creación de dichos contratos inteligentes (smart contracts) y propiedades inteligentes (smart property) sobre la red de Blockchain, de tal forma que se encargue automáticamente de otorgar el cambio del bien en forma de propiedad inteligente según el dinero acordado en forma de criptomoneda, a través del contrato inteligente. En resumidas cuentas, se quiere conseguir que cuando el evento A suceda, entonces la consecuencia B se ponga en marcha de forma automática, sin requerir de ningún intermediario de confianza como podría ser una notaría, consiguiendo ahorrar tiempo y costes significativos.

De esta forma queremos conseguir una mejor experiencia entre compradores y vendedores, consiguiendo una mayor seguridad, lo que implicará usuarios más contentos y la captación de nuevos clientes.

1.2. Motivación

El propósito principal de este proyecto nace al tener conocimiento de está tecnología y ver su gran potencial. Por ello me decidí a intentar desarrollar una aplicación de compra/venta a través de subastas con un notario virtual, el **smart contract**, que será el encargado de cambiar la propiedad del producto y dar el dinero al vendedor una vez se cumpla lo acordado en el smart contract, ni antes ni después, y si alguno incumple algún punto, se podrá reembolsar el dinero.

Esta plataforma de venta a través de pujas pretende ser innovadora a la hora de crear una relación entre vendedores y compradores más fiable, ya que el contrato es público y no se puede alterar una vez está registrado dentro del Blockchain de Ethereum.

De está manera, cualquier persona puede poner productos a la venta y pujar por los productos que hay en venta y no sean suyos.

Esta plataforma es de software libre, gratuita y carece de un servidor centralizado.

1.3. Objetivos y alcance

El objetivo principal de este proyecto es el diseño e implementación de una DAPP o aplicación descentralizada, con la que se pueda interactuar a través de una aplicación web.

Este objetivo tiene asociados los siguientes subobjetivos:

1. Aprender y experimentar lo que son las aplicaciones distribuidas y las criptomonedas usando lenguajes de programación especiales para ello, como es Solidity.

2. Promover una forma de compra/venta entre particulares de una forma innovadora y sin estructuras centralizadas.

3. Sentar las bases para una gran diversidad de trabajos futuros, ya que, es una tecnología innovadora y con mucho futuro en la que hay una gran cantidad de posibles aplicaciones.

Esta aplicación podrá ser utilizada por cualquier usuario con unos conocimientos básicos sobre el manejo de Internet. Su navegación será sencilla e intuitiva gracias al diseño de una interfaz limpia y bien definida, diseñada para poder acceder a ella desde cualquier dispositivo.

La innovación más grande es que vamos a pasar de construir una APP que es lo que se conoce actualmente, a construir una DAPP. La diferencia más grande es que, una APP esta centralizada y dirigida por una empresa la cual tiene el control total de la aplicación y puede desaparecer, mientras que una DAPP esta descentralizada y el control le tienen los propios usuarios que usan la aplicación la cual nunca va a desaparecer y en la cual ellos son los dueños de su información. Más adelante se explicara con más detalle lo que es una DAPP.

Al ser una DAPP y gracias al uso de Blockchain, los usuarios no tendrán que registrarse, ni ceder ningún dato significativo, por lo que no correrán peligro ninguno de sus datos y es uno de los principales atractivos de la aplicación. Pretende ser una aplicación que dote de confianza a los usuarios sin ser una aplicación centralizada, sino que sea una aplicación descentralizada de usuarios para usuarios integramente, creando un mercado de compra/venta entre profesionales y particulares.

El modelo de negocio a seguir que se ha pensado, es una mezcla entre el modelo de negocio de Wallapop y el modelo de negocio de Ebay, intentando coger lo mejor de cada uno de ellos.

La idea principal es centrarse en la expansión internacional y la captación de usuarios, sin un modelo de negocio los primeros 3 años. En esta primera etapa el único ingreso que vamos a tener va a ser el generado por ventas directas en las que no haya pujas o las ventas en las cuales nosotros actuemos como un árbitro más de la aplicación.

En está fase la única comisión que hay por ventas es el 1 % de la venta que va al árbitro, y el 99 % restante al vendedor, en caso de una venta exitosa. Para poder ser un usuario de tipo árbitro y beneficiarte de este premio, tendrás que dejar 5 ETH a modo de fianza, esto hará que los usuarios quieran usar nuestra aplicación para vender y comprar gracias a la confianza que da y para ser árbitro y finalizar subastas y beneficiarte de la pequeña comisión que te llevas.

En la segunda fase, una vez alcanzado una extensión y un número de usuarios activos atractivo en la aplicación, se abrirán las puertas a los anuncios de terceros para ganar algo más de dinero, pero sin ensuciar la web.

En la tercera parte, de nuestro modelo de negocio añadirá una opción a la hora de añadir los productos para que puedan ser patrocinados y de está manera que salgan arriba, que se renueven cada x tiempo solos y que más gente los pueda ver, y se añadirá una opción para usuarios profesionales, los cuales podrán crear una especie de tienda en nuestra aplicación para vender.

De esta manera mejoraríamos el modelo actual de negocio de Wallapop, el cual no dota de confianza una venta a no ser que la venta se haga en mano, así podríamos unir a personas de todo el mundo y no de una zona geográfica concreta solo.

Del modelo de negocio de eBay nos hemos quedado con el modelo de subastas, aunque no es exactamente igual, ya que en nuestro caso se trata de una subasta a ciegas, es decir, tu lanzas una única puja que nadie va a conocer y el resto de usuarios igual, y una vez que acaba el tiempo de lanzar subastas se revelan las pujas que ha habido y la más alta es la ganadora y como premio el precio del producto será el valor de la segunda puja más alta. Las pujas tienen que estar comprendidas entre el precio de comprar ahora que ha definido el vendedor, que es el precio máximo del productos y la puja mínima.

También del modelo de negocio de eBay hemos cogido la idea de crear negocios online para nuestra tercera parte de modelo de negocio, en la cual los usuarios podrán crear e-commerce sin la necesidad de tener una tienda física, ni un stock y lo más importante sin realizar una gran inversión.

1.4. Estructura del proyecto

En este apartado se describen los diferentes capítulos de este documento. La memoria contará con 11 capítulos con sus respectivas secciones y un apartado de webgrafía. A continuación se realizará una breve descripción del contenido de cada uno de los apartados de la memoria.

- **Capítulo 1 - Introducción:** en este capítulo, se expondrá una pequeña introducción al proyecto seguido de las razones por las cuales se a elegido realizar el proyecto. Además se explicarán los objetivos que se desean cumplir junto al alcance del proyecto.
- **Capítulo 2 - Estado del arte:** aquí se pretende detallar un pequeño estudio realizado sobre aplicaciones similares a la propuesta. Para ello, en primer lugar, se describirán las aplicaciones estudiadas y finalmente, se realizará una comparación de las funcionalidades con la aplicación propuesta.
- **Capítulo 3 - Introducción a la tecnología del Blockchain y el origen de Ethereum:** en este tercer capítulo, haré una introducción a la tecnología que voy a utilizar, para comprender mejor los conceptos, junto con el origen de las criptomonedas y porque son interesantes, terminando con el origen de Ethereum.
- **Capítulo 4 - Metodología y Tecnología utilizada:** explicaré con detalles la metodología y el plan de trabajo utilizado para la organización y el desarrollo del proyecto, viendo una pequeña descripción de las tecnologías y las herramientas utilizadas.
- **Capítulo 5 - Planificación y Presupuesto:** en este capítulo, detallaré la planificación temporal llevada a cabo, junto a los distintos presupuesto de la aplicación, viendo con una comparativa de ellos, y por último, se expondrán los costes reales del proyecto.
- **Capítulo 6 - Análisis:** aquí se muestra el análisis previo a la implementación de la aplicación web.
- **Capítulo 7 - Diseño:** en el séptimo capítulo, se pretende exponer el trabajo de diseño previo realizado. Se tratará tanto la arquitectura lógica como la física, al igual que el diseño de algunas de las pantallas de la interfaz de usuario.
- **Capítulo 8 - Implementación:** en este capítulo, se procederá a la explicación del desarrollo llevado a cabo a lo largo del proyecto.
- **Capítulo 9 - Pruebas:** aquí se expondrán las pruebas realizadas sobre el resultado de la implementación del sistema. En primer lugar se verán las pruebas de caja blanca y a continuación, las pruebas de caja negra.
- **Capítulo 10 - Manuales:** en esté capítulo, se redacta el manual de instalación y el manual de usuario.
- **Capítulo 11 - Problemas, conclusiones y trabajos futuros:** se realiza una revisión del trabajo realizado, exponiendo en primer lugar los problemas encontrados, seguidos de las conclusiones y terminando con un apartado de trabajos futuros para mejorar el proyecto.
- **Webgrafía:** en este apartado se recogen todas las web a las que se ha hecho referencia o se ha consultado información para la realización del proyecto.

Capítulo 2

Estado del arte

En la actualidad existen multitud de aplicaciones web o móviles de mercados de compra/venta entre particulares y profesionales. Estás apps son muy diferentes en si mismas, por lo que antes de comenzar a desarrollar nuestra DAPP vamos a realizar un pequeño estudio de las aplicaciones que hay en el mercado actualmente que sirven un servicio parecido al que queremos desarrollar.

2.1. eBay

eBay es un sitio destinado a la subasta de productos a través de Internet. Es uno de los pioneros en este tipo de transacciones, habiendo sido fundado en el año 1995. Desde 2002 y hasta 2015, eBay a sido propietario de PayPal, pero desde Junio de 2015, el director ejecutivo (CEO) de eBay es Devin Weing.

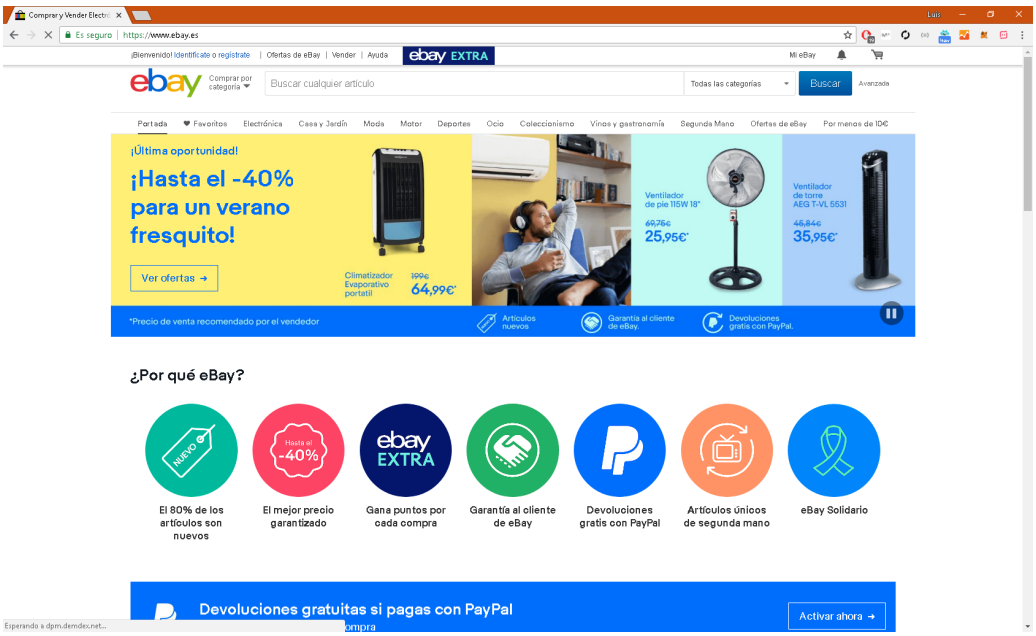


Figura 2.1: Captura de eBay

eBay ofrece 3 características:

- **Subasta:** es la transacción más común en el sitio. El vendedor pone un precio de salida y una duración determinada para el anuncio y mientras dure ese período de tiempo, los compradores pujarán por ella. El ganador de la subasta, se llevara el artículo, bajo las condiciones de entrega y devoluciones impuestas por el vendedor.
- **¡Cómprarlo ahora!:** el vendedor establece un precio fijo y, si el demandante está dispuesto a pagarlo, será suyo.
- **Anuncio clasificado:** venta de artículos bajo esta forma de anuncio, en el que se exponen las características del artículo en cuestión y su precio.

A cambio de publicar su anuncio, eBay cobra una comisión al oferente en caso de venta, en proporción al precio final de la venta.

A pesar de que últimamente se ha aumentado la seguridad y la protección al comprador, no hay que olvidar que cualquier transacción en eBay es un trato en el que no interviene la empresa, por lo que el éxito de la compraventa depende de la buena fe de la otra parte. eBay realmente, es un sistema de intermediación automático, en el que los usuarios pueden clasificar al otro usuario mediante un sistema de puntos positivos o negativos, dependiendo del éxito de la operación. Sin embargo, este sistema ha demostrado ser a menudo fácilmente manipulable, por lo que en el pasado se han dado casos de estafas con vendedores asiáticos y productos de alta tecnología. Para evitar esto, hay que fijarse en la forma de pago que el vendedor ofrece, y si esa forma de pago dispone de la protección al comprador de eBay, que cubre una cierta cantidad en caso de fraude, aunque con unos gastos de 30€por reclamación. [22]

Pero, ¿por qué tuvo ese gran éxito eBay?

eBay ha sido un éxito masivo porque hace que comprar y vender sea extremadamente fácil y eficiente. Antes de que Internet se convirtiera en algo cotidiano, uno podía comprar y vender productos en su vecindario local o dentro de un cierto radio geográfico. A medida que más y más personas accedieron a Internet, y que empresas como eBay entraran en escena, cualquiera podía comprar y vender cualquier cosa en Internet desde cualquier lugar del mundo.

Aunque esto beneficia a todos y mejora el comercio y la economía en general, tiene algunos inconvenientes tales como:

- **Los comerciantes participantes están a merced de la compañia:** La compañia puede decidir si bloquear el comercio del comerciante en su plataforma según su propio criterio en cualquier momento, lo que podría ser un gran golpe para el comerciante, cuyo sustento podría depender de esto.

- Los comerciantes pagan tarifas para mostrar su producto y también pagan comisiones sobre las ventas. Pagar tarifas en sí mismo, no es tan malo, porque eBay ofrece un servicio. Sin embargo, las **tarifas de cotización a veces son demasiado elevadas** para que el comerciante termine con muy poco margen o pase esa tarifa al consumidor.

- **Comerciante y consumidor no poseen ninguno sus datos**. Con esto me refiero, a que no poseen sus propias reseñas, historial de compras, etc, son todas propiedad de la compañia. Por ejemplo, si un comerciante desea trasladar su operación a otro proveedor, no es trivial o casi imposible de exportar sus revisiones u otros datos.

2.2. Wallapop

Wallapop es una empresa española fundada en 2013, que ofrece un sitio web dedicado a la compra y venta de productos de segundamano entre usuarios a través de Internet, con un uso centrado sobre todo en smartphones.

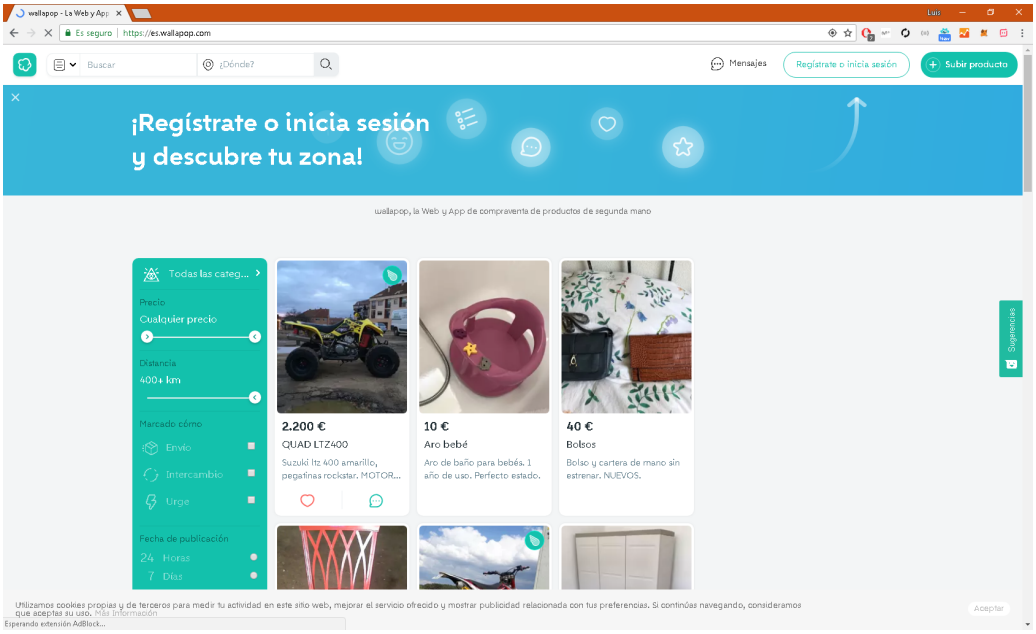


Figura 2.2: Captura de Wallapop

Utiliza la geolocalización para que los usuarios puedan comprar y vender en función de su proximidad geográfica.

Tiene una aplicación móvil tanto para dispositivos iOS como Android. El objetivo principal de Wallapop es ser un negocio de economía colaborativa entre particulares, con un sistema de geolocalización que ofrece la compra y venta de proximidad a través de un chat instantáneo. Se le añadió una función para evitar el uso fraudulento de la aplicación, conocido como *scoring*, que son valoraciones que se hacen entre usuarios y la validación del perfil.

2.3. Vibbo

Vibbo es una marketplace digital multiplataforma a través del cual particulares y profesionales pueden buscar y anunciar artículos en distintas categorías. Es propiedad de Schibsted Sapin, compañia que forma parte del grupo internacional de medios de origen noruego Schibsted Media Group.

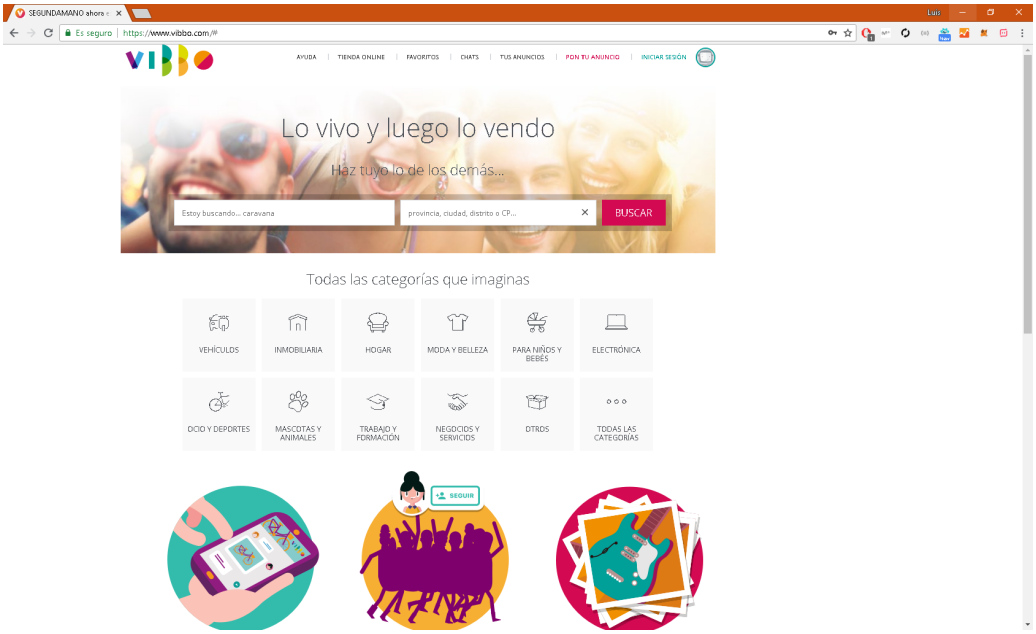


Figura 2.3: Captura de Vibbo

Vibbo fue lanzada en Noviembre de 2015 para renombrar al marketplace generalista de anuncios clasificados *segundamano.es*, que contaba con 37 años en el mercado de la compra y venta de artículos de segunda mano en España.

Vibbo opera a través de una app web y aplicaciones móviles, tanto para iOS como Android. Aunque la gran mayoría de las visitas son a través de smartphones.

2.4. Milanuncios

Milanuncios es una marketplace digital multiplataforma a través del cual particulares y profesionales pueden buscar y anunciar artículos en distintas categorías. Es propiedad de Schibsted Sapin, compañia que forma parte del grupo internacional de medios de origen noruego Schibsted Media Group, al igual que vibbo.

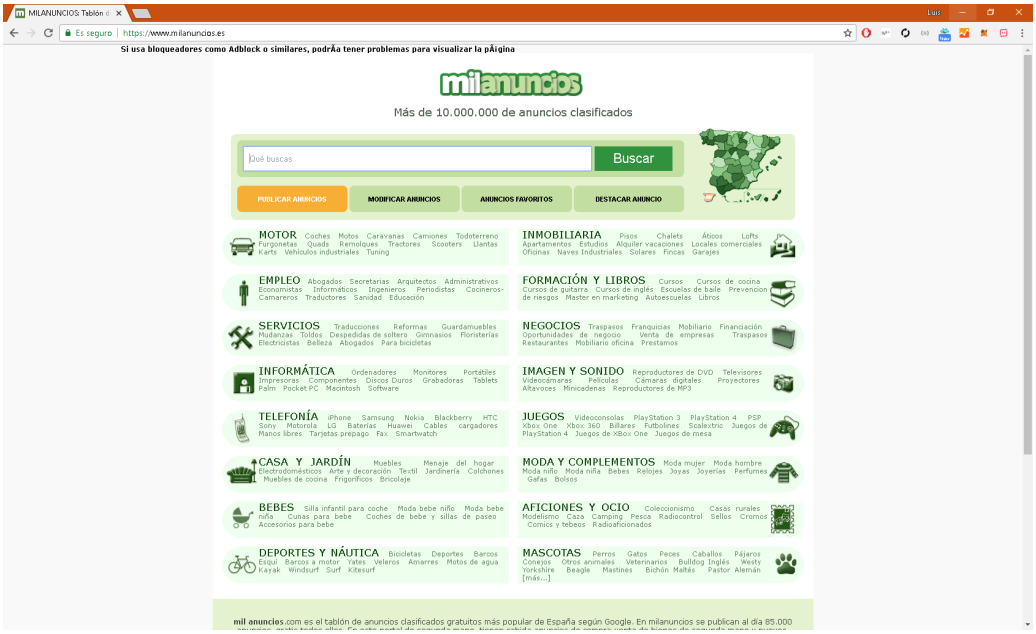


Figura 2.4: Captura de Milanuncios

Milanuncios opera a través de una app web y aplicaciones móviles, tanto para iOS como Android.

2.5. Comparación del análisis con la propuesta

	eBay	Wallapop	Vibbo	Milanuncios	Propuesta
Subir productos	Si	Si	Si	Si	Si
Comprar productos	Si	No	No	No	Si
Subasta de productos	Si	No	No	No	Si
Geolocalización	No	Si	No	No	No
Chat	Si	Si	Si	Si	No
Sin dar información personal*	No	No	No	No	Si
Protección de la compra	Si	No	No	No	Si
Evaluaciones	Si	Si	Si	No	No
Descentralizado	No	No	No	No	Si
Datos públicos**	No	No	No	No	Si
Cuenta imbloqueable	No	No	No	No	Si
Sitio web	Si	Si	Si	Si	Si
Aplicación móvil	Si	Si	Si	Si	No

Tabla 2.1: Comparación del análisis con la propuesta

*Sin dar información personal: como por ejemplo, correos electrónicos, nombres, tarjetas, etc.

**Datos públicos: los datos de la aplicación están al alcance de los usuarios.

Como podemos observar en la *tabla 2.1*, la aplicación propuesta cumple prácticamente con todas las funcionalidades observadas y con las que no cumple se pueden completar en futuras mejoras de la aplicación.

Como se puede observar, hay 4 puntos en los cuales nuestra aplicación mejora al resto de las aplicaciones, esto es debido, al ser una aplicación descentralizada.

Debido a que la aplicación más pionera en el mercado de compra y venta entre usuarios de todas es eBay y es la que más similitudes tiene con la aplicación que quiero realizar, decidí centrarme en mejorar esta aplicación, ya que, construir un mercado en una plataforma descentralizada, como Ethereum, es algo pionero y resuelve diversos problemas que hemos visto que tiene eBay como por ejemplo que, la cuenta del comerciante no se puede bloquear, los datos son públicos, por lo que se pueden exportar fácilmente y las tarifas por transacción son mucho menores en comparación con las empresas centralizadas.

Capítulo 3

Introducción a la tecnología de Blockchain y el origen de Ethereum

La Blockchain es una tecnología muy reciente que se empezó a popularizar a partir del año 2009 gracias a la aparición del Bitcoin. En esta sección se presentan los conceptos más importantes que se deben conocer y la evolución de esta tecnología.

3.1. Conceptos más importantes sobre Blockchain

En está sección veremos como es la arquitectura Blockchain y terminaremos con una definición de Blockchain y sus tipos.

3.1.1. Arquitecturas Distribuidas

Cuando hablamos de Ethereum o el Blockchain, una de los conceptos más interesantes es el relacionado con su arquitectura distribuida, ya que con ella, se elimina la centralización.

Existen tres tipos importantes de estructuras:

- ***Estructura Centralizada***: Toda la estructura está gestionada por un solo nodo y sus usuarios pertenecen a la misma comunidad. Se utiliza principalmente en servicios web, alojados en un servidor centralizado por el que tienen que pasar todas las personas que quieran acceder a ella (e.g. Facebook, Wikipedia, Github).
- ***Estructura Descentralizada***: La infraestructura está dividida en varios nodos operativos que funcionan como su propia estructura centralizada, fragmentado el servidor central en pequeños servidores distribuidos. Cada uno de estos tiene su propio dueño y su comunidad. A pesar de esto, cualquier usuario de la estructura puede acceder a los datos de otros independientemente del nodo al que pertenezcan (e.g. GNU Social, Buddycloud, Diaspora).
- ***Estructura P2P (Peer to Peer)***: Es una estructura totalmente distribuida, particionando los trabajos y la información entre los usuarios de la red llamados “Peers”. Cada uno de estos tienen los mismos privilegios en la infraestructura. Cada usuario controla su aportación a la red distribuida y generalmente todos pertenecen a la misma comunidad (e.g. BitTorrent, Twister, Bitcoin, Ethereum).

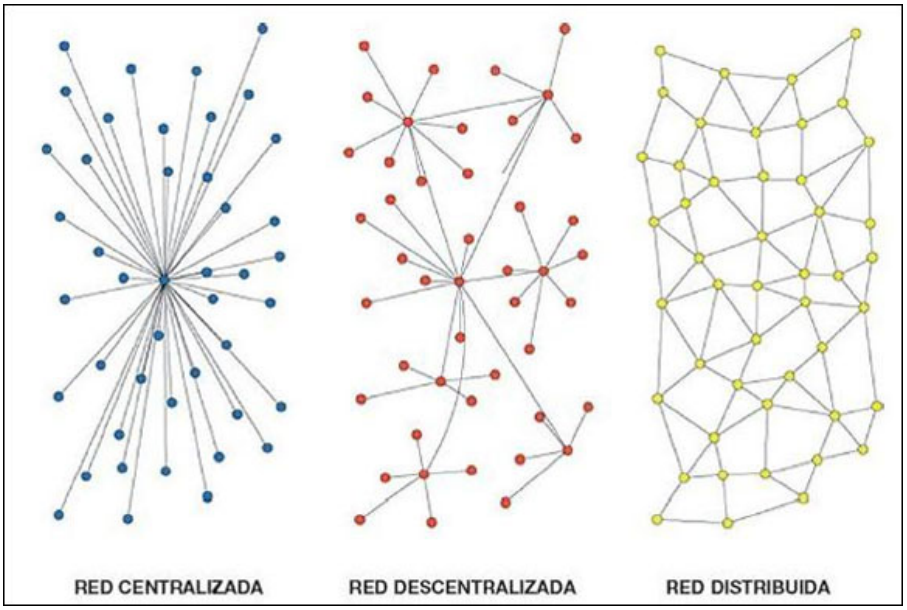


Figura 3.1: Representación gráfica de las arquitecturas de red

Uno de los ejemplos más importantes de una arquitectura distribuida (P2P) es el de Bitcoin, en el que todos los usuarios ven todas las transacciones y tienen los mismos derechos.

Ethereum utiliza la arquitectura P2P, al igual que Bitcoin, consiguiendo así que todo el software desarrollado en este lenguaje sea totalmente libre y descentralizado, ya que todos los usuarios de la red tienen acceso a todos los contratos de la cadena de bloques y al código fuente de estos.

3.1.2. Blockchain

Blockchain o la cadena de bloques es un tipo de base de datos descentralizada que almacena la información en forma de bloques, en cada bloque se guarda información sobre las transacciones realizadas, el tiempo en el que el bloque fue añadido a la cadena de bloques y el hash del bloque anterior. De esta forma conseguimos que la validez de las transacciones futuras pueda ser verificado consultando el último estado de la dirección de envío.

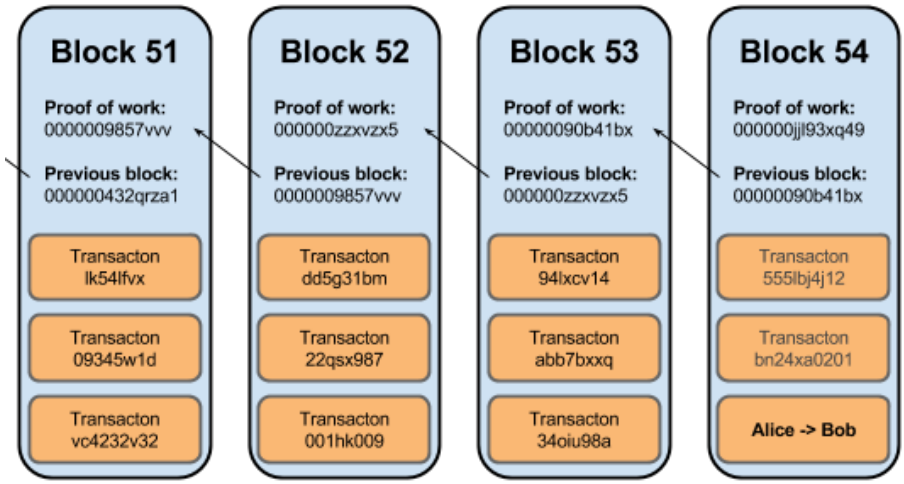


Figura 3.2: Representación de Blockchain [41]

Un ejemplo de como funcionaría Blockchain sería la siguiente imagen:

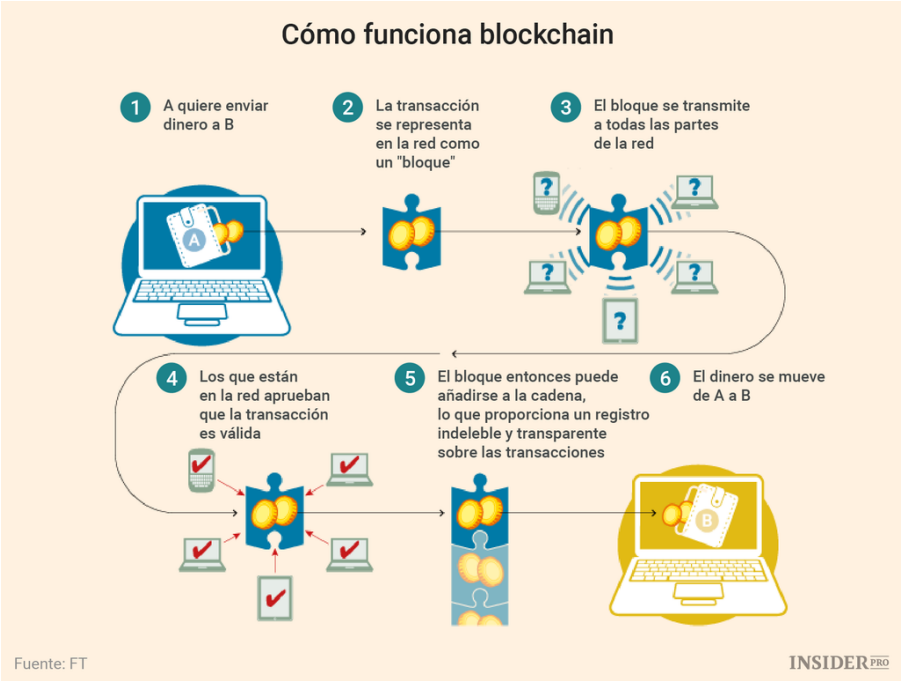


Figura 3.3: Funcionamiento de Blockchain [40]

La inclusión del próximo bloque de la cadena se realiza mediante un sistema de prueba de trabajo, conocido como "*minar* ". [7]

La minería permite a los nodos de la red alcanzar un consenso de formar distribuida. Los diferentes nodos de la red compiten mediante potencia de cálculo computacional para resolver un problema matemático. El nodo que resulta ganador es el encargado de formar el siguiente bloque con las transacciones pendientes de verificar y de añadirlo a la cadena de bloques.

Este sistema permite la creación de nueva moneda mediante la generación del *coinbase* [17] de cada bloque, una pequeña cantidad de moneda que va a parar al generador del bloque junto con la comisión pagada por los usuarios cuyas transacciones enviadas han sido influidas por él.

Actualmente, la tecnología de Blockchain va madurando incrementalmente y con ella, las plataformas que la implementan.

Hoy en día podemos clasificar las redes de Blockchain por sus generaciones, siguiendo la siguiente clasificación:

- **Primera generación:** se basa en la idea de realizar un sistema de registro compartido o un "ledger "donde poder ver las transacciones.
- **Segunda generación:** se extiende la idea anterior donde las plataformas crean una red donde se puedan utilizar criptomonedas y donde se almacenan las relaciones del crédito y divisas definidas por los usuarios.
- **Tercera generación:** plataformas cuyo propósito principal es la creación de aplicaciones descentralizadas usando como tecnología subyacente las plataformas de segunda generación. [13]

3.2. Criptomonedas

En está sección hablaremos del origen de las criptomonedas, ya que son las que dieron a conocer la tecnología Blockchain.

3.2.1. Moneda Descentralizada

Todo empezó en la década de los 80 y los 90, que es cuando empiezan a aparecer las formas de pago descentralizadas como *e-cash* [2], en su mayoría dependen de una primitiva criptografía conocida como primitiva ciega de Chaum (**Chaumian blinding**). Esta ofrecía una **moneda con un alto nivel de privacidad**, pero que no llegaría a despegar debido a su dependencia de un intermediario centralizado. Fue entonces cuando empieza a surgir el concepto del *"dinero electrónico ánonimo"*.

La primera mención del concepto que hoy conocemos como ***criptomoneda*** **aparece en 1998 por Dai Wei**, con su propuesta de dinero electrónico llamada **B-money** [5],por lo que se puede decir, que fue uno de los pioneros en la creación de una moneda electrónica descentralizada. Dai Wei proponía con B-money un *"sistema de efectivo electrónico distribuido y anónimo"*. Surge la idea de la creación de dinero mediante la resolución de puzles computacionales, así como el consenso descentralizado, pero la propuesta se mostró escasa en los detalles de cómo podría ser implementado éste consenso descentralizado en la práctica.

En 2005, **Hal Finney** introdujo el concepto de "**pruebas reutilizables de trabajo**", un sistema que utilizaba algunas ideas de B-money junto con los rompecabezas computacionalmente difíciles de romper de **Adam Back** de tipo Hashcash [3], para volver a crear un concepto de criptomoneda, pero que una vez más se quedaría escasa al apoyarse en el uso de informática de confianza en el backend.

A partir de las ideas de Dai Wei han surgido otras propuestas como *Bit gold* [6] para mejorar la implementación de la criptomoneda haciendo uso de *RPOW* [43], una extensión del sistema de prueba de trabajo de Hashcash .

Pero no fue hasta el año 2009 cuando se implantó por primera vez una *moneda descentralizada*, cuando Satoshi Nakamoto publicó la primera versión del Bitcoin [10]. Esta moneda tenía como primer objetivo crear un sistema de pagos electrónicos completamente descentralizado,

empleando las pruebas criptográficas en lugar de la confianza mediante un sistema *proof of work* o prueba de trabajo. Este mecanismo criptográfico resolvía dos problemas:

- Primero, proporciona un algoritmo de consenso simple y moderadamente eficaz, permitiendo que los nodos de la red, se pongan de acuerdo colectivamente, en un conjunto de actualizaciones perfectas del estado actual del libro contable de Bitcoin.
- Segundo,proporciona un mecanismo para permitir la entrada libre en el proceso de consenso, resolviendo el problema político de decidir quién va a influir en el consenso, y evitando al mismo tiempo los ataques de tipo Sybil.

Lo hace mediante la sustitución de dos barreras:

- La barrera formal para participar, es decir, el requisito de estar registrado como una entidad única en una lista particular.
- La barrera económica, el peso de un solo nodo en el proceso de votación por consenso es directamente proporcional a la potencia de cálculo de la que el nodo dispone.

Desde entonces, se ha propuesto un enfoque alternativo, llamado prueba de participación (proof of stake), en donde el peso de un nodo debe ser proporcional a su posesión de moneda y no a sus recursos computacionales; la discusión sobre los méritos relativos de los dos enfoques van más allá del alcance de este documento, pero cabe señalar que ambos pueden ser utilizados y servir como columna vertebral de una criptomoneda. [32]

3.3. Orígenes de Ethereum

En esta sección veremos la historia y la evolución que se ha seguido y que ha dado lugar a Ethereum.

3.3.1. Bitcoin

Bitcoin [9] es una moneda, como el Euro o el Dólar Estadounidense, que sirve para intercambiar bienes y servicios, sin embargo, esta es una **divisa electrónica** o **moneda virtual**, que presenta características novedosas y destaca por su eficiencia, seguridad y facilidad de intercambio.

A diferencia de otras monedas, se trata de una moneda *descentralizada*, es decir, que nadie la controla, ya que, Bitcoin no tiene un emisor central como tienen el Euro o los dólares, sino que está criptomoneda es producida por personas y empresas de alrededor del mundo que dedican gran cantidad de recursos a la minería de bloques.

El Bitcoin se basa en un sistema de transacciones *peer to peer*, en el que hay un *estado*, que representa el **status** de todas las transacciones de los dueños del dinero. Un ejemplo muy sencillo y fácil de entender es imaginarse Bitcoin como una gran pecera donde, desde fuera, puedes mover el pez que hay dentro si posees las claves necesarias para ello, pero **nunca nadie** podrá introducir o sacar más peces de dicha pecera, por lo que, será inalterable la cantidad total de peces de la pecera, pero si podrá cambiar la posesión de estos, quedando registrado en forma de transacciones dentro de los bloques. [23]

Una vez visto el ejemplo, vemos que de está forma, la posesión de los Bitcoin no implica tener una moneda virtual, sino la existencia de un listado de transacciones en el cual, la ultima persona es el actual beneficiario. En la siguiente imagen se puede ver un ejemplo del funcionamiento de las transacciones en Bitcoin y cómo estas van cambiando su estado mediante las transferencias de Bitcoin.

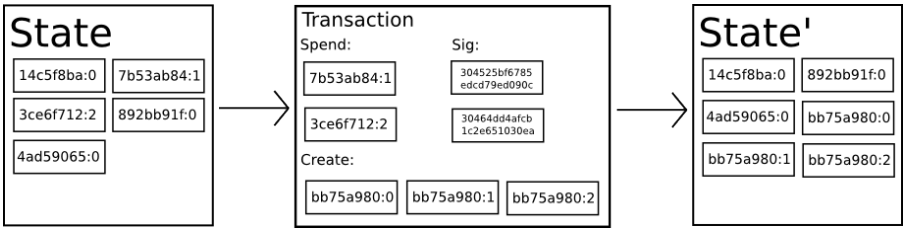


Figura 3.4: Funcionamiento de las transacciones en Bitcoin

En el ejemplo podemos observar, que hay un "estado "inicial (que consiste en el estado de la propiedad de todos los Bitcoin existentes) y una "función de transición de estado "que toma un estado (inicial) y una transacción y emite un nuevo estado, que es el resultado. [32]

El soporte de todo este proceso es el sistema *Blockchain* o *cadena de bloques*, en donde un registro público y accesible por todos los nodos de la red almacena el listado de todas las transacciones. Con ello se consigue que sea prácticamente imposible falsear una transacción, ya que, si alguien quiere engañar a un nodo para hacerle creer que tiene más dinero del que realmente posee, al sincronizarse la información que contiene dicha transacción, esta sería visible por el resto de nodos de la red y estos la rechazarían automáticamente al no poder demostrar la posesión de dicho dinero.

3.3.2. Altcoins

Bitcoin es un proyecto de código abierto, y su código ha sido utilizado como la base para muchos otros proyectos de software. La forma más común de software generado a partir del código fuente de Bitcoin son monedas alternativas descentralizadas, que utilizan los mismos bloques de construcción básicos para implementar las monedas digitales, que se conocen como Altcoins.

En Febrero de 2011 [11], con el aumento del valor de los Bitcoins, se tomo la decisión de crear una criptomoneda paralela con el objetivo de realizar pruebas sin poner en peligro la red de Bitcoin. Así surge Bitcoin Testnet [46], considerada la primera Altcoin. Bitcoin Testnet comparte las mismas características que Bitcoin, ya que, su objetivo es servir como una zona de pruebas. Actualmente estamos en la versión Testnet3.

Más tarde, en Abril de 2011 se creó **Namecoin** [28], con el objetivo de crear un sistema DNS descentralizado que usase la base de datos de Bitcoin directamente. El registro sin censura alguna de dominios *.bit* es la principal función de Namecoin. Este es un funcionamiento similar a .com o .net pero es totalmente alternativo e independiente de la ICANN, el organismo que controla los nombres de dominio de nivel superior [36].

Poco después, en Octubre de ese mismo año, se creo Litecoin [8], que es una criptomoneda similar a Bitcoin pero con algunas características diferentes, como pueden ser, un mayor número

de unidades, menos tiempo entre cada bloque y un algoritmo diferente para alcanzar el consenso entre los nodos de la red.

Desde entonces, han ido surgiendo nuevas criptomonedas, hasta llegar al número actual de 688 Altcoin [4]. La mayoría de estas criptomonedas son una copia de Bitcoin o Litecoin que han modificado alguna característica, como puede ser, el número de monedas en circulación o el tiempo que tarda en confirmarse una de las transacciones, pero hay algunas pocas que han innovado mediante la creación de nuevos algoritmos de minado y la inclusión de nuevas funcionalidades.

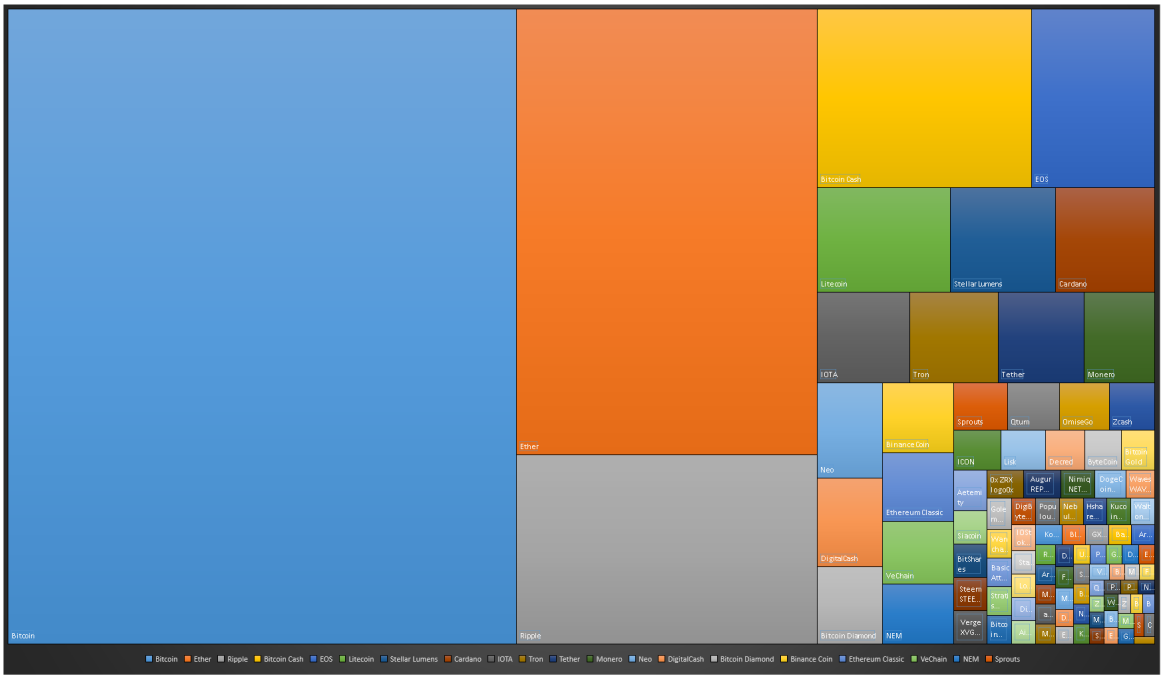


Figura 3.5: Representación de criptomonedas existentes por su volumen [19]

La idea principal que ha surgido con los Altcoins, han sido los diferentes métodos de alcanzar un consenso sobre el nodo siguiente en verificar la validez de las transacciones mediante la inclusión de este al Blockchain y la generación de criptomoneda.

A continuación vamos a ver las diversas formas y conceptos de como se verifican las transacciones y se generan bloques.

- ***Proof of Work***: Prueba de trabajo se define como una medida económica para evitar los ataques de denegación de servicio y otros abusos del servicio, como el correo no deseado en una red, requiriendo algún trabajo del solicitante del servicio, lo que generalmente significa tiempo de procesamiento en una computadora. Monedas como Bitcoin usan Prueba de trabajo.

  Puesto simplemente en términos de criptomoneda, esto significa que los nodos de red requieren cálculos para formar un *libro mayor distribuido*. Este proceso se llama *minería*.

  Un *libro mayor distribuido*, es un sistema descentralizado que puede acordar y realizar un seguimiento del monto correcto que tiene una billetera al proporcionar un historial de cada transacción.

Debido a la naturaleza de cálculo extenuante de Prueba de trabajo, puede convertirse en un proceso costoso. [48]

- ***Proof of Stake***: Proof of Stake es una alternativa a Prueba de trabajo,es un protocolo de consenso distribuido para redes distribuidas, que asegura una red de una criptomoneda, mediante la petición de pruebas de posesión de dichas monedas. Con PoS la probabilidad de encontrar un bloque de transacciones, y recibir el premio correspondiente, es directamente proporcional a la cantidad de monedas que uno tiene acumuladas (evitando así que la confianza venga dada por la cantidad de trabajo invertida).

  Se basan en la suposición de que quienes poseen más unidades de una moneda basada en PoS, están especialmente interesados en la supervivencia y el buen funcionamiento de la red que otorga valor a dichas monedas, y por tanto, son ellos los más indicados para cargar con la responsabilidad de proteger al sistema de posibles ataques. Es por eso que el protocolo los premia con una menor dificultad para encontrar bloques (es inversamente proporcional al número de monedas que demuestren poseer).

  - ***Selección de bloques aleatorizados***: Nxt y Blackcoin usan aleatorización para predecir el siguiente creador de bloque. Lo hacen utilizando un algoritmo para buscar el valor de hash más bajo en combinación con el tamaño de la apuesta. Como todas las apuestas son públicas, los nodos pueden predecir qué apuesta creará el próximo bloque.
  - ***Selección de edad de monedas***: La edad de la moneda se refiere a la edad de las entradas de la transacción.

    La edad de la moneda es igual a la cantidad de monedas enviadas por la edad promedio de estas monedas. La edad se mide en días. La edad se restablece a cero cada vez que se envía una moneda Y cada vez que una moneda proporciona una firma. La edad de la moneda se puede usar para calcular tarifas obligatorias, bloquear recompensas o corregir metas.

    Las monedas no gastadas deben esperar 30 días antes de que puedan comenzar a competir para generar el siguiente bloque. Monedas como Novacoin y Peercoin utilizan este método.

    La recompensa por replanteo provendrá de monedas nuevas generadas al inflar el suministro actual, llamado acuñación, o pueden provenir de tarifas de transacción recicladas, llamadas falsificación.

Variaciones de Proof of Stake

- ***Proof of Stake Anonymous (PoSA)***: Presentado por primera vez por Cloakcoin , las transacciones son encubiertas por otros usuarios que reciben una recompensa por ayudar a la anonimización de la transacción. Otros usuarios proporcionan entradas y salidas a la transacción y hacen imposible determinar la fuente y el destino de la transacción.
- ***Delegated Proof of Stake (DPoS)***: Delegated Proof of Stake se vio por primera vez en Bitshares blockchain. La forma en que funciona es que los usuarios voten por "delegados"a quienes se les da el poder de obtener ganancias al ejecutar un nodo

completo. Se supone que este método es más eficiente y protege a los usuarios de interferencias reguladoras no deseadas.

- ***Proof of Importance (POI)***: Se introdujo una expansión de NEM llamada Prueba de Importancia para promover la actividad económica. A cada cuenta se le asigna un puntaje de importancia que representa su importancia agregada para la economía. Este método ayuda a garantizar que todas las computadoras de la red estén de acuerdo entre sí y puede evitar que las personas gasten monedas que no tienen. Los usuarios que son "importantes"pueden "cosechar"y ganar recompensas.
- ***Proof of Storage***: Prueba de almacenamiento se formuló por primera vez en 2013 . Monedas como Storj usan Prueba de almacenamiento. En lugar de usar una cadena de bloques, la red usa un árbol de bloques. Además, en lugar de ver todas las transacciones enumeradas, el usuario solo verá las transacciones que sean relevantes para ellas. Cada nodo en el árbol de bloques contiene una cadena de bloques. A partir de ahora, no se conoce ninguna prueba práctica de verificación verificable públicamente, y no existe un esquema conocido para verificar de forma independiente que se haya emitido o respondido una auditoría privada verificable según lo reclamado.
- ***Proof of Stake Time (PoST)***: El primero en usar esto es Vericoin . La prueba del tiempo de apuesta usa la edad de la moneda, pero en vez de usar la cantidad de monedas para calcular la edad, usan el período de tiempo que las monedas se han mantenido en la dirección específica. Este método se implementó para evitar que los ricos sean más ricos, lo que hacen muchos métodos de Prueba de Stake.
- ***Proof of Stake Velocity (PoSV)***: Reddcoin es el primero en introducir este método. La prueba de Stake Velocity recompensa a los usuarios en función de la cantidad de monedas que tienen y la forma en que las usan activamente.

- ***Proof of Activity***: El comprobante de actividad (POA) se propuso por primera vez en 2012 como una alternativa al comprobante de participación. Es un método que complementa la prueba de trabajo y ayuda a prevenir un ataque del 51 por ciento, que es cuando un usuario o grupo controla el 51 por cierto o más del hashrate de minería de una red. Prueba de actividad, en resumen, selecciona un par aleatorio de la red para firmar un nuevo bloque. Este método requiere un intercambio continuo de datos. Para reducir el tráfico, la "plantilla"de bloques no incluye la lista de transacciones y, en cambio, la agrega el último firmante.

- ***Proof of Burn (PoB)***: Prueba de quemadura es exactamente lo que se llama. Está proporcionando pruebas de que ha quemado algunas de sus monedas en el proceso de enviar una transacción a una dirección que no es confiable. Este método solo funciona con monedas extraídas de criptomonedas de prueba de trabajo. Los usuarios intentarán quemar la mayor cantidad de monedas para "ganar"la recompensa del bloque. La mayoría de las veces se ha introducido la Prueba de Quemadura para sembrar otras monedas al destruir el valor de una.

- ***Proof of Capacity (PoC)***: Las monedas criptográficas que utilizan la Prueba de capacidad, también conocida como Prueba de espacio, usan la Minería de disco duro para validar nuevos bloques. Burst Coin fue el primero en introducir este concepto. Los mineros de Prueba de trabajo queman recursos, mientras que Prueba de capacidad le permite

usar el espacio asignado en su disco duro para minar. Se usa un algoritmo para crear fragmentos de datos denominados gráficos al mezclar claves públicas de forma repetida. Mientras más espacio tenga, más probabilidades tendrá de explotar un bloque.

- ***Proof of Checkpoint (PoC)***: Prueba de Checkpoint es un sistema híbrido que utiliza cualquier sistema de Proof of Stake con un sistema de prueba de trabajo. La idea de este concepto es mitigar los ataques al sistema Proof of Stake; sin embargo, todavía está sujeto a un ataque en un nodo que ha estado fuera de línea durante un período de tiempo prolongado y, a su vez, puede utilizarse para proporcionar información falsa sobre el blockchain.

  Cada x cantidad de bloques en el sistema de Prueba de Estaca requiere un bloque de Prueba de Trabajo para ser extraído. Cada bloque de prueba de trabajo no contiene transacciones y están directamente vinculados tanto a la red de prueba de trabajo como a la red de prueba de participación. [34]

3.3.3. Bitcoin 2.0

Bitcoin y su desarrollo están creando muchísimo vocabulario propio del ecosistema, como por ejemplo el termino de cadena de bloques que nació para llevar un registro de todas las transacciones realizadas y evitar los dobles gastos del dinero.

El término o etiqueta **Bitcoin 2.0**, que con frecuencia nos encontramos en el mundo de las criptomonedas, se refiere a los nuevos avances conceptuales e implementaciones experimentales en el desarrollo que extienden de manera significativa la flexibidad, potencia y utilidad última de la plataforma subyacente de Bitcoin o las criptomonedas y están así cruzando la barrera de usos más álla de Bitcoin como forma de pago para adentrarse en Bitcoin como protocolo, algo que extenderá el uso del ecosistema criptográfico. [21]

La primera moneda que surgió para ampliar las funcionalidades de la cadena de bloques, fue Namecoin, como hemos visto anteriormente, con el objetivo de crear un servicio de DNS descentralizado. Namecoin utilizá la cadena de bloques para almacenar la información sobre los dominios y el historial de registros, aunque la cadena de bloques puede utilizarse para almacenar cualquier tipo de información. Por ejemplo, Onename [37] almacena objetos JSON con un perfil público de sus usuarios y así crea un sistema de identificación distribuido. Gracias a la idea de Namecoin de utilizar la cadena de bloques como una base de datos distribuida, surgen proyectos interesantes, como Proof of Existence y BTProof, que funcionan como un notario digital, utilizando la cadena de bloques para certificar y probar la existencia de un documento en un momento determinado. Utilizan un hash del documento y lo almacenan en la cadena de bloques mediante una transacción.

Así es como al final surgió el término de Metacoin, que son aplicaciones que usan la cadena de bloques para ampliar sus funcionalidades mediante la inclusión de metadatos.

Algunos desarrolladores han comenzado a crear nuevos protocolos que funcionan como una capa por encima de Bitcoin, con esto se consiguen usos más avanzados de la cadena de bloques. Mastercoin fue uno de los primeros de estos desarrollos y permitió la emisión e intercambio de monedas con diferentes características de manera descentralizada. Aunque Mastercoin no tuvo el éxito que se esperaba y ahora se conoce el proyecto bajo el nombre de Omni [35] se ha centrado en la idea de **Smart Property** que se utiliza para representar activos mediante monedas o transferirlo a través de la cadena de bloques.

Estás aplicaciones resolvían problemas concretos, pero era necesario crear un Altcoin adicional y sus propias reglas de funcionamiento, lo que implicaba que los usuarios tuvieran que hacer uso de un conjunto de Altcoins y protocolos diferentes. Para solucionar esto, en la actualidad han surgido diferentes proyectos que pretenden crear una plataforma de desarrollo de aplicaciones descentralizada basada en la cadena de bloques bajo un mismo protocolo, empleando el término de **Smart Contract** o **Contrato Inteligente**. [44]

Figura 3.6: Representación de Smart Contract [44]

Un contrato inteligente se puede definir como, un acuerdo contractual implementado mediante software. A diferencia de un contrato tradicional, donde ambas partes tienen que recurrir al sistema legal para asegurarse de su cumplimiento, un contrato inteligente es capaz de autoejecutarse una vez se hayan cumplido las condiciones establecidas en él. Por la forma en que ha sido desarrollado Bitcoin, es posible incorporar código a una transacción mediante un lenguaje de Script que dicte bajo que condiciones debe o no ejecutarse una transacción. Proyectos como Counterparty [18] utilizan esta característica dentro de Bitcoin, mientras otros como **Ethereum** han decidido crear su propia cadena de bloques y así ampliar este lenguaje Script para facilitar el desarrollo de contratos inteligentes a los desarrolladores.

En este contexto, surge el término de DAO ()Descentralized Autonomous Organitation), que es una organización que está dirigida a través de reglas codificadas en programas de ordenador llamados contratos inteligentes. Un registro de transacciones financieras de una DAO, así como dichas reglas, están gestionadas a través de un Blockchain. Hay varios ejemplos de este modelo empresarial, como por ejemplo, el ejemplo más famoso es *The DAO*, una DAO para fondos de capital riesgo, que se puso en marcha con $150 millones en crowdfunding en Junio de 2016 y el cual fue inmediatamente pirateado y despojado de $50 millones de dólares americanos en criptomonedas. Dicho hackeo fue revertido unas semanas después, y el dinero fue recuperado al completo, gracias a una versión del Blockchain de Ethereum. Este rescate descentralizado fue posible gracias a la mayoría de votos en la tasa de hash del Blockchain. [38]

Otros ejemplos de DAO son Counterparty [47], que es una plataforma que amplía la funcionalidad de Bitcoin al “escribir en los márgenes”de las transacciones regulares de Bitcoin, abriendo la puerta a la innovación y las funciones avanzadas que no son posibles con el software

ordinario de Bitcoin, y otro ejemplo es Ethereum [15], que es una plataforma descentralizada que ejecuta contratos inteligentes: aplicaciones que se ejecutan exactamente como se programaron sin posibilidad de tiempo de inactividad, censura, fraude o interferencia de terceros.

Aunque las dos buscan un mismo enfoque hay diferencias, por ejemplo, Counterparty emplea la cadena de bloques de Bitcoin, mientras que Ethereum se ejecuta en su propia Blockchain, esto hace que Counterparty parta con una ventaja ya que tiene más usuarios potenciales, pero cuenta con ciertas desventajas, como sus limitaciones en el lenguaje Script o el limitado tamaño del bloque, además de que al hacer uso de la cadena de bloques de Bitcoin se ve limitada su escalabilidad. Ethereum por el contrario, aprovecha la creación de su propia Blockchain lo que le permite incorporar un lenguaje Script más avanzado, un mayor tamaño de bloque y un algoritmo diferente de hash para el minado que permite bloques cada 12 segundos de media, frente al tiempo medio de Bitcoin que crece de forma logarítmica.Además de la cadena de bloques, incluye dos protocolos para el intercambio de ficheros llamado Swarm [31] y otro de mensajes llamado Whisper [53] para crear un entorno completo de desarrollo.

3.3.4. Dapps

Antes de hablar de qué son las Dapps, tenemos que ver porque están surgiendo.

Si preguntamos a varias personas qué son Whatsapp , Instagram, Facebook, etc, nos contestarán lo mismo, que son aplicaciones o apps. Como sabemos una app es una aplicación informática diseñada para ser ejecutada en smartphones, tablet y otros dispositivos. Estas apps permiten a las personas efectuar una tarea concreta de cualquier tipo, ya sea profesional, de ocio, educativa, etc.

Hoy en día el mundo de las aplicaciones no es algo futurista, sino que está muy presente en nuestras vidas. Aplicaciones como Twitter, Wallapop o Dropbox invaden nuestros dispositivos móviles con la intención de poner sus servicios al alcance de la mano.

En algunos casos, estás aplicaciones también existen en el mundo Web. De hecho, Facebook o Amazon empezaron así antes de que pudieramos disfrutar de sus apps en el móvil.

Sin embargo, existe un problema común en todas estas apps a las que tenemos acceso, y es que son centralizadas.

Por ejemplo, si un día a Mark Zuckerberg (creador de Facebook), decide poner un precio de suscripción a su plataforma, el que quiera usarla tendrá que abonar dicha cantidad o, de lo contrario, no podrá acceder a la aplicación. Es decir, las aplicaciones centralizadas dependen directamente de un ente central (normalmente una empresa), que puede decidir cualquier cosa sobre dichas plataformas sin necesidad de tener en cuenta o preguntar a sus usuarios.

Esto no ocurre con las Dapps.

Pero, ¿qué es una Dapps?, se puede decir, que una Dapp es una aplicación descentralizada, es decir, una app que no depende de un sistema central, sino que depende de la comunidad de usuarios que la utilizan.

Está puede ser una app móvil o una aplicación web que interactúa con un contrato inteligente para llevar a cabo su función.

Vamos a ver una pequeña comparación entre aplicaciones web tradicionales con las novedosas Dapps para entender mejor el funcionamiento de estas.

En la arquitectura de las aplicaciones, tanto centralizadas como descentralizadas, existe lo que llamamos la parte cliente (conocido como Frontend) y la parte servidor (conocido como Backend).

Cuando abrimos una aplicación móvil o web, podemos decir, que lo que vemos es el Frontend, que corresponde a la interfaz gráfica de la app y al código informático que tiene tras de sí.

Por otra parte, la parte de Backend, es la que no se ve, es decir, todas las bases de datos, la capacidad de almacenamiento, etc.

Ambas partes (cliente y servidor o fontend y backend) interactúan entre sí constantemente enviándose datos para que la aplicación funcione según se construyó, es decir, para que cumpla sus funciones.

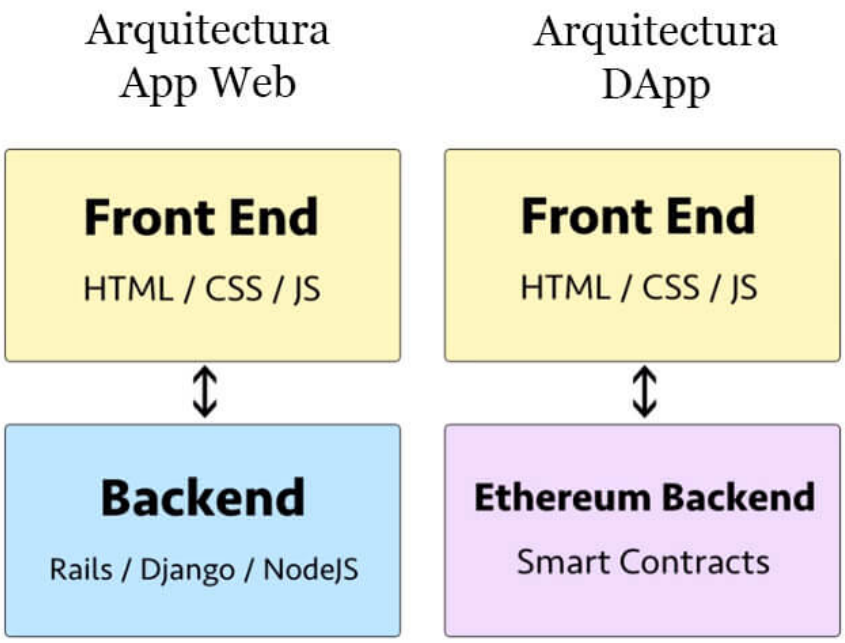


Figura 3.7: Representación de una arquitectura App vs Dapps [1]

Por ejemplo, cuando entramos a Facebook desde la aplicación móvil, podemos ver a todos nuestros amigos, sus perfiles, etc, pero nada de eso está guardado en la aplicación que nos hemos descargado, si fuera así, ni siquiera podríamos descargar la aplicación por el gran tamaño que supondría.

Para poder ver esa información, Facebook necesita acudir a sus servidores (el backend), que es donde reside toda esa información.

Facebook mandará toda esa información desde su backend a su frontend para que nosotros podamos visualizarla.

Comprender esta gran diferencia nos ayudara a entender el funcionamiento de las Dapps.

Ventajas de las Dapps

Al realizar una aplicación descentralizada basada en el entorno Ethereum, nos aporta bastantes ventajas frente a las aplicaciones web tradicionales:

- **Procesamiento de pagos y cobros:** En las Webs tradicionales solemos encontrar pasarelas de pago o integraciones mediante PayPal para poder recibir pagos de los usuarios que navegan a través de ella.

En una Dapps no es necesario hacer integraciones adicionales, ya que, es posible para el usuario el enviar o recibir fondos, en forma de Ether, de una forma directa, sin la figura intermediaria que sería Visa, Mastercard o PayPal, por ejemplo, y con comisiones prácticamente nulas.

- **Cuentas de usuario:** Estamos acostumbrados al hecho de tener que registrarnos prácticamente en cada web de Internet. El tener que crearnos muchas cuentas de usuario con contraseñas y nombres de usuario diferentes puede hacer, que con el paso del tiempo, se nos olviden y tengamos que recurrir al proceso de recuperar contraseña.

  Esto con las Dapps no ocurre, ya que, los usuarios no necesitan registrarse.

  Al crearte una sola cuenta con su clave pública y su clave privada, como en el caso de las Wallets [27], que contienen sus datos, puedes vincularla con cualquier Dapps.

  Para que podamos visualizar este proceso, el registro de una Dapps funciona de forma parecida a como funcionan los inicios de sesión con las cuentas de Google, Facebook o Linkedin: puedes usar tu cuenta creada en estas redes sociales para registrarte en otros sitios web sin necesidad de poner tus datos otra vez.

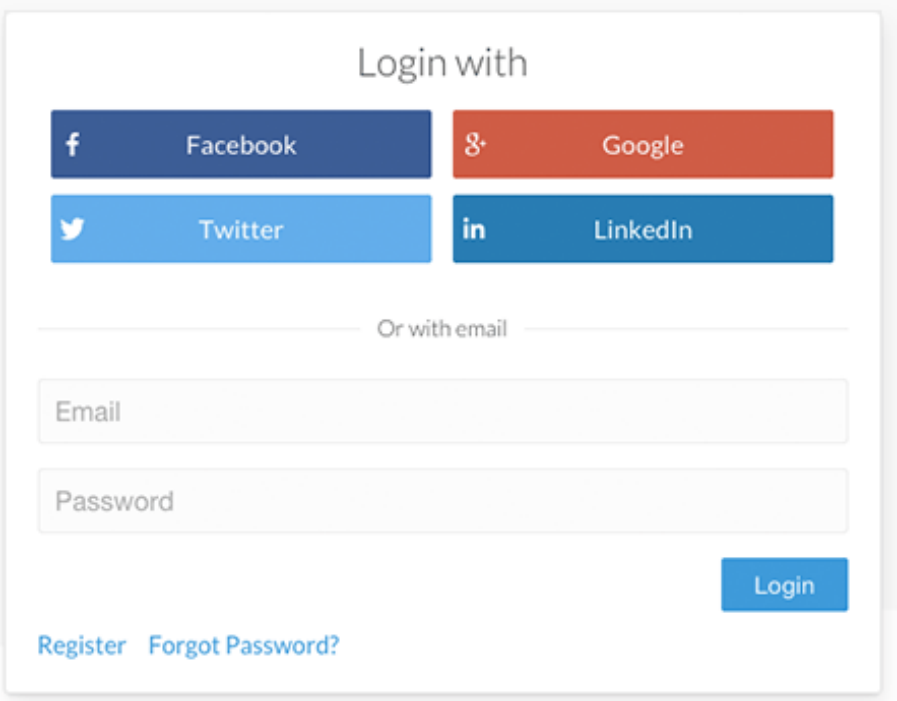


Figura 3.8: Inicio de sesión desde otras redes sociales [1]

Simplemente debemos recordar que estos sistemas son centralizados, lo que quiere decir, que si un día, por ejemplo Twitter deja de existir, si nos hemos registrado en alguna web con nuestra cuenta de Twitter, ya no podremos acceder a esa web y nos tendríamos que crear otra cuenta nueva.

Eso es algo que no ocurriría en las Dapps.

- **Bases de datos:** en el sistema tradicional, los datos son almacenados a través de discos duros, ya sean personales o en servidores externos, mediante servicios en la nube.

Ambas son opciones que tienen sus riesgos, en el caso de los discos duros personales, estos pueden ser hackeados y los datos saldrían a la luz; en el caso de los servicios de la nube, nuestra cuenta puede ser hackeada también, si se hackea a la empresa que proporciona ese servicio, como ha ocurrido ya en incontables ocasiones.[30]

Además, si esta empresa desaparece, nuestros datos también lo harán.

Sin embargo, con las Dapps, el almacenar datos en una Blockchain hace que estos datos permanezcan inmutables, es decir, una vez que se registran esos datos ya no se pueden borrar.

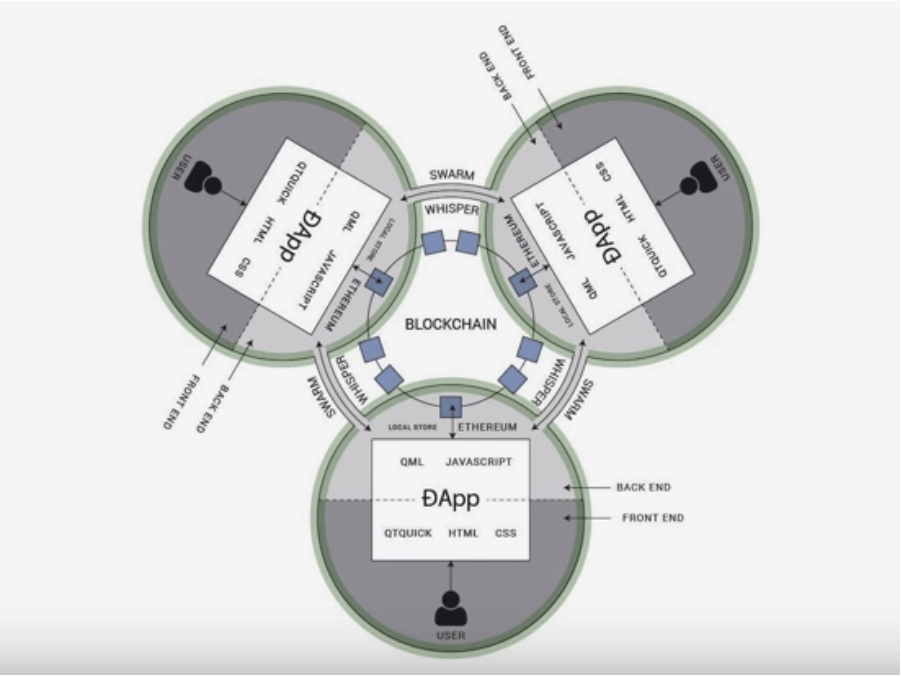


Figura 3.9: Funcionamiento de una Dapps [1]

Los datos permanecen en la cadena de bloques de una forma encriptada, es decir, son ilegibles para cualquier persona excepto para sus propietarios.

Además, el carácter distribuido de la cadena de bloques, hace que esos datos residan en cada ordenador de la red Ethereum, por lo que si desaparecieran de un ordenador, existen muchas otras copias de seguridad.

Como único punto negativo observamos que almacenar una gran cantidad de datos en la cadena de bloques puede resultar bastante costoso y además aumenta de forma notable el tamaño de la misma, aunque ya se está trabajando para mejorarlo y encontrar soluciones escalables.

- **Confianza:** Cuando usamos una aplicación web, podemos ver el código que se ha usado a través de las herramientas de inspección del navegador. De esta forma, el usuario puede verlo desde el frontend. Sin embargo, la interacción del frontend con el backend es algo que no podemos ver a simple vista.

  Con las Dapps, los usuarios pueden estar tranquilos, ya que, pueden inspeccionar el código del contrato inteligente basado en Ethereum a través de su identificador.

En la siguiente imagen vemos un contrato inteligente en la Blockchain Mainnet de Ethereum.

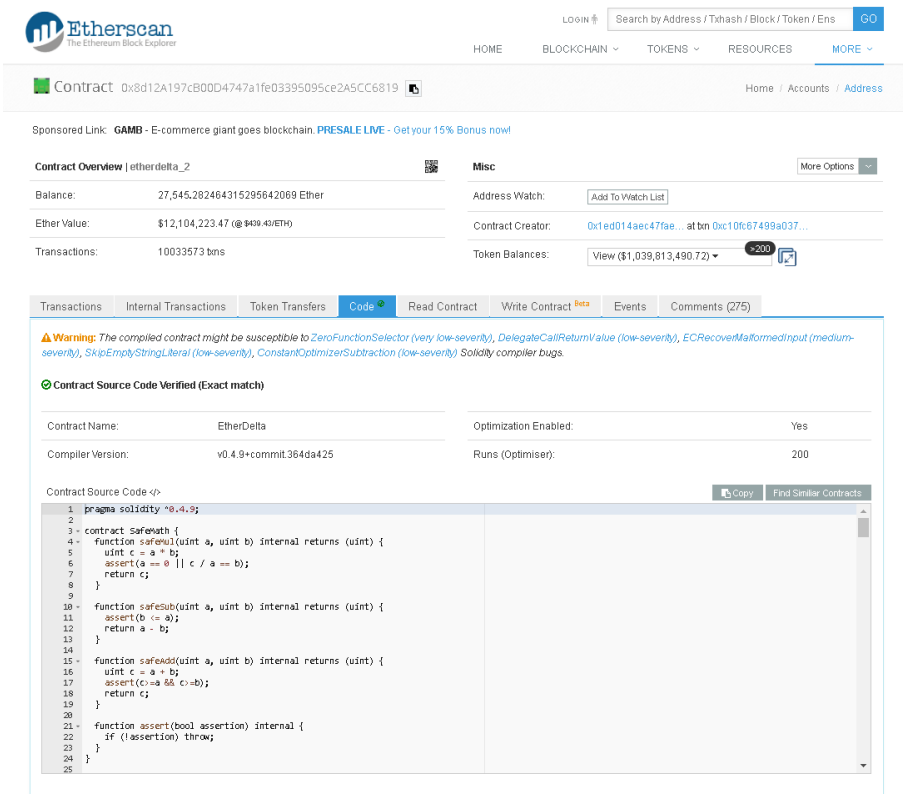


Figura 3.10: Contrato inteligente en el Mainnet de Ethereum

De esta manera se puede verificar el código fuente y ver si tiene fallos de escritura o seguridad, enviando correcciones para que no se puedan robar fondos o información depositada en la Dapps. Esto hace crecer el sentimiento de confianza y seguridad de los usuarios.

Características de una Dapps

- **Descentralización:** Lo primero y más importante de todo, una Dapps tiene que ser descentralizada, es decir, tiene que funcionar de forma autónoma sin que ninguna entidad la controle, dejando todo el poder de decisión sobre la misma en su comunidad de usuarios.
- **Código abierto (Open Source):** una Dapp tiene que ser 100 % código abierto. Esto significa que el código fuente bajo el que está programada la Dapp está abierto a posibles modificaciones y mejoras por parte de los usuarios, todo lo contrario que la gran mayoría de aplicaciones que se utilizan hoy en día, en las que sólo los programadores dentro de la empresa pueden modificar y siempre bajo la supervisión de sus jefes.

  Esas mejoras propuestas por parte de la comunidad que usan la Dapp deben decidirse por consenso de la gran mayoría antes de hacerse efectivas.
- **Blockchain:** Los datos y registros del funcionamiento de la Dapp deben ser almacenados de forma criptográfica a través de un Blockchain público, para así añadir transparencia y seguridad como cualidades de la aplicación descentralizada.

- **Protocolo:** Si la Dapp está basada en Blockchain, eso significa que la información de las operaciones realizadas dentro de la aplicación tiene que ser almacenadas en bloques y estos tienen que ser verificados.

  Esto se da de acuerdo con un protocolo que actúe como prueba de que esas verificaciones son llevadas a cabo.

  Este protocolo puede estar basado en el algoritmo PoW o en PoS, vistos anteriormente.

Tipos de Dapps

Para esta clasificación nos basaremos en base a si poseen su propia Blockchain o utilizan la Blockchain de otra Dapps.

- **Aplicaciones descentralizadas de tipo I**
  Estas son las que tendrían su propia Blockchain independiente. En este caso, y como hemos visto ya, Ethereum sería una de esas Dapps, aunque la más famosa dentro del mundo de las criptomonedas sea Bitcoin. Litecoin, Dash, Monero y muchas otras Altcoin también entrarían en esta clasificación.

- **Aplicaciones descentralizadas de tipo II**
  La característica principal de las Dapps de tipo II es que utilizan la Blockchain de una aplicación descentralizada de tipo I en vez de tener ellas una propia. Este tipo de Dapps son protocolos que funcionan ya sea con sus propios tokens o con los tokens de la Blockcahin en la que operan.

  Un ejemplo de este tipo de Dapps sería Omni Layer. Esta aplicación descentralizada está construida sobre la cadena de bloques de Bitcoin y es una plataforma que sirve para la creación y el comercio de activos digitales y criptomonedas. Al actuar sobre la cadena de bloques de Bitcoin, las transacciones de Omni son también transacciones de Bitcoin.

  Otro ejemplo sería Raiden Network [49], este basado en la cadena de bloques de Ethereum. Esta plataforma ofrece una solución de escalabilidad dentro de la red de Ethereum a la hora de permitir pagos casi instantáneos y de bajo coste. La idea principal de Raiden Network, es aprovechar una red de canales de pago que permitan transferir de forma segura el valor sin necesidad de implicar a la Blockchain de Ethereum en cada transferencia, lo cual multiplicaría su velocidad.

- **Aplicaciones dencentralizadas de tipo III**
  Por último, las Dapps de tipo III son las que utilizan el protocolo de una aplicación descentralizada de tipo II como las que acabamos de ver.

  Estas aplicaciones, también funcionan con sus propios tokens digitales o bien con los de las Dapps en las que se basan, al igual que pasaba en las Dapps de tipo II.

  Una Dapp de tipo III podría ser Safe Network [50], que utiliza el protocolo de Omni Layer para emitir su propia criptomoneda, el Safecoin, que a se vez puede utilizarse para adquirir almacenamiento distribuido de archivos, por ejemplo.

3.3.5. Ethereum

Ethereum es una plataforma abierta de Blockchain que permite a cualquier persona crear y utilizar aplicaciones descentralizadas que funcionan con la tecnología Blockchain. Al igual que Bitcoin, nadie controla ni posee Ethereum, es un proyecto de código abierto construido por muchas personas en todo el mundo. Pero a diferencia del protocolo de Bitcoin, Ethereum fue diseñado para ser adaptable y flexible. Es fácil crear nuevas aplicaciones en la plataforma Ethereum, y con el lanzamiento de Homestead, ahora es seguro para cualquiera usar esas aplicaciones. [52]

En 2014, los fundadores de Ethereum, Vitalik Buterin, Gavin Wood y Jeffrey Wilcke, comenzaron a trabajar en una cadena de bloques de próxima generación que tenía la ambición de implementar una plataforma de contratos inteligentes general, totalmente confiable.

Ethereum incorpora muchas características y tecnologías que serán familiares para los usuarios de Bitcoin, a la vez que introduce muchas modificaciones e innovaciones propias.

Mientras que la cadena de bloques de Bitcoin era puramente una lista de transacciones, la unidad básica de Ethereum es la cuenta . El Blockchain de Ethereum rastrea el estado de cada cuenta, y todas las transiciones de estado en la cadena de bloques de Ethereum son transferencias de valor e información entre cuentas.

Ethereum usa su propia criptomoneda llamada “ether ”[24]. Para la creación de las aplicaciones descentralizadas lo primero que hay que hacer es escribir el contrato inteligente mediante el código y una vez esté escrito y probado, hay que subirlo a la cadena de bloques, donde el contrato inteligente tendrá una dirección única que le identifique desde la cual se puede interactuar con él.

Los smart contracts son abstracciones de programación de alto nivel que se compilan en bytecode EVM y se implementan en el blockchain de Ethereum para su ejecución. Se pueden escribir en Solidity (una biblioteca de lenguajes con similitudes con C y JavaScript), Serpent (similar a Python), LLL (un lenguaje de bajo nivel tipo Lisp) y Mutan (basado en Go, pero obsoleto). También se está desarrollando un lenguaje orientado a la investigación llamado Viper (un lenguaje decidible derivado de Python).

Cuentas Ethereum

Como hemos dicho en el apartado anterior, la unidad básica de Ethereum es la cuenta. Hay dos tipos de cuentas:

- **Cuentas de propiedad externa (EOA):** controladas por claves privadas.
- **Cuentas de contrato:** controladas por su código de contrato y solo pueden ser “activadas ”por un EOA.

Para la mayoría de los usuarios, la diferencia básica entre estos es que los usuarios humanos controlan los EOA, ya que pueden controlar las claves privadas que dan control sobre un EOA. Las cuentas de contrato, por otro lado, se rigen por su código interno. Si son “controlados ”por un usuario humano, es porque están *programados* para ser controlados por un EOA con una dirección determinada, que a su vez está controlada por quien tenga las claves privadas que controlan ese EOA. El término popular “contratos inteligentes ”se refiere al código en una cuenta de contrato: programas que se ejecutan cuando se envía una transacción a esa cuenta. Los usuarios pueden crear nuevos contratos implementando código en Blockchain.

Las cuentas de contrato solo realizan una operación cuando se lo indica un EOA. Por lo tanto, no es posible que una cuenta de contrato realice operaciones nativas, como la generación de números aleatorios o las llamadas API, solo puede hacer estas cosas si se lo solicita un EOA. Esto se debe a que Ethereum requiere que los nodos puedan ponerse de acuerdo sobre el resultado de la computación, lo que requiere una garantía de ejecución estrictamente determinista.

De esta manera se consigue que usuarios y contratos se comporten de una manera indistinguible en la red de Ethereum. Para que exista comunicación entre cuentas existen las transacciones.

Transacciones

El término "transacción "se utiliza en Ethereum para referirse al paquete de datos firmado que almacena un mensaje para ser enviado desde una cuenta de propiedad externa a otra cuenta en la cadena de bloques.

Las transacciones contienen:

- El destinatario del mensaje.
- Una firma que identifica al remitente y prueba su intención de enviar el mensaje a través del Blockchain al destinatario.
- **VALUE**, es la cantidad de WEI (la unidad más pequeña de Ether) para transferir del remitente al destinatario.
- Un campo de datos opcional, que puede contener el mensaje enviado a un contrato.
- **STARTGAS**, es el valor que representa la cantidad máxima de pasos computacionales que la ejecución de la transacción puede tomar.
- **GASPRICE**, es el valor que representa la tarifa que el remitente está dispuesto a pagar por el gas. Una unidad de gas corresponde a la ejecución de una instrucción atómica, es decir, un paso computacional.

De esta manera se consigue que si un usuario quiere usar la red Ethereum para subir contratos a la cadena de bloques, tiene que conseguir ether, por lo tanto tiene que minar, lo que contribuye al funcionamiento de toda la red.

.

Capítulo 4

Metodología, Tecnología y Herramientas utilizada

En este capítulo se va a presentar la metodología y el plan de trabajo elegido para llevar a cabo el proyecto. Y se dará una pequeña descripción de las tecnologías y herramientas empleadas para desarrollar la aplicación.

4.1. Plan de trabajo

Debido a la novedad que supone el proyecto, se utilizará un modelo de desarrollo híbrido, ya que, vamos a combinar el modelo en cascada, añadiendo una parte de prototipado para poder hacer frente a la evolución que tiene esta tecnología. Para ello, seguiremos un conjunto de fases de manera secuencial, estableciendo en cada una de ellas un conjunto de metas, así como un conjunto de actividades, realizando una única iteración de cada una de ellas. Las fases llevadas a cabo han sido:

- **Investigación:** durante esta fase se obtendrá el contexto necesario sobre la tecnología y su lenguaje de programación para adentrarnos en esta nueva tecnología y conocer la forma de trabajar con ella.
- **Análisis:** durante esta fase se planteará como deben de funcionar las implementaciones, que actores van a actuar con la aplicación y qué resultados se deben obtener, así como los requisitos que se deben cumplir.
- **Programación:** en esta fase se inicia la implementación del código, empezando con pequeños programas para aprender como funciona internamente Ethereum. Una vez conocido el funcionamiento de la plataforma, se pasara a realizar el backend y una vez tengamos el backend se realizara el frontend.

  *Durante está fase nos hemos encontrado con pequeños problemas, ya que, el lenguaje de programación de los contratos inteligentes, Solidity, esta en constante desarrollo y había a veces que actualizaban partes y dejaba de funcionar nuestro proyecto. Para solucionarlo, se realizaron pequeños prototipos en ciclos cortos que se iban mejorando, así según evolucionaba Solidity, podíamos encontrar los fallos e ir consiguiendo el prototipo final.
- **Pruebas:** en esta fase se comprobará que los resultados obtenidos de las implementaciones realizadas son los esperados en el análisis y corroborar que la implementación y el funcionamiento se realiza de forma correcta.

4.2. Tecnologías utilizadas

Debido a la naturaleza de la tecnología, se ha intentado realizar el proyecto bajo un enfoque de software libre, esto es debido, a que Ethereum es una plataforma para promover las aplicaciones distribuidas en Internet excluyendo servidores centralizados y que es de código abierto, lo más lógico es intentar que todo el software desarrollado sea libre.

El proyecto se puede dividir en 2 partes, **un contrato inteligente** que funciona como backend y **una interfaz web** que funciona como frontend.

Para el desarrollo del contrato se han utilizado las herramientas que nos facilita Ethereum (lenguaje de programación, tecnologías y aplicaciones). Y para la interfaz web se han utilizado tecnologías web del lado cliente (HTML, CSS, JavaScript y Bootstrap). La conexión entre

el contrato inteligente y la interfaz web se realiza a través de una librería (Web3) que nos proporciona Ethereum.

4.2.1. Ethereum

Lo que es Ethereum lo hemos visto en la sección anterior. Nosotros lo utilizaremos para desarrollar una aplicación descentralizada y segura de compra/venta de una forma sencilla.

Se utiliza el Blockchain de Ethereum para que nuestro contrato sea distribuido por toda la red, para que cualquier persona que este en ella pueda interactuar con nuestro contrato.

4.2.2. Ganache

Ganache es una cadena de bloques personal para el desarrollo de Ethereum que se puede usar para implementar contratos inteligentes, desarrollar sus aplicaciones descentralizadas y ejecutar pruebas. [54]

Está disponible tanto como una aplicación de escritorio como en una herramienta de línea de comandos (anteriormente conocida como TestRPC).

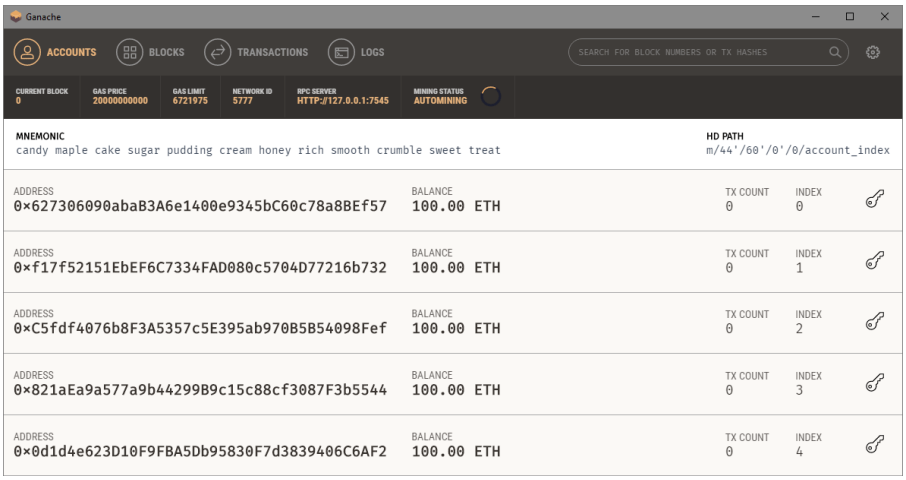


Figura 4.1: Funcionamiento de Ganache aplicación de escritorio [54]

Podemos observar en la *figura 3.1* que hay 4 pestañas disponibles:

1. La pestaña **Accounts** (Cuentas), muestra las cuentas generadas y sus saldos disponibles, por defecto es de 100 Ether. Es la página que sale por defecto.

2. La pestaña **Blocks** (Bloques), muestra cada bloque extraído de la cadena de bloques, junto con el gas utilizado y las transacciones.

3. La pestaña **Transations** (Transacciones), se enumeran todas las transacciones ejecutadas contra la cadena de bloques.

4. La pestaña **Logs** (Registros), muestra los registros para el servidor, útil para la depuración.

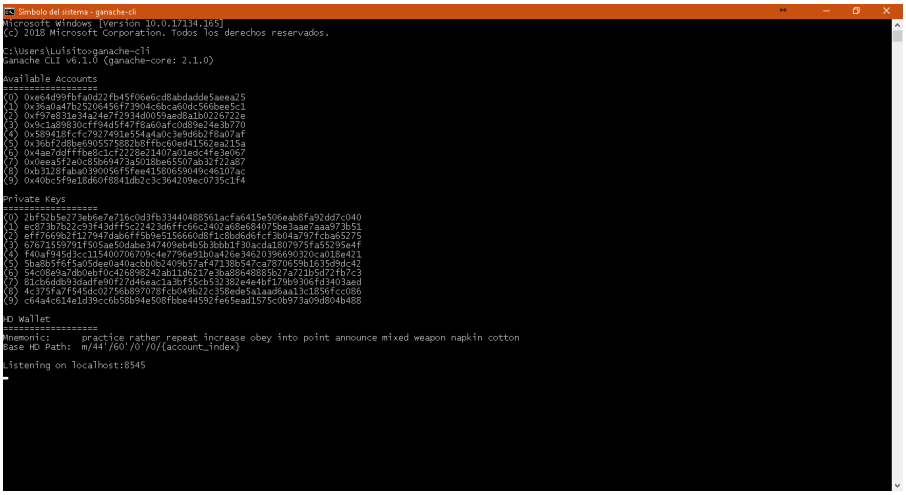


Figura 4.2: Funcionamiento de Ganache aplicación de línea de comandos [25]

Lo que hace es iniciar la cadena de bloques, creando 10 cuentas con 100 Ether cada una.

4.2.3. Truffle

Truffle es el framework más popular para el desarrollo en Ethereum hoy en día. Truffle nos ofrece:

1. Compilación, enlace y despliegue de Smart contracts desde el propio framework.
2. Depuración y testing automatizado de contratos.
3. Framework con scripts de despliegue y migraciones en redes públicas y privadas.
4. Acceso a cientos de paquetes externos y gestión con EthPM y NPM.
5. Consola interactiva para comunicación directa con los contratos.
6. Interacción con contratos mediante scripts externos.

Truffle [55] nos ofrece una serie de proyectos predefinidos o boxes que podemos usar como plantilla para nuestros proyectos.

En este caso vamos a utilizar el proyecto Webpack como plantilla para el desarrollo de nuestro proyecto, ya que, Webpack es un empaquetador de módulos para aplicaciones JavaScript.

4.2.4. Solidity

Solidity [45] es un lenguaje de alto nivel orientado a contratos. Su sintaxis es similar a la de JavaScript y está enfocado específicamente a la Máquina Virtual de Etehreum (EVM).

Solidity está tipado de manera estática y acepta, entre otras cosas, herencias, librerías y tipos complejos definidos por el usuario.

Como se verá, es posible crear contratos para votar, para el crowdfunding, para subastas a ciegas, para monederos multi-firmas, y mucho más.

La mejor manera de probar Solidity ahora mismo es usando Remix [42]

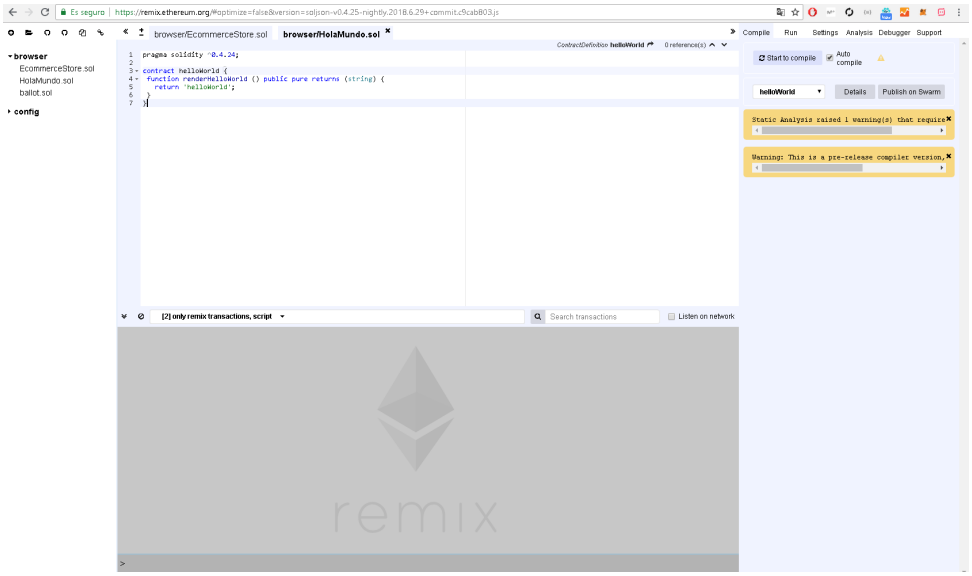


Figura 4.3: Funcionamiento de Remix con ejemplo HolaMundo.sol escrito en Solidity

Tiene los tipos de datos básicos más comunes además de otros específicos como “address ”, que sirve para almacenar direcciones de Ethereum. Al igual que otros lenguajes, Solidity cuenta con un conjunto de variables globales que permiten el acceso a información sobre la transacción.

La manipulación de los datos almacenados en el contrato se realiza por el compilador, ya que es el encargado de asignar su posición, en vez de por el programador.

La ventaja de Solidity frente a otros lenguajes creados anteriormente, es que, se puede dividir el contrato en funciones y emplear estructuras de datos o tablas hash. Así puede verse el contrato como una clase, con sus variables, atributos, métodos y su constructor en caso de ser necesario.

4.2.5. Bootstrap

Bootstrap [14] es un kit de herramientas de código abierto para desarrollar código HTML, CSS y JavaScript creado por Twitter. Sirve para crear proyectos responsive en la web. Es la biblioteca de componentes frontend más popular.

En el proyecto se utilizará para el desarrollo de la interfaz gráfica junto con más código CSS y JavaScript si fuera necesario para hacer la interfaz gráfica más atractiva y agradable al usuario.

4.2.6. JavaScript

Usaremos JavaScript de una forma extensa, ya que se utiliza en el lado servidor para guardar datos en la base de datos y consultar la base de datos y devolver los resultados al frontend. Gracias al framework Web3.js [51], esta es la API de JavaScript compatible con Ethereum que implementa la especificación Generic JSON RPC, se usa en la interfaz para interactuar con la cadena de bloques.

4.2.7. IPFS

IPFS [33] significa Inter Planetary File System. IPFS es un sistema de archivos distribuidos peer to peer que busca conectar todos los dispositivos informáticos con el mismo sistema de archivos. También es un protocolo diseñado para crear un método permanente y descentralizado para almacenar y compartir archivos.

La mayoría de las personas almacena sus archivos en su computadora local o en la nube a través de un proveedor en la nube (como Dropbox, AWS S3, Azure Cloud, etc.). Si tiene un sitio web, los activos pueden ser servidos desde un CDN que distribuye/replica geográficamente sus archivos para un acceso más rápido. Todas estas son soluciones muy buenas porque hacen un buen trabajo almacenando y publicando sus archivos con alta disponibilidad y sin pérdida de datos.

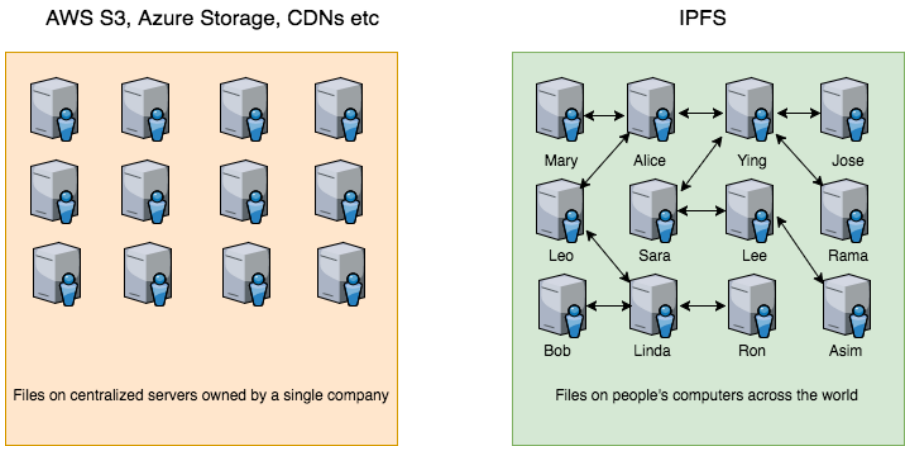


Figura 4.4: Representación de un sistema centralizado vs IPFS

Sin embargo, también tienen algunos inconvenientes que pueden ser preocupantes. Si estos proveedores de servicios tienen una interrupción (y sucede), sus archivos pueden ser inaccesibles. Si almacena archivos contra la política de la compañía, tienen derecho a eliminarlos o bloquearlos. Dependiendo de la cantidad de archivos, el costo de almacenamiento puede ser muy alto.

En el mundo de IPFS, estos proveedores de servicios con servidores centralizados son reemplazados por redes de computadoras peer-to-peer. La idea detrás de Ethereum es que puedes ejecutar tu aplicación sin que nadie la elimine. Del mismo modo, la idea detrás de IPFS es que puede alojar archivos que nadie puede eliminar.

Al igual que cualquiera puede ejecutar un nodo Ethereum, cualquiera puede ejecutar un nodo IPFS y unirse a la red para formar un sistema de archivos global. Los archivos se replican en muchos nodos y es casi imposible perder el acceso a sus archivos y también es resistente a la censura. Esto es similar a cómo su contrato y los datos asociados con él se almacenan en todos los nodos de Ethereum en la red. Cualquier persona en el mundo puede ejecutar un nodo IPFS y en el futuro puede ser compensado a través de Filecoin [**Filecoin**].

Usaremos IPFS, para cuando un usuario enumere un artículo en la tienda, la interfaz cargará los archivos y la descripción del producto en el IPFS y almacenará el hash del archivo cargado en la cadena de bloques.

4.2.8. Otras

Usaremos MongoDB para almacenar la información de los productos fuera de la cadena de bloques.

Usaremos NodeJS server para el servidor backend a través del cual el frontend se comunica con la base de datos. A la vez, usaremos distintas API's simples para que la interfaz pueda consultar y recuperar productos de la base de datos.

4.3. Herramientas utilizadas

En este apartado se analizan las diferentes herramientas utilizadas durante el desarrollo del proyecto. Para ello, se dividirán en dos grupos diferentes, el primero hace referencia a todas aquellas herramientas utilizadas durante las etapas de análisis, y después se explican las herramientas necesarias para la creación de la aplicación web.

4.3.1. Herramientas de análisis

Son las herramientas utilizadas para la creación de los diferentes diagramas y esquemas descritos en el documento.

Dia

Esta aplicación ha sido utilizada para la creación del diagrama E/R, diagrama de jerarquía de usuarios y los diagramas de casos de uso. En la *figura 7.1* podemos ver alguna de sus funcionalidades.

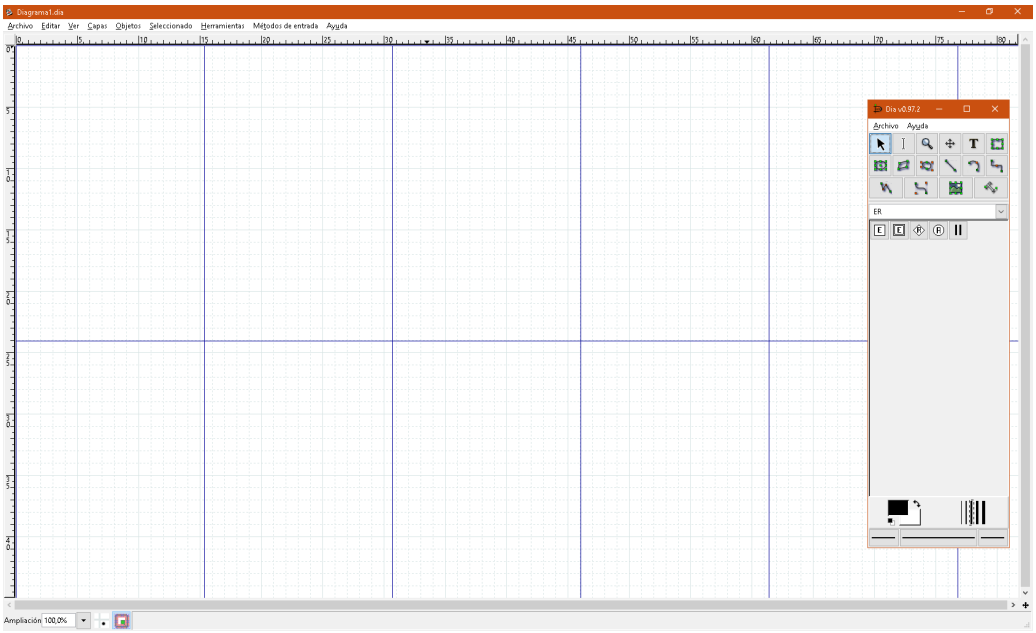


Figura 4.5: Captura de pantalla del programa dia

Paint

Esta aplicación ha sido utilizada para la creación de la arquitectura lógica y física, al igual que para crear las imágenes a través de las capturas de pantalla. En la *figura 7.2* podemos ver algunas de sus funcionalidades.

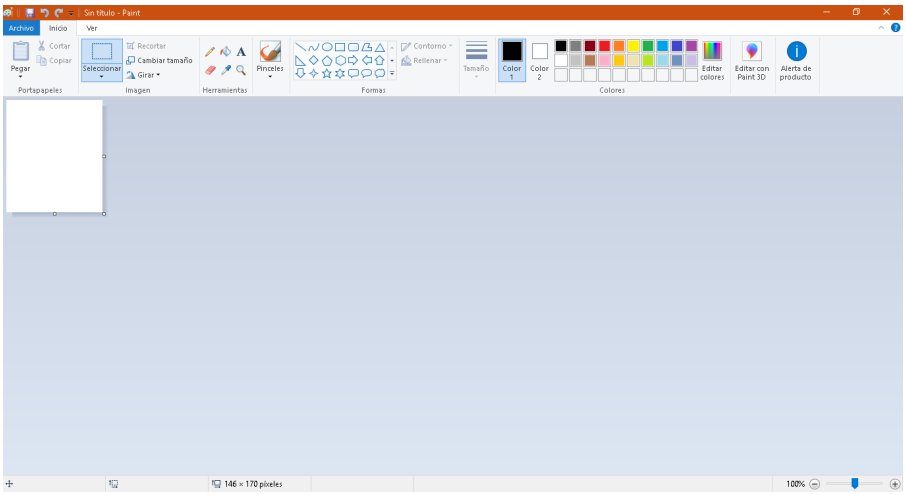


Figura 4.6: Captura de pantalla del programa Paint

TeXstudio

TeXstudio es un entorno de escritura integrado para crear documentos LaTeX. TeXstudio tiene numerosas funciones como resaltado de sintaxis, visor integrado, verificación de referencias y varios asistentes. TeXstudio es de código abierto y está disponible para todos los principales sistemas operativos.

Esta aplicación a sido utilizada para la creación de este documento.

En la *figura 7.3* podemos ver algunas de sus funcionalidades.

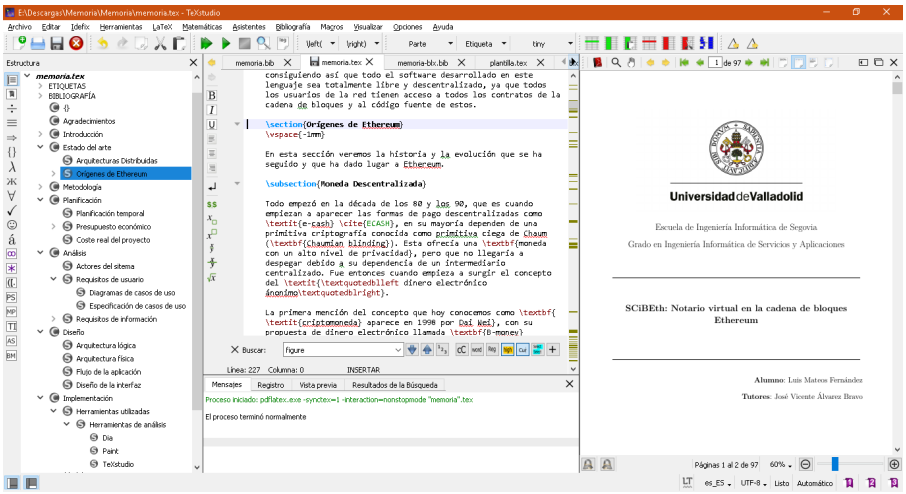


Figura 4.7: Captura de pantalla del programa TeXstudio

4.3.2. Herramientas para el desarrollo web

Para el desarrollo de la aplicación web hay diversas formas y lenguajes de programación para crear una solución. En este proyecto, se ha optado por un desarrollo web desde cero con la implementación de un framework de desarrollo para Ethereum que hace que el desarrollo sea más sencillo.

Utilizamos una caja llamada webpack, está incluye contratos, migraciones, pruebas, ina nterfaz de usuario y una canalización de complilación webpack.

A partir de esta caja se ha ido desarrollando el proyecto.

Remix

Remix es una poderosa herramienta de código abierto que le ayuda a redactar contratos de Solidity directamente desde el navegador. Escrito en Javascript, Remix admite tanto el uso en el navegador como localmente.

Esta herramienta online, se ha utilizado para el desarrollo de los contratos inteligentes, para ver los fallos y ver que compile bien, antes de escribirlos en el proyecto final.

En la *figura 7.4* podemos ver algunas de sus funcionalidades.

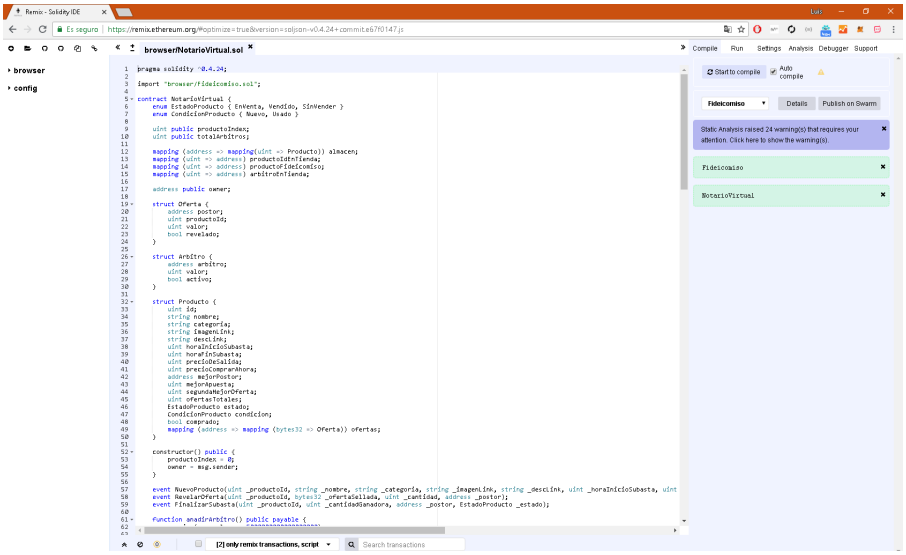


Figura 4.8: Captura de pantalla del programa Remix

Ganache-cli

Ganache CLI, parte del paquete Truffle de herramientas de desarrollo de Ethereum, es la versión de línea de comando de Ganache , su Blockchain personal para el desarrollo de Ethereum.

Ganache CLI utiliza ethereumjs para simular el comportamiento completo del cliente y hacer que el desarrollo de las aplicaciones de Ethereum sea más rápido, más fácil y más seguro. También incluye todas las funciones y características populares de RPC (como eventos) y se puede ejecutar de manera determinista para facilitar el desarrollo.

Usaremos está herramienta en nuestro proyecto para simular el Blockchain de Ethereum, donde se encontrarán nuestros contratos y nuestros clientes.

Ganache-cli nos proporciona 10 cuentas con 100 ETH en cada una de ellas para hacer pruebas e interactuar con ellas.

En la *figura 7.5* podemos ver Ganache-cli en funcionamiento.

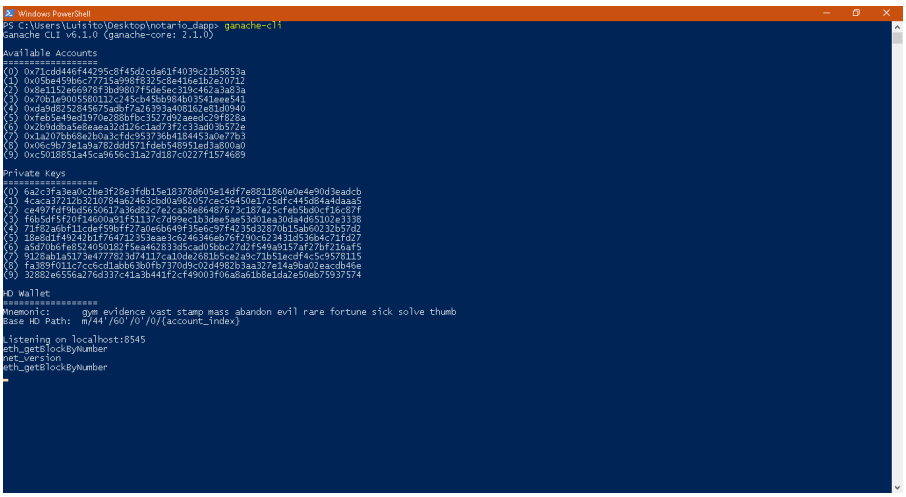


Figura 4.9: Captura de pantalla de Ganache-cli funcionando

NPM

NPM es el administrador de paquetes para JavaScript y el registro de software más grande del mundo.

Utilizaremos npm en nuestro proyecto para instalar distintas bibliotecas que necesitaremos, administrar dependencias para nuestro proyecto, y con ello conseguir una reutilización de código y facilitarnos el trabajo.

- **ethereumjs-util**

  Utilizaremos la biblioteca de ethereumjs-util para generar el hash de nuestra oferta.

- **ipfs-api**

  Utilizaremos la biblioteca de ipfs-api para interactuar con IPFS a través de la interfaz.

- **mongosee**

  Utilizaremos mongosee, un popular controlador de JavaScript para interactuar con la base de datos MongoDB a través de la aplicación NodeJS.

- **express**

  Utilizaremos Expressjs, que es un framework web simple y minimalista para manejar solicitudes web. Usando Express, hacemos uso de nuestra API para hacer las funciones con muy pocas líneas de código.

- **nodemon**

  Uno de los problemas de utilizar NodeJS para desarrollar la aplicación, es que, debes reiniciar la aplicación si realizamos algún cambio, por ello se utiliza la biblioteca nodemon. Nodemon sirve para monitorear el contenido del archivo y reinicia el servicio cuando detecta que el archivo a sido modificado.

IPFS

La definición la podemos encontrar en el apartado *3.2.7. IPFS* en la página *29*.

Nosotros usaremos IPFS para añadir las imágenes y descripciones de los productos y almacenar su hash para así ahorrar espacio en el Blockchain y que salga más barato minar las transferencias que se vean comprometidas, como por ejemplo, añadir un producto.

MongoDB

MongoDB es una base de datos NoSQL orientada a documentos, que usaremos para almacenar los productos y hacer consultas sobre ellos, sin tener que recurrir a la Blockchain, lo que llevaría más carga de trabajo y más precio por transferencias.

Sublime Text 3

Es una herramienta de edición de textos que sirve para el desarrollo de diferentes archivos (.js, .sol, .css, .html, etc). Y que se ha usado para ir desarrollando todo el proyecto.

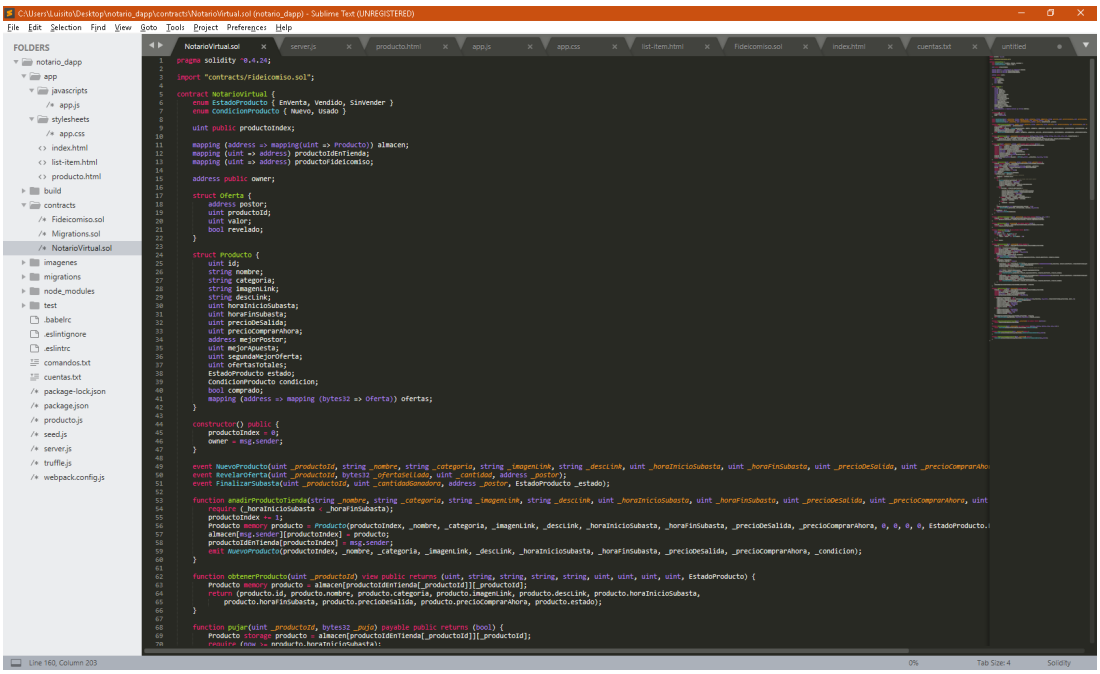


Figura 4.10: Captura de pantalla de Sublime Text 3

Capítulo 5

Planificación y Presupuesto

En este capítulo se va a presentar la planificación del proyecto. Se describirá la planificación temporal y la estimación presupuestaría sobre la cual se llevará a cabo el proyecto.

5.1. Planificación temporal

Como hemos visto en el capítulo anterior, vamos a utilizar un modelo en cascada, con 4 fases (Investigación, Análisis, Programación y Pruebas).

A continuación se mostrara la planificación temporal desglosada por fases, así como los días en los que se estima que comience y acabe cada fase.

Sumando los días de la planificación, limitados por la fecha de inicio (27/02/2018) y el plazo límite para la convocatoria extraordinaria (16/07/2018) , se estima una duración de proyecto de 100 días contando que son 20 días al mes y 8 horas al día, es decir, a tiempo completo. Se ha obtenido el diagrama de Gantt para esta planificación temporal, con una fecha inicial de 27/02/2018 y fin del 16/07/2018.

		Nombre	Duración	Inicio	Terminado
1		**⊟INVESTIGACIÓN**	**35 days?**	**27/02/18 8:00**	**16/04/18 17:00**
2		Redes Blockchain	5 days?	27/02/18 8:00	5/03/18 17:00
3		Smart Contract	5 days?	6/03/18 8:00	12/03/18 17:00
4		Plataformas	10 days?	13/03/18 8:00	26/03/18 17:00
5		Arquitectura	10 days?	27/03/18 8:00	9/04/18 17:00
6		Lenguajes de programación	5 days?	10/04/18 8:00	16/04/18 17:00
7		**⊟ANÁLISIS**	**12 days?**	**17/04/18 8:00**	**2/05/18 17:00**
8		Ethereum	2 days?	17/04/18 8:00	18/04/18 17:00
9		Ebay	1 day?	19/04/18 8:00	19/04/18 17:00
10		Actores	1 day?	20/04/18 8:00	20/04/18 17:00
11		Requisitos	3 days?	23/04/18 8:00	25/04/18 17:00
12		Funcionamiento	5 days?	26/04/18 8:00	2/05/18 17:00
13		**⊟Programación**	**47 days?**	**3/05/18 8:00**	**6/07/18 17:00**
14		Variables	2 days?	3/05/18 8:00	4/05/18 17:00
15		Funciones	3 days?	7/05/18 8:00	9/05/18 17:00
16		Hola Mundo	1 day?	10/05/18 8:00	10/05/18 17:00
17		Otras contratos sencillos	4 days?	11/05/18 8:00	16/05/18 17:00
18		Solidity a fondo	2 days?	17/05/18 8:00	18/05/18 17:00
19		Plataforma	35 days?	21/05/18 8:00	6/07/18 17:00
20		**⊟Pruebas**	**6 days?**	**9/07/18 8:00**	**16/07/18 17:00**
21		Funcionamiento	4 days?	9/07/18 8:00	12/07/18 17:00
22		Actualizaciones Solidity	2 days?	13/07/18 8:00	16/07/18 17:00

Figura 5.1: Tabla de la planificación temporal

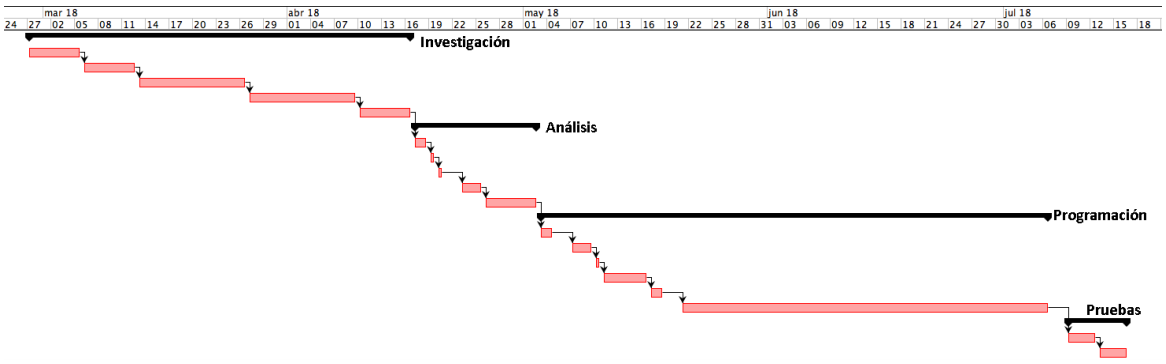


Figura 5.2: Diagrama de Gantt para la planificación temporal

5.2. Presupuesto económico

Para llevar a cabo el presupuesto representativo, primero se analizará la planificación temporal y se calcularán los costes en función de los tiempos previstos, posteriormente se utilizarán el método de puntos de función y de COCOMO II.

Con ello pretendemos elegir la opción más realista y que mejor represente la relación entre la carga de trabajo realizada en el desarrollo del proyecto y el coste del trabajo.

Lo primero que vamos a evaluar, es el coste de software y hardware necesario para llevar a cabo el proyecto. Posteriormente realizaremos un análisis de los costes del personal en función de los roles llevados a cabo.

5.2.1. Presupuesto Hardware y Software

En la tabla siguiente, se muestran todos los componentes Hardware utilizados para la realización del proyecto, junto a ellos, se muestra el valor que representa el porcentaje del uso del componente frente a la estimación de la vida útil del mismo. Así conseguimos calcular el coste de uso del componente durante el periodo de duración.

Elemento	Coste Total	Vida Útil	Uso	Coste Real
Ordenador Portátil	600 €	4 años	10,5 %	63 €
Segunda pantalla	150 €	4 años	10,5 %	15,75 €
Conexión a Internet	35 €/mes	5 meses	100 %	175 €
Ratón USB	15 €	2 años	21 %	3,15 €
Impresora	75 €	4 años	10,5 %	7,875 €
			Total	**264,78 €**

Tabla 5.1: Presupuesto Hardware

Ahora vamos a pasar a los costes relativos a los programas utilizados para la realización del proyecto, tanto, de la memoria como de la aplicación.

Los valores se calcularán del mismo modo que los componentes hardware, es decir, evaluando su coste total en función del tiempo de uso de vida. Las herramientas software utilizadas se detallarán en el siguiente apartado.

A continuación se puede observar la tabla con los costes de software:

Elemento	Coste Total	Vida Útil	Uso	Coste Real
Windows 10	135 €	2 años	21 %	28,35 €
Sublime Text 3	0 €	-	-	0 €
TeXstudio	0 €	-	-	0 €
Startuml	0 €	-	-	0 €
Remix	0 €	-	-	0 €
NodeJS	0 €	-	-	0 €
MongoDB	0 €	-	-	0 €
Notepad++	0 €	-	-	0 €
IPFS	0 €	-	-	0 €
Billetera Ethereum	0 €	-	-	0 €
MetaMask	0 €	-	-	0 €
Google Chrome	0 €	-	-	0 €
			Total	**28,35 €**

Tabla 5.2: Presupuesto Software

5.2.2. Recursos Humanos

Ahora que ya tenemos los presupuestos de costes de hardware y software, vamos a proceder a definir los costes de recursos humanos.

Para este proyecto se han utilizado cuatro tipos de roles diferentes, el de investigador, el de analista de software, el de programador de aplicaciones web (frontend y backend) y el programador de contratos inteligentes con Solidity. En la siguiente tabla se puede observar el coste presupuestado en función del rol.

Rol	Salario Mensual	Salario por horas
Investigador	1.500 €/mes	9,4 €/hora
Analista Software	2.000 €/mes	12,5 €/hora
Programador de aplicaciones web	1.800 €/mes	11,25 €/hora
Programador de Smart Contract	2.250 €/mes	15,63 €/mes

Tabla 5.3: Salario por rol

A continuación se muestra una tabla con el presupuesto en función a las horas trabajadas por el rol en la planificación inicial.

Para calcular las horas se han asignado, al investigador la primera fase de investigación, al analista la fase de análisis y a los programadores la fases de programación y pruebas.

El sueldo se calcula a partir del sueldo medio mensual de los roles, obteniendo el coste por hora y teniendo en cuenta que se trabaja 8 horas al día con 20 días laborables al mes.

Rol	Salario mensual	Salario/horas	Horas	Salario final
Investigador	1.500 €/mes	9,4 €/hora	35 * 8 = 280 horas	2.632 €
Analista Software	2.000 €/mes	12,5 €/hora	12 * 8 = 96 horas	1.200 €
Programador aplicaciones web	1.800 €/mes	11,25 €/mes	53 * 8 = 424 horas	4.770 €
Programador Smart Contract	2.250 €/mes	15,63 €/mes	53 * 8 = 424 horas	6.627,12 €
			Total	**15.229,12 €**

Tabla 5.4: Costes de personal estimados

Finalmente, el presupuesto total estimado será la suma de los presupuestos hardware, software y de recursos humanos, que podemos ver en la siguiente tabla:

Presupuesto	Euros
Hardware	264,78 €
Software	28,35 €
Recursos Humanos	15.229,12 €
Total	**15.522,25 €**

Tabla 5.5: Presupuesto Final Estimado

5.2.3. Método de puntos de función

Este método consiste en realizar una estimación del coste del proyecto software evaluando todas sus funciones, para obtener los puntos de función de cada una, en función a su complejidad, para implementarla.

Para evaluar los puntos de función se han establecido los siguientes grupos:

1. Nº de entradas de usuario: es el equivalente al uso de formularios por parte del usuario.
2. Nº de salidas de usuario: es el equivalente a las funcionalidades que muestran datos al usuario.
3. Nº de peticiones externas: son las consultas que realiza la aplicación a la Blockchain.
4. Grupos lógicos internos: consultas a bases de datos internas.
5. Grupos lógicos externos: consultas a bases de datos externas.

Una vez tenemos repartidos los componentes en grupos, haremos uso de la tabla que nos proporciona el método para asignar la complejidad adecuada a cada uno de ellos.

Ficheros lógicos externos e internos				Salidas y consultas				Entradas			
Registros elementales	Datos elementales			Tipos de ficheros	Datos elementales			Tipos de ficheros	Datos elementales		
	1-19	20-50	>51		1-5	6-19	>20		1-4	5-15	>16
1	Baja	Baja	Media	0-1	Baja	Baja	Media	0-1	Baja	Baja	Media
2-5	Baja	Media	Alta	2-3	Baja	Media	Alta	2-3	Baja	Media	Alta
>6	Media	Alta	Alta	>4	Media	Alta	Alta	>3	Media	Alta	Alta

Figura 5.3: Criterios para evaluar la complejidad de los elementos de cálculo

Por lo que se nos generaría la siguiente tabla para nuestra aplicación:

Grupo	Componente	Complejidad
Entradas	Añadir producto	Media
	Hacer Puja	Baja
	Comprar Producto	Baja
	Finalizar Subasta	Baja
	Revelar Pujas	Baja
	Enviar Reembolso	Baja
	Enviar Pago	Baja
Salidas	Mostrar todos los productos	Baja
	Mostrar un producto	Media
	Mostrar una determinada categoria	Baja
Consultas	Obtención de JSON	Alta
Grupos internos	Base de datos MongoDB	Media
Grupos externos	Consultas a la Blockchain	Alta
	Escucha eventos Blockchain	Alta

Tabla 5.6: Complejidad de nuestros componentes

Observando la tabla se muestra la complejidad de los elementos de la aplicación haciendo uso de la *figura 4.3*.

Ahora pasamos a calcular los puntos de función sin ajustar:

Grupo	**Entradas**			**Salidas**			**Consultas**			**G. Interno**			**G. Externo**			
Complejidad	**B**	**M**	**A**	**B**	**M**	**A**	**B**	**M**	**A**	**B**	**M**	**A**	**B**	**M**	**A**	
Factor	x3	x4	x6	x4	x5	x7	x7	x10	x15	x5	x7	x10	x3	x4	x6	
Número	6	1	0	2	1	0	0	0	1	0	1	0	0	0	2	
Total	18	4	0	8	5	0	0	0	15	0	7	0	0	0	12	**= 69**

Tabla 5.7: Puntos de función sin ajustar

Una vez obtenidos los puntos de función sin ajustar, debemos llevar a cabo un ajuste con un coeficiente obtenido tras la asignación de un valor entero entre 0 y 5 a una serie de variables, en función de la complejidad de éstas:

- El 0 equivale a Sin influencia.
- El 1 equivale a Incidental.
- El 2 equivale a Moderado.
- El 3 equivale a Medio.
- El 4 equivale a Significativo.
- El 5 equivale a Esencial.

Factor	**Valor**
1. Comunicaciones de Datos	5
2. Procesamiento de Datos Distribuido	3
3. Rendimiento	3
4. Configuración Altamente Utilizada	2
5. Tasa de Transacciones	4
6. Entrada de datos en línea	3
7. Eficiencia del Usuario Final	3
8. Actualización en línea	5
9. Complejidad de Procesamiento	4
10. Reusabilidad	2
11. Facilidad de Instalación	1
12. Facilidad de Operación	4
13. Múltiples Localizaciones	1
14. Facilidad de Cambio	2
Factor de ajuste de valor	**42**

Tabla 5.8: Cálculo del coeficiente para el factor de ajuste

Una vez tenemos el coeficiente para calcular el factor de ajuste, debemos utilizar la siguiente formula:

*FA = 0,65 + 0,01 * total coeficiente = 0,65 + 0,01 * 42 = 1,07*

Una vez tenemos el factor de ajuste, debemos multiplicarlo por los puntos de función obtenidos:

*PFA = PF * FA = 69 * 1,07 = 73,83 PF*

Para terminar, debemos realizar el cálculo de la estimación de tiempo. Para ello asumimos la equivalencia de que un punto de función es igual a 5 horas de trabajo, por lo que:

73,83 PF * 5 h/PF = 369,15 horas, unos 47 días si consideramos 8 horas al día de trabajo.

Como se puede observar, nos sale casi la mitad de horas con respecto a lo que habíamos estimado en un principio (100 horas), esto se debe a que en esté método no se tiene en cuenta el tiempo de investigación y aprendizaje llevado a cabo, sino que se centra en las tareas de desarrollo que se deben cumplir.

Ahora repartimos esas horas entre los roles para calcular nuestro presupuesto final. Se utilizarán las horas de la estimación inicial para calcular el % que trabaja cada rol.

Rol	**%**
Investigador	35 %
Analista de Software	12 %
Programador de aplicaciones web	53 %
Programador de Smart Contract	53 %

Tabla 5.9: % de trabajo por rol

Como se puede observar, los dos programadores trabajan el mismo tiempo. Así quedaría el siguiente presupuesto:

Rol	**Sueldo al mes**	**Sueldo a la hora**	**Horas**	**Total**
Investigador	1.500 €	9,4 €/hora	369,15 * 0,35 = 129,2025	1.214,51 €
Analista de Software	2.000 €	12,5 €/hora	369,15 * 0,12 = 44,298	553,73 €
Programador de aplicaciones web	1.800 €	11,25 €/hora	369,15 * 0,53 = 195,6495	2.201,06 €
Programador de Smart Contract	2.250 €	15,63 €/hora	369,15 * 0,53 = 195,6495	3.058,01 €
			Total	**7.027,31 €**

Tabla 5.10: Presupuesto final de puntos de función

5.2.4. Método de COCOMO II

El método del COCOMO II trata de realizar un presupuesto a partir de la estimación de las líneas de código y a partir de los puntos de función previamente obtenidos.

Para proceder a realizar el calculo hay que elegir entre varios modelos, para ver cual se adapta mejor a la aplicación que se pretende realizar.

Estos modelos son:

- **Orgánico:** de pequeño tamaño para proyectos sencillos con pocas innovaciones, desarrollados por equipos pequeños y experimentados en el desarrollo de aplicaciones y con requisitos poco rígidos.
- **Semiacoplado:** para complejidad y tamaño del proyecto software intermedio, formados por varios equipos con experiencia heterogénea y con requisitos relativamente poco rígidos.

- **Empotrado:** para proyectos de software que deben ser desarrollados en un conjunto de hardware, software y restricciones operativas muy restringidas (alto grado de dificultad y requisitos muy rígidos).

Nuestro proyecto se adapta mejor al modelo semiacoplado, porque, tiene una dificultad intermedia y requiere que el equipo de desarrollo web y el de desarrollo smart contract se coordinen.

Para calcular la estimación de líneas de código se identifican primero los tipos de lenguaje que se van a utilizar, en este caso Solidity y JavaScript en su gran mayoría, aunque también se utilizará HTML y CSS para la interfaz web. Como Solidity se parece a JavaScript, se cogerá la estimación de JavaScript para calcular las líneas de código por punto de función.

Language	QSM SLOC/FP Data			
	Avg	Median	Low	High
ABAP (SAP) *	28	18	16	60
ASP*	51	54	15	69
Assembler *	119	98	25	320
Brio +	14	14	13	16
C *	97	99	39	333
C++ *	50	53	25	80
C# *	54	59	29	70
COBOL *	61	55	23	297
Cognos Impromptu Scripts +	47	42	30	100
Cross System Products (CSP) +	20	18	10	38
Cool:Gen/IEF *	32	24	10	82
Datastage	71	65	31	157
Excel *	209	191	131	315
Focus *	43	45	45	45
FoxPro	36	35	34	38
HTML *	34	40	14	48
J2EE *	46	49	15	67
Java *	53	53	14	134
JavaScript *	47	53	31	63
JCL *	62	48	25	221
LINC II	29	30	22	38
Lotus Notes *	23	21	19	40
Natural *	40	34	34	53
.NET *	57	60	53	60
Oracle *	37	40	17	60
PACBASE *	35	32	22	60
Perl *	24	15	15	60
PL/I *	64	80	16	80
PL/SQL *	37	35	13	60
Powerbuilder *	26	28	7	40
REXX *	77	80	50	80
Sabretalk *	70	66	45	109
SAS *	38	37	22	55
Siebel *	59	60	51	60
SLOGAN *	75	75	74	75
SQL *	21	21	13	37
VB.NET *	52	60	26	60
Visual Basic *	42	44	20	60

Figura 5.4: LOC por puntos de función según el lenguaje [29]

Como podemos observar en la imagen de arriba para el lenguaje de JavaScript se estiman unas 53 líneas de código por cada punto de función. Por lo que el cálculo de líneas de código se corresponde con:

LDC = PF * 53 = 69 * 53 = 3657

Tenemos que tener en cuenta una serie de factores de ajuste que se obtendrán en función de las características de la aplicación siguiendo la siguiente tabla:

Factores Conductores del Coste		Valor de los factores					
		Muy bajo	Bajo	Medio	Alto	Muy alto	Extra
Software	Fiabilidad del software requerido	0,75	0,88	1,00	1,15	1,4	
	Tamaño de la base de datos		0,94	1,00	1,08	1,16	
	Complejidad del software	0,70	0,85	1,00	1,15	1,30	1,65
Hardware	Restricciones de rendimiento en tiempo de ejecución			1,00	1,11	1,30	1,66
	Restricciones de memoria			1,00	1,06	1,21	1,56
	Volatilidad del entorno de la máquina virtual		0,87	1,00	1,15	1,30	
	Tiempo de respuesta requerido		0,87	1,00	1,07	1,15	
Personal	Capacidad de los analistas	1,46	1,19	1,00	0,86	0,71	
	Experiencia con el tipo de aplicación	1,29	1,13	1,00	0,91	0,82	
	Experiencia con el hardware	1,21	1,10	1,00	0,90		
	Experiencia con el lenguaje de programación	1,14	1,07	1,00	0,95		
	Capacidad de los programadores	1,42	1,17	1,00	0,86	0,70	
Proyecto	Técnicas modernas de programación	1,24	1,10	1,00	0,91	0,82	
	Utilización de herramientas software	1,24	1,10	1,00	0,91	0,83	
	Restricciones en la planificación temporal del desarrollo	1,23	1,08	1,00	1,04	1,10	

Figura 5.5: Factores de ajuste presupuesto COCOMO II [16]

Teniendo en cuenta la tabla de la *figura 4.5* calculamos los factores de nuestra aplicación:

Factores de coste	**Valor**
Fiabilidad del software requerido	1,15
Tamaño de la base de datos	1,08
Complejidad del software	1,15
Restricciones de rendimiento en tiempo de ejecución	1,00
Restricciones de memoria	1,00
Volatilidad del entorno de la máquina virtual	1,15
Tiempo de respuesta requerido	0,87
Capacidad de los analistas	1,00
Experiencia con el tipo de aplicación	1,29
Experiencia con el lenguaje de programación	1,07
Capacidad de los programadores	1,00
Técnicas modernas de programación	0,91
Utilización de herramientas software	1,00
Restricciones en la planificación temporal del desarrollo	1,04
Total	**1,867**

Tabla 5.11: Total de los factores de ajuste presupuesto COCOMO II

Para obtener el tiempo se hace uso de los modificadores de la siguiente tabla, dependiendo del modelo de proyecto que sea:

Proyecto de software	a	b	c	d
Orgánico	2,4	1,05	2,5	0,38
Semiacoplado	3,0	1,12	2,5	0,35
Integrado	3,6	1,20	2,5	0,32

Tabla 5.12: Tabla de modificadores para el tiempo COCOMO II

Ahora que tenemos todos los datos, vamos a calcular primero el esfuerzo utilizando la siguiente formula:

E = a * (KLDC) ^b * factor de ajuste = 3,0 * (3,657)^1,12 * 1,867 * 1,15 (complejidad) * 1 (personal) * 1,29 (experiencia) = 35,51 personas-mes

Este cálculo indica el número de personas necesarias para terminar el proyecto en un mes. Utilizando este dato, el tiempo necesario será:

t = c * E^d = 2,5 * (35,51)^0,35 = 8,72 meses

Con un número medio de personas de:

nº medio de personas = 35,51/8,72 = 4,07 personas 5 personas

En resumen, necesitamos a 5 personas para realizar el proyecto en 7,102 meses (35,51/5). Como solo se realiza por una persona, se estima que tardará aproximadamente 35,5 meses, lo que equivaldría a 35,5*20 días al mes*8horas al día = 5680 horas.

Ahora vamos a proceder a realizar una división del tanto por ciento de trabajo que realiza cada rol de acuerdo a la planificación inicial.

Rol	Sueldo al mes	Sueldo a la hora	Horas	Total
Investigador	1.500 €	9,4 €/hora	5680 * 0,35 = 1988	18.687,2 €
Analista de Software	2.000 €	12,5 €/hora	5680 * 0,12 = 681,6	8.520 €
Programador aplicación web	1.800 €	11,25 €/hora	5680 * 0,53 = 3010,4	33.867 €
Programador Smart Contract	2.250 €	15,63 €/hora	5680 * 0,53 = 3010,4	47.052,53 €
			Total	**108.126,73 €**

Tabla 5.13: Coste real del personal

5.2.5. Comparativas de los presupuestos

De los dos métodos utilizados para calcular el presupuesto, el que más se acerca a la realidad es el de **estimación por puntos de función**, con un presupuesto de 7.027,31 €, ya que, **COCOMO II** se va a un presupuesto de 108.126,73 €que es muy elevado.

Comparando los resultados de los distintos métodos de estimación de presupuestos, podemos observar que el resultado obtenido con el método de puntos de función, con un presupuesto total de 7.027,31 €, es prácticamente la mitad que el obtenido mediante la estimación por la planificación temporal, que con 100 días de trabajo nos dio un presupuesto de 15.522,25 €. Esto se debe a que en el caso del método de puntos de función no se tiene en cuenta el tiempo empleado en investigación y aprendizaje sobre la tecnología que se va a desarrollar.

5.3. Coste real del proyecto

COCOMO II tiene bastantes limitaciones debido a las desviaciones que se producen, tales como el tema del aprendizaje o la investigación, por lo que son difíciles de representar mediante estimación, por eso vamos a realizar el calculo del presupuesto con los datos reales obtenidos tras la realización del TFG.

Los costes que obtuvimos de hardware y software son los mismo, no varían, el de hardware es de 264,78 €y el de software es de 28,35 €.

Las tareas llevadas a cabo se le asignan a los mismos roles descritos en apartados anteriores (Investigador, Analista de Software, Programador de aplicaciones Web y Programador de Smart Contract), repartiendo las horas realizadas para la realización del TFG:

Rol	**Sueldo al mes**	**Sueldo a la hora**	**Horas**	**Total**
Investigador	1.500 €	9,4 €/hora	50	470 €
Analista de Software	2.000 €	12,5 €/hora	20	250 €
Programador aplicación web	1.800 €	11,25 €/hora	55	618,75 €
Programador Smart Contract	2.250 €	15,63 €/hora	70	1094,1 €
			Total	**2.432,85 €**

Tabla 5.14: Presupuesto real de recursos humanos

Sumando el total de los prepuestos nos da:
264,78 €+ 28,35 €+ 2.432,85 €= **2.725,98 €**

Si lo comparamos con los presupuestos obtenidos por los anteriores métodos hay una diferencia significativa.

En conclusión, debemos tener en cuenta los presupuestos realizados en la planificación inicial, ya que es la que más se acerca a la realidad.

.

Capítulo 6

Análisis

En este capítulo, se muestra el análisis previo a la implementación del contrato inteligente y el desarrollo de la aplicación web.

6.1. Actores del sistema

Un actor es una entidad externa al sistema que guarda una relación con éste y al cual le demanda una funcionalidad. Los actores además de a las personas, incluyen también a sistemas externos y elementos de carácter abstracto en el caso de tener alguna funcionalidad con el sistema que se describe.

Durante el proceso de análisis se han identificado 7 tipos de actores distintos que interactuarán de alguna manera con la web o el contrato inteligente.

1. **Dueño del contrato:** es el actor que creara el contrato y lo implementara en la Blockchain para que la aplicación y el resto de usuarios puedan interactuar con el contrato y hará algunas veces de árbitro.

2. **Vendedor:** es el actor que añade productos a la tienda en forma de subasta y comprar ahora.

3. **Comprador/Pujador:** es el actor que compra o puja por un producto de la tienda.

4. **Árbitro:** es quien finaliza una subasta, en caso de compra directa el árbitro será el dueño del contrato, y en caso de disputa es el que tiene que intervenir.

5. **Usuario general:** es el que accede al sistema para ver información pero no esta registrado en MetaMask.

6. **MetaMask:** sistema externo encargado de gestionar la billetera de cada usuario en la Blockchain y que sirve para unir la aplicación web con la Blockchain.

7. **Reloj del sistema:** Es un actor ficticio. Representa un proceso del sistema que cuando llega a la fecha y hora de una subasta programada, lanza el evento correspondiente.

A continuación se muestra la jerarquía de los actores, que indica la relación de herencia entre funcionalidades y actores. Los actores relacionados con flechas heredan las funcionalidades del actor al cual apuntan.

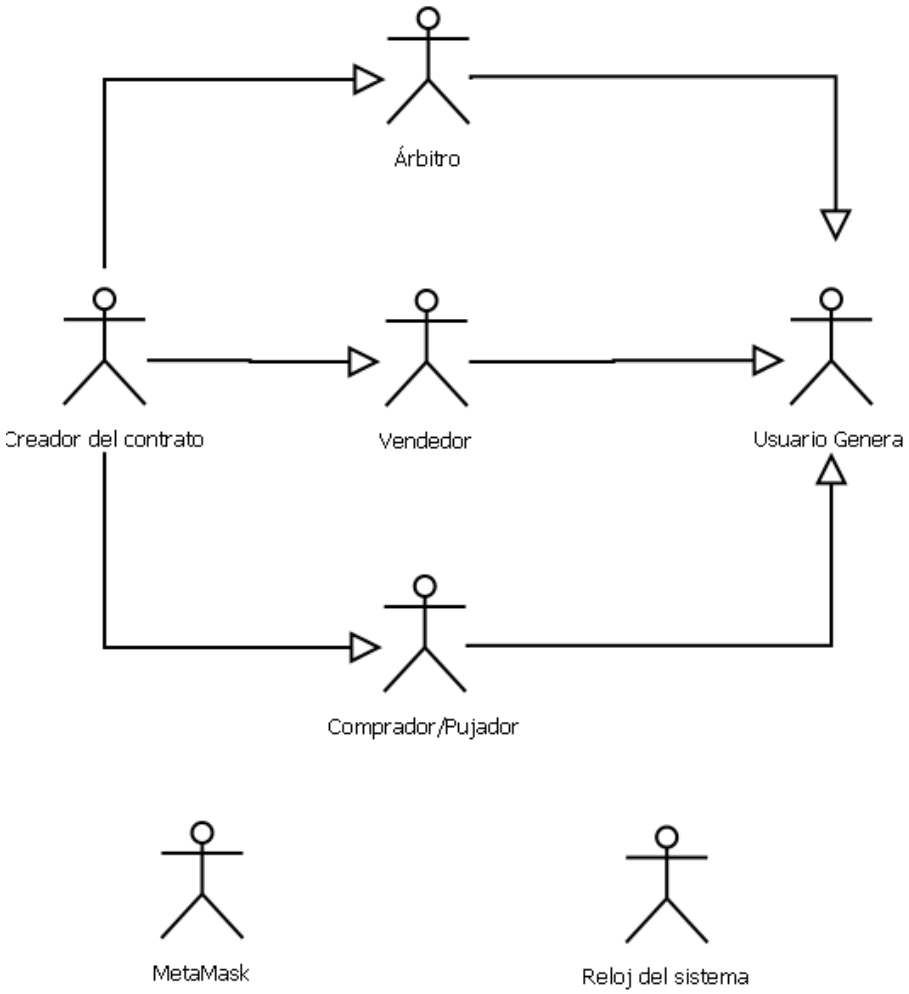


Figura 6.1: Diagrama de la jerarquía de los actores del sistema

6.2. Requisitos de usuario

A continuación se muestran los requisitos de usuarios identificados para la plataforma:

Identificador	Descripción
RU-01	El usuario general podrá visualizar la información de la tienda
RU-02	El usuario general podrá visualizar un producto
RU-03	El vendedor podrá añadir productos a la tienda
RU-04	El comprador/pujador podrá comprar un producto
RU-05	El comprador/pujador podrá pujar por un producto
RU-06	El comprador/pujador podrá revelar una puja
RU-07	El árbitro podrá finalizar una subasta
RU-08	El vendedor podrá enviar una votación de reembolso
RU-09	El vendedor podrá enviar una votación de pago
RU-10	El comprador podrá enviar una petición de reembolso
RU-11	El comprador podrá enviar una petición de pago
RU-12	El árbitro podrá enviar una petición de reembolso
RU-13	El árbitro podrá enviar una petición de pago
RU-14	El usuario general podrá clasificar los productos sin finalizar por categorías
RU-15	El usuario general podrá visualizar los productos en etapa de revelar
RU-16	El usuario general podrá visualizar los productos en etapa de finalizar
RU-17	Una vez finalizada la subasta, el usuario general no se podrán visualizar los productos vendidos en la página de inicio pero si por su id.

Tabla 6.1: Requisitos de usuario del sistema

6.2.1. Diagramas de casos de uso

En este apartado se recogen los diagramas que relacionan los distintos casos de uso, según los actores que los ejecutan o interactúan con ellos.

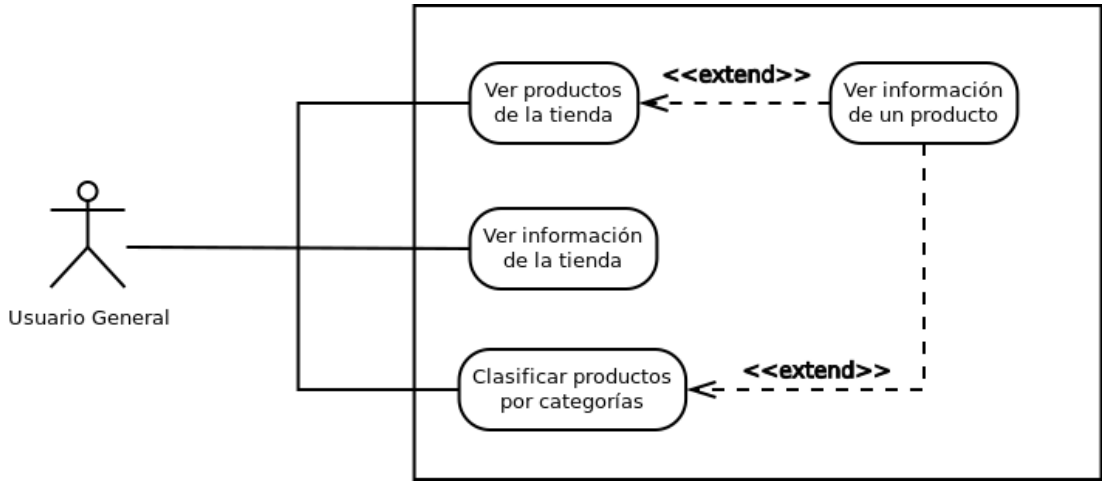


Figura 6.2: Diagrama de casos de uso del usuario general

En la *figura 5.2* podemos observar el diagrama de casos de uso del usuario general, del cual partirán los demás actores del sistema. Como se observa, el usuario general puede ver los productos que hay subidos por los distintos vendedores y el estado en el que estos están (en venta, en etapa de revelar o finalizado), podrá también ver los productos de una determinada categoría nada más, también pueden ver un producto concreto accediendo a toda la información de este y también puede ver la información que proporciona la página en las distintas pestañas.

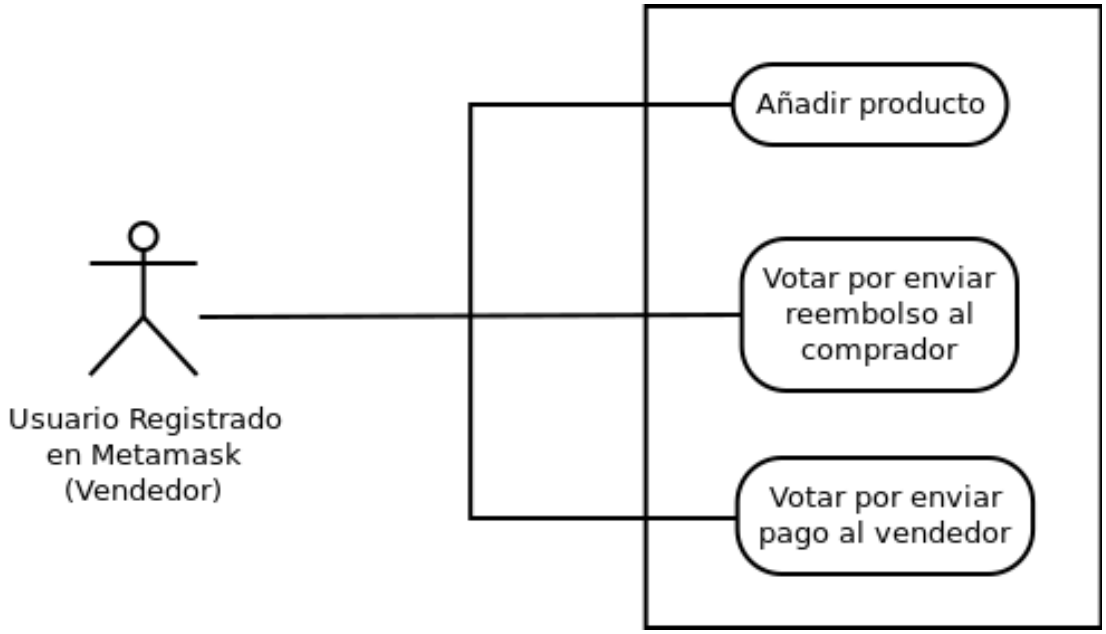


Figura 6.3: Diagrama de casos de uso del vendedor

El vendedor, que es un usuario registrado en MetaMask, podrá realizar, además de los que puede hacer el usuario general, añadir un producto a la tienda y desde el caso de uso de ver un producto podrá votar que se envié un reembolso de algún producto que el haya añadido con anterioridad al comprador y podrá votar también para solicitar el dinero de la venta para que se le pague.

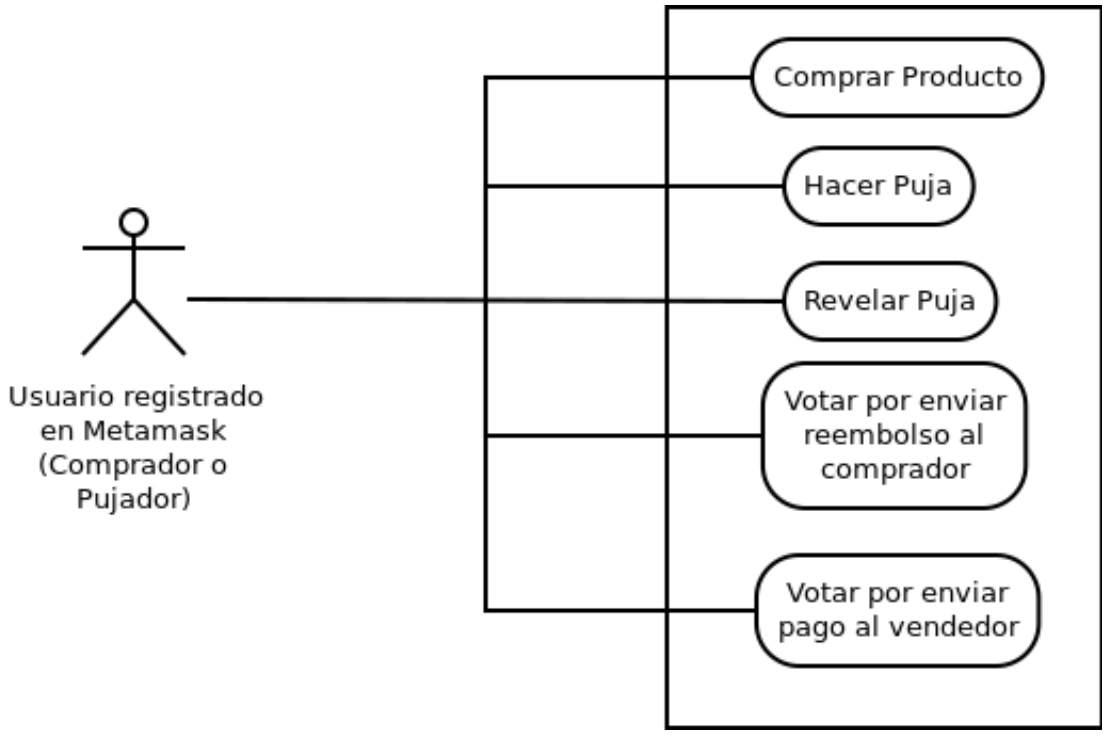


Figura 6.4: Diagrama de casos de uso del comprador/pujador

El comprador/subastador, es como el usuario anterior, un usuario registrado en MetaMask pero que ha comprado o hecho una puja por algún producto. Desde el caso de uso de ver un producto podrá comprar un producto que haya en la tienda bajo un precio fijado, si el producto está en venta, es decir, el tiempo que puso el vendedor para la venta no a acabado; hacer una puja por un producto que este en venta, es decir, que el tiempo de la venta no haya acabado; revelar una puja, en caso de haber hecho una puja y que la subasta este finalizada, para confirmarla y ver si es el ganador o no antes de un periodo de tiempo; podrá votar que se envié el dinero de algún producto que el haya comprado con anterioridad al vendedor y podrá votar también para solicitar el reembolso de la compra realizada.

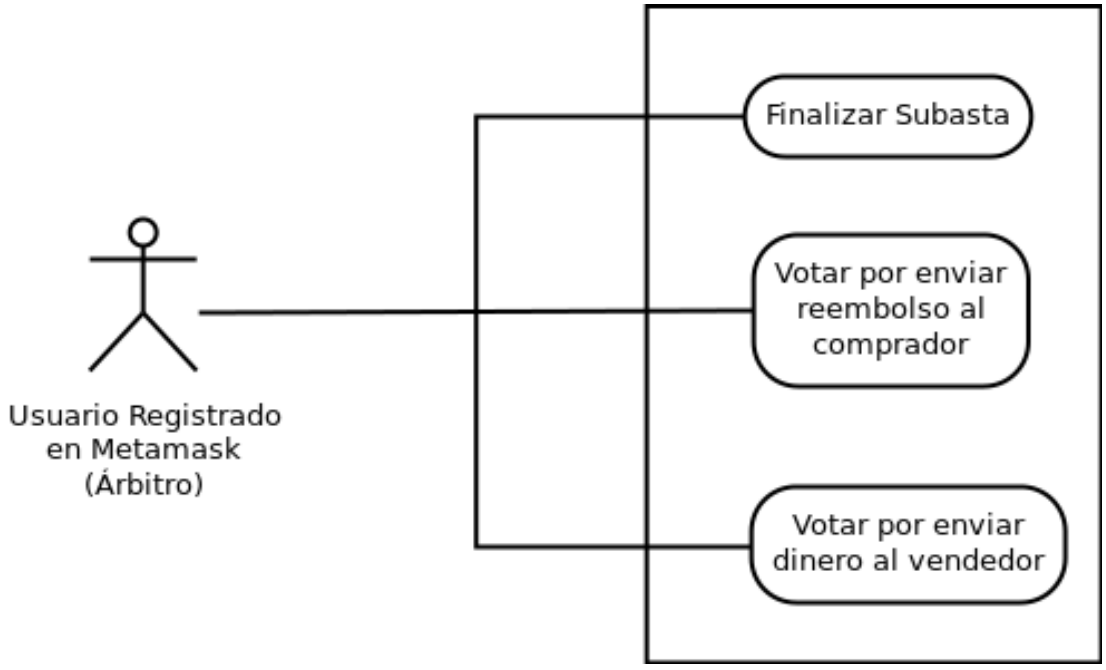


Figura 6.5: Diagrama de casos de uso del árbitro

El árbitro podrá finalizar una subasta una vez está haya llegado al tiempo límite, podrá votar que se envié un reembolso al comprador en caso de una venta con disputas y podrá votar también para enviar el dinero al vendedor en caso de una venta con disputas.

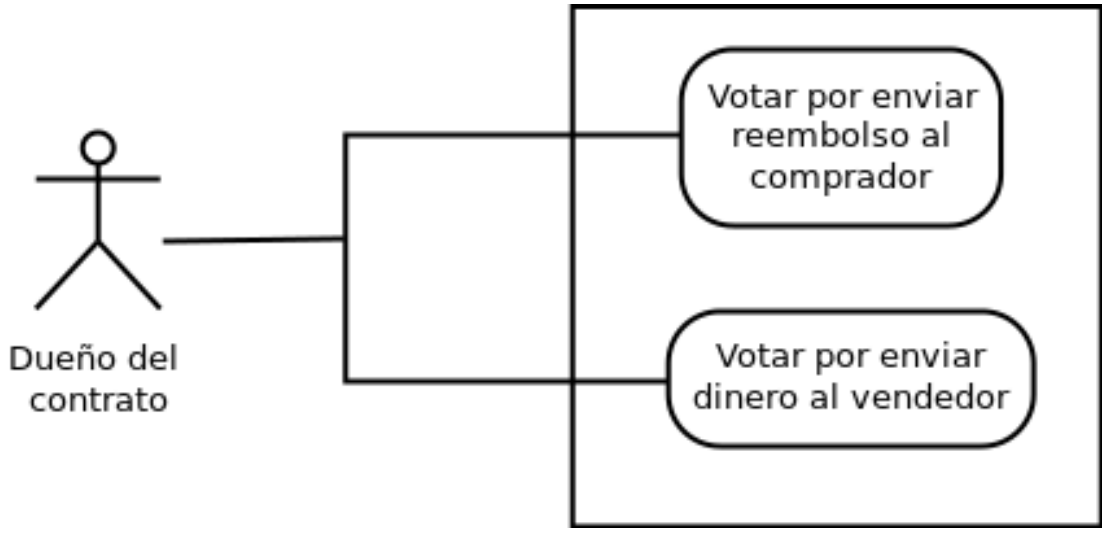


Figura 6.6: Diagrama de casos de uso del dueño del contrato

El dueño del contrato actuara de árbitro en las ventas que no son a través de pujas, por lo que podrá votar que se envié un reembolso al comprador en caso de una venta con disputas y podrá votar también para enviar el dinero al vendedor en caso de una venta con disputas.

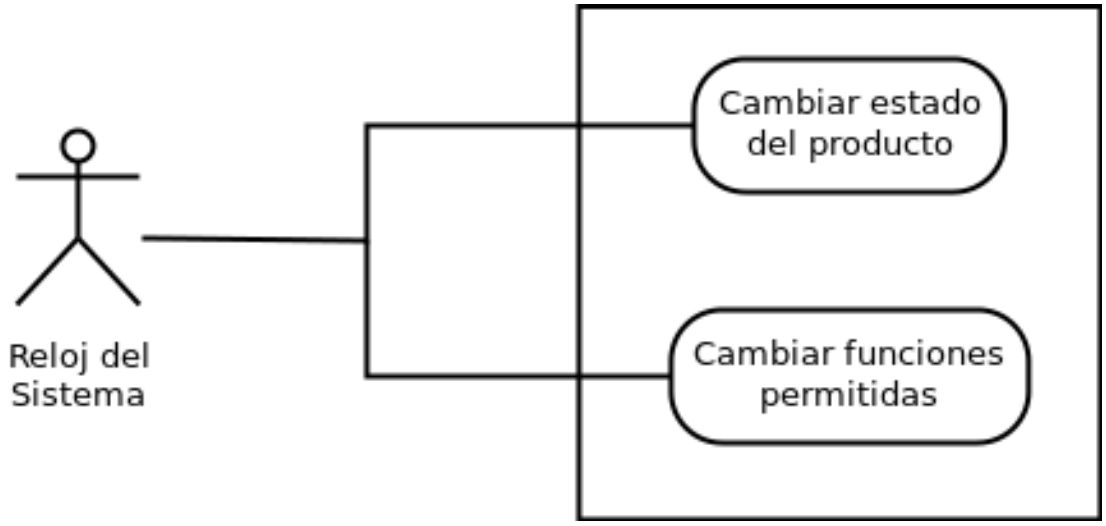


Figura 6.7: Diagrama de casos de uso del reloj

El reloj del sistema podrá cambiar el estado de un producto cuando este llegue al límite de su período de venta y será el sistema de seguridad para permitir que se puedan realizar unas ciertas operaciones u otras.

6.2.2. Especificación de casos de uso

En esta sección se recoge la especificación de todos los casos de uso indicados en los diagramas anteriores. Estará organizada por actores.

- **Usuario general**

CU-01	Ver productos de la tienda
Versión	1.0
Actor Principal	Usuario general
Actores Secundarios	
Disparador	El usuario ingresa en el index de nuestra aplicación web
Descripción	Un usuario podrá ver los productos que están en la tienda a forma de cátalogo
Precondiciones	
Secuencia normal	1. El usuario general ingresa con el navegador en nuestra web 2. El sistema muestra la página de inicio donde se encuentran los productos 2.1. Si el usuario general solicita un producto, se lanza el caso de uso CU-02: Ver información de un producto
Postcondiciones	1. El sistema muestra los productos clasificados por el estado del tiempo de venta
Excepciones	
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.2: CU-01 - Ver productos de la tienda

CU-02	Ver información de un producto
Versión	1.0
Actor Principal	Usuario general
Actores Secundarios	
Disparador	El usuario indica que quiere visualizar un producto
Descripción	Un usuario podrá ver los datos de un producto.
Precondiciones	
Secuencia normal	1. El usuario general solicita ver un producto 2. El sistema muestra la página del producto indicado
Postcondiciones	1. El sistema muestra el producto solicitado
Excepciones	
Prioridad	Alta
Casos relacionados	CU-01
Comentarios	

Tabla 6.3: CU-02 - Ver información de un producto

CU-03	Ver información de la tienda
Versión	1.0
Actor Principal	Usuario general
Actores Secundarios	
Disparador	El usuario indica que quiere visualizar la información de la tienda
Descripción	Un usuario podrá ver los datos de la tienda
Precondiciones	
Secuencia normal	1. El usuario general solicita ver información de la tienda 2. El sistema muestra la página de la información de la tienda
Postcondiciones	1. El sistema muestra la información de la tienda
Excepciones	
Prioridad	Baja
Casos relacionados	
Comentarios	

Tabla 6.4: CU-03 - Ver información de la tienda

CU-04	Clasificar por categorías
Versión	1.0
Actor Principal	Usuario general
Actores Secundarios	
Disparador	El usuario indica que quiere visualizar los productos de una cierta categoría
Descripción	Un usuario podrá ver los productos de una categoría
Precondiciones	
Secuencia normal	1. El usuario general solicita ver los productos de una categoría 2. El sistema muestra los productos en venta de una categoría
Postcondiciones	1. El sistema muestra los productos en venta de una categoría
Excepciones	
Prioridad	Media
Casos relacionados	
Comentarios	

Tabla 6.5: CU-04 - Clasificar por categorías

- **Vendedor**

CU-05	Añadir producto
Versión	1.0
Actor Principal	Usuario Registrado (Vendedor)
Actores Secundarios	
Disparador	El vendedor indica que quiere añadir un producto a la tienda
Descripción	El vendedor podrá añadir un producto a la tienda
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia
Secuencia normal	1. El vendedor solicita añadir un producto 2. El sistema proporciona los campos necesarios necesarios para agregar el producto 3. El vendedor introduce los datos del producto que desea vender 4. MetaMask le pide que pague la transferencia para que sea minada 5. El sistema agrega el producto a la tienda
Postcondiciones	1. El vendedor vuelve al formulario de añadir producto
Excepciones	3.1. El vendedor proporcionas datos incorrectos 3.1.1. El sistema muestrá un mensaje de error 3.1.2. El caso vuelve al punto 2 3.2. El usuario no completa algún campo requerido 3.2.1. El sistema muestra un mensaje de error 3.2.2. El caso vuelve al paso 2 4.1. El usuario no tiene dinero suficiente 4.1.1. El caso vuelve al paso 2 4.2. El usuario cancela el pago 4.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	
Comentarios	

Tabla 6.6: CU-05 - Añadir producto

CU-06	Votar por enviar reembolso al comprador
Versión	1.0
Actor Principal	Usuario Registrado (Vendedor)
Actores Secundarios	
Disparador	El vendedor indica que quiere enviar el reembolso al comprador
Descripción	El vendedor podrá votar para que se le envié el reembolso al comprador
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia 3. El usuario debe haber accedido a la página de un producto CU-02 4. El producto tiene que estar en estado finalizado 5. El usuario debe haber añadido el producto a la tienda
Secuencia normal	1. El vendedor vota para enviar el reembolso al comprador 2. MetaMask le pide que pague la transferencia para que sea minada 3. El sistema agrega el voto al reembolso
Postcondiciones	1. El vendedor vuelve a la página del producto
Excepciones	2.1. El usuario no tiene dinero suficiente 2.1.1. El caso vuelve al paso 2 2.2. El usuario cancela el pago 2.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.7: CU-06 - Votar para enviar reembolso al comprador

CU-07	Votar por enviar dinero al vendedor
Versión	1.0
Actor Principal	Usuario Registrado (Vendedor)
Actores Secundarios	
Disparador	El vendedor indica que quiere que le envién el dinero de la venta
Descripción	El vendedor podrá votar para que se le envién el dinero
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia 3. El usuario debe haber accedido a la página de un producto CU-02 4. El producto tiene que estar en estado finalizado 5. El usuario debe haber añadido el producto a la tienda
Secuencia normal	1. El vendedor vota para reclamar el dinero de la venta 2. MetaMask le pide que pague la transferencia para que sea minada 3. El sistema agrega el voto al envió del pago de la compra
Postcondiciones	1. El vendedor vuelve a la página del producto
Excepciones	2.1. El usuario no tiene dinero suficiente 2.1.1. El caso vuelve al paso 2 2.2. El usuario cancela el pago 2.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.8: CU-07 - Votar para enviar dinero al vendedor

- **Comprador/Pujador**

CU-08	Comprar un producto
Versión	1.0
Actor Principal	Usuario Registrado (Comprador/Pujador)
Actores Secundarios	
Disparador	El comprador/pujador indica que quiere comprar un producto
Descripción	El comprador/pujador podrá comprar un producto que este en venta
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia 3. El usuario debe haber accedido a la página de un producto CU-02 4. El producto tiene que estar en tiempo de venta y sin vender
Secuencia normal	1. El comprador/pujador indica que quiere comprar el producto 2. MetaMask le pide que pague la transferencia para que sea minada 3. El sistema agrega la compra al sistema que será revelada al final del tiempo de venta
Postcondiciones	1. El comprador/pujador vuelve a la página del producto
Excepciones	2.1. El usuario no tiene dinero suficiente 2.1.1. El caso vuelve al paso 2 2.2. El usuario cancela el pago 2.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.9: CU-08 - Comprar un producto

CU-09	Hacer puja
Versión	1.0
Actor Principal	Usuario Registrado (Comprador/Pujador)
Actores Secundarios	
Disparador	El comprador/pujador indica que quiere hacer una puja
Descripción	El comprador/pujador podrá realizar una puja por algún producto de la tienda
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia 3. El usuario debe haber accedido a la página de un producto CU-02 4. El producto tiene que estar en tiempo de venta y sin vender
Secuencia normal	1. El comprador/pujador indica que quiere pujar por el producto 2. El sistema proporciona los campos necesarios para pujar 3. El usuario introduce los datos para realizar la puja 4. MetaMask le pide que pague la transferencia para que sea minada 5. El sistema agrega la puja por el producto
Postcondiciones	1. El comprador/pujador vuelve a la página del producto
Excepciones	3.1. El usuario proporciona datos erroneos 3.1.1. El sistema muestra un mensaje indicandoló 3.1.2. El caso vuelve al paso 2 3.2. El usuario no completa algún campo obligatorio 3.2.1. El sistema muestra un mensaje indicandoló 3.2.2. El caso vuelve al paso 2 4.1. El usuario no tiene dinero suficiente 4.1.1. El caso vuelve al paso 2 4.2. El usuario cancela el pago 4.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.10: CU-09 - Hacer puja

CU-10	Revelar puja
Versión	1.0
Actor Principal	Usuario Registrado (Comprador/Pujador)
Actores Secundarios	
Disparador	El comprador/pujador indica que quiere revelar una puja
Descripción	El comprador/pujador podrá relevar una puja por algún producto de la tienda
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia 3. El usuario debe haber accedido a la página de un producto CU-02 4. El producto tiene que estar en estado revelar 5. El usuario debe haber realizado una puja sobre el producto
Secuencia normal	1. El comprador/pujador indica que quiere revelar una pujar por el producto 2. El sistema proporciona los campos necesarios para revelar la puja 3. El usuario introduce los datos para revelar la puja 4. MetaMask le pide que pague la transferencia para que sea minada 5. El sistema revela la puja por el producto
Postcondiciones	1. El comprador/pujador vuelve a la página del producto
Excepciones	3.1. El usuario proporciona datos erroneos 3.1.1. El sistema muestra un mensaje indicandoló 3.1.2. El caso vuelve al paso 2 3.2. El usuario no completa algún campo obligatorio 3.2.1. El sistema muestra un mensaje indicandoló 3.2.2. El caso vuelve al paso 2 4.1. El usuario no tiene dinero suficiente 4.1.1. El caso vuelve al paso 2 4.2. El usuario cancela el pago 4.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.11: CU-10 - Revelar puja

CU-11	Votar por enviar reembolso al comprador
Versión	1.0
Actor Principal	Usuario Registrado (Comprador/Pujador)
Actores Secundarios	
Disparador	El comprador/pujador indica que quiere recibir un reembolso por la compra
Descripción	El comprador/pujador podrá votar por recibir el reembolso por la compra
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia 3. El usuario debe haber accedido a la página de un producto CU-02 4. El producto tiene que estar en estado finalizado 5. El usuario debe haber ganado el producto
Secuencia normal	1. El comprador/pujador indica que quiere un reembolso 2. MetaMask le pide que pague la transferencia para que sea minada 3. El sistema agrega el voto al reembolso
Postcondiciones	1. El comprador/pujador vuelve a la página del producto
Excepciones	4.1. El usuario no tiene dinero suficiente 4.1.1. El caso vuelve al paso 2 4.2. El usuario cancela el pago 4.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.12: CU-11 - Votar por enviar reembolso al comprador

CU-12	Votar por enviar dinero al vendedor
Versión	1.0
Actor Principal	Usuario Registrado (Comprador/Pujador)
Actores Secundarios	
Disparador	El comprador/pujador indica que quiere enviar el pago al vendedor
Descripción	El comprador/pujador podrá votar por enviar el dinero al vendedor
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia 3. El usuario debe haber accedido a la página de un producto CU-02 4. El producto tiene que estar en estado finalizado 5. El usuario debe haber ganado el producto
Secuencia normal	1. El comprador/pujador indica que quiere enviar el dinero al vendedor 2. MetaMask le pide que pague la transferencia para que sea minada 3. El sistema agrega el voto al envió del pago de la compra
Postcondiciones	1. El comprador/pujador vuelve a la página del producto
Excepciones	4.1. El usuario no tiene dinero suficiente 4.1.1. El caso vuelve al paso 2 4.2. El usuario cancela el pago 4.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.13: CU-12 - Votar por enviar dinero al vendedor

- **Árbitro**

  Los casos de uso *Votar por enviar reembolso al comprador* (CU-14) y el de *Votar por enviar dinero al vendedor* (CU-15) son iguales a los casos de uso del vendedor CU-06 y CU-07 y del comprador CU-11 y CU-12, por lo que se opta por no especificarlos.

CU-13	Finalizar subasta
Versión	1.0
Actor Principal	Usuario Registrado (Árbitro)
Actores Secundarios	
Disparador	El árbitro indica que quiere finalizar una subasta
Descripción	El árbitro podrá finalizar una subasta
Precondiciones	1. El usuario deberá estar registrado en MetaMask 2. El usuario deberá tener algo de ether para poder pagar la transferencia 3. El usuario debe haber accedido a la página de un producto CU-02 4. El producto tiene que estar en estado revelar y haber pasado el tiempo estipulado para revelar la subasta
Secuencia normal	1. El árbitro indica que quiere finalizar la subasta 2. MetaMask le pide que pague la transferencia para que sea minada 3. El sistema agrega el vendedor, el ganador de la subasta/compra y el árbitro que han intervenido
Postcondiciones	1. El árbitro vuelve a la página del producto
Excepciones	4.1. El usuario no tiene dinero suficiente 4.1.1. El caso vuelve al paso 2 4.2. El usuario cancela el pago 4.2.1. El caso vuelve al paso 2
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.14: CU-13 - Finalizar subasta

- **Dueño del contrato**

  Los casos de uso *Votar por enviar dinero al comprador en una venta directa* (CU-17) y el de *Votar por enviar dinero al vendedor en una compra directa* (CU-18) son iguales a los casos de uso del vendedor CU-06 y CU-07 y del comprador CU-11 y CU-12, por lo que se opta por no especificarlos.

- **Reloj del sistema**

CU-19	Cambiar estado del producto
Versión	1.0
Actor Principal	Reloj del sistema
Actores Secundarios	
Disparador	El tiempo de subasta de un producto llega a su fin
Descripción	El reloj del sistema cambia el estado de un producto para que se muestre en el lugar apropiado en la aplicación
Precondiciones	1. Tener al menos un producto en la web
Secuencia normal	1. El reloj del sistema verifica la fecha de finalización del producto 2. El reloj del sistema actualiza el estado si fuera necesario
Postcondiciones	1. El producto cambia de posición a la sección que le toque
Excepciones	
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.15: CU-19 - Cambiar estado del producto

CU-20	Cambiar funciones permitidas
Versión	1.0
Actor Principal	Reloj del sistema
Actores Secundarios	
Disparador	El tiempo de subasta de un producto llega a su fin
Descripción	El reloj del sistema cambia las funciones que se pueden realizar sobre un producto
Precondiciones	1. Tener al menos un producto en la web
Secuencia normal	1. El reloj del sistema verifica la fecha de finalización del producto 2. El reloj del sistema actualiza el estado si fuera necesario
Postcondiciones	1. El producto cambia la información disponible en la vista del producto
Excepciones	
Prioridad	Alta
Casos relacionados	CU-02
Comentarios	

Tabla 6.16: CU-20 - Cambiar funciones permitidas

6.3. Requisitos de información

Los requisitos de información describen todos los aspectos relacionados con los datos creados, gestionados o emitidos por el sistema.

Los requisitos de información se modelan a través del diagrama entidad-relación y se detallan mediante el diccionario de datos.

Los requisitos de información obtenidos tras el análisis son:

- **RI-01**: El Blockchain almacenará los datos del producto.
- **RI-02**: El Blockchain almacenará los datos de las pujas.
- **RI-03**: El contrato de Fideicomiso almacenará el dinero antes de enviarlo.
- **RI-04**: IPFS almacenará los datos de las imágenes y de la descripción.
- **RI-05**: MongoDB almacenará los datos de los productos para mostrarlos cuando no haya conexión con la Blockchain a los usuarios generales.

6.3.1. Modelo conceptual de datos

Para modelar la información con la que se trabajará, se hace un modelo de ER que va a ser representativo, ya que, no vamos a controlar todas las entidades que hay con nuestra aplicación.

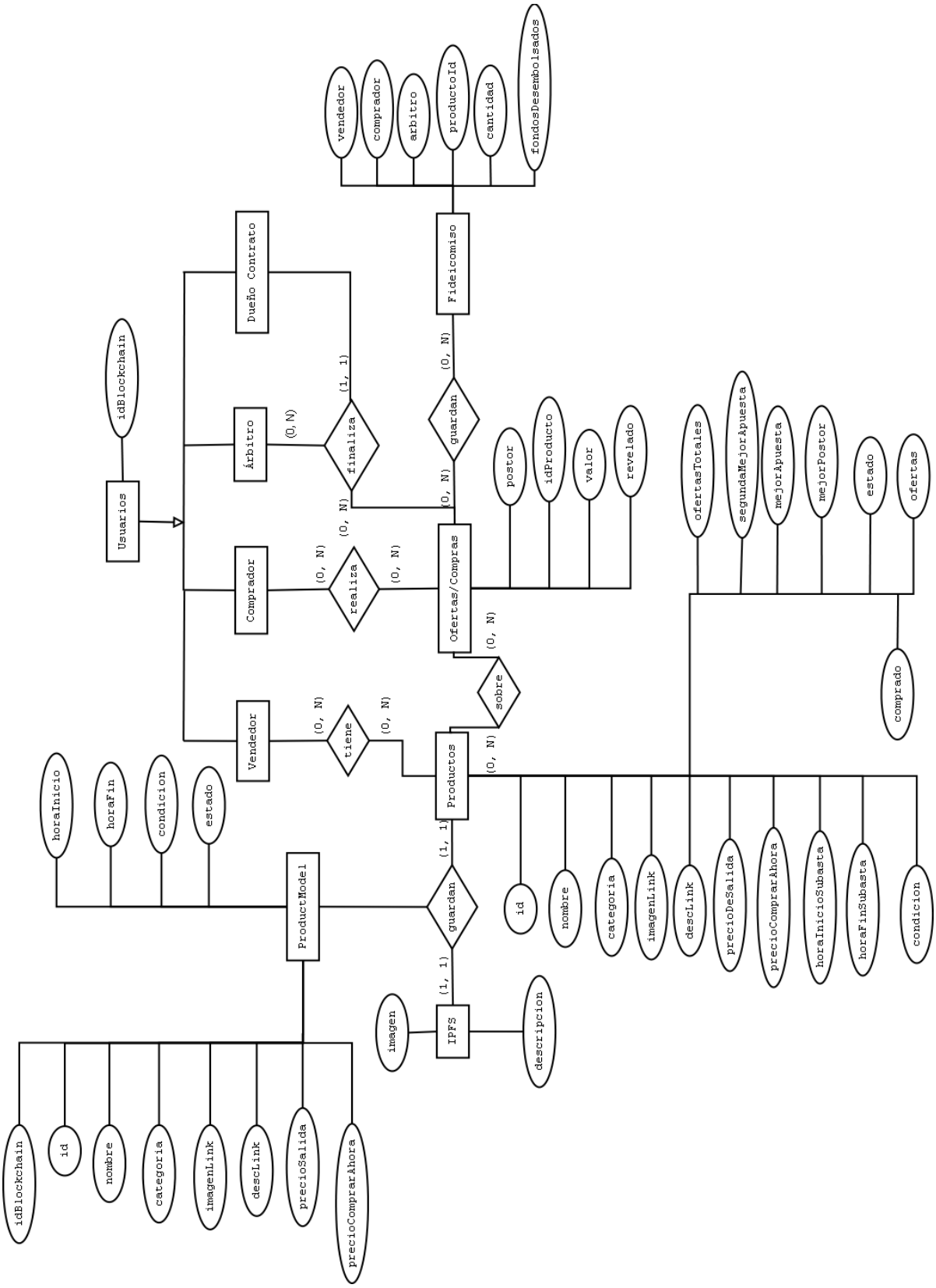


Figura 6.8: Diagrama Entidad-Relación

6.3.2. Diccionario de datos

En este punto se describirán las diferentes entidades almacenadas en la Blockchain y en la base de datos MongoDB:

Blockchain

Entidad Usuarios			
Representa a los usuarios que pueden hacer uso de la aplicación desde la cadena de bloques			
Nombre	**Descripción**	**Tipo de dato**	**Dominio**
idBlockchain	Dirección que apunta a un usuario de Blockchain	address	Dirección Blockchain Ethereum

Tabla 6.17: Entidad Usuarios

Los usuarios deben estar registrados en el Blockchain y no se registraran en la aplicación. La aplicación solo guardará la dirección de Ethereum del usuario cuando pase a hacer uso de la aplicación que se guardará para saber que rol desempeña.

Entidad Ofertas/Compras			
Representa a una oferta/compra que ha sido realizada			
Nombre	**Descripción**	**Tipo de dato**	**Dominio**
postor	Dirección de la cuenta del usuario postor	Address	Dirección Blockchain Ethereum
idProducto	Identificador del producto en la tienda	uint	Número
valor	Valor de la puja enviada	uint	Número
revelado	Indica si una oferta a sido revelada o no	bool	Verdadero o Falso

Tabla 6.18: Entidad Ofertas/Compras

Las ofertas y las compras se realizan sobre los productos y puede que haya más de una oferta por producto de distintos usuarios, pero sí se realiza una compra se finaliza la subasta.

Entidad Productos
Representa a un producto en la tienda

Nombre	Descripción	Tipo de dato	Dominio
id	Id autoincremental único para cada producto	uint	Número
nombre	Nombre identificativo del producto	string	Cadena de caracteres
categoria	Nombre identificativo de la categoria	string	Cadena de caracteres
imagenLink	Link donde se encuentra la imagen	string	Cadena de caracteres
descLink	Link donde se encuentra la descripción	string	Cadena de caracteres
horaInicioSubasta	Número identificativo del inicio de la subasta	uint	Número
horaFinSubasta	Número identificativo del fin de la subasta	uint	Número
precioDeSalida	Número idincativo del precio inicial de la subasta	uint	Número
precioComprarAhora	Número idincativo del precio de comprar ahora	uint	Número
mejorPostor	Dirección identificativa del mejor postor	address	Dirección Blockchain Ethereum
mejorApuesta	Número idincativo de la mejor apuesta	uint	Número
segundaMejorApuesta	Número idincativo de la mejor apuesta	uint	Número
ofertasTotales	Número autoincremental de las ofertas realizadas al producto	uint	Número
estado	Indica el estado de venta de un producto	enum	- Enventa - Vendido - Sin vender
condicion	Indica el estado de procedencia de un producto	enum	- Nuevo - Usado
comprado	Indica el estado de compra de un producto	bool	Verdadero o falso
ofertas	Indica la oferta realizada por un postor	mapping	Oferta

Tabla 6.19: Entidad Productos

Los productos reciben ofertas de los usuarios y cuando finaliza se crea el fideicomiso con el vendedor, el comprador y el árbitro de un producto.

Entidad Fideicomiso
Representa el fideicomiso de un producto

Nombre	Descripción	Tipo de dato	Dominio
productoId	Identifica a un producto de la tienda	uint	Número
comprador	Identifica la dirección Blockchain del comprador	address	Dirección Blockchain Ethereum
vendedor	Identifica la dirección Blockchain del vendedor	address	Dirección Blockchain Ethereum
arbitro	Identifica la dirección Blockchain del árbitro	address	Dirección Blockchain Ethereum
cantidad	Indica la cantidad de dinero por la que se vendio el producto	uint	Número
fondosDesembolsados	Indica si se ha liberado el dinero del fideicomiso o no	bool	Verdadero o falso

Tabla 6.20: Entidad Fideicomiso

El fideicomiso es sobre un producto e indica quien vende y compra el producto y quien árbitra la oferta en caso de disputa.

Es el encargado de almacenar el pago hasta que la compra haya finalizado correctamente.

Entidad IPFS
Representa el hash que se almacena en IPFS de los productos

Nombre	Descripción	Tipo de dato	Dominio
imagen	Dirección hash que apunta a la imagen en IPFS	string	Cadena de caracteres
descripcion	Dirección hash que apunta a la descripción en IPFS	string	Cadena de caracteres

Tabla 6.21: Entidad IPFS

El IPFS es el encargado de almacenar la imagen y la descripción del producto en la cadena de bloques, y guardar en nuestro contrato el hash donde se encuentra para poder recuperarlo y mostrarlo con posterioridad.

MongoDB

Entidad ProductModel			
Representa el producto fuera de la Blockchain			
Nombre	**Descripción**	**Tipo de dato**	**Dominio**
idBlockchain	Identifica el id del producto en la Blockchain	Number	Número
nombre	Indica el nombre del producto	String	Cadena de caracteres
categoria	Indica la categoría del producto	String	Cadena de caracteres
ipfsImagenHash	Identifica el hash del link de la imagen en el IPFS	String	Cadena de caracteres
ipfsDescHash	Identifica el hash del link de la descripción en el IPFS	String	Cadena de caracteres
horaInicioSubasta	Indica la fecha del inicio de la subasta	Number	Número
horaFinSubasta	Indica la fecha de finalización de la subasta	Number	Número
precioDeSalida	Indica el precio del inicio de la subasta	Number	Número
precioComprarAhora	Indica el valor del producto para comprar ahora	Number	Número
condicion	Indica el estado del producto	Number	Número
estado	Indica el estado de compra del producto	Number	Número

Tabla 6.22: Entidad ProductModel

Representa a los productos de la Blockchain para que los usuarios que no estén registrados en MetaMask puedan verlos a forma de catalogo.

.

Capítulo 7

Diseño

En este capítulo se expondrá el trabajo de diseño realizado previo a la creación de la aplicación web. Se tratarán las arquitecturas lógicas y físicas de la aplicación al igual que los diseños de las pantallas de la interfaz.

7.1. Arquitectura lógica

El entorno de desarrollo se ve influenciado por los clientes más avanzados para Ethereum desarrollados en JavaScript.

Teniendo en cuenta estos factores, stack tecnológicos como MEAN JS parecen proporcionar una solución que se adapta a estas necesidades:

- **MongoDB:** Base de datos NoSQL orientada a documentos. Su propósito es el de servir de base de datos off-chain.
- **ExpressJs:** Framework para aplicaciones web.
- **AngularJS:** Framework que permite extender el comportamiento de HTML para aplicaciones web dinámicas. En el stack aplica a la capa de frontend.
- **Node.js:** Plataforma que permite crear tu propio servidor web y construir aplicaciones. Tiene un único hilo de ejecución, utilizando entradas y salidas asíncronas. NPM es el gestor de librerías/paquetes.

Es importante destacar que MEAN JS no tiene que ser necesariamente un stack inamovible. Por ejemplo, reemplazando el frontend AngularJS por otro diferente como Bootstrap. En este caso, por el tiempo reducido, el frontend no se ha desarrollado con AngularJS sino con HTML, CSS, JavaScript y Bootstrap. Adicionalmente a todo esto, y de cara a facilitar el despliegue de las aplicaciones/pruebas de concepto, una de las herramientas que permite realizar esta tarea de una forma más sencilla y orientada a microservicios es Docker, aunque en este proyecto no se utilizará.

Para la escalabilidad, y de cara a la promoción a entornos productivos, se puede emplear una plataforma cloud como AWS o Azure, permitiendo de esta forma, gestionar el hosting de una aplicación Node.JS, y, integrar el código de una forma cercana al plug and play si se utiliza Docker.

Un ejemplo de integración con Ethereum sería el siguiente:

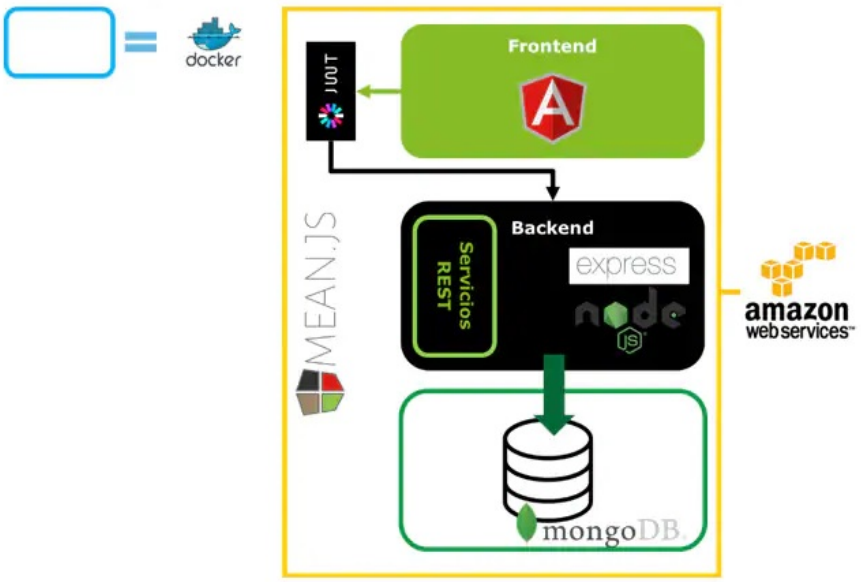


Figura 7.1: Ejemplo de integración con Ethereum [12]

Teniendo esto en cuenta, un ejemplo de arquitectura lógica a alto nivel de Ethereum para una aplicación web descentralizada utilizando Parity como cliente Ethereum y utilizando el stack tecnológico mencionado anteriormente, quedaría de la siguiente manera:

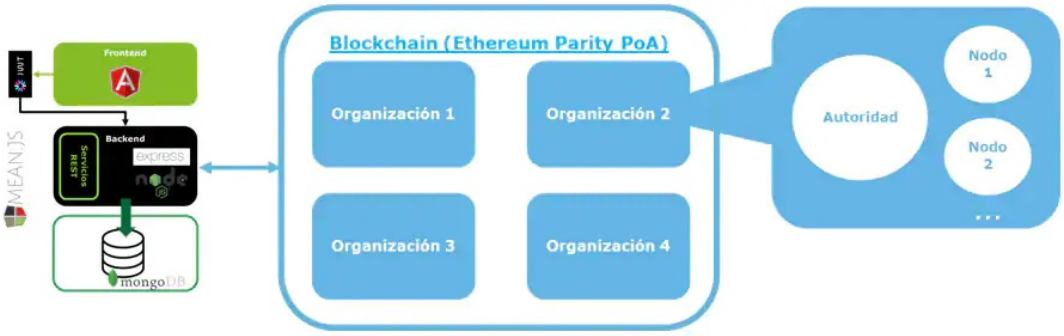


Figura 7.2: Arquitectura lógica de alto nivel [12]

En la *figura 6.2* podemos ver como sería una arquitectura lógica de alto nivel ideal para una aplicación descentralizada, sin embargo, en el proyecto no se a realizado exactamente de esta manera, aunque se a intentado que fuera lo más parecida posible.

Nuestra arquitectura lógica por lo tanto queda de la siguiente manera:

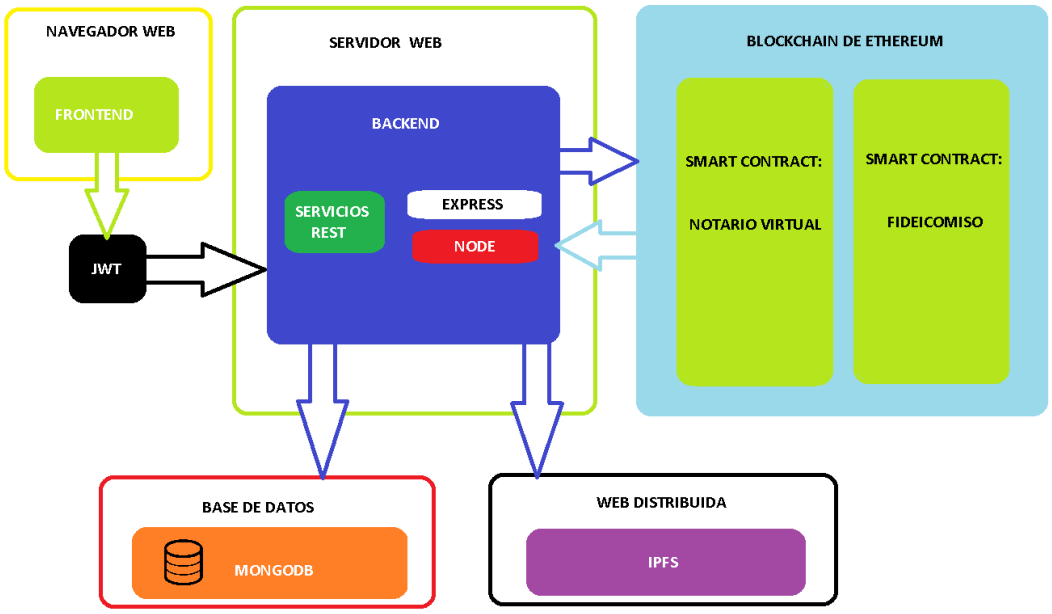


Figura 7.3: Arquitectura lógica de la aplicación

Como se puede observar la arquitectura es parecida a la cliente-servidor de una aplicación convencional. Esta arquitectura consiste en que un cliente envía una serie de peticiones al servidor las cuales generan unas respuestas, que son enviadas de nuevo al cliente.

En nuestro lado cliente tendremos al navegador web, ya que, es la forma en la que el usuario interactua con la aplicación. Respecto al lado servidor web, la iteracción se realizará en dos partes, ya que, usamos una base MongoDB para mostrar los productos y así poderles aplicar diferentes filtros, y por otra parte, utilizaremos los smart contract para realizar las funcionalidades que será nuestra capa que se asemeja a la capa del Modelo en una capa Modelo-Vista-Controlador, es decir, se encargara de las operaciones que se realizan con los datos; y también tendremos la billetera, en este caso MetaMask, que será como la capa de Control, ya que será la encargada de controlar el inicio de sesión del usuario y ver si tiene el dinero suficiente, al igual que sirve para unir nuestra aplicación web con el Blockchain para que puedan interactuar a través de Web3.

También hacemos uso de objetos JSON para comunicar la parte cliente y servidor.

Para terminar con la arquitectura lógica, queda detallar que en la base de datos solo se guardara una copia de los productos subidos a la Blockchain para que sirva de modo catalogo y poder trabajar con ellos, pero la verdadera base de datos donde se trabaja con los datos es el Blockchain, que es donde ocurre todo.

7.2. Arquitectura física

La arquitectura física implementada para la Dapp desarrollada consta de 5 componentes:

1. **Interfaz web:** es una combinación de HTML, CSS, JavaScript y Bootstrap, en la parte de JavaScript se hace un uso masivo de web3js. Los usuarios interactuarán con el Blockchain, IPFS y el servidor NodeJS a través de esta aplicación frontend.

2. **Blockchain:** es el corazón de la aplicación descentralizada, y es donde viven todos los códigos, transacciones y datos. Todos los productos de la tienda, las ofertas de los usuarios y el depósito en garantía viven en la cadena de bloques de Ethereum.

3. **MongoDB:** aunque los productos se almacenan en el Blockchain, no es eficiente consultar el Blockchain para mostrar los productos y aplicar filtros (mostrar solo productos de una categoría específica, ordenar por fecha de expiración, etc). En su lugar utilizamos la base de datos MongoDB para almacenar la información del producto y consultarla para mostrar los productos.

4. **Servidor NodeJS:** este es el servidor backend a través del cual el frontend se comunica con la base de datos. Utilizaremos algunas API's simples para que la interfaz pueda consultar y recuperar productos de la base de datos.

5. **IPFS:** se utilizará para cuando un usuario quiera ver un producto de la tienda, la interfaz cargará la imagen y la descripción del producto en el IPFS y almacenará el hash del archivo cargado en la cadena de bloques.

Por lo tanto, nuestra arquitectura física será:

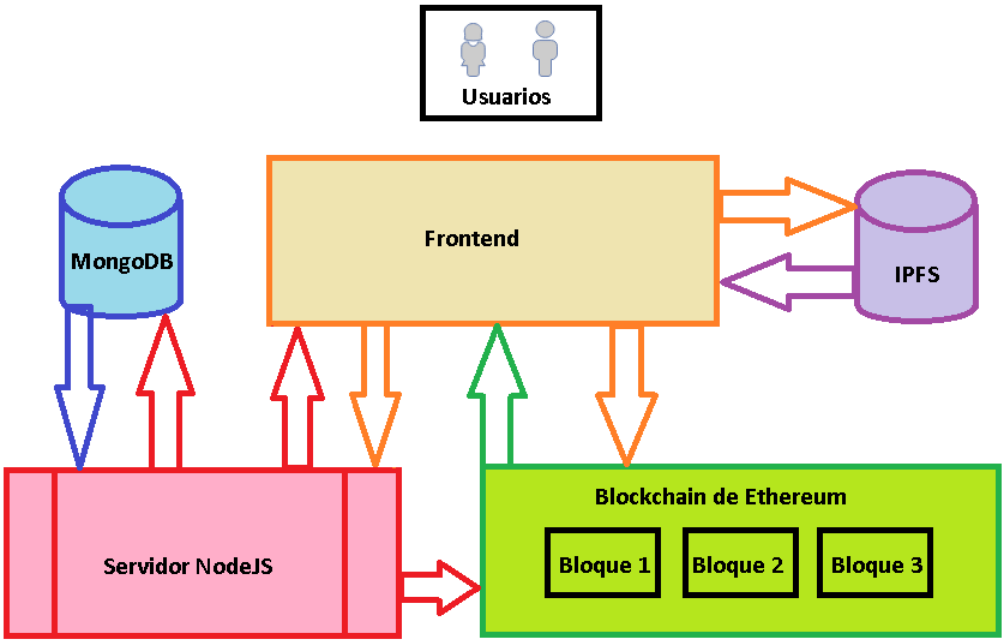


Figura 7.4: Arquitectura física de la aplicación

7.3. Flujo de la aplicación

Para dar sentido a todos los componentes vistos en la arquitectura física, vamos a ver lo que sucede cuando un usuario quiere subir un producto a la tienda:

1. La interfaz web contendrá un formulario HTML donde el usuario ingrese los detalles del producto (nombre, precio de inicio, imagen, descripción, etc) y hace click en guardar. (1)

2. La interfaz web carga la imagen y la descripción del producto en el IPFS y recupera los enlaces de esos archivos cargados. (2) y (3)

3. La interfaz web invoca al contrato inteligente para almacenar la información del producto más los enlaces de IPFS en la cadena de bloques. Una vez se ha guardado con éxito el producto en la cadena de bloques, el contrato envía un evento. El evento contiene toda la información del producto. (4) y (5)

4. El servidor NodeJS está configurado para escuchar estos eventos y cuando un evento es enviado por el contrato, el servidor lee el contenido del evento e inserta el producto en la base de datos MongoDB. (6), (7) y (8)

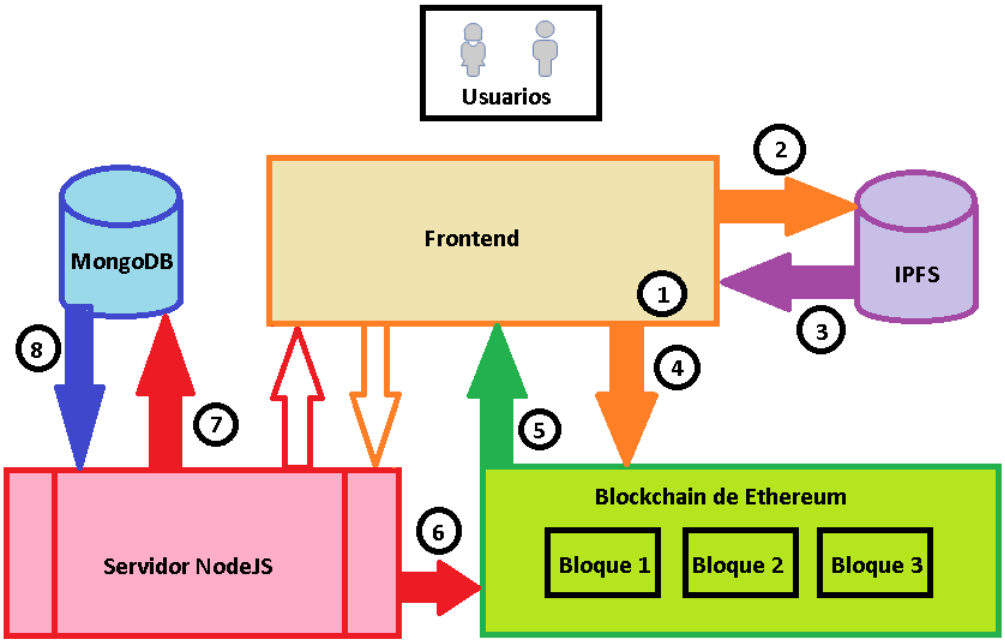


Figura 7.5: Flujo de la aplicación para subir un producto

7.3.1. Diagramas de Secuencia

En está parte se verán algunos diagramas de secuencia de la aplicación, con ellos se muestra la interacción entre los distintos objetos del sistema.

1. **Añadir producto**

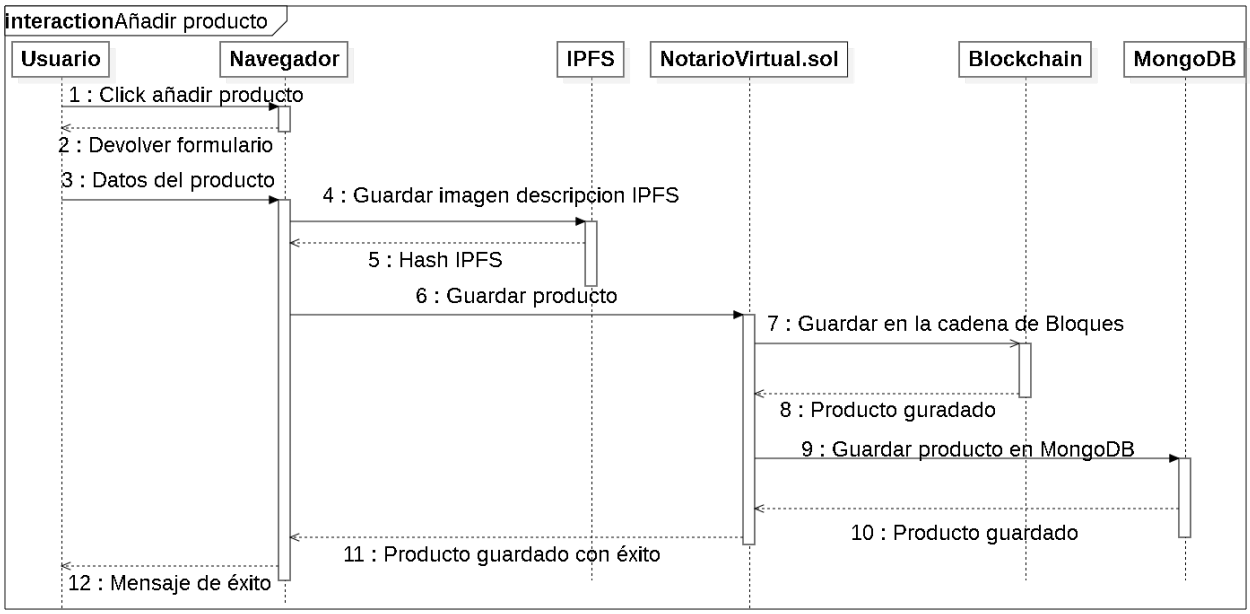


Figura 7.6: Diagrama de secuencia - Añadir Producto

2. **Realizar puja**

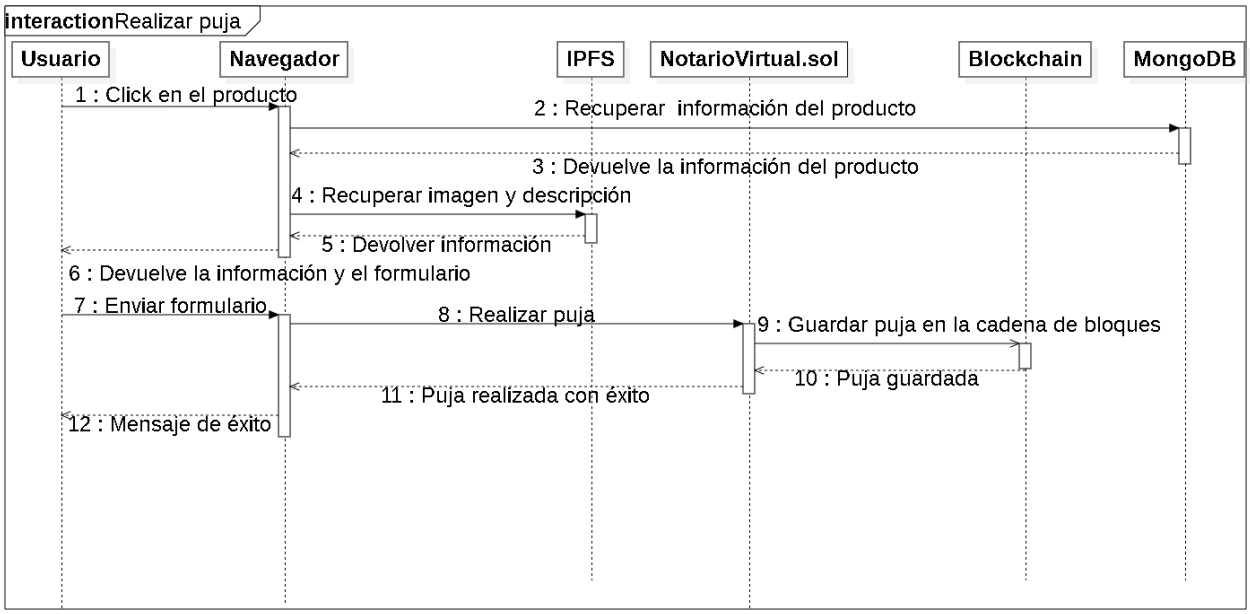


Figura 7.7: Diagrama de secuencia - Hacer puja

3. **Comprar producto**

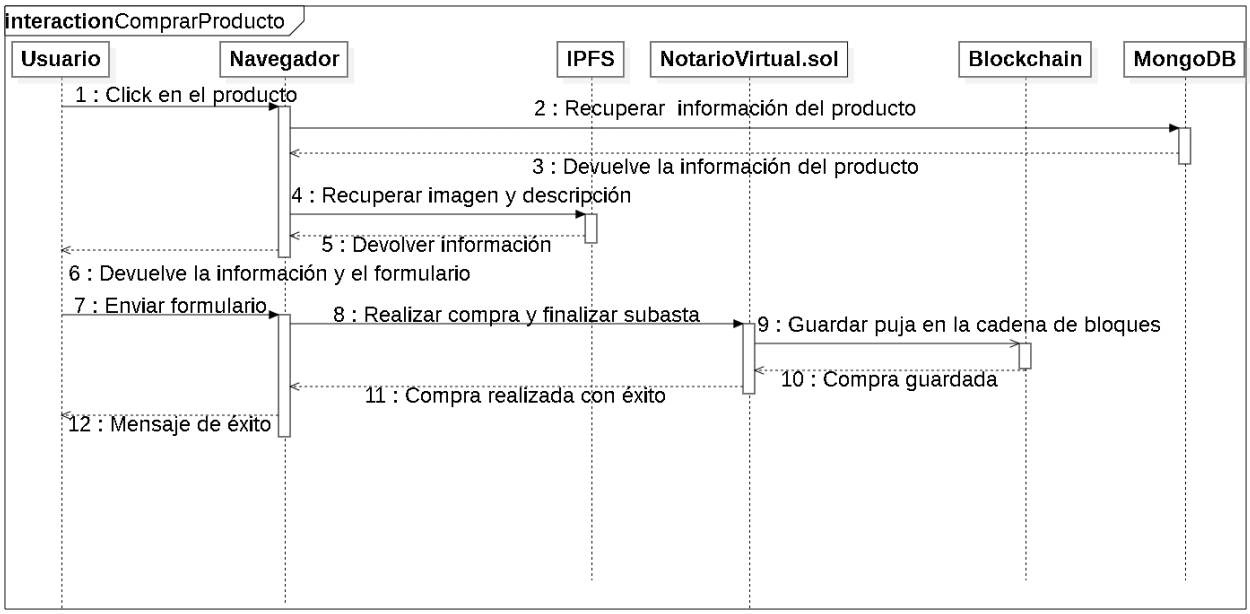


Figura 7.8: Diagrama de secuencia - Comprar producto

7.4. Diseño de la interfaz

El diseño de la interfaz para la aplicación web se ha realizado con el objetivo de lograr una navegabilidad sencilla e intuitiva. De este modo, los usuarios con menos conocimiento, podrán hacer uso de la aplicación sin ningún problema.

El color que se ha elegido como tema de la aplicación web es blanco, ya que queremos dejar una interfaz lo más limpia posible para que se centre en la parte principal, los productos.

En cuanto a la navegabilidad, se ha diseñado un menú con todas las funciones consideradas como principales, y de especial interés para los usuarios. Esté menú puede visualizarse en cualquier momento durante el uso de la aplicación web, ya que se encuentra en la parte superior de la pantalla.

Se ha utilizado un diseño responsive (gracias a la utilización de Bootstrap), para que se pueda hacer a la web con cualquier dispositivo y que de esta forma se adapte perfectamente a nuestra pantalla.

A continuación se desarrolla la explicación detallada de algunas de las pantallas que componen la aplicación con los distintos dispositivos (Smartphone, Tablet y PC).

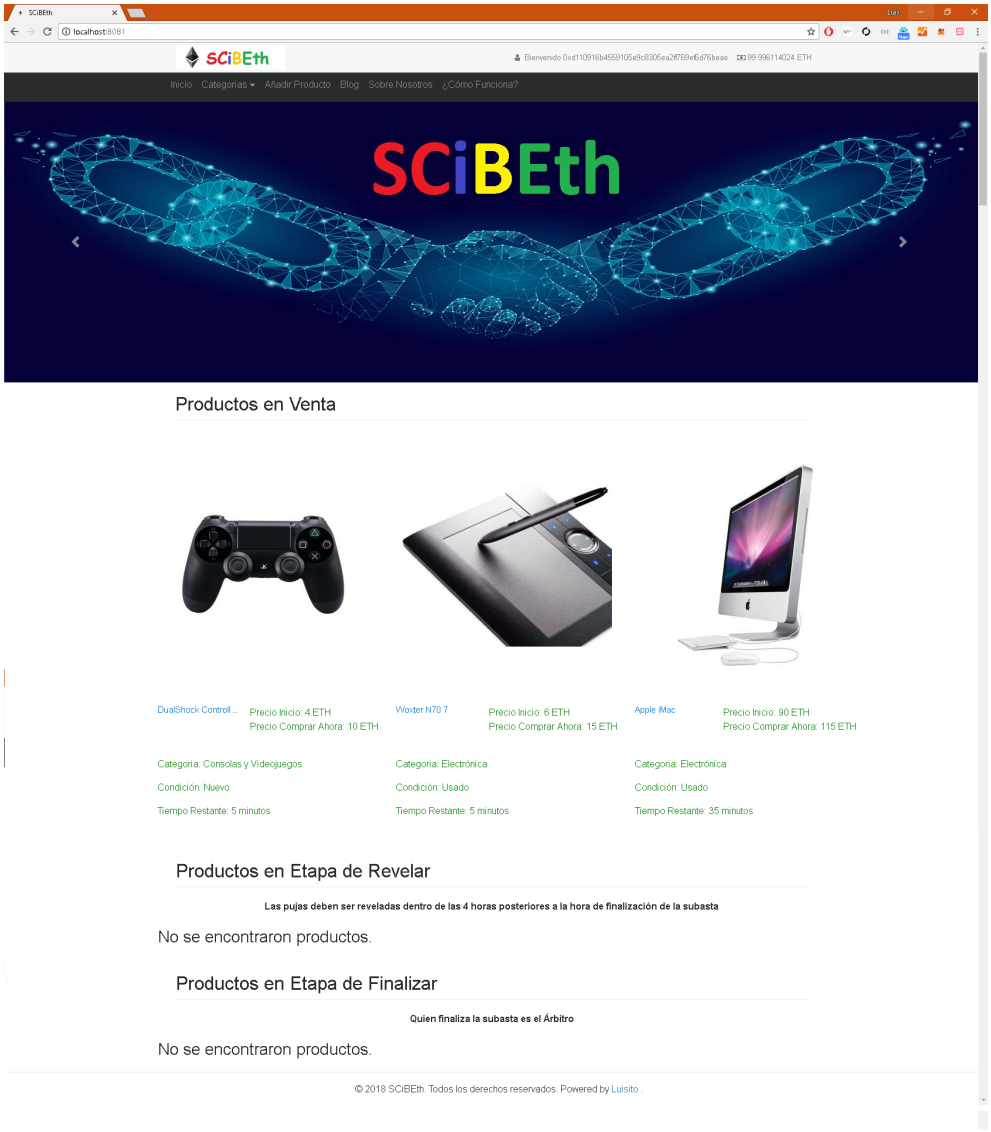
	Inicio desde PC
Descripción	Pantalla de inicio de la web.
Activación	Acceso al index de la web.
Diseño	
Eventos	Acceso a todas las secciones disponibles en el menú. Acceso a cada uno de los productos de la web.

Figura 7.9: Index de la aplicación web desde PC

Tabla 7.1: Diseño inicio PC

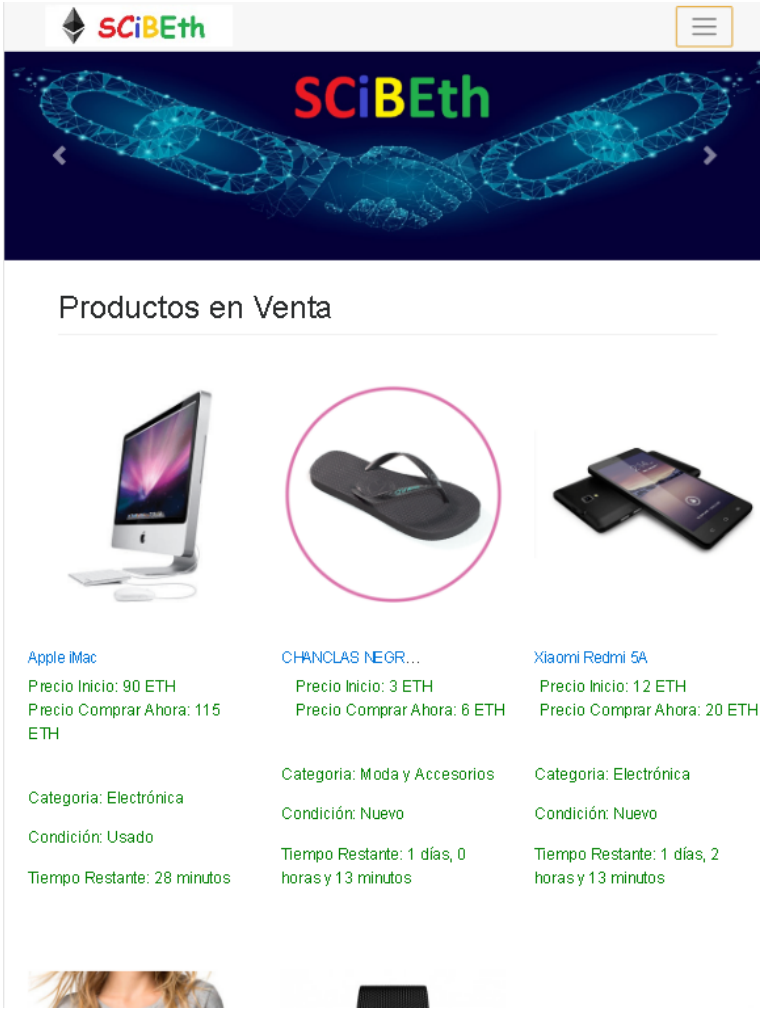
	Inicio desde Tablet
Descripción	Pantalla de inicio de la web visto desde una tablet.
Activación	Acceso al index de la web desde tablet.
Diseño	
Eventos	Acceso a todas las secciones disponibles en el menú. Acceso a cada uno de los productos de la web.

Figura 7.10: Index de la aplicación web desde Tablet

Tabla 7.2: Diseño inicio Tablet

	Inicio desde Smartphone
Descripción	Pantalla de inicio de la web visto desde un smartphone.
Activación	Acceso al index de la web desde smartphone.
Diseño	
	Figura 7.11: Index de la aplicación web desde Smartphone
Eventos	Acceso a todas las secciones disponibles en el menú. Acceso a cada uno de los productos de la web.

Tabla 7.3: Diseño inicio Smartphone

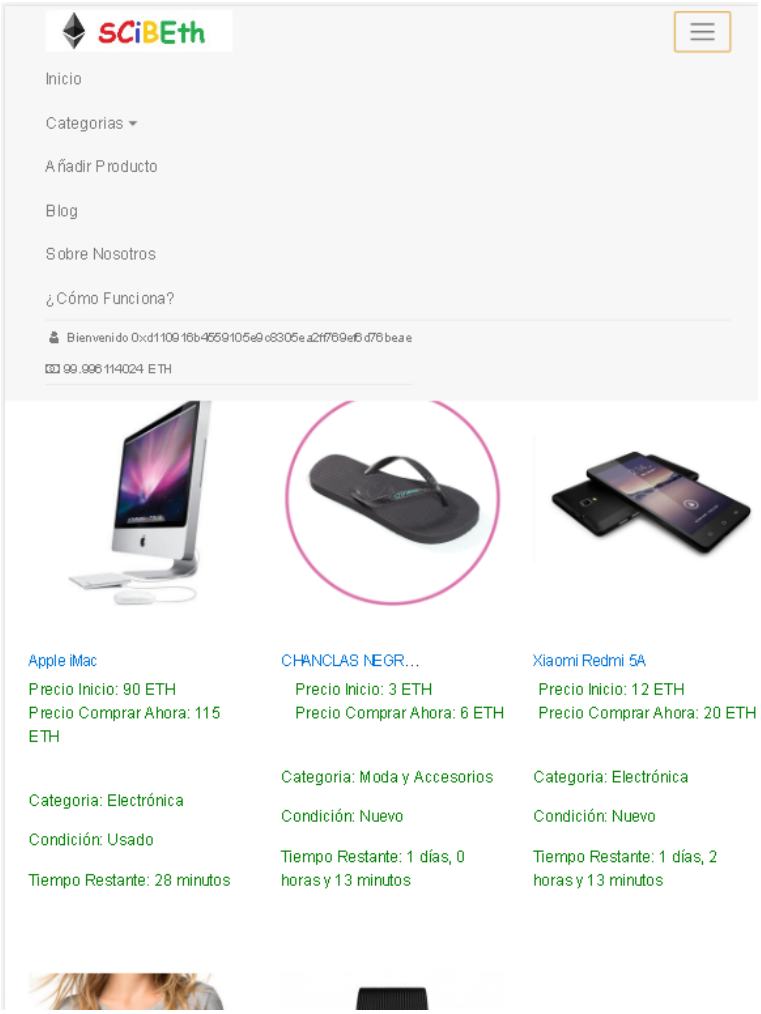
Menú de inicio desde Tablet	
Descripción	Menú desde la pantalla de inicio en una tablet
Activación	Al presionar en el icono de menú que aparece en la esquina superior derecha.
Diseño	
Eventos	Acceso a todas las secciones disponibles en el menú. Acceso a cada uno de los productos de la web.

Figura 7.12: Menú desde pantalla inicio tablet

Tabla 7.4: Diseño menú Tablet

	Menú de inicio desde Smartphone
Descripción	Menú desde la pantalla de inicio en un smartphone
Activación	Al presionar en el icono de menú que aparece en la esquina superior derecha.
Diseño	 Figura 7.13: Menú desde pantalla inicio smartphone
Eventos	Acceso a todas las secciones disponibles en el menú. Acceso a cada uno de los productos de la web.

Tabla 7.5: Diseño menú Smartphone

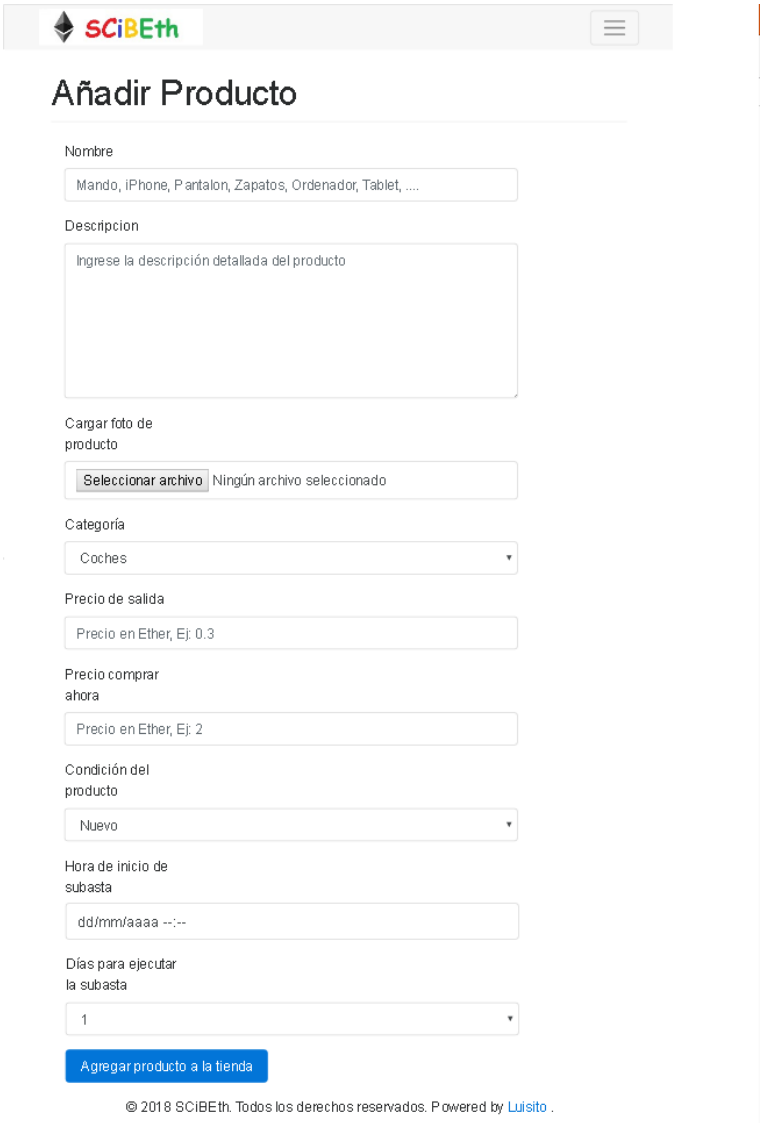
Añadir producto desde Tablet	
Descripción	Se muestra el formulario para agregar una producto a la tienda
Activación	Al presionar sobre la opción del menú: *Añadir producto*
Diseño	
	Figura 7.14: Pantalla añadir producto desde tablet
Eventos	Acceso a todas las secciones disponibles en el menú. Formulario para añadir un producto a la tienda.

Tabla 7.6: Diseño menú Tablet

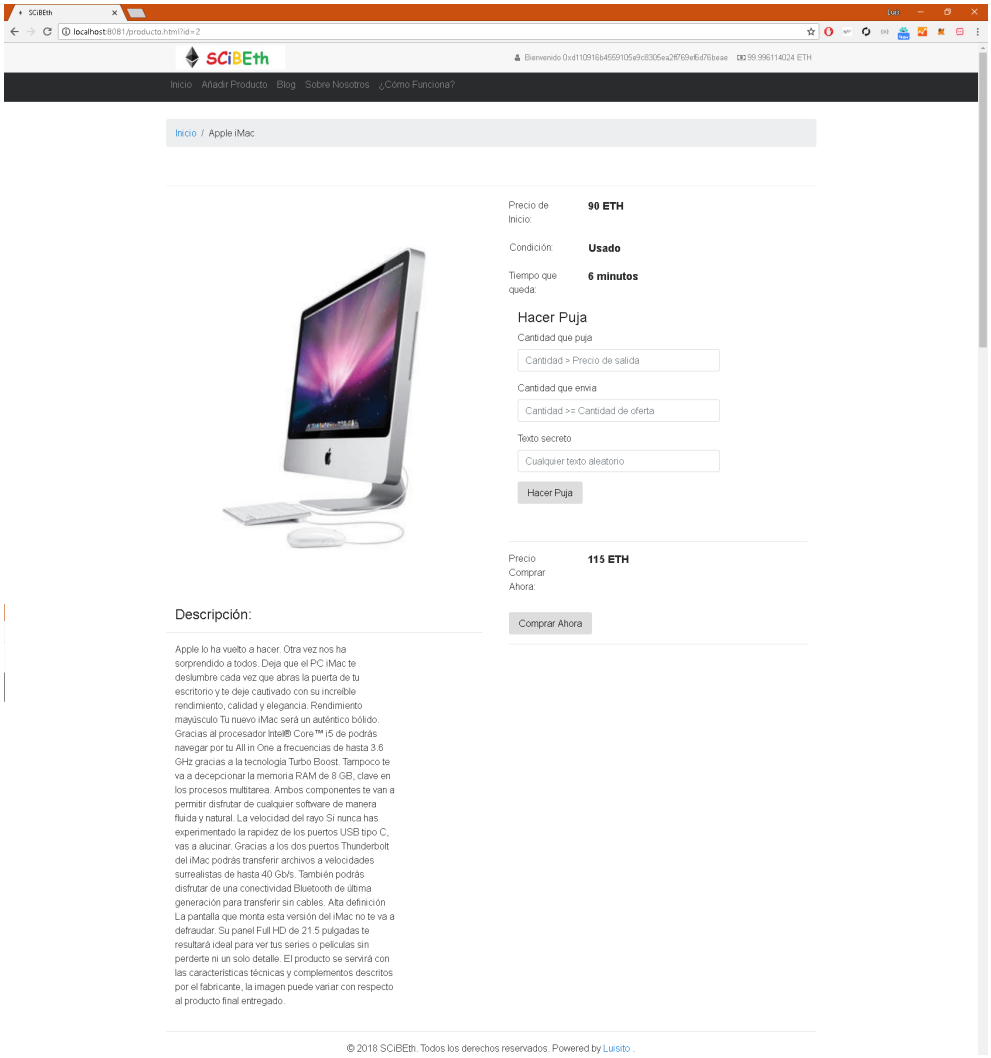
	Pantalla detalle de un producto en venta
Descripción	Se muestra un producto de la tienda al detalle
Activación	Al presionar sobre el nombre o la imagen de un producto en el inicio
Diseño	

Figura 7.15: Pantalla detalles producto

Eventos	Acceso a todas las secciones disponibles en el menú. Acceso a los detalles de un producto. Pujar por el producto. Comprar el producto.

Tabla 7.7: Diseño pantalla detalle de un producto

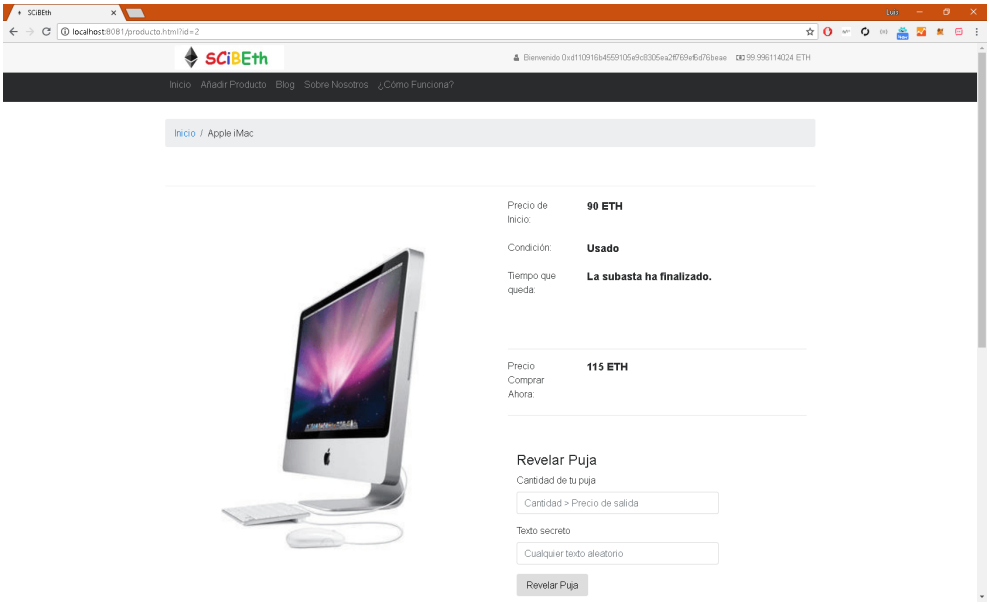
	Pantalla detalle de un producto en etapa de revelar
Descripción	Se muestra un producto de la tienda al detalle en la etapa de revelar
Activación	Al presionar sobre el nombre o la imagen de un producto en el inicio en etapa de revelar
Diseño	
	Figura 7.16: Pantalla detalles producto etapa revelar
Eventos	Acceso a todas las secciones disponibles en el menú. Acceso a los detalles de un producto. Revelar puja.

Tabla 7.8: Diseño pantalla detalle de un producto en etapa de revelar

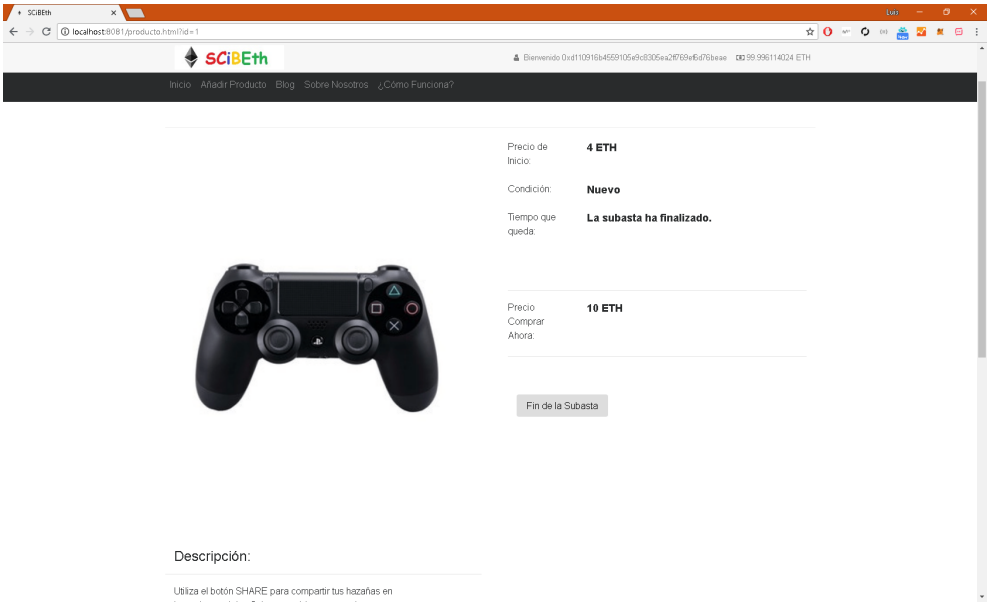
	Pantalla detalle de un producto en etapa de finalizar
Descripción	Se muestra un producto de la tienda al detalle en etapa de finalizar
Activación	Al presionar sobre el nombre o la imagen de un producto en el inicio en etapa de finaliza
Diseño	
	Figura 7.17: Pantalla detalles producto en etapa de finalizar
Eventos	Acceso a todas las secciones disponibles en el menú. Acceso a los detalles de un producto. Finalizar subasta

Tabla 7.9: Diseño pantalla detalle de un producto en etapa de finalizar

Estos son los diseños más representativos de nuestra plataforma, aunque se podrían poner más, no se considera que vayan a aportar más información, ya que el diseño sería igual, pero lo que cambia es la información que contiene.

.

Capítulo 8

Implementación

En este capítulo, se procede a explicar el desarrollo llevado a cabo a lo largo del proyecto. En primer lugar veremos la implementación de la aplicación desarrollada, mostrando su estructura y funcionamiento.

8.1. Estructura del proyecto

En siguiente figura se detalla la estructura de la aplicación web.

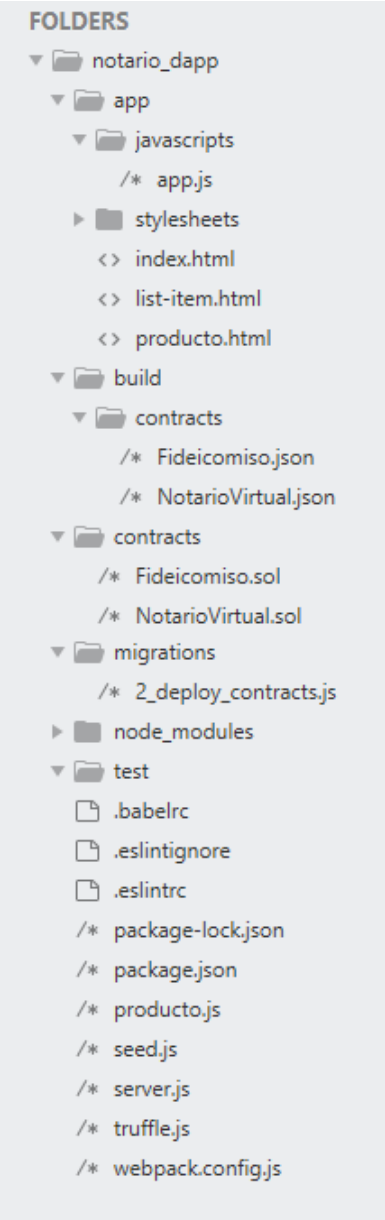


Figura 8.1: Estructura de la aplicación web

Vamos a ver que contiene cada carpeta del proyecto:

- **app:** Este directorio contiene:

    1. **javascripts/app.js:** archivo javascript con las funciones necesarios para comunicar nuestra pagina HTML con nuestro contrato haciendo uso de la librería web3.
    2. **stylesheets/app.css**: estilos para nuestra página HTML
    3. Los **archivos HTML** de nuestras páginas web.

- **build:** Dentro de la carpeta build tenemos otra carpeta llamada contracts, la cual contiene cada uno de los contratos compilados generando todos los datos del contrato a un archivo .json, el cual se usara para migrar el contrato inteligente e interactuar con él.
- **contracts:** aquí estará el código fuente de nuestros smart contracts.
- **migrations:** En este directorio Truffle busca los scripts que utilizara para desplegar nuestros smart contracts a su posterior entorno de ejecución. Los scrips de despliegue están priorizados (fijados en el prefijo numérico de los archivos que hay dentro) y se ejecutaran en dicho orden.
- **node modules:** En está carpeta se encuentran los módulos de NodeJS que utilizamos.
- **test:** Como buen "framework "de desarrollo, Truffle nos ha generado una serie tests unitarios para que podamos tener integración continua (CI) en nuestro proyecto. Para ello, Truffle utilizara el paquete Mocha de pruebas unitarias en NodeJS.
- **package.json:** Es el fichero que contiene cada uno de los framework de npm utilizados para el proyecto y que sirve para instalarlos, por ejemplo, mongoose.
- **producto.js:** Contiene el Schema que se va a seguir para guardar los productos en la base de datos de MongoDB.
- **seed.js:** Archivo de JavaScript que contiene un script para cargar productos en la cadena de bloques a modo de pruebas. Solo se utilizo para probar que se subían los productos antes de tener la parte frontend.
- **server.js:** Sirve para crear una aplicación ExpressJS y que comience a escuchar las solicitudes en el puerto 3000.
- **truffle.js:** Archivo de JavaScript que contiene la información de la red de Blockcahin a la que se va a migrar nuestra aplicación y nuestros contratos.
- **webpack.config.js:** Archivo de JavaScript que contiene los script de configuración de nuestro proyecto, como por ejemplo, las rutas que se van a seguir para ir a cada página web.

8.2. Funcionamiento de la DAPP

En este apartado se van a describir las partes del código que han sido claves para el correcto funcionamiento de la aplicación. Lo dividiremos en dos apartados, lo perteneciente al Backend, en este caso, los contratos inteligentes; y lo perteneciente al cliente, en este caso las funciones JavaScript.

- ***Contratos inteligentes***: En este caso nos encontramos con dos contratos inteligentes, **NotarioVirtual.sol** y **Fideicomiso.sol**. En el primer contrato inteligente es el principal, en el se encuentran las funciones de añadir un producto a la tienda, obtener un producto, pujar, revelar las ofertas, información sobre la puja más alta, finalizar la subasta, comprar un producto y añadir una dirección de fideicomiso para un producto, después

están las funciones que necesitan del contrato de fideicomiso, que son, información sobre el fideicomiso, liberar la cantidad al vendedor y reembolsar la cantidad al comprador.

El contrato de fideicomiso, es el encargado de guardar el dinero para posteriormente liberarlo al vendedor o reembolsarselo al comprador.

- ***Funciones JavaScript***: Las funciones de JavaScript son las que unen el contrato inteligente con el Frontend, sin ellas no podría haber sido posible la realización de la aplicación.

  Aquí podemos ver funciones como la de realizar la puja, revelar la puja, comprar ahora, finalizar la subasta, añadir un producto a la tienda, reembolsar el dinero al comprador, ver los detalles del producto, renderizar un producto a través de su id, guardar un producto en el Blockchain, guardar el hash correspondiente de IPFS, renderizar los productos, y alguna función más de menos importancia.

  Gracias al JavaScript y sus funciones también comprobamos el estado del producto para mostrar unos formularios u otros en cada pantalla de la aplicación.

Capítulo 9

Pruebas

En este capítulo se expondrán las pruebas realizadas sobre el resultado de la implementación del proyecto. En primer lugar se detallarán las pruebas de caja blanca, y para finalizar se analizarán las de caja negra.

9.1. Pruebas de caja blanca

El objetivo de estas pruebas es la comprobación a bajo nivel del correcto funcionamiento de las operaciones desarrolladas en cada una de las aplicaciones. A continuación se muestra un listado de las pruebas llevadas a cabo:

- **Control de acceso:** se comprueba que las funcionalidades sobre el Blockchain y la base de datos no funcionen para aquellos usuarios que no hayan iniciado sesión en MetaMask.
- **Validación de datos en el lado cliente:** se comprueba que tras introducir los datos en los formularios, se controlase su formato y se denegase el envío si no se adaptaban al solicitado.
- **Validación de datos en el lado servidor:** una vez recibidos los datos, se comprueba su formato por si se habían modificado.
- **Acceso a la base de datos:** se comprueba la correcta conexión a la base de datos de la aplicación web, en todas aquellas operaciones de consulta, inserción y actualización.
- **Funcionamiento de operaciones:** se comprueba el funcionamiento de cada una de las operaciones. Se realiza mediante la carga de datos de prueba para trabajar con ellos, y posteriormente con datos reales de la plataforma.
- **Datos JSON:** se comprueba que el envío y recepción de los datos a través de JSON por el método GET funciona correctamente.
- **Conexión con el Blockchain:** se comprueba que la aplicación tiene comunicación con Blockchain y viceversa.
- **Conexión con los Smart Contract:** se comprueba que la aplicación tiene comunicación con los Smart Contract y que esta escucha los eventos que le manda el Smart Contract.

9.2. Pruebas de caja negra

Estos test sirven para verificar la funcionalidad deseada de las aplicaciones desarrolladas, comprobando que se cumplen los requisitos modelados en la fase de análisis.

CP-01: Añadir producto	
Objetivo	Comprobar que se puede añadir un nuevo producto a la tienda
Preconsiciones	Haber iniciado sesión en MetaMask y tener algo de Ether.
Datos de entrada	Nombre: Producto de prueba Descripción: Descripción de prueba Foto: producto.jpeg Categoría: Coches Precio de salida: 12 ETH Precio de comprar ahora: 30 ETH Condición: Nuevo Hora de inicio: 06/07/2018 18:00 Días para ejecutar la subasta: 1
Acción esperada	Sale un mensaje de producto añadido correctamente. Se vuelve a la página del formulario con los datos vacíos.
Secuencia	1. Completar campo nombre 2. Completar campo descripción 3. Subir la imagen del producto 4. Elegir una categoría 5. Completar campo precio de salida 6. Completar campo precio de comprar ahora 7. Elegir la condición del producto 8. Elegir la hora de inicio de la subasta 9. Elegir cuanto quieres que dure la venta 10. Pulsar sobre Agregar producto a la tienda
Resultado	Correcto

Tabla 9.1: CP-01 - Añadir producto

CP-02: Visualizar productos	
Objetivo	Comprobar que el listado de los productos se carga bien
Preconsiciones	
Datos de entrada	
Acción esperada	Obtener un listado de todos los productos en venta
Secuencia	1. Ir a la página de inicio (index.html) 2. Mostrar los productos 2.1. En caso de no haber productos, mostrar un mensaje diciendo que no se encontraron productos
Resultado	Correcto

Tabla 9.2: CP-02 - Visualizar productos

CP-03: Visualizar un producto	
Objetivo	Comprobar que carga bien la información de un producto
Preconsiciones	
Datos de entrada	
Acción esperada	Visualizar un listado con toda la información del producto y las acciones que se pueden realizar
Secuencia	1. Seleccionar un producto 2. Mostrar los detalles del producto y las acciones que se pueden realizar
Resultado	Correcto

Tabla 9.3: CP-03 - Visualizar un producto

CP-04: Hacer una puja	
Objetivo	Comprobar que se realiza una puja por un producto de forma correcta
Preconsiciones	1. Se debe haber accedido a la página del producto 2. El producto debe estar en venta y en tiempo de subasta
Datos de entrada	1. Cantidad que puja: 10 ETH 2. Cantidad que envía: 10 ETH 3. Texto secreto: contraseña
Acción esperada	Realizar una puja por el producto deseado
Secuencia	1. Completar el campo, *Cantidad que puja* y que sea superior al precio de inicio e inferior al de precio de compra 2. Completar el campo, *Cantidad que envía* 3. Escribir un texto secreto para cifrar la puja 4. Pulsar sobre *Hacer puja*
Resultado	Correcto

Tabla 9.4: CP-04 - Hacer una puja

CP-05: Hacer una compra	
Objetivo	Comprobar que se realiza una compra por un producto de forma correcta
Preconsiciones	1. Se debe haber accedido a la página del producto 2. El producto debe estar en venta y en tiempo de subasta
Datos de entrada	
Acción esperada	Realizar una compra por el producto deseado
Secuencia	1. Pulsar sobre *Comprar ahora*
Resultado	Correcto

Tabla 9.5: CP-05 - Hacer una compra

CP-06: Revelar una puja	
Objetivo	Comprobar que se revelan las pujas de forma correcta
Preconsiciones	1. Se debe haber accedido a la página del producto 2. Se debe haber realizado una puja 3. Debe haber finalizado el tiempo de subasta
Datos de entrada	1. Cantidad de tu puja: 10 ETH 2. Texto secreto: contraseña
Acción esperada	Revelar una oferta realizada sobre un producto
Secuencia	1. Completar el campo, Cantidad de tu puja 2. Completar el campo, Texto secreto 3. Pulsar sobre *Revelar puja*
Resultado	Correcto

Tabla 9.6: CP-06 - Revelar una puja

CP-07: Finalizar una subasta	
Objetivo	Comprobar que se finaliza correctamente una subasta
Preconsiciones	1. Se debe haber accedido a la página del producto 2. Debe haber finalizado el tiempo de subasta 3. Debe haber finalizado el tiempo de revelar subastas
Datos de entrada	
Acción esperada	Finalizar la subasta de un producto
Secuencia	1. Pulsar sobre *Fin de la subasta*
Resultado	Correcto

Tabla 9.7: CP-07 - Finalizar subasta

CP-08: Enviar dinero al vendedor	
Objetivo	Comprobar que se envía el dinero al vendedor correctamente
Preconsiciones	1. Se debe haber accedido a la página del producto 2. Se debe haber ganado la subasta 2. Debe ser el árbitro, vendedor o comprador identificado
Datos de entrada	
Acción esperada	Votar por enviar el dinero al vendedor
Secuencia	1. Pulsar sobre Enviar dinero al vendedor 2. En caso de ser 2/3 votos, se liberan los fondos al vendedor y la venta se da por concluida
Resultado	Correcto

Tabla 9.8: CP-08 - Enviar dinero al vendedor

CP-09: Enviar dinero al comprador	
Objetivo	Comprobar que se envía el dinero al comprador correctamente
Preconsiciones	1. Se debe haber accedido a la página del producto 2. Se debe haber ganado la subasta 2. Debe ser el árbitro, vendedor o comprador identificado
Datos de entrada	
Acción esperada	Votar por enviar el dinero al comprador
Secuencia	1. Pulsar sobre Enviar dinero al comprador 2. En caso de ser 2/3 votos, se liberan los fondos al comprador y la venta se da por concluida
Resultado	Correcto

Tabla 9.9: CP-09 - Enviar dinero al comprador

.

Capítulo 10

Manuales

En este capítulo se redactan las guías para poder hacer uso del sistema desarrollado. En primer lugar, se muestra un manual para la instalación y configuración de la aplicación web y después un manual para los usuarios de la aplicación web.

10.1. Manual de instalación

Para facilitar la tarea de instalación de la plataforma sobre el sistema se proporciona este manual de usuario. Aquí se detalla como instalar las herramientas necesarias para que funcione el proyecto.

Lo primero que se debe hacer es copiar la carpeta del proyecto al destino del sistema donde se vaya a ejecutar.

A continuación tenemos que instalar npm con el siguiente comando:

npm install

Después debemos instalar ganache-cli con el siguiente comando:

npm install -g ganache-cli

Y debemos de asegurarnos de que ganache-cli está ejecutándose, con el comando:

ganache-cli

Debemos configurar el archivo *truffle.js* para que apunte a esa red. Por ejemplo: *localhost:8545*.

Una vez tenemos ganache-cli ejecutandose y truffle.js apuntando a esa red, debemos crear los artefactos contractuales json, ejecutando el comando:

truffle compile

Cuando termine de crearse los artefactos, debemos implementar los contratos en la red con el comando:

truffle migrate

Si es preciso, se debe cambiar de proveedor web3 en el fichero *app.js* para que apunte al servidor de ganache-cli.

Debemos instalar IPFS en nuestro sistema, para ello hay que dirigirse a la web https://dist.ipfs.io/#go-ipfs, una vez lo descargues debes extraer el contenido del paquete y acceder a el desde comandos:

cd go–ipfs

ipfs init

ipfs config – –json API.HTTPHeaders.Access–Control–Allow–Origin '["*"]'

Para lanzar IPFS, en una ventana a parte ejecutamos:

ipfs daemon

Después debemos instalar MongoDB en nuestro sistema, en MAC se instalaría con el comando:

brew install mongodb

Una vez instalado MongoDB y le iniciamos con:

mongod y luego con **mongo**

Una vez encendido, lanzamos el servidor de escucha con el comando:

node_modules/.bin/nodemon server.js

A continuación servimos los archivos que están en la carpeta app a través del comando:

npm run dev

Y ya estaría todo listo para poder acceder a nuestra aplicación desde el navegador indicando el host y el puerto en que truffle lo está sirviendo, predeterminadamente es: http://localhost:8080.

Llegados a este punto, tendríamos instalado y en funcionamiento nuestra aplicación web.

10.2. Manual de usuario

En esté apartado se describe el mapa de navegación de la aplicación web y el uso de la aplicación pasando por todas sus pantallas con los distintos diseños dependiendo del dispositivo.

10.2.1. Mapa de navegación

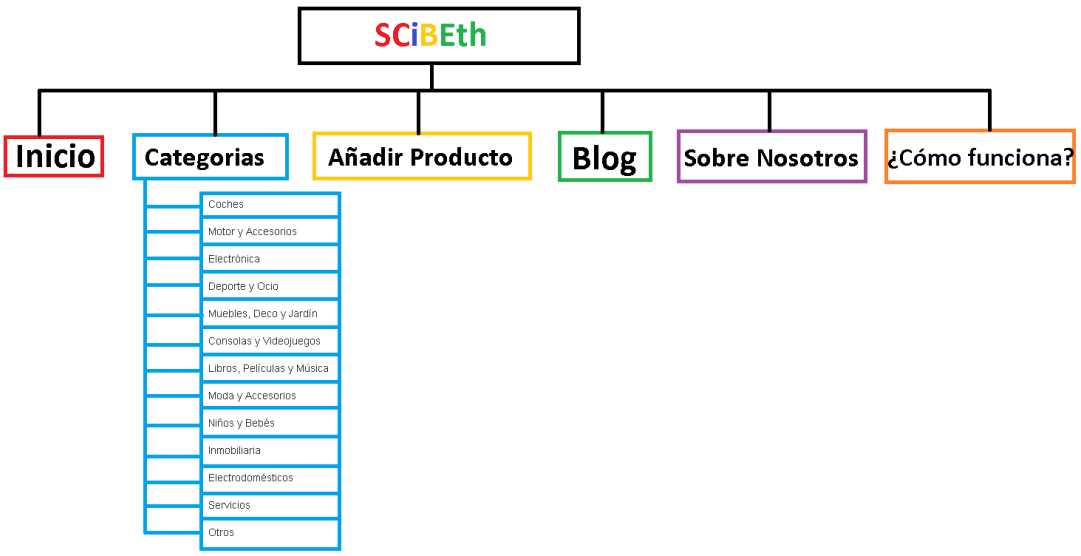


Figura 10.1: Mapa de navegación

En está figura, podemos observar el mapa de navegación de la aplicación web, tenemos las opciones de ir a la página de *Inicio* en la cual se encuentran todos los productos clasificados por el estado de la subasta en el que se encuentren (realizar puja, revelar puja, finalizar puja). La segunda opción solo se encuentra disponible en la pantalla de inicio y es la opción de *Categorías*, está opción te permite ver solo los productos de una categoría que están en estado de realizar puja. La tercera opción es la de *Añadir producto*, está opción nos lleva a un formulario que hay que rellenar con la información del producto que desea añadir a la tienda. La cuarta opción es *Blog*, está opción te lleva a mi Blog en el cual podrá encontrar información interesante sobre mis proyectos. La quinta opción es *Sobre Nosotros* en la cual podrá encontrar información sobre quienes somos. La sexta opción es *¿Cómo funciona?*, en esta página se encuentra información sobre como funciona la aplicación.

10.2.2. Uso de la aplicación

El desarrollo de la aplicación web se ha realizado tratando de crear una navegación sencilla con una interfaz limpia e intuitiva. Por está razón cualquier usuario con un conocimiento mínimo para manejar el ordenador, podrá utilizar nuestra Dapp sin problemas.

En primer lugar se debe acceder a la url de la aplicación web desde un navegador en la dirección `http://localhost:8081` actualmente. Para ello debe estar corriendo el proyecto en su PC. Se debe tener instalado MetaMask.

En cuanto a la navegación dentro de la web, se navega desde el menú principal a las diferentes pantallas. A continuación se muestran las opciones de la navegación junto con las opciones que se pueden realizar:

- **Index:** Se muestran los productos dependiendo de la clasificación (En venta, en etapa de revelar y en etapa de finalizar). En está página también podemos mostrar solo los productos que están en venta de la categoría que seleccionemos.

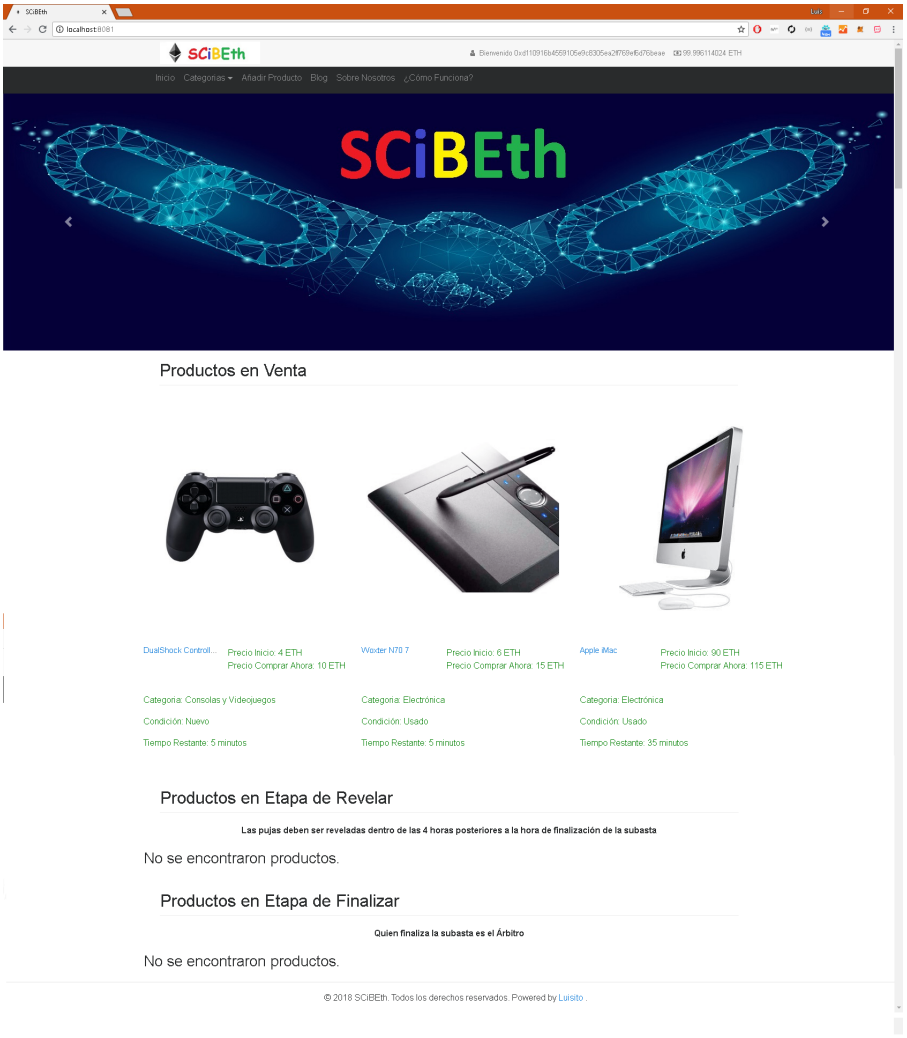


Figura 10.2: Captura de index PC

Los productos en etapa de venta, son por los cuales se puede pujar o comprar ahora. Los productos que ya están en etapa de relevar, son aquellos cuya puja a terminado, bien porque ha terminado el tiempo de la subasta, o porque un usuario le ha comprado ahora, en está etapa hay un tiempo para revelar la puja que hiciste y declarar un ganador. La etapa de finalizar es la etapa en la que entra un producto cuando ya ha sido revelado el ganador y es cuando se pasa la etapa de enviar el dinero, bien al vendedor porque todo a ido correctamente o se le hace un reembolso al comprador, porque algo ha salido mal en la venta.

- **Añadir producto:** Se muestra un formulario con la información que debes introducir para añadirlo a la tienda, si alguna parte del formulario está sin rellenar o mal rellenada como la fecha de inicio que quieres que empiece la puja de tu producto, no se añadirá a la tienda. Cuando el formulario sea enviado saldrán mensajes diciendo que el producto a sido enviado a la cadena de bloques y una vez confirmes la transacción saldrá un mensaje diciendo si la transacción a sido un éxito y se a agregado a la tienda o si por el contrario algo a salido mal.

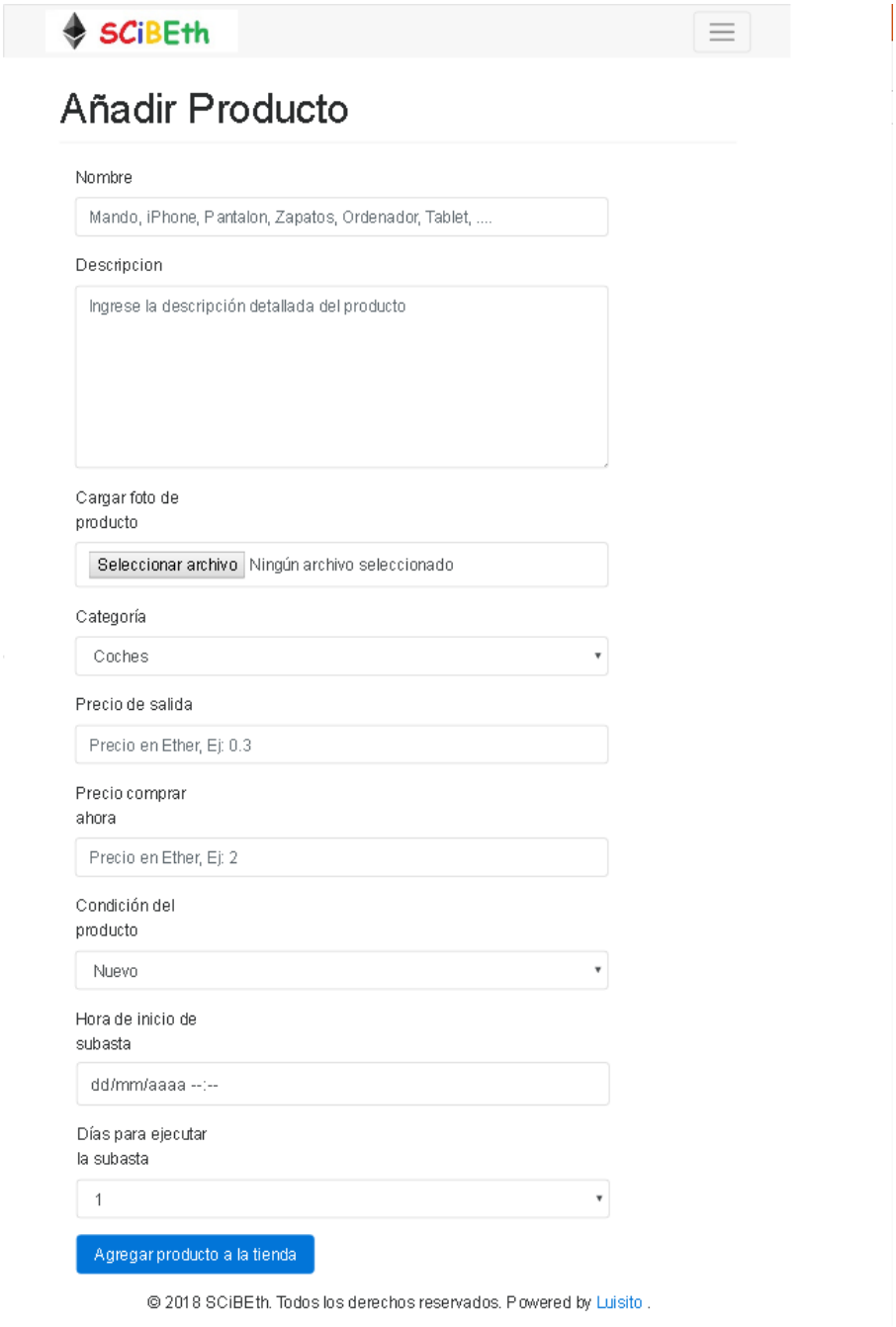


Figura 10.3: Captura de añadir producto desde iPad

En la siguiente imagen podemos ver un formulario completo enviado y el mensaje que sale cuando se envía a la cadena de bloques antes de confirmar nuestra transferencia.

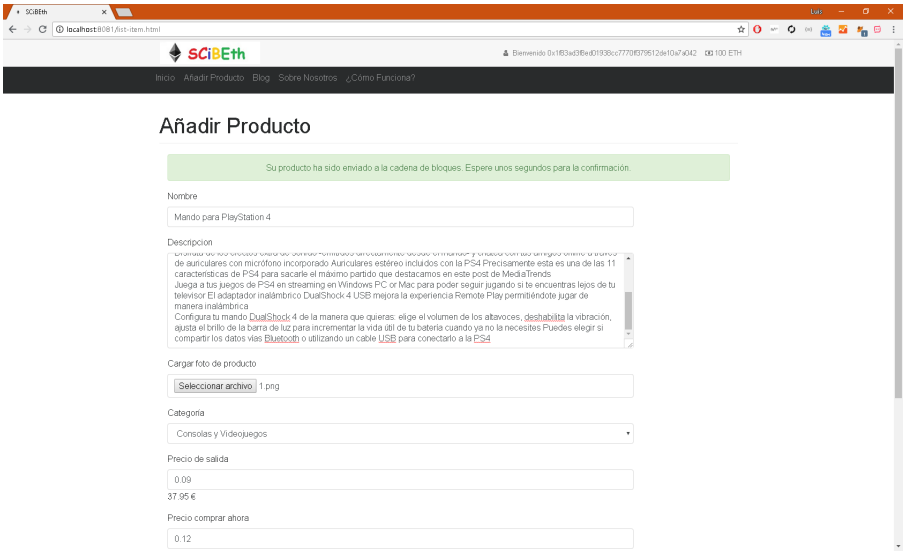


Figura 10.4: Captura de añadir producto PC enviado formulario

En la siguiente imagen se ve la transferencia que queremos realizar para añadir el producto a la cadena de bloques y a nuestra tienda.

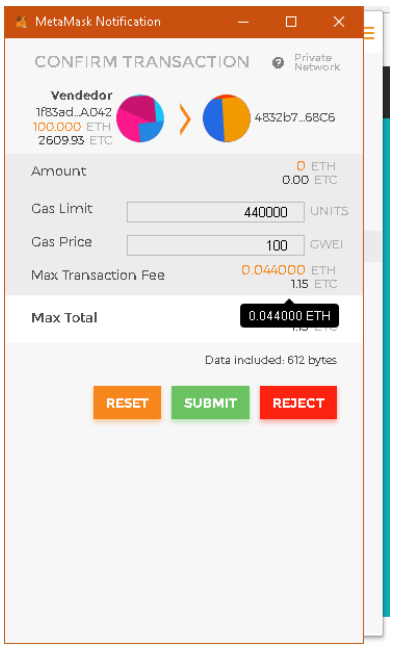


Figura 10.5: Captura de añadir producto PC - MetaMask

En la última imagen se muestra el mensaje de éxito al añadir nuestro producto a la tienda, a partir de ahora le podremos ver en la página de inicio y podremos acceder a él pinchando sobre él.

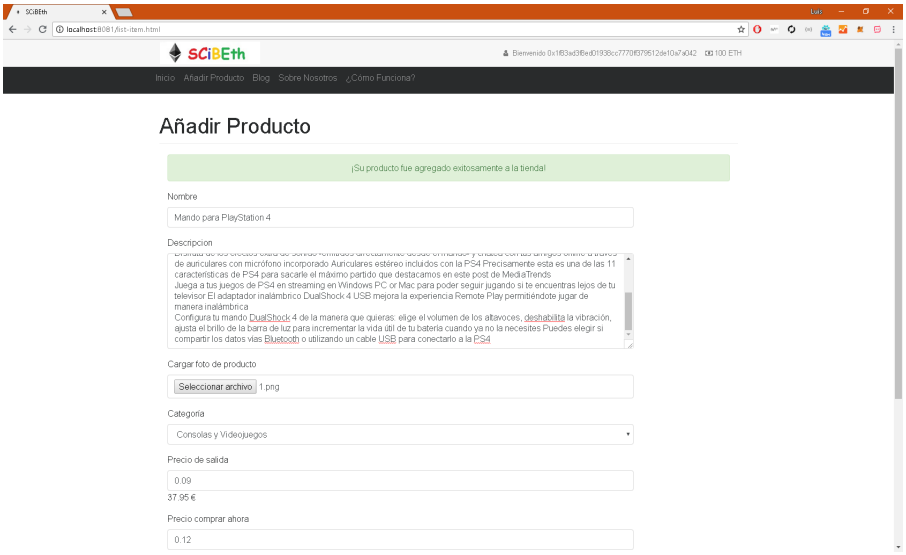


Figura 10.6: Captura de añadir producto desde PC éxito

- **Ver un producto:** Para acceder a la información de un producto, hay que pinchar sobre él en la pantalla de Inicio. Así se mostrará la información de ese producto y sus funcionalidades (formularios), dependiendo de la etapa en la que esté:

  - **En venta:** En la etapa de venta, veremos información relevante sobre el producto, como el precio de salida, el estado en el que está (nuevo o usado), el tiempo que le queda para finalizar la puja, y dos formularios. El primer formulario es para realizar una puja sobre el producto siempre que sea por encima del precio de inicio y por debajo del precio de venta. El segundo formulario es el de comprar ahora, que simplemente será un botón, con el finalizará la subasta y tu serás el ganador del producto.

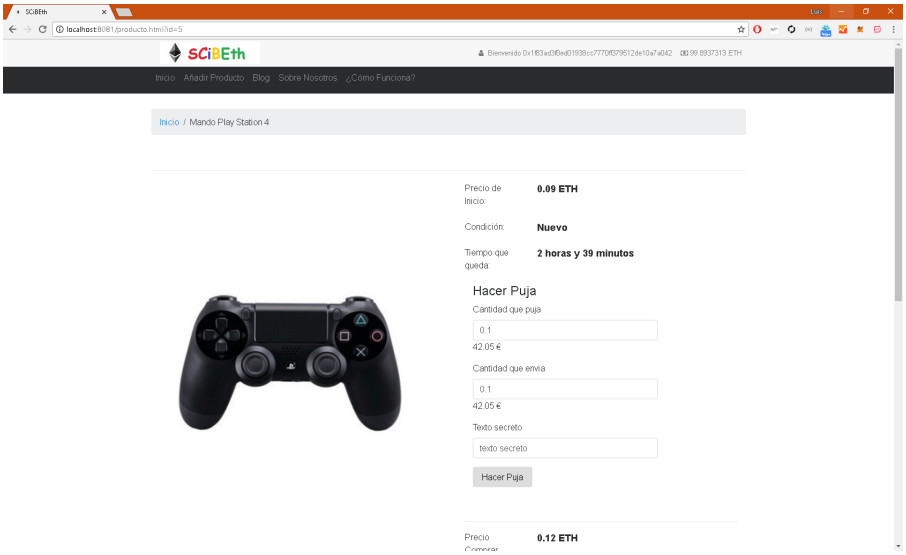


Figura 10.7: Captura de detalles de un producto en etapa de venta

- **En etapa de revelar:** En la etapa de revelar se mostrará información sobre el producto, como el precio de inicio, el estado (nuevo o usado), etc. Se mostrará un formulario para revelar la puja, para revelar quien a sido el ganador de la subasta, si no se revela la puja no puede ser el ganador del producto. Para revelar la puja deberá introducir la cantidad de ETH que a pujado al igual que la palabra secreta que a elegido. Una vez reveladas las pujas y cuando acabe el tiempo de revelar, tendremos un ganador de la subasta y pasaremos a la siguiente etapa.

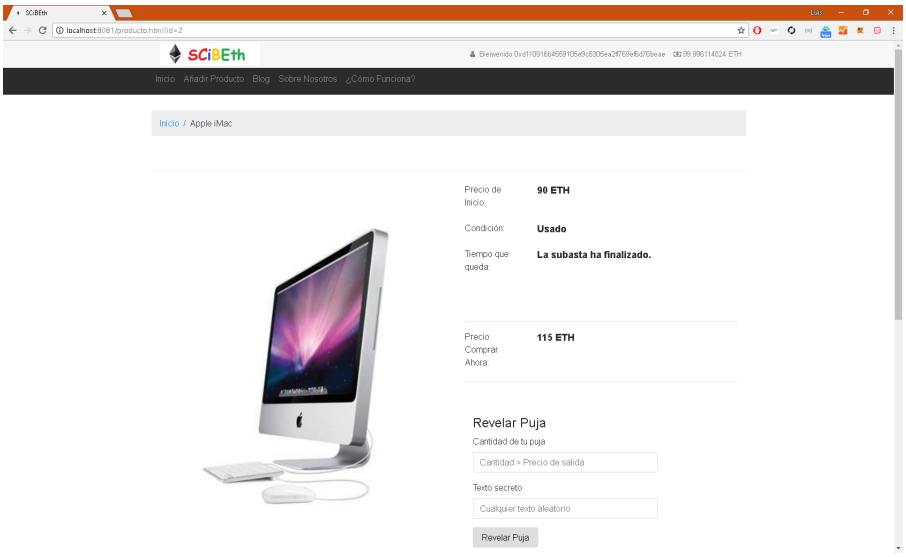


Figura 10.8: Captura de detalles de un producto en etapa de revelar

- **En etapa de finalizar:** Es la penúltima etapa por la que pasa un producto, en esta tiene que haber una tercera persona que actuará a modo de árbitro entre el comprador y el vendedor y se llevará el 1 % de la transacción, esté árbitro sirve para que en caso de disputa entre el vendedor y el comprador decida si se reembolsa el dinero al comprador o se le envía al vendedor en la última etapa de enviar fondos.

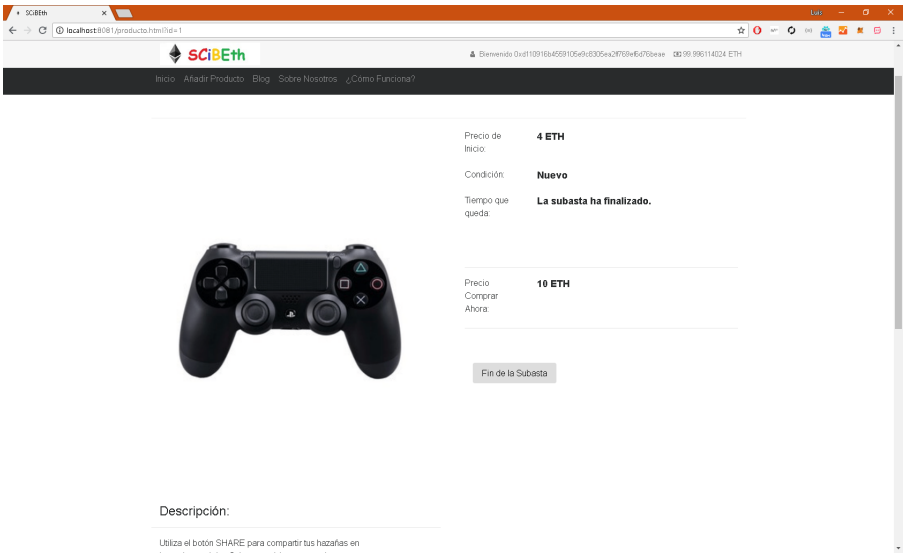


Figura 10.9: Captura de detalles de un producto en etapa de finalizar

- **En etapa de enviar fondos:** En esta etapa 2 de las 3 personas tienen que votar si enviar el dinero al vendedor o al comprador, en caso de disputa intervendrá el árbitro de la puja. Una vez 2 de los 3 hayan votado una misma opción se liberarán los fondos y se dará por finalizada la subasta o venta del producto, dejando de aparecer en la página de inicio.

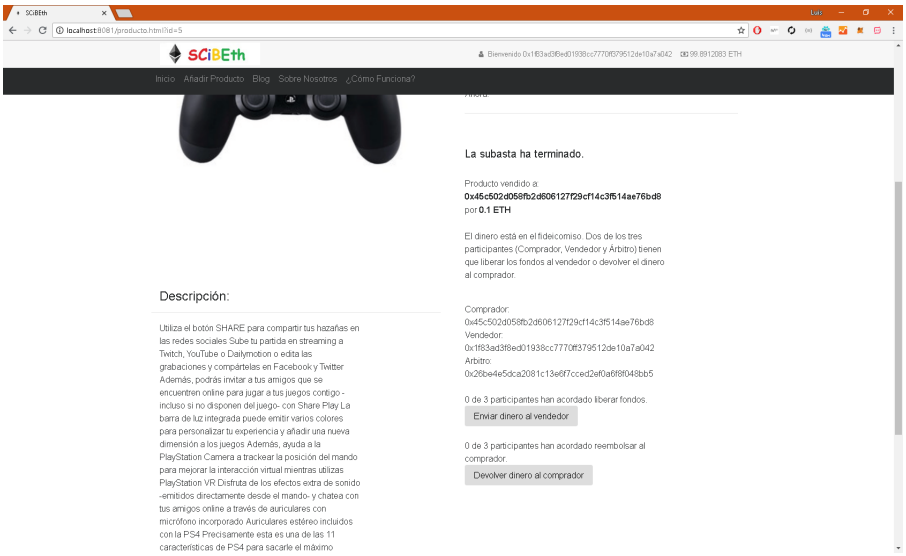


Figura 10.10: Captura de detalles de un producto en etapa de enviar dinero

- **Producto vendido:** Una vez se han enviado los fondos, saldrá una pantalla con la información de la venta del producto y los usuarios que han intervenido en ella, para acceder a ella habrá que acceder poniendo el id del producto en la URL. (Más adelante habrá una opción para mostrarlo en tu cuenta, donde saldrán las ventas y las compras realizadas, pero que no se han incluido en está primera versión).

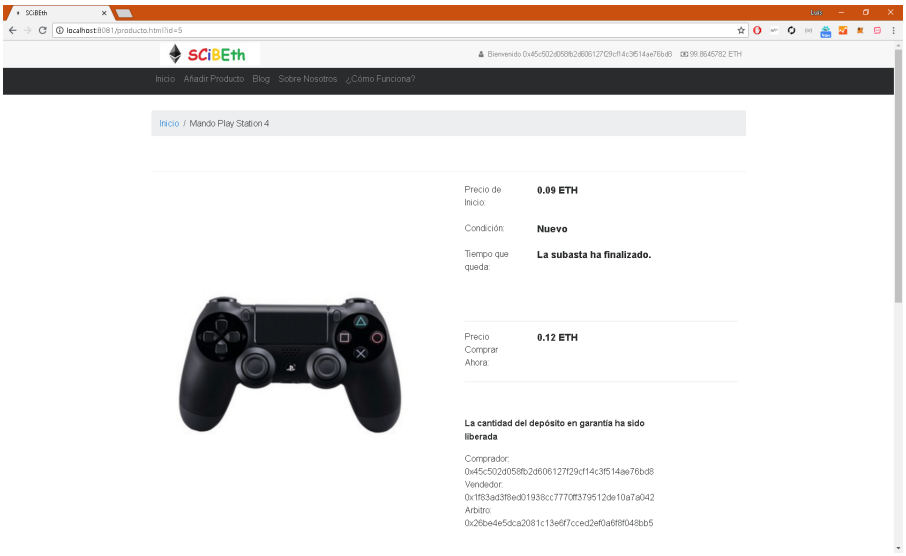


Figura 10.11: Captura de detalles de un producto en etapa de finalizar

- **Iniciar sesión en MetaMask:** Se muestra como se debe iniciar sesión en la billetera que une el navegador con el Blockchain, para poder ser un usuario de la aplicación web, ya que si no estás registrado en el Blockchain no podrás hacer uso de la aplicación.

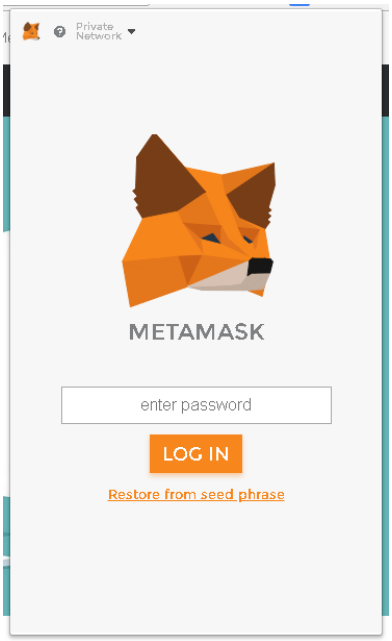


Figura 10.12: Captura de inicio de sesión con MetaMask

- **Importar cuenta en MetaMask:** Se muestra como se debe importar una cuenta del Blockchain a la billetera de MetaMask. De esta forma podrás tener conectada tu billetera con el navegador web, para poder usar la aplicación web.

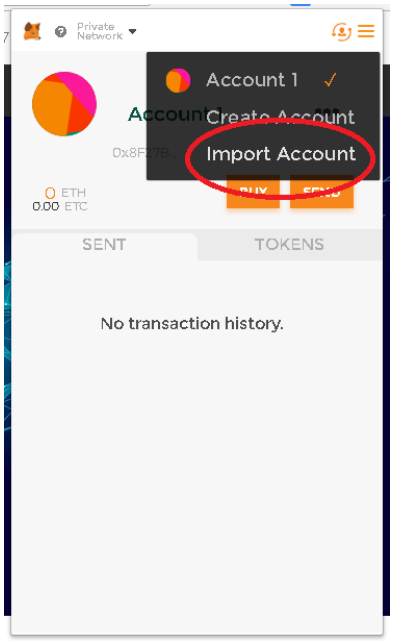


Figura 10.13: Captura de importar cuenta en MetaMask

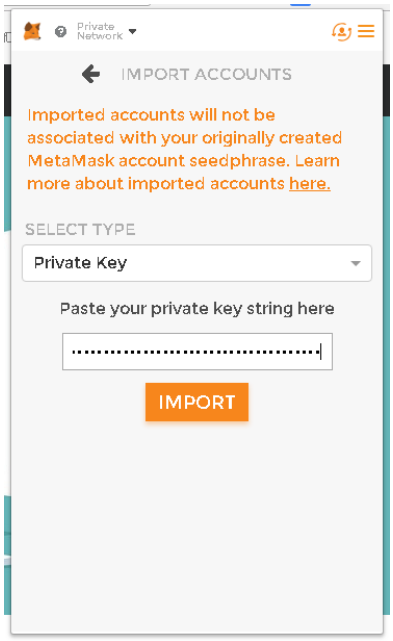


Figura 10.14: Captura de importar cuenta en MetaMask

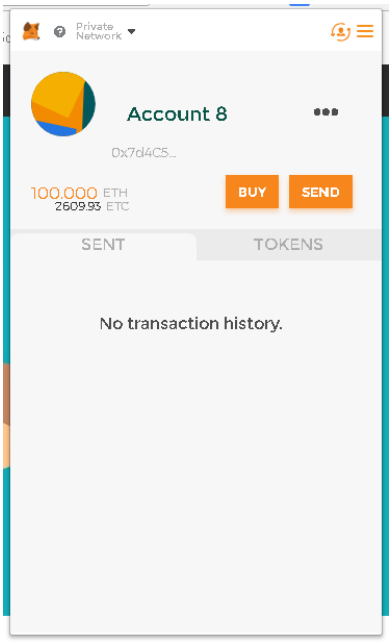


Figura 10.15: Captura de importar cuenta en MetaMask

- **Pujar por un producto:** Se sigue la siguiente secuencia, una vez que estamos dentro del producto, como hemos detallado anteriormente. Lo primero es rellenar el formulario con la cantidad que queremos enviar y la palabra secreta que queramos usar.

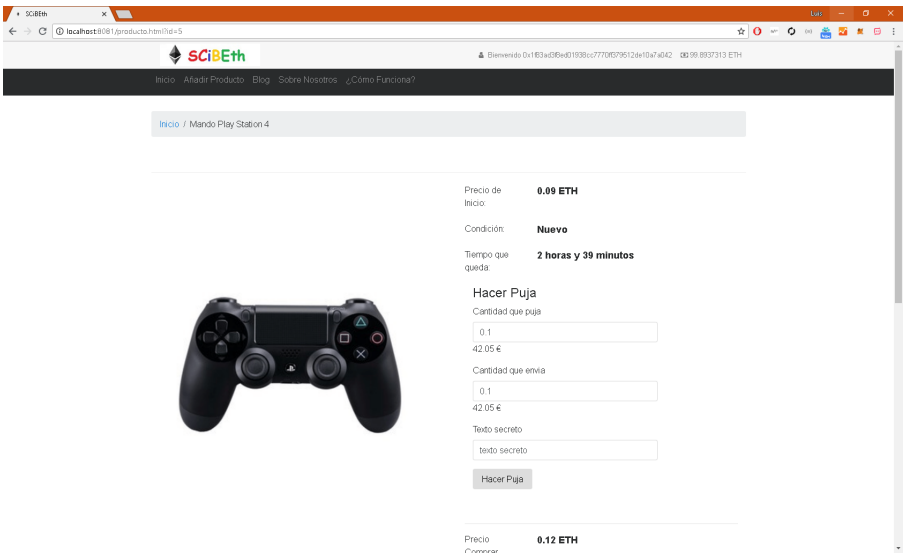


Figura 10.16: Captura de importar pujar por un producto

Lo siguiente es aceptar la transferencia de la cadena de bloques para registrar la puja/oferta en la cadena de bloques y que quede registrada.

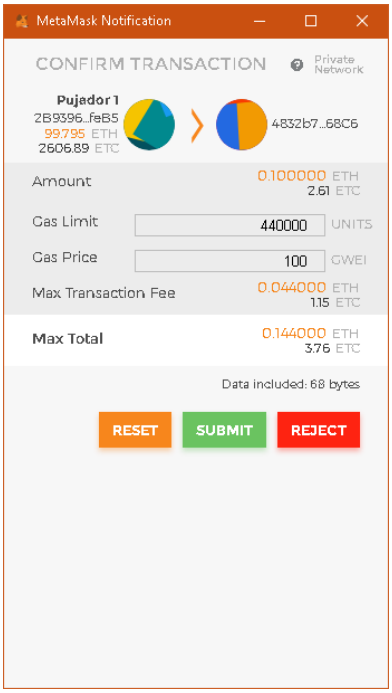


Figura 10.17: Captura de pujar por producto - MetaMask

Si todo a ido correctamente se mostrará un mensaje de éxito y se registrará la puja en la cadena de bloques, para una vez finalizada la subasta se revelen las pujas y saber si tu eres el ganador o no.

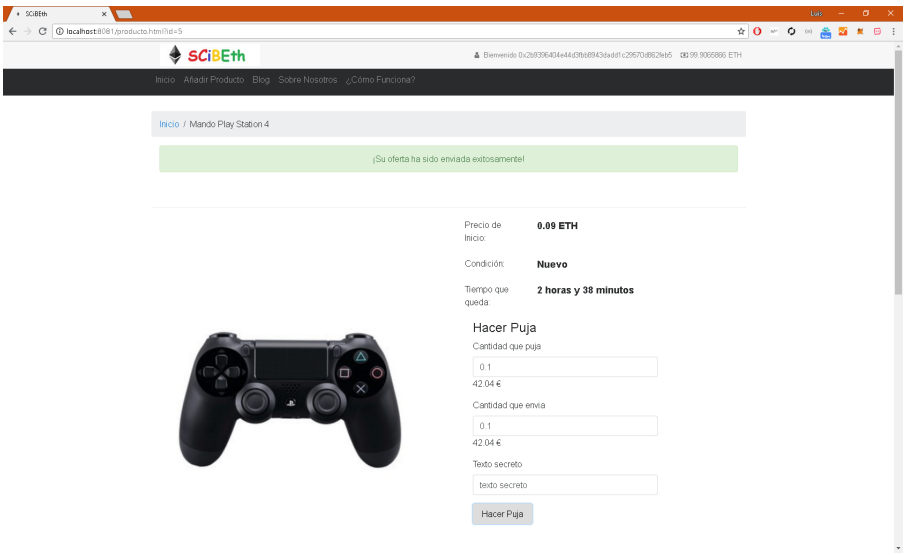


Figura 10.18: Captura de pujar por producto éxito

- **Revelar una puja:** Se sigue la siguiente secuencia, una vez que estamos dentro del producto, como hemos detallado anteriormente. Lo primero es rellenar el formulario con la cantidad que enviamos para la puja y la palabra secreta que utilizamos para realizar la puja.

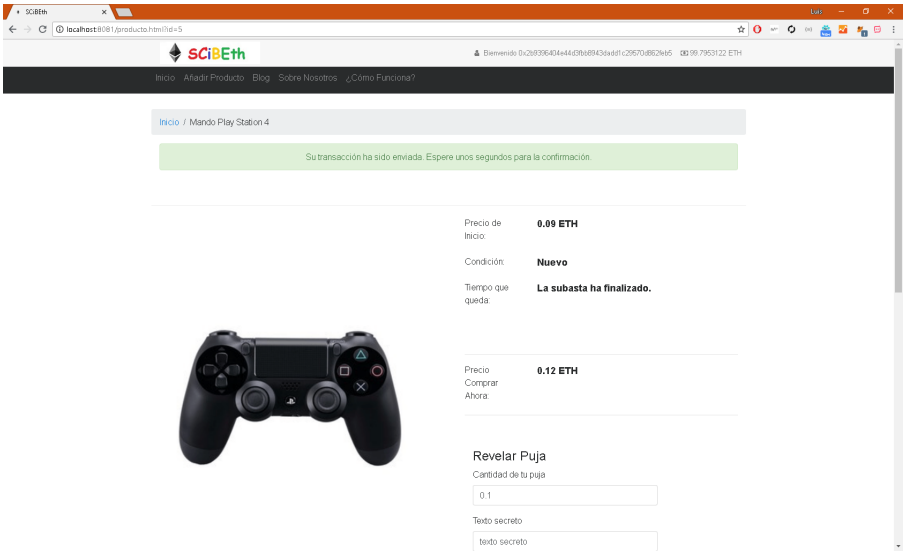


Figura 10.19: Captura de revelar puja por producto

Una vez se envié el formulario a la cadena de bloques, nos saltará la transferencia que vamos a realizar para que la aceptemos.

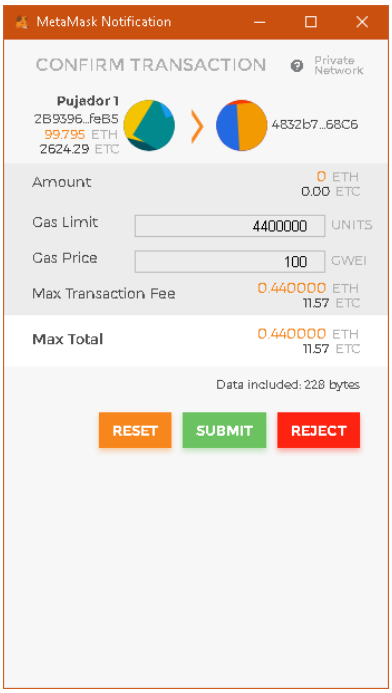


Figura 10.20: Captura de revelar puja por producto MetaMask

Si todos los datos son correctos y la transferencia también nos saldrá un mensaje de

éxito y una vez acabe el tiempo de revelar la subasta sabremos quien es el ganador de la subasta.

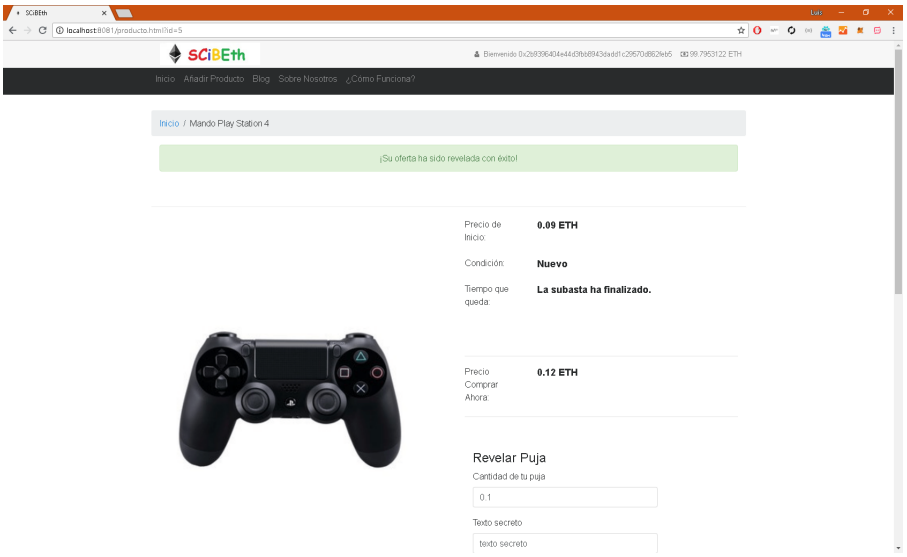


Figura 10.21: Captura de revelar puja por producto éxito

- **Finalizar una subasta:** Se sigue la siguiente secuencia, una vez que estamos dentro del producto, como hemos detallado anteriormente. Saldrá un botón para finalizar la subasta, está no la puede finalizar ni el ganador, ni el vendedor del producto, sino una tercera persona, la cual tendrá que pagar 5 ETH que una vez finalizada la venta se le devolverán más el 1 % de la venta como premio. Una vez la tercera persona pulse el botón le saltará el pago y la transferencia en MetaMask.

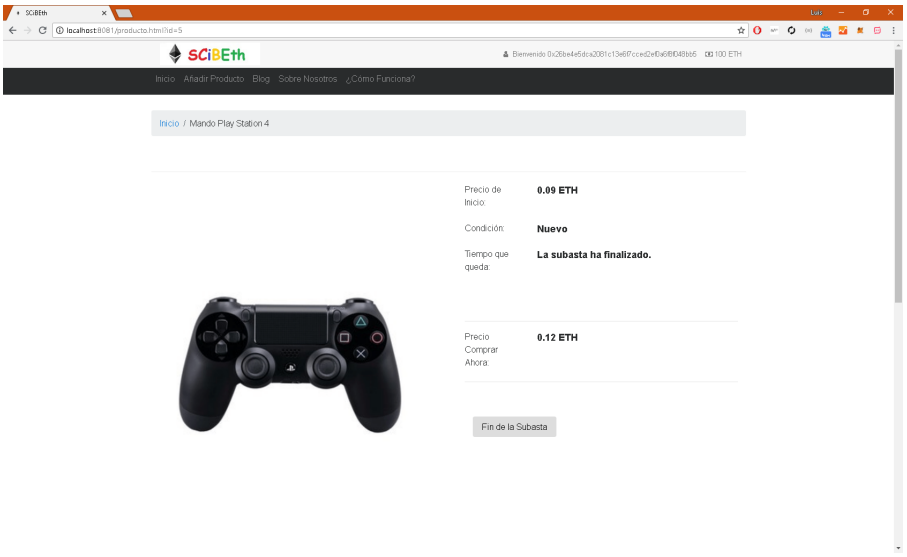


Figura 10.22: Captura de finalizar subasta por producto

En la siguiente imagen podemos ver la transferencia que se va a realizar, y podemos ver los 5 ETH que vamos a enviar a modo de precaución, para saber que ese árbitro va a

actuar de un modo legal y sin hacer trampas para que le sean devueltos junto el 1 % de la venta.

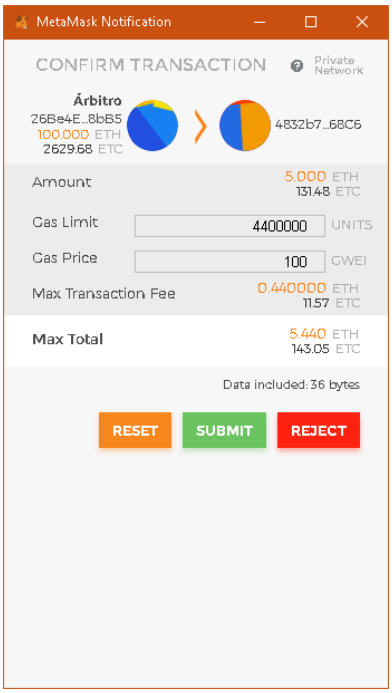


Figura 10.23: Captura de finalizar subasta por producto MetaMask

Hay dos opciones posibles cuando se finaliza una subasta, que haya un pujador/comprador o que nadie haya pujado. En el primero caso, veremos quien a sido el ganador de la subasta y pasaremos a la etapa de enviar los fondos, la cual funciona por medio de votación como veremos a continuación.

1. - Fin de la subasta con algún pujador/comprador:

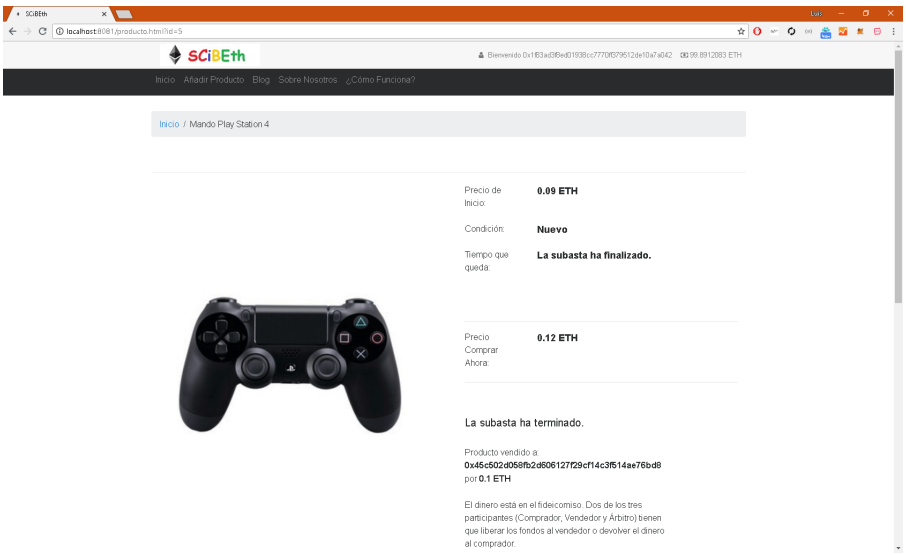


Figura 10.24: Captura de finalizar subasta por producto con pujador

En el caso de que nadie haya hecho ofertas sobre ese producto o no se hayan revelado ofertas, se mostrará la siguiente pantalla y se dará por finalizada la venta del producto.

2. - Fin de la subasta sin ningún pujador/comprador:

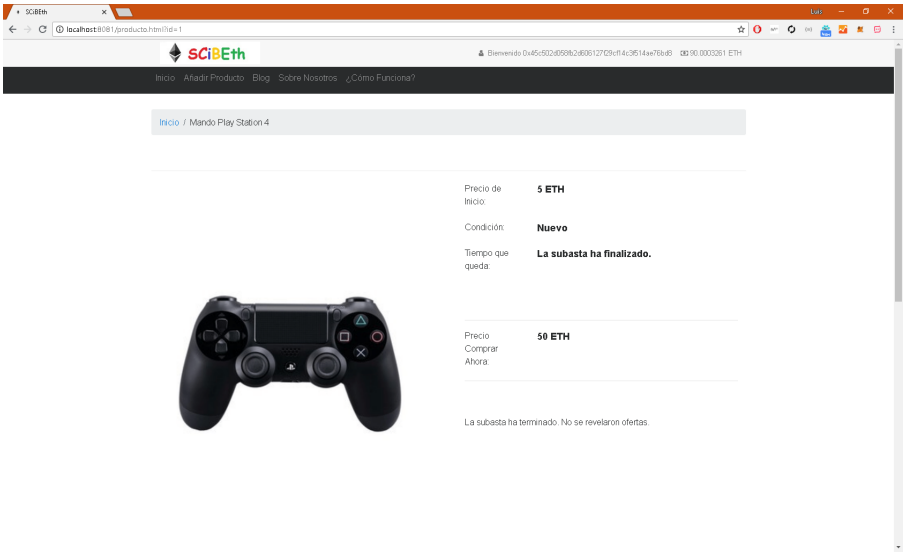


Figura 10.25: Captura de finalizar subasta por producto sin pujador

- **Enviar dinero:** Se sigue la siguiente secuencia, una vez que estamos dentro del producto, como hemos detallado anteriormente. En la siguiente imagen veremos una especie de formulario que funciona por votación, ya que 2 de los 3 participantes en la subasta se tienen que poner de acuerdo para enviar los fondos al vendedor o reembolsar el dinero al comprador. El árbitro solo debe actuar en está parte en caso de disputa o de que uno de los otros dos usuarios no rellenen la votación en varios días.

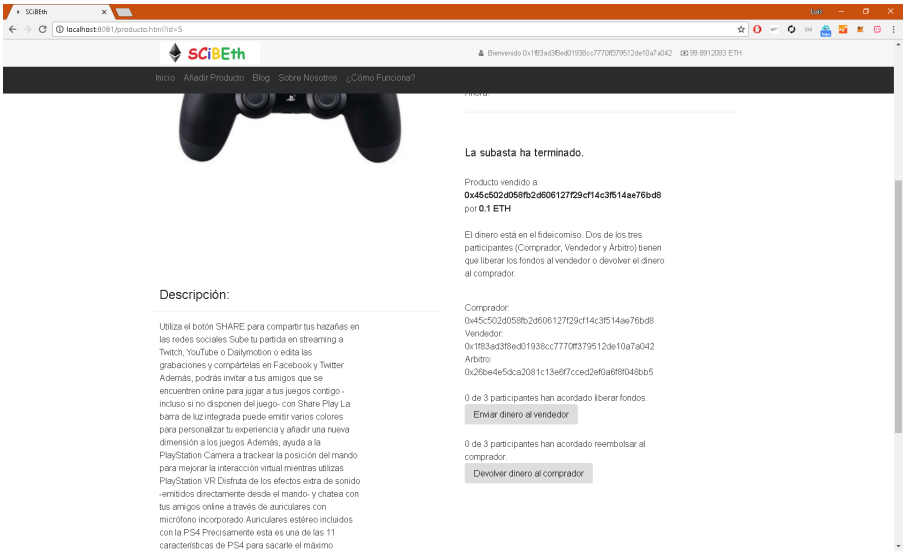


Figura 10.26: Captura de enviar dinero de un producto

Una vez envías tu votación, la cadena de bloques registra la transferencia para guardarla, saltando la siguiente pantalla de MetaMask.

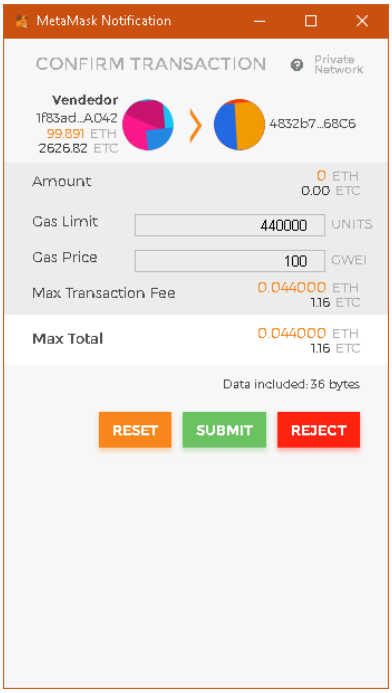


Figura 10.27: Captura de enviar dinero de un producto MetaMask

Si la transacción a ido con éxito, se registrará tu voto y se actualizarán los datos como se muestra en la siguiente imagen.

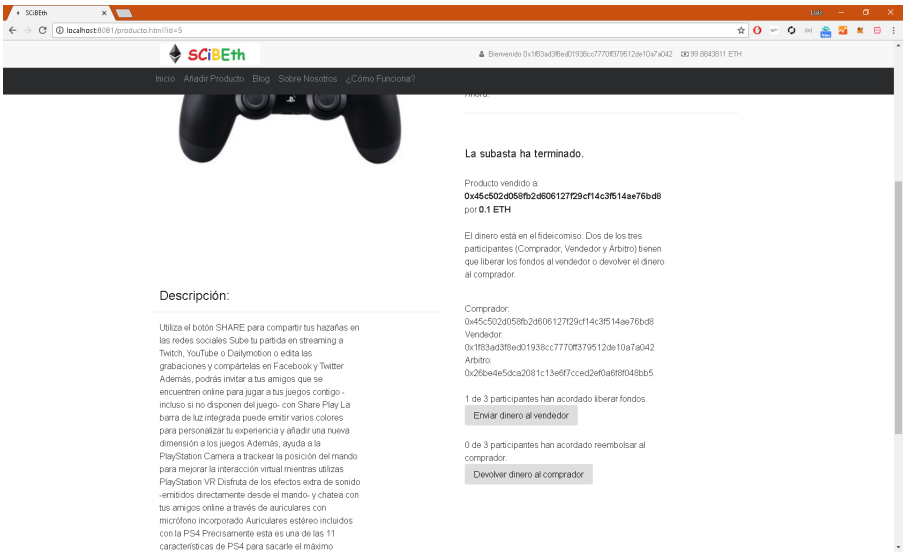


Figura 10.28: Captura de enviar dinero de un producto votación primera

Una vez 2 de los 3 hayan votado lo mismo se liberarán los fondos retenidos en el contrato de Fideicomiso y saldrá la información sobre lo sucedido en la subasta y los usuarios que han intervenido en ella. Con ello se da por concluida la subasta/venta.

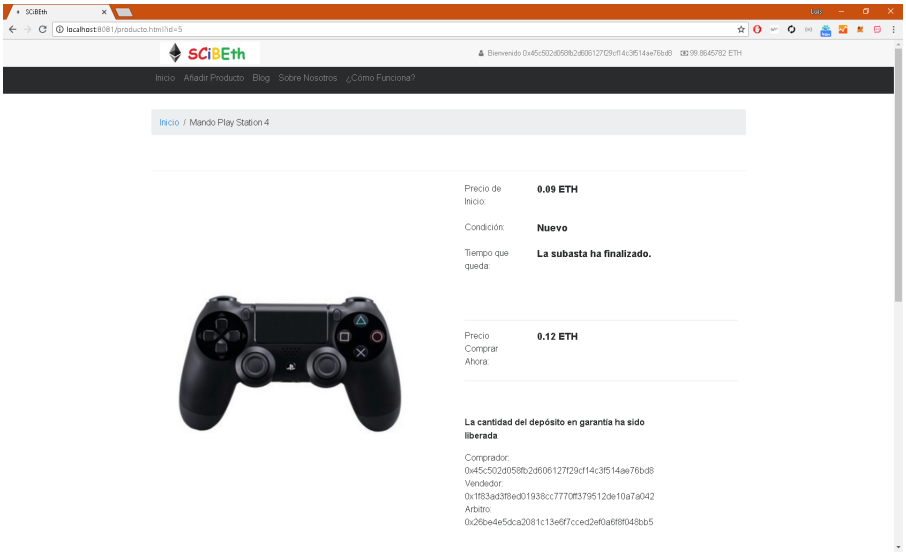


Figura 10.29: Captura de enviar dinero de un producto finalizada

Capítulo 11

Problemas, conclusiones y trabajos futuros

En este capítulo se realizá una revisión del trabajo realizado, analizando los problemas, los resultados y el aprendizaje llevado acabo y finalmente se explicarán que trabajos futuros se pueden realizar para mejorar la aplicación web.

11.1. Problemas surgidos

La primera y mayor dificultad encontrada fue empezar con la programación en Solidity tras la fase de investigación y análisis. Debido al desconocimiento de este ámbito, se tuvo que iniciar por un nivel muy bajo, empezando con vídeos de Youtube de Nicolas Palacio [39].

De este modo fui yendo de unos contratos más simples a otros más complejos utilizando el software que nos proporciona la página web https://ethereum.org/ llamado Ethereum Wallet, esto funciono durante un par de semanas, luego empezó a fallar la billetera sin poder utilizarla para probar he interactuar con los contratos de prueba que iba realizando y me toco buscarme la vida para solucionarlo.

Investigando por Internet, me uní a un grupo privado de desarrollo de Dapps en Facebook, y ahí, empece a ver contratos más complejos, dudas que ponía la gente y a mi me servían y desarrollos de frontend y backend con contratos inteligentes.

Luego investigando más acerca de las Dapps encontré un Meetup de Sevilla [26] donde hablaban todas las semanas sobre desarrollo de Dapps en Ethereum y al que me uní para ver las Meetup de manera online y poder consultar mis dudas a través de emails.

Y por último, gracias al grupo de Facebook conocí una página web que te enseña a programar Dapps en Ethereum construyendo tu propio juego, Cryptozombies [20] y gracias a ello termine de adquirir los conocimientos suficientes para poder escribir mis contratos.

La siguiente dificultad que encontré fue como unir mis contratos inteligentes con la parte frontend de mi aplicación web, investigando por Internet encontré Truffle [55]. Hice un tutorial que tiene que te guía en el proceso de construcción de su primera Dapp, un sistema de seguimiento para una tienda de mascotas.

Una vez que terminé con el, ya estaba listo para adentrarme en mi proyecto de SCiBEth, una tienda descentralizada de usuarios con dos opciones de compras, subastas y compras directas. Aquí me encontré con el problema de programar en JavaScript, ya que, tenia un conocimiento bastante básico, pero con esfuerzo e investigando llegué a adquirir los conocimientos necesarios para sacar la plataforma adelante.

Otro problema que me iba encontrando, es que iban sacando nuevas versiones del lenguaje Solidity, por lo que mis contratos cada vez que los compilaba para comprobar que las funciones que iba realizando funcionaban, y no empezaban a dar problemas porque utilizaban funciones o parámetros que se iban despreciando de unas versiones a otras, por lo que había que comprobar que todo funcionaba antes de adelantar trabajo cada día.

La idea principal era haber realizado el frontend sobre algún framework, en un principio se pensó en AngularJS o VUE.js pero debido al tiempo llevado a cabo en aprender Solidity y JavaScript se decidió realizar la interfaz de usuario simplemente con HTML, CSS, JavaScript y Bootstrap para recuperar el tiempo perdido y que diera tiempo a sacar el proyecto adelante.

11.2. Conclusiones

El principal objetivo de este proyecto era investigar, experimentar y aprender sobre los smart contract y la tecnología de la cadena de bloques. Y este objetivo lo he conseguido mediante la implementación de una aplicación funcional como es el proyecto SCiBEth.

Gracias a la realización del proyecto y el estudió llevado a cabo sobre la cadena de bloques ha resultado fructífera permitiendo ver que es una tecnología con un gran potencial disruptivo enorme que seguro que dará que hablar a lo largo de los próximos años.

Gracias a utilizar una de las redes de Blockchain más conocidas como es Ethereum, ha servido para comprender el principal lenguaje de programación de los Smart Contract (Solidity), siendo capaz de desarrollar un contrato desde cero.

Se han logrado resolver todos los problemas que nos han ido surgiendo durante el desarrollo del proyecto, como los comentados en la sección anterior.

Al final se ha conseguido una aplicación web funcional que cumple los objetivos planteados y que puede servir de base para futuros proyectos orientados al uso por un público de verdad.

11.3. Trabajos futuros

Al haber implementado la aplicación en local, tenemos que estar borrando la base de datos MongoDB cada vez que apagamos nuestra Blockchain (ganache-cli), lo ideal sería alojar la aplicación en una Blockchain aunque fuera una de Test, para que todo el mundo pueda acceder a ella y así no tener que estar borrando la base de datos para que coincida con los datos del Blockchain.

Otra opción a mejorar, es la que concierne al árbitro, ya que, actualmente, cualquiera puede finalizar la subasta y convertirse en árbitro si tiene 5 ETH para pagar. Se podría crear una estructura de árbitros, donde haya que depositar los 5 ETH que se piden para ser árbitro y así ganar el 1 % de la venta y donde puedan dejar de ser árbitros cuando quisieran recuperando los 5 ETH depositados y dejando de ser árbitro de la aplicación.

De esta mejora anterior, podemos sacar otra. Donde el depósito de 5 ETH del árbitro se quema si se descubre que el árbitro es un mal actor del sistema, por ejemplo, en caso de haber pactado con el comprador o vendedor.

Otra mejora sería añadir un chat para que en caso de disputas entre compradores y vendedores, el árbitro pueda actuar de una mejor manera e integrar un servicio externo de seguimiento de envió, para saber si llega o no el producto al destino.

Se podría hacer también un sistema de críticas u opiniones. En las que el comprador ponga una opinión sobre el vendedor y el producto, es decir, el servicio. Y que el vendedor escriba otra sobre el comprador.

Se podría agregar la funcionalidad de que un usuario, al acceder a su perfil, pudiera ver las ventas de los productos subidos a la web y modificar los datos o borrar el anuncio, y donde también, pudiera ver los productos de las compras y subastas que ha ganado.

Y por último, nuestra aplicación solo utiliza Ether como moneda, pero se podría hacer uso de un exachange descentralizado para que cada usuario use la moneda de su preferencia.

Webgrafía

[1] *ĐAPPS, ¿Qué son y para qué sriven las Aplicaciones Decentralizadas o Đapps?* 27-02-2018. miethereum. URL: https://miethereum.com/smart-contracts/dapps/.

[2] *An introduction to ecash.* 27-02-2018. The Wayback Machine. Noviembre 1996. URL: https://web.archive.org/web/19961102121426/http://digicash.com:80/publish/ecash_intro/ecash_intro.html.

[3] *[ANNOUNCE] Hash cash postage implementation.* 01-03-2018. A. Back. Marzo 1997. URL: http://www.hashcash.org/papers/announce.txt.

[4] *¿Qué son las altcoins y qué ofrecen al ecosistema Bitcoin?* 01-03-2018. bit2me. URL: https://blog.bit2me.com/es/que-son-las-altcoins/.

[5] *B-money.* 01-03-2018. Wei Dai. Julio 2016. URL: http://www.weidai.com/bmoney.txt.

[6] *Bit gold.* 05-03-2018. Unenumerated. Diciembre 2008. URL: http://unenumerated.blogspot.com/2005/12/bit-gold.html.

[7] *Bitcoin.* 05-03-2018. Wikipedia. URL: https://en.wikipedia.org/wiki/Bitcoin#Mining.

[8] *Bitcoin, ethereum o litecoin: ¿Cuál es la mejor criptomoneda para ti?* 05-03-2018. cnet. URL: https://www.cnet.com/es/como-se-hace/bitcoin-ethereum-o-litecoin-criptomoneda/.

[9] *Bitcoin, la moneda que está cambiando el mundo.* 28-04-2018. Qué es Bitcoin. URL: https://www.queesbitcoin.info/.

[10] *Bitcoin open source implementation of P2P currency.* 08-03-2018. S. Nakamoto. Febrero 2009. URL: http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source.

[11] *Block 0 - TEST Bitcoin Block Explorer.* 09-03-2018. Block Explorer. URL: https://testnet.blockexplorer.com/blocks-date/2011-02-02.

[12] *Blockchain: desarrollo en Ethereum.* 12-03-2018. deloitte. URL: https://www2.deloitte.com/es/es/pages/technology/articles/blockchain-desarrollo-en-ethereum.html.

[13] *Blockchain primera, segunda y tercera generación! Entendiendo sus diferencias.* 16-03-2018. steemit. URL: https://steemit.com/spanish/@cosmicboy123/blockchain-primera-segunda-y-tercera-generacion-entendiendo-sus-diferencias.

[14] *Bootstrap.* 18-03-2018. Bootstrap. URL: https://getbootstrap.com/.

[15] *Build unstoppable applications.* 27-02-2018. Ethereum. URL: https://ethereum.org/.

[16] Francisco J. González Cabrera. *Tema II.2: Modelado algorítmico de costes.* 05-05-2018. Escuela ingeniería Informática Segovia. 2017.

[17] *Coinbase.* 05-05-2018. Bitcoinwiki. URL: https://en.bitcoin.it/wiki/Coinbase.

[18] *Counterparty.* 08-04-2018. Wikipedia. URL: https://en.wikipedia.org/wiki/Counterparty.

[19] *Cryptocurrency Market Capitalizations.* 08-03-2018. cryptocoincharts. URL: https://cryptocoincharts.info/coins/graphicalComparison.

[20] *Cryptozombies.* 08-04-2018. Cryptozombies. URL: https://cryptozombies.io/es/.

[21] *Definición Bitcoin 2.0: ¿Qué es Bitcoin 2.0?* 18-03-2018. oroyfinanzas. URL: https://www.oroyfinanzas.com/2014/11/definicion-bitcoin-2-0-que-es-bitcoin-2-0/.

[22] *eBay.* 28-03-2018. wikipedia. URL: https://es.wikipedia.org/wiki/EBay.

[23] *El funcionamiento de la Red Bitcoin.* 07-03-2018. Qué es Bitcoin. URL: https://www.queesbitcoin.info/como-funciona-bitcoin/.

[24] *Ether.* 28-02-2018. ethdocs. URL: http://www.ethdocs.org/en/latest/ether.html.

[25] *Ethereum Development Walkthrough (Part 2: Truffle, Ganache, Geth and Mist).* 28-04-2018. hackernoon. URL: https://hackernoon.com/ethereum-development-walkthrough-part-2-truffle-ganache-geth-and-mist-8d6320e12269.

[26] *Ethereum Meetup Sevilla.* 23-05-2018. Meetup. URL: https://www.meetup.com/es-ES/Ethereum-Meetup-Sevilla/?_af_cid=Ethereum-Meetup-Sevilla&rv=wm2&_xtd=gatlbWFpbF9jbGlja9oAJGE5YzU4NGFmLWVlYzItNGI2OS1iYWUwLWJjMzNmNTJhMWIzZA&_af=chapter&https=on.

[27] *ETHEREUM WALLET / MONEDERO.* 27-02-2018. miethereum. URL: https://miethereum.com/ether/wallet-monedero/.

[28] *Freedom of information.* 08-05-2018. namecoin. URL: https://namecoin.org/.

[29] *Function Point Languages Table.* 25-04-2018. qsm. URL: http://www.qsm.com/resources/function-point-languages-table.

[30] *Here Are 10 of the Biggest Corporate Hacks in History.* 25-03-2018. fortune. URL: http://fortune.com/2017/06/22/cybersecurity-hacks-history/.

[31] *Introduction.* 26-04-2018. Swarm. URL: http://swarm-guide.readthedocs.io/en/latest/introduction.html.

[32] *Introduction to Bitcoin and Existing Concepts.* 27-02-2018. ethereum. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[33] *IPFS is the Distributed Web.* 17-05-2018. ipfs. URL: https://ipfs.io/.

[34] *Los diferentes tipos de consenso del Blockchain.* 19-03-2018. Karl Booklover. URL: https://www.karlbooklover.com/consensos-del-blockchain/.

[35] *Mastercoin Seeks Second Start With Omni Reboot.* 09-05-2018. Coindesk. URL: https://www.coindesk.com/mastercoin-new-beginning-omni/.

[36] *Namecoin.* 13-05-2018. Wikipedia. URL: https://es.wikipedia.org/wiki/Namecoin.

[37] *Onename.* 16-05-2018. onename. URL: https://onename.com/.

[38] *Organización autónoma descentralizada*. 29-03-2018. Wikipedia. URL: https://es.wikipedia.org/wiki/Organizaci%C3%B3n_aut%C3%B3noma_descentralizada.

[39] *Perfil de Youtube de Nicolas Palacios*. 03-03-2018. Youtube - Nicolas Palacios. URL: https://www.youtube.com/channel/UC_U6Wft5UemcWBBK2jfRXlA.

[40] *Qué es blockchain: la explicación definitiva para la tecnología más de moda*. 25-05-2018. xataka. URL: https://www.xataka.com/especiales/que-es-blockchain-la-explicacion-definitiva-para-la-tecnologia-mas-de-moda.

[41] *Qué es el blockchain y cómo se usa para combatir el fraude*. 30-03-2018. medium. URL: https://medium.com/@moften/qu%C3%A9-es-el-blockchain-y-c%C3%B3mo-se-usa-para-combatir-el-fraude-b4e480446b2d.

[42] *remix*. 01-03-2018. remix. URL: https://remix.ethereum.org/.

[43] *Rpow*. 25-04-2018. Cryptome. Agosto 2004. URL: http://cryptome.org/rpow.htm.

[44] *Smart contracts, ¿Qué son, cómo funcionan y qué aportan?* 06-04-2018. bit2me. URL: https://blog.bit2me.com/es/que-son-los-smart-contracts/.

[45] *Solidity*. 17-03-2018. solidity. URL: https://solidity.readthedocs.io/en/v0.4.24/.

[46] *Testnet*. 14-04-2018. Bitcoinwiki. URL: https://en.bitcoin.it/wiki/Testnet.

[47] *The Counterparty Platform*. 10-04-2018. Counterparty. URL: https://counterparty.io/platform/.

[48] *The Different Proofs of Crypto Currency*. 18-03-2018. steemit. URL: https://steemit.com/cryptocurrency/@killjoy/the-different-proofs-of-crypto-currency.

[49] *The Raiden Network*. 26-04-2018. Raiden Network. URL: https://raiden.network/.

[50] *The World's First Autonomous Data Network*. 30-04-2018. Safe Network. URL: https://maidsafe.net/.

[51] *Web3.js*. 17-05-2018. github/ethereum. URL: https://github.com/ethereum/web3.js/.

[52] *What is Ethereum?* 28-02-2018. ethdocs. URL: http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html.

[53] *Whisper*. 27-05-2018. github. URL: https://github.com/ethereum/wiki/wiki/Whisper.

[54] *Working with ganache*. 18-05-2018. truffleframework. URL: https://truffleframework.com/docs/ganache/using.

[55] *Your Ethereum swiss army knife*. 18-05-2018. truffleframework. URL: https://truffleframework.com/.