TronAuction: plataforma de subastas online utilizando *Smart Contracts*

Trabajo de Fin de Grado

Grado en ingeniería informática

VNiVERSiDAD
Ð SALAMANCA

Septiembre de 2021

Autor

Carlos Álvarez López

Tutores

Juan Manuel Corchado Rodríguez

Diego Javier Valdeolmillos Villaverde

Agustín San Román Guzmán

Dr. Juan Manuel Corchado Rodríguez, catedrático adscrito al Departamento de Informática y Automática de la Universidad de Salamanca, D. Diego Javier Valdeolmillos Villaverde y D. Agustín San Román Guzmán,

Hacen constar:

Que el trabajo titulado "*TronAuction*: plataforma de subastas online utilizando *Smart Contracts*" ha sido realizado por D. Carlos Álvarez López, con DNI 70918305J y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo de Fin de Grado de la Titulación Grado en Ingeniería Informática de esta Universidad.

Y para que así conste a todos los efectos oportunos.

En Salamanca, a 6 de septiembre de 2021

CORCHADO RODRIGUEZ JUAN MANUEL - 70978310B

Firmado digitalmente por CORCHADO RODRIGUEZ JUAN MANUEL - 70978310B
Fecha: 2021.09.06 11:03:33 +02'00'

Dr. Juan Manuel Corchado Rodríguez

VALDEOLMILLOS VILLAVERDE DIEGO JAVIER - 71933273Z

Firmado digitalmente por VALDEOLMILLOS VILLAVERDE DIEGO JAVIER - 71933273Z
Fecha: 2021.09.06 10:50:40 +02'00'

D. Diego Javier Valdeolmillos Villaverde

SAN ROMAN GUZMAN AGUSTIN - 70919269B

Firmado digitalmente por SAN ROMAN GUZMAN AGUSTIN - 70919269B
Fecha: 2021.09.06 10:53:19 +02'00'

D. Agustín San Román Guzmán

Resumen

La utilización de dinero a través de Internet ya sea para realizar pagos o transferencias, es uno de los mayores motivos de desconfianza por parte de los usuarios de la red. Siempre surge la duda de si el dinero llegará al destino o de si el pago se habrá realizado correctamente, además de la posible desconfianza que suscita la posibilidad de un ataque de los ciberdelincuentes. Es por ello por lo que se están buscando nuevas tecnologías que garanticen la seguridad de nuestras operaciones, de entre todas ellas destaca *blockchain*, una nueva forma de actuar mediante una red distribuida de bloques que aumenta de forma destacada la seguridad de las transacciones.

Pero blockchain, es mucho más, ha supuesto una revolución en el mundo de la economía ya que es la base para el funcionamiento de las criptomonedas, ese dinero digital que está moviendo millones de transacciones económicas trayendo de cabeza a la banca y a los gobiernos, por esa facilidad que presenta para poder actuar sin intermediarios. Su uso se extiende cada vez más en multitud de sectores, como el almacenamiento en la nube, la garantía de la identidad digital, los registros y verificaciones de datos o las cadenas de suministros.

Una utilidad que se le ha querido dar a *blockchain* con este proyecto, es su aplicación en el sector de las subastas por internet, un campo con un progresivo desarrollo y con un futuro prometedor. El ejemplo más claro sería el de las subastas de productos por internet, con plataformas como eBay, totalmente consolidadas. Lo que se pretende con el proyecto es crear una infraestructura que permita el uso de *blockchain* para una plataforma de subastas, utilizando criptomonedas como medio de pago, pero con el trasfondo de una tecnología mucho más segura que va a utilizar *smart contract*, que son fragmentos de código ejecutables en la cadena de bloques. Una vez que este código se despliega en la red no podrá ser modificado, lo cual sumado a la transparencia propia de esta tecnología permite garantizar las transacciones.

Con este proyecto se pretende conseguir una mejora en el proceso de las subastas online, garantizando la seguridad de los pagos, y solucionando los problemas que presentan actualmente las aplicaciones disponibles, como son la existencia de falsas pujas, las comisiones bancarias y el tiempo que conlleva realizar depósitos para subastas oficiales.

Palabras clave: *blockchain*, subasta, criptomoneda, online, *token*, *smart contract*, minería, sistema distribuido, bloque, puja.

Abstract

The use of money over the Internet, whether for payments or transfers, is one of the biggest sources of mistrust on the part of Internet users. There are always doubts as to whether the money will reach its destination or whether the payment has been made correctly, in addition to the possible mistrust raised by the possibility of an attack by cybercriminals. Therefore, new technologies are being sought to guarantee the security of our transactions, among which blockchain stands out, a new way of acting through a distributed network of blocks that significantly increases the security of transactions.

But blockchain is much more, it has brought about a revolution in the world of economics as it is the basis for the functioning of cryptocurrencies, this digital money that is moving millions of economic transactions, turning banking and governments upside down, due to the ease with which it can act without intermediaries. Its use is becoming increasingly widespread in a multitude of sectors, such as cloud storage, digital identity assurance, records and data verification or supply chains.

One of the uses that blockchain has been given with this project is its application in the internet auction sector, a field that is progressively developing and has a promising future. The clearest example would be that of online product auctions, with platforms such as eBay, which are fully consolidated. The aim of the project is to create an infrastructure that allows the use of blockchain for an auction platform, using cryptocurrencies as a means of payment, but against the backdrop of a much more secure technology that will use smart contracts, which are fragments of executable code on the blockchain. Once this code is deployed on the network, it cannot be modified, which, together with the transparency inherent to this technology, allows transactions to be guaranteed.

This project aims to improve the process of online auctions, guaranteeing the security of payments and solving the problems that currently exist in the available applications, such as the existence of false bids, bank commissions and the time it takes to make deposits for official auctions.

Keywords: blockchain, auction, cryptocurrency, online, token, smart contract, mining, distributed system, block, bidding.

Memoria del proyecto

Tabla de contenido

Índice de figuras

Índice de tablas

1 Introducción

El término subasta aparece por primera vez en la época Románica, etimológicamente se trata de un vocablo derivado del latín, sub (bajo) y hasta (lanza). El cual designaba una forma de vender bienes a viva voz. En las subastas, los compradores realizaban una serie de ofertas ascendentes por los productos, de modo que el bien subastado se adjudicaba al comprador con la mayor oferta.

En el Imperio Romano, la palabra sub hizo referencia a las ventas que contaban con la protección de la fuerza pública. Y es por esto, que originalmente las subastas solamente se dedicaban a la venta de botines militares, pero con el paso de los años se fueron ampliaron a prácticamente cualquier tipo de bienes o mercancías, tanto de propiedad pública como privada.

A su vez, el termino hasta (asta) hace referencia al mecanismo de emplear una lanza para sujetar un paño, principalmente de color rojo en el cual se podía leer el nombre del martillero (subastador). Este símbolo era utilizado para indicar el lugar de celebración de una subasta en la vía pública, principalmente en los foros.

A pesar del origen romano de palabra, las primeras subastas tuvieron lugar en Babilonia en el año 500 a.C. Estas se celebraban anualmente y en ellas se vendían mujeres núbiles. Las jóvenes con mayor belleza eran las que alcanzaban una mayor cuantía, mientras que las de peor aspecto tenían que pagar por presentarse a la subasta, llegando a casos en los que el importe de la subasta era negativo.

A pesar de la gran importancia que tuvieron las subastas en la antigua Roma. También cabe destacar el papel que tuvieron en la antigua Grecia. En este caso podían ser públicas o privadas. En las adjudicaciones públicas se vendían los bienes pertenecientes al estado, mientras que en las subastas privadas eran utilizadas por los habitantes para realizar sus ventas de manera anónima. En ambos tipos de subastas los bienes eran adjudicados al mejor postor.

Tras la caída del imperio romano, las subastas dejaron de tener una gran relevancia, y no fue hasta el siglo XIII cuando volvieron a tomar cierta relevancia. Principalmente para el comercio de productos al por mayor, como por ejemplo para la venta de seda o pescado. Tal fue dicha popularidad, que se construyeron grandes edificios exclusivamente para este propósito, como por ejemplo la Lonja de la Seda (Valencia), construida en el sigo XV para aumentar la capacidad de la antigua lonja valenciana. Las subastas de pescado siguen siendo hoy en día el proceso más habitual para la venta del pescado al por mayor. Normalmente, para este propósito, siempre se han utilizado las subastas holandesas. En ellas el subastador comienza subastando los bienes con un precio alto, el cual va reduciendo hasta que un comprador acepta la oferta. De modo que el primer precio aceptado, será el precio de venta de los bienes.

Fue en el siglo XVII cuando obtuvieron su mayor popularidad y aparecieron nuevos métodos, como por ejemplo la limitación temporal, la cual fue introducida por los franceses. Para ello se encendían 3 velas al comenzar la subasta, y cada vez que se realizaba una puja se encendían dos velas más. De modo que la adjudicación no terminaba hasta que todas las velas se habían consumido.

En estos últimos siglos, las subastas han sido una las formas más utilizadas por el poder público para la adjudicación tanto de bienes públicos como privados.

En la expansión de la tecnología, las subastas han encontrado un gran nicho de mercado. La llegada del internet a todos los hogares ha fomentado la aparición de plataformas de subastas online, como por ejemplo eBay. Estas han logrado que este tipo de adjudicaciones tengan cada día una mayor popularidad en los ciudadanos.

En la actualidad, en plena expansión tecnológica de los servicios públicos, las subastas administrativas de bienes públicos y las subastas judiciales de bienes embargados o decomisados no son una excepción. Para ello, han aparecido empresas dedicadas a la subasta online de los activos de administraciones concursales, como por ejemplo *Pacelma Auctions*. Además de todas estas empresas especializadas, la administración pública cuenta con su propia plataforma de subastas online (https://subastas.boe.es/). En ella, se celebran todas las subastas públicas y judiciales publicadas en el boletín oficial del estado.

Todas estas subastas online, tanto públicas como privadas, tienen una serie de desventajas. La principal desventaja son las falsas pujas. Es decir, pujas que al resultar ganadoras no son pagadas, ya bien sea por falta de fondos o por otras cuestiones. Para evitarlas, algunas casas de subastas, como por ejemplo las subastas del estado, solicitan a los pujadores un porcentaje del valor del bien subastado a modo de abal. Este método tiene varios problemas: la dificultad para gestionar todos los abales de los pujadores, así como su posterior devolución, el tiempo transcurrido entre la realización de una transferencia bancaria y la llegada a la cuenta de destino, sobre todo en transferencias internacionales, o las altas comisiones cobradas por realizar estos depósitos.

Otra gran desventaja son las pujas fraudulentas. El principal método de fraude consiste en realizar una puja por un importe superior al precio de reserva (precio mínimo para la venta del bien) y acto seguido otro pujador aliado realiza una puja con un precio muy por encima al valor del producto. De este modo, el segundo pujador renunciará a su puja perdiendo el importe del depósito y será el primer pujador quien resulte ganador de la subasta. Obteniendo así un precio muy favorable a pesar de tener que sumar el precio de puja ganadora con el abal perdido por el segundo pujador.

En otras ocasiones nos encontramos que es el propio subastador quién sobrepuja al resto de pujadores, de este modo consigue que el artículo no se venda si no alcanza el precio que él desea.

Por otro lado, cuando se realizan subastas internacionales podemos encontrar unos tipos de cambio abusivos, normalmente con comisiones superiores al 10%. Estas comisiones son cobradas por la propia entidad bancaria del pujador o por las pasarelas de pago utilizadas por la plataforma, por ejemplo, PayPal.

Finalmente nos encontramos con las irregularidades relacionadas con el propio artículo, en esta sección podemos distinguir entre dos tipos principales. En el primero, el vendedor nunca llega a enviar el artículo de modo que el comprador se queda sin el dinero y sin el producto comprado. Este fraude normalmente no suele ocurrir, ya que normalmente todas las grandes tiendas de comercio electrónico incorporan un sistema que retiene el dinero para que no llegue al vendedor hasta que el comprador no reciba el producto. Y es en el producto donde se basa el segundo tipo de fraude, precisamente en su calidad, que en algunas ocasiones es muy inferior a la anunciada. Normalmente este problema es solucionado de forma similar al anterior, la plataforma online retiene el dinero durante un plazo a partir de que el comprador recibe el artículo para que pueda realizar una reclamación en caso de que los productos no sean como se describían en el anuncio.

En vista de todos estos inconvenientes, y con el propósito de intentar solventarlos utilizando la tecnología blockchain, se ha desarrollado ***TronAuction***. Una plataforma de subastas online en la cual todo el proceso de pagos y realización de las subastas está automatizado mediante s*mart contracts* (un fragmento de código que se almacena y ejecuta en la *blockchain)*; y cuyo principal objetivo es hacer más accesible este mundo a las personas con menos conocimientos especializados y evitar las posibles estafas.

Para solventar los inconvenientes de las subastas online tradicionales, *TronAuction* utiliza las principales características y ventajas de la tecnología *blockchain*. En primer lugar, gracias a la inmutabilidad de la cadena de bloques conseguimos que tanto las pujas realizadas como las subastas creadas no se puedan modificar. Evitando así que se anulen las pujas en el último momento, o que el vendedor modifique los datos de las subastas justo antes de su finalización y en caso de realizarlo en el plazo permitido siempre quedará registrada la modificación públicamente. Además de estas ventajas, también añade una capa de seguridad extra, ya que ningún dato de los almacenados podrá ser alterado por ciberdelincuentes y siempre estará accesibles a pesar del paso de tiempo.

Por otro lado, se utilizará la posibilidad de realizar transacciones mediante un *token* propio, este seguirá el estándar TRC20. Este *token*, al cual se ha denominado TRA, será el método de pago utilizado en la plataforma. El utilizar este método de pago permitirá el uso de una moneda global, evitando las comisiones por cambio de divisas. Además, añadirá una mayor seguridad para los clientes, para ello se utilizará criptografía de clave pública y privada, la cual tiene una mayor seguridad criptográfica frente el típico sistema de nombre de usuario y contraseña. También facilitará el proceso de gestión de las pujas, aprovechando las ventajas de los s*mart contracts*, será este el encargado de comprobar que se pueda realizar una puja (fondos suficientes), de bloquear esos fondos mientras que siga siendo la máxima puja y de liberarlos cuando se sea sobrepujado. De este modo no será necesario el depósito de abales para participar en la subasta, dado que, si un usuario no tiene los fondos necesarios para realizar la puja, esta no se podrá realizar. En el caso de tenerlos, se efectuará la puja y se bloquearán los fondos en la *wallet* (monedero digital en el que se almacenan los activos electrónicos) del pujador. De este modo se evitarán las falsas pujas, fraudulentas y las comisiones cobradas por las entidades bancarias por el depósito de abales.

Finalmente, cabe destacar una mayor velocidad en las transacciones. Desde que se genera una transacción (una puja, una transferencia, etc.) hasta que se completa, normalmente no pasarán más de 3 segundos. Esto es debido al tiempo de bloque de Tron. Es decir, el periodo que transcurre entre el minado de cada bloque. Cuando una transacción es lanzada a la red, y por lo tanto añadida a un bloque no pasarán más de 3 segundos hasta su validación. Ofreciendo así una gran ventaja frente a los medios de pago tradicionales, cuyas transferencias normalmente tardan 24 horas en llegar pudiendo incluso llegar a tardar días en el caso de las transferencias internacionales.

En el presente documento se desarrollará el proyecto software *TronAuction*. Para ello se presentarán los objetivos del proyecto, seguido de los conceptos teóricos relacionados con la tecnología *blockchain* necesarios para la comprensión y desarrollo del proyecto. Descritos los conceptos teóricos se describirá las técnicas y herramientas utilizadas para la implementación de la plataforma. A continuación, se presentarán los aspectos más relevantes del desarrollo. Para finalizar con la exposición de los resultados, las conclusiones y las líneas de trabajo futuras.

Este documento va acompañado de cinco anexos, los cuales abarcarán la planificación temporal y la viabilidad del proyecto, la especificación de los requisitos del software, las especificaciones de diseño, la documentación técnica de la programación y finalmente un manual de usuario.

2 Objetivos

En los últimos años, debido a la gran expansión de la tecnología y en especial con la llegada de los *smartphones* a prácticamente toda la población, el comercio electrónico ha tomado una gran popularidad, y en especial en este último año, debido a la situación epidemiológica que vivimos. A pesar de esta gran expansión, los principios del comercio online se mantienen inalterados a pesar de sus actuales inconvenientes. Por ello se desarrolla *TronAuction*, una plataforma de subastas online que busca solventar gran parte de los inconvenientes de las subastas online mediante la aplicación de tecnologías emergentes, como, por ejemplo, *blockchain*.

TronAuction tiene como principal objetivo facilitar y mejorar la experiencia de los usuarios en el proceso de compra online, implementando tecnología basada en la cadena de bloques al alcance de cualquier usuario. Para ello, se busca transparencia junto con simplicidad, es decir que pueda ser accesible por cualquier persona independientemente de sus conocimientos técnicos, pero a su vez estas tengan un total control sobre sus datos, productos y *wallets.* Para lograr satisfacer a los clientes, en la mayor medida posible, se han establecido los siguientes objetivos:

- **Gestionar usuarios.** El sistema deberá de ser capaz de registrar nuevos usuarios, modificarlos y de eliminarlos. Estas acciones se deberán desarrollar a través de una interfaz amigable e intuitiva para el usuario, evitando así posibles errores causados por un mal uso de la aplicación. Además, deberá de prevenir la creación de usuarios con emails no válidos, para ello se deberá verificar la dirección correo electrónico antes de acceder a la plataforma por primera vez. Con este sistema también se previene la creación de cuentas masivas.
- **Gestionar *wallets*.** El sistema debe ser capaz de crear *wallets* asociadas a un usuario, así como eliminarlas. Para facilitar el uso de los monederos será el sistema el encargado de la seguridad y del manejo de estos. Se deberá crear una *wallet* por cada usuario al registrarse. El usuario podrá saber los datos de su monedero en todo momento, así como operar con él.
- **Gestionar transferencias de *tokens*.** El sistema permitirá a los usuarios enviar *tokens* desde su monedero al monedero que ellos consideren oportuno, ya bien sea a un monedero de otro usuario como a cualquier monedero externo a la aplicación. Además, deberá permitir recibir fondos en las *wallets* principales de los usuarios.
- **Gestionar subastas.** Se permitirá a los usuarios la creación de nuevas subastas, pero solo se permitirá su modificación y su eliminación en determinadas circunstancias. Para ello, todas las subastas serán creadas en el *smart contract* y las modificaciones se registrarán en el mismo.
- **Gestionar pujas.** El sistema deberá permitir a los usuarios realizar las pujas que ellos consideren oportunas siempre y cuando dispongas de los fondos suficientes. Con esta medida se evitarán falsas pujas.

En cuanto a los objetivos personales, se busca obtener un mayor conocimiento en herramientas y tecnologías emergentes que no han sido desarrolladas a lo largo del Grado en Ingeniería Informática. Así como realizar un primer acercamiento formal al mundo de la investigación científica. Finalmente, se espera obtener cierta experiencia en el desarrollo completo de un proyecto software, así como en la implementación de soluciones funcionales con tecnologías emergentes.

3 Conceptos teóricos

3.1 Sistema distribuido

Se trata de un sistema en los que los componentes hardware o software se encuentran en distintos ordenadores unidos mediante una red. La comunicación entre estos ordenadores se realizará mediante el intercambio de mensajes.

En la actualidad, prácticamente ningún sistema informático está diseñado para ser ejecutado únicamente en una máquina. De hecho, con el paso de tiempo han ido incrementado el número de nodos que forman parte de una infraestructura. Por ello, los sistemas distribuidos cobran una gran relevancia. Facilitando el aumento del número de usuarios en los sistemas, y permitiendo una gran escalabilidad de estos.

El uso de estos sistemas es esencial para la contabilidad distribuida, también conocida como *blockchain*. Se trata de poder intercambiar información entre dos usuarios a través de internet sin la utilización de intermediarios. Para ello se utilizan bloques que se van enlazando linealmente, pero que necesitan de unos equipos (mineros) que autorizan los enlaces mediante la resolución de acertijos matemáticos. Estos mineros a su vez obtienen unos beneficios mediante recompensas con la moneda que utilice la cadena de bloques con la que estén trabajando.

3.2 Criptografía asimétrica

La criptografía es el conjunto de técnicas utilizadas para alterar la representación de los mensajes. Las primeras técnicas criptográficas utilizaban el cifrado simétrico, el cual consiste en utilizar una clave secreta compartida entre los usuarios que quieran cifrar o descifrar mensajes. Para cifrarlos se utiliza la clave compartida para aplicar sustituciones y permutaciones en el mensaje.

En la actualidad la criptografía asimétrica es una de las técnicas criptográficas más potentes, la cual utiliza complejas fórmulas matemáticas basadas en transformaciones, para crear un par de claves (una pública y una privada). Este sistema resuelve algunos problemas complejos de revolver mediante una clave secreta, como puede ser la distribución de claves o la firma digital.

Este par de claves debe seguir los siguientes requisitos: la obtención de la clave privada, conocida la pública, debe ser computacionalmente imposible; debe ser imposible obtener el mensaje cifrado conociendo el texto cifrado y la clave pública; tanto el descifrado, conociendo la clave privada, como el cifrado, conociendo la clave privada, debe ser sencillo.

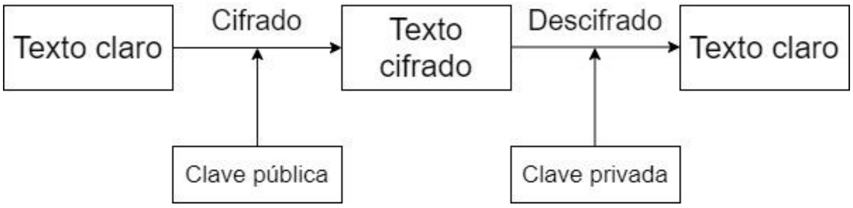

Figura 1: Cifrado asimétrico

El funcionamiento de la criptografía asimétrica se basa en utilizar dos claves distintas. La clave pública, la cual como su nombre indica es conocida por todos los usuarios y es utilizada por los emisores de los mensajes para cifrarlos. Y la clave privada, la cual solo conoce su propietario y es utilizada por este para descifrar los mensajes recibidos.

3.3 Blockchain

3.3.1 Introducción

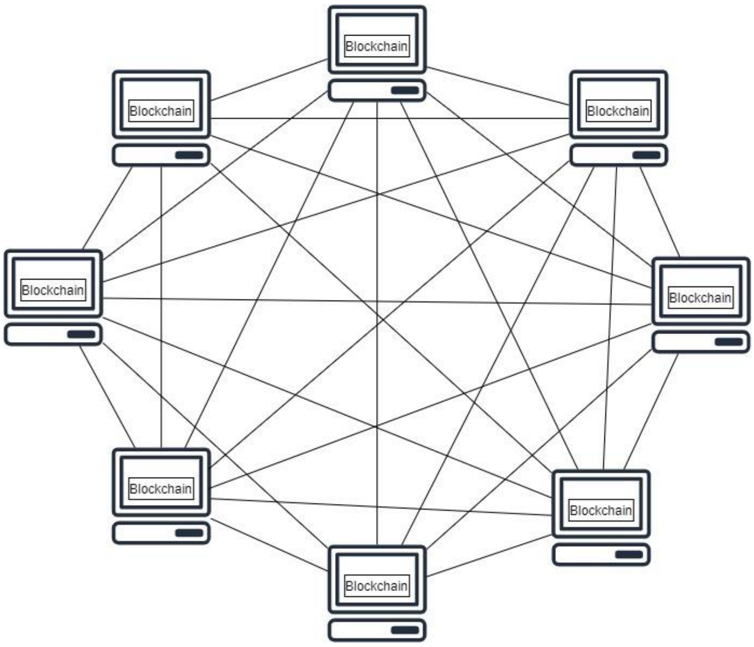

Figura 2. Diagrama arquitectura blockchain

Blockchain o cadena de bloques en español, es una tecnología que se basa en un registro distribuido (una lista de transacciones con una cierta información) en una red de nodos p2p, es decir una red en la que todos los nodos son iguales entre sí, y por lo tanto ninguno tiene privilegios especiales frente al resto. Este registro distribuido utiliza una estructura de datos compuesta por una serie de bloques, los cuales van encadenados uno detrás de otro. Para ello, en cada bloque se hace referencia al bloque anterior, de modo que todos los bloques siguen un orden, y en caso de que faltara alguno, o hubiera bloques duplicados, sería muy fácil identificarlo.

Estructuralmente estos bloques se dividen en dos partes. La primera será el encabezado, donde se incluirá el *hash* del propio bloque, el *hash* del bloque anterior (una referencia al bloque anterior), la fecha y el *nonce* (recuento de transacciones de la *wallet*). Se entiende por *hash*, el resultado de aplicar la función criptográfica de igual nombre a los datos del bloque. La función recibe como parámetro de entrada un flujo de datos de longitud variable y genera a raíz de ellos un identificador único e irrepetible. Gracias a esta función, si alguno de los datos del bloque se alterase, el *hash* no coincidiría con el almacenado.

La segunda parte contendrá la información de todas las transacciones que se han realizado, ya bien sea la transferencia de alguna criptomoneda, el despliegue o la interacción con un *smart contract*, etc. De estas transacciones se almacenarán los parámetros de entrada, los de salida y *hash* de esta.

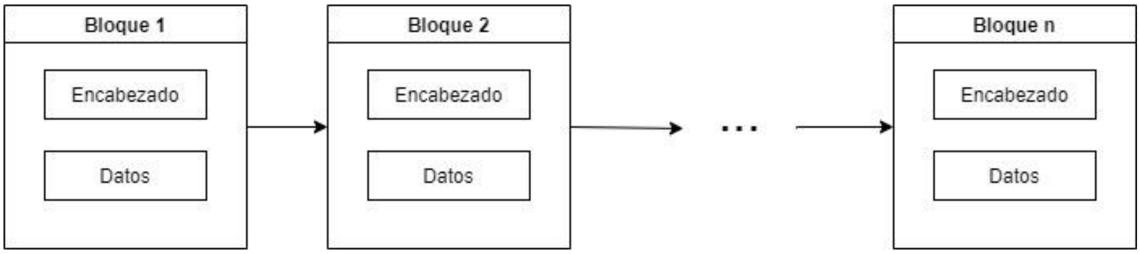

Figura 3. Estructura típica de una cadena de bloques (blockchain)

En todos los nodos de la red siempre habrá almacenada una copia de la cadena actualizada en tiempo real, por lo tanto, si un nodo desaparece no supondrá ningún problema para el sistema. A la hora de introducir un nuevo bloque, todos los nodos deben estar de acuerdo con su inclusión, para ello se utilizará un algoritmo de consenso.

En *blockchain* normalmente se utiliza el algoritmo de consenso *Proof of Work* (PoW). El funcionamiento de este algoritmo es realmente sencillo. Cuando un nuevo bloque quiere ser añadido a la cadena, este será enviado a todos los nodos de la red, los cuales tratarán de calcular el *hash* del bloque completo. Esta operación que puede parecer simple en un primer momento, no lo es, ya que el hash debe reunir una serie de condiciones como pueden ser: ser inferior a un número, tener un determinado número de ceros seguidos, etc. Todas estas condiciones están definidas y dependerán de la cadena de bloques utilizada. Otra característica que debe cumplir esta función *hash,* es que, una vez conocido su resultado, este será fácilmente comprobable. Por lo tanto, los nodos competirán utilizando todos sus recursos hardware para alcanzar la solución lo antes posible y una vez alcanzada la comunicarán al resto de la red, que la comprobarán y validarán. Cuando el bloque esté validado por más de la mitad de la red será añadido a la cadena.

Para poder sostener la cadena e incentivar que haya un mayor número de usuarios, cuando un nodo consigue calcular el *hash* de un bloque recibe una recompensa en la criptomoneda de la cadena. Este proceso para añadir un nuevo bloque es el famoso minado.

El tiempo transcurrido entre el minado de cada bloque se conoce como tiempo de bloque, y es el tiempo que tardan los mineros en descubrir el *hash* del bloque. Para que este tiempo se cumpla de forma regular, la función *hash* va variando su dificultad en función del tiempo que se ha tardado en minar el último bloque. En el caso de *Ethereum* se tiene un tiempo de bloque de 15 segundos, mientas que para *Bitcoin* es de 10 minutos. Gracias a este sistema se garantiza la seguridad de la red, ya que para que un atacante pudiera incorporar un bloque fraudulento debería de tener el control de al menos el cincuenta y uno por ciento de red, lo cual, hoy en día, es imposible.

Una vez añadido un bloque a la cadena, este no podrá ser modificado ni eliminado, por ello la tecnología *blockchain* es capaz de garantizar la inmutabilidad de sus datos almacenados, ya que hay una copia de estos en cada uno de los miles de nodos distribuidos por todo el mundo, por lo tanto, al igual que en el caso anterior, haría falta conseguir el control de más de la mitad de los nodos de la red para poder modificar los datos registrados.

3.3.2 Tipos de cadenas de bloques

En la actualidad se pueden distinguir tres tipos de *blockchains*, las públicas, las privadas y las híbridas. El primer tipo en aparecer fueron las redes públicas. Como su nombre indica, sus datos e implementación pueden ser accedidos por cualquier persona a través de internet. En este tipo de redes cualquiera puede formar parte de ellas. Al tener todos sus datos públicos, se obtiene una transparencia total, de modo que cualquier persona puede revisar o auditar las transacciones realizadas.

En las redes públicas, se utiliza normalmente el algoritmo de consenso PoW. Para mantener económicamente la red, cuando un usuario quiere realizar una transacción debe pagar una comisión, las cuales se destinarán al pago de las recompensas de los mineros. Algunas de las redes públicas más famosas son *Bitcoin*, *Ethereum*, *Monero*, etc.

Por otro lado, tenemos las redes privadas o permisionadas. Las cuales siguen la misma estructura que las redes públicas, pero dependen de una autoridad central encargada de dar acceso a los usuarios, gestionar las transacciones, etc. En estas redes el mantenimiento económico depende de la empresa que la gestiona, por ello normalmente no existe la minería ni las criptomonedas. La red privada más utilizada es *Hyperledger*.

Finalmente nos encontramos con las redes híbridas, las cuales tienen las mejores características de las redes públicas y las privadas. En este tipo de redes el acceso está controlado por una o varias autoridades, mientras que los datos almacenados son públicos. Esto quiere decir que solamente podrán incorporar datos los nodos autorizados, mientras que los datos podrán ser accedidos por cualquier persona. Con este método se alcanza un almacenamiento muy seguro, a la vez que transparente. Como a la hora de introducir nuevos datos, estos están controlados por una autoridad central, no es necesario el uso de la minería.

3.3.3 Otros algoritmos de consenso

3.3.3.1 Proof of Authority

Anteriormente se ha hablado del algoritmo de consenso PoW, el cual es utilizado en las redes en las que los participantes no se conocen entre sí y por lo tanto el grado de confianza entre ellos es nulo (redes públicas). Además de este algoritmo, existen una serie de algoritmos denominados *Proof of Authority*. Estos están diseñados para trabajar principalmente en entornos donde los participantes se conocen entre sí y hay un alto grado de confianza (redes permisionadas) Algunos de ellos son:

- ***Clique*.** Se trata de un algoritmo de consenso diseñado principalmente para utilizarse en redes privadas o de pruebas, pero también puede ser utilizado en redes públicas. Su funcionamiento se basa en los intervalos de tiempo. En cada intervalo, un nodo conocido como validador propone un bloque, el cual enviará al resto de los nodos validadores. Cada uno añadirá su firma al bloque cuando esté de acuerdo en la validación. Este bloque será confirmado si la mayoría de los validadores lo han firmado (la mitad más uno). Estos validadores están definidos en el *genesis block* de la cadena y para su modificación (eliminar o añadir nuevos firmantes) es necesario que la mayoría de ellos esté de acuerdo. Una característica destacable de este algoritmo es la posibilidad de que se produzcan *forks* (divisiones) de la cadena. De modo que no todos los bloques validados serán incluidos en la *mainchain.*

- **IBFT 1.0.** Es una implementación del protocolo de Fallos Bizantinos. Este algoritmo de consenso ofrece un gran rendimiento a la vez que evita la división de la *blockchain*. Se suele utilizar en redes privadas y permisionadas, principalmente en *Quorum*. En el caso de las redes públicas no se suele utilizar, ya que solo garantiza la integridad cuando menos de un tercio de los nodos son corruptos.

- **IBFT 2.0.** Este algoritmo de consenso basa su funcionamiento en *clique*. Las principales diferencias son que a la hora de validar un bloque es necesario que más de dos tercios de los nodos validadores estén de acuerdo. Además, este protocolo evita que haya bifurcaciones en la cadena, de modo que todos los bloques validados serán incluidos a la *mainchain.* Por otro lado, para que este algoritmo funcione correctamente es necesario que haya al menos 4 nodos validadores.

3.3.3.2 Proof of Stake

En los últimos años han aparecido nuevas propuestas de algoritmos de consenso, como podría ser PoS (*Proof of Stake*) el cual pretende sustituir la prueba de trabajo de los algoritmos PoW, por una prueba de participación.

Con esta nueva tecnología, los nodos “mineros” se sustituyen por nodos validadores de transacciones. Para que un simple nodo pueda funcionar como nodo validador, es necesario que tengan una cierta participación (de aquí el nombre de prueba de participación). Es decir, tener una cierta cantidad de la criptomoneda propia de la cadena de bloques en la que quieran ser validadores.

En los algoritmos PoS, para validar un bloque ya no es necesario realizar complejos cálculos matemáticos para su verificación, sino que un bloque será validado y por lo tanto añadido a la *blockchain* cuando suficientes (cantidad especificada en cada blockchain) nodos verificadores lo confirmen. De este modo desaparecerán las potentes granjas de minado, ya que los nuevos mineros no serán los nodos con mayor potencia de cálculo, si no los nodos con mayor poder económico sobre la red. Por ello se tratará de un sistema mucho más eficiente energéticamente.

Una de las principales *blockchains* que utilizan este algoritmo es Ethereum 2.0, la cual es una bifurcación de la red Ethereum que se aprobó el 1 de diciembre de 2020, y está pendiente de aplicación. Estas dos cadenas de bloques se diferencian principalmente en su algoritmo de consenso.

3.3.3.3 Delegated Proof of Stake

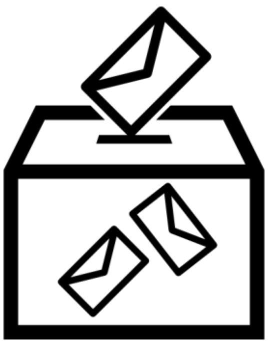

Figura 4: Algoritmo Delegated Proof of Stake

El algoritmo de consenso DPoS, o también conocido como algoritmo de prueba de participación delegada, tiene como objetivo que todos los participantes de la red tengan el poder sobre ella, y no como ocurre con el algoritmo PoS, que solamente los más ricos pueden tomar decisiones y validar bloques. Para ello, DPoS propone un sistema basado en la democracia, es decir, en él todos los usuarios que tengan criptomonedas eligen una serie de representantes (delegados) mediante una votación. Estos representantes serán los encargados tanto de validar los bloques como de tomar las decisiones sobre la cadena de bloques.

A la hora de elegir a los representantes, cada usuario tiene tantos votos como número de criptomonedas tenga. Cada cadena tiene especificado el número de representantes que debe haber, así como el número de bloques que debe validar un delegado al ser elegido. Por norma general se eligen a los representantes cada 1-2 minutos. Con esta medida se consigue que los representantes sean confiables, ya que en caso contrario nunca más serían elegidos y no podrían obtener las recompensas por el minado de los bloques.

Las principales ventajas de este sistema es que todos los usuarios de la red pueden participar en sus decisiones, además maximiza las ganancias de los propietarios de las criptomonedas, ya que cuantas más tengan, mayor participación en la red tendrán. Por otro lado, se tendrá una *blockchain* más económica, debido a que el costo de los equipos disminuye sustancialmente.

En cuanto a las desventajas, se considera que este tipo de redes pueden promover la centralización del poder, ya que son los usuarios más ricos quienes mayor control tienen. A esto se debe sumar, que el correcto funcionamiento de la red solo se alcanzará cuanto toda la comunidad esté activa y participando en las votaciones. En caso contario la *blockchain* puede fallar. Esto es principalmente debido a que hay un gran número de usuarios que consideran que sus votos no son relevantes en comparación con las grandes fortunas de la cadena.

3.3.4 Smart contracts

Los *smart contracts,* o contratos inteligentes, son un fragmento de código almacenado en la *blockchain* que se ejecuta de manera autónoma cuando se cumplen una serie de requisitos prestablecidos, o bien de manera manual, a demanda del programador. Todo ello garantizando la inmutabilidad, la transparencia y la seguridad de los datos.

En la vida cotidiana un contrato es un acuerdo entre dos o más personas en el que se definen una serie de cláusulas y condiciones. Normalmente estos contratos son documentos escritos con un carácter jurídico, y en ocasiones requieren de un notario para su verificación, convirtiéndolos así en un caro trámite.

Gracias a los contratos inteligentes, estos intermediarios pueden ser eliminados, ya que es el propio contrato inteligente quien realizará una serie de acciones en función de los requisitos previos. Por ejemplo, si tuviéramos un contrato inteligente encargado de la venta de una propiedad inmobiliaria, el *smart contract* se encargará de todos los trámites. Si el comprador realiza el pago, el contrato automáticamente le transferirá los fondos al vendedor, y la titularidad del inmueble al comprador. En caso de que no se realice el pago, o que este no sea el acordado, devolverá el dinero al comprador y no se realizará transferencia alguna. Se podría incluso programar de tal modo que se le devuelva al vendedor solamente una parte del dinero, perdiendo así una señal a favor del vendedor.

Este tipo de contratos cuenta con dos de las principales ventajas de la tecnología *blockchain:* la transparencia y la inmutabilidad. Gracias a estas dos características los usuarios pueden conocer todas y cada una de las cláusulas y condiciones que van a firmar (transparencia), junto con la garantía de que no se van a modificar y que siempre serán las mismas (inmutabilidad).

Por lo tanto, con el uso de los contratos inteligentes se consigue una disminución de los costes, una mayor autonomía ya que se eliminan los intermediarios, y finalmente una mayor velocidad y seguridad; utilizando la cadena de bloques pública nunca se podrá perder la información ni modificarla.

3.3.5 Eventos

Los *events,* o eventos en español, es uno de los escasos mecanismos que tiene un *smart contract* para devolver información al exterior. Son un tipo de notificaciones que son lanzadas a demanda del programador cuando ocurre algún hito en el *smart contract*. Por ejemplo, se podría emitir un evento cuando se transfiera una propiedad, indicando la propiedad transferida, el vendedor y el comprador.

Estos eventos quedarán escritos permanentemente en la *blockchain*, de modo que son accesibles públicamente y en cualquier momento. A través de ello se puede monitorizar el estado de un *smart contract* de manera sencilla. Por ejemplo, en el caso que se ha venido describiendo a lo largo de este documento, se podría conocer el número de transferencias que se han realiza en un periodo de tiempo, quién ha sido el vendedor que más propiedades ha vendido, etc.

3.3.6 Criptomonedas

Una criptomoneda es un activo digital, es decir una forma de dinero virtual. Este dinero se basa en la criptografía asimétrica o criptografía de clave pública pasa asegurar la integridad de las transacciones, así como la titularidad de los activos. Este tipo activos digitales tiene una marcada diferencia con el dinero *fiat*. En primer lugar, al ser un activo digital es posible realizar transferencias a cualquier parte del mundo en cuestión de segundos. En segundo lugar, al utilizar la tecnología blockchain como base, no existe ninguna autoridad central o intermediarios, como podrían ser los bancos, en las transacciones. Finalmente, al estar basadas en la tecnología *blockchain*, hereda de ella sus principales ventajas:

- **La transparencia.** Al ser públicos todos los datos almacenados en la cadena, cualquiera puede ver de dónde provienen las monedas y a dónde van.
- **La seguridad.** Con la tecnología *blockchain* nos aseguramos de que los datos son inmutables, por lo tanto, nos da un sistema mucho más seguro que los actuales. Al fin y al cabo, lo datos están replicados en miles de ordenadores.
- **La privacidad.** Al utilizar un par de claves públicas y privadas para realizar las transacciones en vez de un nombre, apellidos y un número de identificación, se garantiza la privacidad de las transacciones.

Como anteriormente se ha mencionado, se utiliza la criptografía asimétrica. El par de claves utilizado para firmar las transacciones recibe el nombre de *wallet* (monedero o billetera), el cual tendrá asociado una cantidad de criptomonedas. El equivalente en dinero *fiat* sería una cuenta bancaria. La clave pública será la que se mostrará en la red, a ella se enviarán las transacciones o aparecerá como el origen de estas. Mientras que la clave privada, solamente se utilizará para firmar las transacciones y en consecuencia autorizará a utilizar las criptomonedas almacenadas en la *wallet*. Es por esto, que es de vital importancia guardar la clave privada en un lugar seguro, asegurándonos de que nadie la conozca, a la vez de no perderla, ya que si esto ocurriera todas las criptomonedas asociadas a la *wallet* no podrían ser accedidas nunca más.

3.3.7 Diferencias entre criptomoneda y *token*

En muchas ocasiones se utiliza el término criptomoneda y *token* como sinónimos, pero en la realidad existen ciertas diferencias. Una criptomoneda es la moneda digital nativa de la cadena de bloques, como puede ser el Ether en el caso de Ethereum, el Bitcoin en la propia red de Bitcoin o ADA en la red de Cardano. Todas ellas tienen las mismas características, pueden ser transferidas entre *wallets* y pueden ser minadas. Además, normalmente pueden ser divisibles y tienen un suministro limitado. Por ello se suelen utilizar como el dinero en efectivo, para el pago de bienes o servicios.

Por otro lado, los *tokens* son un activo digital vinculado a un determinado proyecto, por ejemplo, el BNB del *exchange* Binance. La principal diferencia con las criptomonedas es que no necesitan una *blockchain* propia para existir, sino que su creación y funcionamiento se basa en un contrato inteligente, y por lo tanto se podrá utilizar cualquier cadena de bloques que permita el despliegue de *smart contracts*. Al igual que las criptomonedas pueden transferirse, pero no es posible su minado. Ya que, al no tener una cadena de bloques propia, no hay transacciones que verificar.

3.3.8 Tipos de *wallets*

A la hora de almacenar las claves públicas y privadas existen diversos tipos de *wallets*, las más importantes son:

- ***Wallets* físicas o por hardware.** Su aspecto es similar a un pendrive, y al igual que este, se conecta mediante el puerto USB al ordenador. Este tipo de *wallets* es realmente útil para evitar el robo de información cuando un ordenador es infectado por algún tipo de *malware,* ya que es el propio hardware del monedero quien firma las transacciones, de modo que la clave privada nunca llegará al ordenador, solo llegará la transacción firmada. Esta transacción lleva asociado un *hash*, lo cual hace que sea imposible su modificación. En caso de cualquier alteración del contenido, el *hash* se vería afectado y no coincidiría. Algunos diseños de wallets físicas se pueden observa en la siguiente imagen:

Figura 5. Wallets físicas o por hardware.

- ***Wallets* de escritorio.** No son más que un software ejecutado en el ordenador del usuario. Este software descarga una copia de la *blockchain*, la cual mantiene actualizada y es capaz de operar directamente en la red. La principal desventaja es que, en caso de un ciberataque, los atacantes podrían llegar a conseguir las claves privadas del usuario.

- ***Wallets* de papel.** Se trata simplemente de imprimir o escribir el par de claves en un folio. Este sistema es el más seguro, ya que no existe ningún tipo de acceso remoto a la información. Presenta la desventaja de que en caso de deterioro o extravío del papel es imposible recuperar la wallet, perdiendo así todos los fondos almacenados en ella.

- ***Wallets* online**: Se trata de monederos almacenados en la red. Hay dos tipos de ellas, las que se basan en una página web o las que son una extensión para el navegador. Las primeras suelen integrarse en los populares *exchanges*, plataformas web que permiten la compra y venta de criptomonedas con dinero fiat y el intercambio de fondos entre criptomonedas. En la figura, se recoge una comparativa de los principales *exchanges*. Las segundas, las extensiones para navegador, tienen la ventaja de que son capaces de firmar las transacciones en cualquier página web. Cosa que es realmente útil, ya que no implica revelar la clave privada a la aplicación web, si no que directamente se envía la transacción firmada localmente.

Figura 6. Comparativa de las principales exchanges.

3.4 *Token* TRC20

El TRC20 es un estándar para la creación de *tokens* funcionales, es decir todos y cada uno de estos *tokens* serán iguales en tipo y en valor. Por ejemplo, si un *token* tuviera un valor de un euro, el resto de los *tokens* valdrían un euro. La plantilla del TRC20 es un smart contract escrito en Solidity (lenguaje de programación específico para smart contract), diseñado para ejecutarse sobre la red de Tron. Esta definición cuenta con estructura y una serie de métodos predefinidos para facilitar su implementación. Con él, se puede crear de manera relativamente sencilla un *token* propio con las características que uno desee, de hecho, es esta flexibilidad la que lo llevó a ser un estándar, el cual hoy en día es usado en numerosos proyectos de gran relevancia.

Como se ha mencionado anteriormente, la implementación del TRC20 cuenta con una serie de estructuras de datos y predefinidos entre los que caben destacar la transferencia de *tokens* entre *wallets*, obtener el saldo de una cuenta o la definición de suministro total de *tokens* que existen en la red. Además, se definen una serie de eventos, como, por ejemplo, cuando se realiza una transferencia de *tokens*.

4 Metodología, técnicas y herramientas

4.1 Metodología

A lo largo del proyecto se utilizarán las metodologías ágiles debido a la mayor rapidez de desarrollo, así como por su enfoque hacia el producto final. Estas metodologías se basan en la utilización de iteraciones a lo largo del ciclo de vida del proyecto, dando lugar a sucesivos refinamientos de los productos obtenidos. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. De esta forma se minimizan los riesgos al obtener prototipos verificables en cortos periodos de tiempo.

Dentro de las metodologías ágiles, se utilizará Proceso Unificado Ágil (AUP, del inglés *Agile Unified Process*), es una versión simplificada del Proceso Unificado de Rational (RUP, del inglés *Rational Unified Process*), que combina conceptos propios del proceso unificado tradicional con técnicas ágiles, con el objetivo de mejorar la productividad. Es un enfoque intermedio entre la XP (*eXtreme Programming*) y el RUP, adaptándose y evitando las carencias o puntos débiles de estas dos metodologías. La Programación Extrema es considerada en ocasiones demasiado liviana por no especificar cómo crear algunos artefactos, mientras que del Proceso Unificado de Rational se huye por tener demasiadas disciplinas y requerir demasiados artefactos, lo que conlleva mucha documentación y tiempos de desarrollo excesivos. Por ello en el Proceso Unificado Ágil se agrupan en una única disciplina denominada Modelado, las disciplinas de Modelado de Negocio, Requisitos y Análisis y Diseño.

Las principales características del Proceso Unificado Ágil son: iterativo e incremental, dirigido por los casos de uso, centrado en la arquitectura, enfocado en los riesgos y con un ciclo de vida propio. Aquellos elementos con más riesgos o con mayor prioridad se abordan en las etapas más tempranas, para ir obteniendo prototipos que se puedan ir evaluando.

Algunos de los principios de la alianza ágil que se van a cumplir perfectamente en el desarrollo de este proyecto son:

- Satisfacer al cliente mediante la entrega temprana de software con valor. Al tratarse de un trabajo fin de grado el cliente final serán los tutores, así como uno mismo. Gracias a este enfoque enseguida se irán teniendo evoluciones.
- Aceptar que los requisitos cambien, incluso en etapas avanzadas del desarrollo. Esto será de gran utilidad debido a la amplia temática del proyecto, de modo que se podrá añadir nuevas funcionalidades a lo largo del desarrollo.
- Entregar software funcional en el periodo de tiempo más corto posible

El ciclo de vida general de un proceso unificado ágil sigue la siguiente representación:

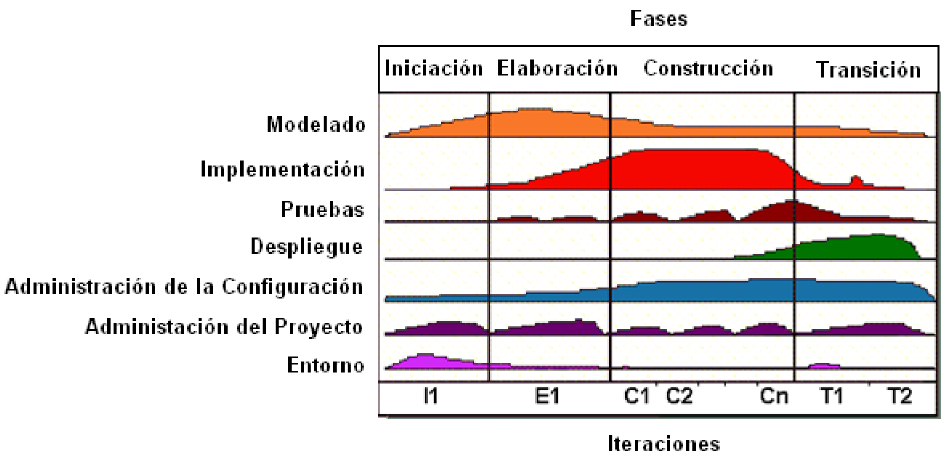

Figura 7: Fases del ciclo de vida

Se componen de 4 fases (columnas) que se abordan de forma secuencial, que son:

- **Iniciación**. Su objetivo es identificar el alcance inicial del proyecto. Entre las actividades que se realizan están la estimación de costes y plazos, definición de los requisitos, estimación de recursos y el cronograma inicial de fases.
- **Elaboración**. Su objetivo es que el equipo de desarrollo profundice en los requisitos del sistema y en validar la arquitectura.
- **Construcción**. Su objetivo es desarrollar el software, de forma incremental, basado en las prioridades de los participantes. Las actividades comienzan con el diseño, para pasar a la implementación y a las pruebas.
- **Transición**. Se despliega el sistema en el entorno real de producción.

A lo largo de las cuatro fases se desarrollan actividades relativas a siete disciplinas (filas). Como se puede observar en la representación del modelo, estas disciplinas no tienen por qué aplicarse en todas las fases, así por ejemplo la disciplina de las Pruebas sólo se aplica en las fases de Construcción y Transición, y en cada fase se pueden aplicar con un peso distinto, por ejemplo, la disciplina de Implementación tiene mucha más importancia en la fase de Implementación que en las dos anteriores. Las siete disciplinas son:

- **Modelado**. Se trata de plasmar la lógica de negocio de la aplicación, determinar el dominio del problema y buscar una solución viable.
- **Implementación**. Transformar los modelos desarrollados en código ejecutable que se pueda probar mediante pruebas básicas unitarias.
- **Pruebas**. Realizar una serie de pruebas que verifiquen que se cumplen los objetivos propuestos y que se verifican todos los requisitos demandados.
- **Despliegue**. Desarrollar un plan para poner a disposición del cliente el sistema para que los usuarios lo puedan utilizar.
- **Gestión de la configuración**. Gestionar los artefactos que se han ido produciendo, así como las versiones de la aplicación y sus cambios.

- **Gestión del proyecto**. Gestionar los elementos internos del proyecto, como los riesgos, el personal asignado, los tiempos de entrega, el progreso y la asignación de las tareas.
- **Entorno**. Proporcionar los elementos necesarios para el desarrollo, desde orientaciones de tipo legislativo, hasta material, pasando por requisitos hardware y software.

En este proyecto, la fase de iniciación tendría un componente fuerte de la disciplina de modelado que estaría compuesta por el apartado de Identificación y descripción del problema y por el apartado de Objetivos. Se llevaría a cabo mediante una única iteración.

Esta fase también tendría apartados importantes de las disciplinas Administración del proyecto y Entorno. En Administración del proyecto se determinan los participantes (en este caso solo uno), y las viabilidades técnicas. En cuanto al Entorno hay que determinar las herramientas software que se van a utilizar. En estos primeros momentos ya se está utilizando la herramienta REM (*REquirements Management*), que soporta la ingeniería de, y *Visual Paradigm* para el diseño gráfico de varios de los artefactos software que se van a obtener.

4.2 Técnicas

4.2.1 Elicitación de requisitos de Durán y Bernárdez

Para la definición de los requisitos software se utilizará la metodología de elicitación de requisitos propuesta por Amador Durán Toro y Beatriz Bernárdez Jiménez en el año 2000, la cual es ampliamente utilizada por la industria del software.

El objetivo de la metodología de Durán y Bernárdez es definir las tareas que se deben realizar, así como los productos finales que se deben obtener y las técnicas a utilizar para lograr una correcta definición de los requisitos.

Esta metodología distingue dos tipos de productos finales: los productos entregables que son aquellos que se entregan al cliente en las fechas acordadas y los productos no entregables, aquellos productos utilizados para el desarrollo del software pero que no son entregados al cliente.

Las tareas propuestas para alcanzar una correcta elicitación de requisitos son las siguientes:

- **Tarea 1:** Obtener información sobre el dominio del problema y el sistema actual (opcional)
- **Tarea 2:** Preparar y realizar las reuniones de elicitación/negociación.
- **Tarea 3:** Identificar/revisar los objetivos del sistema.
- **Tarea 4:** Identificar/revisar los requisitos de almacenamiento de información.
- **Tarea 5:** Identificar/revisar los requisitos funcionales.
- **Tarea 6:** Identificar/revisar los requisitos no funcionales.
- **Tarea 7:** Priorizar objetivos y requisitos.

Por norma general las tareas se deben realizar de forma iterativa en orden ascendente. Pero en algunas ocasiones es posible que pueden realizarse simultáneamente o en orden distinto las tareas 4, 5 y 6.

4.2.2 Patrón MVC

Para la implementación del proyecto se ha utilizado el patrón arquitectónico MVC, cuyas siglas en inglés son: “*Model View Controller”.* El MVC está englobado en la categoría de los patrones para sistemas interactivos, por ello es principalmente utilizado en aplicaciones destinadas a la interacción con el usuario.

Es una realidad que las interfaces de usuario están en constante cambio, ya bien sea para extender la funcionalidad de la aplicación o para adaptarse a un entorno gráfico totalmente distinto. Por ello, crear un sistema flexible capaz de adaptarse dinámicamente a los nuevos requisitos es muy costoso a la vez que lo convierte en un producto propenso a los fallos. Para dar solución a todos estos problemas, se diseñó el patrón MVC, el cual propone aislar la interfaz gráfica de la lógica y los datos de la aplicación. De modo que si una de estas tres partes cambia el resto de las partes no se verán afectadas. Para ello se dividirá el proyecto en las siguientes partes:

- **Modelo (“*Model*”).** En él se encapsularán los datos almacenados por la aplicación, así como las funcionalidades relacionadas con estos.

- **Vista (“*View*”).** Esta parte englobará las interfaces gráficas encargadas de presentar la información al usuario y de interactuar con el mismo.

- **Controlador (“*Controller*”).** Se incluirán los elementos encargados de la lógica de control, haciendo de interfaz entre la vista y el controlador. Estos elementos realizarán todas las transformaciones de los datos necesarias para la correcta visualización. Además, se incluirán las funcionalidades propias de la aplicación.

Como el principal objetivo del presente proyecto es ofrecer a los usuarios una plataforma lo más sencilla e intuitiva posible. Se ha elegido este patrón arquitectónico debido a propiedades orientadas a la interfaz de usuario ayudando a realizar una aplicación web con una mayor flexibilidad de cara a futuras mejoras. Además, la estructuración del proyecto siguiendo los requisitos del MVC facilita la ordenación de los ficheros software de cara a una programación más intuitiva y sencilla.

4.2.3 Patrón DAO

El patrón arquitectónico DAO (*Data Access Object)* permite separar la lógica de acceso a datos de la lógica de negocio. Gracias a este desacoplamiento, es posible cambiar el acceso a los datos (por ejemplo, cambiar de una base de datos MySQL a una base de datos PostgreSQL) sin necesidad de modificar la lógica de negocio. El modelo DAO implementa los métodos necesarios para añadir, modificar, eliminar y consultar (métodos CRUD) la información almacenada en el motor de base de datos.

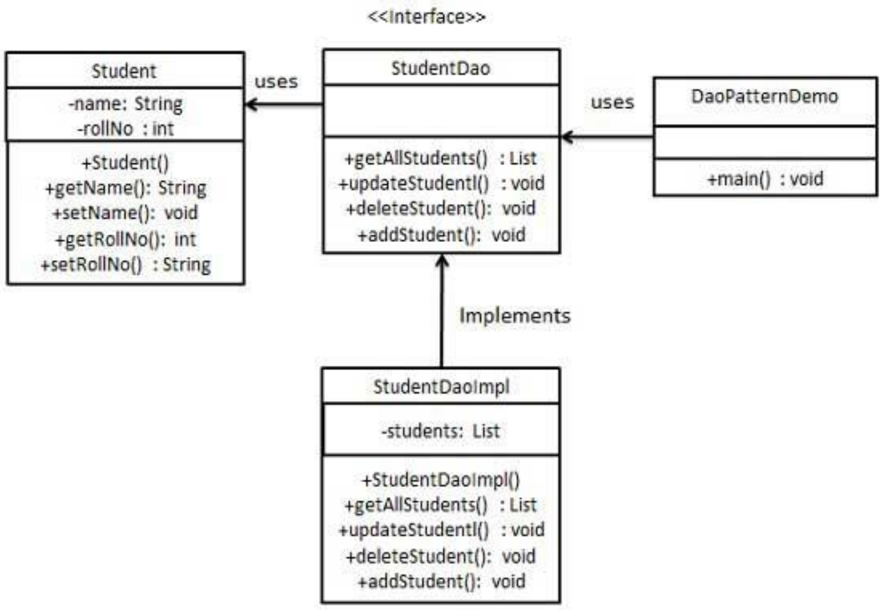

Figura 8: ejemplo diagrama clases DAO

En la figura número 8 se puede observar un ejemplo de aplicación del patrón DAO. La clase S*tudent* utiliza la interfaz del DAO (*StudentDAO),* para el acceso a los datos. Esta interfaz es implementada por la clase *StudentDaoImpl.* la cual incluye la lógica para el acceso a los datos, en este caso almacenados en *DaoPatternDemo.* Con esta implementación, se consigue que, aunque cambie el modelo de datos no se vea afectada la lógica de negocio, ya que esta estará haciendo uso de la interfaz y esta no cambiará, solo su implementación interna.

4.3 Herramientas

4.3.1 Lenguajes de programación

4.3.1.1 Java

Para la realización del proyecto se ha utilizado el lenguaje orientado a objetos Java. Este lenguaje desarrollado en el año 1995 por la empresa *Sun Microsystem* está basado en C y C++ y buscando crear un lenguaje de alto nivel para obtener una mayor simplicidad que estos dos. Durante su creación se buscaban cumplir una serie de requisitos fundamentales, entre los que caben destacar:

- Ser un lenguaje orientado a objetos.
- Permitir ejecutar el código en cualquier máquina independientemente de su sistema operativo. Para ello se utiliza la máquina virtual de Java (JVM) permitiendo que el código sea ejecutado en cualquier sistema.
- Ser un lenguaje de alto nivel, hasta el día de la aparición de Java la mayor parte de los lenguajes de programación eran bastante complejos de utilizar, por ello se buscaba obtener una mayor simplicidad. Permitiendo desarrollar software de una manera más rápida a la vez que sencilla.

Es debido a esta portabilidad entre sistemas, su orientación a objetos junto con su gran popularidad en la industria los principales motivos por los que se ha elegido este lenguaje para el desarrollo del proyecto.

4.3.1.2 Javascript

Javascript es un lenguaje de programación interpretado, esto quiere decir que la mayoría de sus instrucciones se ejecutan directamente sin necesidad de compilar el código. Principalmente se utiliza en aplicaciones web, permitiendo de manera sencilla ejecutar código en el navegador del cliente. Gracias a la gran popularidad que está alcanzando en los últimos años, no es raro ver aplicaciones utilizando este lenguaje para el desarrollo del *backend*.

Algunas de las características más reseñables son:

- Permite definir dinámicamente objetos.
- Utiliza un tipado dinámico de variables, por ello no es necesario especificar el tipo de las variables
- Es un lenguaje orientado a objetos, permitiendo el uso de clases y estructuras.
- Es un lenguaje de alto nivel

A lo largo del presente proyecto se utilizará para ejecutar código en la máquina del cliente, más concretamente en el navegador. Permitiendo realizar peticiones al servidor en tiempo real, o incluso modificar la apariencia de la página web.

4.3.1.3 Solidity

Solidity es un lenguaje de programación estático orientado a objetos diseñado específicamente para el desarrollo de *smart contracts* en diferentes cadenas de bloques, principalmente en *Ethereum*. Aunque también es utilizado por las redes de *Tron*, *Tendermint* o *Counterparty*.

Figura 9: Logotipo de Solidity

Este lenguaje fue propuesto por Gavin Wood en el año 2014 y posteriormente ha sido desarrollado por el equipo de *Ethereum* hasta llegar a convertirse en el lenguaje principal de *Ethereum*.

A lo largo del proyecto se utilizará Solidity como lenguaje de programación para el desarrollo de los *smart contracts.*

4.3.1.4 UML

Unifed Modeling Languaje o UML es un lenguaje diseñado para especificar, visualizar, construir y documentar los artefactos de un sistema software. Para ello define un lenguaje de modelado visual, implementado una interfaz gráfica entendible por las personas y con la capacidad de modelar sistemas orientados a objetos de gran complejidad. Este lenguaje será utilizado en el proyecto para las fases de análisis y diseño del sistema.

UML dispone de un gran número de herramientas de análisis y diseño, para el desarrollo del presente proyecto se han utilizado las siguientes:

- Diagramas de caso de uso: En ellos se relacionan los actores y los caos de usos presentes en el sistema.
- Diagramas de clases: Modelan las clases del sistema, aportando una vista gráfica e intuitiva de las relaciones entre ellas.
- Diagramas de paquetes: Su objetivo es separar el sistema en diversos paquetes y establecer las relaciones de dependencia entre ellos.

- Diagramas de secuencia: Modelan la interacción entre los objetos del sistema en los diferentes casos de uso.

4.3.2 Herramientas *Blockchain*

A lo largo del siguiente apartado se describirán las herramientas utilizadas para el desarrollo blockchain.

4.3.2.1 *Tron*

Figura 10: Logo de la plataforma Tron

Tron es una cadena de bloques fundada por Justin Sun in 2017, en sus orígenes fue un simple *token* ERC-20 corriendo sobre la red de Ethereum, pero en junio de 2018 se convirtió en una *blockchain* completa. La criptomoneda de esta nueva red se llamaría TRX.

Tron es una plataforma de código libre y descentralizada enfocada en crear un ecosistema de entretenimiento descentralizado. Por ello, desde un principio se ha convertido en uno de los principales competidores de Ethereum de cara a ser la cadena preferida por los desarrolladores de *Dapps* (aplicaciones descentralizadas).

Las principales características técnicas de *Tron* son las siguientes:

- Utiliza como algoritmo de consenso el DPoS o Prueba delegada de participación.

- Tiene 27 delegados que se encargan de validar los bloques en la red y estos delegados son elegidos cada 6 horas.

- El tiempo entre bloques es de 3 segundos, aportando una mayor rapidez entre transacciones que sus competidores. Por ejemplo, 15 segundos en Ethereum.

- Como se mencionó anteriormente su criptomoneda se llama TRX, que a su vez se pueden dividir en *sun*. 1 TRX = 1000000 *sun*.

- Al igual que ocurre en *Ethereum*, realizar una transacción en la red de *Tron* lleva un coste asociado. Este se mide en ancho de banda y energía. El ancho de banda es una comisión que se debe pagar en todas las transacciones, mientras que la energía solo será necesaria en aquellas transacciones que estén implicados *smart contracts.* Esta energía se corresponde con el tiempo de CPU necesario para ejecutar el método del contrato, cada milisegundo de cómputo será una unidad de energía. En *Tron*, los usuarios disponen de 5000 puntos de ancho de banda gratuitos diarios, lo que implica realizar de manera gratuita hasta 25 transacciones diarias. Cuando el usuario necesita más ancho de banda existen dos opciones: la primera de ellas es comprar ancho de banda con los TRX que tiene el usuario; la segunda consiste en que el usuario congele una cantidad de sus TRX durante 72 horas. Esto quiere decir, que el usuario dejará una cantidad de sus *tokens* bloqueados durante 72 horas a cambio de ancho de banda, una vez que ha transcurrido este tiempo, los *tokens* serán desbloqueados y el usuario podrá operar con ellos. A la hora de conseguir energía se seguirán los mismos métodos. Por lo tanto, con la segunda opción se podrá operar en la red de *Tron* de manera totalmente gratuita, cosa que no pasa en la red de *Ethereum*.

- Para que un usuario tenga derecho a votar a sus representantes, debe tener una cantidad de TRX congelada, de esta forma se asegura que solamente los usuarios reales de la red puedan ejercer su derecho al voto. El usuario tendrá tantos votos como monedas tenga congeladas.

Como es de esperar no todo iban a ser ventajas en esta cadena de bloques, y la principal desventaja está relacionada con la descentralización. Se considera que *Tron* está en cierto modo centralizada, ya que la mitad de los TRX se almacenan únicamente en 10 *wallets*.

Se utilizará *Tron* como cadena de bloques para el presente proyecto por sus grandes ventajas frente a otras redes, principalmente frente a *Ethereum*. Se tiene que las transacciones se ejecutarán 5 veces más rápidas, por lo tanto, se tendrá una aplicación mucho más rápida y funcional. Además, se podrán obtener transacciones totalmente gratuitas mediante la congelación de TRX. Por otro lado, se trata de una red en continua expansión y cada vez más utilizada por las aplicaciones descentralizadas; si a esto se le suma su gran reputación, tenemos una tecnología idónea para este proyecto. Finalmente, al utilizar el algoritmo DPoS tenemos una de las *blockchains* más cuidadosas con el medio ambiente, a pesar de su posible centralización, la cual recae principalmente en el equipo de desarrollo. Las ventajas son muy superiores a las desventajas dando lugar a la decisión tomada.

4.3.2.2 Shasta

Shasta se trata de una testnet o red de pruebas para proyectos basados en la red de *Tron*. Es decir, es una cadena de bloques idéntica a la red de Tron, con la peculiaridad de que el TRX es gratuito. Este tipo de redes permiten desarrollar las aplicaciones de manera gratuita, y una vez que se pongan en producción será realmente sencillo cambiar a la *mainnet* (red principal, Tron), ya que es una copia idéntica.

En el caso de este proyecto, realmente se ha utilizado la red de Shasta ya que no tiene ningún coste, y permite trabajar con una *blockchain* real permitiendo que en futuro se pueda migrar a la red principal en caso de que fuera necesario.

4.3.2.3 TronLink

TronLink es una extensión de *Chrome*, que no solo ofrece un gestor de *wallets* en el navegador, sino que también permite conectar aplicaciones web con la red de *Tron* y sus principales *testnet*.

A la hora de realizar una transacción vía web en una cadena de bloques existen dos opciones. La primera de ellas consiste en que la propia aplicación gestione tu *wallet* de manera interna, y por lo tanto la firma de la transacción y su envío lo hará el *backend* propio de la web. Con este sistema se consigue que sea mucho más fácil para los usuarios que no tienen conocimientos en *blockchain*, ya que no se deben de encargar de la gestión de sus *wallets*. Mientras que la segunda opción consistiría en usar *TronLink*, en este caso será el usuario quien desde su propio navegador firme la transacción con la *wallet* que elija. La principal ventaja es que el usuario es el encargado de controlar su monedero, y por lo tanto la aplicación en cuestión no tiene acceso a su clave privada.

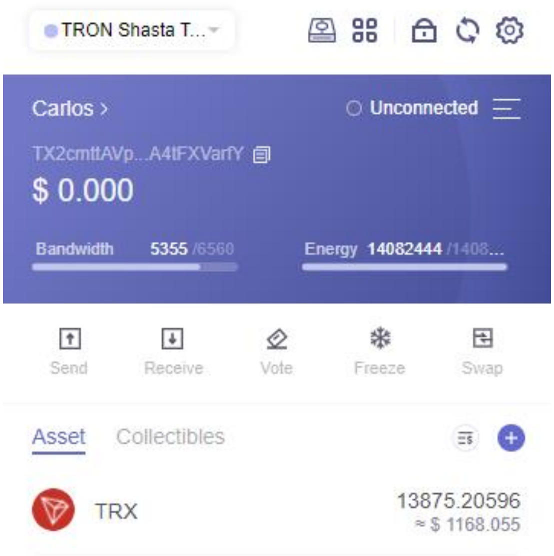

Figura 11: Ejemplo de TronLink

En la figura 11 se puede observar cómo se gestionaría una *wallet* a través de *TronLink*, en ella, el usuario puede seleccionar a qué red conectarse, en este caso a *Shasta*. También puede ver la energía y el ancho de banda disponible (por haber congelado *tokens*). Puede enviar o recibir TRX, votar al representante (siempre que tenga derecho a ello), congelar sus *tokens* para ganar energía o ancho de banda y finalmente intercambiar sus TRX por otras monedas. También se puede observar la cantidad de criptomonedas asociadas a la wallet y su valor en dólares. En este caso sería un valor ficticio, ya que el valor de los TRX en *Shasta* es 0 (el valor mostrado sería si esos *tokens* estuvieran en la *mainnet*).

A pesar de las ventajas que tiene *TronLink*, se ha optado no utilizarlo en la aplicación final, ya que uno de los principales objetivos del presente proyecto es que pueda ser utilizado por cualquier persona sin conocimientos en *blockchain*. Es por ello por lo que todas las *wallets* y firmas de transacciones se llevarán a cabo en el *backend* de *TronAuction* de manera automatizada. Sin embargo, sí que ha sido ampliamente utilizado durante el desarrollo de los *smart contracts*, permitiendo vincular nuestra wallet a aplicaciones web como *tronide.io* o *tronscan*, herramientas fundamentales para el correcto desarrollo del proyecto.

4.3.2.4 Tronide.io

Tronide.io es un entorno de desarrollo web para *Solidity*. Este IDE permite la edición de archivos a la vez que implementa una serie de compiladores de *Solidity* (desde la versión 0.4.24 hasta la 0.8.6). Gracias a esta diversidad se podrá elegir la versión necesaria para nuestro proyecto. Además, permite el despliegue de *smart contract* directamente sobre las distintas redes de *Tron*, tanto la *mainnet* como las *testnet*. Para ello hace uso de *Tronlink*, por lo tanto, es necesario que el usuario disponga de una *wallet* con energía y ancho de banda suficientes para poder ejecutar las operaciones.

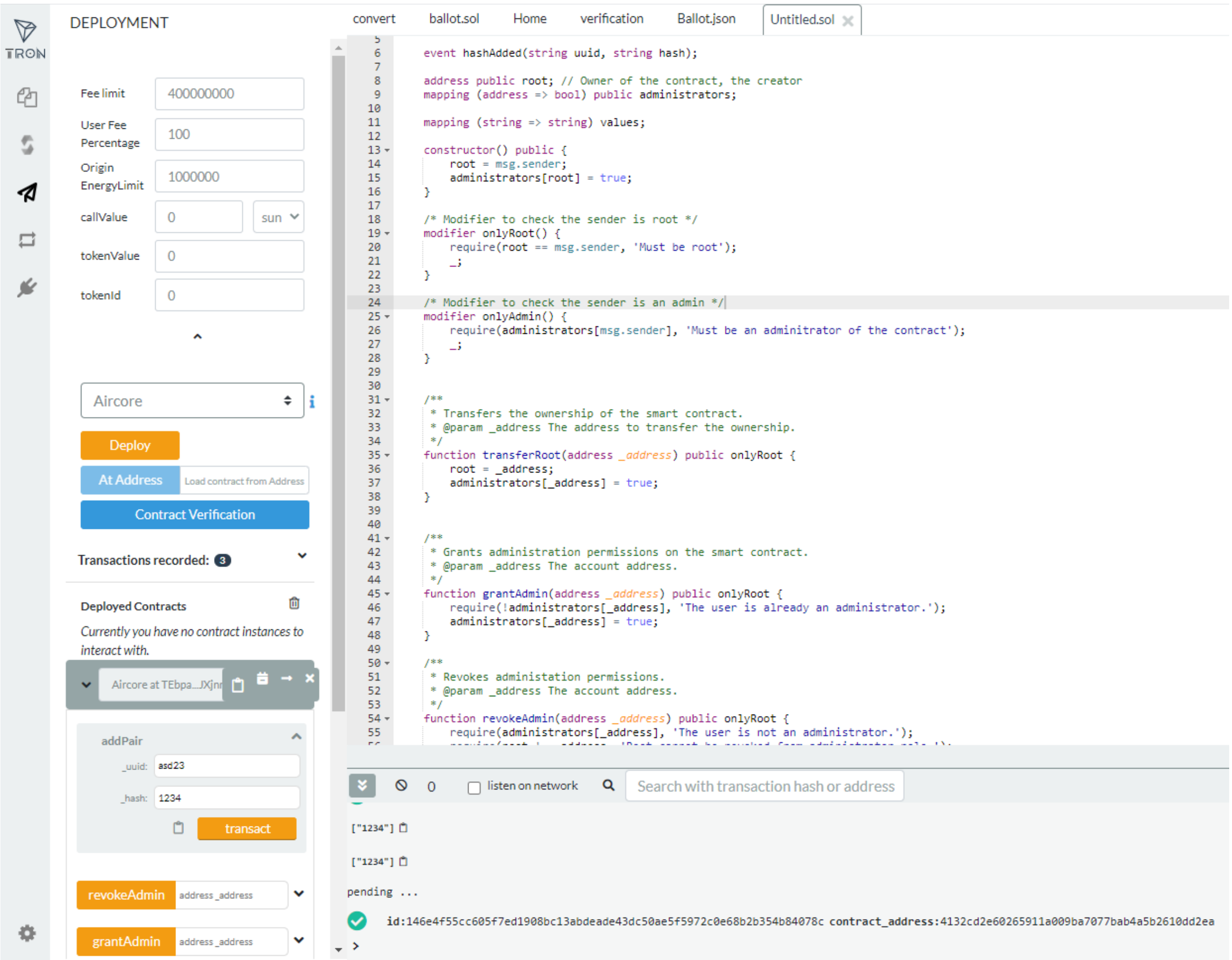

Figura 12: Captura de tronide.io

En la Figura 12 se pueden observar alguna de las funcionalidades de *tronide.io* como por ejemplo la edición de código, y la posibilidad de desplegar *smart contract*. Además, como se puede observar, una vez que un *smart contract* ha sido desplegado sobre la red permite llamar a los distintos métodos de este.

Como ya se sabe gran parte de *Tron* está basada en *Ethereum*, y esta aplicación no es una excepción. Se trata de una pequeña versión del IDE de *Ethereum* (https://remix.ethereum.org/), e incluso se podría considerar una copia incompleta, ya que la interfaz y el funcionamiento es el mismo, pero hay una serie de características que en el IDE de *Tron* no están implementadas.

A lo largo del proyecto se ha utilizado *tronide.io* para el desarrollo y las primeras pruebas de los *smart contracts*. Ya que una sencilla compilación de estos, así como un rápido despliegue, tras el cual será posible interactuar con los *smart contracts*.

4.3.2.5 Tronscan

Tronscan es un explorador de bloques para la red de *Tron*. Un explorador de bloques tendría una función similar a un navegador de internet. Permite a sus usuarios explorar toda la cadena de bloques, desde el primer bloque hasta el último minado. A través de estas aplicaciones web se pueden ver todas las transacciones que han sucedido, las que ha realizado una determinada *wallet*, también permite ver los fondos de los que dispone una *wallet*, ver de dónde provienen esos fondos o incluso ver todas las llamadas que ha recibido un *smart contract*. Por lo tanto, es una herramienta indispensable en el mundo *blockchain*, y de gran ayuda para desarrolladores como para inversores.

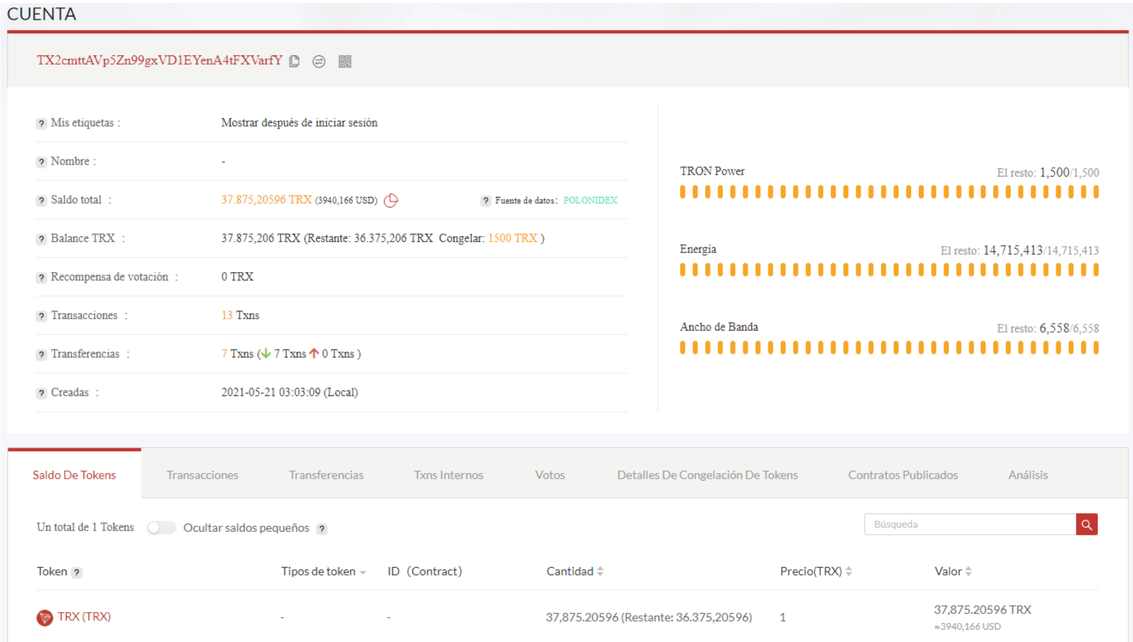

Figura 13: Vista de una wallet en tronscan

En la figura 13 se puede observar los datos disponibles en una *wallet*, entre los que destacan:

- La dirección pública de la propia *wallet*.
- El saldo total de TRX, así como el saldo disponible y el congelado.
- El número de transacciones que ha realizado el monedero, así como el número de transferencias tanto enviadas como recibidas.
- La cantidad de votos disponibles (*Tron power*), el ancho de banda y la energía disponibles.
- Navegando en las pestañas inferiores se podría ver una lista con todas las transacciones realizadas, una lista con las trasferencias, los *tokens* que tiene, los votos emitidos, detalles sobre los *tokens* congelados, los contratos que ha desplegado o finalmente un análisis de todos estos datos mostrados en gráficos.

4.3.2.6 Trongrid

Trongrid es una suite de herramientas para los desarrolladores de *Tron*. La principal utilidad es ayudar a conectar las aplicaciones descentralizadas con la red de *Tron*. Sin la existencia de esta herramienta sería necesario sincronizar un nodo para poder enviar transacciones, con el consiguiente gasto de hardware, eléctrico, etc. Para evitar esta situación, se creó *Trongrid*, una herramienta que permite a los usuarios tener un nodo remoto sin coste alguno. Pudiendo simplemente interactuar con algunos de sus nodos disponibles (en diversas regiones), y no tener que crear uno localmente.

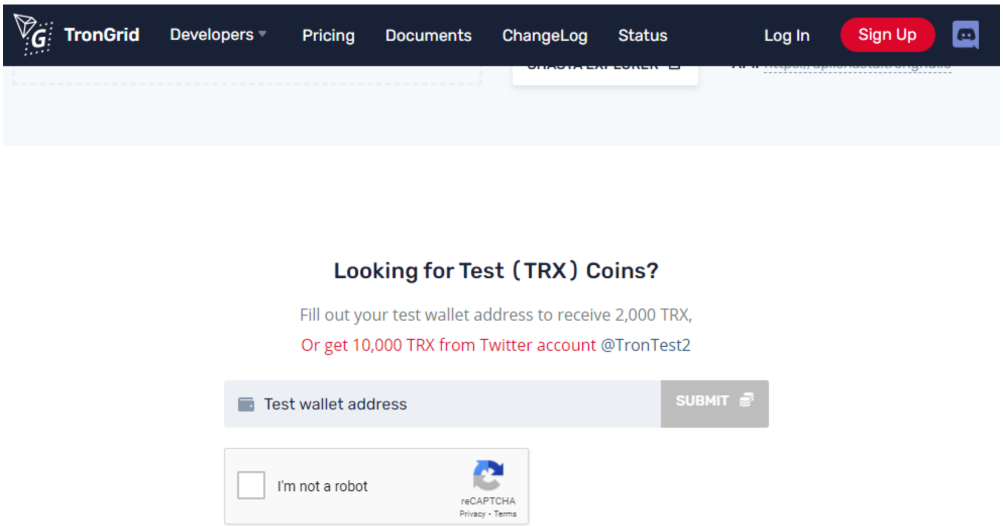

Figura 14: Faucet de trongrid

Por otro lado, dispone de un *faucet* para la red *Shasta*. Cuando un usuario de una *testnet* quiere criptomonedas para poder realizar transacciones en la misma, debe utilizar un *faucet*. Un *faucet* es una herramienta que transfiere gratuitamente una serie de monedas a la *wallet* que se elija. Estas monedas se pueden obtener de dos formas en función del *faucet*. La primera, introduciendo la dirección pública de la *wallet* en la página web del *faucet* y la segunda publicando un *tweet* en el que se mencione la cuenta de Twitter del *faucet* y la dirección pública de la *wallet* en la que se quieren recibir los fondos. Tras realizar alguno de estos dos pasos, en unos pocos segundos se tendrán las monedas en el monedero, y se podrá empezar a utilizar la *testnet*. En el caso de *trongrid* los dos métodos están disponibles, con el primero se pueden conseguir 2000 TRX cada 24 horas y con el segundo 10000 TRX cada 24 horas.

4.3.2.7 *Truffle*

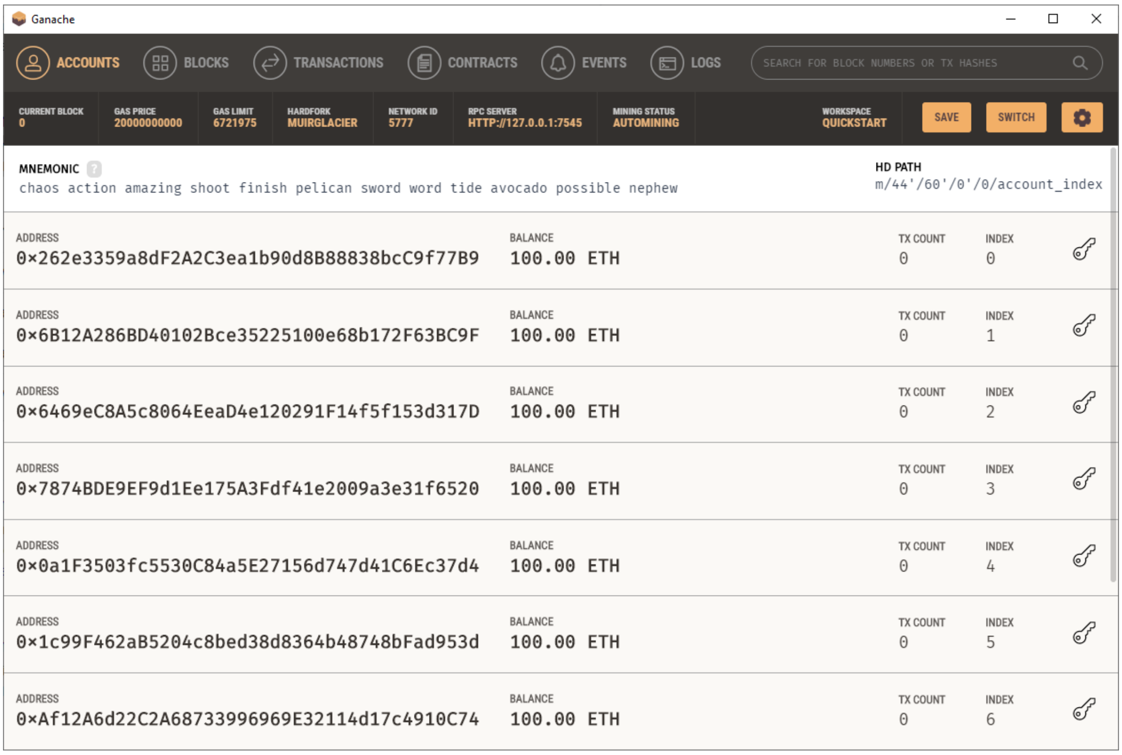

Figura 15: Wallets en Ganache

Truffle es un entorno de desarrollo con funcionalidades ampliadas respecto a *tronide.io*, alguna de estas funcionalidades son el despliegue de *smart contracts* de forma automática en múltiples plataformas como puede ser *Ethereum*, *Tron*, etc.

Además, permite realizar pruebas unitarias a contratos inteligentes, para ello utiliza una cadena de bloques de pruebas de manera local (Ganache) y con una interfaz gráfica. Esta red tiene la característica de que los bloques se minan a demanda, es decir que cuando se ejecute una transacción se minará un bloque sin tener que esperar un determinado tiempo entre cada bloque. Gracias a esta característica se pueden ejecutar muchas operaciones en cortos periodos de tiempo, realizando las pruebas mucho más rápidas que si se tuvieran que ejecutar en una red real, además su coste será 0. Para facilitar aún más las pruebas se generan de manera automática 10 *wallets* con 100 ETH cada una, resultando mucho más rápido y sencillo que conseguir criptomonedas a través de los *faucet* de las *testnets*.

4.3.3 Aplicación Web

En el siguiente apartado se describirán las herramientas más relevantes utilizadas para el desarrollo la aplicación Web.

4.3.3.1 Play framework

Play es un *framework open-source* escrito en *Scala*, aunque también puede ser utilizado por aplicaciones Java. Incluye todos los componentes necesarios para la creación de aplicaciones web y de servicios REST. Por ejemplo, incluye un servidor HTTP integrado, un mecanismo propio para la gestión de las rutas, protección para CRSF, etc. Algunas de sus características son:

- Las aplicaciones siguen el patrón modelo vista controlador.
- Utiliza la compilación en caliente, esto quiere decir que una vez implementada o modificada una funcionalidad de la aplicación web, simplemente con recargar la página compilará los archivos de código modificados y enseñará el resultado del trabajo realizado de manera inmediata. Con este método, consigue optimar en gran medida el tiempo de desarrollo.
- Soporte para Java y para Scala.
- La arquitectura de *Play* utiliza *Akka* para conseguir un consumo de recursos mínimo a la vez que predecible. Gracias a ello las aplicaciones que utilizan este *framework* se podrán escalar tanto vertical como horizontalmente.

4.3.3.2 MySQL

MySQL es un sistema de gestión de bases de datos relacionales desarrollado por Oracle y que se basa en SQL (lenguaje de consulta estructurado).

Las principales características de una base de datos relacional son:

- Almacena la información en una serie de tablas (relaciones) con un identificador único.
- Cada tabla se compone de filas (registros) y columnas (campos).
- Cada registro en una tabla debe tener un campo único (clave primaria).
- Para relacionar dos tablas se hará mediante las claves primarias y foráneas.

En cuanto a MySQL, es el sistema gestor de bases de datos más utilizado hoy en día. Algunas de las ventajas que han hecho esto posible son: ser un software libre y gratuito, no requerir de un *hardware* de alto rendimiento para conseguir realizar operaciones en un tiempo aceptable o su alta seguridad ante accesos no deseados.

Para el desarrollo del proyecto, se ha decido utilizar este software para almacenar todos los datos persistentes necesarios para el correcto funcionamiento de la aplicación. Como por ejemplo la información de inicio de sesión, los datos relacionados con las subastas, etc.

4.3.4 Herramientas auxiliares

4.3.4.1 IntelliJ IDEA

IntelliJ es un IDE o entorno de desarrollo integrado para diversos lenguajes de programación, entre los que cabe destacar Java, Android, *Solidity*, Html, etc. Aparte de soportar numerosos lenguajes de programación, también incluye una serie de *frameworks* de manera nativa, como puede ser *Django*, *Play* o *Spring*. Desde que *IntelliJ* se lanzó al mercado en el año 2001 por la empresa *JetBrains* ha tenido una gran aceptación por la comunidad, hasta el punto de convertirse en la actualizad en uno de los entornos de desarrollo más utilizados por la industria.

IntelliJ IDEA cuenta con dos versiones distintas, *ultimate* y *community*. La primera de ellas es una versión de pago y con muchas más opciones orientadas al desarrollo web, como por ejemplo la integración de un cliente HTTP o el soporte a lenguajes como JavaScript, HTML, etc.

Para el desarrollo del proyecto se ha utilizado la versión *ultimate* por sus funcionalidades añadidas para el desarrollo web. Para ello, se ha utilizado la licencia gratuita disponible para la comunidad universitaria.

4.3.4.2 Git y Github

Git es un *software* para el control de versiones desarrollado por Linus Torvalds. Sus rasgos más característicos son la simplicidad, la eficiencia y ser un software libre. Gracias a todos ellos se ha convertido en el *software* de control de versiones más utilizado hoy en día.

Github se trata de una plataforma online para el control de versiones colaborativa. Para su funcionamiento hace uso de *Git*.

4.3.4.3 Draw.io

Se trata de una herramienta web para la creación de diagramas y esquemas. Destaca por ser una herramienta muy intuitiva a la par de sencilla, aspectos que no le restan funcionalidad.

Esta herramienta ha sido utilizada en el proyecto para la creación de los diversos esquemas mostrados.

4.3.4.4 Visual Paradigm

Visual Paradigm es una herramienta CASE (Ingeniería del *Software* Asistida por Ordenador), utilizada para la creación de múltiples diagramas UML, como por ejemplo de casos de uso, de secuencia, de despliegue, etc.

Esta herramienta ha sido utilizada para la creación de todos los diagramas de la fase de análisis y diseño.

4.3.4.5 Microsoft Proyect

Microsoft Proyecto forma parte de la *suite Office*, propiedad de la empresa *Microsoft*. Es ampliamente utilizado en el ámbito de la gestión de proyecto para planificar y gestionar las tareas. Una vez realizada la planificación permite la generación automática de informes y diagramas, como por ejemplo los diagramas de Gantt.

5 Aspectos relevantes del desarrollo del proyecto

5.1 Introducción

En el presente apartado se pretende resumir los detalles más importantes para una correcta comprensión del proyecto, algunos de estos aspectos pueden ser el ciclo de vida utilizado, los actores que interactúan con el sistema o que requisitos busca satisfacer.

5.2 Ciclo de vida utilizado

Para el desarrollo del presente proyecto se ha seguido un ciclo de vida basado en el proceso unificado. Por lo tanto, se tratará de un ciclo de vida iterativo e incremental, dirigido por los casos de uso y centrado en la arquitectura.

Como se ha descrito anteriormente, el proceso unificado se basa en seguir una serie de iteraciones, las cuales se van adaptando a las distintas fases de este. En este caso, se han realizado un total de cinco iteraciones. La primera de ellas se corresponde con la fase de iniciación, en ella se han llevado a cabo tareas relacionadas con el modelado de la aplicación. Por ejemplo, la investigación de los conceptos teóricos o el análisis de las tecnologías existentes.

La segunda iteración se corresponde con la fase de elaboración, esta se centra en el modelo de requisitos, tanto funcionales, como no funcionales o de información.

La tercera iteración continúa con la fase de elaboración, centrada en el diseño de la aplicación. Como por ejemplo el diseño arquitectónico y el diseño de los datos.

La cuarta iteración se corresponde con la fase de implementación, en ella se han desarrollado tanto los *smart contracts* como la aplicación web. Además, se ha iniciado una primera fase de pruebas.

La quinta y última iteración se corresponde con la fase de transición, en ella se han desarrollado las últimas pruebas al producto y se han dado los últimos retoques al proyecto.

5.3 Estimación del coste y el esfuerzo

En un proyecto software es de vital importancia conocer una estimación de los costes que conllevará el desarrollo. En este caso, al seguirse el proceso unificado y este estar dirigido por los casos de uso, se ha realizado una planificación basada en estos.

En este caso se seguirá el método de estimación de puntos de casos (UCP – *Use Case Points)*, propuesto por Gustav Karner en el año 1998. Para realizar la estimación utiliza el cálculo de tres variables:

- Puntos de caso de uso sin ajustar (UUCP – *Unadjusted Use Case Points*)
    - Complejidad de los casos de uso (UUCW – *Unadjusted Use Case Weigth)*
    - Complejidad de los actores (UAW – *Unadjusted Actor Weight)*
- Factores de complejidad técnica (TCF – *Technical Complexity Factor*)
- Factores de complejidad del entorno (ECF – Enviroment Complexity Factor)

Para llegar a la estimación final de los puntos de caso de uso se utilizará la siguiente fórmula: UCP = UUCP * TCF * ECF

En primer lugar, se debe calcular la complejidad de los casos de uso, para ello se considerará el número de pasos de los escenarios, la complejidad de la interfaz de usuario, el número de entidades de la base de datos a las que accede, y el número de clases implicadas. A cada caso de uso se le debe adjudicar un factor (valor numérico entre 5 y 15 que indica la complejidad del caso de uso). El baremo será el siguiente:

- **Simple** (factor = 5): Caso de uso con 3 o menos pasos.
- **Medio** (factor = 10): Caso de uso con entre 4 y 7 paso.
- **Complejo** (factor = 15): Caso de uso con más de 7 pasos.

Una vez evaluados los factores de complejidad de los casos de uso, simplemente se sumará el factor de cada uno en función de su complejidad. En el caso del presente proyecto se obtiene la siguiente complejidad:

$$\mathrm{UUCW} = UC_{simples} * 5 + UC_{medios} * 10 + UC_{complejos} * 15 = 10 * 5 + 19 * 10 + 0 * 15 = 240$$

Una vez evaluada la complejidad de los casos de uso, se debe evaluar la complejidad de los actores. Para ello se considerará el número y complejidad de estos, siguiendo el siguiente baremo:

- **Simple** (factor 1). El actor es un sistema y la aplicación se comunica con él mediante la invocación de métodos.
- **Medio** (factor 3). El actor es un sistema y la aplicación se comunica con él mediante un protocolo de red.
- **Complejo** (factor 3). Persona con una interfaz gráfica.

Una vez evaluados los factores de complejidad de los actores, se sumará el factor de cada uno:

$$\mathrm{UAW} = ACT_{simples} * 5 + ACT_{medios} * 10 + ACT_{complejos} * 15 = 1 * 1 + 1 * 2 + 3 * 3 = 12$$

Finalmente se deben de evaluar los factores de complejidad técnica y del entorno.

Para calcular los puntos de los factores de complejidad del entorno, se definen ocho factores a evaluar (definidos en el anexo I). A cada uno de esos factores se le asignará un peso (W) acorde a su impacto y una complejidad percibida por el equipo de desarrollo (F).

En el presente proyecto la suma ponderada de todos los factores del entorno es:

$$EPoints = (1.5 * 3) + (-1 * 0) + (0.5 * 2) + (0.5 * 2) + (1 * 3) + (1 * 4) + (-1 * 4) + (2 * 4) = 17.5$$

Una vez evaluados los factores del entorno se calculará el factor total de complejidad del entorno mediante la siguiente fórmula:

$$ECF = 1.4 + (-0.03 * EPoints) = 1.4 - 0.03 * 17.5 = 0.875$$

Al igual que con los factores de complejidad del entorno, para evaluar los factores de complejidad técnica se definen trece factores a evaluar de igual modo que en caso anterior. La suma pondera de todos los factores es:

$$TPoints = (2 * 3) + (1 * 5) + (1 * 1) + (1 * 2) + (1 * 2) + (0.5 * 3) + (0.5 * 2) + (2 * 3) + (1 * 2) + (1 * 1) + (1 * 5) + (1 * 0) + (1 * 1) = 33.5$$

Y para calcular el factor total de complejidad técnica se aplicará la siguiente fórmula:

$$TCF = 0.6 + (0.01 * TPoints) = 0.6 + 0.01 * 33.5 = 0.936$$

Una vez calculados todos estos valores se podrán calcular los puntos de casos de uso mediante la siguiente formula:

$$UCP = UUCP * TCF * ECF = 252 * 0.936 * 0.875 = 206.388$$

El autor de la metodología indica que el número de horas por cada punto de uso es de 20 horas, en investigaciones posteriores se ha llegado a la conclusión de que el tiempo debe estar entre 15 y 30 horas. En este caso se ha considerado el primer valor:

$$Total_{horas} = 206.388 * 15 = 3095\ horas$$

Por lo tanto, para el desarrollo del proyecto se necesitarán 3095 horas.

5.4 Planificación temporal

La planificación temporal de un proyecto es la herramienta que permite programar y gestionar las tareas que se deben desarrollar. Esta planificación marcará el tiempo necesario para el desarrollo del proyecto, que influirá de manera directa en el éxito y la calidad de este.

Para realizar una correcta planificación temporal se identificarán todas las tareas del proyecto, prestando especial atención a aquellas que son críticas, las cuales deben estar en continuo seguimiento para evitar posibles retrasos. Una vez definidas las tareas se evaluarán los recursos necesarios para su realización y finalmente se realizará una planificación de las tareas diarias a ejecutar.

Proyecto	**152 días?**	**lun 01/02/21**	**mar 31/08/21**
Iteración 1: fase de iniciación	**22 días?**	**lun 01/02/21**	**mar 02/03/21**
1.1 Modelado	**22 días?**	**lun 01/02/21**	**mar 02/03/21**
Tarea 1.1.1 Investigación de los conceptos teóricos	13 días	lun 01/02/21	mié 17/02/21
Tarea 1.1.2 Analisis de las tecnologías existentes	5 días?	jue 18/02/21	mié 24/02/21
Tarea 1.1.3 Identificar objetivos	4 días?	jue 25/02/21	mar 02/03/21
Iteración 2: fase de elaboración (I)	**21 días?**	**mié 03/03/21**	**mié 31/03/21**
2.1 Modelado	**3 días?**	**mié 03/03/21**	**vie 05/03/21**
Tarea 2.1.1. Pequeñas correcciones de la iteración anterior	3 días?	mié 03/03/21	vie 05/03/21
2.2 Modelado de requisitos	**18 días?**	**lun 08/03/21**	**mié 31/03/21**
Tarea 2.2.1 Catalogo de requisitos funcionales y definición de los actores	8 días?	lun 08/03/21	mié 17/03/21
Tarea 2.2.2 Catalogo de requisitos no funcionales	6 días?	jue 18/03/21	jue 25/03/21
Tarea 2.2.3 Catálogo de requisitos de información	4 días?	vie 26/03/21	mié 31/03/21
Iteración 3: fase de elaboración (II)	**20 días?**	**jue 01/04/21**	**mié 28/04/21**
3.1 Modelado de requisitos	**2 días?**	**jue 01/04/21**	**vie 02/04/21**
Tarea 3.1.1 Refinamiento de requisitos	2 días?	jue 01/04/21	vie 02/04/21
3.2 Diseño	**18 días?**	**lun 05/04/21**	**mié 28/04/21**
Tarea 3.2.1 Diseño arquitectónico	5 días?	lun 05/04/21	vie 09/04/21
Tarea 3.2.2 Diseño de datos	3 días?	lun 12/04/21	mié 14/04/21
Tarea 3.2.3 Vista de iteración	9 días?	jue 15/04/21	mar 27/04/21
Tarea 3.2.4 Modelo de despliegue	1 día?	mié 28/04/21	mié 28/04/21
Iteración 4: fase de construcción	**66 días?**	**jue 29/04/21**	**jue 29/07/21**
4.1 Modelo de diseño	**3 días**	**jue 29/04/21**	**lun 03/05/21**
Tarea 4.1.1 Refinamiento del diseño	3 días	jue 29/04/21	lun 03/05/21

Figura 16. Ejemplo de catálogo de tareas junto con su asignación temporal

En la figura anterior se puede ver parte de la planificación temporal del presente proyecto. En la columna izquierda se puede observar una lista de las tareas a realizar, divididas en las distintas iteraciones propias del ciclo de vida utilizado. En la parte derecha de la tabla se pueden observar los recursos asignados, número de días, a cada una de estas tareas. Y finalmente en las dos últimas columnas se puede ver la fecha prevista de inicio y finalización de cada una de las tareas.

Una vez realizada esta especificación de tareas y su correspondiente asignación de recursos se puede crear el diagrama de Gantt, el cual permite una representación gráfica de las actividades a realizar frente a una escala de tiempo. En la figura siguiente se puede observar el diagrama de Gantt del presente proyecto, cada una de las barras azules representa una actividad, y su longitud determina su duración.

Este diagrama permite ver fácilmente el orden de ejecución de las actividades y cuáles se realizarán de forma paralela.

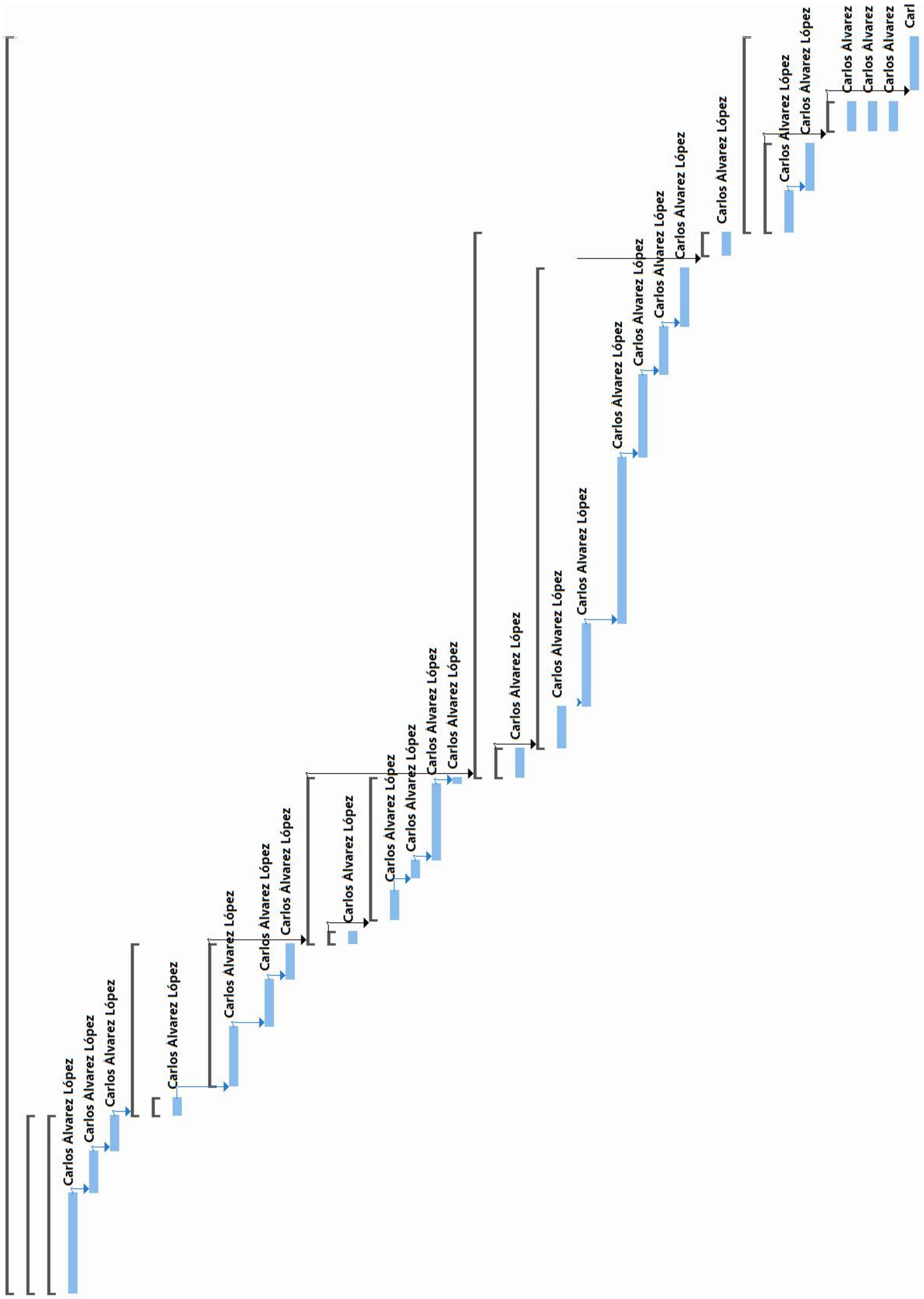

Figura 17. Diagrama de Gantt

5.5 Especificación de requisitos

La especificación de requisitos en un proyecto software supone una primera toma de contacto con el sistema a desarrollar, en ella se busca definir requisitos necesarios con un lenguaje claro y sencillo, con un carácter no técnico.

Para la especificación formal de estos se ha seguido la metodología de elicitación de requisitos propuesta por Amador Durán y Beatriz Bernárdez. La cual propone una serie de tablas para describir los distintos requisitos, que se tratarán en los siguientes apartados.

5.5.1 Listado de participantes

Al ser este proyecto un trabajo de fin de grado, con el que el estudiante busca superar el mismo, el principal participante será el estudiante, pero siempre supervisado por los distintos tutores.

Estas figuras se encuentras descritas mediante las tablas de Durán y Bernárdez en el Anexo II. A continuación, se muestra una tabla de ejemplo:

Participante	**Carlos Álvarez**
Rol	Jefe de proyecto y desarrollador
Organización	Universidad de Salamanca
Desarrollador	Si
Cliente	Si
Usuario	No
Comentarios	Se encargará de todas las tareas relacionadas con la investigación, análisis, diseño, implementación, pruebas y documentación

Tabla 1. Especificación del participante Carlos Álvarez

5.5.2 Objetivos del proyecto

Los objetivos son aquellos aspectos que el proyecto debe cumplir para que el sistema se pueda considerar completo.

Los objetivos buscados a lo largo del proyecto son:

- Gestionar usuarios
- Sistema *blockchain*
- Gestionar *wallets*
- Transferir *tokens*
- Gestionar subastas
- Filtrar subastas
- Gestionar pujas
- Enviar notificaciones

Todos estos objetivos se detallan mediante las tablas propuestas por la metodología usada. En la Tabla 2, se muestra la definición del objetivo gestionar subasta, uno de los principales del sistema.

OBJ-0005	**Gestionar subastas**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	Se permitirá a los usuarios la creación de nuevas subastas, pero solo se permitirá su modificación y su eliminación en determinadas circunstancias. Para ello, todas las subastas serán creadas en el *smart contract* y las modificaciones se registrarán en el mismo
Subobjetivos	**[OBJ-0003] Gestionar *wallets***: Cuando un usuario desee crear una subasta, el sistema debe consultar su *wallet* para recuperar su clave privada, e interactuar con el *smart contract* **[OBJ-0002] Sistema *blockchain***: Cuando se quiera crear o modificar una subasta, se deberá registrar los datos en la cadena de bloques
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 2. Descripción del objetivo gestionar subastas

5.5.3 Requisitos de información

Los requisitos de información representan los datos que el sistema debe almacenar de forma persistente. Los datos que el sistema debe almacenar se pueden ver en la siguiente tabla:

ID	**Nombre**	**Descripción**
IRQ-0001	Usuarios	El sistema deberá guardar la información relativa a los usuarios de la aplicación
IRQ-0002	Susbastas	El sistema deberá almacenar la información relativa a las subastas creadas

IRQ-0003	*Wallets*	El sistema deberá almacenar los datos de las *wallets* de los usuarios
IRQ-0004	Pujas	El sistema deberá controlar las pujas realizadas por los usuarios
IRQ-0005	Notificaciones	El sistema deberá almacenar un histórico de las notificaciones enviadas

Tabla 3. Catálogo de requisitos de información

Al igual que en los casos anteriores cada uno de estos requisitos se definirá con su correspondiente tabla. Para la descripción de todos los requisitos, tanto funcionales, no funcionales o de información, siempre se deben definir una serie de campos, como los objetivos o los requisitos asociados, la descripción, importancia, etc. En este caso, se muestra la definición del requisito *wallet*.

IRQ-0003	***Wallet***	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	0BJ-0003 Gestionar *wallets*	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0005 Interfaces Gráficas	
Descripción	El sistema deberá almacenar la información correspondiente a *las wallets*	
Datos específicos	• ID • ID del usuario • Ruta de almacenamiento • Nombre del archivo • Contraseña	
Tiempo de vida	**Medio**	**Máximo**
	-	-
Ocurrencias simultáneas	**Medio**	**Máximo**
	-	-

Importancia	Importante
Urgencia	Inmediatamente
Estado	Validado
Estabilidad	Alta
Comentarios	Ninguno

Tabla 4. Definición del requisito de información wallet

5.5.4 Requisitos no funcionales

El principal objetivo de los requisitos no funcionales es definir cómo será el sistema final utilizando una serie de restricciones en la calidad del producto. Tiene en cuenta aspectos como la seguridad, el rendimiento o las interfaces gráficas.

En el presente proyecto se han identificado una serie de requisitos no funcionales, principalmente centrados en el rendimiento y la seguridad.

ID	Nombre	Descripción
NFR-0001	Privacidad de la información	El sistema deberá tener la seguridad suficiente para que *no se expongan los datos privados de los compradores y vendedores, respetando la Ley Orgánica de Protección de Datos y la RGPD*
NFR-0002	Portabilidad	El sistema deberá poseer un diseño "*Responsive*" a fin de garantizar su correcta visualización en múltiples ordenadores personales, teléfonos inteligentes y tabletas
NFR-0003	Tiempo de aprendizaje	El tiempo de aprendizaje del manejo del sistema por un usuario deberá ser menor a 2 horas
NFR-0004	Tiempo de respuesta	Toda funcionalidad del sistema deberá responder al usuario en menos de 7 segundos
NFR-0005	Concurrencia de vista	El sistema deberá ser capaz de operar correctamente con hasta 3000 dispositivos de forma concurrente
NFR-0006	Interfaces gráficas	El sistema deberá poseer interfaces gráficas bien formadas y organizadas
NFR-0007	Conexiones seguras	Se debe garantizar la seguridad de las comunicaciones

Tabla 5. Catálogo de requisitos no funcionales

Todos estos requisitos funcionales se han definido formalmente en el Anexo II, un ejemplo sería la tabla que se muestra a continuación:

NFR-0004	**Tiempo de respuesta**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens* OBJ-0005 Gestionar subastas OBJ-0006 Visualizar subastas OBJ-0007 Gestionar pujas OBJ-0008 Enviar notificaciones
Requisitos asociados	IRQ-0001 Usuarios IRQ-0002 Subastas IRQ-0003 *Wallets* IRQ-0004 Pujas IRQ-0005 Notificaciones IRQ-0006 Sesiones UC-0001 Iniciar sesión UC-0002 Cerrar sesión UC-0003 Registrar usuario UC-0004 Modificar usuario UC-0005 Eliminar usuario UC-0006 Recuperar usuario UC-0007 Verificar email UC-0008 Crear *wallet* UC-0009 Eliminar *wallet* UC-0010 Recuperar *wallet* UC-0011 Realizar transacción *blockchain*

	UC-0012 Firmar transacción UC-0013 Enviar transacción UC-0014 Transferir *tokens* UC-0015 Consultar *tokens* UC-0016 Comprar *tokens* UC-0017 Vender *tokens* UC-0018 Crear subasta UC-0019 Modificar subasta UC-0020 Eliminar subasta UC-0021 Recuperar subasta UC-0022 Realizar puja UC-0023 Congelar *tokens* UC-0024 Descongelar *tokens* UC-0025 Finalizar subasta UC-0026 Registar envío UC-0027 Modificar envío UC-0028 Enviar notificación UC-0029 Escuchar eventos
Descripción	Toda funcionalidad del sistema deberá responder al usuario en menos de 7 segundos
Importancia	Importante
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 6. Descripción del requisito no funcional tiempo de respuesta

5.5.5 Actores del sistema

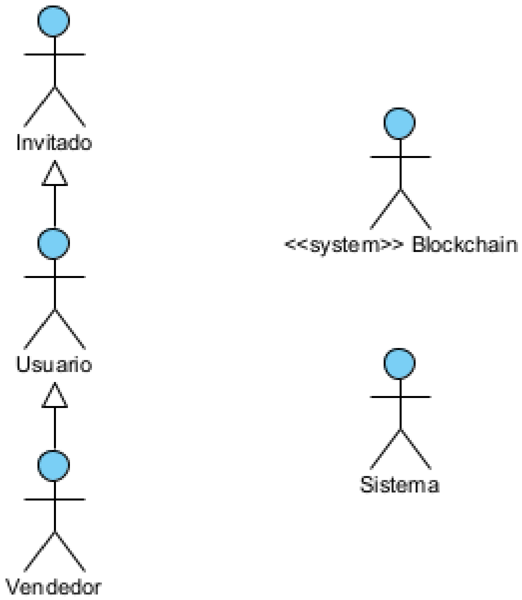

Figura 18. Jerarquía de actores

En la figura número 18 se puede observar la jerarquía de los actores del sistema. Por un lado, encontramos a los usuarios físicos de la aplicación, que los podemos dividir en tres tipos. Los invitados, que son aquellos usuarios que no han iniciado sesión en la aplicación. Luego están los usuarios, que son todos aquellos usuarios básicos que pueden realizar compras en la página y publicar subastas. Y finalmente los vendedores, que son aquellos usuarios que tienen alguna subasta en curso.

Por otro lado, nos encontramos al actor sistema, que será el encargado de ejecutar acciones automáticas dentro la aplicación, y el actor *blockchain* que representa a la API de *Shasta* utilizada para interactuar con la cadena de bloques.

Todos estos actores se describen con sus correspondientes tablas como se puede observar en el siguiente ejemplo.

ACT-02	**Usuario**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El actor representa a un usuario de la plataforma que está registrado y ha iniciado sesión

Comentarios	Ninguno

Tabla 7. Descripción del actor usuario

5.5.6 Requisitos funcionales o casos de uso

Los requisitos funcionales son aquellos que describen una funcionalidad especifica de la aplicación. Estos requisitos se definen mediante los casos de uso, estos casos de uso se mostrarán en un diagrama junto a los actores para modelar las interacciones entre ellos. Además de modelarse en el diagrama de casos de uso, también se realizará una especificación formal de los mismos.

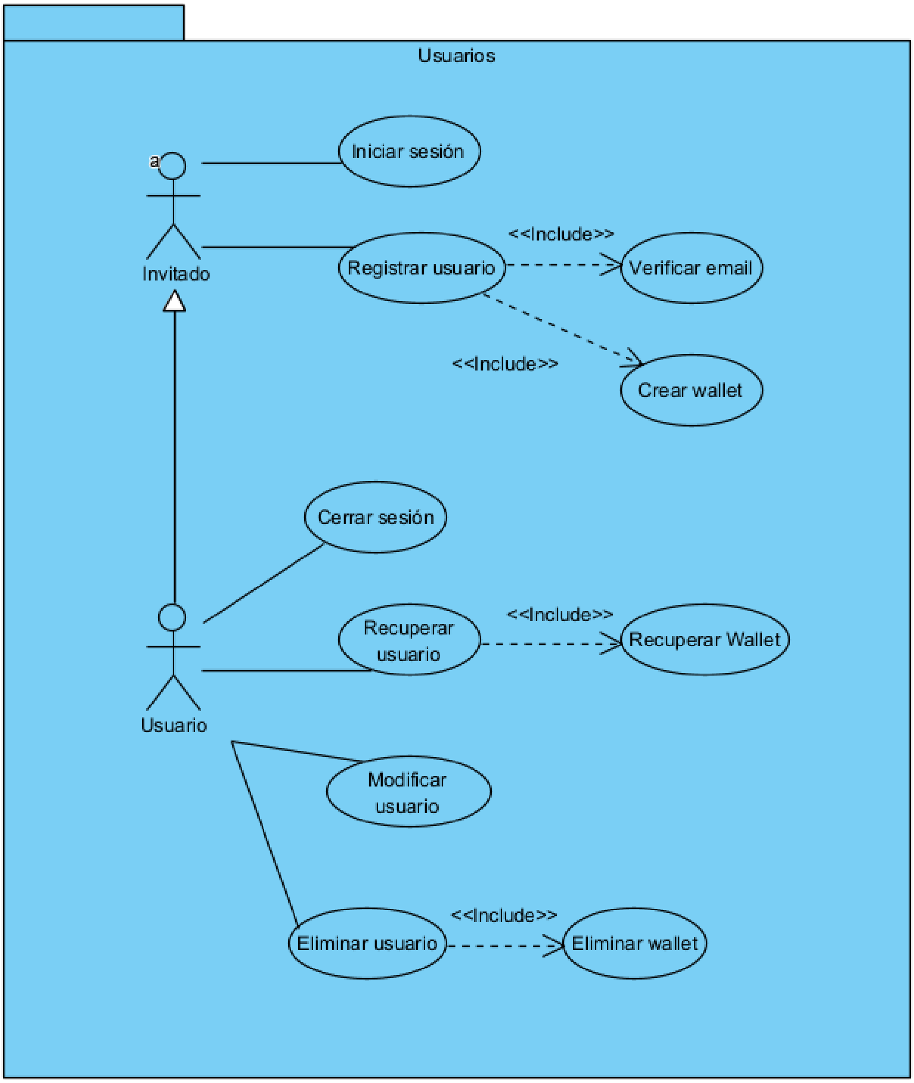

Figura 19. Diagrama de casos de uso del paquete Usuarios

UC-0001	Iniciar sesión	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0001 Gestionar usuarios	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios	
Descripción	El sistema debe permitir el inicio de sesión a los usuarios	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-01 (*Invitado*) selecciona la opción de inicio de sesión
	2	El sistema solicita la información de inicio de sesión, en particular el nombre de usuario y la contraseña
	3	El usuario introduce los datos correspondientes
	4	El sistema comprueba los datos de inicio de sesión
	5	El sistema crea una nueva sesión
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si el sistema detecta algún error en la validez de los datos volverá al paso 2, mostrando un error
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 8. Especificación formal del requisito funcional iniciar sesión

5.5.7 Matrices de trazabilidad

Las matrices de trazabilidad relacionan los casos de uso del sistema con los elementos de la especificación formal de este. Concretamente encontraremos las siguientes matrices:

- Matriz de trazabilidad de objetivos
- Matriz de trazabilidad de objetivos no funcionales
- Matriz de trazabilidad de requisitos de información

A continuación, se muestra el resultado de la matriz de trazabilidad de los requisitos de información.

	IRQ-0001	IRQ-0002	IRQ-0003	IRQ-0004	IRQ-0005	IRQ-0006
UC-0001	X	-	-	-	-	X
UC-0002	X	-	-	-	-	X
UC-0003	X	-	X	-	-	-
UC-0004	X	-	-	-	-	-
UC-0005	X	-	X	-	-	-
UC-0006	X	-	X	-	-	-
UC-0007	X	-	-	-	-	-
UC-0008	-	-	X	-	-	-
UC-0009	-	-	X	-	-	-
UC-0010	-	-	X	-	-	-
UC-0011	X	-	X	-	-	-
UC-0012	X	-	X	-	-	-
UC-0013	X	-	X	-	-	-
UC-0014	X	-	X	-	-	-
UC-0015	X	-	X	-	-	-
UC-0016	X	-	X	-	-	-
UC-0017	X	-	X	-	-	-
UC-0018	-	X	X	-	-	-
UC-0019	-	X	X	-	-	-
UC-0020	-	X	X	-	-	-
UC-0021	-	X	X	-	-	-
UC-0022	-	-	X	X	-	-
UC-0023	-	-	X	-	-	-
UC-0024	-	-	X	-	-	-
UC-0025	-	X	X	X	-	-
UC-0026	-	X	-	-	-	-
UC-0027	-	X	-	-	-	-
UC-0028	X	-	-	-	X	-
UC-0029	-	-	-	-	X	-

Tabla 9. Matriz de trazabilidad de los requisitos de información

5.6 Diseño arquitectónico

Mediante el diseño arquitectónico se define la estructura del sistema a implementar, así como su organización, enlazando el modelo de requisitos con el diseño final de la aplicación. En este apartado se identificarán los principales componentes que forman la estructura del proyecto y la relación entre ellos.

Para crear un diseño arquitectónico, primero se definirán los patrones arquitectónicos a utilizar. A continuación, se dividirá el sistema en paquetes y finalmente se realizará el diagrama de clases de diseño.

5.6.1 Patrones arquitectónicos

5.6.1.1 Modelo Vista Controlador

Este patrón ha sido definido en el apartado de técnicas del presente documento.

Como el principal objetivo de este proyecto es ofrecer a los usuarios una plataforma lo más sencilla e intuitiva posible, se ha elegido este patrón arquitectónico debido a propiedades orientadas a la interfaz de usuario, ayudando a realizar una aplicación web con una mayor flexibilidad de cara a futuras mejoras. Además, la estructuración del proyecto siguiendo los requisitos del MVC facilita la ordenación de los ficheros software de cara a una programación más intuitiva y sencilla.

5.6.1.2 Patrón DAO

Al igual que el patrón anterior, ya está definido en apartados anteriores del proyecto.

La principal característica buscada al utilizar este patrón es desvincular el acceso a los datos de la propia lógica de la aplicación. De modo que, si en etapas futuras del proyecto fuera necesario cambiar el tipo de almacenamiento, esta modificación se pueda realizar de manera sencilla.

5.6.2 División de paquetes

La división del sistema en paquetes se realiza mediante diagrama de paquetes. A partir de este diagrama es posible visualizar de forma gráfica cuántos paquetes compondrán la aplicación, qué estructura siguen y cómo se relacionan entre sí.

El presente desarrollo sigue una arquitectura de capas basada en el patrón modelo vista controlador, el cual aporta un cierto grado de modularización, dividiendo el código en función de su utilidad.

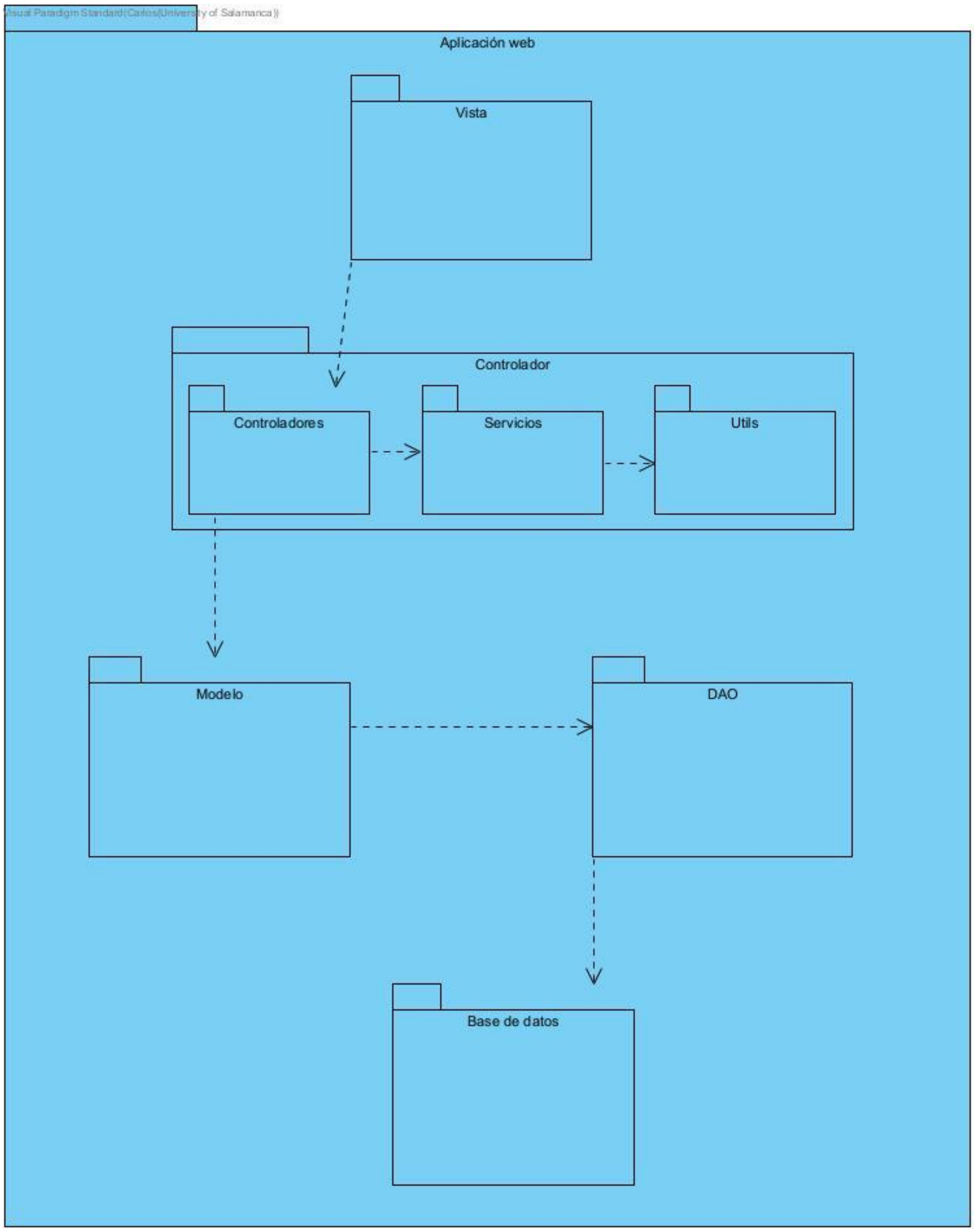

Figura 20. Diagrama de paquetes del sistema

En la figura número 20, se puede observar la peculiaridad de que el paquete controlador está dividido en 3 subpaquetes, de este modo se separa la lógica propia de la aplicación que se encuentra en el paquete controladores, de la lógica genérica la cual podrá ser reutilizada en otros proyectos. Gracias a esta peculiaridad se consigue una aplicación con un alto grado de reusabilidad.

5.6.3 Diagrama de clases del sistema

El diagrama de clases del sistema permite visualizar de un modo gráfico las relaciones existentes entre las distintas clases. A modo de ejemplo se presenta el diagrama de clases del paquete modelo.

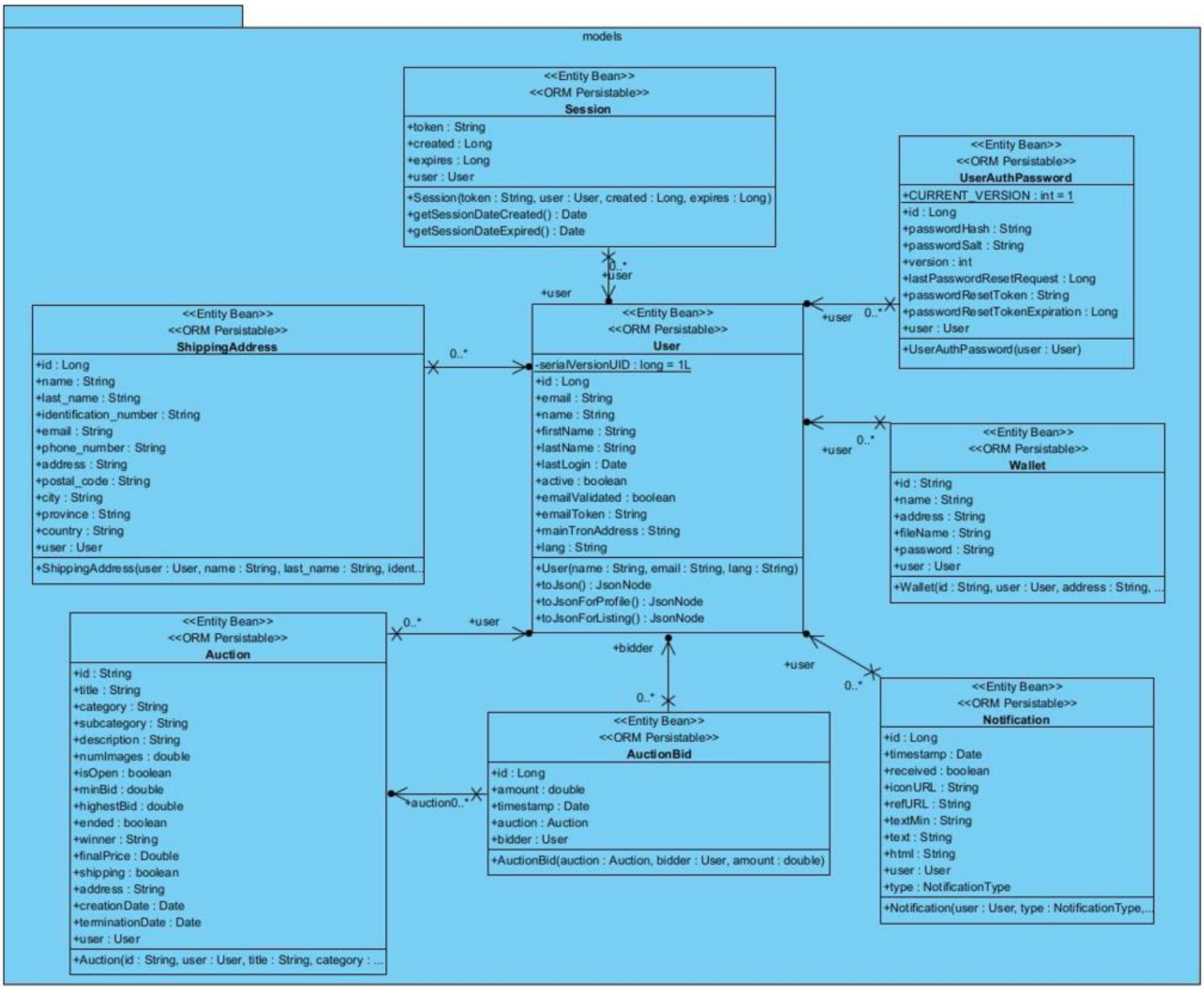

Figura 21. Diagrama de clases del paquete modelo

5.7 Diseño de datos

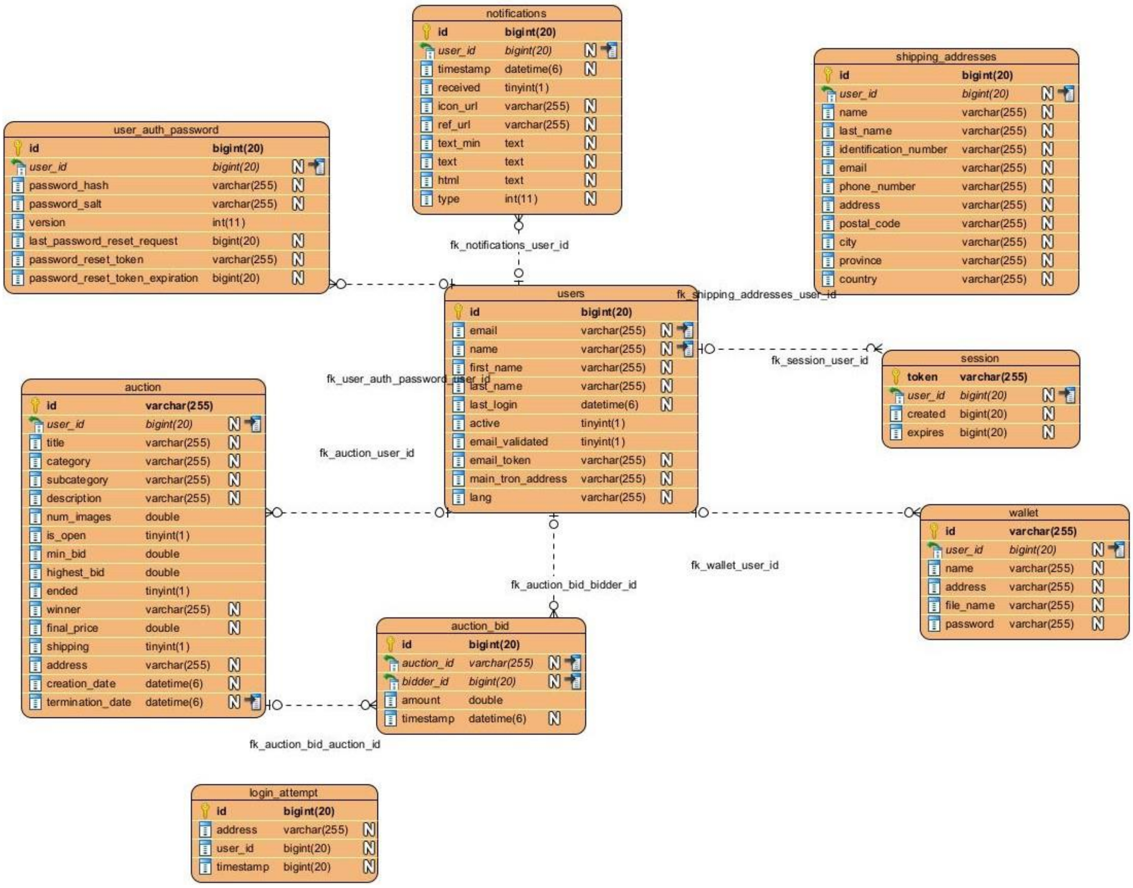

Figura 22. Diagrama entidad relación

La información necesaria para el correcto funcionamiento de la aplicación se almacenará en una base de datos MySQL. De cara a visualizar la estructura de los datos almacenados se utilizan los diagramas entidad relación. En los cuales se muestran las diferentes tablas de la base de datos, y las relaciones que existen entre ellas.

5.8 Vista de interacción

El principal objetivo de la vista de iteración es modelar el intercambio de información observable entre los elementos conectados.

Para mostrar este intercambio de información se utilizan los diagramas de secuencia, que muestran la iteración entre distintos objetos centrándose en los mensajes enviados.

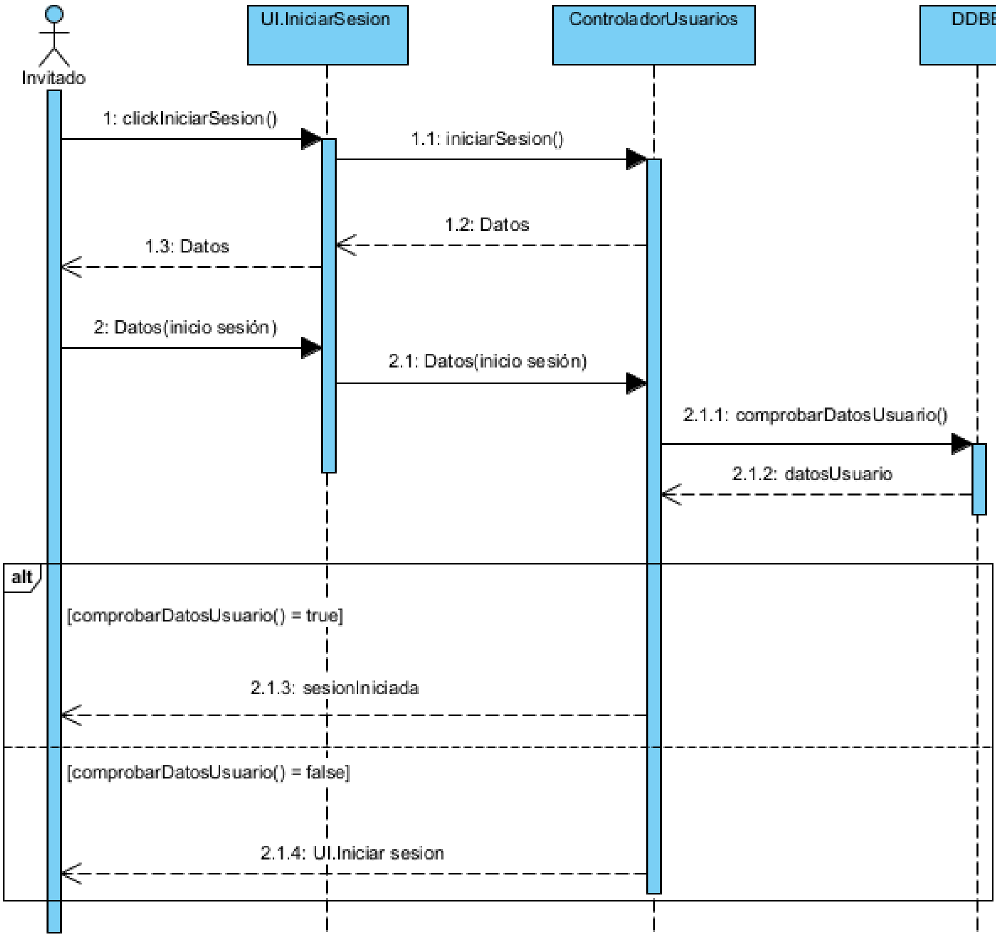

Figura 23. Diagrama de secuencia iniciar sesión

5.9 Desarrollo de los *smart contracts*

Para la lógica *blockchain* de este proyecto se ha implementado un *smart contract*, el cual será el encargado de gestionar las subastas y las pujas, así como de mantener el *token* propio de la plataforma.

Este *smart contract* lo podemos dividir en 4 partes, la primera se encargará del almacenamiento de los datos (atributos), la segunda será la definición de los eventos, en la tercera encontraremos las funciones modificadoras y finalmente en la cuarta los métodos que implementan la funcionalidad.

5.9.1 Atributos

En esta parte del *smart contract* se almacenarán los datos necesarios para su funcionamiento.

```
mapping (address => uint256) public balances; // user tokens

/**
 * Represents an auction.
 */
struct Auction {
    bool closed;
    bool ended;

    address owner; // The owner for the auction

    uint256 minBid;
    mapping (address => uint256) bids; // Bids made by the users
    address[] bidders; // User that bid in the auction
    address highestBid; // Hightest bid

    uint256 expiration; // Expiration
}

mapping (uint256 => Auction) public auctions; // Auctions mapping
```

Figura 24. Datos smart contract

En la figura número 24 se pueden observar los datos almacenados en el *smart contract*. En primer lugar, una tabla *hash* que relaciona la *wallet* de un usuario con el número de *tokens* que tiene. En segundo lugar, se define la estructura que debe tener una subasta, en este caso constará de dos valores lógicos (*booleans)* para saber si la subasta está activa y si la subasta es a sobre cerrado; también se almacenará la *wallet* del vendedor, la puja mínima, una lista con los pujadores y una tabla *hash* con sus pujas, finalmente se almacena la *wallet* del mayor postor y la fecha de finalización de la subasta. Por último, en tercer lugar, se crea una tabla *hash* que almacena cada subasta relacionándola con su id.

5.9.2 Eventos

```
/**
 * Event emitted when an auction is created.
 * @param _id The auction identifier
 * @param _owner The owner of the auction.
 * @param _min_bid The minimal amount in order to bid.
 * @param _expiration Expiration timestamp for the auction
 */
event LogCreateAuction(uint256 _id, address indexed _owner, uint256 _min_bid, uint256 _expiration);

/**
 * Event emitted when an user bids for an auction.
 * @param _id The auction identifier
 * @param _bidder Bidder address
 * @param _amount bid amount
 * @param _highest True if it is the highest bid
 */
event LogBid(uint256 _id, address indexed _bidder, uint256 _amount, bool _highest);

/**
 * Event emitted when an auction ends normally.
 * @param _id The auction identifier
 * @param _winner The winner of the auction.
 * @param _amount The amount paid
 */
event LogAuctionEnded(uint256 _id, address indexed _winner, uint256 _amount);
```

Figura 25. Eventos del smart contract

En la figura superior se puede observar un fragmento que contiene la definición de una serie de eventos. Estos eventos son:

- *LogCreateAuction*. Se lanzará cuando se cree una nueva subasta, los parámetros que lleva asociados son el id de la subasta, la dirección pública de la *wallet* del vendedor, la puja mínima y la fecha de finalización de la subasta.
- *LogBid*. Este evento se emitirá cuando se realice una nueva puja, los datos que llevará asociados son el id de subasta, la dirección pública del pujador, la puja realizada y si es la máxima puja.
- *LogAuctionEnded*. Este evento se lanzará al finalizar una subasta, en él se indicará el id de la subasta, la dirección del pujador ganador y la cantidad pujada.

5.9.3 Modificadores

Los modificadores en *Solidity* son unas funciones especiales que permiten modificar el comportamiento de los métodos. Su funcionamiento es el siguiente: en primer lugar, se deben definir los modificadores, para ello se le asignarán un identificador único, la condición que se debe cumplir para que el método que los utilice se ejecute y un mensaje que se mostrará en caso de que no se cumpla la condición especificada. En segundo lugar, una vez ya creado el modificador se utilizará en los métodos que se crea oportuno, para ello bastará con especificar su nombre después de los parámetros de la función.

```
/**
 * Checks if an auction is in progress.
 * @param _auction The auction identifier
 */
modifier auctionInProgress(uint256 _auction) {
    require(!auctions[_auction].ended, 'The auction already ended.');
    _;
}
```

Figura 26. Ejemplo modificador en el smart contract

En la Figura 26 se puede observar el modificador *auctionInProgress*. Este modificador recibe como parámetro el id de la subasta, y en su condición comprueba que la subasta no haya terminado.

5.9.4 Métodos

```
/**
 * Ends the auction (by the application)
 * The auction must exists and be in progress
 * @param _id The auction id
 */
function endAuction(uint256 _id) public onlyAdmin notPaused auctionInProgress(_id) {
    auctions[_id].ended = true;

    address owner = auctions[_id].owner;
    address winner = auctions[_id].highestBid;
    uint256 price = auctions[_id].bids[winner];

    // Transfer founds from winner to owner and the platform
    balances[owner] = balances[owner].add(price * 95 / 100);
    balances[root] = balances[root].add(price * 5 / 100);

    // Remove the bid amount for the winner
    auctions[_id].bids[winner] = 0;
    unfreezeTokens(_id);

    emit LogAuctionEnded(_id, winner, price);
}
```

Figura 27. Ejemplo de método del smart contract

En la figura anterior se puede visualizar el método encargado de finalizar las subastas. Este método recibe como parámetros de entrada el id de la subasta. Utiliza los siguientes modificadores:

- *onlyAdmin*. Restringe su llamada a únicamente el administrador/creador del contrato.
- *notPaused*. Exige que el contrato no haya sido pausado por el administrador.
- *auctionInProgress(_id)*. Comprueba que la subasta a finalizar esté en curso.

Una vez dentro del método, el primer paso es finalizar la subasta, a continuación, se transfieren de manera automática el 95% del importe de la puja al vendedor y el 5% restante se transfiere al creador de la plataforma de subasta. El siguiente paso es descongelar los fondos del resto de pujadores y emitir el evento de que la subasta ha finalizado.

5.9.5 Test unitarios

Las pruebas unitarias son un mecanismo utilizado para comprobar que un fragmento de código funciona correctamente. Se basan en aislar una parte del código para garantizar que no se van a producir fallos en ninguna ejecución.

```
Contract: TronAuction test
  √ El creador del contrato tiene el control del mismo (es root) (404ms)
  √ Otra cuenta no tiene el control del contrato (no es root) (386ms)
  √ El creador del contrato tiene permisos de administrador (354ms)
  √ Otra cuenta no tiene permisos de administrador (457ms)
  √ Dar permisos de administrador a la cuenta 2 (932ms)
  √ El creador del contrato mintea 100 TRA a la cuenta 1 (1357ms)
  √ El creador del contrato mintea 100 TRA a la cuenta 2 (892ms)
  √ El creador del contrato mintea 100 TRA a la cuenta 3 (1205ms)
  √ La cuenta 1 genera una nueva subasta (1113ms)
  √ La cuenta 2 puja 10 TRA en la subasta (1317ms)
  √ La cuenta 3 puja 15 TRA en la subasta (1779ms)
  √ La cuenta 2 intenta pujar 10 TRA y falla (2104ms)
  √ La cuenta 2 puja 20 TRA en la subasta (1368ms)
  √ Se finaliza la subasta (1018ms)
  √ La cuenta 2 ha pagado los 20 TRA (718ms)
  √ El creador del contrato ha recibido la comisión por la subasta (640ms)
  √ La cuenta 1 ha recibido el pago del articulo (506ms)
  √ Los fondos de la cuenta 3 se han descongelado (551ms)
  √ La subasta finalizo correctamente (378ms)
  √ Transferir 10 TRA de la cuenta 2 a la cuenta 3 (480ms)


20 passing (19s)
```

Figura 28. Pruebas unitarias del smart contract

Estas pruebas se le han realizado al *smart contract,* debido a la necesidad de que funcione correctamente en todas las circunstancias por llevar el peso de la aplicación relacionado con la validación económica.

5.10 Interfaz gráfica de usuario

Para la interfaz gráfica de usuario se ha utilizado una web, desarrollada con *Play Framework*. Este marco de trabajo incluye todos los componentes necesarios para la creación de aplicaciones web y de servicios REST. En el diseño de la interfaz se ha optado por una línea clara para que destaquen las imágenes de los productos y no haya grandes contrastes entre colores, primando la importancia que tiene la forma de presentar un producto en una subasta. Se ha buscado una interfaz lo más sencilla posible, para facilitar el uso a los usuarios. En cuanto a los colores escogidos, ha sido la paleta de rojos propia de la red de *Tron*, por ser la tecnología en la que se basa toda la aplicación.

Es principal objetivo de esta aplicación web es interactuar con los *smart contracts* desde cualquier dispositivo de manera gráfica y sencilla, sin que el usuario necesite grandes conocimientos técnicos en la materia.

La pantalla principal de la aplicación puede ser accedida tanto por personas registradas como por personas no registradas. En ella, los usuarios pueden ver las novedades de los patrocinadores, mediante un carrusel que va alternado en la parte central los últimos productos.

En la parte inferior aparecen las categorías más demandadas por los usuarios, y se puede pulsar sobre cualquiera de ellas para que se filtren los productos. En la esquina superior derecha se puede acceder al espacio personal del usuario. Todo esto se puede observar en la siguiente imagen.

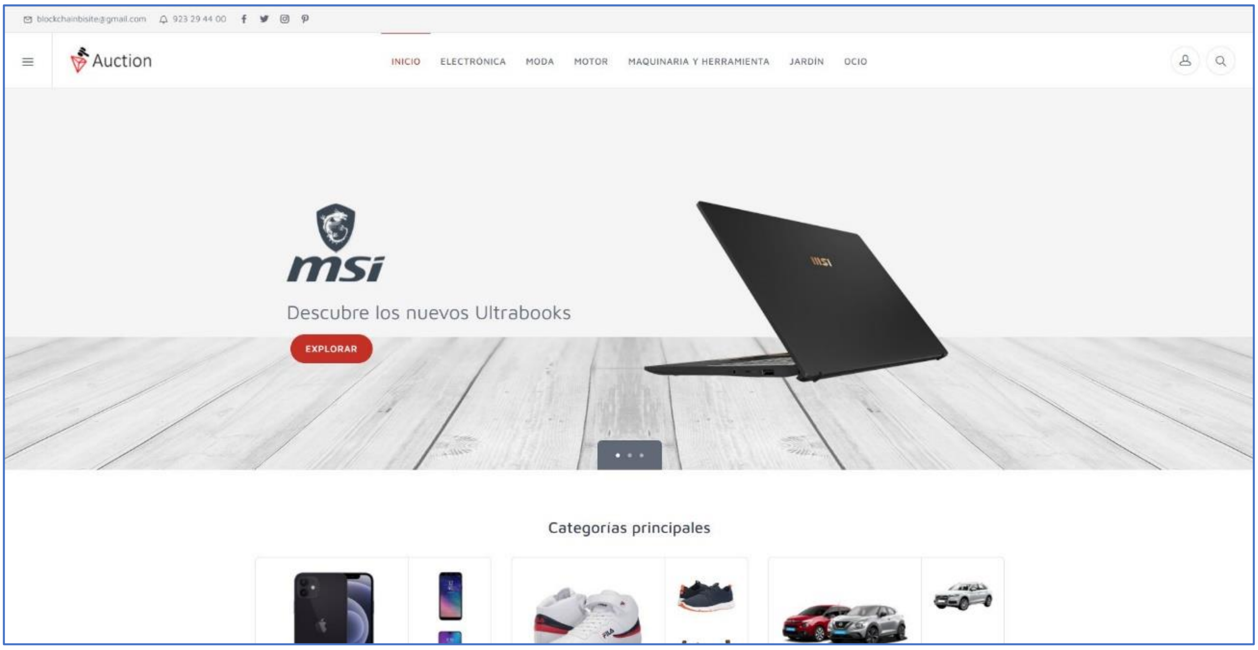

Figura 29. Vista principal de la interfaz gráfica

En la parte superior, se encuentran las distintas categorías de la aplicación, si se hace clic sobre alguna de ellas se mostrará un menú desplegable con las distintas subcategorías, este se puede observar a continuación.

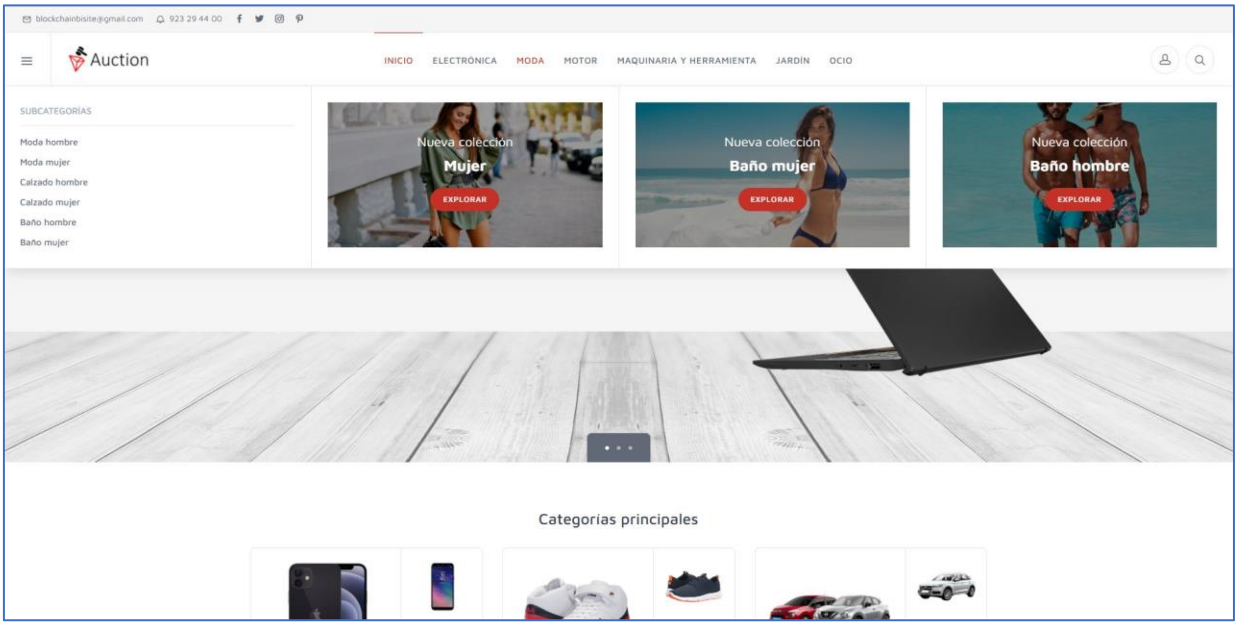

Figura 30. Subcategorías interfaz gráfica

Para poder acceder a todas las funcionalidades de la aplicación, los usuarios deben estar registrados. Para ello rellenarán los datos del formulario de registro. Una vez enviados estos, se recibirá un correo electrónico para su validación. Para terminar el registro, el usuario tendrá que pulsar sobre un enlace que vendrá insertado en el correo electrónico.

En la Figura 31 se recogen los formularios tanto para iniciar sesión, como para registrarse.

En caso de que no se reciba el correo de validación, se puede solicitar su reenvío en la pantalla que aparece después de registrar los datos del nuevo usuario.

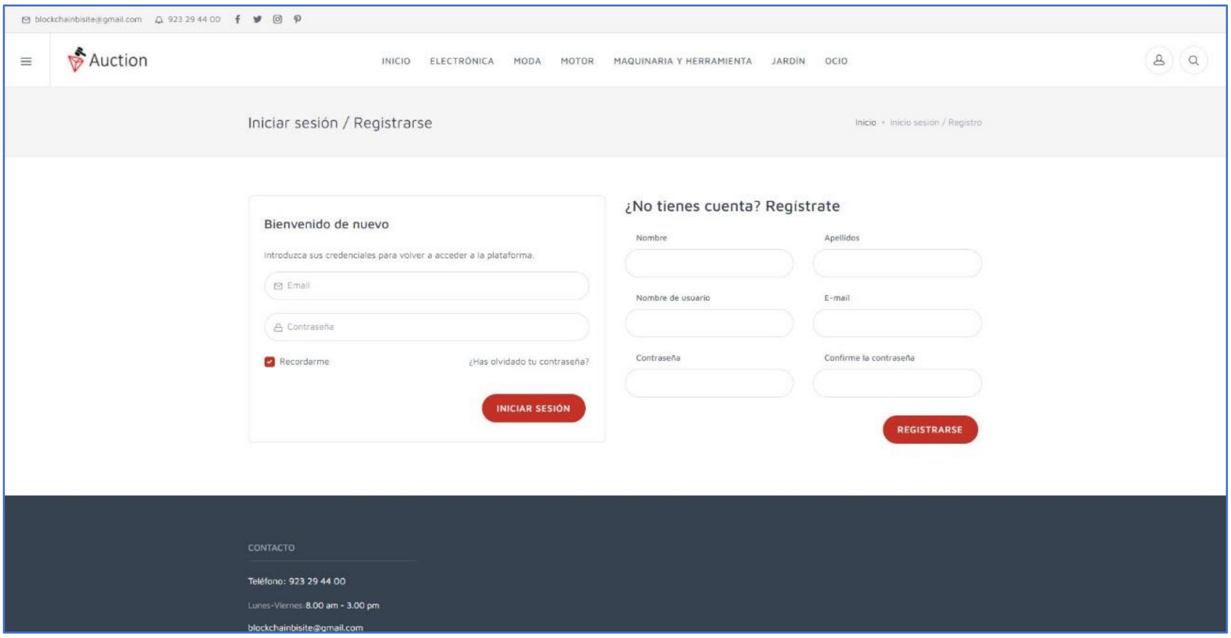

Figura 31. Formulario de registro/inicio de sesión de la interfaz gráfica

Dentro de la aplicación, cada usuario dispondrá de su propio espacio personal. En él podrá consultar sus datos personales, su cartera de criptomonedas, así como los datos relativos a las subastas y las pujas.

En la siguiente figura se puede apreciar el espacio personal de un usuario, con sus distintos subapartados. En este caso en concreto se ha elegido el subapartado “Mi perfil” que permite consultar los datos personales y poder cambiar la contraseña de acceso.

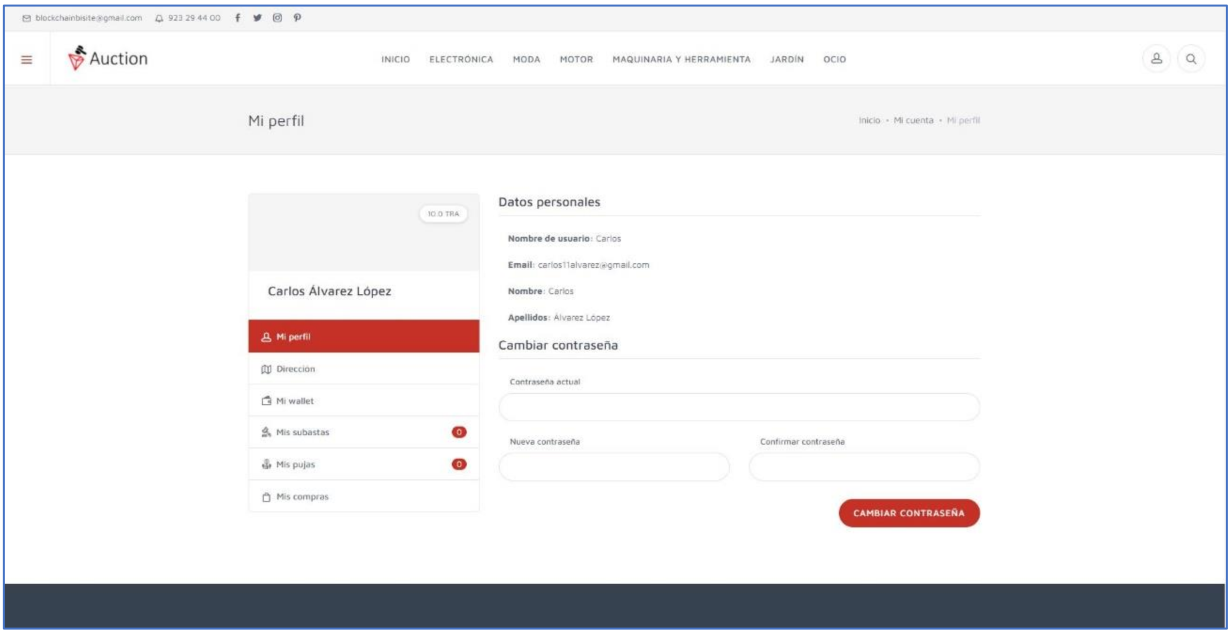

Figura 32. Apartado personal del usuario en la interfaz gráfica

Pinchando en cada uno de los subapartados de la columna de la izquierda irán apareciendo nuevas subpantallas para la gestión de los correspondientes datos. Por ejemplo, si pinchamos sobre el subapartado "Mi wallet", la nueva subpantalla mostrará la clave privada y la clave pública del usuario. Así como los métodos para comprar y vender los *tokens* de la aplicación. Véase la Figura 33.

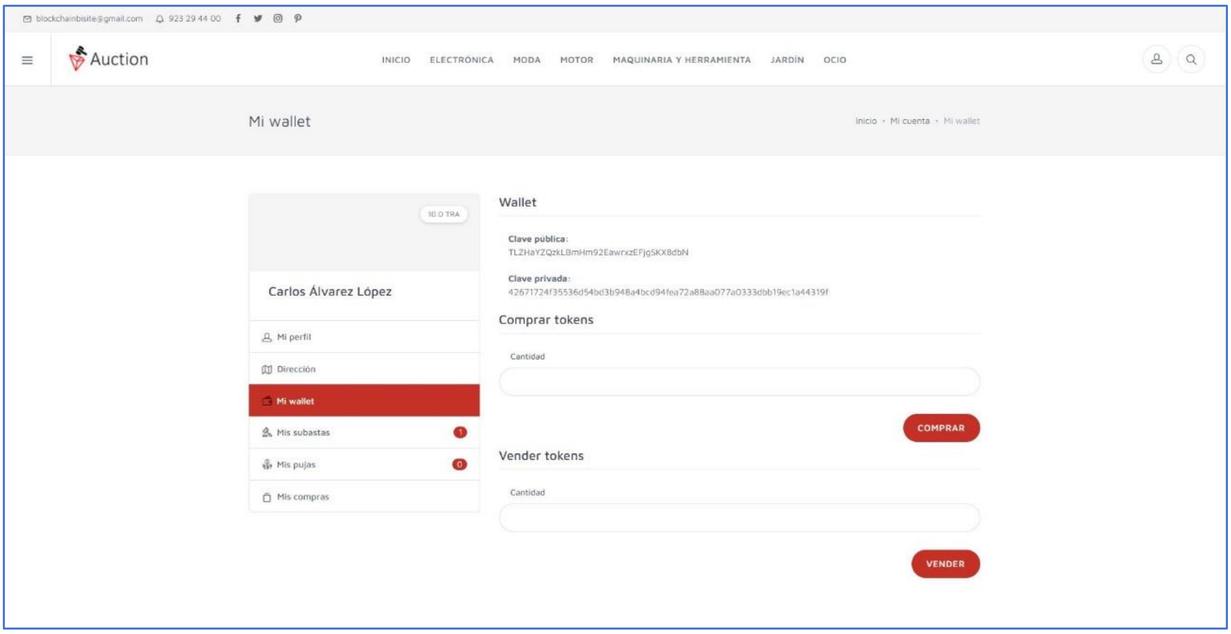

Figura 33. Vista de la wallet en la interfaz gráfica

Otra de los apartados centrales de la aplicación sería la gestión de las subastas, la cual se puede realizar desde el subapartado "Mis subastas", que da acceso a una lista con las subastas en curso y permite la creación de nuevas subastas, como se puede ver en la siguiente figura.

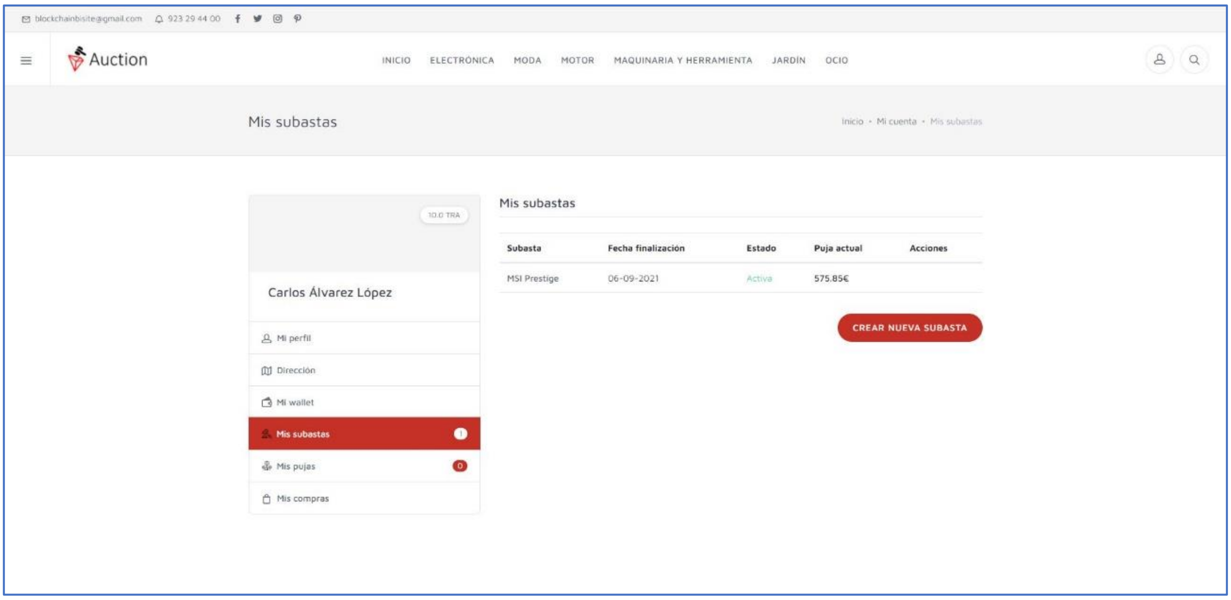

Figura 34. Gestión de las subastas en la interfaz gráfica

En las dos últimas opciones de menú, se puede encontrar un histórico tanto de las pujas realizadas por el usuario, como de las subastas ganadas.

Como se indicó anteriormente el usuario dispondrá de botones para crear nuevas subastas. En caso de que lo pulse aparecerá un formulario de recogida de datos con elementos desplegables desde los que se podrá introducir los datos de la nueva subasta. Se recoge en la figura siguiente.

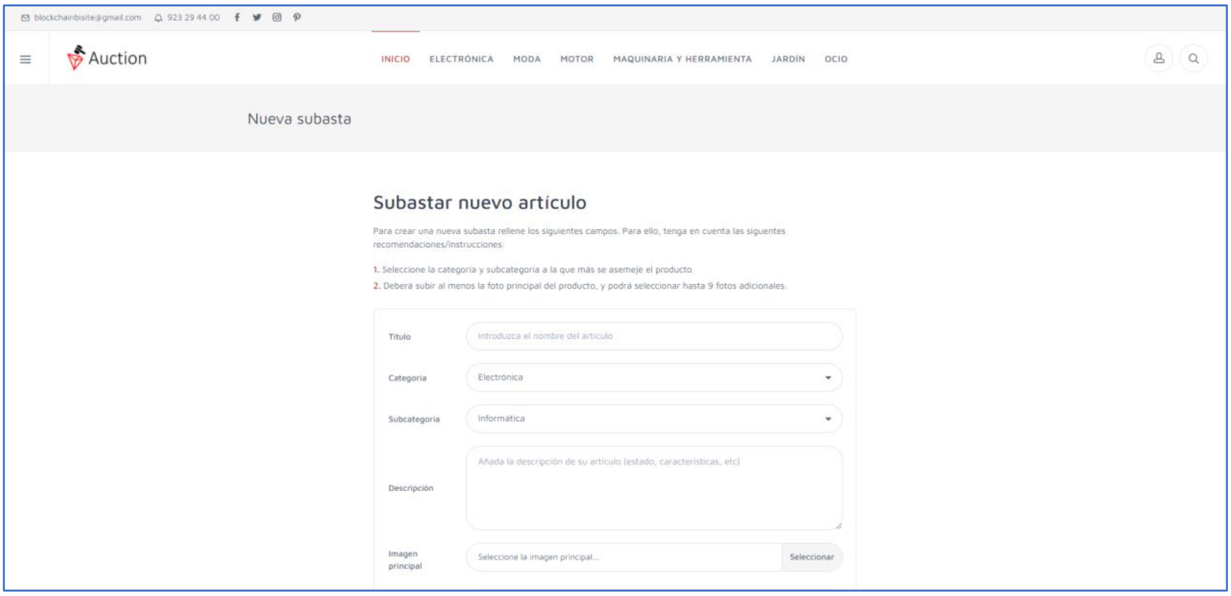

Figura 35. Crear nueva subasta en la interfaz gráfica

Otro aspecto destacado de la interfaz es el catálogo de subastas que muestra una matriz con los artículos que se están subastando, y al pulsar sobre cada uno de ellos muestra su vista detallada.

Para que sea más fácil la navegación cuando haya muchos productos en la parte izquierda aparece una lista con las categorías para poder filtrar los artículos. Un ejemplo aparece en la Figura 36.

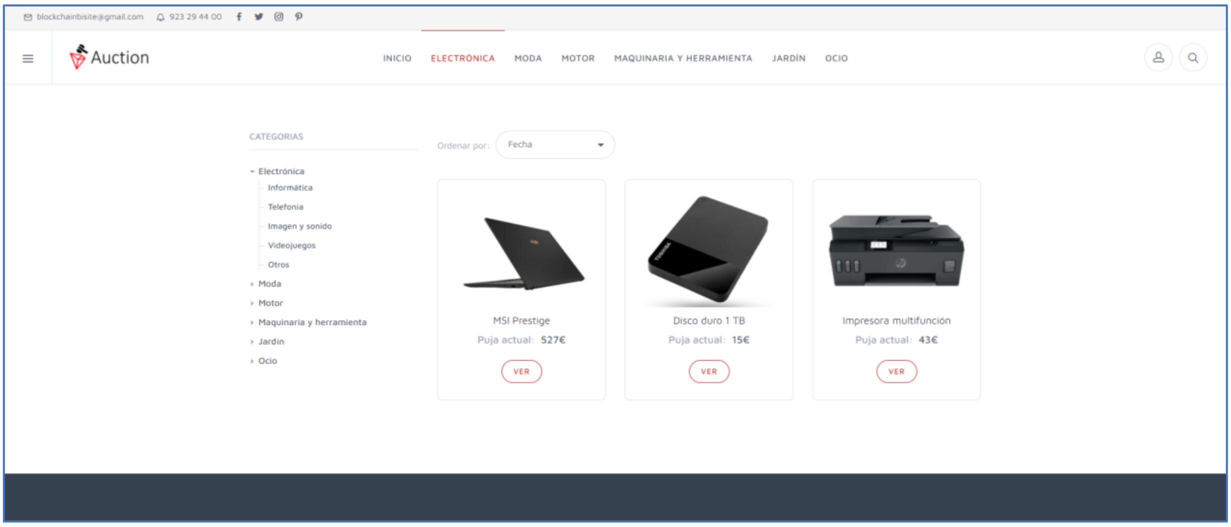

Figura 36. Catálogo de productos subastados en la interfaz gráfica

Cuando un usuario accede a la descripción de una subasta, podrá ver los datos del artículo, la puja actual, la fecha de finalización y si lo desea podrá hacer una puja por un importe superior al de la puja actual. Esto se puede observar en la figura posterior.

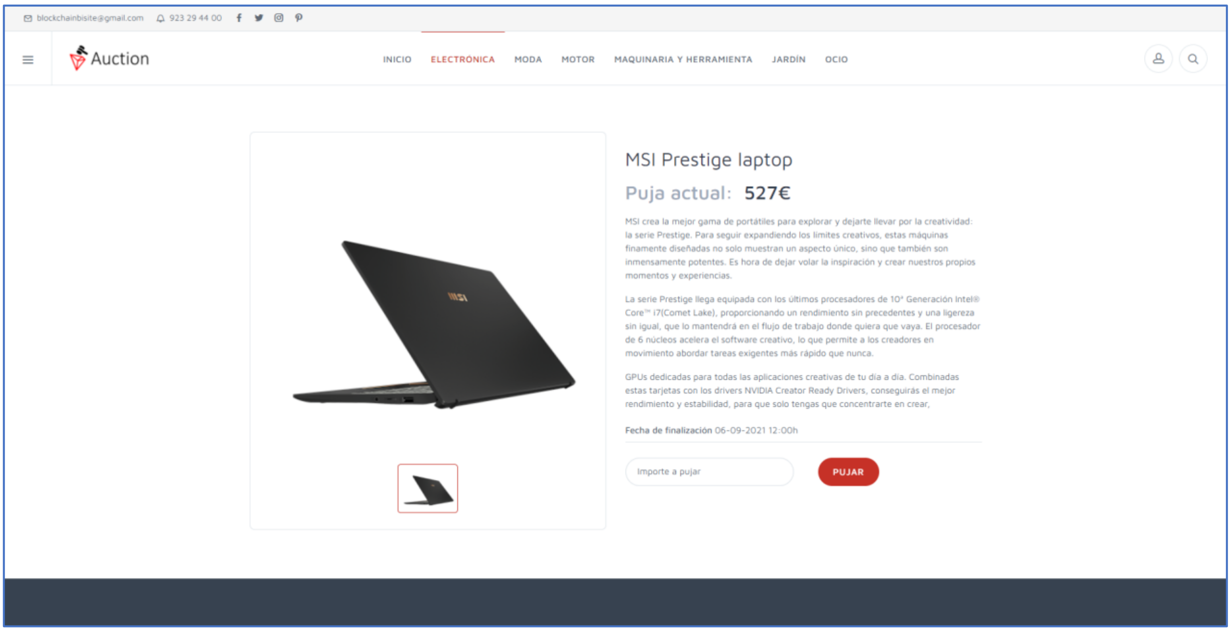

Figura 37. Detalle de una subasta en la interfaz gráfica

La ventaja que presenta una aplicación web es que está accesible desde cualquier navegador independientemente del sistema operativo, además al haberse creado una Web *Responsive,* el contenido se adaptará a cualquier tamaño de pantalla, ya bien sea un *smartphone,* una *tablet,* o un ordenador de cualquier tipo.

Véase por ejemplo el aspecto que tendría para un dispositivo Android con una relación de aspecto de pantalla de 18:9, como se recoge en la siguiente imagen.

Figura 38. Vista de la aplicación web desde un móvil Android

6 Conclusiones

El desarrollo de un proyecto completo es la mejor forma de tomar contacto con la realidad del mundo empresarial, sirve para darse cuenta de la envergadura que puede tener el desarrollo global de una aplicación informática y de la necesidad del trabajo en equipo. Se ha constatado, que, aunque el proyecto puede ser desarrollado por una única persona, no va a ser la forma habitual de llevarlo a cabo en un puesto de trabajo, porque los tiempos de desarrollo se alargan mucho y la búsqueda de soluciones siempre es más compleja.

Un aspecto para destacar al abordar un proyecto completo es que se tiene una visión de conjunto de todas las fases y técnicas necesarias y sirve para poner en práctica muchos de los conceptos y herramientas vistos durante la carrera, que en algunas ocasiones se han visto en ejemplos más reducidos, o en casos muy delimitados, pero sin integrarse y relacionarse unos elementos con otros.

Otro aspecto que destacar es la necesidad de la investigación como parte esencial del trabajo de los desarrolladores, ya sea investigación de técnicas, *frameworks*, tecnologías, entornos, etc. Se ha constatado que para poder buscar soluciones a los requisitos que se nos planteen siempre va a ser necesario tener que estudiar las últimas tendencias, al igual que para poder solventar las dificultades que nos vayan surgiendo. Esta búsqueda de soluciones, que siempre será necesaria, hace que la necesidad de formación sea una constante en nuestro futuro profesional.

El haber tenido que enfrentarme a técnicas muy novedosas y de actualidad como puede ser el mundo de las criptomonedas y el de *blockchain*, que pueden haberse tocado de forma transversal en alguna asignatura de la carreta, pero que no tienen un tratamiento específico en ninguna asignatura en concreto, ha servido para completar mi formación y para que a la hora de acceder al mundo laboral la experiencia adquirida pueda servir para tener continuidad en la empresa.

El tema central de este proyecto es el uso de criptomonedas para el pago en subastas. Como se ha demostrado a lo largo del desarrollo, las ventajas que aporta son muchas, pero sin embargo las perspectivas de futuro que puede tener el pago, ya no solo en subastas sino en tiendas virtuales, en servicios a domicilio, o incluso en tiendas físicas, son inciertas. Se están escribiendo multitud de artículos sobre el futuro de las criptomonedas y las posiciones son de lo más dispares, desde los que dicen que es un mercado totalmente efímero que colapsará, hasta los que afirman que será el dinero del futuro que sustituirá al dinero físico tal como lo conocemos. En medio de estos extremos se entrecruzan multitud de intereses de todo tipo, especulativos, políticos, recaudatorios, etc., por lo que saber si este tipo de pago se implantará es difícil, las dudas que se plantean son muchas, si será cuestión de tiempo, si habrá que esperar a que los gobiernos lo regulen, o cuando menos que lo admitan y no lo sancionen.

Respecto a la utilización de *blockchain*, parece una técnica totalmente contrastada, y lo que es muy importante, muy segura (no infalible), que sin lugar a duda tiene muchas posibilidades y que cada vez se va a utilizar más es aspectos muy variados de la vida diaria. Ya se ha demostrado su utilidad para el seguimiento de la trazabilidad de productos alimentarios por parte de grandes cadenas de alimentación, para empresas cárnicas, en aplicaciones de ciberseguridad, en votaciones electrónicas, en sistemas de puntos de fidelización, para la gestión de contratos de trabajo, etc. Por lo tanto, se trata de una herramienta con una gran perspectiva de futuro, que va a seguir adelante, pero que va a poder hacer uso en mayor o menor medida de las criptomonedas según sea la evolución de estas últimas.

7 Líneas de trabajo futuras

Una aplicación informática siempre es susceptible de ser mejorada, y en el caso de las aplicaciones web es muy habitual realizar cambios para mejorar su aspecto. En este sentido el aspecto visual podría y debería adaptarse a la identidad corporativa de la empresa que la vaya a utilizar. Relacionado con este aspecto también podría incorporarse mecanismos para hacer accesible la aplicación a personas con discapacidad.

En la actualidad la aplicación está utilizando una red de pruebas (*testnet*) que tiene las mismas características funcionales que la red principal de Tron *(mainnet)*. La principal diferencia entre estas dos redes es el número de nodos (muy inferior en la red de pruebas), por lo que las posibilidades de alterar un registro de la cadena son mayores, ya que es más fácil conseguir el control de la mitad de la red. De cara a una implementación comercial, sería interesante el uso de la red principal a pesar de las comisiones económicas cobradas por cada transacción realizada, que serían asumibles a cambio de una mayor confianza.

Además de utilizar la red principal de Tron, se podrían explorar otras alternativas confiables, pero con un tiempo entre bloques menor. Como, por ejemplo, la cadena de bloques española Alastria, desarrollada por un convenio entre grandes empresas de diversos sectores entre las que destacan el banco Santander, Acciona, el grupo AXA, Caixa Bank, BBVA, o CEPSA. La principal ventaja de esta cadena es su tiempo entre bloques, de solamente un segundo, dando lugar a transacciones mucho más rápidas, y por lo tanto a una aplicación con mayor usabilidad. Como aspecto a tener en cuenta, habría que considerar que Alastria se trata de una *blockchain* privada, pudiendo participar única y exclusivamente sus socios. A pesar de ello, no se vería comprometida la integridad de esta, ya que el número de socios y por lo tanto de nodos es notable. Por otro lado, a pesar de ser una cadena de bloques privada sus transacciones son públicas, garantizando la transparencia propia de esta tecnología.

Si fuera necesario mejora la seguridad de la aplicación, podría implementarse un sistema de validación de usuarios basado en la autentificación de varios factores, al menos dos, es decir añadir uno más, es decir, además de pedir usuario y contraseña, se solicita un segundo factor, que puede ser, por ejemplo, un código pin, o una cadena alfanumérica generada y enviada a un móvil registrado para el usuario. Si aún se quisiera más seguridad se podrían añadir más factores, como algún elemento gráfico a completar para evitar el ataque por fuerza bruta que suelen utilizar los *bots*.

Otro de los aspectos sobre el que se podría trabajar es el tema de la utilización de las redes sociales. Si bien se podría implementar el acceso mediante perfiles de redes, habría que cuestionarlo mucho por el tema de la vulnerabilidad y la facilidad de los ataques por suplantación de identidad. Lo que sí se podría utilizar es el carácter publicitario y divulgativo que pueden aportar las redes sociales para dar a conocer las subastas que se vayan a producir en la aplicación, para ello se podrían incorporar botones para poder difundir en las principales redes.

Finalmente, un aspecto a considerar sería el desarrollo de una aplicación para móviles, por varios aspectos, primero debido a la gran difusión que tienen estos dispositivos actualmente y a las buenas capacidades de cómputo que aportan. Y en segundo lugar por la gran aceptación que tienen en el sector financiero debido a su gran portabilidad y la capacidad de uso en cualquier escenario. Hay que tener en cuenta que la hora de finalización de una subasta puede darse en cualquier momento, y no siempre los pujadores dispondrán de un ordenador a mano. A pesar de disponer la aplicación web de una vista especial para dispositivos móviles, siempre es mucho más intuitivo y rápido el uso de una aplicación nativa.

8 Bibliografía

Ambler, S. W. (2005). *The Agile Unified Process (AUP)*. Obetenido del sitio oficial de AUP: http://www.ambysoft.com/unifiedprocess/agileUP.html

Beck, K. (1999). *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional.

Bit2Me Academy. (2021). sitio oficial de Bit2Me Academy. Obtenido del sitio oficial de Bit2Me Academy: https://academy.bit2me.com/

Developers Tron Network. (2021). *The TRON Developer Hub*. Obtenido del sitio oficial de Developers Tron Network: https://developers.tron.network/

Durán Toro, A., & Bernárdez Jiménez, B. (2002). *Metodología para la Elicitación de Requisitos de Sistemas Software (versión 2.3.* Sevilla: Informe Técnico LSI-2000-10, Universidad de Sevilla.

García-Peñalvo, F. J., Moreno García, M. N., García-Holgado, A. y Vázquez-Ingelmo, A. "UML. Unified Modeling Language," Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2019-2020, F. J. García-Peñalvo y A. Vázquez-Ingelmo, Eds., Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2020. [Online]. Disponible en: https://bit.ly/2HY6TOO. doi: 10.5281/zenodo.3688621.

García-Peñalvo, F. J. y Vázquez-Ingelmo, A., Recursos docentes de la asignatura Ingeniería de Software I. Grado en Ingeniería Informática. Curso 2018-2019, Salamanca, España: Grupo GRIAL, Universidad de Salamanca, 2019. [Online]. Disponible en: https://goo.gl/ZNf7SH. doi: 10.5281/zenodo.2596446.

Johnson, J. (1995). Chaos: The Dollar Drain of IT Project Failures. *Application Development Trends 2(1)*, 41-47.

Moreno, M. N. (2020). Apuntes de la asignatura gestión de proyectos.

Moreno, M. N. (2020). Apuntes de la asignatura Ingeniería del Software II.

Pressman, R. S. (2002). *Ingeniería del Software: Un Enfoque Práctico.* McGraw-Hill.

Play framework. (2021). *The High Velocity Web Framework For Java and Scala*. Obtenido del sitio oficial de Play framework: https://www.playframework.com/

TronGrid. (2021). *Easy access to the Tron network*. Obtenido del sitio oficial de TronGrid: https://www.trongrid.io/

Truffle. (2021). *Sweet tools for smart contracts*. Obtenido del sitio oficial de Truffle: https://www.trufflesuite.com/

TronIDE. (2021). Sitio oficial de *TronIDE*. Obtenido del sitio oficial de TronIDE: http://www.tronide.io/

TronLink. (2021). *TronLink Wallet. Link TRON Ecosystem*. Obtenido del sitio oficial de TronLink: https://www.tronlink.org/

TronScan. (2021). *Tron BlockChain Explorer.* Obtenido del sitio oficial de TronScan: https://tronscan.org/#/

TronAuction: plataforma de subastas online utilizando *Smart Contracts*

Anexo I: Plan del proyecto *software*

Grado en ingeniería informática

Septiembre de 2021

Autor

Carlos Álvarez López

Tutores

Juan Manuel Corchado Rodríguez

Diego Javier Valdeolmillos Villaverde

Agustín San Román Guzmán

Tabla de contenido

Índice de figuras

Índice de tablas

1. Introducción

En el presente documento se presenta el ciclo de vida y la planificación temporal diseñada para el proyecto *TronAuction*: plataforma de subastas online utilizando *smart contracts*.

En la primera parte del documento se encuentra la descripción del modelo de ciclo de vida junto con los factores de complejidad técnica y del entorno. En la segunda parte, una vez escogido el ciclo de vida idóneo para el proyecto, se expondrá la planificación temporal. En ella se divide el proyecto en tareas, asignándole a cada una un periodo de tiempo y una serie de recursos.

2. Modelo de ciclo de vida

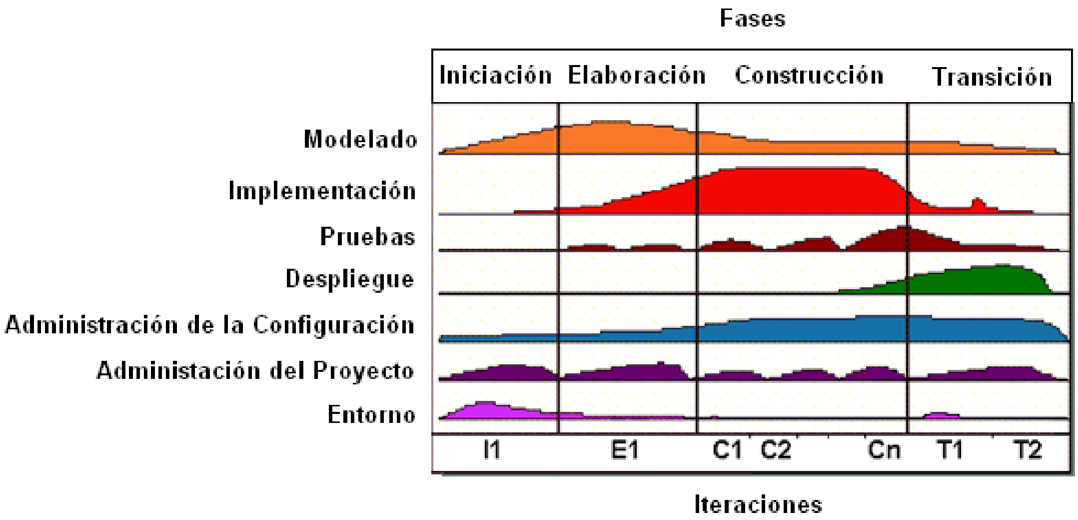

Figura 1: Ciclo de vida del proceso unificado

Para el desarrollo del presente proyecto se ha seguido un ciclo de vida basado en el proceso unificado. Por lo tanto, se tratará de un ciclo de vida iterativo e incremental. El cual estará compuesto por las siguientes cuatro fases:

- **Iniciación**. Su objetivo es identificar el alcance inicial del proyecto. Entre las actividades que se realizan están la estimación de costes y plazos, definición de los requisitos, estimación de recursos y el cronograma inicial de fases. El artefacto que se obtiene es el Documento de definición del proyecto.

- **Elaboración**. Su objetivo es que el equipo de desarrollo profundice en los requisitos del sistema y en validar la arquitectura. Los artefactos que se obtendrían son el Modelo del dominio, el Modelo de Procesos, el Modelo funcional de alto nivel y la Arquitectura básica.

- **Construcción**. Su objetivo es desarrollar el software, de forma incremental, basado en las prioridades de los participantes. Las actividades comienzan con el diseño, para pasar a la implementación y a las pruebas.

- **Transición**. Se despliega el sistema en el entorno real de producción.

A lo largo de las cuatro fases se desarrollan actividades relativas a siete disciplinas (filas). Como se puede observar en la representación del modelo, estas disciplinas no tienen por qué aplicarse en todas las fases, así por ejemplo la disciplina de las Pruebas sólo se aplica en las fases de Construcción y Transición, y en cada fase se pueden aplicar con un peso distinto, por ejemplo, la disciplina de Implementación tiene mucha más importancia en la fase de Implementación que en las dos anteriores. Las siete disciplinas son:

1. **Modelado**. Se trata de plasmar la lógica de negocio de la aplicación, determinar el dominio del problema y buscar una solución viable.
2. **Implementación**. Transformar los modelos desarrollados en código ejecutable que se pueda probar mediante pruebas básicas unitarias.

3. **Pruebas**. Realizar una serie de pruebas que verifiquen que se cumplen los objetivos propuestos y que se verifican todos los requisitos demandados.
4. **Despliegue**. Desarrollar un plan para poner a disposición del cliente el sistema para que los usuarios lo puedan utilizar.
5. **Gestión de la configuración**. Gestionar los artefactos que se han ido produciendo, así como las versiones de la aplicación y sus cambios.
6. **Gestión del proyecto**. Gestionar los elementos internos del proyecto, como los riesgos, el personal asignado, los tiempos de entrega, el progreso y la asignación de las tareas.
7. **Entorno**. Proporcionar los elementos necesarios para el desarrollo, desde orientaciones de tipo legislativo, hasta material, pasando por requisitos hardware y software., de modo que los módulos y artefactos del proyecto se han construido de manera gradual. Iteración tras iteración.

Todas estas fases se realizarán de manera iterativa en una o más iteraciones. Obteniendo un producto final evaluable en cada una de ellas.

3. Estimación del esfuerzo

A lo largo de este apartado se estimarán los costes temporales y el esfuerzo de desarrollo del proyecto según la funcionalidad requerida por este. Para ello se seguirá el método de estimación de puntos de casos (UCP – *Use Case Points)*, propuesto por Gustav Karner en el año 1998. Para realizar la estimación utiliza el cálculo de tres variables:

- Puntos de caso de uso sin ajustar (UUCP – *Unadjusted Use Case Points*)
    - Complejidad de los casos de uso (UUCW – *Unadjusted Use Case Weigth)*
    - Complejidad de los actores (UAW – *Unadjusted Actor Weight)*
- Factores de complejidad técnica (TCF – *Technical Complexity Factor*)
- Factores de complejidad del entorno (ECF – Enviroment Complexity Factor)

Para llegar a la estimación final de los puntos de caso de uso se utilizará la siguiente fórmula: UCP = UUCP * TCF * ECF

3.1 Puntos de caso de uso sin ajustar

3.1.1 Complejidad de los casos de uso (UUCW)

Para calcular la complejidad de los casos de uso, se considerará el número de pasos de los escenarios, la complejidad de la interfaz de usuario, el número de entidades de la base de datos a las que accede, y el número de clases implicadas. El baremo será el siguiente:

- **Simple** (factor = 5): Caso de uso con 3 o menos pasos.
- **Medio** (factor = 10): Caso de uso con entre 4 y 7 paso.
- **Complejo** (factor = 15): Caso de uso con más de 7 pasos.

ID	Nombre	Complejidad
UC-0001	Iniciar sesión	Media
UC-0002	Cerrar sesión	Simple
UC-0003	Registrar usuario	Media
UC-0004	Modificar usuario	Media
UC-0005	Eliminar Usuario	Media
UC-0006	Recuperar usuario	Media
UC-0007	Verificar email	Simple
UC-0008	Crear *wallet*	Simple
UC-0009	Eliminar *wallet*	Simple

UC-0010	Recuperar *wallet*	Simple
UC-0011	Realizar transacción *blockchain*	Simple
UC-0012	Firmar transacción	Simple
UC-0013	Enviar transacción	Media
UC-0014	Transferir *tokens*	Media
UC-0015	Consultar *tokens*	Media
UC-0016	Comprar *tokens*	Media
UC-0017	Vender *tokens*	Media
UC-0018	Crear subasta	Media
UC-0019	Modificar subasta	Media
UC-0020	Eliminar subasta	Media
UC-0021	Recuperar subasta	Media
UC-0022	Realizar puja	Media
UC-0023	Congelar *tokens*	Simple
UC-0024	Descongelar *tokens*	Simple
UC-0025	Finalizar subasta	Media
UC-0026	Registrar envío	Media
UC-0027	Modificar envío	Media
UC-0028	Enviar notificación	Simple
UC-0029	Escuchar eventos	Media

Tabla 1: Complejidad de los casos de uso

CÁLCULO DEL UUCW

Una vez evaluados los factores de complejidad de los casos de uso, simplemente se sumará el factor de cada uno en función de su complejidad:

$$\mathrm{UUCW} = UC_{simples} * 5 + UC_{medios} * 10 + UC_{complejos} * 15 = 10 * 5 + 19 * 10 + 0 * 15 = 240$$

3.1.2 Complejidad de los actores (UAW)

Se considera el número y complejidad de los actores. Para ello se sigue el siguiente baremo:

- **Simple** (factor 1). El actor es un sistema y la aplicación se comunica con él mediante la invocación de métodos.
- **Medio** (factor 3). El actor es un sistema y la aplicación se comunica con él mediante un protocolo de red.
- **Complejo** (factor 3). Persona con una interfaz gráfica.

ID	Nombre	Complejidad
ACT-01	Invitado	Compleja
ACT-02	Usuario	Compleja
ACT-03	Vendedor	Compleja
ACT-04	*<<System>> Blockchain*	Media
ACT-05	Sistema	Simple

Tabla 2: Complejidad de los actores

CÁLCULO DEL UAW

Una vez evaluados los factores de complejidad de actores o, simplemente se sumará el factor de cada uno en función de su complejidad:

$$\text{UAW} = ACT_{simples} * 5 + ACT_{medios} * 10 + ACT_{complejos} * 15 = 1 * 1 + 1 * 2 + 3 * 3 = 12$$

3.1.3 Cálculo de los puntos de caso de uso sin ajustar (UUCP)

Una vez calculada la complejidad de los casos de uso y de los actores, se procede a calcular los puntos de caso de uso sin ajustar:

$$UUCP = UAW + UUCW = 240 + 12 = 252$$

3.2 Factores de complejidad técnica

Para calcular los puntos de los factores de complejidad técnica, se definen trece factores a evaluar. A cada uno de esos factores se le asignará un peso (W) acorde a su impacto y una complejidad percibida por el equipo de desarrollo (F)

- **Sistemas distribuidos**: 3

  El sistema utiliza la tecnología blockchain, por lo tanto, estará prepara para funcionar con una red distribuida y descentralizada de nodos.

- **Rendimiento**: 5

  El sistema estará utilizado por personas sin grandes conocimientos en informática, las cuales pueden seguir inseguridades en el proceso si se demora demasiado. Por lo tanto, una página que tarda en cargar o una transacción que se demore unos minutos es inadmisible, por lo tanto, el rendimiento es un punto clave. Es por ello por lo que las operaciones deben de realizarse en el menor periodo de tiempo posible.

- **Eficiencia del usuario final:** 1

  Se trata de una aplicación web que podrá ser accedida por cualquier tipo de personas, que no incurrirá en ningún tipo de aspecto técnico, de modo que cualquier persona con unos mínimos conocimientos técnicos podrá hacer uso de ella.

- **Procesamiento interno complejo:** 2

  Se trata de un sistema que no debe de utilizar grandes cantidades de datos, pero si que debe de almacenar la información relativa a todas las subastas activas, así como la información de todos los usuarios y sus *wallets*.

- **Reusabilidad**: 2

  Se pretende que parte de los módulos desarrollados en el presente proyecto puedan ser reutilizados para futuras implementaciones. En especial la parte relacionada con la tecnología blockchain.

- **Facilidad de instalación:** 3

  Para la instalación del proyecto deben de instalarse diversas herramientas, con sus respectivas configuraciones. Las cuales están en continuo desarrollo al ser *blockchain* una tecnología emergente, y por lo tanto su instalación puede variar en cuestión de meses o incluso semanas.

- **Facilidad de uso:** 2

  Se busca que la interfaz de la aplicación lo más sencilla posible, siguiendo la estructura de cualquier plataforma online: La única dificultad añadida a los usuarios es familiarizarse en el funcionamiento de una plataforma de subastas y a sus distintos tipos.

- **Portabilidad:** 3

  La aplicación una vez instalada en su correspondiente servidor no será necesario portarlo a otros entornos. Aún así, al tratar de una aplicación java, sería posible realizar un cambio de plataforma de manera sencilla

- **Facilidad de cambio:** 2

  El tipo de aplicación realizada está bastante definida y encapsulada. Por lo tanto, no es frecuente que se realicen grandes cambios en un futuro. A pesar de ello, se busca realizar una programación ordenada y documentada para facilitar esta tarea en un futuro.

- **Concurrencia:** 1

  La aplicación está diseñada para trabajar con un gran volumen de usuarios, pero sus operaciones internas no se solaparán unas a otras.

- **Características especiales de seguridad:** 5

  El sistema va a trabajar con activos económicos, por ello es fundamental garantizar su integridad.

- **Acceso directo a terceras partes**: 0

  No se permitirá a terceras partes acceder al sistema.

- **Se requiere entrenamiento especial del usuario**: 1

  Simplemente se tratará de interactuar con una aplicación web, por lo tanto, será una tarea con la que cualquier usuario está familiarizado hoy en día.

CÁLCULO DE DEL TCF

Una vez evaluados los factores de complejidad técnica se procede a calcular sus puntos:

En primer lugar, se realiza la suma ponderada de todos los factores:

$$TPoints = (2*3)+(1*5)+(1*1)+(1*2)+(1*2)+(0.5*3)+(0.5*2)+(2*3) + (1*2)+(1*1)+(1*5)+(1*0)+(1*1) = 33.5$$

Una vez tenemos ponderados los factores se procede a calcular el TCF

$$TCF = 0.6 + (0.01 * TPoints) = 0.6 + 0.01 * 33.5 = 0.936$$

3.3 Factores de complejidad del entorno

Para calcular los puntos de los factores de complejidad del entorno, se definen ocho factores a evaluar. A cada uno de esos factores se le asignará un peso (W) acorde a su impacto y una complejidad percibida por el equipo de desarrollo (F)

- **Familiaridad con UML:** 3

  El desarrollador ya ha estudiado y trabajado con la notación UML en el pasado.

- **Trabajadores a tiempo parcial:** 0

  No habrá ningún desarrollador parcial en proyecto.

- **Capacidad de los analistas:** 2

  El desarrollador ya ha trabajado en algunos proyectos de análisis durante sus estudios, pero nunca en uno de tal envergadura.

- **Experiencia en la aplicación:** 2

  El desarrollador tiene cierta experiencia en el desarrollo de aplicaciones *blockchain*.

- **Experiencia en la orientación a objetos:** 3

  El desarrollador ha trabajado a lo largo de los últimos 4 años con lenguajes de programación orientados a objetos.

- **Motivación:** 4

  Se trata de un proyecto novedoso a la par de interesante, el cual se enfrentará con un gran entusiasmo.

- **Dificultad del lenguaje de programación:** 4

  Debido a la envergadura de la solución buscada se utilizarán diversos lenguajes de programación en función de las diferentes partes.

- **Estabilidad de los requisitos:** 4

  Al utilizar el proceso unificado, los requisitos pasarán por varias iteraciones a lo largo del proyecto. Aun así, si hubiera algún cambio en los requisitos, este sería fácilmente abordable.

CALCULO DE DEL ECF

Una vez evaluados los factores de complejidad del entorno se procede a calcular sus puntos:

En primer lugar, se realiza la suma ponderada de todos los factores:

$$EPoints = (1.5 * 3) + (-1 * 0) + (0.5 * 2) + (0.5 * 2) + (1 * 3) + (1 * 4) + (-1 * 4) + (2 * 4) = 17.5$$

Una vez tenemos ponderados los factores se procede a calcular el ECF

$$ECF = 1.4 + (-0.03 * EPoints) = 1.4 - 0.03 * 17.5 = 0.875$$

3.4 Cálculo de los puntos de casos de uso

Una vez calculados los puntos de caso de uso sin ajustar (UUCP), junto con los factores de complejidad técnica (TCF) y del entorno (ECF), se procederá a calcular el total de los puntos de caso de uso (UCP).

$$UCP = UUCP * TCF * ECF = 252 * 0.936 * 0.875 = 206.388$$

El autor de esta metodología indica que se debe considerar que cada punto de caso de uso conlleva una duración de unas 20 horas de trabajo. Investigaciones posteriores, como por ejemplo la de Kirsters Ribu (2001), concluye que cada punto de casos de uso se ejecutará en un intervalo entre 15 y 30, siendo más realista la primera cifra. Es por ello por lo que será la cifra considerada para realizar los cálculos.

$$Total_{horas} = 206.388 * 15 = 3095\ horas$$

4. Descripción de tareas, subtareas e hitos

A continuación, se presenta el catálogo de tareas e hitos basado en el ciclo de vida expuesto en el apartado 2 del presente documento. Por lo tanto, este catálogo se dividirá en iteraciones, al final de las cuales se alcanzará un hito. Cada una de estas iteraciones tendrá una serie de tareas asociadas, descritas en la siguiente tabla:

Nombre de la tarea			Descripción
Comienzo del proyecto			Punto de inicio del proyecto, a partir de él se realizará la planificación temporal
Interacción inicial			Esta iteración se corresponderá con la fase de iniciación
1.1	Modelado		Se centrará en una investigación previa, tanto de los conceptos teóricos, como de las tecnologías existentes.
	1.1.1	Investigación de los conceptos teóricos	Investigación centrada en el concepto de cadena de bloques y de los tipos de subasta.
	1.1.2	Análisis de las tecnologías existentes	Análisis de las soluciones blockchain existentes, buscando la solución óptima para el desarrollo del proyecto
	1.1.3	Identificar objetivos	Identificación de objetivos del proyecto
Hito 01 – Fin de la iteración inicial			Final de la primera iteración del proyecto, en este punto ya se tendrá un conocimiento más técnico de la problemática a abordar
Segunda iteración			Fase de elaboración
2.1	Modelado		Se centrará en una investigación previa, tanto de los conceptos teóricos, como de las tecnologías existentes
	2.1.1	Pequeñas correcciones de la iteración anterior	Revisión del modelado de la iteración anterior
2.2	Modelo de requisitos		Especificación de requisitos
	2.2.1	Catálogo de requisitos funcionales y definición de los actores	Especificación de requisitos funcionales mediante el modelado por casos de uso.
	2.2.2	Catálogo de requisitos no funcionales	Especificación de los requisitos no funcionales.

	2.2.3	Catálogo de requisitos de información	Especificación de los requisitos de información (almacenamiento de persistente)
Hito 02 – Fin de la segunda iteración			Final de la segunda iteración
Tercera iteración			Será la segunda iteración de la fase de elaboración. Se centrará en el modelo de diseño
3.1	Modelo de requisitos		Especificación de requisitos
	3.1.1	Refinamiento de los requisitos	Refinamiento de los requisitos
3.2	Diseño		A partir de los requisitos, realizar el diseño del sistema
	3.2.1	Diseño arquitectónico	Diseño de la estructura interna del sistema
	3.2.2	Diseño de datos	Diseño de la estructura de los datos persistentes
	3.2.3	Vista de iteración	Creación de los diagramas de secuencia
	3.2.4	Modelo de despliegue	Creación del modelo de despliegue de la aplicación
Hito 03 – Fin de la tercera iteración			Finalización de la fase de elaboración
Cuarta iteración			Iteración que se corresponde con la fase de construcción
4.1	Modelo de diseño		Realizar el diseño del sistema
	4.1.1	Refinamiento del diseño	Pequeños cambios en el modelo de diseño
4.2	Implementación		Programación del sistema y sus partes
	4.2.1	Implementación del *token* TRC-20 propio	Creación del *token* a utilizar como moneda de pago en la aplicación
	4.2.2	Implementación del *smart contract* para la gestión de las subastas	Creación del *smart contract* encargado de la gestión automática de las subastas
	4.2.3	Implementación de la interfaz web	Codificación de la plantilla de la aplicación web
	4.2.4	Implementación del sistema de subastas	Codificación del sistema de subastas, y de su conexión con *blockchain*
	4.2.5	Implementación del sistema de pujas	Codificación del sistema de pujas, y de su conexión con *blockchain*
	4.2.6	Implementación del sistema de notificaciones	Codificación del sistema de notificaciones y el Daemon encargado de explorar los eventos de la cadena de bloques

4.3	Pruebas		Realización de pruebas.
	4.3.1	Pruebas a los *smart contract*	Se realizarán pruebas unitarias a los *smart contract* para garantizar su integridad y correcto funcionamiento
Hito 04 – Fin de la cuarta iteración			Fin de la fase de elaboración
Quinta y última iteración			Iteración que se corresponde con la fase de transición
5.1	Implementación		Programación del sistema y sus partes
	5.1.1	Retoques en la implementación.	Ultimas mejoras de la interfaz
	5.1.2	Elaboración de los manuales	Elaboración de los manuales necesarios
5.2	Pruebas		Realización de pruebas.
	5.2.1	Prueba del sistema de subastas	Se comprobará que el sistema de subastas funcione correctamente y que los *smart contracts* respondan correctamente
	5.2.2	Prueba del sistema de pujas	Se comprobará que las pujas se realicen correctamente
	5.2.3	Pruebas del sistema de notificaciones	Se comprobará que se reciban correctamente todos los eventos de la cadena de bloques, y que se notifique a los usuarios
5.3	Últimos retoques		Elaboración de las conclusiones y aspectos futuros.
Fin del proyecto			Finalización del trabajo fin de grado y su presentación

Tabla 3. Tabla de descripción de tareas e hitos.

5. Asignación de recursos

Proyecto	**152 días?**	**lun 01/02/21**	**mar 31/08/21**
Iteración 1: fase de iniciación	**22 días?**	**lun 01/02/21**	**mar 02/03/21**
1.1 Modelado	**22 días?**	**lun 01/02/21**	**mar 02/03/21**
Tarea 1.1.1 Investigación de los conceptos teóricos	13 días	lun 01/02/21	mié 17/02/21
Tarea 1.1.2 Analisis de las tecnologías existentes	5 días?	jue 18/02/21	mié 24/02/21
Tarea 1.1.3 Identificar objetivos	4 días?	jue 25/02/21	mar 02/03/21
Iteración 2: fase de elaboración (I)	**21 días?**	**mié 03/03/21**	**mié 31/03/21**
2.1 Modelado	**3 días?**	**mié 03/03/21**	**vie 05/03/21**
Tarea 2.1.1. Pequeñas correcciones de la iteración anterior	3 días?	mié 03/03/21	vie 05/03/21
2.2 Modelado de requisitos	**18 días?**	**lun 08/03/21**	**mié 31/03/21**
Tarea 2.2.1 Catalogo de requisitos funcionales y definición de los actores	8 días?	lun 08/03/21	mié 17/03/21
Tarea 2.2.2 Catalogo de requisitos no funcionales	6 días?	jue 18/03/21	jue 25/03/21
Tarea 2.2.3 Catálogo de requisitos de información	4 días?	vie 26/03/21	mié 31/03/21
Iteración 3: fase de elaboración (II)	**20 días?**	**jue 01/04/21**	**mié 28/04/21**
3.1 Modelado de requisitos	**2 días?**	**jue 01/04/21**	**vie 02/04/21**
Tarea 3.1.1 Refinamiento de requisitos	2 días?	jue 01/04/21	vie 02/04/21
3.2 Diseño	**18 días?**	**lun 05/04/21**	**mié 28/04/21**
Tarea 3.2.1 Diseño arquitectónico	5 días?	lun 05/04/21	vie 09/04/21
Tarea 3.2.2 Diseño de datos	3 días?	lun 12/04/21	mié 14/04/21
Tarea 3.2.3 Vista de iteración	9 días?	jue 15/04/21	mar 27/04/21
Tarea 3.2.4 Modelo de despliegue	1 día?	mié 28/04/21	mié 28/04/21
Iteración 4: fase de construcción	**66 días?**	**jue 29/04/21**	**jue 29/07/21**
4.1 Modelo de diseño	**3 días**	**jue 29/04/21**	**lun 03/05/21**
Tarea 4.1.1 Refinamiento del diseño	3 días	jue 29/04/21	lun 03/05/21

Figura 2. Asignación de recursos (I).

4.2 Implementación	**59 días?**	**mar 04/05/21**	**vie 23/07/21**
Tarea 4.2.1 Implementación del token TRC-20 propio	5 días?	mar 04/05/21	lun 10/05/21
Tarea 4.2.2 Implementación del smart contract para la gestión de las subastas	10 días?	mar 11/05/21	lun 24/05/21
Tarea 4.2.3 Implementación de la interfaz web	20 días?	mar 25/05/21	lun 21/06/21
Tarea 4.2.4 Implementación del sistema de subastas	10 días?	mar 22/06/21	lun 05/07/21
Tarea 4.2.5 Implementación del sistema de pujas	6 días?	mar 06/07/21	mar 13/07/21
Tarea 4.2.6 Implementación del sistema de notificaciones	8 días?	mié 14/07/21	vie 23/07/21
4.3 Pruebas	**4 días?**	**lun 26/07/21**	**jue 29/07/21**
Tarea 4.3.1 Pruebas unitarias a los smart contracts	4 días?	lun 26/07/21	jue 29/07/21
Iteración 5: fase de transición	**23 días?**	**vie 30/07/21**	**mar 31/08/21**
5.1 Implementación	**11 días?**	**vie 30/07/21**	**vie 13/08/21**
Tarea 5.1.1 Retoques en la implementación	5 días?	vie 30/07/21	jue 05/08/21
Tarea 5.1.2 Elaboración de los manuales	6 días	vie 06/08/21	vie 13/08/21
5.2 Pruebas	**5 días?**	**lun 16/08/21**	**vie 20/08/21**
Tarea 5.2.1 Prueba del sistema de subastas	5 días?	lun 16/08/21	vie 20/08/21
Tarea 5.2.2 Prueba del sistema de pujas	5 días?	lun 16/08/21	vie 20/08/21
Tarea 5.2.3 Prueba del sistema de notificaciones	5 días?	lun 16/08/21	vie 20/08/21
5.3 Últimos retoques	7 días?	lun 23/08/21	mar 31/08/21

Figura 3. Asignación de recursos (II)

6. Diagrama de Gantt

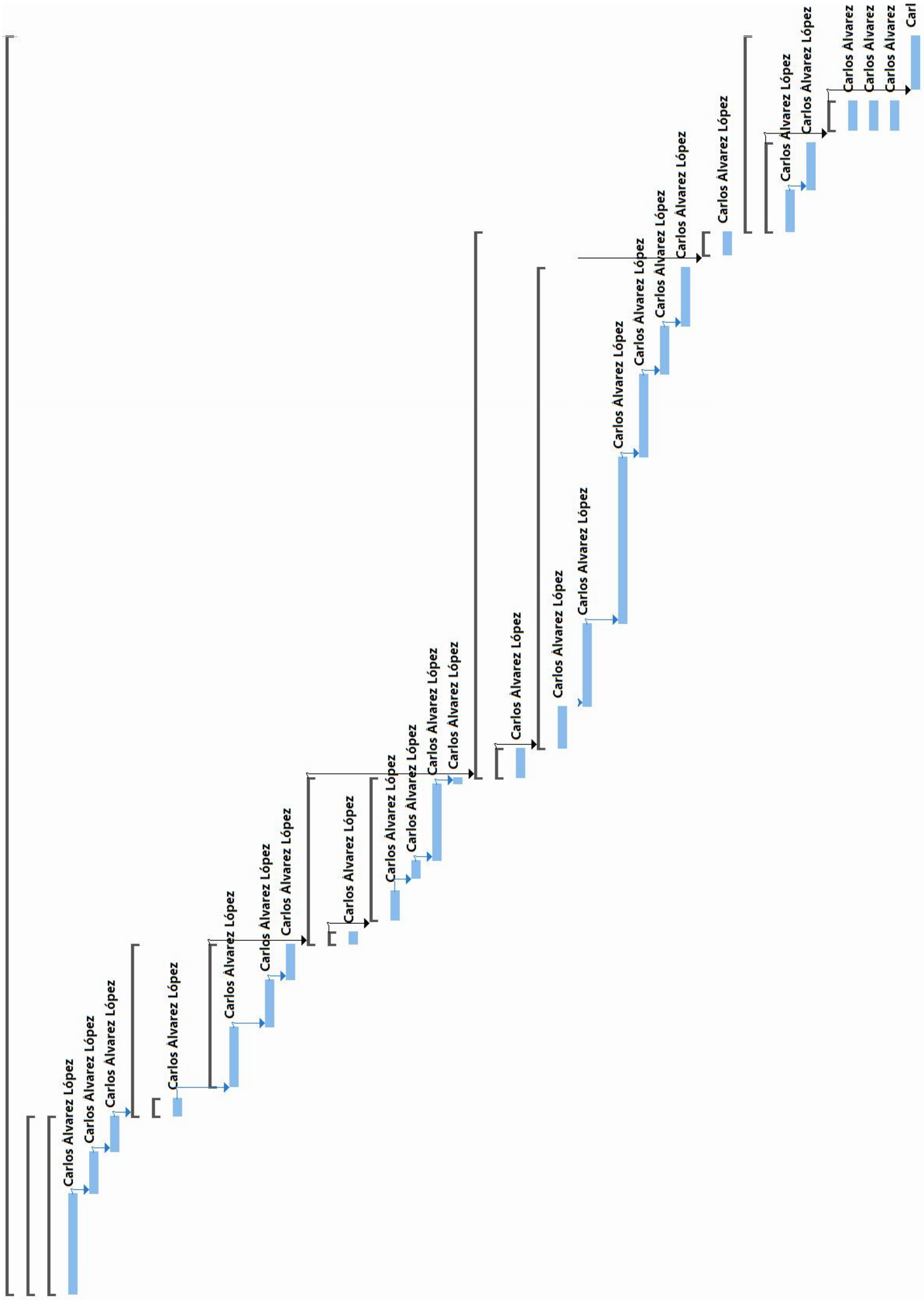

Figura 4. Diagrama de Gantt

TronAuction: plataforma de subastas online utilizando *Smart Contracts*

Anexo II: Especificación de requisitos del software

Grado en ingeniería informática

VNiVERSiDAD D SALAMANCA

Septiembre de 2021

Autor

Carlos Álvarez López

Tutores

Juan Manuel Corchado Rodríguez

Diego Javier Valdeolmillos Villaverde

Agustín San Román Guzmán

Tabla de contenido

Índice de figuras

Índice de tablas

1 Introducción

En el presente documento se presentará la especificación de requisitos para el proyecto *TronAuction*: plataforma de subastas online utilizando *smart contracts.*

En la primera parte del documento se encuentran definidos los participantes y objetivos del proyecto. En la segunda parte se expondrá la especificación formal de los requisitos (de información, no funcionales, y funcionales), y finalmente se expondrán las matrices de trazabilidad.

2 Listado de participantes

Participante	**Carlos Álvarez**
Rol	Jefe de proyecto y desarrollador
Organización	Universidad de Salamanca
Comentarios	Se encargará de todas las tareas relacionadas con la investigación, análisis, diseño, implementación, pruebas y documentación

Tabla 1: Participante Carlos Álvarez

Participante	**Juan Manuel Corchado**
Rol	Tutor
Organización	Universidad de Salamanca
Comentarios	Tutor del proyecto, y por lo tanto se encargará del asesoramiento y correcciones

Tabla 2: Participante Juan Manuel Corchado

Participante	**Diego Javier Valdeolmillos**
Rol	Tutor
Organización	Universidad de Salamanca
Comentarios	Tutor del proyecto, y por lo tanto se encargará del asesoramiento y correcciones

Tabla 3: Participante Diego Javier Valdeolmillos

Participante	**Agustín San Román**
Rol	Tutor
Organización	Universidad de Salamanca
Comentarios	Tutor del proyecto, y por lo tanto se encargará del asesoramiento y correcciones

Tabla 4: Participante Agustín San Román

3 Objetivos del proyecto

A lo largo del apartado se presentará la especificación formar de los objetivos del sistema siguiendo la metodología de Durán y Bernárdez.

OBJ-0001	**Gestionar usuarios**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El sistema deberá de ser capaz de registrar nuevos usuarios, modificarlos y de eliminarlos
Subobjetivos	**[OBJ-0003] Gestionar *wallets***: Al registrase un nuevo usuario, se le debe crear una nueva wallet
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 5: OBJ-0001 Gestionar usuarios

OBJ-0002	**Sistema *blockchain***
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El sistema se comunicará con el *smart contract* desplegado en la cadena de bloques. Deberá poder ejecutar sus métodos, así como leer sus eventos
Subobjetivos	Ninguno
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 6: OBJ-0002 Sistema blockchain

OBJ-0003	Gestionar *wallets*
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El sistema debe ser capaz de crear *wallets* asociadas a un usuario, así como modificarlas y eliminarlas. Para facilitar el uso de los monederos será el sistema el encargado de la seguridad y del manejo de estos
Subobjetivos	Ninguno
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 7: OBJ-0003 Gestionar wallets

OBJ-0004	Transferir *tokens*
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El sistema permitirá a los usuarios enviar *tokens* desde su monedero al monedero que ellos consideren oportuno, ya bien sea a un monedero de otro usuario, como a cualquier monedero externo a la aplicación
Subobjetivos	**[OBJ-0003] Gestionar *wallets***: Cuando un usuario desee transferir una serie de *tokens*, el sistema debe consultar la *wallet* del usuario para recuperar su clave privada
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 8: OBJ-0004 Transferir tokens

OBJ-0005	Gestionar subastas
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	Se permitirá a los usuarios la creación de nuevas subastas, pero solo se permitirá su modificación y su eliminación en determinadas circunstancias. Para ello, todas las subastas serán creadas en el *smart contract* y las modificaciones se registrarán en el mismo
Subobjetivos	**[OBJ-0003] Gestionar *wallets***: Cuando un usuario desee crear una subasta, el sistema debe consultar su *wallet* para recuperar su clave privada, e interactuar con el *smart contract* **[OBJ-0002] Sistema *blockchain***: Cuando se quiera crear o modificar una subasta, se deberá registrar los datos en la cadena de bloques
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 9: OBJ-0005 Gestionar subastas

OBJ-0006	Filtrar subastas
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El sistema deberá poder mostrar las subastas creadas, así como ordenarlas y filtrarlas a gusto de los usuarios
Subobjetivos	**[OBJ-0005] Gestionar subastas**: Se deben recuperar los datos de las subastas creadas
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 10: OBJ-0006 Visualizar subastas

OBJ-0007	Gestionar pujas
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	Se debe permitir a los usuarios la puja en las subastas activas
Subobjetivos	**[OBJ-0003] Gestionar *wallets***: Cuando un usuario desee crear una subasta, el sistema debe consultar su *wallet* para recuperar su clave privada, e interactuar con el *smart contract* **[OBJ-0002] Sistema *blockchain***: Cuando se quiera realizar una puja, se deberá registrar los datos en la cadena de bloques **[OBJ-0005] Sistema *blockchain***: Cuando se quiera realizar una puja, esta debe de ir asociada a una subasta
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 11: OBJ-0007 Gestionar pujas

OBJ-0008	Enviar notificaciones
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
***Descripción**	El sistema enviará a los usuarios una notificación cuando se realice una puja, cuando termine una subasta propia, o cuando resulte ganador
Subobjetivos	**[OBJ-0002] Sistema *blockchain***: El sistema escuchará los eventos del *smart contract*
Importancia	Vital
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 12: OBJ-0008 Enviar notificaciones

4 Catálogo de requisitos del sistema

4.1 Catálogo de requisitos de información

ID	Nombre	Descripción
IRQ-0001	Usuarios	El sistema deberá guardar la información relativa a los usuarios de la aplicación
IRQ-0002	Susbastas	El sistema deberá almacenar la información relativa a las subastas creadas
IRQ-0003	*Wallets*	El sistema deberá almacenar los datos de las *wallets* de los usuarios
IRQ-0004	Pujas	El sistema deberá controlar las pujas realizadas por los usuarios
IRQ-0005	Notificaciones	El sistema deberá almacenar un histórico de las notificaciones enviadas
IRQ-0006	Subastas	El sistema almacenará información relativa a las sesiones abiertas por los usuarios

Tabla 13: Catálogo de requisitos de información

4.2 Catálogo de requisitos no funcionales

ID	Nombre	Descripción
NFR-0001	Privacidad de la información	El sistema deberá tener la seguridad suficiente para que *no se expongan los datos privados de los compradores y vendedores, respetando la Ley Orgánica de Protección de Datos y la RGPD*
NFR-0002	Portabilidad	El sistema deberá poseer un diseño "*Responsive*" a fin de garantizar su correcta visualización en múltiples ordenadores personales, teléfonos inteligentes y tabletas
NFR-0003	Tiempo de aprendizaje	El tiempo de aprendizaje del manejo del sistema por un usuario deberá ser menor a 2 horas
NFR-0004	Tiempo de respuesta	Toda funcionalidad del sistema deberá responder al usuario en menos de 7 segundos
NFR-0005	Concurrencia de vista	El sistema deberá ser capaz de operar correctamente con hasta 3000 dispositivos de forma concurrente
NFR-0006	Interfaces gráficas	El sistema deberá poseer interfaces gráficas bien formadas y organizadas
NFR-0007	Conexiones seguras	Se debe garantizar la seguridad de las comunicaciones

Tabla 14: Catálogo de requisitos no funcionales

4.3 Catálogo de requisitos funcionales

ID	**Nombre**	**Descripción**
UC-0001	Iniciar sesión	El sistema deberá permitir el inicio de sesión, cuando el actor Usuario pulse el botón de iniciar sesión
UC-0002	Cerrar sesión	El sistema deberá permitir el cierre de sesión, cuando el actor Usuario pulse el botón de cerrar sesión
UC-0003	Registrar usuario	El sistema deberá permitir registrar un nuevo usuario, cuando el actor Usuario pulse el botón registro
UC-0004	Modificar usuario	El sistema deberá permitir modificar los datos personales, cuando el actor Usuario pulse el botón modificar datos personales
UC-0005	Eliminar Usuario	El sistema deberá permitir eliminar los datos de un usuario del sistema, cuando el actor Usuario lo solicite
UC-0006	Recuperar usuario	El sistema deberá permitir al usuario modificar sus datos personales
UC-0007	Verificar email	El sistema deberá verificar automáticamente el email indicado
UC-0008	Crear *wallet*	El sistema deberá crear una nueva *wallet* de manera automática
UC-0009	Eliminar *wallet*	El sistema debe permitir el borrado automático de datos asociados a una *wallet*
UC-0010	Recuperar *wallet*	El sistema deberá permitir la recuperación automática de los datos asociados a una *wallet*
UC-0011	Realizar transacción *blockchain*	El sistema deberá permitir interactuar con el *smart contract* desplegado en la cadena de bloques
UC-0012	Firmar transacción	El sistema deberá permitir la firma de las transacciones antes de enviarlas a la cadena de bloques
UC-0013	Enviar transacción	El sistema deberá de permitir enviar transacciones firmadas a la cadena de bloques
UC-0014	Transferir *tokens*	El sistema deberá permitir el envío de *tokens* entre dos *wallets*
UC-0015	Consultar *tokens*	El sistema deberá permitir consultar al usuario su saldo
UC-0016	Comprar *tokens*	El sistema deberá permitir al usuario comprar *tokens*
UC-0017	Vender *tokens*	El sistema deberá permitir a los usuarios intercambiar dinero por *tokens*
UC-0018	Crear subasta	El sistema deberá permitir la creación de una nueva subasta por el actor Vendedor

UC-0019	Modificar subasta	El sistema deberá permitir la modificación de una subasta por el actor Vendedor
UC-0020	Eliminar subasta	El sistema deberá permitir la eliminación de una subasta por el actor Vendedor
UC-0021	Recuperar subasta	El sistema deberá permitir recuperar los datos de una subasta en curso
UC-0022	Realizar puja	El sistema deberá permitir la realización de pujas por parte de los usuarios
UC-0023	Congelar *tokens*	El sistema deberá permitir congelar los *tokens* del usuario cuando realice una puja
UC-0024	Descongelar *tokens*	El sistema deberá descongelar los *tokens* congelados de todas las pujas que no hayan resultado ganadoras
UC-0025	Finalizar subasta	El sistema deberá de ser capaz de terminar de manera automática una subasta
UC-0026	Registrar envío	El sistema deberá permitir al vendedor registrar el envío de un nuevo artículo
UC-0027	Modificar envío	El sistema deberá permitir al vendedor modificar los datos de un envío
UC-0028	Enviar notificación	El sistema deberá permitir el envío de notificaciones vía email a los usuarios
UC-0029	Escuchar eventos	El sistema deberá de ser capaz de recuperar los datos de los eventos del *smart contract*

Tabla 15: Catálogo de requisitos funcionales

5 Especificación de los requisitos del sistema

5.1 Requisitos de información

IRQ-0001	**Usuarios**	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	0BJ-0001 Gestionar usuarios	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0005 Interfaces Gráficas	
Descripción	El sistema deberá almacenar la información correspondiente a *los usuarios.*	
Datos específicos	• Nombre • NIF o DNI • Domicilio • Teléfono • Correo electrónico	
Tiempo de vida	**Medio**	**Máximo**
	-	-
Ocurrencias simultáneas	**Medio**	**Máximo**
	-	-
Importancia	Importante	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 16: Requisito de información IRQ-0001

IRQ-0002	Subastas	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	0BJ-0005 Gestionar subasta OBJ-0006 Filtrar subastas	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0005 Interfaces Gráficas	
Descripción	El sistema deberá almacenar la información correspondiente a *las subastas.*	
Datos específicos	• ID • Título • Tipo de subasta • Categoría • Subcategoría • Descripción • Imágenes • Fecha de creación • Fecha finalización	
Tiempo de vida	**Medio**	**Máximo**
	-	-
Ocurrencias simultáneas	**Medio**	**Máximo**
	-	-
Importancia	Importante	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 17: Requisito de información IRQ-0002

IRQ-0003	***Wallets***	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	0BJ-0003 Gestionar *wallets*	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0005 Interfaces Gráficas	
Descripción	El sistema deberá almacenar la información correspondiente a *las wallets*	
Datos específicos	• ID • ID del usuario • Dirección • Nombre del archivo • Contraseña	
Tiempo de vida	**Medio**	**Máximo**
	-	-
Ocurrencias simultáneas	**Medio**	**Máximo**
	-	-
Importancia	Importante	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 18: Requisito de información IRQ-0003

IRQ-0004	Pujas	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	0BJ-0007 Gestionar pujas	
Requisitos asociados	IRQ-0002 Subastas NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0005 Interfaces Gráficas	
Descripción	El sistema deberá almacenar la información correspondiente a *las pujas*	
Datos específicos	● ID ● ID de la subasta ● ID del pujador ● Cantidad ● Fecha	
Tiempo de vida	Medio	Máximo
	-	-
Ocurrencias simultáneas	Medio	Máximo
	-	-
Importancia	Importante	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 19: Requisito de información IRQ-0004

IRQ-0005	Notificaciones	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	0BJ-0008 Enviar notificaciones	
Requisitos asociados	IRQ-0002 Subastas IRQ-0004 Pujas NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0005 Interfaces Gráficas	
Descripción	El sistema deberá almacenar la información correspondiente a *las notificaciones enviadas*	
Datos específicos	• ID • ID del usuario • Contenido • Fecha	
Tiempo de vida	**Medio**	**Máximo**
	-	-
Ocurrencias simultáneas	**Medio**	**Máximo**
	-	-
Importancia	Importante	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 20: Requisito de información IRQ-0005

IRQ-0006	Sesiones	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	0BJ-0001 Gestionar usuarios	
Requisitos asociados	IRQ-0001 Usuarios NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0005 Interfaces Gráficas	
Descripción	El sistema deberá almacenar la información correspondiente a las notificaciones enviadas	
Datos específicos	• ID • ID del usuario • *Token* de sesión • Fecha de creación • Fecha de finalización	
Tiempo de vida	Medio	Máximo
	-	-
Ocurrencias simultáneas	Medio	Máximo
	-	-
Importancia	Importante	
Urgencia	Inmediatamente	
Estado	Validado	
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 21: Requisito de información IRQ-0006

5.2 Requisitos no funcionales

NFR-0001	**Privacidad de la información**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0003 Gestionar *Wallets* OBJ-0007 Gestionar pujas OBJ-0008 Enviar notificaciones
Requisitos asociados	IRQ-0001 Usuarios IRQ-0003 *Wallets* IRQ-0004 Pujas IRQ-0005 Notificaciones IRQ-0006 Sesiones UC-0001 Iniciar sesión UC-0002 Cerrar sesión UC-0003 Registrar usuario UC-0004 Modificar usuario UC-0005 Eliminar usuario UC-0006 Recuperar usuario UC-0007 Verificar email UC-0008 Crear *wallet* UC-0009 Eliminar *wallet* UC-0010 Recuperar *wallet* UC-0012 Firmar transacción UC-0015 Consultar *tokens* UC-0016 Comprar *tokens* UC-0017 Vender *tokens* UC-0022 Realizar puja UC-0028 Enviar notificación

Descripción	El sistema deberá tener la seguridad suficiente para que *no se expongan los datos privados de los trabajadores, proveedores o tratantes, respetando la Ley Orgánica de Protección de Datos y la RGPD*
Importancia	Importante
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Toda la información personal, debe estar almacenada de tal forma que sea imposible acceder sin permiso

Tabla 22: Requisito no funcional NFR-0001

NFR-0002	**Portabilidad**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens* OBJ-0005 Gestionar subastas OBJ-0006 Visualizar subastas OBJ-0007 Gestionar pujas OBJ-0008 Enviar notificaciones
Requisitos asociados	IRQ-0001 Usuarios IRQ-0002 Subastas IRQ-0003 *Wallets* IRQ-0004 Pujas IRQ-0005 Notificaciones IRQ-0006 Sesiones UC-0001 Iniciar sesión UC-0002 Cerrar sesión UC-0003 Registrar usuario

	UC-0004 Modificar usuario UC-0005 Eliminar usuario UC-0006 Recuperar usuario UC-0007 Verificar email UC-0008 Crear *wallet* UC-0009 Eliminar *wallet* UC-0010 Recuperar *wallet* UC-0011 Realizar transacción *blockchain* UC-0012 Firmar transacción UC-0013 Enviar transacción UC-0014 Transferir *tokens* UC-0015 Consultar *tokens* UC-0016 Comprar *tokens* UC-0017 Vender tokens UC-0018 Crear subasta UC-0019 Modificar subasta UC-0020 Eliminar subasta UC-0021 Recuperar subasta UC-0022 Realizar puja UC-0023 Congelar *tokens* UC-0024 Descongelar *tokens* UC-0025 Finalizar subasta UC-0026 Registar envío UC-0027 Modificar envío UC-0028 Enviar notificación UC-0029 Escuchar eventos
Descripción	El sistema deberá poseer un diseño "Responsive" a fin de garantizar su correcta visualización en múltiples ordenadores personales, teléfonos inteligentes y tabletas
Importancia	Importante
Urgencia	hay presión
Estado	En construcción

Estabilidad	Alta
Comentarios	Ninguno

Tabla 23: Requisito no funcional NFR-0002

NFR-0003	**Tiempo de aprendizaje**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens* OBJ-0005 Gestionar subastas OBJ-0006 Visualizar subastas OBJ-0007 Gestionar pujas
Requisitos asociados	UC-0001 Iniciar sesión UC-0002 Cerrar sesión UC-0003 Registrar usuario UC-0004 Modificar usuario UC-0005 Eliminar usuario UC-0006 Recuperar usuario UC-0007 Verificar email UC-0008 Crear *wallet* UC-0009 Eliminar *wallet* UC-0010 Recuperar *wallet* UC-0014 Transferir *tokens* UC-0015 Consultar *tokens* UC-0016 Comprar *tokens* UC-0017 Vender *tokens* UC-0018 Crear subasta UC-0019 Modificar subasta

	UC-0020 Eliminar subasta UC-0021 Recuperar subasta UC-0022 Realizar puja UC-0026 Registar envío UC-0027 Modificar envío
Descripción	El tiempo de aprendizaje del manejo del sistema por un usuario deberá ser menor a 2 horas
Importancia	Importante
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 24: Requisito no funcional NFR-0003

NFR-0004	**Tiempo de respuesta**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens* OBJ-0005 Gestionar subastas OBJ-0006 Visualizar subastas OBJ-0007 Gestionar pujas OBJ-0008 Enviar notificaciones
Requisitos asociados	IRQ-0001 Usuarios IRQ-0002 Subastas IRQ-0003 *Wallets* IRQ-0004 Pujas IRQ-0005 Notificaciones

	IRQ-0006 Sesiones UC-0001 Iniciar sesión UC-0002 Cerrar sesión UC-0003 Registrar usuario UC-0004 Modificar usuario UC-0005 Eliminar usuario UC-0006 Recuperar usuario UC-0007 Verificar email UC-0008 Crear *wallet* UC-0009 Eliminar *wallet* UC-0010 Recuperar *wallet* UC-0011 Realizar transacción *blockchain* UC-0012 Firmar transacción UC-0013 Enviar transacción UC-0014 Transferir *tokens* UC-0015 Consultar *tokens* UC-0016 Comprar *tokens* UC-0017 Vender *tokens* UC-0018 Crear subasta UC-0019 Modificar subasta UC-0020 Eliminar subasta UC-0021 Recuperar subasta UC-0022 Realizar puja UC-0023 Congelar *tokens* UC-0024 Descongelar *tokens* UC-0025 Finalizar subasta UC-0026 Registar envío UC-0027 Modificar envío UC-0028 Enviar notificación UC-0029 Escuchar eventos
Descripción	Toda funcionalidad del sistema deberá responder al usuario en menos de 7 segundos

Importancia	Importante
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 25: Requisito no funcional NFR-0004

NFR-0005	**Concurrencia de vista**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens* OBJ-0005 Gestionar subastas OBJ-0006 Visualizar subastas OBJ-0007 Gestionar pujas OBJ-0008 Enviar notificaciones
Requisitos asociados	UC-0001 Iniciar sesión UC-0002 Cerrar sesión UC-0003 Registrar usuario UC-0004 Modificar usuario UC-0005 Eliminar usuario UC-0006 Recuperar usuario UC-0007 Verificar email UC-0008 Crear *wallet* UC-0009 Eliminar *wallet* UC-0010 Recuperar *wallet* UC-0014 Transferir *tokens* UC-0015 Consultar *tokens*

	UC-0016 Comprar *tokens* UC-0017 Vender *tokens* UC-0018 Crear subasta UC-0019 Modificar subasta UC-0020 Eliminar subasta UC-0021 Recuperar subasta UC-0022 Realizar puja UC-0026 Registar envío UC-0027 Modificar envío
Descripción	El sistema deberá ser capaz de operar correctamente con hasta 30 dispositivos de forma concurrente
Importancia	Importante
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 26: Requisito no funcional NFR-0005

NFR-0006	**Interfaces gráficas**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens* OBJ-0005 Gestionar subastas OBJ-0006 Visualizar subastas OBJ-0007 Gestionar pujas OBJ-0008 Enviar notificaciones
Requisitos asociados	UC-0001 Iniciar sesión

	UC-0002 Cerrar sesión UC-0003 Registrar usuario UC-0004 Modificar usuario UC-0005 Eliminar usuario UC-0006 Recuperar usuario UC-0007 Verificar email UC-0008 Crear *wallet* UC-0009 Eliminar *wallet* UC-0010 Recuperar *wallet* UC-0014 Transferir *tokens* UC-0015 Consultar *tokens* UC-0016 Comprar *tokens* UC-0017 Vender tokens UC-0018 Crear subasta UC-0019 Modificar subasta UC-0020 Eliminar subasta UC-0021 Recuperar subasta UC-0022 Realizar puja UC-0026 Registar envío UC-0027 Modificar envío
Descripción	El sistema deberá poseer interfaces gráficas bien formadas y organizadas
Importancia	Importante
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 27: Requisito no funcional NFR-0006

NFR-0007	Conexiones seguras
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens* OBJ-0005 Gestionar subastas OBJ-0006 Visualizar subastas OBJ-0007 Gestionar pujas OBJ-0008 Enviar notificaciones
Requisitos asociados	IRQ-0001 Usuarios IRQ-0002 Subastas IRQ-0003 *Wallets* IRQ-0004 Pujas IRQ-0005 Notificaciones IRQ-0006 Sesiones UC-0001 Iniciar sesión UC-0002 Cerrar sesión UC-0003 Registrar usuario UC-0004 Modificar usuario UC-0005 Eliminar usuario UC-0006 Recuperar usuario UC-0007 Verificar email UC-0008 Crear *wallet* UC-0009 Eliminar *wallet* UC-0010 Recuperar *wallet* UC-0011 Realizar transacción *blockchain* UC-0012 Firmar transacción UC-0013 Enviar transacción UC-0014 Transferir *tokens*

	UC-0015 Crear subasta UC-0016 Modificar subasta UC-0017 Eliminar subasta UC-0018 Recuperar subasta UC-0019 Realizar puja UC-0020 Congelar *tokens* UC-0021 Descongelar *tokens* UC-0022 Finalizar subasta UC-0023 Registar envío UC-0024 Modificar envío UC-0025 Enviar notificación UC-0026 Escuchar eventos
Descripción	Se debe de garantizar la seguridad de las comunicaciones.
Importancia	Importante
Urgencia	Hay presión
Estado	En construcción
Estabilidad	Alta
Comentarios	Ninguno

Tabla 28: Requisito no funcional NFR-0007

5.3 Requisitos funcionales

5.3.1 Actores del sistema

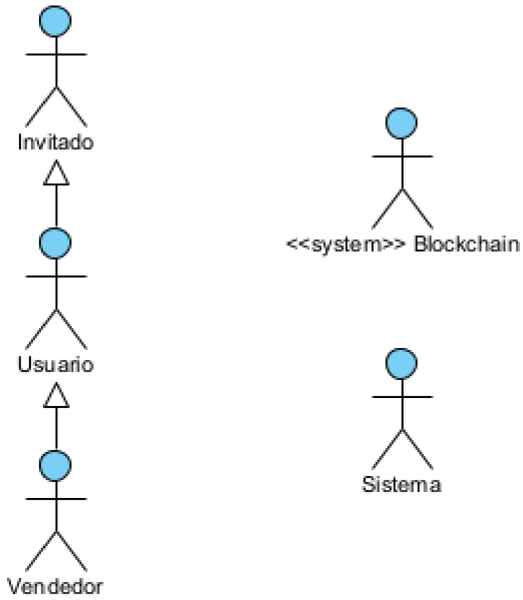

Ilustración 1: Jerarquía de actores del sistema

ACT-01	Invitado
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El actor representa a un usuario sin iniciar sesión en la plataforma
Comentarios	Ninguno

Tabla 29: Actor ACT-01

ACT-02	Usuario
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El actor representa a un usuario de la plataforma que está registrado y ha iniciado sesión
Comentarios	Ninguno

Tabla 30: Actor ACT-02

ACT-03	**Vendedor**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El actor representa a un vendedor
Comentarios	Ninguno

Tabla 31: Actor ACT-03

ACT-04	**<<System>> *Blockchain***
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El actor representa la API del nodo de *Shasta* proporcionado por *Trongrid* para la interacción con la cadena de bloques
Comentarios	Ninguno

Tabla 32: Actor ACT-04

ACT-05	**Sistema**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Descripción	El actor representa al subsistema que realiza tareas de modo automático
Comentarios	Ninguno

Tabla 33: Actor ACT-05

5.3.2 Casos de usos

5.3.2.1 Usuarios

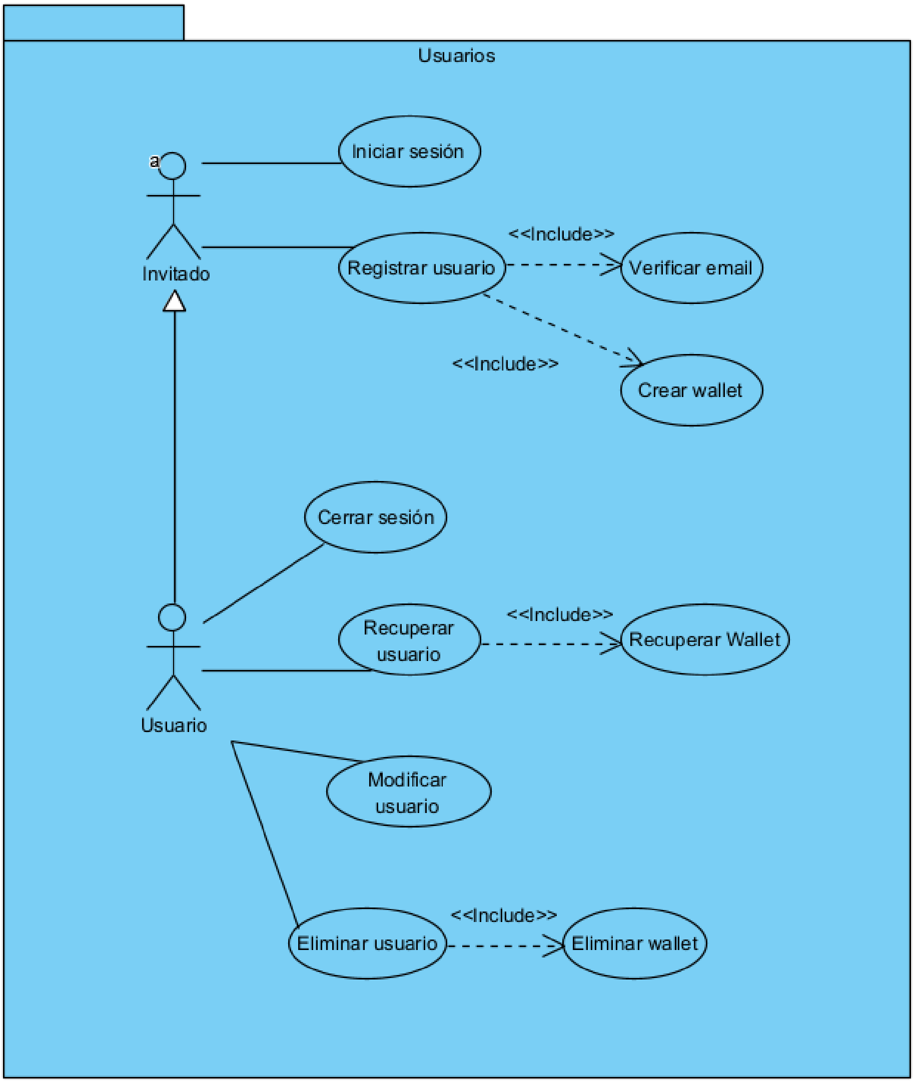

Figura 1: Casos de uso del paquete usuario

UC-0001	**Iniciar sesión**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje

	NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios	
Descripción	El sistema debe permitir el inicio de sesión a los usuarios	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-01 (*Invitado*) selecciona la opción de inicio de sesión
	2	El sistema solicita la información de inicio de sesión, en particular el nombre de usuario y la contraseña
	3	El usuario introduce los datos correspondientes
	4	El sistema comprueba los datos de inicio de sesión
	5	El sistema crea una nueva sesión
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si el sistema detecta algún error en la validez de los datos volverá al paso 2, mostrando un error
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 34: Requisito funcional UC-0001

UC-0002	**Cerrar sesión**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje

	NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios	
Descripción	El sistema debe permitir el inicio de sesión a los usuarios	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de cerrar sesión
	2	El sistema cierra y elimina la sesión del usuario
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	-	-
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 35: Requisito funcional UC-0002

UC-0003	**Registrar usuario**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0003 Gestionar *Wallets*
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas

	NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 *Wallets* UC-0001 Iniciar sesión UC-0007 Verificar email UC-0008 Crear *wallet*	
Descripción	El sistema debe permitir el inicio de sesión a los usuarios	
Precondición	Ninguna	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-01 (*Invitado*) selecciona la opción de registrar nuevo usuario
	2	El sistema solicita la información personal del nuevo usuario
	3	El usuario introduce los datos correspondientes
	4	Se ejecuta el caso de uso ***UC-0007 Verificar email***
	5	Se ejecuta el caso de uso ***UC-0008 Crear wallet***
	6	El sistema almacena los datos del usuario
	7	El sistema muestra en inicio del caso de uso ***UC-0001 Iniciar sesión***
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si el sistema detecta algún error en la validez de los datos volverá al paso 2, mostrando un error
	5	Si se detecta algún error en la creación de la *wallet*, se notificará al usuario y terminará el caso de uso
	6	Si se detecta algún error en el almacenamiento de los datos, se notificará al usuario y terminará el caso de uso
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 36: Requisito funcional UC-0003

<table>
<tr><th>UC-0004</th><th colspan="2">Modificar usuario</th></tr>
<tr><td>Versión</td><td colspan="2">1.0</td></tr>
<tr><td>Autores</td><td colspan="2">Carlos Álvarez López</td></tr>
<tr><td>Fuentes</td><td colspan="2">Carlos Álvarez López</td></tr>
<tr><td>Objetivos asociados</td><td colspan="2">OBJ-0001 Gestionar usuarios</td></tr>
<tr><td>Requisitos asociados</td><td colspan="2">NFR-0001 Privacidad de la información
NFR-0003 Tiempo de aprendizaje
NFR-0004 Tiempo de respuesta
NFR-0005 Concurrencia de vista
NFR-0006 Interfaces gráficas
NFR-0007 Conexiones seguras
IRQ-0001 Usuarios</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema debe permitir la modificación de sus datos a los usuarios</td></tr>
<tr><td>Precondición</td><td colspan="2">Debe existir el usuario</td></tr>
<tr><td rowspan="6">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El sistema recupera la información del usuario, detallada en el IRQ-0001 (Usuario)</td></tr>
<tr><td>2</td><td>Un usuario de tipo ACT-02 (usuario) selecciona la opción de modificar sus datos personales</td></tr>
<tr><td>3</td><td>El usuario introduce los datos personales que desea modificar</td></tr>
<tr><td>4</td><td>El sistema almacena los datos del usuario</td></tr>
<tr><td>6</td><td>El sistema notifica, a través de un aviso general, del cambio realizado sobre el usuario</td></tr>
<tr><td>Postcondición</td><td colspan="2">Ninguna</td></tr>
<tr><td rowspan="3">Excepciones</td><td>Paso</td><td>Acción</td></tr>
<tr><td>3</td><td>En caso de que se produzca algún error en la lectura de los datos, se le notifica al usuario y el caso de uso finaliza</td></tr>
<tr><td>4</td><td>Si se produce algún error al actualizar los datos se le notifica al usuario y el caso de uso terminará</td></tr>
<tr><td>Estabilidad</td><td colspan="2">Alta</td></tr>
<tr><td>Comentarios</td><td colspan="2">Ninguno</td></tr>
</table>

Tabla 37: Requisito funcional UC-0004

UC-0005	Eliminar usuario	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0003 Gestionar *Wallets*	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 *Wallets* UC-0009 Eliminar *wallet*	
Descripción	El sistema debe permitir la eliminación de un usuario	
Precondición	Debe de existir el usuario	
Secuencia normal	**Paso**	**Acción**
	1	El sistema recupera la información del usuario, detallada en el IRQ-0001 (*Usuario*)
	2	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de eliminar sus datos personales
	3	El usuario confirma la solicitud de borrado de la información
	4	Se ejecuta el caso de uso ***UC-0009 Eliminar wallet***
	5	El sistema elimina el usuario
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si se produce algún error al eliminar, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 38: Requisito funcional UC-0005

UC-0006	Recuperar usuario	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0001 Gestionar usuarios OBJ-0003 Gestionar *Wallets*	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 Wallets UC-0009 Recuperar *wallet*	
Descripción	El sistema debe permitir recuperar la información de un usuario	
Precondición	Debe de existir el usuario	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de mostrar sus datos personales
	2	Se ejecuta el caso de uso ***UC-00010 Recuperar wallet***
	3	El sistema recupera la información del usuario, detallada en el IRQ-0001 (*Usuario*)
	4	El sistema devuelve la información del usuario
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si se produce alguna excepción al ejecutar el caso de uso, se le notifica al usuario y el caso de uso termina
	3	Si se produce algún error al recuperar la información, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 39: Requisito funcional UC-0006

<table>
<tr><th>UC-0007</th><th colspan="2">Verificar email</th></tr>
<tr><td>Versión</td><td colspan="2">1.0</td></tr>
<tr><td>Autores</td><td colspan="2">Carlos Álvarez López</td></tr>
<tr><td>Fuentes</td><td colspan="2">Carlos Álvarez López</td></tr>
<tr><td>Objetivos asociados</td><td colspan="2">OBJ-0001 Gestionar usuarios</td></tr>
<tr><td>Requisitos asociados</td><td colspan="2">NFR-0001 Privacidad de la información
NFR-0003 Tiempo de aprendizaje
NFR-0004 Tiempo de respuesta
NFR-0005 Concurrencia de vista
NFR-0006 Interfaces gráficas
NFR-0007 Conexiones seguras
IRQ-0001 Usuarios
UC-0003 Registrar usuario</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema debe comprobar que el email que ha introducido el usuario es suyo</td></tr>
<tr><td>Precondición</td><td colspan="2">-</td></tr>
<tr><td rowspan="4">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El sistema envía un correo electrónico de verificación al usuario</td></tr>
<tr><td>2</td><td>El usuario valida su correo electrónico</td></tr>
<tr><td>3</td><td>El sistema valida el correo electrónico del usuario</td></tr>
<tr><td>Postcondición</td><td colspan="2">Ninguna</td></tr>
<tr><td rowspan="2">Excepciones</td><td>Paso</td><td>Acción</td></tr>
<tr><td>2</td><td>Si se produce algún problema al validar el email, se le notifica al usuario y el caso de uso termina</td></tr>
<tr><td>Estabilidad</td><td colspan="2">Alta</td></tr>
<tr><td>Comentarios</td><td colspan="2">Ninguno</td></tr>
</table>

Tabla 40: Requisito funcional UC-0007

UC-0008	Crear *wallet*	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0003 Gestionar *Wallets*	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0003 *Wallet* UC-0003 Registrar usuario	
Descripción	El sistema debe de crear una nueva *wallet*	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	El sistema crea y almacena una nueva *wallet*
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	1	Si se produce algún problema al crear la *wallet*, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 41: Requisito funcional UC-0008

UC-0009	Eliminar *wallet*
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0003 Gestionar *Wallets*

Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0003 *Wallet* UC-0009 Eliminar usuario	
Descripción	El sistema debe comprobar que el email que ha introducido el usuario es suyo	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	El sistema recupera la *wallet* a eliminar
	2	El sistema elimina la *wallet*
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	1	Si no se encuentra la *wallet* a eliminar, se le notifica al usuario y el caso de uso termina
	2	Si se produce algún problema al eliminar la *wallet*, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 42: Requisito funcional UC-0009

UC-0010	**Recuperar *wallet***
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0003 Gestionar *Wallets*
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0001 Usuarios

	UC-0006 Recuperar usuario UC-0011 Realizar transacción *blockchain*	
Descripción	El sistema debe de comprobar que el email que ha introducido el usuario es suyo	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	El sistema recupera la *wallet*
	2	El sistema devuelve la *wallet*
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	1	Si se produce algún problema al recuperar la *wallet*, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 43: Requisito funcional UC-0010

5.3.2.2 Tokens

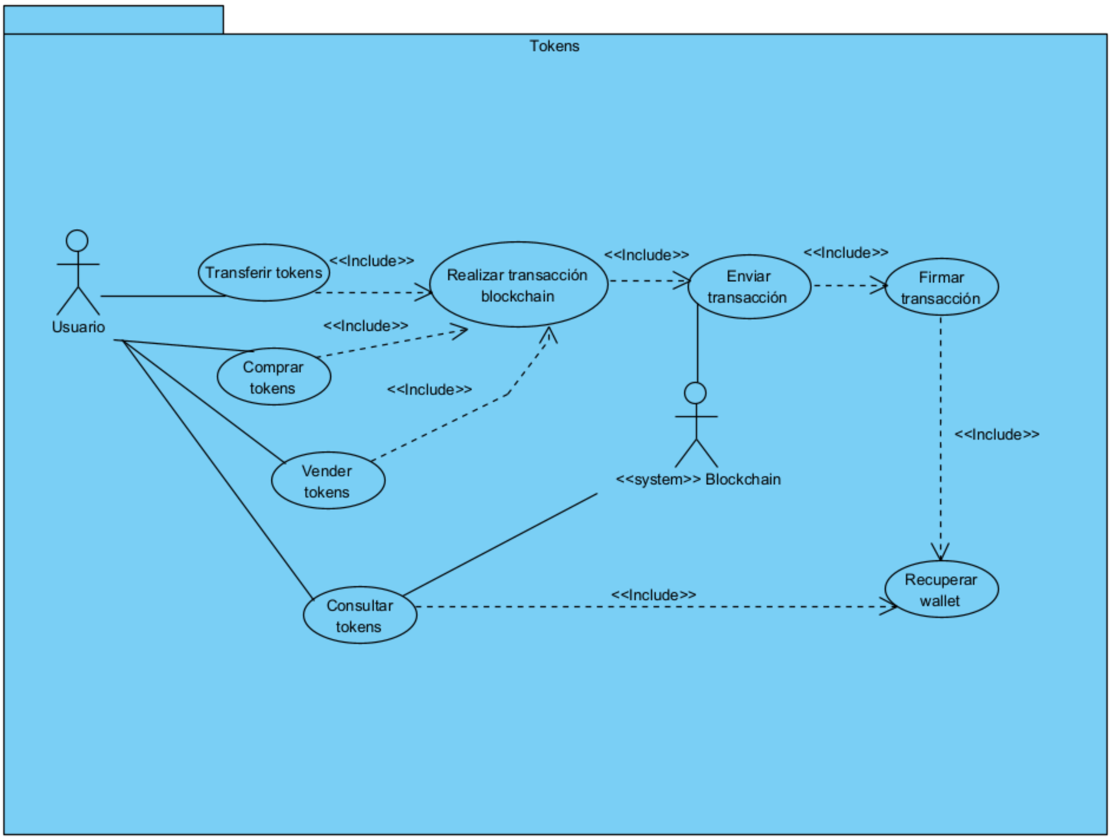

Figura 2: Casos de usos del paquete tokens

UC-0011	**Realizar transacción *blockchain***	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets*	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-0012 Firmar transacción UC-0013 Enviar transacción UC-0014 Transferir *tokens* UC-0015 Consultar *tokens* UC-0016 Comprar *tokens* UC-0017 Vender *tokens* UC-0018 Crear subasta UC-0019 Modificar subasta UC-0020 Eliminar subasta UC-0022 Realizar puja UC-0023 Congelar *tokens* UC-0024 Descongelar *tokens* UC-0025 Finalizar subasta	
Descripción	El sistema debe permitir la realización de transacciones con la cadena de bloques	
Precondición	-	
	Paso	**Acción**

Secuencia normal	1	El sistema recupera la transacción que debe incorporar a la cadena de bloques
	2	Se ejecuta el caso de uso ***UC-00013 Enviar transacción,*** recuperando la información de la transacción enviada
	3	El sistema devuelve la información de la transacción
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	1	Si se produce algún error al crear la transacción, se le notifica al usuario y el caso de uso termina
	2	Si se produce alguna excepción al ejecutar el caso de uso, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 44: Requisito funcional UC-0011

UC-00012	**Firmar transacción**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets*
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00011 Realizar transacción *blockchain*
Descripción	El sistema debe permitir firmar transacciones con la clave privada del usuario
Precondición	Debe haberse creado correctamente la transacción

Secuencia normal	Paso	Acción
	1	Se ejecuta el caso de uso ***UC-00010 Recuperar wallet,*** recuperando la *wallet* del usuario
	2	El sistema firma la transacción con la clave privada de la *wallet*
	3	El sistema devuelve la transacción firmada
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	1	Si se produce alguna excepción al ejecutar el caso de uso, se le notifica al usuario y el caso de uso termina
	2	Si se produce algún error al firmar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 45: Requisito funcional UC-0012

UC-00013	Enviar transacción
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets*
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción UC-00011 Realizar transacción *blockchain*
Descripción	El sistema debe permitir el envío de transacciones a la cadena de bloques
Precondición	-

Secuencia normal	**Paso**	**Acción**
	1	Se ejecuta el caso de uso ***UC-00012 firmar transacción,*** recuperando la transacción firmada
	2	El sistema envía la transacción firmada al actor **ACT-04 <<System>>** ***Blockchain***
	3	El actor **ACT-04 <<System>>** ***Blockchain*** devuelve el *hash* de la transacción y emite un evento
	4	El sistema devuelve el *hash* de la transacción
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	1	Si se produce alguna excepción al ejecutar el caso de uso, se le notifica al usuario y el caso de uso termina
	2	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 46: Requisito funcional UC-0013

UC-00014	**Transferir *tokens***
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens*
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios

	IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-00011 Realizar transacción *blockchain*	
Descripción	El sistema debe permitir la transferencia de *tokens* entre *wallets*	
Precondición	Que el usuario disponga de los *token*s que quiere enviar	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de transferir *token*s
	2	El usuario introduce la cantidad de *token*s a transferir
	3	El sistema comprueba que el usuario dispone de suficientes *token*s
	4	El sistema prepara la transacción y la añade a la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando el *hash* de la transacción
	5	El sistema confirma al usuario la transferencia
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	3	Si el usuario no dispone de *tokens* suficientes, se le notifica y el caso de uso termina
	4	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 47: Requisito funcional UC-0014

UC-00015	**Consultar *tokens***
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López

Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens*	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-00011 Realizar transacción *blockchain*	
Descripción	El sistema debe permitir consultar el balance de *token*s de una *wallet*	
Precondición	Que el usuario disponga de los *token*s que quiere enviar	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de consultar *token*s
	2	Se ejecuta el caso de uso ***UC-00010 Recuperar wallet,*** recuperando la *wallet* del usuario
	3	El sistema consulta el balance *token*s de la *wallet* del usuario al actor **ACT-04 <<System>> *Blockchain***
	4	El actor **ACT-04 <<System>> *Blockchain*** devuelve el balance de la *wallet*
	5	El sistema devuelve la información al usuario
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si se produce alguna excepción en la ejecución del caso de uso, se le notifica al usuario y el caso de uso termina

	3	Si se produce algún error al consultar el balance, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 48: Requisito funcional UC-0015

UC-00016	**Comprar *tokens***	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *tokens*	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-00011 Realizar transacción *blockchain*	
Descripción	El sistema debe permitir la compra de *tokens*	
Precondición	Que el usuario disponga de los *token*s que quiere enviar	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de comprar *token*s

	2	El usuario introduce la cantidad de *token*s a comprar
	3	El sistema cobra al usuario los *token*s
	4	El sistema prepara la transacción de añadir *token*s al usuario y la hace efectiva en la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando el hash de la transacción
	5	El sistema confirma al usuario la transferencia
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	3	Si el usuario no dispone de fondos suficientes, se le notifica y el caso de uso termina
	4	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 49: Requisito funcional UC-0016

UC-00017	**Vender *tokens***
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0004 Transferir *token*s
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0003 *Wallets*

	UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-00011 Realizar transacción *blockchain*	
Descripción	El sistema debe permitir la venta de *token*s	
Precondición	Que el usuario disponga de los *token*s que quiere vender	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de vender *token*s
	2	El usuario introduce la cantidad de *token*s a vender
	3	El sistema comprueba que el usuario dispone de suficientes *token*s
	4	El sistema prepara la transacción y la añade a la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando el hash de la transacción
	5	El sistema paga al usuario los *token*s vendidos
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	3	Si el usuario no dispone de *tokens* suficientes, se le notifica y el caso de uso termina
	4	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 50: Requisito funcional UC-0017

5.3.2.3 Subasta

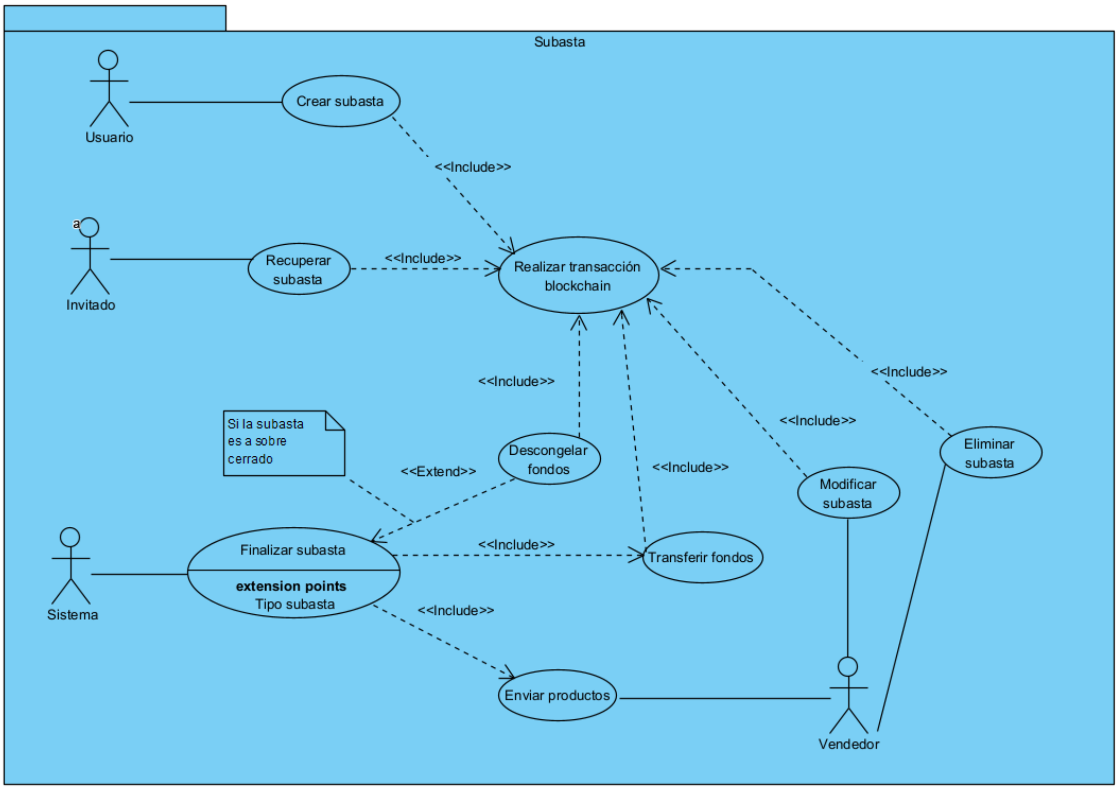

Figura 3: Casos de uso del paquete subasta

UC-0018	Crear subasta
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0005 Gestionar subastas
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras

	IRQ-0002 Subastas IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-00011 Realizar transacción *blockchain*	
Descripción	El sistema debe permitir la creación de subastas	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de crear nueva subasta
	2	El sistema solicita la información asociada al IRQ-0002 Subastas
	3	El usuario introduce la información solicitada
	4	El sistema comprueba la información
	5	El sistema almacena la información
	6	El sistema prepara la transacción y la añade a la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando el hash de la transacción
	7	El sistema notifica al usuario la creación de la subasta
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si la información introducida no es válida, se le notifica al usuario y se vuelve al paso 2
	5	Si el sistema no puede almacenar la información proporcionada, se notifica al usuario y el caso de uso termina
	6	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 51: Requisito funcional UC-0018

UC-0019	**Modificar subasta**

Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0005 Gestionar subastas	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0002 Subastas IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-00011 Realizar transacción *blockchain*	
Descripción	El sistema debe permitir la modificación de subastas	
Precondición	Que la modificación esté dentro del plazo establecido	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-03 (*vendedor*) selecciona la opción de modificar una subasta
	2	El usuario selecciona, entre sus subastas, la que quiere modificar
	3	El sistema solicita la nueva información
	3	El usuario introduce la información solicitada
	4	El sistema comprueba la información
	5	El sistema almacena la información
	6	El sistema prepara la transacción y la añade a la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando el *hash* de la transacción

	7	El sistema notifica al usuario la modificación de la subasta
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si la información introducida no es válida, se le notifica al usuario y se vuelve al paso 2
	5	Si el sistema no puede almacenar la información proporcionada, se notifica al usuario y el caso de uso termina
	6	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 52: Requisito funcional UC-0019

UC-0020	**Eliminar subasta**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0005 Gestionar subastas
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0002 Subastas IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción*

	UC-00011 Realizar transacción *blockchain*	
Descripción	El sistema debe permitir la eliminación de subastas	
Precondición	Que la eliminación esté dentro del plazo establecido	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-03 (*vendedor*) selecciona la opción de eliminar una subasta
	2	El usuario selecciona, entre sus subastas, la que quiere eliminar
	3	El sistema solicita confirmación
	4	El usuario confirma la eliminación
	5	El sistema elimina la información
	6	El sistema prepara la transacción y la añade a la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando el hash de la transacción
	7	El sistema notifica al usuario la eliminación de la subasta
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si el usuario no confirma la eliminación, el caso de uso termina
	5	Si el sistema no puede eliminar la subasta, se notifica al usuario y el caso de uso termina
	6	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 53: Requisito funcional UC-0020

UC-0021	**Recuperar subasta**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets*

	OBJ-0005 Gestionar subastas	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0002 Subastas IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-00011 Realizar transacción *blockchain*	
Descripción	El sistema debe permitir la visualización de las subastas	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-01 (*invitado*) selecciona la opción de ver una subasta
	2	El usuario selecciona la subasta que quiere consultar
	3	El sistema solicita la información a la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando los datos de la subasta
	4	El sistema muestra la información de la subasta
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	3	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 54: Requisito funcional UC-0021

5.3.2.4 Pujas

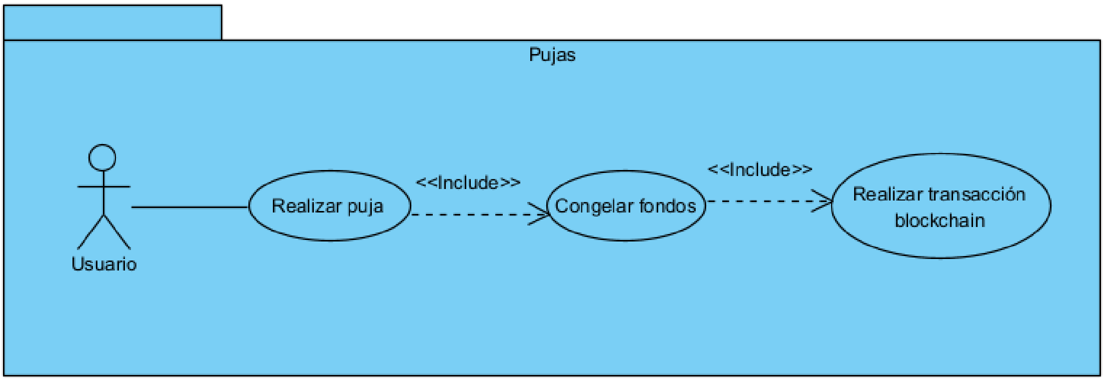

Figura 4> Casos de uso del paquete pujas

UC-0022	**Realizar puja**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0007 Gestionar pujas
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0003 *Wallets* IRQ-0004 Pujas UC-0009 Recuperar *wallet* UC-0011 Realizar transacción *blockchain* UC-0012 Firmar transacción *UC-0013 Enviar transacción* UC-0023 Congelar *tokens*

Descripción	El sistema debe permitir la realización de pujas	
Precondición	El usuario dispone de fondos suficientes para la realización de la puja	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-02 (*usuario*) selecciona la opción de pujar
	2	El sistema solicita el importe a pujar
	3	El usuario introduce los datos solicitados
	4	Se ejecuta el caso de uso ***UC-00023 Congelar tokens,*** congelando los tokens del usuario
	5	El sistema prepara la transacción de puja y la registra en *blockchain* ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando los datos de la subasta
	6	El sistema confirma la puja al usuario
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si se produce algún error en la ejecución del caso de uso, se le notifica al usuario y el caso de uso termina
	5	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 55: Requisito funcional UC-0022

UC-0023	**Congelar *tokens***	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0007 Gestionar pujas	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-0011 Realizar transacción *blockchain* UC-0022 Realizar puja	
Descripción	El sistema debe permitir la congelación de *tokens*	
Precondición	El usuario dispone de fondos suficientes	
Secuencia normal	**Paso**	**Acción**
	1	El sistema prepara la transacción para bloquear los tokens
	2	El sistema registra la operación en la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando el *hash* de la transacción
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 56: Requisito funcional UC-0023

UC-0024	**Descongelar *tokens***	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0007 Gestionar pujas	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0003 *Wallets* UC-0009 Recuperar *wallet* UC-00012 Firmar transacción *UC-0013 Enviar transacción* UC-0011 Realizar transacción *blockchain* UC-0025 Finalizar subasta	
Descripción	El sistema debe permitir la descongelación de *tokens*	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	El sistema prepara la transacción para desbloquear los tokens
	2	El sistema registra la operación en la cadena de bloques ejecutando el caso de uso ***UC-00011 Realizar transacción blockchain,*** recuperando el *hash* de la transacción
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si se produce algún error al enviar la transacción, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 57: Requisito funcional UC-0024

<table>
<tr><th>UC-0025</th><th colspan="2">Finalizar subasta</th></tr>
<tr><td>Versión</td><td colspan="2">1.0</td></tr>
<tr><td>Autores</td><td colspan="2">Carlos Álvarez López</td></tr>
<tr><td>Fuentes</td><td colspan="2">Carlos Álvarez López</td></tr>
<tr><td>Objetivos asociados</td><td colspan="2">OBJ-0002 Sistema blockchain
OBJ-0003 Gestionar Wallets
OBJ-0005 Gestionar subastas</td></tr>
<tr><td>Requisitos asociados</td><td colspan="2">NFR-0001 Privacidad de la información
NFR-0004 Tiempo de respuesta
NFR-0007 Conexiones seguras
IRQ-0002 Subastas
IRQ-0003 Wallets
IRQ-0004 Pujas
UC-0009 Recuperar wallet
UC-00012 Firmar transacción
UC-0013 Enviar transacción
UC-00011 Realizar transacción blockchain
UC-0014 Transferir tokens
UC-0024 Descongelar tokens</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema debe permitir la finalización de subastas</td></tr>
<tr><td>Precondición</td><td colspan="2">-</td></tr>
<tr><td rowspan="5">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El sistema comienza el caso de uso de manera automática cuando hay que finalizar una subasta</td></tr>
<tr><td>2</td><td>En caso de que sea una subasta a sobre cerrado, el sistema desbloquea los fondos de los pujadores no ganadores, para ello ejecuta el UC-0024 Descongelar tokens</td></tr>
<tr><td>3</td><td>El sistema transfiere los fondos de la cuenta del comprador a la cuenta del vendedor, para ello utiliza el UC-0014 Transferir tokens</td></tr>
<tr><td>4</td><td>El sistema finaliza la subasta</td></tr>
<tr><td>Postcondición</td><td colspan="2">Ninguna</td></tr>
</table>

Excepciones	**Paso**	**Acción**
	2	Si se produce algún error al ejecutar el caso de uso, el caso de uso termina
	3	Si se produce algún error al ejecutar el caso de uso, el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 58: Requisito funcional UC-0025

UC-0026	**Registrar envío**	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0005 Gestionar subastas	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0002 Subastas	
Descripción	El sistema debe permitir registrar los envíos	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-03 (*vendedor*) selecciona la opción de registrar envío
	2	El usuario selecciona, entre sus subastas finalizadas, la que se corresponde con el envío
	3	El sistema solicita el número de seguimiento

	4	El usuario facilita los datos solicitados
	5	El sistema registra el envío del producto
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si se produce algún al recuperar los datos solicitados, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 59: Requisito funcional UC-00026

UC-0027	**Modificar envío**	
Versión	1.0	
Autores	Carlos Álvarez López	
Fuentes	Carlos Álvarez López	
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0003 Gestionar *Wallets* OBJ-0005 Gestionar subastas	
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0003 Tiempo de aprendizaje NFR-0004 Tiempo de respuesta NFR-0005 Concurrencia de vista NFR-0006 Interfaces gráficas NFR-0007 Conexiones seguras IRQ-0002 Subastas	
Descripción	El sistema debe permitir modificar los envíos	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	Un usuario de tipo ACT-03 (*vendedor*) selecciona la opción de modificar envío
	2	El usuario selecciona, entre sus subastas finalizadas, la que se corresponde con el envío a modificar

	3	El sistema solicita el nuevo número de seguimiento
	4	El usuario facilita los datos solicitados
	5	El sistema registra la modificación del envío del producto
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	4	Si se produce algún al recuperar los datos solicitados, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 60: Requisito funcional UC-0027

5.3.2.5 Notificaciones

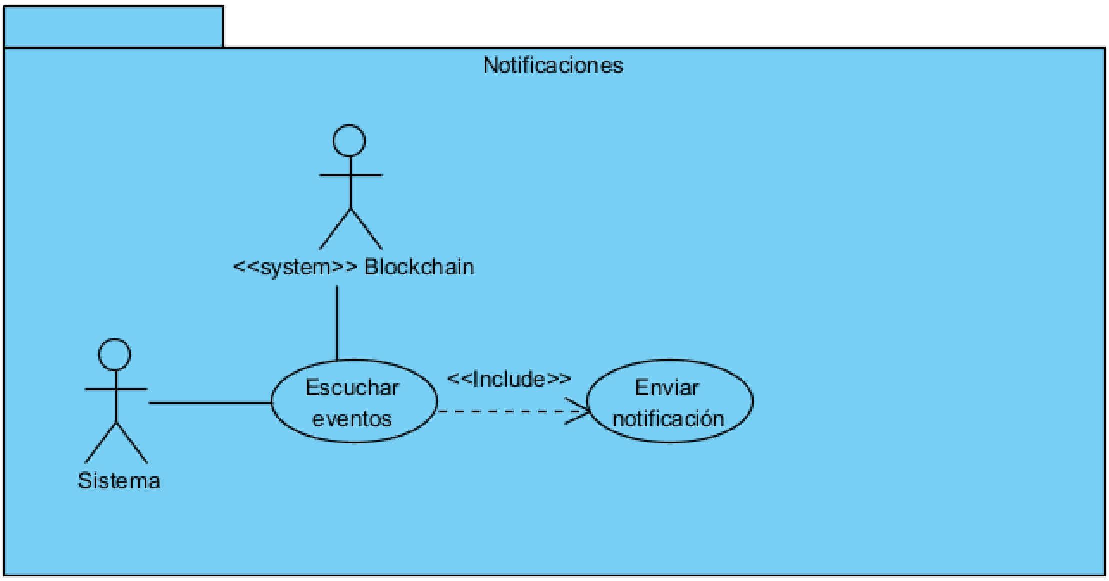

Figura 5: Casos de uso del paquete notificaciones

UC-0027	**Enviar notificación**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0008 Enviar notificaciones

Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0001 Usuarios IRQ-0005 Notificaciones UC-0029 Escuchar eventos	
Descripción	El sistema debe permitir enviar notificaciones a los usuarios	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	El sistema almacena la notificación
	2	El sistema envía por correo electrónico la notificación al usuario
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si se produce algún al enviar la notificación, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 61: Requisito funcional UC-0028

UC-0029	**Escuchar eventos**
Versión	1.0
Autores	Carlos Álvarez López
Fuentes	Carlos Álvarez López
Objetivos asociados	OBJ-0002 Sistema *blockchain* OBJ-0008 Enviar notificaciones
Requisitos asociados	NFR-0001 Privacidad de la información NFR-0004 Tiempo de respuesta NFR-0007 Conexiones seguras IRQ-0005 Notificaciones

	UC-0028 Enviar notificaciones	
Descripción	El sistema debe permitir modificar los envíos	
Precondición	-	
Secuencia normal	**Paso**	**Acción**
	1	El actor **ACT-04 <<System>> *Blockchain*** emite un nuevo evento
	2	El sistema escucha el evento
	3	El sistema genera y almacena una nueva notificación
	4	El sistema envía la notificación a los usuarios, para ello ejecuta ***UC-0028 enviar notificación***
Postcondición	Ninguna	
Excepciones	**Paso**	**Acción**
	2	Si se produce algún error al escuchar los eventos, se le notifica al usuario y el caso de uso termina
Estabilidad	Alta	
Comentarios	Ninguno	

Tabla 62: Requisito funcional UC-0029

6 Matrices de trazabilidad

6.1 Matriz de trazabilidad de objetivos

	OBJ-0001	OBJ-0002	OBJ-0003	OBJ-0004	OBJ-0005	OBJ-0006	OBJ-0007	OBJ-0008
UC-0001	X	-	-	-	-	-	-	-
UC-0002	X	-	-	-	-	-	-	-
UC-0003	X	-	X	-	-	-	-	-
UC-0004	X	-	-	-	-	-	-	-
UC-0005	X	-	X	-	-	-	-	-
UC-0006	X	-	X	-	-	-	-	-
UC-0007	X	-	-	-	-	-	-	-
UC-0008	-	-	X	-	-	-	-	-
UC-0009	-	-	X	-	-	-	-	-
UC-0010	-	-	X	-	-	-	-	-
UC-0011	-	X	X	-	-	-	-	-
UC-0012	-	X	X	-	-	-	-	-
UC-0013	-	X	X	-	-	-	-	-
UC-0014	-	x	X	X	-	-	-	-
UC-0015	-	X	X	X	-	-	-	-
UC-0016	-	X	X	X	-	-	-	-
UC-0017	-	X	X	X	-	-	-	-
UC-0018	-	X	X	-	X	-	-	-
UC-0019	-	X	X	-	X	-	-	-
UC-0020	-	X	X	-	X	-	-	-

UC-0021	-	X	X	-	X	-	-	-
UC-0022	-	X	X	-	-	-	X	-
UC-0023	-	X	X	-	-	-	X	-
UC-0024	-	X	X	-	-	-	X	-
UC-0025	-	X	X	-	X	-	-	-
UC-0026	-	X	X	-	X	-	-	-
UC-0027	-	X	X	-	X	-	-	-
UC-0028	-	X	-	-	-	-	-	X
UC-0029	-	X	-	-	-	-	-	X

Tabla 63: Matriz de trazabilidad de objetos

6.2 Matriz de trazabilidad requisitos no funcionales

	NRF-0001	**NRF-0002**	**NRF-0003**	**NRF-0004**	**NRF-0005**	**NRF-0006**	**NRF-0007**
UC-0001	X	-	X	X	X	X	X
UC-0002	X	-	X	X	X	X	X
UC-0003	X	-	X	X	X	X	X
UC-0004	X	-	X	X	X	X	X
UC-0005	X	-	X	X	X	X	X
UC-0006	X	-	X	X	X	X	X
UC-0007	X	-	X	X	X	X	X
UC-0008	X	-	-	X	-	-	X
UC-0009	X	-	-	X	-	-	X
UC-0010	X	-	-	X	-	-	X
UC-0011	X	-	-	X	-	-	X
UC-0012	X	-	-	X	-	-	X
UC-0013	X	-	-	X	-	-	X
UC-0014	X	-	X	X	X	X	X
UC-0015	X	-	X	X	X	X	X
UC-0016	X	-	X	X	X	X	X
UC-0017	X	-	X	X	X	X	X
UC-0018	X	-	X	X	X	X	X
UC-0019	X	-	X	X	X	X	X
UC-0020	X	-	X	X	X	X	X
UC-0021	X	-	X	X	X	X	X
UC-0022	X	-	X	X	X	X	X
UC-0023	X	-	-	X	-	-	X

UC-0024	X	-	-	X	-	-	X
UC-0025	X	-	-	X	-	-	X
UC-0026	X	-	X	X	X	X	X
UC-0027	X	-	X	X	X	X	X
UC-0028	X	-	-	X	-	-	X
UC-0029	X	-	-	X	-	-	X

Tabla 64: Matriz de trazabilidad de requisitos no funcionales

6.3 Matriz de trazabilidad de requisitos de información

	IRQ-0001	IRQ-0002	IRQ-0003	IRQ-0004	IRQ-0005	IRQ-0006
UC-0001	X	-	-	-	-	X
UC-0002	X	-	-	-	-	X
UC-0003	X	-	X	-	-	-
UC-0004	X	-	-	-	-	-
UC-0005	X	-	X	-	-	-
UC-0006	X	-	X	-	-	-
UC-0007	X	-	-	-	-	-
UC-0008	-	-	X	-	-	-
UC-0009	-	-	X	-	-	-
UC-0010	-	-	X	-	-	-
UC-0011	X	-	X	-	-	-
UC-0012	X	-	X	-	-	-
UC-0013	X	-	X	-	-	-
UC-0014	X	-	X	-	-	-
UC-0015	X	-	X	-	-	-
UC-0016	X	-	X	-	-	-
UC-0017	X	-	X	-	-	-
UC-0018	-	X	X	-	-	-
UC-0019	-	X	X	-	-	-
UC-0020	-	X	X	-	-	-
UC-0021	-	X	X	-	-	-
UC-0022	-	-	X	X	-	-
UC-0023	-	-	X	-	-	-
UC-0024	-	-	X	-	-	-
UC-0025	-	X	X	X	-	-
UC-0026	-	X	-	-	-	-
UC-0027	-	X	-	-	-	-
UC-0028	X	-	-	-	X	-
UC-0029	-	-	-	-	X	-

Tabla 65: Matriz de trazabilidad de requisitos de información

TronAuction: plataforma de subastas online utilizando *Smart Contracts*

Anexo III: Especificación de diseño

Grado en ingeniería informática

Septiembre de 2021

Autor

Carlos Álvarez López

Tutores

Juan Manuel Corchado Rodríguez

Diego Javier Valdeolmillos Villaverde

Agustín San Román Guzmán

Tabla de contenido

Índice de figuras

1 Introducción

En el presente documento se presentará la especificación de diseño para el proyecto *TronAuction*: plataforma de subastas online utilizando *smart contracts*.

El modelo de diseño en un sistema *software* es una abstracción de la implementación del sistema. Esta abstracción se realiza en busca de un *software* de calidad. En la primera parte del documento se presentará el modelo arquitectónico utilizado, seguido del modelo de datos y de la vista de iteración. Finalmente se expondrá el modelo de despliegue.

2 Diseño arquitectónico

Mediante el diseño arquitectónico se define la estructura del sistema a implementar, así como su organización. Enlazando el modelo de requisitos con el diseño final de la aplicación. En este apartado se identificarán los principales componentes que componen la estructura del proyecto y la relación entre ellos.

2.1 Patrón modelo vista controlador

Figura 1: modelo vista controlador

Para la implementación del proyecto se ha utilizado el patrón arquitectónico MVC, cuyas siglas en inglés son: "*Model View Controller*". El MVC está englobado en la categoría de los patrones para sistemas interactivos, por ello es principalmente utilizado en aplicaciones destinadas a la interacción con el usuario.

Es una realidad que las interfaces de usuario están en constante cambio, ya bien sea para extender la funcionalidad de la aplicación o para adaptarse a un entorno gráfico totalmente distinto. Por ello, crear un sistema flexible capaz de adaptarse dinámicamente a los nuevos requisitos es muy costoso a la vez que lo convierte en un producto propenso a los fallos. Para dar solución a todos estos problemas, se diseñó el patrón MVC, el cual propone aislar la interfaz gráfica de la lógica y los datos de la aplicación. De modo que, si una de estas tres partes cambia, el resto de las partes no se verán afectadas. Para ello se dividirá el proyecto en las siguientes partes:

- **Modelo ("*Model*").** En él se encapsularán los datos almacenados por la aplicación, así como las funcionalidades relacionadas con estos.

- **Vista ("*View*").** Esta parte englobará las interfaces gráficas encargadas de presentar la información al usuario y de interactuar con el mismo.

- **Controlador ("*Controller*").** Se incluirán los elementos encargados de la lógica de control, haciendo de interfaz entre la vista y el controlador. Estos elementos realizarán todas las transformaciones de los datos necesarias para la correcta visualización. Además, se incluirán las funcionalidades propias de la aplicación.

Como el principal objetivo del presente proyecto es ofrecer a los usuarios una plataforma lo más sencilla e intuitiva posible, se ha elegido este patrón arquitectónico debido a propiedades orientadas a la interfaz de usuario, ayudando a realizar una aplicación web con una mayor flexibilidad de cara a futuras mejoras. Además, la estructuración del proyecto siguiendo los requisitos del MVC facilita la ordenación de los ficheros software de cara a una programación más intuitiva y sencilla.

2.2 Patrón DAO

El patrón arquitectónico DAO (*Data Access Object)* permite separar la lógica de acceso a datos de la lógica de negocio. Gracias a este desacoplamiento, es posible cambiar el acceso a los datos (por ejemplo, cambiar de una base de datos MySQL a una base de datos PostgreSQL) sin necesidad de modificar la lógica de negocio. El modelo DAO implementa los métodos necesarios para añadir, modificar, eliminar y consultar (métodos CRUD) la información almacenada en el motor de base de datos.

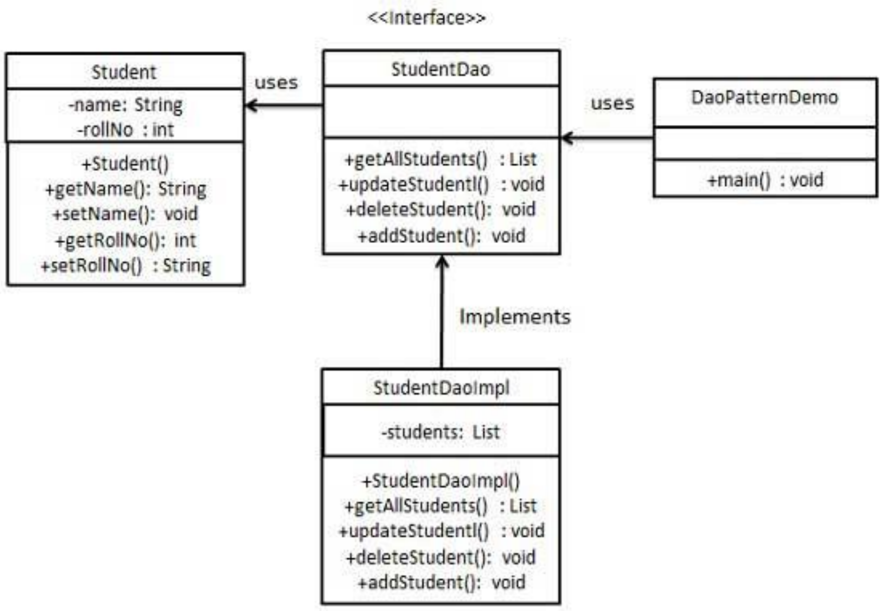

Figura 2: patrón DAO

En la figura número 2 se puede observar un ejemplo de aplicación del patrón DAO. La clase *Student* utiliza la interfaz del DAO (*StudentDAO),* para el acceso a los datos. Esta interfaz es implementada por la clase *StudentDaoImpl,* la cual incluye la lógica para el acceso a los datos, en este caso almacenados en *DaoPatternDemo.* Con esta implementación, se consigue que, aunque cambie el modelo de datos no se vea afectada la lógica de negocio, ya que esta estará haciendo uso de la interfaz y esta no cambiará, solo su implementación interna.

2.3 División de paquetes

A continuación, se muestra el diagrama de paquetes de la aplicación. A partir de este diagrama es posible visualizar de forma gráfica cuántos paquetes compondrán la aplicación, qué estructura siguen y cómo se relacionan entre sí.

El presente desarrollo sigue una arquitectura de capas basada en el patrón modelo vista controlador, el cual aporta un cierto grado de modularización, dividiendo el código en función de su utilidad.

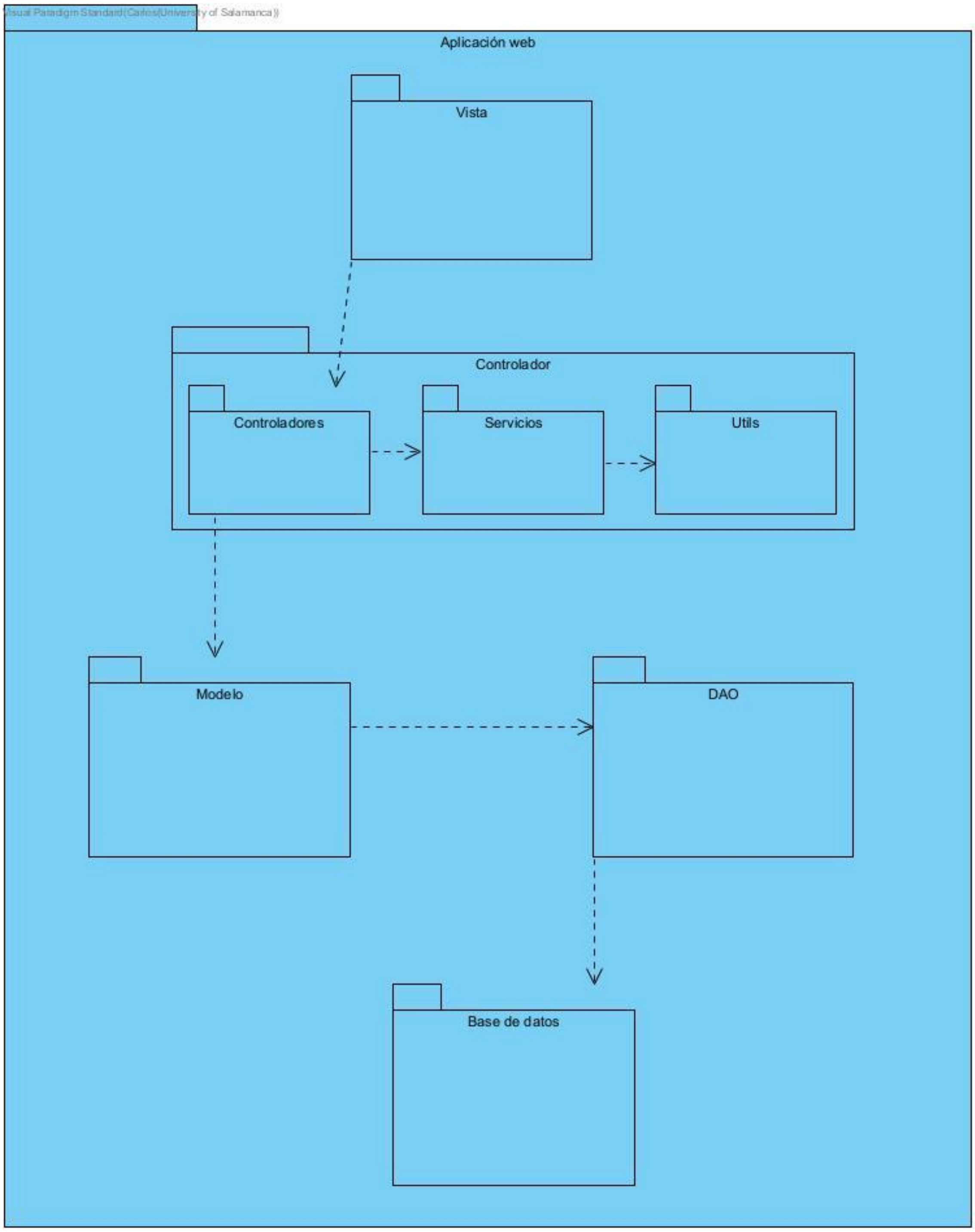

Figura 3: diagrama de paquetes

En la figura número 3, se puede observar la peculiaridad de que el paquete controlador está dividido en 3 subpaquetes, de este modo se separa la lógica propia de la aplicación que se encuentra en el paquete controladores, de la lógica genérica la cual podrá ser reutilizada en otros proyectos. Gracias a esta peculiaridad se consigue una aplicación con un alto grado de reusabilidad.

2.4 Diagrama de clases del sistema

El diagrama de clases del sistema permite modelar las distintas clases a utilizar, así como las relaciones existentes entre ellas. El diagrama de clases se expondrá dividido por los paquetes principales para facilitar su visualización. El paquete controlador al tener una gran extensión, primero se muestra completo y apaisado, pero a continuación se muestra dividido en tres partes para facilitar su visualización.

En el caso del paquete vista, no se realizará diagrama de clases al tratarse de vistas HTML, las cuales se renderizarán desde los distintos métodos de los controladores.

2.4.1 Paquete controlador

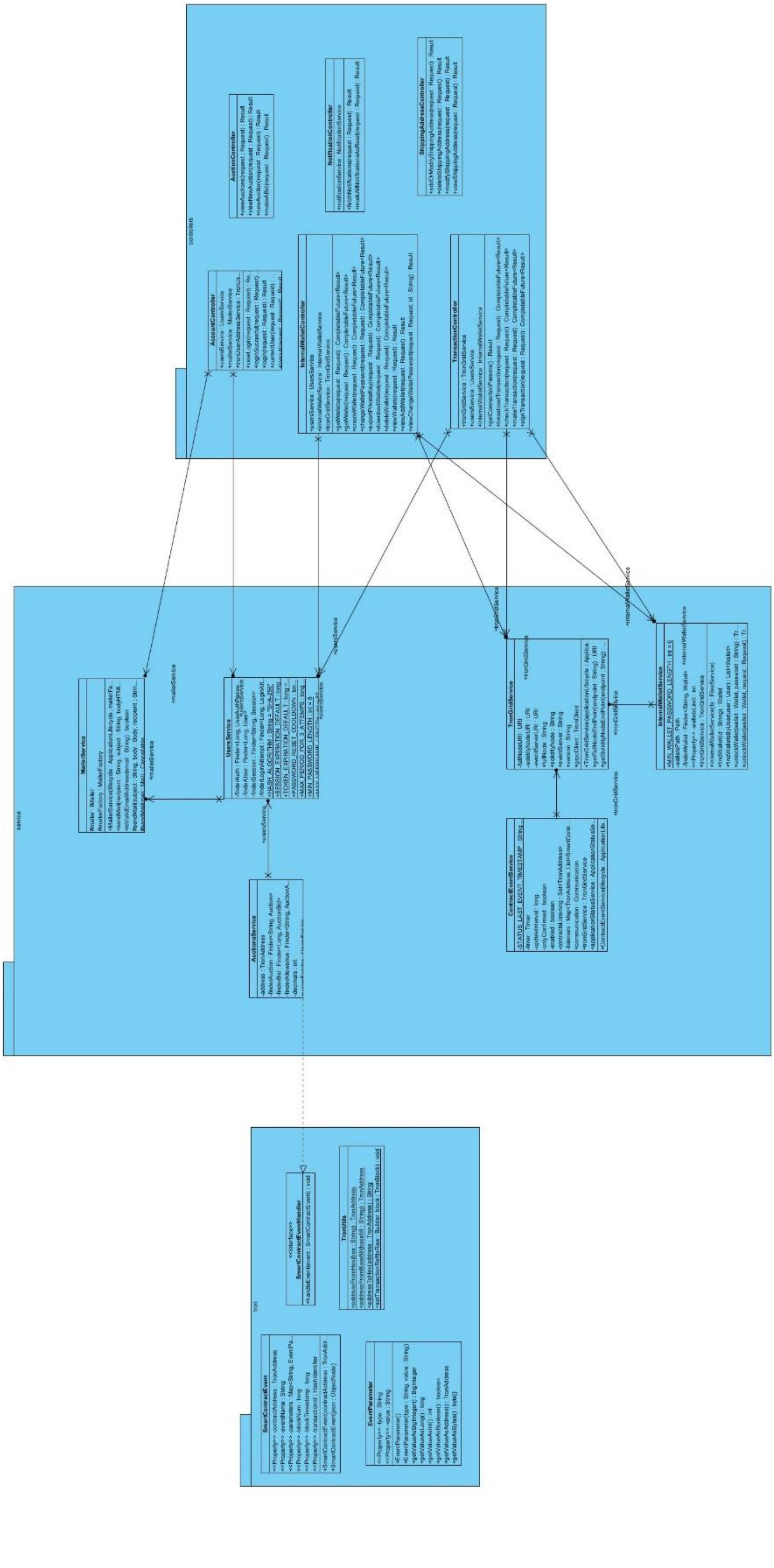

Figura 4: diagrama de clases del paquete controlador completo

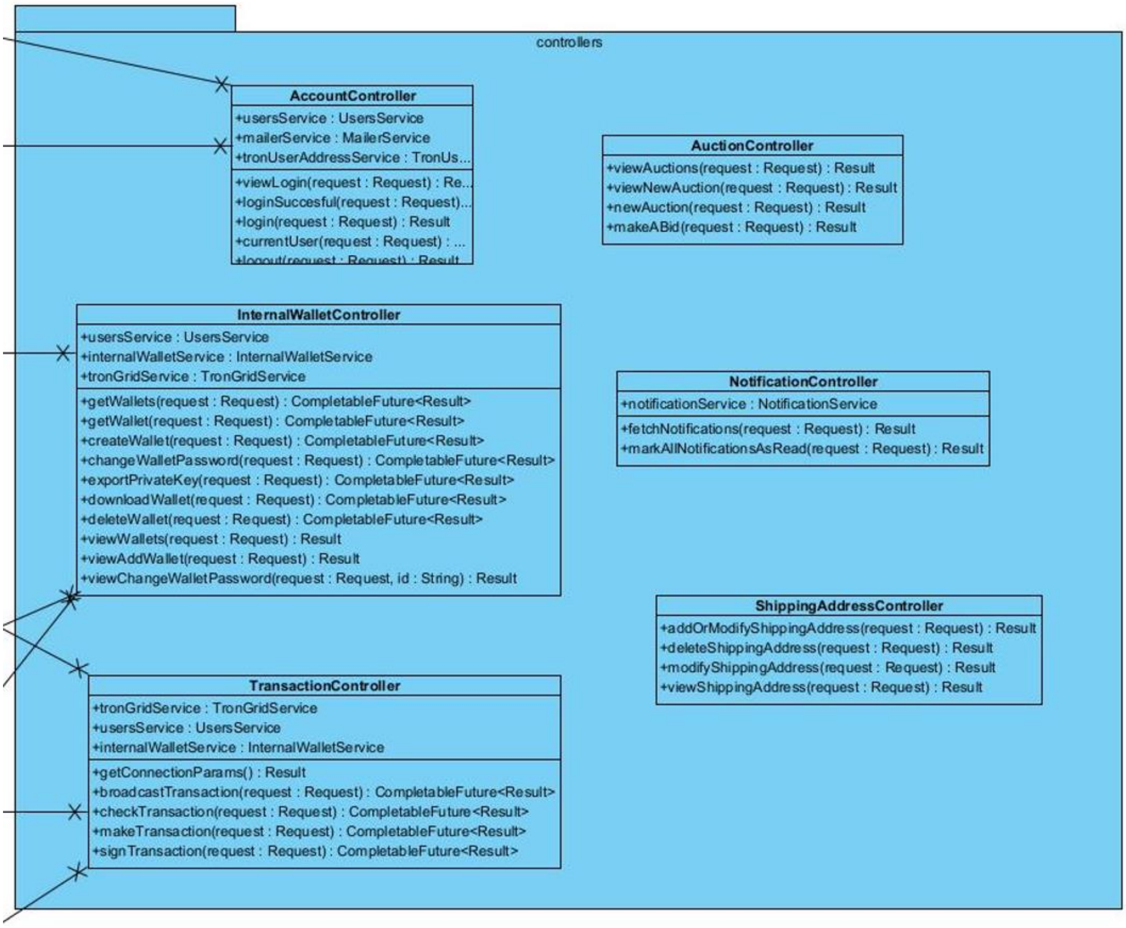

Figura 5:diagrama de clases del paquete controlador detallado (I)

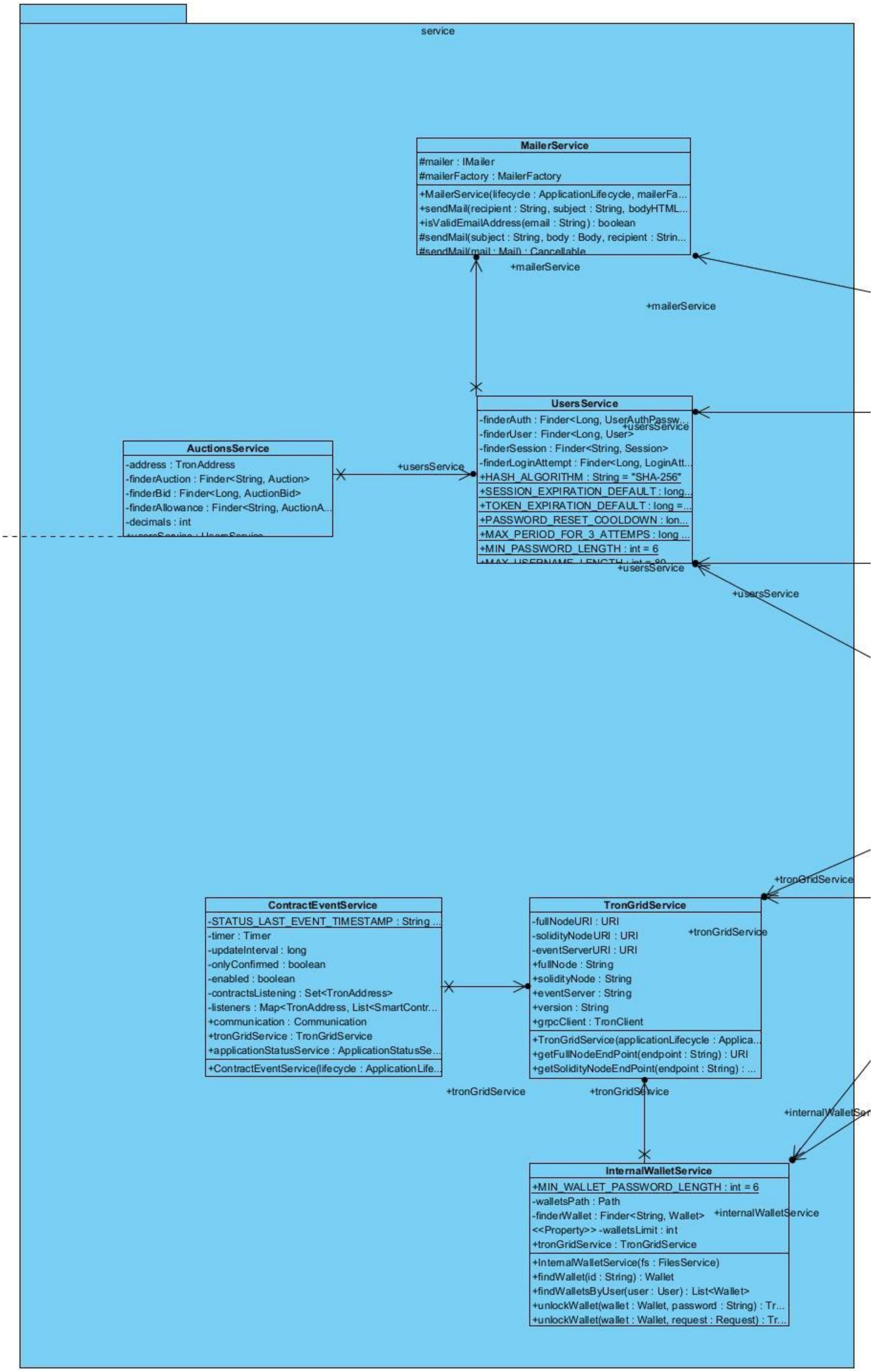

Figura 6: diagrama de clases del paquete controlador detallado (II)

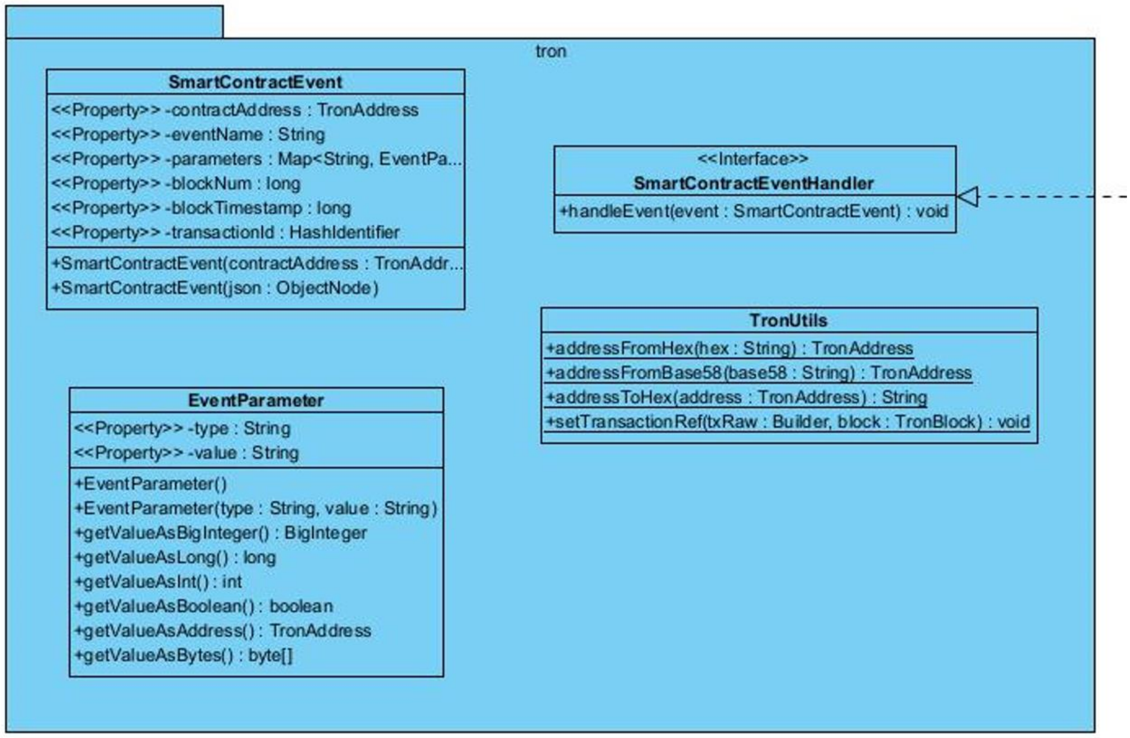

Figura 7: diagrama de clases del paquete controlador detallado (III)

2.4.2 Paquete modelo

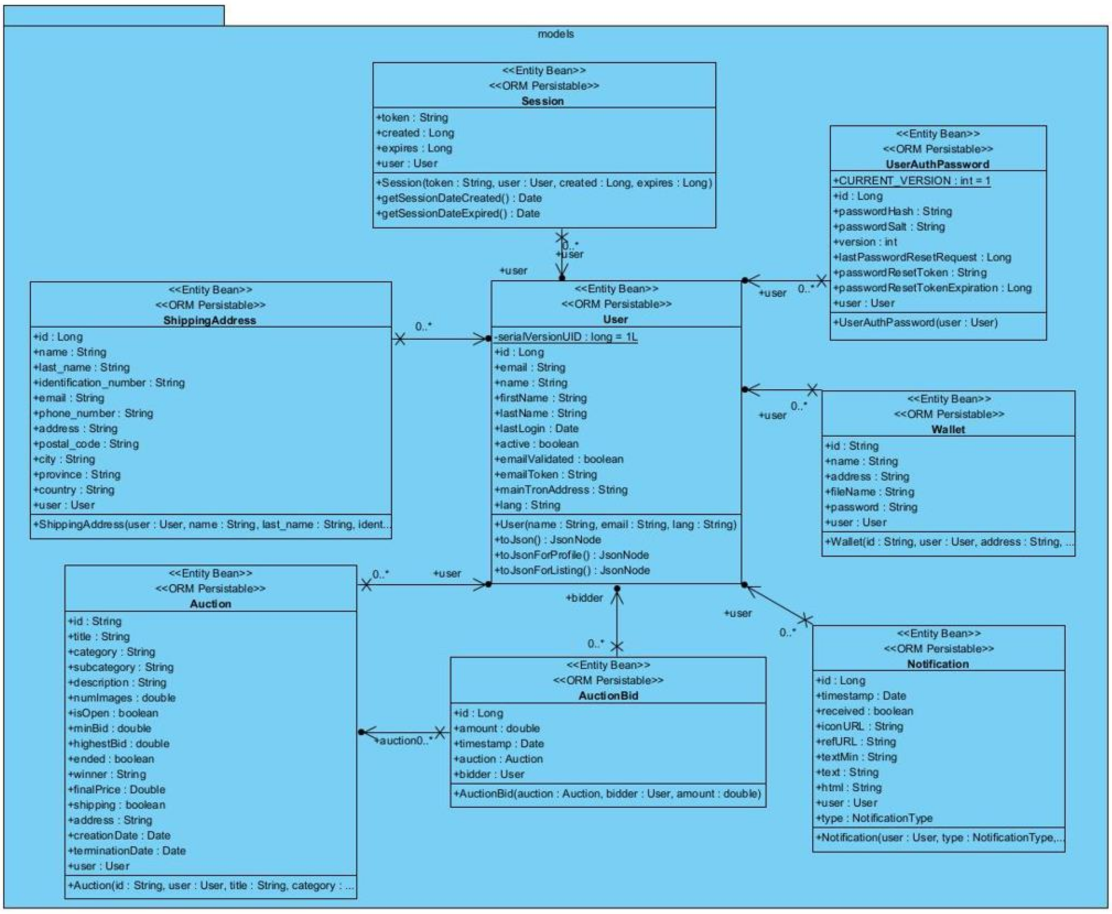

Figura 8:diagrama de clases del paquete modelo detallado

3 Diseño de datos

La información persistente de la aplicación, como, por ejemplo, los datos de los usuarios o la información de una subasta serán almacenados en una base de datos MySQL.

MySQL es un sistema de gestión de bases de datos relacionales desarrollado por Oracle y que se basa en SQL (lenguaje de consulta estructurado).

Las principales características de una base de datos relacional son:

- Almacena la información en una serie de tablas (relaciones) con un identificador único.
- Cada tabla se compone de filas (registros) y columnas (campos).
- Cada registro en una tabla debe de tener un campo único (clave primaria).
- Para relacionar dos tablas se hará mediante las claves primarias y foráneas.

En cuanto a MySQL, es el sistema gestor de bases de datos más utilizado hoy en día. Algunas de las ventajas que han hecho esto posible son: ser un software libre y gratuito, no requerir de un *hardware* de alto rendimiento para conseguir realizar operaciones en un tiempo aceptable o su alta seguridad ante accesos no deseados.

3.1 Diagrama entidad relación

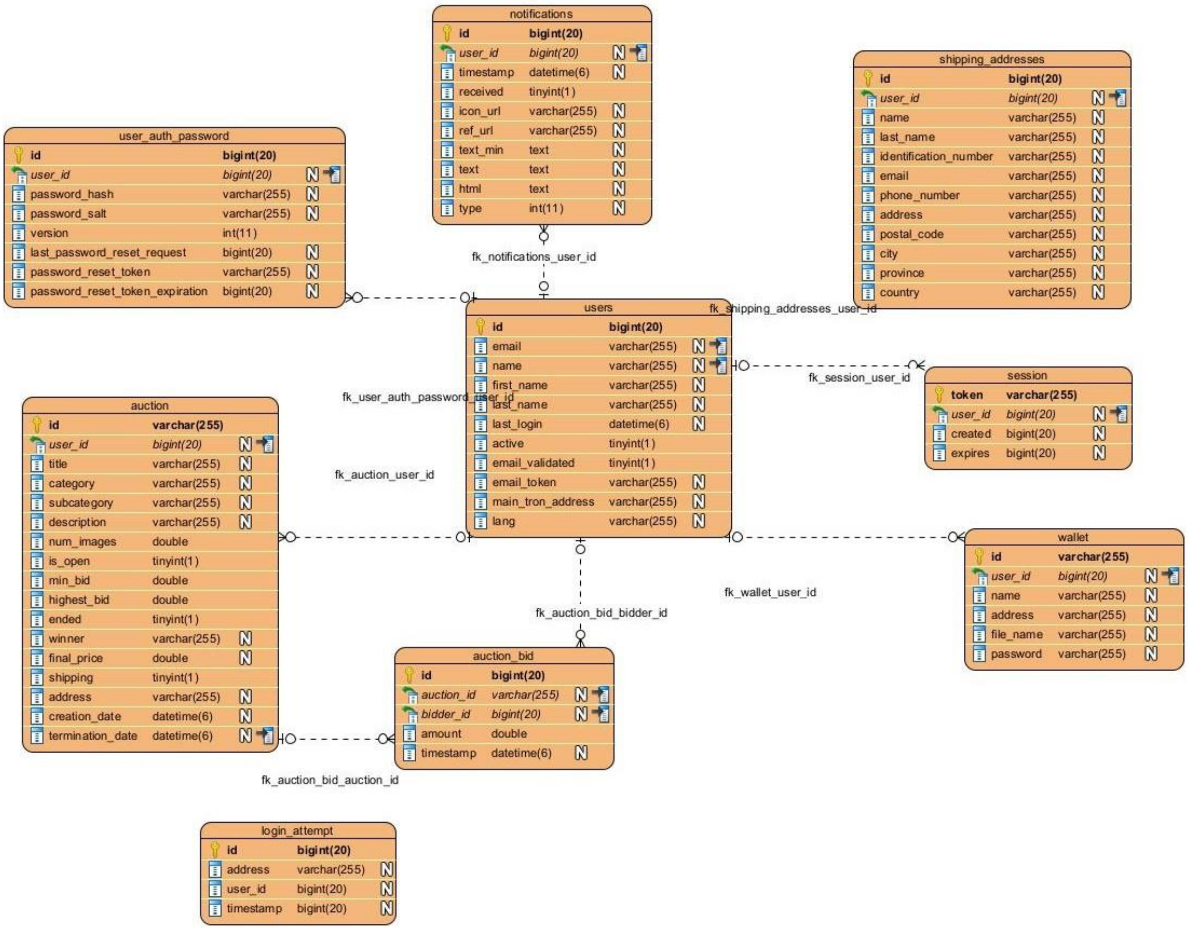

Figura 9: diagrama entidad relación

4 Vista de interacción

El principal objetivo de la vista de interacción es modelar el intercambio de información observable entre los elementos conectados. Este intercambio de información se realiza mediante mensajes.

4.1 Diagramas de secuencia

Los diagramas de secuencia se utilizan para mostrar la interacción entre dos objetos, centrándose en la secuencia de mensajes enviados. A continuación, se muestran los diagramas de secuencia de la aplicación.

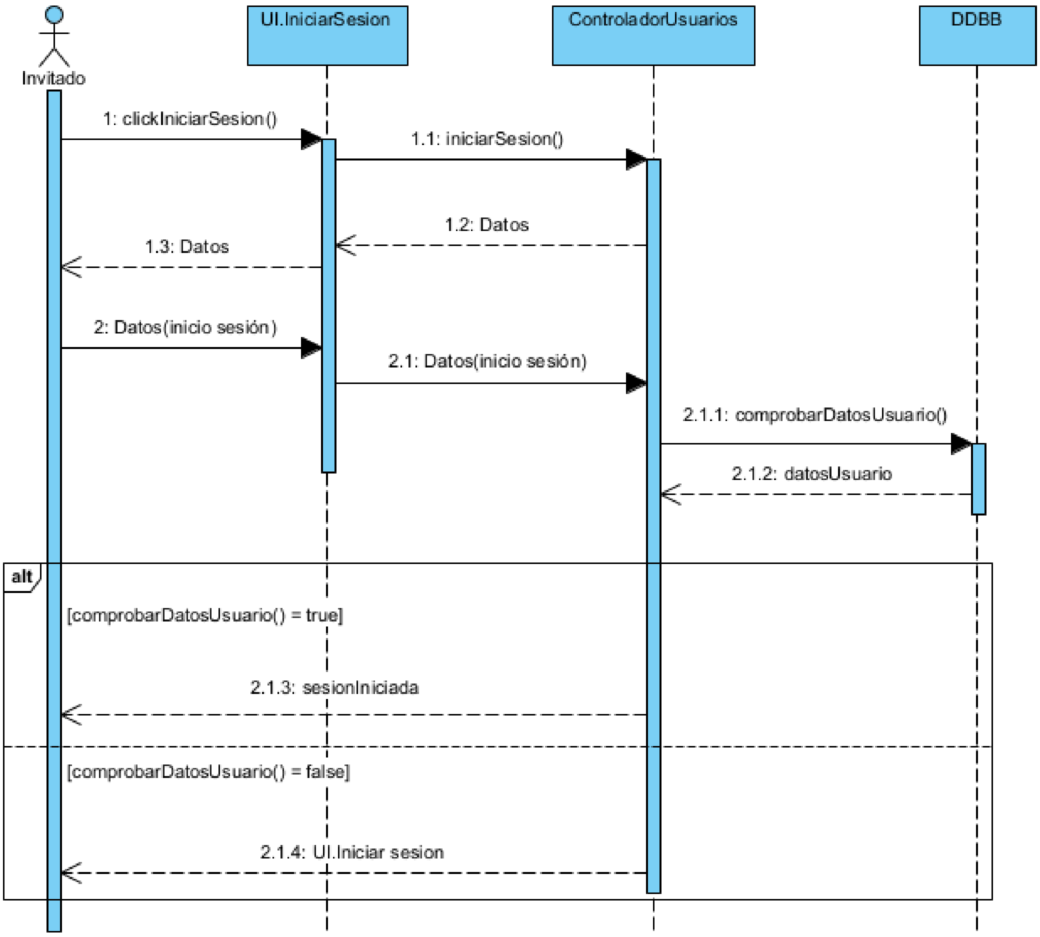

Figura 10. Diagrama de secuencia iniciar sesión

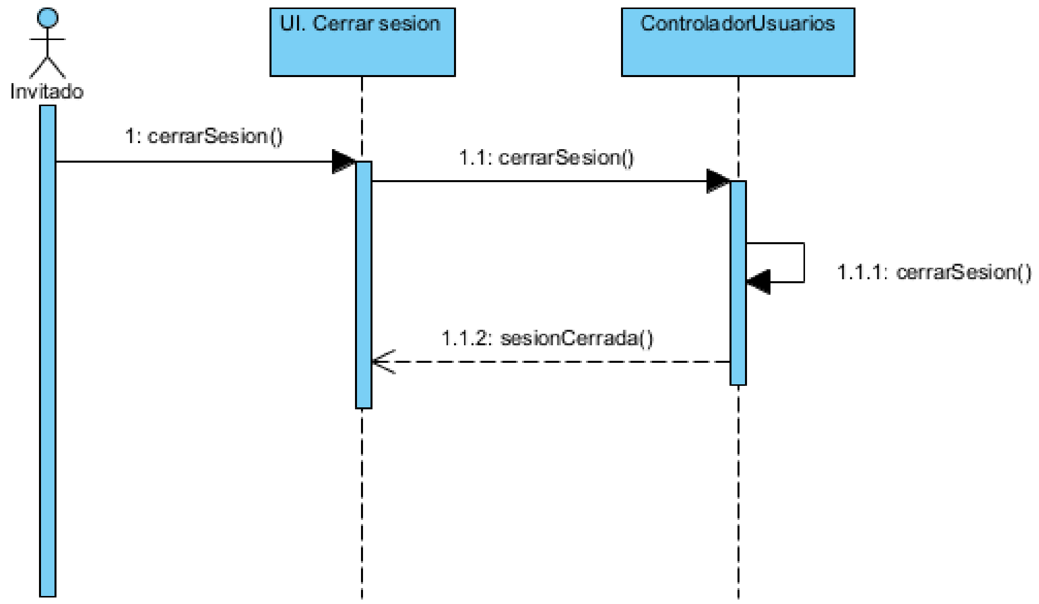

Figura 11. Diagrama de secuencia cerrar sesión

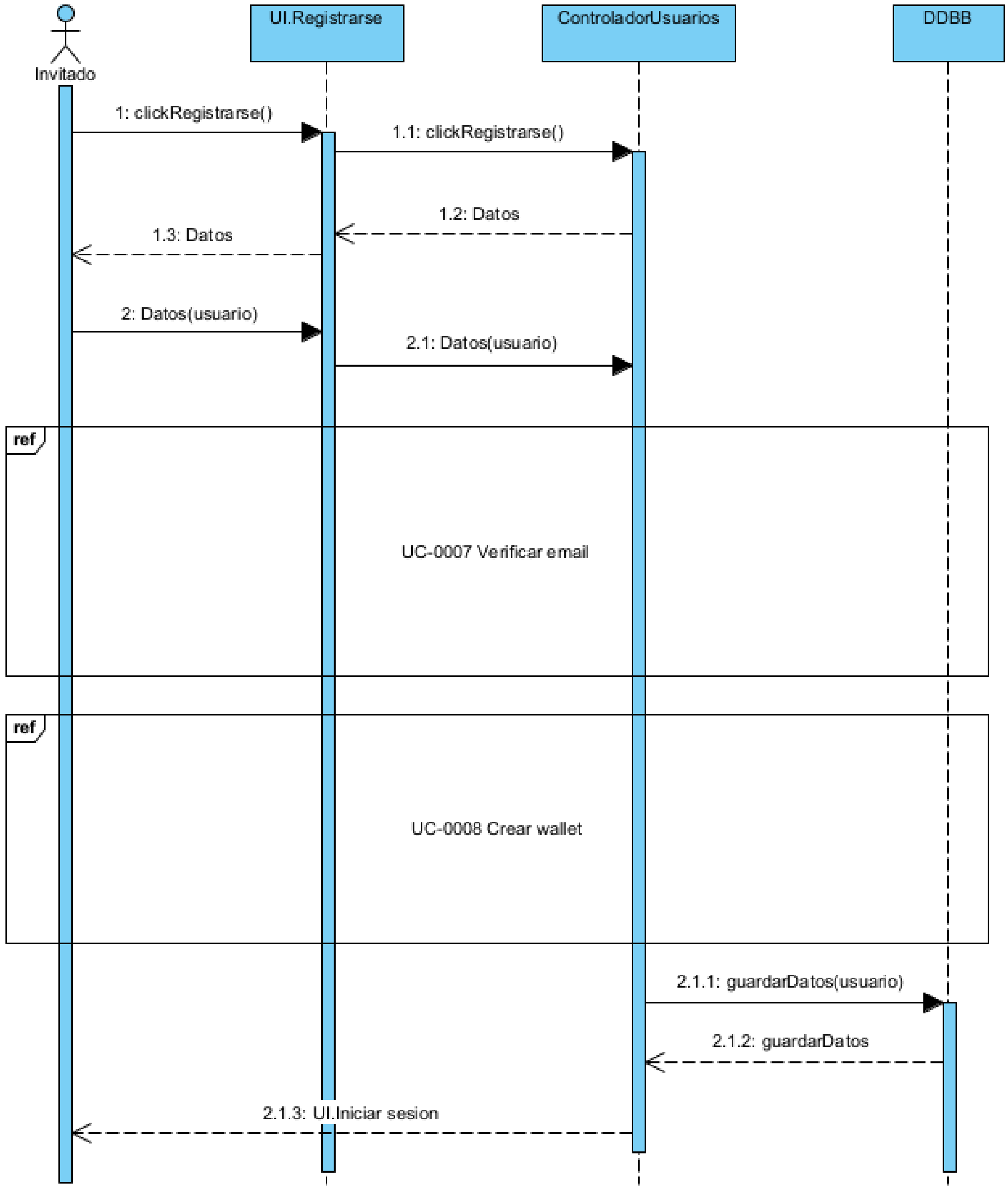

Figura 12. Diagrama de secuencia registrar usuario

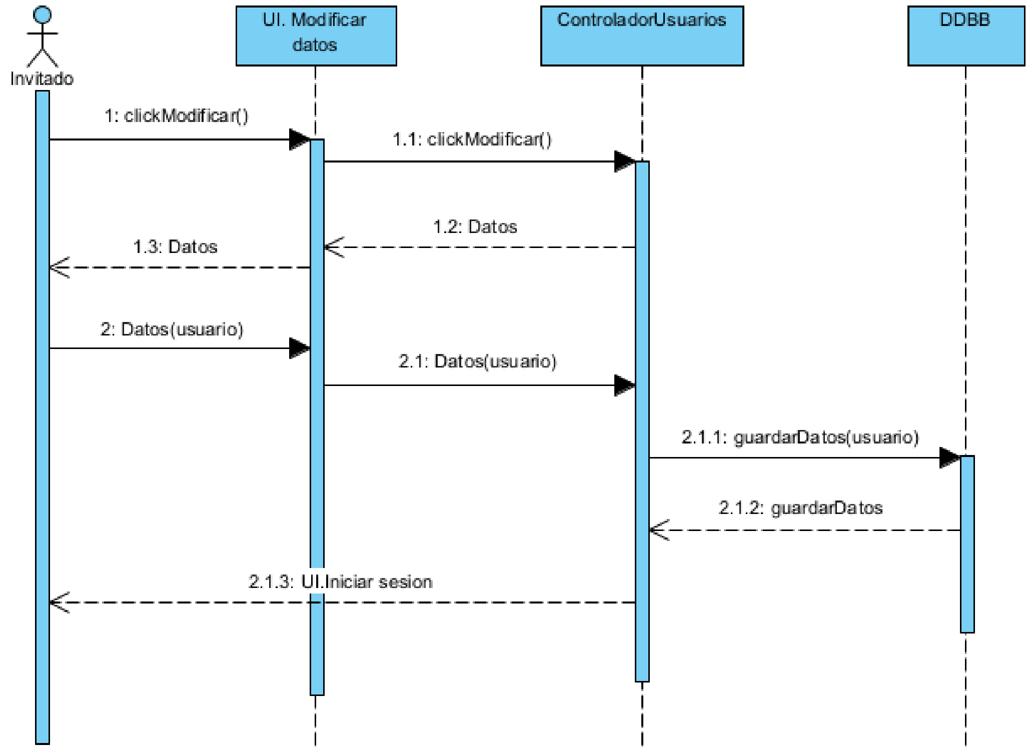

Figura 13. Diagrama de secuencia modificar usuario

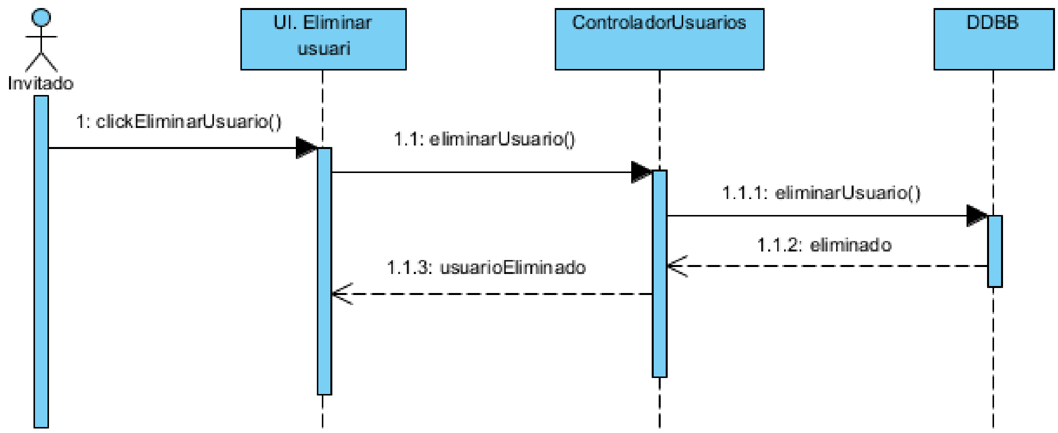

Figura 14. Diagrama de secuencia eliminar usuario

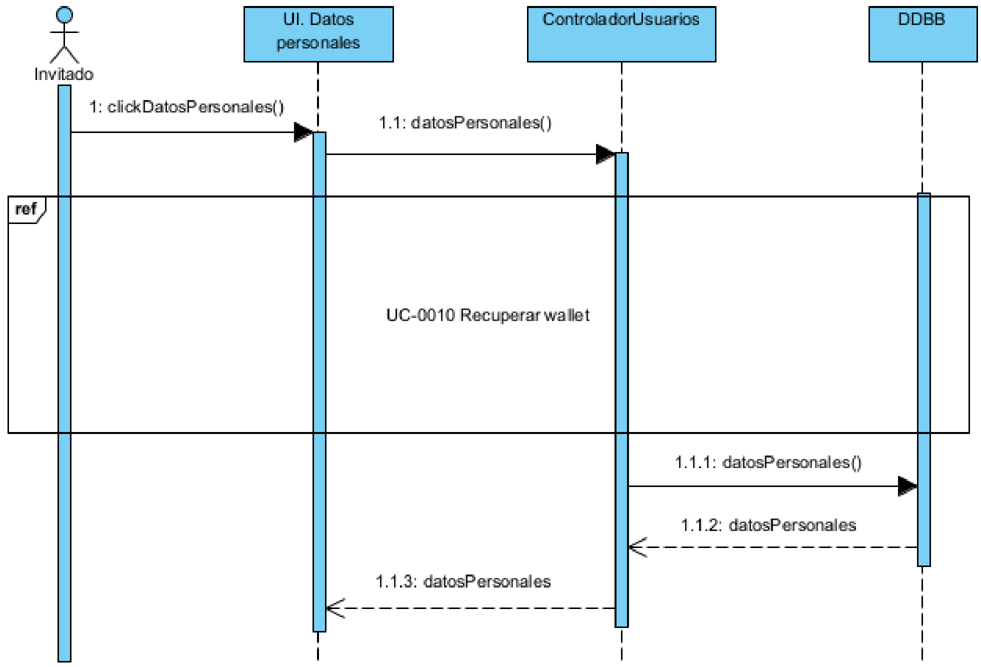

Figura 15. Diagrama de secuencia recuperar usuario

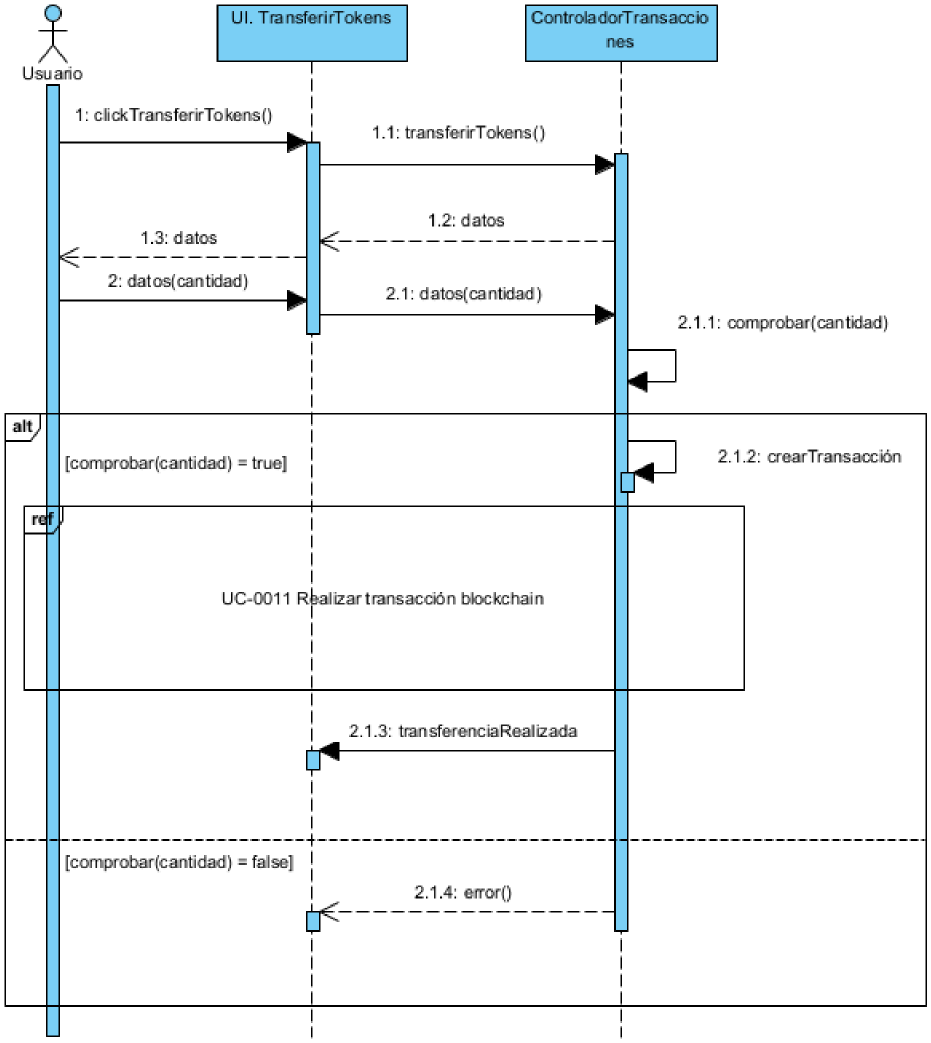

Figura 16. Diagrama de secuencia transferir tokens

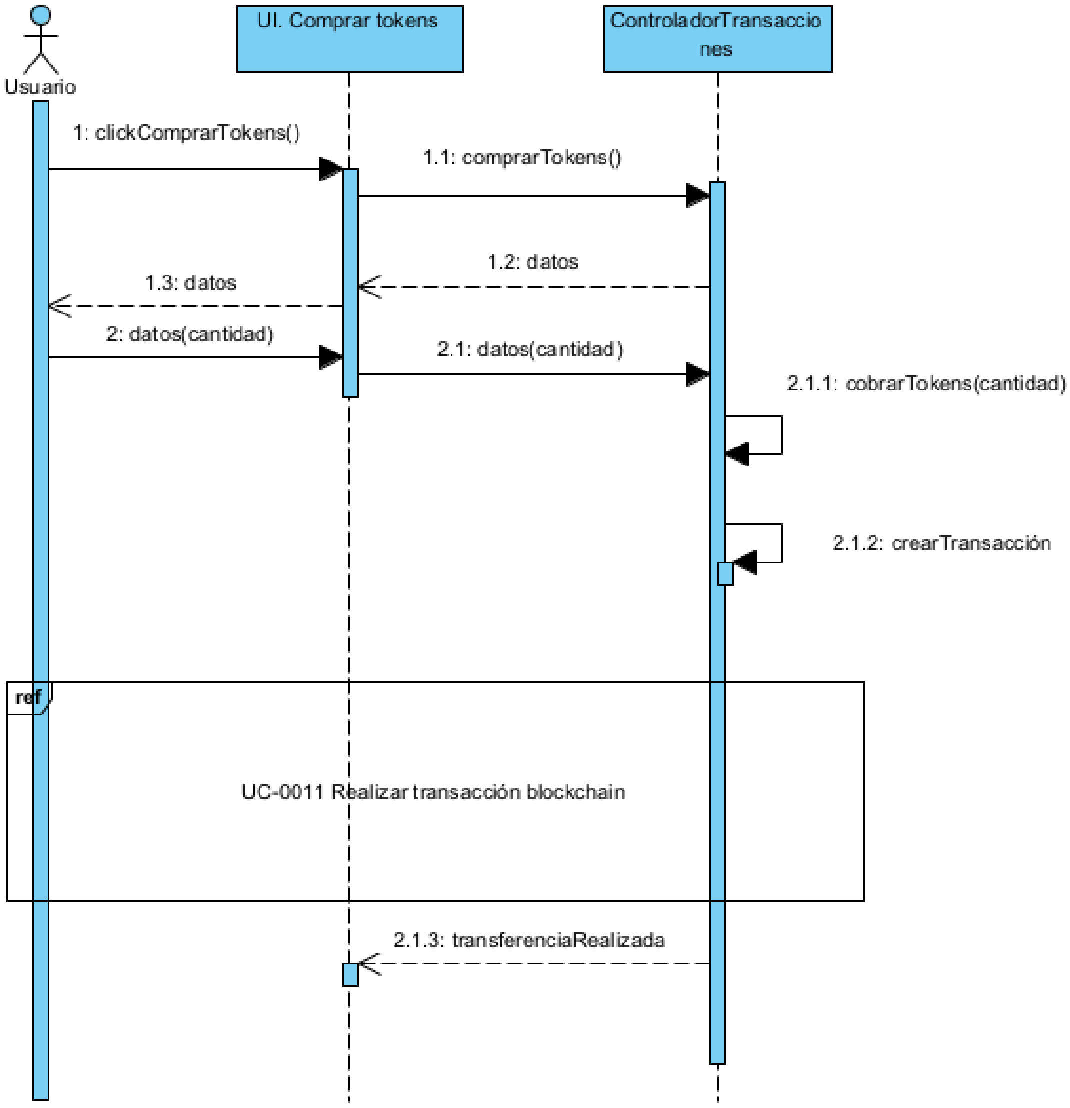

Figura 17. Diagrama de secuencia comprar tokens

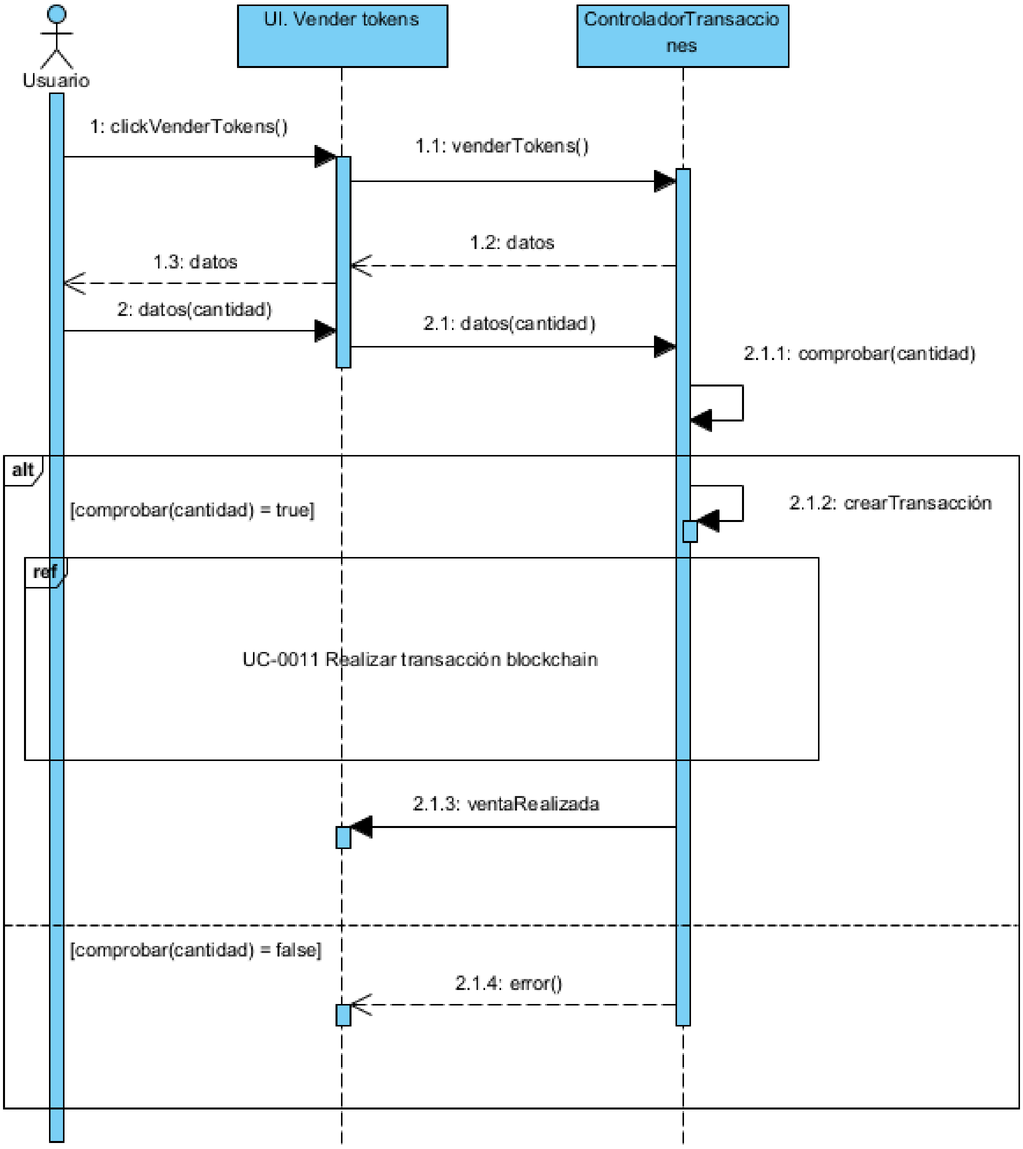

Figura 18. Diagrama de secuencia vender tokens

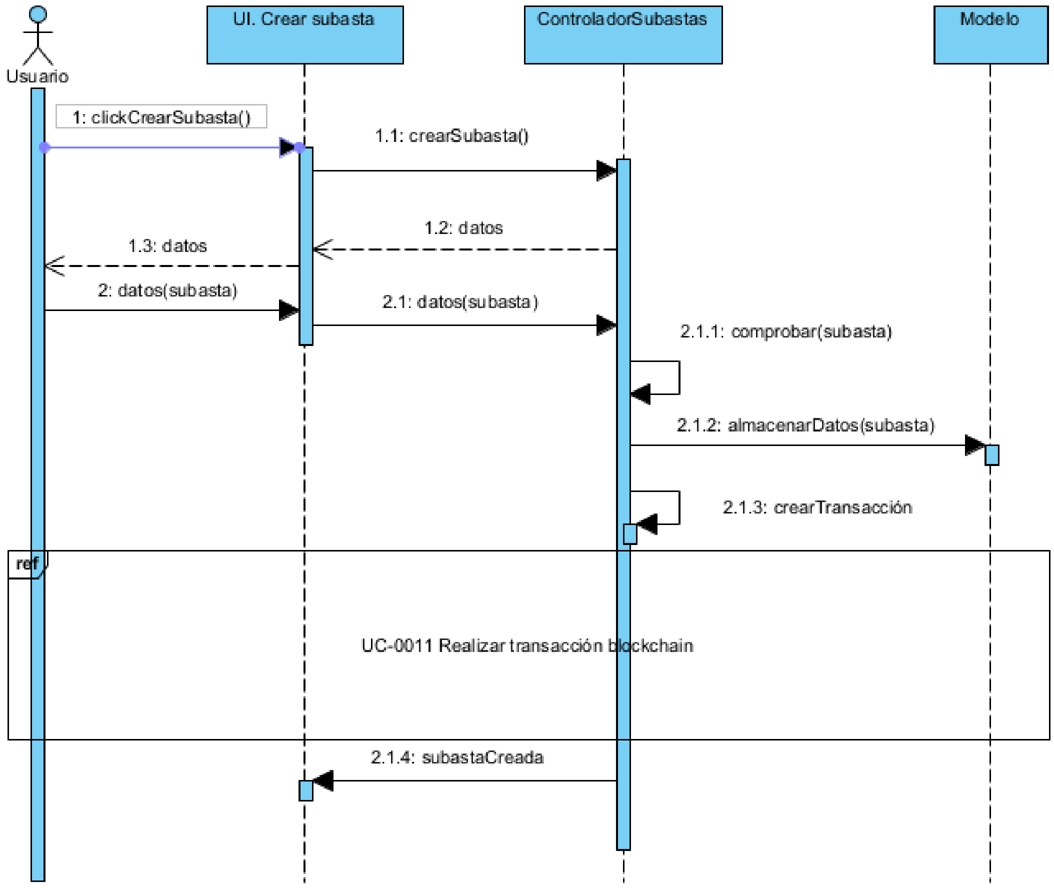

Figura 19. Diagrama de secuencia crear subasta

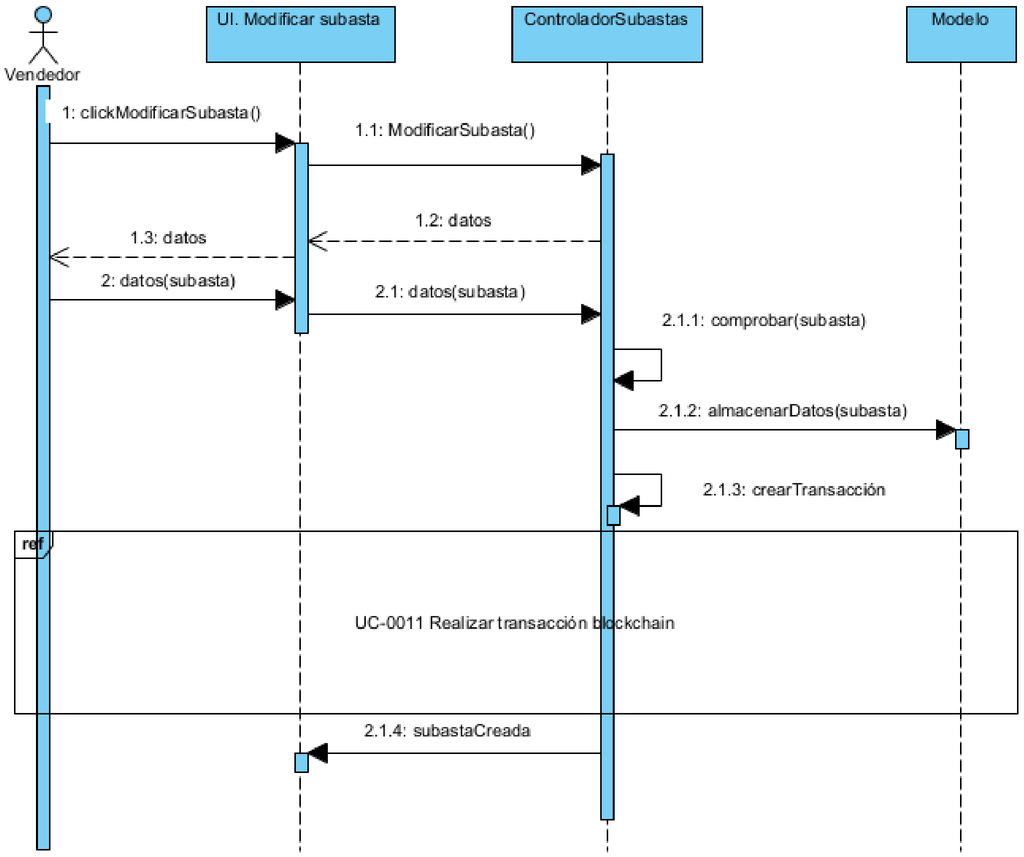

Figura 20. Diagrama de secuencia modificar subasta

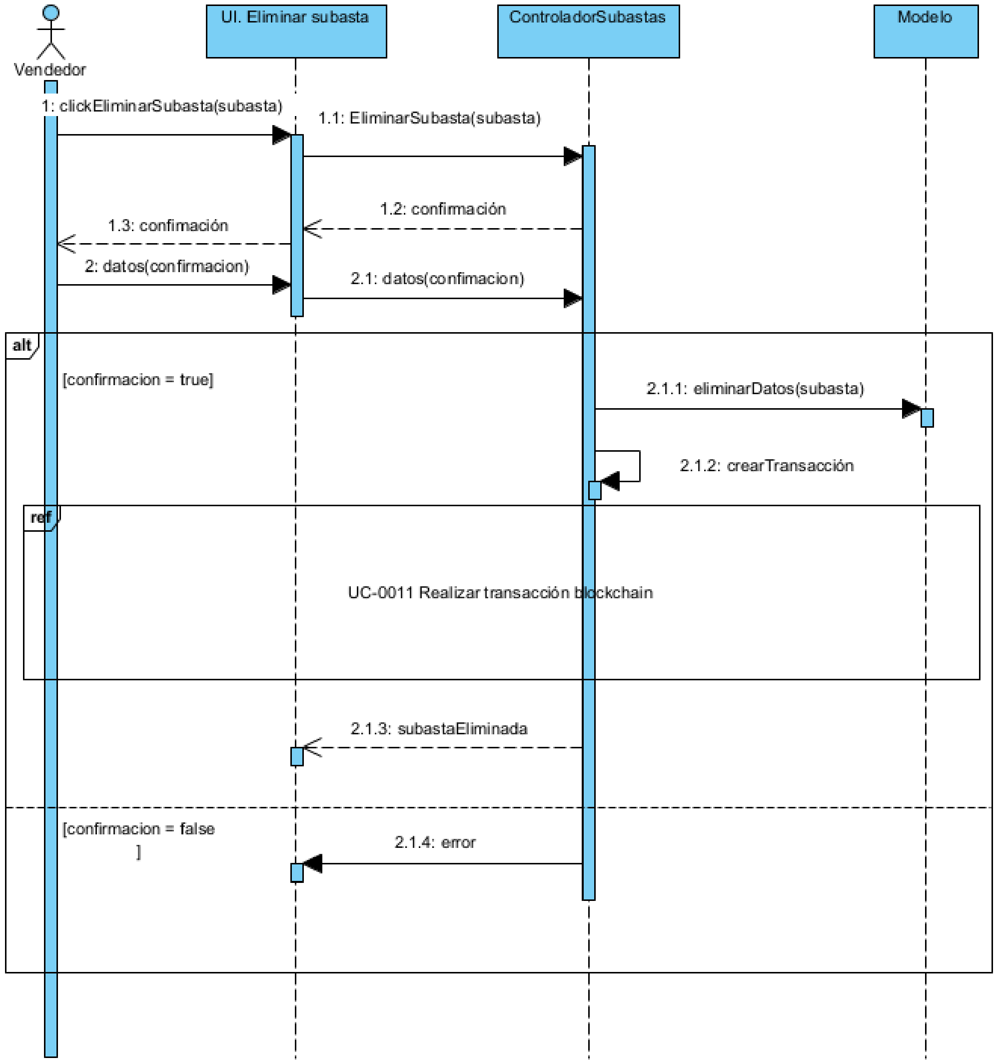

Figura 21. Diagrama de secuencia eliminar subasta

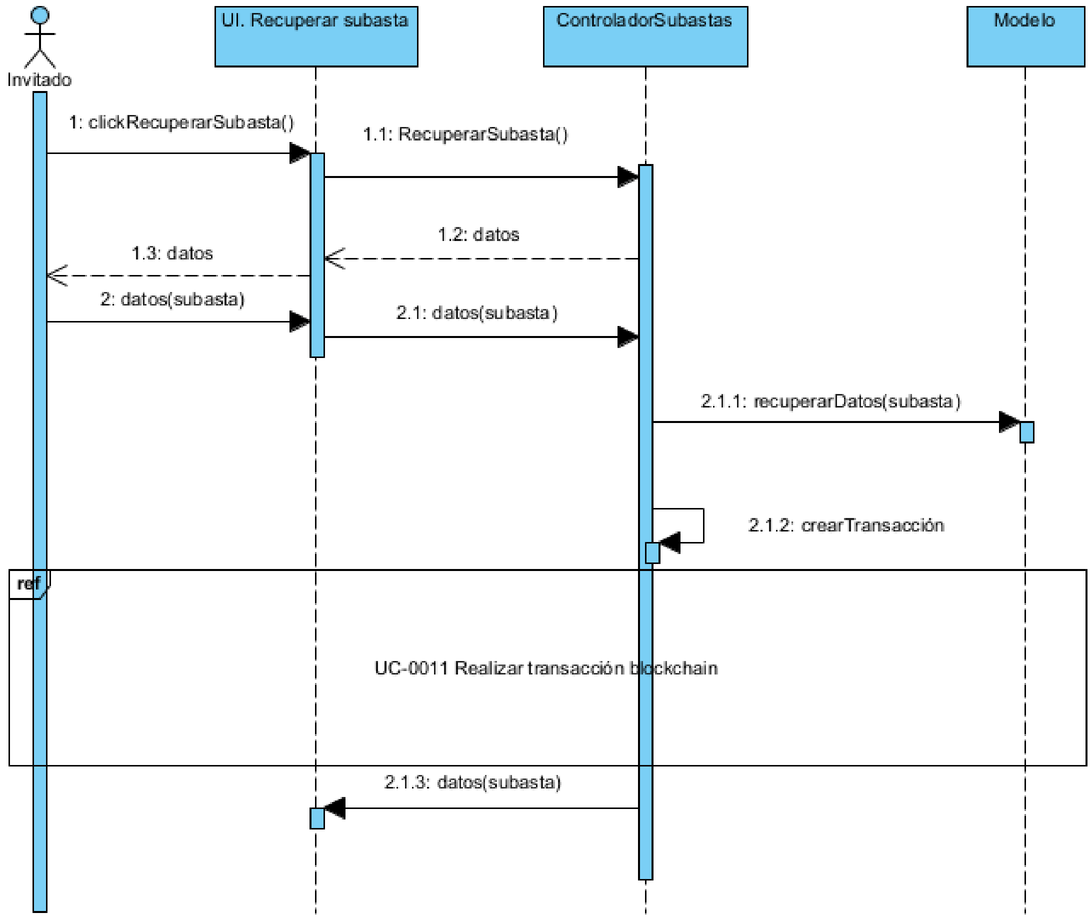

Figura 22. Diagrama de secuencia recuperar subasta

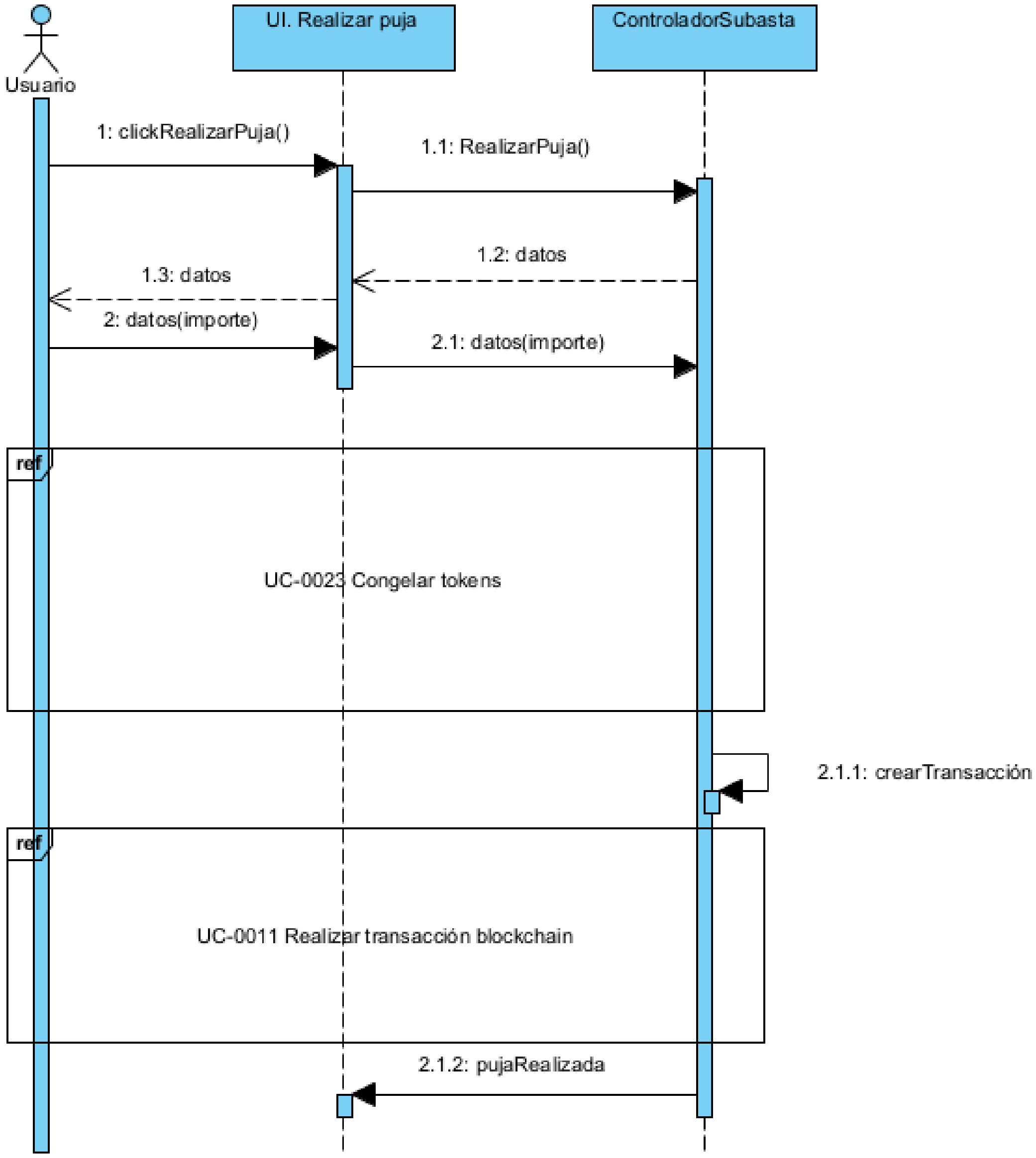

Figura 23.Diagrama de secuencia realizar puja

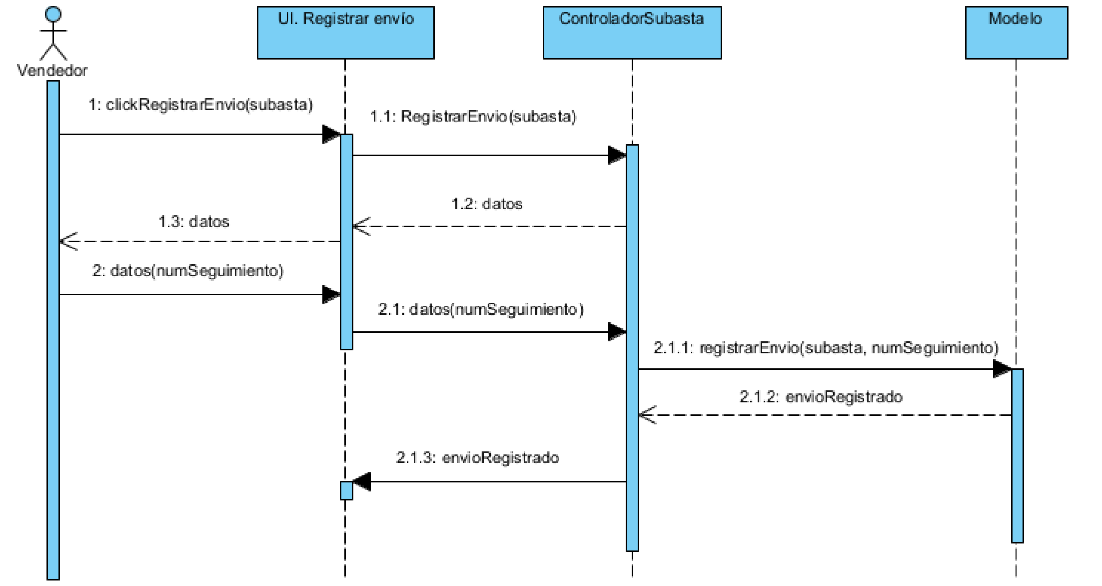

Figura 24. Diagrama de secuencia registrar envío

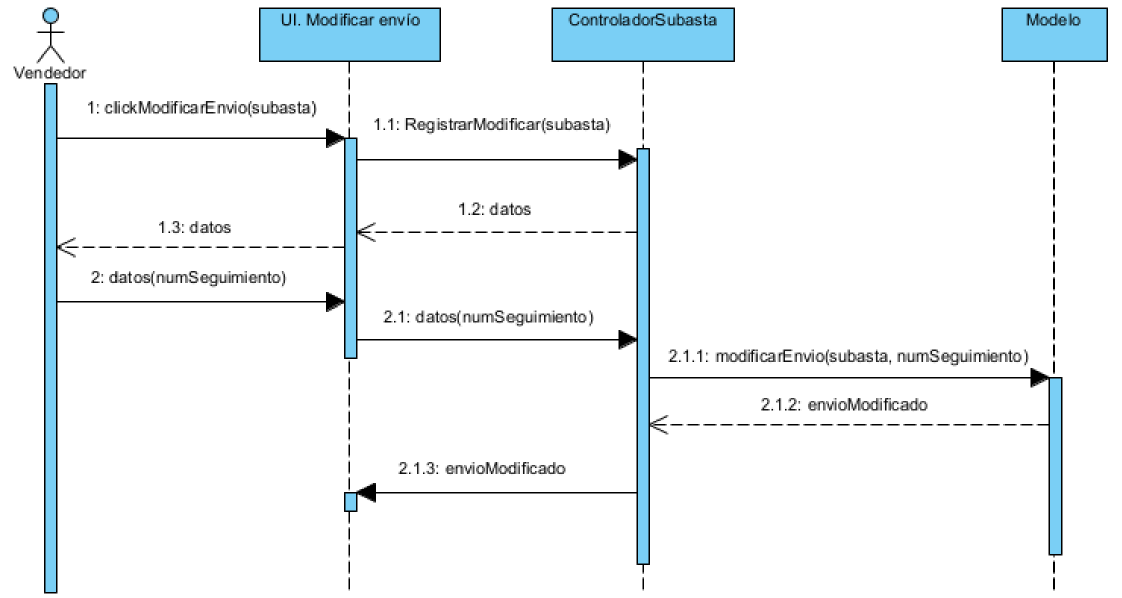

Figura 25. Diagrama de secuencia modificar envío

5 Modelo de despliegue

El diagrama de despliegue permite describir la distribución física del sistema, donde cada nodo representa un dispositivo.

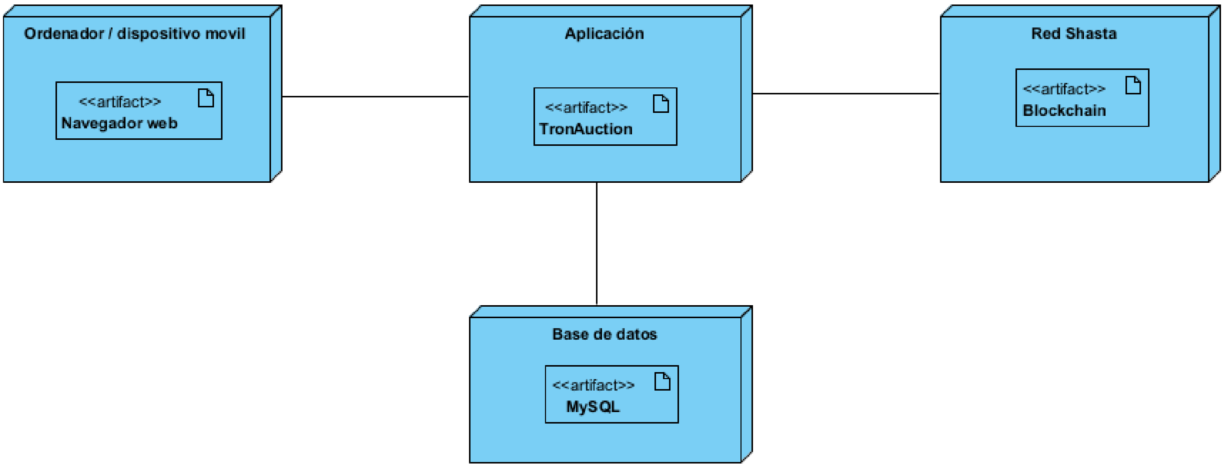

Figura 26: diagrama de despliegue

TronAuction: plataforma de subastas online utilizando *Smart Contracts*

Anexo IV: Documentación técnica de programación

Grado en ingeniería informática

VNiVERSiDAD
Ð SALAMANCA

Septiembre de 2021

Autor

Carlos Álvarez López

Tutores

Juan Manuel Corchado Rodríguez

Diego Javier Valdeolmillos Villaverde

Agustín San Román Guzmán

Tabla de contenido

1. Introducción

En el presente documento se recoge la documentación técnica del proyecto referida a su programación o codificación. Para ello se ha utilizado la herramienta *JavaDoc*. Una utilidad que genera código HTML navegable, a partir de las anotaciones realizadas mediante unos comentarios con etiquetas especiales en el código fuente de la aplicación. De esta forma a la vez que se va escribiendo código, se va documentando, y al final se obtiene una documentación en conjunto mucho más clara, pero sobre todo mucho más fácil de utilizar, debido a los enlaces que crea entre todos sus componentes: clases, atributos y métodos.

Además, al ser *JavaDoc* el estándar de la industria, y el propio formato que utiliza Java para documentar su API, facilita su interpretación por futuros desarrolladores.

Se adjunta por tanto una carpeta que contiene una estructura de ficheros organizados en subcarpetas, que puede ser recorrida con cualquier navegador, a partir del fichero de entrada *index.html*.

TronAuction: plataforma de subastas online utilizando *Smart Contracts*

Anexo V: Manuales de usuario

Grado en ingeniería informática

Septiembre de 2021

Autor

Carlos Álvarez López

Tutores

Juan Manuel Corchado Rodríguez

Diego Javier Valdeolmillos Villaverde

Agustín San Román Guzmán

Tabla de contenido

Índice de figuras

1. Introducción

En el presente documento se presenta el manual de usuario del proyecto *TronAuction*: plataforma de subastas online utilizando *smart contracts*.

El documento se dividirá en varias secciones relacionadas con la funcionalidad de la aplicación.

2. Vista principal

En la figura posterior se observa la pantalla inicial de la aplicación, esta vista puede ser accedida tanto por personas registradas como por personas no registradas. En ella, los usuarios pueden ver las novedades de los patrocinadores, mediante un carrusel que va alternado en la parte central los últimos productos. En la parte inferior aparecen las categorías más demandadas por los usuarios, y se puede pulsar sobre cualquiera de ellas para que se filtren los productos.

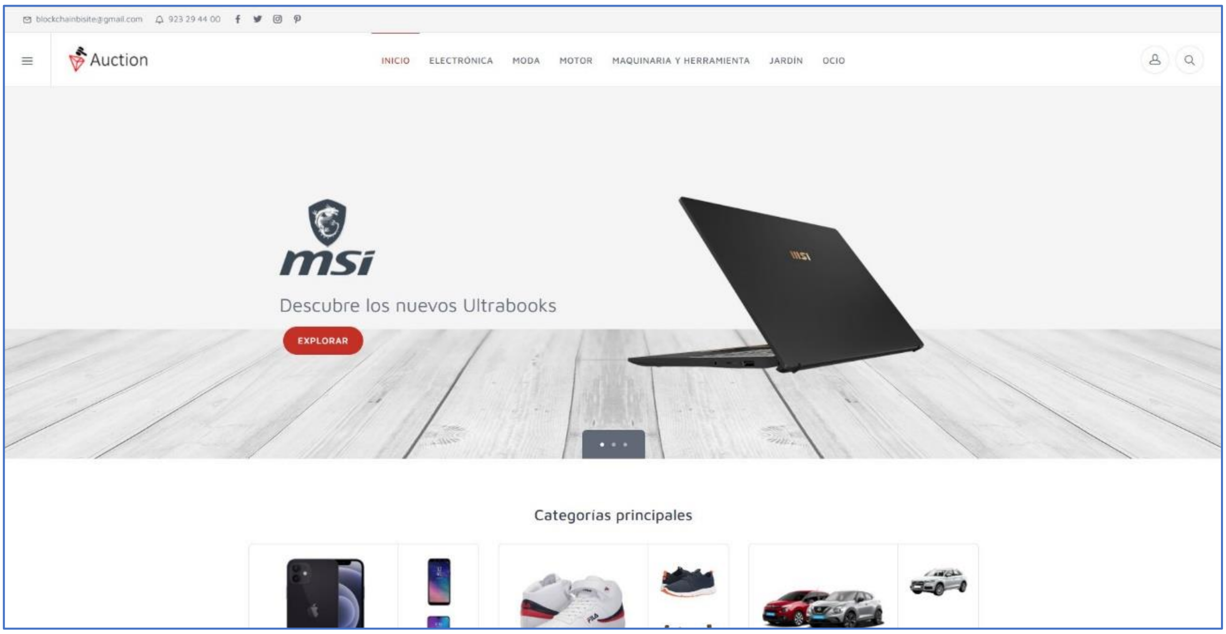

Figura 1. Vista principal

También desde esta pantalla, pinchando en el icono de la esquina superior derecha se puede acceder al espacio personal del usuario.

En la figura 2 se puede apreciar como cambia de aspecto cuando se ha filtrado por una categoría principal, en este caso ha sido por la categoría “moda”, lo que ha hecho que se desplieguen sus subcategorías mediante nuevos iconos.

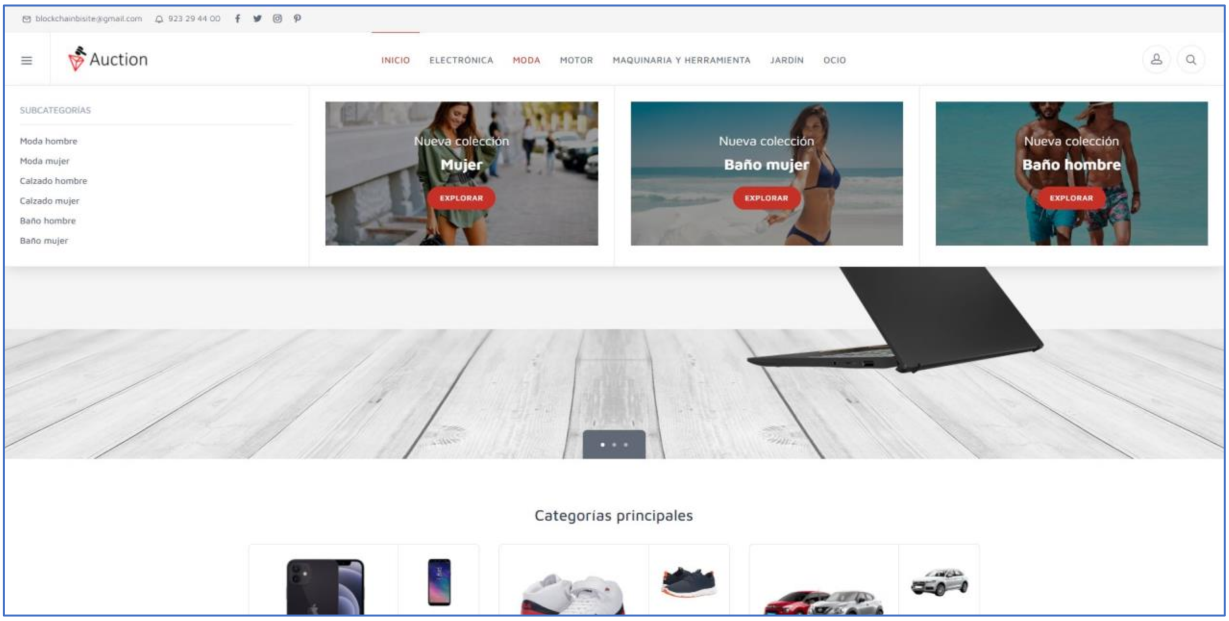

Figura 2. Filtro categorías

3. Registrarse

La primera vez que un usuario quiera usar la aplicación, debe registrarse en la misma. Para ello, rellenará los datos solicitados en la Figura 3.

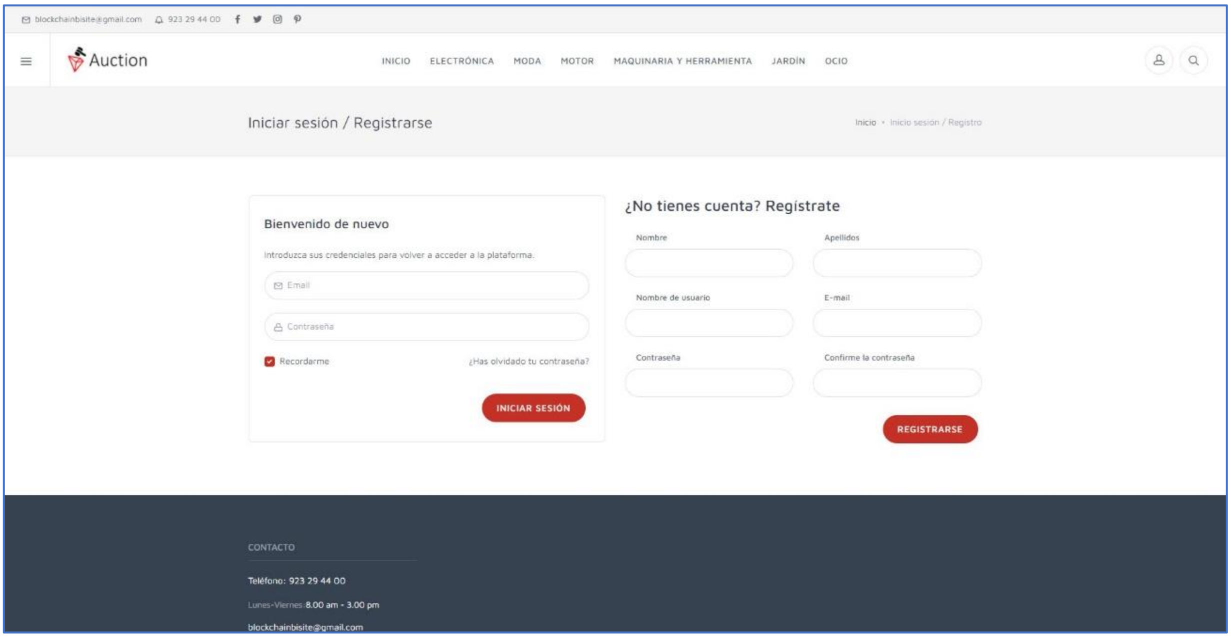

Figura 3. Registro/inicio sesión

Si todo es correcto, se completará y se mostrará un mensaje al usuario para que valide su correo electrónico, como se muestra en la figura 4.

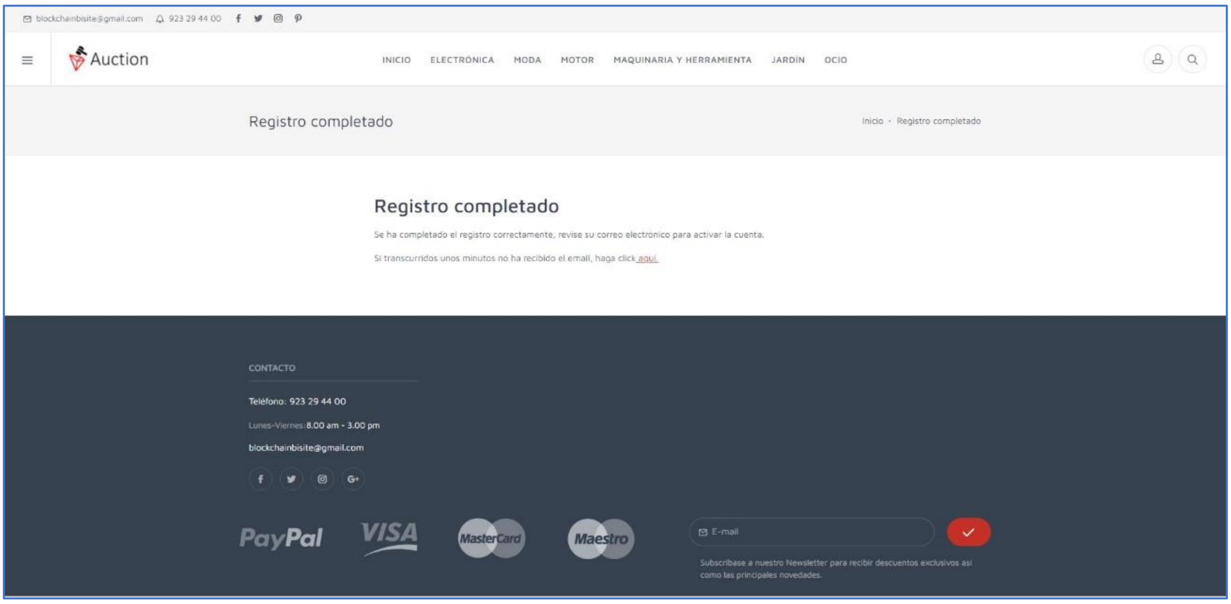

Figura 4. Solicitud de validación de email

Para poder validar su cuenta, los usuarios recibirán un email como el que se muestra en la Figura 5.

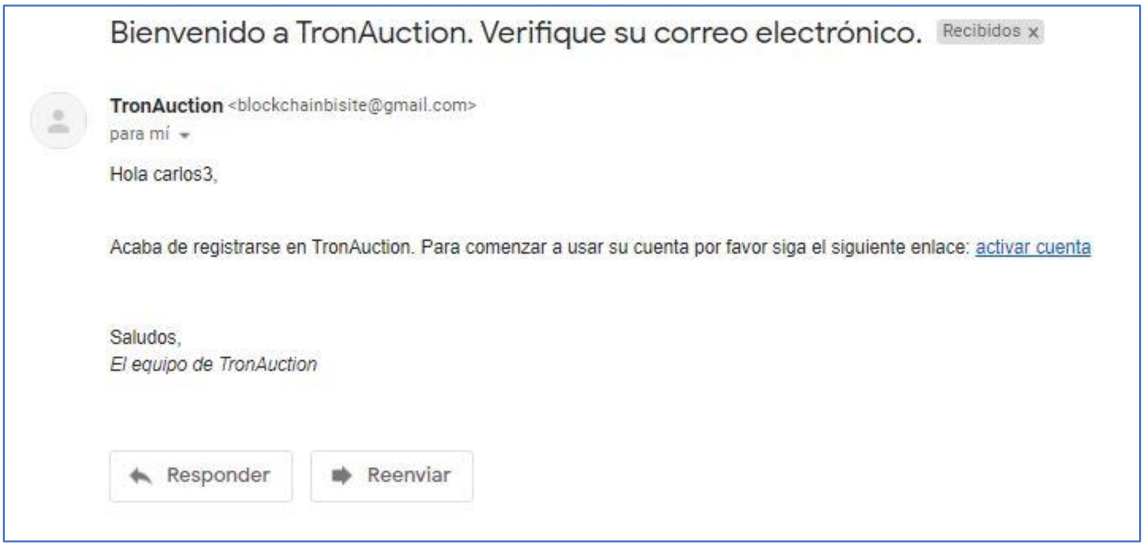

Figura 5. Email validación

Basta con hacer clic en el enlace proporcionado, y el usuario será redireccionado a la Figura 6. Donde pulsará en “validar email” para finalizar todo el proceso.

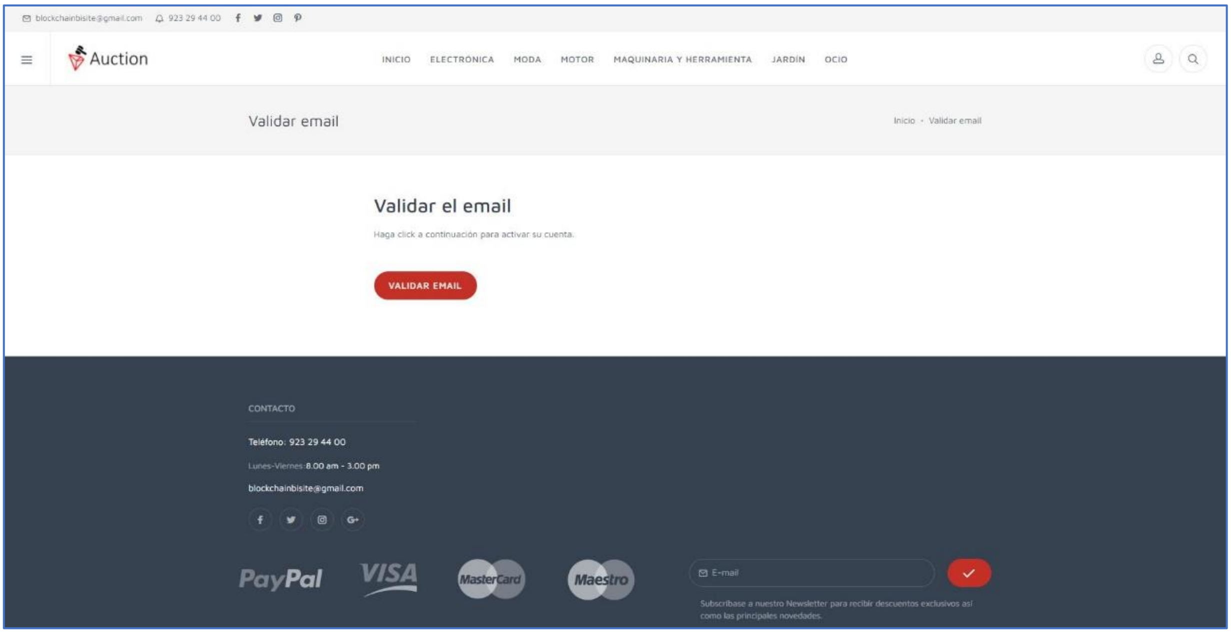

Figura 6. Fin del proceso de validación

4. Reenviar email validación

Si el usuario tras rellenar sus datos de registro no recibe el email de validación, podrá enviar un nuevo email mediante el formulario de la Figura 7. Al cual se puede acceder desde la vista expuesta en la Figura 4.

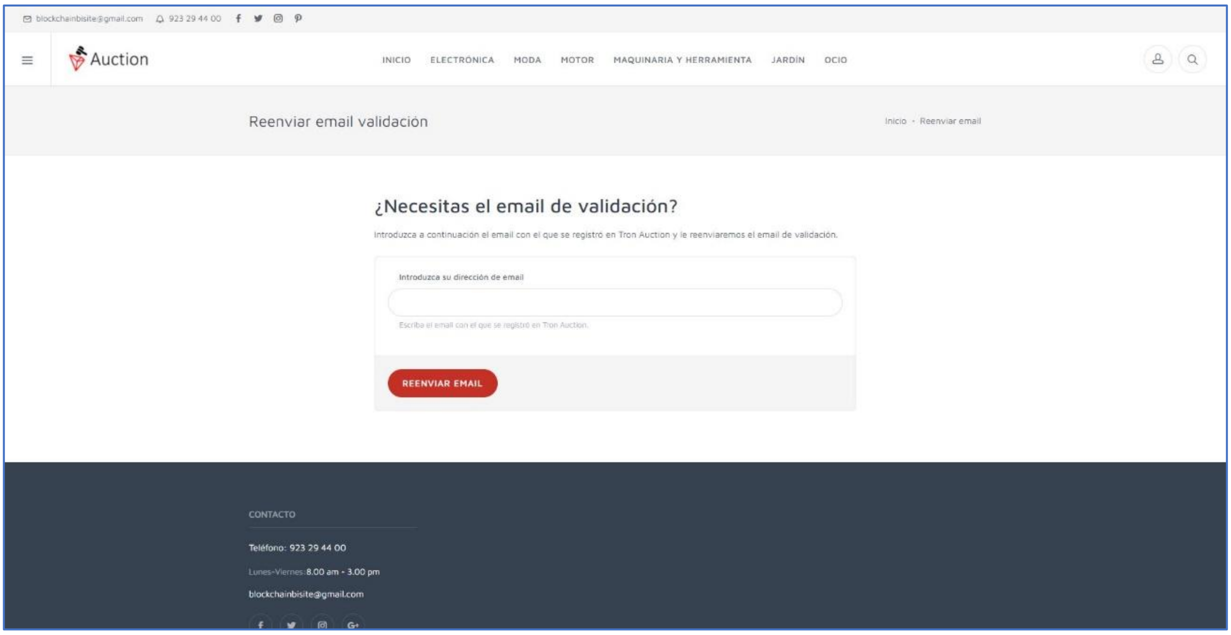

Figura 7. Reenviar email de validación

5. Cuenta del usuario

Cada usuario tendrá acceso a una zona personal denominada “mi cuenta”, en ella podrá consultar sus datos personales, cambiar su contraseña, ver su *wallet*, ver sus pujas, etc. Todas estas funcionalidades se describen a continuación.

5.1 Mi perfil

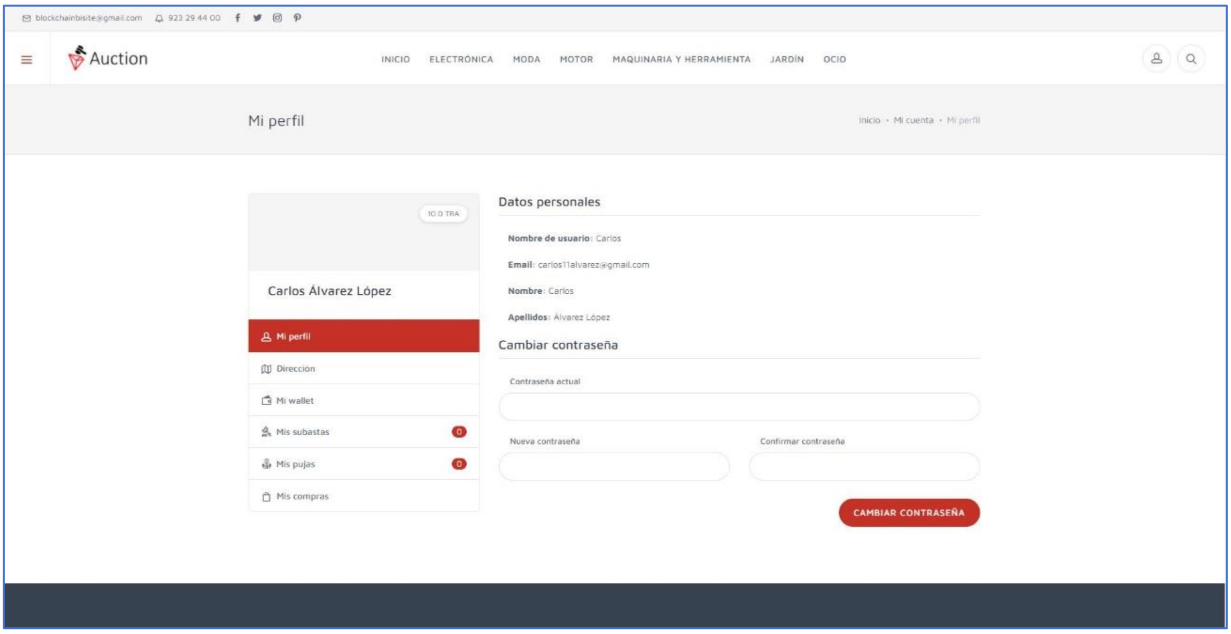

Figura 8. Mi perfil

En el apartado “Mi perfil”, el usuario podrá consultar sus datos personales y cambiar su contraseña.

5.2 Dirección

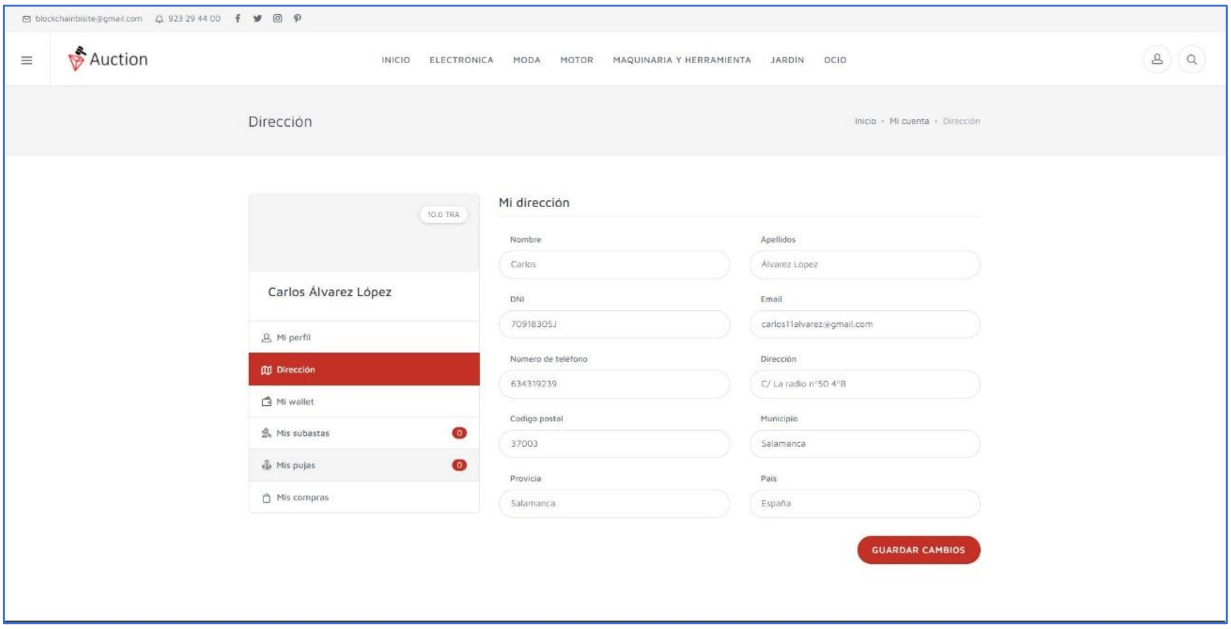

Figura 9. Mi dirección

En el apartado "Mi dirección del perfil", cada usuario podrá ver su dirección, así como modificarla. Para ello, modificará los datos y pulsará sobre el botón de modificar dirección.

5.3 Mi *wallet*

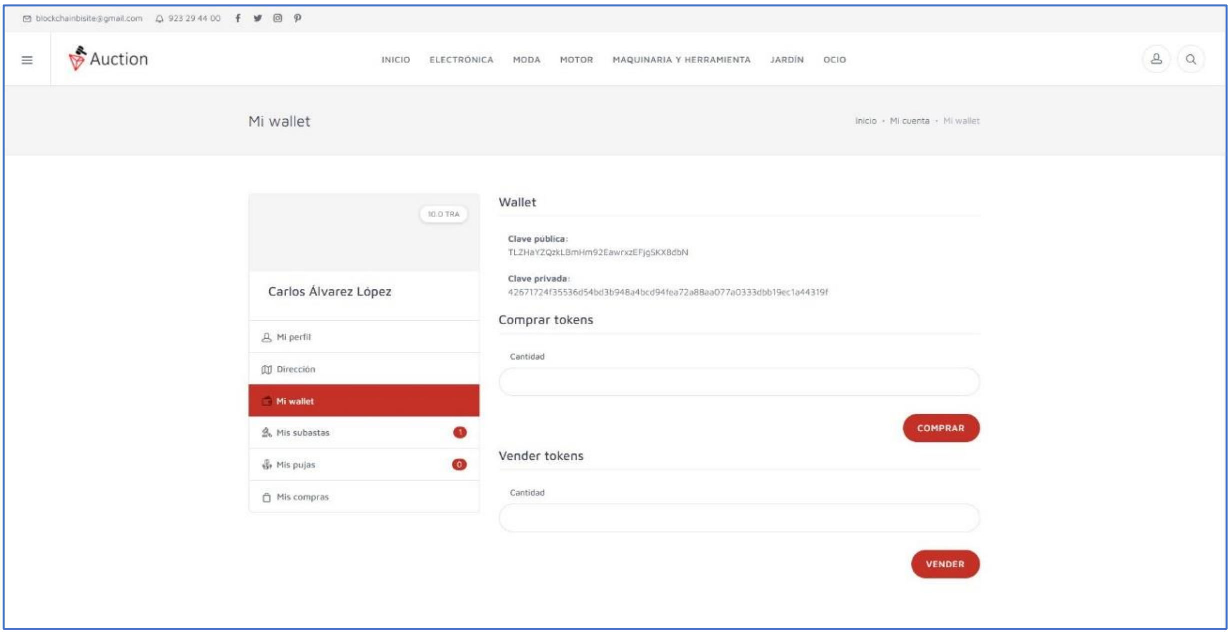

Figura 10. Mi wallet

En la sección "Mi *wallet*", el usuario podrá ver los datos de su monedero (tanto la clave privada, como la clave pública). Además, dispone de dos opciones, una para comprar *tokens,* y otra para venderlos. Estas dos opciones deberían implementar una plataforma tradicional de pago para la compra y la venta de criptomonedas. Pero al tratarse de un trabajo fin de grado, y no de un proyecto en producción, los usuarios podrán obtener y vender las monedas mediante el formulario disponible en la sección.

5.4 Mis subastas

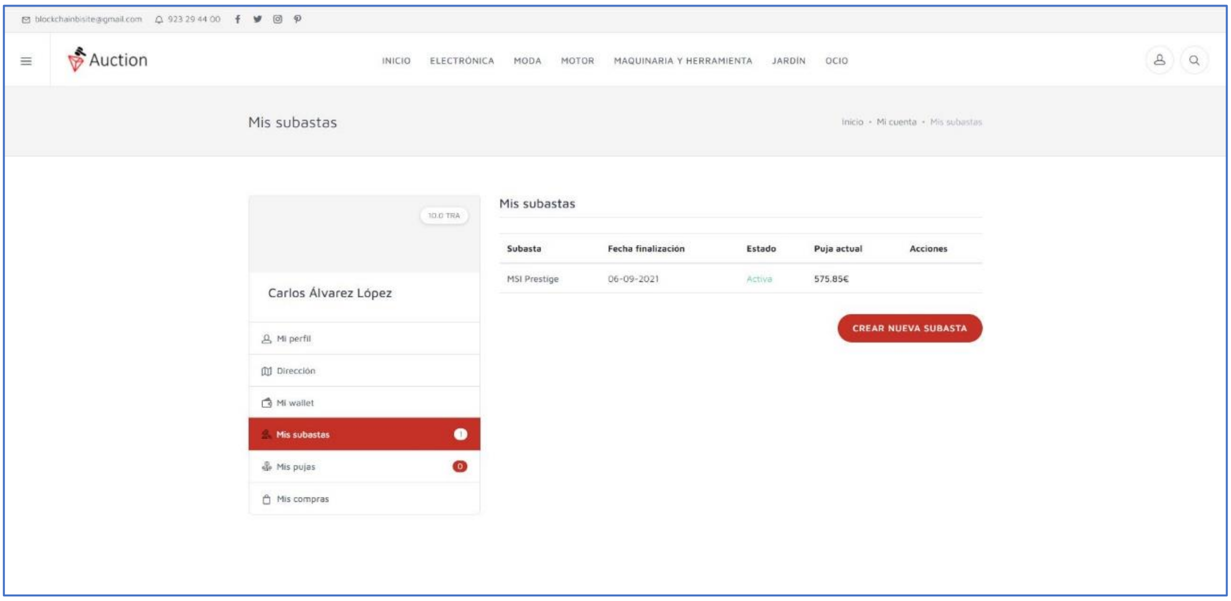

Figura 11. Mis subastas

En esta vista el vendedor podrá ver las subastas que tiene en curso, en el estado en el que están, la puja máxima actual y una serie de acciones. Estas acciones serán, eliminar o modificar la subasta (siempre y cuando no hayan pasado más de 2 horas desde la creación de la subasta) y registrar envío (cuando la subasta haya terminado).

Además, dispone de la función crear nueva subasta. La cual se expondrá más adelante.

5.5 Mis pujas

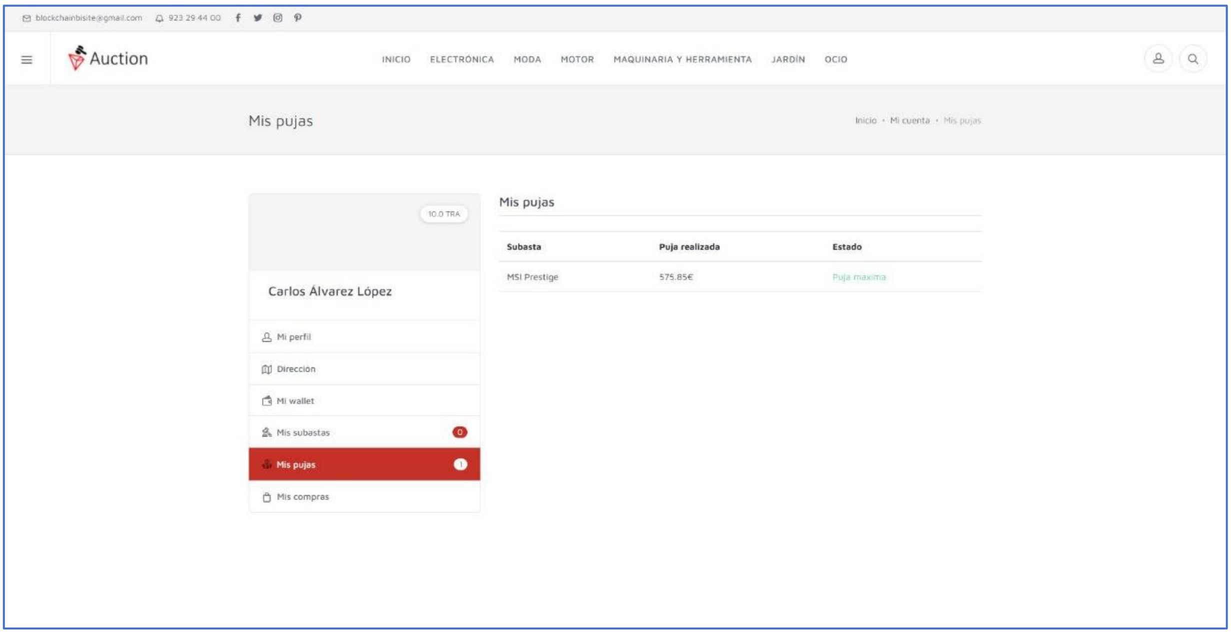

Figura 12. Mis pujas

En este apartado el usuario podrá ver sus pujas realizadas, y su estado (si es la máxima puja, si es la puja ganadora o si ha sido sobrepujado por otro pujador).

5.6 Mis compras

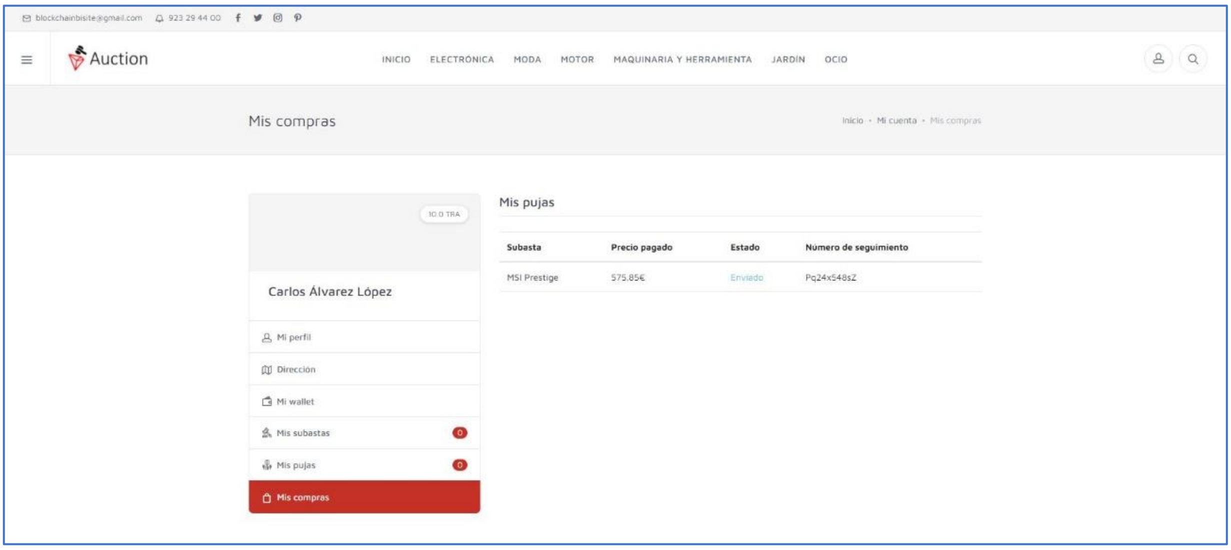

Figura 13. Mis compras

En este último apartado de la vista del perfil, el usuario podrá ver las subastas ganadas, el estado del envío, así como el número de seguimiento de este.

6. Crear nueva subasta

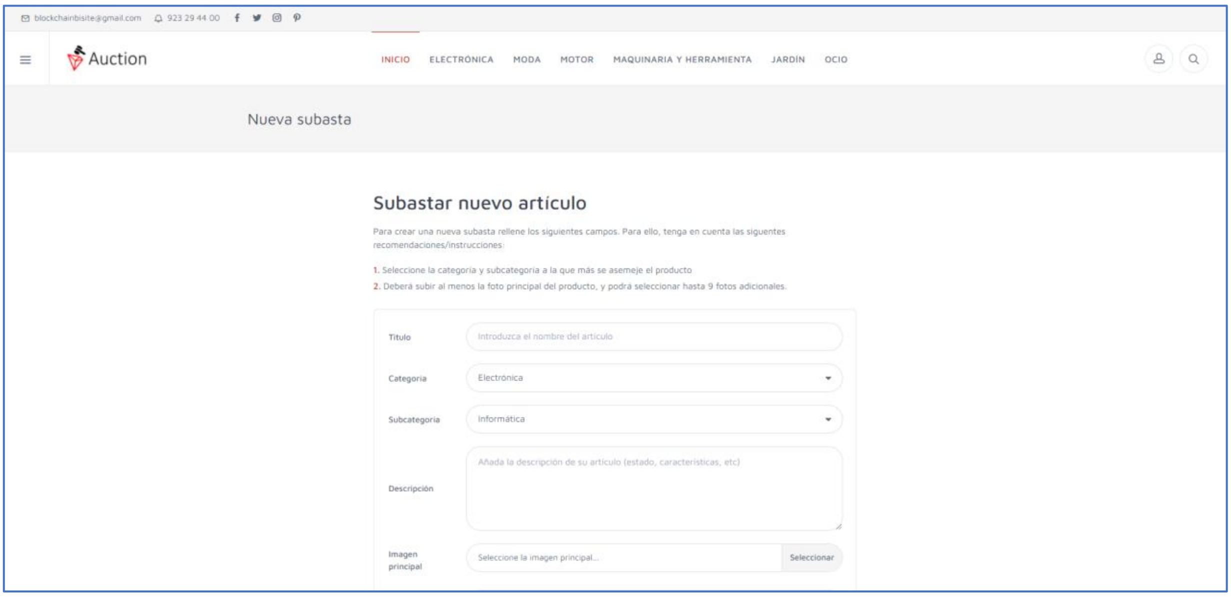

Figura 14. Crear nueva subasta

Cuando un vendedor quiere registrar una nueva subasta debe rellenar el formulario que se observa en la Figura 14. En él debe introducir el título del artículo, definir la categoría y la subcategoría a la que pertenece, introducir una pequeña descripción, introducir el precio de salida, y finalmente la fecha de finalización de la subasta.

7. Catálogo de subastas

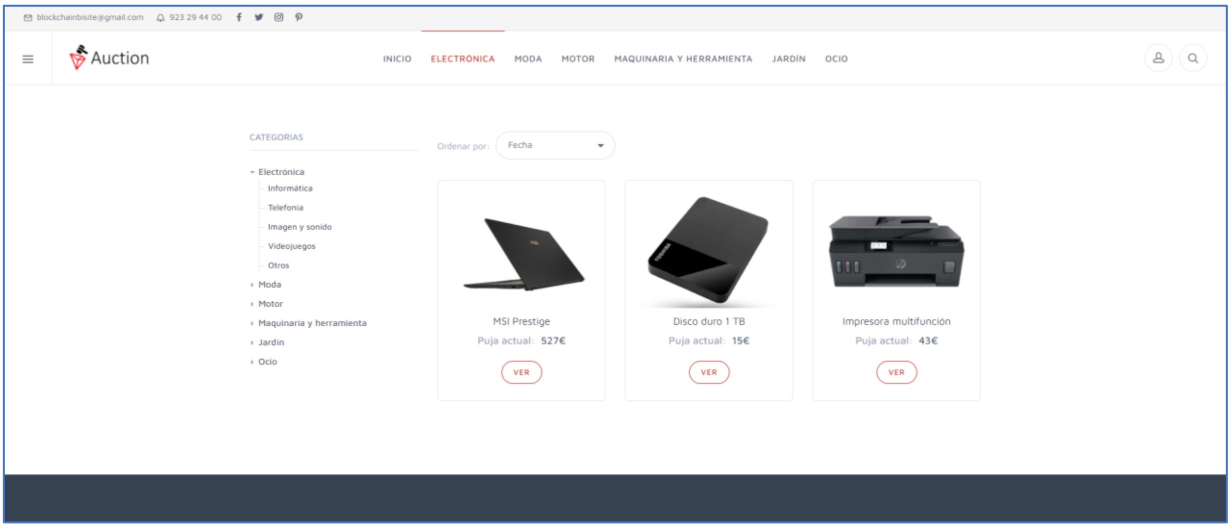

Figura 15. Catálogo de subastas

Los usuarios podrán acceder desde los botones superiores (electrónica, moda, motor, etc.) al catálogo de las subastas de dicha categoría. En él podrán ver los artículos que se están subastando actualmente y acceder a su descripción.

8. Descripción subasta

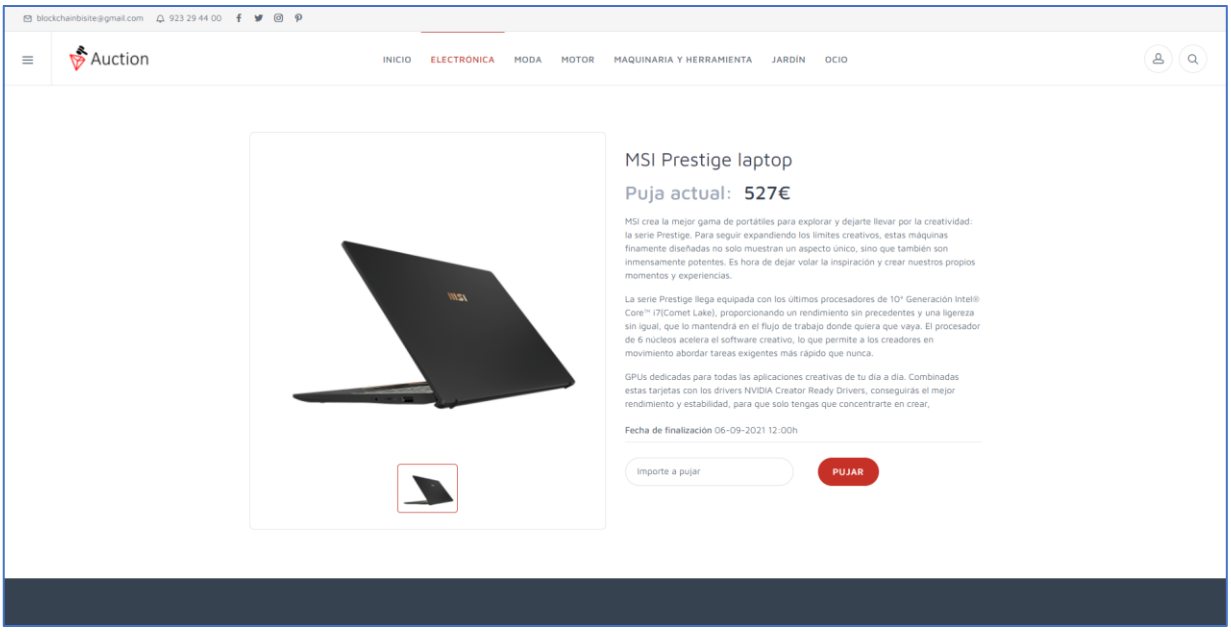

Figura 16. Descripción subasta

Cuando un usuario accede a la descripción de una subasta, podrá ver los datos del artículo, la puja actual, la fecha de finalización y si lo desea podrá hacer una puja por un importe superior al de la puja actual.

9. Vista en dispositivos móviles

La aplicación web se ha desarrollado de modo que sea Web *Responsive*, es decir que se adapte de manera automática al tamaño del dispositivo en el que se esté visualizando. A continuación, se muestran algunos ejemplos de vistas del navegador web de un terminal Android, con una relación de aspecto de pantalla de 18:9.

Figura 17. Vista principal dispositivo Android

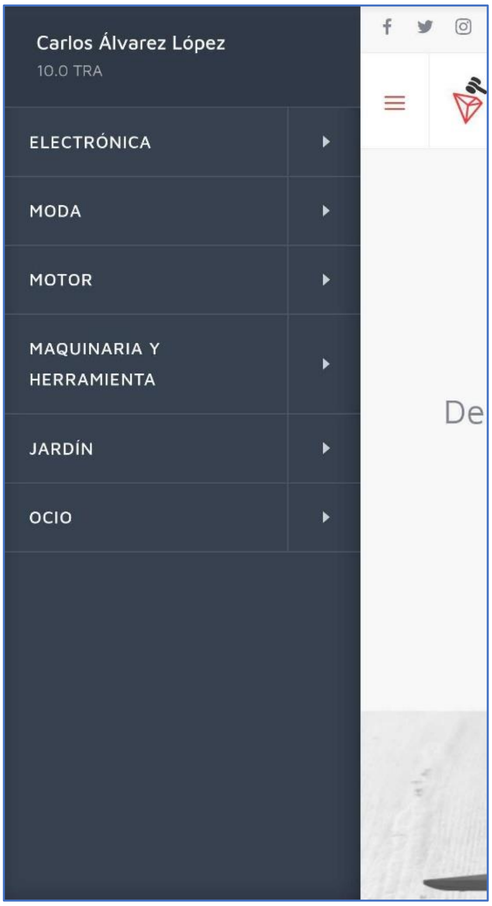

Figura 18. Panel de navegación lateral dispositivo Android

Figura 19.Vista de una subasta dispositivo Android