



# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Implementación de un sistema básico de Blockchain y su  
aplicación en la trazabilidad de datos médicos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Montoliu Rico, Pablo

Tutor/a: Blanquer Espert, Ignacio

Director/a Experimental: ALIC, ANDREI STEFAN

CURSO ACADÉMICO: 2021/2022

## **Agradecimientos**

A mis dos tutores, Nacho y Andy, por todo el tiempo y la atención que me han dedicado. Agradecerles también la confianza depositada en mí y el haberme dado la oportunidad de hacer este trabajo con el que he disfrutado tanto.

## Resumen

---

En este trabajo se expone la implementación de un software de almacenamiento de datos de forma segura mediante la creación de una cadena *blockchain* así como una explicación detallada de sus características básicas y de los componentes necesarios para crear una cadena de bloques desde cero. El objetivo de la creación de esta cadena es proporcionar una solución al almacenamiento y la trazabilidad de datos privados en un entorno clínico.

**Palabras clave:** libro mayor distribuido, cadena de bloques, blockchain híbrida, prueba de trabajo, red entre iguales, trazabilidad.

## Abstract

---

This paper exposes the implementation of a secure data storage software by creating a Blockchain chain, as well as a detailed explanation of its essential characteristics and the necessary components to create a chain of blocks from scratch. The objective of creating this chain is to provide a solution to the storage and traceability of private data in a clinical environment.

**Keywords :** distributed ledger technology, hybrid blockchain, proof of work, peer-to-peer network, traceability.



# Tabla de contenidos

<b>Introducción</b>	<b>8</b>
1.1. Motivación	8
1.2. Objetivos	9
1.3. Metodología	9
1.4. Estructura del documento	10
1.5. Convenciones	11
<b>Estado del arte</b>	<b>12</b>
2.1. Tecnología Blockchain	12
2.2. Herramientas para la construcción de Libros de Registros Distribuidos Blockchain	15
2.2.1 Función Hash y SHA-256	15
2.2.2 Algoritmos de consenso	16
2.2.3 Regla de la cadena más larga	16
2.2.4 Proof of Work (PoW)	17
2.2.5 Alternativas actuales a Proof of Work	17
2.3. Vulnerabilidades de una red Blockchain	18
2.3.1 Posibles ataques	19
2.4. Aplicaciones Blockchain	21
2.5. Almacenamiento de historias clínicas digitales.	23
<b>Análisis del problema</b>	<b>24</b>
3.1. Solución propuesta	24
3.2. Especificación de requisitos	24
3.2.1 Requisitos funcionales	24
3.2.2 Requisitos no funcionales	26
<b>Diseño de la solución</b>	<b>31</b>
4.1. Arquitectura del Sistema	31
4.2. Diseño Detallado	33
4.3. Endpoints y funcionalidades	36
4.3.1 Endpoints para la construcción de la red	36
4.3.2 Endpoints para la gestión de datos	40
4.3.3 Endpoints para la gestión de la aplicación	44
4.4. Tecnología Utilizada	50
<b>Desarrollo de la solución propuesta</b>	<b>52</b>
5.1 Configuración de contenedores Docker	52
5.2 Configuración de servidores MongoDB	52

5.3 Partes relevantes del código	54
5.3.1 blockchain.js	54
5.3.2 model.js	57
5.3.3 netNode.js	58
5.3.4 index.js	63
5.4 Codificación de imágenes	64
<b>Pruebas</b>	<b>66</b>
6.1. Pruebas de uso	66
6.1.1 Puesta en marcha de servidores y nodos	66
6.1.2 Pruebas de ejecución	69
6.2. Pruebas de rendimiento	73
6.3. Discusión de los resultados	75
<b>Conclusiones</b>	<b>77</b>
7.1 Relación del trabajo con los estudios cursados	78
<b>Trabajos futuros</b>	<b>80</b>
<b>Referencias</b>	<b>82</b>
<b>Apéndice 1. Relación con los Objetivos de Desarrollo Sostenible (ODS)</b>	<b>84</b>
Objetivo 3: Garantizar una vida sana y promover el bienestar para todos en todas las edades	84
Objetivo 16: Promover sociedades justas, pacíficas e inclusivas	85

# 1. Introducción

---

A día de hoy, el uso de los datos médicos obtenidos en la práctica clínica supone una fuente de conocimiento esencial para el análisis de cualquier enfermedad o de cualquier suceso que pueda tener implicación directa en la salud de las personas. Estos datos ayudan cada día a profesionales del sector sanitario a crear modelos predictivos que les asistan tanto en el diagnóstico de enfermedades como en el tratamiento, la terapia y el seguimiento de estas.

Sin embargo, los datos recopilados requieren de unos procesos de mejora, análisis y selección que hacen que estos adquieran un mayor valor informativo. Teniendo en cuenta esto y el hecho de las implicaciones tanto éticas como legales que conlleva el uso de datos clínicos de pacientes, hace que la utilización de estos datos requiera de una trazabilidad y de un almacenamiento seguro que permita establecer la cadena de valor y de responsabilidad en su acceso.

Por tanto, este trabajo se centra en la realización de un sistema de almacenamiento seguro empleando una cadena *blockchain* con la que se otorgará transparencia, seguridad y trazabilidad al uso y almacenamiento de datos médicos.

## 1.1. Motivación

En ocasiones nos referimos a la sociedad actual como la sociedad de la información, y es que si hay algo que juega un papel relevante en esta sociedad avanzada en la que vivimos son los datos. Con ellos podemos desde tratar de mejorar el crecimiento económico de una nación hasta tratar de prever cuándo y dónde ocurrirá la siguiente pandemia.

Los datos de miles de millones de personas son adquiridos y tratados a diario por empresas de todo el mundo con distintos fines, muchas veces con el completo desconocimiento de la persona que facilitó dichos datos. Estas razones hacen que a menudo surjan preguntas como:

¿Son tratados de manera responsable y eficiente?

¿Son almacenados de manera segura?

¿Son tratados sólo por aquellas entidades a las que dimos nuestro consentimiento para su tratamiento?

Este Trabajo de Fin de Grado está motivado por el hecho de responder a esas preguntas de forma afirmativa y conseguir mediante el uso de la tecnología *blockchain*, anonimato, eficiencia, seguridad y transparencia en la forma en la que se usan los

datos de las personas, especialmente en un ámbito tan crítico como puede ser en ámbito clínico.

A su vez este trabajo ha sido motivado por el interés en saber cómo funcionan a un nivel interno las tan en boga redes *blockchain* que acaparan miles y miles de artículos de actualidad.

## 1.2. Objetivos

El objetivo principal de esta solución era lograr la implementación de una cadena blockchain con la que poder interactuar y almacenar datos desde distintos nodos de manera demostrativa. La creación de esta cadena estaba enfocada a la mejora en la seguridad y en la trazabilidad del almacenamiento de los historiales médicos digitales gracias a la integración de una cadena de bloques en el sistema de procesamiento de datos sanitario que a día de hoy sigue siendo dependiente, bien de organizaciones centralizadas y dirigidas por el estado en el caso de algunas comunidades o bien de empresas privadas que se encargan del procesamiento de estos datos.

Para ello, esta implementación está diseñada para permitir la conexión de múltiples nodos entre sí, almacenar datos de forma segura en una cadena de bloques y poder obtener estos datos de vuelta cuando sean requeridos.

Con esta aplicación se pretende lograr la seguridad en el almacenamiento y acceso de los datos médicos de los pacientes así como permitir el acceso de forma segura y transparente a estos.

## 1.3. Metodología

Para tratar de lograr una solución que cumpliera de la manera más acertada posible los objetivos señalados en el apartado anterior se ha seguido una metodología de trabajo clásica que ha constado de los siguientes puntos o apartados:

1. Definición de la funcionalidad del sistema
2. Análisis en profundidad de la tecnología blockchain así como de sus aplicaciones derivadas.
3. Estudio de soluciones similares a la planteada.
4. Estudio de las tecnologías disponibles para la implementación del proyecto.
5. Selección y preparación de las tecnologías a utilizar en el desarrollo del proyecto.
6. Construcción de una cadena de bloques aislada.
7. Implementación de las funcionalidades de la cadena.
8. Construcción de una red para interactuar con la cadena.

9. Integración de mongodb para almacenar la cadena de forma permanente sin necesidad de mantener un nodo conectado.
10. Pruebas y validación del sistema.

## 1.4. Estructura del documento

El siguiente documento está dividido en los siguientes apartados:

### Estado del arte

En este apartado el lector podrá encontrar una breve introducción a la tecnología blockchain y a sus características principales acompañada del análisis de los elementos que la componen. Así mismo, también encontrará un análisis de otras alternativas similares a la propuesta y una breve introducción al estado del almacenamiento de historias clínicas digitales en la actualidad.

### Análisis del problema

En este apartado se lleva a cabo un análisis detallado del problema y se establecen los requisitos que deberá cumplir la aplicación para satisfacer las necesidades observadas. Además de esto, también se puede encontrar un texto explicativo en el que se explica de forma detallada la solución propuesta.

### Diseño de la solución

A lo largo de este apartado, el lector encontrará aspectos como la arquitectura de la solución, de qué componentes consta esta y una explicación más detallada del diseño de la aplicación. Aparte de estos apartados, también se encuentra en este apartado un resumen de todos los endpoints y funcionalidades presentes en la aplicación, desde los que nos permiten construir la red de nodos hasta aquellos que nos permiten la actualización de la cadena.

Por último se exponen todas las tecnologías que han sido empleadas en el desarrollo de esta solución.

### Desarrollo de la solución propuesta

En el apartado de desarrollo de la solución propuesta, se encuentra un resumen de la configuración necesaria para el correcto funcionamiento del sistema. Además de esta configuración, el lector puede encontrar los fragmentos de código que se han considerado de mayor relevancia acompañados por distintas aclaraciones sobre su relevancia y su estructura.

### Pruebas

A lo largo de esta sección se presentan las pruebas que se han realizado para verificar la validez de nuestra solución. También se exponen los resultados de distintas pruebas de eficacia.

### Conclusiones

Este apartado contiene una breve redacción analizando el proceso de aprendizaje a lo largo de la construcción de la aplicación. Se expone que nuevos

conceptos o habilidades se han aprendido y qué relación guardan los conocimientos aplicados en este trabajo, con todo lo aprendido durante la carrera.

También se puede encontrar un breve resumen con los problemas acaecidos durante el desarrollo de la solución y cómo se logró solucionarlos.

### **Trabajos futuros**

Este último apartado contiene aquellas mejoras que no han formado parte de la aplicación por falta de tiempo. Estas mejoras están pensadas para mejorar la eficiencia y la funcionalidad del sistema y podrían ser de especial interés para aquellos que deseen desarrollar una hipotética solución basada en esta aplicación.

### **1.5. Convenciones**

- Las palabras que se hallen escritas en un idioma extranjero aparecerán en cursiva
- Las citas textuales externas a la obra aparecerán entrecomilladas.
- El código fuente aparece escrito en letra courier new y con un fondo negro.

## 2. Estado del arte

---

En esta sección se analiza y trata toda tecnología cuya comprensión resulte necesaria para poder realizar un correcto análisis de este proyecto. El lector encontrará un breve resumen de la evolución de las tecnologías implicadas en el proyecto así como una introducción a todos los conceptos y términos de los que se hablará durante el desarrollo de la solución. También se exponen aspectos como cuáles pueden ser las posibles vulnerabilidades de una cadena de bloques o como la situación del tratamiento de datos médicos en nuestro país.

### 2.1. Tecnología Blockchain

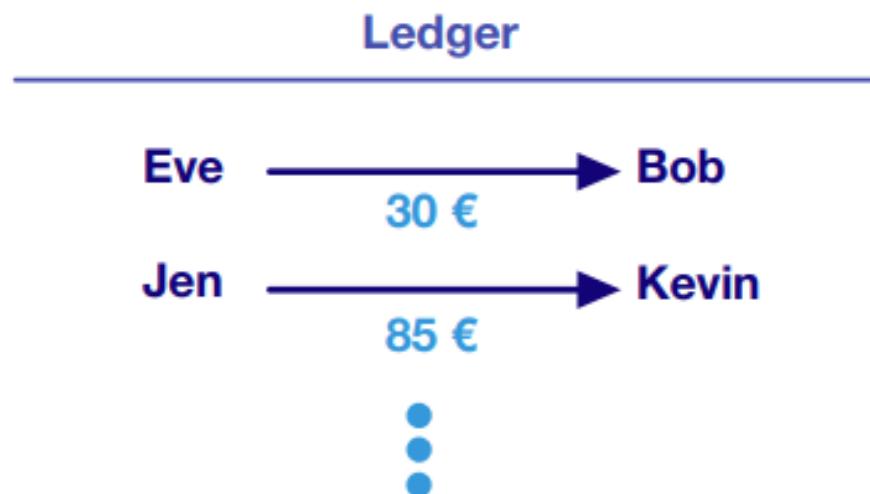
Si hacemos un repaso de la evolución de esta tecnología a lo largo de los años, quizás el momento más importante para el surgimiento de la tecnología *blockchain* sea la aparición de Bitcoin en el 2008. Aunque la tecnología ya existía antes de la aparición de Bitcoin, este fue el primer ejemplo de *blockchain* tal y como lo conocemos hoy. Bitcoin sentó las bases de la tecnología de manera práctica, convirtiéndose en la primera criptomoneda establecida y la primera aplicación sustancial de *blockchain*.

Otro momento a destacar en la evolución de la tecnología *blockchain* llegaría con el lanzamiento de la plataforma Ethereum y la llegada de los contratos inteligentes y las aplicaciones descentralizadas o DApps. Aunque algunas de estas aplicaciones quedan lejos del ámbito de estudio de este trabajo, parten de las mismas bases, por ello para comprenderlas mejor y poder manejar estos términos comenzaremos explicando las bases.

- **¿Qué es una *blockchain*?**

Antes que nada, en esta sección vamos a realizar una breve explicación de qué es una *blockchain*. Dicho de manera simple, una *blockchain* no es más que un ledger, libro de registros o libro mayor, inmutable y distribuido. Por ahora estas palabras pueden sonar algo complejas, pero si las descomponemos un poco puede resultar más sencillo entenderlas. Primero, un ledger o un libro de registros, es simplemente una colección de transacciones, pero tratemos de entenderlo con un ejemplo.

Eve le envía a Bob 30€ y Kevin le envía a Jen 80€. El ledger o libro mayor sería simplemente un documento en el que se anotarían esas transacciones.



**Figura 1.0 : Ilustración Ledger**

Ahora, ¿Qué significa que una *blockchain* sea inmutable?

Esto significa que no puede ser editada, nunca. Esto implica que, cuando una transacción es almacenada en la cadena, no puede deshacerse. Tampoco se pueden editar los valores almacenados, cómo los datos que transportaba, el emisor o el receptor. Una vez se ha realizado la transacción, no hay nada de ella que se pueda editar porqué es inmutable.

A día de hoy, vemos muchas plataformas, aplicaciones y redes centralizadas.

Tomemos Instagram, por ejemplo. Cada persona que usa Instagram, tiene que confiar la protección y el correcto procesamiento de sus datos a la compañía.

Comparado con esto, *blockchain* es diferente. La tecnología *blockchain* no está centralizada como Instagram, Google, o muchas otras entidades. En lugar de esto, *blockchain* utiliza una red descentralizada, lo que significa que cualquier red blockchain no está controlada por una única entidad o empresa, sino que es controlada por sus usuarios. Las *blockchain*, como por ejemplo Bitcoin, son soportadas por todos los miles de usuarios que participan en la red. De esta forma, todos nuestros datos, o el ledger en este caso, no están a merced de una única compañía o entidad. Este es uno de los principales beneficios de una *blockchain* dado que al estar distribuida, no tenemos porqué confiar nuestros datos a una única organización. En lugar de esto, nuestros datos están almacenados en toda la red de nodos que forman las distintas personas que componen la red y que actúan de manera independiente.

Cada computador que contribuye a la formación de la red *blockchain* recibe el nombre de nodo, y cada uno de estos nodos tiene la misma copia exacta del ledger o libro mayor. Por lo tanto, los datos del ledger están almacenados y sincronizados a lo largo y ancho de toda la red.

Entonces, para resumir y volviendo a la frase del principio, una *blockchain* es un ledger inmutable y distribuido. Es decir, es un libro mayor o libro de registros donde se almacenarán todas las transacciones llevadas a cabo de manera terminal y sin posibilidad de edición posterior y que se distribuye a través de una red gestionada por personas independientes. [31]

#### - **Tipos de Blockchain**

Una vez explicado el concepto de *blockchain*, cabe destacar que en la actualidad coexisten distintos enfoques dentro de lo que se podría englobar como tecnologías *blockchain*:

- **Blockchains públicas**: este tipo de cadenas fueron las primeras en aparecer. Son las únicas cadenas que mantienen los tres principios fundamentales de la tecnología (pseud-anonimato, no dependencia en terceros y inmutabilidad) por ello son las únicas que deberían mantener el nombre de *blockchains*. Estas cadenas están caracterizadas por ofrecer libre acceso tanto a sus datos como al software de la cadena, por lo que cualquier persona puede copiar el código o tratar de mejorarlo. Entre sus principales características se encuentran el funcionamiento completamente transparente y abierto y la completa disponibilidad de los datos almacenados. Además estas cadenas no están reguladas ni regidas por ninguna entidad centralizada.

Ejemplos de esta clase de cadenas son Bitcoin, Ethereum y Monero.

- **Blockchains privadas**: estas cadenas tienen un enfoque totalmente distinto al planteado en un principio por Bitcoin. En este tipo de cadenas, el poder reside sobre un nodo o una autoridad central que se encarga de distribuir los distintos privilegios sobre el resto de usuarios.

Esto implica que la unidad central sea la única que pueda dar o revocar el permiso de acceso a la red a cualquier usuario. Otra característica a destacar es que el acceso al libro de transacciones es privado.

Ejemplos de esta clase de cadenas son Corda, Quorum y Ripple.

- **Blockchains híbridas**: este tipo de cadenas fusiona componentes existentes en cadenas de bloques públicas con componentes de cadenas de bloques privadas, tratando de utilizar las ventajas y las mejores partes de ambos mundos.

Aunque una entidad privada pueda ser dueña de la cadena de bloques híbrida, esta no puede cambiar transacciones. Este tipo de cadenas permiten a las organizaciones establecer un sistema privado basado en permisos junto a un sistema público sin permisos, lo que les permite

administrar quién puede acceder a datos particulares almacenados en la cadena de bloques y qué datos particulares almacenados en la cadena de datos se harán públicos.

Cuando un usuario se une a una cadena de bloques híbrida, tiene acceso completo a la red. La identidad del usuario estará asegurada y protegida frente al resto de usuarios.

Una de las principales ventajas de este tipo de cadenas es que los piratas informáticos externos no tendrían posibilidad de intentar un ataque de tipo 51% ya que la cadena opera dentro de un ecosistema cerrado, sólo perteneciendo a la organización podrían tratar de realizar un ataque semejante.

Por último señalar que en nuestro caso, sería una cadena de tipo híbrida ya que sólo aquellas personas con la correspondiente autorización podrían tener acceso a la cadena. [22]

## **2.2. Herramientas para la construcción de Libros de Registros Distribuidos Blockchain**

En primera instancia, como se ha definido en el apartado anterior, hay que lograr que nuestro Libro de registros sea inmutable, para esto necesitaremos que nuestros bloques tengan una determinada estructura que nos permita enlazarlos unos con otros.

Una parte fundamental de esta estructura serían los atributos que contienen el hash del bloque actual y el hash del bloque anterior, ya que estos son los que de alguna manera enlazan los bloques formando una cadena y logran gran parte de la inmutabilidad señalada de la *blockchain*.

A continuación se explican de forma detallada las herramientas y tecnologías empleadas en la construcción de nuestro libro de registros y que permiten enlazar unos bloques con otros.

### **2.2.1 Función Hash y SHA-256**

Una función hash o función resumen es una función criptográfica que transforma uno o varios elementos de entrada en una serie de caracteres de salida, con una longitud fija o variable, dependiendo del algoritmo hash que estemos utilizando. Una de las características más relevantes de las funciones hash es que podemos generar una salida a partir de unos datos de entrada, pero es imposible tratar de volver a obtener esos datos de entrada a partir de la salida. En este caso empleamos un algoritmo hash con longitud de salida fija (SHA-256) que devolverá una salida de longitud fija independientemente del tamaño de los datos de entrada. [17]

Los hashes criptográficos son usados normalmente para proteger las contraseñas y no guardarlas en texto claro en una base de datos, en nuestro caso lo utilizaremos para generar un valor de salida a partir de los datos del bloque, que nos servirá para poder hacer uso del algoritmo Proof of Work, que tratará de generar un hash que comience por una cantidad de 0's prefijada que nos servirá como identificador del bloque, esto lo conseguirá llamando a la función de hashing (hashBlock) y editando un único parámetro de los que componen el bloque, el **nonce**, en este caso particular, como es usual encontrar en la mayoría de las cadenas, es simplemente una variable que incrementamos en una unidad cada vez que tenemos que llamar a la función hashing para recalcular el valor. Dependiendo del número de 0's por el que debe comenzar el hash habrá que hacer un mayor o un menor número de iteraciones, a mayor número de 0's, mayor número de iteraciones, i.e. mayor nonce.

En este caso particular se ha optado por la función de hashing SHA-256. El algoritmo SHA-256 es uno de los más utilizados hoy en día dado que ofrece un muy buen equilibrio entre la seguridad que garantiza y el coste computacional necesario para aplicarlo. Las siglas SHA hacen referencia a Secure Hash Algorithm o Algoritmo de hash seguro y 256 hace referencia a la codificación de 256 bits que emplea (32 bytes) que devuelve una cadena de 64 letras. [\[18\]](#)

Esto significa que los valores de salida del hash no son únicos porque hay un número finito de valores de salida (256 bits), por un número infinito de entradas, lo que significa que algunas entradas generarán el mismo resultado. Cuando esto sucede se dice que ha sucedido una **colisión** pero esto es prácticamente imposible ya que después de todo hay  $2^{256}$  salidas posibles.

### 2.2.2 Algoritmos de consenso

Un algoritmo de consenso es un mecanismo mediante el cual se permite a los usuarios o a las computadoras ponerse de acuerdo sobre una configuración distribuida. Este, debe asegurar que todos los participantes del sistema puedan ponerse de acuerdo sobre una única fuente de verdad, incluso si algunos de estos participantes fallan. Es decir, el sistema debe ser tolerante a fallos bizantinos. Se dice que un sistema es tolerante a fallos bizantinos (TFB o BFT) cuando éste es capaz de resistir la clase de fallos derivados del problema de los generales bizantinos. Esto significa que un sistema BFT es capaz de continuar operativo incluso si alguno de los nodos falla o actúa de manera maliciosa. [\[13\]](#)[\[14\]](#)

### 2.2.3 Regla de la cadena más larga

En caso de fallo en algún nodo o en el caso de que alguno de los nodos estuviera actuando de forma maliciosa, el algoritmo de consenso nos daría una manera de comparar un nodo con todos los otros nodos dentro de la red de manera que podríamos confirmar que tenemos los datos correctos en ese nodo. Hay distintos algoritmos de consenso aplicados en distintas redes, pero para nuestra red vamos a utilizar *la regla de la cadena más larga*. Básicamente, *la regla de la cadena más larga* analiza un nodo y la copia de la cadena en ese nodo, comparando la longitud de la cadena en un nodo con la longitud de la cadena en otros nodos. Durante esta

comparación, si se encuentra una cadena que tiene mayor longitud que la cadena que está presente en el nodo seleccionado, el algoritmo reemplazará la cadena que está en el nodo seleccionado por la cadena más larga en la red.

La teoría detrás de esto es que debemos confiar en la cadena más larga para mantener los datos correctos, ya que supuso más trabajo y poder computacional crear esa cadena que las otras más cortas. La cadena más larga tiene más bloques y cada uno de esos bloques fue minado usando Proof of Work. Por esta razón, podemos asumir que la mayor parte de la red ha contribuido a la creación de la cadena más larga y por tanto a no ser que se estuviera dando el caso de un ataque del 51%, la cadena más larga es la cadena a confiar.

#### 2.2.4 Proof of Work (PoW)

La prueba de trabajo o proof of work es una prueba criptográfica en la que tal y como su nombre indica, un nodo debe llevar a cabo un esfuerzo o un trabajo para poder registrar un nodo en la red. Esto consigue evitar ciertos comportamientos indeseados en la red y es la pieza fundamental para evitar ataques del tipo Sybil [-Vulnerabilidades de una red Blockchain-Posibles ataques-Sybil attack]. Ya que el coste para tratar de engañar al resto de la red, supondría un coste energético inabarcable para el atacante.

Hay distintas formas de conseguir esta carga extra de trabajo, una de ellas consiste en conseguir un hash para el bloque que comience por un valor concreto, en nuestro caso por determinada cantidad de 0's (cuatro en el ejemplo). [\[15\]](#)

La Prueba de Trabajo, funciona de una forma bastante sencilla. De hecho, el proceso que se lleva a cabo, se puede dividir a grandes rasgos en las siguientes etapas:

1. **Etapa 1:** El cliente o **nodo** envía una solicitud para añadir un bloque. En este punto, la red le asigna una tarea computacionalmente costosa. Esta tarea puede ser resuelta con el fin de recibir un incentivo económico, que en nuestro caso ha sido omitido pues es usada con el fin de almacenar información, no con fines económicos.
2. **Etapa 2:** Comienza la resolución del acertijo. Esto conlleva el uso potencia de computación hasta que se resuelve el puzzle criptográfico. Este proceso es el que recibe el nombre de **minería**.
3. **Etapa 3:** Una vez resuelta la tarea computacional, el cliente comparte esta con la red para su verificación. En este punto, se verifica rápidamente que la tarea cumpla con los requisitos exigidos. Si lo hace, se añade el bloque a la cadena. En caso contrario, se rechaza la adhesión y la solución presentada del problema.
4. **Etapa 4:** Con la confirmación que la tarea ha sido cumplida, el nodo añade de forma segura el bloque a la cadena.

#### 2.2.5 Alternativas actuales a Proof of Work

Como no entran directamente dentro del ámbito de este trabajo, se realiza una mención superficial del resto de alternativas existentes como son:

- Practical Byzantine Fault Tolerance (PBFT),
- Proof of Burn (PoB),
- Proof of Capacity,
- Proof of Elapsed Time (PoET),
- Proof of Stake (PoS). [\[16\]](#)

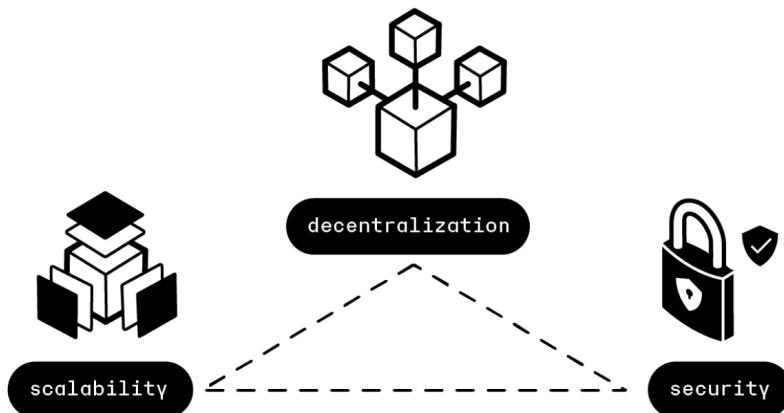
Cabe señalar que al tratarse de un sistema no monetario, prácticamente la mitad de estas opciones queda descartada, a excepción de dos: Proof of Capacity, donde se demuestra la confiabilidad al sistema cediendo espacio de tu disco duro en lugar de carga de trabajo (cuanto más espacio cedas más posibilidades tienes de minar el bloque); y Proof of Elapsed Time, el cual es señalado como uno de los algoritmos de consenso más justos y es ampliamente utilizado en redes blockchain autorizadas. En este algoritmo, cada validador en la red tiene una oportunidad justa de crear su propio bloque. Todos los nodos lo hacen esperando una cantidad aleatoria de tiempo, agregando una prueba de su espera en el bloque. Los bloques creados se transmiten a la red para su revisión. El ganador es el validador que tiene menos valor en el temporizador en la parte de prueba. El bloque del nodo validador ganador se añade a la cadena de bloques. Hay comprobaciones adicionales en el algoritmo para evitar que los nodos siempre ganen las elecciones, evitando que los nodos generen un valor de temporizador más bajo.

### 2.3. Vulnerabilidades de una red Blockchain

Desde que apareció Bitcoin en 2009, la tecnología *blockchain* se ha postulado como una de las tecnologías con potencial suficiente para cambiar el mundo tal y como lo conocemos. Este potencial es tal, que aún hoy, 13 años más tarde se siguen descubriendo frecuentemente nuevas aplicaciones y nuevos campos de aplicación para la tecnología.

La tecnología *blockchain*, por otra parte, aún se encuentra en sus primeras etapas de desarrollo y debe superar una serie de obstáculos antes de ser ampliamente adoptada e integrada en nuestra sociedad. Las redes *blockchain* descentralizadas utilizan tecnología avanzada que todavía está por desarrollar y comprender completamente. Sabemos cómo deberían funcionar tales redes y a qué propósito deberían servir conceptualmente. Sin embargo, cuando se trata de poner la teoría en práctica, los desarrolladores luchan por crear *blockchains* que tengan las tres propiedades principales recomendadas. La escalabilidad, la descentralización y la seguridad deben considerarse características esenciales de la tecnología *blockchain*. Sin embargo, con los principios que entendemos actualmente, retener todos los componentes es increíblemente desafiante.

De estas tres propiedades fundamentales, surge el principal dilema respecto a este tipo de redes y que todos los expertos en el mundo de la tecnología *blockchain* se están esforzando por resolver.[\[1\]](#)



**Figura 2.0 : Ilustración Trilema**

El trilema de blockchain hace referencia a la imposibilidad de lograr una red con estos tres elementos (seguridad, descentralización y escalabilidad) equilibrados, y la necesidad de sacrificar algunos aspectos de estos para lograr el correcto funcionamiento de la red. Como la característica principal de una *blockchain* es la seguridad, la mayoría de iniciativas enfatizan en la seguridad y en la descentralización, por lo que cadenas de host como Bitcoin o Ethereum sufren grandes carencias de escalabilidad que impiden que las criptomonedas sean alternativas de pago realmente viables a escala internacional. Por ejemplo la red Bitcoin, en el momento de escribir este trabajo y sin tener en cuenta posibles alternativas externas a la red como podría ser The Lightning Network [12] es solo capaz de procesar siendo optimistas hasta siete transacciones por segundo, ya que el tamaño medio de transacción es de 0.5kb y el tamaño máximo de bloque es de 1024 kb (1mb), este nuevo bloque se añade cada 10 minutos, por lo que nos deja los siguientes datos:

$$1024 / 0.5 = 2048 \text{ transacciones por bloque.}$$

$$10 * 60 = 600 \text{ segundos}$$

$$2048 / 600 = 3.41 \text{ tps (transacciones por segundo)}$$

Por lo que el resultado real de tps medio sería de 3.5, bastante inferior a los 7 que se suelen indicar e increíblemente lejos del ratio necesario para poder llegar a ser un sistema de pago virtual como por ejemplo los implementados por Visa y Mastercard, capaces de soportar hasta 24.000 tps y 5.000 tps respectivamente. [11]

Este problema de escalabilidad viene dado principalmente por el intento de conseguir una red lo más resistente posible ante ataques como el Sybil Attack que en este trabajo, al igual que en Bitcoin, se consigue evitar gracias a la Prueba de Trabajo o Proof of Work (PoW).

### 2.3.1 Posibles ataques

- **Sybil Attack**

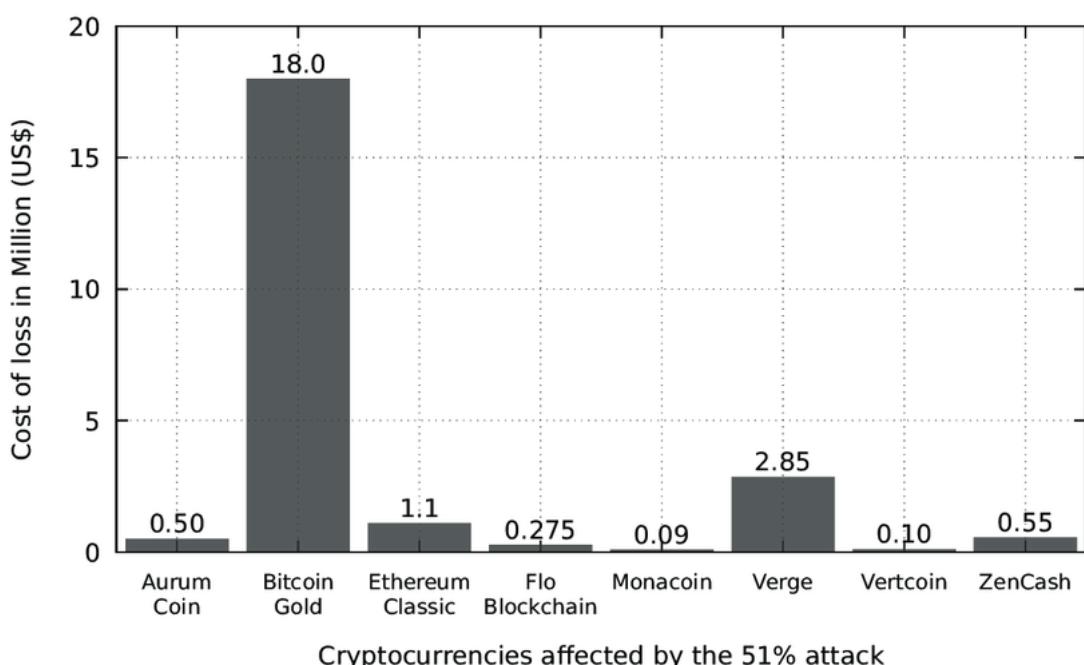
Un ataque Sybil es un tipo de amenaza de seguridad en un sistema en línea donde una persona trata de hacerse con el control de la red creando múltiples cuentas o nodos.

Esto podría llevarse a cabo mediante la creación de distintas direcciones IPs en un mismo nodo de manera que podría actuar como si hubiesen más nodos hasta lograr sobrepasar el 51% de los nodos de la red y tener el poder de imponer su cadena, que ahora tendría más probabilidades de ser la más larga, sobre la del resto de los nodos. Esto es evitado en nuestro caso y en el caso de Bitcoin o Ethereum gracias al algoritmo Proof of Work, que requiere al nodo y a las distintas cuentas falsas un esfuerzo extra para participar en la cadena, y tratar de mantener ese esfuerzo para tal cantidad de cuentas es una tarea inabordable tanto en términos de poder computacional como en términos energéticos. [9]

#### - 51% Attack

Un ataque del 51% sería un tipo de ataque dirigido hacia una blockchain con la intención de conseguir más de la mitad del cómputo generado por el resto de nodos o mineros de esa cadena. Gracias a este ataque, los ciberdelincuentes tendrían una mayor probabilidad de lograr obtener la cadena más larga y así conseguir que los cambios ilícitos que han llevado a cabo, sean validados y tomados como los originales.

En el caso de una criptomonedas, tal y como se muestra en la siguiente figura, los efectos de un ataque del 51% podrían ser desastrosos.



**Figura 2.1:** Gráfica pérdidas por ataque 51%

Pero no todas las criptomonedas o todas las cadenas de bloques pueden sufrir este tipo de ataque. Solo aquellas que utilizan Proof of Work como mecanismo de consenso.

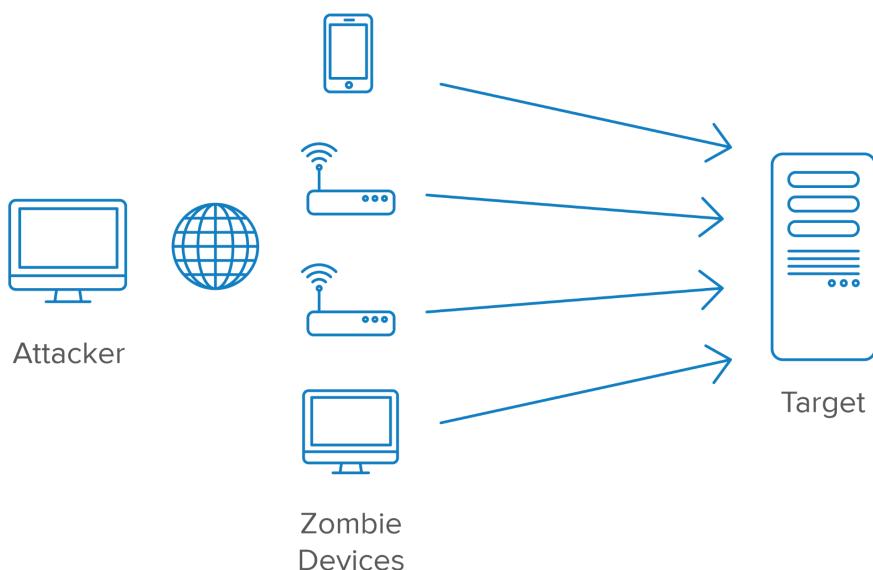
La principal defensa contra este tipo de ataques es tener una red lo más extensa posible, cuanto mayor sea el número de nodos de la red, mayor será la

cantidad de energía que necesitará el atacante para conseguir un poder computacional del 51%. [10]

- **Denial of service attack (DoS)**

En seguridad informática, un ataque de denegación de servicio o DoS, es un ataque a una red que provoca que un recurso o un servicio deje de ser accesible para usuarios legítimos. Normalmente se logra este objetivo mediante el consumo del ancho de banda de la red de la víctima o la sobrecarga de los recursos computacionales del sistema atacado.

Los ataques DoS se generan mediante la saturación de los puertos con múltiples flujos de información, haciendo que el servidor se sobrecargue y no pueda seguir prestando su servicio. Por eso se le denomina *denegación*, pues hace que el servidor no pueda atender la cantidad enorme de solicitudes.



**Figura 2.2 : Ilustración Denegación de servicio**

## 2.4. Aplicaciones Blockchain

- **BigchainDB**

BigchainDB es un software con propiedades de redes blockchain tales como descentralización o inmutabilidad y también con propiedades propias de una base de datos, como serían una tasa de transacciones eficiente, baja latencia y la indexación y consulta de datos estructurados. Fue lanzado a principios del año 2016 y ha ido mejorando y puliendo sus aspectos con el paso de los años. BigchainDB, a partir de su versión pasó a ser tolerante a fallos bizantinos por lo que ahora puede soportar hasta la pérdida de una tercera parte de los nodos. [3]

BigchainDB permite a los desarrolladores y a las empresas la implementación de plataformas y aplicaciones blockchain con una base de datos escalable. En lugar de tratar de escalar la tecnología blockchain, BigchainDB comienza con una base de datos distribuida y luego agrega características de blockchain - control descentralizado, inmutabilidad y la capacidad de crear y transferir activos. [24]

	Bitcoin Blockchain	Distributed Database	BIGCHAIN DB
Immutability	✓		✓
No Central Authority	✓		✓
Assets Over Network	✓		✓
High Throughput		✓	✓
Low Latency		✓	✓
Rich Permissioning		✓	✓
Query Capabilities	✓		✓

**Figura 2.3 :** Tabla comparación entre blockchain tradicional, una base de datos distribuida y el modelo de BigchainDB.

BigchainDB es compatible con una amplia gama de industrias y casos de uso, desde la identidad y la propiedad intelectual hasta las cadenas de suministro, la energía, el IoT y los ecosistemas financieros. Con un alto rendimiento, latencia de menos de un segundo y una potente funcionalidad para automatizar los procesos de negocio, BigchainDB se ve, actúa y se siente como una base de datos, pero tiene las características principales de blockchain que las empresas desean.

BigchainDB ofrece una implementación completa de su versión 1.0 en Github lo que permite estudiar más a fondo su solución, buscar posibles fallas en ella y aprender de esta. [2]

La arquitectura de BigchainDB 2.0 está compuesta por dos capas, la capa de consenso y la capa de almacenamiento. Cada nodo ejecuta su propia base de datos en MongoDB para almacenar los datos estructurados y manejar las peticiones. Cada transacción propuesta no es almacenada en la capa de MongoDB hasta que no es almacenada en la cadena. [7] El protocolo de consenso **Tendermint** garantiza que esta cadena sea BFT (Byzantine Fault Tolerant) al menos hasta con dos terceras partes de los nodos funcionando correctamente.[6]

- **Tendermint Core**

Tendermint Core es un software creado por la empresa Tendermint, que fue fundada por el desarrollador y creador del *whitepaper* original, Jae Kwon. [4][5]

Tendermint Core es un sistema que logra tolerancia a fallos siempre que al menos dos tercios de los participantes sean honestos. Para conseguir esto, se utiliza el mecanismo de consenso Proof of Stake (PoS) y para cada período, se selecciona un nodo aleatorio de un conjunto de validadores. Ese nodo debe proponer el siguiente bloque (en algo llamado sistema de turnos). Si los otros validadores están contentos con él, se agrega el nuevo bloque y se actualiza la cadena. La finalidad es instantánea: a diferencia de Bitcoin o Ethereum, no es necesario esperar a las confirmaciones para asegurarte de que tu transacción es válida.

La principal ventaja y la novedad que ha hecho de este software una ventaja tan potente es que Tendermint Core hace uso de una arquitectura modular, es decir, la capa de aplicación se puede encontrar disociada de la de consenso o la de redes, lo que implica que se puede conectar cualquier capa de aplicación al stack. Sin necesidad de preocuparse por los algoritmos de consenso.

Esta separación de la interfaz de aplicación y el mecanismo de consenso consigue una mayor flexibilidad y sencillez para lograr que aplicaciones centralizadas puedan incorporar cualquier lenguaje de programación en su lógica empresarial. [44]

## **2.5. Almacenamiento de historias clínicas digitales.**

Las historias clínicas, la información de Hacienda y Seguridad Social y expedientes judiciales, entre otros datos, se hallan almacenados en servidores gestionados por las administraciones públicas. La ubicación y gestión de estos centros de datos, es confiada de forma parcial a empresas externas.

En el caso de la información clínica y de salud, las comunidades autónomas tienen delegada la competencia de su gestión, que implica el almacenamiento y tratamiento de dicha información.

El Sistema Nacional de Salud dispone de un nodo central y una intranet sanitaria (ambos alojados en esa instalación principal), pero las historias clínicas digitales de los ciudadanos se encuentran en servidores ubicados en su comunidad autónoma. "Estos servidores contienen los episodios e informes médicos, acompañados por imágenes, resultados de pruebas de laboratorio, etc.". [33]

En el caso de los hospitales privados, estos firman acuerdos de privacidad y pasan auditorías documentales, pero dónde y cómo guardan los datos depende de ellos. Cabe destacar que todos los centros, incluidos los de ámbito privado, pueden acceder a los historiales almacenados en el resto de las instalaciones sanitarias.

El uso de los datos clínicos para investigación requiere de una motivación específica, la aprobación de un comité ético y la de identificación (vía seudonimización, eliminando los datos personales por un identificador cuya relación con los datos personales sólo conoce el proveedor o la anonimización completa, en la que se destruye dicha relación) de los datos. El riesgo que supone el tratamiento de los datos clínicos hace necesario garantizar la trazabilidad de su uso para garantizar el cumplimiento del propósito de su uso.

### 3. Análisis del problema

---

Una vez introducidos en el ámbito del trabajo y puestos en situación, no es difícil darse cuenta del amplio abanico de oportunidades que nos ofrece la tecnología *blockchain*. Estas facilidades unidas a la necesidad de una utilización y de un tratamiento serio y seguro de nuestros datos, en este caso los datos médicos, son las que han promovido la realización de este trabajo.

#### 3.1. Solución propuesta

La solución propuesta en este trabajo, consistirá en la creación de una cadena de bloques en la que poder almacenar datos ya sean médicos como en este ejemplo o de cualquier otro tipo de forma segura. Para ello, se creará una cadena de bloques con javascript y un sistema que nos permita actualizar esa cadena y conectar y sincronizar a todos los nodos que formen parte de la red. Cada vez que un nodo desee añadir nuevos datos a esta cadena deberá enviar una transacción donde irán almacenados los datos a añadir. Una vez la transacción haya sido validada será añadida a la cadena cuando uno de los nodos decida añadir un nuevo bloque con información a la cadena, para esto deberá llevar a cabo un proceso de minado para garantizar la seguridad de la cadena.

Está cadena será almacenada a su vez en un servidor MongoDB propio de cada nodo que a su vez estará alojado en un contenedor Docker.

Para lograr estos objetivos primero se comenzará con la construcción de una cadena básica, luego se llevará a cabo la construcción de un sistema de transacciones o envío de datos que nos permita enviar los datos que queremos almacenar de un nodo a otro, después se deberá construir la infraestructura que permita la creación, gestión y simulación de una red de nodos y por último se añadirán los correspondientes servidores MongoDB a cada nodo para que almacene la cadena que está corriendo en ese instante sobre estos.

#### 3.2. Especificación de requisitos

En esta especificación de requisitos software (ERS) aparece una descripción completa del comportamiento del sistema que se va a desarrollar, incluyendo un conjunto de los casos de uso que describen las interacciones que tendrán los usuarios con nuestro software. A continuación se exponen los distintos tipos de requisitos a cumplir por el proyecto.

##### 3.2.1 Requisitos funcionales

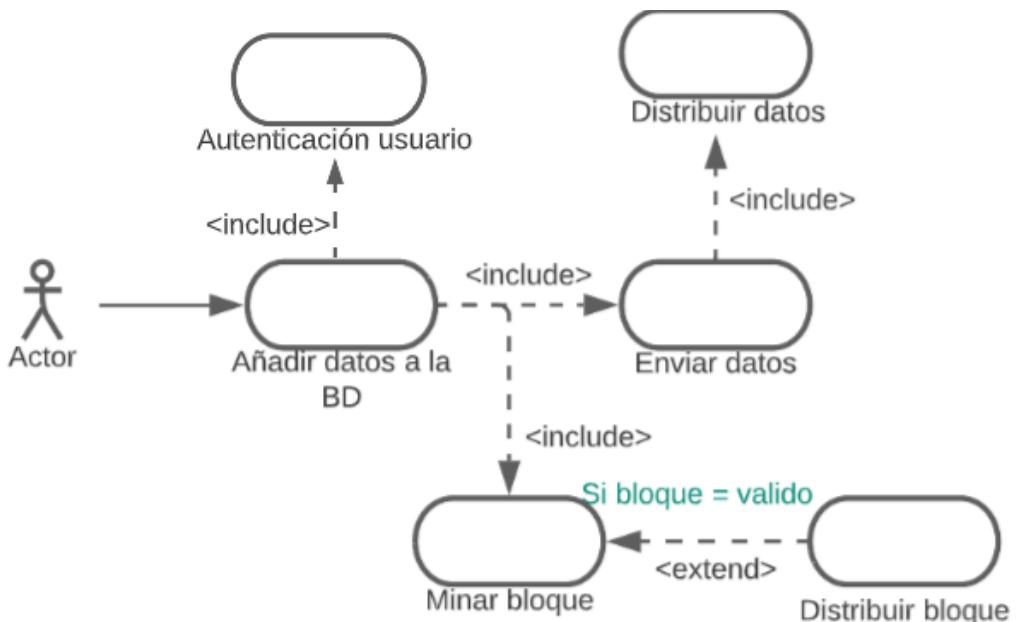
Dentro de los requisitos funcionales, encontramos el conjunto de casos de uso que describe la totalidad de las interacciones que tendrán los usuarios con el software.

Es decir, en ellos encontramos todos los servicios que deberá proporcionar el sistema una vez finalizado.

Los casos de uso de nuestro sistema serán los siguientes:

#### Añadir datos a la base de datos:

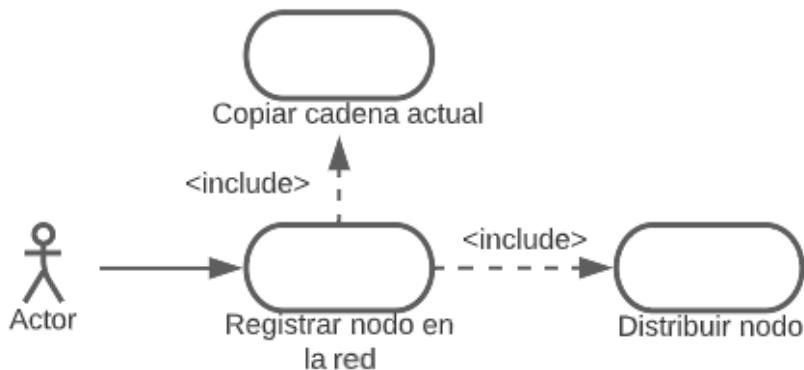
Si el usuario, en este caso alguien perteneciente al personal sanitario, desea añadir información a la base de datos, deberá disponer en primera instancia de una clave y de un usuario o identificador como personal sanitario para poder añadir información a nuestra base de datos. Una vez autenticado, puede enviar su nuevo paquete de datos en una “transacción” que se distribuirá por toda la red. Una vez distribuida y validada esta “transacción”, será añadida a la cadena de forma definitiva e irreversible una vez se mine o añada un nuevo bloque a la cadena, en este bloque se almacenarán todas las transacciones o paquetes de datos que estaban pendientes de validación hasta ese instante.



**Figura 3.0 : Caso de uso 1**

#### Registrar nuevo nodo en la red

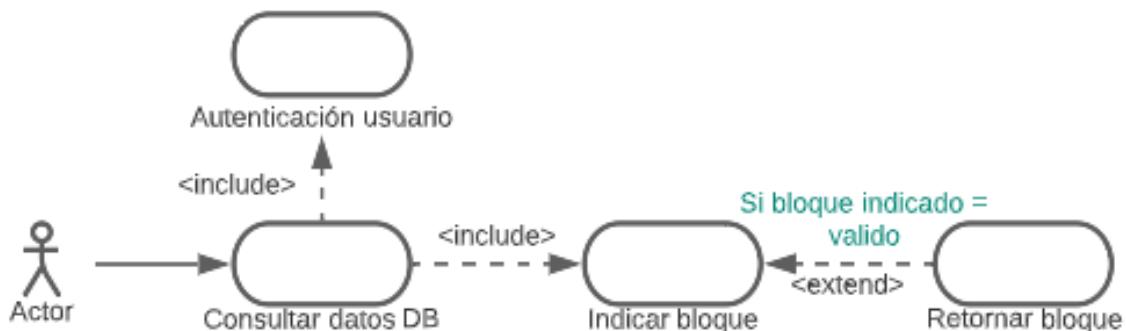
Si un nuevo nodo o usuario quiere pasar a formar parte de la red de nodos donde se almacena nuestra cadena, deberá en primera instancia, registrar su nodo en la red, para ello lanzará una petición para distribuir su nodo a la red a uno de los nodos de la red que será el encargado de hacer llegar al resto de nodos la adhesión de este nuevo nodo de la red. Una vez todos los nodos han añadido a este nuevo nodo y nuestro nodo ya conoce el resto de nodos de la red, el siguiente paso sería realizar una copia de la cadena que está corriendo en el resto de nodos a nuestro nuevo nodo, este paso se ejecutará de manera automática una vez se haya registrado el nodo.



**Figura 3.1 : Caso de uso 2**

#### Consultar datos de la cadena

Si un nodo desea consultar la información almacenada en un bloque, deberá, primero que nada autenticarse para poder tener acceso a la base de datos, una vez autenticado, deberá indicar el índice del bloque deseado tener acceso, si el bloque indicado existe, el sistema le devolverá la información almacenada en este bloque.



**Figura 3.2 : Caso de uso 3**

#### **3.2.2 Requisitos no funcionales**

##### **Análisis de la seguridad**

Antes de comenzar este apartado, me gustaría especificar claramente que el sistema de autenticación de esta aplicación está por desarrollar y sería un aspecto a tener en cuenta en caso de querer traspasar esta aplicación de un entorno demostrativo a un entorno real. En nuestro caso la autenticación consiste en los permisos y autenticación necesarios que ofrece la plataforma de MongoDB para tener acceso a la edición de una determinada base de datos, pero en un futuro, podría realizarse un sistema que requiriese de unas credenciales como sanitario desde la aplicación web, con estas credenciales se podría tener acceso a la creación y lectura

de bloques e incluso se podrían usar para generar certificados que permitieran identificar a un nodo como válido al igual que se realiza con las conexiones SSH.

Cómo ya se ha explicado brevemente en el apartado 2.3 Vulnerabilidades de una red blockchain, nuestra aplicación estará expuesta principalmente a tres tipos de ataques: ataques Sybil, ataques del 51%, y ataques DoS.

#### Protección frente ataques Sybil:

Frente a este tipo de ataques, nuestra aplicación, con una forma de autenticación ya depurada y no la meramente demostrativa que está implementada, se muestra como una alternativa robusta frente a otras posibles soluciones, ya que, en primera instancia, si un nodo malicioso quisiera tener acceso a la cadena debería en primer lugar disponer de credenciales de personal sanitario por lo que no podría simular nuevos nodos sin tener esas credenciales. Y en segundo lugar, si una vez conseguidas esas credenciales tratase de lograr un ataque del 51% gracias a la replicación de nodos, tendría que consumir una gran cantidad de energía en el minado de nuevos bloques debido a la aplicación de PoW.

#### Protección frente ataques 51%:

En este caso, además de que los nodos maliciosos tendrían que tener acceso a credenciales de personal sanitario y se les debería permitir el acceso a la cadena, lo cual sería bastante improbable, además deberían de conseguir una mayoría del 51% de la capacidad de cómputo de la red, que, suponiendo que está formada por la mayoría de los centros hospitalarios de un país sería algo complejo de lograr. [51]

#### Protección frente a ataques DoS:

Gracias a que la cadena está distribuida y almacenada en todos y cada uno de los nodos de la red, sería necesario derribar todos y cada uno de estos nodos para conseguir una completa denegación del servicio.

### Análisis energético o de eficiencia algorítmica

En cuanto a la eficiencia algorítmica o energética, este proyecto presenta bastantes carencias, en primer lugar, el algoritmo empleado para lograr una mayor seguridad frente a ataques del 51%, es altamente ineficiente al tratarse de un puzzle criptográfico que lleva un gran gasto de recursos. Esto podría solventarse aplicando otro mecanismo de consenso capaz de mantener la seguridad mejorando la eficiencia del programa, un ejemplo de estos algoritmos podrían ser, bien los indicados en el apartado de Alternativas actuales a Proof of Work dentro de 2.2 Herramientas para la construcción de Libros de Registros Distribuidos Blockchain como Proof of Elapsed Time o Proof of Capacity o bien Tendermint Core, empleado por BigchainDB.

En este caso se ha optado por Proof of Work por su sencillez de aplicación, la fiabilidad que ha demostrado con el paso de los años y la gran variedad de ejemplos disponibles en internet.

## Análisis del marco legal y ético

Al tratarse de un proyecto enfocado al almacenamiento de datos sanitarios, debemos tener en cuenta la actual legislación sobre el tratamiento de este tipo de datos así como la normativa que establece la protección de datos. La protección de datos es un derecho fundamental que se desarrolla a partir del artículo 18.4 de la Constitución Española. Cada persona de forma individual es la única facultada para decidir lo que se puede o no hacer con sus datos personales. [21]

La Ley Oficial de Protección de Datos (LOPD) define como dato de carácter personal cualquier información que identifique o haga posible la identificación de una persona física. Desde el 24 de octubre de 2007 esta ley es de obligado cumplimiento para todos los responsables de ficheros que contengan datos de carácter personal, ya sea en soporte papel o soporte informático.

A nivel Europeo, el Reglamento General de Protección de Datos (Reglamento 2016/679 del Parlamento Europeo y del Consejo de 27 de abril de 2016), establece el marco Europeo en el tratamiento de datos de carácter personal, haciendo referencia explícita en su artículo 4.15 a los datos "relativos a la salud física o mental de una persona física, incluida la prestación de servicios de atención sanitaria, que revelen información sobre su estado de salud".

Además de esto, existen otras regulaciones adicionales en el tratamiento de datos sanitarios y un marco legal emergente en el tratamiento de datos personales mediante técnicas de Inteligencia Artificial.

En el ámbito sanitario toda historia clínica representa un soporte de datos personales sobre la salud. Debe entenderse por datos de salud toda información concerniente a la salud pasada, presente y futura, física o mental, de un individuo, ya sea relativa a buena salud, enfermedad, diagnóstico, tratamiento, incluyendo, además, información sobre hábitos con el alcohol, drogas o sexuales. [20]

Apartados que exige la ley para el tratamiento de los datos:

### 1. Calidad de los datos

Sólo se podrán recoger datos de los pacientes que sean necesarios con la finalidad médica que se pretende, no excesivos, estériles o impertinentes para el fin médico.

### 2. Información y consentimiento

Al paciente se le debe informar de la existencia de un fichero (modelo por escrito), de la finalidad para la que sus datos se recogen, destinatario de la información, quien es el responsable de ese fichero, y la posibilidad de rectificación y cancelación. Además, es obligatorio solicitar el consentimiento del paciente para tratar sus datos. (salvo excepciones legales).

### 3. Seguridad

Hay que garantizar la seguridad de los datos obtenidos que se consideran de nivel alto, para ello es obligatorio tener un “documento de seguridad”. En este documento se establecerá el protocolo seguido por la clínica como por ejemplo las medidas que garantizan la seguridad de los datos.

### 4. Comunicación de los datos

Si creemos que los datos obtenidos van a ser comunicados a terceros se debe informar y solicitar el consentimiento del paciente. No se debe olvidar que la ley solo permite que se comuniquen datos para el cumplimiento de fines “directamente relacionados” con su proceso y en funciones legítimas entre cedente y cesionario.

### 5. Acceso, rectificación, cancelación y oposición.

Es obligatorio informar al paciente de la posibilidad de acceso a sus datos así como de la posible rectificación o cancelación de estos.

### 6. Inscripción de ficheros en la agencia española de protección de datos

Cada fichero que se cree debe ser notificado a la Agencia Española de Protección de Datos, así como todas sus modificaciones o cancelaciones.

Resaltar que hace referencia a la existencia del fichero, no a su contenido.

## Análisis de la seguridad de la información

La seguridad de la información es el conjunto de medidas preventivas y reactivas de las organizaciones y sistemas tecnológicos que permiten resguardar y proteger la información buscando mantener la confidencialidad, la disponibilidad e integridad de datos.[\[27\]](#)

### 1. Integridad

Es la propiedad que busca mantener los datos libres de modificaciones no autorizadas. Resumido de una forma simple, la integridad es mantener con exactitud la información tal cual fue generada, sin ser manipulada ni alterada por personas o procesos no autorizados.

La integridad de los datos se puede aplicar a cualquier base de datos y se puede clasificar en dos tipos principales: [\[26\]](#)

## INTEGRIDAD FÍSICA

La integridad física sería la protección de los datos alojados en la base de datos frente a factores externos, como podrían ser desastres naturales o

usuarios informáticos maliciosos. En nuestro caso, gracias a la distribución de la base de datos en cada uno de los nodos, la necesidad de autenticación y que una vez almacenado un dato no puede volver a ser modificado, podríamos considerar que la integridad física de los datos está garantizada.

## INTEGRIDAD LÓGICA

La integridad lógica trata la racionalidad de los datos presentes en la base de datos relacional. En nuestro caso, al tratarse de una base de datos no relacional carece de importancia.

### 2. Confidencialidad

Esta propiedad está lograda en parte gracias a la necesidad de autenticación del personal para acceder a los datos, pero podría ser mejorada añadiendo un cifrado hash a los datos para que no fueran visibles hasta que se accediese al bloque con los debidos permisos.

Además se podría añadir un contador de registros que llevase cuanta de quien accede a cada recurso así como el número de veces que ha accedido y las fechas

### 3. Disponibilidad

La disponibilidad es una característica o cualidad de la información que consiste en la capacidad de encontrarse a disposición de quienes necesitan tener acceso a ella. Dicho de una forma resumida, la disponibilidad es la posibilidad de acceso a la información y a los sistemas por personas autorizadas en el momento que así lo requieran.

Esta cualidad se considera lograda por el proyecto ya que gracias a que funciona sobre una red distribuida, tiene una gran tolerancia a fallos, por lo que las probabilidades de falla en el sistema y de no disponibilidad son mínimas.

## Autenticación

Es la propiedad que permite identificar el generador de la información. Por ejemplo al recibir un mensaje de alguien, estar seguro de que es de ese alguien el que lo ha mandado, y no una tercera persona haciéndose pasar por la otra (suplantación de identidad). En un sistema informático se suele conseguir este factor con el uso de cuentas de usuario y contraseñas de acceso. [28]

Esta propiedad estaría lograda en parte en nuestro proyecto gracias a la necesidad de autenticación tanto para leer como para escribir datos en MongoDB pero podría ser ampliamente mejorado como ya se ha especificado con anterioridad mediante un sistema de registro de usuarios o personal sanitario más avanzado y que contuviese comprobaciones de pertenencia a la organización.

# 4. Diseño de la solución

---

## 4.1. Arquitectura del Sistema

Esta solución está basada en una arquitectura P2P o peer-to-peer. Esta arquitectura se caracteriza por la igualdad entre todos los nodos de la red. Cada nodo actuaría de manera simultánea como cliente y como servidor respecto a los demás nodos de la red. Esta arquitectura sería la opuesta a la clásica arquitectura cliente-servidor donde los nodos pueden tener distintos roles, unos tendrían el rol de ofrecer un servicio, servidores, y otros de demandar este servicio, clientes.

Gracias a esta arquitectura, *blockchain* logra eliminar la dependencia de los nodos clientes respecto de los nodo servidores y las posibles prácticas abusivas de estos además de garantizar la disponibilidad de los datos al tenerlos alojados en tantos nodos como participantes tenga la red.

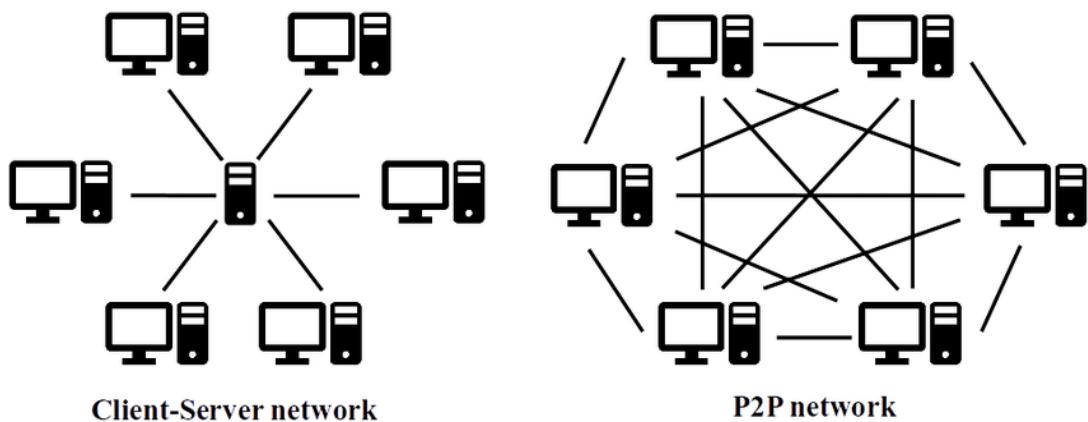


Figura 4.0 : Esquema tipos de redes

Nuestra solución, al igual que la solución de BigchainDB contará con dos capas, en primer lugar está la capa que se encargará del algoritmo de consenso y de implementar todas las funcionalidades de la cadena, esta capa también será la encargada de almacenar la cadena en un array dentro de la propia memoria del ordenador que esté ejecutando el programa. Esta cadena estará distribuida por el resto de nodos de la red por lo que todos los nodos dispondrán de una idéntica copia de la *blockchain*. La segunda capa será la capa de almacenamiento o capa de datos que consiste en una copia en MongoDB de la cadena que está corriendo en ese momento en todos los nodos de la red. Esta cadena se actualizará cada vez que se mine (/mine) un nuevo bloque, justo en el instante posterior a la actualización a la cadena local. En esta cadena, a diferencia de en la cadena local, solo se podrán ver aquellas transacciones que ya hayan sido confirmadas.

Esto es útil ya que en caso de fallo en la red y caída de todos los nodos que mantenían una copia de la cadena, nuestra aplicación arrancaría cargando una copia idéntica de la cadena, además nos permite la consulta de los datos de la cadena desde la aplicación de MongoDB Compass, lo cual puede resultar de gran utilidad.

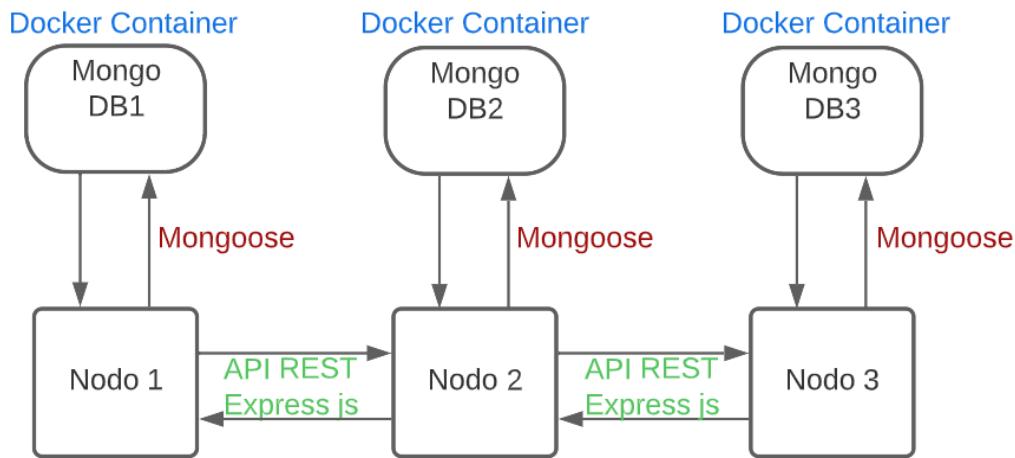
Tambiénaremos uso de MongoDB para almacenar las imágenes de la consulta, estas serán almacenadas en una base de datos distinta a la de la cadena para evitar una sobrecarga de la aplicación a la hora de realizar el minado, facilitar la lectura de la cadena y reducir los tiempos de ejecución.

Un aspecto que puede resultar llamativo es la estructura seguida en la implementación de este código, concretamente la estructura seguida en el fichero *blockchain.js*, en esta, se hace uso de funciones constructor para construir nuestra estructura de datos de *blockchain* en lugar de usar clases. La razón que ha motivado el uso de funciones constructoras en lugar de hacer uso de las clases es que aparte de ser una forma de trabajar nueva para mí y por lo tanto más llamativa a la hora de aprender, lo cierto es que en JavaScript no existen realmente las clases, las clases en realidad son como una forma de endulzar o dar otra apariencia a la parte superior del constructor y del objeto prototype.

## Componentes de la aplicación

Esta solución consta de cinco componentes principales:

- **Nodos o usuarios** para crear y distribuir los datos que posteriormente serán almacenados en la cadena de bloques. Estos nodos serán a su vez los encargados de hospedar su propia copia de la cadena.
- Una **API** que permitirá tanto la gestión de la red de nodos permitiendo, controlando y gestionando las nuevas adhesiones a la red como las interacciones de los nodos con la cadena de bloques tales como la adhesión de bloques o el envío de datos al resto de la red. Todas las funciones de esta API han sido implementadas haciendo uso de la infraestructura de aplicaciones Express que proporciona Node.js.
- Una **base de datos donde almacenar la blockchain**, esta base de datos estará almacenada en MongoDB y a su vez estará alojada dentro de un contenedor Docker. En esta base de datos se almacenarán los bloques de la cadena con sus respectivos datos. Cada uno de estos nodos tendrá acceso a una base de datos distinta y particular hospedada en contenedores Docker distintos que será actualizada de forma simultánea a la del resto de nodos. Las comunicaciones entre los servidores Mongo y nuestra aplicación se han creado y gestionado con la librería mongoose proporcionada por Node.js.
- Una **base de datos donde almacenar imágenes**, con la finalidad de no sobrecargar la blockchain y lograr una mayor rapidez de ejecución las imágenes introducidas serán almacenadas en una base de datos distinta a la que está almacenada cadena, está, al igual que la anterior también estará soportada por MongoDB y hospedada a su vez dentro de un contenedor.
- Un **modelo de datos** que defina la estructura de los bloques a almacenar en la cadena para evitar posibles incongruencias en la estructura de los datos de los bloques almacenados y mantener un orden en estos.
- **Contenedores Docker** que serán los encargados de hospedar las bases de datos de MongoDB.



**Figura 4.1 : Arquitectura de la aplicación**

## 4.2. Diseño Detallado

Como ya se ha recalcado en apartados anteriores, hay que lograr que nuestro Libro de registros sea inmutable, con el objetivo de lograr este fin, la estructura de los bloques que añadiremos a la cadena estará compuesta de los siguientes atributos:

- Un índice que indica la posición en la que fue añadida el bloque,
- un atributo timestamp que indica el momento de creación del bloque,
- un atributo transactions donde se almacenarán todas las transacciones que estaban pendientes hasta la fecha,
- un atributo nonce que se explicará más adelante,
- el hash del bloque actual generado mediante la función hash criptográfica SHA-256
- y el hash del bloque que le precede.

```
const newBlock = {
    index: this.chain.length + 1,
    timestamp: Date.now(),
    transactions: this.pendingTransactions,
    nonce: nonce,
    hash: hash,
    previousBlockHash: previousBlockHash,
};
```

**Fragmento de código 1: blockchain.js**

Como se puede observar en el anterior fragmento de código, nuestro bloque consta de los siguientes atributos:

1. **index**: Un índice que nos indicará en qué posición se ha añadido el bloque.
2. **timestamp**: Una marca temporal que nos indicará en qué momento fue creado el bloque.
3. **transactions**: Un array de transacciones en el que incluir todas las transacciones que estaban pendientes de validación en el momento de creación del bloque.
4. **nonce**: este *nonce* es la prueba de que hemos creado este nuevo bloque de forma legítima.
5. **hash**: hash resultado de llamar a la función de hashing *hashBlock* con los datos del bloque actual, el hash del bloque anterior y el nonce obtenido de la llamada a la función *proofOfWork*.
6. **previousBlockHash**: hash del bloque anterior.

Para añadir un nuevo bloque a la cadena, antes habría que hacer una llamada al endpoint /mine.

Otro aspecto importante a señalar sería la estructura de las transacciones:

```
const newTransaction = {  
  data: {  
    datos: newData.datos,  
    address: newData.address,  
    responsable: newData.responsable,  
    paciente: newData.paciente,  
    imageUniHash: sg(newData.images, timer),  
    imagesHash: sg(newData.images, ""),  
  }, // .map((i) => btoa(i)),  
  transUUID: v4().split("-").join(""),  
  timestamp: timer,  
  /**Unique Id for every transaction */  
};
```

**Fragmento de código 2: blockchain.js**

Como se puede observar en la figura anterior, las transacciones constan de 3 atributos:

1. **data**: este atributo u objeto de la transacción contendrá toda la información de esta y está compuesto de los siguientes apartados:

- Un primer apartado de datos donde podrán ir anotados los distintos informes de la consulta y demás datos relativos a esta.
- Un apartado `address` o dirección en el que registrar la dirección del centro desde el que se han subido los datos a la cadena.
- Otro apartado donde quede registrado el responsable de la consulta o tratamiento.
- Otro donde quede registrado el paciente
- Un apartado para almacenar el hash de la imagen junto con el temporizador, encargado de identificar a la imagen junto con el momento en el que fue introducida para poder diferenciarla de una misma imagen que haya sido introducida en otro momento.
- Y un último apartado `imagesHash` donde se almacena el hash de la imagen introducida, que nos permitirá comprobar la integridad de esta además de servirnos de identificador para después localizarla en la base de datos.

2. **TransUUID**: Universal Unique Identifier (UUID) o Identificador único universal para identificar de manera única cada transacción. Un UUID es una etiqueta de 128 bits utilizada para identificar cualquier clase de información en un sistema informático. La representación estándar de un UUID como es en nuestro caso suele ser en una cadena de 32 números hexadecimales. En este caso hemos aplicado la versión 4 que utiliza un generador de números pseudoaleatorios o función hashing (MD5) para generar el UUID de 128 bits, lo que nos permite un total de  $2^{128}$  combinaciones, es decir, las probabilidades de colisión son prácticamente nulas de  $1/(2^{128})$ . [\[19\]](#)

3. **timestamp**: este timestamp almacena la fecha en la que fue creada la transacción.

Estas transacciones serán almacenadas en un array fuera de la cadena antes del minado de un nuevo bloque, la principal razón de esta decisión es el hecho de que el responsable o doctor desee revisar la correctitud de los datos que van a ser introducidos a la cadena antes de que estos sean almacenados de manera definitiva e irreversible en esta.

También es importante señalar que a diferencia de otras muchas aplicaciones blockchain estas transacciones no llevan un coste asociado, esto es en primer lugar, porque al tratarse de un sistema enfocado en un ámbito sanitario no he creído conveniente añadir ningún tipo de coste ya que no consideraba oportuna ni lógica la creación de ningún tipo de moneda o de coste para el hecho de almacenar datos de



un paciente por una persona o profesional que ya ha sido autenticado con anterioridad en la aplicación.

### 4.3. Endpoints y funcionalidades

En este apartado se describen todas las funcionalidades e interacciones que el usuario puede llevar a cabo en el sistema. Dentro del archivo netNode.js encontramos todos los *endpoints* de nuestra aplicación, estos van desde la creación de la red de nodos a la inserción de nuevos bloques de datos a la cadena.

Una parte fundamental de la tecnología *blockchain* son los nodos que componen la red sobre la que se distribuye la cadena. Para que estos nodos puedan interactuar entre sí y escribir sobre una misma cadena, primero deberán estar registrados en la misma red. En esta ocasión se ha llevado a cabo la implementación de distintos *endpoints* para poder registrar nuevos nodos y crear a partir de estos una red en la que poder interactuar

A continuación se muestran los *endpoints* creados para crear y gestionar esta red y se explica de forma detallada qué llamadas son necesarias para la construcción de la red así como el orden en que deberán ser ejecutadas.

#### 4.3.1 Endpoints para la construcción de la red

##### Register-and-broadcast-node

La función de este *endpoint* es registrar el nuevo nodo y hacer broadcast del nuevo nodo al resto de nodos que ya estaban presentes en la red. Estos otros nodos, simplemente aceptarán el nuevo nodo y lo añadirán a su lista de nodos activos usando el *endpoint* /register-node.

/register-and-broadcast-node

POST (JSON)

```
{  
    "newNodeUrl": "http://localhost:3005"  
}
```

##### Register-node

El *endpoint* /register-node es el *endpoint* en el cual todos los nodos de la red recibirán el broadcast enviado desde el *endpoint* /register-and-broadcast-node. La única tarea de este *endpoint* será registrar el nuestro nuevo nodo en el nodo que recibe la petición. No es obligatorio llamar de forma directa a este nodo, su llamada se realiza

de forma secundaria a la llamada a /register-and-broadcast-node pero se puede realizar el registro de forma manual en cada nodo cambiando el atributo “newNodeUrl” del Body de la petición en cada nodo.

```
/register-node  
POST (JSON)  
{  
    "newNodeUrl": "http://localhost:3002"  
}
```

## Register-nodes-bulk

Este *endpoint* se ejecutará una vez el nuevo nodo haya sido registrado en la red y se haya realizado el broadcast.

Su función es registrar de forma rápida todos los nodos que ya se encuentran presentes en red en nuestro nuevo nodo.

Por ejemplo, si quisiéramos añadir el nodo 3005 a la red, una vez finalizado el /register-and-broadcast-node llamaríamos a <http://localhost:3002/register-nodes-bulk> con el cuerpo de mensaje compuesto por los nodos ya existentes en la red, para registrar todos los nodos de la red dentro de este nuevo nodo.

```
/register-nodes-bulk  
POST (JSON)  
{  
    "allNetworkNodes": [  
        "http://localhost:3002",  
        "http://localhost:3003",  
        "http://localhost:3004"
```

## Aclaraciones sobre la construcción de la red:

Los *endpoints* /register-and-broadcast-node y /register-node son esencialmente distintos. Básicamente, lo que sucederá aquí es que cada vez que queramos registrar un nuevo nodo en nuestra red es que vamos a hacer una llamada al *endpoint* /register-and-broadcast-node. Este *endpoint* registrará el nuevo nodo en su propio servidor y luego transmitirá este nuevo nodo a todos los demás nodos de la red. Esos

nodos de red simplemente aceptarán el nuevo nodo de red dentro del *endpoint* /register-node, y es que todo lo que estos nodos deben hacer es simplemente registrar los nodos que se han transmitido mediante el broadcast. Solo queremos que registren el nuevo nodo, no es necesario que transmitan el nuevo nodo porque ya ha sido transmitido anteriormente. Si todos los demás nodos de la red también transmitieran el nuevo nodo, eso reduciría gravemente el rendimiento de nuestra red *blockchain* y daría lugar a un bucle infinito que colapsaría nuestra *blockchain*. Por lo tanto, cuando todos los demás nodos de la red reciban la URL del nuevo nodo, solo queremos que la registren y no la difundan.

## RESUMEN DE LA CONSTRUCCIÓN DE LA RED.

Ahora hagamos un repaso del *endpoint* /register-and-broadcast-node de nuevo de forma resumida para lograr una mejor comprensión de cómo se construye la red. Cuando queramos registrar un nuevo nodo en nuestra red, el primer paso debe ser llamar al *endpoint* /register-and-broadcast-node. Lo primero que se está llevando a cabo dentro de este *endpoint* es tomar el valor de newNodeUrl y registrarlo en el nodo actual enviándolo dentro del array networkNodes. El siguiente paso que se llevará a cabo es transmitir este newNodeUrl al resto de nodos de nuestra red. Para esto utilizamos un bucle forEach. Todo lo que está pasando dentro de este bucle es que estamos haciendo una solicitud a cada uno de los otros nodos de nuestra red. Estamos haciendo esta solicitud al *endpoint* /register-node. Entonces estamos almacenando todas estas solicitudes en nuestro array de promesas de /register-node, y luego simplemente estamos ejecutando todas esas solicitudes. Una vez que todas estas solicitudes se completen sin ningún error, podemos asumir que el newNodeUrl ha sido registrado exitosamente con todos nuestros otros nodos de red. Una vez completado nuestro broadcast, lo siguiente que queremos hacer es registrar todos los nodos de red que ya están presentes dentro de nuestra red en nuestro nuevo nodo. Para ello, hacemos una sola solicitud a nuestro nuevo nodo y llamamos al *endpoint* /register-nodes-bulk. Los datos que pasamos a este *endpoint* son las URL de todos los nodos que ya están presentes dentro de nuestra red.

Todos estos pasos se llevarán a cabo dentro del *endpoint* /register-and-broadcast-node, que se encargará a su vez de llamar a los otros dos *endpoints*, /register-nodes-bulk y /register-node.

## EXPLICACIÓN GRÁFICA DE LA CONSTRUCCIÓN DE LA RED

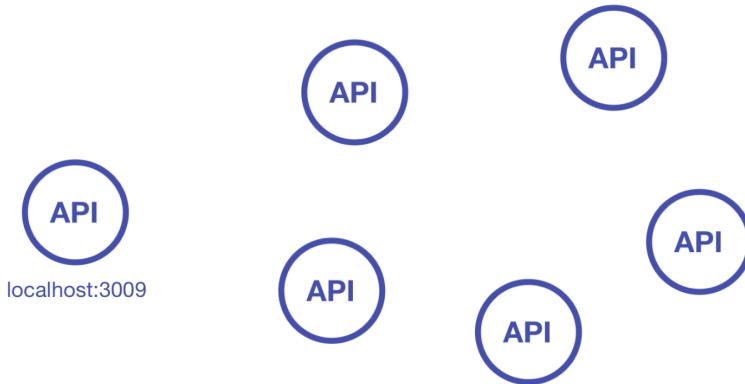


Figura 4.1 : Estado inicial de la red

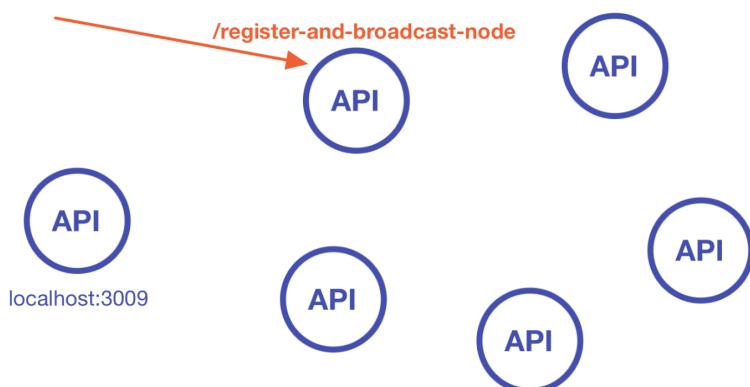


Figura 4.2 : Estado de la red tras ejecutar register-and-broadcast-node

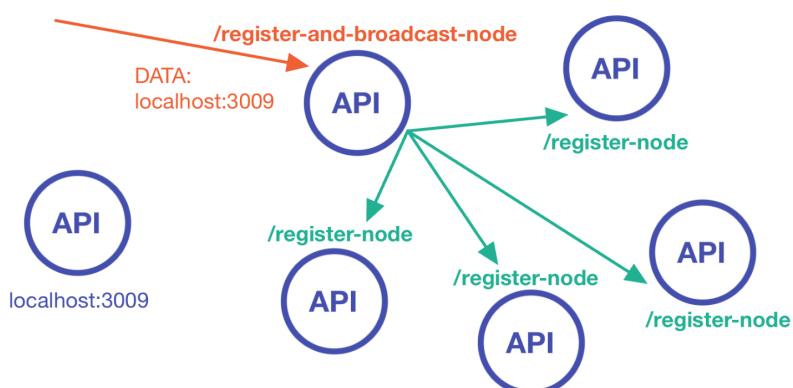
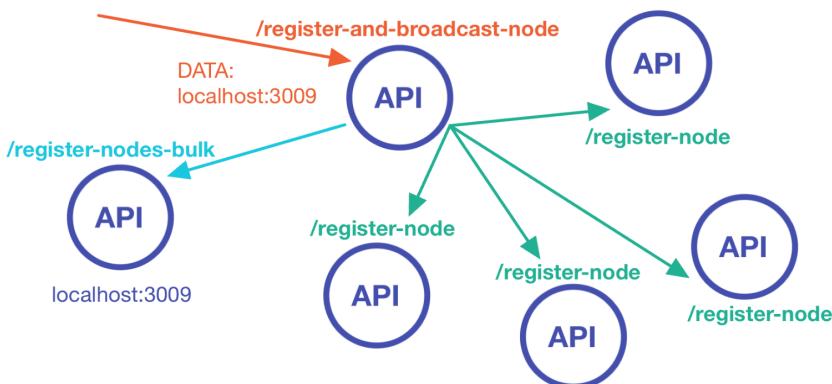


Figura 4.3 : Distribución del nuevo nodo tras ejecutar register-and-broadcast-node



**Figura 4.4 :** Llamada a register-nodes-bulk para finalizar el registro

#### 4.3.2 Endpoints para la gestión de datos

##### Blockchain

Una vez puesto en marcha el servidor, para poder acceder al resto de funcionalidades de la cadena será necesario ejecutar este *endpoint* en primer lugar para cargar la cadena actualizada y así poder realizar operaciones sobre esta.

A parte de esto, este *endpoint* será el encargado de mostrar en pantalla la cadena actual actualizada, con este *endpoint* podemos mostrar todos los bloques que la componen así como todos los nodos que forman la red. Está diseñado para realizar las pruebas más que para un caso de uso real donde podría resultar más interesante obtener los bloques de manera individual. Otra actualización necesaria y futurable sería mostrar los datos de los bloques cifrados con una función hash en lugar de exponer toda la información al solicitar el bloque.

/blockchain

GET

The screenshot shows a Postman interface with a GET request to `http://localhost:3001/blockchain`. The response status is 200 OK, with a timestamp of 2022-08-14T12:36:38.820Z, size of 8.63 KB, and a duration of 16 ms. The response body is a JSON object representing a blockchain chain with five blocks. Each block has an '\_id', 'index', 'timestamp', 'transactions' (empty array), 'nonce', 'previousBlockHash', 'hash', and a '\_v' field set to 0.

```
{
  "chain": [
    {
      "_id": "62f8ec5637e6e08433152ac5",
      "index": 1,
      "timestamp": "2022-08-14T12:36:38.820Z",
      "transactions": [],
      "nonce": 58,
      "previousBlockHash": "0",
      "hash": "0",
      "_v": 0
    },
    {
      "_id": "62f8ec5637e6e08433152ac6",
      "index": 2,
      "timestamp": "2022-08-14T12:36:38.910Z",
      "transactions": [],
      "nonce": 18140,
      "previousBlockHash": "0",
      "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100",
      "_v": 0
    },
    {
      "_id": "62f8ec6b37e6e08433152acb",
      "index": 3,
      "timestamp": "2022-08-14T12:36:59.757Z",
      "transactions": [],
      "nonce": 92894,
      "previousBlockHash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100",
      "hash": "00002778916a7dadc7260a1b6cff17be291bda44445d157e48da55fe9dbb06b3",
      "_v": 0
    },
    {
      "_id": "62f919986915bdcbdf3eec29",
      "index": 4,
      "timestamp": "2022-08-14T15:49:44.252Z",
      "transactions": [],
      "nonce": 92100,
      "previousBlockHash": "00002778916a7dadc7260a1b6cff17be291bda44445d157e48da55fe9dbb06b3",
      "hash": "0000d10859b91ab623a0bab0ddccbc68635b06f6fd582017b055e787303a569",
      "_v": 0
    },
    {
      "_id": "62f91f0716620aeaf6a7ae2b",
      "index": 5,
      "timestamp": "2022-08-14T16:12:55.740Z",
      "transactions": [],
      "nonce": 125853,
      "previousBlockHash": "0000d10859b91ab623a0bab0ddccbc68635b06f6fd582017b055e787303a569"
    }
  ]
}
```

**Figura 4.5 :** Cadena mostrada tras ejecutar el endpoint /blockchain

## Transaction

Con este *endpoint* se facilita a los nodos la creación de nuevas estructuras de datos para añadirlas a la cadena y distribuirlas por toda la red. La estructura de datos elegida para estas sería la mostrada en la siguiente imagen pero podría variar en función de las necesidades de los usuarios de la aplicación. Este método no está pensado para llamarse directamente sino para servir de soporte al *endpoint* que se menciona a continuación.

/transaction

POST (JSON)

```
{
  "data": {
    "datos": "Datos del paciente: ++++++",
    "address": "HospitalDrPesetOffice",
    "responsable": "Jim Halpert",
    "paciente": "23898947V",
    "images": [ "...", ... ]
  }
}
```

## Transaction Broadcast

Versión actualizada del anterior que realizando llamadas en serie a este consigue crear y distribuir la transacción por toda la red para que todos los nodos tengan acceso a esta.

/transaction/broadcast

POST (JSON)

```
{  
    "data": {  
        "datos": "Datos del paciente: ++++++",  
        "address": "HospitalDrPesetOffice",  
        "responsable": "Jim Halpert",  
        "paciente": "23898947V",  
        "images": ["...", "..."]  
    }  
}
```

Las siguientes imágenes ilustran el funcionamiento de este endpoint, y como al ser ejecutado añade la nueva transacción al array de transacciones pendientes (pendingTransactions) de todos los nodos disponibles en la red en ese momento.

```
8257d5fb16327ebbed", "__v":0}], "pendingTransactions": [{"data": {"datos": "Datos del paciente: ++++++", "address": "HospitalDrPesetOffice", "responsable": "Jim Halpert", "paciente": "23898947V", "images": []}, "transUUID": "07a9d90401454bd1981a2f6aed5e6acc"}], "networkNodes": [{"http://localhost:3002"}, {"currentNodeUrl": "http://localhost:3001"}]
```

**Figura 4.6 :** Transacción añadida en el nodo 3001

```
"000035ee66e25c695872a6c30f33827bf2729f9de5032057b98ad80aa67b4e1a", "__v":0}], "pendingTransactions": [{"data": {"datos": "Datos del paciente: ++++++", "address": "HospitalDrPesetOffice", "responsable": "Jim Halpert", "paciente": "23898947V", "images": []}, "transUUID": "cb45a9b37cc341939db42f50159e3d65"}, {"data": {"datos": "Datos del paciente: ++++++", "address": "HospitalDrPesetOffice", "responsable": "Jim Halpert", "paciente": "23898947V", "images": []}, "transUUID": "07a9d90401454bd1981a2f6aed5e6acc"}], "networkNodes": [{"http://localhost:3001"}, {"currentNodeUrl": "http://localhost:3002"}]
```

**Figura 4.7 :** Transacción añadida en el nodo 3002

## Mine

Este *endpoint* permitirá a todos los nodos añadir nuevos bloques a la cadena mediante un proceso de minado. Una vez se haya minado el nuevo bloque se almacenará en este toda la información (transacciones) que estaban a la espera de ser validadas.

Este minado se realiza de forma manual y no automática ya que se considera que un doctor o responsable puede desear revisar con anterioridad los datos que van a ser introducidos en la cadena de forma definitiva.

/mine

GET

En la siguiente imagen se puede observar el comportamiento del endpoint /mine, y como este almacena las transacciones pendientes en su array transactions.

```

2   "note": "Se ha completado el minado de un nuevo nodo",
3   "block": {
4     "index": 3,
5     "timestamp": 1662576211567,
6     "transactions": [
7       {
8         "data": {
9           "datos": "Datos del paciente: ++++++",
10          "address": "HospitalDrPresetOffice",
11          "responsable": "Jim Halpert",
12          "paciente": "23898947V",
13          "imageUniHash": [
14            "9ec736e59445f8059c839b08236f07d986ec685f5f9bf0ba1f9b40c4dea1b2aab",
15            "05e5010a3963338bd9e1f636f50563f557d048cdf95ee1d43c759511869c3ed8"
16          ],
17          "imagesHash": [
18            "0affff57e9209a27e7b97f9fcfc5c0c91d031a1a570d7512b09dd40fe6558e384",
19            "0c443a986602d976dc802fc2112b09b2418bd456f164de87d964e31265959021"
20          ]
21        },
22        "transUUID": "5094af06438a4e11b776afb3c6db4b4e",
23        "timestamp": "2022-09-07T18:43:28.271Z"
24      }
25    ],
26    "nonce": 7457,
27    "hash": "000089c549cd34935dc673713e5262306bf25fd3f0844b69fb329ba7",
28    "previousBlockHash": "0"
29  }

```

**Figura 4.8 :** Ejecución del endpoint /mine desde Postman

Por último para comprobar que el minado ha sido efectivo vamos a volver a realizar una llamada al endpoint /blockchain para observar que el nuevo bloque ha sido añadido correctamente a la cadena. En la imagen que subsigue a este texto podemos ver que el nuevo bloque ha sido añadido de manera satisfactoria.

```

BcmQ6avv//Snf9+ndfr/wOdaitdFAMAQwAAAABJRU5ErkJgg=="}},"transUUID":"06b5ae97cf3c43c9a3fb8902e2a9cc10"}],"nonce":77387
,"previousBlockHash":"0000791aae470b01cf117b932581cf41f988490775a2a94db618c891d4dccbd37","hash":"000036d8a30767b44b33f
3908c31646e353d1c777f0157dafb3e6d7457e2f6ca","__v":0}, {"_id": "6314fb8e5600ba2518d5a80d", "index": 13, "timestamp": "2022-09-04T19:25:02.608Z", "transactions": [{"data": {"id": "Datos del paciente: ++++++", "address": "HospitalDrPresetOffice", "responsable": "Jim Halpert", "paciente": "23898947V", "images": [], "transUUID": "1f78b4f211a748a7a6ba965fe3082b21"}, {"nonce": 79858, "previousBlockHash": "000036d8a30767b44b33f3908c31646e353d1c777f0157dafb3e6d7457e2f6ca", "hash": "00002ee9e088d24c4306f9ae9832ad9c828ab126b6ef00e775b3a37d7fa06e05", "__v": 0}, {"_id": "6314fdad5600ba2518d5a812", "index": 14, "timestamp": "2022-09-04T19:34:05.378Z", "transactions": [{"data": {"datos": "Datos del paciente: ++++++", "address": "HospitalDrPresetOffice", "responsable": "Jim Halpert", "paciente": "23898947V", "images": []}, {"transUUID": "d5841e0ade6b43e79260bc92c36658c4"}]}, {"nonce": 96963, "previousBlockHash": "00002ee9e088d24c4306f9ae9832ad9c828ab126b6ef00e775b3a37d7fa06e05", "hash": "0000ba593f68a6cb5e3cef0b8beb3563969aee2942bf27a30ff7b793653be54a", "__v": 0}], "pendingTransactions": [], "networkNodes": [], "currentNodeUrl": "http://localhost:3001"}}

```

**Figura 4.9 :** Comprobación de la correcta adhesión del bloque.

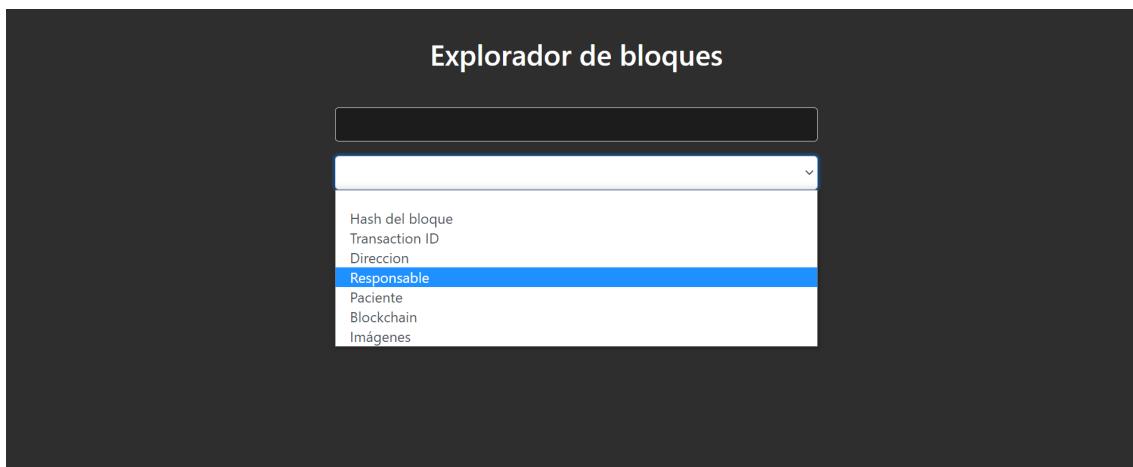
#### 4.3.3 Endpoints para la gestión de la aplicación

##### View

Ejecutaremos este *endpoint* para acceder a la web de la aplicación. Desde esta ventana realizaremos las llamadas a las distintas funcionalidades de la cadena.

/views

GET



**Figura 4.10 :** Menú de inicio

Una vez dentro de la app podemos realizar las siguientes operaciones:

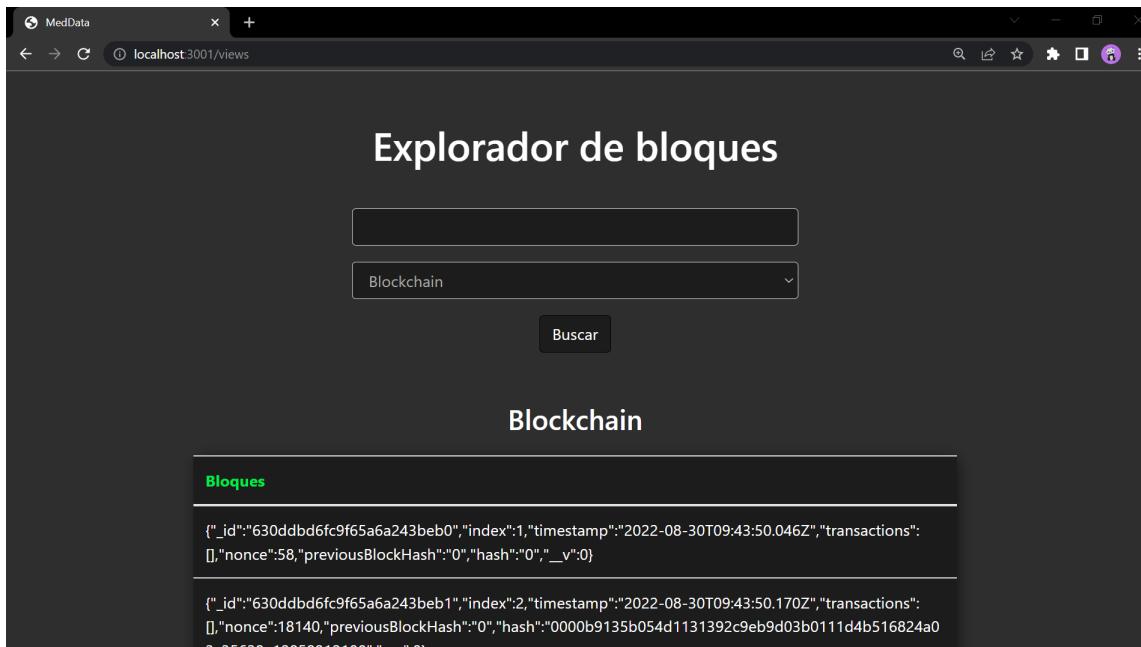
1. Obtener un bloque particular indicando su hash.
2. Obtener la información de una “transacción” o de un conjunto de datos introduciendo su ID (TransUUID).
3. Obtener todos los bloques que provengan de una determinada dirección, centro u hospital.
4. Obtener todos los bloques almacenados por un responsable X.
5. Obtener todos los bloques en los que aparecen datos de un cierto paciente Y.
6. Obtener la blockchain completa.
7. Obtener una imagen introduciendo su hash.

##### Blockchain

Al igual que la función del mismo nombre mostrada en el apartado 4.3.2 que devolvía un objeto JSON con la cadena completa, esta función devolverá, esta vez a través de la interfaz web, todos los bloques que componen la cadena.

/blockchain1

GET



**Figura 4.11 : Ejecución opción Blockchain**

## Hash del bloque

Con esta opción se le facilita al usuario la opción de obtener los datos de un determinado bloque indicando su *hash*.

/block/:blockHash

GET

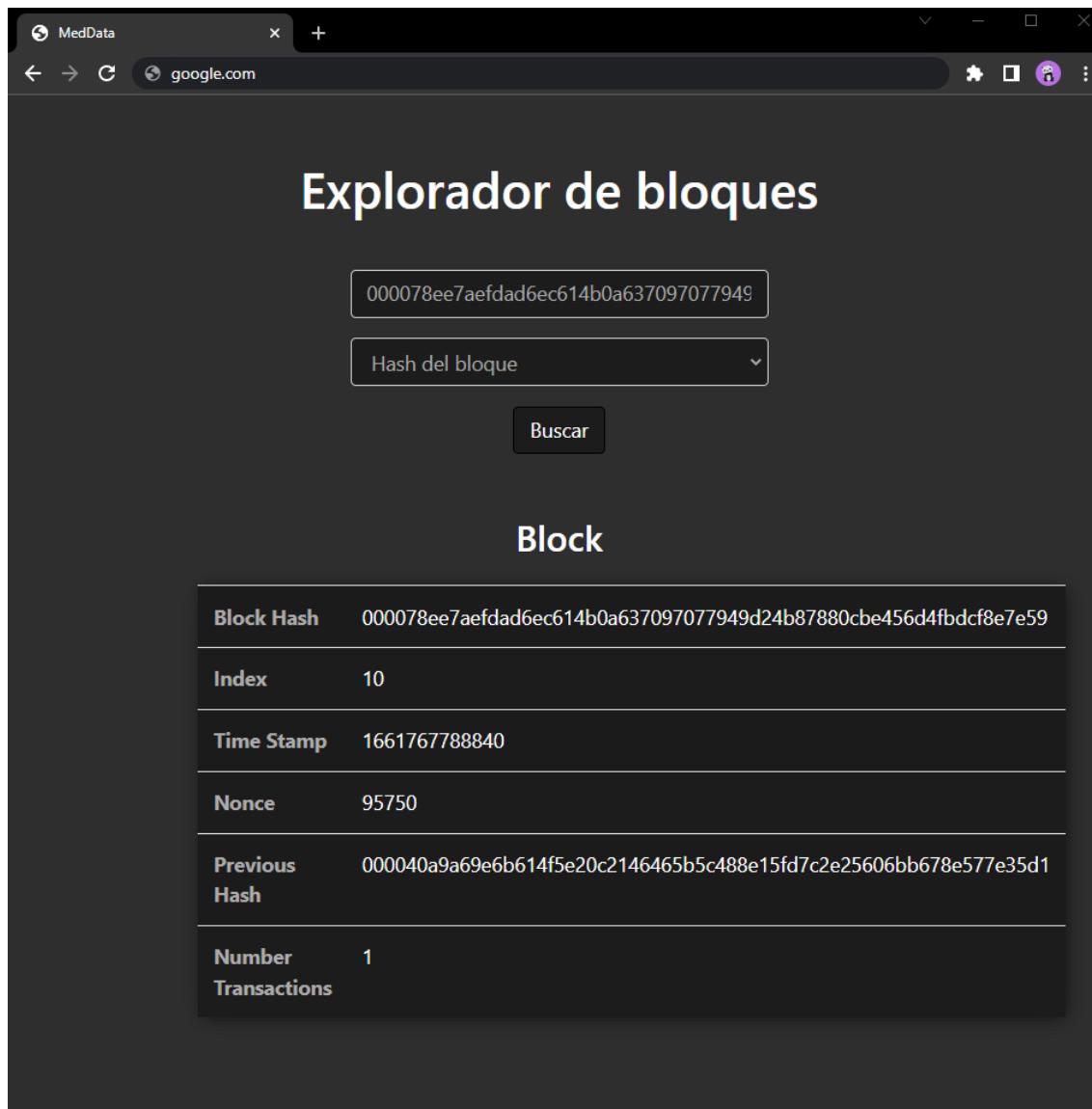


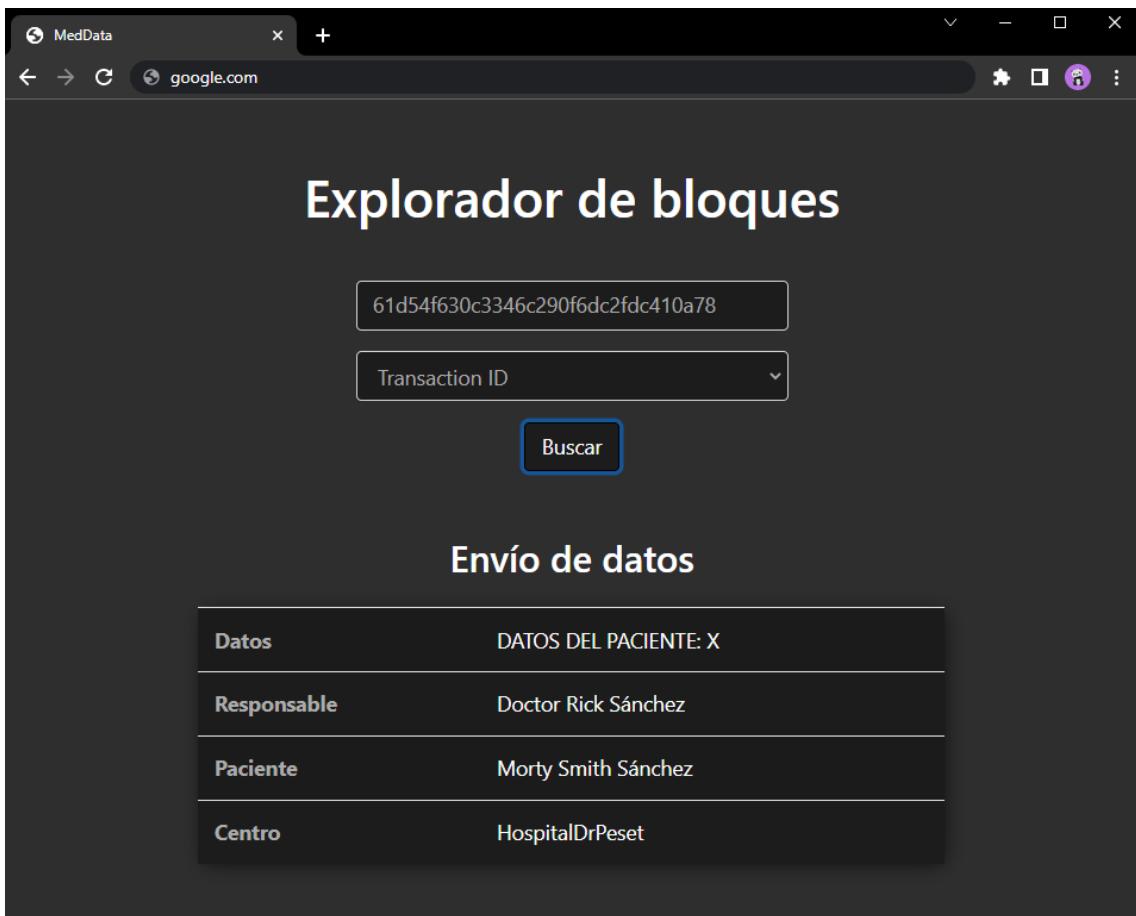
Figura 4.12 : Ejecución opción Hash del bloque

### Transaction ID

Esta opción del desplegable nos permitirá acceder a los datos contenidos en una transacción simplemente indicando su identificador (TransUUID).

/transaction/:transactionId

GET



**Figura 4.13 :** Ejecución opción Transaction ID

## Dirección

Con este *endpoint* se busca obtener todos los datos que han sido registrados desde un centro en concreto para poder obtener de forma rápida una lista de todos los movimientos de datos desde ese centro.

/address/:address

GET

The screenshot shows a dark-themed web application titled "Explorador de bloques". At the top, there are two input fields: "HospitalDrPresetOffice" and "Direccion", with a "Buscar" button below them. Below these fields is a section titled "Address" containing a table with four columns: "Responsable", "Paciente", "Centro", and "Fecha". The table lists five entries:

Responsable	Paciente	Centro	Fecha
Jim Halpert	23898947V	HospitalDrPresetOffice	2022-09-07T18:43:28.271Z
Diego Contreras	23898947V	HospitalDrPresetOffice	2022-09-07T18:46:02.108Z
Márton Hásch	23898947V	HospitalDrPresetOffice	2022-09-07T18:46:58.768Z
Michael Scott	23898933V	HospitalDrPresetOffice	2022-09-07T18:50:34.086Z

Figura 4.14 : Ejecución opción Dirección

## Responsable

Esta opción del menú desplegable nos permite obtener todos los pacientes tratados por un doctor, el centro en el que se llevó a cabo y los datos obtenidos y almacenados durante el tratamiento.

/doctor/:doctor

GET

The screenshot shows the same dark-themed web application. In the "Responsable" section, the input field contains "Doctor Rick Sánchez" and the dropdown menu shows "Responsable". A "Buscar" button is located below the input fields. Below this section is a table titled "Responsable" with three columns: "Datos", "Paciente", and "Centro". The table displays one row of data:

Datos	Paciente	Centro
DATOS DEL PACIENTE: X	Morty Smith Sánchez	HospitalDrPreset

Figura 4.15 : Ejecución opción Responsable

## Paciente

Esta opción del menú desplegable permite al usuario de la aplicación obtener todos los datos de un determinado paciente al introducir su identificador. Estos datos son, la información de la consulta, el responsable de dicha consulta y el centro desde el cual se almacenaron los datos.

En lugar de usar el nombre del paciente como identificador, este podría ser identificado por su DNI o su número SIP para lograr una mayor anonimidad. Otra opción interesante sería realizar una función hash de su nombre para que tuviera un identificador único y anónimo al mismo tiempo. Los mismos cambios son aplicables al doctor.

/patient/:patient

GET

The screenshot shows a dark-themed user interface titled 'Explorador de bloques'. At the top, there is a search bar containing the identifier '23898947V' and a dropdown menu set to 'Paciente'. Below these are two buttons: a blue 'Buscar' button and a smaller grey 'Cancelar' button. The main area is titled 'Paciente' and contains a table with four columns: 'Datos', 'Responsable', 'Centro', and 'Fecha'. There are three rows of data, each representing a patient record:

Datos	Responsable	Centro	Fecha
Datos del paciente: ++++++	Jim Halpert	HospitalDrPresetOffice	2022-09-07T18:43:28.271Z
Datos del paciente: ++++++	Diego Contreras	HospitalDrPresetOffice	2022-09-07T18:46:02.108Z
Datos del paciente: ++++++	Márton Hársch	HospitalDrPresetOffice	2022-09-07T18:46:58.768Z

**Figura 4.16:** Ejecución opción Paciente

## Imágenes

Gracias a esta opción podemos obtener la imagen que deseemos desde la interfaz simplemente indicando el hash con el que fue almacenada.

/images/:imagen

GET



Figura 4.17: Ejecución opción Imágenes

#### 4.4. Tecnología Utilizada

Como ya se ha aclarado con anterioridad, la solución planteada para el almacenamiento de datos médicos pasa por la implementación de una cadena de bloques en la que poder almacenar estos de forma eficiente y segura. Para esta solución se ha decidido emplear los siguientes componentes: una API con la que poder interactuar, una cadena de bloques para poder almacenar los datos, una base de datos en la que poder almacenar esta cadena de bloques y por último una serie de nodos que nos permitan hacer uso de esta cadena. Para construir esta solución, se ha optado por utilizar las siguientes tecnologías:

1. **JavaScript**: la selección de este lenguaje de programación no ha sido casual, si bien tal vez no sea el más adecuado debido a la ausencia de tipado y no se suela recomendar para aplicaciones de este tipo decantándose normalmente por lenguajes con tipado como Golang o Typescript, dada la naturaleza demostrativa de la implementación, la cercanía de este lenguaje y la gran variedad de documentación y soporte en internet de **Node.js**, han hecho que este lenguaje se convirtiera en la mejor opción para desarrollar tanto la cadena como la gestión de las peticiones.
2. **Express**: el uso de esta infraestructura para aplicaciones web Node.js ha venido marcada por la sencillez y facilidad que proporciona para la creación de APIs (Application Programming Interface) en Node.js. [23]
3. **MongoDB**: MongoDB es un sistema de bases de datos NoSQL, no relacional y de código abierto. En vez de almacenar los datos en tablas tal y como se haría en una base de datos relacional, MongoDB guarda estructuras de datos BSON(Binary JSON), que son una representación binaria de las estructuras de datos y los mapas, gracias a esto logra que la integración de aplicaciones sea rápida y sencilla, por esto y por la gran facilidad para crear conexiones y realizar peticiones entre MongoDB y Node.js que proporciona la librería

**Mongoose** para Javascript, se ha decidido emplear MongoDB como base de datos para almacenar los datos de nuestra cadena. [8]

4. **MongoDB Compass:** Es una interfaz gráfica de usuario o GUI que facilita la consulta, la agregación y el análisis de los datos de una base de datos en MongoDB. [7]
5. **Docker:** Docker es una plataforma de código abierto que permite la automatización del despliegue de aplicaciones dentro de contenedores software. En este caso se ha utilizado un contenedor docker para hospedar al servidor MongoDB en el que se almacena la cadena de bloques. [29]
6. **Postman:** Postman es una plataforma API para que los desarrolladores diseñen, construyan prueben e iteren sus API. En este proyecto ha sido de gran utilidad para la prueba y el testeo de todas los *endpoints* ofrecidos por nuestra API REST. [30]
7. **Github:** Github es un servicio basado en la nube que aloja un sistema de control de versiones (VCS) llamado Git. Este me ha permitido mostrar los cambios que se iban añadiendo, crear ramas del proyecto para no generar fallos en el principal y hacer un
8. **AngularJS:** AngularJS, es un framework de JavaScript de código abierto, mantenido por Google, este se utiliza para crear y mantener aplicaciones web de una sola página. En este proyecto se ha hecho un uso bastante rudimentario de esta herramienta con la que se podrían implementar multitud de funcionalidades distintas.
9. **HTML y CSS:** HTML es el lenguaje de marcado que se emplea para estructurar y dar significado a las páginas web. En nuestro caso se ha empleado de una manera bastante sencilla junto con Bootstrap para crear la interfaz con la que interactuar. Por otra parte, CSS es el lenguaje de reglas de estilo que hemos empleado para dar forma y color a la interfaz estructurada por HTML.

## 5. Desarrollo de la solución propuesta

---

Para el desarrollo de la solución propuesta ha sido necesario realizar la configuración de los siguientes aspectos:

### 5.1 Configuración de contenedores Docker

Para la puesta en marcha de un servidor mongoDB desde docker habría que ejecutar el siguiente comando desde el terminal de Windows en este caso:

```
docker run -dit --name mongo -p 27017:27017 -e MONGO_INITDB_ROOT_USERNAME=root  
-e MONGO_INITDB_ROOT_PASSWORD=root_pass mongo
```

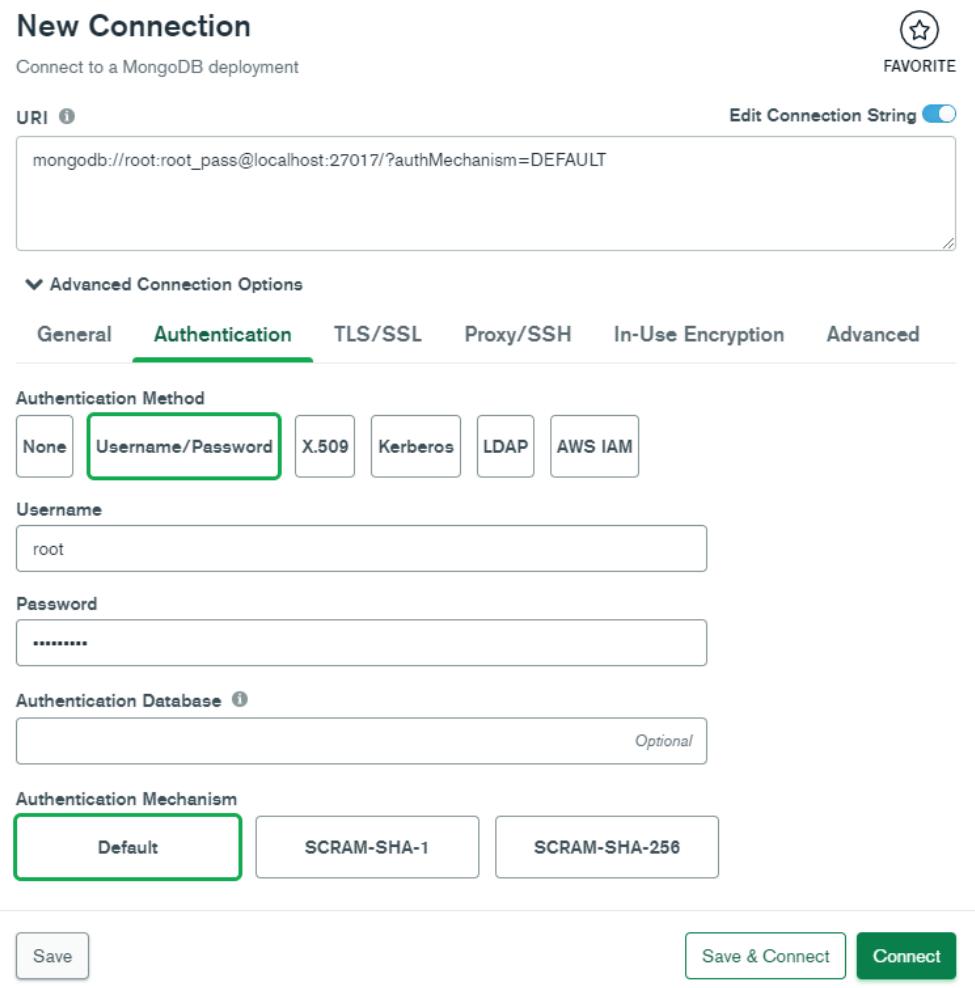
Indicando el número de puerto en el que deseamos que escuche el servidor, un nombre de usuario y una contraseña.

Si queremos crear más contenedores para hospedar más servidores mongoDB, lo cual será necesario ya que habrá un servidor por nodo, habría que repetir el comando cambiando el primer número de puerto al número deseado para reconducir las peticiones hacia este y cambiar el nombre del contenedor en nuestro caso mongo, por otro nombre de libre elección que no haya sido escogido anteriormente. También podemos elegir un nombre de usuario (`MONGO_INITDB_ROOT_USERNAME`) distinto y una contraseña nueva (`MONGO_INITDB_ROOT_PASSWORD`).

### 5.2 Configuración de servidores MongoDB

Para permitir el acceso restringido a la base de datos y evitar operaciones de actualización o eliminación ya que blockchain no sigue el patrón CRUD ( Create Read Update Delete) sino que solamente permite operaciones de inserción y de lectura, habrá que seguir una serie de pasos:

1. En primer lugar nos conectaremos, introduciendo el usuario y la contraseña usados en la creación del contenedor, al servidor que está corriendo en Docker desde MongoDB Compass.



**Figura 5.0 :** Conexión con MongoDB Compass

2. En segundo lugar crearemos una database desde el apartado *databases* y le daremos el nombre que prefiramos, Blockchain para este ejemplo.
3. Una vez creada la database, abriremos el terminal integrado de MongoDB Compass e introduciremos el comando `> use Blockchain` para acceder al ámbito de esa base de datos.
4. Una vez en el ámbito de nuestra base de datos, procederemos a crear los nuevos usuarios que podrán acceder a esta y realizar operaciones. También añadiremos una serie de roles para estos para permitir la creación, la lectura de bloques y el borrado de bloques desde la API. Para ello ejecutamos los siguientes comandos:

```
a. db.createRole({role:"test", privileges:
  [ {resource:{db:"Blockchain", collection:
  "blockschemas"}, actions:[ "find", "insert"] }})

b. db.createUser({ user: "usuario",
```

```
        pwd: "contraseña",
        roles:[{role: "test" ,
        db:"Blockchain"}] })
```

### 5.3 Partes relevantes del código

En este apartado se muestran fragmentos de la implementación de las partes más cruciales del programa junto a una pequeña explicación teórica de estas implementaciones.

Cómo hay una gran cantidad de código a analizar, esta sección estará estructurada en los distintos ficheros utilizados para crear la aplicación.

Se puede acceder al código de la aplicación a través del siguiente enlace [\*montoliws/BlockChain\\_Thesis \(github.com\)\*](https://montoliws/BlockChain_Thesis (github.com))

#### 5.3.1 blockchain.js

En este fichero se encuentran implementadas la mayor parte de las funcionalidades básicas de la cadena, estas funciones serán llamadas desde el fichero *netNode.js* que realiza las funciones de API.

##### Función hashBlock

*blockchain.js*

```
Blockchain.prototype.hashBlock = function (
    previousBlockHash,
    currentBlockData,
    nonce
) {
    const dataAsString = previousBlockHash + nonce.toString() +
    JSON.stringify(currentBlockData);
    const hash = sha256(dataAsString);
    return hash;
};
```

Fragmento de código 3: *blockchain.js*

Lo que realizará este método, será tomar un bloque de la cadena y realizar la función hash sobre sus datos para devolver una cadena o *string* de longitud fija. Para ello obtendremos concretamente los valores `previousBlockHash`, `currentBlockData` y el `nonce` del bloque del que vamos a obtener el nuevo hash y los introducimos en la función de hashing sha256 que ya ha sido explicada con anterioridad en el apartado **Función Hash y SHA-256** dentro del punto 2.2. Herramientas para la construcción de Libros de Registros Distribuidos Blockchain.

### Función proofOfWork

*blockchain.js*

```
Blockchain.prototype.proofOfWork = function (
    previousBlockHash,
    currentBlockData
) {
    let nonce = 0;

    let hash = this.hashBlock(previousBlockHash,
        currentBlockData, nonce);

    while (hash.substring(0, 4) !== "0000") {
        nonce++;

        hash = this.hashBlock(previousBlockHash,
            currentBlockData, nonce);
    }

    return nonce;
};
```

Fragmento de código 4: *blockchain.js*

Los aspectos del funcionamiento de esta función ya han sido explicados detalladamente en el apartado **Función Hash y SHA-256** dentro del punto 2.2. Herramientas para la construcción de Libros de Registros Distribuidos Blockchain.

En esta implementación se pueden observar con facilidad todos los atributos de una función `proofOfWork`, el contador o `nonce` y el bucle `while` encargado de realizar la llamada al método `hashBlock` hasta conseguir una cadena o *string* que comience por la cantidad de 0s indicada, 4 en nuestro caso, la razón de la elección de esta cantidad ha sido por razones de ahorro de energía y tiempo a la hora de realizar el *debugging*, si se desea aumentar la seguridad de la red, aumentar el número de 0s al comienzo

del *string*, aumentaría el coste computacional y pondría más dificultades a los nodos que deseasen tomar el control de la red.

### Función chainValidation

*blockchain.js*

```
Blockchain.prototype.chainValidation = function (blockchain) {  
    let cadenaValida = true;  
  
    for (var i = 1; i < blockchain.length; i++) {  
  
        const bloquePrev = blockchain[i - 1];  
  
        const bloqueAct = blockchain[i];  
  
        const hashBlockconst =this.hashBlock(  
            bloquePrev["hash"],  
            { transactions: bloqueAct["transactions"], index:  
                bloqueAct["index"] },  
            bloqueAct["nonce"]  
        );  
  
        if (hashBlockconst.substring(0, 4) !== "0000") {  
            cadenaValida = false;  
        }  
  
        if (bloqueAct["previousBlockHash"] !== bloquePrev["hash"]) {  
            cadenaValida = false;  
        }  
    }  
  
    const genesisBlck = blockchain[0];  
  
    const transactionsCheck = genesisBlck["transactions"].length  
    === 0;  
  
    const nonCheck = genesisBlck["nonce"] === 58;  
  
    const previousBlockHashCheck =  
        genesisBlck["previousBlockHash"] === "0";  
  
    const hashCheck = genesisBlck["hash"] === "0";  
  
    if (
```

```

    transactionsCheck !== true ||
    nonCheck !== true ||
    previousBlockHashCheck !== true ||
    hashCheck !== true
  ) {

  cadenaValida = false;
}

return cadenaValida;
};

```

**Fragmento de código 5:** blockchain.js

Este método validará si una cadena es válida o no. Vamos a utilizar este método, `chainValidation` para validar las otras cadenas dentro de la red cuando las comparamos con la cadena alojada en el nodo actual. Para validar que la cadena de bloques es legítima, simplemente vamos a iterar a través de cada bloque dentro de la cadena de bloques y verificar si todos los hash se alinean correctamente.

Iteramos en toda la cadena de bloques comprobando dos cosas básicamente:

- Primero comprobar que todos los hashes están ordenados de forma adecuada y que el hash del bloque anterior realmente coincide con el valor almacenado.
- Segundo, hacemos hash de cada bloque y comprobamos que realmente el hash del bloque comienza por cuatro ceros. Si no es así, entonces indicamos que la cadena es inválida.
- Finalmente comprobamos que las propiedades del bloque génesis, el bloque inicial, sean las correctas.

### 5.3.2 model.js

En este fichero se crea el esquema que deberán seguir los bloques para poder ser almacenados dentro de la base de datos de MongoDB y se exporta este modelo con el nombre de `BlockSchema` para poder hacer uso de él en el resto de ficheros que componen la aplicación

```
let BlockchainSchema = new Schema({
  index: {
    required: true,
    type: Schema.Types.Number,
  },
  timestamp: {
    required: true,
    type: Schema.Types.Date,
    default: Date().now,
  },
  transactions: {
    required: true,
    type: Schema.Types.Array,
  },
  nonce: {
    required: true,
    type: Schema.Types.Number,
  },
  previousBlockHash: {
    required: false,
    type: Schema.Types.String,
  },
  hash: {
    required: true,
    type: Schema.Types.String,
  },
}) ;
module.exports = mongoose.model(
  "BlockSchema",
  BlockchainSchema,
  "blockschemas"
) ;
```

**Fragmento de código 6: model.js**

### 5.3.3 netNode.js

En este fichero se encuentran todos los *endpoints* que permiten al usuario interactuar con nuestra cadena de bloques. Todos estos *endpoints* han sido implementados gracias a la librería Express, que será necesario importar al comienzo.

Al tratarse de un fichero de más de trescientas líneas se explicarán brevemente algunas de las implementaciones más relevantes y significativas, dejando al lector la posibilidad de indagar de manera más profunda en el resto del código por su cuenta.

### Función /mine

*netNode.js*

```
app.get("/mine", async function (req, res) {

  const chain = await blockchainModel.find();

  meddata.chain = chain;

  const requestPromises = [];

  if (meddata.chain.length == 0) {

    const genesis = meddata.createNewBlock(58, "0", "0");

    let newBlockGen = new blockchainModel(genesis);

    newBlockGen = await newBlockGen.save((err) => {

      if (err)

        return console.log(chalk.red("cannot save the block",
err.message));

      console.log(chalk.green("Block saved on DB"));

    });

  }

  await meddata.networkNodes.forEach((networkNodeUrl) => {

    const requestOptions = {

      uri: networkNodeUrl + "/receive-new-block",

      method: "POST",

      body: { newBlock: genesis },

      json: true,

    };

  });

});
```

```
    requestPromises.push(rp(requestOptions)) ;

}) ;

}

const lastBlock = meddata.chain[meddata.chain.length - 1] ;
const previousBlockHash = lastBlock["hash"] ;

const currentBlockData = {
  transactions: meddata.pendingTransactions,
  index: lastBlock["index"] + 1,
} ;

const nonce = meddata.proofOfWork(previousBlockHash,
currentBlockData) ;

const newBlockHash = meddata.hashBlock(
  previousBlockHash,
  currentBlockData,
  nonce
) ;

const newBlock = meddata.createNewBlock(
  nonce,
  previousBlockHash,
  newBlockHash
) ;

let newBlockMod = new blockchainModel(newBlock) ;
console.log(newBlockMod) ;

newBlockMod = await newBlockMod.save((err) => {
  if (err)
```

```

        return console.log(chalk.red("cannot save the block",
err.message));

    console.log(chalk.green("Block saved on DB"));

}) ;

await meddata.networkNodes.forEach((networkNodeUrl) => {

const requestOptions = {

uri: networkNodeUrl + "/receive-new-block",

method: "POST",

body: { newBlock: newBlock },

json: true,

};

requestPromises.push(rp(requestOptions));

}) ;

Promise.all(requestPromises).then((data) => {

res.json({

note: "Se ha completado el minado de un nuevo nodo",

block: newBlock,

}) ;

}) ;

}) ;

```

#### Fragmento de código 7: netNode.js

Esta extensa función que nos permite la creación y adhesión de un nuevo bloque a nuestra cadena se ejecuta cuando algún nodo realice una petición GET al endpoint /mine.

Lo primero que se realiza es una actualización de la cadena llamando al método .find() que nos proporciona mongoose. Esto cargará todos los bloques que hay almacenados hasta ese instante en la cadena de MongoDB en la cadena con la que estamos utilizando en ese momento.

Una vez actualizada, si la cadena está vacía, i.e. estamos empezando de cero, se añadirá un primer bloque automáticamente, el bloque Génesis. Si por el contrario, ya se habían almacenado bloques en esta cadena, añadiremos el bloque siguiendo los siguientes pasos:

1. Obtenemos el hash del bloque previo y los datos a almacenar.
2. Obtenemos el nonce llamando a la función proofOfWork

### Función /register-and-broadcast-node

*netNode.js*

```
app.post("/register-and-broadcast-node", function (req, res) {  
  
    const newNodeUrl = req.body.newNodeUrl;  
  
    if (meddata.networkNodes.indexOf(newNodeUrl) == -1)  
        meddata.networkNodes.push(newNodeUrl);  
  
  
    const regNodesPromises = [];  
  
    meddata.networkNodes.forEach((networkNodeUrl) => {  
  
        const requestOptions = {  
  
            uri: networkNodeUrl + "/register-node",  
            method: "POST",  
            body: { newNodeUrl: newNodeUrl },  
            json: true,  
        };  
  
  
        regNodesPromises.push(rp(requestOptions));  
    });  
  
  
    Promise.all(regNodesPromises)  
        .then((data) => {  
  
            const bulkRegisterOptions = {  
  
                uri: newNodeUrl + "/register-nodes-bulk",  
                method: "POST",  
                body: {  
  
                    allNetworkNodes: [...meddata.networkNodes,  
meddata.currentNodeUrl],  
  
                },  
            },  
        );  
});
```

```

        json: true,
    };

    return rp(bulkRegisterOptions);
}

.then((data) => {
    res.json({
        note: "El nuevo nodo ha sido correctamente registrado en la red.",
    });
});

})
;
}

```

#### Fragmento de código 8: netNode.js

Como se puede apreciar, esta función se ejecutará al recibir una petición post en el servidor en la que recibiremos la dirección del nodo que desea ser registrado en la red. Una vez recibida la petición se llamará al endpoint `/register-node` en cada uno de los nodos que conforman la red. Este endpoint registrará de forma individual en cada nodo al nuevo nodo.

El método `Promise.all(regNodesPromises)` devuelve una promesa que termina correctamente cuando todas las promesas en el argumento iterable (`regNodesPromises`) han sido concluídas con éxito, o bien rechaza la petición con el motivo pasado por la primera promesa que es rechazada.

Si todas las promesas son resueltas entraríamos entonces dentro del `.then()` que ejecutaría la llamada a `/register-nodes-bulk` con la dirección del nuevo nodo que quería pasar a formar parte de la red para que este registre a su vez a todos los nodos que ya formaban parte de la red.

La variable `data` son los datos que recibimos de la promesa de arriba. No vamos a utilizar estos datos pero hay que hacer este paso dentro de este endpoint por lo que solo podemos hacerlo una vez la promesa se ha completado.

#### 5.3.4 index.js

En este fichero se introducen tanto el usuario como la contraseña para acceder a la base de datos creada en MongoDB y posteriormente nos conectamos haciendo uso de la función

mongoose.connect("mongodb://localhost:27017/Blockchain", options);  
e indicando los datos del usuario introducidos anteriormente en el apartado **options**.

```
let mongoose = require("mongoose");

let BlockchainModel = require("./model");

const options = {

  auth: {

    username: "user",

    password: "PASSWORD",

  },

};

mongoose.connect("mongodb://localhost:27017/Blockchain", options);

let connectionCallback = () => {};

module.exports.onConnect = (callback) => {

  connectionCallback = callback;

};
```

Fragmento de código 9: index.js

## 5.4 Codificación de imágenes

Para poder añadir imágenes a la información de la cadena, es necesario transformar estas a texto para poder introducirlas dentro del array `images`, para esto se ha utilizado el formato Base64. Base64 es un grupo de esquemas de codificación de binario a texto que representa los datos binarios por medio de una cadena ASCII.

Los esquemas de codificación Base64 son comúnmente usados cuando se necesita codificar datos binarios para que sean almacenados y transferidos sobre un medio diseñado para tratar con datos textuales como es nuestro caso. Esto es para asegurar que los datos se mantienen intactos y sin modificaciones durante la transmisión. Base64 es utilizado en una amplia gama de aplicaciones, como MIME o OpenPGP.

Para codificar las imágenes que queremos introducir en la cadena, podemos utilizar una de las distintas herramientas online, en nuestro caso, hemos utilizado [Base64 Decode and Encode - Online](#) que nos facilita tanto la codificación de la imagen como su decodificación.

Una vez obtenido el texto a partir de la imagen, cada vez que se mine un nuevo bloque con una transacción que contenga la imagen, se introducirá este texto en la base de datos específica de las imágenes. Para obtener la imagen que finalmente se mostrará en la aplicación web, deberemos llamar en primer lugar a la base de datos para obtener las imágenes codificadas y después introducir la siguiente línea de código para volver a transformarlas en .png de nuevo y poder mostrarlas:

```
</div>
```

**Fragmento de código 10:** index.html

## 6. Pruebas

---

A lo largo de este apartado se muestran las pruebas realizadas para verificar el correcto funcionamiento de la aplicación. Aparte de esto, también se exponen los resultados de distintas pruebas llevadas a cabo con el objetivo de medir la eficiencia de la aplicación.

### 6.1. Pruebas de uso

En este apartado se muestran, la puesta en marcha de todos los nodos y servidores, y la ejecución de gran parte de las funcionalidades de la cadena junto las convenientes comprobaciones para cerciorarnos del correcto comportamiento de la cadena.

Para llevar a cabo las pruebas de uso, se ha instalado una máquina virtual Ubuntu en Oracle VM VirtualBox con el fin de realizar pruebas con nodos hospedados en máquinas remotas. A esta máquina se le ha asignado una IP propia y se le ha permitido establecer conexiones con la máquina anfitrión desde los ajustes de red.

#### 6.1.1 Puesta en marcha de servidores y nodos

Desde el fichero *package.json* vamos a gestionar las conexiones con el objeto *scripts*, en este vamos a crear los distintos nodos que vamos a poner en ejecución.

```
"scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1",  
    "node_1": "nodemon --watch dev -e js BEG/netNode.js 3001  
http://localhost:3001",  
    "node_2": "nodemon --watch dev -e js BEG/netNode.js 3002  
http://localhost:3002",  
    "node_3": "nodemon --watch dev -e js BEG/netNode.js 3003  
http://localhost:3003",  
    "node_4": "nodemon --watch dev -e js BEG/netNode.js 3004  
http://localhost:3004",  
    "node_5": "nodemon --watch dev -e js BEG/netNode.js 3005  
http://localhost:3005"  
},
```

Fragmento de código 11: package.json

Para arrancar los distintos nodos ejecutaremos el comando `npm run node_X` desde la consola de node y en X indicamos el número del nodo que queremos lanzar.

Antes de lanzar todos estos nodos, debemos poner en marcha desde Docker todos los servidores mongo en los que vamos a almacenar tanto la cadena como las imágenes.

Como cada nodo debe tener su propia base de datos en la que almacenar la cadena de manera aislada al resto y vamos a llevar a cabo estas pruebas utilizando solamente tres nodos, vamos a crear tres contenedores para hospedar las distintas bases de datos y asignamos a cada uno un número de puerto distinto para acceder a través de este a cada base de datos por separado.

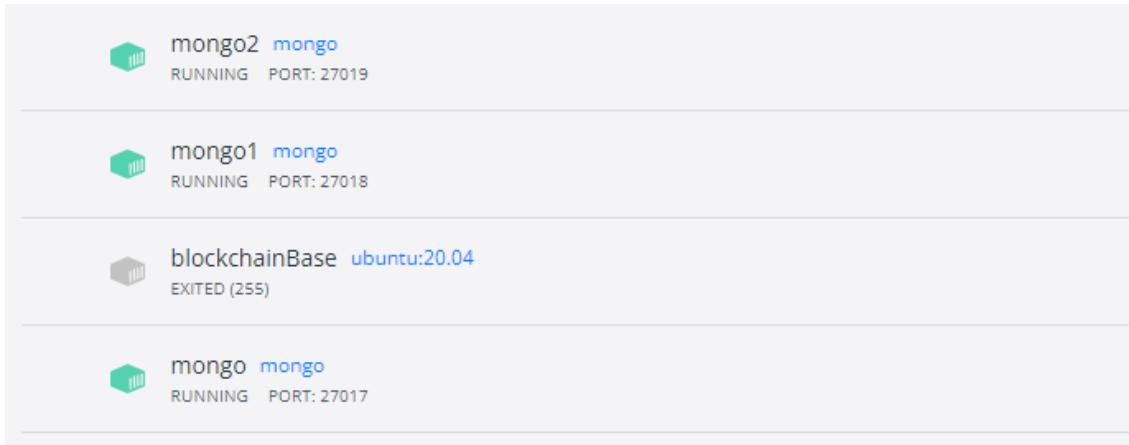


Figura 6.0 : contenedores docker

Una vez puestos en marcha los tres contenedores con sus respectivas bases de datos ya configuradas (consultar de nuevo apartados **5.1 Configuración de contenedores Docker** y **5.2 Configuración de servidores MongoDB**) procedemos a lanzar desde nuestro ordenador principal los nodos 1 y 3 (node\_1 y node\_3).

Para ello abrimos un primer terminal y ejecutamos la orden `npm run node_1`:

```
PS C:\Users\Monto\Documents\GitHub\BlockChain_Thesis\BEG> npm run node_1
> devel@1.0.0 node_1 C:\Users\Monto\Documents\GitHub\BlockChain_Thesis\BEG
> nodemon --watch dev -e js BEG/netNode.js 3001 http://localhost:3001

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): dev
[nodemon] watching extensions: js
[nodemon] starting `node BEG/netNode.js 3001 http://localhost:3001 netNode.js`
Aplicación escuchando en el puerto: 3001
```

Figura 6.1 : Puesta en marcha nodo 1

Una vez puesto en marcha el primer nodo, antes de poner en marcha el segundo, es importante cambiar la dirección url a la que se conectará nuestro cliente de MongoDB ya que cada máquina cuenta con su propia base de datos aislada. Para esto cambiamos la dirección url de la conexión anterior por la url nueva cambiando la IP o el número de puerto, en nuestro caso, al estar hospedados en contenedores Docker podemos cambiar únicamente el número de puerto.

```
mongoose.connect("mongodb://localhost:27017/Blockchain", options);
```

Fragmento de código 12: index.js

```
mongoose.connect("mongodb://localhost:27018/Blockchain", options);
```

Fragmento de código 13: index.js

Una vez realizados estos cambios abrimos un segundo terminal desde el mismo ordenador y ejecutamos la orden `npm run node_2`:

```
PS C:\Users\Monto\Documents\GitHub\BlockChain_Thesis\BEG> npm run node_2
> devel@1.0.0 node_2 C:\Users\Monto\Documents\GitHub\BlockChain_Thesis\BEG
> nodemon --watch dev -e js BEG/netNode.js 3002 http://localhost:3002

[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): dev
[nodemon] watching extensions: js
[nodemon] starting `node BEG/netNode.js 3002 http://localhost:3002 netNode.js`
Aplicación escuchando en el puerto: 3002
```

Figura 6.2 : Puesta en marcha nodo 2

Después de haber arrancado estos dos nodos, vamos a arrancar un tercero desde la máquina virtual. En este caso será necesario cambiar la IP por la del ordenador principal ya que Docker se está ejecutando en este último. Además de esto al igual que en el resto de casos también será necesario cambiar el número de puerto.

```
mongoose.connect("mongodb://192.168.1.164:27019/Blockchain", options);
```

Fragmento de código 14: index.js

Una vez cambiados tanto la IP como el número de puerto, iniciamos el nodo desde consola.

```
scipio@scipio-VirtualBox:~/BEG$ npm run node_3
> devel@1.0.0 node_3 /home/scipio/BEG
> nodemon --watch dev -e js BEG/netNode.js 3003 http://localhost:3003

[nodemon] 2.0.19
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): dev
[nodemon] watching extensions: js
[nodemon] starting `node BEG/netNode.js 3003 http://localhost:3003 netNode.js`
Aplicación escuchando en el puerto: 3003
```

Figura 6.3: Puesta en marcha nodo 3

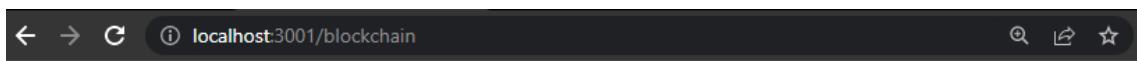
Una vez están todos los nodos en ejecución, podemos proceder a realizar las pruebas pertinentes sobre la cadena.

### 6.1.2 Pruebas de ejecución

La primera prueba que podemos llevar a cabo es comprobar el correcto funcionamiento de la red y la sincronización entre los nodos.

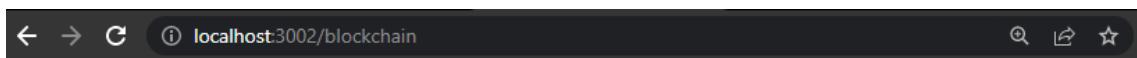
Para ello, partiremos de una nueva cadena e iremos realizando distintas operaciones sobre esta desde los distintos nodos.

Antes de comenzar a realizar operaciones, vamos a observar el estado de la cadena en cada uno de los nodos. Como se puede observar en la imágenes a continuación, el array `networkNodes` está vacío, esto nos indica que los nodos aún no están conectados entre sí.



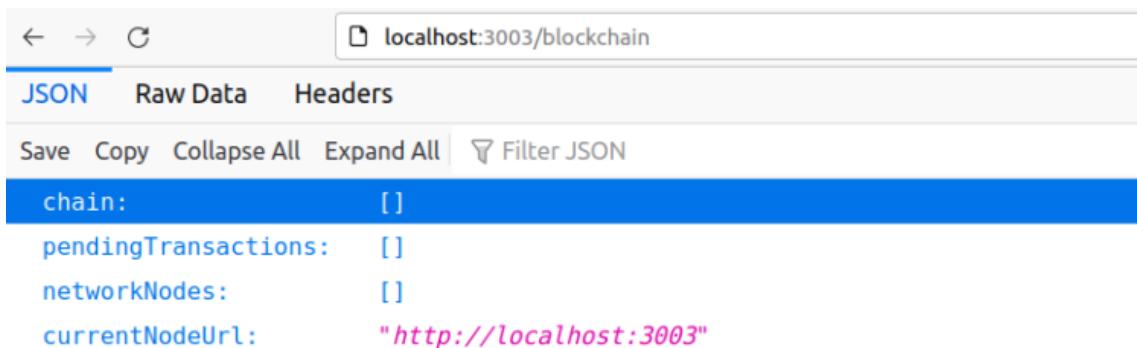
```
{"chain":[],"pendingTransactions":[],"pendingImages":[],"networkNodes":[],"currentNodeUrl":"http://localhost:3001"}
```

**Figura 6.4:** Cadena inicial nodo 1



```
{"chain":[],"pendingTransactions":[],"pendingImages":[],"networkNodes":[],"currentNodeUrl":"http://localhost:3002"}
```

**Figura 6.5:** Cadena inicial nodo 2



```
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
chain: []
pendingTransactions: []
networkNodes: []
currentNodeUrl: "http://localhost:3003"
```

**Figura 6.6:** Cadena inicial nodo 3

En primer lugar, vamos a crear la red con los tres nodos que hay en funcionamiento llamando en uno de ellos al endpoint `/register-and-broadcast-node` indicando el nuevo nodo que se desea añadir. Esta llamada habrá que realizarla tantas veces como nodos queramos añadir, dos en nuestro caso.

Añadimos en primera instancia el nodo que se encuentra en la máquina virtual:

## Implementación de un sistema básico de Blockchain y su aplicación en la trazabilidad de datos médicos

The screenshot shows a POST request to <http://localhost:3001/register-and-broadcast-node>. The Body tab is selected, showing a JSON payload with a single key-value pair: "newNodeUrl": "http://192.168.56.111:3003". The response status is 200 OK, and the note says "El nuevo nodo ha sido correctamente registrado en la red."

Figura 6.7: Registro nodo 3 en la red

Realizamos esta misma operación indicando la url del nodo 2.

The screenshot shows a POST request to <http://localhost:3001/register-and-broadcast-node>. The Body tab is selected, showing a JSON payload with a single key-value pair: "newNodeUrl": "http://localhost:3002". The response status is 200 OK, and the note says "El nuevo nodo ha sido correctamente registrado en la red."

Figura 6.8: Registro nodo 2 en la red

Una vez registrados ambos nodos, podemos comprobar si ha funcionado correctamente recargando la página <http://localhost:300X/blockchain> en cualquiera de los nodos.

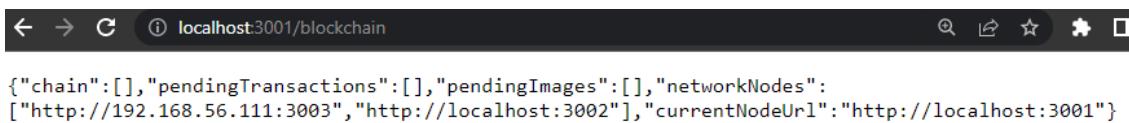


Figura 6.9: Comprobación registro nodos

Una vez comprobado que la red está en funcionamiento y que los nodos están conectados entre sí podemos empezar a realizar operaciones sobre la cadena.

Para comprobar la correcta sincronización de los nodos, la primera operación que vamos a llevar a cabo va a ser el minado de un bloque. Al ser este el primer bloque

que vamos a añadir a la cadena, se añadirán dos bloques en lugar de uno, el bloque génesis o inicial, con hash igual a 0 y nonce igual a 58, y el bloque que hemos minado que tendrá como atributo previousHash un 0 ya que el hash del primer bloque es 0.

Ejecutamos la orden /mine y observamos los resultados:



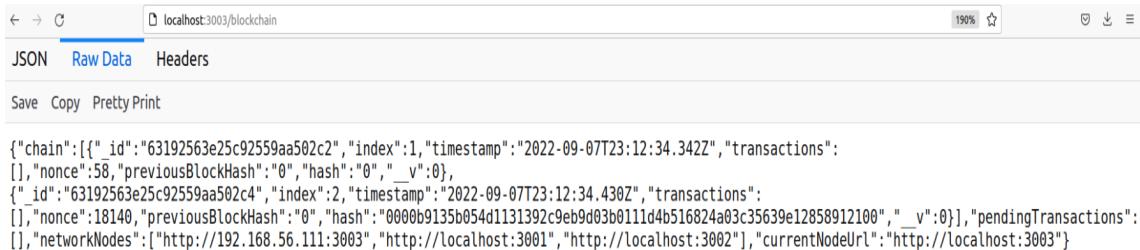
```
{"chain": [{"_id": "63192562d8c5297a1ef36d46", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": [], "nonce": 58, "previousBlockHash": "0", "hash": "0", "v": 0}, {"_id": "63192562d8c5297a1ef36d47", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": [], "nonce": 18140, "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "v": 0}], "pendingTransactions": [], "pendingImages": [], "networkNodes": [{"http://192.168.56.111:3003", "http://localhost:3002"}, {"currentNodeUrl": "http://localhost:3001"}]}
```

**Figura 6.10:** Comprobación distribución bloques nodo 1



```
{"chain": [{"_id": "63192562afec816f73663606", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": [], "nonce": 58, "previousBlockHash": "0", "hash": "0", "v": 0}, {"_id": "63192562afec816f73663608", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": [], "nonce": 18140, "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "v": 0}], "pendingTransactions": [], "pendingImages": [], "networkNodes": [{"http://192.168.56.111:3003", "http://localhost:3001"}, {"currentNodeUrl": "http://localhost:3002"}]}
```

**Figura 6.11:** Comprobación distribución bloques nodo 2



```
localhost:3003/blockchain
JSON Raw Data Headers
Save Copy Pretty Print

{"chain": [{"_id": "63192563e25c92559aa502c2", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": [], "nonce": 58, "previousBlockHash": "0", "hash": "0", "v": 0}, {"_id": "63192563e25c92559aa502c4", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": [], "nonce": 18140, "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "v": 0}], "pendingTransactions": [], "networkNodes": [{"http://192.168.56.111:3003", "http://localhost:3001", "http://localhost:3002"}, {"currentNodeUrl": "http://localhost:3003"}]}
```

**Figura 6.12:** Comprobación distribución bloques nodo 3

Como se puede observar, la red está correctamente coordinada y los bloques minados se distribuyen satisfactoriamente por toda la red. La siguiente comprobación que vamos a realizar es que las transacciones se añaden por igual a todas las cadenas.

Para ello crearemos una transacción con postman, la enviaremos y después realizaremos el minado del nuevo bloque en el que se almacenará esta transacción.

```
{ "chain": [{"_id": "63192562d8c5297a1ef36d46", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": [], "nonce": 58, "previousBlockHash": "0", "hash": "0", "v": 0}, {"_id": "63192562d8c5297a1ef36d47", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": [], "nonce": 18140, "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "v": 0}, {"_id": "631925ded8c5297a1ef36d4d", "index": 3, "timestamp": "2022-09-07T23:14:38.558Z", "transactions": [{"data": {"Datos del paciente: -----", "address": "HospitalDrPesonOffice", "responsable": "Michael Scott", "paciente": "23898933V", "imageUniHash": "3d2a4442d56e0ce1de0608d5cd89d75d0a1f622a9010fc4b164fbfaa48cc52c"}, "imagesHash": ["0c443a986602d976dc802fc2112b09b2418bd456f164de87d964e3126595021"]}], "transUUID": "1ed9d92f9fd14cb28cb2814e735d7cd1", "timestamp": "2022-09-07T23:14:33.397Z"}], "nonce": 81703, "previousBlockHash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "hash": "00006d638b2875241737713ec3fa2a10e6fc07e360bc326f0b6a1174b3a6aa4", "v": 0}], "pendingTransactions": [], "pendingImages": [], "networkNodes": [{"http://192.168.56.111:3003", "http://localhost:3002"}, {"currentNodeUrl": "http://localhost:3001"}]}
```

**Figura 6.13:** Comprobación distribución transacción nodo 1

# Implementación de un sistema básico de Blockchain y su aplicación en la trazabilidad de datos médicos

```
{"chain": [{"_id": "63192562afec816f73663606", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": [], "nonce": 58, "previousBlockHash": "0", "hash": "0", "v": 0}, {"_id": "63192562afec816f73663608", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": []}, {"_id": "18140", "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "v": 0}, {"_id": "631925deafe816f7366360c", "index": 3, "timestamp": "2022-09-07T23:14:38.558Z", "transactions": [{"data": {"datos": "Datos del paciente: -----", "address": "HospitalDrPresetOffice", "responsable": "Michael Scott", "paciente": "23898933V", "imageUniHash": ["3d2a4442d56e0ce1de0608d5dc89d75d0a1f622a9010fc4b164fbfaa48cc52c"]}, "imagesHash": ["0c443a986602d976dc802fc2112b09b2418bd456f164de87d964e31265959021"}]}, {"transUUID": "1ed9d92f9fd14cb28cb2814e735d7cd1", "timestamp": "2022-09-07T23:14:33.397Z"}], "nonce": 81703, "previousBlockHash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "hash": "00006d63d8b2875241737713ec3fa2a10e6fc07e360bc326f0b6a1174b3a6aa4", "v": 0}], "pendingTransactions": [], "pendingImages": [], "networkNodes": [{"http://192.168.56.111:3003", "http://localhost:3001"}, {"currentNodeUrl": "http://localhost:3002"}]
```

**Figura 6.14:** Comprobación distribución transacción nodo 2

```
{"chain": [{"_id": "63192563e25c92559aa502c2", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": []}, {"_id": "63192563e25c92559aa502c4", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": []}, {"_id": "18140", "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "v": 0}, {"_id": "631925df25c92559aa502c8", "index": 3, "timestamp": "2022-09-07T23:14:38.558Z", "transactions": [{"data": {"datos": "Datos del paciente: -----", "address": "HospitalDrPresetOffice", "responsable": "Michael Scott", "paciente": "23898933V", "imageUniHash": ["3d2a4442d56e0ce1de0608d5dc89d75d0a1f622a9010fc4b164fbfaa48cc52c"]}, "imagesHash": ["0c443a986602d976dc802fc2112b09b2418bd456f164de87d964e31265959021"}]}, {"transUUID": "1ed9d92f9fd14cb28cb2814e735d7cd1", "timestamp": "2022-09-07T23:14:33.397Z"}], "nonce": 81703, "previousBlockHash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "hash": "00006d63d8b2875241737713ec3fa2a10e6fc07e360bc326f0b6a1174b3a6aa4", "v": 0}], "networkNodes": [{"http://192.168.56.111:3003", "http://localhost:3001"}, {"http://localhost:3002", "http://localhost:3003"}, {"currentNodeUrl": "http://localhost:3003"}]
```

**Figura 6.15:** Comprobación distribución transacción nodo 3

Una vez comprobado el correcto funcionamiento tanto del minado de nuevos bloques como de la adhesión de nuevas transacciones, procederemos a eliminar un nodo y a seguir realizando operaciones con el resto, una vez haya sido alterado el estado de la cadena, volveremos a lanzar el nodo que estaba caído y a ejecutar el endpoint /consensus que aplicará la regla de la cadena más larga, sustituyendo forzosamente la cadena del nodo que había quedado desconectado por la cadena más larga dentro de la red.

Para ello, desconectamos el nodo 2, y llevamos a cabo el minado de nuevos bloques en los dos nodos restantes. Después de añadir dos nuevos bloques a la cadena mientras el nodo 2 estaba desconectado, volvemos a conectarlo, quedando la red en el siguiente estado:

```
{"chain": [{"_id": "63192562d8c5297a1ef36d46", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": []}, {"_id": "63192562d8c5297a1ef36d47", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": []}, {"_id": "18140", "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "v": 0}, {"_id": "631925ded8c5297a1ef36d4d", "index": 3, "timestamp": "2022-09-07T23:14:38.558Z", "transactions": [{"data": {"datos": "Datos del paciente: -----", "address": "HospitalDrPresetOffice", "responsable": "Michael Scott", "paciente": "23898933V", "imageUniHash": ["3d2a4442d56e0ce1de0608d5dc89d75d0a1f622a9010fc4b164fbfaa48cc52c"]}, "imagesHash": ["0c443a986602d976dc802fc2112b09b2418bd456f164de87d964e31265959021"}]}, {"transUUID": "1ed9d92f9fd14cb28cb2814e735d7cd1", "timestamp": "2022-09-07T23:14:33.397Z"}], "nonce": 81703, "previousBlockHash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "hash": "00006d63d8b2875241737713ec3fa2a10e6fc07e360bc326f0b6a1174b3a6aa4", "v": 0}, {"_id": "63192892d8c5297a1ef36d52", "index": 4, "timestamp": "2022-09-07T23:26:10.245Z", "transactions": []}, {"_id": "2949", "previousBlockHash": "00006d63d8b2875241737713ec3fa2a10e6fc07e360bc326f0b6a1174b3a6aa4", "index": 5, "timestamp": "2022-09-07T23:27:30.197Z", "transactions": []}, {"_id": "168726", "previousBlockHash": "0000db2facdfb1f49ceef8621299b6a3896b4c232c2599938f7a9a8bbc9c2880", "index": 6, "timestamp": "2022-09-07T23:27:30.197Z", "transactions": []}], "pendingTransactions": [], "pendingImages": [], "networkNodes": [{"caedc585c8fc15671078f6ad691fe308cebe535e76b5", "v": 0}], "currentNodeUrl": "http://localhost:3003"}]
```

**Figura 6.16:** Estado cadena en los nodos 1 y 3

```
{"chain": [{"_id": "63192562afec816f73663606", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": []}, {"nonce": 58, "previousBlockHash": "0", "hash": "0", "__v": 0}, {"_id": "63192562afec816f73663608", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": []}, {"nonce": 18140, "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "__v": 0}, {"_id": "631925deafec816f7366360c", "index": 3, "timestamp": "2022-09-07T23:14:38.558Z", "transactions": [{"data": {"datos": "Datos del paciente: -----", "address": "HospitalDrPresetOffice", "responsable": "Michael Scott", "paciente": "23898933V", "imageUniHash": ["3d2a4442d56e0ce1de008d5cd89d75d0a1f622a9010fc4b164fbfaa48cc52c"]}, "imagesHash": ["0c443a986602d976dc802fc2112b09b2418bd456f164de87d964e3126595021"]}], "transUUID": "1ed9d92f9fd14cb28cb2814e735d7cd1", "timestamp": "2022-09-07T23:14:33.397Z"}], "nonce": 81703, "previousBlockHash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "hash": "00006d63d8b2875241737713ec3fa2a10e6fc07e360bc326f0b6a1174b3a6aa4", "__v": 0}, "pendingTransactions": [], "pendingImages": [], "networkNodes": [], "currentNodeUrl": "http://localhost:3002"}
```

**Figura 6.17:** Estado cadena en el nodo 2

Cómo se puede observar, el nodo 2 ya no forma parte de los nodos que componen la red y además tiene dos bloques menos que los nodos 1 y 3. Para solucionar esto, volveremos a llamar al endpoint /register-and-broadcast-node y una vez vuelva a formar parte de la red, llamaremos al endpoint /consensus.

```
{"note": "La cadena actual ha sido reemplazada.", "chain": [{"_id": "63192563e25c92559aa502c2", "index": 1, "timestamp": "2022-09-07T23:12:34.342Z", "transactions": []}, {"nonce": 58, "previousBlockHash": "0", "hash": "0", "__v": 0}, {"_id": "63192563e25c92559aa502c4", "index": 2, "timestamp": "2022-09-07T23:12:34.430Z", "transactions": []}, {"nonce": 18140, "previousBlockHash": "0", "hash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "__v": 0}, {"_id": "631925df25c92559aa502c8", "index": 3, "timestamp": "2022-09-07T23:14:38.558Z", "transactions": [{"data": {"datos": "Datos del paciente: -----", "address": "HospitalDrPresetOffice", "responsable": "Michael Scott", "paciente": "23898933V", "imageUniHash": ["3d2a4442d56e0ce1de008d5cd89d75d0a1f622a9010fc4b164fbfaa48cc52c"]}, "imagesHash": ["0c443a986602d976dc802fc2112b09b2418bd456f164de87d964e3126595021"]}], "transUUID": "1ed9d92f9fd14cb28cb2814e735d7cd1", "timestamp": "2022-09-07T23:14:33.397Z"}], "nonce": 81703, "previousBlockHash": "0000b9135b054d1131392c9eb9d03b0111d4b516824a03c35639e12858912100", "hash": "00006d63d8b2875241737713ec3fa2a10e6fc07e360bc326f0b6a1174b3a6aa4", "__v": 0}, {"_id": "63192898e25c92559aa502cc", "index": 4, "timestamp": "2022-09-07T23:26:10.245Z", "transactions": []}, {"nonce": 2949, "previousBlockHash": "00006d63d8b2875241737713ec3fa2a10e6fc07e360bc326f0b6a1174b3a6aa4", "hash": "0000db2facdfb1f49cef8621299b6a3896b4c232c2599938f7a9a8bbc9c2880", "__v": 0}, {"_id": "631928e3e25c92559aa502d0", "index": 5, "timestamp": "2022-09-07T23:27:30.197Z", "transactions": []}, {"nonce": 168726, "previousBlockHash": "0000db2facdfb1f49cef8621299b6a3896b4c232c2599938f7a9a8bbc9c2880", "hash": "000081e2fdd8a1f490d7caedc585c8fc15671078f6ad691fe308cebe535e76b5", "__v": 0}]}]
```

**Figura 6.18:** Comprobación algoritmo de consenso

Como se observa en la imagen anterior, al ejecutar el algoritmo de consenso, cualquier cadena con una longitud menor a la cadena de mayor longitud procede a ser reemplazada por esta, dificultando así que los nodos con intenciones maliciosas se hagan con el control de la cadena. Cabe señalar también que con un número de nodos tan bajo es obvio que la seguridad garantizada por este algoritmo es de bajo nivel, pero conforme aumente el número de nodos de la red, aumentará simultáneamente el nivel de seguridad.

## 6.2. Pruebas de rendimiento

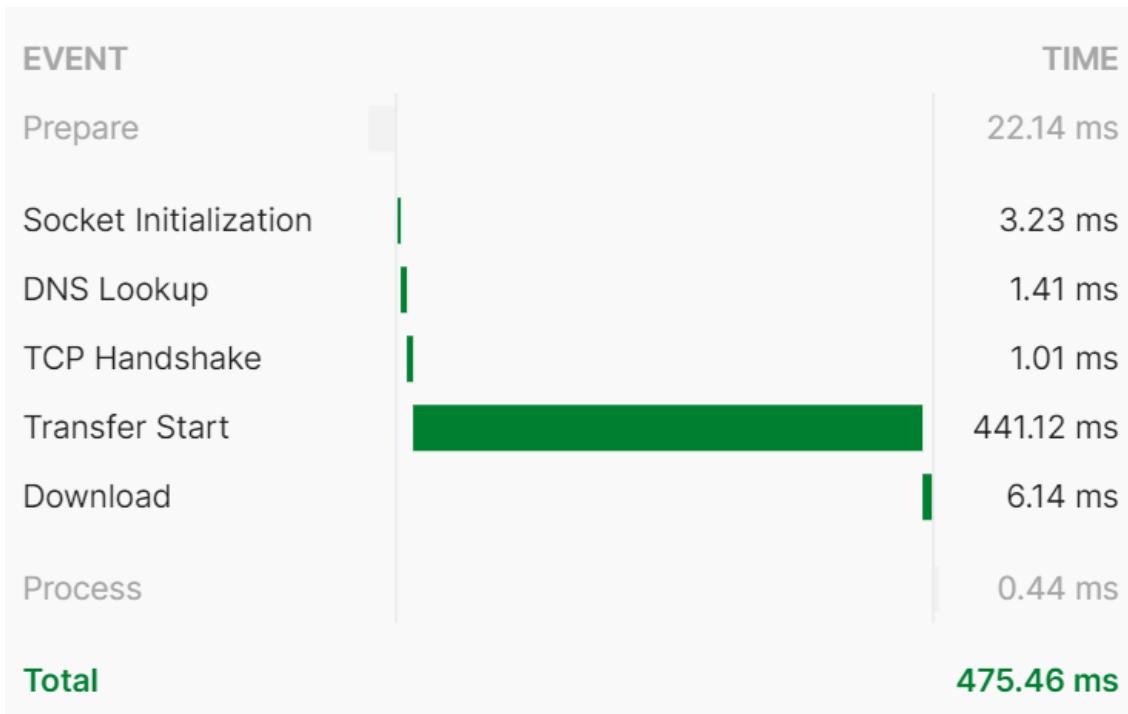
Las pruebas de rendimiento se limitan a analizar el tiempo empleado por la aplicación en llevar a cabo las distintas funciones.

Podemos medir este tiempo de distintas maneras, en esta guía se muestran solamente dos de las múltiples formas que hay de obtener este tiempo.

### Obtener los tiempos de ejecución con Postman

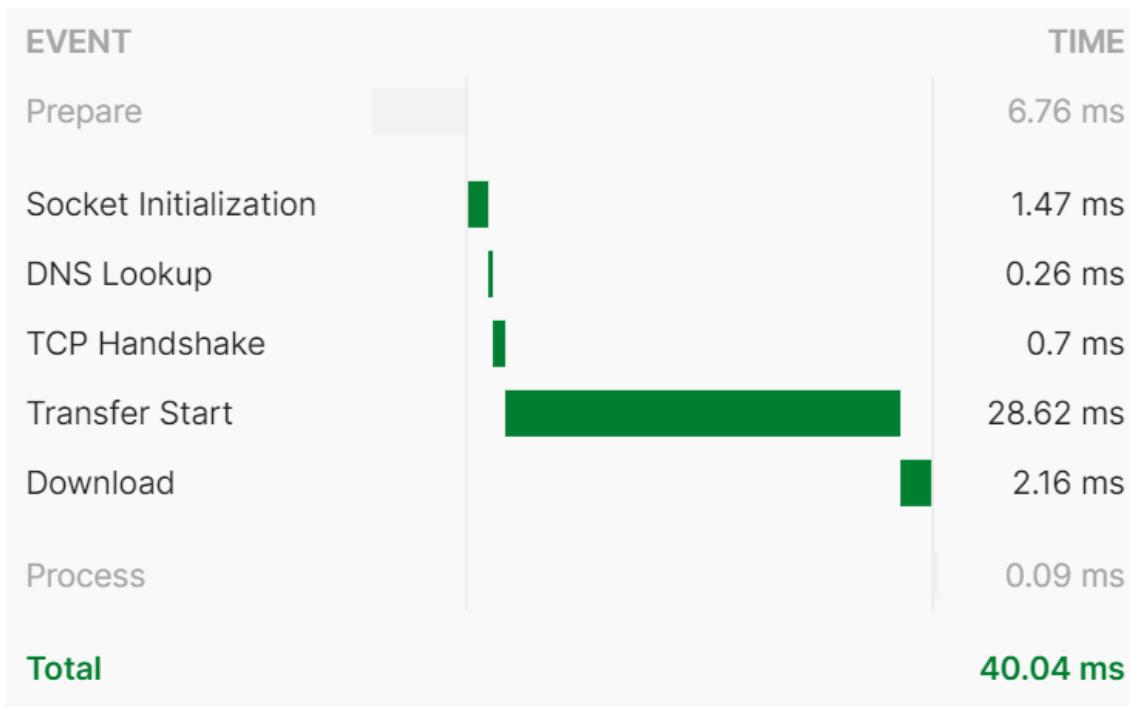
La aplicación de postman nos facilita un indicador con los tiempos de respuesta que nos indica de manera exacta cuánto tiempo se ha consumido y en qué apartado de la petición se ha consumido.

Uno de los principales indicadores del rendimiento de nuestra aplicación es el tiempo que tarda en ejecutarse la función /mine. Así que procedemos a ejecutarla y a observar los resultados.



**Figura 6.19:** Medidor tiempo petición /mine Postman

Otro buen indicador de rendimiento es el tiempo que tarda en crearse una transacción y propagarse tras llamar al endpoint /transaction/broadcast. Así que procedemos a calcularlo.



**Figura 6.19:** Medidor tiempo petición /transaction Postman

#### Obtener los tiempos de ejecución con console.time()

Añadiendo la linea `console.time("X")` al principio del método y la linea `console.timeEnd("X")` al final, podemos obtener cuánto tiempo ha tardado en ejecutarse ese método.

En las figuras siguientes se puede ver el tiempo mostrado por pantalla para las mismas llamadas a las funciones `/mine` y `/transaction/broadcast` que se muestran en el medidor de Postman.

```
timeMining: 401.623ms
```

**Figura 6.20:** Medidor tiempo de minado

```
timeTransaction: 1.109ms
```

**Figura 6.21:** Medidor tiempo de transacción

### 6.3. Discusión de los resultados

Los resultados de las pruebas son realmente satisfactorios y verifican haber alcanzado todas las funcionalidades correctamente. Como se ha observado, la red es capaz de mantenerse sincronizada, de soportar la caída de algunos de sus nodos y de llevar a cabo transacciones de datos y minado de nuevos bloques de manera satisfactoria. Además de esto, los bloques son enlazados de manera correcta y las

transacciones son almacenadas en estos satisfactoriamente. Por lo que los casos de uso planteados han sido resueltos correctamente.

Gracias al correcto funcionamiento del algoritmo de consenso también podemos asegurar que siempre se cumplirá la regla de la cadena más larga, aumentando de esta forma la seguridad de nuestra base de datos.

Respecto a los resultados temporales, cualquier tipo de conclusión que se pretendiese obtener de ellos sería precipitada, si comparamos el tiempo medio de minado de un bloque en nuestra cadena con el tiempo medio de minado en una cadena algo más conocida como podría ser Bitcoin, las diferencias de tiempo son abismales, a Bitcoin le tomaría de media 10 minutos realizar el minado de un nuevo bloque, por los 0.5s de la nuestra.

Esto no quiere indicar nada, seguramente, si queremos aumentar la seguridad en nuestra cadena se deba aumentar el número de 0's a introducir en el hash, pero al tratarse de una cadena híbrida con autenticación necesaria no sería necesario llegar a un extremo como el de bitcoin, dependiendo de múltiples factores como el número de nodos en la red o la cantidad de transacciones ejecutadas por minuto interesaría aumentar o incrementar este número, pero cabe señalar que la cantidad usada durante las pruebas ha sido seleccionada para garantizar al mismo tiempo la rapidez y el ahorro energético.

## 7. Conclusiones

---

Este proyecto ha sido una gran oportunidad para aprender, y disfrutar aprendiendo sobre la tecnología blockchain en todos sus aspectos. Si bien hoy en día a nivel práctico y empresarial podría resultar algo disparatado la creación de una cadena de bloques desde cero y la implementación de toda su seguridad habiendo alternativas bastante viables y eficientes, el hecho de crear la cadena desde cero y ver cómo interactúan todas las tecnologías que componen *blockchain* es una tarea primordial a nivel didáctico y que considero necesaria para lograr un conocimiento verdaderamente profundo sobre esta tecnología. Con esta creación desde cero lo que he logrado ha sido una gran comprensión de todos los términos de *blockchain* que suelen ser explicados sólamente desde un marco teórico como por ejemplo el mecanismo de consenso Proof of Work, y que al llevarlos a un terreno práctico y tener que construirlos desde cero són comprendidos desde una perspectiva más cercana y directa.

Tomando las funcionalidades y facilidades que nos ofrecen muchas plataformas *blockchain*, podemos tener acceso directo a muchas de sus ventajas y comprender algunos aspectos de forma superficial, pero no podremos ahondar en los cimientos básicos que componen esta tecnología y que eran la principal prioridad de este trabajo.

Durante la realización de este trabajo he adquirido una gran cantidad de conocimientos sobre JavaScript, lenguaje al cual ya había sido introducido en la asignatura de Tecnologías de los sistemas de información en la red (TSR) pero en el cual aún tenía algunas carencias a la hora de crear código desde cero. La naturaleza asíncrona de JavaScript ha sido uno de los mayores quebraderos de cabeza a la hora de llevar a cabo este trabajo ya que no estaba acostumbrado al uso de lenguajes asíncronos y la mayoría de operaciones de actualización que se ejecutaban sobre la cadena eran de esta naturaleza. Por esta razón me ha tocado aprender a lidiar con los procesos asíncronos, con las promesas y con los *callbacks*. He de decir que la solución muchas veces era mucho más sencilla de lo que yo intentaba realizar y que las librerías de Mongo y de Mongoose proporcionan herramientas más que suficientes para realizar cualquier tipo de operación sobre la cadena. Para la resolución de estos problemas o de estos apartados de la aplicación que no lograba hacer funcionar correctamente trataba de ceñirme lo máximo posible a las guías de uso oficiales que proporcionaban las herramientas que utilizaba como por ejemplo Mongoose, y en caso de no obtener una solución deseada trataba de encontrarla en blogs donde la gente plantea problemas similares como por ejemplo Stack Overflow. He de decir que la mayoría de las veces por no decir la totalidad, lo que hay en la guía de la herramienta es lo que hay, y ceñirse a eso te puede ahorrar más de una pérdida de tiempo. También señalar que para la resolución de estos errores siempre he contado con la ayuda y la opinión de mis tutores que me han propuesto siempre su punto de vista y me han facilitado algunas soluciones.

Este trabajo también ha sido una gran oportunidad para desarrollar APIs con Express, está ha sido una de las partes que he considerado más enriquecedoras ya que no

recuerdo haber tenido nociones básicas de programación de APIs durante la carrera y me ha parecido un apartado de gran utilidad, este aspecto me ha permitido sacarle el máximo partido a la aplicación pudiendo gestionar todo tipo de peticiones Web.

Otro campo en el que he aprendido desde cero y que me ha sido de gran utilidad ha sido en la creación y la gestión de contenedores Docker, en este apartado quiero agradecer también a mi tutor por toda la ayuda que me ha proporcionado y toda la paciencia que ha tenido para conmigo. Además de aprender desde cero a gestionar y crear contenedores Docker, también he tenido que aprender a gestionar y utilizar desde cero una base de datos no relacional como es MongoDB y sobre la cual tampoco se trabaja en la carrera. Ha sido muy interesante y prácticamente el mayor reto de este trabajo el hecho de aprender a actualizar y a gestionar la cadena almacenada en Mongo desde Node.js y aprender a hacer uso de todas las operaciones CRUD mediante llamadas a la API.

Para la codificación y decodificación de imágenes he tenido que aprender también el funcionamiento del formato base64 y como hacer uso de este para transformar la imagen a binario para poder introducirla en la cadena y luego volver a codificarla de nuevo para poder mostrarla en la aplicación web.

Otra aplicación que he descubierto por recomendación de terceros y que me ha sido de gran utilidad durante todo el desarrollo de este trabajo sería Postman que me ha ayudado a realizar todas las pruebas necesarias y comprobar el correcto funcionamiento de la API de la cadena.

Respecto a la completitud del trabajo, creo que he llegado más lejos con esta implementación de lo que pretendía en un principio y estoy muy contento con el resultado, es evidente que desde un punto de vista empresarial o práctico esta aplicación tiene multitud de carencias y de aspectos a mejorar, pero desde una perspectiva académica considero que he adquirido más conocimientos de los que podría haber imaginado y que los objetivos planteados en un principio han sido logrados en su mayoría. Si bien el sistema no es plenamente funcional y los métodos de autenticación y la interfaz de uso son incompletos, en los aspectos principales del trabajo como son la construcción de una cadena robusta, segura y funcional, la gestión de los nodos o el hecho de mantener la cadena actualizada y coordinada a través de todos los nodos con sus respectivas bases de datos, considero que el trabajo ha sido realmente satisfactorio.

## 7.1 Relación del trabajo con los estudios cursados

Durante este trabajo me he dado cuenta de las grandes carencias que tenía, y que sigo teniendo como programador. Me ha servido para mejorar en muchos aspectos y para adquirir conocimientos que o bien había olvidado o bien aún no tenía en mi haber.

El grado de Ingeniería Informática, incluye una amplia gama de conocimientos, de estos, solamente una minoría han sido puestos en práctica a lo largo de este trabajo.

Si bien he echado en falta algunos conocimientos que por determinadas razones no se han impartido en las aulas, considero que ha habido una gran cantidad de conocimientos aprendidos en clase que han sido aplicados en este proyecto.

Un ejemplo de esto sería la comprensión de JavaScript y de Node.js a la que ya he hecho mención anteriormente llevada a cabo en la asignatura de TSR.

Otro ejemplo de conocimientos que han sido de utilidad podría ser también todos los aspectos aprendidos sobre HTML, CSS y Bootstrap aprendidos en las asignaturas de DEW y DCU que aunque no han sido el foco principal de este proyecto sí que han sido de utilidad a la hora de crear la sencilla página web desde la que interactuar con la cadena.

También más próximo a la realización de este trabajo y de gran utilidad serían todos los conocimientos sobre criptografía y funciones de cifrado adquiridos en la asignatura de la rama de Tecnologías de la información RCO que me han permitido comprender con anterioridad la función SHA-256 así como toda la familia de los hashes.

Cómo he explicado con anterioridad, muchas de las aplicaciones y de los conocimientos que han hecho posible la realización de este trabajo han sido obtenidos de manera individual fuera de las aulas, ya fuera porque se trata de una tecnología algo novedosa o que tal vez se salga del campo de estudio de mi rama. Pero los conocimientos más importantes para la realización de este trabajo o de cualquier otro como son la capacidad para buscar la información correcta y la solución de problemas, forman parte de los conocimientos que he adquirido durante la duración de estos estudios y que me han ayudado a mejorar en muchos aspectos.

## 8. Trabajos futuros

---

Al tratarse de una aplicación tan extensa y con fines más bien demostrativos o didácticos, es una realidad que hay aspectos de la implementación general que han quedado un poco más desatendidos. Podríamos hacer la siguiente enumeración de los aspectos a mejorar:

### Usabilidad

La interfaz que se ha creado es una interfaz que meramente facilita las posibles pruebas de la aplicación pero queda muy lejos de cualquier diseño de interfaz a nivel profesional, en primer lugar no se ha tenido en cuenta la opinión de los posibles usuarios a la hora de diseñarla. En segundo lugar, es parcialmente incompleta, puesto que tampoco facilita operaciones tan importantes como la creación de transacciones, la adhesión de un nuevo nodo a la red de nodos, la visualización de toda la cadena o el minado de un nuevo bloque.

### Autenticación

Otro aspecto en el que se podrían realizar importantes mejoras sería en la autenticación, tanto de los pacientes como de los doctores o responsables, sería interesante permitir dos opciones, una que sólo permitiera la lectura y otra la edición también, esto es posible de manera directa cambiando los permisos de MongoDB y creando reglas para cada tipo de usuario, el problema es que esto debe hacerse por código de momento y es algo rudimentario y aparatoso. Sería interesante incluir un apartado de autenticación en la aplicación web que permitiera gestionar los permisos de los usuarios dependiendo de su rol.

### Privacidad

Uno de los aspectos principales de esta aplicación era lograr la una mayor privacidad para los usuarios, y este es uno de los apartados en los que más ha dejado que desear la aplicación. Si bien los datos dentro de la cadena son seguros, creo de vital importancia, en primer lugar que quede registro de quienes tienen acceso a esos datos, lo cual sería interesante para el paciente y poseedor de los datos. Y en segundo lugar la anonimidad de esos datos, bien cifrando el nombre del paciente con una función hash o bien usando un distintivo único al que solo paciente y responsable tengan acceso.

### Eficiencia energética

Al ser una implementación demostrativa se ha optado por un algoritmo de consenso como Proof Of Work que no es precisamente conocido por su eficiencia. En primer lugar la cantidad de ceros con la que funciona actualmente ha sido introducida para simplemente facilitar las pruebas de la aplicación y que estas lleven el menor tiempo posible dentro de un límite razonable por lo que se debería realizar un análisis en

profundidad de cuál sería la cantidad óptima. En segundo lugar este podría ser sustituido por algunas opciones ya mencionadas como Proof of Elapsed Time, Proof of Capacity o bien Tendermint Core, empleado por BigchainDB según las preferencias y conclusiones del futuro desarrollador.

#### Rapidez de ejecución

Además de la eficiencia energética, el uso de recursos como el algoritmo Proof of Work, conlleva importantes lastres a la hora de obtener buenos resultados en lo que a la velocidad de respuesta se refiere. Estos retrasos se aumentan en cuanto mayor es la información que va a ser incluida dentro de un bloque, sobretodo en el caso de las imágenes, por lo que buscar una alternativa a la solución actual es un factor necesario de cara a futuras implementaciones.

## 9. Referencias

---

- [1] [¿Qué es el Trilema de la Blockchain? | Bybit Learn](#)
- [2] [bigchaindb/bigchaindb: Meet BigchainDB. The blockchain database. \(github.com\)](#)
- [3] [BigchainDB • The blockchain database.](#)
- [4] Jae Kwon. Tendermint: Consensus without Mining, fall 2014.
- [5] [LPgoWO18TCeMlggJVakt\\_tendermint.pdf \(relayto.com\)](#)
- [6] Tendermint. <https://tendermint.com/>.
- [7] MongoDB. [MongoDB: The Developer Data Platform | MongoDB | MongoDB](#)
- [8] [Mongoose v6.4.4: Connecting to MongoDB \(mongoosejs.com\)](#)
- [9] [What is Sybil attack: How Blockchains Prevent Sybil Attacks - Phemex Academy](#)
- [10] [51% Attacks — MIT Digital Currency Initiative](#)
- [11] [Qué son las Transacciones Por Segundo \(TPS\) - Phemex](#)
- [12] [Lightning Network](#)
- [13] [Byzantine Fault Tolerance Explained | Binance Academy](#)
- [14] Learn Blockchain Programming With Javascript: Build Your Very Own Blockchain And Decentralized Network With Javascript And Node.Js Eric Traub (page 467 Consensus Algorithms).
- [15] [Proof-of-work \(PoW\) | ethereum.org](#)
- [16] [Vulnerability: Proof of Work vs. Proof of Stake | by Robert Greenfield IV | Medium](#)
- [17] [Algoritmos HASH: Qué son, seguridad, uso y funcionamiento \(redeszone.net\)](#)
- [18] [How SHA-256 Works Step-By-Step | Boot.dev](#)
- [19] [What is UUID? \(techtarget.com\)](#)
- [20] [BOE.es - BOE-A-2018-16673 Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.](#)
- [21] [La nube de la Administración: dónde se guardan los datos de los españoles - Datagestion : Datagestion](#)
- [22] [Hybrid Blockchain Explained. Straight to the point! | by The Writing Zone | DataDrivenInvestor](#)
- [23] [Express - Infraestructura de aplicaciones web Node.js \(expressjs.com\)](#)
- [24] [BigchainDB 2.0 Whitepaper • BigchainDB](#)
- [25] [Ataque del 51% en redes Blockchain: qué es y cómo puede perjudicarte \(redeszone.net\)](#)

- [26] [Integridad de datos en una base de datos: ¿por qué es importante? Astera](#)
  - [27] [Seguridad de la información, un conocimiento imprescindible \(obsbusiness.school\)](#)
  - [28] [Autenticación - Wikipedia, la enciclopedia libre](#)
  - [29] Docker [Home - Docker](#)
  - [30] Postman [Postman API Platform | Sign Up for Free](#)
- [31] Learn Blockchain Programming With Javascript: Build Your Very Own Blockchain And Decentralized Network With Javascript And Node.js Eric Traub (page 44 What is a blockchain?).

#### REFERENCIAS DE IMÀGENES

- [1] Figura 2.0 [cover-14.png \(1458×600\) \(ledger.com\)](#)
- [2] Figura 2.1 [Loss from 51% attack on eight cryptocurrencies that have most recently... | Download Scientific Diagram \(researchgate.net\)](#)
- [3] Figura 2.3 [Features & Use Cases • • BigchainDB](#)

# Apéndice 1. Relación con los Objetivos de Desarrollo Sostenible (ODS)

El 25 de septiembre de 2015 los líderes mundiales adoptaron 17 Objetivos de Desarrollo Sostenible (ODS) para proteger el planeta, luchar contra la pobreza y tratar de erradicarla con el objetivo de construir un mundo más próspero, justo y sostenible para las generaciones futuras. Estos objetivos se fijaron dentro de la Agenda 2030 sobre el desarrollo sostenible.



Con los 17 ODS se buscó involucrar a gobiernos, empresas, sociedad civil y también a las personas a título individual. Dentro de cada objetivo se trazan diferentes metas y cada una de ellas cuenta con sus propios indicadores que sirven para determinar si el objetivo se cumple o no. Dentro del conjunto de ODS se encuentran aquellos objetivos enfocados en el avance de energía limpia, el trabajo decente y el crecimiento económico, el consumo y la producción responsable, la acción contra el clima o la industria, y los orientados a la innovación e infraestructura.

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. Fin de la pobreza.				x
ODS 2. Hambre cero.				x
ODS 3. Salud y bienestar.	x			
ODS 4. Educación de calidad.				x
ODS 5. Igualdad de género.				x
ODS 6. Agua limpia y saneamiento.				x

ODS 7. <b>Energía asequible y no contaminante.</b>				x
ODS 8. <b>Trabajo decente y crecimiento económico.</b>				x
ODS 9. <b>Industria, innovación e infraestructuras.</b>			x	
ODS 10. <b>Reducción de las desigualdades.</b>				x
ODS 11. <b>Ciudades y comunidades sostenibles.</b>				x
ODS 12. <b>Producción y consumo responsables.</b>				x
ODS 13. <b>Acción por el clima.</b>				x
ODS 14. <b>Vida submarina.</b>				x
ODS 15. <b>Vida de ecosistemas terrestres.</b>				x
ODS 16. <b>Paz, justicia e instituciones sólidas.</b>	x			
ODS 17. <b>Alianzas para lograr objetivos.</b>				x

En este apartado se mencionan cuáles de los ODS mencionados anteriormente están relacionados con este proyecto.

## 3 SALUD Y BIENESTAR



### Objetivo 3: Garantizar una vida sana y promover el bienestar para todos en todas las edades

Al ser un proyecto con aplicación directa en el ámbito de la salud, este proyecto trata de mejorar directamente aspectos relacionados con la obtención de una vida más

saludable para todos en todas las edades. Las facilidades ofrecidas en el tratamiento de los datos así como en su accesibilidad pueden ser de gran ayuda tanto para los profesionales de este ámbito como para los pacientes que deseen consultar sus datos.



### **Objetivo 16: Promover sociedades justas, pacíficas e inclusivas**

Los conflictos, la inseguridad, las instituciones débiles y el acceso limitado a la justicia continúan suponiendo una grave amenaza para el desarrollo sostenible.

Gracias a la tecnología blockchain y a las redes centralizadas, se está consiguiendo crear sistemas con una gran distribución del poder y se están haciendo grandes esfuerzos para acabar con los monopolios económicos.

Gracias a este tipo de cadenas, está garantizada la igualdad en el tratamiento de los datos así como la seguridad en el tratamiento de estos. Promoviendo un modelo de almacenamiento que cumpla con todas las reglas establecidas sin prácticamente necesidad de inspecciones ya que la propia arquitectura de la solución garantiza el cumplimiento de estos aspectos legales.

También, permitiendo el acceso a la información a la ciudadanía y protegiendo sus identidades se estaría cumpliendo la regla 16.10 del objetivo, que propone lo siguiente: garantizar el acceso público a la información y proteger las libertades fundamentales, de conformidad con las leyes nacionales y los acuerdos internacionales.

