UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo de un sistema de votaciones distribuido en la red blockchain de Ethereum

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Bernat Correas, Xavier

Tutor/a: León Fernández, Antonio

CURSO ACADÉMICO: 2022/2023

Resumen

Este proyecto de final de grado es un estudio sobre la tecnología de cadena de bloques, conocida como Blockchain, encontrándose esta tecnología en continuo desarrollo y con una gran perspectiva de cara a los próximos años. En el proyecto se profundizará en la tecnología Blockchain de Ethereum, cuya plataforma de código abierto se centra en la creación de aplicaciones descentralizadas. Más en concreto, el proyecto se centrará en los contratos inteligentes, Smart Contracts, viendo sus usos, ventajas y aplicaciones en la actualidad.

Una vez realizado el contexto teórico, el proyecto se centrará en el proceso de desarrollo de una aplicación Smart Contract que se ejecuta en la red Blockchain de Ethereum, usando el lenguaje de programación Solidity, para permitir a los usuarios participar en un proceso de votación distribuido y fiable. Está aplicación estará apoyada por otra aplicación basada en javascript que servirá para obtener los datos y mostrarlos en una web de prueba

Resum

Aquest projecte de final de grau és un estudi sobre la tecnologia de cadena de blocs, coneguda com a Blockchain, trobant-se aquesta tecnologia en continu desenvolupament i amb una gran perspectiva de cara als pròxims anys. Al projecte s'aprofundirà en la tecnologia Blockchain d'Ethereum, de la qual la plataforma de codi obert se centra en la creació d'aplicacions descentralitzades. Més en concret, el projecte se centrarà en els contractes intel·ligents, Smart Contracts, veient els seus usos, avantatges i aplicacions en l'actualitat. Una vegada realitzat el context teòric, el projecte se centrarà en el procés de desenvolupament d'una aplicació Smart Contract que s'executa a la xarxa Blockchain d'Ethereum, utilitzant el llenguatge de programació Solidity, per permetre als usuaris participar en un procés de votació distribuït i fiable. Aquesta aplicació estarà suportada per una altra aplicació basada en JavaScript que servirà per obtenir les dades i mostrar-les en una web de prova.

Abstract

This final degree project is a study of blockchain technology, which is in continuous development and has great prospects for the coming years. The project will deepen into Ethereum's Blockchain technology, whose open source platform focuses on the creation of decentralised applications. More specifically, the project will focus on Smart Contracts, looking at their uses, advantages and current applications.

Once the theoretical background has been provided, the project will focus on the development process of a Smart Contract application running on the Ethereum Blockchain network, using the Solidity programming language, to allow users to participate in a distributed and reliable voting process. This application will be supported by another javascript-based application that will serve to obtain the data and display it on a test website.

ÍNDICE

Parte 1. Introducción

1.1. Introducción

Desde la aparición de la tecnología Blockchain, han aparecido una infinidad de proyectos que se basaban en esta tecnología, con estos se ha buscado la descentralización en muchas especialidades de la actualidad.

La simplicidad, fiabilidad y variabilidad que aporta esta tecnología son los principales motivos de que se hable tanto de esta. Además, una gran cantidad de estudios demuestran que esta tecnología permite una revolución en bastantes ámbitos de la vida cuotidiana, como es el almacenamiento de información, la identificación de personas, la alimentación o la más conocida, en la economía con la aparición de las criptomonedas.

Es un hecho que el uso por el que la gente suele reconocer esta tecnología es el de las cripto monedas. De hecho, la aparición de la tecnología fue a través de la Bitcoin, en lo que profundizaré en los siguientes apartados. Pero la que más me ha intrigado y motivado a realizar el proyecto es la de un sistema de votación, lo cual puede revolucionar el formato de elecciones que conocemos en la actualidad.

Por todo eso, este proyecto expondrá los conceptos de la tecnología Blockchain, a través de la plataforma Ethereum y los contratos inteligentes. Apoyando estos conceptos a través del desarrollo de un Smart Contract que permita realizar la simulación de unas elecciones, sin importar el contexto a través del lenguaje Solidity.

1.2. Objetivos

Como ya he dicho, la tecnología Blockchain aplicada en un Smart Contract que permita la realización de un sistema de votación descentralizado puede mejorar mucho los sistemas de votaciones que se suelen utilizar en la actualidad.

Pero para poder llegar hasta este objetivo final, primero hay que recoger una serie de prerrequisitos, u objetivos previos:

- Comprensión del funcionamiento de la tecnología Blockchain.
- Conocimientos de la implementación de Smart Contracts en la actualidad.
- Conocimientos sobre las plataformas descentralizadas, especialmente sobre Ethereum.
- Desarrollo de la aplicación Smart Contract que permita participar en un sistema de votaciones.

1.3. Metodología

El proyecto consistirá en un conjunto de bloques:

- Comenzaremos estudiando los fundamentos de la Blockchain, empezando por sus orígenes, y conociendo conceptos como su funcionamiento, además de los usos que tiene en la actualidad.
- En segundo lugar, se realizará un pequeño estudio sobre los Smart Contracts y su impacto
- Posteriormente, se realizará un estudio ya más centrado en Ethereum, pasando por su máquina virtual y terminando por el conocimiento del lenguaje Solidity.
- Una vez obtenidos todos los conocimientos previos, se realizará un breve estudio de lo necesario para realizar el Smart Contract y sus interacciones con los usuarios.
- Posteriormente, se realizará la aplicación y se observarán tanto los resultados como los problemas o mejoras.
- Finalmente, se desarrollará una conclusión tanto de los resultados como de las opciones que hay para mejorar la aplicación.

Parte 2. Fundamentos teóricos de la Blockchain

2.1 Orígenes de la Blockchain

Normalmente se dice que el origen de las Blockchain se encuentra junto con el origen del bitcoin, pero no es realmente así, pues en 1991 los científicos Stuart Haber [4] y W.Scott Stornetta [5] presentaron una solución computacionalmente factible para los documentos digitales que cuentan con un sello de tiempo, con el objetivo de evitar su manipulación o alteración. Este sistema utilizó una cadena de bloques con seguridad criptográfica para almacenar documentos con sello de tiempo.

Más tarde, en 1992 se incorporó al diseño los árboles de Merkle [2], los cuales se explicarán más adelante, que aumentó la eficiencia, pues permitía el almacenamiento de varios documentos en un solo bloque. A pesar de ello, esta tecnología no fue utilizada y su patente caducó en 2004 [1].

Posteriormente, en 2008, una persona o grupo de personas, pues todavía no se sabe la verdadera identidad, utilizando de seudónimo Satoshi Nakamoto [7], publicó un libro que introdujo un sistema de pagos electrónicos descentralizado punto a punto, Bitcoin.

Este sistema se basaba en el algoritmo de prueba Hashcash, pero utilizando un protocolo peer-to-peer [9] descentralizado para rastrear y verificar las transacciones, el cual proporcionaba la doble protección.

No fue hasta el enero de 2009 que Satoshi Nakamoto minó el primer bloque en Bitcoin, donde recibió una recompensa de 50 bitcoins.

Años después, más concreto en 2013, Vitalik Buterin [10] (fundador de Ethereum) dijo que Bitcoin necesitaba un lenguaje de scripting para crear aplicaciones descentralizadas, por lo que inició el desarrollo de una plataforma de computación distribuida basada en Blockchain, Ethereum, donde se presentaba la figura de los Smart Contracts.

2.2 Estructura y funcionamiento de la Blockchain

Las Blockchain son estructuras de datos distribuidas, descentralizadas y seguras, permiten el almacenamiento y la verificación de transacciones si la necesidad de terceros, utilizándose estas en una gran variedad de ámbitos.

Estas estructuras consisten en una seria de bloques que contienen una gran cantidad de información sobre las transacciones, como son las cantidades transferidas, la fecha exacta a la que se realiza la transacción, las direcciones de los dos “usuarios” que realizan la transacción, además de la información de bloques anteriores gracias al denominado hash criptográfico, cosa que indica que no puede haber modificaciones en la cadena, pues con el hash se detectaría muy rápido.

Antes de continuar, hay que explicar lo que es el hash criptográfico, las funciones hash o resumen convierte un bloque en una serie de caracteres con una longitud fija, independientemente de la longitud de lo introducido en la entrada, siendo distinto a la mínima modificación en el bloque de la entrada, dicho esto, las funciones hash criptográficas, son aquellas que se utilizan en el ámbito criptográfico.

El funcionamiento de las Blockchain es bastante simple y se encuentra basada en una red de nodos distribuidos, es decir, no hay nodo principal sino todos se encuentran en el mismo nivel. Cada nodo es un ordenador, o dispositivo conectado a la red el cual ejecuta, y almacena una copia de toda la cadena de bloques, parecido a lo que ocurre en utorrent donde te descargas una aplicación a través de otros peers, para posteriormente guardarla y ayudar con tu información a obtener la aplicación a otros peers. Cuando una transacción es realizada, esta es enviada a la red donde los nodos validan esta transacción y la añaden a la cadena de bloques.

Por otra parte, buscando la seguridad e inmutabilidad, que no puede ser modificado, de la Blockchain, se utilizan varios mecanismos, como es el de proof-of-work [27], donde los diferentes nodos compiten por resolver el siguiente bloque a través de la resolución de algoritmos criptográficos, siendo el primero en resolver el bloque el que lo añade, después de ser validado, a la cadena obteniendo una recompensa. Este proceso se denomina minería, y en la actualidad que han construido macro fábricas con centenares de nodos tratando de resolver los bloques y obtener las recompensas. Estas recompensas suelen ser económicas, como es el caso de los bitcoins en Bitcoin o el Ether en Ethereum.

2.3 Tipos de Blockchain

En la actualidad existen tres tipos de Blockchain:

- **Blockchain pública:** Estas Blockchain fueron las primeras en aparecer, y son aquellas que dan total libertad de acceso. Estas son totalmente abiertas, pues el público puede observar sus datos, software y desarrollo. Para poder realizarlo, estas cadenas tienen una gran implementación de seguridad, para así evitar la presencia de nodos maliciosos que puedan alterar su funcionamiento. Estas son descentralizadas, pues no hay ningún nodo que sea central o dominante, todos se encuentran en el mismo nivel. Normalmente la economía de estas depende de la minería y de las comisiones de las transacciones. Algún ejemplo seria Bitcoin o Ethereum.
- **Blockchain privadas:** Los elementos son muy parecidos a los de las Blockchain públicas, pero la gran diferencia es que estas dependen de una unidad central que controla todas las acciones que suceden dentro de la Blockchain, Un ejemplo de esto sería a la hora de dar permiso, a la del acceso a los usuarios o a la hora de tener el acceso para saber

las diferentes transacciones. Por otra parte, ya no dependen de la minería, sino de la empresa controladora de la Blockchain. Algunos ejemplos conocidos de estas Blockchain son Hyperledger [11] o Corda [12].

- **Blockchain híbrida:** Como su nombre indica, estas Blockchain son una mezcla de las dos anteriores, intentando aprovechar los beneficios de ambas. Por ejemplo, la participación en ella es privada, sin embargo, cualquier nodo puede acceder a las transacciones que ocurren. Por otra parte, al igual que en las privadas no hay minería. Por ello, estas son usadas en gobiernos y empresas que deben compartir y almacenar datos de forma segura.

Ahora, conociendo los tipos de Blockchain que hay se podría ver las diferentes aplicaciones en las que estas se utilizan:

- **Criptomonedas**: Este es el uso más conocido de las cadenas, pues todo el mundo ha oído hablar de la Bitcoin. En este caso, las Blockchain han llegado a la economía, pues con estas monedas se realizan transacciones, ventas, e incluso hay casos donde los pagos por la realización de trabajo se realizan en ellas. Esto es debido a que las criptomonedas tienen una relación de equivalencia con las monedas físicas, siendo el valor de las criptomonedas más variante.
- **Sanidad**: La arquitectura en la que se basa la base de datos de Blockchain es ideal para el almacenamiento de historiales médicos.
- **Liquidaciones de seguros**: Las aplicaciones basadas en blockchain minimizan el riesgo a fraude al mismo tiempo que agiliza el reembolso por pérdidas en la industria de los seguros.
- **Videojuegos**: Las Blockchain pueden proporcionar una mayor aleatoriedad en juegos de azar, como son los dados, póker… También sirven mejor para registrar estadísticas de los jugadores, e incluso puede servir para el sistema de compras dentro de los diferentes videojuegos al tener cierta similitud con las criptomonedas.
- **Medios de comunicación**: En este caso sería para proporcionar más seguridad de los archivos multimedia y así evitar, o al menos reducir, la piratería.
- **Compilación de datos**: Las Blockchain pueden utilizarse como método de registro de una gran cantidad de datos.
- **IoT**: Las IoT permiten que los diferentes dispositivos con conexión a internet envíen datos a redes privadas de Blockchain, para crear registros de las transacciones.
- **Sistema de votaciones**: La seguridad e inmutabilidad que aportan las Blockchain permite evitar los fraudes en las elecciones, dando más veracidad a los datos.

2.4 Elementos clave de la Blockchain

Principalmente hay tres elementos clave dentro de las Blockchain:

- **Participantes:** Estos son todos los que van a jugar un papel dentro de la Blockchain, ya sea las empresas que administran la red (en el caso de las privadas o híbridas), los mineros (en el caso de las públicas), o los simples usuarios. Dependiendo del tipo de Blockchain que sea, los diferentes participantes tendrán un acceso más o menos libre (libertad total en el caso de las Blockchain públicas, o restringida en el resto) y los diferentes permisos.
- **Activos:** Una vez conocidos los participantes, se debería saber cuáles van a ser los activos, también conocidos como tokens, que se va a intercambiar en la red durante las diferentes transacciones.
- **Transacciones:** Este elemento es muy importante pues permite saber cómo se va a movilizar la red. A través de estas se registran las interacciones que ocurren en la red, que pueden ir desde la transferencia de los activos, hasta la creación de un nuevo bloque.

2.5 Smart Contracts

De forma simplificada, los Smart Contracts son contratos los cuales se ejecutan a la hora de realizar transacciones entre algunas partes, sin la necesidad de intermediarios o terceros, como una especie de contratos no físicos. Estos fueron ideados por Nick Szabo [16], un catedrático de ciencia computacional americano.

Estos contratos son líneas de código, que se ejecutan siempre que se cumplan una serie de condiciones impuestas. Por ello se afirma que fueron ideadas para dar más seguridad que en los contratos tradicionales a las transacciones.

En la actualidad, estos se utilizan en las redes de Blockchain, lo que le da seguridad e inmutabilidad puesto que una vez han sido subidas a la cadena, ya no son modificables por ninguna de las partes, lo que provoca que haya más transparencia y seguridad entre los participantes. Siendo las conocidas Bitcoin y Ethereum, aunque en estas hay diferencias.

- En las redes de Bitcoin, los Smart Contracts solamente se utilizan en aspectos relacionados con las transacciones de su criptomoneda, el Bitcoin.
- En las redes de Ethereum, hay una gran variedad de utilidades de estos contratos, por ejemplo, el que se realizará en el trabajo más adelante.

El funcionamiento de los Smart Contracts es bastante sencillo, pues el primer paso es que exista el contrato hecho por un programador, una vez encontrado el código a ejecutar, se observan los datos necesarios para realizar la transacción, se definen y se ejecuta el contrato. Una vez

ejecutado, se verifica la cartera del cliente para darle validez, y al validarse la cartera se cobra la transacción y se ejecuta por sí mismo.

Una vez conocido el funcionamiento de los Smart Contracts, hay que ver sus beneficios respecto a los contratos tradicionales:

- Reducción de los gastos: esto se debe a la no necesidad de intermediarios, ya sea personas o las comisiones.
- Agilización del proceso: al ser los Smart Contracts autónomos y, el hecho de que no haya contacto humano intermedio agiliza mucho el proceso y aumenta la velocidad de la transacción.
- Independencia: por lo mismo que la agilización, al no haber contacto humano las gestiones las realizan los propios participantes.
- Seguridad y fiabilidad: al encontrarse en la red distribuida, el contrato es inmutable ya que se encuentra duplicado por toda la red en la cadena. Esto también provoca que no se pueda perder y, que en el caso de que una de las partes no cumpla, la otra se encuentre protegida.
- Precisión: al ser automatizadas, las transacciones se protegen de posibles errores humanos.

Parte 3. Fundamentos teóricos de Ethereum

3.1 Qué es Ethereum

Ethereum es una red descentralizada basada en el código abierto y la tecnología Blockchain, que principalmente sirve para la ejecución de contratos inteligentes entre dos partes. Los desarrolladores pueden crear y publicar aplicaciones distribuidas en la plataforma.

Las transacciones o intercambios se realizan con el pago de Ether, que es la moneda o token que proporciona Ethereum. El Ether tiene varios usos, sirve tanto como pago para realizar una operación computacional en la red de Ethereum, como moneda de con la que se realizan las transacciones, e incluso como recompensa para los mineros.

La red Ethereum, también cuenta con una máquina virtual descentralizada EVM, Ethereum Virtual Machine, que no es otra cosa que un conjunto de nodos los cuales sirven para ejecutar un conjunto de scripts o instrucciones que dan una versatilidad a la hora de realizar las operaciones.

Finalmente, debido a que no todas las operaciones tienen el mismo gasto o esfuerzo computacional, aparece la figura del gas, el cual de forma resumida podríamos decir es la prima que se necesita para hacer la transacción exitosamente.

En resumen, Ethereum necesita una serie de elementos para funcionar:

- La máquina virtual de Ethereum EVM
- El Ether
- El gas

3.1.1 Máquina virtual de Ethereum o EVM

La Máquina virtual de Ethereum es un entorno que sirve para la ejecución de los Smart Contracts, que se encuentra en funcionamiento continuo, ininterrumpido e invariable, pues es donde se encuentran la totalidad de las cuentas de Ethereum y sus Smart Contracts. También es la EVM quien define las normas para que se calcule un nuevo estado válido de bloque a bloque.

La EVM es una pila de registro de 256 bits, diseñada para ejecutar los códigos como el desarrollador pretende. El código se ha implementado para su codificación en algunos lenguajes como C ++, Java… Pero después de dicha codificación, ya se compila en bytecode [31] EVM.

Durante su ejecución, la EVM tiene una memoria temporal que no es conservada entre transacciones. Sin embargo, los contratos sí que contienen un Merkle Patricia Trie [2] de almacenamiento, el cual se asocia a la cuenta que los despliega.

Un Merkle Patricia Trie es la organización de los datos para una finalidad, esto permite que haya una rápida verificación y eficiencia en una Blockchain la cual tiene una considerable cantidad de datos, a pesar de que el creador lo pensara inicialmente para las firmas digitales. La primera vez que se utilizó este árbol en relación con la Blockchain fue con el bitcoin donde utiliza los hashes para verificar los datos. Estos árboles se implementan de tal forma que, realizando los hashes a las diferentes transacciones, para luego almacenarlos en un nodo por hash, siendo este el más específico (de ahí que se llame nodo hoja). Una vez obtenido un nodo hoja por cada transacción realizada, se van juntando por parejas para formar otros nodos, pero que tienen dos hashes de la rama anterior, realizando un hash a cada nodo obtenido. Este modus operandi se realiza todas las veces necesarias hasta llegar hasta el nodo raíz, donde todas las transacciones del bloque ya están dentro de un nodo que tiene su hash. Siendo este último hash el que se almacena en el encabezado del bloque.

El bytecode compilado en el Smart Contract se ejecuta con códigos de operación de la EVM, los cuales realizan las operaciones comunes en las pilas XOR, AND, etc. Aunque en este caso la EVM también añade operaciones de pilas que son enfocadas a las Blockchain, como son el ADDRESS, BLOCKHASH…

Volviendo con la EVM, algunas características de la EVM son:

- Un gran nivel de seguridad: la EVM está enfocada a la seguridad, pues el código que se ejecuta en ella no suele ser confiable para introducirla en la red, por eso impone restricciones para la ejecución:
    - La ejecución de una operación computacional no es totalmente gratuita, pues se debe pagar una tarifa, y así impedir algunos ataques.
    - La ejecución de un programa es aislada, por lo que no puede acceder o modificar estados de otros programas. Derivado de este, un programa no puede ejecutar otro programa EVM.
    - La ejecución de un programa produce transacciones de estado iguales para implementaciones que comiencen con estados idénticos. El estado de las transacciones solo cambia si estas son válidas, lo cual se produce al ser validadas con el proceso de minería. El cual, al igual que en todas las plataformas Blockchain involucra a los mineros, los cuales compiten para ser el creador del bloque y obtener la recompensa. Suele ser económica a través de Ether, en el caso de Ethereum.
- Estas son descentralizadas: la EVM se encuentra totalmente descentralizada, pues es un conjunto de miles de nodos que ejecutan la máquina de forma coordinada entre ellas. Esto

provoca la descentralización pues la máquina seguirá funcionando siempre que haya al menos un nodo, y aunque desaparecieran muchos de estos nodos, la maquina continuaría en ejecución siguiendo sus instrucciones.

- Acceso ilimitado: gracias al funcionamiento descrito anteriormente, la EVM permite que se acceda al sistema desde todas las partes del planeta sin ninguna censura. Lo importante es evitar la aparición de terceros que pueden modificar o alterar sus funciones.
- Permite el desarrollo de gran cantidad de aplicaciones descentralizadas o dApps [25] (decentraliced apps), pudiendo estas aplicaciones ejecutarse sobre una misma cadena sin entorpecer su funcionamiento o al de las otras ejecuciones que haya sobre esa cadena.
- Puede ejecutar un conjunto de códigos de operación llamados OP_CODES, que son comandos que sirven para que la EVM ejecute alguna acción. Sin estos comandos la EVM no podría funcionar.

3.1.2 Ether

El Ether es la moneda o token que se utiliza en Ethereum, esta se consigue a través de transacciones de compra de monedas o como recompensa para los mineros al añadir bloques a la cadena. Esta se utiliza a la hora de realizar las transacciones. Por lo que es una de las bases de la red Ethereum. Las cuentas de Ethereum pueden tener una cartera donde se guarda el Ether que esta contiene.

La validez de la moneda obtenida depende de la cadena de bloques, pues puede pasar la casualidad de que dos mineros obtengan el bloque simultáneamente, pero ahí hay un problema, tenemos dos cadenas. La resolución de este problema es bastante sencilla, ambas cadenas son válidas por el momento, más en concreto hasta que se resuelva el siguiente bloque. Al resolverse el siguiente bloque, para conseguirlo se habrá utilizado uno de los dos bloques que se habían obtenido anteriormente, por lo que la cadena donde se encuentra ese bloque pasa a ser la valida, invalidando la otra cadena, y por lo tanto el bloque de esa cadena, reduciendo la recompensa a su minero.

En lo que representa a la gestión del Ether, en 2014 con su oferta inicial fue:

- 60 millones de Ethers fueron creados.
- 12 millones se distribuyeron entre la fundación Ethereum y contribuyentes al desarrollo.
- 5 Ethers se creaban al cerrar un bloque VÁLIDO.
- 3 Ethers se creaban al cerrar un bloque tío, un bloque que se encuentra fuera de la cadena principal.

Para medir la cantidad del Ether, se emplean diferentes nombres por cantidades, de menor a mayor [28]. Esto se representa en la Tabla 1:

Nombre Unidad	Equivalencia en Wei
wei	1
kwei	1000
Kwei	1000
babbage	1000
femtoether	1000
mwei	1000000
Mwei	1000000
lovelace	1000000
picoether	1000000
gwei	1000000000
Gwei	1000000000
shannon	1000000000
nanoether	1000000000
nano	1000000000
szabo	1000000000000
microether	1000000000000
micro	1000000000000
finney	1000000000000000
milliether	1000000000000000
milli	1000000000000000
ether	1000000000000000000
eth	1000000000000000000
kether	10000000000000000000
grand	10000000000000000000
mether	1000000000000000000000
gether	1000000000000000000000000
tether	1000000000000000000000000000

Tabla 1. Equivalencias Unidades Ether

3.1.3 Gas

El gas no es otra cosa que la medición del esfuerzo computacional realizado en la red para la ejecución de una operación, debido a que todas las operaciones realizadas en la red de Ethereum no son igual de complejas. Por eso, para la realización de una operación hace falta una prima para conseguir la transacción con éxito total.

Las comisiones del gas se pagan en Ether, pues esta es la moneda o token de la red, aunque estos precios se indican en gwei o gigaWei, que sigue siendo menor que un ether pues 1 Gwei es equivalente a nanoEther.

Por todo eso, para que se ejecute una transacción, el emisor debe fijar un límite de gas y una cantidad que está dispuesto a pagar por el gas de ella. Obviamente, si el emisor no tiene suficiente gas, la transacción se invalida.

3.2 Smart Contracts en Ethereum con Solidity

Un Smart Contract en resumen es un programa que se ejecuta en la Blockchain de Ethereum. Se forma a partir de un conjunto de funciones y datos que residen en una dirección específica de la Blockchain de Ethereum.

Estos contratos son como una especie de cuentas de Ethereum, es decir, que tienen un pago y pueden ser objeto de transacciones. A pesar de eso, no hay ningún usuario que los controle, sino que estos se despliegan y se ejecutan en la red. Para que las cuentas asociadas a los usuarios puedan interactuar con un Smart Contract, esta tiene que enviar una transacción para ejecutar una función definida en el contrato, no el contrato. Los Smart Contract no pueden borrarse por defecto, y las interacciones con ellos son inalterables.

Cualquier persona puede escribir un Smart Contract y desplegarlos en la red. Solamente tiene que aprender a codificar en un lenguaje y tener suficiente Ether para poder desplegar el contrato en la red, siendo el movimiento este una simple transacción, por lo que se tiene que pagar un gas al igual que en una transferencia de Ether, La diferencia reside en que el coste del gas del despliegue de un contrato es mucho más elevado que el de una transferencia de Ether.

Para poder programar en la Blockchain de Ethereum hay varios lenguajes de programación, en concreto este proyecto se centra en Solidity [40], la cual se detallará más adelante en la sección de tecnologías utilizadas.

Los Smart Contracts en Ethereum programados en Solidity se utilizan en una gran variedad de aplicaciones, ya sea la creación de juegos, redes sociales, o sistemas de votaciones. También se pueden crear tokens como es el ERC-20 [32]. Para realizarlos es necesario un conocimiento básico sobre programación y sobre todo de la Blockchain de Ethereum. Los desarrolladores deben tener la capacidad de escribir un código eficaz y seguro que garantice el correcto funcionamiento y evitar su vulnerabilidad ante los ataques.

En resumen, los Smart Contracts son muy útiles a la hora de crear aplicaciones descentralizadas en la Blockchain de Ethereum. El uso de estos está creciendo bastante y se espera que en futuro tenga más importancia dentro de la tecnología Blockchain.

Parte 4. Especificación de los requisitos de la aplicación desarrollada

Entendiendo ya los conceptos teóricos que se van a aplicar, se van a dar las especificaciones de los requisitos.

4.1 Introducción

El propósito que tiene la aplicación es el de la realización de un sistema de votaciones, adaptable para cualquier sistema de votación, que aporte seguridad y veracidad. Esto se debe, a que se le otorga inmutabilidad, dando mayor validez a los resultados gracias a la tecnología en la que se basa.

En este caso el proyecto se ha realizado en un entorno local, pues para utilizarlo en un entorno productivo deberían realizarse algunos cambios que se expondrán en el apartado 7.2.

Lo que se busca es que, de forma gratuita, o minimizando los gastos lo máximo posible, sea posible la realización de unas votaciones de forma que no se puedan corromper, a través de la tecnología Blockchain.

También hay que explicar una serie de definiciones que aparecen en la Tabla 2.

Usuario	Persona utiliza el servicio, teniendo una cartera, para participar en el sistema
Red	Es la red de Blockchain en la que se encuentra el contrato
Contrato	Es la aplicación que realiza el servicio

Tabla 2. Definiciones

Para recoger y especificar los requisitos de la aplicación, se han seguido las especificaciones definidas por “The Institute of Electrical and Electronics Engineers” [20].

4.2 Descripción general

4.2.1 Perspectiva del producto

La aplicación debe funcionar como un sistema de votación, que permita realizar unas votaciones, en cualquier ámbito, de forma segura.

- **Requisitos del sistema**

  El sistema debe tener conexión directa con una red de Blockchain, pues es necesario pues el contrato se encontrará dentro de esta, y cualquier cambio dentro del sistema será añadido a ella.

- **Interfaz de usuario**

  Es necesario que el usuario tenga, por un lado, acceso a un navegador web, y por otro lado a una cartera de la Blockchain utilizada, en este caso de Ethereum.

- **Interfaz Hardware**

  Es necesario un ordenador con acceso a internet y a la Blockchain.

- **Interfaz de comunicación**

  Es necesario que se pueda acceder a la red Blockchain, tanto para saber el estado en el que se encuentra, como para utilizar el contrato.

- **Requisitos de adaptación de la aplicación**

  La aplicación se debe utilizar en un ordenador, el cual debe tener acceso a la red Blockchain y tener una cartera en ella.

4.2.2 Funciones del sistema

Se podría decir que, a modo de introducción y en general, la aplicación permite a los usuarios realizar lo siguiente:

- Consultar su cartera.
- Inscribir candidatos.
- Consultar los candidatos.
- Realizar su voto.
- Consultar los votos.
- Ver el ganador.

4.2.3 Características de los usuarios

El usuario que se busca en la aplicación es el que quiere participar en un sistema de votaciones. En principio lo que se buscaría es un grupo de personas que desean realizar unas votaciones y busquen la fiabilidad que la Blockchain aporta.

4.2.4 Restricciones, dependencias y requisitos tecnológicos

Para el uso de la aplicación, lo que se busca es que el usuario que desea utilizar la aplicación debe utilizar un navegador que sea compatible con HTML, CSS y JavaScript, pues estos lenguajes se utilizan generar la interfaz de usuario (web), así como para comunicarse bidireccionalmente con la aplicación Solidity. En la actualidad ya no es un gran problema, pues la gran mayoría de los navegadores soportan los lenguajes.

Más en concreto, JavaScript se va a utilizar para enlazar solidity con HTML, pasando la información entre ellas.

También el usuario debería algún software de criptomoneda que permita interactuar con la Blockchain de Ethereum, pues esta va a ser la utilizada en la aplicación.

4.3 Requerimientos funcionales

4.3.1 Usuarios

En el caso de la aplicación los usuarios deberán estar registrado en la red de Ethereum, pues sin estar registrados, no podrán realizar ninguna transacción y no podrán hacer ningunas de las funciones que el sistema ofrece.

Registrarse en esta red es tan sencillo, como a través de cualquier software que lo permita, como es el caso de metamask, crear una cartera de Ethereum. Por otra parte, al estar en una red de pruebas, se puede utilizar Blockchains de prueba.

Los Usuarios podrán acceder a todas las funciones del sistema descritas en el apartado 4.2.2. como son:

- Consultar su cartera.

  Nada más entrar, podrá observar cual es la identificación de su cartera virtual

- Inscribir candidatos.

  El usuario puede inscribir todos los candidatos que desee, eso si para inscribirlos tendrá que cumplir unas características, pudiéndose modificar estas, para las distintas elecciones.

- Consultar los candidatos.

  El usuario podrá consultar el nombre y la identificación de los candidatos que se presentan a las votaciones.

- Realizar su voto.

  El usuario podrá realizar un solo voto a algún presentado, pues si ya ha votado no se le permitirán más.

- Consultar los votos.

  El usuario podrá consultar el número de votos que tiene X candidato.

- Ver el ganador.

El usuario podrá consultar el candidato que va ganando en el momento de su consulta.

4.4 Atributos

4.4.1 Seguridad

Principalmente, la seguridad de la aplicación reside en la que aporta el propio sistema de Blockchain, pues está ya proporciona los cifrados correspondientes para que no se pueda corromper la aplicación, ya sea por alteración de los datos o por la falsificación, pues todo se encuentra registrado en la propia red.

4.4.2 Disponibilidad

El usuario tiene accesibilidad completa a la aplicación siempre que desee, siempre que este se encuentre registrado en la red. Pues el contrato en el que se basa se encuentra en la red de Ethereum.

4.4.3 Portabilidad

En la actualidad, al encontrarse solamente en la red de pruebas, deberá utilizarse un ordenador con acceso a internet, y un navegador, y a la red de Ethereum. Aunque, una de las posibles mejoras que se aportará en los apartados futuros, puede ser su implementación para otros dispositivos.

4.4.4 Mantenimiento

El desarrollo de las Blockchain y sus contratos requieren una mentalidad diferente a la que se tiene con la web tradicional, esto es debido a que los contratos funcionan de forma autónoma proporcionando una información transparente, inviolable e inmutable.

Para el fácil mantenimiento se han seguido unas prácticas de programación, que hacen que el código del contrato se más limpio.

Primeramente, se ha seguido una nomenclatura sencilla pero que te dicen lo que se va a realizar en ella, me refiero a las funciones o las variables. Por ejemplo, una función que se denomina “ganador”, obviamente te va a dar el ganador de las votaciones.

Además, en el contrato se ha seguido un orden natural, como, por ejemplo, para ver que candidatos se han presentado a las votaciones, primero se deberán a ver presentado alguno, por lo que primero se observa la presentación y posteriormente quien se ha presentado.

En este caso, cada vez que se interactúe con el contrato se debe pagar una pequeña cantidad, en el caso de la red de pruebas es simbólico, por lo que se debe firmar con la cartera usada, cosa que da el feedback al usuario de que ha funcionado, al igual que si hay algún error en la transacción, con el control de errores que identifica por qué no se puede realizar esa transacción.

Finalmente, hay que comentar que, a lo largo del código, se han ido añadiendo una serie de comentarios para que cualquier usuario que lea el código, entienda el funcionamiento de cada interacción y conjunto de líneas de código.

Parte 5. Análisis

5.1 Casos de uso

Esta es la primera fase del análisis, que permite representar las interacciones entre los actores (en este caso solamente está el usuario) y las funcionalidades que la aplicación (Smart Contract) aporta.

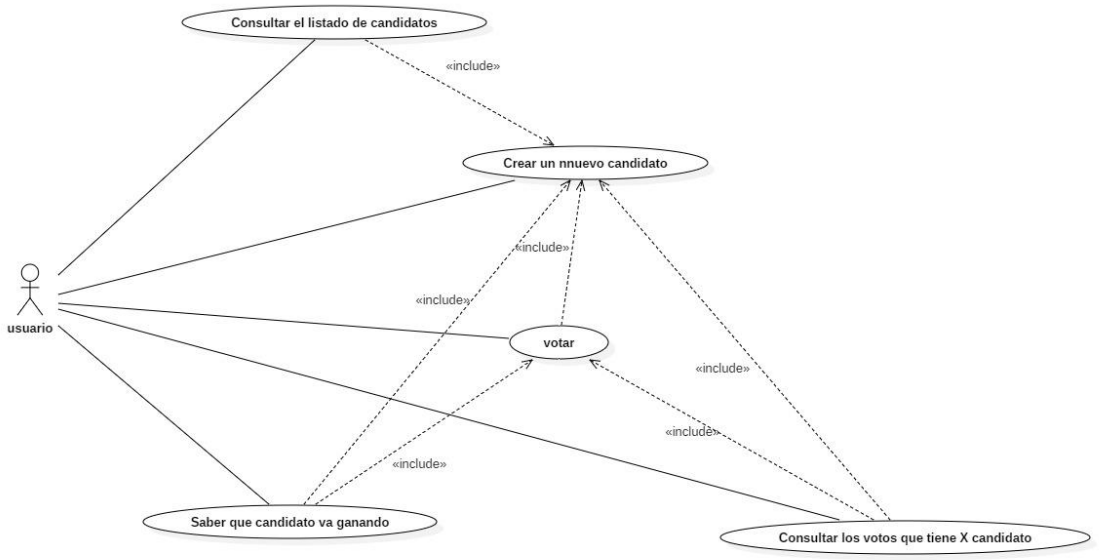

Figura 1. Diagrama de casos de uso de la aplicación, realizada con la aplicación StarUML

Tras mostrar el diagrama general, paso a describir los casos de uso individuales

Caso	Crear un nuevo candidato	
Actores	Usuario	
Precondiciones	Que se haya conectado una cartera de Ethereum	
Acción de un actor 1. Usuario introduce los datos del candidato selecciona ‘añadir candidato’. 3. Aparece el software de criptomonedas preguntándole si quiere seguir adelante con la transacción, el usuario pincha en ‘aceptar’.		**Acción del sistema** 2. El sistema comprueba unos requisitos: Primero, que la edad del candidato es mayor a 18, segundo que la identificación es de 9 dígitos y todavía no se encuentra en uso. 4.El candidato queda añadido a la Blockchain.
Secuencias alternativas	2.Si encuentre error en uno de los requisitos, muestra un mensaje de error explicando el requisito erróneo y se vuelve a antes de seleccionar ‘añadir candidato’. 3. Si no el usuario pincha en ‘no aceptar, se vuelve a antes de seleccionar ‘añadir candidato’.	
Postcondiciones	El candidato añadido queda inscrito en la cadena.	

Tabla 3. Especificación expandida de ‘crear un nuevo candidato’

Caso	Consultar listado de candidatos	
Actores	Usuario	
Precondiciones	Que haya sido inscrito algún candidato de forma correcta	
Acción de un actor En este caso el actor no realiza nada		**Acción del sistema** 1. Si hay inscrito algún candidato, aparece un listado de ellos en el apartado de 'candidatos'
Secuencias alternativas	1. Si no hay inscrito ningún candidato, no aparece nada en el apartado de 'candidatos' a la espera de que se inscriba uno.	
Postcondiciones	-	

Tabla 4. Especificación expandida de 'Consultar listado de candidatos'

Caso	Votar	
Actores	Usuario	
Precondiciones	Que se haya conectado una cartera de Ethereum	
Acción de un actor 1. El usuario introduce la identificación del candidato que desea votar y selecciona 'votar'. 3. Aparece el software de criptomonedas preguntándole si quiere seguir adelante con la transacción, el usuario pincha en 'aceptar'.		**Acción del sistema** 2. El sistema comprueba los siguientes requisitos: primero, que la cartera de la que se pretende votar no haya votado todavía y segundo, que la identificación a la que se pretende votar exista entre los candidatos. 4. El voto que se relaciona al candidato con el identificador correspondiente
Secuencias alternativas	2. Si encuentre error en uno de los requisitos, muestra un mensaje de error explicando el requisito erróneo y se vuelve a antes de seleccionar 'votar'. 3. Si no el usuario pincha en 'no aceptar, se vuelve a antes de seleccionar 'votar'.	
Postcondiciones	Se le suma +1 al número de votos que tiene asignada esa identificación	

Tabla 5. Especificación expandida de 'votar'

Caso	Consultar los votos de X candidato	
Actores	Usuario	
Precondiciones	Que se haya inscrito algún candidato y/o se haya votado	
Acción de un actor 1. El usuario introduce la identificación del candidato que desea ver y selecciona 'ver votos'.		**Acción del sistema** 2. El sistema saca el número de votos que tiene el identificador demandado, mostrando ese número.
Secuencias alternativas	2. En el caso de que el identificador no exista se mostrará por defecto que el número de votos es 0.	
Postcondiciones	-	

Tabla 6. Especificación expandida de 'Consultar los votos de X candidato'

Caso	Saber qué candidato va ganando	
Actores	Usuario	
Precondiciones	Que se haya inscrito algún candidato y/o se haya votado	
Acción de un actor El usuario no realiza nada		**Acción del sistema** 1. El sistema recorre todo el array de Candidatos con sus votos, viendo quien tiene más, mostrándolo en el apartado 'Ganador'
Secuencias alternativas	1. Si no hay inscrito ningún candidato, no aparece nada en el apartado de 'ganador' a la espera de que se inscriba uno. 1. En el caso de que no haya habido ningún voto se indicará, o bien el nombre del único candidato presentado o bien que hay empate, pues es la realidad en ambos casos. 1. En el caso de que haya votos y empate, aparecerá que hay un empate, por lo que no hay ganador en ese momento.	
Postcondiciones	-	

Tabla *7*. Especificación expandida de 'Saber qué candidato va ganando'

5.2 Diagramas de secuencia

A continuación, se mostrarán los diagramas de secuencia que han sido realizados a través de un software especializado en diseño UML llamado StarUML [45].

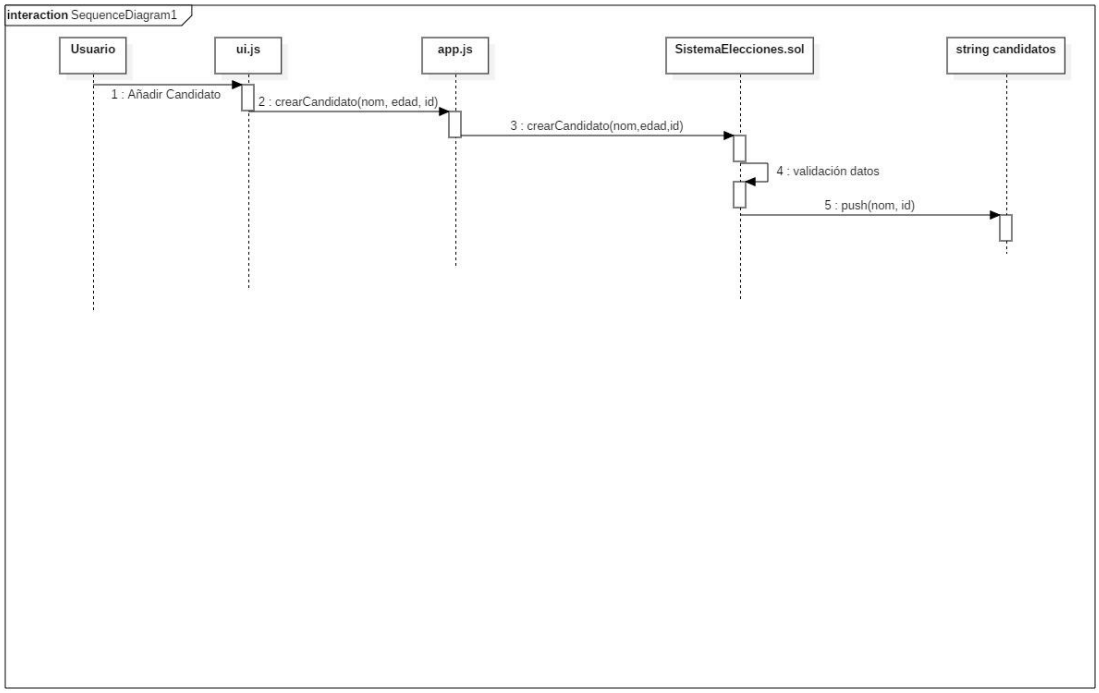

Tabla *8*. Diagrama de secuencia de 'crear un nuevo candidato'

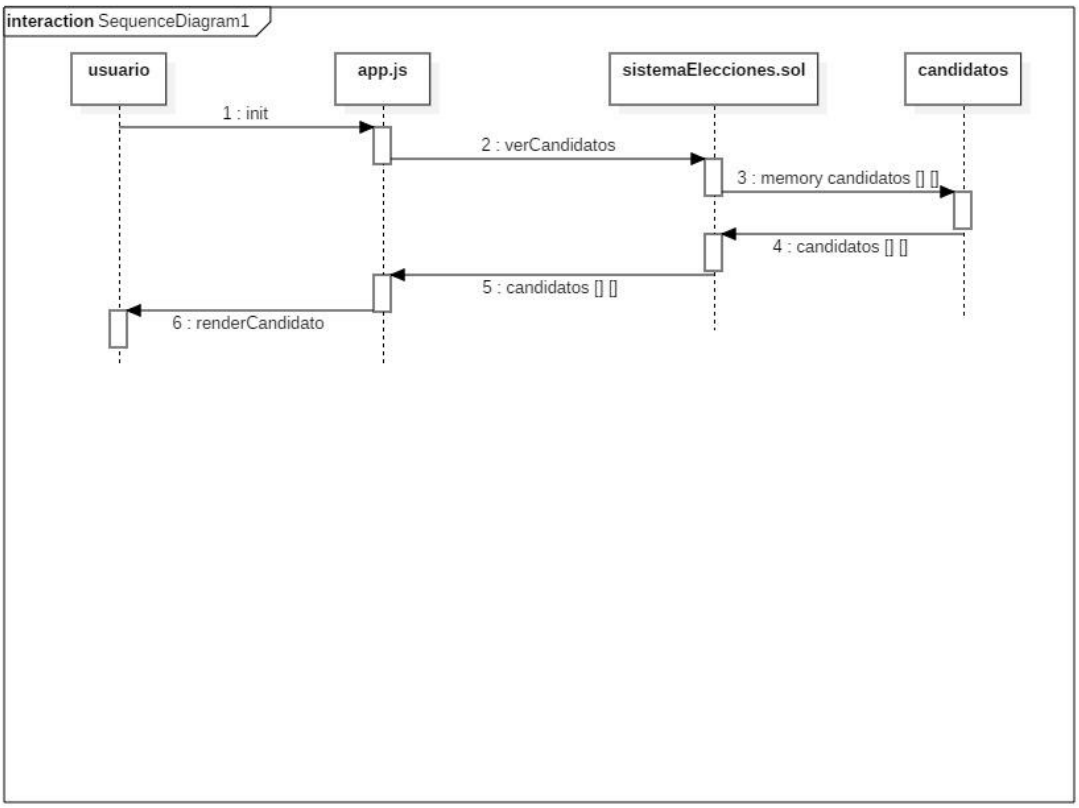

Tabla *9*. Diagrama de secuencia de 'Consultar listado de candidatos'

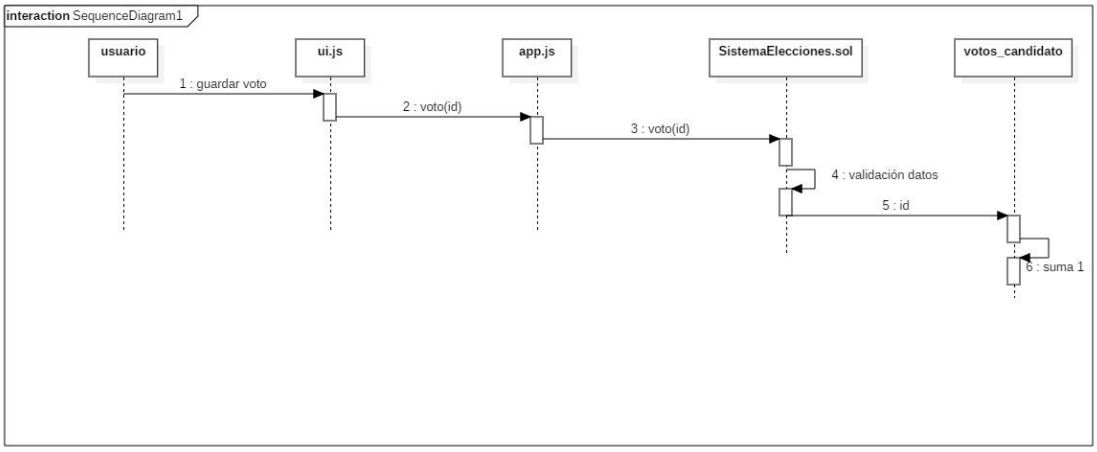

Tabla *10*. Diagrama de secuencia de 'votar'

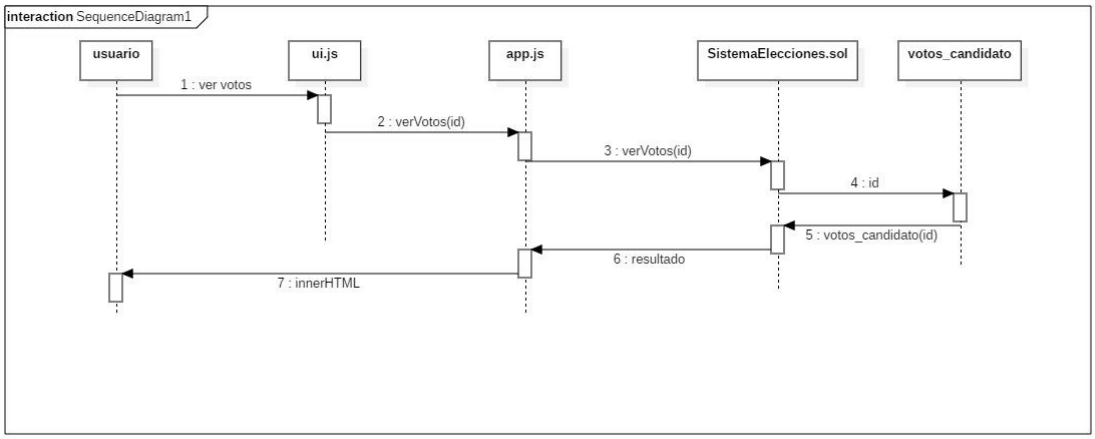

Tabla *11*. Diagrama de secuencia de 'Consultar los votos de X candidato'

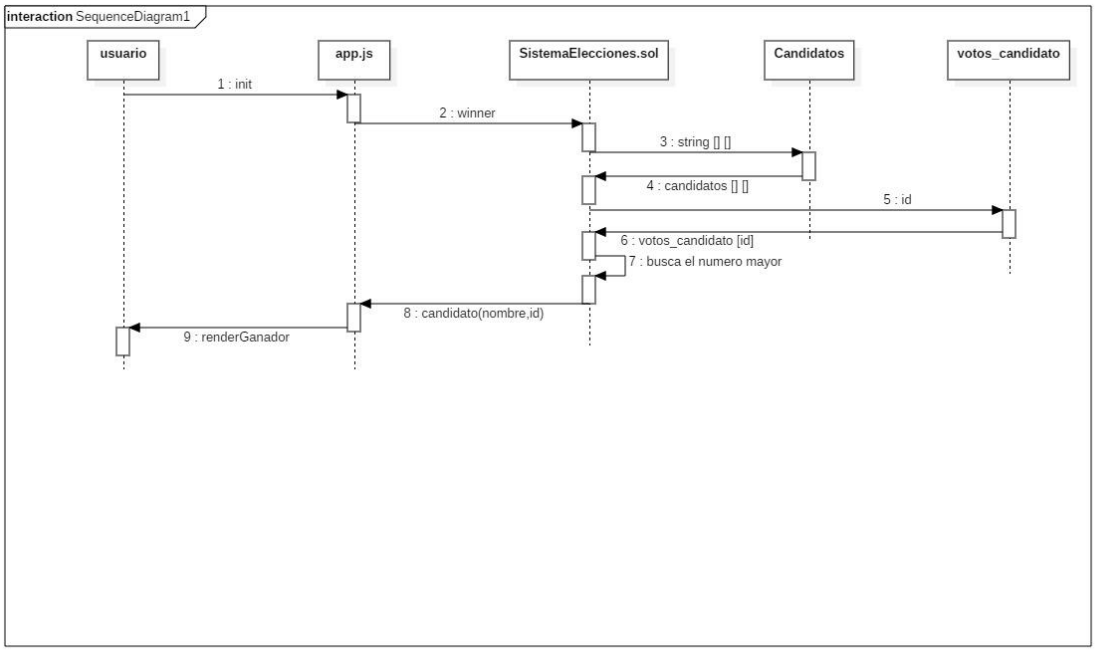

Tabla 13. Diagrama de secuencia de 'Saber qué candidato va ganando'

5.3 Análisis de usabilidad

Continuando con el proceso de análisis, hay que hablar de la usabilidad, que no es otra cosa que la sencillez que una página web ofrece a sus usuarios que la utilizan, por lo que las webs con buena usabilidad son aquellas de las que diseño y desarrollo es pensado para los usuarios, denegándoles una gran dificultad a la hora de su uso.

Para realizar dicho análisis voy a realizar las "heurísticas de usabilidad", que consiste en identificar las heurísticas o principios, en este caso voy a utilizar las de la lista de Jakob Nielsen [34] y aplicarlas en la aplicación para ver su evaluación

Heurísticas de Jackob Nielsen:

- Visibilidad del estado del sistema

  En este caso, hay un gran problema, pues para la actualización de la información es necesario un refresco de página manual del usuario
- Coincidencia entre el sistema y el mundo real

  Viendo la interfaz, se observa la claridad y sencillez de palabras a la hora de realizar las acciones, pues se usa un lenguaje claro y fácil de entender para cualquier tipo de usuario
- Control y libertad del usuario

  En este caso la libertad de usuario es difícil de implementar debido al funcionamiento propio de las Blockchain, pues revertir las acciones implica una gran dificultad
- Consistencia y estándares

  Como se observa en la interfaz los colores son coherentes y como se diría vulgarmente "no hacen daño a la vista". Por otra parte, si se observa el código del contrato, las anotaciones y explicaciones son constantes a la par de que los nombres de las funciones y datos hacen referencia directa a su uso
- Prevención de errores

  La prevención se encuentra en la aplicación, pues antes de que el usuario cometa un error, se le prohíbe la acción explicándose el motivo
- Reconocimiento en lugar de recordar

  El reconocimiento se aplica mostrando por pantalla los nombres e identificadores de los candidatos, pues casi todas las acciones que hace el usuario son recordando estas informaciones las cuales se encuentran siempre en pantalla
- Flexibilidad y eficiencia de uso

  Esta heurística no se encuentra bien aplicada, pues la web es muy rígida y no permite una gran variedad de uso
- Diseño estético y minimalista

La web realizada consta de una página donde se pueden realizar todas las funcionalidades que se ofrecen, sin mucho texto

- Ayuda a los usuarios a reconocer, diagnosticar y recuperarse de los errores
  En este caso, los errores son fácilmente reconocibles, pues gracias al control de errores, aparece el error que se está cometiendo al intentar realizar alguna acción

En conclusión con las heurísticas, por lo general las aplicación cumple la mayoría de estas, aunque se podría implementar una serie de funcionalidades para la mejoría de algunas, como por ejemplo un refresco automático después de cada transacción.

5.4 Análisis de la interfaz gráfica

En el caso de esta aplicación la interfaz gráfica consta de solamente una página, donde se muestra todo lo que se puede realizar en el Smart Contract.

Para ello se ha realizado, a través de la herramienta Balsamiq Wireframes [46], una prueba de cómo debería verse el programa después de ser realizado:

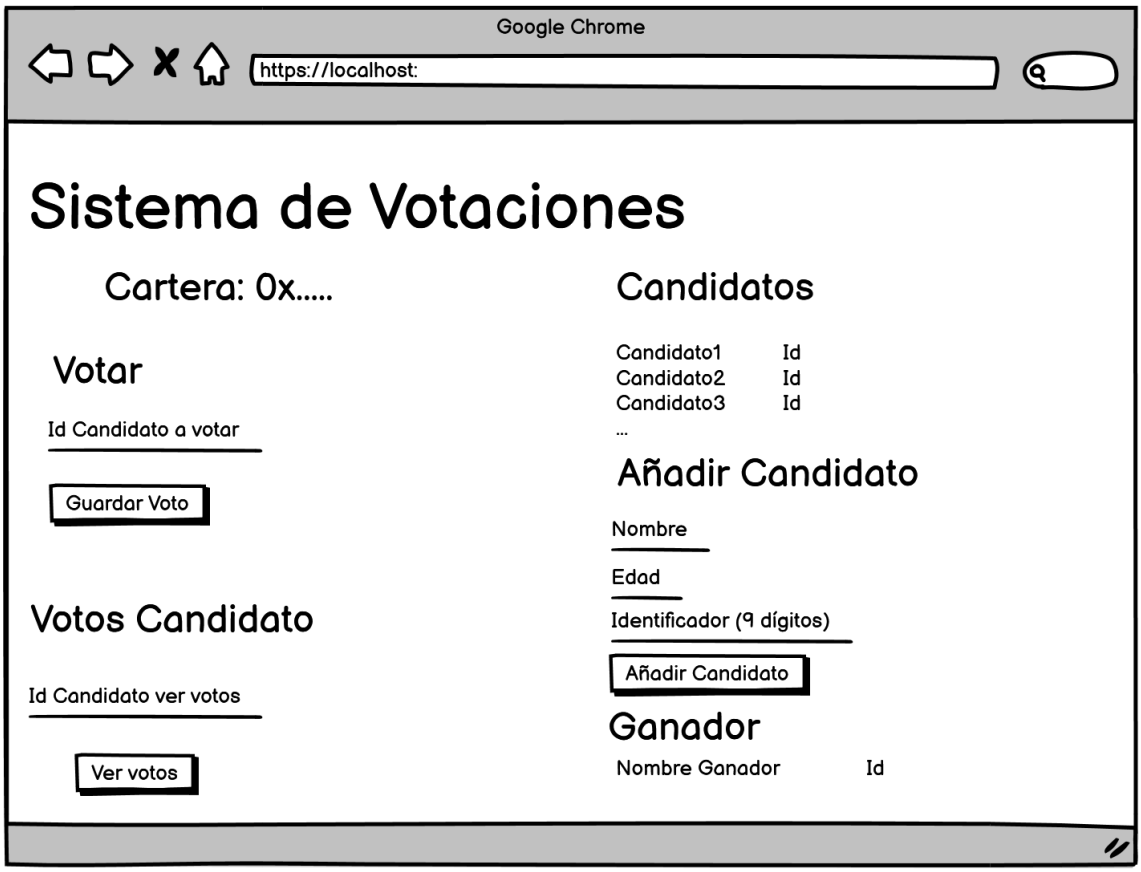

Figura 2. Ejemplo de cómo se vería la aplicación

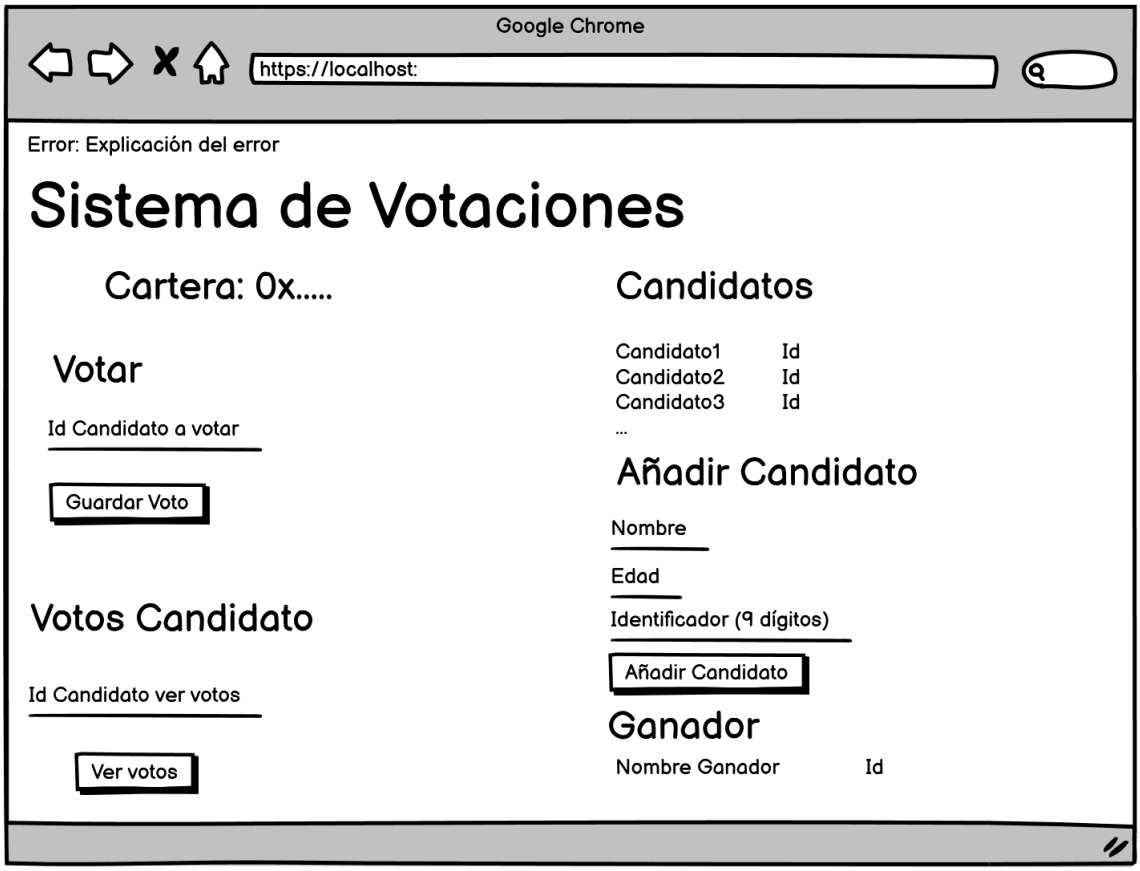

Figura 3. Ejemplo de cómo se vería la página con un error

Parte 6. Implementación

Tras la realización de la dApp, la interfaz se vería de la siguiente forma:

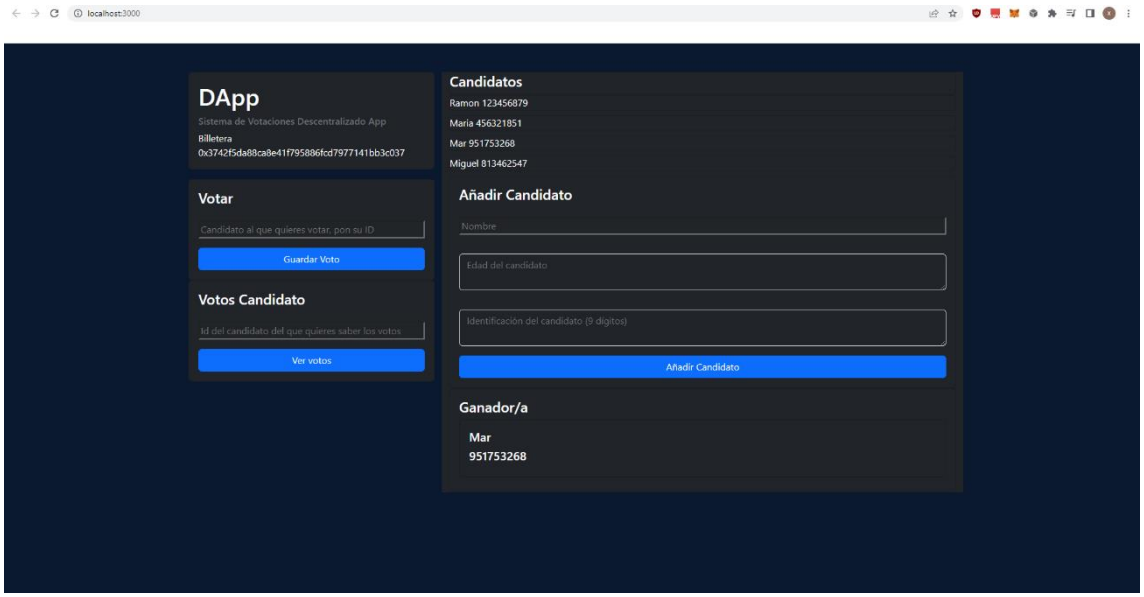

Figura 4. Como se vería la dApp

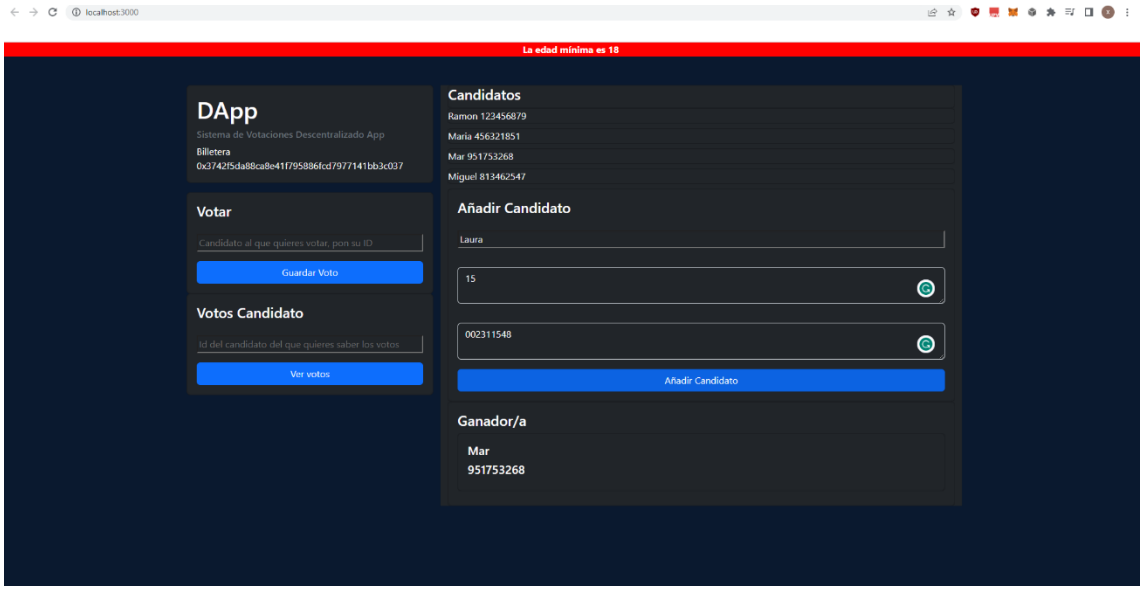

Figura 5. Como se mostraría un error en la dApp

Por otra parte, el proyecto se encuentra en el siguiente repositorio alojado en GitHub:

“https://github.com/XavierBernat/tfg-xavier/tree/main”.

Viendo lo expuesto en el apartado anterior, pues en ese se ha observado todas las características de la aplicación, además del análisis de sus funcionalidades y elementos que la conforman, queda ver a través de que se ha realizado.

Por eso, en este apartado del proyecto se da paso a la explicación de las tecnologías y herramientas que se han utilizado para la realización del proyecto.

6.1 Arquitectura del Proyecto

El concepto de arquitectura, dentro de un proyecto, se centra en la organización y estructura de este, pues es donde se expone sus divisiones, las interacciones entre sus componentes y la gestión de los datos.

En el caso del proyecto trabajado se pueden distinguir cuatro capas: la de usuario o cliente, la de la red Blockchain, la de la wallet y la del servidor web. Estas se interconectan de la forma descrita en la siguiente figura.

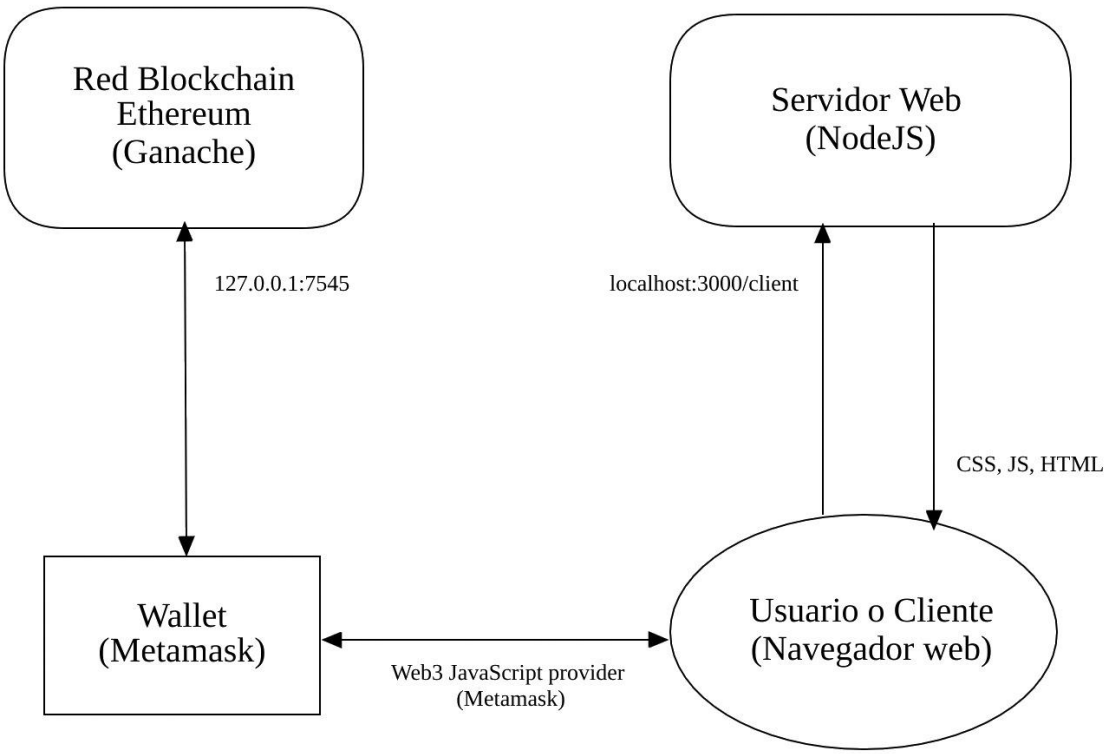

Figura 6. Arquitectura del proyecto

Pasando a las descripciones individuales de cada capa:

- **Capa de Wallet**

  Para crear una wallet de prueba, usamos una extensión para el navegador llamada MetaMask. Esta permite al usuario la gestión de las claves privadas y la firma de las transacciones realizadas en la red Blockchain. También inyecta una API de JavaScript global almacenada en la variable window.ethereum. Es un objeto WebProvider (MetaMask Ethereum Provider Object) que permite que el resto de código JavaScript pueda solicitar cuentas Ethereum, leer datos de la BLockchain a la que está conectada el usuario y se firmen tanto mensajes como transacciones. Por eso, podríamos decir que actúa como capa intermedia entre el navegador web o usuario y la Blockchain.

- **Capa de Cliente (Navegador)**

  En este proyecto, esta capa actúa principalmente como "middleware" principal, ya que su principal función, además de presentar la aplicación al usuario, radica en interconectar las diferentes capas de la aplicación. El código JavaScript ejecutado en el navegador web utiliza el objeto Provider de Ethereum proporcionado por MetaMask (comentado en la capa anterior) para comunicarse con la red Blockchain. También se comunica con el servidor web de forma local (localhost) en el puerto 3000.

- **Capa de red Blockchain**

  En esta capa es donde reside la lógica del programa, se realizan las transacciones y se almacenan los datos, a través de los diferentes bloques que se crean. Gracias a Web3Provider, se le permite al navegador enviar y obtener la información que desee a la red Blockchain e interaccionar con el Smart Contract.

- **Capa del Servidor Web**

  En este caso el servidor web solo se conecta con el cliente para proporcionarle los ficheros estáticos del frontend (JavaScript, hojas de estilo CSS y HTML). Esto se debe a que tanto la lígoca de negocio, como el almacenamiento de los datos se realiza en la red Blockchain, por lo que su función pasa a ser solamente la de proporcionar la información necesaria al navegador para que muestre la interfaz al usuario, pues la red Blockchain se comunica directamente con el navegador, no con el servidor web.

Resumiendo, la arquitectura del proyecto se basa en una pieza central (navegador), que se comunica con las otras tres capas (Wallet, red Blockchain y servidor web). Siendo una de ellas (wallet) una capa intermedia entre el navegador y la red Blockchain.

6.2 Tecnologías utilizadas

- **Solidity [40]**

  Este lenguaje de programación ha sido el utilizado a la hora de la realización del Smart Contract, pues este es con el que se realizan los contratos basados en la Blockchain de Ethereum.

  Esto se debe a que Solidity es un lenguaje de programación orientado a objetos. Este se basa en la sintaxis de C++ y está diseñado para ser fácil de aprender y de usar para los desarrolladores de software.

- **Bootstrap 5.2.3**

  Es un conjunto de herramientas para facilitar la creación de sitios web. Me ha servido para agilizar el diseño web, ya que he utilizado algunos de sus componentes, tanto como CSS como JavaScript.

- **Web3**

  Es un objeto provider de Ethereum proporcionado por la extensión de Chrome MetaMask. Esta es el enlace entre la aplicación web y la red de Ethereum, por lo que a través de ella se permite la interacción de las aplicaciones web con las redes Blockchain

- **Node.js**

  Este es un entorno de ejecución abierto y multiplataforma que se utiliza para crear aplicaciones de servidor y ejecutar código en el lado del servidor, como ocurre en el proyecto.

6.3 Herramientas utilizadas

- **Google Chrome**

  Este ha sido el navegador utilizado a la hora de recoger la información del proyecto. Además, también con él se han realizado las pruebas de funcionamiento del proyecto.

- **Remix [41]**

Esta IDE, es el que ha ayudado a la realización del contrato .sol previo a su unión con el resto. Ya que este ofrece una gran facilidad a la hora de escribir el código, y realizar pruebas de funcionamiento básicas.

- **Visual Studio Code [39]**

  Este editor de código ha sido el utilizado para la realización del programa, pues en él se ha añadido el código .sol que tiene el contracto, además del HTML y el JavaScript

- **NPM**

  Es un sistema de gestión de paquetes para entornos de ejecución JavaScript (en este caso Node.js). Permite, entre otras cosas, descargar, instalar y gestionar los módulos que contiene las funciones y bibliotecas para el desarrollo de aplicaciones node.

- **Metamask [42]**

  Esta extensión de Google, que también se puede utilizar como aplicación, es un software de criptomoneda, el cual se ha utilizado para la interacción con la Blockchain de Ethereum. Para ello se la han añadido cuentas de prueba, con Ether inifinito, pero que solo se pueden utilizar en pruebas.

- **Truffle [44]**

  Este framework ha sido utilizado durante el desarrollo del Smart Contract en la red de Ethereum, más en concreto ha sido el encargado de compilar y desplegar el contrato en la red, de prueba eso sí.

- **Ganache [43]**

  Esta es una red de Blockchain de Ethereum personal, que se utiliza para las pruebas de los contratos. Está ha permitido la realización del proyecto sin ningún costo en su red. También ha sido la encargada de dar las cuentas de prueba a Metamask

6.4 Explicación del código

6.4.1 Smart Contract

Principalmente, lo que se comentará en este subapartado se encuentra en el fichero **SistemaElecciones.sol,** dentro de *"contracts"*

Antes que nada, se debería comentar la parte previa a la creación del contrato:

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.19;
pragma experimental ABIEncoderV2;
```

Figura 7. Previo contrato

La primeria línea indica la licencia de software que aplica al código del proyecto. En este caso es la licencia MIT, la cual permite el uso, modificación y distribución del código. Por otra parte, el "*pragma solidity*" es para indicar la versión de Solidity que se está utilizando, mientras que el "*pragma experimental*" es para habilitar el uso del ABI encoder, que sirve para codificar y decodificar.

Ya dentro del propio contrato, se encuentran las diferentes funciones, que realiza las operaciones permitidas dentro del contrato, y derivando de el en la aplicación.

```
function crearCandidato(
    string memory _nombreCandidato,
    uint _edadCandidato,
    string memory _idCandidato
) public {
    //Hash de los datos del candidato
    require(_edadCandidato > 17, unicode"La edad mínima es 18");
    bytes32 hash_Candi = keccak256(
        abi.encodePacked(_nombreCandidato, _edadCandidato, _idCandidato)
    );
    //A continuación se verá si el id tiene un tamaño de 9 carácteres
    bytes memory idBytes = bytes(_idCandidato);
    require(
        idBytes.length == 9,
        unicode"El ID del candidato debe tener exactamente 9 caracteres"
    );
    // Verificar si ya existe un candidato con la misma ID
    for (uint i = 0; i < candidatos.length; i++) {
        if (
            keccak256(abi.encodePacked(candidatos[i][1])) ==
            keccak256(abi.encodePacked(_idCandidato))
        ) {
            // Si ya existe un candidato con la misma ID, lanzamos una excepción
            revert("Ya existe un candidato con la misma ID");
        }
    }

    //Almacenar hash de los datos del candidato ligados a su nombre
    ID_candidato[_nombreCandidato] = hash_Candi;

    //Almacenar nombre del candidato con funcion push
    //candidatos.push(_nombreCandidato);
    candidatos.push([_nombreCandidato, _idCandidato]);
}
```

Figura 8. Función crearCandidato

Esta función es la que se encarga de crear los nuevos candidatos, cosas a comentar de ella seria, los **require,** esto aparecerá a lo largo del contrato, pues son las condiciones que se debe cumplir para realizar la transacción.

Después de pasar los require se verifica la existencia de un candidato con esta id, se puede modificar para que lo que compruebe sea otro dato, recalcar que Keccak256 es una función hash, por eso se utiliza durante la codificación. Una vez verificado todo, se añade a la lista de candidatos con un simple push.

```
function verCandidatos() public view returns (string[][] memory) {
    // Inicializamos una nueva matriz de cadenas para almacenar los nombres y Id's de los candidatos
    string[][] memory nomId = new string[][](candidatos.length);

    // Recorremos la matriz candidatos y almacenamos los nombres e Id's de cada candidato en la nueva matriz
    for (uint i = 0; i < candidatos.length; i++) {
        // Creamos una nueva sub-matriz para almacenar el nombre y los Id's del candidato actual
        nomId[i] = new string[](2);
        nomId[i][0] = candidatos[i][0]; // Almacenamos el nombre del candidato
        nomId[i][1] = candidatos[i][1]; // Almacenamos la Id del candidato
    }

    // Devolvemos la nueva matriz de nombres y Id's
    return nomId;
}
```

Figura 9. Función verCandidatos

Esta función se utiliza para mostrar un listado de nombres y las Id's que tienen relacionada, para ello a través de un for se van obteniendo los diferentes candidatos presentados, los cuales se guardan dentro de la matriz nomID, que posteriormente se devuelva a la llamada de la función

```
function voto(string memory _candidatoId) public {
    //Calculo del hush de la persona que ha intentado votar
    bytes32 hash_votante = keccak256(abi.encodePacked(msg.sender));

    //Verificación de que el votante NO ha votado
    for (uint i = 0; i < votantes.length; i++) {
        require(votantes[i] != hash_votante, unicode"Usted ya ha votado!");
    }

    //Se verifica la existencia del candidato
    bool existe = false;
    for (uint i = 0; i < candidatos.length; i++) {
        if (
            keccak256(abi.encodePacked(candidatos[i][1])) ==
            keccak256(abi.encodePacked(_candidatoId))
        ) {
            existe = true;
            break;
        }
    }
    require(existe, unicode"El candidatoindicado No existe");

    //Almacenamos el hash del votante dentro del array de votantes
    votantes.push(hash_votante);

    //Añadir un voto al candidato
    votos_candidato[_candidatoId]++;
}
```

Figura 10. Función voto

El uso de la siguiente función es el de añadir un nuevo voto. La cosa es que en el sistema que he implementado cada usuario puede añadir todos los candidatos que quiera, pero solo puede votar una vez, por eso lo primero que se revisa es que el candidato no haya votado todavía, con un require, explicado anteriormente, luego se revisa la existencia del candidato al que se quiere votar, en este caso se vota a las id's, una vez revisado se añade el voto al candidato.

```
function verVotos(string memory _candidatoId) public view returns (uint) {
    return votos_candidato[_candidatoId]; //Obtener los votos que tiene el candidato
}
```

Figura 11. Función verVotos

Está función es como verCandidatos, pero más simplificada, pues solo se desea ver el número de votos que tiene dicho candidatos, en el caso de poner un candidato inexistente devuelve 0.

```
function winner() public view returns (string memory, string memory) {
    //Esta variable va a obtener el nomre del ganador
    string memory idGanador = candidatos[0][1];
    string memory nomGanador = candidatos[0][0];
    bool empate;

    for (uint i = 1; i < candidatos.length; i++) {
        if (
            votos_candidato[idGanador] < votos_candidato[candidatos[i][1]]
        ) {
            idGanador = candidatos[i][1];
            nomGanador = candidatos[i][0];
            empate = false;
        } else {
            if (
                votos_candidato[idGanador] ==
                votos_candidato[candidatos[i][1]]
            ) {
                empate = true;
            }
        }
    }
    if (empate == true) {
        nomGanador = unicode"¡Hay un empate entre candidatos!";
        idGanador = "";
    }
    return (nomGanador, idGanador);
}
```

Figura 12. Función winner

Está función saca el nombre e id del candidato que tiene más votos, o un texto en caso de empate. Para ello recorre un for comparando votos, siempre quedándose en la comprobación el que tiene más votos de los dos. En caso de que durante la comprobación salga empate se marca con un booleano, el cual desaparece si en la siguiente comparación el nuevo candidato supera el empate. Al final de recorrer el for se comprueba el booleano para saber si hay empate, si no se ve el ganador.

Para finalizar la parte del Contrato, me gustaría salir de la *"contracts"*.

Primero, dentro de *"migrations"* se encuentra "*2_deploy_Sistema.js"*

```
const SistemaElecciones = artifacts.require("SistemaElecciones");

module.exports = function (deployer){
    deployer.deploy(SistemaElecciones);
};
```

Figura 13. 1_deploy_Sistema.js

En este archivo básicamente, se está cogiendo el contrato para desplegarlo.

Luego, fuera de cualquier carpeta se encuentra el archivo "*truffle-config.js"*

```
// options below to some value.
//
development: {
 host: "127.0.0.1",     // Localhost (default: none)
 port: 7545,            // Standard Ethereum port (default: none)
 network_id: "*",       // Any network (default: none) Se puede dejar con *, pues esto indica que cualquier red puede ser conectada
},
//
// An additional network, but with some advanced options…
```

Figura 14. truffle-config.js

Este archivo, permite configurar el proyecto Truffle, en concreto, lo que importa para el proyecto es ver donde se conecta a la red Blockchain, de ahí la información no comentada, el host, el puerto o la id de red, la cual aparece como "*" debido a que con los otros parametros sirve para identificar la red, aunque en otros casos deberia ponerse el nombre.

6.4.2 Configuración del Servidor

Para este apartado, debería explicar dos cosas previamente:

- Bootstrap: es un framework que facilita el desarrollo de interfaces de usuario y el diseño web, proporcionando estilos CSS predefinidos, scripts de JavaScript…
- Lite-Server: es un servidor de desarrollo diseñado para facilitar la visualización de aplicaciones web estáticas durante el desarrollo de estas.

Ahora sí, yendo con el código sin entrar en ninguna carpeta, *"bs-config.json"*

Figura 15. bs-config.json

En este archivo se encuentra la configuración para el servidor local, donde se indica los directorios donde se buscarán los archivos, lo que se encuentra dentro de "baseDir". También se configuran las rutas adicionales, donde apuntar cuando se busca (en este caso) "/libs", y, finalmente el puerto donde se ejecuta el servidor.

También está el archivo *"package.json"*

Figura 16. package.json

En este archivo se encuentra la configuración usada por npm, para administrar el proyecto Node.js, donde se ven algunas propiedades importantes, como son las dependencias (bibliotecas externas necesarias para que el proyecto funcione), el nombre del proyecto, la versión utilizada, el archivo principal del proyecto (en este caso donde se encuentra la configuración de truffle), los scripts y algunos metadatos del proyecto.

6.4.3 Interacción con el contrato

En este apartado se mostrará como el frontend se relaciona con el contrato, a través de JavaScript, todo lo citado en este apartado se encuentra en la carpeta client.

El primer paso sería conectarse al contrato, eso se encuentra en el archivo **app.js**

```
loadContracts: async () => {
    const res = await fetch("SistemaElecciones.json")
    const voteContractJSON = await res.json()       //obtengo el JSON

    App.contracts.voteContract = TruffleContract(voteContractJSON)  //Convierto el JSON

    App.contracts.voteContract.setProvider(App.web3Provider)        //Conectar a metamask el contracto

    App.voteContract = await App.contracts.voteContract.deployed() //Intentar utilizar el contrato
},
```

Figura 17. Funcion loadContracts

En esta funcion se intenta conectar al contrato a través del archivo json del contrato, que se genera en compilamiento/despliegue del contrato, se hace siguiendo los pasos que aparecen en la figura.

Por otra parte, también hay un control de errores, que se encuentra en *"app.js"*.

```
const extraerMensajeDeError = (e) => {
    let errorMessage = 'Ha ocurrido un error';

    if (typeof e.reason !== 'undefined') {
        errorMessage = e.reason;
    } else if (typeof e.message !== 'undefined') {
        const rawError = e.message;
        const rawErrorMatches = rawError.match(/"reason"\s*:\s*"([^"]*)"/);

        if (typeof rawErrorMatches[1] !== 'undefined') {
            errorMessage = rawErrorMatches[1];
        }
    }

    return errorMessage;
};
```

Figura 18. control de errores

Lo que ocurre aquí es que, al haber un error, se crea un mensaje predefinido, luego de crearse va verificando propiedades dentro del error, en caso de que exista una propiedad modifica el mensaje predefinido, mostrando el mensaje que quede al final.

Otra interacción muy importante, seria para obtener un bloque y así poder explicarlo a posteriori, esto se explica en el apartado 7.

```
getBlockDetails: async (blockNumber) => {
    try {
      const block = await App.web3.eth.getBlock(blockNumber); // Utilizar la instancia de web3 para obtener detalles del bloque
      console.log('Block Details:', block);
      // Aquí puedes realizar cualquier acción adicional con los detalles del bloque
    } catch (error) {
      console.error('Error al recuperar detalles del bloque:', error);
    }
  },
```

Figura 19. getBlockDetails

Esta función es simple, a través de web3 intenta obtener el bloque especificado, si lo encuentra lo muestra por consola, en caso contrario muestra por consola un error.

A partir de este momento, las interacciones son de dos tipos, si es formulario (entonces tiene botón) o si solo muestra (que no lo tiene). Solo se va a exponer uno de cada, entiéndanse los otros por igual, pero con pequeñas modificaciones.

Si tiene botón, primero pasa por *"ui.js"*, pero, la interacción con este archivo se mostrará luego.

```
voto: async (nombreVoto) => {
    try {
        const resultado = await App.voteContract.voto(nombreVoto, {
            from: App.account
        })
        document.getElementById("#mensajeError").innerText = "Voto realizado correctamente"
        console.log('Voto realizado correctamente')
    } catch (e) {
        const errorMessage = extraerMensajeDeError(e);
        document.getElementById('mensajeError').innerText = errorMessage;
        document.getElementById('mensajeError').style.display = 'block';
    }
},
```

Figura 20. voto en app.js

Con el dato obtenido de ui.js, se intenta llamar al contrato para realizar la acción pertinente en la Blockchain, en caso contrario se llama al error explicado anteriormente.

Por otra parte, si no es un formulario es diferente, dentro del archivo *"app.js"*

```
renderGanador: async () => {
    const ganador = await App.voteContract.winner()
    console.log(ganador)

    let html = ''

    let elementoGanador = `
    <div class="card card-body bg-dark mb-2">
            <div className="card-header">
                <h5>${ganador[0]}</h5>
                <h5>${ganador[1]}</h5>
            </div>
    </div>
    `
    html = elementoGanador;
    document.querySelector('#verGanador').innerHTML = html;
},
```

Figura 21. renderGanador

En este caso, se llama al contrato para realizar la acción, obteniendo el dato. Pero como lo deseado es mostrarlo por pantalla se crea una variable donde se guarda la información de la forma que se desea visualizar, posteriormente se le asigna esta variable a una etiqueta puesta anteriormente en el HTML con lo que se acaba mostrando el mensaje de la forma deseada.

6.4.4 Interacción con la Blockchain

Primero se busca conectar con una cuenta Ethereum, wallet, para eso, dentro del archivo *"app.js"*, se encuentra:

```
loadEthereum: async () => {
    if (window.ethereum) {
      App.web3Provider = window.ethereum;
      console.log('ethereum existe');
      try {
        await window.ethereum.request({ method: 'eth_requestAccounts' }); // enlazar cuenta ethereum
        App.web3 = new Web3(App.web3Provider); // Obtener la instancia de web3 utilizando el proveedor de MetaMask
      } catch (error) {
        console.error('Error al solicitar cuenta de MetaMask:', error);
      }
    } else if (window.web3) {
      App.web3Provider = window.web3.currentProvider;
      App.web3 = new Web3(App.web3Provider);
      web3.eth.handleRevert = true;
    } else {
      console.log('No hay instalado ningún navegador ethereum. Intenta usar MetaMask.');
    }
  },
```

Figura 22. loadEthereum

En ella, primero intenta ver si existe el objeto global window.ethereum inyectado por MetaMask, que proporciona una interfaz para interactuar con la red Ethereum desde una aplicación web. Luego intenta obtener la cuenta del usuario.

Si no es el caso se prueba con el window.web3, que es otra interfaz para interactuar con la red Ethereum. Y se vuelve a intentar obtener la cuenta.

Si este no funciona muestra que prueba a descargar una wallet, como por ejemplo MetaMask.

Después de conectarse con la wallet, se intenta obtener la cuenta del usuario a través de:

```
loadAccount: async () => {

    const account = await window.ethereum.request({ method: 'eth_requestAccounts' })
    App.account = account[0]

},
```

Figura 23. loadAccount

6.4.5 Interacción con la interfaz de usuario

En este apartado es donde entra en juego el html, en este caso *"index.html"*, pero también otro archivo, el ui.js.

Este último, solo aparece cuando hay botones, y su interacción es la siguiente.

Previamente se ha seleccionado el elemento que tiene una id en html con la línea:

const voteForm = document.querySelector("#voteForm")

```
voteForm.addEventListener("submit", e => {
    e.preventDefault(); //Cancelar el autorefresh

    const nombre = voteForm["nombreVoto"].value;

    App.voto(nombre);
```

Figura 24. voteForm en ui.js

Una vez obtenido, se obtiene el texto escrito dentro, y luego se llama a la función de app.js añadiendo como información el nombre.

Por otra parte, el HTML es más extenso, primero en la cabecera se indica la configuración básica de la web:

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="/libs/bootstrap/dist/css/bootstrap.min.css">
    <link rel="stylesheet" href="./main.css">
    <title>Sistema de Votación</title>
</head>
```

Figura 25. Cabecera HTML

En esta se indica, que los caracteres van a ser los de UTF-8, el título de la página web, así como carga las hojas de estilo requeridas para la correcta visualización del sitio.

```css
body {
    background: #0A192F;
    color: #fff;
}
```

Figura 26. main.css

En este caso el main.css solo indica que el fondo va a ser oscuro y la letra blanca.

Luego aparece el cuerpo del HTML, que indica las diferentes apariciones, con las distintas id's y los textos que aparecen, por lo general son bastante semejantes entre ellos, con la variedad de que algunos son formularios y otros muestran por pantalla solamente.

Finalmente, aparecen los scripts que enlazan el HTML a diferentes url's explicadas a lo largo del punto, excepto por el segundo, que lo enlaza a la biblioteca Web3.js.

```
<script src="./libs/@truffle/contract/browser-dist/truffle-contract.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/web3@1.5.2/dist/web3.min.js"></script>
<script src="./app.js"></script>
<script src="./ui.js"></script>
```

Figura 27. Scripts index.html

Parte 7. Despliegue

Primero hay que entender que el hecho de vincular el contrato Solidity, con los archivos JavaScript y HTML indica la construcción de la dApp, o aplicación descentralizada.

7.1 Entorno de desarrollo

En el proyecto se ha utilizado un entorno de desarrollo de prueba, pero basado en la red de Ethereum, para ello se ha utilizado herramientas como son truffle, ganache o node.js. Para la realización del despliegue ha sido necesario ganache, el cual ha creado la red de prueba a través del puerto 7545 y el host "127.0.0.1". Una vez enlazado el contrato con esa red, se ha compilado y desplegado el contrato con la ayuda de truffle.

Para ello se tienen que realizar los siguientes pasos:

1. Compilar código: se realiza con el comando *truffle compile*, o con *truffle compile --all* en el caso que se quiera realizar como si lo anteriormente compilado no existiera.
2. Desplegar el contrato: se realiza con el comando *truffle deploy*, o con *trufle deploy –all*, por el mismo motivo que en el apartado anterior.
3. Posteriormente, se inicia el servidor: se realiza con el comando *npm run dev*, proporcionado por de node.js.

Después de esto, se encontrará la dApp iniciada, con lo cual se podrá observar el correcto funcionamiento de ella. Para ayuda de esto, he realizado un vídeo que se encuentra en Youtube donde se muestra lo descrito en este apartado: "https://youtu.be/D_XVNQfoeI4"

En el caso que se quisiera trasladar fuera del servidor de pruebas, se podría implementar la posible solución que se expondrá en el punto siguiente.

Finalizando con esta parte, se puede comprobar la correcta implementación de la red Blockchain en el proyecto analizando uno de sus bloques, gracias a la función añadida en app.js llamada *getBlockDetails*. El bloque obtenido se obtiene por consola y, se observa en la siguiente imagen:

```
Block Details:                                                                                                   app.js:172
  {hash: '0x35f81d33d0fa6064adbc536cb225bfd5f6332f3be9cbfe4e236909b7ca010518', parentHash: '0x8abb77507a0ce51af7a5ebca46e2c300ba9dc76c24b14c
▼ 529264dd7c57fdddab', sha3Uncles: '0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347', miner: '0x0000000000000000000000000
  000000000000000', stateRoot: '0xeca5e0aeee6e3b87bb324cfd8f0544d405a4a16309d4e971976441486b286216', …}
    baseFeePerGas: "0x342770c0"
    difficulty: "0"
    extraData: "0x"
    gasLimit: 6721975
    gasUsed: 22200
    hash: "0x35f81d33d0fa6064adbc536cb225bfd5f6332f3be9cbfe4e236909b7ca010518"
    logsBloom: "0x0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
    miner: "0x0000000000000000000000000000000000000000"
    mixHash: "0x66a4b019649333e82ed625d30cccfd437e478f1aff64d1dfe2673d46c60602eb"
    nonce: "0x0000000000000000"
    number: 1
    parentHash: "0x8abb77507a0ce51af7a5ebca46e2c300ba9dc76c24b14c529264dd7c57fdddab"
    receiptsRoot: "0x67b40c98d1eb5d5f50381e0d55b533d1753980584fe366ef3e799adb98cd4a18"
    sha3Uncles: "0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347"
    size: 859
    stateRoot: "0xeca5e0aeee6e3b87bb324cfd8f0544d405a4a16309d4e971976441486b286216"
    timestamp: 1686756087
    totalDifficulty: "0"
  ▼ transactions: Array(1)
      0: "0x685d6939b8b300a2b42eda016e8bddace44b44ba1ca64acb4b12b54aac2ca4b2"
      length: 1
    ▶ [[Prototype]]: Array(0)
    transactionsRoot: "0x3a36fff097fbcf71553de83b2a5a49a9b68bb7e2d318bb106e21494fb40ac200"
  ▼ uncles: Array(0)
      length: 0
```


Figura 28: Bloque perteneciente a la red Blockchain

En este caso se trata del primer bloque se aparece en la cadena, tras la primera transacción.

7.2 Redes de Ethereum de Pago

La posible solución que permitiría introducirla en un entorno que no sea de pruebas, seria a través de la propia red de Ethereum. La principal diferencia por la que se ha optado por la red de pruebas es, por la obligación del pago de un gas o tarifa, debido a que el despliegue en la red de Ethereum tiene un coste (económico).

Para la realización de esto, hay que entender que el compilado y despliegue es el mismo, con la ayuda de truffle. Pero el despliegue se realiza en la red de Ethereum y no en la de pruebas.

Una vez se ha desplegado el contrato en la red de Ethereum, es necesario que la dApp sea publicada en algún lugar accesible en la web, para ello sería conveniente el uso de un servicio de alojamiento como IPFS, Amazon Web Services… Realizando todos los procesos que eso conlleva, como es el traslado de los archivos a los lugares pertinentes o la configuración de su dominio.

El gran problema de esta solución es el hecho de que deja de ser gratuito su uso, tanto para el que despliega el contrato como para los usuarios.

Parte 8. Conclusiones

Antes de exponer las conclusiones me gustaría decir que es la primera vez he interactuado en el entorno de Ethereum y todo lo que depende de él, como es Solidity, por lo que se ha empezado desde la base total.

Primeramente, debería analizar los objetivos que se expusieron al inicio del proyecto:

- Comprensión del funcionamiento de la tecnología Blockchain.
  A lo largo de los primeros puntos se ha desarrollado una gran variedad de información relacionada con el tema, pues se ha dicho que esta tecnología se encuentra formada por una serie de bloques entrelazados, de ahí su nombre, dentro de los cuales se encuentra la información de todas las transacciones realizadas en la cadena. Además, al encontrarse descentralizada, pues todos los nodos tienen la cadena, es inmutable, finalizando con el proceso de minería, siendo el que forma los nuevos bloques.
- Conocimientos de la implementación de Smart Contracts en la actualidad.
  En el proyecto también se ha expuesto la idea de Smart Contract, pues esta se creó para tratar de renovar los contratos tradicionales, dando más veracidad a los contratos, siendo estos regidos a unas condiciones. En la actualidad se aplican en las Blockchains para la realización de infinidad de cosas.
- Conocimientos sobre las plataformas descentralizadas, especialmente sobre Ethereum.
  También ha sido desarrollado el tema de las plataformas descentralizadas, centrándose en la red de Ethereum, donde se han conocido sus principales elementos, como son: la EVM, el Ether, el gas. Finalmente, conociendo la aplicación de los Smart Contracts en este.
- Desarrollo de la aplicación Smart Contract que permita participar en un sistema de votaciones.
  Finalmente se ha desarrollado el dApp, o aplicación descentralizada, basada en un Smart Contract que permite la creación de un sistema de votaciones. Habiéndose introducido este en una red Blockchain de pruebas basada en la red Ethereum.
  A pesar de encontrarse en una red de pruebas, permite observar el funcionamiento completo de una red de Blockchain, más en concreto la de Ethereum. Pues se realizan todas las transacciones que se demandan.

Por lo que se podría concluir que el proyecto ha sido de gran utilidad, porque me ha permitido comprender el gran mundo que se encuentra detrás de las Blockchains, pues mucha gente las asocia únicamente a las Criptomonedas. Además, me ha ayudado al a consolidar conocimientos de programación, pues tenía conocimientos básicos de HTML y JavaScript, ampliando estos con Solidity.

Dejando aparte los conocimientos técnicos, este proyecto me ha ayudado a la hora de comprender el cumplimiento de objetivos con plazos temporales. Cosa que me refuerza de cara al mundo laboral en un futuro.

Parte 9. Bibliografía

[1] Binance Academy. (s.f.). Historia de blockchain. Recuperado de https://academy.binance.com/es/articles/history-of-blockchain

[2] Smith, J., & Johnson, A. (2018). The Importance of Merkel Trees in Blockchain Technology. Journal of Cryptocurrency Studies,

[3] Bashir, I. (2018). Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Decentralized Applications. Wiley.

[4]Wikipedia. (s.f.). Stuart Haber. En Wikipedia. Recuperado de https://en.wikipedia.org/wiki/Stuart_Haber

[5] Bit2Me Academy. (s.f.). ¿Quién es W. Scott Stornetta? Recuperado de https://academy.bit2me.com/quien-es-w-scott-stornetta/

[6] Centro de Investigación y Seguridad Nacional (CISEN). (s.f.). Blockchain. Recuperado de https://www.ciset.es/glosario/709-blockchain

[7] CriptoNoticias. (s.f.). ¿Quién es Satoshi Nakamoto, el creador de Bitcoin? Recuperado de https://www.criptonoticias.com/criptopedia/quien-satoshi-nakamoto-creador-bitcoin/

[8] CryptoNews. (s.f.). Blockchain. Recuperado de https://cryptonews.net/es/news/blockchain/17290459/

[9] Panda Security. (s.f.). WP PCIP qué es P2P. Recuperado de http://resources.pandasecurity.com/enterprise/solutions/8.%20WP%20PCIP%20que%20es%20p2p.pdf

[10] Cointelegraph. (s.f.). Who is Vitalik Buterin? Recuperado de https://es.cointelegraph.com/learn/who-is-vitalik-buterin

[11] Hyperledger. (s.f.). About. Recuperado de https://www.hyperledger.org/about

[12] Corda (s.f.). Recuperado de https://corda.net/

[13] Armadillo Amarillo. (s.f.). Aplicaciones de blockchain: 11 usos de blockchain que no conocías. Recuperado de https://www.armadilloamarillo.com/blog/aplicaciones-de-blockchain-11-usos-de-blockchain-que-no-conocias/

[14] Kriptomat. (s.f.). Los mejores casos de uso de blockchain. Recuperado de https://kriptomat.io/es/blockchain/los-mejores-casos-de-uso-de-blockchain/

[15] SG Comunicación. (s.f.). El blockchain y sus aplicaciones. Recuperado de https://sg.com.mx/revista/47/el-blockchain-y-sus-aplicaciones

[16] Nick Szabo: Rankia. (s.f.). Nick Szabo. Recuperado de https://www.rankia.com/diccionario/biografias/nick-szabo

[17] Ethereum. (s.f.). Smart Contracts. Recuperado de https://ethereum.org/en/developers/docs/smart-contracts/#:~:text=A%20%22smart%20contract%22%20is%20simply,be%20the%20target%20of%20transactions

[18] Iberdrola. (s.f.). Smart Contracts. Recuperado de https://www.iberdrola.com/innovacion/smart-contracts

[19] IEBSchool. (s.f.). Smart Contract y blockchain. Recuperado de http://www.iebschool.com/blog/smart-contract-blockchain-tecnologia/

[20] IEEE Xplore. (2000). [Abstract]. Recuperado de http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=720574

[21] Diedrich, H. (2018). Ethereum: Blockchains, Digital Assets, Smart Contracts, Decentralized Autonomous Organizations. Apress.

[22] Ethereum. (s.f.). Ethereum Virtual Machine (EVM). Recuperado de https://ethereum.org/es/developers/docs/evm/

[23] Modi, R. (2017). Solidity Programming Essentials: Develop and Design. Packt Publishing.

[24] Bit2Me Academy. (s.f.). ¿Qué es Ethereum Virtual Machine (EVM)? Recuperado de https://academy.bit2me.com/que-es-ethereum-virtual-machine-evm/

[25] ITDO. (s.f.). ¿Qué es una Dapp y en qué se diferencia de una aplicación normal? Recuperado de https://www.itdo.com/blog/que-es-una-dapp-y-en-que-se-diferencia-de-una-aplicacion-normal/#:~:text=Una%20Dapp%20(aplicaci%C3%B3n%20descentralizada)%20es,descentralizada%20peer%2Dto%2Dpeer.

[26] Ethereum. (s.f.). Gas. Recuperado de https://ethereum.org/es/developers/docs/gas/

[27] proof-of-work: Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Journal of Cryptocurrency,

[28] EthereumJS. (s.f.). Ethereum Units. Recuperado de https://github.com/ethereumjs/ethereumjs-units/blob/master/units.json

[29] Infante, R. (2019). Building Ethereum DApps: Decentralized Applications on the Ethereum Blockchain. Apress.

[30] P.V., K. K. (2018). Hands-On Blockchain with Ethereum: Build decentralized applications with Solidity, Ethereum, and JavaScript. Packt Publishing.

[31] Li, W., & Zhang, J. (2020). An Efficient Bytecode Optimization Technique for Embedded Systems. Journal of Systems Architecture

[32] Ethereum. (s.f.). ERC-20: Título de la página. Recuperado de https://ethereum.org/es/developers/docs/standards/tokens/erc-20/

[33] ESIC Business & Marketing School. (s.f.). La moneda Ethereum: El Ether, ¿para qué se diseñó y cómo se gestiona? Recuperado de https://www.esic.edu/rethink/tecnologia/la-moneda-ethereum-el-ether-para-que-se-diseno-y-como-se-gestiona

[34] Nielsen, J. (1993). Usability Engineering. Morgan Kaufmann.

[35] Nielsen, J. (2000). Designing Web Usability: The Practice of Simplicity. New Riders.

[36] Nielsen, J., & Budiu, R. (2013). Mobile Usability. New Riders.

[37] Johnson, B. (2020). Visual Studio Code: End-to-End Editing and Debugging Tools for Web Developers. Apress.

[38] Zhu, R. (2019). Mastering Visual Studio Code: Develop and Deploy Next-Generation Web Applications. Packt Publishing.

[39] Microsoft. (s.f.). Visual Studio Code documentation. Recuperado de la documentación oficial proporcionada por Microsoft para Visual Studio Code.

[40] Antonopoulos, A. M., & Wood, G. (2018). Mastering Ethereum: Building Smart Contracts and DApps. O'Reilly Media.

[41] Remix. (s.f.). Remix Documentation. Recuperado de https://remix.ethereum.org/docs/

[42] Chittoda, J. (2020). Mastering Blockchain Programming with Solidity: Write Production-Ready Smart Contracts for Ethereum and Hyperledger Fabric. Packt Publishing.

[43] Truffle Suite. (s.f.). Ganache. Recuperado de https://www.trufflesuite.com/docs/ganache

[44] Infante, R. (2019). Building Ethereum DApps: Decentralized Applications on the Ethereum Blockchain. Apress.

[45] StarUML. (s.f.). Working with UML Diagrams - Information Flow Diagram. Recuperado de https://docs.staruml.io/working-with-uml-diagrams/information-flow-diagram

[46] Balsamiq. (s.f.). Intro - Balsamiq Wireframes Desktop. Recuperado de https://balsamiq.com/wireframes/desktop/docs/intro/#:~:text=Balsamiq%20Wireframes%20is%20a%20user,before%20any%20code%20is%20written.