UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una app web para la gestión de historiales médicos mediante blockchain

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Lanza Nieto, Javier

Tutor/a: Molina Marco, Antonio

CURSO ACADÉMICO: 2023/2024

Resumen

En este trabajo se profundizará en el estudio de la tecnología blockchain y su aplicación en el desarrollo de una aplicación web destinada a la gestión de historiales médicos. Para ello, se utilizará Ethereum Blockchain, que permitirá administrar el acceso a los historiales, así como a la propia aplicación mediante su lenguaje destinado a la definición de Smart Contracts, denominado Solidity. Esta tecnología interactuará con una interfaz *frontend* desarrollada en React.js destinada al usuario, que podrá ser tanto un paciente como personal sanitario. El objetivo de la app es otorgar el control sobre quién tiene acceso a su historial, pudiendo concederlo de una manera sencilla y rápida en cualquier momento. El uso de blockchain posibilitará obtener transparencia y trazabilidad sobre todos los cambios realizados en los historiales médicos, generando además un sistema completamente descentralizado. La implantación de una solución como la propuesta proporcionaría un mejora en la interoperabilidad del sistema sanitario actual.

Palabras clave: Blockchain, Solidity, React, DApp, Health, Web app

Abstract

This work will go deep into the study of blockchain technology and its application in the development of a web application for handling medical records. Ethereum Blockchain is utilized for this purpose, allowing the management of access to the medical records as well as the application itself through its Smart Contracts definition language, Solidity. This technology will interact with a frontend interface developed in React.js for the users, who can be either patients or healthcare personnel. The main goal is to provide control over who has access to their medical history, allowing users to grant access easily and quickly at any time.The use of blockchain enables transparency and traceability of all changes made to medical records, creating a fully decentralized system. The implementation of a solution like this would provide an improvement in the interoperability of the current healthcare system.

Key words: Blockchain, Solidity, React, DApp, Health, Web app

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación

Tradicionalmente, el control de la información médica del paciente siempre ha estado en manos de la administración, lo cual ha dificultado que los pacientes puedan acceder fácilmente a su propia información. Esto a menudo genera largos tiempos de espera y trámites burocráticos, además de limitar el control del paciente sobre quién puede ver o modificar sus datos. Este proyecto surge para solucionar precisamente este tipo de situaciones, sirviendo como apoyo a los métodos actuales y proporcionando un mayor control al usuario.

La constante evolución de nuestra sociedad nos lleva a explorar más allá de las posibilidades que hasta ahora han ido surgiendo, con la finalidad de encontrar aquello que mejor se adapte a lo que necesitamos. Esto es más aplicable aún si cabe a nuestro sector, el tecnológico, dónde cada día emergen nuevas tecnologías que pueden aplicarse de una manera innovadora en diversos proyectos. En estos, nuestro rol es evaluar las necesidades de cada uno y buscar las alternativas que mejor se ajusten a ellos.

Todo esto, unido a la sensibilidad que conllevan los datos a tratar, nos impulsa a usar una base segura, confiable y eficiente. Aquí es donde el papel más relevante es asumido por Blockchain, una innovadora tecnología capaz de gestionar este tipo de datos de una manera descentralizada.

Esta tecnología se posiciona para desbancar a las alternativas actuales para el desarrollo de software, a pesar de haber surgido como un apoyo a las criptomonedas. Se trata de una manera totalmente diferente de manejar los datos conforme a lo que conocemos hoy en día en el sector de la programación.

Personalmente, explorar esta tecnología y sus capacidades me parece algo sumamente atractivo, más todavía para emplearla en un sector de tal delicadeza y sensibilidad como el de la salud.

1.2 Objetivos

El objetivo principal de este proyecto es explorar y demostrar las capacidades de esta tecnología para el desarrollo de software y, más concretamente, en ámbitos más sensibles como es el de la sanidad. También buscamos realizar un profundo estudio y una comparativa respecto a las alternativas actuales y metodologías utilizadas tradicionalmente, para analizar sus ventajas y desventajas, tratando además de vislumbrar su futuro en el campo del desarrollo de aplicaciones.

El foco del proyecto consiste en el desarrollo de una DApp (aplicación descentralizada) basada en la tecnología Blockchain, cuya funcionalidad sea la gestión de historiales médicos. Concretamente, el proyecto está pensado para ser utilizado por pacientes y médicos, con el fin de poder gestionar la información médica de los pacientes de una manera cómoda y confortable para el usuario.

1.3 Estructura de la memoria

Durante los próximos capítulos, trataremos de indagar en todo aquello que resulte interesante para el desarrollo de nuestro proyecto, trantando de comprender las necesidades del mismo y aplicar la mejor solución psoible.

Comenzaremos con un análisis del estado del arte para conocer un poco más a fondo la tecnología blockchain y el recorrido de los histpriales clínicos en nuestro país. Posteriormente, detallaremos conceptos y aspectos de una de las plataformas más populares de blockchain, sobre la que construiremos nuestro proyecto.

Durante el capítulo 4, estableceremos el marco de trabajo para la implementación de nuestra DApp seleccionando las tecnologías necesarias y el modelo de desarrollo a seguir. En el quinto capítulo, hablaremos de todas las fases por las que el proyecto pasará para ser implementado tratando de justificar las decisiones tomadas.

Finalizaremos con las conclusiones donde se realizará una valoración del cumplimiento de los objetivos establecidos y se establecerán las líneas de trabajo futuro en caso de continuar con el desarrollo de la aplicación.

CAPÍTULO 2
Estado del arte

2.1 Blockchain: Historia

Para comprender la Blockchain en profundidad, debemos comenzar por adentrarnos en su pilar fundamental, la criptografía. Más concretamente hemos de conocer dos técnicas de cifrado desarrolladas a lo largo de la década de los 1960 y que intervienen de manera fundamental en ella. Una es el cifrado asimétrico o cifrado de clave pública, un método que consta de un par de claves, una pública mediante la que se cifran los datos y una privada con la que se descifran. Por otro lado, contamos con la técnica conocida como función hash, que consiste en la generación de un código de longitud fija a partir de una cadena de datos, garantizando la integridad de estos. Para esta última función actualmente contamos con diferentes maneras de generar dicha cadena, que veremos en profundidad más adelante.

2.1.1. El por qué de la blockchain

La aportación más importante para esta tecnología viene de Stuart Haber, un estudiante de matemáticas en la Universidad de Harvard que, posteriormente, obtuvo un doctorado en la Universidad de Columbia gracias a un trabajo sobre criptografía. Tras esto, en 1987, Haber empezó a trabajar como investigador en Bell Communications Research (Bellcore). En 1989, llegó a Bellcore el físico Wakefield Scott Stornetta, el cual se convertiría en su socio y colaborador. Como ya hemos comentado, para esta época ya se habían desarrollado diversas técnicas de cifrado y criptografía, lo que permitía asegurar la autenticidad, integridad y confidencialidad de un documento, pero seguía existiendo un problema.¿Cómo se podía garantizar la fecha en la que dicho documento se había generado?

Haber y Stornetta se dedicaron a buscar una forma de solventar esta situación, un sistema que permitiera certificar cuándo se había emitido o modificado un documento. Fue en 1991 cuando finalmente publicaron el resultado de su investigación: "Cómo poner un sello de tiempo a un documento digital" [4], lo que resultó ser una solución práctica para avalar que una vez firmado un documento digital en una fecha concreta no fuera posible manipularlo. Esta técnica utilizaba una cadena de bloques que almacenaba los documentos con su sello de fecha.

2.1.2. La tecnología blockchain y su moderado impacto

En el estudio realizado por ambos investigadores, se describe cómo mediante el uso de funciones *hash* y firmas digitales se conseguía incluir esta marca inmutable de tiempo, almacenando y encadenando los documentos uno detrás de otro. Esto es gracias a que la tecnología blockchain es una base de datos en la que estos se almacenan en grupos denominados bloques,

cada uno con una cierta capacidad de almacenamiento que al llenarse se cierra y se vincula al bloque anterior. Una vez cerrado el bloque su contenido no se puede modificar, permitiendo tan solo la lectura del mismo. Por tanto, se puede afirmar que esta tecnología consiste en una cadena de bloques interconectados y sellados, lo que da origen a su nombre, blockchain.

En 1992, Haber y Stornetta incorporaron a esta técnica los árboles *hash* de Merkle, unas estructuras de datos jerárquicas que otorgaron a esta tecnología la capacidad para ligar varios documentos a una única función hash. Esto produjo una mejora significativa de eficiencia para su solución. Dos años más tarde, ambos fundaron la empresa Surety Technologies para poner en marcha el primer servicio comercial de sellado de tiempo. Sin embargo el impacto de esta tecnología resultó ser limitado, lo que llevó a la expiración de su patente en 2004.

2.1.3. Las primeras propuestas de su uso

En 1997 se publicó por parte de la Agencia de Seguridad Nacional de Estados Unidos (NSA) un trabajo de investigación en el que se describe cómo crear una criptomoneda : "How to Make a Mint: the Cryptography of Anonymous Electronic Cash" [5]. Un año más tarde, el ingeniero informático Wei Dai publicó un trabajo sobre la arquitectura de B-money, una moneda virtual que nunca se llegó a hacer realidad, tratando la descentralización del sistema y la protección del anonimato. Ese mismo año, el informático Nick Szabo presentó un documento técnico[6] dónde se proponía el desarrollo de un sistema monetario descentralizado al que denominó Bit Gold.

A diferencia de la solución publicada por Haber y Stornetta, para crear una criptomoneda es necesario solventar el problema del doble gasto. Como su propio nombre indica, este se refiere a la posibilidad de que una persona use la misma unidad monetaria varias veces. Es un concepto que surge con los sistemas de moneda digital, pues con los modelos físicos las unidades monetarias (monedas, billetes, etc...) son entregadas, provocando que la persona que lo hace deje de ser poseedora de estas. Esta situación puede ser explicada con el conocido como "El problema de los generales bizantinos" que detallaremos más adelante.

Para impedir el doble gasto, Szabo propuso el uso del sistema de prueba de trabajo (Proof of work) usado en *hashcash*, una solución creada por el británico Adam Back en 1997 a modo de mecanismo de defensa ante el abuso sistemático de los recursos de internet como el spam o los ataques de denegación de servicio (DoS). Este sistema consiste en pedirle al emisor que realice un trabajo computacional diseñado para que a una computadora regular le tome algún tiempo procesarlo, provocando que un usuario común deba esperar unos segundos para poder realizar un envío (de un correo electrónico por ejemplo) a la vez que otro que quiera realizar miles al mismo tiempo le resulte más complicado.

Esto cuadraba a la perfección para Szabo debido a que el valor monetario de Bit Gold se basaba en el coste de cómputo de aquellos recursos informáticos utilizados. Sin embargo, no tuvo éxito a causa de que este sistema provocaba que unas mismas monedas tuvieran distinto valor dependiendo del coste de cómputo requerido.

2.1.4. Bitcoin como el primer gran uso de blockchain

Tras todos los intentos fallidos de las diferentes versiones de esta tecnología, no se haría hueco, entre los distintos sistemas desarrollados, hasta 2008. A lo largo de ese año una o varias personas, bajo el pseudónimo de Satoshi Nakamoto, publicaron un documento que describe un sistema electrónico de pago descentralizado entre pares, bautizado como Bitcoin [7]: "Bitcoin: A Peer-to-Peer Electronic Cash System" [8]. En la práctica, Bitcoin resultó ser una moneda

virtual que escapaba al control de cualquier gobierno, banco o entidad financiera, es decir una moneda fuera del control centralizado.

El sistema de Bitcoin consiste en la emisión de un número determinado de monedas virtuales identificables, cuya propiedad puede ser traspasada, quedando un sello de esta operación en un registro público basado en blockchain consultable abiertamente. Evitando que estas bases de datos estén en manos de alguna empresa o institución, múltiples copias de estas se distribuyen en diferentes ordenadores que voluntariamente desean participar en el sistema. Estos computadores conocidos como mineros, son los encargados de validar y registrar cada transacción recibiendo una pequeña compensación a cambio de resolver un algoritmo matemático complejo (prueba de trabajo). Todo esto sirve para resolver el problema del doble gasto entre otros, generando un mecanismo de consenso basado en un modelo mejorado de la solución *hashcash* mencionada anteriormente.

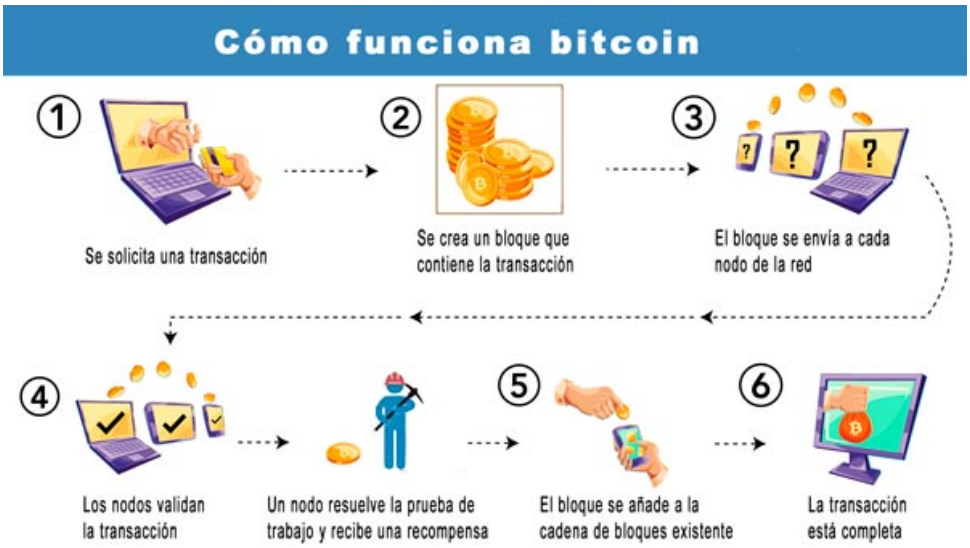

Figura 2.1: Imagen explicativa de Bitcoin extraída de [3]

2.1.5. Un paso adelante para Blockchain: Contratos Inteligentes

La aparición de Bitcoin provocó el surgimiento de muchos sistemas similares en el mercado como pueden ser Cardano, Tether, Polkadot, etcétera. No obstante, la aplicación de blockchain como una tecnología independiente que podía ser usada para otros cometidos, comenzó a ser planteada por diferentes personas. Una de ellas fué Vitalik Buterin, uno de los primeros creadores del código de Bitcoin, quién en 2013 comenzó a desarrollar una nueva plataforma que pudiera albergar contratos inteligentes (Smart contracts) utilizando blockchain, pero no fué hasta 2015 cuándo lanzó oficialmente su plataforma llamada Ethereum [9]. Para esta, un contrato inteligente es cualquier acción que pueda ser programada en su plataforma y esté basada en enviar y recibir transacciones, así como guardar los balances de las mismas.

Los denominados contratos inteligentes no son más que programas que se implementan y se ejecutan en esta cadena de bloques, existiendo para ello un lenguaje de programación específico que puede ser utilizado por cualquiera. Existen gran cantidad de programas ejecutándose en Ethereum y son conocidos como DApps. Bajo este modelo de desarrollo se han creado infinidad de juegos y sistemas de intercambio financiero, llegando incluso a construir su propia criptomoneda llamada Ether (ETH), convirtiéndose así en una de las aplicaciones

más grandes de la blockchain. El funcionamiento del contrato inteligente entre dos partes se ilustra en la Figura 2.2.

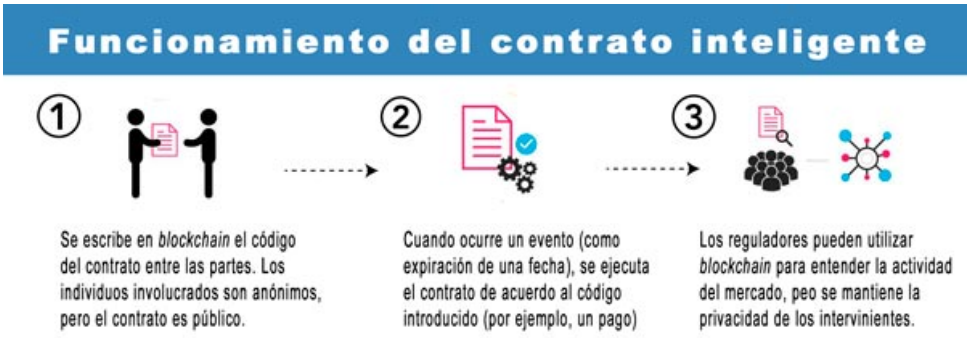

Figura 2.2: Imagen explicativa de un contrato inteligente extraída de [3]

A pesar de su similitud con Bitcoin, Ethereum resuelve el sistema de consenso de una manera diferente. Mientras que bitcoin emplea la ya mencionada prueba de trabajo (PoW), esta plataforma emplea una prueba de participación (PoS) en la que el validador de cada bloque es escogido al azar bajo unas determinadas condiciones.

2.1.6. El despegue de Blockchain

En la actualidad, la disposición de un registro inmutable y descentralizado de datos ha sido capaz de despertar la imaginación por parte de desarrolladores y empresas para adaptar esta tecnología a las necesidades particulares y emplearla en una gran diversidad de aplicaciones. A fin de comprender la evolución de blockchain, podemos observar algunas aplicaciones de la misma que han tenido lugar hasta ahora, como en el voto electrónico, los NFT (Non -Fungible Token) o la trazabilidad de los alimentos.

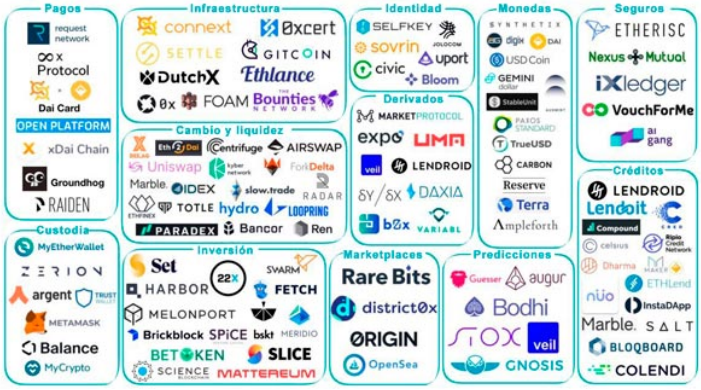

Figura 2.3: Ejemplos de aplicaciones construidas sobre blockchain extraída de [3]

2.2 Funcionamiento de Blockchain al detalle

Ahora que somos conocedores de la cronología de Blockchain, es un buen momento para indagar en las profundidades de su funcionamiento y comentar algunos aspectos relevantes de esta tecnología.

2.2.1. ¿Qué es exactamente Blockchain?

Para resolver esta pregunta podemos ver blockchain como un libro de contabilidad inmodificable y compartido que nos facilita el registro de transacciones y registro de activos en una red. Por activos nos podemos referir a prácticamente cualquier cosa que sea considerada del valor necesario para ser rastreada, pudiendo ser algo tangible (una casa, dinero en efectivo, tierra) o no (patentes, marca, derechos de autor).

La importancia de esta tecnología recae en su capacidad para proporcionar información inmediata, transparente y compartida permitiendo la visión de todos los detalles de cualquier transacción de extremo a extremo, aportando mayor confianza al usuario. Esto además se complementa con el uso de la criptografía, sobre todo con el uso de las funciones hash.

¿Cómo funciona?

Al producirse una transacción, esta se registra como un bloque de datos almacenando la información que se desee (como quién, qué, cuándo, dónde, cuánto,...) incluido el estado de esta. Cuando el bloque se cierra este se conecta al anterior y así sucesivamente, formando una cadena de datos en la que se registra cualquier tipo de movimiento o cambio de propiedad. Estos bloques confirman el tiempo exacto y la secuencia de transacciones entre ellos, evitando que se pueda modificar el contenido de alguno o insertar un bloque entre dos ya existentes. Cada bloque adicional refuerza la verificación del anterior y el resto de la cadena.

Problema de los generales bizantinos

Este problema es una analogía que explica la dificultad lógica que alberga un sistema distribuido para comunicar sus nodos y llegar a un consenso. En él, un ejército bizantino se divide en grupos dispersos con la finalidad de atacar un castillo y deben ponerse de acuerdo entre todos los sub ejércitos para hacerlo o retirarse al mismo tiempo. Si uno de los generales que comanda uno de estos sub ejércitos decide sabotear el plan y traicionar a sus aliados, el plan fracasaría.

La gran dificultad de este problema reside en la comunicación entre las partes y cómo ponerse de acuerdo para lograr conquistar el castillo. Para poder solucionarlo se deben establecer algoritmos con reglas que nos permitan evaluar sobre un estándar que información debe o no ser tratada como verdadera. Este problema se asemeja a cómo Blockchain debe tratar de que los nodos lleguen a un consenso y evitar que nodos malintencionados corrompan los datos de la red con transacciones fraudulentas. Para eso se instauraron en esta tecnología los conocidos como mecanismos de consenso.

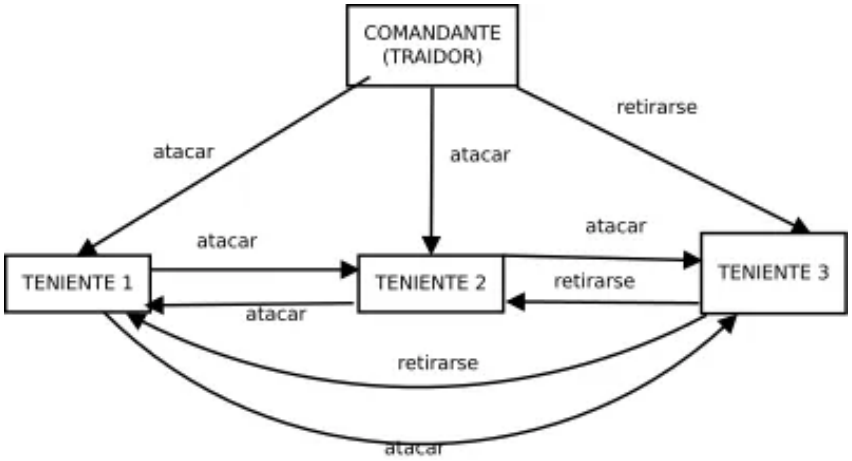

Figura 2.4: Problema solucionado [13]

En la figura 2.4, podemos observar como el problema se soluciona con el consenso general entre las partes implicadas, poniendo la información que tienen en común y llegando a un acuerdo. Para esto, es necesario esclarecer unas bases o condiciones sobre las que tomar la decisión. A esto lo llamamos mecanismo de consenso.

Mecanismos de consenso

Hablando de consenso en la tecnología blockchain hacemos referencia al proceso en el que los pares de una red llegan a un acuerdo sobre el estado verdadero de los datos en esta, para lograr tolerancia a fallos en el sistema. Los pares de esta red, conocidos como nodos, son los encargados de determinar qué transacciones deben ser válidas y cuáles no mediante un conjunto de normas llamados mecanismos de consenso que ayudan a proteger esta red de comportamientos malintencionados o maliciosos.

Para ello, existen muchos tipos de mecanismos que se utilizan en base a las necesidades de la red. A continuación describimos algunos de los más usados o considerados más importantes:

- Prueba de trabajo (PoW): como hemos comentado, este mecanismo es usado por Bitcoin y fue el primer mecanismo de consenso creado. Generalmente, es considerado el más fiable y seguro, aunque presenta algunos problemas de escalabilidad. En este los mineros compiten entre sí para resolver algoritmos computacionalmente complejos, siendo el primero en lograr obtener el *hash* quién se gana el derecho a formar el nuevo bloque y confirmar las transacciones, obteniendo por ello una cantidad predefinida de criptomonedas como recompensa. Debido a la cantidad elevada de recursos computacionales y de energía que requiere, es comúnmente criticado por su impacto medioambiental, llevando a otras plataformas a buscar protocolos más sostenibles y eficientes como el de prueba de participación (PoS).
- Prueba de participación (PoS): este popular método gira en torno a un proceso conocido como “apuesta”, debido a que en él los validadores se comprometen a apostar una moneda virtual a cambio de la oportunidad de ser elegidos al azar para validar un bloque, reportándoles una recompensa si lo consiguen. Al igual que en la lotería, a mayor cantidad apostada mayor probabilidad de ganar. Diferenciándose del mecanismo PoW, los que contribuyen al sistema obtienen una comisión por transacción. PoS es considerado una alternativa más sostenible que PoW, pero al favorecer a las entidades con mayor número de monedas suscita críticas por su potencial para conducir a la centralización.

- Prueba de participación delegada (DPoS): sigue siendo un mecanismo PoS pero modificado para estar basado en un sistema de votación que se fundamente en la reputación a la hora de lograr el consenso. En este los usuarios de la red "votan" añadiendo sus fichas a un fondo de apuestas para seleccionar los conocidos como testigos (productores de bloques) que aseguren la red en su nombre. Los más votados obtienen el derecho a validar las transacciones, recibiendo una recompensa por ellos normalmente repartida entre quienes les votaron. Esto ayuda a incentivar que los testigos sean honestos en todo momento debido a la posibilidad de ser expulsados y sustituidos por otros si incumplen sus responsabilidades o tratan de validar transacciones fraudulentas. Aunque es menos extendido que PoS, es considerado más eficiente y financieramente inclusivo que este.

Figura 2.5: Tipos de mecanismos de consenso existentes de [14]

2.2.2. Funciones Hash y su aportación

Como ya hemos comentado anteriormente, el nombre de *hash* se usa para identificar una función criptográfica muy importante en el mundo de la informática, cuyo objetivo primordial es codificar datos para formar una cadena de caracteres única. Para ello, carece de relevancia la cantidad de datos introducidos inicialmente de la función.

¿Cómo funcionan?

Llevar a cabo esta encriptación es posible gracias a una serie de complejos procesos matemáticos y lógicos trasladados a un ordenador con el fin de usarlos desde este mismo. Desde ahí, podemos introducir cualquier serie de datos que serán procesados para devolvernos una cadena de caracteres de longitud fija y única. Además, una gran ventaja de estas funciones es

que realizar el proceso inverso resulta prácticamente imposible. Por eso se considera como un proceso de un solo sentido, asegurando que una vez realizada la función *hash* sea casi inviable obtener los datos en su formato original.

Para poder entenderlo de una mejor manera, podemos compararlo al proceso de realizar un pastel. En este caso, cada uno de los ingredientes de este serían los datos introducidos y los pasos para su realización, el proceso de encriptación de los datos. Al final de este proceso, obtendríamos un pastel cuyos ingredientes resultan irrecuperables en su estado original.

Características de las funciones hash

Todo lo anterior, da lugar a realizar un pequeño análisis sobre las diferentes características que estas funciones albergan:

- Eficiencia: los algoritmos de *hash* son muy fáciles de calcular y no requieren de una gran potencia de cálculo.
- Compresibilidad: sin importar el tamaño de los datos, la salida siempre tendrá una cantidad de caracteres fija dependiendo de la función concreta. Por ejemplo, con SHA-256 la salida siempre consta de 64 caracteres.
- Funcionamiento de tipo avalancha: esto quiere decir que un mínimo cambio en la entrada de datos generará un *hash* totalmente distinto.
- Resistencia débil y fuerte a colisiones: la unicidad de los hashes está asegurada, provocando que sea imposible generar dos hashes exactamente iguales.
- Irreversibilidad: como ya hemos comentado, obtener un *hash* y tratar de obtener la cadena original resulta casi impracticable..

Nivel de seguridad

Las funciones actuales de *hash* constan con un muy alto nivel de seguridad, aunque no son totalmente infalibles. Como ejemplo de esto tenemos una de las funciones *hash* más utilizadas en internet hasta el año 1996, denominada MD5. En ese mismo año fue cuando se consiguió romper su seguridad, dejando a esta obsoleta y abandonada de su uso.

Por otro lado, funciones como RIPEMD-160 y SHA-256 albergan tal complejidad que su seguridad aún está garantizada, pues se calcula que para esta última harían falta miles de años usando supercomputadoras actuales para romperla.

A continuación mostraremos dos esquemas que representan el funcionamiento de dos de las funciones mencionadas, MD5 y SHA-256.

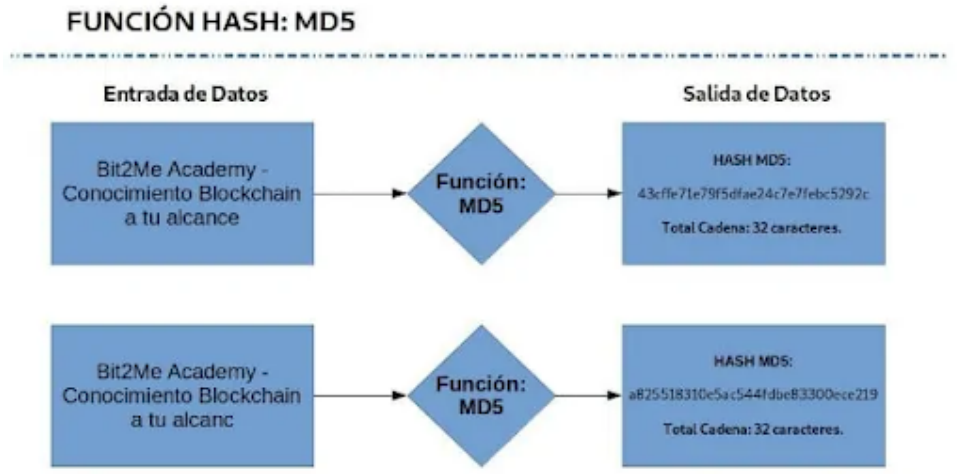

Figura 2.6: Ejemplo del funcionamiento de MD5 extraída de [15]

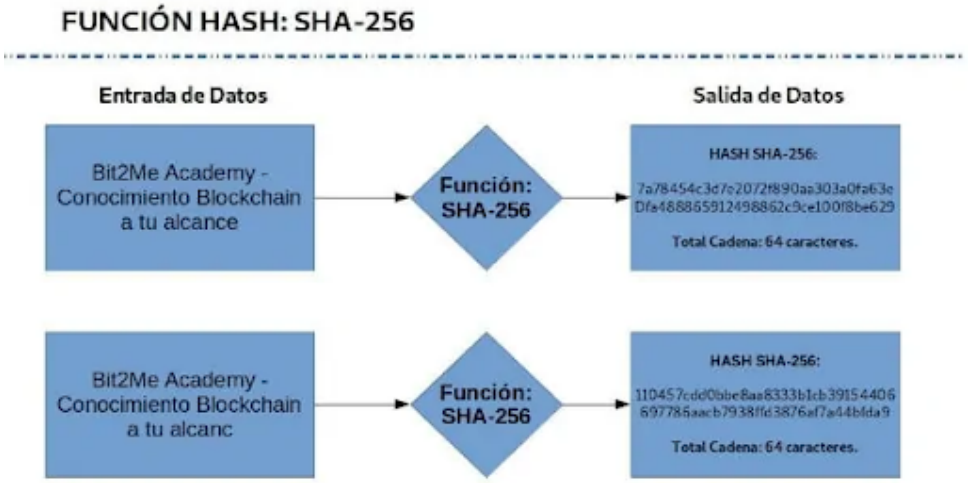

Figura 2.7: Ejemplo del funcionamiento de SHA-256 extraída de [15]

Su aportación a la Blockchain

Gracias a su velocidad, eficiencia, unicidad y coste, estas funciones son muy utilizadas en la tecnología blockchain que, a pesar de su gran evolución, siempre ha mantenido sus bases sobre una fuerte criptografía y las funciones de *hash*. Esto se vió aún más potenciado con la llegada de los contratos inteligentes a blockchain, ya que muchos de los datos manejados en estos son altamente sensibles e incluso en ocasiones resulta demasiada información para ser almacenada en una cadena de bloques.

La mejor manera de resolver estas situaciones es a través de las funciones *hash*, permitiendo que el contrato pueda ser público pero encriptando aquellos datos que quieran mantenerse privados, como nombres, direcciones o datos de terceros, conservando así la visibilidad de estos solo entre las partes interesadas.

Estos hashes también son utilizados para versionar los contratos de manera que un contrato público tiene un *hash* único derivado del contenido de este. Si el contrato es modificado, el anterior es terminado y se genera uno nuevo con otro *hash* diferente, determinando cual es el

contrato correcto a aplicar y facilitando su control. De esta manera, la realización de un contrato y su *hash* se pueden considerar testigos inalterables de lo que se realiza en él, resultando en un uso muy útil por ejemplo en sistemas de compraventa.

2.2.3. Tipos de redes Blockchain

Como en todo, en blockchain también contamos con diferentes tipos de redes conformadas según los intereses de quienes la implementan. Entre ellos distinguimos:

- Redes públicas: son aquellas a las que cualquiera puede unirse y participar. Su mayor desventaja es la gran potencia computacional que requieren, menor privacidad en sus transacciones y su seguridad reducida respecto a otras.
- Redes privadas: similares a las públicas excepto por el hecho de que existe una organización que gestiona la red y decide quién puede participar, ejecutar un protocolo de consenso y mantener el libro mayor compartido. Dependiendo de su uso, todo esto puede significar un aumento de confianza entre los participantes.
- Redes autorizadas: estas son redes públicas o privadas sobre las que se establecen ciertas restricciones sobre quién puede participar en la red y en qué transacciones. Normalmente requieren de un permiso o invitación para poder formar parte de ellas.
- Blockchains de consorcio: en ellas, las responsabilidades de mantener una blockchain pueden recaer sobre diferentes organizaciones que determinan quién puede enviar transacciones o acceder a los datos. Este tipo resulta ideal para empresas en las que todos los participantes necesitan permisos y tienen parte de la responsabilidad de la blockchain.

2.2.4. Desventajas y limitaciones de Blockchain

Hasta ahora hemos comentado los numerosos beneficios que esta innovadora tecnología nos ofrece. Sin embargo, debemos también tener en cuenta las desventajas que posee para ser capaces de obtener una visión objetiva de esta y analizar verdaderamente sus posibles aplicaciones según las restricciones que estas nos aportan.

- Dependencia de criptomonedas: esto es debido a que actualmente las redes de blockchain en su mayoría utilizan las criptomonedas para hacer posibles las transacciones y el funcionamiento de estas redes, llevando consigo el inconveniente de la fluctuación del precio real que suelen tener estas monedas y generando un problema de estimación y variación de costes.
- Coste: el uso de estas redes si bien puede disminuir los costes operacionales requeridos, también generan una necesidad de mayor inversión en su fase de implementación inicial debido al desconocimiento y falta de profesionales con conocimientos profundos sobre esta.
- Disrupción cultural: la adopción total de blockchain requiere un cambio cultural muy significativo dentro de aquellas organizaciones que deseen implementar esta tecnología suponiendo un desafío implementarla. Todo ligado a la reciente emergencia de esta, puede provocar que muchas empresas sean reacias a adoptarla por temor.
- Escalabilidad: respecto a las tecnologías actuales, un blockchain que no esté correctamente implantado y ajustado a las necesidades concretas puede volverse altamente problemático a medida que crece su infraestructura, afectando a la velocidad y eficiencia de sus transacciones.

- Información pública: este punto, a pesar de ser una ventaja, puede volverse problemático ya que si se hace público por error algún dato que no debería serlo no sería posible eliminarlo o modificarlo.

Sin embargo, la mayor parte de estas limitaciones puede ser corregida de antemano con un análisis profundo de las necesidades que tenemos en nuestra red y con la participación de profesionales bien preparados y formados para el uso de esta tecnología.

2.2.5. Impacto de Blockchain en el sector sanitario

Si bien es cierto que blockchain nos lleva a hablar comúnmente de conceptos como las criptomonedas, la capacidad de estas redes para realizar transacciones seguras les lleva a tener grandes aplicaciones en otros campos como lo es el sector sanitario. Esta red nos permite enfrentarnos a los problemas actuales en este sector como son la falsificación de medicamentos o la gestión deficiente e insegura de datos críticos almacenados. La principal aplicación de esta tecnología en el sector está destinada a ser que los sistemas se centren en el paciente y hagan más sencillo el intercambio de datos clínicos de manera segura y eficiente.

Algunos de sus usos que podemos identificar a primera vista son, la verificación y trazabilidad de los medicamentos, la recopilación y transferencia de datos de dispositivos portátiles (marcapasos, pulseras inteligentes, ...) a hospitales o el intercambio de información del paciente entre diferentes centros médicos por el mundo. A continuación describiremos algunos proyectos actuales de blockchain en el sector sanitario.

- MiPasa: se trata de un proyecto conjunto entre IBM, Microsoft y Oracle junto a la Universidad John Hopkins y la OMS utilizado para recopilar datos confiables y de calidad en la detección temprana e inmutable de portadores asintomáticos de enfermedades como el coronavirus.

- Chronicled (San Francisco): aplica blockchain y el internet de las cosas (IoT) a las cadenas de suministro para mejorar la trazabilidad de productos sensibles a la temperatura y/o el paso del tiempo, como las vacunas.

- Medicalchain: una plataforma blockchain destinada a crear un registro electrónico inmutable y compartirlo con instituciones médicas si el paciente lo permite, admitiendo la telemedicina con pacientes a través de sesiones en línea.

2.3 Historiales Médicos

En España, la ley 41/2002 [20] de 14 de noviembre, de Autonomía del Paciente y los Derechos y Obligaciones en materia de Información y Documentación Clínica en su artículo 14 nos indica que la historia clínica se compone por el conjunto de documentos relativos a los procesos asistenciales, identificando a todo el personal sanitario que ha intervenido en ellos. El objetivo principal de esta es lograr la máxima integración posible de la documentación clínica de cada paciente, como mínimo en el ámbito de cada centro. Por tanto, podemos definir un historial médico como un documento legal que se encarga de recoger todos los datos relativos a los servicios sanitarios y la salud de un paciente para proporcionar una asistencia médica adecuada.

2.3.1. ¿Cómo se utiliza?

Una historia clínica se usa cada vez que un paciente acude a un centro sanitario para proporcionar acceso a toda la información médica del paciente al personal médico que lo requiera. Posibilitando así una atención médica personalizada basándose en los datos del paciente, aportando la información necesaria para un correcto diagnóstico que apoye la decisión sobre qué tratamiento es el más adecuado para cada paciente. A su vez, esta se convierte en la herramienta de control y evolución del paciente.

2.3.2. Datos a incluir en una historia clínica

Independientemente del formato que se utilice para almacenarla, debe incluir una serie de datos mínimos sobre el paciente como:

- Datos del paciente que permitan su identificación.
- Anamnesis y exploración física.
- Informes de urgencia.
- Evolución clínica cronológicamente.
- Órdenes médicas cursadas.
- Exploraciones complementarias solicitadas por el personal sanitario.
- Hoja de interconsulta.
- Consentimiento informado del paciente en el que da permiso para la realización de tratamiento o intervenciones quirúrgicas si lo requiere.
- Informe de anestesia, quirófano y anatomía patológica.
- Evolución y planificación de cuidados tras intervención.
- Aplicación terapéutica de enfermería.
- Gráfico de constantes del paciente.
- Informe clínico de alta del paciente.

Cabe destacar que actualmente esta información comentada a incluir puede depender de cada comunidad autónoma en España y ubicación del centro concreto. Sin embargo, aunque varíe, el tipo de datos incluido suele ser similar a esto.

2.3.3. Historia clínica electrónica (HCE) en España

La historia clínica electrónica es el equivalente digital a un historial médico en papel, es decir un registro informatizado de todos los datos médicos relevantes para la atención sanitaria de un paciente.

Historia y situación de la HCE en España

En nuestro país, la historia clínica electrónica fue implantada por primera vez entre 1993 y 1994 en Andalucía. Esta experiencia se llevó a cabo en tres centros de salud de esta comunidad mediante el envío telemático de los partes de incapacidad temporal, lo que sentó las bases del Proyecto TASS. Se trataba de un proyecto de colaboración entre la Consejería de Salud Andaluza y el Ministerio de Trabajo y Seguridad Social para implantar por primera vez una tarjeta sanitaria en la comunidad andaluza.

En el año 2005 el Gobierno de España comenzó con la puesta en marcha de las prescripciones electrónicas que han servido para reducir en un 15 por ciento [21] aproximadamente las visitas médicas de atención primaria en aquellas comunidades en las que se utilizan. Cinco años más tarde ya se habían enviado más de 250 millones de prescripciones electrónicas y más de un 95 % [21] de los médicos recurrieron a la HCE.

Más adelante, en el año 2014, el Hospital de Dénia (Comunidad Valenciana) fue declarado como primer centro clínico español en alcanzar el nivel 7 de la Sociedad de Sistemas de Información y Gestión en Sanidad (HIMSS) tras implantar la historia clínica para toda la documentación relativa a los pacientes, lo que le acredita como un centro 100 % [21] digitalizado. Desde entonces, cada comunidad autónoma está haciendo esfuerzos para implantar el HCE mientras que el gobierno central trata de asegurar la interoperabilidad a través del programa Sanidad en línea. Todo esto ha llevado a España a ocupar una posición de liderazgo internacional en lo referente a la historia clínica electrónica.

El gran problema de la HCE: la interoperabilidad

El principal problema que nos encontramos con la HCE en España es la interoperabilidad. Esto es debido a que cada comunidad gestiona las historias clínicas a su manera, incluso dentro de una comunidad cada centro lo realiza de una forma diferente, lo que dificulta el intercambio de historias ocasionando que desde diferentes centros no se pueda acceder a la misma información de un paciente. Por ejemplo alguien que usualmente es tratado en Madrid, si un día necesita ser tratado en otro lugar como puede ser Valencia, es altamente probable que desde el centro valenciano no sea posible acceder a la información registrada desde el centro madrileño.

Parece que se está tratando de tomar medidas ante esta situación con soluciones como una tarjeta sanitaria única que favorecerá la interconexión de los datos médicos de un paciente dentro del territorio español. De cualquier manera, las soluciones propuestas parecen ser lejanas, pues para ellas sería necesaria un proceso de estandarización para los historiales médicos que todavía no existe.

CAPÍTULO 3
Ethereum

Para el desarrollo de este proyecto se ha escogido como red de blockchain a Ethereum. Como ya hemos comentado previamente, Ethereum es una de las plataformas actuales más importantes de este ecosistema y cuenta con una serie de características que consideramos necesarias de un análisis más profundo.

3.1 Transacciones en Ethereum

Las transacciones en Ethereum son instrucciones firmadas criptográficamente que se emiten desde cuentas que las inician para actualizar el estado de la red, siendo una de las más sencillas la transferencia de ETH de una cuenta a otra. Aquellas transacciones que modifiquen el estado de la red deberán ser transmitidas por toda la red para su posterior validación y propagación. Estas transacciones requieren una tarifa y han de incluirse en un bloque validado.

Una transacción debe incluir los siguientes datos:

- **Desde:** la dirección del remitente, que debe firmar la transacción.
- **A:** la dirección de recepción de la transacción.
- **Firma:** identificador del remitente que se genera cuando el remitente autoriza la transacción.
- **Nonce:** un contador que se incrementa secuencialmente indicando el número de transacciones de la cuenta.
- **Valor:** la cantidad de ETH a transferir del remitente al destinatario.
- **Datos de entrada:** entrada opcional para incluir datos arbitrarios.
- **Límite de gas:** cantidad máxima de gas que el remitente está dispuesto a pagar como tasa.
- **Máximo gas prioritario:** máximo de gas consumido que recibirá el validador del bloque como propina.
- **Tasa máxima de gas:** máxima comisión de gas destinada al pago de la transacción.

En el apartado 3.4, explicaremos más a fondo el concepto de gas.

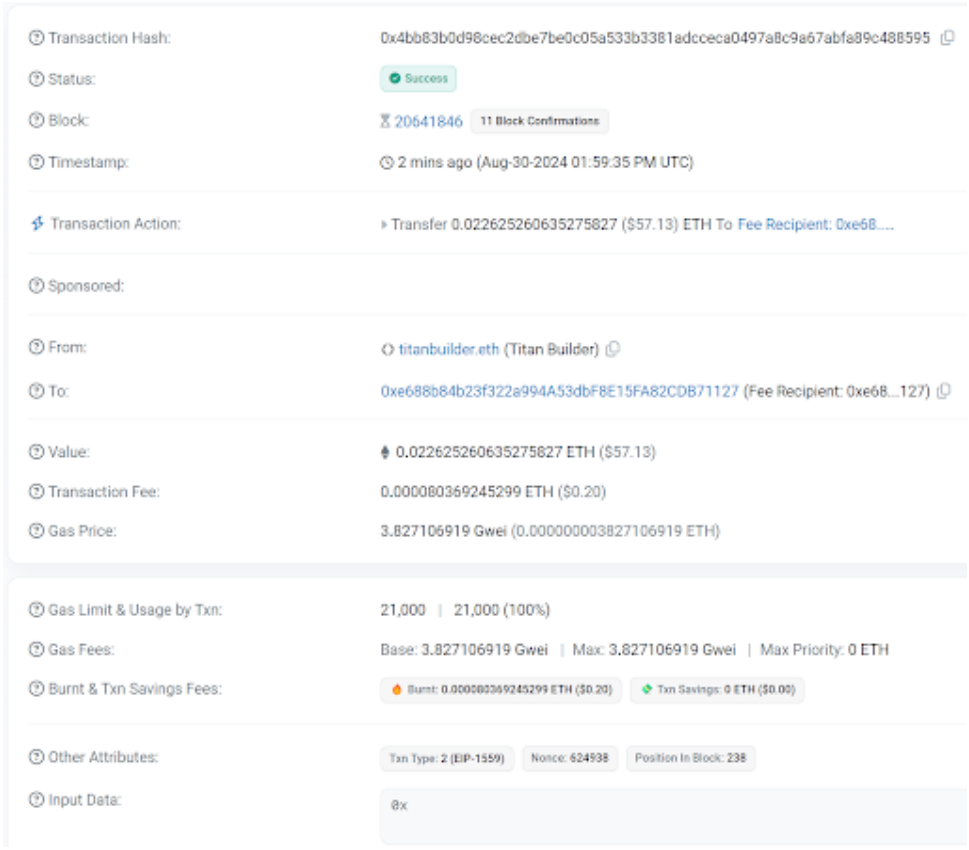

Figura 3.1: Ejemplo de transacción obtenido de [23]

3.2 Bloques de Ethereum

Los bloques en Ethereum son lotes de transacciones que contienen el *hash* del bloque anterior para poder vincularlos formando así una cadena. Esto es debido a que el *hash* deriva del contenido del mismo y si se tratara de modificar alguno serían invalidados los siguientes, previniendo así el fraude.

Cada bloque puede contener diferentes tipos de información dependiendo de su cometido y de los datos introducidos por él. A continuación se presenta un ejemplo de un bloque cualquiera de la red principal de Ethereum.

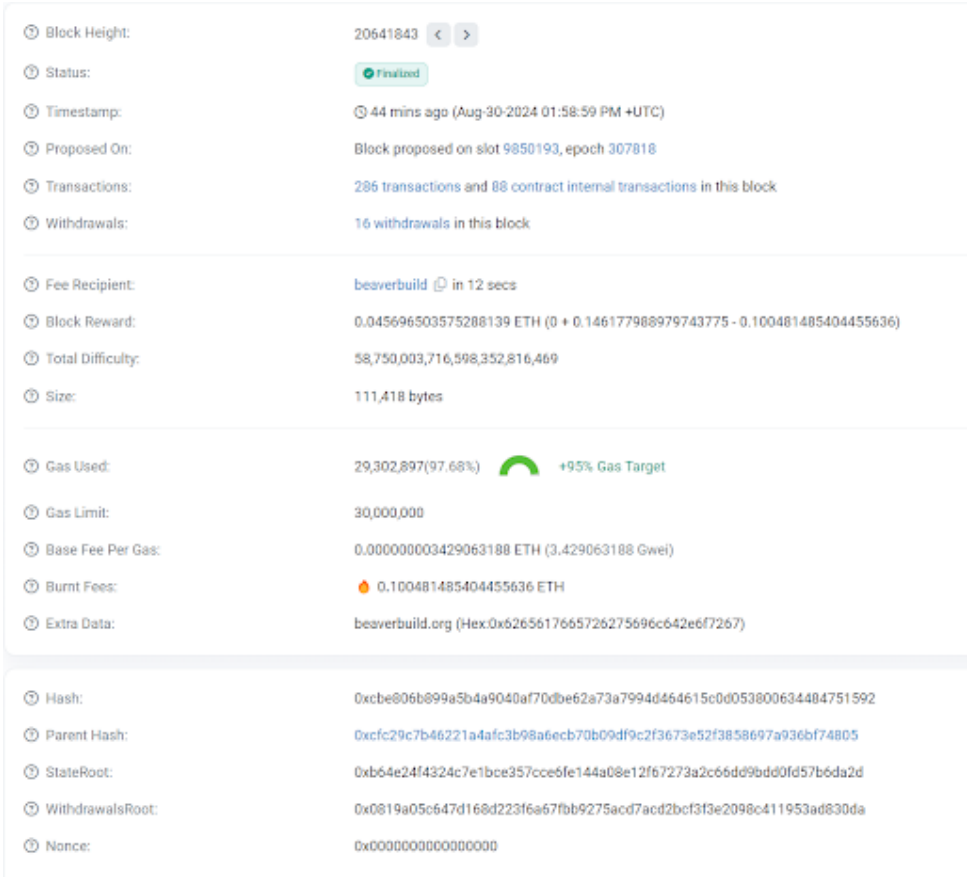

Figura 3.2: Ejemplo de bloque obtenido de [23]

Alguno de los campos más relevantes de los observables en la figura 3.2 son el tamaño de bloque o la cantidad de transacciones que contiene. También podemos ver la recompensa otorgada al validador por sellarlo y el gas utilizado en el proceso.

3.3 Máquina virtual de Ethereum (EVM)

No es posible describir la representación física de la EVM, pero existe como una única entidad sustentada por miles de computadoras conectadas que ejecutan un cliente de Ethereum. El protocolo propio de esta plataforma está diseñado específicamente para mantener el funcionamiento ininterrumpido e inmutable de esta máquina de estado especial.

Esta máquina es el entorno en el que conviven todas las cuentas de Ethereum y los contratos inteligentes. En cualquier bloque, Ethereum tiene un estado único y la EVM es la encargada de definir las reglas de cálculo para un nuevo estado bloque a bloque.

La máquina virtual de Ethereum es la encargada del despliegue y ejecución de los contratos inteligentes mientras que aquellas transacciones más simples no requieren de sus recursos. En estos contratos es la que gestiona funcionalidades como operaciones aritméticas y lógicas, acceso a la memoria, pila y almacenamiento, llamadas y operaciones de flujo de control, trabajando siempre con un tamaño de palabra de 32 bytes. Además de esto, podemos considerarla omnisciente debido al conocimiento total que tiene sobre todo tipo de información que podamos imaginar relacionada a una cuenta, desde su dirección hasta saldo en ETH.

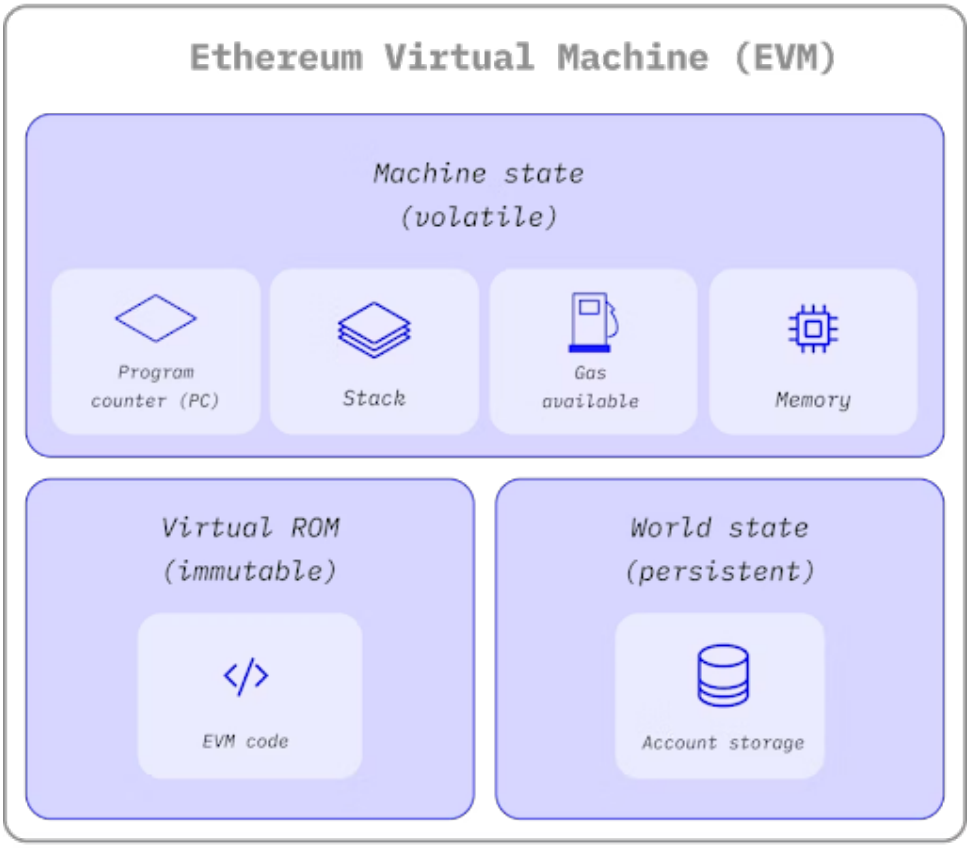

Figura 3.3: Imagen extructural de la EVM [22]

3.4 ¿Qué es el gas?

Anteriormente hemos mencionado el término gas y es necesario que lo comprendamos debido a su papel fundamental en Ethereum. El gas hace referencia a la unidad que mide el esfuerzo computacional para ejecutar alguna operación en esta red. Esto sirve como una manera de evitar el doble gasto comentado previamente ya que, al establecer una tarifa por los recursos informáticos utilizados aseguramos que la red de Ethereum no sea vulnerable a ello. La tarifa de gas es la cantidad de gas utilizado multiplicado por el coste unitario del propio gas. Cabe comentar que aunque la transacción no resulte exitosa, se debe pagar igualmente esta tarifa.

Esta tarifa es pagada en ETH y suele ser expresada en gwei, una unidad en la que cada gwei es equivalente a 10^{-9} ETH (0,000000001 ETH). Como hemos visto, se puede establecer la cantidad máxima que se desea pagar por la transacción, pero hemos de ser cuidadosos, pues una cantidad baja pondrá en riesgo la disposición de un validador a adoptarla puesto que la recompensa obtenida será menor y una cantidad elevada nos podría hacer desperdiciar ETH.

Por eso es necesario entender que la cantidad total de gas a pagar se divide en la tarifa base establecida en el protocolo que indica la cantidad mínima a pagar para que la transacción se considere válida y la tarifa prioritaria, que consiste en la propina que se añade a la base destinada a los validadores. Esto nos deja la siguiente fórmula para calcular el total de gas a pagar por una transacción:

- Unidades de gas usadas * (Tarifa básica + Tarifa prioritaria)

El límite de gas hace referencia a la cantidad máxima que está dispuesto a pagar el usuario por su transacción y este dependerá de la complejidad de la transacción y/o del contrato inteligente que esté asociado.

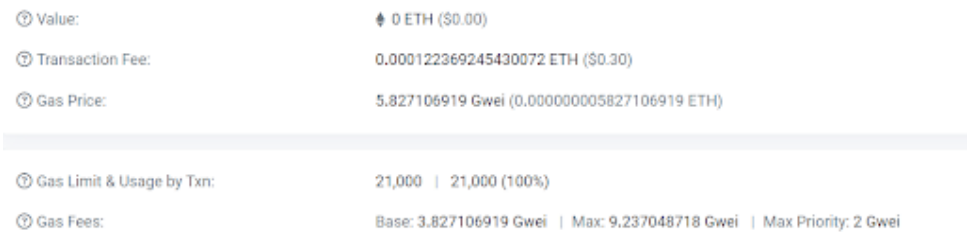

Figura 3.4: Ejemplo de datos de Gas en una transacción obtenido de [23]

3.5 Redes de Ethereum

Hasta ahora Ethereum cuenta con gran cantidad de redes propias que son públicas y están en continuo funcionamiento. Desde la red principal, pasando por redes de prueba utilizadas por los desarrolladores para observar el funcionamiento de nuevos protocolos o comprobar el funcionamiento de los contratos en entornos similares a los de producción, hasta redes destinadas puramente al desarrollo.

Las redes de Ethereum son conjuntos de ordenadores que utilizan el protocolo de esta plataforma para crear una blockchain. Aunque solo existe una red principal, podemos crear otro tipo de red independiente para otros fines dependiendo de lo que necesitemos.

Los tipos más importantes de sus redes son:

- **Red principal:** es la red de producción de Ethereum pública primaria en la que las transacciones son registradas en el libro mayor de contabilidad distribuido. Cuando hablamos de transacciones de criptomonedas o similares reales nos referimos a las que ocurren en esta red.
- **Redes de pruebas:** estas redes son vitales debido a la importancia que adquiere testar el código desarrollado antes de implementarlo en la red principal. Además, la mayoría de aplicaciones distribuidas integradas con contratos inteligentes suelen albergar diferentes copias distribuidas en este tipo de redes. Algunos ejemplos de ellas que son totalmente públicas son Sepolia o Holesvoice.
- **Redes de desarrollo:** en este caso hablamos de redes privadas muy similares a las de pruebas, que podemos comparar con un servidor de desarrollo local tradicional a la hora de elaborar un producto o aplicación software.

3.6 Smart Contracts

La realidad es que el término de contrato inteligente es inapropiado ya que estos programas que hemos visto con anterioridad, realmente no son ni contratos, ni inteligentes. Los podemos definir como programas que se ejecutan en la blockchain de Ethereum y se componen por un

grupo de código y datos que existen en una dirección específica. Internamente estos contratos son tratados como un tipo de cuenta que tiene un saldo y que puede ser objetivo de una transacción.

Sin embargo, a diferencia de otros tipos de cuentas, no están controlados por un usuario y están ubicados en la red, ejecutándose según se haya programado. Las cuentas de usuario pueden comunicarse con un contrato enviando transacciones que ejecuten algún método o función definida en este. Cabe resaltar que por defecto un contrato no puede ser eliminado y las interacciones con ellos son irreversibles.

Estos contratos son públicos y se pueden considerar API abiertas con capacidad de invocar desde ellos a otros contratos existentes. De hecho, un contrato puede implementar a otro dentro de sí mismo. Sus limitaciones residen en que por sí solos no son capaces de obtener información de eventos que ocurren en el "mundo real" porque se limitan al ámbito de la cadena. Esta limitación puede ser solucionada por los conocidos como oráculos, que son los encargados de procesar los formatos fuera de la cadena y ponerlos a su disposición.

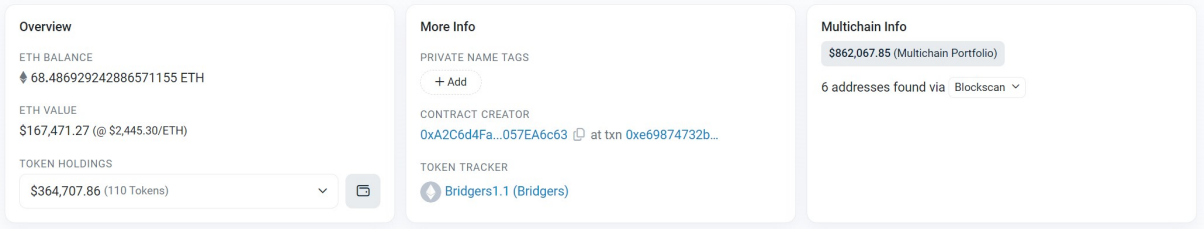

Figura 3.5: Ejemplo de contrato real obtenido de [23]

3.7 Solidity

Solidity es un lenguaje de alto nivel orientado a objetos utilizado para implementar contratos en la red de Ethereum que usa una sintaxis similar a la de JavaScript y se ejecuta a través de la EVM. Se compone de un tipado estático que acepta, entre otras cosas, herencia, librerías y tipos complejos que son definidos por el usuario.

Los tipos que existen dentro de Solidity son similares a los existentes en la mayoría de lenguajes orientados a objetos. Los más básicos e importantes de Solidity son:

- **Booleanos:** los valores que puede adoptar son true y false, adquiriendo los más que conocidos operadores lógicos && (and), || (or) , ! (not) , == (equal) y != (not equal). Para declararlos usamos *bool*.

- **Enteros:** números enteros con o sin signo y para su declaración se usan las palabras clave de uint8 a uint256 (sin signo desde 8 a 256 bits) y int8 a int256. Cabe mencionar que las palabras *int* y *uint* por sí solas corresponden a su versión de 256 bits.

- **Cadenas:** estos se corresponden con cadenas de caracteres y son declarados mediante la palabra *string*.

Otros tipos que no encontramos habitualmente en otros lenguajes y que merece la pena conocer son:

- **Address:** contienen un valor de 20 bytes y hacen referencia a direcciones cuyas propiedades podemos consultar mediante sus miembros como *balance* que nos devuelve el saldo de esta.
- **Mapping:** estos son declarados como mapping(TipoClave =>TipoValor) donde TipoClave puede ser cualquier tipo excepto mapping. Para comprender este tipo de una manera sencilla, podemos verlo como un diccionario o tabla *hash* donde al realizar una "llamada" introduciendo un valor del tipo TipoClave nos devolverá el valor correspondiente de tipo TipoValor.

Como los contratos se ejecutan dentro de la EVM, cabe destacar que podemos obtener acceso a ciertos objetos globales. Uno de esos objetos que resulta muy útil es msg, que es la llamada de la transacción. Mediante este podemos acceder a msg.address que nos devuelve la dirección desde la que se originó la transacción y a msg.value que nos devuelve el valor enviado en esta.

3.7.1. Entornos de desarrollo para Solidity

La compilación y ejecución de contratos inteligentes es facilitada por diferentes entornos de desarrollo como:

- **Visual Studio Code:** mediante la instalación de una extensión que incluye el compilador de Solidity.
- **SublimeText:** tan solo con un paquete que permite resaltar la sintaxis de este lenguaje.
- **Atom:** con un plugin llamado Eheratom que incluye desde resaltar la sintaxis hasta un entorno en tiempo de ejecución.
- **IDEs JetBrains:** todos los entornos de JetBrains lo ofrecen mediante un plugin.
- **Remix:** siendo este un entorno de desarrollo basado en navegador que integra un compilador y un entorno en tiempo de ejecución para Solidity.

3.7.2. Estructura de un contrato inteligente

La realidad es que la estructura de un contrato inteligente puede resultar muy parecida a la definición de una clase en un lenguaje de programación común debido a su orientación a objetos. Es recomendable anotar el fichero fuente de un contrato con una versión de pragma para que este no se compile con una posterior de la deseada que pueda incluir algún cambio incompatible.

También es posible declarar eventos y lanzarlos cuando lo consideremos necesario lo que nos proporciona información de gran valor a la hora de depurar. Por ejemplo, en un contrato en el que definimos a un usuario, podemos crear un evento que devuelva la información que deseemos asociada al usuario y lanzarlo cuando creamos uno nuevo, permitiendo visualizar datos acerca del mismo.

```
pragma solidity >= 0.7.0;
contract Coin {
    // The keyword "public" makes variables
    // accessible from other contracts
    address public minter;
    mapping (address => uint) public balances;
    // Events allow clients to react to specific
    // contract changes you declare
    event Sent(address from, address to, uint amount);
    // Constructor code is only run when the contract
    // is created
    constructor() {
        minter = msg.sender;
    }
    // Sends an amount of newly created coins to an address
    // Can only be called by the contract creator
    function mint(address receiver, uint amount) public {
        require(msg.sender == minter);
        require(amount < 1e60);
        balances[receiver] += amount;
    }
    // Sends an amount of existing coins
    // from any caller to an address
    function send(address receiver, uint amount) public {
        require(amount <= balances[msg.sender], "Insufficient balance.");
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Figura 3.6: Ejemplo de contrato inteligente con estructura completa

CAPÍTULO 4
Marco de trabajo

Para el desarrollo de este proyecto seguiremos una estructura común en el desarrollo de aplicaciones web, que divide una aplicación en dos secciones. Una de ellas es conocida como *frontend* y se refiere a la parte con la que interactúan los usuarios, habitualmente compuesta por html y css para construir la estructura y diseño de la interfaz de usuario y por JavaScript, que proporciona la interactividad necesaria. La otra parte se denomina *backend* y se encarga de gestionar la lógica de la aplicación y la base de datos, otorgando los datos necesarios a la parte *frontend*. Ahora veremos el modelo de desarrollo a seguir para posteriormente seleccionar las tecnologías que utilizaremos a lo largo del proceso de implementación de nuestra aplicación.

4.1 Modelo de desarrollo

Para desarrollar este proyecto seguiremos un modelo de desarrollo en cascada. Este modelo se caracteriza por seguir un procedimiento lineal que divide los procesos en sucesivas fases del proyecto. Al contrario que otros modelos iterativos como la metodología ágil, las fases de este se ejecutan solo una vez. La elección de este modelo se debe a que facilita la generación de documentación de una manera cronológicamente ordenada y permite estimar los costes y la carga de trabajo al comienzo del proyecto.

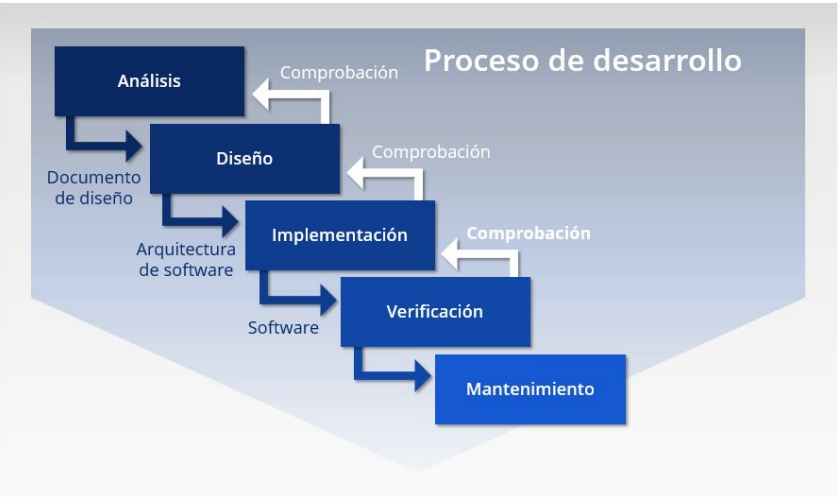

Figura 4.1: Esquema del modelo de desarrollo en cascada [35]

4.2 Tecnologías del Front-End

4.2.1. React

Esta librería [25] de JavaScript es una de las más populares actualmente en el desarrollo tanto de aplicaciones web, como móviles. Una de sus particularidades es el uso de JSX, una extensión de la sintaxis de JS que sirve para incrustar elementos HTML en objetos de JavaScript, permitiendo simplificar las estructuras de código complejas. Su punto clave reside en la manera de enfocar el desarrollo de interfaces, dónde estas son divididas en partes independientes y reutilizables denominadas componentes. Estos componentes funcionan de manera similar a las funciones ya que permiten el paso de argumentos que pueden ser utilizados dentro de estos.

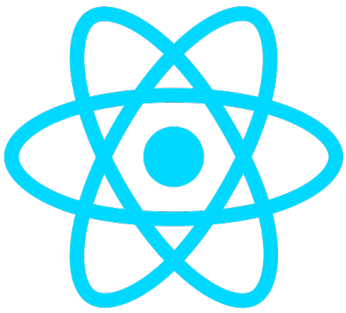

Figura 4.2: Logo de React

4.2.2. Vite

Esta herramienta de compilación tiene como objetivo otorgar una una experiencia de desarrollo y ágil para proyectos web. Un aspecto interesante de Vite [27] es que viene con configuraciones predeterminadas listas para usar, integrándose a la perfección con librerías como React. Vite nos servirá de gran ayuda especialmente a la hora de iniciar nuestro proyecto proporcionando una plantilla sobre la que empezar a desarrollar el código que deseemos.

Figura 4.3: Logo de Vite

4.2.3. NodeJS

Node.js [28] es un entorno JavaScript en tiempo de ejecución que se creó para solucionar el hecho de que JS solo pudiera ser usado en páginas web complementando la parte *frontend*. Este entorno se ejecuta en el motor de tiempo de ejecución JavaScript V8, convirtiendo el código en uno de nivel más bajo que la computadora puede ejecutar sin necesidad de interpretarlo primero, aumentando su velocidad.

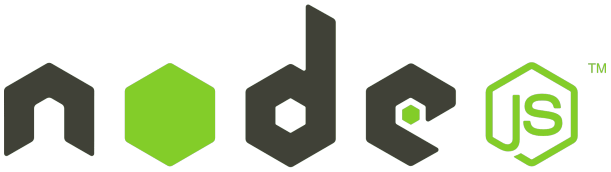

Figura 4.4: Logo de NodeJS

4.3 Tecnologías del Back-End

En nuestro caso concreto, la parte *backend* estará compuesta en su totalidad por blockchain, sustituyendo el uso de una base de datos tradicional debido a que los datos serán almacenados en la cadena de bloques.

4.3.1. Ethereum

El entorno de blockchain que utilizaremos será el previamente analizado Ethereum. Hemos escogido esta red por encima de cualquier otra existente debido a su gran comunidad de desarrolladores que aportan gran cantidad de documentación y librerías que nos facilitarán la elaboración de nuestra DApp. Para el código, utilizaremos el lenguaje Solidity, previamente comentado.

Figura 4.5: Logo de Ethereum

4.3.2. Truffle

Truffle Suite [30] es una herramienta que resulta esencial en el desarrollo de aplicaciones descentralizadas sobre Ethereum, permitiendo a los desarrolladores desarrollar, implementar y vincular Smart Contracts de una forma sencilla y rápida. Destaca por su facilidad de uso desde la interfaz de línea de comandos, la automatización de pruebas que provee y una gestión de dependencias simplificada. Bien es cierto que puede resultar ser algo complicada de utilizar para desarrolladores más novatos y su limitación reside en que está diseñada para ser utilizada únicamente con Ethereum.

Fué creada en 2015 por Tim Coulter y a día de hoy se ha convertido en una de las herramientas principales para un desarrollador de contratos inteligentes sobre la red de Ethereum. Actualmente cuenta con diversas plantillas de desarrollo que pueden ser generadas con comandos para comenzar a escribir el código sobre ellas.

Figura 4.6: Logo de Truffle

4.3.3. Ganache

Ganache [31] es una blockchain Ethereum personal que es utilizada para probar contratos inteligentes. Sobre ella construiremos nuestra aplicación debido a que funciona de la misma manera que una blockchain pero en local. De esta manera, el ETH que se maneja dentro de ella es ficticio, aunque simulando los costes reales. En ella también podemos personalizar nuestra red para establecer la cantidad total de gas, balance y cuentas. Ganache también nos permite visualizar los eventos generados por nuestros contratos inteligentes, así como todas las transacciones y bloques de cada dirección. Se puede asemejar a un servidor o base de datos local que se usa en el desarrollo de software tradicional.

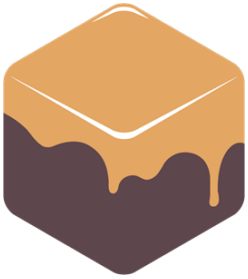

Figura 4.7: Logo de Ganache

4.4 Otras tecnologías

4.4.1. Visual Studio Code

Como entorno de desarrollo utilizaremos Visual Studio Code [33], que es un editor de código libre y multiplataforma desarrollado por Microsoft. Nos será muy útil gracias a la integración que nos aporta con una gran variedad de tecnologías y la personalización de entorno mediante extensiones que nos permite.

Figura 4.8: Logo de Visual Studio Code

4.4.2. Git

Git [34] es un sistema de código abierto que permite almacenar y mantener un historial de todos los cambios realizados en un proyecto. Esto nos permite poder realizar un seguimiento de todas las fases por las que va pasando nuestro desarrollo visualizando los cambios realizados en cada una.

Figura 4.9: Logo de Git

CAPÍTULO 5
Desarrollo del caso de estudio

Con las diferentes tecnologías necesarias seleccionadas y el modelo de desarrollo a seguir explicado, podemos comenzar con el desarrollo de nuestro proyecto. Para ello, pasaremos por cada una de las fases que conlleva el desarrollo de un producto software con la estructura de un modelo en cascada.

5.1 Estimación

Esto no es considerado una fase como tal. Sin embargo, resulta relevante tratar de estimar la carga de trabajo y tiempo que puede llevarnos desarrollar el proyecto. Las fases que tendremos en cuenta en la estimación de nuestro proceso son:

- **Análisis:** en esta fase nos dedicaremos a analizar la arquitectura de la aplicación y los posibles requisitos de nuestro proyecto.
- **Estudio de las tecnologías:** al enfrentarnos a unas tecnologías tan innovadoras es necesario incluir esta parte debido al tiempo que nos puede llevar conocer a fondo la Blockchain y su implementación.
- **Diseño:** aquí nos centraremos en dar forma a la aplicación de una manera visual en la que realizaremos esquemas que nos indiquen las diferentes pantallas a implementar. Esta fase nos permite vislumbrar aspectos que posteriormente necesitaremos implementar para el correcto funcionamiento de la DApp.
- **Desarrollo:** la fase de desarrollo es la que alberga la mayor carga de trabajo y, por tanto, la que requiere invertir mayor cantidad de tiempo y esfuerzo.
- **Pruebas:** una vez que la fase de desarrollo esté finalizando, podremos comenzar a probar diferentes aspectos de la aplicación para asegurarnos de que realmente realiza lo que deseamos y reducir la cantidad de posibles errores.

5.1.1. Coste del proyecto

De esta manera, el plan de trabajo es el siguiente:

PLANIFICACIÓN	Inicio	Final	Marzo	Abril	Mayo	Junio	Agosto
Proyecto	06/03/24	20/08/24					
Análisis	06/03/24	26/03/24					
Estudio de tecnologías	06/03/24	15/04/24					
Diseño	18/03/24	30/04/24					
Desarrollo	15/04/24	15/08/24					
Pruebas	27/05/24	20/08/24					

Tabla 5.1: Planificación del proyecto

5.1.2. Coste del proyecto

Para realizar un presupuesto, tendremos en cuenta que un desarrollador de software cobra una media de unos 12 euros la hora. El proyecto durará aproximadamente 24 semanas durante las cuáles trabjará más de lo habitual debido al desconocimiento inicial sobre estas tecnologías. Podemos establecer una media de 35 horas a la semana. Esto resulta en unos costes personales de:

- 12 €/h * (24 semanas * 35 horas) = 10080 €.

5.2 Análisis

Durante esta fase estudiaremos el problema y trataremos de encontrar que requiere tener nuestra aplicación para solucionarlo. Para ello pasaremos por la fase de especificación de requisitos y utilizaremos técnicas habituales como casos de uso que nos ayudarán en este proceso. También veremos qué estructura de carpetas nos corresponde seguir teniendo en cuenta las necesidades de la aplicación y las tecnologías elegidas.

5.2.1. Propuesta para solucionar el problema

Como bien hemos comentado en capítulos anteriores, el mayor problema de los historiales clínicos se centra en la interoperabilidad de ellos debido a que cada centro puede gestionarlos de una manera diferente, impidiendo así el acceso de la información de un paciente entre diferentes centros. Además, se suma el inconveniente de que para los pacientes también resulta verdaderamente complicado acceder a su propia información y gestionar quién tiene acceso a esta. Debemos tener también en cuenta que los datos deben ser almacenados y transferidos con la máxima seguridad posible debido a la sensibilidad de estos. Por ello proponemos el desarrollo de una aplicación descentralizada basada en blockchain. De esta manera, la información no dependerá de un ente centralizado y estará disponible para todos, sumando que la seguridad está garantizada al utilizar esta tecnología.

Por otro lado, debemos ver que el objetivo real de este proyecto es servir como apoyo a los sistemas actuales, proporcionando tan solo los datos que resulten verdaderamente importantes a la hora de atender a un paciente. Con este objetivo tan solo almacenaremos tan solo la siguiente información de un historial:

- **Datos que permitan la identificación del paciente**
- **Alergias**
- **Discapacidades o patologías**

- **Informes:** que recogerán el suceso o incidente ocurrido por el que el paciente acude, el tratamiento o procedimiento seguido por el personal médico y las observaciones o diagnóstico del médico. Además, cada informe recogerá la fecha en la que se realizó, el nombre del médico y un número de informe que deberá corresponder con el número de historia clínica a la que sirve de apoyo, por si se requiere consultar más información.

Estos datos son los que consideramos verdaderamente relevantes a consultar a la hora de atender a una persona, sobre todo en casos de urgencia. Recalcar que el sistema a construir funcionará a modo de prototipo para demostrar la utilidad de la tecnología blockchain.

5.2.2. Especificación de requisitos

Requisitos funcionales

Por requisitos funcionales nos referimos a aquellas funcionalidades que permiten al usuario lograr los objetivos de cara al usuario en la aplicación. Son puntos específicos que debe cumplir el sistema para el correcto funcionamiento de los casos de uso para cada usuario y que a menudo son identificados por las siglas RF seguidas de un número identificativo. A continuación se muestra una tabla con ellos.

Requisito	Descripción
RF-1	La app almacenará las solicitudes entre usuarios.
RF-2	Una vez aceptada una solicitud, el sistema colocará al usuario paciente en la lista de pacientes del usuario médico, sin importar quién de los dos la haya enviado y quien la haya aceptado.
RF-3	Un usuario médico deberá tener la opción en todo momento de cambiar entre su perfil de médico y paciente con una sola cuenta.
RF-4	El sistema deberá almacenar y mostrar correctamente la información correspondiente a los historiales médicos.
RF-5	La aplicación permitirá a los usuarios nuevos registrarse para crear una cuenta.
RF-6	La aplicación permitirá a los usuarios iniciar sesión en una cuenta existente manteniendo todos sus datos en diferentes sesiones.
RF-7	El servicio permitirá a los médicos filtrar entre sus pacientes por nombre y apellidos.
RF-8	La aplicación permitirá a los médicos forzar la adición de un paciente por casos de urgencia sin la necesidad de aceptación por parte de estos.
RF-9	La aplicación distinguirá entre usuarios médicos y pacientes a la hora de enviar solicitudes para evitar errores.

Tabla 5.2: Requisitos Funcionales

Requisitos no funcionales

Entendemos por requisitos no funcionales aquellas restricciones impuestas al sistema en términos de seguridad, rendimiento, disponibilidad, durabilidad, etc. Son comúnmente iden-

tificados de igual manera que los requisitos funcionales pero con las siglas RNF. Ahora mostraremos una tabla con aquellos que se apliquen a nuestra app.

Requisito	Descripción
RNF-1	La app deberá ser visible desde los navegadores más comunes como Opera, Brave, Chrome, Firefox,...
RNF-2	La información estará replicada en todos los nodos de la red blockchain.
RNF-3	La aplicación deberá informar en la medida de lo posible de todos sus errores.
RNF-4	El servicio deberá informar al usuario de todos los éxitos en sus acciones.
RNF-5	Las limitaciones de almacenamiento y rendimiento serán aquellos impuestos por la red.
RNF-6	La información personal de cada usuario será únicamente mostrada bajo el consentimiento del mismo, a excepción de cuando se realiza por urgencia, notificando al mismo.
RNF-7	La información no podrá ser modificada, sólo será posible añadir nueva.
RNF-8	Debido a la sensibilidad de los datos, se almacenarán completamente encriptados por blockchain.

Tabla 5.3: Requisitos no Funcionales

Casos de uso

Para comprender mejor la interacción entre el usuario y la aplicación analizaremos los casos de uso. Dentro de nuestro sistema debemos tener en cuenta dos tipos de usuario, médico y paciente. Ambos son necesarios para el funcionamiento de la aplicación pues las acciones a realizar requerirán de ambos en su mayoría. Procedemos a mostrar y explicar alguno de los casos de uso de nuestra aplicación.

Diagrama de casos de uso del médico:

Figura 5.1: Diagrama de casos de uso del médico

Diagrama de casos de uso del paciente:

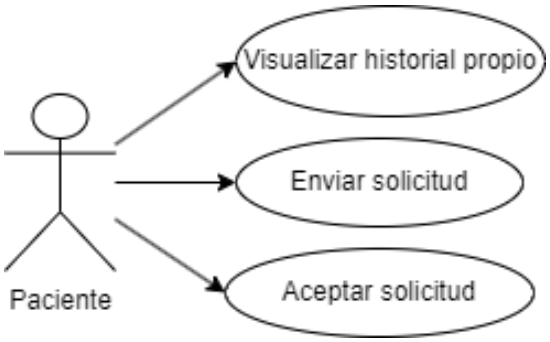

Figura 5.2: Diagrama de casos de uso del paciente

A continuación exponemos una tabla ejemplo de caso de uso para que comprendamos que tipo de información se incluirá en los casos de uso que analicemos.

Nombre	Nombre otorgado al caso de uso.
Descripción	Una pequeña descripción de que se hace en el caso.
Precondición/es	Condición o condiciones que deben cumplirse para poder ser realizado si las hubieran.
Secuencia principal	Serie de pasos que se siguen para ser realizado.
Alternativas/Errores	Posibles errores y su gestión.
Postcondición	Condición que debe cumplirse cuando se termine el caso si la hubiera.

Tabla 5.4: Tabla de ejemplo de un caso de uso

Nombre	Visualizar el historial de un paciente.
Descripción	Permite visualizar el historial médico de un paciente.
Precondición/es	El paciente cuyo historial se quiere visualizar debe estar previamente asignado al médico que lo desea.
Secuencia principal	1.El médico inicia sesión en la aplicación. 2.El sistema devuelve en la pantalla de inicio los pacientes del médico. 3.El médico busca al paciente y pulsa sobre el botón correspondiente. 4.El sistema devuelve el historial del paciente.
Alternativas/Errores	Si sucede un error el sistema muestra una ventana modal informando sobre este.
Postcondición	El médico debe poder ver el historial y sus datos asociados.

Tabla 5.5: Caso de uso de visualizar historial médico de un paciente. Correspondiente a el **RF-4**

Nombre	Añadir información a un historial
Descripción	Permite añadir nuevos datos al historial de un nuevo paciente.
Precondición/es	Debe completarse previamente el caso de uso "Visualizar el historial de un paciente".
Secuencia principal	1.El médico selecciona el tipo de información que desea añadir y pulsa sobre el botón añadir. 2.El sistema devuelve una pantalla que permite al médico introducir la información correspondiente. 3.El médico añade los datos que desee y pulsa sobre confirmar. 4.El sistema almacena los datos mostrandolos en el apartado correspondiente.
Alternativas/Errores	Si sucede un error el sistema muestra una ventana modal informando sobre este. Si el médico no cumplimenta un campo necesario el sistema informa al médico.
Postcondición	La información debe de haber sido almacenada correctamente.

Tabla 5.6: Caso de uso de añadir información a un historial. Correspondiente a el **RF-4**

Nombre	Aceptar solicitud.
Descripción	Permite aceptar una solicitud que tenga el usuario.
Precondición/es	Debe tener alguna solicitud pendiente.
Secuencia principal	1.El usuario se dirige a la sección solicitudes. 2.El sistema muestra una pantalla con las solicitudes pendientes de la aceptación del usuario. 3.El usuario busca la solicitud que desea y pulsa sobre aceptar. 4.El sistema informa del éxito, vinculando ambos usuarios.
Alternativas/Errores	Si ocurre algún error se informa al usuario mediante una alerta
Postcondición	Los usuarios deben haber sido vinculados con éxito.

Tabla 5.7: Caso de uso de aceptar solicitud. Correspondiente a el **RF-1**

Nombre	Enviar solicitud.
Descripción	Permite enviar una solicitud al tipo de usuario opuesto a fin de vincular un médico con un paciente.
Precondición/es	El usuario al que se envía la petición debe ser del tipo opuesto (médico-paciente).
Secuencia principal	1. El usuario se dirige a la sección enviar solicitud. 2. El sistema muestra una pantalla que permite al usuario introducir el nombre del usuario y un botón de enviar. 3. El usuario introduce el nombre del usuario al que desea enviar la solicitud y pulsa sobre enviar. 4. El sistema procesa la petición y se la envía al usuario, informando del éxito de envío.
Alternativas/Errores	Si no se encuentra al usuario el sistema muestra una alerta. Si el usuario no es del tipo opuesto se muestra una alerta indicándolo.
Postcondición	La petición debe haber sido enviada correctamente.

Tabla 5.8: Caso de uso de enviar solicitud. Correspondiente a el **RF-1**

Nombre	Visualizar historial propio.
Descripción	Permite al paciente visualizar la información de su historial.
Precondición/es	No.
Secuencia principal	1.El paciente inicia sesión en la aplicación. 2.El sistema devuelve en la pantalla de inicio el historial del paciente.
Alternativas/Errores	Si ocurre algún error se informa al usuario mediante una ventana modal.
Postcondición	El historial debe poder verse.

Tabla 5.9: Caso de uso de visualizar historial propio. Correspondiente a el **RF-4**

Nombre	Añadir paciente de urgencia
Descripción	Permite al médico añadir a un paciente sin su consentimiento explícito para casos de urgencia.
Precondición/es	No.
Secuencia principal	1. El médico se dirige a la sección enviar solicitud. 2. El sistema muestra una pantalla que permite al usuario introducir el nombre del usuario y un botón de forzar. 3. El médico introduce el nombre del usuario al que desea enviar la solicitud y pulsa sobre forzar. 4. El sistema advierte al médico de que solo debe ser realizado en caso de urgencia y que el paciente será informado de este acto. 5. El médico confirma la acción. 6. El sistema vincula a ambos usuarios de manera inmediata.
Alternativas/Errores	Si ocurre algún error se informa al usuario mediante una ventana modal.
Postcondición	El paciente aparecerá en la lista de pacientes del médico.

Tabla 5.10: Caso de uso de añadir paciente de urgencia. Correspondiente a el **RF-8**

5.3 Diseño

Es el turno de diseñar nuestras interfaces de usuario mediante maquetación con el uso de mockups. Estas maquetaciones las realizaremos desde Figma, una herramienta de diseño colaborativa líder en el mercado.

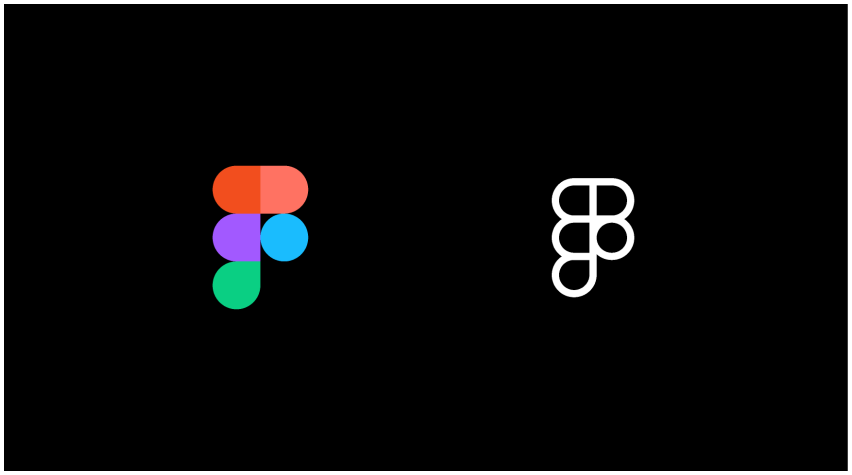

Figura 5.3: Logo de Figma

En las próximas figuras mostramos algunos diseños básicos de las interfaces de usuario de ciertas pantallas, en los que nos basaremos para implementar nuestra UI más adelante.

Figura 5.4: Diseño de la pantalla de inicio de sesión

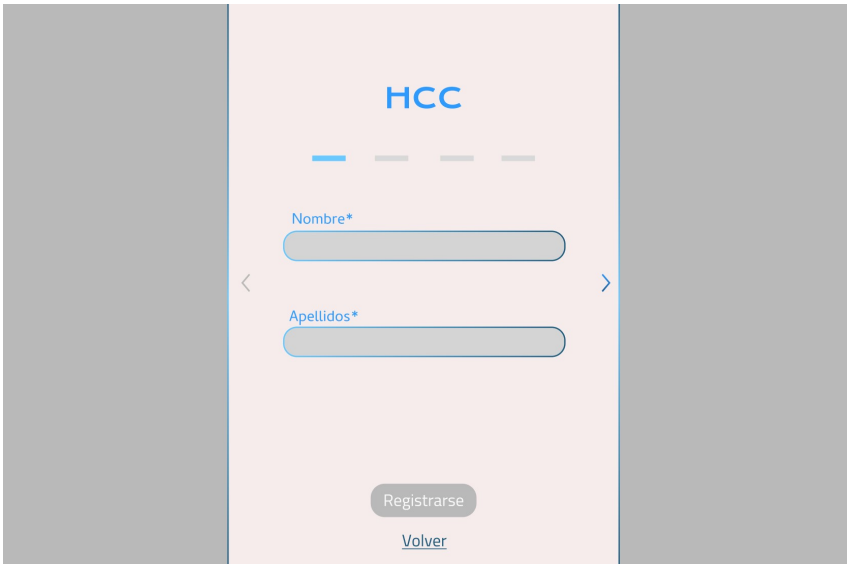

Figura 5.5: Diseño de la pantalla de registro inicial

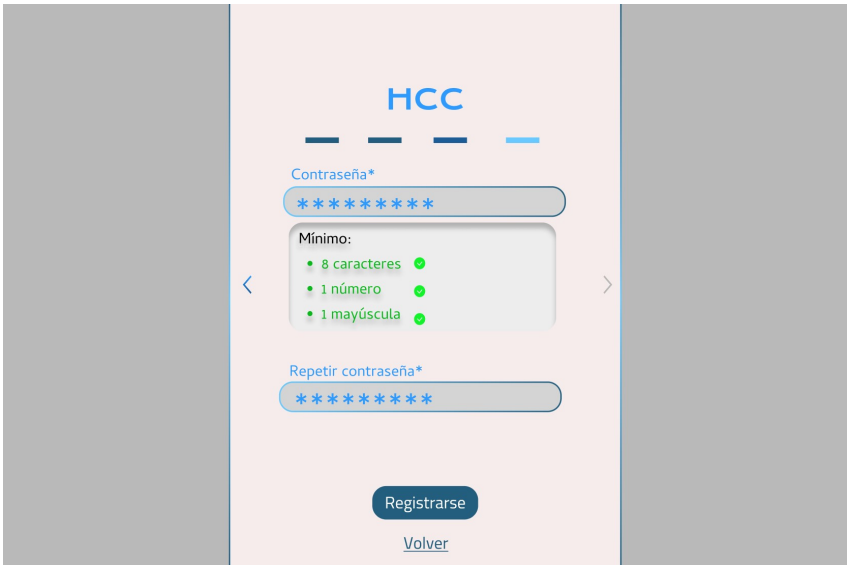

Figura 5.6: Diseño de la pantalla de registro final

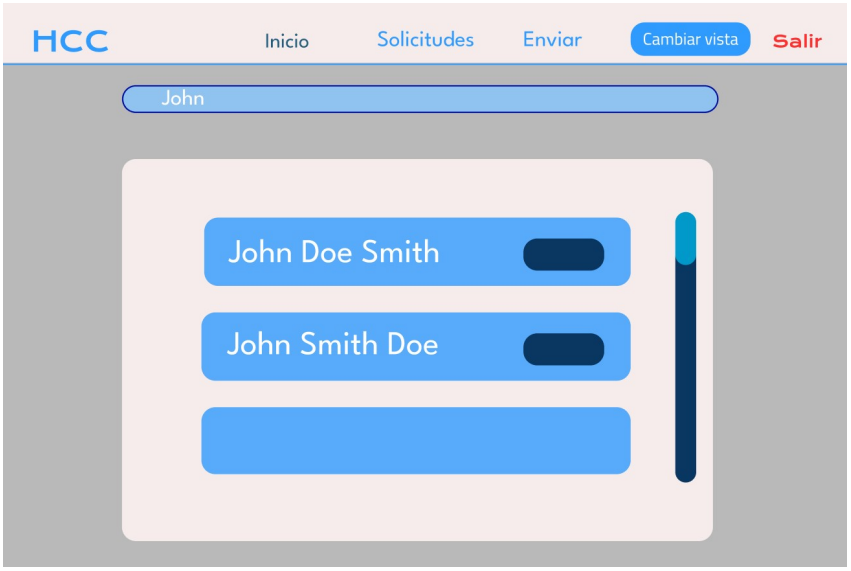

Figura 5.7: Diseño de la pantalla de lista de pacientes

Figura 5.8: Diseño de la pantalla de enviar solicitud

Figura 5.9: Diseño de la pantalla de visualizar historial

El resto de pantallas han sido diseñadas y serán desarrolladas más adelante siguiendo el estilo de estos diseños, reutilizando gran parte de los elementos que hemos incluido en estas maquetas.

5.4 Implementación

A medida que nuestros diseños van tomando forma, podemos comenzar con el desarrollo del código que compone nuestra DApp. Comenzaremos preparando nuestro entorno para finalmente ir detallando los contratos inteligentes desarrollados y crear las interfaces de usuario necesarias.

5.4.1. Entorno

Comenzaremos creando un directorio que envuelva el proyecto y dentro de este escribiremos el siguiente comando para crear la estructura de nuestro proyecto en react:

```
$ npm create vite@latest
```

Una vez ejecutada esta línea, vite nos hace seguir una serie de pasos en los que debemos ir eligiendo las tecnologías que queremos incluir en el proyecto. Esto es debido a que vite tiene compatibilidad con gran cantidad de tecnologías y versiones. En nuestro caso seleccionamos react, con JavaScript y css nativo.

Cuando completamos los pasos, se crea la carpeta src junto con plantillas de archivos importantes como nuestro App.jsx y su archivo css correspondiente. Ahora mismo podríamos iniciar el proyecto de manera local y veríamos una pantalla que muestra ejemplos de algunos componentes estilizados como botones o títulos de página. Es el momento de ir modificando los archivos css generados para ir dando nuestros propios estilos a los componentes más generales de nuestra aplicación. Con nuestra estructura *frontend* prácticamente montada, podemos comenzar a utilizar truffle para iniciar nuestro *backend* en blockchain. Para ello, dentro de nuestro directorio ejecutamos las siguientes líneas:

```
$ npm install -g truffle
$ truffle init
```

Con ellas, en primera instancia instalamos truffle de manera global y posteriormente iniciamos nuestro proyecto truffle, lo que creará los directorios /contracts, /migrations, /build, y /test junto a un archivo muy importante, truffle_config.js. En este archivo es donde deberemos configurar nuestro proyecto una vez instalemos ganache e iniciemos nuestra red local de blockchain. Hablando de ganache, una vez instalada su aplicación, deberemos configurar la red tal y como queramos. Ganache te ofrece la opción de iniciar una red de manera rápida imitando las configuraciones predeterminadas de Ethereum.

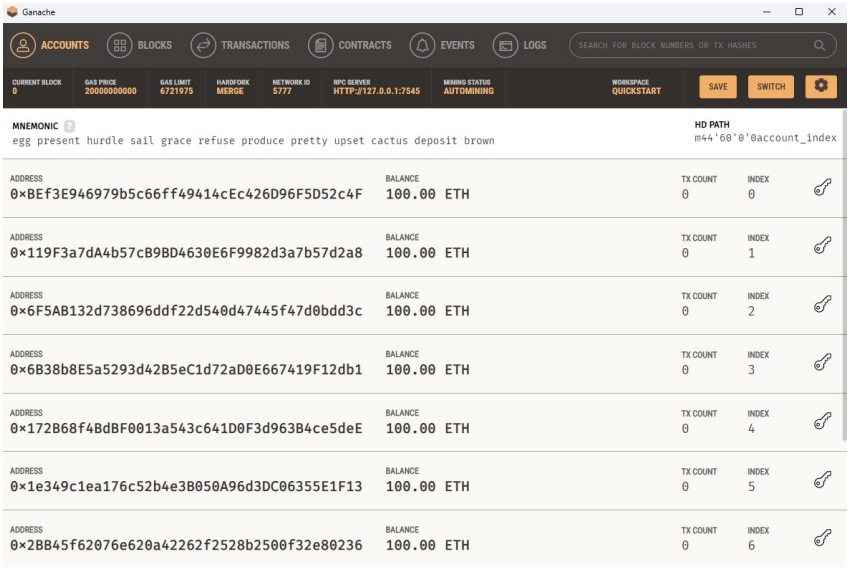

Figura 5.10: Visualización de la red desde Ganache

En nuestro caso podemos observar como se ha creado una red blockchain con diez cuentas diferentes y un balance de 100 ETH en cada una. Ciertos apartados en los que nos debemos fijar son el puerto en el que se ejecuta la red para poder conectarnos a ella y el precio de gas que nos será útil a la hora de desarrollar nuestros contratos y comunicarnos con ellos.

Con nuestra red blockchain configurada y creada, es el turno de dirigirnos a nuestro archivo de configuración truffle para establecer la conexión con esta. Este archivo está compuesto por líneas de código comentadas que están pensadas para ser descomentadas y modificadas solo si las necesitamos. Para nuestro caso nos deberemos dirigir a el apartado “networks” y dejarlo de la siguiente manera:

```
networks: {
 development: {
      host: "127.0.0.1",
      port: 7545,
      network_id: "*",
    },
  compilers: {
    solc: {
      version: "0.8.4",
      settings: {
        optimizer: {
          enabled: true,
          runs: 200
        },
       evmVersion: "byzantium"
      }
    }
  }
```

De esta manera ya hemos configurado nuestra conexión a la red local y establecido la versión de nuestro compilador para solidity junto con la optimización que le corresponde. Con todo esto ya habremos configurado nuestro entorno y es turno de comenzar el desarrollo.

5.4.2. Estructura a seguir

A pesar de ser una app compuesta por una parte *frontend* y otra *backend*, la simplicidad que nos ofrece Truffle para construir contratos inteligentes nos permite incluir todo en un mismo proyecto sin que este sea demasiado grande. Para la parte de blockchain, nos bastará con cuatro carpetas:

- **Contracts:** en ella se guardarán los contratos que necesitemos implementar.
- **Migrations:** dónde incluiremos las conocidas como migraciones, que se encargan de desplegar nuestros contratos sobre la red de blockchain.
- **Build:** aquí se guardan de manera automáticas los datos de compilación, así como la dirección de las cuentas de nuestros contratos.
- **Test:** directorio en el que almacenaremos nuestros tests sobre la parte blockchain.

Para la parte *frontend* de react, utilizaremos la carpeta Models para almacenar los modelos de datos donde recibiremos la información recibida de blockchain para transformarlos en objetos de JavaScript y la carpeta Src para incluir todo lo que conlleva el desarrollo del *frontend* en sí.

A su vez, dentro de src tendremos diferentes directorios siguiendo un modelo clásico para el desarrollo en React:

- **Assets:** para incluir recursos a utilizar como imágenes o logos.
- **Components:** dónde desarrollaremos los componentes independientes y reutilizables de nuestra app con JSX
- **Pages:** aquí desarrollaremos la vista de nuestras páginas react con la sintaxis JSX.
- **Services:** en la que crearemos nuestras llamadas a los contratos inteligentes para ser utilizadas desde los archivos de react.

5.4.3. Contratos inteligentes

Comenzaremos definiendo qué estructuras de datos necesitaremos para la implementación de nuestra aplicación. Necesitaremos al menos dos contratos, uno para gestionar los usuarios que llamaremos UserManager y otro para los historiales clínicos, ReportManager. Una vez definidas nuestras estructuras de datos en sus correspondientes contratos, podemos comenzar a gestionar aquellos métodos o funciones que necesitemos implementar en nuestra parte *backend*. Para ello resulta verdaderamente importante utilizar primero los mappings, que nos permitirán obtener valores como los usuarios totales que existen en una dirección u obtener un usuario a través de su identificador o nombre.

```
1 mapping(address => uint) public usersCount;
2 mapping(address => mapping(uint => User)) public users;
3 mapping(address => mapping(string => uint)) private userIdByUsername;
```

Aquí podemos apreciar como en "users" se utiliza un mapping al que deberemos pasar una dirección y un valor de tipo uint para que nos devuelva el usuario al que le corresponde ese valor como identificador. Un ejemplo del empleo de estos mappings es en la siguiente función:

```
function getUserByUsername(string memory _username) public view
returns (User memory) {
    uint userId = userIdByUsername[msg.sender][_username];
    if (userId == 0) {
        return invalidUser;
    }
    return users[msg.sender][userId]; // Devolver el usuario encontrado
}
```

Este método se corresponde con la obtención de un usuario por su nombre. Como podemos observar, primero obtenemos el id del usuario y después si este es igual a 0 significa que no lo ha encontrado, por lo tanto devolvemos invalidUser (una instancia de usuario que tiene valores no utilizados para ningún otro) que nos permitirá controlar el error desde nuestra parte frontal. En el caso contrario, si se ha encontrado un id válido se devuelve al usuario correspondiente.

Como cada usuario debe tener un historial clínico, debemos incluirlo en el modelo del mismo, proporcionando la siguiente función para la obtención del historial:

```
function getReport(uint _id) public view returns (ReportManager.Report memory) {
      User memory u = getUserById(_id);
      return reportManager.getReport(u.reportId);
}
```

En este método accedemos al usuario mediante su identificador, llamando a un método que actúa de manera muy similar al anterior, y devolvemos el resultado de una llamada a otra función situada en ReportManager, que se encarga de obtener el historial por id. Si observamos bien nos damos cuenta de que hay una dependencia del UserManager con el ReportManager, cuya mejor solución posible es mediante el principio de inyección de dependencias.

Inyección de dependencias

La inyección de dependencias es un patrón de diseño que pertenece a los denominados principios SOLID. Estos principios son un conjunto de patrones o reglas sobre las que se recomienda desarrollar nuestras aplicaciones para generar el código más limpio y mantenible posible. Este principio en concreto tiene como objetivo gestionar las dependencias de otros servicios o clases de una manera más limpia. Por ejemplo, sin utilizarlo, el código presentado anteriormente quedaría tal que así:

```
function getReport(uint _id) public view returns (ReportManager.Report memory) {
      ReportManager reportManager = new ReportManager(...)
      User memory u = getUserById(_id);
      return reportManager.getReport(u.reportId);
 }
```

Esto visto desde un solo método puede no parecer gran cosa, pero si requerimos de su uso de manera recurrente en diferentes funciones, deberíamos crear una instancia nueva y diferente en cada método, provocando una menor comprensión del código y más longitud que puede resultar innecesaria.

Para la aplicación de este patrón, debemos dirigirnos al constructor del UserManager y escribir lo siguiente:

```
 ReportManager reportManager;
```

```
2
3  constructor(address _reportManagerAddress) {
4      reportManager = ReportManager(_reportManagerAddress);
5  }
```

Aquí primero declaramos una propiedad del UserManager del tipo ReportManager y posteriormente le pasamos al constructor la dirección donde se despliega este contrato para que genere la nueva instancia, asignando el valor de esta a su propiedad. Por último, en nuestra migración de los contratos, deberemos pasarle dicha dirección al UserManager para que se construya correctamente y tenga el uso de su propiedad disponible en todo el contrato.

```
1 module.exports = async function(deployer) {
2   await deployer.deploy(ReportManager);
3   const reportManagerInstance = await ReportManager.deployed();
4
5   await deployer.deploy(UserManager, reportManagerInstance.address);
6 };
```

En este código podemos apreciar como a la hora de desplegar el UserManager, se le pasa como argumento la dirección de la instancia de nuestro ReportManager.

5.4.4. Conexión con nuestros contratos

Ahora es el turno de definir nuestros servicios para realizar llamadas a los métodos de nuestros contratos. Cabe destacar que antes de desarrollar estos servicios, hemos de definir los modelos de datos que recibirán la información necesaria de nuestros objetos blockchain y los métodos que transformarán los datos a objetos de JavaScript.

Al inicio de nuestros servicios, debemos de definir las siguientes cosas:

```
1 const web3 = new Web3('http://localhost:7545');
2 const contractAddress = UserManagerContract.networks['5777'].address;
3 const userManager = new web3.eth.Contract(UserManagerContract.abi,
4 contractAddress);
```

- **web3:** Servicio de web3 al que debemos pasar la dirección en la que se despliega nuestro servicio de Ganache.
- **contractAddress:** Dirección de nuestro contrato que obtenemos desde el archivo de compilación generado por Truffle.
- **userManager:** Instancia de nuestro contrato obtenida mediante el servicio de web3 y la dirección del contrato.

Con esto definido, podemos pasar a definir funciones que utilizarán estas variables para obtener los datos deseados.

```
1  export async function getUserByUsername(username){
2      try{
3          const accounts = await web3.eth.getAccounts();
4          let validUser = await userManager.methods.getUserByUsername(username)
5              .call({ from: accounts[0], gas: 2000});
6          validUser = User.fromBc(validUser);
7          if(validUser.id > 0){
8              return validUser;
9          }else{
10             return false;
```

```
11          }
12      }catch(error){
13          console.error(error);
14          throw error;
15      }
16  }
```

Este método es utilizado para llamar a una de las funciones que hemos comentado previamente en los contratos inteligentes, y gestionar su respuesta. Podemos observar aspectos interesantes, como la asignación del límite de gas que queremos gastar en la llamada. Si en vez de obtener datos deseamos crearlos, debemos sustituir "call" por "send". En general, obtenemos la respuesta de la función del contrato inteligente, la transformamos a un objeto JS, y comprobamos si el id es válido para devolver el usuario o en su defecto, devolver el valor booleano "false".

Tanto en esta sección como en la anterior, podríamos extendernos para explicar cada uno de los métodos que hemos creado y sus particularidades. Sin embargo, nos hemos limitado a explicar aquellos conceptos más relevantes sobre cómo crear contratos inteligentes e interactuar con ello debido a que consideramos que a partir de esto, se puede extrapolar lo suficiente para crear cualquier función e interactuar con ella.

5.4.5. Interfaces de usuario

En cuanto a la parte *frontend*, nuestro objetivo es tratar de aprovechar al máximo la capacidad de reutilización de componentes para facilitar el desarrollo. La mejor manera de observar nuestros resultados a la hora del desarrollo de interfaces es visualizar el resultado final de nuestras pantallas. Esto resulta interesante para evaluar si nos hemos sabido adaptar al diseño realizado y ver los cambios que hayan podido surgir. En las siguientes figuras, veremos el resultado final de nuestra UI en diferentes secciones de la aplicación.

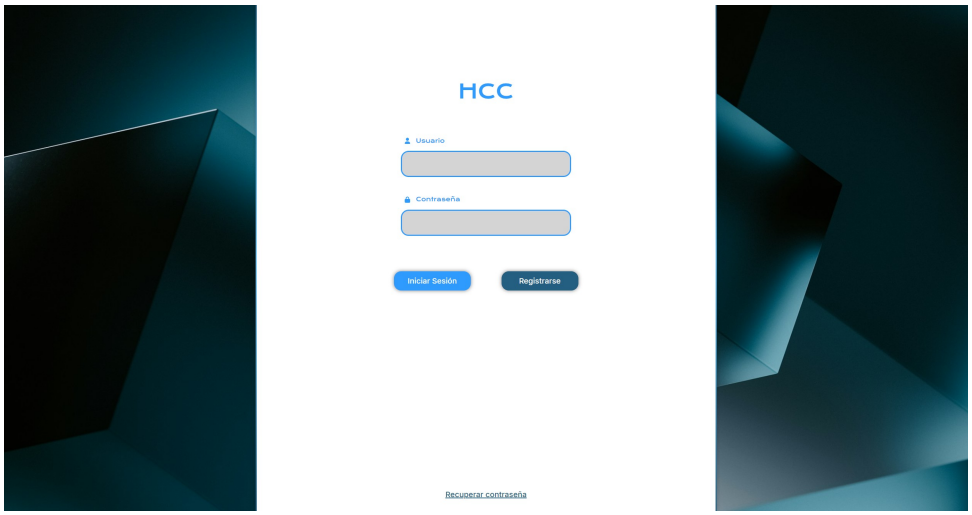

Figura 5.11: Pantalla de inicio de sesión

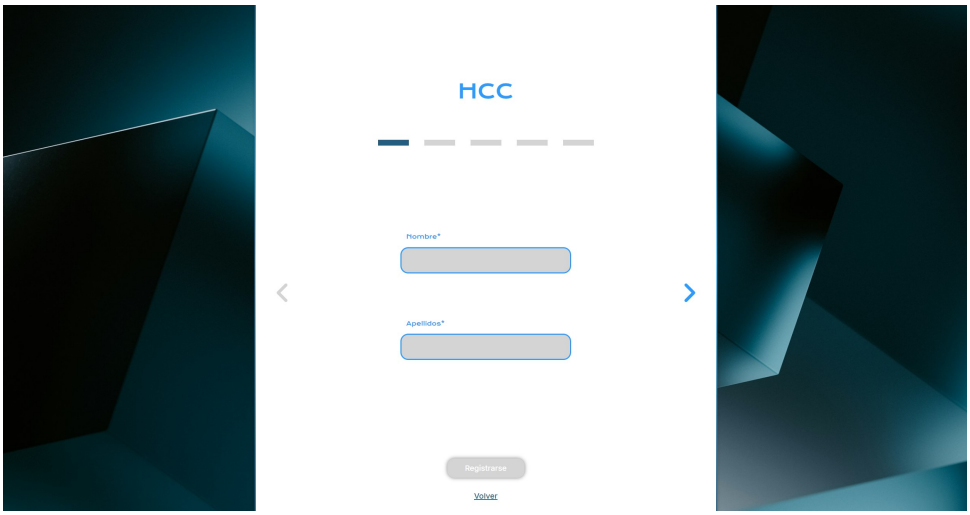

Figura 5.12: Pantalla de registro inicial

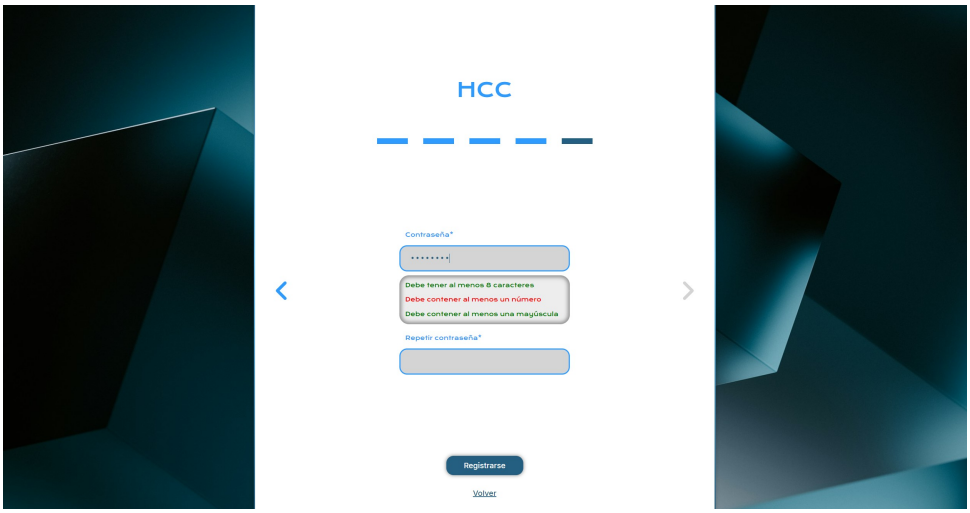

Figura 5.13: Pantalla de registro final

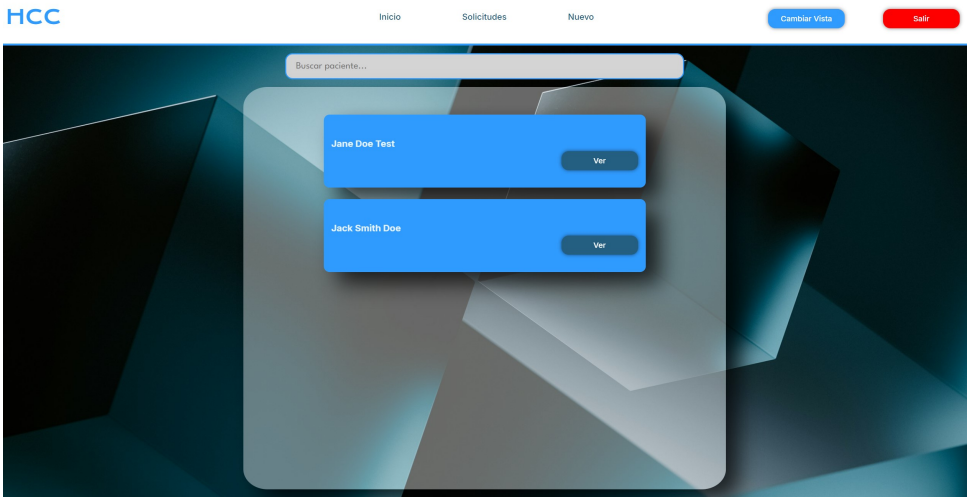

Figura 5.14: Pantalla de lista de pacientes

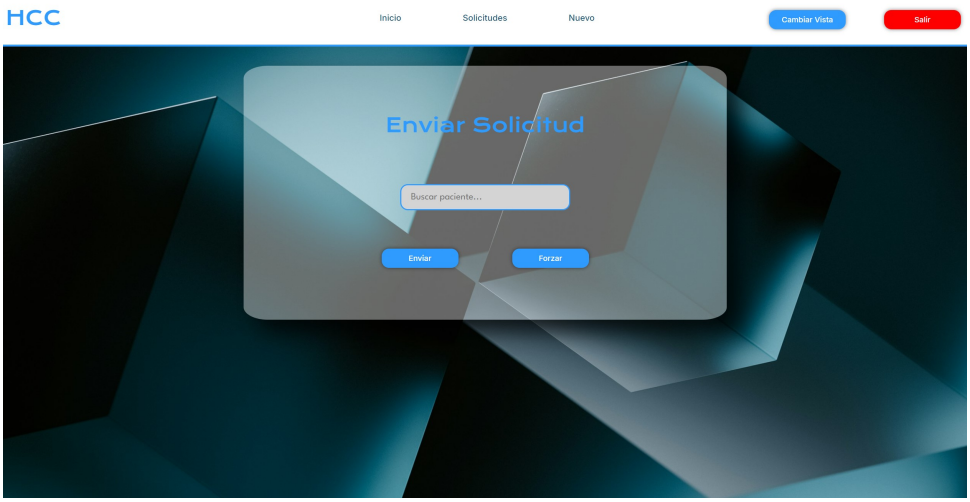

Figura 5.15: Pantalla de enviar solicitud

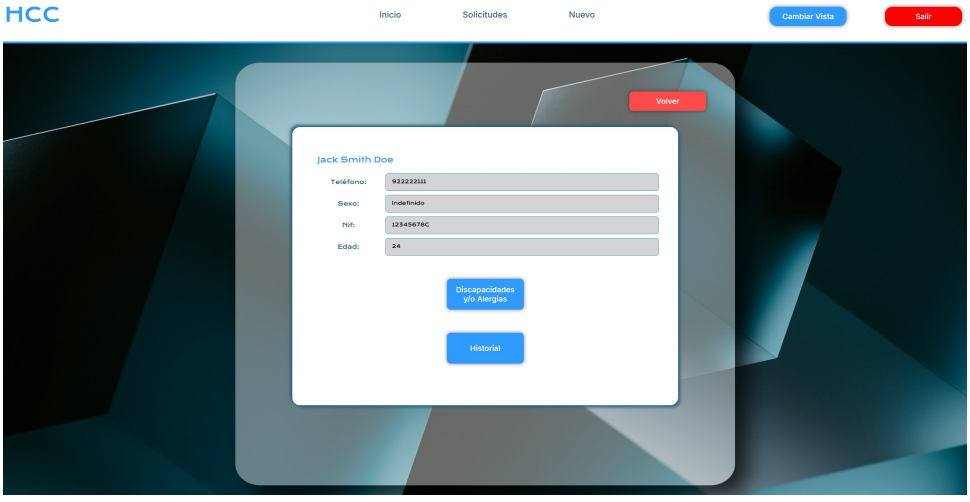

Figura 5.16: Pantalla de visualizar historial

En general, se puede considerar que a la hora de la implementación, no solo se ha igualado el diseño, sino que ha sido mejorado. Esto se ve reforzado con la inclusión de alertas procedentes de una librería llamada Sweet Alerts.

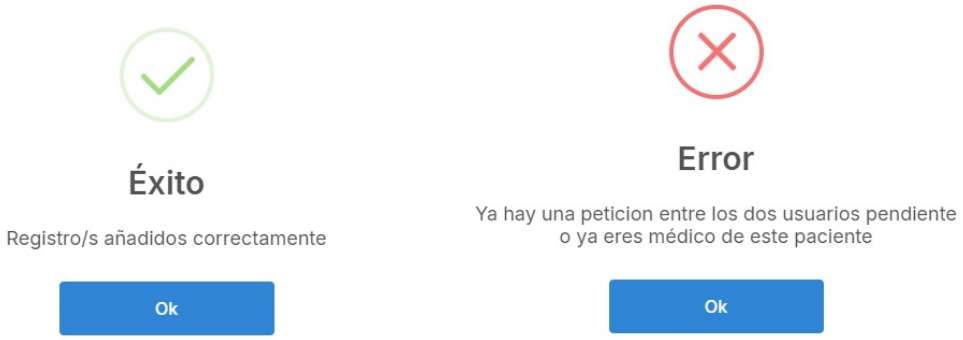

Figura 5.17: Alertas de la aplicación

5.5 Pruebas

Por último, llegamos a la última fase del desarrollo del proyecto. La realización de pruebas es un concepto crucial a la hora de asegurar la calidad del software desarrollado. Puesto que gran parte de la seguridad, aspecto clave en nuestra situación, recae de manera implícita en la red de blockchain, en esta fase tan solo nos quedaría presentar ejemplos de algunos tests implementados para asegurar el correcto funcionamiento de algunos métodos concretos. Esto principalmente tiene como objetivo, mostrar cómo se puede emplear testing en blockchain con las tecnologías que hemos incluido.

Truffle nos permite realizar test sobre nuestros contratos directamente en el lenguaje JavaScript, aunque desde Solidity también se puede realizar. En nuestro caso utilizaremos directamente JS. Para ello, creamos en el directorio /test el archivo UserManager.test.js. Dentro de este archivo y de otro similar para el contrato ReportManager, hemos desarrollado las pruebas necesarias para comprobar el funcionamiento de la aplicación. A continuación mostraremos ejemplos y veremos cómo ejecutar las pruebas.

Al inicio de los tests deberemos declarar los contratos y variables que necesitaremos utilizar para los tests. Dentro del before es donde debemos inicializar los contratos como si de la migración se tratase.

```
1  const UserManager = artifacts.require("UserManager");
2  const ReportManager = artifacts.require("ReportManager");
3
4
5  contract("UserManager", accounts => {
6      let userManager;
7      let reportManager;
8
9
10     const owner = accounts[0];
11     const testUsername = "testuser";
12     const testName = "Test";
13     const testLastname = "User";
14     const testNif = "12345678A";
```

```
    const testSex = "M";
    const testBdate = "1990-01-01";
    const testPassword = "password123";
    const testEmail = "test@example.com";
    const testPhone = 1234567890;


    before(async () => {
        reportManager = await ReportManager.new({ from: owner });

        userManager = await UserManager.new(reportManager.address, { from: owner })
            ;
    });
```

Con esto definido, se puede comenzar a diseñar las pruebas concretas de los métodos:

```
it("should return the invalid user when username does not exist", async () => {
   const result = await  userManager.getUserByUsername("nonexistentuser", { from:
       owner });

 assert.equal(result.id, 0, "El ID deber a ser 0 para un usuario inexistente");
        assert.equal(result.username, "", "El nombre de usuario deber a ser una
            cadena vac a para un usuario inexistente");
});
```

En este método probamos que si tratamos de obtener un usuario con un nombre que no existe nos devolverá un usuario inválido (con id igual a 0 y un nombre de usuario vacío).

Esta es la estructura de un test y a partir de ella podemos extrapolar para realizar los tests que deseemos. Como hemos comentado antes, también hemos realizado tests para el ReportManager.

```
it("should emit ReportCreated event on report creation", async () => {
        const completeName = "Jane Smith";
        const phone = 9876543210;
        const nif = "87654321B";
        const sex = "F";
        const bdate = "1985-05-15";


        const result = await reportManager.createEmptyReport(
            completeName,
            phone,
            nif,
            sex,
            bdate
        );


        const log = result.logs[0];


        assert.equal(log.event, "ReportCreated", "Event ReportCreated should be
            emitted");
        assert.equal(log.args.completeName, completeName, "Complete name in event
            should match");
        assert.equal(log.args.phone.toNumber(), phone, "Phone in event should match
            ");
});
```

Este test se encarga de comprobar que al crear un historial para un nuevo usuario se emita un evento con la información esperada. Por último, para ejecutar los test nos basta con escribir en la terminal el siguiente comando:

```
$ truffle test
```

Al ejecutarse, la consola nos mostrará el resultado de nuestros test, quedando en nuestro caso así:

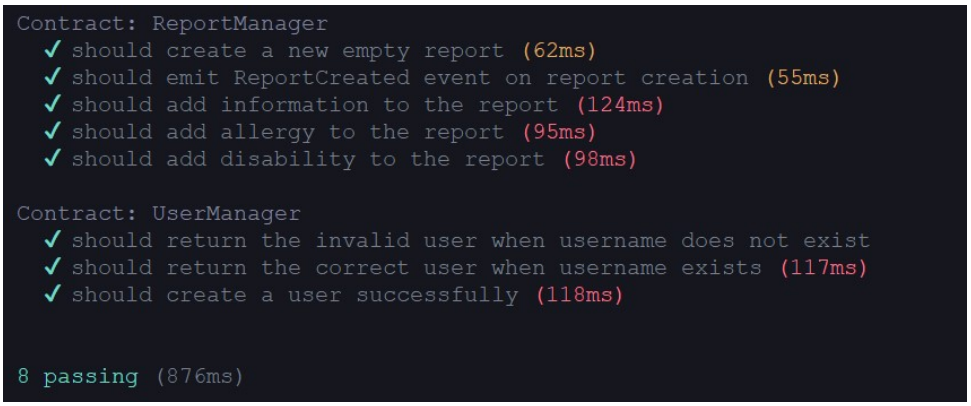

Figura 5.18: Resultados de las pruebas

El resultado nos muestra el nombre que hemos dado a nuestros tests, si han pasado o no y el tiempo de ejecución que le ha llevado a cada uno. Hay que tener en cuenta que el tiempo de ejecución puede variar en base a nuestra máquina y conexión a la red de blockchain. De igual manera, tan solo se muestran 8 tests debido a que nuestro objetivo es probar aquellas funcionalidades más básicas sobre las que trabajan el resto de métodos y funciones. Esto se debe a que nuestro objetivo es sacar un mínimo producto viable para probar que blockchain es capaz de albergar una aplicación de este estilo.

CAPÍTULO 6
Conclusiones

Tras haber terminado el desarrollo de nuestro proyecto, podemos asegurar que los objetivos principales propuestos se han cumplido de manera exitosa, pues el desarrollo de la DApp ha sido posible y hemos logrado explorar las capacidades de la tecnología blockchain en el desarrollo de software. La realidad es que la aplicación implementada cumple a su vez con los diferentes subobjetivos que han ido surgiendo con su desarrollo, adaptándose a todos los requisitos identificados en su fase correspondiente y manteniendo las pautas establecidas en la fase de diseño.

En cuanto a la comparativa de blockchain respecto a otras tecnologías, podemos decir que blockchain a pesar de sus esfuerzos puede no resultar tan efectivo. Esto se debe principalmente a sus limitaciones de eficiencia, donde el tiempo medio por transacción es superado por otros sistemas actuales como bases de datos tradicionales. Es importante destacar que esto no es algo fijo, pues todo depende del contexto en el que se sitúe el proyecto y las necesidades de este. Con esto nos referimos a que una base de datos que no esté perfectamente diseñada puede acabar teniendo una peor eficiencia que una red de blockchain bien diseñada. Esto sumado a que en términos de seguridad, facilita la vida de los desarrolladores gracias a las técnicas que esta tecnología emplea por sí sola. Por eso es importante la figura de un ingeniero informático capaz de aportar unos conocimientos profundos en cada una de las fases del desarrollo de software, pudiendo analizar las necesidades de los proyectos e identificar qué diseño de tecnologías y arquitectura son las mejores para este.

Lo que no podemos negar, es que blockchain es una tecnología más que prometedora, pues los avances que ha tenido en los últimos años nos dan a ver que su futuro en el desarrollo de software es ilusionante. Por ejemplo, podemos ver la gran cantidad de aplicaciones que se están ejecutando ahora mismo con Solidity en la red de Ethereum y darnos cuenta de que este lenguaje ni siquiera ha llegado aún a su versión 1.0.

No obstante, queda remanente un punto a comentar. Un proyecto desplegado en la blockchain de Ethereum, actualmente requiere de su criptomoneda ETH para funcionar. Esto es un asunto que podría resultar problemático para su implementación, pues el precio de esta moneda es variable. Por ello podemos decir, que la mayor desventaja que tiene blockchain hasta ahora es su vinculación con las criptomonedas, un problema difícil de resolver pero que en caso de lograrlo, estaríamos ante una tecnología que nos obligaría a cambiar nuestra visión completamente en el entorno del desarrollo software.

6.1 Trabajo futuro

Debido al constante cambio en las tecnologías, en caso de querer continuar con la implementación de esta DApp resulta más que necesario trabajar en ir adaptando este proyecto con el tiempo a la evolución del software y la sociedad en sí misma. Esto corresponde a la fase de mantenimiento. Un supuesto que resultaría ideal teniendo en cuenta el contexto de nuestro proyecto, sería el desarrollo de una red blockchain propia. Más aún si se tratase de una red particular del estado, con el objetivo de desvincularse completamente de las criptomonedas para crear un servicio público y gratuito de cara a los ciudadanos. Esto requeriría de un estudio más profundo aún de las capacidades de la blockchain para desligarse de estas.

En cuanto a la DApp desarrollada hasta ahora, un camino de evolución pasaría necesariamente por el aumento del número de tests y pruebas con el fin de mejorar más aún la tolerancia a fallos del sistema. Por otro lado, la capacidad de integrarse con los sistemas actuales de manera que, no fuera siquiera necesario el hecho de que un médico introdujera la información correspondiente dentro de la aplicación, y que esta recogiera de manera automática los datos que son introducidos en los historiales clínicos electrónicos con los que contamos hasta ahora. Otra opción, consistiría en ampliar esta aplicación hasta una sustitución total de los sistemas actuales.

La implementación de las mejoras comentadas y un establecimiento claro de los posibles caminos a seguir, otorgarían una gran cantidad de valor al proyecto.

Bibliografía

[1] Alexander Preukschat. *Blockchain: La revolución industrial de internet*.

[2] Yves Michel Leoricher, Frederic Goujon y Dr Bilal Chouli. *Blockchain: de la teoría a la práctica, de la idea a la implementación*.

[3] Historia de Blockchain: https://carballar.com/historia-de-blockchain.

[4] Cómo poner un sello de tiempo a un documento digital: https://link.springer.com/article/10.1007/BF00196791.

[5] *"How to Make a Mint: the Cryptography of Anonymous Electronic Cash"*: https://www.academia.edu/5761579/How_to_Make_a_Mint_The_Cryptography_of_Anonymous_Electronic_Cash.

[6] *"Bit Gold: Towards Trust-Independent Digital Money"*: https://web.archive.org/web/20140406003811/http://szabo.best.vwh.net/bitgold.html.

[7] Bitcoin: https://bitcoin.org/es/.

[8] *"Bitcoin: A Peer-to-Peer Electronic Cash System"*: https://bitcoin.org/files/bitcoin-paper/bitcoin_es.pdf.

[9] Ethereum: https://ethereum.org/es/.

[10] *¿Qué es blockchain?*: https://www.ibm.com/es-es/topics/blockchain.

[11] *¿Qué es blockchain y cómo funciona?*: https://www.finect.com/usuario/vanesamatesanz/articulos/que-blockchain-criptomonedas-guia-facil.

[12] *El problema de los generales bizantinos*-Platzi: https://platzi.com/clases/2443-historia-bitcoin/40366-el-problema-de-los-generales-bizantinos/.

[13] *El problema de los generales bizantinos*-Marvin Soto: https://marvin-soto.medium.com/el-problema-de-los-generales-bizantinos-pgb-e0cb8c4279c2. https://platzi.com/clases/2443-historia-bitcoin/40366-el-problema-de-los-generales-bizantinos/.

[14] *¿Qué es el consenso? Guía para principiantes*: https://crypto.com/university/es/consensus-mechanisms-explained.

[15] *¿Qué es un hash?*: https://academy.bit2me.com/que-es-hash/.

[16] *¿Qué es haschash?*: https://wiki.lemon.me/blockchain/que-es-hashcash/.

[17] *¿Qué es el doble gasto?*: https://uphold.com/es/learn/beginner/the-double-spend-problem.

[18] *Que es el blockchain: definición, ejemplos, tipos y ventajas*: https://www.lisainstitute.com/blogs/blog/que-es-blockchain-tipos-ejemplos-ventajas.

[19] *Qué es una historia clínica y qué debe incluir*: https://www.igaleno.com/blog/que-es-historia-clinica/.

[20] *BOE: Ley 41/2002*: https://www.boe.es/eli/es/l/2002/11/14/41/con.

[21] *La Historia Clínica Electrónica (HCE) en España*: https://clinic-cloud.com/blog/historia-clinica-electronica-hce-espana.

[22] *Documentación de desarrollo en Ethereum*: https://ethereum.org/es/developers/docs/.

[23] *Etherscan*: https://etherscan.io.

[24] Documentación de Solidity en español: https://solidity-es.readthedocs.io/es/latest/.

[25] React: https://es.react.dev.

[26] *¿Qué es react?*: https://www.hostinger.es/tutoriales/que-es-react.

[27] Vite: https://es.vitejs.dev/guide/.

[28] NodeJS: https://nodejs.org/en/.

[29] *¿Qué es node?*: https://openwebinars.net/blog/que-es-nodejs/.

[30] Truffle Suite: https://archive.trufflesuite.com.

[31] *¿Qué es Truffle Suite?*: https://keepcoding.io/blog/que-es-truffle/.

[32] Ganache: https://archive.trufflesuite.com/docs/ganache/.

[33] Visual Studio Code: https://code.visualstudio.com.

[34] Git: https://git-scm.com.

[35] *El modelo en cascada: desarrollo secuencial de software*: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/.

[36] Figma: https://www.figma.com/es-es/.

[37] *Inyección de Dependencia y su utilidad*: https://www.arquitecturajava.com/el-patron-de-inyeccion-de-dependencia/.

[38] *Ventajas y desventajas de la tecnología blockchain*: https://www.esic.edu/rethink/marketing-y-comunicacion/ventajas-desventajas-tecnologia-blockchain-c.

[39] *Crea tu primera Aplicación Descentralizada - React, Solidity, Truffle*: https://www.youtube.com/watch?v=PaAFI-bMLEg.

[40] Laia Soler Izquierdo: *Desarrollo de una Dapp basada en Ethereum y React*: https://upcommons.upc.edu/bitstream/handle/2117/134051/DegreeThesis_LaiaSoler.pdf.

[41] Lucía Muñoz Martínez: *Desarrollo de un Servicio Basado en Blockchain para la Venta de Entradas*: https://oa.upm.es/71281/1/TFG_LUCIA_MUNOZ_MARTINEZ.pdf.
1

[1] Enlaces revisados por última vez el 1 de septiembre de 2024.

APÉNDICE A
Objetivos de desarrollo sostenible

Objetivos de desarrollo sostenible	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				✓
ODS 2. Hambre cero.				✓
ODS 3. Salud y bienestar.	✓			
ODS 4. Educación de calidad.				✓
ODS 5. Igualdad de género.				✓
ODS 6. Agua limpia y saneamiento.				✓
ODS 7. Energía asequible y no contaminante.				✓
ODS 8. Trabajo decente y crecimiento económico.				✓
ODS 9. Industria, innovación e infraestructuras.	✓			
ODS 10. Reducción de las desigualdades.		✓		
ODS 11. Ciudades y comunidades sostenibles.				✓
ODS 12. Producción y consumo responsables.				✓
ODS 13. Acción por el clima.				
ODS 14. Vida submarina.				✓
ODS 15. Vida de ecosistemas terrestres.				✓

Tabla A.1: Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS)

Resulta evidente que este proyecto está altamente relacionado con el ODS de Salud y bienestar. Esto se debe a que uno de los principales fines de este, consiste en la gestión de la información médica, perfeccionando la forma en la que se distribuye actualmente. El proyecto aporta una considerable mejora de interoperabilidad entre los diferentes centros sanitarios de nuestro país.

Definitivamente, garantizar el acceso a los servicios sanitarios es una de las prioridades de este ODS y precisamente en eso se centra nuestra aplicación, proporcionando acceso a la información médica personal de una manera descentralizada a cualquier usuario que la necesite, pudiendo conceder acceso a ella a los médicos y personal sanitario. De esta manera evitamos parte de los más que conocidos problemas de atención sanitaria en nuestro país. Un médico debe de ser capaz de acceder a los datos de un paciente si va a atenderlo, independientemente de su comunidad de origen, más aún si se trata de un caso de urgencia sanitaria.

Con este proyecto solucionamos problemas tanto del personal sanitario, como de los pacientes que requieren de su atención. Esto resulta en un más eficiente servicio médico en España y por tanto en uno mejor. Todos tenemos derecho a conocer nuestra información médica y a acceder a ella en cualquier momento, sin necesidad de esperas y trámites que nos acaban haciendo desistir de ello.

En parte el proyecto también se relaciona con el ODS de reducción de las desigualdades, pues favorece a que todos tengamos el mismo derecho de acceder a nuestra información. También, evitando los problemas de interoperabilidad, proporcionamos la capacidad de atender a cualquier paciente sin la necesidad de que estos sean tratados prácticamente a ciegas por no poder acceder a sus datos. Claramente esto, es una reducción de desigualdades, pues la capacidad de recibir atención médica en cualquier lugar en el que nos encontremos, no nos pone en una situación de desventaja respecto a otras personas que siempre tengan la capacidad de acceder al mismo centro sanitario.

Por último, es necesario destacar el innegable vínculo que alberga nuestro proyecto con el ODS de Industria, innovación e infraestructura. Actualmente, blockchain es una de las tecnologías más innovadoras existentes y, a pesar de que hay gran cantidad de aplicaciones que la implementan, nunca se ha utilizado en un proyecto de esta envergadura. Ya hemos hablado previamente de lo prometedora que resulta la tecnología blockchain. En un mundo tan digitalizado, nuestros esfuerzos por innovar deben dirigirse en este camino, el de construir nuevos desarrollos en tecnologías que puedan llegar a mejorar las diferentes alternativas con las que contamos hoy en día.

No solo estamos hablando de innovar en el sentido tecnológico, pues en el ámbito de la salud no existe nada parecido hasta ahora. Además, esta aplicación mejora la infraestructura sanitaria actual de nuestro país, que cuenta con numerosos problemas para gestionar la atención médica a los pacientes debido a la mala interoperabilidad existente. Estamos hablando de algo que va más allá de lo público y de lo privado, de algo que no entiende de límites ni fronteras a la hora de permitir que la sanidad llegue a todos los rincones de nuestro país y, quién sabe si con el debido crecimiento, al mundo entero.