



*ugr* | Universidad  
de **Granada**

TRABAJO FIN DE GRADO  
INGENIERÍA INFORMÁTICA

# Creación y configuración de una red Blockchain y desarrollo de una aplicación

---

**Autor**

Manuel Ros Rodríguez

**Directores**

José Luis Garrido Bullejos  
María Bermúdez Edo



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN

—  
Granada, 8 de septiembre de 2020







# **Creación y configuración de una red Blockchain y desarrollo de una aplicación**

Manuel Ros Rodríguez

**Palabras clave:** sistemas distribuidos, blockchain, Ethereum, smart contracts, trazabilidad

## **Resumen**

En este proyecto se crea una red blockchain (cadena de bloques) utilizando la tecnología Ethereum, mediante máquinas virtuales en un servidor y posteriormente se diseña e implementa una aplicación que hace uso de la red blockchain creada.

Las tecnologías blockchain son un campo de actualidad y se pueden utilizar para crear sistemas distribuidos, donde se utiliza como base de datos la misma red blockchain, que es una buena solución para situaciones donde se quiere gestionar una gran cantidad de información de forma inalterable y transparente. Las tecnologías blockchain están en constante actualización.

No solo se utilizan como base de datos, sino que permiten subir código en ellas y ejecutarlo. Debido a que se tiene una copia de la red blockchain en cada nodo participante, se consigue una alta redundancia, garantizando una alta disponibilidad y reduciendo las probabilidades de que se pierdan datos.

Las tecnologías blockchain son famosas por ser utilizadas para crear criptomonedas, como Bitcoin, pese a ser una alternativa muy atractiva para la creación de sistemas distribuidos descentralizados, por lo que consideramos que no se está utilizando todo su potencial.

Para mostrar cómo funciona y aprovechar las características y ventajas que ofrece una red blockchain desarrollamos una aplicación como prueba de concepto que realiza un seguimiento de la trazabilidad de los suministros médicos, tema que podría ser de utilidad en la actualidad debido a la pandemia global COVID-19.

Junto al desarrollo de la aplicación que se subirá en la red blockchain también se ha diseñado una API (Application Programming Interface) REST (Representational State Transfer), implementada en Node.js, que se comunicará con la aplicación dentro de la red blockchain. Se incluye una página web simple para demostrar el uso de la API REST y poder interactuar con la red blockchain.

El seguimiento de la trazabilidad de los suministros médicos se realiza sin tener en cuenta el origen del suministro médico, centrándonos en España y partiendo de una versión simplificada de la realidad. Tener en cuenta el

origen implicaría conocer la normativa médica y aduanera de los diferentes países, cuestión que está fuera del objetivo de este trabajo. Con la ayuda de expertos en estos temas se podría expandir la aplicación para cubrir la trazabilidad desde el origen y cumplir las normativas existentes.

# **Creation and configuration of a Blockchain network and development of an application**

Manuel Ros Rodríguez

**Keywords:** distributed systems, blockchain, Ethereum, smart contracts, traceability

## **Abstract**

In this project we create a blockchain network using the Ethereum technology, with virtual machines and design as well as develop an application that makes use of the blockchain network created.

Blockchain technologies are a very interesting and current area of investigation that can be used to create distributed systems, where the blockchain network itself is the database and it is a good solution for situations where we want to handle a big amount of information in an unalterable and transparent way. The blockchain technologies are constantly being updated.

They are not only used as a database, they allow the upload and execution of code. Every node that participates has a copy of the blockchain network, making it highly fault-tolerant and with high data redundancy.

Although it is a very interesting alternative for developing decentralized distributed systems, blockchain technologies are famous for being used to create cryptocurrencies. For that reason, we consider that this technology is not being used to its full potential.

To show its potential and demonstrate how an application that uses the blockchain network works, we design and develop an application as proof of concept where we follow medical supplies traceability. This can be a very useful application taking into account the COVID-19 global pandemic we are all facing.

Apart from the development of the application that will be uploaded to the blockchain network we design a REST (Representational State Transfer) API (Application Programming Interface), made with Node.js, that will communicate with the application in the blockchain network. We include a simple webpage to demonstrate the use of the REST API and to allow interactions with the blockchain network.

For the traceability of medical supplies, we do not take into account the origin of the supplies and we focus in Spain, using a simplified version of reality. Expanding the traceability to include the origin of the supplies will require knowing the medical and customs laws of other countries and that would be outside the scope of this project. With the help of experts on this subject we would be able to easily expand the application to cover the entire medical supplies traceability and comply with existing laws.



---

Yo, **Manuel Ros Rodríguez**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, con DNI 76653644A, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Manuel Ros Rodríguez

Granada a 8 de septiembre de 2020.



---

**D. José Luis Garrido Bullejos**, Profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

**Dña. María Bermúdez Edo**, Profesora del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

**Informan:**

Que el presente trabajo, titulado *Creación y configuración de una red Blockchain y desarrollo de una aplicación*, ha sido realizado bajo su supervisión por **Manuel Ros Rodríguez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada 8 de septiembre de 2020.

**Los directores:**

**José Luis Garrido Bullejos**

**María Bermúdez Edo**



# **Agradecimientos**

Quiero dar las gracias a mis tutores, José Luis Garrido Bullejos y María Bermúdez Edo, por la ayuda ofrecida, sus recomendaciones y el tiempo que han dedicado a este proyecto.

Agradecer a mis amigos, por su compañía y apoyo durante estos años de estudio.

Por último, dar las gracias a mis padres, por su paciencia y apoyo.



# Índice general

<b>1. Introducción</b>	<b>17</b>
1.1. Motivación . . . . .	18
1.2. Descripción del problema y planteamiento inicial para la propuesta de una solución . . . . .	18
1.3. Objetivos . . . . .	20
1.4. Planificación temporal . . . . .	21
1.5. Presupuesto . . . . .	22
<b>2. Tecnología blockchain: Fundamentos</b>	<b>25</b>
2.1. ¿Que es una red blockchain? . . . . .	26
2.2. Conceptos sobre redes blockchain . . . . .	26
2.2.1. Bloque génesis . . . . .	27
2.2.2. Tipos de nodos . . . . .	27
2.2.3. Tipos de servicios . . . . .	28
2.3. Protocolos para añadir bloques (Protocolos de consenso) . . .	34
2.3.1. Proof of Work . . . . .	34
2.3.2. Proof of Authority . . . . .	34
2.3.3. Proof of Stake . . . . .	35
<b>3. Estado del Arte</b>	<b>36</b>
3.1. Bitcoin . . . . .	37
3.2. Ethereum . . . . .	38
3.3. Otras tecnologías blockchain en la actualidad . . . . .	39
3.3.1. Quorum . . . . .	39
3.3.2. Hyperledger Fabric . . . . .	39

<b>4. Creación de una red blockchain basada en la plataforma Ethereum</b>	<b>40</b>
4.1. Elección de la tecnología blockchain . . . . .	41
4.2. Herramientas utilizadas . . . . .	42
4.2.1. Proxmox . . . . .	43
4.2.2. Puppet . . . . .	43
4.2.3. Geth . . . . .	44
4.3. Configuración del servidor . . . . .	48
4.4. Privacidad de la red blockchain . . . . .	57
4.5. Creación de la red blockchain utilizando Puppet . . . . .	59
4.5.1. Configurar bloque génesis . . . . .	60
4.5.2. Instalación de Ethstats . . . . .	62
4.5.3. Instalación de los nodos bootnode . . . . .	62
4.5.4. Instalación de los nodos signer . . . . .	63
4.5.5. Instalación de los servicios opcionales (Explorer, Wallet, Faucet, Dashboard) . . . . .	63
4.6. Creación de la red blockchain de forma manual . . . . .	68
4.6.1. Instalación de Ethstats . . . . .	68
4.6.2. Configuración manual del bloque génesis . . . . .	69
4.6.3. Instalación del bootnode . . . . .	70
4.6.4. Instalación de los nodos signer . . . . .	73
<b>5. Desarrollo de una aplicación descentralizada</b>	<b>75</b>
5.1. Descripción general de la trazabilidad de suministros médicos	76
5.2. Visión general del sistema . . . . .	77
5.3. Metodología de desarrollo utilizada . . . . .	78
5.4. API de Ethereum . . . . .	79
5.5. Programación en Ethereum, diferencias respecto a programación orientada a objetos . . . . .	80
5.6. Especificación de requisitos del sistema . . . . .	82
5.6.1. Requisitos de información (RI) . . . . .	82
5.6.2. Requisitos funcionales (RF) . . . . .	83
5.6.3. Requisitos no funcionales (RNF) . . . . .	85

5.7. Historias de usuario (Product backlog) . . . . .	86
5.7.1. Primer Sprint . . . . .	86
5.7.2. Segundo Sprint . . . . .	89
5.8. Arquitectura del software . . . . .	92
5.9. Diagramas de clases . . . . .	94
5.10. Base de datos del sistema . . . . .	100
5.11. Definición de la API para la interacción con la aplicación . . . . .	104
5.11.1. Administrador . . . . .	106
5.11.2. Comunidades autónomas . . . . .	107
5.11.3. Almacenes . . . . .	110
5.11.4. Suministros . . . . .	117
<b>6. Conclusiones y trabajo futuro</b>	<b>130</b>
6.1. Conclusiones . . . . .	131
6.2. Trabajo futuro . . . . .	133
6.3. Valoración personal del desarrollo del trabajo . . . . .	134
<b>Bibliografía</b>	<b>139</b>

# **Capítulo 1**

## **Introducción**

Este capítulo describe brevemente la motivación personal para llevar a cabo este trabajo, realiza una descripción general del problema y el planteamiento inicial para la propuesta de una solución, se especifican los objetivos a alcanzar, la planificación temporal y por último el presupuesto del proyecto.

## 1.1. Motivación

Durante mi segundo año de carrera cursé la asignatura Sistemas Concurrentes y Distribuidos, donde descubrí mi interés y pasión por este tema. Las asignaturas que han estado centradas en temas similares han acabado siendo las que más me han interesado de la carrera, mencionando en especial a la asignatura de 3º Curso de la Especialidad de Ingeniería del Software denominada Desarrollo de Sistemas Distribuidos, que como su nombre indica dejaba de lado la parte de paralelismo para centrarse más en el desarrollo de software para sistemas distribuidos en general. Por eso cuando me planteé el tema del TFG decidí que sería algo relacionado con este campo.

En las primeras reuniones con mis tutores José Luis Garrido Bullejos y posteriormente María Bermúdez Edo, se mencionó que un posible tema podría ser estudiar en profundidad y poner en práctica la tecnología blockchain (cadena de bloques) con funcionalidad smart contracts (contratos inteligentes) para el desarrollo de dapps (aplicaciones software descentralizadas). Me pareció una idea muy interesante, ya que no solo está relacionado con sistemas distribuidos, sino que además es un tema de actualidad e innovador. Por lo que elegí este TFG: La creación de una red blockchain y desarrollo de aplicaciones descentralizadas bajo smart contracts.

Es un campo que hoy en día está llegando a ser más conocido, siendo los ejemplos más destacables las monedas virtuales (criptomonedas) como el Bitcoin o Ethereum, que le dan la principal fama, cuando es una alternativa de alto interés para la creación de otros tipos de sistemas distribuidos, gracias a su capacidad de gestionar información de manera transparente, segura e inalterable.

## 1.2. Descripción del problema y planteamiento inicial para la propuesta de una solución

El problema a tratar en este TFG es la creación de una red blockchain, para ello se utilizará como base alguna tecnología/plataforma existente como Ethereum o similares. Para mostrar su potencial se desarrollará una aplicación descentralizada como prueba de concepto sobre dicha red. Por lo que

se tendrá que configurar un servidor físico, crear y configurar máquinas virtuales en él, instalar en ellas los nodos necesarios de la red blockchain, y diseñar e implementar la aplicación a desarrollar.

La tecnología blockchain es útil para situaciones donde se quiere gestionar una alta cantidad de información, de forma descentralizada sin que se pueda modificar pero que cualquier persona la pueda consultar. Por ejemplo, sustituyendo a los registros públicos o para seguir la trazabilidad de algún proceso de negocio o procesamiento administrativo.

La creación de una red blockchain requiere la instalación de un conjunto de nodos, que se comunicarán entre ellos y se necesitarían algunos tipos de nodos obligatorios, lo cual tendremos que investigarlo. Estos nodos se instalarán como se ha mencionado en máquinas virtuales de un único servidor físico, por lo que tendremos que asegurarnos de que se puedan comunicar entre ellas y con el exterior, pues sería interesante poder tener un nodo en el ordenador que se está utilizando para el trabajo y así facilitar el desarrollo de la aplicación.

Otra parte importante estaría relacionada con la privacidad de la red blockchain, habría que asegurarse que la red blockchain fuese de verdad privada, para evitar posibles ataques de nodos maliciosos. Adicionalmente, sería muy interesante incluir algún tipo de página web o similar o ver alguna forma de monitorizar nuestra red blockchain y obtener información de ella. Así podríamos obtener información durante el desarrollo de la aplicación y saber si nos estamos comunicando de forma correcta con ella o ver si los nodos creados están funcionando bien y sincronizando la red blockchain.

Se va a desarrollar una aplicación que utilice redes blockchain y esto implica el desarrollo de contratos inteligentes, que tienen su propio lenguaje de programación, por lo que hay que aprender a programar en este lenguaje nuevo, además hay que diseñar e implementar algún tipo de API propia que haga uso de las API disponibles en la tecnología blockchain seleccionada para interactuar con la red blockchain de modo que facilite y permita la ejecución de los contratos.

Como para programar la aplicación se utilizan conceptos diferentes a clases (contratos) esto provoca que debamos investigar las diferencias que hay entre contratos y clases, ya que podría provocar que un diseño orientado a objeto tradicional no encajase con los contratos y por tanto tengamos que adaptarlo e intentar obtener los beneficios que nos da el diseño orientado a objetos.

Sería interesante ver cómo se pueden superar las limitaciones existentes en las redes blockchain para conseguir el máximo rendimiento posible. El uso de redes blockchain tiene sus peculiaridades, como tener que esperar varios segundos para que se empiece a ejecutar el código y por tanto se

modifique la información guardada.

Como problema a resolver con la aplicación inicialmente se planteó la trazabilidad de los alimentos, de forma que una persona en un supermercado podría, por ejemplo, con una aplicación del móvil escanear un código y saber exactamente el recorrido y las alteraciones que ha tenido el producto. Aunque no se terminó de tomar una decisión en este aspecto, sino que se decidió retrasar un poco la elección final de la aplicación a desarrollar.

Con la llegada del COVID-19 y después de hablarlo con los tutores se decidió desarrollar una aplicación similar a la mencionada, la cual también aprovecha las características de las tecnologías blockchain al máximo y resuelve un problema que está ocurriendo en la actualidad: La trazabilidad en el suministro y distribución de material sanitario (equipos de protección individual, test, mascarillas, etc.).

Nos encontramos hoy en día con noticias que hablan sobre mascarillas que no cumplen los estándares de calidad o el caos de la distribución de materiales. Utilizando una red blockchain para seguir su trazabilidad se podría ayudar en parte a resolver estos problemas de forma más automática, pues conoceríamos su origen y destino, y las decisiones tomadas y pasos seguidos en el proceso de distribución.

Para delimitar el alcance de este TFG y debido a las complejidades existentes en las normativas legales tanto de España como de otros países por los que pasaría el material sanitario decidimos limitar la aplicación a lo que correspondería al ámbito nacional y dentro de este utilizamos una versión simplificada de la realidad.

### 1.3. Objetivos

El objetivo general de este proyecto es la creación de una red blockchain, incluyendo la instalación y configuración del software de infraestructura necesario, y el desarrollo de una aplicación descentralizada a desplegar y ejecutar en dicha red que, a modo de prueba de concepto, permita llevar a cabo la trazabilidad del material médico. Para alcanzar este objetivo general se proponen los siguientes objetivos específicos:

1. Estudiar e investigar sobre tecnologías blockchain. Se pretende llevar a cabo una investigación y selección de una tecnología blockchain que pueda servir como plataforma base para crear una red blockchain, así como el estudio de los tipos de nodos y servicios relacionados con la tecnología blockchain elegida.
2. Llevar a cabo las tareas de gestión y soporte para instalar y configu-

rar una red virtual de servidores como plataforma distribuida para la creación de una red blockchain. Se configurará un servidor físico y se crearán y configurarán en él varias máquinas virtuales, las cuales tendremos que hacer accesibles desde el exterior.

3. Construcción de una red blockchain sobre la plataforma mencionada en el objetivo anterior. Se instalarán los nodos necesarios y se configurará la red blockchain.
4. Determinar y desarrollar una aplicación de interés que pueda mostrar la validez y funcionamiento de la red blockchain. Se estudiarán y analizarán las API (Application Programming Interface) disponibles de la tecnología blockchain elegida y se seleccionará la más adecuada para la aplicación. Así mismo, se llevará a cabo el diseño e implementación de la aplicación descentralizada hasta alcanzar un prototipo funcional.

#### 1.4. Planificación temporal

Al inicio del proyecto se realizó una planificación temporal de las tareas a realizar como medio para alcanzar los objetivos propuestos, la cual se ha ido modificando conforme avanzaba el proyecto. En este apartado mostraremos la planificación temporal final del proyecto.

Durante el desarrollo del proyecto, desde la planificación inicial pasando por los múltiples cambios que se han producido en ella hasta llegar a la finalización del proyecto, hemos aplicado un enfoque ágil, siguiendo la filosofía de las metodologías de desarrollo ágiles, para estar preparados y adaptarnos a cualquier cambio que pueda ocurrir, ya que resulta conveniente para un proyecto de este tipo con una parte de estudio, investigación, creación y uso de tecnologías recientes como es una red blockchain, y con una segunda parte de desarrollo de una aplicación soportada por dichas tecnologías en constante actualización. Por tanto, debido a que el proyecto se basa en el uso de estas tecnologías novedosas, es normal que ocurran cambios y sea necesario explorarlas para conocer en profundidad las carencias, deficiencias, adaptaciones, etc., que aún requieren. El seguir un enfoque ágil ha permitido abordar los cambios y necesidades que se han ido presentando sin problemas. En el Capítulo 5 titulado “Desarrollo de una aplicación descentralizada”, se hace uso de la metodología de desarrollo ágil SCRUM adaptada a un equipo de desarrollo formado por una sola persona, en ese capítulo detallamos como hemos aplicado esta metodología para el desarrollo de la aplicación.

La duración total del proyecto ha sido de 6 meses, desde marzo hasta agosto. Para la realización de la planificación temporal se ha utilizado un

diagrama de Gantt, que se muestra en la Figura 1, donde se indican las tareas realizadas con el tiempo que ha llevado realizarlas.

## 1.5. Presupuesto

Para el desarrollo de este proyecto necesitaríamos cubrir el coste del personal y el equipamiento.

Necesitaríamos el siguiente equipamiento:

- Ordenador para desarrollo y documentación.
- Servidor dedicado alquilado de la empresa Soyousart en el que montar la red blockchain.

El coste del personal sería el sueldo de un ingeniero informático que desarrollaría el proyecto a una media equivalente a tiempo completo, es decir, con días dedicados completamente al TFG pero otros días centrados en el resto del curso académico y días a medias donde se avanza ambos. Para el sueldo mensual se toma como referencia la información recogida en Indeed [1].

A continuación mostramos una tabla con el presupuesto:

Gastos elegibles	Importe solicitado
<b>GASTOS DE PERSONAL</b>	
Sueldo Ingeniero Informático 2300 euros/mes durante 6 meses	13800
<b>GASTOS DE EJECUCIÓN</b>	
Ordenador personal para desarrollo y documentación	1000
Instalación del servidor E3-SSD-1-32 de la empresa Soyous-tart	24.2
Asignación e instalación de bloque de IP (Internet Protocol) para uso en máquinas virtuales	9.68
Alquiler mensual del servidor E3-SSD-1-32 de la empresa Soyousstart 48.4 euros/mes durante 6 meses	290.4
<b>GASTOS TOTAL EUROS</b>	<b>15124.28</b>

Tabla 1: Presupuesto del proyecto

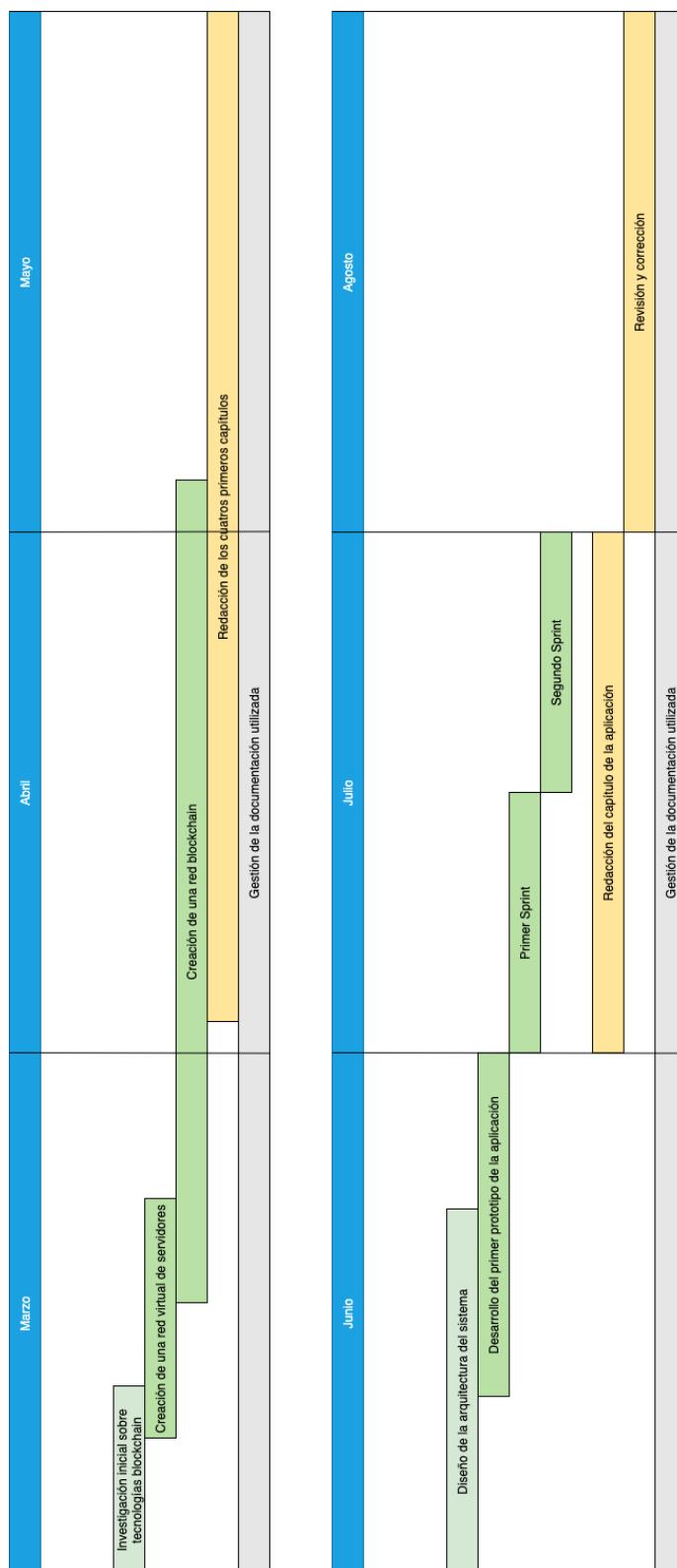


Figura 1: Diagrama de Gantt con la planificación temporal final

## **Capítulo 2**

# **Tecnología blockchain: Fundamentos**

En este capítulo describiremos que es una red blockchain, explicaremos una serie de conceptos básicos, como los tipos de nodos y servicios en las redes blockchain y terminaremos hablando sobre algunos de los protocolos de consenso más conocidos.

## 2.1. ¿Qué es una red blockchain?

Una red blockchain [2] es una base de datos distribuida de transacciones, que, como su nombre indica, está formada por una cadena (chain) de bloques (block), cada bloque contiene un conjunto de transacciones que se han realizado.

Las transacciones se agrupan por bloques, los cuales contienen un valor hash generado por el bloque anterior (enlazándolos y creando la cadena) y un valor hash creado a partir del hash del bloque anterior y las transacciones que contiene este bloque (véase la Figura 2).

El proceso por el que se añade un bloque a la red blockchain dependerá del protocolo utilizado y una vez añadido, esa información es retransmitida a todos los nodos que forman la red blockchain.

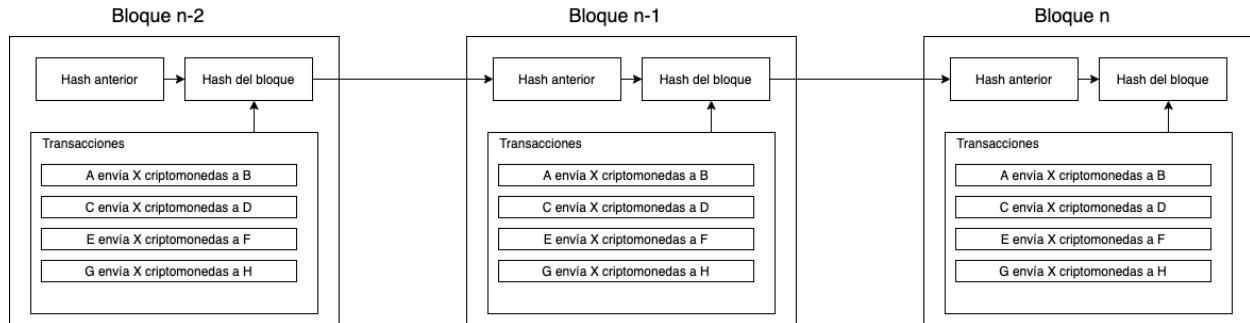


Figura 2: Esquema de una red blockchain

## 2.2. Conceptos sobre redes blockchain

En este apartado explicaremos los conceptos sobre redes blockchain que necesitamos conocer para poder utilizar y crear una. Nos centramos en la tecnología blockchain Ethereum y en la implementación Geth, los motivos por los que hemos elegido la tecnología Ethereum y Geth los exponemos en el capítulo 4: “Creación de una red blockchain basada en la plataforma Ethereum”

A la hora de crear una red blockchain Ethereum necesitaremos conocer

los tipos de nodos [3] y servicios básicos que hay disponibles. Para distinguir cuales son los servicios básicos nos hemos basado en los incluidos en Puppeth, herramienta que facilita la creación de una red blockchain Ethereum, en el apartado “4.2.2. Puppet” se hablará de ella.

### 2.2.1. Bloque génesis

Un bloque génesis [3] es un tipo especial de bloque, es el primer bloque de la red blockchain y es configurado a través de un archivo JSON (JavaScript Object Notation). Contiene la configuración de la red blockchain que se creará utilizando este bloque. La Figura 3 muestra un ejemplo del archivo JSON de configuración.

Figura 3: Ejemplo de archivo JSON de configuración para bloque génesis [3]

### 2.2.2. Tipos de nodos

Un nodo es un ordenador que está ejecutando un proceso que mantiene una copia actualizada de la red blockchain, dependiendo del tipo de nodo pueden tener responsabilidades adicionales, como añadir nuevos bloques en la red blockchain.

**a. Nodo básico**

Un nodo básico [3] no tiene ninguna responsabilidad adicional, lo único que hace es mantener una copia de la red blockchain.

**b. Bootnode**

Un bootnode (también conocido como bootstrap node) [3] permite a los nodos que acabamos de crear encontrar al resto de nodos que forman la red blockchain.

**c. Signer**

Un signer [3] es un nodo que tiene la responsabilidad adicional de añadir bloques en la red blockchain cuando se utiliza el protocolo Proof of Authority. En el apartado “2.3. Protocolos para añadir bloques (Protocolos de consenso)” se explica que es este protocolo.

**d. Miner**

Un miner [3] es un nodo que tiene la responsabilidad adicional de añadir bloques en la red blockchain cuando se utiliza el protocolo Proof of Work. En el apartado “2.3. Protocolos para añadir bloques (Protocolos de consenso)” se explica que es este protocolo.

### 2.2.3. Tipos de servicios

**a. Ethstats**

Ethstats [4] es un dashboard que obtiene información de los nodos de nuestra red y nos la muestra de forma gráfica, facilitando la monitorización de nuestra red blockchain (véase la Figura 5).

**b. Explorer**

El Explorer [5] es una interfaz web que analiza la red blockchain y nos permite ver qué transacciones se han creado, los emisores y receptores de estas transacciones, las llamadas al código subido en la red blockchain que se hacen, los bloques que se crean, cuantas transacciones contienen, etc. Realiza un análisis completo de la red blockchain (véase la Figura 6).

Esta página web se conectará a un nodo que hayamos creado, al cual tendremos que activarle los servicios RPC (Remote Procedure Call) y WebSocket [6]. Utiliza estos servicios para comunicarse con el nodo y obtener información. El Explorer utiliza el servicio RPC para obtener la siguiente información:

- Información sobre los bloques.

- Información sobre las transacciones.
- Información sobre las cuentas.

El servicio WebSocket lo utiliza para ser avisado de cuando se ha creado un nuevo bloque.

#### c. Wallet

Una Wallet [7] es una interfaz web, que podría montarse en un servidor, pero es recomendable por motivos de seguridad que cada usuario la descargue y ejecute en local. Se conectará a un nodo de la red blockchain y nos permitirá la creación de transacciones, la subida de código a la red blockchain y la interacción con este código (véase la Figura 7).

#### d. Faucet

El Faucet [8] es una página web simple donde indicas la dirección de una cuenta y el Faucet te envía criptomonedas, permitiendo así a cualquier persona probar la red blockchain. Se conectará a un nodo y es necesario tener una cuenta con criptomonedas disponibles y estas criptomonedas serán las que enviará el Faucet (véase la Figura 4).

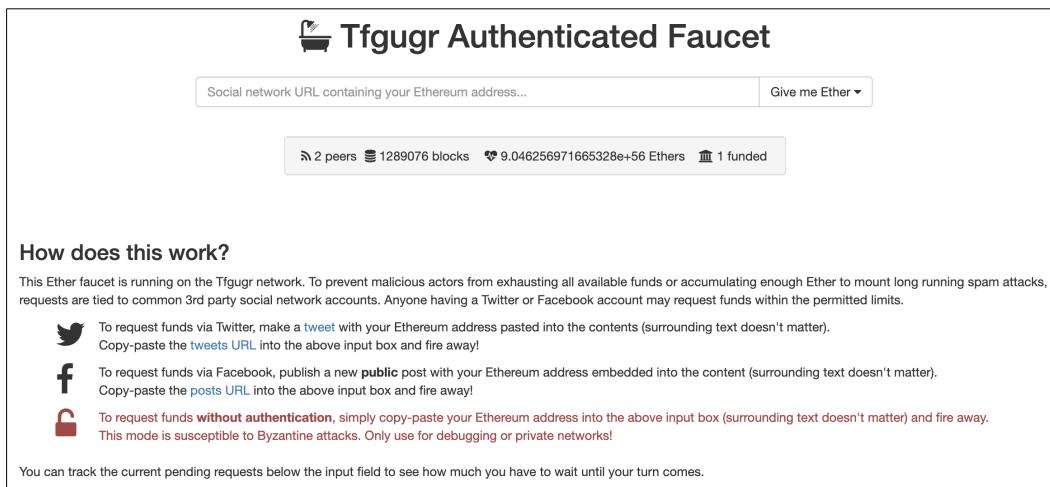


Figura 4: Faucet incluido en Puppeth

#### e. Dashboard

El Dashboard es una página web que permite acceder al Faucet, Wallet, Explorer y al Ethstats desde la misma página (véase la Figura 8).

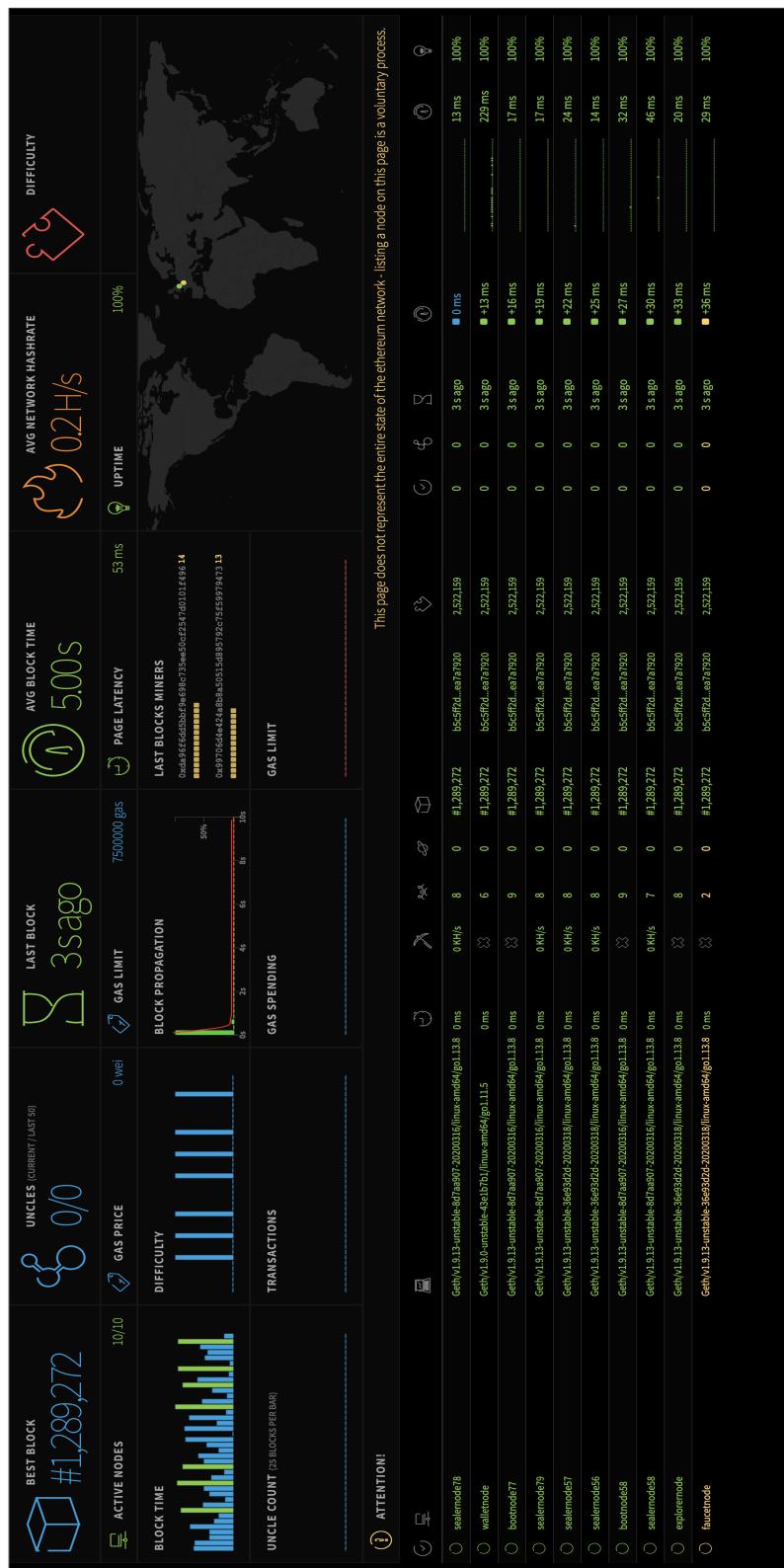


Figura 5: Ethstats incluido en Puppeth

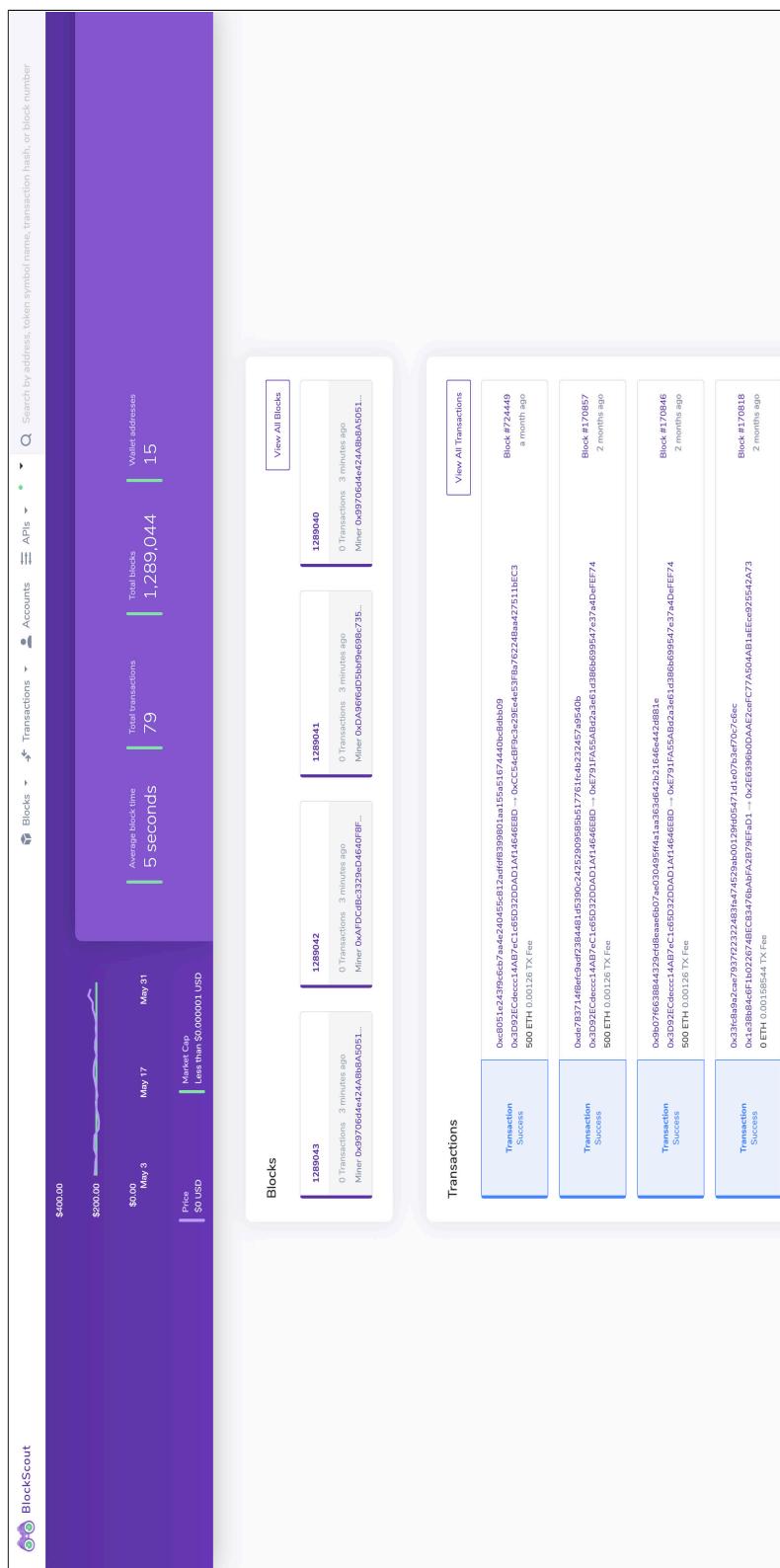


Figura 6: Explorer incluido en Puppet

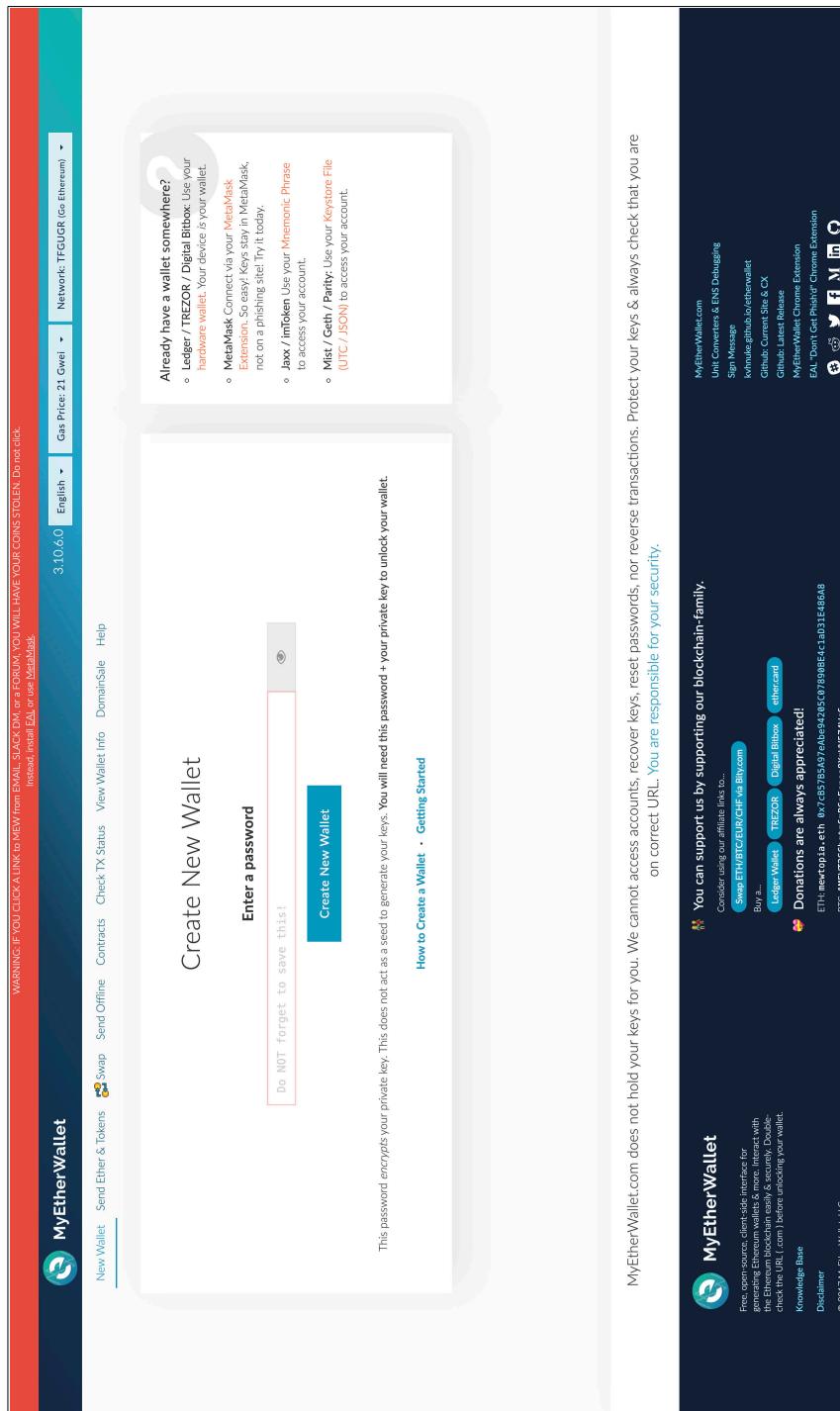


Figura 7: Wallet incluida en Puppeth

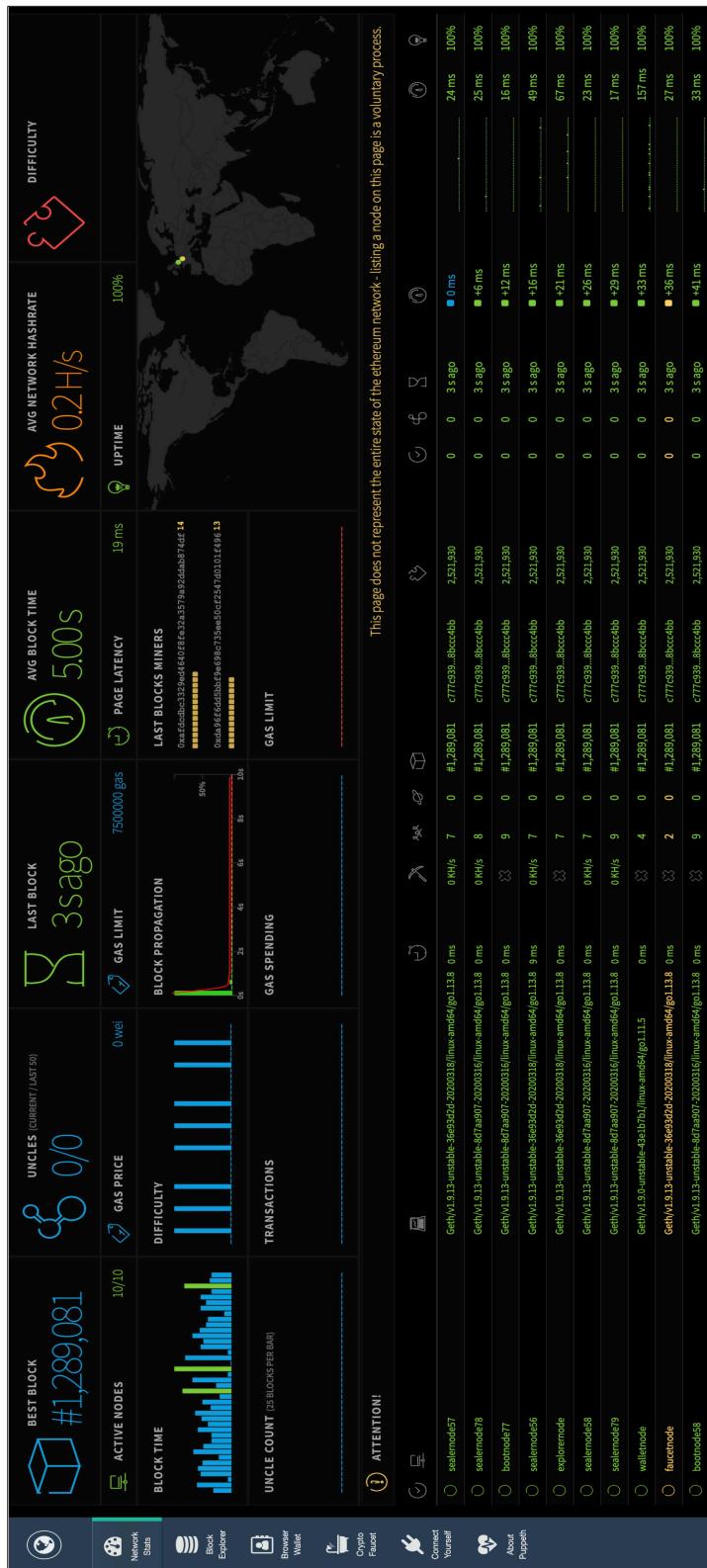


Figura 8: Dashboard incluido en Puppet

## 2.3. Protocolos para añadir bloques (Protocolos de consenso)

En este apartado explicaremos algunos de los protocolos más conocidos. La elección de un protocolo u otro dependerá del uso que queramos darle a la red blockchain creada y estará limitado por los protocolos disponibles en la tecnología que decidamos utilizar.

### 2.3.1. Proof of Work

La idea general en Proof of Work [9] es que existe un conjunto de nodos, que están intentando resolver un puzzle complejo y aquel que lo resuelva primero es el que añadirá un bloque. Para dar un ejemplo más concreto explicaremos la implementación de Proof of Work que utiliza bitcoin [10]:

El conjunto de nodos que intentan resolver el puzzle se llaman “mineros” y el proceso de resolver el puzzle se llama “minar”. El puzzle consiste en generar el valor hash de cada bloque como se ha mencionado antes, pero añadiendo una restricción, ese valor hash debe tener al principio un número determinado de bits con valor cero.

Para conseguir esto utilizan un valor llamado nonce, que van incrementando hasta que el hash resultante cumple la restricción y una vez conseguido el bloque se añade a la red blockchain.

Como minar implica utilizar recursos computacionales y consumir electricidad, existe un incentivo económico para motivar a los nodos, se recomienda al minero que ha añadido el bloque con criptomonedas.

### 2.3.2. Proof of Authority

En Proof of Authority [9] tenemos un conjunto de nodos especiales que son los únicos capaces de añadir bloques en la red blockchain, explicaremos a continuación la implementación que utiliza Ethereum [11].

En este caso a esos nodos especiales se les conoce como nodos signer y tienen dos restricciones a la hora de añadir bloques:

- Siguen el orden por el que aparecen en la lista ordenada de nodos autorizados, pero no es un orden fijo, lo que se hace es que si no es tu turno de añadir un bloque se retrasa el añadir un bloque una cantidad aleatoria de tiempo, pero si sí es tu turno no tienes ningún retraso. Y puede llegar a saltarse el turno de algún signer. Para determinar si es tu turno hace el siguiente cálculo:

BLOCK\_NUMBER % SIGNER\_COUNT == SIGNER\_INDEX

Siendo BLOCK\_NUMBER el número del bloque actual, SIGNER\_COUNT el número de nodos signer que hay y SIGNER\_INDEX el índice en la lista del signer actual. Si la igualdad se cumple, es el turno del signer SIGNER\_INDEX.

- Hay una variable SIGNER\_LIMIT cuyo valor es el resultado del siguiente calculo:

$\text{floor}(\text{SIGNER\_COUNT} / 2) + 1$

Cada signer solo podrá añadir un bloque de SIGNER\_LIMIT bloques consecutivos. (Si el valor de la variable es 3, entonces un signer añade el bloque 0, no puede añadir el bloque 1 y 2 pero el bloque 3 ya sí puede añadirlo).

### 2.3.3. Proof of Stake

La idea general en Proof of Stake [12] es que se elige aleatoriamente al nodo que añadirá el siguiente bloque basándose en la inversión (stake) que tiene cada nodo en la red blockchain, cuanta más inversión tiene más probabilidades de ser elegido. Esta inversión puede ser una mezcla de muchos factores como por ejemplo la cantidad de criptomonedas que tiene o la antigüedad de estas monedas.

## **Capítulo 3**

### **Estado del Arte**

En este capítulo hablaremos de algunas tecnologías blockchain destacadas en la actualidad, centrándonos principalmente en Bitcoin y Ethereum.

### 3.1. Bitcoin

Bitcoin [2] apareció por primera vez en 2008 en un documento [10] escrito por Satoshi Nakamoto, un seudónimo utilizado por una persona o grupo de personas, donde se describe un protocolo de comunicación que se podría llegar a utilizar para crear unas monedas virtuales (criptomonedas) y poder realizar pagos sin depender de una entidad intermediaria. Lo descrito en el documento pasaría a conocerse como la tecnología blockchain.

Acabó convirtiéndose en una de las criptomonedas más famosas y llegó a alcanzar el impresionante valor de 20.000 dólares por bitcoin en diciembre de 2017 [13] (véase la Figura 9).

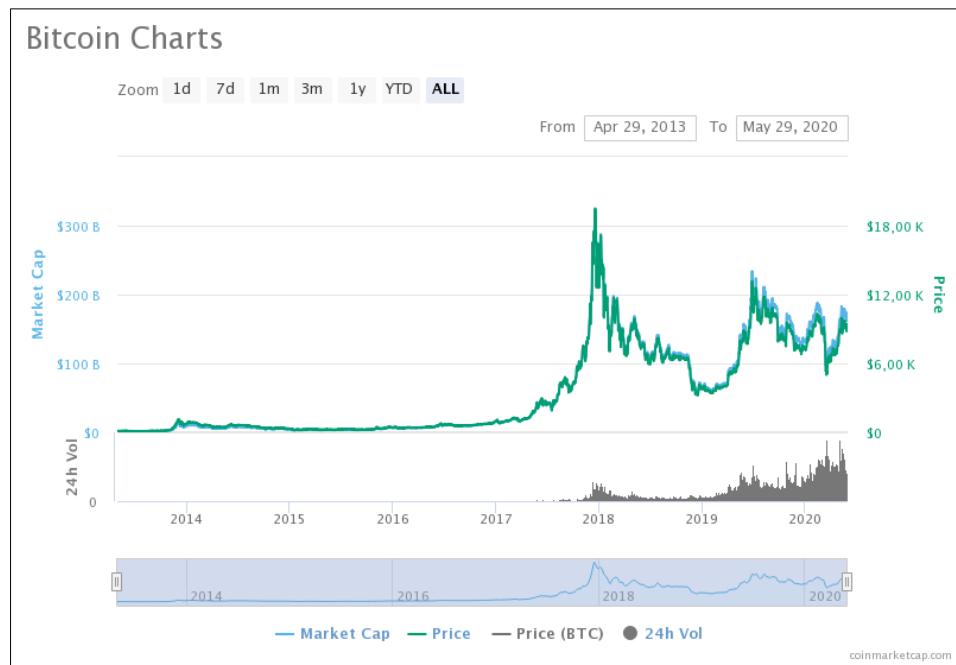


Figura 9: Gráfica mostrando la evolución del precio de Bitcoin [13]

Aunque es interesante mencionarlo por ser el origen de las redes blockchain, es cierto que su uso está limitado a criptomonedas y no se puede utilizar para la creación de sistemas distribuidos.

### 3.2. Ethereum

Ethereum [14] es una tecnología blockchain que tiene asociada una criptomonedas llamada Ether y que incluye una característica revolucionaria que la diferencia de Bitcoin: permite subir código a la red blockchain y ejecutar este código. Al código que subimos a la red blockchain se le conoce como smart contracts (Contratos inteligentes). Fue la primera tecnología blockchain en permitir la ejecución de smart contracts. Nos permite utilizar dos protocolos de consenso: Proof of Work y Proof of Authority.

Existen múltiples implementaciones de Ethereum [15], tanto oficiales como no oficiales, por lo que se debe de tener cuidado para evitar utilizar alguna versión que no se actualice con frecuencia.

Los contratos son programados en lenguajes de alto nivel [16] como Solidity o Viper, que luego son compilados y se convierten en código máquina ejecutable por la Ethereum Virtual Machine (EVM) [14]. Esta EVM puede utilizar tres zonas de memoria [17] diferentes durante la ejecución:

- Stack: La EVM no utiliza una arquitectura de registros, sino de pila, que utilizará para ejecutar todas las instrucciones máquina.
- Memoria: Cada vez que se hace una llamada a una función del contrato, se crea una nueva instancia de memoria para esta llamada y al terminar se borrará. Es lineal y podemos acceder por byte.
- Storage: Cada contrato dispone de un storage, un área de almacenamiento persistente donde puede guardar datos.

Para enviar transacciones en Ethereum necesitamos utilizar una cuenta [14], las cuentas tienen como identificador (nombre de usuario) una dirección y existen dos tipos de cuentas, cuya diferencia está en quien las controla:

- Externally Owned Accounts: Estas cuentas son controladas por usuarios. Se utiliza para enviar Ether a otras cuentas o si el receptor de la transacción es una cuenta contrato, para ejecutar el código del contrato indicado.
- Contract Accounts: Estas cuentas son controladas por los contratos, es decir, identifican a un contrato que está subido en la red blockchain.

Cuando creamos una cuenta nueva [18] se nos pedirá una contraseña y se generará un archivo keystore, con información encriptada de la cuenta. Para poder utilizar la cuenta se necesita saber la dirección, la contraseña utilizada y tener el archivo keystore.

### 3.3. Otras tecnologías blockchain en la actualidad

En la actualidad existen muchas tecnologías blockchain que han seguido los pasos de Ethereum y permiten la ejecución de smart contracts. Vamos a hablar sobre dos de ellas que nos han parecido interesantes por los cambios que ofrecen: Quorum y Hyperledger Fabric.

#### 3.3.1. Quorum

Quorum [19] se caracteriza por estar más centrada en la creación de redes blockchain privadas, ofreciendo múltiples protocolos Proof of Authority y creando el concepto de transacciones privadas, solo ejecutables por nodos que tengan permiso para ello.

#### 3.3.2. Hyperledger Fabric

Hyperledger [20] es un proyecto colaborativo fundado por The Linux Foundation enfocado en el desarrollo de tecnologías blockchain.

Una de las redes blockchain que tienen entre manos es Hyperledger Fabric [21], la cual permite la programación de smart contracts en lenguajes como Java y tiene un enfoque modular, permitiendo un alto grado de configuración.

## **Capítulo 4**

# **Creación de una red blockchain basada en la plataforma Ethereum**

En este capítulo empezaremos por mencionar la tecnología blockchain que hemos elegido y justificaremos nuestra decisión, mencionaremos las herramientas utilizadas para la creación de la red blockchain, se explicará como se configuró el servidor y las máquinas virtuales, entraremos en detalle en la privacidad de una red blockchain y por último describiremos el proceso de creación y configuración de una red blockchain, de dos formas distintas.

#### **4.1. Elección de la tecnología blockchain**

Decidimos utilizar Ethereum de entre todas las tecnologías blockchain existentes en la actualidad por los siguientes motivos:

- Ethereum es la primera tecnología blockchain en ofrecer la posibilidad de ejecutar smart contracts.
- Tiene una gran cantidad de documentación, que facilita el trabajo de crearla y desarrollar una aplicación.
- Incluye un protocolo Proof of Authority que es perfecto para una red blockchain privada, como es este caso.
- Está en constante actualización, no es un proyecto abandonado y pese a ser antigua continúan innovando.

Dentro de Ethereum disponemos de varias implementaciones [15], destacamos tres de ellas por ser las únicas oficiales:

- Aleth [22] (Programado en C++)
- Geth [23] (Programado en Go)
- Trinity [24] (Programado en Python)

Utilizamos Geth, ya que de entre las tres implementaciones es la más desarrollada y está constantemente en actualización, aparte de tener mejor documentación.

A la hora de elegir el protocolo de consenso, Ethereum nos da dos opciones, Proof of Work y Proof of Authority. Como la red que se quiere crear es privada, el protocolo que mejor se adapta a nuestras necesidades es Proof of Authority, ya que todos los nodos que están participando en la red serán nodos que hayamos incluido y por tanto podemos confiar en ellos. De esta forma evitamos el consumo de recursos que supondría tener ordenadores minando.

## 4.2. Herramientas utilizadas

La red blockchain se creó utilizando Puppet y Geth. Todos los nodos se crearon en un mismo servidor, utilizando Proxmox para la creación de las máquinas virtuales.

Se eligió Proxmox debido a que alquilamos un servidor de la compañía Soyousart y por tanto estamos limitados en las opciones de sistemas operativos que podemos instalar. Soyousart nos ofrece los sistemas operativos que aparecen en la Figura 10.



Figura 10: Sistemas operativos que se pueden instalar

Teniendo en cuenta la lista de sistemas operativos, tenemos que elegir de entre las siguientes tres opciones que permiten la creación de máquinas virtuales:

- Citrix XenServer
- Proxmox

- VMware

Una de las condiciones que teníamos era que fuese open source, por lo que VMware quedó descartado. Lo que nos llevó a elegir Proxmox antes que Citrix fue que la documentación de Proxmox nos pareció más completa y fácil de seguir [25].

#### 4.2.1. Proxmox

Proxmox [26] es un entorno de virtualización open source que instala un panel de control web para facilitar su uso. Aparte de crear máquinas virtuales permite gestionar cluster, copias de seguridad, etc.

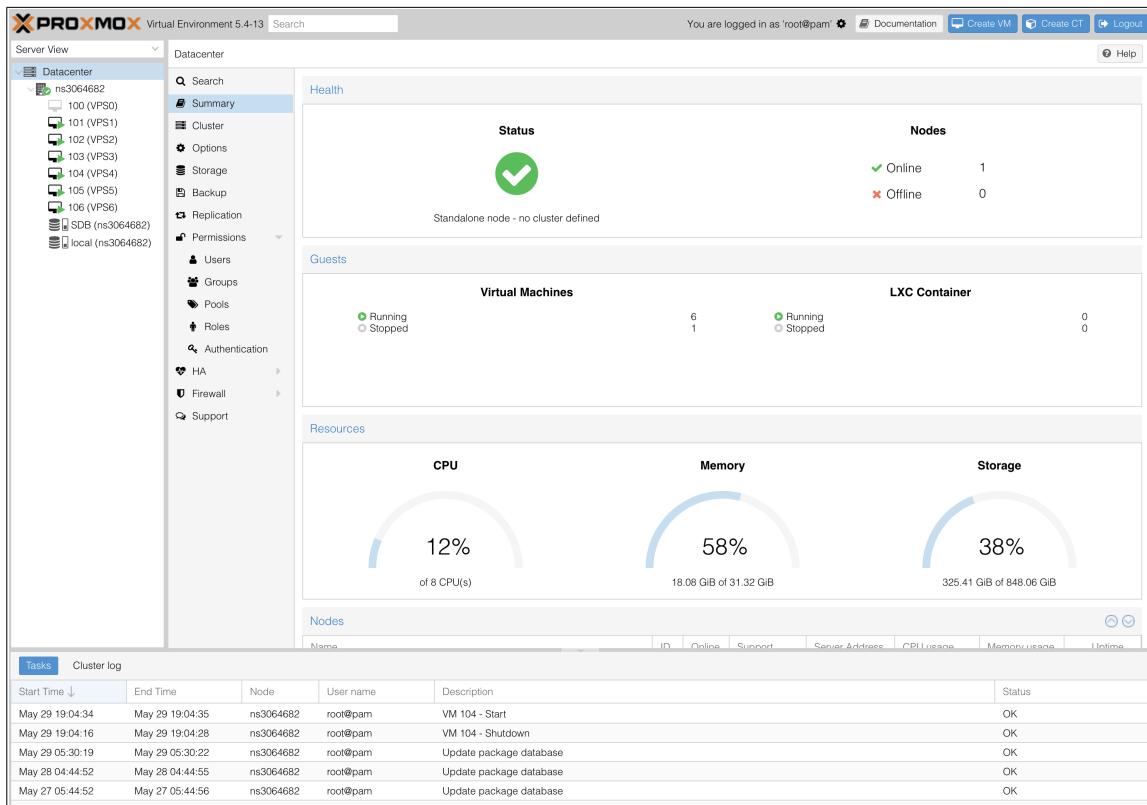


Figura 11: Interfaz web de Proxmox instalada en el servidor

#### 4.2.2. Puppeth

Puppeth [27] es un programa que busca simplificar el proceso de creación de una red blockchain Ethereum de principio a fin, encargándose de

la configuración del bloque génesis, creación de los nodos con las opciones adecuadas y creación de los servicios deseados incluyendo el servidor web para poder acceder a ellos.

De forma que lo único que tiene que hacer el usuario es ir respondiendo a las preguntas que hace Puppeth y este lo traducirá en las opciones adecuadas de configuración (véase la Figura 12).

Para crear los nodos y servicios es necesario conectarse a otros servidores, para ello utiliza SSH (Secure SHell). Cada componente de la red que crea está dentro de un Docker. Un contenedor Docker [28] se puede ver como una máquina virtual que permite aislar al software del entorno en el que se ejecuta.

			Signer account	0x99706d4e424A8b8A50515D895792c75f59979473
mano@51.178.122.157	51.178.122.157	dashboard	Ethstats service	188.165.116.177
			Explorer service	51.178.122.156
			Faucet service	188.165.116.179
			Wallet service	188.165.116.178
			Website address	51.178.122.157
			Website listener port	80
		nginx	Shared listener port	80
		sealnode	Data directory	/home/mano/sealerData
			Ethstats username	sealernode57
			Gas ceil (target maximum)	10.000 MGas
			Gas floor (baseline target)	7.500 MGas
			Gas price (minimum accepted)	0.000 GWei
			Listener port	30303
			Peer count (all total)	50
			Peer count (light nodes)	0
			Signer account	0xAFDCdBc3329eD4640F8Fe32a3579A92ddAb874Df
mano@51.178.122.158	51.178.122.158	bootnode	Data directory	/home/mano/bootData
			Ethstats username	bootnode58
			Listener port	30303
			Peer count (all total)	512
			Peer count (light nodes)	256
		sealnode	Data directory	/home/mano/sealerData
			Ethstats username	sealernode58
			Gas ceil (target maximum)	10.000 MGas
			Gas floor (baseline target)	7.500 MGas
			Gas price (minimum accepted)	0.000 GWei
			Listener port	30304
			Peer count (all total)	50
			Peer count (light nodes)	0
			Signer account	0x6f0bA1cC42Febc57341BC7D27C7Ef66c5b35B42E

```

What would you like to do? (default = stats)
1. Show network stats
2. Manage existing genesis
3. Manage tracked machines
4. Manage network components
> █

```

Figura 12: Menú principal de Puppeth

#### 4.2.3. Geth

Geth (Go Ethereum) [23] es la implementación de Ethereum que decidimos utilizar, principalmente se usa para creación de nodos e incluye una gran cantidad de opciones y una serie de utilidades, en la Figura 13 se mue-

tran las utilidades que incluye.

```

NAME:
  geth - the go-ethereum command line interface

  Copyright 2013–2019 The go-ethereum Authors

USAGE:
  geth [options] command [command options] [arguments...]

VERSION:
  1.9.14-stable-6d74d1e5

COMMANDS:
  account           Manage accounts
  attach            Start an interactive JavaScript environment (connect to node)
  console           Start an interactive JavaScript environment
  copydb            Create a local chain from a target chaindata folder
  dump              Dump a specific block from storage
  dumpconfig        Show configuration values
  dumpgenesis       Dumps genesis block JSON configuration to stdout
  export             Export blockchain into file
  export-preimages  Export the preimage database into an RLP stream
  import             Import a blockchain file
  import-preimages  Import the preimage database from an RLP stream
  init               Bootstrap and initialize a new genesis block
  inspect            Inspect the storage size for each type of data in the database
  js                 Execute the specified JavaScript files
  license            Display license information
  makecache          Generate ethash verification cache (for testing)
  makedag            Generate ethash mining DAG (for testing)
  removedb          Remove blockchain and state databases
  retesteth          Launches geth in retesteth mode
  show-deprecated-flags Show flags that have been deprecated
  version            Print version numbers
  wallet             Manage Ethereum presale wallets
  help, h            Shows a list of commands or help for one command

```

Figura 13: Salida del comando Geth con la opción help

El uso principal que se le dará a Geth es el de la creación de nodos. En el apartado “2.2.2. Tipos de nodos” hablábamos de los tipos de nodos que hay y para que se utilizan, ahora explicaremos como utilizar Geth para crear los distintos tipos de nodos.

Existe un paso previo que hay que hacer siempre antes de crear un nodo, que es inicializar el directorio de datos que utilizará el nodo [3]. Para conseguir esto se ejecuta Geth [29] con las siguientes opciones:

```
$ geth init --datadir <directorio> <archivoGenesis.json>
```

Donde la opción --datadir indica cual es el directorio que inicializamos. Y archivoGenesis.json indica el archivo de configuración del bloque génesis de la red blockchain que vamos a utilizar.

#### a. Nodo básico

Si creamos un nodo sin darle ninguna opción extra se creará un nodo básico [3]. El comando de Geth [29] para crear este nodo es:

```
$ geth --datadir <directorio> --networkid <idRed> --port <puerto>
  --bootnodes <enode>
```

En la opción --datadir indicamos el directorio previamente inicializado. En la opción --networkid indicamos la id de la red, se explicará lo que es y el valor que darle en el apartado “4.4. Privacidad de la red blockchain”. En la opción --bootnodes indicamos los bootnodes de nuestra red blockchain, utilizando lo que se conoce como enode.

Un enode [30] es un identificador de un nodo, que está dividido en dos partes. La primera de ellas es un identificador en hexadecimal del nodo, que está separado por el símbolo arroba de la segunda parte. Esta segunda parte es una dirección IP donde indicamos el puerto que utiliza el nodo para comunicarse con otros nodos. Un ejemplo genérico de enode sería el siguiente:

```
enode://<valorHexadecimal>@<ip>:<puerto>
```

#### b. Bootnode

Para crear un bootnode [3] le añadiremos a Geth [29] la siguiente opción:

```
--nat extip:<ipNodo>
```

Al añadir esta opción le estamos indicando al nodo su IP y aparte del comportamiento de un nodo básico tendrá la responsabilidad adicional de un bootnode.

El comando final será el siguiente:

```
$ geth --datadir <directorio> --networkid <idRed> --port <puerto>
--nat extip:<ipNodo>
```

#### c. Signer

Para crear un nodo signer [3] tendremos que utilizar una cuenta Ethereum de las incluidas en el bloque génesis para este propósito. Cómo incluir estas cuentas en el bloque génesis y la configuración de este se explican en el apartado “4.6.2. Configuración manual del bloque génesis”. Para crear el nodo mediante Geth [29] le añadimos la siguiente opción:

```
--unlock <direccionCuentaSigner> --mine
```

Con la opción --unlock estamos indicando la dirección de la cuenta a utilizar, que intentará desbloquear y con la opción --mine le indicamos que será uno de los nodos especiales que añadirán bloques en la red

blockchain. Este tipo de nodo solo aparece si utilizamos el protocolo Proof of Authority.

Si recordamos cómo funcionan las cuentas Ethereum, para poder desbloquearla y utilizarla necesitamos su archivo keystore, que debemos copiar en la carpeta keystore que se encuentra dentro del directorio de datos inicializado. Si no copiamos el archivo no se puede desbloquear la cuenta.

El comando completo sería el siguiente:

```
$ geth --datadir <directorio> --networkid <idRed> --port <puerto>
--bootnodes <enode> --unlock <direccionCuentaSigner> --mine
```

#### d. Miner

Para crear un nodo miner [3] tendremos que utilizar una cuenta Ethereum, solo que en este caso no es necesario que este incluida en el bloque génesis, para crearlo mediante Geth [29] le añadimos la siguiente opción:

```
--mine --miner.threads=1 --miner.etherbase=<direccionCuenta>
```

Ahora se indica la cuenta a utilizar mediante la opción --miner.etherbase y con la opción --miner.threads indicamos el número de hebras de la CPU (Central Processing Unit) a utilizar. Este tipo de nodo solo aparece si utilizamos el protocolo Proof of Work.

Como se ha mencionado antes cuando explicábamos Proof of Work (en el apartado “2.3.1. Proof of Work”), cuando se utiliza este protocolo, los nodos que minan bloques son recompensados con Ether por cada bloque que añaden, que se envía a la cuenta indicada.

En el caso de los nodos miner, no es necesario desbloquear ninguna cuenta, por lo que no hace falta copiar el archivo keystore.

El comando completo sería el siguiente:

```
$ geth --datadir <directorio> --networkid <idRed> --port <puerto>
--bootnodes <enode> --mine --miner.threads=<numHebras> --
miner.etherbase=<direccionCuenta>
```

Existe otra opción de Geth [29] que incluiremos en todos los nodos:

```
--ethstats='<nombrenodo>:<password>@<ip>:<puerto>'
```

Se utiliza para conectar los nodos al servicio Ethstats creado. Donde nombreNodo será el nombre con el que aparezca el nodo en Ethstats, Password es la contraseña que se le ha dado al servicio Ethstats al crearlo y ip:puerto es la IP del servidor donde tenemos Ethstats instalado y el puerto que utiliza.

### **4.3. Configuración del servidor**

Para la creación de la red se ha utilizado el siguiente servidor de la compañía Soyousart [31]:

<b>E3-SSD-1-32</b>		
<b>Características</b>		
<b>Procesador</b>	<b>Cores/Threads</b>	<b>Frecuencia</b>
Intel Xeon E3-1245v2	4c/8t	3,4GHz /3,8GHz
<b>RAM</b>	<b>Discos</b>	<b>Ancho de banda</b>
32GB DDR3 1333 MHz	SoftRAID 2x480GB SSD	250 Mbit/s
<b>Tráfico</b>	<b>IPv4</b>	<b>IPv6</b>
Ilimitado	1	/64
<b>Anti-DDoS</b>	<b>IP Failover</b>	<b>Espacio de backup</b>
Incluido	16 incluidas y hasta 128 opcionales (1,00 €/mes por IP + IVA)	100 GB (incluido) y hasta 10 TB opcional

Figura 14: Servidor utilizado de Soyousart [31]

La empresa Soyousart pone a disposición de sus usuarios una interfaz web (véase la Figura 15) mediante la cual puedes gestionar el servidor contratado, permitiendo instalar el sistema operativo deseado. Como ya se ha mencionado, vamos a instalar Proxmox.



Figura 15: Interfaz web para gestionar el servidor contratado de Soyoutstart

Una vez terminado el proceso de instalación, controlado por Soyousart, se nos envía por correo la contraseña del usuario root y la dirección IP del servidor junto con el puerto utilizado por la interfaz web de Proxmox.

El siguiente paso fue montar las máquinas virtuales [25]. Para montarlas necesitamos utilizar un storage, que es la abstracción que utiliza Proxmox para almacenamiento de archivos. Existe un storage creado por defecto por Proxmox cuyo nombre es “local”. Tendremos que subir al storage local la imagen ISO (International Organization for Standardization) del sistema operativo que queremos utilizar.

Una vez subida la ISO, haremos clic en la opción Create VM (Virtual Machine), en la esquina superior derecha de la Figura 16, a continuación indicaremos las opciones que se han modificado de la configuración de la máquina virtual:

- General: Le damos un nombre a la máquina virtual, el nombre que le hemos ido dando es VPS<Número>(Virtual Private Server).
- OS (Operating System): Seleccionamos la ISO que hemos guardado en el storage local. Se ha instalado Ubuntu Server 18.04.4.
- System: Activamos Qemu Agent, que permitirá a Proxmox mostrar información adicional y realizar ciertas acciones de forma más eficiente.
- Hard Disk: Dejamos las opciones por defecto. Con 32 GB es suficiente para su propósito.
- CPU: Decidimos aumentar el número de cores ya que un solo socket con un core nos parecía muy poco, por lo que lo cambiamos a 2 sockets y 2 cores.
- Memory: Cambiamos los 512 MB por defecto a 4 GB.
- Network: Dejamos las opciones por defecto, aunque luego tocará cambiar la dirección MAC (Media Access Control).

Iniciamos la VM creada y terminamos la instalación de Ubuntu. Para encenderla haremos clic en Start y para entrar en ella clic en Console. (En la esquina superior derecha de la Figura 16).

Es importante que durante la instalación seleccionemos la opción Install OpenSSH Server. El resto de opciones las dejamos por defecto.

El siguiente paso será configurar la red de la máquina virtual, para poder acceder mediante SSH. Para realizar la configuración de la red se ha seguido un tutorial creado por la empresa OVH [32].

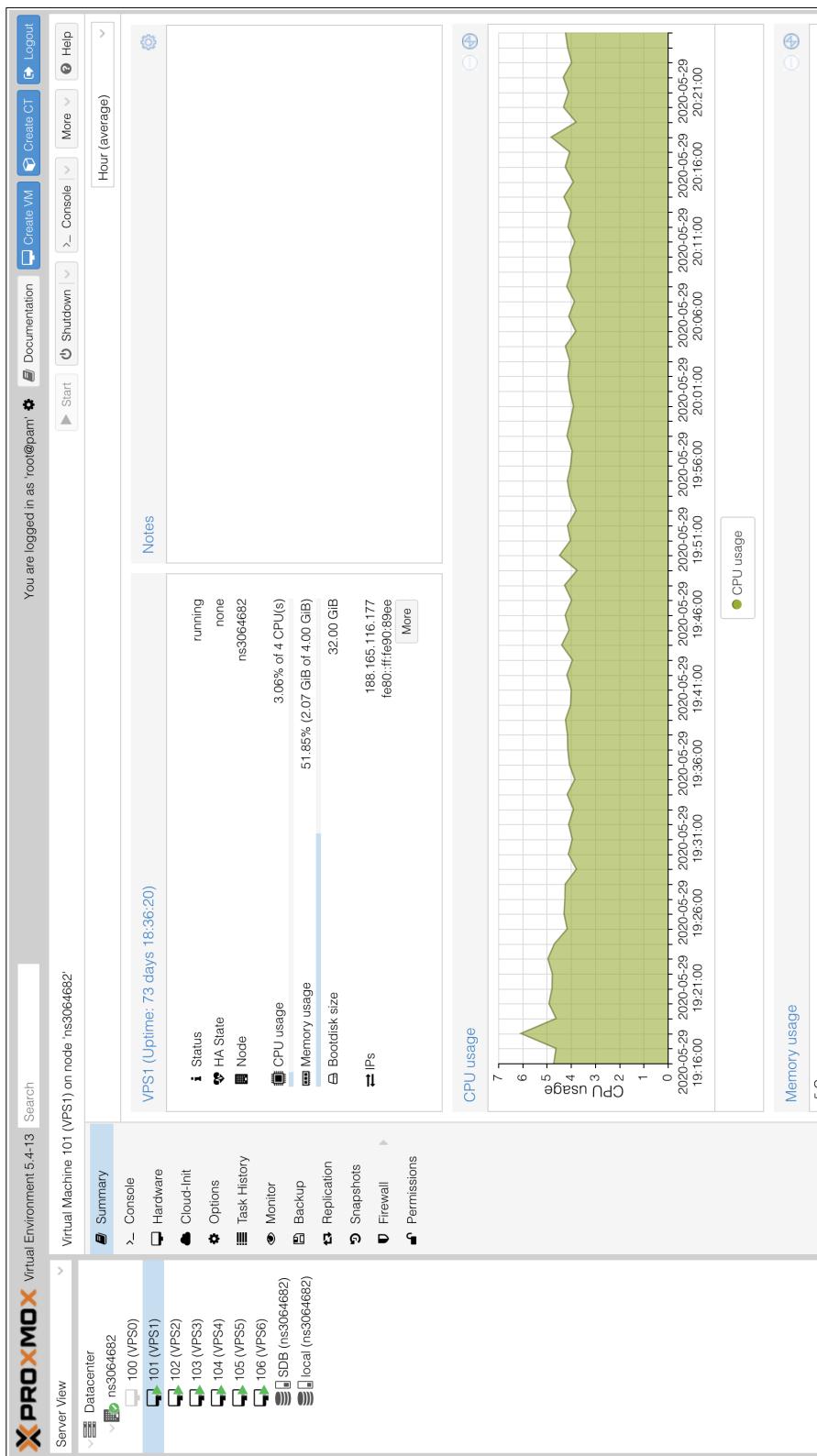


Figura 16: Interfaz de Proxmox en el servidor, mostrando el menú de una máquina virtual

Consiste en modificar el archivo “/etc/netplan/50-cloud-init.yaml”, de forma que quede así:

```
network:
  ethernets:
    (interface-name):
      addresses:
        - FAILOVER_IP/32
      nameservers:
        addresses:
          - 213.186.33.99
      search: []
      optional: true
      routes:
        - to: 0.0.0.0/0
          via: GATEWAY_IP
          on-link: true
  version: 2
```

Figura 17: Captura de pantalla del tutorial de OVH [32]

Previo a este paso tenemos que contratar un grupo de IP de Soyousart y crear una MAC virtual para cada una de ellas, que serán las que utilicemos en este paso (véase la Figura 18).

The screenshot shows a web-based management interface for IP addresses. At the top, there's a header bar with 'IP' on the left and 'Manage IPs' on the right. Below this is a navigation bar with 'Back to dashboard', 'Service' (set to 'ns3064682.ip.91.121.82.eu'), 'Failover IP' (set to '91.121.82.196'), 'Manage IPs' (selected), 'My organisations', 'Order IPs', and a help icon.

The main area is titled 'Manage IPs' and contains a table with the following data:

	IP address	Country	Reverse	Virtual MAC	Alerts
<b>+</b>	2001:4d0:187c4::/64	-	-	-	<span style="color: green;">●</span>
<b>-</b>	91.121.82.196	-	-	-	<span style="color: green;">●</span>
<b>-</b>	188.165.116.176/30	<span style="background-color: #e6f2ff; border: 1px solid black; padding: 2px;">FO</span>	ES	ip 176.ip-188-165-116.eu.	<span style="color: green;">●</span>
	188.165.116.176	ES	ip 176.ip-188-165-116.eu.	02:00:00:3ctb:4b	<span style="color: green;">●</span>
	188.165.116.177	ES	ip 177.ip-188-165-116.eu.	02:00:00:90:89:ee	<span style="color: green;">●</span>
	188.165.116.178	ES	ip 178.ip-188-165-116.eu.	02:00:00:df8c:6b	<span style="color: green;">●</span>
	188.165.116.179	ES	ip 179.ip-188-165-116.eu.	02:00:00:42:db:f7	<span style="color: green;">●</span>
<b>-</b>	51.178.122.156/30	<span style="background-color: #e6f2ff; border: 1px solid black; padding: 2px;">FO</span>	ES	ip 156.ip-51-178-122.eu.	<span style="color: green;">●</span>
	51.178.122.156	ES	ip 156.ip-51-178-122.eu.	02:00:00:40:aac:3c	<span style="color: green;">●</span>
	51.178.122.157	ES	ip 157.ip-51-178-122.eu.	02:00:00:5a:22:46	<span style="color: green;">●</span>
	51.178.122.158	ES	ip 158.ip-51-178-122.eu.	02:00:00:83:56:20	<span style="color: green;">●</span>
	51.178.122.159	ES	ip 159.ip-51-178-122.eu.	02:00:00:dd:01:32a	<span style="color: green;">●</span>

Figura 18: Panel de configuración de IP de Soyoutstart

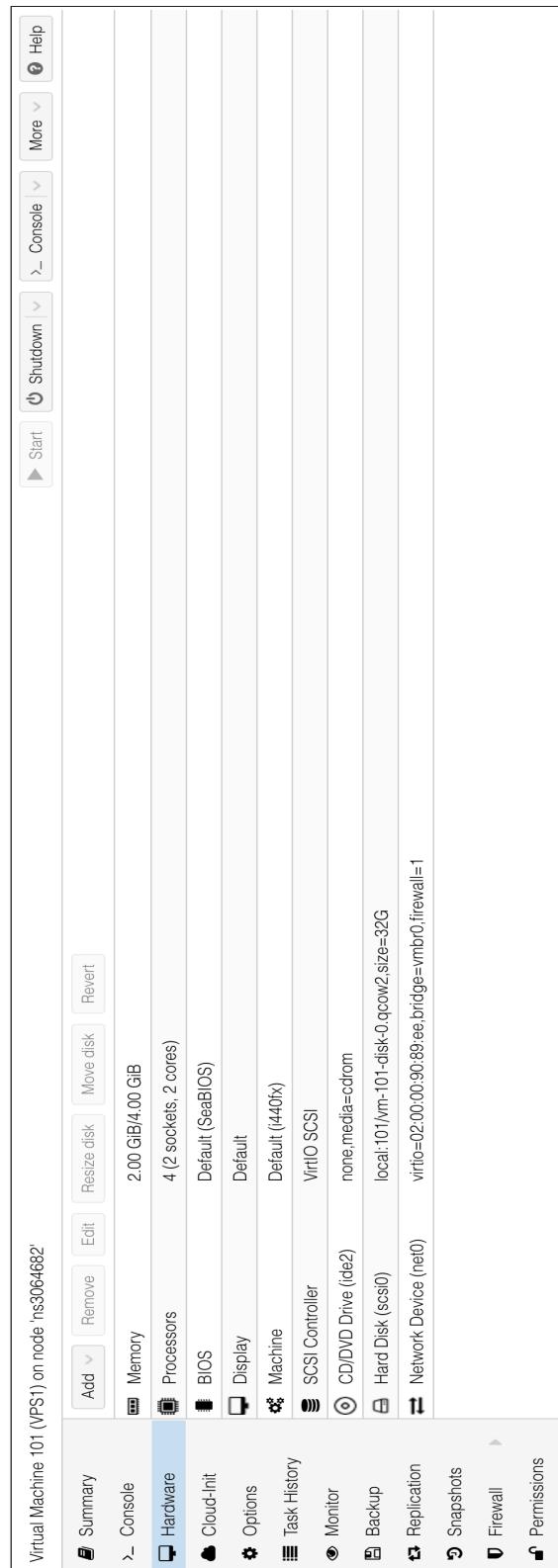


Figura 19: Menú de una máquina virtual de la interfaz de Proxmox, dentro de la opción Hardware.

Para asignarle una dirección MAC a una máquina virtual, iremos al menú de cada máquina virtual en Proxmox y hacemos clic en Hardware, después en Network Device y cambiaremos la dirección MAC (véase la Figura 19).

Una vez cambiadas las direcciones MAC, entramos en las máquinas virtuales mediante Console para añadir las IP contratadas. Modificamos la plantilla del tutorial de la siguiente forma:

- FAILOVER\_IP: Una de las IP contratadas.
- GATEWAY\_IP: La IP del servidor Proxmox, cambiando el último número de la IP por 254.

Una vez modificado el archivo ejecutamos el siguiente comando para que se apliquen los cambios realizados:

```
$ sudo netplan try
```

Confirmamos desde otro ordenador que la configuración ha sido un éxito y podemos acceder mediante SSH a la máquina virtual.

Ahora instalaremos todo el software que necesitamos en la máquina virtual para luego clonarla y así tendremos todas las máquinas virtuales necesarias para la red blockchain. A cada máquina debemos cambiarle la dirección MAC y volver a modificar el archivo “50-cloud-init” con la IP correcta.

Necesitaremos instalar en cada máquina:

- Qemu Agent [33]: Necesitamos instalarlo para poder utilizar el Qemu Agent que habíamos activado antes en el proceso de configuración. Para instalarlo ejecutamos el siguiente comando:

```
$ sudo apt-get install qemu-guest-agent
```

- Docker Engine [28]: Necesitamos instalarlo para que Puppeth pueda crear contenedores Docker, que permiten aislar al software del entorno de ejecución. Para instalar Docker Engine ejecutamos los siguientes comandos:

```
$ sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
gnupg-agent \
software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
    apt-key add -

$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Con esto ya hemos instalado Docker Engine y se ha creado el grupo de usuarios docker. Podemos añadir usuarios a este grupo y así evitar tener que utilizar sudo. En nuestro caso tendremos que hacerlo obligatoriamente, para que Puppeth pueda utilizar Docker. El comando para añadir un usuario es el siguiente [34]:

```
$ sudo usermod -aG docker $USER
```

- Docker Compose [35]: Es una herramienta que facilita la gestión de contenedores Docker. Es necesario instalarlo ya que Puppeth lo utiliza. Para instalar Docker Compose ejecutamos los siguientes comandos:

```
$ sudo curl -L "https://github.com/docker/compose/releases/
    download/1.25.5/docker-compose-$(uname -s)-$(uname -m)" -o /
    usr/local/bin/docker-compose

$ sudo chmod +x /usr/local/bin/docker-compose
```

- Geth [36]: Se utilizará para crear los distintos nodos que formarán nuestra red blockchain. Al instalar Geth, se nos instalará también otras herramientas adicionales como Puppeth. Para instalar Geth ejecutamos los siguientes comandos:

```
$ sudo add-apt-repository -y ppa:ethereum/ethereum
$ sudo apt-get update
$ sudo apt-get install ethereum
```

Ya tenemos nuestra máquina virtual configurada y la clonamos, haciendo clic en More y luego en Clone (esquina superior derecha de la Figura 16). La

clonaremos un total de seis veces, por lo que tendremos la máquina virtual repetida siete veces. Dejaremos una de copia de seguridad, por si queremos empezar de cero y las otras seis restantes se utilizarán para crear nodos. En cada una de ellas crearemos dos componentes, ya sean tipos de nodos o servicios, para conseguir un total de cinco nodos signer, dos nodos bootnode, un servicio Wallet, que se conectará a un nodo básico que está en la misma máquina virtual, un servicio Faucet con su correspondiente nodo básico y por último un servicio Explorer con otro nodo básico.

Aunque hemos incluido todo tipo de nodos y servicios, los únicos tipos de nodos necesarios para conseguir una red blockchain funcional son los nodos de tipo signer (o miner en caso de Proof of Work).

Una vez clonadas las máquinas, deberemos cambiar las direcciones MAC y modificar sus IP como se ha explicado antes, para hacerlas accesibles desde el exterior.

#### **4.4. Privacidad de la red blockchain**

Antes de continuar con la creación de la red blockchain, es importante saber cómo podemos garantizar que nuestra red blockchain sea efectivamente privada. Para empezar, nos centraremos en el protocolo de descubrimiento de nodos [37] que se utiliza. Por cómo funciona, es posible que un nodo malicioso que quisiese acceder a nuestra red empezase a comunicarse con uno de nuestros nodos y pasase a propagarse la información de este nodo malicioso al resto de nodos de nuestra red. Dependiendo del protocolo de consenso utilizado en la red el daño que podría causar sería diferente:

- En el caso de una red Proof of Work, ese nodo malicioso podría minar y por tanto añadir bloques a nuestra red.
- En el caso de una red Proof of Authority, no podría añadir bloques nuevos, ya que para ello hace falta tener acceso a una cuenta signer incluida en el bloque génesis [3] o poder acceder al resto de nodos signer para comenzar una votación e incluir un nuevo signer, cosa que no podría realizar ya que es necesario usar la API clique [38], para lo cual necesitaría acceder al servidor y entrar en la consola del nodo [39]. Se podría acceder a la API clique desde fuera si tuviésemos los puertos RPC abiertos [40] con esta API incluida, algo que debemos evitar.

Para evitar que ese nodo malicioso entre en nuestra red, tendremos que realizar una de estas dos opciones:

- Podríamos utilizar la opción --netrestrict [3], a la cual le indicamos una o varias direcciones IP, incluyendo máscaras CIDR (Classless Inter-Domain Routing) de forma que el nodo solo aceptará comunicaciones del rango de IP indicado.
- Se podría utilizar también una combinación entre la opción --nodiscover [29] y el método de la API admin [41] admin.addPeer(). Mediante la opción --nodiscover desactivamos el protocolo de descubrimiento de nodos, por lo que debemos añadir los nodos de forma manual. Para añadir nodos de forma manual se utiliza el método addPeer() mencionado. Si fuésemos por este camino, nos ahorraríamos el uso de los bootnodes, ya que estaríamos haciendo nosotros su trabajo de forma manual.

Otra amenaza a la seguridad en las redes blockchain es lo que se conoce como Replay attack [42], que consiste en la grabación de una comunicación, para repetirla posteriormente. Aplicándolo al contexto de las redes blockchain sería repetir una transacción que ha ocurrido en una red blockchain Ethereum en otra, de manera que si el atacante tiene cuentas en ambas redes blockchain (cuenta B) y es una transferencia de dinero de una cuenta A a la cuenta B, si tiene suerte y existe una cuenta A en la otra red blockchain podría obtener el doble de criptomonedas, de dos redes blockchain distintas.

Para solucionar este problema se utiliza la chainId (la id de la red blockchain) [43] que es un número entero utilizado en el proceso de firma de transacciones, de forma que dos transacciones idénticas en redes blockchain Ethereum distintas tendrán distinto hash.

A parte de lo mencionado antes, debemos tener en cuenta los dos siguientes aspectos:

- networkId: La id de la red [3] es un número entero que le asignamos a cada nodo mediante la opción --networkid de Geth. Dos nodos solo se comunicarán si tienen la misma id de red.
- Bloque génesis: Para que dos nodos de una red blockchain se conecten, aparte de la misma id de red necesitan haber sido inicializados con el mismo bloque génesis [3].

Juntando toda esta información, con elegir un método para evitar la intrusión de nodos maliciosos y asignarle un valor aleatorio a la id de red e id de la red blockchain sería suficiente para garantizar que nuestra red sea privada. Le daremos a ambas id el mismo valor.

## 4.5. Creación de la red blockchain utilizando Puppeth

Para la creación de la red blockchain mediante Puppeth utilizamos el siguiente tutorial de Medium [44], menos las partes relacionadas con la configuración de servidores AWS (Amazon Web Services), la compilación de Go y la compilación de Ethereum.

Hay que tener en cuenta que las tecnologías blockchain, incluido Ethereum, están en constante actualización, por lo que el tutorial mencionado y los tutoriales en general se desactualizan con facilidad y contienen información que ya no es cierta. Se ha seguido el tutorial indicado pero adaptado a las versiones actuales del software.

El primer paso será crear las cuentas, que haremos en nuestro ordenador. Para ello primero creamos un archivo con la contraseña que utilizaremos para las cuentas. Tendremos que descargar Geth para nuestro ordenador si no lo tenemos ya. Crearemos un total de diez cuentas.

A cada una de estas cuentas le asociaremos un rol excepto a dos cuentas que tendremos de reserva, sin ningún rol. Utilizaremos cinco cuentas para los nodos signer y tres cuentas serán pre-funded. Las cuentas pre-funded son cuentas que desde el principio tienen Ether.

El comando de Geth a ejecutar para crear una cuenta es el siguiente:

```
$ nano clave  
$ ./geth account new --password clave
```

La salida de ejecutar el comando es la siguiente:

```
manopc: geth manolo$ ./geth account new --password clave  
INFO [06-01|18:40:01.349] Maximum peer count  
ETH=50 LES=0 total=50  
Your new key was generated  
Public address of the key: 0x34425da30717136709Bac9EdFb478236512169DC  
Path of the secret key file: /Users/manolo/Library/Ethereum/keystore/UTC--2020-06-01T16-40-01.351365000Z--34425da30717136709bac9edfb478236512169dc  
- You can share your public address with anyone. Others need it to interact with you.  
- You must NEVER share the secret key with anyone! The key controls access to your funds!  
- You must BACKUP your key file! Without the key, it's impossible to access account funds!  
- You must REMEMBER your password! Without the password, it's impossible to decrypt the key!
```

Figura 20: Salida del comando geth account new

Es recomendable guardar en un archivo de texto la dirección de cada cuenta y la información contenida en el archivo JSON, o al menos el directorio en el que se encuentra, ya que más adelante se nos pedirá esta información.

Una vez tengamos las cuentas podremos ejecutar Puppeth y empezar el proceso de creación de la red blockchain.

```
What would you like to deploy? (recommended order)
1. Ethstats - Network monitoring tool
2. Bootnode - Entry point of the network
3. Sealer - Full node minting new blocks
4. Explorer - Chain analysis webservice
5. Wallet - Browser wallet for quick sends
6. Faucet - Crypto faucet to give away funds
7. Dashboard - Website listing above web-services
> █
```

Figura 21: Componentes que Puppeth permite instalar.

Para terminar de crear nuestra red blockchain realizaremos los siguientes pasos:

1. Configurar bloque génesis
2. Instalación de Ethstats
3. Instalación de los nodos bootnode
4. Instalación de los nodos signer
5. Instalación de los servicios opcionales (Explorer, Wallet, Faucet, Dashboard)

#### **4.5.1. Configurar bloque génesis**

En este apartado mencionaremos algunas de las opciones de configuración más importantes que tiene disponible el bloque génesis [3]:

- La id de la red blockchain (chainId).
- La elección del protocolo de consenso.
- Cuáles serán las cuentas pre-funded.
- En el caso de una red blockchain Proof of Authority quienes serán los nodos signer iniciales.
- El límite de gas de los bloques. Limita los cálculos que se pueden hacer en un solo bloque que sería equivalente a decir que limita el gas [45] total que se puede consumir en un solo bloque.

Está muy relacionado con la ejecución de código en la red blockchain, para provocar la ejecución de este código se envía una transacción. Y

en esta transacción especificamos la cantidad de gas que estamos dispuestos a utilizar y los Ether que estamos dispuestos a pagar por este gas. Si sobra gas se devuelve el dinero pagado por ese gas sobrante.

El gas es una unidad abstracta que es consumida por cada operación realizada en una transacción. Por ejemplo, sumar dos números o modificar una variable consumiría gas. Si durante la ejecución de una transacción nos quedamos sin gas, los cambios provocados por la ejecución se revierten.

El objetivo del gas es limitar el procesamiento que se realiza en una transacción, para evitar posibles ataques como podría ser un bucle infinito que dejase paralizado a los nodos y pagar por el consumo de recursos de la red blockchain (debemos tener en cuenta que esto se creó con una red pública en mente en la que todo el mundo puede participar).

Como nuestra red es privada, los problemas que arregla el uso del gas no existen (pues estaríamos pagándonos a nosotros mismos por el consumo de recursos que nosotros mismos damos) y como solo nosotros tenemos acceso no tenemos el problema de los posibles ataques de bucles infinitos, por tanto, el uso del gas no es un beneficio sino una limitación, ya que tenemos que asegurarnos de que las cuentas que utilicemos tengan Ether. Gracias a las cuentas pre-funded esto no supone ningún problema.

Las cuentas pre-funded por defecto se les da mucho Ether por Puppeth y nosotros manualmente podemos darle incluso más. Cuando hablamos de mucho estamos hablando de números del estilo  $9 \times 10^{56}$ . De esta forma solo tenemos que tener cuidado de utilizar las cuentas pre-funded.

Puppeth no nos permite modificar el límite de gas de los bloques, pero podemos modificarlo manualmente, como se verá en el apartado “4.6.2. Configuración manual del bloque génesis”.

Para crear un bloque génesis seleccionamos la opción 2 del menú principal de Puppeth (Configure new genesis). Y seleccionamos crear uno nuevo de cero.

Durante el proceso de configuración Puppeth nos irá haciendo preguntas y realizará los cambios necesarios al archivo JSON. En nuestra red blockchain utilizamos el protocolo Proof of Authority, crearemos bloques cada 5 segundos, tendremos un total de 5 nodos signer y 3 cuentas pre-funded.

A la hora de elegir cada cuanto tiempo creamos un bloque debemos tener en cuenta que significa añadir un bloque en la red blockchain. Al enviar una transacción a un contrato se ejecuta su código, el cual puede realizar

modificaciones sobre la información guardada, las cuales se escriben cuando se añade un bloque. Partiendo de esto debemos reflexionar sobre dos aspectos para elegir un tiempo de creación de bloques adecuado para nuestra situación:

- Dependiendo del gas que consuma cada transacción y del límite de gas de los bloques, necesitaremos una mayor o menor cantidad de bloques para procesar estas transacciones
- Si queremos que se hagan al instante o que tarden un tiempo nos interesará crear bloques con mayor frecuencia o no.

Elegimos crear bloques cada 5 segundos para que a la hora de programar nuestra aplicación no tengamos que esperar demasiado tiempo para ver los resultados de la ejecución, pero sin ser demasiado rápido para que se pueda apreciar que las modificaciones en la red blockchain no se aplican hasta que se añade el bloque.

#### **4.5.2. Instalación de Ethstats**

El siguiente paso será instalar Ethstats. Debemos instalarlo antes de crear los nodos ya que se informará a los nodos de su presencia para que se comuniquen entre ellos. Utilizamos Ethstats para monitorizar el estado de nuestra red blockchain.

Para crearlo mediante Puppeth elegiremos la opción 4, Deploy network components y elegimos Ethstats. Nos pedirá elegir en qué servidor instalarlo y como aún no hemos añadido ninguno habrá que hacerlo ahora. Al igual que con la configuración del bloque génesis, Puppeth nos irá guiando en la configuración del servicio.

#### **4.5.3. Instalación de los nodos bootnode**

Ahora debemos crear los nodos bootnode, que utilizamos para que, al añadir un nuevo nodo a nuestra red blockchain, no tengamos que ir informándole del resto de nodos que forman la red blockchain manualmente, el bootnode se encarga de hacer esto por nosotros.

Como siempre, Puppeth simplifica el proceso, solo tenemos que elegir la opción de crear un bootnode y responder a las preguntas que nos haga. Para nuestra red hemos creado dos nodos bootnode, ya que tener uno solo implica que si este falla, no podríamos añadir nuevos nodos hasta que lo arreglemos.

#### 4.5.4. Instalación de los nodos signer

Los nodos signer son los nodos especiales que solo existen en las redes Proof of Authority y son los encargados de añadir nuevos bloques. Puppeth utiliza el nombre sealer para hacer referencia tanto a los nodos miner como a los nodos signer y creará uno u otro dependiendo del protocolo de consenso elegido.

Es muy importante que si estamos utilizando Puppeth en la misma máquina donde vamos a instalar algún tipo de nodo, que no instalemos en esta un signer, si lo hacemos la red no funcionará. Parece ser un bug de Puppeth o de Geth.

Debemos tener el archivo JSON de la cuenta preparado, ya que Puppeth nos pedirá que copiemos el contenido del archivo. La cuenta debe de ser una de las que añadimos como signer en el bloque génesis.

#### 4.5.5. Instalación de los servicios opcionales (Explorer, Wallet, Faucet, Dashboard)

Ya se ha explicado previamente que es cada servicio, pero los servicios que incluye Puppeth lamentablemente están desactualizados e incluyen algunos bugs, a continuación comentaremos como podemos corregirlos.

**Explorer:** Es muy importante incluir un Explorer, ya que nos permitirá obtener información sobre las transacciones y bloques que hay en nuestra red blockchain. El Explorer incluido en Puppeth no se instala bien y no podemos acceder al servicio, el error que da es que no se ha indicado una contraseña para el superusuario de la base de datos y por tanto no se ha podido iniciar. Para solucionarlo [46] debemos modificar el código de Geth, más específicamente la línea 40 del archivo:

“go-ethereum/cmd/puppeth/module\_explorer.go”

La línea original es la siguiente:

```
echo '/usr/local/bin/docker-entrypoint.sh postgres &' >> explorer.sh  
&& \
```

Le añadimos la siguiente modificación:

```
echo 'POSTGRES_HOST_AUTH_METHOD=trust /usr/local/bin/docker-entrypoint  
.sh postgres &' >> explorer.sh && \
```

La solución hace que no pida contraseña y por tanto la base de datos se inicia sin problemas. Por motivos de seguridad debemos tener en cuenta que al hacer este cambio estamos permitiendo a cualquier contenedor que

se haya instalado en este mismo servidor el acceso a la base de datos del Explorer.

Si la solución anterior supone un problema, podemos darle una contraseña a la base de datos.

```
INFO [03-08|15:50:22.967] Ethereum protocol stopped
INFO [03-08|15:50:22.967] Transaction pool stopped
*****
WARNING: POSTGRES_HOST_AUTH_METHOD has been set to "trust". This will allow
anyone with access to the Postgres port to access your database without
a password, even if POSTGRES_PASSWORD is set. See PostgreSQL
documentation about "trust":
https://www.postgresql.org/docs/current/auth-trust.html
In Docker's default configuration, this is effectively any other
container on the same system.

It is not recommended to use POSTGRES_HOST_AUTH_METHOD=trust. Replace
it with "-e POSTGRES_PASSWORD=password" instead to set a password in
"docker run".
*****
INFO [03-08|15:50:23.518] Maximum peer count          ETH=50 LES=0 total=5
INFO [03-08|15:50:23.518] Smartcard socket not found, disabling    err="stat /run/pcscd"
```

Figura 22: Aviso de los posibles problemas de seguridad de la modificación realizada

Una vez hecho el cambio compilamos el código de Geth, ejecutamos Puppeth e instalamos de nuevo el servicio, que ya si funcionará. A fecha de 29 de abril de 2020 el bug aún no ha sido arreglado, un desarrollador de Ethereum se ha asignado el bug en GitHub por lo que es cuestión de tiempo que se arregle.

**Wallet:** Utilizamos la Wallet para realizar transferencias, crear contratos e interactuar con ellos. Dependiendo de si tenemos ya un programa que utiliza la API de Ethereum, es posible que no necesitemos incluir una Wallet. La versión que viene incluida en Puppeth no funciona correctamente con la creación de contratos, cuando se crea un contrato el nodo que la Wallet utiliza deja de sincronizar la red blockchain.

En este caso para arreglarlo decidimos directamente utilizar una versión más actualizada. Tenemos dos opciones, o bien instalar la versión moderna de la Wallet [7] o instalar la que ahora se conoce como versión vintage [47]. Ambas opciones tienen el bug arreglado.

Puppeth usa una versión desactualizada de la versión vintage y lo modifica para que en las opciones de redes blockchain que podemos utilizar solo aparezca nuestra red blockchain. Quisimos imitar lo que ellos hacían, pero no estaba claro cómo hacerlo en la versión moderna, en la versión vintage si conseguimos encontrar la parte del código que había que modificar y por eso fue la que utilizamos.

La Wallet la montaremos en un servidor web, para simplificar el probar

nuestra red blockchain creada. Hemos utilizado Apache. Este servicio necesita conectarse a un nodo, por lo que tendremos que crearlo. Pese a ser un nodo local, como queremos realizar esa modificación, necesitaremos abrir sus puertos RPC y hacerlo accesible desde el exterior, ya que no funciona con localhost.

Para abrir los puertos RPC [40] debemos añadir las siguientes opciones al comando Geth que inicia el nodo:

```
$ geth <opciones> --rpc --rpccaddr=0.0.0.0 --rpccorsdomain "*" 
```

Nos vamos al archivo “etherwallet-master.js” dentro de la carpeta js y buscaremos la parte del código donde aparecen las distintas opciones a las que podemos conectarnos, que es la sección que se muestra en la Figura 23. Podemos modificarlo para que se muestre únicamente nuestra red blockchain, para ello eliminamos todos los “nodos” que aparecen, que son las distintas opciones a las que podemos conectarnos y dejamos solo eth\_mew, que luego modificaremos para que se conecte a nuestra red blockchain, evitando que el usuario tenga que añadir nuestra red blockchain de forma manual y posibles confusiones que puedan ocurrir al existir varias opciones.

```

17167 nodes nodeList = {
17168   eth_mew: {
17169     name: "ETH",
17170     blockExplorerTX: "https://etherscan.io/tx/[[txHash]]",
17171     blockExplorerAddr: "https://etherscan.io/address/[[address]]",
17172     type: nodes.nodeType.ETH,
17173     eip155: true,
17174     chainId: 1,
17175     tokenList: require("./tokens/ethTokens.json"),
17176     abiList: require("./abiDefinitions/ethAbi.json"),
17177     service: "myetherwallet.com",
17178     lib: new nodes.customNode("https://api.myetherwallet.com/eth", "")
17179   },
17180   eth_etherscan: {
17181     name: "ETH",
17182     blockExplorerTX: "https://etherscan.io/tx/[[txHash]]",
17183     blockExplorerAddr: "https://etherscan.io/address/[[address]]",
17184     type: nodes.nodeType.ETH,
17185     eip155: true,
17186     chainId: 1,
17187     tokenList: require("./tokens/ethTokens.json"),
17188     abiList: require("./abiDefinitions/ethAbi.json"),
17189     service: "etherscan.io",
17190     lib: require("./nodeHelpers/etherscan")
17191   },
17192   eth_infura: {
17193     name: "ETH",
17194     blockExplorerTX: "https://etherscan.io/tx/[[txHash]]",
17195     blockExplorerAddr: "https://etherscan.io/address/[[address]]",
17196     type: nodes.nodeType.ETH,
17197     eip155: true,
17198     chainId: 1,
17199     tokenList: require("./tokens/ethTokens.json"),
17200     abiList: require("./abiDefinitions/ethAbi.json"),
17201     service: "infura.io",
17202     lib: new nodes.infuraNode("https://mainnet.infura.io/mew")
17203   },

```

Figura 23: Parte del código de etherwallet-master.js que hay que modificar

Modificaremos las opciones del “nodo” que queda de la siguiente forma para que se conecte a nuestra red blockchain:

- name: El nombre que queremos que aparezca.
- blockExplorerTX y blockExplorerAddr: La dirección de nuestro Explorer. Cuando la Wallet genere una transacción, dará un enlace que nos llevará a la transacción en el Explorer.
- chainId: La id de nuestra red blockchain, utilizada en el proceso de creación de transacciones.
- service: Al conectarse aparecerá en la página web “Provided by <nombre>”, con esta opción elegimos este nombre.
- lib: Aquí pondremos la dirección de nuestro nodo al que se conectará la Wallet y el puerto que utiliza. Como se ha mencionado antes, no funciona localhost por lo que debemos poner la IP completa, aunque nos estemos conectando a un nodo local.

En la Figura 24 se muestra el archivo con las modificaciones realizadas.

```

17160 nodes nodeList = {
17161   eth_mew: {
17162     name: "TFG",
17163     blockExplorerTX: "http://51.178.122.156",
17164     blockExplorerAddr: "http://51.178.122.156",
17165     type: nodes.nodeType.ETH,
17166     eip155: true,
17167     chainId: 52036,
17168     tokenList: require("./tokens/ethTokens.json"),
17169     abiList: require("./abiDefinitions/ethAbi.json"),
17170     service: "TFG",
17171     lib: new nodes.customNode("http://188.165.116.178", "8545")
17172   },
17173 };

```

Figura 24: Parte del código de etherwallet-master.js ya modificado

**Faucet:** Es una página web donde indicas una dirección de una cuenta y te envía Ether, es interesante incluirlo si tenemos pensado permitir a personas externas probar la red blockchain con facilidad, pues utilizando el servicio conseguirían Ether en su cuenta y podrían realizar transferencias, subir contratos e interactuar con ellos.

Otro servicio desactualizado, al principio funciona bien, pero acaba fallando. En este caso no encontramos ninguna otra versión reciente por lo que decidimos crear uno básico (véase la Figura 25), haciendo uso de una combinación de Node.js, la API de Ethereum Web3js y un nodo creado en el mismo servidor. En el capítulo 5, “Desarrollo de una aplicación descentralizada”, se habla en más detalle de la API.



Figura 25: Faucet básico

Para mostrarlo en funcionamiento hemos introducida una dirección y se muestra una captura de pantalla de la transacción en el Explorer:

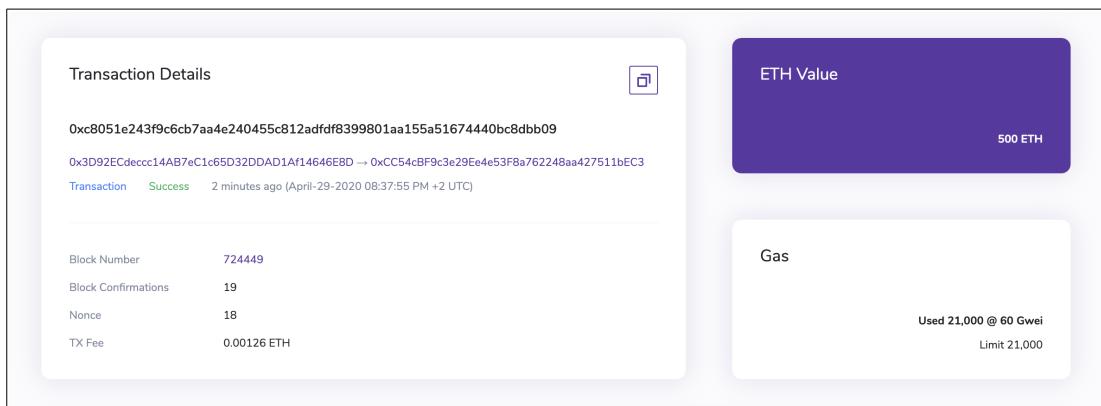


Figura 26: Transacción creada por el Faucet

**Dashboard:** Funciona a la perfección, nos permite acceder al Faucet, Wallet, Explorer y Ethstats. Es interesante incluirlo si tenemos pensado utilizar todos los servicios y cambiar con frecuencia de uno a otro, pues así los tendríamos todos en el mismo sitio.

En el apartado “4.4. Privacidad de la red blockchain” se habla de un par de opciones de Geth que hay que considerar para hacer que nuestra red sea completamente privada. Puppeth no incluye estas opciones, luego tendríamos que modificar el código para que las incluya y volver a compilar, como hicimos con el Explorer. Otra opción sería crear directamente la red de forma manual, obteniendo así completa libertad sobre las opciones a utilizar. El apartado siguiente explicará cómo crearla manualmente, con los comandos mínimos necesarios, sin opciones extras.

## **4.6. Creación de la red blockchain de forma manual**

En este apartado mostraremos como crear una red blockchain sin la ayuda de Puppeth. Mencionar que no hay ningún problema en mezclar nodos creados manualmente con nodos creados por Puppeth. La única diferencia entre ambos es que Puppeth utiliza un contenedor Docker para el proceso.

Al crearlo de forma manual decidimos aprovechar y buscar un Ethstats que encajase mejor con una red Proof of Authority en vez de utilizar el mismo que incluía Puppeth. Se utilizarán máquinas virtuales creadas mediante Proxmox en un servidor, igual que la red blockchain creada con Puppeth. Tendremos un Ethstats, un bootnode y dos nodos signer. Para conectarnos a los servidores y ejecutar los comandos se ha utilizado SSH.

### **4.6.1. Instalación de Ethstats**

Para instalar el nuevo Ethstats elegido se seguirán las instrucciones indicadas en su página de GitHub [48]. Necesitamos tener instalado Node.js. Los comandos a ejecutar para instalar Ethstats son los siguientes:

```
$ git clone https://github.com/goerli/ethstats-server  
$ cd ethstats-server  
$ npm install  
$ sudo npm install -g grunt-cli
```

Con esto ya tenemos el repositorio en nuestro servidor y las dependencias instaladas. Para crear los archivos estáticos se ejecuta el siguiente comando:

```
$ grunt poa
```

Por último, se utiliza el siguiente comando para ejecutar el servicio:

```
$ WS_SECRET="contraseña" npm start
```

La dirección por defecto es `http://<ip>:3000`. Para cambiar el puerto se utiliza una variable de entorno, el comando para ejecutarlo utilizando el puerto 80 sería el siguiente:

```
$ sudo WS_SECRET="contraseña" PORT=80 npm start
```

Ya podemos acceder al Ethstats desde el exterior (véase la Figura 27).

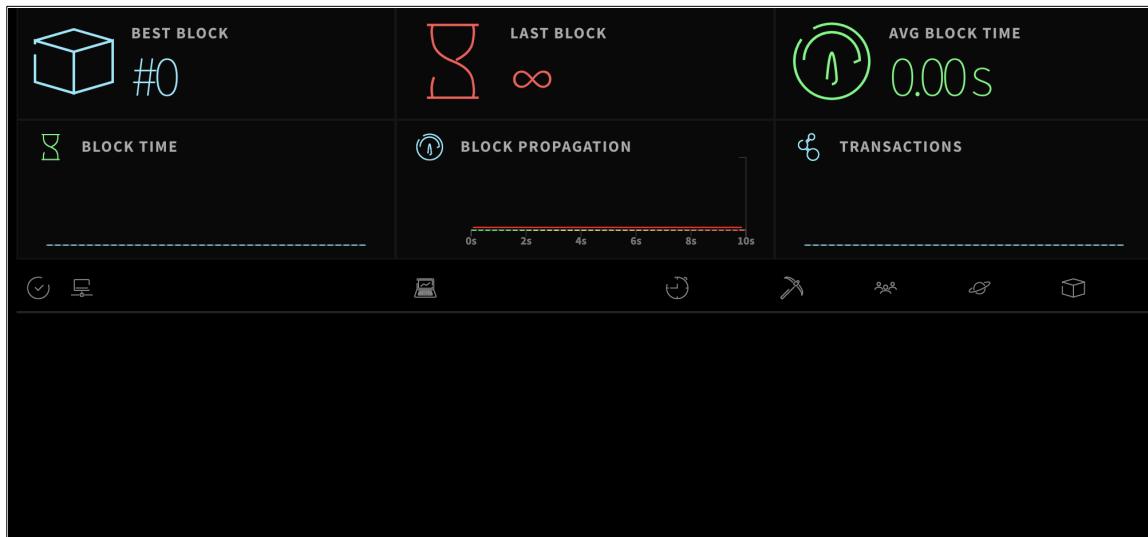


Figura 27: Captura de pantalla recortada del servicio Ethstats instalado

#### 4.6.2. Configuración manual del bloque génesis

Al igual que con Puppeth, antes de continuar con la creación de los nodos necesitamos configurar el bloque génesis.

Crearemos un archivo con formato JSON. Para ayudarnos en la configuración hemos partido del ejemplo mostrado en la documentación oficial de Geth [3].

El bloque génesis final queda así:

```

1  {
2    "config": {
3      "chainId": 1527,
4      "homesteadBlock": 0,
5      "eip150Block": 0,
6      "eip155Block": 0,
7      "eip158Block": 0,
8      "byzantiumBlock": 0,
9      "constantinopleBlock": 0,
10     "petersburgBlock": 0,
11     "clique": {
12       "period": 5,
13       "epoch": 30000
14     },
15   },
16   "difficulty": "1",
17   "gaslimit": "8000000",
18   "extradata": "0x0000000000000000000000000000000000000000000000000000000000000000DA",
19   "alloc": {
20     "23389B82d8cab10a7CaBFab6eA9Aa7c2B07dD899": { "balance": "300000" },
21     "CC54cBF9c3e29Ee4e53F8a762248aa427511bEC3": { "balance": "400000" }
22   }
23 }
```

Figura 28: Bloque génesis utilizado

Hemos modificado los siguientes campos:

- chainId: Le hemos dado un número aleatorio.
- Clique: Hemos añadido el apartado Clique, para indicar que utilice el protocolo Proof of Authority y period lo hemos dejado a 5, por lo que habrá un bloque nuevo cada 5 segundos.
- Extradata: Es un campo formado por treinta y dos 0, después las direcciones de las cuentas que utilizarán nuestros nodos signer y por último sesenta y cinco 0.
- Alloc: Aquí añadimos las cuentas que queremos que sean pre-funded y la cantidad de Ether que se les dará.
- gasLimit: Mediante este campo podemos modificar el límite de gas de los bloques.

Con estos cambios ya tenemos nuestro bloque génesis preparado.

#### **4.6.3. Instalación del bootnode**

El proceso de creación de un nodo bootnode, junto a los comandos a utilizar fueron explicados en el apartado “4.2.3. Geth”, ahora recordaremos detalles importantes y nos centraremos en dar ejemplos concretos, que fueron los utilizados para crear nuestra red blockchain. Se mostrará la versión general del comando y después la versión específica utilizada.

El primer comando por tanto que deberemos ejecutar es el que inicializa el directorio que utilizará el nodo, para esto necesitaremos tener el bloque génesis copiado (el archivo JSON) en todos los servidores donde montaremos un nodo, ya que es necesario para su inicialización. En la Figura 29 se muestra la salida que da la ejecución de este comando.

```
$ geth init --datadir <directorio> <archivoGenesis.json>
$ geth init --datadir ./boot tfg.json
```

```
[mano@vps1:~$ geth init --datadir ./boot tfg.json
INFO [05-04|17:15:16.840] Maximum peer count
INFO [05-04|17:15:16.840] Smartcard socket not found, disabling
INFO [05-04|17:15:16.847] Allocated cache and file handles
INFO [05-04|17:15:16.858] Writing custom genesis block
INFO [05-04|17:15:16.860] Persisted trie from memory database
INFO [05-04|17:15:16.860] Successfully wrote genesis state
INFO [05-04|17:15:16.860] Allocated cache and file handles
INFO [05-04|17:15:16.872] Writing custom genesis block
INFO [05-04|17:15:16.873] Persisted trie from memory database
INFO [05-04|17:15:16.873] Successfully wrote genesis state
ETH=50 LES=0 total=50
err="stat /run/pcscd/pc:
database=/home/mano/boo:
nodes=3 size=397.00B tir:
database=chaindata
database=/home/mano/boo:
nodes=3 size=397.00B tir:
database=lightchaindata
```

Figura 29: Salida del comando geth init.

Ahora creamos el tipo de nodo específico que queremos, en este caso un bootnode, utilizando como directorio de datos el indicado en el comando de inicialización e indicando como Ethstats el instalado previamente:

```
$ geth --datadir <directorio> --networkid <id> --nat extip:<ip> --
ethstats=<direccionEthstats>
$ geth --datadir ./boot --networkid 1527 --nat extip:51.254.147.197 --
ethstats='bootnode:contraseña@51.254.147.196:80'
```

Hemos inicializado el bootnode de forma correcta. En la Figura 30 mostramos parte de la salida que nos da la ejecución del comando anterior, es muy importante que nos fijemos en la línea “Started P2P networking”. En la parte derecha aparece “self=enode://...”, ahí tenemos indicado el enode del bootnode, que es lo que se copiará en la opción --bootnodes de Geth del resto de nodos creados, para indicarles cual es el bootnode. También se podría obtener el enode accediendo a la API admin [41] y utilizando nodeInfo.

## Creación de una red blockchain basada en la plataforma Ethereum 72

```
[mano@vps1:~$ geth --datadir ./boot --networkid 1527 --nat extip:51.254.147.197 --e
INFO [05-04|17:29:04.486] Maximum peer count                                ETH=50 LES=0 to
INFO [05-04|17:29:04.486] Smartcard socket not found, disabling           err="stat /run/
INFO [05-04|17:29:04.488] Starting peer-to-peer node                         instance=Geth/v
INFO [05-04|17:29:04.488] Allocated trie memory caches                      clean=256.00MiB
INFO [05-04|17:29:04.488] Allocated cache and file handles                  database=/home/
INFO [05-04|17:29:04.515] Opened ancient database                           database=/home/
INFO [05-04|17:29:04.516] Initialised chain configuration                   config="{ChainI
inople: 0 Petersburg: 0 Istanbul: <nil>, Muir Glacier: <nil>, Engine: clique}"
INFO [05-04|17:29:04.516] Initialising Ethereum protocol                     versions="[65 6
WARN [05-04|17:29:04.516] Upgrade blockchain database version                 from=<nil> to=7
INFO [05-04|17:29:04.518] Loaded most recent local header                  number=0 hash=a
INFO [05-04|17:29:04.518] Loaded most recent local full block                number=0 hash=a
INFO [05-04|17:29:04.518] Loaded most recent local fast block               number=0 hash=a
INFO [05-04|17:29:04.518] Regenerated local transaction journal             transactions=0
INFO [05-04|17:29:04.544] Allocated fast sync bloom                      size=512.00MiB
INFO [05-04|17:29:04.552] Stored checkpoint snapshot to disk            number=0 hash=a
INFO [05-04|17:29:04.554] Initialized fast sync bloom                    items=3 errorra
INFO [05-04|17:29:04.557] Mapped network port                          proto=tcp extpo
INFO [05-04|17:29:04.557] Mapped network port                    proto=udp extpo
INFO [05-04|17:29:04.558] New local node record                  seq=1 id=61925d
INFO [05-04|17:29:04.559] Started P2P networking                  self=enode://99
a9b00f3aab...8635ff6e3b7488051.254.147.197:30303
INFO [05-04|17:29:04.560] Stats daemon started
INFO [05-04|17:29:04.565] IPC endpoint opened                      url=/home/mano/
```

Figura 30: Salida del comando geth para crear el nodo.

Si nos fijamos en la Figura 31 veremos que aparece el bootnode creado.

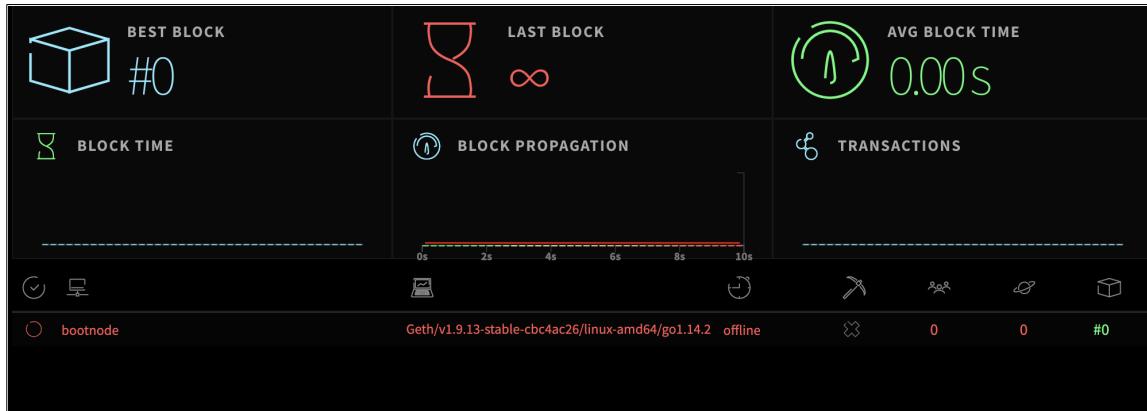


Figura 31: Captura de pantalla recortada del servicio Ethstats, con el bootnode creado.

#### **4.6.4. Instalación de los nodos signer**

El proceso de creación de un nodo signer, junto a los comandos a utilizar fueron explicados en el apartado “4.2.3. Geth”, ahora recordaremos detalles importantes y nos centraremos en dar ejemplos concretos, que fueron los utilizados para crear nuestra red blockchain. Se mostrará la versión general del comando y después la versión específica utilizada.

Al igual que en los nodos bootnode, el primer paso es inicializar el directorio que utilizará el nodo:

```
$ geth init --datadir <directorio> <archivoGenesis.json>
$ geth init --datadir ./sealer tfg.json
```

Ahora haremos uso de la opción --bootnodes, como se ha mencionado antes. También se utilizará la opción --unlock <direccionCuenta> y al igual que ocurría en Puppeth al crear un signer, es necesario utilizar el archivo JSON de la cuenta. Solo que en este caso no necesitamos su contenido, sino que tenemos que copiarlo en la carpeta keystore dentro del directorio que utiliza el nodo. Esta cuenta debe de ser una de las que añadimos en el bloque génesis como signer.

Por tanto, el comando de Geth para crear un signer es el siguiente:

```
$ geth --unlock <direccionCuenta> --mine --datadir <directorio> --
  networkid <id> --ethstats=<direccion> --bootnodes <enodeBootnodes>

$ geth --unlock 0xDA96f6dD5bbf9e698c735EE50CF2547d0101f496 --mine --
  datadir ./sealer --networkid 1527 --ethstats='sealer1:
  contrasena@51.254.147.196:80' --bootnodes enode://99
  a919347c5df544565cbc...007f900d384da9b00f3aab8635ff6e3b7488@51
  .254.147.197:30303

$ geth --unlock 0x3E1bA0338ef893D2BedEc5C328BddE6DdbFaB0Fe --mine --
  datadir ./sealer --networkid 1527 --ethstats='sealer2:
  contrasena@51.254.147.196:80' --bootnodes enode://99
  a919347c5df544565cbc...007f900d384da9b00f3aab8635ff6e3b7488@51
  .254.147.197:30303
```

Repetiremos el proceso para crear otro signer, cambiando el directorio y la cuenta utilizada.

Con esto ya tenemos nuestra red blockchain en funcionamiento y se ha empezado a generar bloques. En la Figura 32 mostramos parte de la salida del nodo signer, centrándonos en la creación de bloques. Ethstats también está funcionando (véase la Figura 33).

## Creación de una red blockchain basada en la plataforma Ethereum 74

```
INFO [05-04|17:55:34.004] ↗ mined potential block
INFO [05-04|17:55:39.008] Imported new chain segment
INFO [05-04|17:55:39.008] 🔒 block reached canonical chain
INFO [05-04|17:55:39.008] Commit new mining work
INFO [05-04|17:55:40.342] Looking for peers
INFO [05-04|17:55:44.000] Successfully sealed new block
INFO [05-04|17:55:44.001] ↗ mined potential block
INFO [05-04|17:55:44.001] Commit new mining work
INFO [05-04|17:55:44.001] Signed recently, must wait for others
INFO [05-04|17:55:44.001] number=18 hash=b851e9...5f053d
INFO [05-04|17:55:44.001] blocks=1 txs=0 mgas=0.000 elap
INFO [05-04|17:55:44.001] number=12 hash=af5b5b...99a424
INFO [05-04|17:55:44.001] number=20 sealhash=17de76...1a0%
INFO [05-04|17:55:44.001] peercount=1 tried=107 static=6
INFO [05-04|17:55:44.001] number=20 sealhash=17de76...1a0%
INFO [05-04|17:55:44.001] number=20 hash=fdac9a...d0ec80
INFO [05-04|17:55:44.001] number=21 sealhash=bc69...e59t
```

Figura 32: Salida del nodo signer, centrandonos en la parte de creación de bloques.

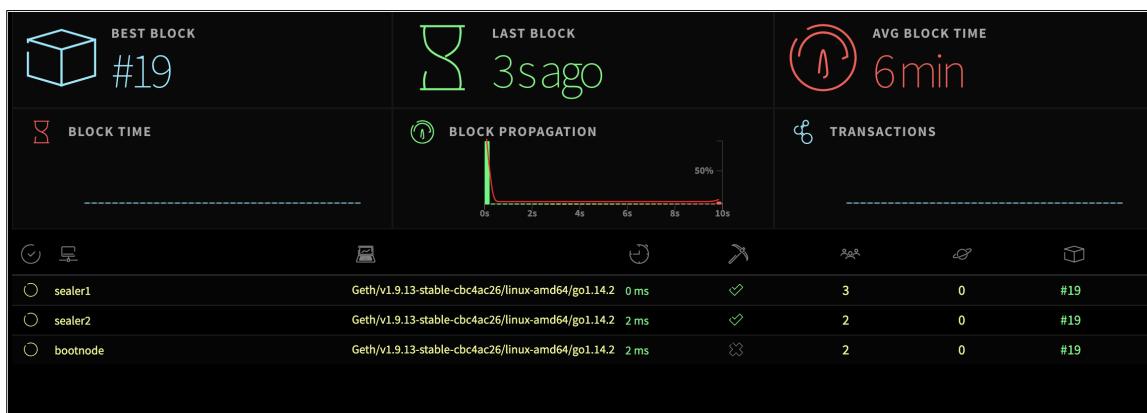


Figura 33: Captura de pantalla recortada de Ethstats con la red blockchain funcionado.

## **Capítulo 5**

# **Desarrollo de una aplicación descentralizada**

En este capítulo detallamos el proceso de desarrollo de la aplicación, empezamos con una descripción general de la trazabilidad de los suministros médicos, después hablaremos sobre la visión general del sistema, explicaremos la metodología de desarrollo utilizada, se selecciona una de las API de Ethereum, se justifica esta decisión y entraremos en detalle en las diferencias existentes entre la programación en Ethereum y la programación orientada a objetos.

Con esta primera parte terminada entramos en el diseño y desarrollo, con la especificación de requisitos, mostramos las historias de usuario, describimos la arquitectura del software, mostramos y explicamos los diagramas de clases, describimos la base de datos del sistema y terminamos con una definición de la API que se comunicará con la aplicación en la red blockchain.

## 5.1. Descripción general de la trazabilidad de suministros médicos

Nos centraremos en la trazabilidad de suministros médicos en el ámbito nacional y utilizando una versión simplificada de la realidad, buscando demostrar la utilidad y operación de la red blockchain creada. La implementación de esta aplicación se realiza como prueba de concepto para validar nuestro sistema de blockchain. La trazabilidad de los suministros médicos desde el origen implicaría conocer la legislación de los diferentes países en cuanto a normativa médica y aduanera, así como lidiar con los diferentes grados de transparencia de información sobre empresas en cada país. Todo esto está fuera del objetivo de este trabajo. Sin embargo, cabe resaltar que con la ayuda de expertos en importación de suministros médicos de los principales países proveedores, la aplicación se podría extender fácilmente para abordar la trazabilidad desde el origen y cumplir las normativas existentes.

Un suministro médico puede que haya sido comprado por una comunidad autónoma o que haya sido comprado por el gobierno central, para luego repartirlo entre las comunidades autónomas. Dentro de una comunidad autónoma se repartirán entre los distintos distritos sanitarios existentes, obviando a las provincias.

Durante el viaje pasará por varios almacenes y de cada suministro guardaremos los almacenes por los que ha pasado. De forma que cualquier persona pueda comprobar el recorrido que ha seguido.

Cada suministro es único y tiene asignado un identificador. Lo mismo ocurre con los almacenes y las comunidades autónomas. De cada suministro guardamos el número de unidades que incluye y el tipo de suministro

médico que es. Guardamos una lista de tipos de suministros (Guantes de nitrilo, mascarillas FPP2, etc.) y podremos añadir nuevos tipos. Un ejemplo de suministro médico sería una caja de 500 mascarillas FFP3.

De cada almacén, aparte de información básica, se guardarán los suministros y las unidades que tiene de cada tipo de suministro en cada momento. De los distritos sanitarios, que consideramos un tipo de almacén específico, guardamos la población de cada uno y las unidades de cada tipo de suministro que tiene asignado, aunque aún no hayan llegado al distrito, para utilizarlo en el reparto.

De las comunidades autónomas guardamos la población y las unidades de cada tipo de suministro que tiene asignada cada comunidad autónoma.

Con la información guardada, se realiza un reparto de cada suministro, de forma que cuando llega un suministro a España, en el caso de que no tenga una comunidad autónoma preasignada (lo haya comprado esa comunidad autónoma) se reparte entre las comunidades autónomas, realizando un balanceo dependiendo de sus necesidades. En nuestro caso se utiliza un criterio simple como demostración, donde comprobamos la población y las unidades del tipo de suministro de cada comunidad autónoma y se asigna el suministro a la comunidad que tiene una mayor necesidad. Se podría guardar más información y hacer criterios de reparto más complejos, teniendo en cuenta múltiples factores. Después se aplica el mismo criterio entre los distintos distritos sanitarios de la comunidad autónoma elegida (o pasamos a este paso directamente si es un suministro preasignado). De esta forma elegimos el distrito sanitario al que se enviará el suministro.

Cuando un suministro médico llega a su distrito sanitario asignado puede ser utilizado, lo que haría que redujésemos las unidades de este tipo de suministro tanto del almacén en el que está guardado como en la comunidad autónoma en la que se encuentra, simulando con esto la apertura del cargamento.

## 5.2. Visión general del sistema

Queremos con la aplicación desarrollada seguir la trazabilidad de los suministros médicos. Utilizaremos una red blockchain privada, que es donde se desplegará nuestra aplicación. Para comunicarnos con ella haremos uso de servidores, los cuales tendrán una API REST (API Representational State Transfer) para comunicarse con el resto del mundo.

Cada almacén tendrá sensores o bien un ordenador que se utilizará para avisar de que un suministro ha llegado, en ese momento envía una petición HTTP (Hypertext Transfer Protocol) a la API REST del servidor de la

aplicación y este procesa la petición provocando llamadas a métodos de la aplicación existente en la red blockchain, la cual guardará esta información en la red blockchain, de forma que pasaría a ser inalterable y transparente. Cada servidor sería a su vez un nodo de la red blockchain, permitiendo esta comunicación con la aplicación.

Esta aplicación en la red blockchain permitirá gestionar la información relacionada con los almacenes, las comunidades autónomas y los suministros médicos. Cualquiera con acceso al identificador de un suministro podría obtener la información de toda su trazabilidad, por ejemplo, mediante una aplicación móvil leyendo un código QR (Quick Response) en el cargamento la cual mandaría una petición HTTP al servidor.

Al decidir utilizar una API REST tenemos la posibilidad de crear múltiples aplicaciones cliente que se comuniquen con ella como el sensor o aplicación móvil mencionada y hemos decidido centrarnos en una interfaz web, que ilustraría el uso de la API.

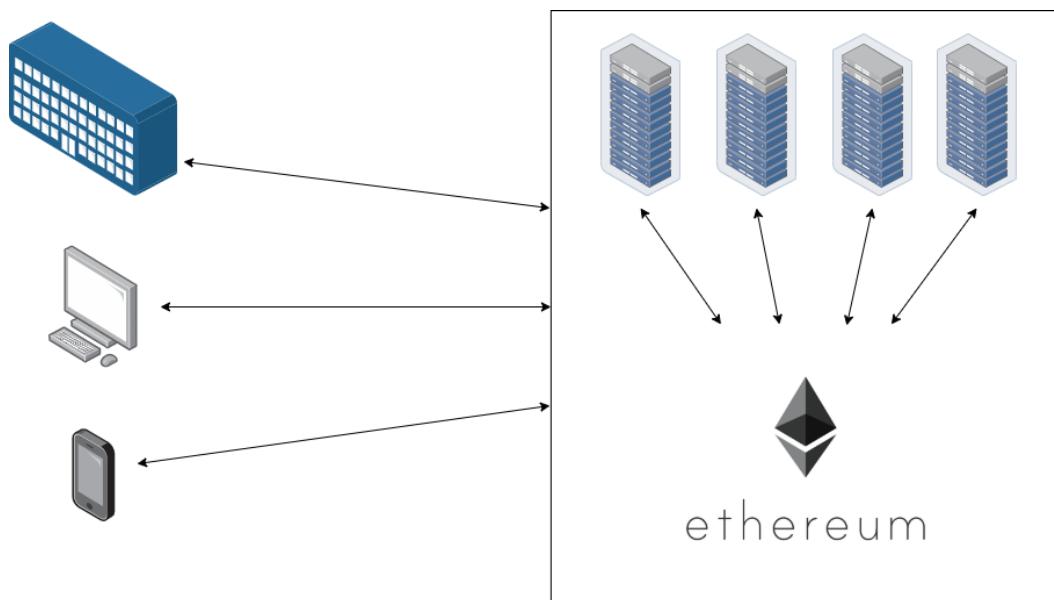


Figura 34: Visión general del sistema (Se ha utilizado el logo de Ethereum y los iconos pertenecen a drawio [49])

### 5.3. Metodología de desarrollo utilizada

Para el desarrollo de la aplicación se ha utilizado la metodología de desarrollo ágil SCRUM, adaptándola a un equipo de desarrollo formado por una sola persona. Nos ofrece flexibilidad y adaptabilidad a los cambios que con-

frecuencia ocurren en el desarrollo de software, permitiendo además un enfoque incremental, empezando con un prototipo de la aplicación y luego ir complicándola y añadiendo más funcionalidad.

Más específicamente, de todas las prácticas y herramientas que ofrece SCRUM [50], hemos utilizado el Product Backlog, una lista que contiene todo lo que sabemos que va a necesitar el producto, que estará en constante cambio y la división del tiempo de desarrollo en Sprint, que tendrán un Sprint Backlog, que contiene las funcionalidades del Product Backlog que vamos a implementar en este incremento del producto (en este Sprint). El Sprint Backlog puede cambiar durante el Sprint. SCRUM incluye múltiples roles para diferentes personas como Product Owner o Scrum Master y distintas reuniones como el Daily Scrum, pero al ser un desarrollo realizado por una sola persona hemos descartado estas prácticas.

De las metodologías de desarrollo ágiles en general se han utilizado las historias de usuario, que serán los componentes que formen el Product Backlog y Sprint Backlog. Las historias de usuario pueden contener una estimación y una prioridad, en nuestro caso hemos decidido incluirlos, para ayudarnos a decidir en qué historias de usuario centrarnos primero en los Sprint. Distinguimos tres niveles de prioridad, siendo 1 el más prioritario, 2 prioridad media y 3 el menos prioritario. Para la estimación utilizamos lo que se conoce como puntos de historia, una estimación relativa a otras estimaciones anteriores (realizadas ambas estimaciones por el mismo equipo), teniendo en cuenta que una historia de usuario de 4 puntos de historia no es el doble de grande que una historia de usuario de 2 puntos de historia.

La división del tiempo de desarrollo en Sprint y las historias de usuario que tenemos se encuentran detalladas en el apartado “5.7. Historias de usuario (Product backlog)”.

## 5.4. API de Ethereum

Ethereum pone tres API a nuestra disposición:

- GO API (propia de Geth) [51]
- JSON RPC API [52]
- Web3.js [53]

La API Web3.js utiliza la API JSON RPC, facilitando el acceso a ella, por lo que descartamos usar directamente la API JSON RPC. Nos queda elegir entre la API de GO y Web3.js, ambas nos permiten interactuar con la

red blockchain y elegir una o la otra nos obligará a utilizar el lenguaje de programación GO o JavaScript (Node.js).

En nuestro caso nos decidimos por utilizar la API Web3.js por la forma en la que funciona Node. Node.js es muy eficiente cuando tenemos múltiples operaciones de E/S, gracias a su ejecución de ellas de forma asíncrona [54]. El enviar una transacción a la red blockchain (llamar a un método de un contrato) es una operación muy similar a una de E/S, por cómo funciona la red blockchain, las operaciones tienen un retraso obligatorio de varios segundos (el tiempo que se tarde en añadir un bloque) por lo que Node.js es ideal y muy eficiente.

## 5.5. Programación en Ethereum, diferencias respecto a programación orientada a objetos

De los lenguajes disponibles para programar en Ethereum [16], utilizamos el lenguaje de programación Solidity, ya que es el más avanzado y desarrollado, por lo que es más estable y contiene menos bugs.

Cuando programamos en una red blockchain Ethereum, a los programas que creamos y ejecutamos se les conoce como contratos [55]. Estos contratos son similares a una clase de programación orientada a objetos en tanto que tienen variables de instancia y métodos que modifican y acceden a estas variables, pero tienen una serie de diferencias que es importante conocer:

- Crear un objeto de una clase, cuando estamos hablando de contratos, sería equivalente a subir de nuevo el contrato a la red blockchain, de forma que cada copia subida tendrá su propio estado y podremos comunicarnos con ellas, utilizando sus direcciones. Un contrato puede llegar a subir otro contrato a la red blockchain. Cuando subimos un contrato a la red blockchain se le asigna una dirección.
- Para llamar a un método de un contrato se realiza una transacción en la red blockchain que se envía a la dirección donde tenemos subido el contrato. Cuando el método que se llama no modifica la red blockchain (no modifica la información guardada) no se realiza una transacción y por tanto comienza su ejecución al instante.
- Un contrato puede llamar a los métodos de otro contrato, si conoce su dirección. Estas llamadas generan lo que se conoce como transacciones internas [56]. Estas transacciones internas se generan a partir de una transacción normal, una vez que la transacción normal se haya añadido en un bloque, se crearán las transacciones internas y por tanto estarán ya añadidas en la red blockchain. Cuando el método que se

llama no modifica la red blockchain tampoco se generan transacciones internas, aunque este método llame a otros contratos.

- Es importante tener en cuenta que, aunque asignemos a una variable una visibilidad private, con esto lo único que conseguimos es evitar que otros contratos puedan modificarla y conseguir información de ella (no se crean métodos get y set), es decir, cualquier persona externa a la red blockchain con acceso a ella puede ver el valor que tiene.

Como se puede apreciar, se podría realizar un diseño idéntico al de la programación orientada a objetos, por ejemplo, teniendo una clase (contrato) Almacén que creará objetos Suministro (subirá el contrato Suministro varias veces) y tendrá un array donde los guarda (las direcciones de los contratos que sube a la red blockchain). Aunque realizar este diseño de forma directa cuando usamos contratos presenta ciertos problemas:

- El consumo del gas. Cada operación que se haga durante la ejecución de un método consumirá una cantidad determinada de gas. De entre las distintas operaciones existentes la de creación de contratos es la más cara que hay [57] y si hiciésemos un diseño orientado a objetos puro acabaríamos haciendo muchas de estas operaciones. En una red blockchain privada no nos afecta, pero si intentásemos pasar nuestra aplicación a una red blockchain pública el coste sería demasiado grande.
- El código que subimos a la red blockchain es immutable, no podemos modificarlo. Para arreglar entonces bugs o añadir funcionalidades tenemos que volver a subir el contrato a la red blockchain y actualizar la dirección de este contrato que tienen el resto de contratos que interactúan con él. También debemos tener en cuenta como transferir los datos guardados en las variables del contrato desactualizado al nuevo, podría hacerse mediante métodos get o se podría utilizar el patrón Eternal Storage [58]. Aplicando este patrón y volviendo al ejemplo anterior, ya no subiríamos solo un contrato Suministro, sino que implicaría subir otro contrato adicional SuministroStorage, duplicando la cantidad de contratos que tendríamos que subir y haciéndolo aún más caro.
- Se sube el contrato entero. Subir un contrato implica subir el código entero a la red blockchain, por lo que en vez de añadir unos valores a un array pasaríamos a subir todo el código de nuevo y entonces añadir esos valores, por lo que la red blockchain aumentará de tamaño más rápido y acabará ocupando más, especialmente si subimos una cantidad muy grande de contratos, que es lo que ocurriría en nuestro caso.

- Los contratos en sí se pueden ver como tablas de una base de datos. Las distintas variables de instancia que tiene un contrato son guardadas en la red blockchain y por tanto se podrían considerar como las columnas de la tabla. Siguiendo esta lógica la subida de un contrato por cada Suministro nuevo sería similar a crear una nueva tabla en vez de añadir una fila a la ya existente.

Por todos estos motivos, hemos decidido seguir un diseño algo más alejado de la programación orientada a objetos. Lo que hacemos es utilizar varios contratos diferentes, buscando obtener los beneficios del diseño orientado a objetos, pero estos contratos no los creamos varias veces, sino que los consideramos Singleton, encajando a la perfección con la idea de que la aplicación en la red blockchain es nuestra base de datos, reduciendo el tamaño que ocuparía la red blockchain y permitiéndonos al haber reducido el consumo de gas, llevar nuestros contratos a otras redes blockchain en un futuro.

## 5.6. Especificación de requisitos del sistema

En este apartado indicaremos cuales son los requisitos funcionales, no funcionales y de información del sistema a desarrollar:

### 5.6.1. Requisitos de información (RI)

- RI\_1: El sistema debe guardar información sobre las comunidades autónomas.

**Contenido:** Nombre de la comunidad autónoma, información sobre ella, población y unidades por tipo de suministro.

- RI\_2: El sistema debe guardar información sobre los almacenes.

**Contenido:** Identificador asociado al almacén, comunidad a la que pertenece, información sobre el almacén, los suministros almacenados y las unidades por tipo de suministro.

- RI\_3: El sistema debe guardar más información sobre los almacenes que son distritos sanitarios.

**Contenido:** Población del distrito sanitario y las unidades por tipo de suministro que tiene asignadas.

- RI\_4: El sistema debe guardar información sobre los suministros.

**Contenido:** Identificador asociado al suministro, tipo de suministro que es, unidades de este tipo que incluye, el distrito sanitario asignado y si ha sido utilizado.

- RI\_5: El sistema debe guardar información sobre los almacenes por los que pasa cada suministro.

**Contenido:** Por cada viaje se guarda el almacén del que salió y la fecha en la que salió, el almacén al que llega y la fecha en la que llega.

- RI\_6: El sistema debe guardar información sobre los tipos de suministros existentes.

**Contenido:** Los tipos de suministros existentes.

- RI\_7: El sistema debe guardar información sobre las cuentas almacén y administrador, que tendrán acceso limitado a la funcionalidad de la aplicación.

**Contenido:** Usuario (dirección de la cuenta Ethereum).

- RI\_8: El sistema debe guardar información sobre la cuenta autorizada, la única que podrá llamar a los métodos de los contratos creados.

**Contenido:** Usuario (dirección de la cuenta Ethereum).

### 5.6.2. Requisitos funcionales (RF)

- RF\_1: El sistema puede añadir nuevas comunidades autónomas.
- RF\_2: El sistema puede devolver toda la información guardada sobre una comunidad autónoma específica.
- RF\_3: El sistema puede devolver una lista de todas las comunidades autónomas guardadas.
- RF\_4: El sistema puede informar si la comunidad autónoma indicada ha sido ya guardada en el sistema.
- RF\_5: El sistema permite la modificación de la información guardada de las comunidades autónomas.
- RF\_6: El sistema puede añadir nuevos almacenes, los cuales están asociados a una comunidad autónoma ya existente.
- RF\_7: El sistema puede devolver toda la información guardada sobre un almacén específico.
- RF\_8: El sistema puede devolver una lista de todos los almacenes guardados.

- RF\_9: El sistema puede informar si el almacén indicado ha sido ya guardado en el sistema.
- RF\_10: El sistema puede informar si el almacén indicado es un distrito sanitario.
- RF\_11: El sistema permite la modificación de la información guardada de los almacenes.
- RF\_12: El sistema puede añadir nuevos tipos de suministro.
- RF\_13: El sistema puede informar de los tipos de suministro existentes.
- RF\_14: El sistema puede informar de si el tipo de suministro indicado ha sido ya incluido en el sistema.
- RF\_15: El sistema puede añadir nuevos suministros.
- RF\_16: El sistema puede repartir estos suministros entre comunidades autónomas.
- RF\_17: El sistema puede repartir estos suministros entre los distritos sanitarios de una comunidad autónoma.
- RF\_18: El sistema puede añadir nuevos suministros, que están ya preasignados a una comunidad autónoma específica.
- RF\_19: El sistema permite la transferencia de un suministro de un almacén a otro.
- RF\_20: El sistema permite la utilización de un suministro.
- RF\_21: El sistema puede devolver toda la información guardada sobre un suministro específico.
- RF\_22: El sistema puede devolver una lista de todos los suministros existentes, utilizados o no.
- RF\_23: El sistema puede informar si existe ya un suministro con el identificador indicado.
- RF\_24: El sistema puede informar del almacén actual en el que se encuentra un suministro específico.
- RF\_25: El sistema puede informar del número total de unidades que guarda un almacén específico.
- RF\_26: El sistema puede informar de las unidades de cada tipo de suministro que tiene un almacén específico.

- RF\_27: El sistema puede informar de las unidades de un tipo de suministro concreto que tiene un almacén específico.
- RF\_28: El sistema puede informar de la comunidad autónoma a la que está asignada un suministro.
- RF\_29: El sistema puede informar de la trazabilidad completa de un suministro, indicando los almacenes por los que ha pasado y en qué fecha.
- RF\_30: El sistema permite el uso de cuentas Ethereum como cuentas almacén, asociadas a almacenes.
- RF\_31: El sistema permite el uso de cuentas Ethereum como cuentas administrador.
- RF\_32: El sistema dispone de una interfaz web simple para comunicarse con la API y demostrar su uso.

#### 5.6.3. Requisitos no funcionales (RNF)

- RNF\_1: Toda la información relacionada con los almacenes, suministros, la trazabilidad de estos y las comunidades autónomas deben guardarse en la red blockchain, utilizándola como base de datos, permitiendo así aprovechar sus características.
- RNF\_2: Se debe utilizar un sistema de identificación como mecanismo de seguridad para que solo las personas autorizadas puedan utilizar la API de acceso a la aplicación.
- RNF\_3: El procesamiento de las peticiones y escritura en la red blockchain debe de ser lo suficientemente eficiente como para aguantar la carga de la trazabilidad de todos los suministros.
- RNF\_4: Deben existir múltiples servidores, cada uno con un nodo de la red blockchain, para reducir al máximo posible el tiempo en el cual no está disponible la aplicación y que así afecte lo mínimo posible al reparto de suministros médicos.
- RNF\_5: El sistema debe de ser tolerante a fallos, para afectar lo mínimo posible al reparto de suministros médicos.
- RNF\_6: La información que devuelve el sistema debe seguir un formato estándar de representación e intercambio de datos (por ejemplo, JSON), para así facilitar el procesamiento de esta misma.

## 5.7. Historias de usuario (Product backlog)

En vez de mostrar el Product Backlog primero y luego volver a repetir las historias de usuario cuando hablemos de los Sprint Backlog, mostramos directamente los Sprint Backlog, consiguiendo así una mayor legibilidad.

Decidimos realizar 2 Sprint, con una duración de 15 días y 16 días respectivamente. El primer Sprint fue desde el 1 de julio hasta el 15 de julio y el segundo Sprint desde el 16 de julio hasta el 31 de julio.

### 5.7.1. Primer Sprint

Identificador: HU-1	Estimación: 1	Prioridad: 1
Como usuario quiero poder añadir nuevas comunidades autónomas para que la aplicación cubra toda España.		
<b>Pruebas de aceptación</b>		
<ol style="list-style-type: none"><li>1. La interfaz web manda la petición a la API de forma correcta.</li><li>2. La API responde correctamente a la petición.</li><li>3. La comunidad autónoma se añade correctamente.</li></ol>		
<b>Descomposición en tareas</b>		
<ol style="list-style-type: none"><li>1. Diseño y modificación de los diagramas de secuencia y clase.</li><li>2. Implementación de esta opción en la interfaz web.</li><li>3. Implementación de esta opción en la API.</li><li>4. Implementación de la funcionalidad en la red blockchain.</li></ol>		

Identificador:	HU-2	Estimación:	1	Prioridad:	1
Como usuario quiero poder añadir un almacén nuevo en la aplicación para poder guardar suministros médicos en él.					
<b>Pruebas de aceptación</b>					
<ol style="list-style-type: none"> <li>1. La interfaz web manda la petición a la API de forma correcta.</li> <li>2. La API responde correctamente a la petición.</li> <li>3. El almacén se añade correctamente.</li> <li>4. En caso de que la comunidad indicada, donde el almacén está localizado, no exista se produce un error y se avisa de ello.</li> </ol>					
<b>Descomposición en tareas</b>					
<ol style="list-style-type: none"> <li>1. Diseño y modificación de los diagramas de secuencia y clase.</li> <li>2. Implementación de esta opción en la interfaz web.</li> <li>3. Implementación de esta opción en la API.</li> <li>4. Implementación de la funcionalidad en la red blockchain.</li> </ol>					

Identificador:	HU-3	Estimación:	1	Prioridad:	1
Como usuario quiero poder añadir nuevos suministros médicos a la aplicación para poder seguir su trazabilidad.					
<b>Pruebas de aceptación</b>					
<ol style="list-style-type: none"> <li>1. La interfaz web manda la petición a la API de forma correcta.</li> <li>2. La API responde correctamente a la petición.</li> <li>3. El suministro se añade correctamente.</li> <li>4. En caso de que el almacén indicado donde aparece por primera vez el suministro no exista se produce un error y se avisa de ello.</li> </ol>					
<b>Descomposición en tareas</b>					
<ol style="list-style-type: none"> <li>1. Diseño y modificación de los diagramas de secuencia y clase.</li> <li>2. Implementación de esta opción en la interfaz web.</li> <li>3. Implementación de esta opción en la API.</li> <li>4. Implementación de la funcionalidad en la red blockchain.</li> </ol>					

Identificador:	HU-4	Estimación:	3	Prioridad:	1
Como usuario quiero que el sistema gestione la trazabilidad de los suministros para que pueda saber si estos suministros médicos son fiables y pueden ser utilizados.					
<b>Pruebas de aceptación</b>					
<ol style="list-style-type: none"> <li>1. Se tiene en cuenta cuando un suministro sale de un almacén y se guarda la información.</li> <li>2. Se tiene en cuenta cuando un suministro entra en un almacén y se guarda la información.</li> </ol>					
<b>Descomposición en tareas</b>					
<ol style="list-style-type: none"> <li>1. Diseño y modificación de los diagramas de secuencia y clase.</li> <li>2. Implementación de esta opción en la interfaz web.</li> <li>3. Implementación de esta opción en la API.</li> <li>4. Implementación de la funcionalidad en la red blockchain.</li> </ol>					

Identificador:	HU-5	Estimación:	3	Prioridad:	2
Como usuario quiero que la aplicación se encargue de repartir y asignar los suministros, teniendo en cuenta además si ha sido uno que yo he comprado a parte, para poder centrarme en otros asuntos importantes y no tener que coordinar todo.					
<b>Pruebas de aceptación</b>					
<ol style="list-style-type: none"> <li>1. Se tiene en cuenta si un suministro tiene una comunidad autónoma preasignada.</li> <li>2. Se reparte correctamente los suministros.</li> </ol>					
<b>Descomposición en tareas</b>					
<ol style="list-style-type: none"> <li>1. Diseño y modificación de los diagramas de secuencia y clase.</li> <li>2. Implementación de la funcionalidad necesaria para realizar el reparto de suministros.</li> </ol>					

### 5.7.2. Segundo Sprint

Identificador:	HU-6	Estimación:	8	Prioridad:	1
Como usuario quiero poder obtener todo tipo de información, desde la trazabilidad de un suministro hasta información específica de un almacén, como las unidades por tipo de suministro e información similar para tener una idea clara del estado del reparto de los suministros médicos.					
<b>Pruebas de aceptación</b>					
<ol style="list-style-type: none"><li>1. La interfaz web manda la petición a la API de forma correcta.</li><li>2. La API responde correctamente a la petición.</li><li>3. El formato JSON utilizado es correcto.</li><li>4. Se obtiene la información correcta para cada petición.</li><li>5. Si se pide información sobre un objeto (ya sea un suministro, almacén o comunidad autónoma) y no existe, ocurre un error y avisa de ello.</li></ol>					
<b>Descomposición en tareas</b>					
<ol style="list-style-type: none"><li>1. Decidir las opciones en la API que habrá disponibles.</li><li>2. Elección del formato JSON adecuado para cada petición de información.</li><li>3. Diseño y modificación de los diagramas de secuencia y clase.</li><li>4. Implementación de estas opciones en la interfaz web.</li><li>5. Implementación de estas opciones en la API.</li><li>6. Implementación de la obtención de datos en la red blockchain.</li></ol>					

Identificador: HU-7	Estimación: 3	Prioridad: 1
Como usuario quiero que se tenga en cuenta cuando un suministro ha llegado a su destino y ha empezado a ser utilizado para tener una idea clara de la situación actual.		
<b>Pruebas de aceptación</b>		
<ol style="list-style-type: none"><li>1. Un suministro no se puede utilizar si no ha llegado a su distrito sanitario asignado.</li><li>2. Una vez utilizado se puede seguir obteniendo información de él, no se borra ninguna información.</li><li>3. No podemos utilizar un suministro varias veces.</li></ol>		
<b>Descomposición en tareas</b>		
<ol style="list-style-type: none"><li>1. Diseño y modificación de los diagramas de secuencia y clase.</li><li>2. Implementación de esta opción en la interfaz web.</li><li>3. Implementación de esta opción en la API.</li><li>4. Implementación de la funcionalidad en la red blockchain.</li></ol>		

Identificador: HU-8	Estimación: 5	Prioridad: 3
Como usuario quiero que el sistema sea seguro para evitar la manipulación y perdida de suministros médicos.		
<b>Pruebas de aceptación</b>		
<ol style="list-style-type: none"><li>1. Necesitamos identificarnos para acceder a cierta funcionalidad de la API.</li><li>2. Dependiendo del tipo de cuenta podemos acceder a cierta funcionalidad y a otra no.</li><li>3. Se utilizan cuentas Ethereum y se permite el inicio de sesión en ellas mediante los mecanismos ofrecidos por Ethereum.</li><li>4. Los contratos solo aceptan peticiones que vengan de una cuenta especial o de los otros contratos.</li></ol>		
<b>Descomposición en tareas</b>		
<ol style="list-style-type: none"><li>1. Diseño y modificación de los diagramas de secuencia y clase.</li><li>2. Implementación de la seguridad en la interfaz web.</li><li>3. Implementación de la seguridad en la API.</li><li>4. Implementación de la seguridad en la red blockchain.</li></ol>		

Identificador: HU-9	Estimación: 3	Prioridad: 2
Como usuario quiero poder modificar la información de los almacenes y comunidades autónomas que se han añadido en el sistema para poder mantener la información actualizada, sin que esto afecte a la trazabilidad.		
<b>Pruebas de aceptación</b>		
<ol style="list-style-type: none"><li>1. La interfaz web manda la petición a la API de forma correcta.</li><li>2. La API responde correctamente a la petición.</li><li>3. La información se modifica correctamente.</li><li>4. En caso de que el almacén o comunidad autónoma indicada no exista se produce un error y se avisa de ello.</li></ol>		
<b>Descomposición en tareas</b>		
<ol style="list-style-type: none"><li>1. Diseño y modificación de los diagramas de secuencia y clase.</li><li>2. Implementación de estas opciones en la interfaz web.</li><li>3. Implementación de estas opciones en la API.</li><li>4. Implementación de la funcionalidad en la red blockchain.</li></ol>		

## 5.8. Arquitectura del software

En este apartado describiremos la arquitectura del software. Estamos ante una arquitectura cliente-servidor por capas, donde tenemos clientes que se comunicarán con servidores y este proceso de comunicación se encuentra dividido en varias capas, de forma que cada capa actúa como cliente de la capa inferior, las capas no tienen conocimiento de las capas superiores. Cada capa solo puede interactuar con la capa que tiene justamente en un nivel inferior y responder a las peticiones recibidas de una capa superior.

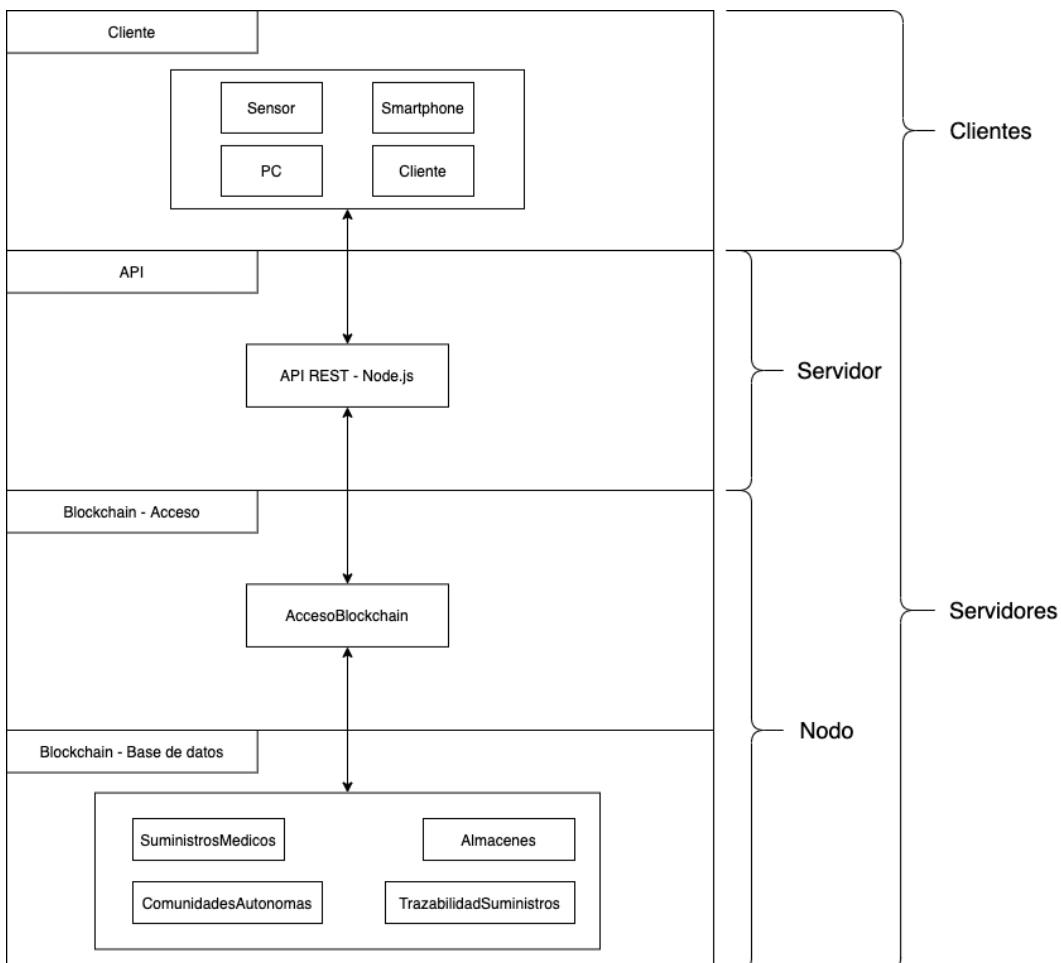


Figura 35: Arquitectura del software

La primera capa de todas es la de los clientes, donde tenemos los distintos dispositivos que harán uso de nuestro sistema, ya sea un ordenador que se encuentra en el almacén y conforme llegan suministros médicos se vaya actualizando manualmente o quizás, en un futuro, completamente automatizado y mediante el uso de sensores. Quizás algún médico comprobando con su smartphone la trazabilidad del suministro de EPI (Equipo de Protección Individual) que van a utilizar. Estos clientes se comunicarán con los servidores y más específicamente con la capa justamente inferior, la de la API REST.

Al ser una API REST para comunicarnos con ella realizamos peticiones HTTP, lo cual nos ofrece una gran flexibilidad y permite con facilidad ser utilizada por muchos tipos de dispositivos diferentes. Devuelve la información en formato JSON. Esta API se comunicará mediante Web3.js con la capa in-

ferior, que es el nodo de la red blockchain. Más específicamente se comunica con un contrato concreto, AccesoBlockchain.

El contrato AccesoBlockchain se encarga de redirigir la petición recibida de la API al contrato de la red blockchain pertinente y de convertir la información recibida en el formato JSON que entenderá la API. Nos ofrece un punto de entrada al resto de contratos de la red blockchain, permitiendo a la API abstraerse de la implementación de la aplicación en la red blockchain.

Por último, tenemos la capa base de datos, formada por el resto de contratos, cuyos métodos son llamados por AccesoBlockchain.

Estas dos últimas capas son las que se encuentran dentro de la red blockchain y en cada servidor tendremos un nodo instalado.

## 5.9. Diagramas de clases

En este apartado mostramos los diagramas de clases, donde aparecen los contratos existentes y como se relacionan entre ellos.

Lo hemos separado en tres diagramas, en el primero mostramos las relaciones de herencia de los contratos (véase la Figura 36).

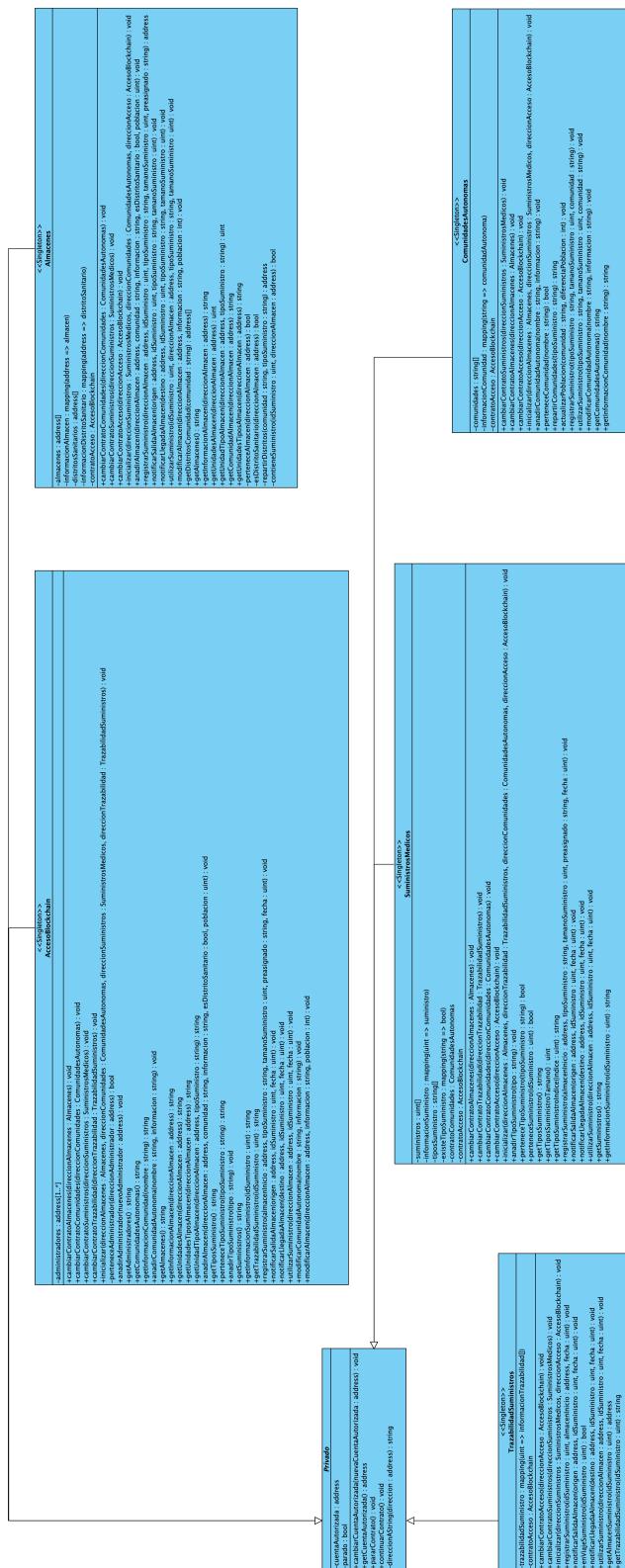


Figura 36: Diagrama de clases con las relaciones de herencia

Todos los contratos heredan del contrato Privado, en él tenemos el código relacionado con la limitación del acceso a la red blockchain de forma que solo se acepten llamadas a métodos realizadas desde la cuenta indicada (cuenta autorizada) y donde se incluye el código para poder bloquear los contratos y evitar que se utilicen, utilizando el patrón de diseño Emergency Stop [59] (Para el caso en que haya algún bug crítico y deseamos impedir que se siga utilizando).

A parte de este código común, todos los contratos tienen incluido código para aceptar llamadas a métodos que vengan de los otros contratos, esta limitación de acceso no se ha podido incluir en Privado, ya que habría que tener variables del tipo de los otros contratos y al heredar de Privado no nos permite el compilador hacer import del resto de contratos en el contrato Privado.

En el segundo diagrama de clases (véase la Figura 37) mostramos las relaciones entre los distintos contratos existentes pero sin incluir los struct y en el tercer diagrama de clases (véase la Figura 38) se muestra las relaciones entre los distintos contratos sin incluir el contrato AccesoBlockchain pero incluyendo los struct.

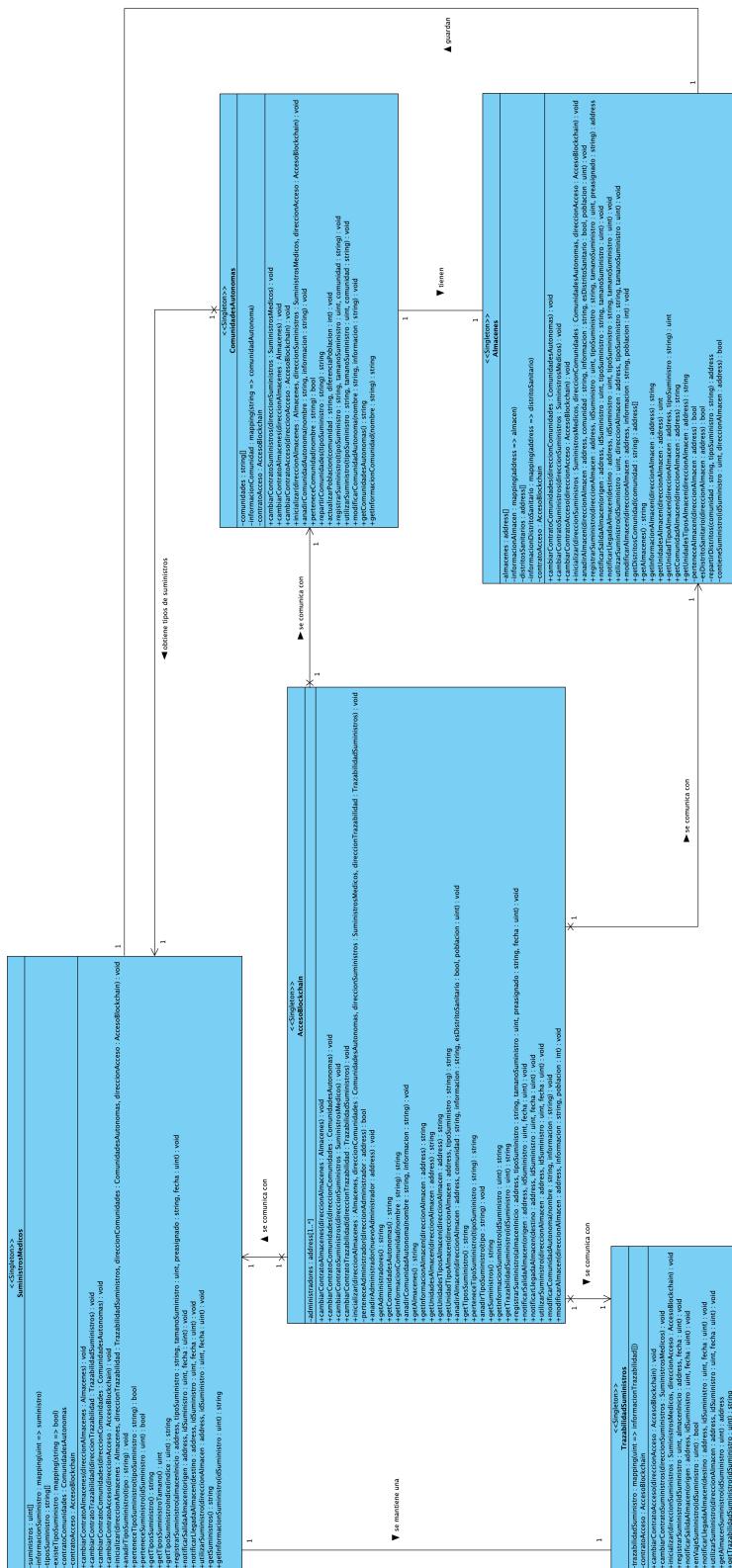


Figura 37: Diagrama de clases con las relaciones entre contratos sin incluir los struct

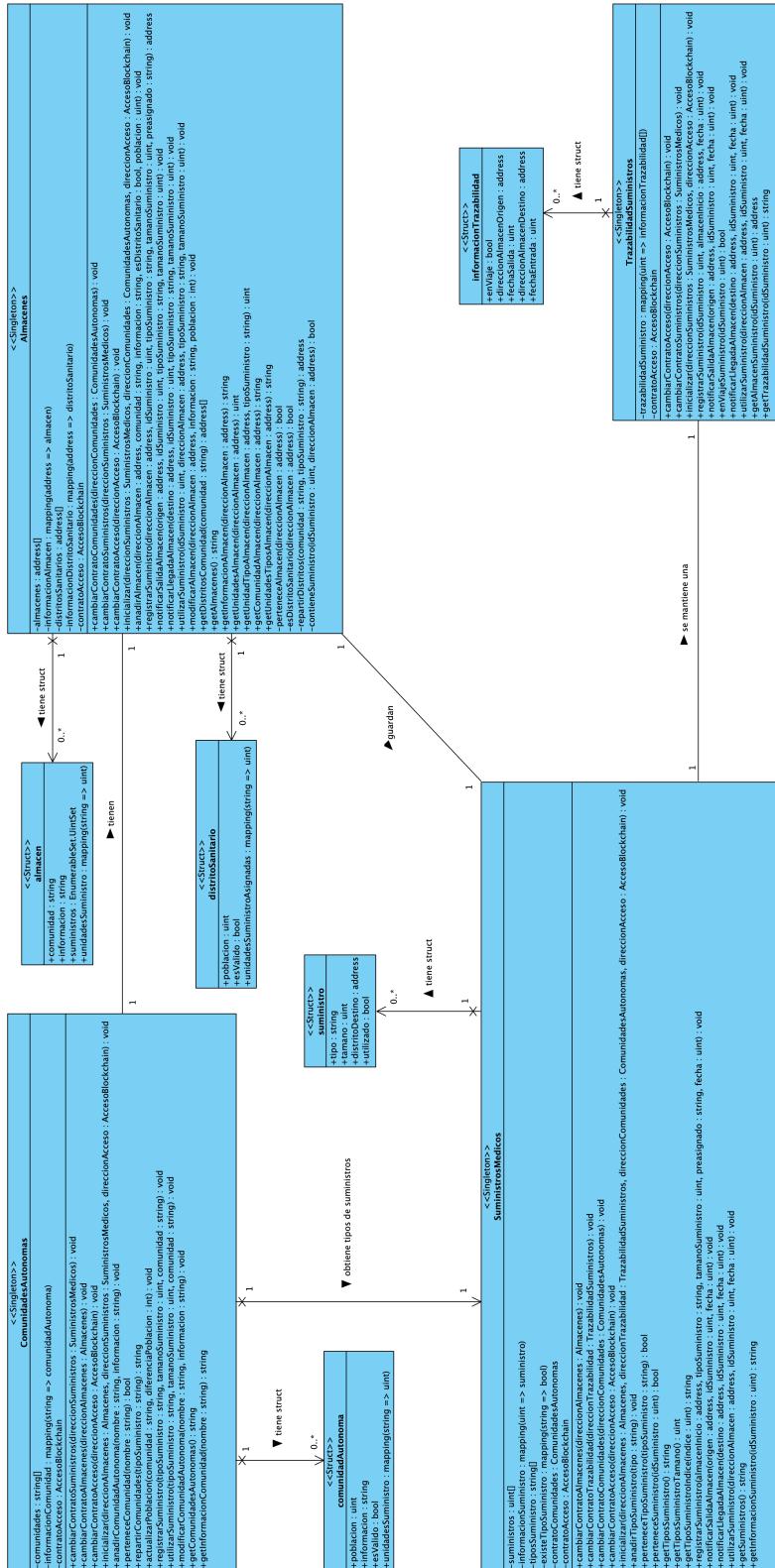


Figura 38: Diagrama de clases con las relaciones entre contratos con struct pero sin incluir AccesoBlockchain

Debemos tener en cuenta un matiz que ocurre al utilizar contratos y es que con tener la dirección de un contrato se podría llamar a sus métodos, por lo que alguien desde fuera podría provocar la ejecución de uno de los métodos de un contrato. Para evitar esta situación se añade el control de acceso mencionado antes, se comprueba la dirección desde la que se ha llamado al contrato y si no pertenece a nuestros contratos no se ejecuta el método. Pero esto implica que tendríamos que guardar las direcciones de contratos cuyos métodos no llamamos, pero de los que si recibimos llamadas, como AccesoBlockchain. A la hora de crear el diagrama de clases decidimos no incluir este tipo de relaciones en él, pues acabaría confundiendo luego cuando se vea que, por ejemplo, nadie llama a los métodos de AccesoBlockchain pese a existir una asociación bidireccional entre ellos en el diagrama.

El contrato AccesoBlockchain contiene las direcciones del resto de contratos, para poder reenviar la petición de la API al contrato adecuado y se encarga de pasar a formato JSON la información que recibe del resto de contratos.

El contrato SuministrosMedicos guarda toda la información relacionada con los suministros médicos existentes e incluye toda la funcionalidad relacionada con los suministros, para poder obtener información de estos o añadir nuevos suministros al sistema. Se relaciona con los contratos Almacenes y TrazabilidadSuministros.

El contrato TrazabilidadSuministros guarda toda la información relacionada con la trazabilidad de los suministros como los almacenes por los que pasan y fechas de salida y llegada a cada almacén. Incluye toda la funcionalidad relacionada con la trazabilidad y se relaciona con el contrato SuministrosMedicos.

El contrato Almacenes guarda toda la información relacionada con los almacenes, incluyendo que suministros tiene cada almacén, cuantas unidades por tipo de suministro y la comunidad a la que pertenece el almacén. Consideramos a los distritos sanitarios como almacenes y por tanto también se guarda su información en este contrato. Incluye toda la funcionalidad relacionada con los almacenes y se relaciona con SuministrosMedicos y con ComunidadesAutonomas.

El contrato ComunidadesAutonomas guarda toda la información sobre las comunidades autónomas existentes. También guarda la cantidad de unidades por tipo de suministro asignadas a la comunidad autónoma específica. Incluye toda la funcionalidad relacionada con las comunidades autónomas. Se relaciona con SuministrosMedicos y con Almacenes.

## 5.10. Base de datos del sistema

Como ya se ha mencionado, la red blockchain es una base de datos en sí, por tanto, toda la información que hemos querido guardar está representada en las variables de instancia de los contratos creados.

Antes de empezar explicaremos un tipo de dato muy útil de Solidity, el mapping.

**mapping(address => int):** Esto sería una asociación entre valores claves de tipo address (dirección) y valores enteros, de forma que dado un valor address te devolverá un valor int y en el caso de que sea la primera vez que se utiliza ese valor clave para acceder al valor guardado, está inicializado al valor por defecto del tipo de dato en cuestión.

En Solidity un array se declara de la siguiente forma:

**address[] arrayDirecciones**

El tipo de datos address hace referencia a una dirección de Ethereum, es decir, a una cuenta Ethereum. Las cuentas Ethereum utilizan como nombre de usuario (identificador) una dirección, que es un valor en hexadecimal.

De las comunidades autónomas tenemos las siguientes variables (en el contrato ComunidadesAutonomas):

- **struct comunidadAutonoma:** Formado por las siguientes variables:
  - **uint poblacion:** La población de la comunidad autónoma. Se modifica automáticamente al modificar las poblaciones de los distritos sanitarios que se encuentran en la comunidad autónoma.
  - **string informacion:** Información sobre la comunidad autónoma.
  - **bool esValido:** Variable utilizada para comprobar si una instancia de este struct ha sido inicializada.
  - **mapping(string => uint) unidadesSuministro:** Mapping que contiene las unidades que hay por tipo de suministro asignadas a esta comunidad autónoma.
- **mapping(string => comunidadAutonoma) informacionComunidad:** Mapping que contiene la información asociada a una comunidad autónoma identificado por un string (Su nombre).
- **string[] comunidades:** Array que contiene todos los identificadores de las comunidades autónomas añadidas en el sistema.

De los almacenes tenemos las siguientes variables (en el contrato Almacenes):

- **struct almacen:** Formado por las siguientes variables:
    - **string comunidad:** Indica la comunidad autónoma a la que pertenece el almacén.
    - **string informacion:** Información sobre el almacén.
    - **EnumerableSet.UintSet suministros:** Un set con los identificadores de los suministros que están guardados en el almacén.
    - **mapping(string => uint) unidadesSuministro:** Indica las unidades que hay por cada tipo de suministro en el almacén.
  - **mapping(address => almacen) informacionAlmacen:** Mapping que contiene la información asociada a un almacén identificado por una dirección.
  - **address[] almacenes:** Array que contiene todos los identificadores de los almacenes añadidos en el sistema.
  - **struct distritoSanitario:** Formado por las siguientes variables:
    - **uint poblacion:** Población del distrito sanitario.
    - **bool esValido:** Variable utilizada para comprobar si una instancia de este struct ha sido inicializada.
    - **mapping(string => uint) unidadesSuministroAsignadas:** Indica las unidades que hay por cada tipo de suministro asignadas a este distrito sanitario (suministros que tienen como destino a este distrito sanitario) aunque aún no hayan llegado al distrito.
  - **mapping(address => distritoSanitario) informacionDistritoSanitario:** Mapping que contiene la información asociada a un distrito sanitario identificado por una dirección.
  - **address[] distritosSanitarios:** Array que contiene todos los identificadores de los distritos sanitarios añadidos en el sistema.
- Los distritos sanitarios también tendrán un struct almacen inicializado, guardado en informacionAlmacen, pero su identificador se guarda en el array distritosSanitarios mencionado. De esta forma cuando sabemos que es un distrito sanitario podremos acceder a informacion-DistritoSanitario para tener acceso a la información específica de un distrito sanitario.
- De los suministros tenemos las siguientes variables (en el contrato SuministrosMedicos):
- **struct suministro:** Formado por las siguientes variables:

- **string tipo:** Indica el tipo de suministro que es.
  - **uint tamano:** Indica el tamaño del suministro, las unidades que contiene.
  - **address distritoDestino:** El distrito sanitario que se le ha asignado en el reparto.
  - **bool utilizado:** Indica si el suministro ha sido utilizado o no.
- **mapping(uint =>suministro) informacionSuministro:** Mapping que contiene la información asociada a un suministro identificado por un número entero.
  - **uint[] suministros:** Array que contiene todos los identificadores de los suministros añadidos al sistema.
  - **string[] tiposSuministro:** Array que contiene todos los tipos de suministros añadidos en el sistema.
  - **mapping(string =>bool) existeTipoSuministro:** Mapping que utilizamos para comprobar si un tipo de suministro está incluido en el sistema con un orden de eficiencia O(1).

De la trazabilidad de los suministros tenemos las siguientes variables (en el contrato TrazabilidadSuministros):

- **struct informacionTrazabilidad:** Formado por las siguientes variables:
  - **bool enViaje:** Indica si el suministro está en viaje de un almacén a otro.
  - **address direccionAlmacenOrigen:** El identificador del almacén del que ha salido.
  - **uint fechaSalida:** La fecha en la que salió del almacén.
  - **address direccionAlmacenDestino:** El identificador del almacén al que ha llegado.
  - **uint fechaEntrada:** La fecha en la que llegó al almacén destino.
- **mapping(uint =>informacionTrazabilidad[]) trazabilidadSuministro:** Mapping que contiene un array con la trazabilidad de un suministro identificado por un número entero. Cada vez que un suministro salga de un almacén crearemos un nuevo valor en el array con la información de la salida y cuando llegue a un almacén volveremos a acceder a este mismo valor y le daremos sus valores correspondientes a los dos últimos campos.

Por último, tenemos una variable común a todos los contratos, que es la dirección de la cuenta autorizada. La cuenta autorizada es una cuenta que utilizamos para subir los contratos e interactuar con ellos. Si intentamos llamar a estos contratos usando otra cuenta no se ejecutarán y se lanzará una excepción. Los contratos por tanto solo aceptarán mensajes de los otros contratos y la cuenta autorizada. Dependiendo del método se permitirá acceso a la cuenta autorizada o a los contratos.

Tendremos que iniciar sesión en la cuenta autorizada cada vez que iniciemos la API. Para evitar tener la contraseña de la cuenta autorizada guardada en el código de la API podemos desbloquear la cuenta al iniciar el nodo mediante la opción siguiente, indicándole el archivo donde tenemos la contraseña guardada [18]:

```
geth <otrasOpciones> --unlock <direccionCuenta> --password <archivo>
```

Sobre la autenticación de usuarios, distinguimos dos tipos de cuentas, cuentas almacén (asociadas a un almacén, utilizamos el identificador del almacén como nombre de usuario) y cuentas administrador (cuyas direcciones se guardan en el contrato AccesoBlockchain), que pueden acceder a funcionalidad diferente.

Cada almacén tendrá una cuenta almacén y será el único con la contraseña de esa cuenta, de forma que si recibimos una petición de una cuenta almacén específica sabemos que de verdad lo ha enviado ese almacén, pues es el único que conocería la contraseña de su cuenta.

Para el inicio de sesión no utilizamos usuario y contraseña tradicionales, sino que utilizamos cuentas Ethereum.

Si recordamos lo explicado en la sección “3.2. Ethereum” sobre cuentas Ethereum, para utilizar una cuenta necesitamos saber su contraseña, su dirección y tener su archivo keystore.

En la red blockchain se guardarán solo las direcciones y cada servidor tendrá una copia de los archivos keystore de las cuentas. Cuando iniciamos sesión se intentará desbloquear la cuenta con la contraseña dada y se comprobará si la cuenta es almacén o administrador. Gracias a cómo funciona las cuentas Ethereum no es necesario que guardemos las contraseñas en la red blockchain, lo cual sería problemático ya que sería visible para todo el que tenga acceso a la red blockchain.

Para la creación de cuentas se utilizaría Geth y luego se incluiría la dirección de la cuenta creada en los contratos. Por último, habría que copiar el archivo keystore en todos los servidores, para poder iniciar sesión en esta cuenta independientemente del servidor al que nos conectemos.

## 5.11. Definición de la API para la interacción con la aplicación

En este apartado explicaremos la API REST [60] y las interacciones que se producen entre el software cliente, la API y los métodos de los contratos mediante diagramas de secuencia.

En caso de ser peticiones GET donde se devuelve contenido JSON con la información pedida la respuesta tendrá estado 200. Si no devuelve nada estado 204. En las peticiones PUT cuando la modificación se haya realizado con éxito se devuelve una respuesta vacía con estado 204. En las peticiones POST la respuesta estará vacía y tendrá estado 201 si ha tenido éxito. En caso de error se incluye contenido JSON con el error ocurrido y el estado de la respuesta variará dependiendo del error.

En el caso de la autenticación de usuario, si ha habido un problema con las credenciales se devuelve una respuesta con estado 401. En caso de estar intentando acceder a una funcionalidad restringida para la cuenta, la respuesta tendrá estado 403.

Las cuentas administrador pueden acceder a la funcionalidad relacionada con la creación y modificación de comunidades autónomas, almacenes y tipos de suministro médico. Las cuentas almacén pueden acceder a la funcionalidad relacionada con la creación y gestión de los suministros médicos. No es necesario autenticarse para la obtención de información.

Para autenticarnos y poder acceder a estas funcionalidades incluiremos dos variables con formato x-www-form-urlencoded en la petición, solo será necesario incluirlas si queremos acceder a una funcionalidad restringida:

```
usuario=<usuario>&password=<password>
```

Todos los diagramas de secuencia siguen el mismo formato, donde los bloques break muestran la condición para que se lance una excepción y lo que ocurre. Este lanzamiento de excepciones lo representamos con un *throw error*. En el caso de las excepciones es la API la encargada de crear el contenido con formato JSON (ya que los contratos no pueden en estos casos). Cuando en el diagrama aparece que se devuelve JSON, nos referimos a que se devuelve un string con el contenido JSON con el formato indicado en su apartado. La comunicación entre cliente y API son peticiones HTTP, la comunicación entre API y AccesoBlockchain son llamadas a métodos mediante Web3.js y a partir de ahí ya son llamadas a métodos entre contratos (clases).

En el caso de que haya ocurrido un error, el formato JSON de la respuesta es el siguiente:

```
{  
    "error": "<error>"  
}
```

Para la API REST se ha utilizado Express y su documentación oficial [61]. Para la programación con Web3.js se ha utilizado la documentación oficial de Web3.js [62]. Para la creación de la interfaz web se ha utilizado el apartado HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) y JavaScript de la página web w3schools [63]. Para la programación de los contratos con el lenguaje de programación Solidity se ha utilizado la documentación oficial de Solidity [55].

Para compilar los contratos y subirlos a la red blockchain se ha utilizado OpenZeppelin [64]. También lo hemos utilizado para facilitar la interacción con los contratos que permite Web3.js y utilizamos la implementación del tipo de dato set que ofrecen.

Se ha incluido en la entrega de este trabajo, junto al código del sistema, un archivo llamado “estructura-e-instrucciones-ejecucion-codigo.txt” donde se explica los comandos necesarios para poder ejecutar el código entregado. Para poder ejecutar este código es necesario conectarnos a alguna red blockchain, los pasos para crear una red blockchain no lo hemos incluido en el archivo ya que en este mismo trabajo se explican. Más específicamente se explican en el capítulo 4 titulado “Creación de una red blockchain basada en la plataforma Ethereum”.

### 5.11.1. Administrador

- GET /administradores

Formato JSON de la respuesta:

```
{
    "resultado": ["<administrador1>", "<administrador2>", ...]
}
```

Devuelve los nombres de usuario (direcciones de las cuentas Ethereum) de los administradores existentes en el sistema.

Diagrama de secuencia:

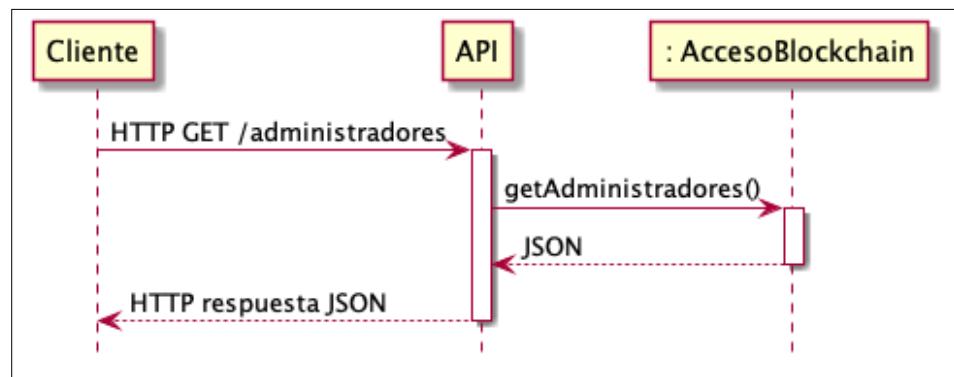


Figura 39: Diagrama de secuencia de la petición GET /administradores

- POST /administradores

Variables con formato x-www-form-urlencoded de la petición:

```
nuevoAdministrador=<direccionAdministrador>
```

Añade una nueva cuenta administrador al sistema.

Requiere iniciar sesión con una cuenta administrador.

Diagrama de secuencia:

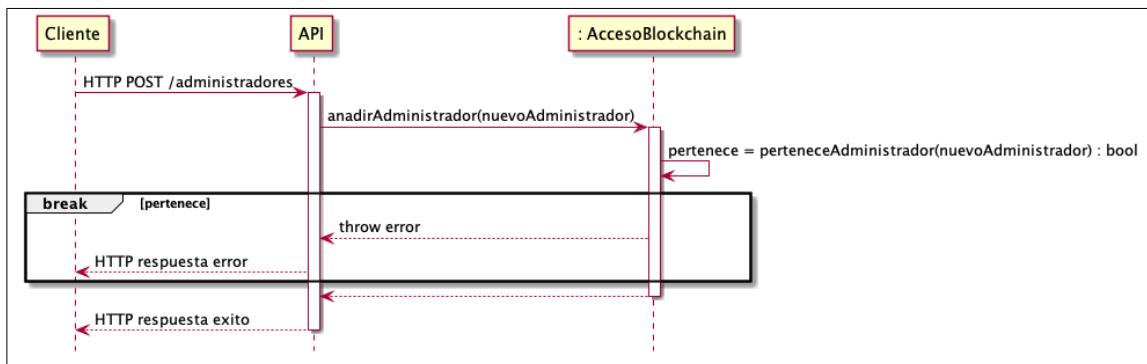


Figura 40: Diagrama de secuencia de la petición POST /administradores

### 5.11.2. Comunidades autónomas

- **GET /comunidades**

Formato JSON de la respuesta:

```
{
    "resultado" : ["<comunidad1>" ,
                   "<comunidad2>" , ... ]
}
```

Devuelve los identificadores de todas las comunidades autónomas existentes en el sistema.

Diagrama de secuencia:

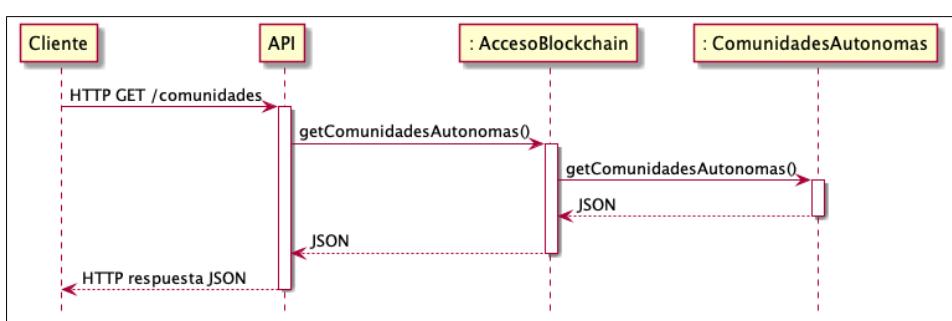


Figura 41: Diagrama de secuencia de la petición GET /comunidades

■ GET /comunidades/<nombre>

Formato JSON de la respuesta:

```
{
    "nombre": "<nombre>",
    "informacion": "<informacion>",
    "poblacion": <poblacion>,
    "distritos": [<direccion1>, <direccion2>, ...],
    "tiposSuministro": {
        "<tipo1>": <unidades1>,
        "<tipo2>": <unidades2>,
        ...
    }
}
```

Devuelve información sobre la comunidad autónoma indicada.

Diagrama de secuencia:

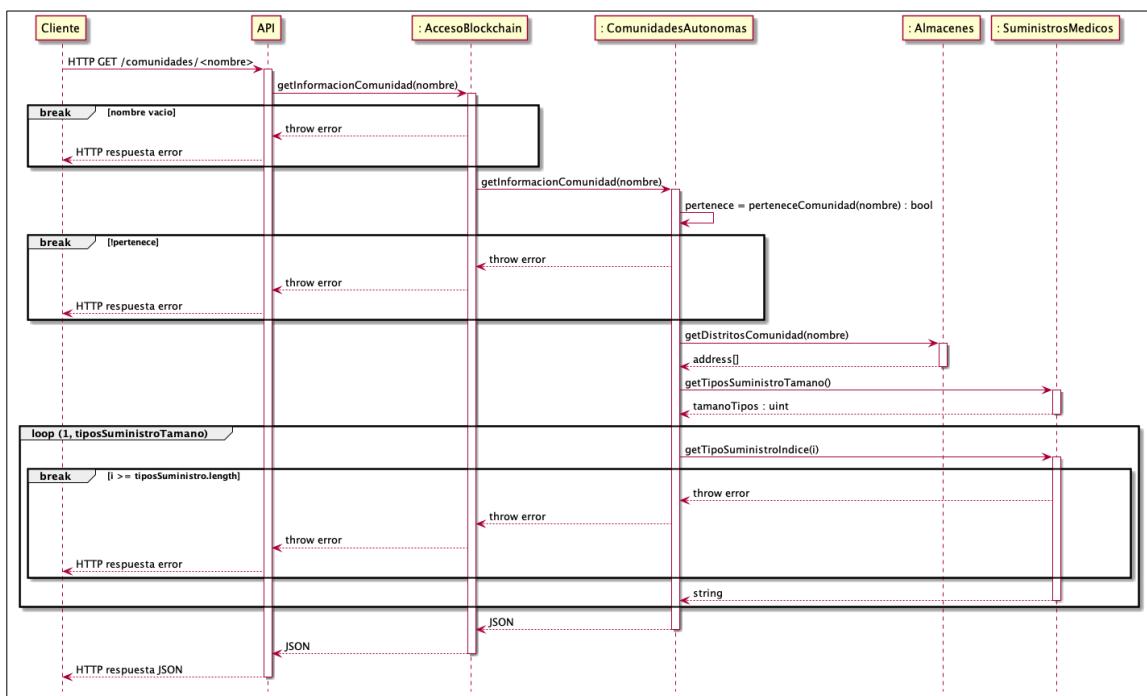


Figura 42: Diagrama de secuencia de la petición GET /comunidades/<nombre>

■ **POST /comunidades**

Variables con formato x-www-form-urlencoded de la petición:

```
nombre=<nombre>&
informacion=<informacion>
```

Añade una nueva comunidad autónoma al sistema.

Requiere iniciar sesión con una cuenta administrador.

Diagrama de secuencia:

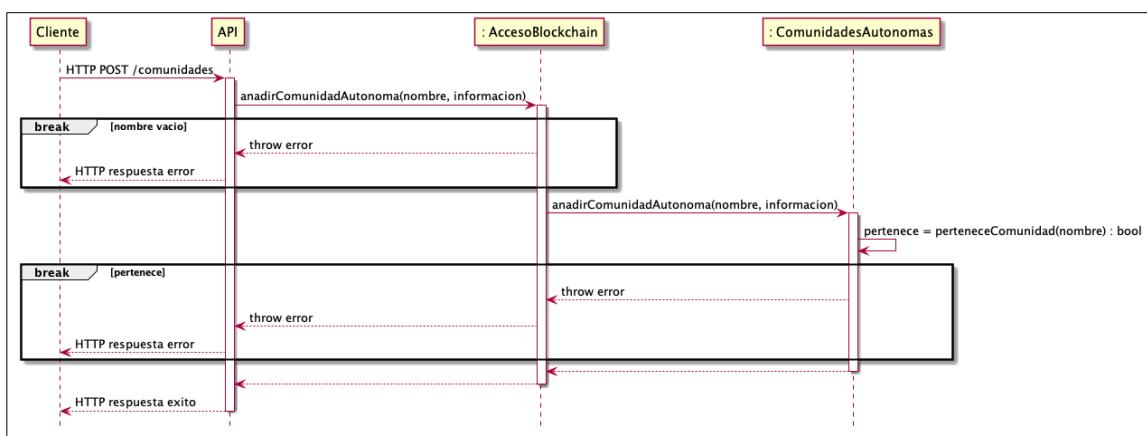


Figura 43: Diagrama de secuencia de la petición POST /comunidades

■ **PUT /comunidades/<nombre>**

Variables con formato x-www-form-urlencoded de la petición:

```
informacion=<informacion>
```

Modifica la información de la comunidad autónoma indicada.

Requiere iniciar sesión con una cuenta administrador.

Diagrama de secuencia:

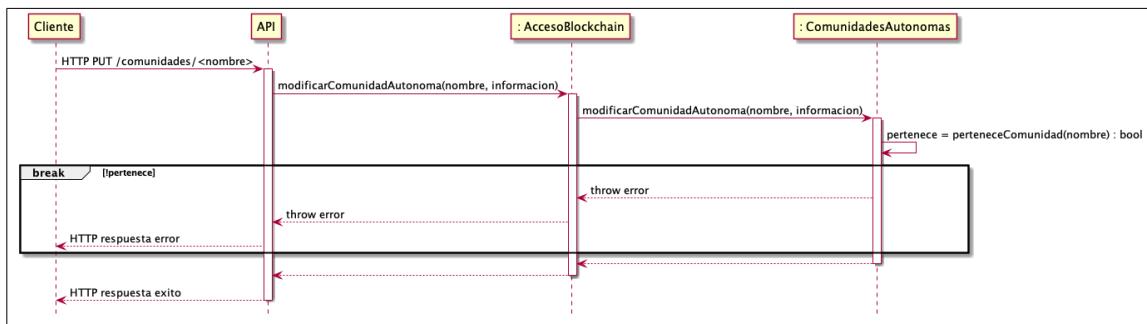


Figura 44: Diagrama de secuencia de la petición PUT /comunidades/<nombre>

### 5.11.3. Almacenes

- **GET /almacenes**

Formato JSON de la respuesta:

```
{
    "resultado": ["<almacen1>",
                  "<almacen2>", ...]
}
```

Devuelve los identificadores de todos los almacenes existentes en el sistema.

Diagrama de secuencia:

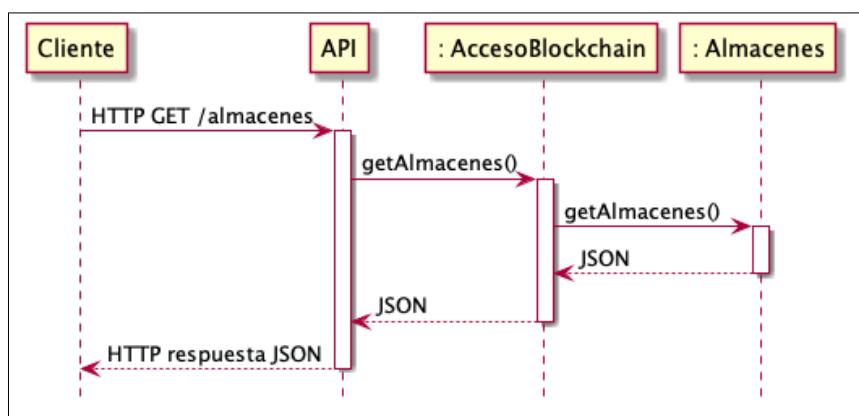


Figura 45: Diagrama de secuencia de la petición GET /almacenes

- **GET /almacenes/<direccionAlmacen>**

Formato JSON de la respuesta:

```
{
    "direccion": "<direccion>",
    "comunidad": "<nombre>",
    "informacion": "<informacion>",
    "esDistrito": <bool>,
    "poblacion": <poblacion>,
    "suministros": [<idSuministro1>, <idSuministro2>, ...],
    "tiposSuministro": {
        "<tipo1>": <unidades1>,
        "<tipo2>": <unidades2>,
        ...
    }
}
```

Devuelve información sobre el almacén indicado.

La variable población solo se incluye si estamos ante un distrito sanitario.

Diagrama de secuencia:

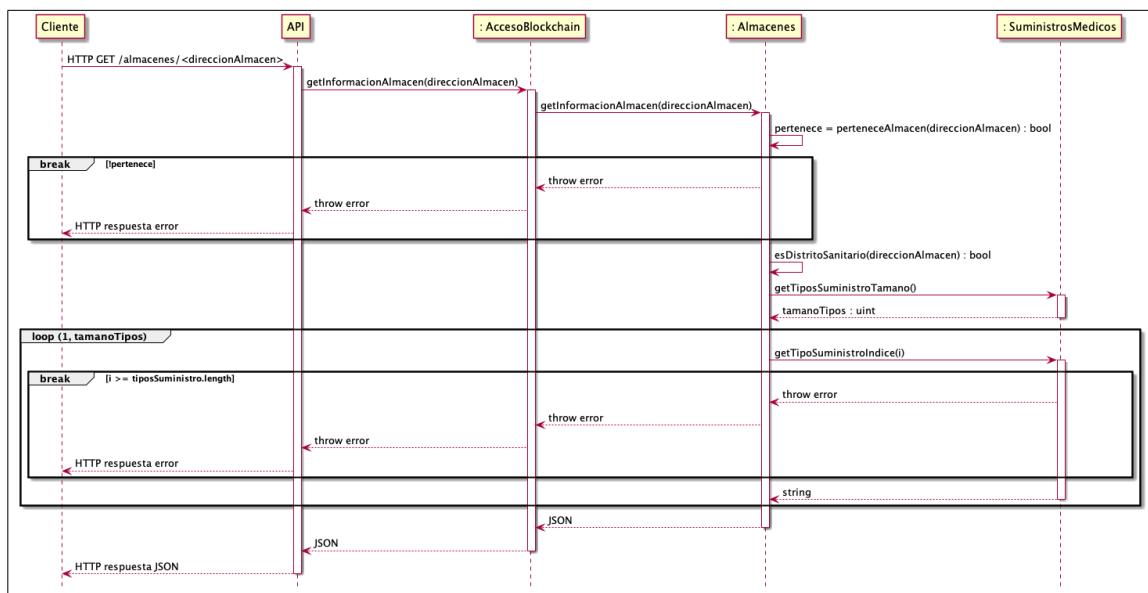


Figura 46: Diagrama de secuencia de la petición GET /almacenes/<direccionAlmacen>

#### ■ POST /almacenes

Variables con formato x-www-form-urlencoded de la petición:

```
direccionAlmacen=<direccion>&
comunidad=<nombre>&
informacion=<informacion>&
esDistritoSanitario=<bool>&
poblacion=<poblacion>
```

Añade un nuevo almacén al sistema.

Requiere iniciar sesión con una cuenta administrador. Si no es un distrito sanitario, se ignora el valor de la variable población.

Diagrama de secuencia:

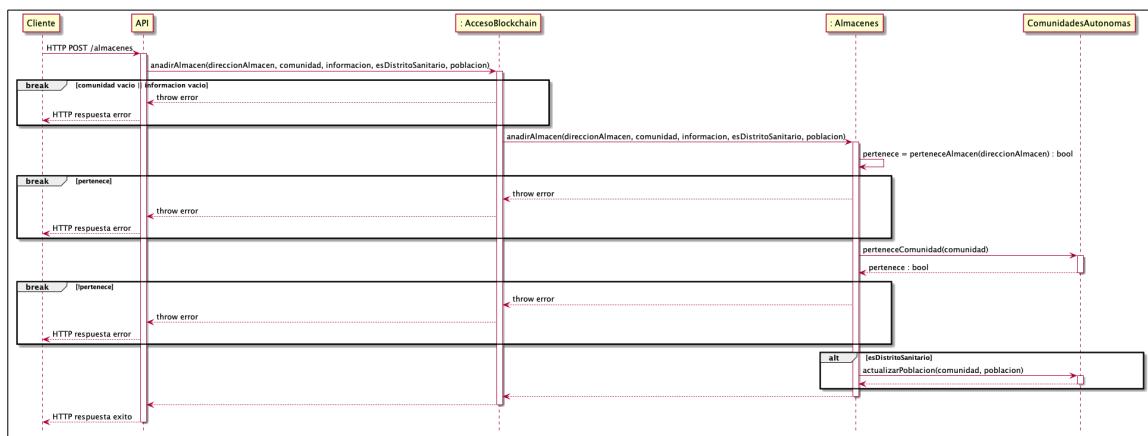


Figura 47: Diagrama de secuencia de la petición POST /almacenes

■ **PUT /almacenes/<direccionAlmacen>**

Variables con formato x-www-form-urlencoded de la petición:

```
informacion=<informacion>&
poblacion=<poblacion>
```

Modifica la información o población del almacén indicado.

Requiere iniciar sesión con una cuenta administrador. Si no queremos modificar la información del almacén, añadimos la variable informacion, pero no le damos ningún valor y si no queremos modificar la población le damos un valor a la población de -1.

Diagrama de secuencia:

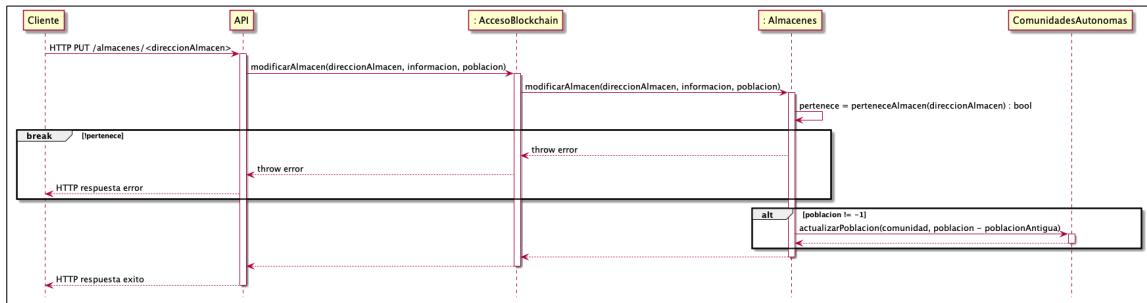


Figura 48: Diagrama de secuencia de la petición PUT /almacenes/<direccionAlmacen>

■ GET /almacenes/<direccionAlmacen>/unidades

Formato JSON de la respuesta:

```
{
    "resultado" : <unidades>
}
```

Devuelve las unidades en total que tiene el almacén indicado.

Diagrama de secuencia:

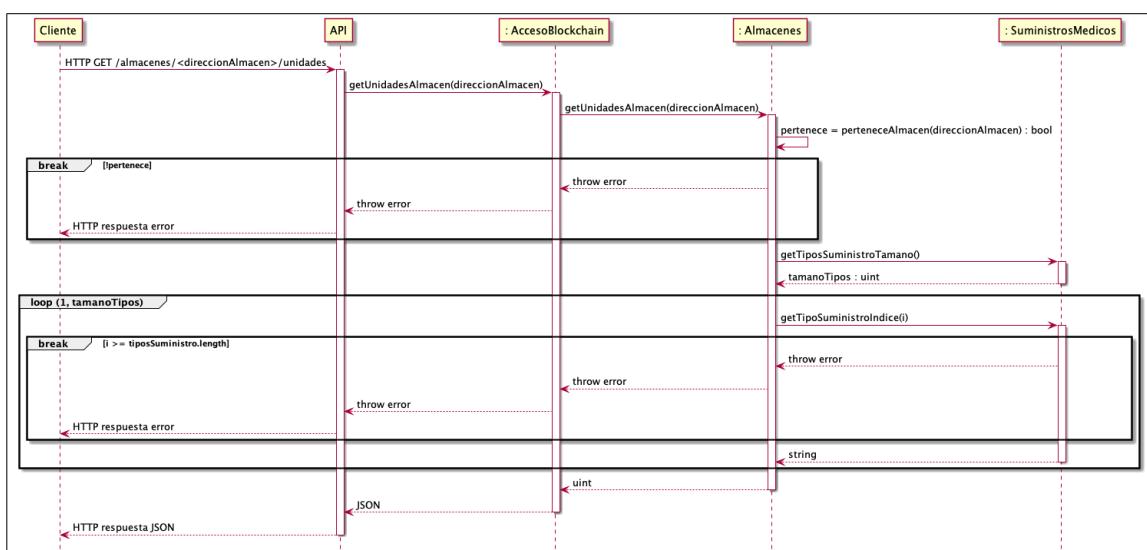


Figura 49: Diagrama de secuencia de la petición GET /almacenes/<direccionAlmacen>/unidades

■ GET /almacenes/<direccionAlmacen>/unidades-por-tipo

Formato JSON de la respuesta:

```
{
    "<tipo1>":<unidades1>,
    "<tipo2>":<unidades2>,
    ...
}
```

Devuelve las unidades por tipo de suministro que tiene el almacén indicado.

Diagrama de secuencia:

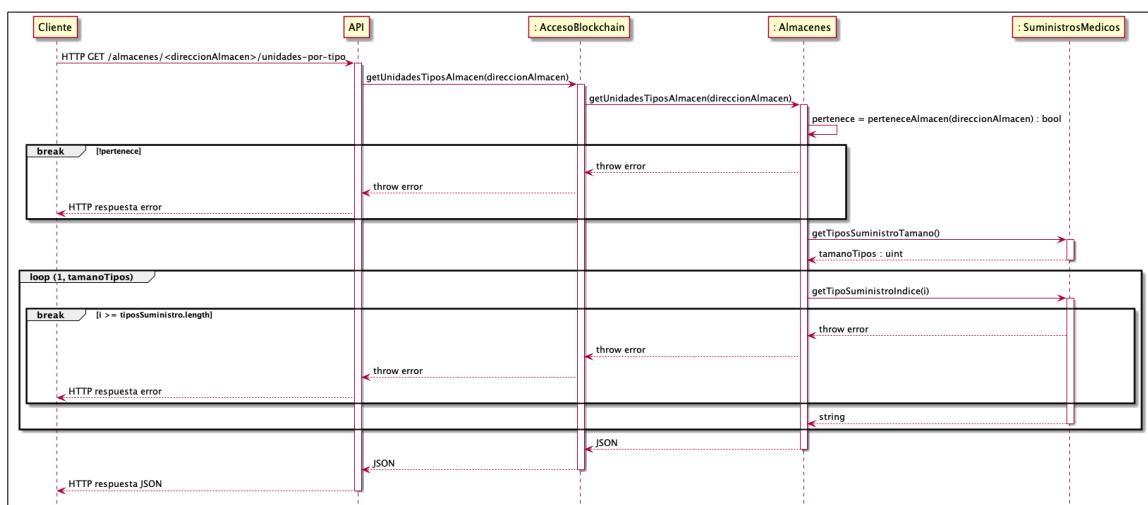


Figura 50: Diagrama de secuencia de la petición GET /almacenes/<direccionAlmacen>/unidades-por-tipo

■ GET /almacenes/<direccionAlmacen>/unidades/<tipoSuministro>

Formato JSON de la respuesta:

```
{
    "resultado" : <unidades>
}
```

Devuelve las unidades que tiene el almacén indicado de un tipo de suministro específico.

Diagrama de secuencia:

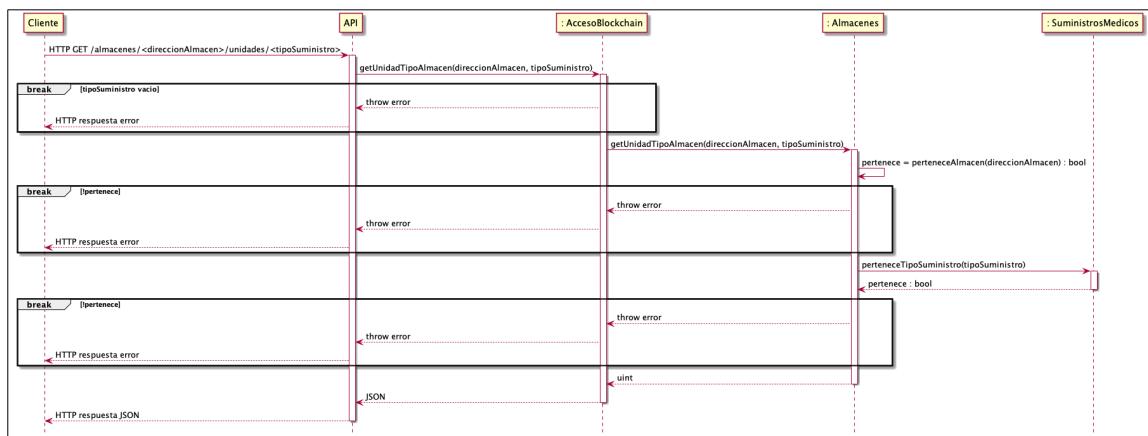


Figura 51: Diagrama de secuencia de la petición GET /almacenes/<direccionAlmacen>/unidades/<tipoSuministro>

#### 5.11.4. Suministros

- **GET /tipos-suministro**

Formato JSON de la respuesta:

```
{
    "resultado": ["<tipo1>", "<tipo2>", ...]
}
```

Devuelve los tipos de suministros existentes en el sistema.

Diagrama de secuencia:

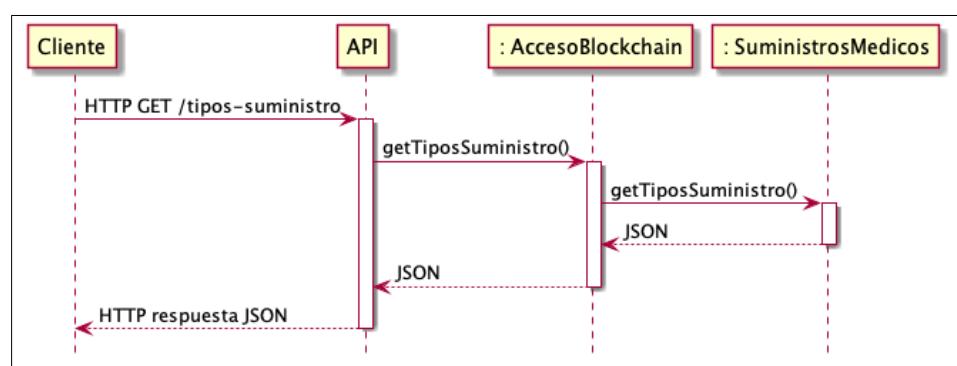


Figura 52: Diagrama de secuencia de la petición GET /tipos-suministro

- **GET /tipos-suministro/<tipoSuministro>**

La respuesta no devuelve ningún contenido JSON, tendrá estado 204 si existe el tipo de suministro o 404 si no existe.

Diagrama de secuencia:

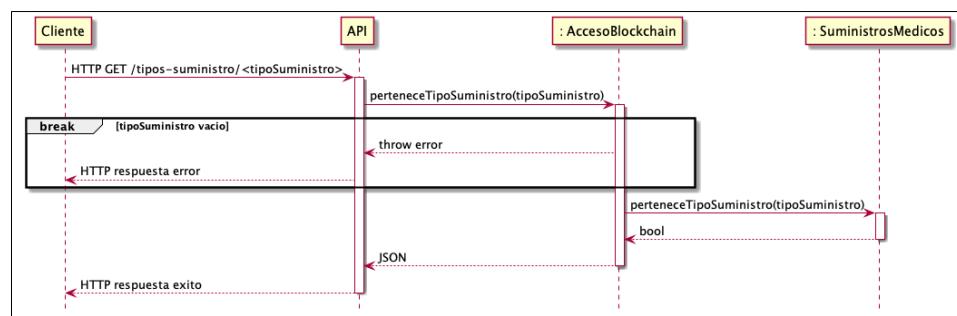


Figura 53: Diagrama de secuencia de la petición GET /tipos-suministro/<tipoSuministro>

**■ POST /tipos-suministro**

Variables con formato x-www-form-urlencoded de la petición:

```
tipo=<tipo>
```

Añade un nuevo tipo de suministro al sistema.

Requiere iniciar sesión con una cuenta administrador.

Diagrama de secuencia:

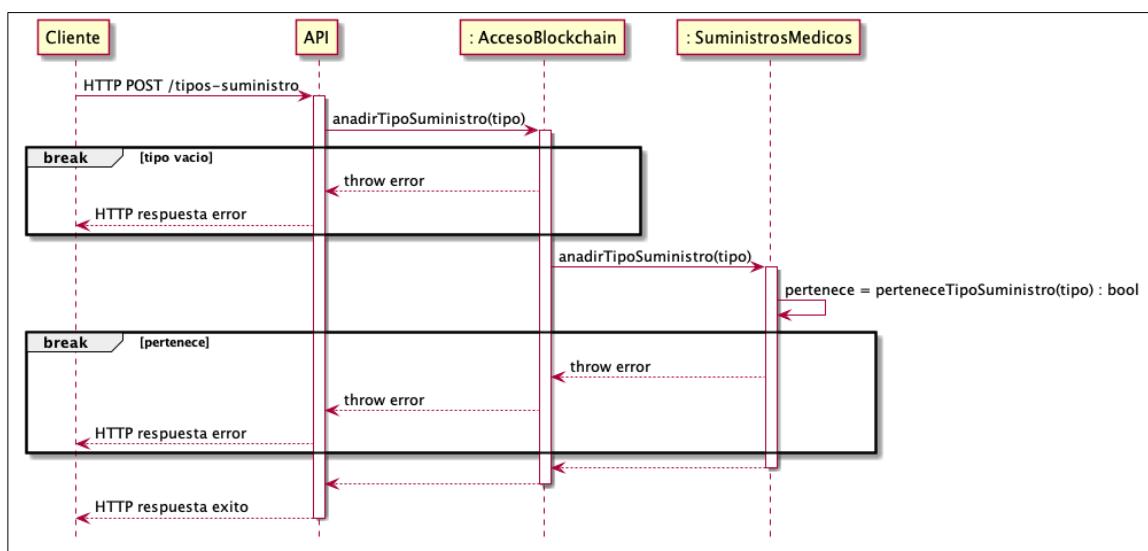


Figura 54: Diagrama de secuencia de la petición POST /tipos-suministro

- GET /suministros

Formato JSON de la respuesta:

```
{  
    "resultado": [<idSuministro1>,  
                  <idSuministro2>, ...]  
}
```

Devuelve los identificadores de todos los suministros existentes en el sistema.

Diagrama de secuencia:

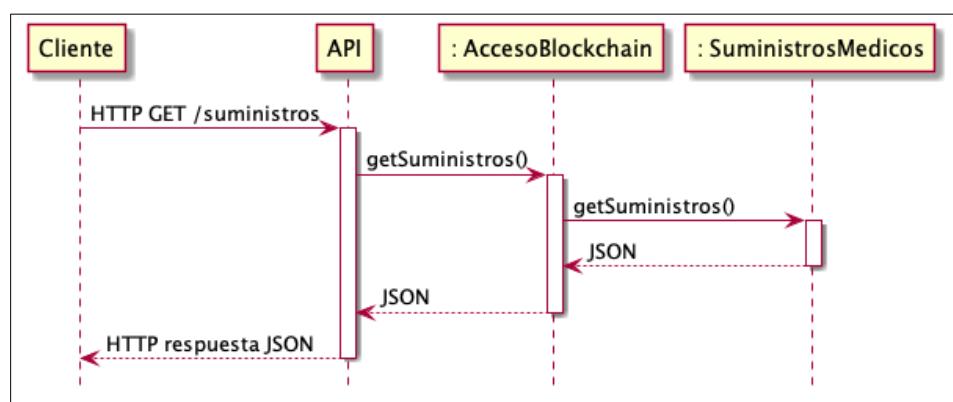


Figura 55: Diagrama de secuencia de la petición GET /suministros

■ GET /suministros/<idSuministro>

Formato JSON de la respuesta:

```
{
    "id": <idSuministro>,
    "tipo": "<tipoSuministro>",
    "tamano": <tamano>,
    "almacenActual": "<almacen>",
    "comunidadAsignada": "<nombre>",
    "distritoDestino": "<distrito>",
    "enViaje": <bool>,
    "utilizado": <bool>
}
```

Devuelve información sobre el suministro indicado. En el caso de que el suministro este en viaje (variable enViaje tiene valor true) la variable almacenActual indica el almacén del que acaba de salir. En el caso de que el suministro haya sido utilizado (variable utilizado tiene valor true) entonces la variable almacenActual indica el ultimo almacén en el que ha estado (que será su distrito sanitario asignado).

Diagrama de secuencia:

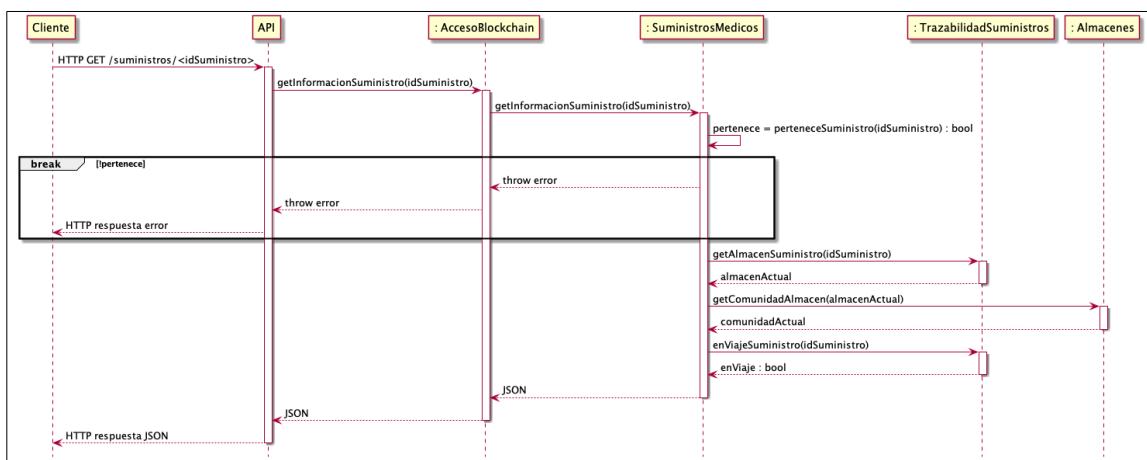


Figura 56: Diagrama de secuencia de la petición GET /suministros/<idSuministro>

**■ POST /suministros**

Variables con formato x-www-form-urlencoded de la petición:

```
tipoSuministro=<tipo>&
tamanoSuministro=<tamano>&
preasignado=<comunidad>&
fecha=<fecha>
```

Añade un suministro nuevo al sistema, esta creación de un suministro representa la llegada de un suministro médico a España.

En el caso de no tener ninguna comunidad preasignada, se añade la variable preasignado sin darle ningún valor. La fecha se rellenará automáticamente con el valor correcto (Date.now()).

Requiere iniciar sesión con una cuenta almacén y la dirección de la cuenta utilizada nos indica el almacén al que ha llegado el suministro.

Diagrama de secuencia en la página siguiente debido a su tamaño:

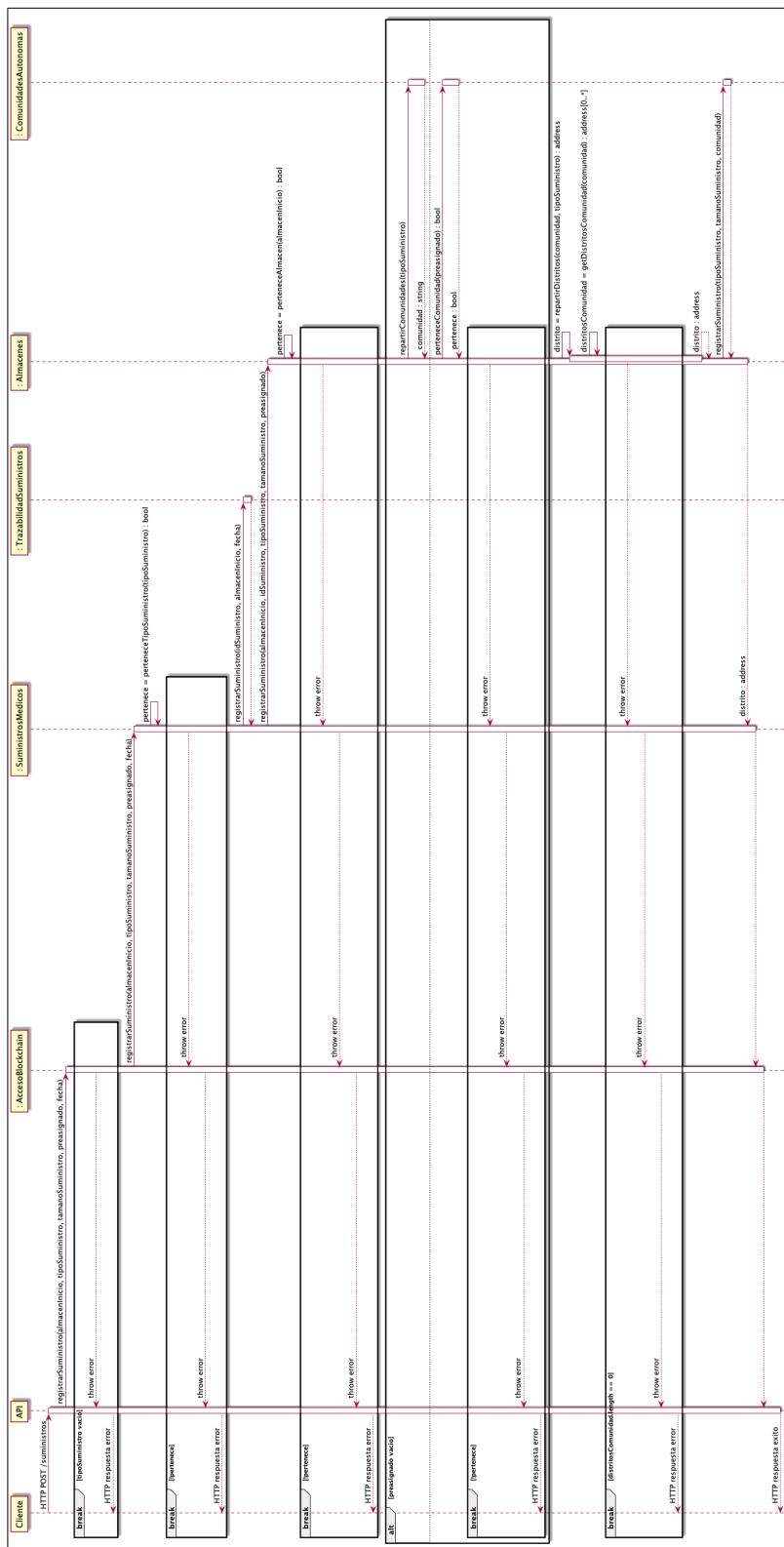


Figura 57: Diagrama de secuencia de la petición POST /suministros

■ GET /suministros/<idSuministro>/trazabilidad

Formato JSON de la respuesta:

```
{
    "resultado": [
        {"almacenOrigen": "<direccion1>" ,
         "almacenDestino": "<direccion2>" ,
         "fechaSalida": <fecha1> ,
         "fechaEntrada": <fecha2>} ,
        {"almacenOrigen": "<direccion1>" ,
         "almacenDestino": "<direccion2>" ,
         "fechaSalida": <fecha1> ,
         "fechaEntrada": <fecha2>} ,
        ...
    ]
}
```

Devuelve la trazabilidad completa del suministro indicado.

Diagrama de secuencia:

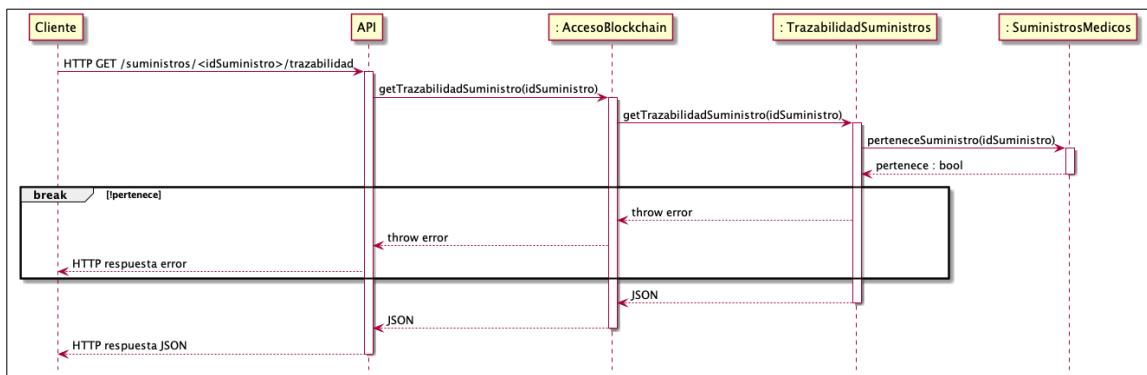


Figura 58: Diagrama de secuencia de la petición GET /suministros/<idSuministro>/trazabilidad

- **POST /suministros/<idSuministro>/trazabilidad (tipo salida)**

Variables con formato x-www-form-urlencoded de la petición:

```
tipo="salida"&
idSuministro=<idSuministro>&
fecha=<fecha>
```

Se envía esta petición cuando un suministro sale de un almacén y guardará esta información para mantener la trazabilidad, además de modificar las variables necesarias para mantener el estado consistente.

Requiere iniciar sesión con una cuenta almacén y la dirección de la cuenta utilizada nos indica el almacén del que sale el suministro.

Diagrama de secuencia en la página siguiente debido a su tamaño:

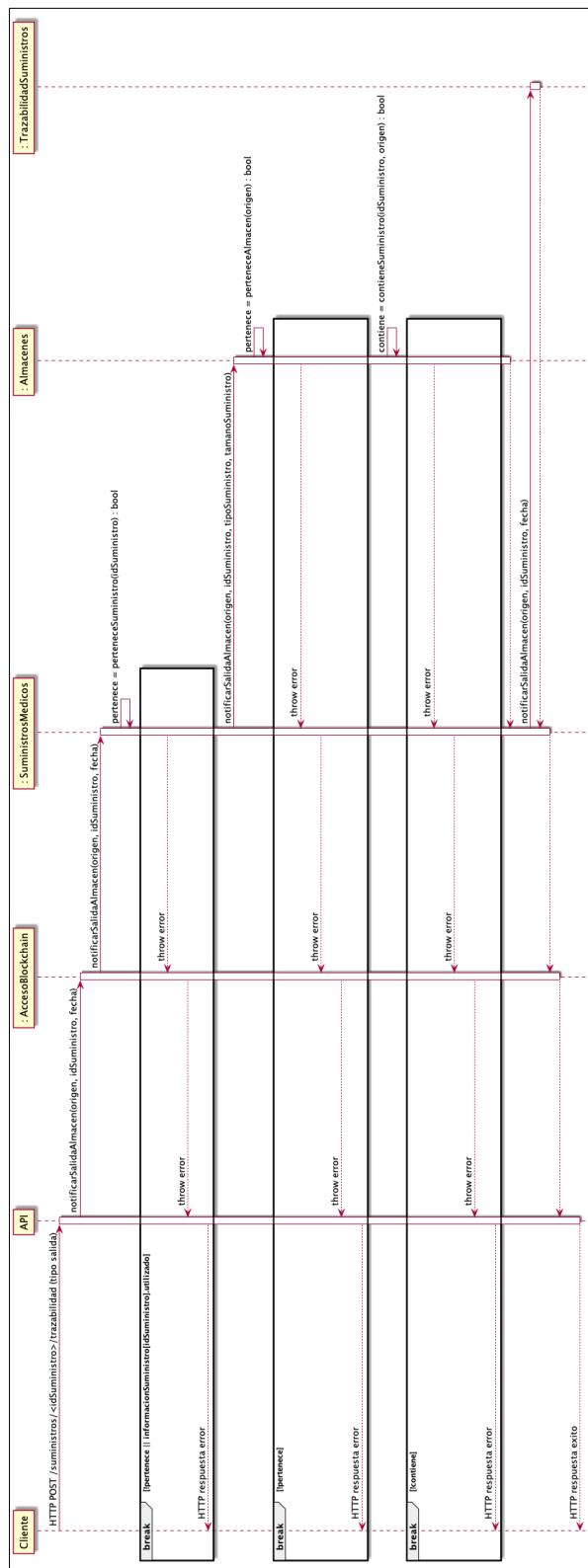


Figura 59: Diagrama de secuencia de la petición POST /suministros/{idSuministro}/trazabilidad (tipo salida)

- **POST /suministros/<idSuministro>/trazabilidad (tipo llegada)**

Variables con formato x-www-form-urlencoded de la petición:

```
tipo="llegada"&
idSuministro=<idSuministro>&
fecha=<fecha>
```

Se envía esta petición cuando llega un suministro a un almacén. Se guardará esta información para mantener la trazabilidad y se modificarán las variables necesarias para mantener el estado consistente.

Requiere iniciar sesión con una cuenta almacén y la dirección de la cuenta utilizada nos indica el almacén al que ha llegado el suministro.

Diagrama de secuencia en la página siguiente debido a su tamaño:

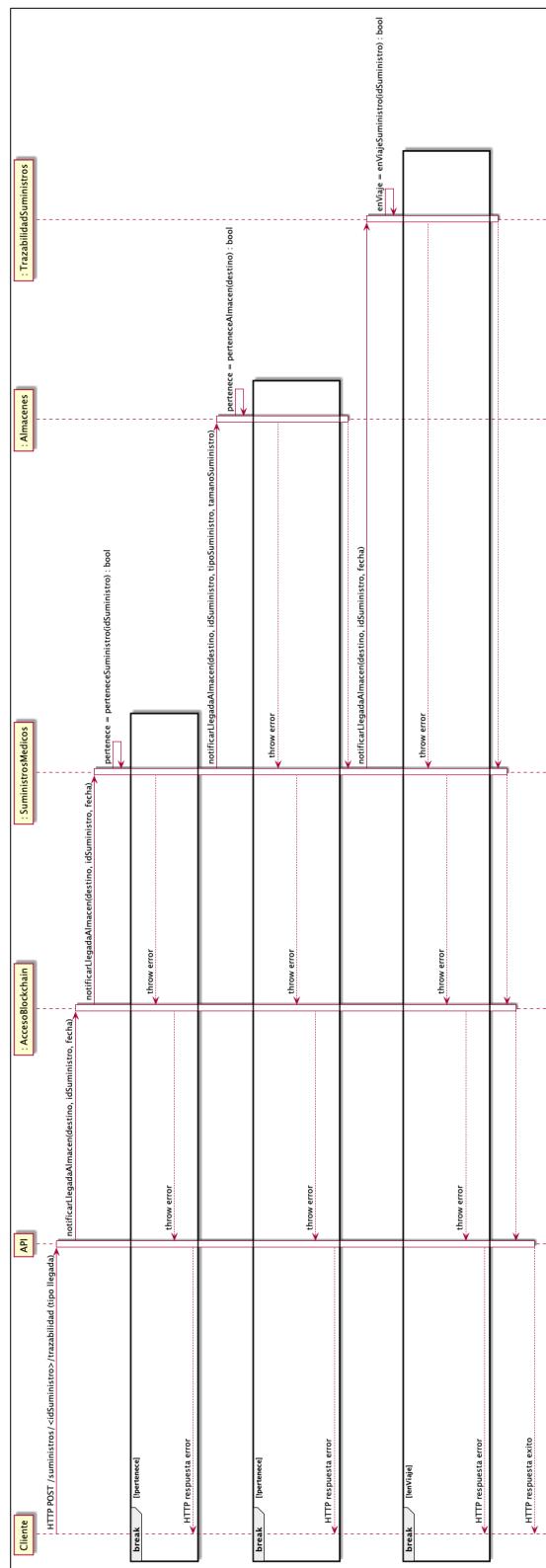


Figura 60: Diagrama de secuencia de la petición POST /suministros/<idSuministro>/trazabilidad (tipo llegada)

- **POST /suministros/<idSuministro>/trazabilidad (tipo utilizar)**

Variables con formato x-www-form-urlencoded de la petición:

```
tipo="utilizar"&
idSuministro=<idSuministro>&
fecha=<fecha>
```

Se envía esta petición cuando se abre un suministro, de forma que se empezarán a utilizar las unidades individuales que guarda. Una vez abierto se elimina del almacén y se reducen las unidades de este tipo de suministro que tiene asignada la comunidad autónoma y el distrito sanitario al que pertenece.

Requiere iniciar sesión con una cuenta almacén y la dirección de la cuenta utilizada nos indica el distrito sanitario que está abriendo el suministro, que debe coincidir con el distrito asignado al suministro.

Diagrama de secuencia en la página siguiente debido a su tamaño:

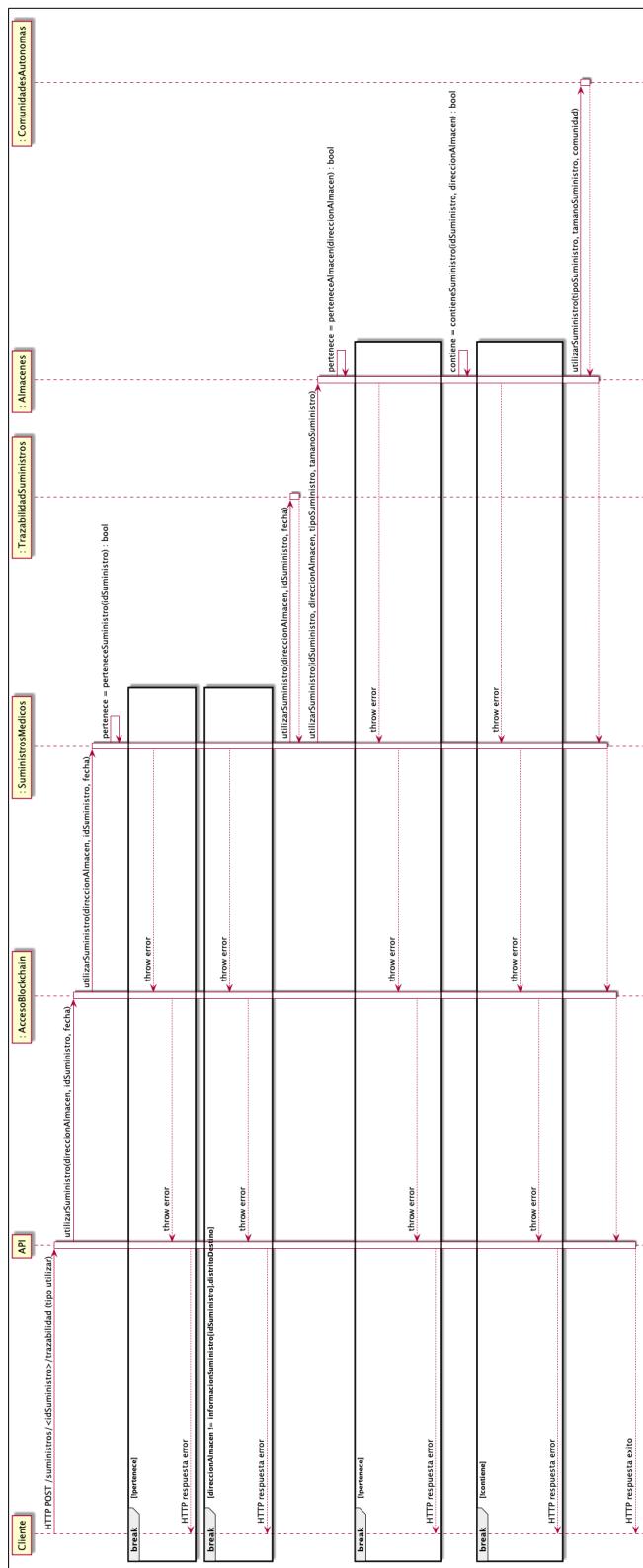


Figura 61: Diagrama de secuencia de la petición POST /suministros/<idSuministro>/trazabilidad (tipo utilizar)

## **Capítulo 6**

# **Conclusiones y trabajo futuro**

En este capítulo se exponen las conclusiones del trabajo, el posible trabajo futuro que se podría hacer a partir de él y una valoración personal del desarrollo del trabajo.

## 6.1. Conclusiones

El objetivo general del proyecto ha sido la creación de una red blockchain y el desarrollo como prueba de concepto de una aplicación descentralizada que se desplegará y ejecutará en dicha red y que nos permite seguir la trazabilidad de los suministros médicos. Revisaremos los objetivos específicos mostrados en el apartado “1.3. Objetivos”, comprobando si se han cumplido de forma satisfactoria.

El primer objetivo específico trata sobre la investigación de tecnologías blockchain y los elementos que componen una red blockchain, durante esta investigación elegimos la tecnología Ethereum y vimos que está en innovación y actualización constante. Llegamos a entender bien cómo funcionan las redes blockchain, con el enlazado de los bloques que hacen formando la cadena y entendiendo en detalle cómo funcionan los distintos nodos que la componen, cosa que resulta muy útil y es necesario tener este dominio ya que al estar en constante actualización los tutoriales se desactualizan rápido y en el tutorial que se siguió para la creación de una red blockchain descubrimos múltiples errores. Esta constante actualización nos obliga a estar al día en este campo. Este objetivo se aborda en las secciones “2. Tecnología blockchain: Fundamentos”, “3. Estado del Arte” y “4.1. Elección de la tecnología blockchain”. Consideramos que este objetivo específico de investigación se ha cumplido correctamente.

El segundo objetivo específico es la creación de una red virtual de servidores, que consiste en la configuración de un servidor, la creación y configuración de máquinas virtuales en él y la configuración de estas últimas para hacerlas accesibles desde el exterior. El uso de máquinas virtuales ha sido clave para el proyecto, pues no solo abarata los costes, sino que además nos permite hacer copias de una máquina virtual ya configurada, pudiendo clonarla tantas veces como queramos para ir probando distintas configuraciones y nodos en la creación de la red blockchain con facilidad. Este objetivo se aborda en la sección “4. Creación de una red blockchain basada en la plataforma Ethereum” y se ha cumplido con éxito.

El siguiente objetivo específico es la creación de una red blockchain, donde se instalan los nodos y se configura la red blockchain. Debido a ser una tecnología actual y en desarrollo, tenemos muchos problemas con los bugs. Uno de los bugs más problemáticos es el que ocurre cuando se crea un signer en la misma máquina donde se está utilizando Puppeth, ya que este

nodo signer no funciona bien y por tanto tampoco la red blockchain, que no empieza a generar bloques y se queda atascada, por lo que debemos evitar esta situación.

Destacamos otros bugs que nos ocurrieron, como el de Faucet, que dejaba de funcionar al poco tiempo, en este caso lo que hicimos fue crear un Faucet de cero. Algo similar ocurrió con el Explorer, que nos obligó a modificar el código de Geth y volver a compilar. Con el otro servicio Wallet también nos encontramos otro bug y tuvimos que instalar una versión más reciente de la incluida en Puppeth. Este objetivo se aborda en la sección “4. Creación de una red blockchain basada en la plataforma Ethereum” y se ha cumplido con éxito.

Con la red blockchain funcional creada, pasamos al último objetivo específico, el de desarrollo de una aplicación descentralizada que haga uso de ella. En este apartado aprendimos como programar en la red blockchain, lo cual resultó tener sus similitudes y diferencias respecto a la programación orientada a objetos. Se comparten muchos detalles como el poder separar el código en clases (contratos), crear nuevas instancias de los contratos, realizar llamadas entre estos, pero con muchos matices que hacen que nos replanteemos el diseño que se haría si se siguiese el diseño orientado a objetos.

Crear una nueva instancia implica subir el contrato de nuevo a la red blockchain y para realizar llamadas entre métodos de contratos se tienen que conocer las direcciones en las que están subidos. Esto junto a aspectos como el espacio utilizado por la red blockchain o el consumo del gas y otros que se explican el apartado “5.5. Programación en Ethereum, diferencias respecto a programación orientada a objetos” hace que haya que realizar un diseño basado en Singleton, es decir, en clases de las cuales creamos solo una instancia (se suben solo una vez).

Otra de las limitaciones que presenta la programación en la red blockchain es la dificultad de arreglar bugs, pues implica volver a subir el contrato y por tanto transferir los valores de las variables entre el contrato antiguo y el nuevo.

Y para llamar a los métodos de un contrato cuando estos modifican la red blockchain tenemos que crear una transacción y por tanto esperar a que sea añadida en un bloque. Esto último nos lleva a utilizar Node.js para crear la API, gracias a su forma eficiente de manejar operaciones asíncronas. Utilizando Node.js se creó una API REST para comunicarnos con la aplicación en la red blockchain.

Durante el desarrollo de la aplicación el principal problema que nos encontramos fue los errores genéricos que muestra Geth, que no informaba con exactitud de lo que estaba causando el problema y provocaba que tu-

viésemos que ir comprobando poco a poco donde puede haber ocurrido el error, si es en la API REST, quizás un bug de la API Web3.js, o un bug de Geth o por supuesto un bug de nuestra aplicación. Teniendo en cuenta la documentación que a veces no entra en detalles, la dificultad de hacer este proceso de búsqueda de bug y al ser una tecnología reciente el no poder encontrar información sobre el problema en Internet hace que sea difícil arreglar los problemas que surjan.

Por mencionar uno específico que fue el que más problema nos dio es el de necesitar especificar un límite de gas de una transacción en una operación que no genera una transacción, lo cual de primeras no tiene mucho sentido, pero sospechamos que, aunque no se genere una transacción, utiliza el mecanismo del límite de gas para evitar quedarse atascado en un bucle infinito. Muy relacionado con este problema está la necesidad de especificar una cuenta que envíe la transacción, en esas mismas operaciones que no generan transacciones, cosa que tampoco tiene sentido. En este caso sospechamos que debido a la reutilización de código compartirán alguna parte que haga que tengamos que indicar la cuenta, aunque no se vaya a utilizar.

Como se puede apreciar, el desarrollo de aplicaciones descentralizadas para las redes blockchain incluye problemas únicos a este sistema y, por tanto, conforme avanzábamos, acabó provocando muchos cambios, pero gracias al uso de las metodologías de desarrollo de software ágil pudimos adaptarnos a estos cambios sin problemas y llegar a terminar la aplicación.

Consideramos que este objetivo específico también se ha cumplido con éxito, ya que hemos llegado a implementar las funcionalidades que teníamos pensado incluir. Este objetivo se aborda en la sección “5. Desarrollo de una aplicación descentralizada”. Como hemos cumplido de forma satisfactoria todos los objetivos específicos consideramos que se ha cumplido sin problemas el objetivo general del proyecto.

## 6.2. Trabajo futuro

De cara al futuro, existen muchas formas en las que se podría ampliar este proyecto. Mencionaremos las ampliaciones que pueden ser más interesantes y directas a llevar a cabo:

- Gracias a utilizar una API REST se puede desarrollar una aplicación para plataformas y dispositivos móviles, lo cual sería algo muy interesante para demostrar el uso de la red blockchain con este tipo de tecnologías. Por ejemplo, la aplicación móvil podría leer un código QR que tendría el identificador del suministro y recibir toda la trazabilidad del suministro específico.

- Pasar de una prueba de concepto a una versión que cumpla la legislación española en lo referente al reparto y seguimiento de los suministros médicos.
- Una vez hecho lo anterior se podría pasar de cubrir la trazabilidad de los suministros médicos solo en España a tener cobertura global, pudiendo seguir la trazabilidad desde la fábrica en la que se produjo hasta llegar al hospital en el que se utiliza. Si fuésemos por este camino sería interesante intentar obtener financiación y llegar a hacer el proyecto una realidad.
- Realizar un análisis del consumo de gas, realizando múltiples ejecuciones, con el objetivo de optimizar el código e intentar reducir este consumo al máximo.

El gas es una unidad abstracta y cada operación que se haga consume una cantidad determinada de esta. En el caso de estar en una red blockchain pública, el gas que consumamos se traduciría en dinero real, por lo que es un tema de suma importancia si nuestro contrato va a ser ejecutado en una red blockchain pública.

### 6.3. Valoración personal del desarrollo del trabajo

Como valoración personal cabe mencionar que trabajar con las tecnologías blockchain ha sido una experiencia muy enriquecedora, que ha puesto a prueba mis conocimientos adquiridos durante el estudio del grado gracias a las diferencias y particularidades propias que implica el trabajar con una red blockchain. Con el proyecto terminado me encuentro satisfecho con los nuevos conocimientos que he adquirido y siento que un proyecto de este estilo, que cubre varias áreas diferentes y ha implicado poner en práctica conocimientos adquiridos y aprender tecnologías recientes y prometedoras a nivel profesional, ha sido perfecto para terminar mi preparación académica y prepararme para el futuro laboral o la continuación de mis estudios en un máster.

# Bibliografía

- [1] *Página web Indeed, media salarial de ingenieros informáticos en España.* dirección: <https://es.indeed.com/salaries/ingeniero-inform%C3%A1tico-Salaries> (visitado 03-09-2020).
- [2] M. Crosby, P. Pattanayak, S. Verma, V. Kalyanaraman y col., “Blockchain technology: Beyond bitcoin,” *Applied Innovation*, vol. 2, n.º 6-10, págs. 8-13, 2016.
- [3] *Documentación oficial de Geth, red privada.* dirección: <https://geth.ethereum.org/docs/interface/private-network> (visitado 07-08-2020).
- [4] *GitHub de una implementación de Ethstats.* dirección: <https://github.com/cubedro/eth-netstats> (visitado 07-08-2020).
- [5] *GitHub de Blockscout, un Explorer.* dirección: <https://github.com/poanetwork/blockscout> (visitado 07-08-2020).
- [6] *Documentación oficial de Blockscout, configuración del nodo.* dirección: <https://docs.blockscout.com/for-developers/information-and-settings/client-settings-parity-geth-ganache> (visitado 07-08-2020).
- [7] *GitHub de MyEtherWallet, implementación de una Wallet.* dirección: <https://github.com/MyEtherWallet/MyEtherWallet> (visitado 07-08-2020).
- [8] J. Ng, *Artículo de Medium sobre el servicio Faucet*, dic. de 2019. dirección: <https://medium.com/coinmonks/ropsten-ethereum-faucet-how-it-works-c5703f769c2a> (visitado 07-08-2020).
- [9] S. Gupta y M. Sadoghi, “Blockchain Transaction Processing,” en *Encyclopedia of Big Data Technologies*, S. Sakr y A. Zomaya, eds. Cham: Springer International Publishing, 2018, págs. 4-5, ISBN: 978-3-319-63962-8. DOI: 10.1007/978-3-319-63962-8\_333-1. dirección: [https://doi.org/10.1007/978-3-319-63962-8\\_333-1](https://doi.org/10.1007/978-3-319-63962-8_333-1).
- [10] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” págs. 2-4, 2008. dirección: <https://bitcoin.org/bitcoin.pdf> (visitado 07-08-2020).

- [11] P. Szilágyi, *Ethereum Improvement Proposals, EIP-225: Clique proof-of-authority consensus protocol*, mar. de 2017. dirección: <https://eips.ethereum.org/EIPS/eip-225> (visitado 07-08-2020).
- [12] J. Siim, “Proof-of-stake,” en *Research Seminar in Cryptography*, 2017, págs. 2-3.
- [13] Página web coinmarketcap, criptomonedas bitcoin. dirección: <https://coinmarketcap.com/currencies/bitcoin/> (visitado 07-08-2020).
- [14] V. Buterin, *Página web oficial de Ethereum, whitepaper*, 2013. dirección: <https://ethereum.org/whitepaper/> (visitado 07-08-2020).
- [15] *Wiki oficial de Ethereum, implementaciones de Ethereum*, 11 de jun. de 2020. dirección: <https://eth.wiki/en/clients/clients-tools-dapp-browsers-wallets-and-other-projects> (visitado 07-08-2020).
- [16] *Wiki oficial de Ethereum, lenguajes para programar Smart Contracts, apartado Programming Languages that Compile into EVM*, 11 de jun. de 2020. dirección: [https://eth.wiki/en/concepts/evm/ethereum-virtual-machine-\(evm\)-awesome-list](https://eth.wiki/en/concepts/evm/ethereum-virtual-machine-(evm)-awesome-list) (visitado 07-08-2020).
- [17] *Documentación oficial de Solidity, memoria en la Ethereum Virtual Machine, apartado Storage, Memory and the Stack*. dirección: <https://solidity.readthedocs.io/en/latest/introduction-to-smart-contracts.html> (visitado 07-08-2020).
- [18] *Documentación oficial de Geth, gestión de cuentas*. dirección: <https://geth.ethereum.org/docs/interface/managing-your-accounts> (visitado 07-08-2020).
- [19] *Documentación oficial de Quorum*. dirección: <http://docs.goquorum.com/en/latest/> (visitado 07-08-2020).
- [20] *Wiki oficial de Hyperledger*, feb. de 2020. dirección: <https://wiki.hyperledger.org/> (visitado 07-08-2020).
- [21] *Documentación oficial de Hyperledger Fabric*. dirección: <https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html> (visitado 07-08-2020).
- [22] *GitHub de Aleth, implementación de Ethereum*. dirección: <https://github.com/ethereum/aleth> (visitado 07-08-2020).
- [23] *GitHub de Go Ethereum (Geth), implementación de Ethereum*. dirección: <https://github.com/ethereum/go-ethereum> (visitado 07-08-2020).
- [24] *GitHub de Trinity, implementación de Ethereum*. dirección: <https://github.com/ethereum/trinity> (visitado 07-08-2020).
- [25] “Documentación oficial de Proxmox, versión PDF,” págs. 106-109, 158-170, jul. de 2020. dirección: <https://pve.proxmox.com/pve-docs/pve-admin-guide.pdf> (visitado 07-08-2020).

- [26] *Documentación oficial de Proxmox*, ago. de 2020. dirección: [https://pve.proxmox.com/wiki/Main\\_Page](https://pve.proxmox.com/wiki/Main_Page) (visitado 07-08-2020).
- [27] P. Szilágyi, *Blog oficial de Ethereum, presentación de Puppeth*, abr. de 2017. dirección: <https://blog.ethereum.org/2017/04/14/geth-1-6-puppeth-master/> (visitado 07-08-2020).
- [28] *Documentación oficial de Docker*. dirección: <https://docs.docker.com/engine/> (visitado 07-08-2020).
- [29] *Documentación oficial de Geth, opciones del comando geth*. dirección: <https://geth.ethereum.org/docs/interface/command-line-options> (visitado 07-08-2020).
- [30] *Wiki oficial de Ethereum, formato enode*, 11 de jun. de 2020. dirección: <https://eth.wiki/en/fundamentals/enode-url-format> (visitado 07-08-2020).
- [31] *Servidor E3-SSD-1-32 de Soyousart*. dirección: <https://www.soyousart.com/es/ofertas/1801sys47.xml> (visitado 07-08-2020).
- [32] *Tutorial de OVH de configuración de red de máquinas virtuales*, mayo de 2019. dirección: <https://docs.ovh.com/gb/en/dedicated/network-bridging/> (visitado 07-08-2020).
- [33] *Documentación oficial de Proxmox, sección Qemu guest agent*, mayo de 2020. dirección: <https://pve.proxmox.com/wiki/Qemu-guest-agent> (visitado 07-08-2020).
- [34] *Documentación oficial de Docker, configuración tras la instalación*. dirección: <https://docs.docker.com/engine/install/linux-postinstall/> (visitado 07-08-2020).
- [35] *Documentación oficial de Docker, herramienta Docker Compose*. dirección: <https://docs.docker.com/compose/> (visitado 07-08-2020).
- [36] *Documentación oficial de Geth, instalación de Geth*. dirección: <https://geth.ethereum.org/docs/install-and-build/installing-geth> (visitado 07-08-2020).
- [37] F. Lange, *Documentación oficial del protocolo de descubrimiento de nodos utilizado por Ethereum*, oct. de 2019. dirección: <https://github.com/ethereum/devp2p/blob/master/discv4.md> (visitado 07-08-2020).
- [38] *Documentación oficial de Geth, API clique*. dirección: <https://geth ethereum.org/docs/rpc/ns-clique> (visitado 07-08-2020).
- [39] *Documentación oficial de Geth, consola del nodo*. dirección: <https://geth.ethereum.org/docs/interface/javascript-console> (visitado 07-08-2020).

- [40] *Documentación oficial de Geth, configuración de JSON-RPC API.* dirección: <https://geth.ethereum.org/docs/rpc/server> (visitado 07-08-2020).
- [41] *Documentación oficial de Geth, API admin.* dirección: <https://geth.ethereum.org/docs/rpc/ns-admin> (visitado 07-08-2020).
- [42] H. C. Van Tilborg y S. Jajodia, *Encyclopedia of cryptography and security.* Springer Science & Business Media, 2014, pág. 1042.
- [43] V. Buterin, *Ethereum Improvement Proposals, EIP-155: Simple replay attack protection,* oct. de 2016. dirección: <https://eips.ethereum.org/EIPS/eip-155> (visitado 07-08-2020).
- [44] C. Cusce, *Artículo de Medium, creación de una red blockchain Ethereum que utilice Proof of Authority,* jun. de 2018. dirección: <https://medium.com/@collin.cusce/using-puppeth-to-manually-create-an-ethereum-proof-of-authority-clique-network-on-aws-ae0d7c906cce> (visitado 07-08-2020).
- [45] *Wiki oficial de Ethereum, explicación sobre gas, apartado Gas and Fees,* 11 de jun. de 2020. dirección: <https://eth.wiki/fundamentals/design-rationale> (visitado 07-08-2020).
- [46] Nombre de usuario: blazejkrzak, *GitHub de Geth, bug en la instalación del Explorer incluido en Puppeth,* feb. de 2020. dirección: <https://github.com/ethereum/go-ethereum/issues/20706> (visitado 07-08-2020).
- [47] *GitHub de etherwallet, versión anterior a MyEtherWallet.* dirección: <https://github.com/MyEtherWallet/etherwallet> (visitado 07-08-2020).
- [48] *GitHub de otro Ethstats, incluye versión específica para redes Proof of Authority.* dirección: <https://github.com/goerli/ethstats-server> (visitado 07-08-2020).
- [49] *Drawio, herramienta para dibujo de diagramas. Se ha utilizado la herramienta y los iconos que pone a disposición para sus usuarios para el diagrama de visión general del sistema.* dirección: <https://github.com/jgraph/drawio> (visitado 07-08-2020).
- [50] K. Schwaber y J. Sutherland, *The Scrum Guide,* nov. de 2017. dirección: <https://www.scrumguides.org/scrum-guide.html> (visitado 07-08-2020).
- [51] *Documentación oficial de Geth, GO API.* dirección: <https://geth.ethereum.org/docs/dapp/native> (visitado 07-08-2020).
- [52] *Wiki oficial de Ethereum, JSON RPC,* 11 de jun. de 2020. dirección: <https://eth.wiki/json-rpc/API> (visitado 07-08-2020).
- [53] *GitHub de Web3.js.* dirección: <https://github.com/ethereum/web3.js> (visitado 07-08-2020).

- [54] *Documentación oficial de Node.js.* dirección: <https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/> (visitado 07-08-2020).
- [55] *Documentación oficial de Solidity, lenguaje de programación de smart contracts de Ethereum.* dirección: <https://solidity.readthedocs.io/en/latest/index.html> (visitado 07-08-2020).
- [56] L. Wu, S. Wu, Y. Zhou, R. Li, Z. Wang, X. Luo, C. Wang y K. Ren, “EthScope: A Transaction-centric Security Analytics Framework to Detect Malicious Smart Contracts on Ethereum,” *arXiv preprint arXiv:2005.08278*, pág. 3, 2020.
- [57] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” págs. 25, 28-36, jun. de 2020. dirección: <https://ethereum.github.io/yellowpaper/paper.pdf> (visitado 07-08-2020).
- [58] *Patrón de diseño de Solidity: Eternal Storage*, ene. de 2019. dirección: [https://fravoll.github.io/solidity-patterns/eternal\\_storage.html](https://fravoll.github.io/solidity-patterns/eternal_storage.html) (visitado 07-08-2020).
- [59] *Patrón de diseño de Solidity: Emergency Stop*, abr. de 2018. dirección: [https://fravoll.github.io/solidity-patterns/emergency\\_stop.html](https://fravoll.github.io/solidity-patterns/emergency_stop.html) (visitado 07-08-2020).
- [60] M. Masse, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. .ºReilly Media, Inc.", 2011, págs. 11-34, 47-48, 68-70.
- [61] *Documentación oficial de Express.* dirección: <https://expressjs.com/> (visitado 07-08-2020).
- [62] *Documentación oficial de Web3.js.* dirección: <https://web3js.readthedocs.io/en/v1.2.11/index.html> (visitado 07-08-2020).
- [63] *Página web w3schools, apartado Learn HTML, Learn CSS y Learn JavaScript.* dirección: <https://www.w3schools.com/> (visitado 07-08-2020).
- [64] *Documentación oficial de OpenZeppelin.* dirección: <https://docs.openzeppelin.com/openzeppelin/> (visitado 07-08-2020).