Trabajo de Fin de Grado

Grado en Ingeniería de las Tecnologías de Telecomunicación

Mecanismo de registro de transacciones de información sanitaria basado en blockchain

Autora: Marta García Benzal

Tutor: Jorge Calvillo Arbizu

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2023

Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Mecanismo de registro de transacciones de información sanitaria basado en blockchain

Autora:
Marta García Benzal

Tutor:
Jorge Calvillo Arbizu
Profesor Ayudante Doctor

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2023

Trabajo de Fin de Grado: Mecanismo de registro de transacciones de información sanitaria basado en blockchain

Autora: Marta García Benzal

Tutor: Jorge Calvillo Arbizu

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2023

El Secretario del Tribunal

Agradecimientos

Con este proyecto pongo fin a mis estudios en Ingeniería de las Tecnologías de Telecomunicación. La consecución de este título no habría sido posible sin las personas que me han acompañado en esta etapa. Por ello, quiero aprovechar estas líneas para agradecer su apoyo.

En primer lugar, quiero agradecer a Jorge Calvillo, tutor de este trabajo, por la asignación del tema y por la atención rápida, eficaz y cercana durante estos meses de trabajo. Gracias también al resto de docentes del grado por todos los conocimientos transmitidos. A día de hoy ya estoy comprobando que todo lo aprendido en estos años va a serme de gran utilidad y me va a abrir muchas puertas en el futuro.

En segundo lugar, quiero agradecer a Pablo, Rocío, María, Isa, Itxaso, Gabriel, Ana, Paula, Marta y Miguel por apoyarme, animarme y escucharme cada vez que lo he necesitado. En Huelva, Sevilla, Francia o cualquier lugar: estar con vosotros es sentirse en casa.

Gracias también a Fernando, Celia, Ángel, Rubio, Juanjo, Manolo, Jacinto, Julio, Carlos y Alejandra. Gracias, Pablo, por descubrirme nuevas amistades cuando más lo necesitaba. Lo mejor que me llevo del grado sois vosotros y todos los momentos vividos que llevaré conmigo para siempre.

Gracias, Vicente, por cuidarme y creer en mí. Por hacer siempre todo lo posible por verme bien en los malos momentos y por hacer aún mejores los buenos. Eres un verdadero tesoro y yo tengo la suerte de haberte encontrado en este camino.

Por último, gracias, papá, por ayudarme y apoyarme, por ser un gran ejemplo para mí, por enseñarme lo que es ser ingeniero y por tener siempre algo nuevo que enseñarme. Gracias, David, por aconsejarme y escucharme siempre. Ojalá logres todas las metas que te propongas, que yo estaré siempre cerca tuya para celebrarlo. Gracias, mamá, por ser mi mayor apoyo en estos años. Por celebrar conmigo cada triunfo y llorar cada mal momento, por escuchar día tras día mi rutina sin importar la monotonía de algunas etapas. Gracias también a mis tíos, abuelos y primos, por seguir de cerca mi evolución en el grado y alegraros de cada pasito que he dado. No sabéis lo afortunada que me siento de tener una familia como vosotros.

Lo mejor de cumplir este gran objetivo es poder compartirlo con vosotros. Os quiero con todo mi corazón.

Marta García Benzal

Sevilla, 2023

Resumen

En el sector sanitario, los datos son especialmente valiosos. El volumen de datos generados en este sector ha crecido exponencialmente en los últimos años, impulsado por la adopción de tecnologías digitales, la creciente demanda de atención médica y la necesidad de analizar grandes cantidades de información para mejorar la calidad de la atención.

Compartir datos sanitarios es una tarea compleja debido a la naturaleza altamente sensible de la información que se maneja. Además, la complejidad del sector de la salud hace que sea difícil intercambiar datos entre diferentes organizaciones, ya que estos datos se almacenan en diferentes formatos y sistemas de información, lo que dificulta el intercambio entre diferentes organizaciones y proveedores de atención médica.

A pesar de estas dificultades, este intercambio es fundamental para la investigación y la mejora de la calidad de la atención médica. Por ello, en este proyecto se buscará desarrollar una solución que acompañe a esta distribución de datos. Este mecanismo deberá responder a los principios de seguridad, trazabilidad, descentralización e interoperabilidad para solventar las dificultades expuestas.

Con este fin, habrá dos pilares fundamentales en el proyecto: blockchain y FHIR. La primera es una tecnología que permitirá desarrollar y desplegar una red para registrar intercambios de información entre entidades sanitarias, y cubrirá los principios de seguridad, trazabilidad y descentralización. La segunda se trata de un estándar diseñado para facilitar la interoperabilidad entre diferentes sistemas de información sanitaria, por lo que dará cobertura al cuarto principio planteado.

En resumen, en el presente trabajo se abordarán los siguientes aspectos:

- Desarrollo de una red blockchain para implementar el mecanismo de registro de transacciones.
- Uso del estándar FHIR para definir el formato y estructura de las transacciones.
- Desarrollo de una aplicación formada por una API REST y una interfaz gráfica de usuario para facilitar la interacción con la red.

Abstract

In the healthcare field, data is particularly valuable. The volume of data generated in this sector has grown exponentially in recent years, powered by digital transformation, increasing demand for medical care and the need to analyse large amounts of information in order to improve healthcare quality.

Sharing health-related data is a challenging task due to the extremely sensitive nature of the information being handled. Additionally, the complexity of the healthcare sector hinders to exchange data between different organizations, owing to the storage of this data in diverse formats and information systems.

Despite these difficulties, data sharing is essential for research and improving medical care quality. Therefore, this project aims to develop a solution that facilitates this data distribution while adhering to principles of security, traceability, decentralization, and interoperability to overcome the mentioned adversities.

In this regard, this project will rely on two main pillars: blockchain and FHIR. The first technology will allow to develop and deploy a network that registers information exchanges between medical companies, covering the principles of security, traceability, and decentralization. The second one is a standard designed to provide interoperability among different healthcare information systems, hence it will cover the fourth suggested principle.

In a nutshell, this project will focus on the following aspects:

- Development of a blockchain network to implement the transaction log mechanism.
- Use of FHIR standard to define the format and structure of the transactions.
- Development of an application comprising a REST API and a graphical user interface to enhance the interaction with the network.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

ACK	Acknowledgement
API	Application Programming Interface
CLI	Command Line Interface
CSR	Solicitud de Firma de Certificado
EHR	Electrinic Health Record
FHIR	Fast Healthcare Interoperability Resource
JSON	JavaScript Object Notation
JWT	JSON Web Token
MSP	Membership Service Provider
PoS	Proof of Stake
PoW	Proof of Work
RGPD	Reglamento General de Protección de Datos
SDK	Software Development Kit
SGBD	Sistema Gestor de Bases de Datos
SNS	Sistema Nacional de Salud
UE	Unión Europea

1 INTRODUCCIÓN

En este capítulo se presentarán los motivos que han llevado a la realización de este proyecto, cuyo resultado será el desarrollo de una red blockchain para dar soporte al registro de transacciones de información sanitaria entre organizaciones. Además, se expondrán los objetivos que se pretenden lograr con la solución adoptada, así como la metodología y el plan de trabajo seguidos en el desarrollo del proyecto.

1.1. Motivación y objetivos

El desarrollo incesante de la tecnología en todos los ámbitos de nuestras vidas, incluido el de la salud, ha permitido que las organizaciones sanitarias abandonen el almacenamiento de datos sobre historiales médicos, diagnósticos, resultados de pruebas, imágenes médicas y tratamientos de forma impresa, a favor de un registro digital, en lo que se conoce como EHR (registro electrónico de salud). Esta digitalización, que surgió hace ya décadas, supone cada vez más una ventaja, debido a la creciente cantidad de datos de pacientes que necesitan ser almacenados [1]. A menudo, parte de estos datos son compartidos con aseguradoras médicas, laboratorios y otras entidades cuando participan en la asistencia de un determinado paciente.

Según recoge el Reglamento General de Protección de Datos (RGPD), en uno de los 7 derechos de los que disponen los ciudadanos de la UE, el derecho a la portabilidad de los datos consiste en otorgar a cualquier ciudadano europeo el derecho a consentir que cualquier empresa que trate sus datos personales de forma autorizada se los ceda o los transfiera a cualquier otra empresa que este les indique en un formato estructurado, inteligible, e interoperable [2]. La portabilidad de los datos permite que los usuarios autoricen el intercambio, de manera limitada y controlada, de sus datos personales entre empresas u organizaciones. De esta forma, se enriquecen las experiencias y los servicios de los consumidores.

Sin embargo, en el ámbito sanitario, esta portabilidad o cesión no está soportada entre organizaciones en la mayoría de las situaciones, sobre todo cuando conviven organizaciones públicas y privadas [3].

Uno de los principales motivos de esta causa es que los datos relacionados con la salud e historias clínicas son considerados altamente sensibles, por lo que su protección es de obligado cumplimiento para cualquier entidad y su compartición debe realizarse de forma segura.

Este proyecto plantea una solución a la problemática expuesta, poniendo el foco en el paciente para mejorar la continuidad asistencial de este, a la vez que garantiza la privacidad de sus datos sanitarios. La motivación del proyecto gira en torno al desarrollo de un mecanismo que registre flujos de información sanitaria entre distintas organizaciones.

Además de dar soporte al derecho de portabilidad, esta solución permitiría mejorar y agilizar los diagnósticos y tratamientos, reducir costes innecesarios por pruebas duplicadas e impulsar la investigación médica al poder compartir los datos con fines de estudio y educación, siempre bajo el consentimiento del paciente.

El mecanismo desarrollado debe cumplir diversos requisitos:

- Ha de ser seguro, trazable y garantizar el no repudio.
- Debe acogerse a las regulaciones de protección de datos en el ámbito sanitario.
- Debe ser interoperable y distribuido, dado que la información será compartida entre organizaciones independientes.

Por todo ello, se ha escogido desarrollar esta solución basándose en las redes blockchain. A modo de resumen, se trata de una tecnología que está cobrando gran importancia en la actualidad por sus características de seguridad, transparencia, descentralización y consenso entre los participantes [4].

El objetivo principal del proyecto es desarrollar el mecanismo planteado anteriormente, de forma que se registre qué organización es la emisora de los datos, qué organización los recibe, con qué fin se comparten, durante

cuánto tiempo pueden compartirse y a qué paciente afectan. Todo ello utilizando la tecnología blockchain.

Además, se proponen otros objetivos secundarios, tales como:

- Montaje de un escenario de prueba en el que la red se despliegue en varios equipos.
- Desarrollo de una API REST para acceder a la red.
- Desarrollo de una interfaz gráfica de usuario para interactuar con la red.

Para cumplir estos objetivos, deben satisfacerse los siguientes:

- Estudio de la tecnología Hyperledger Fabric y despliegue de las soluciones de prueba.
- Análisis del estándar FHIR para definir la estructura de las transacciones.
- Análisis y elección de los lenguajes de programación y entornos de ejecución disponibles y estudio de su uso y funcionamiento.
- Pruebas de validación de las distintas partes y de la solución al completo.

La estructura del mecanismo que será desarrollado se ilustra de forma básica en la *Figura 1-1. Diagrama de estructura básica de la solución*. Se desarrollará una red blockchain en la que participarán varias organizaciones. Cada organización dispone de los datos de sus pacientes almacenados en registros EHR y de una copia de la cadena de bloques. El funcionamiento básico es el siguiente:

1) Un paciente firma un consentimiento en el que autoriza a la organización 1, la cual posee sus datos sanitarios, a ceder parte de (o todos) ellos a la organización 2.
2) La organización 1 solicita compartir dichos datos con la organización 2.
3) Todas las organizaciones que participan de la red consensúan la validez o no de la solicitud.
4) En caso de ser validada, estas organizaciones registran una transacción en la que se incluyen, entre otros, los datos mostrados en la figura.
5) Cada organización añade a su cadena de bloques la transacción realizada.

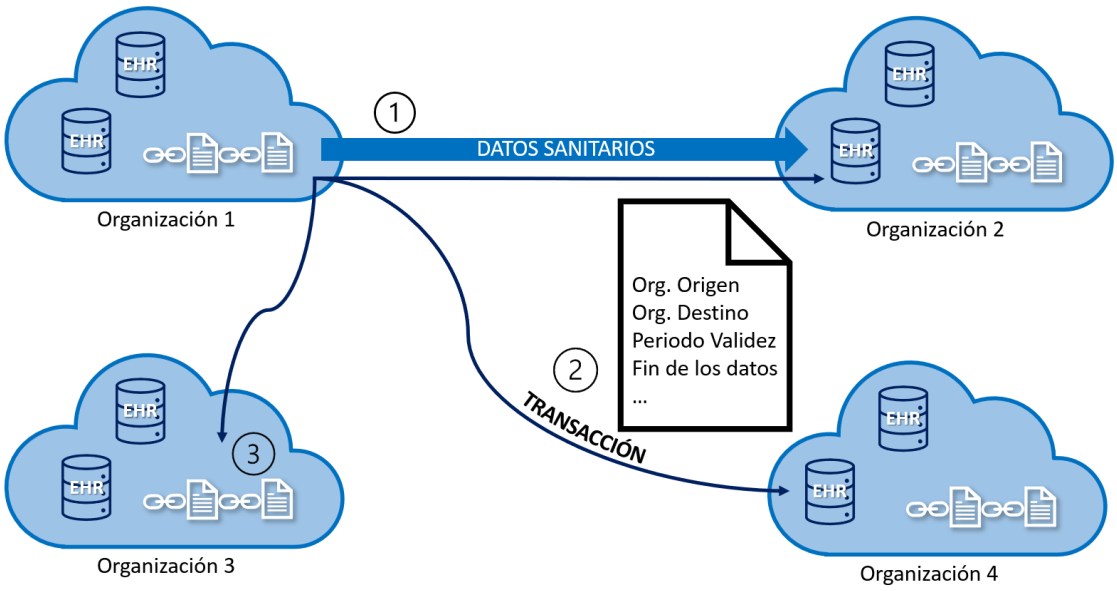

Figura 1-1. Diagrama de estructura básica de la solución

1.2. Metodología

El desarrollo ágil de software es una metodología para la construcción de software basada en ciclos de desarrollo iterativos e incrementales, donde los requisitos y las soluciones van evolucionando según las necesidades del proyecto.

La metodología Scrum sigue el modelo de desarrollo ágil de software que ayuda a los equipos a desarrollar productos en periodos cortos, permitiendo obtener de forma rápida una realimentación por parte del cliente, permitiendo adaptaciones y una mejora continuada [5].

Para el desarrollo de este proyecto, se ha seguido una metodología basada en Scrum, con ciertas variaciones, al tratarse de un proyecto personal en el que no hay un cliente real.

Se ha tomado la unidad de trabajo fundamental en Scrum, las iteraciones. Además, cada iteración se ha basado en las distintas fases establecidas por la metodología, que son:

- Plan
- Análisis
- Diseño
- Construcción
- Prueba
- Despliegue

Un esquema con el funcionamiento de la metodología se muestra en la *Figura 1-2. Funcionamiento metodología de desarrollo ágil.*

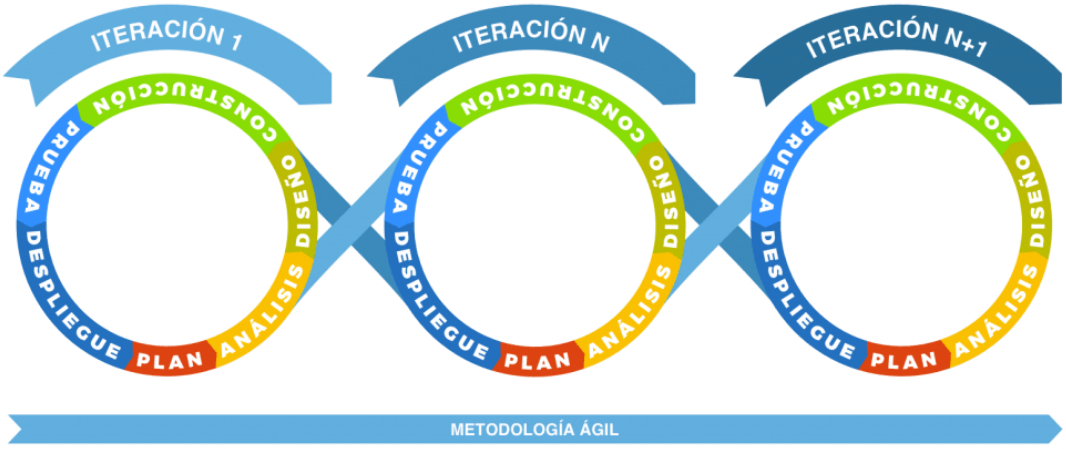

Figura 1-2. Funcionamiento metodología de desarrollo ágil

En este proyecto, se ha diseñado una solución básica para cumplir los objetivos marcados anteriormente. A medida que se ha evolucionado en la primera iteración, han surgido otras funcionalidades o características que se han considerado fundamentales o útiles para lograr un resultado lo suficientemente completo. Estas nuevas necesidades se han añadido en un "backlog" o lista de tareas y se han desarrollado en iteraciones posteriores, dando lugar al proyecto completo al finalizar la última iteración.

Las distintas iteraciones y etapas de trabajo que se han desarrollado se exponen en el apartado siguiente.

1.3. Plan de trabajo

A continuación, se listan las distintas tareas que se han realizado a lo largo del proyecto, agrupadas en las iteraciones en las que se han llevado a cabo. Cabe destacar que, al tratarse de un proyecto en el que ha sido mi primer contacto con la mayoría de las tecnologías y herramientas empleadas, se ha requerido una documentación sobre estas en la fase de planteamiento y análisis, al principio de cada iteración.

Primera iteración – red básica y chaincode

- Documentación sobre redes blockchain en general y sobre la herramienta Hyperledger Fabric en particular.
- Documentación sobre Docker y Docker compose.
- Diseño de la red blockchain básica.
- Configuración de la red basada en la red de prueba proporcionada por Hyperledger Fabric.
- Pruebas del chaincode de prueba proporcionado por Hyperledger Fabric, usando la interfaz de línea de comandos propia de la herramienta.
- Diseño del chaincode propio.
- Desarrollo y pruebas del chaincode propio.

Segunda iteración – API REST

- Documentación sobre el funcionamiento básico de Node.js para construir una API REST ofrecida a través de un servidor HTTPS.
- Diseño y desarrollo de la API REST.
- Generación de certificado autofirmado con openSSL.
- Implementación del certificado en la API.
- Pruebas de la API en Postman.

Tercera iteración – interfaz de usuario web

- Documentación sobre el funcionamiento de Reactjs para desarrollo de interfaz web sencilla.
- Diseño y desarrollo de la interfaz básica.
- Pruebas de la interfaz básica.

Cuarta iteración – mejoras en la configuración de red y en el chaincode

- Adición de funcionalidad de búsqueda en el chaincode inicial (se pasó de 4 a 5 opciones de búsqueda).
- Adaptación en la interfaz web para soportar la nueva funcionalidad.
- Mejoras en la API y en la interfaz.
- Adición de 2 nodos en el servicio de ordenamiento para comprobar el funcionamiento del algoritmo de consenso.

Quinta iteración – despliegue distribuido usando Docker Swarm

- Documentación sobre configuración y funcionamiento de Docker Swarm.
- Diseño de la red distribuida y preparación del entorno (2 máquinas virtuales adicionales).
- Adaptación de la configuración de la red para el despliegue en varios equipos.
- Despliegue final y realización de pruebas en los distintos equipos, a través de la interfaz web.

A continuación, se añade una tabla con la estimación horaria para cada grupo de tareas descrito, acompañada de la duración real del desempeño de estas tareas.

	TAREA	DURACIÓN ESTIMADA (horas)	DURACIÓN REAL (horas)	
Primera iteración	Documentación y diseño de la red	40 h	45 h	Duración de la iteración: **124 h**
	Configuración de la red y despliegue	40 h	60 h	
	Pruebas del chaincode de prueba	5 h	5 h	
	Diseño y desarrollo del chaincode propio	15 h	12 h	
	Pruebas y correcciones del chaincode	3 h	2 h	
Segunda iteración	Documentación	20 h	20 h	Duración de la iteración: **72 h**
	Diseño y desarrollo de la API	35 h	38 h	
	Generación e implementación del certificado	2 h	2 h	
	Pruebas y correcciones de la API	10 h	12 h	
Tercera iteración	Documentación	15 h	15 h	Duración de la iteración: **62 h**
	Diseño de la interfaz	12 h	10 h	
	Desarrollo de la interfaz	20 h	22 h	
	Pruebas y correcciones de la interfaz	10 h	15 h	

Tabla 1-1. Plan de trabajo I

	TAREA	DURACIÓN ESTIMADA (horas)	DURACIÓN REAL (horas)	
Cuarta iteración	Adición de nuevo filtro de búsqueda en chaincode	2 h	2 h	Duración de la iteración: **30 h**
	Adaptación de la interfaz para la nueva funcionalidad	2 h	3 h	
	Mejoras en la API y en la interfaz	10 h	15 h	
	Adición de dos nodos en el servicio de ordenamiento	10 h	10 h	
Quinta iteración	Documentación	20 h	18 h	Duración de la iteración: **92 h**
	Diseño de la red distribuida	5 h	7 h	
	Preparación del entorno	4 h	4 h	
	Adaptación de la configuración de la red	15 h	20 h	
	Despliegue final	15 h	18 h	
	Pruebas finales	20 h	25 h	
DURACIÓN TOTAL		**330 h**	**380 h**	

Tabla 1-2. Plan de trabajo II

2 ESTADO DEL ARTE

En este capítulo se definirá, desde un punto de vista teórico, el punto de partida del proyecto. Se abordarán las cuestiones más relevantes sobre las redes blockchain, el estándar FHIR y el RGPD, que constituyen la base del trabajo. Además, se explicarán las distintas plataformas y herramientas utilizadas para el diseño y desarrollo de la solución.

2.1. Marco teórico

2.1.1. Redes blockchain

2.1.1.1. ¿Qué es la tecnología blockchain?

Blockchain es una tecnología que permite crear aplicaciones en las que varias partes pueden registrar transacciones directamente, sin la necesidad de una autoridad central de confianza para garantizar que se verifiquen las transacciones. Esto se consigue usando una red peer-to-peer donde cada participante en la red tiene acceso a un libro mayor compartido en el que se registran las transacciones. Estas transacciones son, por diseño, inmutables y verificables criptográficamente. A alto nivel, la tecnología blockchain consta de tres componentes: un libro mayor distribuido (en adelante, *ledger*, por el frecuente uso del término en inglés), un algoritmo de consenso y contratos inteligentes (en adelante, *smartcontract*, por el frecuente uso del término en inglés) [6].

- Una ledger es un registro transaccional que mantiene todo el historial de cambios de datos. Las ledgers son inmutables y de solo escritura (no borrado) por diseño, y las transacciones son verificables de forma independiente por cada miembro de una red. Cada uno de estos miembros mantiene una copia de la ledger.
- Un algoritmo de consenso es un mecanismo para garantizar que todos los miembros de la red puedan ponerse de acuerdo respecto a una fuente única de verdad. En blockchain, se aplican algoritmos de consenso para la ejecución de código y registro de transacciones [7].
- Los smartcontracts definen reglas entre los participantes de la red en forma de código ejecutable. Es el mecanismo que permite generar transacciones y registrarlas en la ledger.

2.1.1.2. Características y ventajas

Algunas de las características de blockchain son las siguientes [4]:

- **Inmutabilidad**: Una vez añadida una transacción, nadie puede eliminarla o modificarla. Además, todos los bloques en el registro tienen un hash único y contienen el hash del bloque anterior. Modificar todos los bloques en todos los nodos de la red es prácticamente imposible.
- **Descentralización**: No existe una sola autoridad que gobierne la red o que ejerza un control total sobre esta.
- **Altamente segura**: Los registros cuentan con un cifrado extremo a extremo, lo que ayuda a prevenir el fraude y la actividad no autorizada. Además, el almacenamiento de la información de forma descentralizada también dificulta que la información sea visible por personas no deseadas.
- **Registros distribuidos**: los datos se registran de manera idéntica en múltiples ubicaciones. Todos los participantes de la red ven la misma información al mismo tiempo, lo que brinda total transparencia.

- **Consenso**: Para agregar una transacción a la ledger, cada nodo debe verificar su validez. Si no se cumplen los requisitos de consenso, la transacción u operación se considera inválida.
- **Eficiencia y velocidad**: La automatización proporcionada por los smartcontracts reduce la intervención humana y la dependencia de terceros para la ejecución de procesos, haciéndola más rápida.

2.1.1.3. Redes blockchain permisionadas

Existen, principalmente, tres tipos de redes blockchain [8]:

- **Redes públicas**: son aquellas en las que cualquier persona tiene la libertad de unirse a la red, visualizar e incluso realizar y validar transacciones. En estas redes, se suele proporcionar un incentivo a los usuarios para que participen en el consenso resolviendo en sus equipos puzles matemáticos o criptográficos (lo que se conoce como minería en el ámbito de blockchain). Este tipo de algoritmos de consenso se denomina *Proof Of Work* (PoW).

- **Redes privadas**: son aquellas en las que los permisos de participación en la red están restringidos, normalmente a una única organización.

- **Redes permisionadas**: pueden definirse como redes privadas para un grupo de organizaciones o entidades. La participación en la red está restringida a los miembros de estas organizaciones. El algoritmo de consenso es mantenido por una serie de nodos que han sido preseleccionados y que cuentan con la confianza de manera previa. Este tipo de algoritmos se denomina *Proof of Stake* (PoS).

En este proyecto, esta tecnología será la utilizada en la red desarrollada para dar soporte al mecanismo de registro de transacciones planteado. Sus características de inmutabilidad, descentralización, seguridad y consenso hacen que sea una solución altamente adecuada, debido principalmente a dos motivos:

- Los datos sanitarios son considerados altamente sensibles, por lo que es imprescindible proteger estos datos y conocer quién los posee, sin que esa información pueda ser alterada.
- Dado que se pretende desarrollar una red en la que participen entidades independientes, es necesario que todas actúen en concordancia y tengan el mismo nivel de autoridad.

Además, puesto que en esta red participará únicamente un grupo de organizaciones sanitarias, se ha optado por desarrollar una red blockchain permisionada.

2.1.2. FHIR

FHIR (*Fast Healthcare Interoperability Resource*) es un estándar de interoperabilidad desarrollado por HL7 (organización de desarrollo de estándares de atención médica), diseñado para permitir el intercambio electrónico de datos de atención médica entre diferentes sistemas. El objetivo principal de FHIR es abordar las crecientes necesidades de digitalización en la industria de la salud y simplificar el intercambio de datos sin comprometer la integridad de la información. FHIR tiene como objetivo hacer que los registros de salud electrónicos (EHR) estén disponibles, sean detectables y fácilmente comprensibles para las partes interesadas a medida que los pacientes se mueven dentro del ecosistema de atención médica [9].

2.1.2.1. Recursos

FHIR parte del concepto fundamental de recursos. Un recurso es la unidad básica de interoperabilidad. Son representaciones de conceptos del mundo sanitario: paciente, médico, problema de salud, etcétera [10].

Los recursos tienen una serie de características comunes:

- Un pequeño conjunto de propiedades principales que la gran mayoría de los sistemas soportan actualmente.

- Un mecanismo de extensión que permite a los implementadores añadir nuevas propiedades de manera sencilla.
- Una identificación a través de la cual puede ser registrado, localizado y recuperado.
- Un componente (elementos narrativos) que permiten una visión legible de los datos almacenados en el recurso.

Estos pueden utilizarse en su forma más simple o agruparse en forma de mensajes (asemejando los recursos a los segmentos de los mensajes), documentos (como una colección de recursos agrupados) o incluso servicios (empleando uno o más recursos).

Los recursos FHIR se agrupan en módulos para organizar y guiar la configuración. A su vez, estos módulos se dividen en niveles para mostrar en qué ámbito debe usarse cada módulo [11].

- Nivel 1 – módulo básico: marco de referencia sobre el que se desarrollan el resto de los módulos. Este módulo define las documentaciones base.

- Nivel 2 – implementación y configuración: incluye los módulos para soportar la implementación y vinculación a recursos externos.

- Nivel 3 – administración: describe los datos básicos para realizar un seguimiento de los pacientes.

- Nivel 4 – mantenimiento de registros e intercambio de datos: incluye módulos de recursos relacionados con patologías, diagnósticos, medicación, flujo de trabajo y finanzas.

- Nivel 5 – razonamiento clínico: proporciona el razonamiento para afrontar el proceso asistencial.

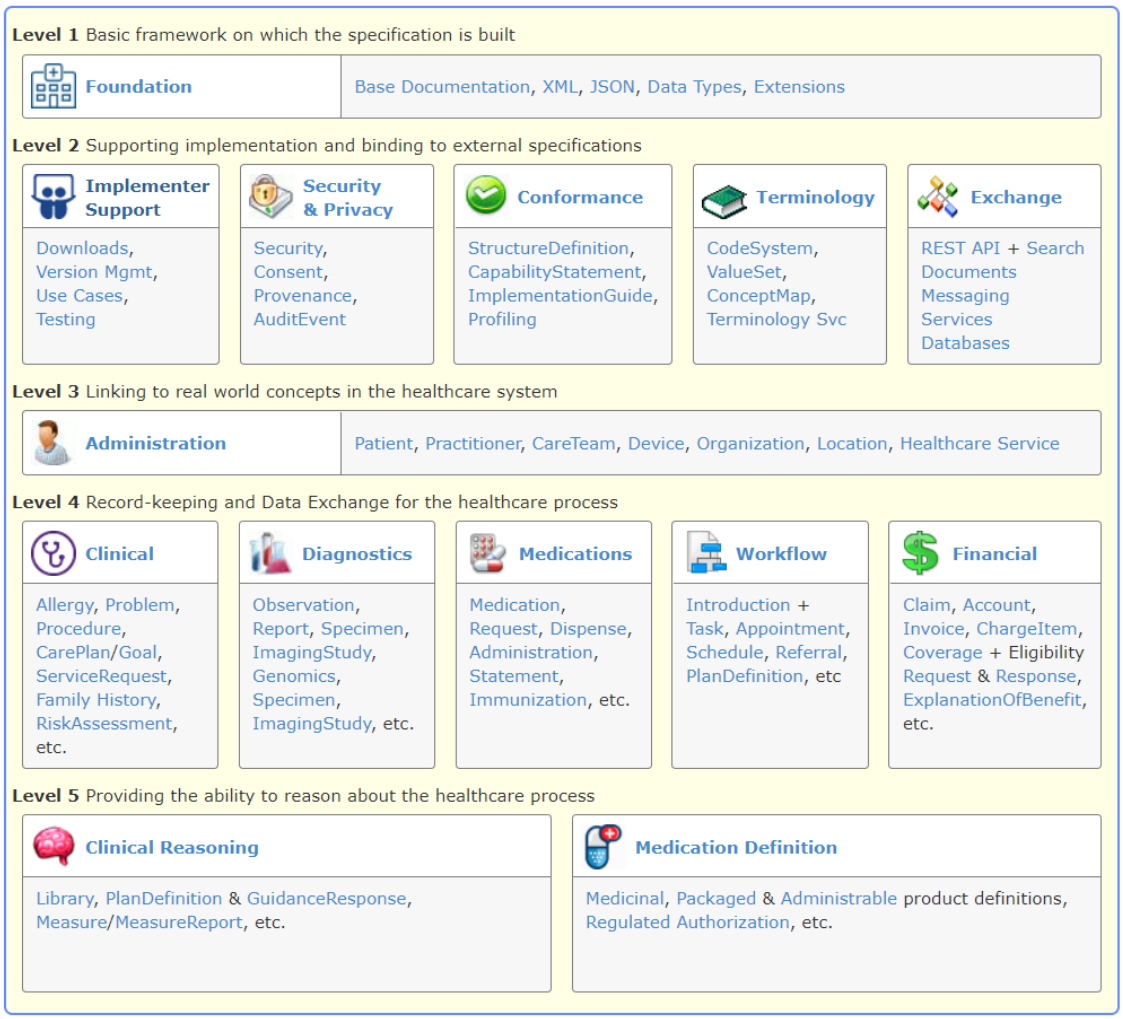

Figura 2-1. Niveles de recursos FHIR

En este proyecto, el recurso principal que se usará será AuditEvent.

El recurso AuditEvent tiene como objetivo facilitar el registro de eventos auditables o que necesiten ser gestionados. Por ello, es el recurso adecuado para representar los eventos de intercambio de información sanitaria entre organizaciones que se quiere registrar en este proyecto [12].

A continuación, se muestra la estructura del recurso:

Structure

Name	Flags	Card.	Type	Description & Constraints
AuditEvent	TU		DomainResource	Event record kept for security purposes Elements defined in Ancestors: id, meta, implicitRules, language, text, contained, extension, modifierExtension
type	Σ	1..1	Coding	Type/identifier of event Audit Event ID (Extensible)
subtype	Σ	0..*	Coding	More specific type/id for the event Audit Event Sub-Type (Extensible)
action	Σ	0..1	code	Type of action performed during the event AuditEventAction (Required)
period		0..1	Period	When the activity occurred
recorded	Σ	1..1	instant	Time when the event was recorded
outcome	Σ	0..1	code	Whether the event succeeded or failed AuditEventOutcome (Required)
outcomeDesc	Σ	0..1	string	Description of the event outcome
purposeOfEvent	Σ	0..*	CodeableConcept	The purposeOfUse of the event PurposeOfUse (Extensible)
agent		1..*	BackboneElement	Actor involved in the event
type		0..1	CodeableConcept	How agent participated ParticipationRoleType (Extensible)
role		0..*	CodeableConcept	Agent role in the event SecurityRoleType (Example)
who	Σ	0..1	Reference(PractitionerRole | Practitioner | Organization | Device | Patient | RelatedPerson)	Identifier of who
altId		0..1	string	Alternative User identity
name		0..1	string	Human friendly name for the agent
requestor	Σ	1..1	boolean	Whether user is initiator
location		0..1	Reference(Location)	Where
policy		0..*	uri	Policy that authorized event
media		0..1	Coding	Type of media Media Type Code (Extensible)
network		0..1	BackboneElement	Logical network location for application activity
address		0..1	string	Identifier for the network access point of the user device
type		0..1	code	The type of network access point AuditEventAgentNetworkType (Required)
purposeOfUse		0..*	CodeableConcept	Reason given for this user PurposeOfUse (Extensible)
source		1..1	BackboneElement	Audit Event Reporter
site		0..1	string	Logical source location within the enterprise
observer	Σ	1..1	Reference(PractitionerRole | Practitioner | Organization | Device | Patient | RelatedPerson)	The identity of source detecting the event
type		0..*	Coding	The type of source where event originated Audit Event Source Type (Extensible)
entity	I	0..*	BackboneElement	Data or objects used *+ Rule: Either a name or a query (NOT both)*
what	Σ	0..1	Reference(Any)	Specific instance of resource
type		0..1	Coding	Type of entity involved AuditEventEntityType (Extensible)
role		0..1	Coding	What role the entity played AuditEventEntityRole (Extensible)
lifecycle		0..1	Coding	Life-cycle stage for the entity ObjectLifecycleEvents (Extensible)
securityLabel		0..*	Coding	Security labels on the entity SecurityLabels (Extensible)
name	Σ I	0..1	string	Descriptor for entity
description		0..1	string	Descriptive text
query	Σ I	0..1	base64Binary	Query parameters
detail		0..*	BackboneElement	Additional Information about the entity
type		1..1	string	Name of the property
value[x]		1..1		Property value
valueString			string	
valueBase64Binary			base64Binary	

Figura 2-2. Estructura del recurso AuditEvent

Para trabajar con los recursos, FHIR ofrece varios formatos. En este caso, se van a usar en formato JSON. Además, no todos los campos resultan de utilidad para este proyecto. A continuación, se muestra el formato JSON del recurso con los campos que serán utilizados:

```
{
  “resourceType” : “AuditEvent”,

  “type” : { Coding }, // Tipo del evento. En este caso, tendrá siempre el valor “receive” [13]
                          (se registra un evento cada vez que una organización reciba datos).

  “action” : “<code>”, // Tipo de la acción. En este caso, será siempre “E” [14] (execute, el
                          evento se produce al ejecutar una transacción en el sistema).

  “recorded” : “<instant>”, // Instante en el que se produce el evento (YYYY-MM-DD hh:mm:ss) [15].
```

```
  "purposeOfEvent" : [{ CodeableConcept }], // Propósito de uso de los datos. Podrá tomar
                                              los valores "HRESCH" (Healthcare Research, para
                                              fines de investigación), "TRAIN" (para fines
                                              educativos) o "TREAT" (para prestar servicios
                                              de tratamiento médico) [16].

  "agent" : [{ // Actor involucrado en el evento.

    "type" : { CodeableConcept }, // Tipo del actor. Será siempre "PROV" (proveedor de
                                     servicios médicos) [17].

    "role" : [{ CodeableConcept }], // Rol del actor en el evento. Podrá tomar los valores
                                       "AUT" [18] (autor, origen) o "IRCP" [19] (receptor de
                                       información).

    "who" : { Reference(Device|Organization|Patient|Practitioner|

    PractitionerRole|RelatedPerson) }, // Referencia a recurso Organization para
                                          especificar la organización origen o destino.
  }],


  "source" : { // Fuente que registra el evento. En este caso, será la organización
                  origen.

    "observer" : { Reference(Device|Organization|Patient|Practitioner|

    PractitionerRole|RelatedPerson) }, // Referencia a recurso Organization.

  },


  "entity" : [{ // Datos adicionales.

    "what" : { Reference(Any) }, // Referencia a recurso Patient para especificar el
                                    paciente al que pertenecen los datos.

    "detail" : [{ // Propiedades adicionales.

      "type" : "<string>", // Nombre de la propiedad. En este caso, se registrará aquí la
                              fecha en la que termina el periodo de validez durante el cual
                              la organización destino puede hacer uso de los datos. Tomará
                              el valor "end of validity period".

      "value" : "<string>" // Fecha en formato YYYY-MM-DD hh:mm:ss.

    }]

  }]
}
```

Figura 2-3. Formato JSON recurso AuditEvent

En el recurso anterior pueden observarse referencias a dos recursos:

Recurso Organization [20]: se registrará el identificador de la organización y su nombre. Se podrá tener los pares 01 – SAS, 02 – Quirón, 03 – HLA y 04 – Viamed.

```
{
  “resourceType” : “Organization”,
  “identifier” : [{ Identifier }],
  “name” : “<string>”
}
```

Figura 2-4. Formato JSON recurso Organization

Recurso Patient [21]: se registrar el identificador (DNI) del paciente.

```
{
  “resourceType” : “Patient”,
  “identifier” : [{ Identifier }]
}
```

Figura 2-5. Formato JSON recurso Patient

Los campos mostrados en las figuras anteriores serán los básicos de ambos tipos de recursos, pudiendo añadirles otros si se considerara necesario.

2.1.3. RGPD

El Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, denominado Reglamento general de protección de datos («RGPD»), regula el tratamiento que realizan personas, empresas u organizaciones de los datos personales relacionados con personas en la Unión Europea.

El RGPD define «datos personales» como “toda información sobre una persona física identificada o identificable; se considerará persona física identificable toda persona cuya identidad pueda determinarse, directa o indirectamente, en particular mediante un identificador, como por ejemplo un nombre, un número de identificación, datos de localización, un identificador en línea o uno o varios elementos propios de la identidad física, fisiológica, genética, psíquica, económica, cultural o social de dicha persona.” [22]

En concreto, los datos personales relativos a la salud son considerados como datos sensibles, especialmente protegidos, y, tras la publicación del RGPD, como una categoría especial de datos personales.

Además, entre los artículos 13 y 22 del RGDP se recogen un conjunto de derechos que cualquier ciudadano de la UE puede ejercer sobre sus datos personales [23]. Estos derechos son:

- Derecho de acceso.
- Derecho de rectificación.
- Derecho de supresión («olvido»).
- Derecho de limitación del tratamiento.
- Derecho a la portabilidad.
- Derecho de oposición.
- Derecho a no ser objeto de decisiones individuales automatizadas.

En concreto, resultan de especial interés para este proyecto los dos siguientes [24]:

- Derecho a la portabilidad

El derecho de portabilidad habilita al interesado para solicitar al responsable del tratamiento sus datos personales en un formato estructurado, de uso común, de lectura mecánica e interoperable, para poder trasladarlos a otro responsable, cuando medie el consentimiento del interesado o sea necesario para la ejecución de un contrato.

- Derecho de acceso

Todo interesado tiene derecho a conocer si sus datos personales están siendo tratados, para lo que puede ejercer el derecho de acceso, dirigiéndose al responsable del tratamiento para ello.

Si sus datos están siendo tratados, podrá solicitar la siguiente información sobre ellos:

- Copia de los datos objeto del tratamiento.
- Con qué finalidad se están tratando.
- Qué categorías de datos se están tratando.
- Si los datos se han cedido a terceros.
- El plazo previsto de conservación.
- El origen de los datos (cuando no se hayan obtenido directamente del interesado).
- Si se están usando en decisiones automatizadas o elaboración de perfiles.
- Si se han transferido fuera de la UE.

Tal y como se ha mencionado en el capítulo 1, este proyecto pretende desarrollar un mecanismo que acompañe a la transmisión de datos sanitarios entre distintas organizaciones, registrando información relevante relativa a estos flujos de información. Este mecanismo permitiría dar soporte al derecho de portabilidad.

La información que se va a registrar pretende acogerse al derecho de acceso, de forma que en cada transacción se almacenará:

- El paciente al que pertenecen los datos transmitidos.
- La organización origen y la organización destino.
- Propósito del tratamiento de los datos.
- Periodo de validez durante el cual la organización destino está autorizada a almacenar y tratar los datos.

2.2. Herramientas software, tecnologías y lenguajes

2.2.1. Hyperledger Fabric

2.2.1.1. ¿Qué es Hyperledger Fabric?

Hyperledger Fabric es una plataforma de tecnología de libro mayor distribuido (DLT) de código abierto, diseñada para contextos empresariales, que ofrece algunas capacidades de diferenciación clave sobre otras plataformas populares de blockchain [25]. Estas características son las siguientes:

- **Código abierto**: al ser una plataforma de código abierto, todos pueden usarla libremente. Además, Hyperledger Fabric pertenece al proyecto Hyperledger, establecido bajo la Fundación Linux, que tiene una larga historia de nutrir proyectos de código abierto bajo gobierno abierto.

- **Lenguajes de programación**: Fabric es la primera plataforma de blockchain que admite contratos inteligentes creados en lenguajes de programación de uso general como Java, Go y Node.js, en lugar de lenguajes específicos de dominio restringidos.
- **Permisionada**: a diferencia de una red pública sin permiso, los participantes se conocen entre sí. Esto significa que, si bien los participantes pueden no confiar completamente entre sí, una red puede operarse bajo un modelo de gobernanza que se basa en que la confianza existe entre los participantes.
- **Protocolos de consenso**: Uno de los diferenciadores más importantes de la plataforma es su compatibilidad con protocolos de consenso conectables que permiten que la plataforma se personalice de manera más eficaz para adaptarse a casos de uso particulares y modelos de confianza.
- **No requiere una criptomoneda**: Evitar una criptomoneda reduce algunos vectores de riesgo o de ataque significativos. Además, la ausencia de operaciones de minería criptográfica reduce los costes de implementación.
- **Modularidad**: Fabric tiene una arquitectura altamente modular y configurable, que permite la innovación, la versatilidad y la optimización para una amplia gama de casos de uso de la industria.

La combinación de estas características convierte a Fabric en una de las plataformas de mejor rendimiento disponibles en la actualidad tanto en términos de procesamiento de transacciones como de latencia de confirmación de transacciones.

Hyperledger Fabric constituirá la base fundamental del proyecto. Será la plataforma usada para construir la red blockchain sobre la que se desarrollará la solución anteriormente presentada.

Figura 2-6. Logo Hyperledger Fabric

2.2.1.2. Conceptos clave y elementos de la arquitectura

Peers

Los nodos pares, o *peers*, son el elemento fundamental de la red. Cada peer soporta una instancia de uno o varios smartcontracts y de una o varias ledgers.

Además, pueden ser peers básicos o clasificarse en la categoría de *endorsing peers* (aquellos que se encargan de validar transacciones) o *anchor peers* (aquellos con los que los peers de otras organizaciones pueden comunicarse para descubrir la red).

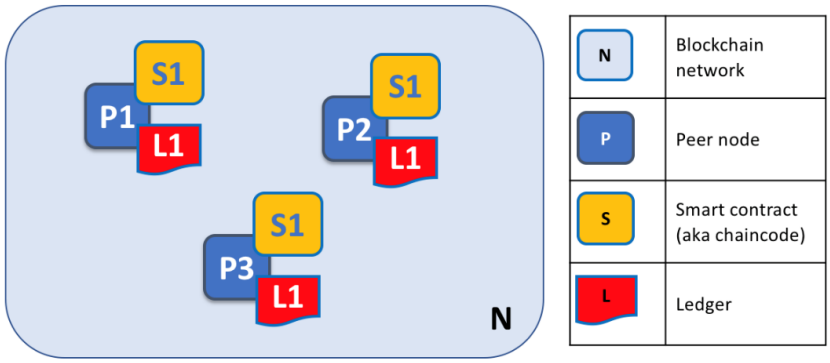

Figura 2-7. Esquema de peers, smartcontracts y ledgers

Servicio de ordenamiento

Está formado por otro tipo de nodos, los *orderers*. En las redes blockchain públicas, los peers realizan tanto la ejecución de smartcontracts como la ordenación de las transacciones en bloques. En hyperledger fabric, estas funciones se separan, de forma que existen nodos dedicados únicamente a ordenar las transacciones en bloques. Esto asegura que todos los peers de la red tengan una copia idéntica de la ledger. Además, esta separación mejora el rendimiento y la escalabilidad.

Assets

Los *assets* son las identidades que existen en la ledger. Son pares clave-valor.

Smartcontract o Chaincode

Chaincode es el nombre que reciben los smartcontracts en fabric. El chaincode es el mecanismo para interactuar con la ledger. Se trata de reglas escritas en código ejecutable que indican qué operaciones se pueden hacer en la ledger (añadir, leer, actualizar o borrar assets) y cómo deben hacerse.

Canal

Un canal es una subred privada de comunicación entre dos o más organizaciones dentro de una red. En cada red debe existir, como mínimo, un canal.

El canal está formado por varias organizaciones (cada una con sus propios peers regulares y con, al menos, un anchor peer), una ledger, un chaincode y un servicio de ordenamiento.

Membership Service Provider (MSP)

Al ser una red permisionada, los participantes de la red necesitan una forma de probar su identidad frente al resto de participantes para poder operar en la red. De esta gestión de identidades se encargan los Proveedores de Servicios de Membresía (Membership Service Provider, MSP).

Los MSP pueden registrar participantes almacenando información sobre ellos (qué rol tienen, qué privilegios…). Con esta información, se generan certificados, que actúan como la identidad de los participantes en la red.

Para crear estas identidades digitales, se usa Fabric CA. Fabric CA es un mecanismo de generación de Autoridades de Certificación (CA), usadas para crear el material criptográfico de los participantes de la red en todos sus roles: peer, orderer, admin (nodo encargado de tareas administrativas) o clientes (usuarios que pretenden interactuar con la red).

Organizaciones

En fabric, una organización está formada por el conjunto de nodos gestionados por un mismo MSP.

Un esquema de la arquitectura básica de una red de Hyperledger Fabric puede verse en la *Figura 2-8. Esquema de arquitectura básica de Hyperledger Fabric*. En este caso, la red está formada por dos organizaciones y una "organización" orderer, que sirve para agrupar y gestionar los nodos orderer, que proveerán el servicio de ordenamiento.

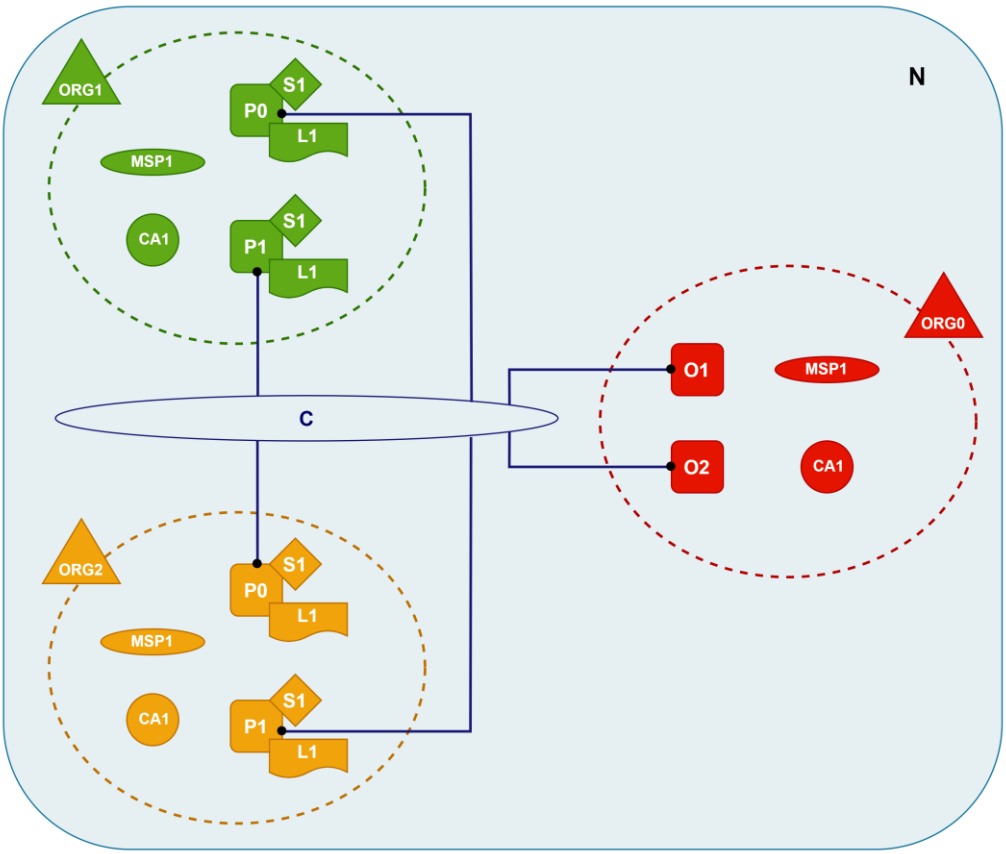

Figura 2-8. Esquema de arquitectura básica de Hyperledger Fabric

P	Nodo peer	ORG	Organización
S	Smartcontract	C	Canal
L	Ledger	C	Conexión con canal
CA	Autoridad de Certificación	O	Nodo orderer
MSP	Membership Services Provider		

Tabla 2-1. Leyenda de la Figura 2-8

Ledger

En Hyperledger Fabric, la ledger está formada por dos componentes.

El primer componente es el "estado mundial" o *world state*. Es una base de datos que almacena, en pares clave-valor, el estado actual de los bienes sobre los que se quiere registrar información (los llamados assets).

El segundo componente es la blockchain en sí. Se trata de un registro de transacciones que recoge todos los cambios que han resultado en el world state. Este componente es inmutable una vez escrito. De forma resumida, la blockchain está formada por bloques, que contienen:

- Una cabecera, que almacena el hash de las transacciones del bloque, así como el hash de la cabecera del bloque anterior.
- Una zona de datos, que contiene las transacciones del bloque.
- Una zona de metadatos.

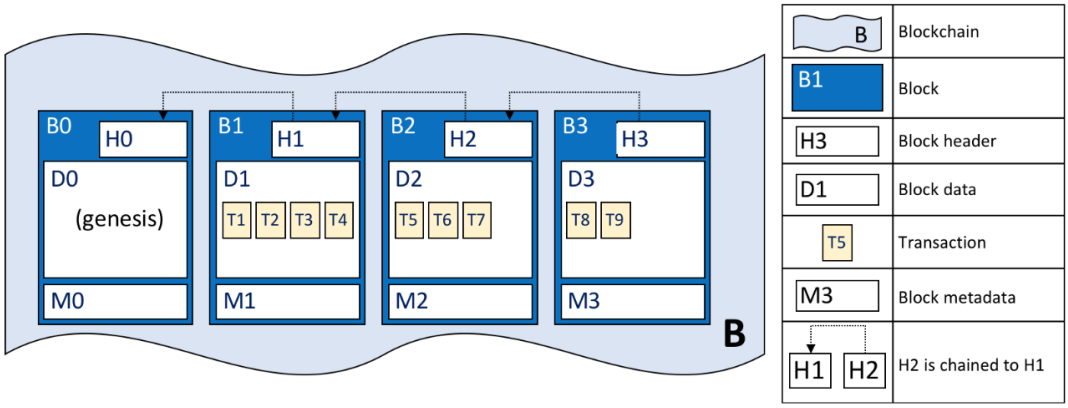

Figura 2-9. Estructura de la ledger

En el caso que nos ocupa, los assets no podrán sufrir modificaciones (no se podrá, por ejemplo, establecer un fin de tratamiento distinto para los datos de un paciente, dado que esto se registraría en otra transacción), por lo que el world state solo servirá para realizar consultas del estado de la blockchain.

Gateway

Hyperledger Fabric Gateway es un componente que proporciona la conexión entre aplicaciones cliente y la red blockchain. Funciona como un puente entre ambas, de forma que las aplicaciones pueden interactuar con la red sin tener acceso directo a los nodos y la ledger.

SDK

El SDK de Hyperledger proporciona APIs y herramientas para desarrollar aplicaciones con las que realizar acciones como conectar con la red, registrar transacciones y consultar la ledger.

Raft

A partir de la versión 2.x, el algoritmo de consenso recomendado en Fabric es Raft. Raft sigue un modelo de «líder y seguidor». El nodo líder es el encargado de recibir peticiones, ordenar las transacciones y replicar la ledger a los seguidores. Si el orderer peer líder falla, se inicia una votación en la que los seguidores eligen al

nodo que pasará a ser el líder [26].

Mientras que, en otros algoritmos, como Kafka, los nodos pertenecen a un mismo clúster que debe ser gestionado por una única organización, en Raft, cada organización puede tener sus propios nodos orderers participando en el servicio de ordenamiento. Esto aumenta la descentralización del sistema [27].

Un esquema del proceso de cambio de estado usando el algoritmo Raft se muestra en la *Figura 2-10. Funcionamiento de RAFT para cambio de estado*. El funcionamiento es el siguiente:

1. El cliente realiza una solicitud de cambio (lectura o escritura).
2. El nodo leader recibe la solicitud y la registra.
3. El leader manda el cambio a los nodos followers.
4. Los followers registran el cambio.
5. Los followers confirman al leader la recepción (envían un ACK).
6. Si el líder recibe confirmación de la mayoría de los followers, confirma el cambio.
7. El líder confirma el cambio a los followers, de forma que el sistema queda actualizado de manera consensuada.

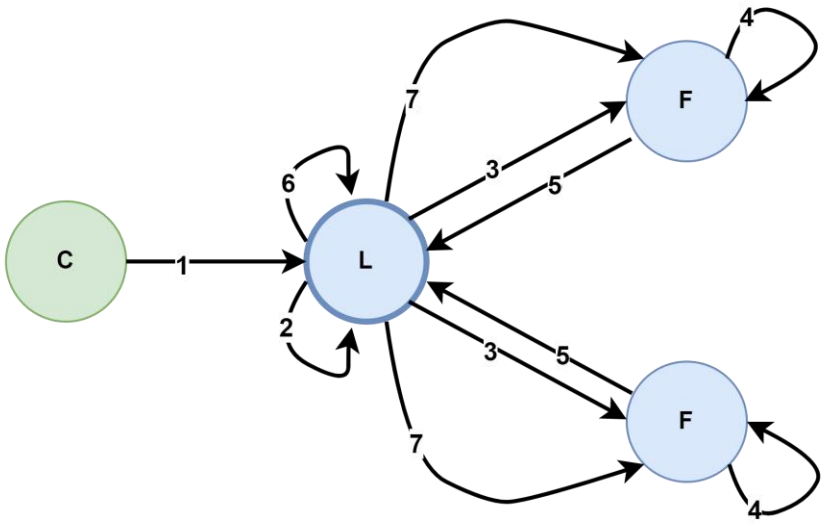

Figura 2-10. Funcionamiento de RAFT para cambio de estado

Flujo de transacciones

El flujo de transacciones de Hyperledger Fabric puede resumirse en los siguientes pasos [28]:

1. Desde una aplicación cliente se hace una propuesta de transacción (crear, leer, actualizar o borrar assets) y se manda firmada a los endorser peers de las organizaciones, a través de la *gateway*.

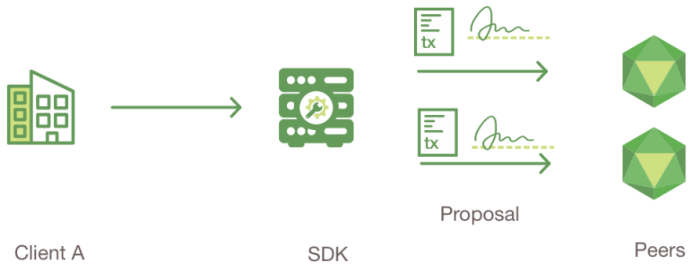

Figura 2-11. Fase 1 del flujo de transacciones

2. Los endorser peers verifican:
   - Que la propuesta está bien formada.
   - Que la transacción no se ha registrado anteriormente.
   - Que la firma del cliente es válida.
   - Que el cliente está autorizado para hacer la operación solicitada.

   Después, ejecutan la transacción invocando al chaincode, generan y firman la respuesta y la mandan a la aplicación.

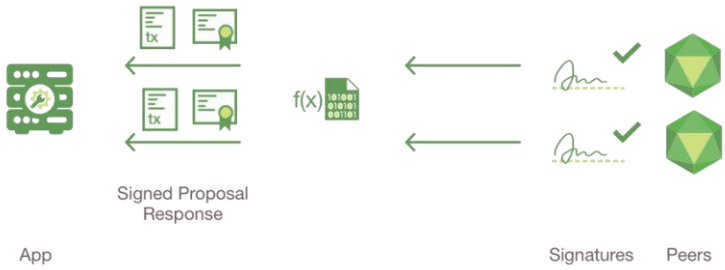

Figura 2-12. Fase 2 del flujo de transacciones

3. La aplicación comprueba las firmas de las respuestas y compara estas respuestas entre sí, para ver que son iguales. Además, si la operación que se desea realizar implica actualizar la ledger, se comprueba el cumplimiento de las políticas de los endorsers (que han respondido todos los endorser peers requeridos, etcétera).

Figura 2-13. Fase 3 del flujo de transacciones

4. La aplicación genera un mensaje que contiene la propuesta de transacción y la respuesta y la envía al servicio de ordenamiento. Después, los orderers ordenan la transacción y la añaden a un bloque.

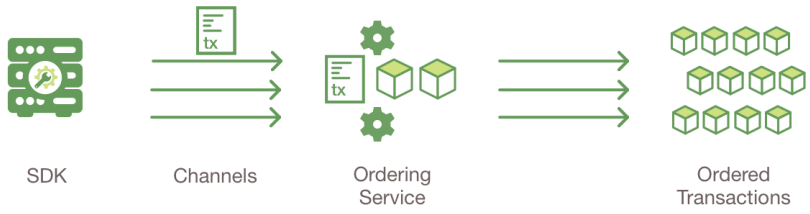

Figura 2-14. Fase 4 del flujo de transacciones

5. Cuando los peers de la red reciben un bloque, comprueban que se han cumplido las políticas y que no se han producido cambios en la ledger desde que se ejecutó la transacción en el paso 2. Se marca la transacción como válida o inválida.

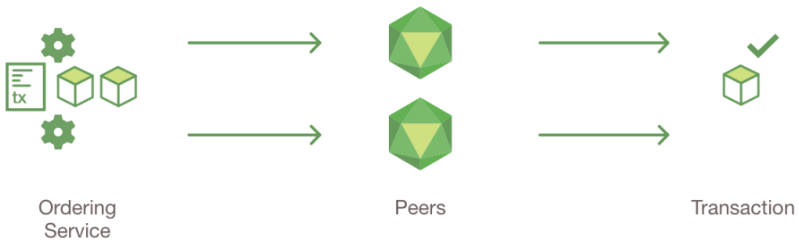

Figura 2-15. Fase 5 del flujo de transacciones

6. Los peers marcan la transacción como válida o inválida y actualizan la ledger con el bloque de transacciones válidas. Después, notifican al cliente de que la transacción ha sido añadida a la cadena.

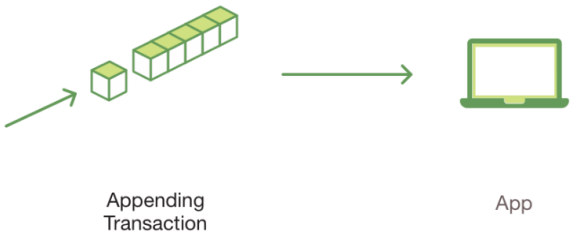

Figura 2-16. Fase 6 del flujo de transacciones

2.2.2. JWT

JSON Web Token (JWT) es un estándar abierto definido en la RFC 7519. Determina un mecanismo compacto para transmitir información segura entre dos partes con un objeto JSON. Esta información se puede verificar porque está firmada digitalmente [29].

En este proyecto, se van a usar para uno de los fines más comunes de estos tokens, la autenticación en APIs y mantenimiento de sesión. Así, al iniciar sesión, el usuario recibirá un token que incluirá en las solicitudes posteriores.

2.2.3. Docker

Docker es una herramienta usada para crear, desplegar y ejecutar aplicaciones en un entorno de contenedores. Un contenedor es un paquete ligero, ejecutable y autónomo que incluye todo lo que una aplicación necesita para ser ejecutada, incluyendo código, librerías, dependencias y runtime [30].

Con los contenedores, las aplicaciones son portables y autocontenidas, lo que facilita el despliegue de estas en diferentes plataformas y entornos.

En este proyecto, todos los peers, oderers, Cas y otros elementos de la red serán serán componentes desplegados en contenedores Docker.

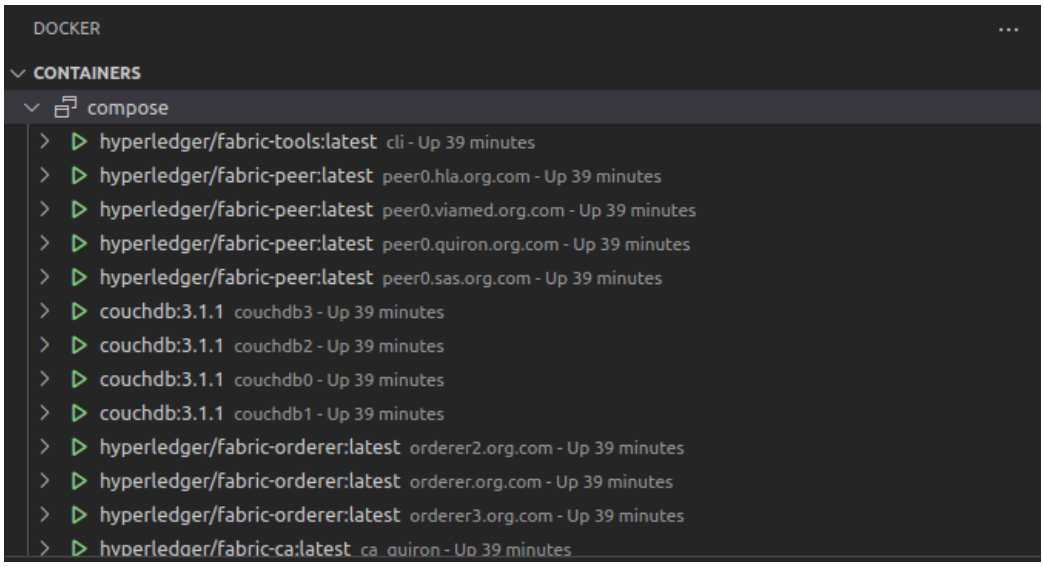

Figura 2-17. Algunos contenedores del proyecto.

2.2.4. Docker compose

Docker compose es una herramienta para definir y ejecutar aplicaciones multi-contenedor. Con Docker compose, pueden usarse ficheros YAML para definir los distintos componentes de la aplicación, tales como servicios, dependencias y configuración [31].

Docker compose se utilizará, en el caso que ocupa, para un primer despliegue en un solo equipo.

2.2.5. Docker swarm

Docker swarm permite ejecutar aplicaciones multi-contenedor en modo clúster. Es decir, mientras que con Docker compose el despliegue se realiza en un solo equipo, con Docker Swarm se puede desplegar en múltiples nodos.

En Docker Swarm, los nodos se clasifican en administradores (o managers) y trabajadores (o workers). Los managers realizan la orquestación de los contenedores y la administración del clúster, mientras que los workers reciben y ejecutan tareas desde los managers [32].

Docker swarm será utilizado en este proyecto para el despliegue de la red en un escenario con 3 equipos, donde el primero actuará como manager y los dos restantes lo harán como workers.

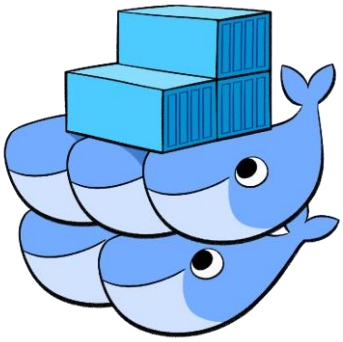

Figura 2-18. Logo Docker swarm

2.2.6. NodeJS

NodeJS es un entorno de ejecución para JavaScript que se utiliza para desarrolla aplicaciones web del lado del servidor [33].

En concreto, cabe destacar Express.js, un framework de Node ampliamente utilizado para el desarrollo de APIs. Proporciona herramientas para el manejo de rutas e implementación de funcionalidades como el manejo de sesiones y la autenticación de usuarios [34].

NodeJS es una de las opciones en las que Hyperledger Fabric ofrece su SDK y será la utilizada en esta solución.

Figura 2-19. Logo NodeJS

2.2.7. Go

Go, también conocido como Golang, es un lenguaje de programación de código abierto creado por Google en 2009. Go se caracteriza por su sintaxis clara y concisa, su tipado estricto y su enfoque en la concurrencia y la eficiencia [35].

Este lenguaje ha sido escogido entre las distintas alternativas que Hyperledger Fabric propone para el desarrollo del chaincode.

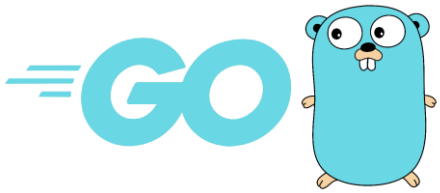

Figura 2-20. Logo Golang

2.2.8. Postman

Postman es un entorno de desarrollo de APIs que permite diseñar, probar y monitorizar servicios REST. Implementa un cliente HTTP con el que podemos interactuar con los 'endpoints' a través de los principales métodos HTTP: GET, POST, PUT y DELETE [36].

Postman será usado para realizar pruebas de la red a través de la API implementada en NodeJS, antes del desarrollo de la interfaz de usuario.

2.2.9. ReactJS

ReactsJS es una librería de JavaScript ampliamente usada para desarrollar interfaces de usuario. Desde su lanzamiento, su uso ha ido incrementando notablemente, convirtiéndose en una de las tecnologías front-end más utilizadas [37].

El elemento principal de la arquitectura de React es el componente. Un componente es una pieza independiente y reusable que, junto a otros componentes, conforma la interfaz de usuario.

ReactJS es la solución elegida para desarrollar, en este proyecto, una interfaz de usuario web sencilla y de fácil uso.

Figura 2-21. Logo React

2.2.10. Axios

Axios es una librería de JavaScript para realizar operaciones como cliente HTTP. Permite hacer peticiones Ajax de forma cómoda y potente para consumir servicios web y APIs REST [38].

Axios se basa en promesas, lo que facilita aprovechar la funcionalidad de JavaScript para construir peticiones asíncronas.

Esta será la librería utilizada desde la interfaz web para realizar las distintas operaciones ofrecidas por la API de NodeJS.

3 RESULTADOS

En este capítulo se describirá detalladamente el proceso de desarrollo del proyecto dividido en una primera sección de diseño de la solución, una segunda sección de implementación de las distintas partes de esta y una última sección de pruebas.

A pesar de que se ha realizado el proyecto siguiendo la metodología expuesta en la sección *1.2*, se ha optado por agruparlo en las tres secciones mencionadas, de forma que en cada una se presenta el diseño, la implementación y las pruebas de la solución final al completo. Esto se ha realizado para evitar la redundancia que podría provocar el hacer una presentación de estas secciones por cada iteración.

3.1 Diseño

Una vez finalizada la fase de análisis, que comprende la detección del problema, el estudio de la situación actual y el planteamiento del proyecto que puede solucionar este problema se procede a presentar el diseño de la red finalmente implementada. Un esquema de esta red puede observarse en la *Figura 3-1. Esquema de los elementos de la red del proyecto.* y está formada por los siguientes elementos:

- Cuatro organizaciones sanitarias, nombradas SAS, Quirón, HLA y Viamed.
- Un nodo peer, una autoridad de certificación, una base de datos para el "world state", una instancia del chaincode (llamado, en este caso, testqueries) y un MSP en cada organización.
- Un canal (llamado, en este caso, mychannel).
- Un servicio de ordenamiento, formado por tres nodos orderers, una autoridad de certificación y un MSP.

Todos estos componentes conforman la red, de nombre mynetwork.

Además, se muestra un diagrama de la arquitectura del proyecto en la *Figura 3-2*. En este diagrama pueden observarse las distintas partes de la solución, representadas en los contenedores Docker en los que se van a desplegar. Se ha realizado una agrupación meramente informativa según la organización a la que cada contenedor está asociado y según el tipo de contenedor (peer, chaincode, CA…). El despliegue se realizará, en el caso de Docker Compose, en un solo equipo y, en el caso de Docker Swarm, siguiendo la distribución de la *Figura 3-42. Nombres, direcciones IP y contenedores de los equipos del escenario.*

Cabe destacar que, además de los elementos mencionados anteriormente, se despliega un contenedor command-line interface (CLI), que permitirá interactuar con los nodos de la red para algunas partes del despliegue o para realizar pruebas antes de desarrollar la API REST.

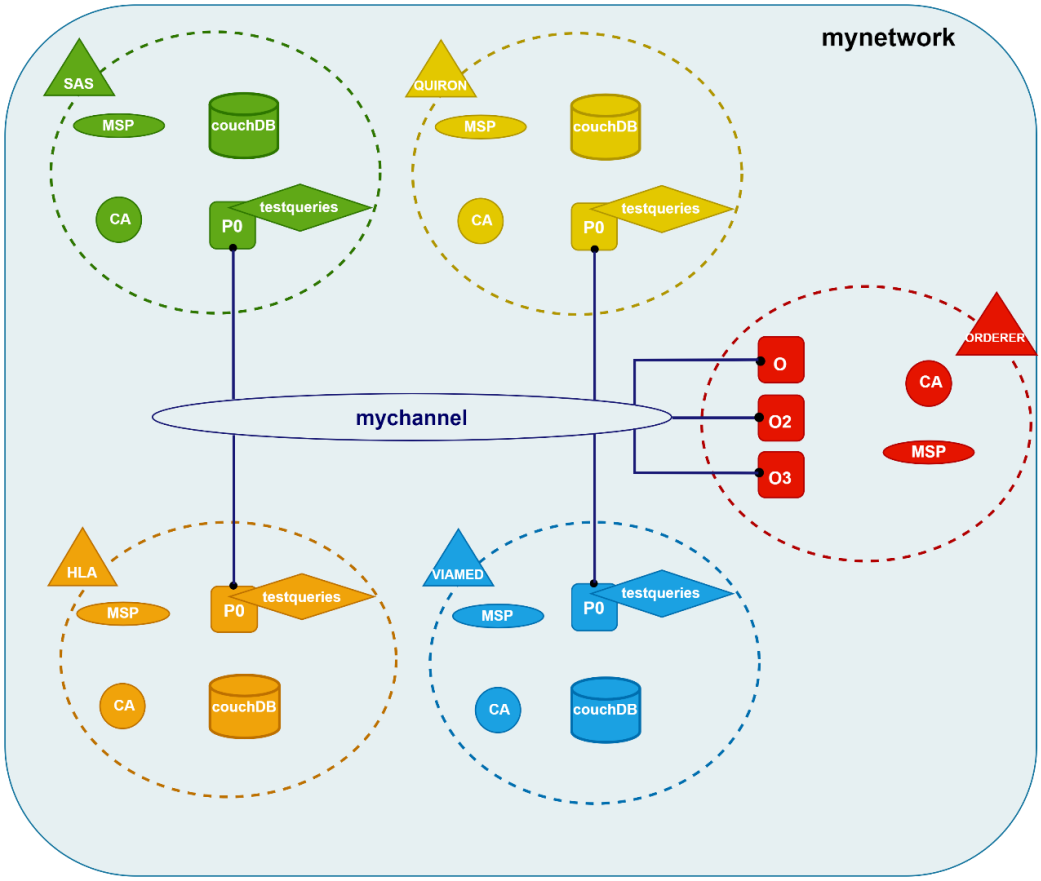

Figura 3-1. Esquema de los elementos de la red del proyecto.

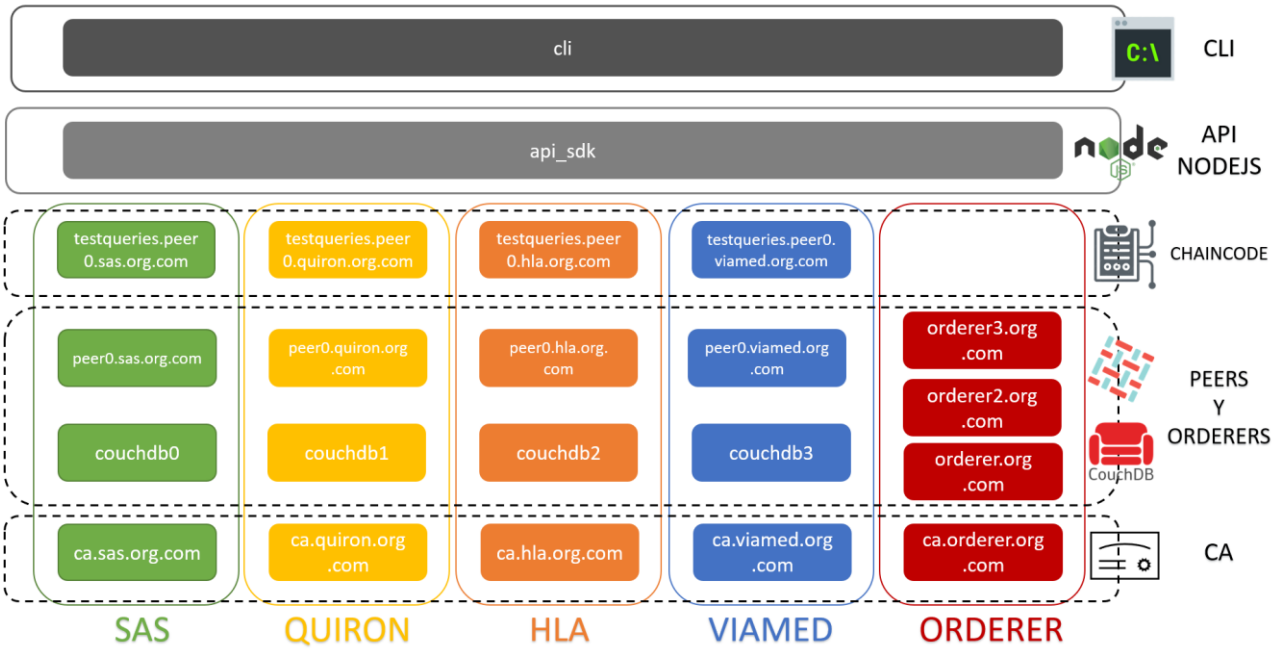

Figura 3-2. Arquitectura de la red en contenedores.

Por último, se muestra a continuación la estructura de directorios del proyecto, cuyos subdirectorios y ficheros se irán detallando en la sección siguiente.

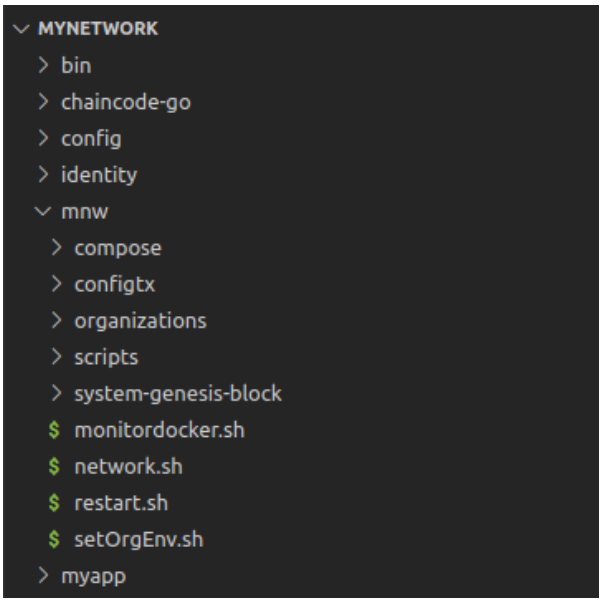

Figura 3-3. Estructura de directorios del proyecto.

3.2. Implementación

3.2.1. Configuración inicial de la red

En esta sección se explicará, en un primer lugar, qué partes deben configurarse para desplegar la red descrita anteriormente usando Docker compose (es decir, se desplegará en un solo equipo). Después, se expondrán las distintas partes del despliegue y la puesta en marcha de la red.

Autoridades de certificación

En primer lugar se han de configurar las Autoridades de Certificación (CA). Para cada uno de estos servicios, han de especificarse parámetros como la imagen del contenedor a utilizar, variables de entorno y los puertos en los que va a escuchar el servicio. Además, debe incluirse un fichero en el que se configuren las características del servidor de CA, tales como la información necesaria para las solicitudes de firma de certificado (CSR).

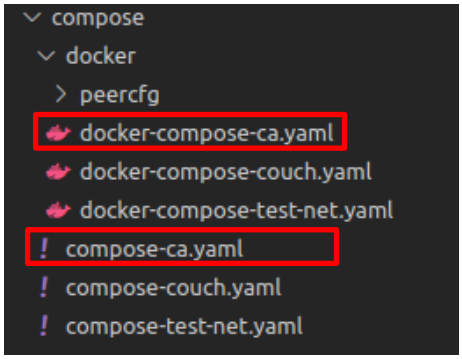

Figura 3-4. Ficheros de configuración de los servicios de CA.

```yaml
networks:
  test:
    external: true
    name: mynetwork

services:

  ca_sas:
    image: hyperledger/fabric-ca:latest
    labels:
      service: hyperledger-fabric
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
      - FABRIC_CA_SERVER_CA_NAME=ca-sas
      - FABRIC_CA_SERVER_TLS_ENABLED=true
      - FABRIC_CA_SERVER_PORT=7054
      - FABRIC_CA_SERVER_OPERATIONS_LISTENADDRESS=0.0.0.0:17054
    ports:
      - "7054:7054"
      - "17054:17054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw -d'
    volumes:
      - ../organizations/fabric-ca/sas:/etc/hyperledger/fabric-ca-server
    container_name: ca_sas
    networks:
      - test
```

Figura 3-5. Fragmento de compose-ca.yaml para la CA de la organización SAS.

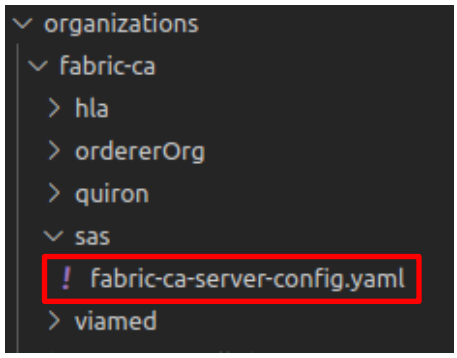

Figura 3-6. Fichero de configuración del servidor de CA.

Peers y couchDBs

Una vez configuradas las CAs, se procede a configurar los peers. Para ello, se utilizarán dos ficheros en los que se establecerán las imágenes a utilizar, las variables de entorno y otros parámetros de configuración. Además, puede usarse otro fichero en el directorio peercfg para la configuración que no se añada en las variables de entorno. En estos ficheros, también se configurará el contenedor CLI.

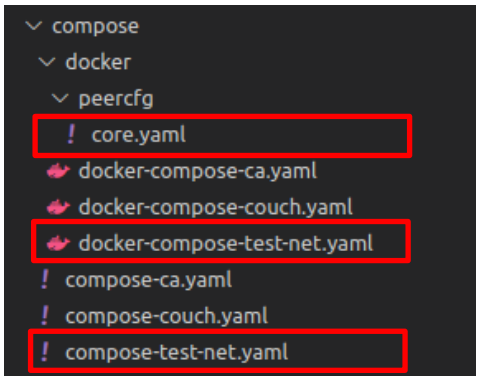

Figura 3-7. Ficheros de configuración de los peers.

```yaml
peer0.sas.org.com:
  container_name: peer0.sas.org.com
  image: hyperledger/fabric-peer:latest
  labels:
    service: hyperledger-fabric
  environment:
    - FABRIC_CFG_PATH=/etc/hyperledger/peercfg
    - FABRIC_LOGGING_SPEC=INFO
    #- FABRIC_LOGGING_SPEC=DEBUG
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_PROFILE_ENABLED=false
    - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
    - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
    - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
    # Peer specific variables
    - CORE_PEER_ID=peer0.sas.org.com
    - CORE_PEER_ADDRESS=peer0.sas.org.com:7051
    - CORE_PEER_LISTENADDRESS=0.0.0.0:7051
    - CORE_PEER_CHAINCODEADDRESS=peer0.sas.org.com:7052
    - CORE_PEER_CHAINCODELISTENADDRESS=0.0.0.0:7052
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.sas.org.com:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.sas.org.com:7051
    - CORE_PEER_LOCALMSPID=SasMSP
    - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/fabric/msp
    - CORE_OPERATIONS_LISTENADDRESS=peer0.sas.org.com:9444
    - CHAINCODE_AS_A_SERVICE_BUILDER_CONFIG={"peername":"peer0sas"}
    - CORE_CHAINCODE_EXECUTETIMEOUT=300s
  volumes:
      - ../organizations/peerOrganizations/sas.org.com/peers/peer0.sas.org.com:/etc/hyperledger/fabric
      - peer0.sas.org.com:/var/hyperledger/production
  working_dir: /root
  command: peer node start
  ports:
    - 7051:7051
    - 9444:9444
  networks:
    - test
```

Figura 3-8. Fragmento de compose-test-net.yaml para el peer de la organización SAS.

```yaml
peer0.sas.org.com:
  container_name: peer0.sas.org.com
  image: hyperledger/fabric-peer:latest
  labels:
    service: hyperledger-fabric
  environment:
    #Generic peer variables
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=mynetwork
  volumes:
    - ./docker/peercfg:/etc/hyperledger/peercfg
    - ${DOCKER_SOCK}:/host/var/run/docker.sock
```

Figura 3-9. Fragmento de docker-compose-test-net.yaml para el peer de la organización SAS.

Una vez configurados los peers, se pasa a configurar la base de datos que dará soporte al world state. Hyperledger Fabric ofrece la posibilidad de usar para ello CouchDB. CouchDB es un SGBD de Apache, de código abierto y NoSQL. Recoge la información en documentos con formato JSON [39].

Se deberá configurar una instancia de CouchDB para cada peer.

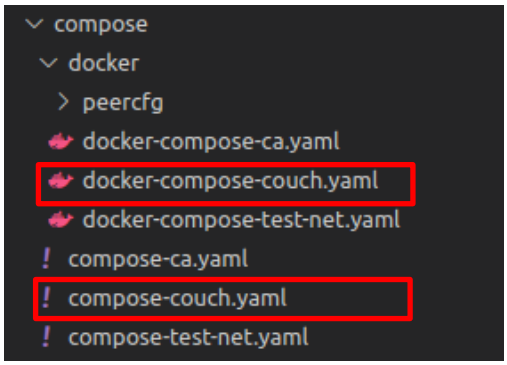

Figura 3-10. Ficheros de configuración de couchDB.

```yaml
couchdb1:
  container_name: couchdb1
  image: couchdb:3.1.1
  labels:
    service: hyperledger-fabric
  environment:
    - COUCHDB_USER=admin
    - COUCHDB_PASSWORD=adminpw
  ports:
    - "7984:5984"
  networks:
    - test

peer0.quiron.org.com:
  environment:
    - CORE_LEDGER_STATE_STATEDATABASE=CouchDB
    - CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb1:5984
    - CORE_LEDGER_STATE_COUCHDBCONFIG_USERNAME=admin
    - CORE_LEDGER_STATE_COUCHDBCONFIG_PASSWORD=adminpw
  depends_on:
    - couchdb1
```

Figura 3-11. Fragmento de compose-couch.yaml para la CouchDB de la organización Quirón.

Orderers

En el mismo fichero en el que se han configurado los peers (compose-test-net.yaml) se configurarán los orderers. En este caso habrá tres contenedores, uno para cada nodo orderer.

```
orderer2.org.com:
  container_name: orderer2.org.com
  image: hyperledger/fabric-orderer:latest
  labels:
    service: hyperledger-fabric
  environment:
    - FABRIC_LOGGING_SPEC=INFO
    - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
    - ORDERER_GENERAL_LISTENPORT=8050
    - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
    - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
    # enabled TLS
    - ORDERER_GENERAL_TLS_ENABLED=true
    - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
    - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
    - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
    - ORDERER_GENERAL_CLUSTER_CLIENTCERTIFICATE=/var/hyperledger/orderer/tls/server.crt
    - ORDERER_GENERAL_CLUSTER_CLIENTPRIVATEKEY=/var/hyperledger/orderer/tls/server.key
    - ORDERER_GENERAL_CLUSTER_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
    - ORDERER_GENERAL_BOOTSTRAPMETHOD=none
    - ORDERER_CHANNELPARTICIPATION_ENABLED=true
    - ORDERER_ADMIN_TLS_ENABLED=true
    - ORDERER_ADMIN_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
    - ORDERER_ADMIN_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
    - ORDERER_ADMIN_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
    - ORDERER_ADMIN_TLS_CLIENTROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
    - ORDERER_ADMIN_LISTENADDRESS=0.0.0.0:8053
    - ORDERER_OPERATIONS_LISTENADDRESS=orderer2.org.com:9441
  working_dir: /root
  command: orderer
  volumes:
      - ../organizations/ordererOrganizations/org.com/orderers/orderer2.org.com/msp:/var/hyperledger/orderer/msp
      - ../organizations/ordererOrganizations/org.com/orderers/orderer2.org.com/tls/:/var/hyperledger/orderer/tls
      - orderer2.org.com:/var/hyperledger/production/orderer
  ports:
    - 8050:8050
    - 8053:8053
    - 9441:9441
  networks:
    - test
```

Figura 3-12. Fragmento de compose-test-net.yaml para el orderer2

Algoritmo de consenso, directorio MSP y políticas

Además de los servicios a desplegar, deben configurarse otros aspectos en el fichero configtx.yaml (*Figura 3-3. Estructura de directorios del proyecto.*). Esta configuración comprende:

- **Directorio MSP y políticas de firma para cada organización**: se establecerá el directorio base que actuará como MSP (aquí se almacenarán todos los certificados necesarios para los integrantes de la organización).

  Con las políticas de firma, cada organización puede especificar qué miembros o roles pueden realizar distintas operaciones en el canal y la ledger. Las palabras reservadas que pueden usarse son AND, OR y NoutOf (y, o, número sobre, por ejemplo, 11 de 20). En este caso, podrán hacer lecturas los roles de administrador, peer y cliente; podrán escribir los roles de administrador y cliente; podrá hacer funciones de administración el rol de administrador y podrá validar transacciones el rol de peer.

```yaml
- &Hla
    Name: HlaMSP

    ID: HlaMSP

    MSPDir: ../organizations/peerOrganizations/hla.org.com/msp

    Policies:
        Readers:
            Type: Signature
            Rule: "OR('HlaMSP.admin', 'HlaMSP.peer', 'HlaMSP.client')"
        Writers:
            Type: Signature
            Rule: "OR('HlaMSP.admin', 'HlaMSP.client')"
        Admins:
            Type: Signature
            Rule: "OR('HlaMSP.admin')"
        Endorsement:
            Type: Signature
            Rule: "OR('HlaMSP.peer')"
```

Figura 3-13. Fragmento de configtx.yaml para el MSP y políticas de firma de la organización HLA.

- **Políticas implícitas**: son políticas que sirven para validar las políticas de firma (están a un nivel menos profundo que estas últimas). Esto puede ser útil para evitar conflictos en caso de cambios en las políticas de firma. Por ejemplo, en este caso, la política de firma de "Endorsement" (roles que pueden validar transacciones) debe ser validada por la mayoría de los que pertenecen a esta política de firma. Las palabras reservadas que pueden usarse son ANY, ALL y MAJORITY (cualquiera, todos, la mayoría).

```yaml
Application: &ApplicationDefaults

    Organizations:

    Policies:
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
        LifecycleEndorsement:
            Type: ImplicitMeta
            Rule: "MAJORITY Endorsement"
        Endorsement:
            Type: ImplicitMeta
            Rule: "MAJORITY Endorsement"
```

Figura 3-14. Fragmento de configtx.yaml para políticas implícitas.

- **Algoritmo de consenso**: por último, cabe destacar la configuración de los perfiles. En esta sección, se establecerá la configuración a tener en cuenta para crear el primer bloque de la red, el llamado bloque génesis.

  Como puede observarse en la *Figura 3-15. Fragmento de configtx.yaml para el bloque génesis y el algoritmo de consenso.*, se establecen las organizaciones que forman parte de la aplicación, así como el tipo de algoritmo de consenso y los participantes en este. Para este proyecto, se usará el algoritmo RAFT, detallado en capítulos anteriores, y estará formado por tres orderers. Para cada uno de ellos se define el nombre, el puerto y la ubicación de los certificados correspondientes.

```
Profiles:

    TwoOrgsApplicationGenesis:
        <<: *ChannelDefaults
        Orderer:
            <<: *OrdererDefaults
            OrdererType: etcdraft
            EtcdRaft:
                Consenters:
                - Host: orderer.org.com
                  Port: 7050
                  ClientTLSCert: ../organizations/ordererOrganizations/org.com/orderers/orderer.org.com/tls/server.crt
                  ServerTLSCert: ../organizations/ordererOrganizations/org.com/orderers/orderer.org.com/tls/server.crt
                - Host: orderer2.org.com
                  Port: 8050
                  ClientTLSCert: ../organizations/ordererOrganizations/org.com/orderers/orderer2.org.com/tls/server.crt
                  ServerTLSCert: ../organizations/ordererOrganizations/org.com/orderers/orderer2.org.com/tls/server.crt
                - Host: orderer3.org.com
                  Port: 9050
                  ClientTLSCert: ../organizations/ordererOrganizations/org.com/orderers/orderer3.org.com/tls/server.crt
                  ServerTLSCert: ../organizations/ordererOrganizations/org.com/orderers/orderer3.org.com/tls/server.crt

            Organizations:
                - *OrdererOrg
            Capabilities: *OrdererCapabilities
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *Sas
                - *Quiron
                - *Hla
                - *Viamed
            Capabilities: *ApplicationCapabilities
```

Figura 3-15. Fragmento de configtx.yaml para el bloque génesis y el algoritmo de consenso.

Chaincode

Por último, se procede a configurar el chaincode. Para este proyecto, se ha desarrollado en lenguaje Golang.

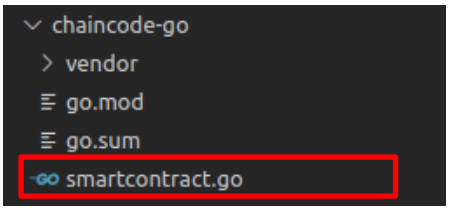

Figura 3-16. Ubicación del smartcontract.

En el fichero del chaincode pueden diferenciarse las siguientes partes:

- **Importación de paquetes**. Cabe destacar el paquete "shim", que implementa la interfaz "ChaincodeStubInterface" y ofrece métodos para acceder y modificar la ledger, y el paquete "peer", que implementa el mecanismo "Protocol Buffer", utilizado para serializar datos estructurados.

```
import (
    "bytes"
    "encoding/json"
    "fmt"
    "strconv"

    "github.com/hyperledger/fabric-chaincode-go/shim"
    pb "github.com/hyperledger/fabric-protos-go/peer"
)
```

Figura 3-17. Importación de paquetes en el chaincode.

- **Definición de los assets**. Para definir los assets, se crean tipos de tipo estructura, dentro de las cuales se especifican los campos del asset. Además, se añade la etiqueta del JSON para su posterior conversión a este formato.

  Esta estructura se ha formado a partir de los campos de los recursos FHIR mencionados en las figuras *Figura 2-3. Formato JSON recurso AuditEvent*, *Figura 2-4. Formato JSON recurso Organization* y *Figura 2-5. Formato JSON recurso Patient*.

```
type SmartContract struct {
}

type Organization struct {
    ResourceType string `json:"resourceType"`
    Identifier   string `json:"identifier"`
    Name         string `json:"name"`
}

type Agent struct {
    Type string       `json:"type"`
    Role string       `json:"role"`
    Who  Organization `json:"who"`
}

type Detail struct {
    Type  string `json:"type"`
    Value string `json:"value"`
}

type Entity struct {
    ResourceType string `json:"resourceType"`
    Identifier   string `json:"identifier"`
    Detail       Detail `json:"detail"`
}

type Asset struct {
    Identifier     string       `json:"identifier"`
    ResourceType   string       `json:"resourceType"`
    Type           string       `json:"type"`
    Action         string       `json:"action"`
    Recorded       string       `json:"recorded"`
    PurposeOfEvent string       `json:"purposeOfEvent"`
    Agent1         Agent        `json:"agent1"`
    Agent2         Agent        `json:"agent2"`
    Source         Organization `json:"source"`
    Entity         Entity       `json:"entity"`
}
```

Figura 3-18. Definición de los assets en el chaincode.

- **Definición de funciones**. Se dispondrá de una función principal, "Invoke", que será a la que se llame desde la línea de comandos o desde la API de Node. Dentro de esta función, se invocará a la función encargada de realizar la operación que se ha solicitado, pasada como parámetro a Invoke.

```
func (s *SmartContract) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    function, args := stub.GetFunctionAndParameters()
    fmt.Println(" ")
    fmt.Println("starting invoke, for - " + function)

    // Handle different functions
    if function == "init" {
        return s.Init(stub)
    } else if function == "readBySource" {
        return s.GetAssetsBySourceOrg(stub, args)
    } else if function == "readByDestination" {
        return s.GetAssetsByDestinationOrg(stub, args)
    } else if function == "readByDate" {
        return s.GetAssetsByEndDate(stub, args)
    } else if function == "readByPatient" {
        return s.GetAssetsByPatient(stub, args)
    } else if function == "newTransaction" {
        return s.CreateAsset(stub, args)
    } else if function == "readAll" {
        return s.GetAllAssets(stub, args)
    }

    // error out
    fmt.Println("Received unknown invoke function name - " + function)
    return shim.Error("Received unknown invoke function name - '" + function + "'")
}
```

Figura 3-19. Función Invoke del chaincode.

Las distintas funciones implementadas se resumen en la *Tabla 3-1. Funciones del chaincode.*.

Nombre de la función	Recibe	Devuelve	Funcionalidad
CreateAsset	ChaincodeStubInterface, puntero al SmartContract, datos para el nuevo asset	pb.Response con mensaje de éxito o error	Completa los campos del asset con los argumentos recibidos. Convierte a formato JSON y añade el asset a la ledger. Crea claves compuestas con el id del asset y la org. Origen, org. Destino, fecha de fin de periodo de validez y DNI del paciente para posteriores búsquedas.
GetAllAssets	ChaincodeStubInterface, puntero al SmartContract	pb.Response con mensaje de éxito y la lista de assets o error	Recorre la ledger y añade cada asset a una lista para devolerla con todos los assets.
GetAssetsBySourceOrg	ChaincodeStubInterface, puntero al SmartContract, nombre de la org. Origen	pb.Response con mensaje de éxito y la lista de assets o error	Recorre la ledger usando la clave compuesta correspondiente para añadir a una lista los assets que tengan como org. Origen la que se ha pasado como parámetro.
GetAssetsByDestinationOrg	ChaincodeStubInterface, puntero al SmartContract, nombre de la org. Destino	pb.Response con mensaje de éxito y la lista de assets o error	Recorre la ledger usando la clave compuesta correspondiente para añadir a una lista los assets que tengan como org. Destino la que se ha pasado como parámetro.
GetAssetsByEndDate	ChaincodeStubInterface, puntero al SmartContract, fecha fin de periodo de validez	pb.Response con mensaje de éxito y la lista de assets o error	Recorre la ledger usando la clave compuesta correspondiente para añadir a una lista los assets que tengan como fecha de fin de periodo de validez la que se ha pasado como parámetro.
GetAssetsByPatient	ChaincodeStubInterface, puntero al SmartContract, DNI del paciente	pb.Response con mensaje de éxito y la lista de assets o error	Recorre la ledger usando la clave compuesta correspondiente para añadir a una lista los assets que tengan como DNI del paciente el que se ha pasado como parámetro.

Tabla 3-1. Funciones del chaincode.

```go
func (s *SmartContract) GetAssetsBySourceOrg(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    if len(args) != 1 {
        return shim.Error("Incorrect number of arguments. Expecting 1 (Source Organization name).")
    }

    sourceOrg := args[0]
    CompKey := "sourceOrg~indentifier"
    resultsIterator, err := stub.GetStateByPartialCompositeKey(CompKey, []string{sourceOrg})
    if err != nil {
        return shim.Error(err.Error())
    }
    defer resultsIterator.Close()

    var buffer bytes.Buffer
    buffer.WriteString("[")

    bArrayMemberAlreadyWritten := false
    for resultsIterator.HasNext() {
        queryResponse, err := resultsIterator.Next()

        if err != nil {
            return shim.Error(err.Error())
        }
        _, compositeKeyParts, err := stub.SplitCompositeKey(queryResponse.Key)
        if err != nil {
            return shim.Error(err.Error())
        }

        returnedAssetID := compositeKeyParts[1]
        assetAsBytes, err := stub.GetState(returnedAssetID)
        if err != nil {
            return shim.Error(err.Error())
        }
        if bArrayMemberAlreadyWritten {
            buffer.WriteString(",")
        }
        buffer.WriteString("{\"Key\":")
        buffer.WriteString("\"")
        buffer.WriteString(returnedAssetID)
        buffer.WriteString("\"")

        buffer.WriteString(", \"Record\":")
        buffer.WriteString(string(assetAsBytes))
        buffer.WriteString("}")
        bArrayMemberAlreadyWritten = true
    }

    buffer.WriteString("]")

    return shim.Success(buffer.Bytes())
}
```

Figura 3-20. Función GetAssetsBySourceOrg del chaincode.

3.2.2. Despliegue con Docker Compose

Tras finalizar la configuración, el siguiente paso es desplegar y poner en marcha la red. Con este fin, Hyperledger Fabric facilita un conjunto de scripts para hacer el despliegue de forma ordenada, cómoda y rápida.

A continuación, se enumeran y explican las fases o pasos que se deben seguir para desplegar la red.

1. **Levantar la red**

   a) Levantar los contenedores de las CA.

   b) Crear MSP para cada organización, registrar e inscribir a los usuarios.

      El MSP en Hyperledger Fabric no es más que una estructura de directorios que contienen los certificados TLS y de la organización. En este paso, se crearán los directorios necesarios y se hará el registro e inscripción de un rol peer, un rol usuario y un rol administrador.

      En el proceso de registro se creará una identidad para el usuario en el sistema, asignándole un

certificado y una clave privada. En este punto, el usuario aún no tiene acceso de la red.

En el proceso de inscripción, el usuario utilizará su certificado y clave para obtener acceso a la red y así activar su identidad.

c) Generar los CCPs

Los perfiles de conexión son ficheros en formato JSON o YAML que describen la topología de la red a nivel de peers, orderers y CAs. Estos ficheros configuran la conexión de una aplicación o cliente (en este caso, el SDK) con una red de Hyperledger. Se generará un CCP para cada organización.


```
{
    "name": "mynetwork-viamed",
    "version": "1.0.0",
    "client": {
        "organization": "Viamed",
        "connection": {
            "timeout": {
                "peer": {
                    "endorser": "300"
                }
            }
        }
    },
    "organizations": {
        "viamed": {
            "mspid": "ViamedMSP",
            "peers": [
                "peer0.viamed.org.com"
            ],
            "certificateAuthorities": [
                "ca.viamed.org.com"
            ]
        }
    },
    "peers": {
        "peer0.viamed.org.com": {
            "url": "grpcs://localhost:13051",
            "tlsCACerts": {
                "path": "organizations/peerOrganizations/viamed.org.com/tlsca/tlsca.viamed.org.com-cert.pem"
            },
            "grpcOptions": {
                "ssl-target-name-override": "peer0.viamed.org.com",
                "hostnameOverride": "peer0.viamed.org.com"
            }
        }
    },
    "certificateAuthorities": {
        "ca.viamed.org.com": {
            "url": "https://localhost:10054",
            "caName": "ca-viamed",
            "tlsCACerts": {
                "path": "organizations/peerOrganizations/viamed.org.com/ca/ca.viamed.org.com-cert.pem"
            },
            "httpOptions": {
                "verify": false
            }
        }
    }
}
```


Figura 3-21. Perfil de conexión de la organización Viamed.

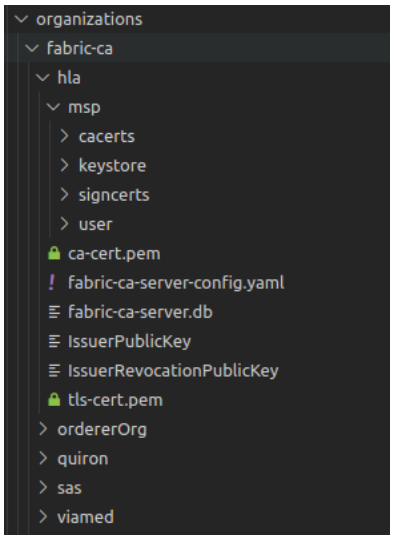

Figura 3-22. Directorios tras la fase 1 del despliegue (I)

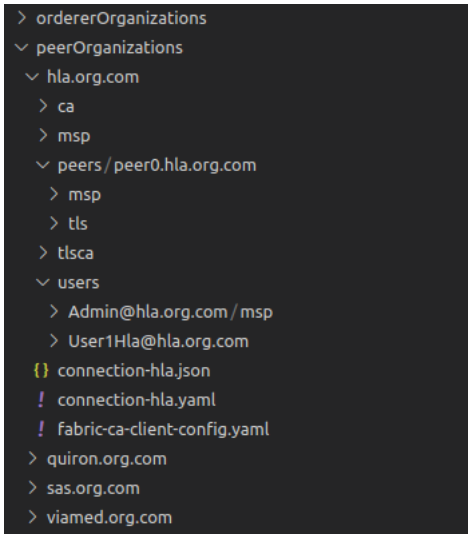

Figura 3-23. Directorios tras la fase 1 del despliegue (II).

d) Levantar los contenedores de los peers y couchDB

2. Crear los "Channel Artifacts" (elementos del canal)

a) Crear el bloque génesis

El bloque génesis es el primer bloque de la blockchain y contiene la información sobre la red indicada en configtx.yaml.

b) Crear el canal

En este paso, se creará el canal y se unirán a este los tres nodos orderers. En este punto, los orderers habrán interactuado entre ellos para definir quién será el leader y quiénes los followers (sección *2.2.1.2 – Raft*).

c) Unir los peers al canal

d) Establecer anchor peers

Tal y como se ha descrito en la sección *2.2.1.2 – Peers*, los anchor peers permiten que otros peers se comuniquen con ellos para describir la red. En este paso, se añadirá un anchor peer por cada organización.

```
osnadmin channel join --channelID $CHANNEL_NAME --config-block ./channel-artifacts/${CHANNEL_NAME}.block \
    -o localhost:7053 --ca-file "$ORDERER_CA" --client-cert "$ORDERER_ADMIN_TLS_SIGN_CERT" \
    --client-key "$ORDERER_ADMIN_TLS_PRIVATE_KEY" >&log.txt
osnadmin channel join --channelID $CHANNEL_NAME --config-block ./channel-artifacts/${CHANNEL_NAME}.block \
    -o localhost:8053 --ca-file "$ORDERER_CA" --client-cert "$ORDERER2_ADMIN_TLS_SIGN_CERT" \
    --client-key "$ORDERER2_ADMIN_TLS_PRIVATE_KEY" >&log.txt
osnadmin channel join --channelID $CHANNEL_NAME --config-block ./channel-artifacts/${CHANNEL_NAME}.block \
    -o localhost:9053 --ca-file "$ORDERER_CA" --client-cert "$ORDERER3_ADMIN_TLS_SIGN_CERT" \
    --client-key "$ORDERER3_ADMIN_TLS_PRIVATE_KEY" >&log.txt
```

Figura 3-24. Creación del canal y unión de los orderers.

```
[orderer.consensus.etcdraft] becomePreCandidate -> 2 became pre-candidate at term 1 channel=mychannel node=2
[orderer.consensus.etcdraft] poll -> 2 received MsgPreVoteResp from 2 at term 1 channel=mychannel node=2
[orderer.consensus.etcdraft] campaign -> 2 [logterm: 1, index: 3] sent MsgPreVote request to 3 at term 1 channel=mychannel node=2
[orderer.consensus.etcdraft] campaign -> 2 [logterm: 1, index: 3] sent MsgPreVote request to 1 at term 1 channel=mychannel node=2
[orderer.consensus.etcdraft] poll -> 2 received MsgPreVoteResp from 3 at term 1 channel=mychannel node=2
[orderer.consensus.etcdraft] stepCandidate -> 2 [quorum:2] has received 2 MsgPreVoteResp votes and 0 vote rejections channel=mychannel node=2
[orderer.consensus.etcdraft] becomeCandidate -> 2 became candidate at term 2 channel=mychannel node=2
[orderer.consensus.etcdraft] poll -> 2 received MsgVoteResp from 2 at term 2 channel=mychannel node=2
[orderer.consensus.etcdraft] campaign -> 2 [logterm: 1, index: 3] sent MsgVote request to 1 at term 2 channel=mychannel node=2
[orderer.consensus.etcdraft] campaign -> 2 [logterm: 1, index: 3] sent MsgVote request to 3 at term 2 channel=mychannel node=2
[orderer.consensus.etcdraft] poll -> 2 received MsgVoteResp from 3 at term 2 channel=mychannel node=2
[orderer.consensus.etcdraft] stepCandidate -> 2 [quorum:2] has received 2 MsgVoteResp votes and 0 vote rejections channel=mychannel node=2
[orderer.consensus.etcdraft] becomeLeader -> 2 became leader at term 2 channel=mychannel node=2
[orderer.consensus.etcdraft] run -> raft.node: 2 elected leader 2 at term 2 channel=mychannel node=2
[orderer.consensus.etcdraft] run -> Leader 2 is present, quit campaign channel=mychannel node=2
[orderer.consensus.etcdraft] run -> Raft leader changed: 0 -> 2 channel=mychannel node=2
[orderer.consensus.etcdraft] run -> Start accepting requests as Raft leader at block [0] channel=mychannel node=2
```

Figura 3-25. Votación y elección de leader y followers en RAFT.

3. **Desplegar el chaincode**

   a) Empaquetar el chaincode en un archivo .tar.gz e instalar en los peers.

   b) Comprobaciones de la instalación.

   Para finalizar con el despliegue, se debe comprobar la correcta instalación del chaincode, aprobar la definición de este y confirmarla. Una vez realizados estos pasos, la red estará desplegada y lista para ser usada.

3.2.3. Pruebas en CLI

Una vez desplegada la red, se puede hacer uso del CLI y los comandos que ofrece para comprobar el correcto funcionamiento del chaincode. A continuación, se presenta un ejemplo de creación de un asset y de lectura de todos los assets.

```
marta@manager:~/mynetwork/mnw$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.
org.com --tls --cafile ${PWD}/organizations/ordererOrganizations/org.com/tlsca/tlsca.org.com-cert.pem -C $CH
ANNEL_NAME -n testqueries --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD}/organizations/peerOrganiz
ations/sas.org.com/tlsca/tlsca.sas.org.com-cert.pem --peerAddresses localhost:9051 --tlsRootCertFiles ${PWD}
/organizations/peerOrganizations/quiron.org.com/tlsca/tlsca.quiron.org.com-cert.pem --peerAddresses localhos
t:11051 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/hla.org.com/tlsca/tlsca.hla.org.com-cert.p
em --peerAddresses localhost:13051 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/viamed.org.com/
tlsca/tlsca.viamed.org.com-cert.pem -c '{"Args":["newTransaction", "01", "01","Sas","02","Quiron","2022-08-1
2/10:00:00","48123231B","TREAT", "2022-07-15/13:30:00"]}'
2023-01-25 17:37:55.676 CET 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful.
result: status:200
```

Figura 3-26. Creación de asset a través del CLI.

```
marta@manager:~/mynetwork/mnw$ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.
org.com --tls --cafile ${PWD}/organizations/ordererOrganizations/org.com/tlsca/tlsca.org.com-cert.pem -C $CH
ANNEL_NAME -n testqueries --peerAddresses localhost:7051 --tlsRootCertFiles ${PWD}/organizations/peerOrganiz
ations/sas.org.com/tlsca/tlsca.sas.org.com-cert.pem --peerAddresses localhost:9051 --tlsRootCertFiles ${PWD}
/organizations/peerOrganizations/quiron.org.com/tlsca/tlsca.quiron.org.com-cert.pem --peerAddresses localhos
t:11051 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/hla.org.com/tlsca/tlsca.hla.org.com-cert.p
em --peerAddresses localhost:13051 --tlsRootCertFiles ${PWD}/organizations/peerOrganizations/viamed.org.com/
tlsca/tlsca.viamed.org.com-cert.pem -c '{"Args":["readAll"]}'
2023-01-25 17:39:59.147 CET 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincode invoke successful.
result: status:200 payload:"[{\"identifier\":\"01\",\"resourceType\":\"AuditEvent\",\"type\":\"receive\",\"a
ction\":\"C\",\"recorded\":\"2022-07-15/13:30:00\",\"purposeOfEvent\":\"TREAT\",\"agent1\":{\"type\":\"PROV\
",\"role\":\"AUT\",\"who\":{\"resourceType\":\"Organization\",\"identifier\":\"01\",\"name\":\"Sas\"}},\"age
nt2\":{\"type\":\"PROV\",\"role\":\"IRCP\",\"who\":{\"resourceType\":\"Organization\",\"identifier\":\"02\",
\"name\":\"Quiron\"}},\"source\":{\"resourceType\":\"Organization\",\"identifier\":\"01\",\"name\":\"Sas\"},
\"entity\":{\"resourceType\":\"Patient\",\"identifier\":\"48123231B\",\"detail\":{\"type\":\"End of validity
 period date\",\"value\":\"2022-08-12/10:00:00\"}}},{\"identifier\":\"02\",\"resourceType\":\"AuditEvent\",\
"type\":\"receive\",\"action\":\"C\",\"recorded\":\"2022-07-15/13:30:00\",\"purposeOfEvent\":\"HRESCH\",\"ag
ent1\":{\"type\":\"PROV\",\"role\":\"AUT\",\"who\":{\"resourceType\":\"Organization\",\"identifier\":\"02\",
\"name\":\"Quiron\"}},\"agent2\":{\"type\":\"PROV\",\"role\":\"IRCP\",\"who\":{\"resourceType\":\"Organizati
on\",\"identifier\":\"03\",\"name\":\"Hla\"}},\"source\":{\"resourceType\":\"Organization\",\"identifier\":\
"02\",\"name\":\"Quiron\"},\"entity\":{\"resourceType\":\"Patient\",\"identifier\":\"48123231B\",\"detail\":
{\"type\":\"End of validity period date\",\"value\":\"2022-09-12/10:00:00\"}}}]"
```

Figura 3-27. Lectura de assets a través del CLI.

Por último, se puede acceder a la dirección de couchDB de los peers (http://127.0.0.1:<puertoCouchDB>/utils/) para visualizar fácilmente el contenido de la ledger.

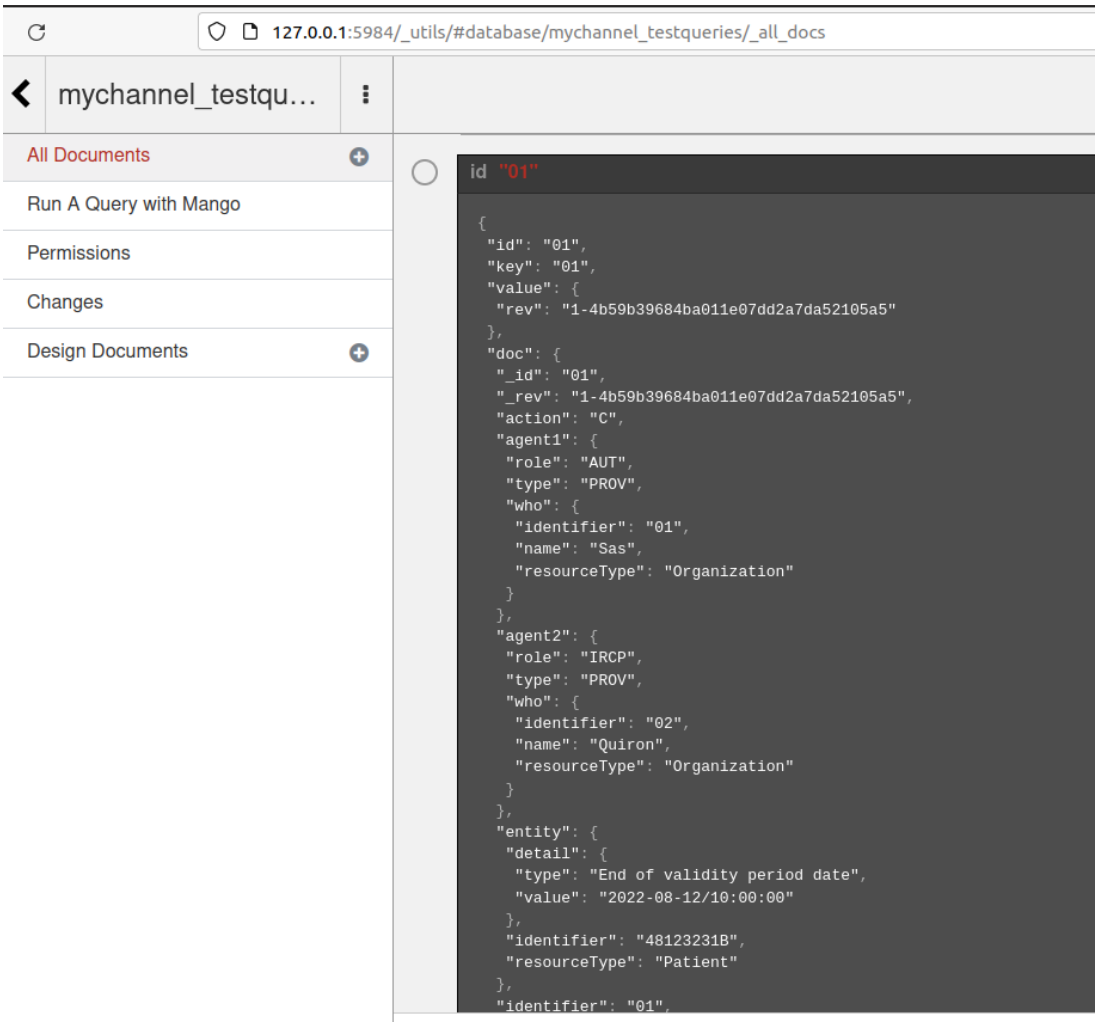

Figura 3-28. Contenido de la ledger desde couchDB.

3.2.4. Aplicación

En esta sección, se abordará el uso de las herramientas del SDK de Hyperledger para desarrollar la aplicación de la red. Se ha desarrollado basándose en el código de prueba de Hperledger Fabric y se ha estructurado siguiendo como ejemplo el código compartido por el usuario samlinux en GitHub [40].

Esta aplicación dará soporte a dos tipos de acciones:

- **CA actions**: comprende el registro e inscripción (register y enroll) de usuarios en la organización a los que no se les haya asignado una identidad durante el proceso de despliegue.
- **Ledger actions**: comprende las acciones de interacción con la red, es decir, operaciones de escritura y lectura en la ledger. Estas acciones serán accesibles a través de la API REST.

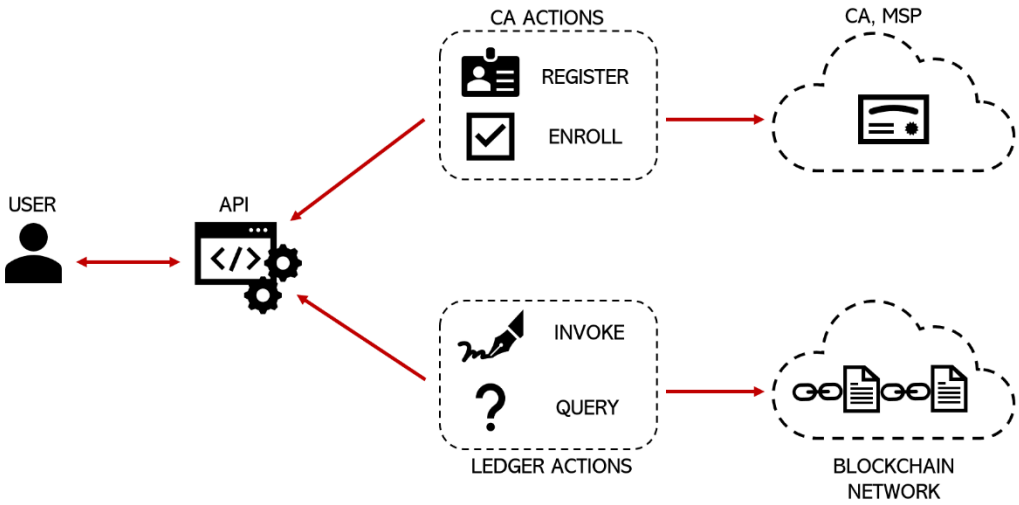

Figura 3-29. Acciones del SDK

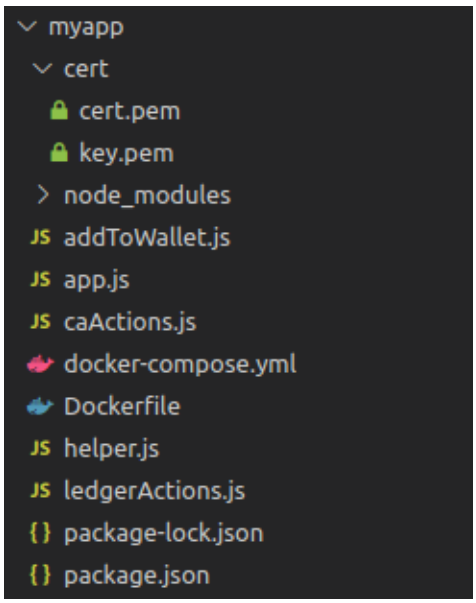

Figura 3-30. Estructura de directorios de la aplicación.

3.2.4.1. CA Actions

Además de los usuarios que se han registrado en una organización durante el despliegue, el SDK de Hyperledger permite añadir usuarios con la red en marcha. En caso de utilizar el segundo método, las identidades de los usuarios se almacenan en lo que se denomina "wallet". Hyperledger ofrece tres formas de implementar estos wallets:

- Sistema de ficheros.
- Sistema in-memory, utilizando la memoria de la aplicación.
- Uso de base de datos.

Para este proyecto, se ha escogido el sistema de ficheros.

De forma resumida, el proceso para dar de alta a un nuevo usuario usando el SDK es el siguiente:

1. Obtener el CCP (fichero del perfil de conexión) de la organización.
2. Crear una instancia de cliente de CA a partir de los datos del CCP.
3. Obtener el directorio del wallet o crearlo si no existe.
4. Obtener la identidad y usuario del administrador
5. Registrar y obtener la clave secreta, llamada secret, que servirá para autenticar al usuario durante el proceso de registro.
6. Inscribir utilizando la clave secreta para obtener los datos de la identidad del usuario
7. Construir una identidad X.509 con el certificado y la clave privada proporcionados en el paso anterior.
8. Guardar la nueva identidad en el wallet.

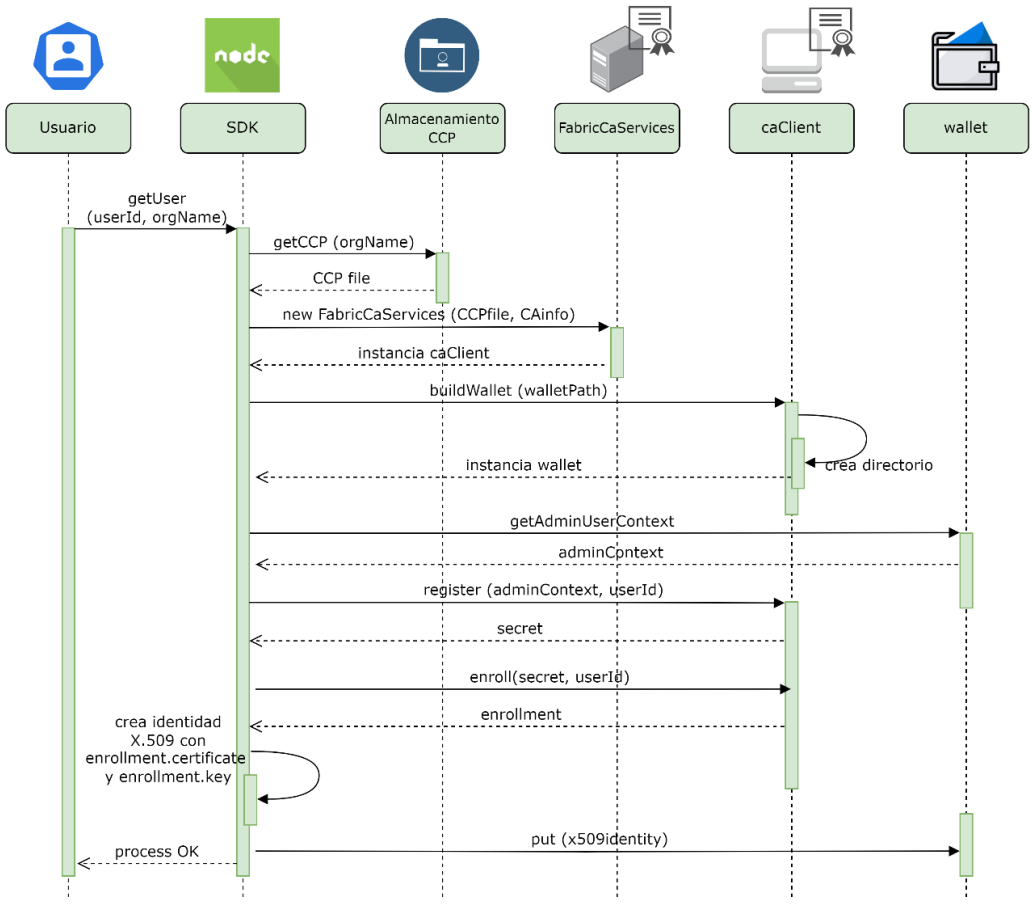

Figura 3-31. Secuencia de alta de un usuario en la red.

Cabe destacar que para dar de alta un usuario es necesario que exista la identidad de un administrador en el wallet. Es decir, primero se debe hacer el alta del administrador, de forma similar a la descrita (sin los pasos 4, 5 y 6 y utilizando el nombre y contraseña del administrador como secret).

Para este proyecto, se dispondrá de un número prefijado de usuarios (uno por cada organización). Una vez realizada el alta, los wallets tendrán un aspecto como el de la figura.

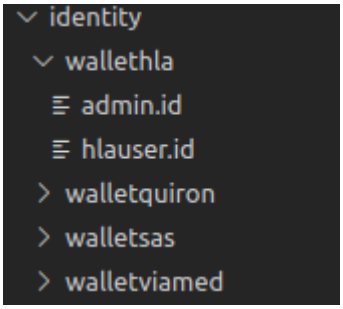

Figura 3-32. Wallets tras la creación de identidades.

3.2.4.2. LedgerActions

Para que los usuarios puedan interactuar con la red, la API será accesible a través de los siguientes endpoints:

- **/login** [método POST]: será el punto de autenticación. Cuando el cliente quiera interactuar con la red, deberá acceder primero a este endpoint, añadiendo en el cuerpo de la petición el nombre de la organización y el identificador del usuario. Se comprobará si dicho usuario existe en el wallet de la organización indicada.

  Si el usuario existe, se procederá a generar un JSON Web Token (JWT). Para ello, se utilizará una cabecera que contenga el algoritmo de firma y el periodo de validez del token, los datos del cuerpo de la petición y una clave. Una vez firmado el token, se devolverá al cliente. Este deberá añadirlo en la cabecera *Authorization* con el formato *Bearer <token>* de las próximas peticiones para mantener su sesión activa.

```
app.post('/login', async function (req, res) {
  console.log('Name: ' + orgname);
  var userid = req.body.userid;
  let isRegistered = await caActions.checkExists(userid, orgname);

  if (isRegistered) {
    const token = jwt.sign({ orgname, userid}, jwtKey, {
      algorithm: "HS256",
      expiresIn: jwtExpirySeconds,
    })
    console.log("token:", token);
    return res.status(200).json({response: token});
  } else {
    return res.status(403).json({ error: 'The user does not exist in the wallet' });
  }
});


app.use(function(req, res, next) {
  if (!req.headers.authorization) {
    return res.status(403).json({ error: 'No credentials sent!' });
  }
  next();
});
```

Figura 3-33. Endpoint /login.

Cada vez que se reciba una petición al resto de los endpoints, se comprobará, con una función middleware, que se incluye el token en esta cabecera.

- **/ledger/invoke** [método POST]: será el endpoint que permita crear assets y añadirlos a la ledger. Para cada petición recibida, se comprobará, usando la clave, que el token de la cabecera es válido. A partir de este token, se obtendrá el usuario que está creando el asset y su organización.

  En el cuerpo de la petición deben viajar los valores del nuevo asset (organización destino, propósito de uso, ID del paciente…). Además, en este punto se genera aleatoriamente el campo ID del asset y se obtendrá la fecha y hora para registrarla en el campo recorded.

  Una vez obtenidos los datos, se usará *ledgerActions.js*, parte de la aplicación encargada de evaluar y confirmar las transacciones.

```
app.post('/ledger/invoke', async function (req, res) {

  const tokenR =  req.headers.authorization.split(' ')[1];
  console.log("tokenR:", tokenR);
  var payload
  try {
    // Parse the JWT string and store the result in `payload`.
    payload = jwt.verify(tokenR, jwtKey)
  } catch (e) {
    if (e instanceof jwt.JsonWebTokenError) {
      // if the error thrown is because the JWT is unauthorized, return a 401 error
      return res.status(401).end()
    }
    // otherwise, return a bad request error
    return res.status(400).end()
  }
  console.log("payload:", payload)
  var identifier = Nanoid.nanoid();
  var orgname = payload.orgname;
  var userId = payload.userid;
  var idOrgS = req.body.idOrgS;
  var nameOrgS = req.body.nameOrgS;
  var idOrgD = req.body.idOrgD;
  var nameOrgD = req.body.nameOrgD;
  var validityDate = req.body.validityDate;
  var idPatient = req.body.idPatient;
  var purpose = req.body.purpose;

  let date_ob = new Date();
  let date = ("0" + date_ob.getDate()).slice(-2);
  let month = ("0" + (date_ob.getMonth() + 1)).slice(-2);
  let year = date_ob.getFullYear();
  let hours = date_ob.getHours();
  let minutes = date_ob.getMinutes();
  let seconds = date_ob.getSeconds();

  var recorded = year + "-" + month + "-" + date + "/" + hours + ":" + minutes + ":" + seconds;

    // Evaluate the specified transaction.
    const result = await ledgerActions.invoke(orgname, userId, identifier, idOrgS,
      nameOrgS, idOrgD, nameOrgD, validityDate, idPatient, purpose, recorded);
    console.log(`Transaction has been evaluated, result is: ${result.toString()}`);

    if (result.split(' ')[0] == 'Error.') {
      console.error(`Failed to evaluate transaction: ${result}`);
      res.status(500).json({error: result});
      process.exit(1);
    }

    else {
      res.status(200).json({response: result.toString()});
    }
});
```

Figura 3-34. Endpoint /ledger/invoke.

- **/ledger/query** [método GET]: será el endpoint a través del cual se realizarán lecturas de la ledger. De forma análoga al endpoint anterior, se verificará el token. En caso de éxito, se obtendrá el tipo de lectura deseado y el argumento necesario de los parámetros de la URL de la petición.

Una vez obtenidos los datos, se usará *ledgerActions.js*. El funcionamiento de esta parte de la aplicación puede resumirse en los siguientes pasos:

1. Obtener el CCP (fichero del perfil de conexión) de la organización.
2. Acceder al directorio del wallet de la organización y comprobar si existe la identidad.
3. Obtener una instancia de pasarela (Gateway) a través de la cual se podrá acceder a la red.
4. Obtener una instancia de la red a partir de la session y el nombre del canal.
5. Obtener una instancia del chaincode.
6. Llamar al chaincode para evaluar la transacción pasándole la función de lectura que se ha solicitado realizar.
7. Obtener el resultado y cerrar la conexión con la red.

```
exports.query = async function (orgName, UserId, fun, data) {
  try {
    let result;
    // build CCP
    const ccp = helper.buildCCP(orgName);

    // setup the wallet to hold the credentials of the application user
    const walletName = '/identity/wallet' + orgName;
    const walletPath = path.join(__dirname, walletName);
    const wallet = await helper.buildWallet(Wallets, walletPath);

    let identity = wallet.get(UserId);
    if (!identity) {
      result = 'Error. An identity for the user ' + UserId + ' does not exist in the wallet.';
      console.log(result);
      return result;
    }
    else {

      // Create a new gateway instance for interacting with the fabric network.
      const gateway = new Gateway();

      // setup the gateway instance
      await gateway.connect(ccp, {
        wallet,
        identity: UserId,
        discovery: { enabled: true, asLocalhost: false }
      });

      // Build a network instance based on the channel where the smart contract is deployed
      const network = await gateway.getNetwork(channelName);

      // Get the contract from the network.
      const contract = network.getContract(chaincodeName);

      if(fun === 'readBySource'){
        let sOrg = data
        result = await contract.evaluateTransaction('readBySource', sOrg);
        console.log(`${helper.prettyJSONString(result.toString())}`);
      }
      else if(fun === 'readByDestination'){
        let dOrg = data
        result = await contract.evaluateTransaction('readByDestination', dOrg);
        console.log(`${helper.prettyJSONString(result.toString())}`);
      }
      else if(fun === 'readByDate'){
        let date = data
        result = await contract.evaluateTransaction('readByDate', date);
        console.log(`${helper.prettyJSONString(result.toString())}`);
      }
      else if(fun === 'readByPatient'){
        let idPatient = data
        result = await contract.evaluateTransaction('readByPatient', idPatient);
        console.log(`${helper.prettyJSONString(result.toString())}`);
      }
      else if(fun === 'readAll'){
        result = await contract.evaluateTransaction('readAll');
        console.log(`${helper.prettyJSONString(result.toString())}`);
      }
      else {
        console.log('...')
        result = 'Error';
      }
      // disconnect form the network
      gateway.disconnect();

      return result.toString();;
    }
  }
  catch(e){
    result = 'Error. ' + e.message;
    return result;
  }
}
```

Figura 3-35. Función query de ledgerActions.js

3.2.4.3. Configuración y despliegue del servidor

Una vez desarrollados los endpoints, es necesario configurar la aplicación como servidor para que pueda atender las solicitudes de los clientes. Para este proyecto, se ha optado por un servidor HTTPS, usando un certificado autofirmado.

```
https.createServer({
  key: fs.readFileSync('cert/key.pem'),
  cert: fs.readFileSync('cert/cert.pem'),
  passphrase: 'tfgmgb16'
},app).listen(PORT, function(){
  console.log('Https server running on port ' + PORT);
});
```

Figura 3-36. Creación del servidor HTTPS.

En este punto, se podrá desplegar el servidor en un contenedor con la ayuda del fichero *docker-compose.yml*. Además, se dispone del fichero *Dockerfile*, que se encargará de iniciar la aplicación al levantar el contenedor. De esta forma, se podrá acceder al servidor con la URL https://localhost:4000.

3.2.4.4. Pruebas con Postman

Una vez el servidor esté en marcha, se pueden realizar pruebas para interactuar con la red. En este caso, se ha usado el cliente HTTP proporcionado por Postman.

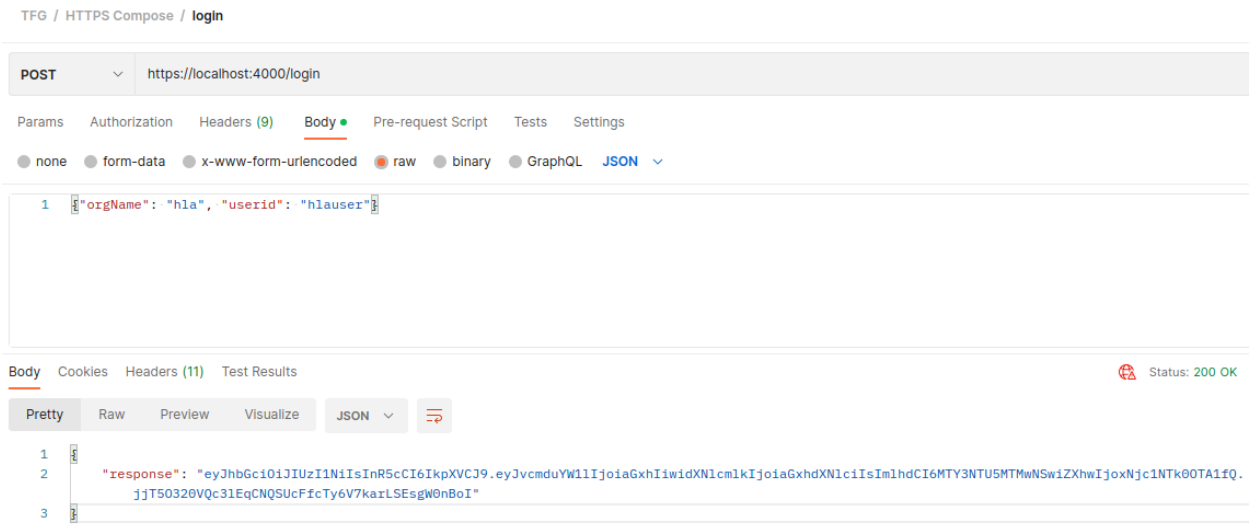

Figura 3-37. Ejemplo de login con Postman.

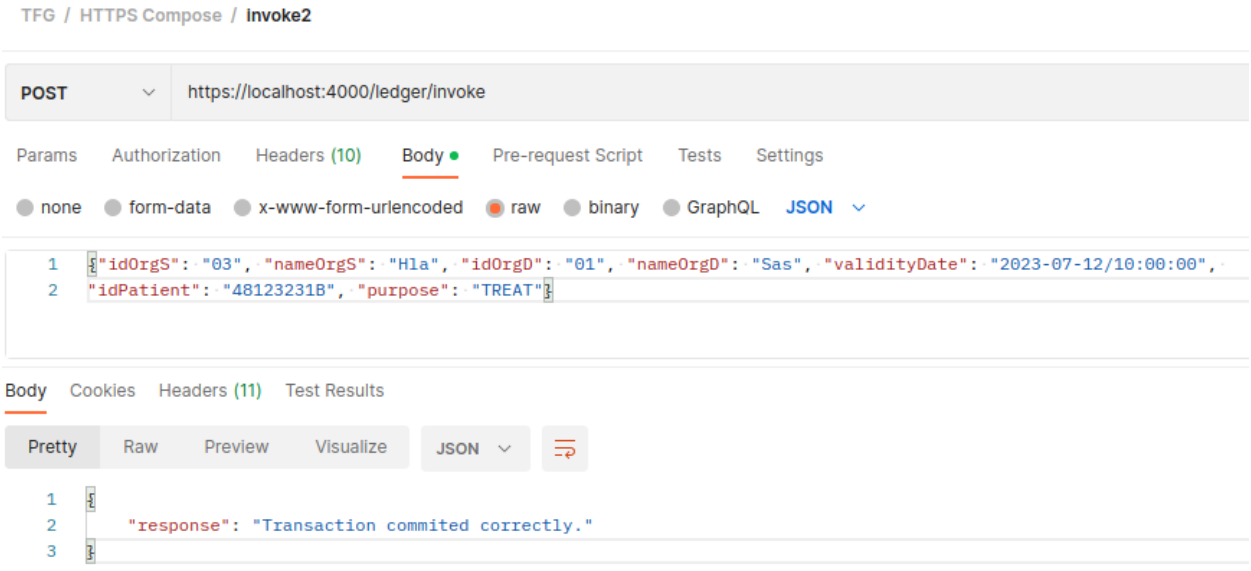

Figura 3-38. Ejemplo de invoke con Postman.

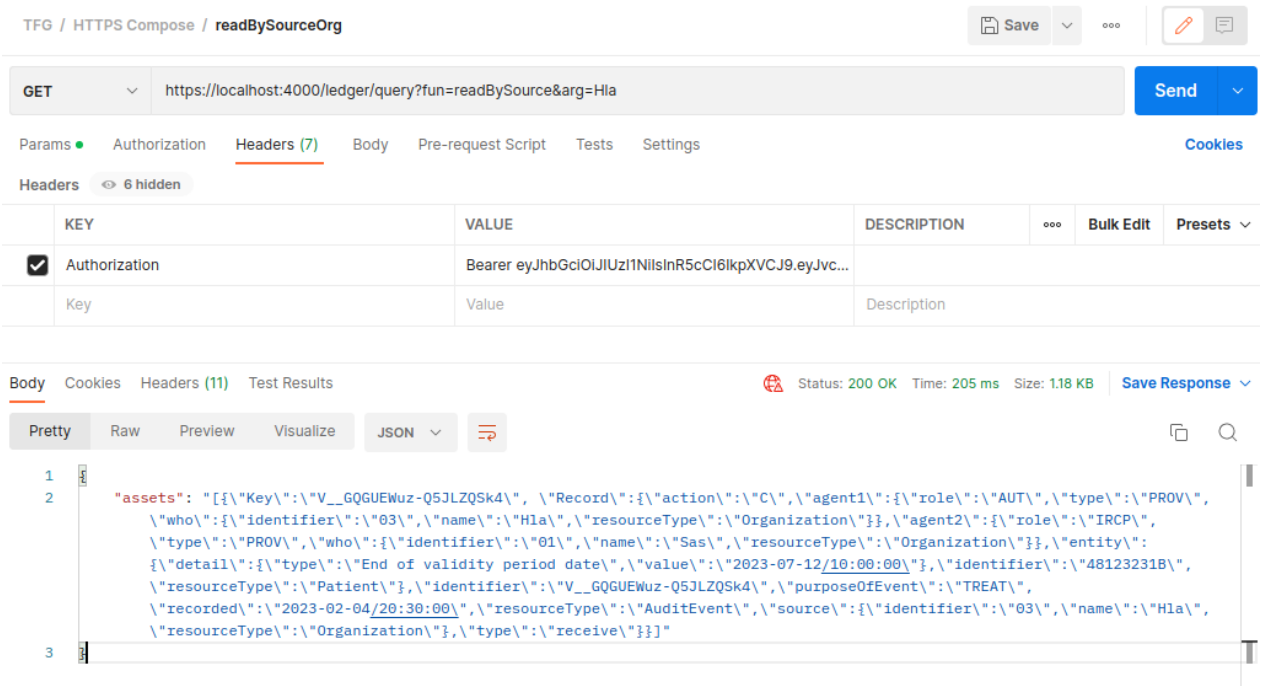

Figura 3-39. Ejemplo de query y cabecera authorization con Postman.

3.2.5. Interfaz gráfica de usuario

Para facilitar el uso de la aplicación se ha desarrollado una sencilla interfaz de usuario web, usando la librería de JavaScript React. Para el desarrollo, se han usado como base los repositorios compartidos en GitHub por el usuario Chhaileng Peng para la parte del consumo de una aplicación REST [41] y por el usuario Techomoro para desarrollar una web multi-página [42].

Esta solución estará estructurada en los siguientes componentes:

- Componente de navegación para desplazarse a la página principal desde las distintas secciones.
- Componente **Login** para introducir el nombre de la organización e identidad del usuario. Desde este componente, se llamará al endpoint */login* para obtener en JW Token.
- Componente **Home** (página principal) en el que aparecerá un listado de las funcionalidades que se podrán realizar.
- Componentes **ReadAll**, **ReadByDate**, **ReadByDestination**, **ReadByPatient** y **ReadBySource**, que permitirán elegir el argumento según la funcionalidad escogida y realizar una búsqueda filtrada en la ledger llamando al endpoint */ledger/query*.
- Componente **NewTransaction**, desde el que se podrá añadir una transacción a la ledger rellenando un formulario. Desde este componente, se llamará al endpoint */ledger/invoke*.

Las distintas acciones sobre la ledger que se pueden realizar desde la interfaz serán detalladas en la *Sección 3.3*.

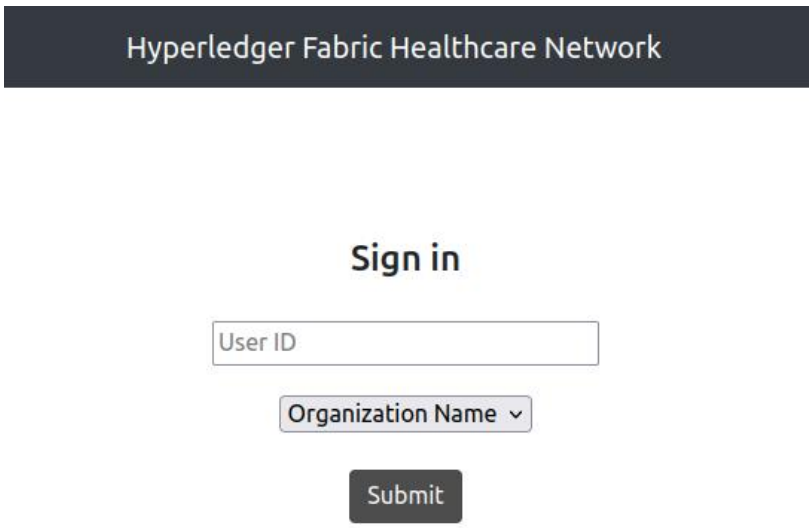

Figura 3-40. Página de inicio de sesión de la interfaz.

Una vez iniciada, la interfaz será accesible en la dirección http://172.16.17.11:3000. Para las distintas llamadas a la API, se ha utilizado la librería Axios, que actúa como cliente HTTP.

```
onFormSubmit(e) {
    e.preventDefault();
    const token = localStorage.getItem("token");
    if (token) {
        axios.defaults.headers.common["Authorization"] = 'Bearer ' + token;
        const url = 'https://127.0.0.1:4000/ledger/query?fun=readByDate&arg=' + this.state.arg;
        axios.get(url).then(res => {
        if (res.status === 200) {
            this.setState({assets: JSON.parse(res.data.assets)})
            this.setState({petitionMade: true})
        } else {
            alert('Something went wrong')
        }
    }).catch((error) => {
        if (error.response) {
            // The request was made and the server responded with a status code that falls out of the range of 2xx
            console.log(error.response.data);
            console.log(error.response.status);
            console.log(error.response.headers);
            alert('Server responded with status code != from 2xx.\nData: ' + error.response.data
            + '\nStatus: ' + error.response.status + '\nHeaders: ' + error.response.headers)
        } else if (error.request) {
            // The request was made but no response was received
            console.log(error.request);
            alert('The request was made but no response was received')
        } else {
            // Something happened in setting up the request that triggered an Error
            console.log('Error', error.message);
            alert('Request error. Message: ' + error.message)
        }
        console.log(error.config);
        });
    } else {
        alert('Token error');
        console.log('Token error');
    }
}
```

Figura 3-41. Ejemplo de petición con Axios para readByDate.

3.2.6. Configuración y despliegue con Docker Swarm

Para plantear un caso de uso más cercano a la realidad, se ha optado por diseñar un pequeño escenario en el que participen varios equipos, que puedan representar varias organizaciones. Para desplegar la solución del proyecto en este escenario, se ha usado la herramienta Docker Swarm.

Por simplicidad, se ha optado por incluir tres *hosts* en el escenario, situados en la misma subred, y agrupar organizaciones en ellos, ya que se ha considerado cantidad suficiente para mostrar el funcionamiento con Swarm.

El primero de los equipos actuará como manager del swarm y albergará a las organizaciones Orderer y SAS; el segundo, será de tipo worker y albergará a las organizaciones Quirón y HLA y el tercero, también de tipo worker, albergará a la organización Viamed.

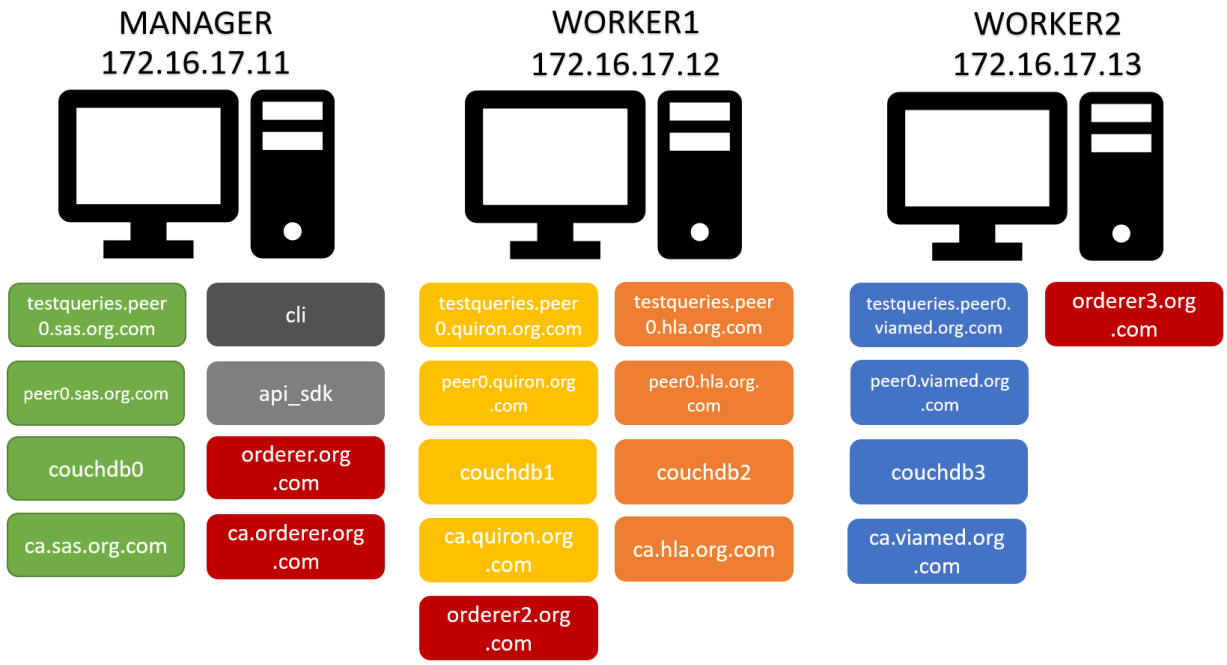

Figura 3-42. Nombres, direcciones IP y contenedores de los equipos del escenario.

Cambios en la configuración de Hyperledger Fabric

- **Autoridades de certificación**: en la configuración de las CA, debe añadirse la sección "*deploy*", en la que se indicará, entre otros, el equipo en el que se debe desplegar el contenedor correspondiente. También se añadirá el nombre del equipo o *hostname*.

  Además, se dividirá el fichero de configuración de las CA en 5 ficheros, uno para cada organización.

  Por último, se debe añadir el nombre de la autoridad de certificación (ca.<*nombreOrg*>.org.com) en la sección de equipos de las CSR

```
ca_viamed:
  deploy:
    replicas: 1
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 3
    placement:
      constraints:
        - node.hostname == worker2
  image: hyperledger/fabric-ca:latest
  hostname: ca.viamed.org.com
  labels:
    service: hyperledger-fabric
```

Figura 3-43. Cambios en la configuración de CAs para swarm.

- **Peers y CouchDB**: para el despliegue con swarm, se han configurado los peers y las CouchDB en el mismo fichero, habiendo un fichero para cada organización. Cabe destacar que, además de añadir la sección "*deploy*", se debe añadir la sección "extra_hosts", en la que se mapean el resto de los nombres del escenario a sus correspondientes direcciones IP.

```
extra_hosts:
  - "orderer.org.com:172.16.17.11"
  - "orderer2.org.com:172.16.17.12"
  - "orderer3.org.com:172.16.17.13"
  - "peer0.quiron.org.com:172.16.17.12"
  - "peer0.hla.org.com:172.16.17.12"
  - "peer0.viamed.org.com:172.16.17.13"
```

Figura 3-44. Sección extra_hosts en la configuración del swarm para el peer0 de la organización SAS.

Despliegue

El despliegue de la red seguirá la misma secuencia que para Docker compose. En este caso, en vez de utilizar un único script, se dispondrá de un script para cada organización. El levantamiento de los contenedores se realizará desde el manager, pero estos se levantarán en el equipo correspondiente debido a la sección *deploy*.

El registro e inscripción de los usuarios, la unión de los peers al canal y el despliegue del chaincode se realizará desde cada equipo. Por último, el contenedor de la aplicación se desplegará en el manager.

```
marta@manager:~/mynetwork/myapp$ docker service ps peer_hla_peer0_hla
ID             NAME                  IMAGE                                NODE      DESIRED STATE   CURRENT STATE
qgg5zm8cs1kf   peer_hla_peer0_hla.1  hyperledger/fabric-peer:latest       worker1   Running         Running 27 minutes ago
marta@manager:~/mynetwork/myapp$ docker service ps peer_viamed_peer0_viamed
ID             NAME                        IMAGE                                NODE      DESIRED STATE   CURRENT STATE
8ipidljjlu9j   peer_viamed_peer0_viamed.1  hyperledger/fabric-peer:latest       worker2   Running         Running 27 minutes
marta@manager:~/mynetwork/myapp$ docker service ps orderer_orderer2
ID             NAME                  IMAGE                                NODE      DESIRED STATE   CURRENT STATE
vzk32b9huivk   orderer_orderer2.1    hyperledger/fabric-orderer:latest    worker1   Running         Running 27 minutes ago
marta@manager:~/mynetwork/myapp$ docker service ps orderer_orderer3
ID             NAME                  IMAGE                                NODE      DESIRED STATE   CURRENT STATE
ohfvhg16fyq2   orderer_orderer3.1    hyperledger/fabric-orderer:latest    worker2   Running         Running 27 minutes ago
```

Figura 3-45. Comprobación de despliegue en los equipos correspondientes.

Uso

La interacción con la red no se ve alterada por este despliegue. Desde cada equipo, se puede acceder a la interfaz de usuario para realizar acciones sobre la ledger. Estos cambios se verán reflejados en la copia de la ledger que tiene cada peer, de forma que podrán también observarse desde la interfaz del resto de equipos

3.3. Pruebas finales

En esta sección se expondrá el proceso de interacción con la red a través de la interfaz gráfica de usuario.

3.3.1. Inicio de sesión

En primer lugar, al acceder a la interfaz se mostrará una ventana, en la que el usuario deberá introducir el nombre de la organización y el identificador de usuario.

En caso de que no coincidan o que no exista el usuario introducido, se mostrará un modal con un mensaje de error.

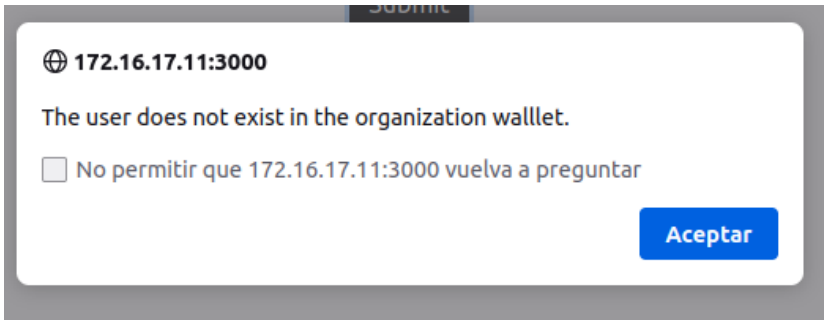

Figura 3-46. Error en login.

3.3.2. Pantalla principal

Una vez que se ha iniciado sesión con éxito, se mostrará una vista con las distintas interacciones que se pueden realizar. Estas son:

- Nueva transacción.
- Leer todas las transacciones.
- Buscar transacciones por organización de origen.
- Buscar transacciones por organización de destino.
- Buscar transacciones por fecha de fin del periodo de validez.
- Buscar transacciones por identificador de paciente.

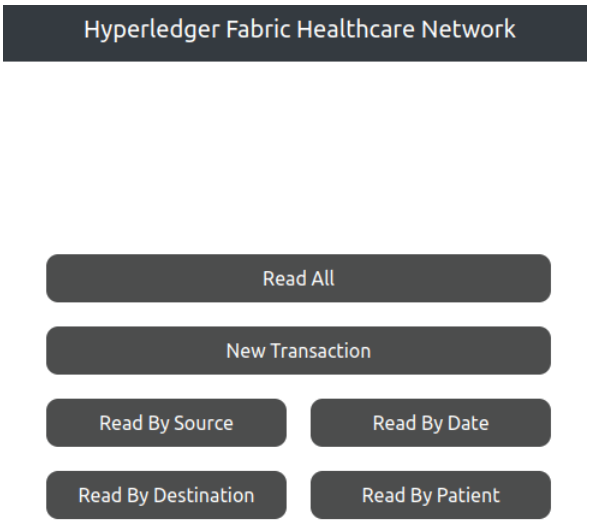

Figura 3-47. Vista principal de la interfaz.

3.3.3. Casos de uso

NewTransaction

Esta acción permitirá introducir transacciones en la blockchain. Para ello, se debe rellenar un formulario con la organización de destino, la fecha de fin del periodo de validez, el identificador del paciente y el propósito de uso de los datos. Una vez completados, al pulsar el botón *"Submit"*, se registrará la transacción, mostrando un modal de éxito al finalizar.

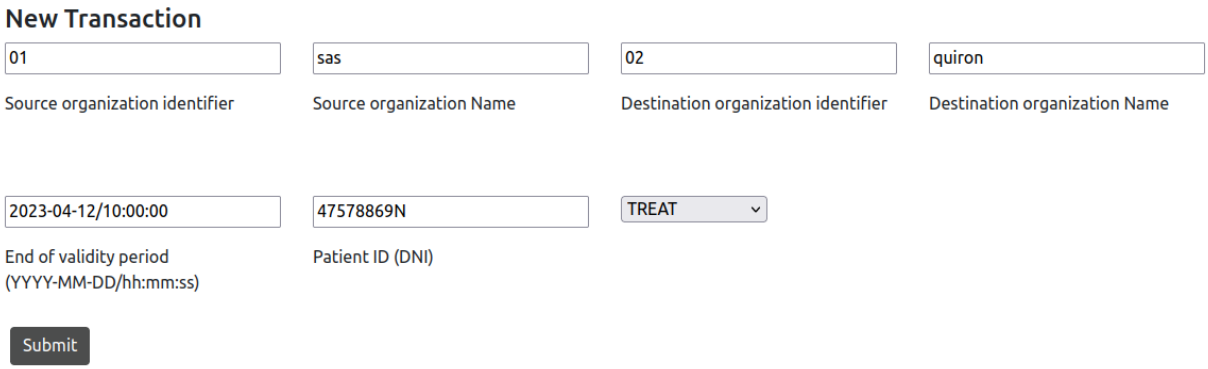

Figura 3-48. NewTransaction en interfaz.

ReadAll

Desde esta sección se podrá ver un resumen sobre todos los assets registrados en la ledger en forma de lista.

Figura 3-49. Resultado de ReadAll en la interfaz.

ReadBySource

Esta acción permitirá filtrar la búsqueda de assets en función de la organización de origen. Se debe introducir una organización en el campo de búsqueda y clicar en *"Search"*.

Si la búsqueda ha sido exitosa, se mostrarán los resultados de forma similar al caso anterior. Además, se podrán ver los assets completos en formato JSON pulsando el botón *"Show in JSON format"*.

En caso de no haber ningún resultado, aparecerá el mensaje *"There is not any asset that matches the search"*, permitiendo realizar de nuevo la búsqueda.

Figura 3-50. Resultado de ReadBySource en la interfaz.

ReadByDestination, ReadByDate y ReadByPatient

Estas acciones tienen un comportamiento análogo al del ReadBySource, permitiendo filtra la búsqueda por organización de destino, fecha hasta la que la organización de destino puede conservar los datos (fin del periodo de validez) e identificador del paciente.

Figura 3-51. Fragmento del resultado de una búsqueda en formato JSON.

4 CONCLUSIONES Y LÍNEAS FUTURAS

El desarrollo de este proyecto ha supuesto para mí un gran reto debido a la variedad, novedad y dificultad de las tecnologías empleadas. Sin embargo, todo este esfuerzo se ve compensado por la enorme cantidad de conocimientos que he adquirido. He podido trabajar con herramientas y tecnologías punteras y estoy segura de que este aprendizaje me ayuda a tener una visión de la situación de mercado actual y me será de gran utilidad en mis próximos pasos.

Una vez finalizado el proyecto, puedo realizar un análisis de las ventajas e inconvenientes de las tecnologías utilizadas, del grado de cumplimiento de los objetivos planteados y de las posibles mejoras que se podrían realizar.

4.1. Conclusiones sobre las tecnologías empleadas

En líneas generales, considero que el desarrollo de un mecanismo para compartir datos sanitarios entre distintas organizaciones sanitarias es algo realizable en un futuro próximo. De hecho, se trata de un tema de gran actualidad en España y así lo confirman las jornadas y reuniones que se están llevando a cabo entre el IDIS (Instituto para el Desarrollo y la Integración de la Sanidad) y otras entidades para buscar una solución a esta problemática [43].

La tecnología blockchain, en concreto las redes blockchain permisionadas, serían muy adecuadas para esta solución debido a sus condiciones de seguridad, trazabilidad y descentralización. Estas redes, unidas al estándar de FHIR, lograrían satisfacer las necesidades de interoperabilidad en los datos de los pacientes y aumentarían la eficiencia del sistema sanitario tanto público como privado.

En cuanto a Hyperledger Fabric, a pesar de que su modularidad es una gran ventaja, también hace que el desarrollo de una red sea una tarea compleja. Esto se debe a la gran cantidad de opciones de configuración que se manejan. En este proyecto, se ha desarrollado una red básica, pero, aun así, se ha tenido que realizar un gran estudio del funcionamiento de Fabric.

Sin embargo, es una plataforma muy potente y utilizada por entidades de gran importancia como IBM, Airbus o BNP Paribas. Además, es un proyecto de código abierto. Por ello, considero que es una opción de plataforma blockchain adecuada.

El mayor inconveniente que he detectado en Hyperledger Fabric es la gestión de identidades. El hecho de que el servidor de la API almacene las identidades de los usuarios en los wallets hace que la gestión de identidades sea centralizada. Además, solo se necesita conocer el nombre de un usuario registrado en la organización para poder interactuar con la red, ya que desde la API simplemente se comprueba que exista una identidad para dicho usuario. Esto puede suponer un problema de seguridad.

Además, el uso de contenedores Docker para desplegar la red de Hyperledger Fabric proporciona una solución flexible, escalable, portátil y fácil de implementar. A pesar de ello, aprender a manejar esta herramienta también es complicado y ha supuesto una alta inversión de tiempo.

Por último, considero que el uso de Golang para el desarrollo del chaincode, NodeJS para la API y React para la interfaz ha sido una buena elección. A pesar de que otras opciones habrían sido igualmente válidas, han sido lenguajes y entornos completamente nuevos para mí, por lo que añaden un gran valor a mi aprendizaje durante el proyecto.

En general, considero que las tecnologías empleadas han sido adecuadas, si bien requieren un alto grado de comprensión para poder trabajar con ellas.

4.2. Cumplimiento de objetivos y líneas futuras

Opino que el objetivo principal del proyecto, desarrollar un mecanismo para registrar flujos de información sanitaria entre organizaciones, ha sido alcanzado de forma satisfactoria.

Tras un estudio en profundidad de las tecnologías escogidas, se ha conseguido desplegar la red blockchain con Hyperledger Fabric con cuatro organizaciones y un peer en cada una de ellas, además del servicio de ordenamiento.

Además, se ha conseguido desarrollar la API con NodeJS y la interfaz con React. El haber sido capaz de llevar a cabo un proyecto donde la mayoría de las herramientas eran nuevas para mí me hace estar muy satisfecha con mi trabajo.

Sin embargo, considero que hay bastante margen de mejora para posibles líneas futuras. Las principales mejoras detectadas son las siguientes:

- **Gestión de identidades**: para solucionar el problema detectado, podrían usarse identificadores distribuidos (DIDs). DID es un modelo de identidad digital descentralizado que permite a los usuarios tener control sobre sus propios datos sin depender de una autoridad centralizada. En lugar de almacenar la información de identidad en una base de datos centralizada (el wallet), los usuarios pueden crear un DID, que es un identificador único y verificable. Los DIDs agrupan la información pública necesaria para autenticar a un usuario, como su certificado X.509. Así, al registrar a un usuario, este podría crear un DID con su certificado y usarlo para identificarse al interactuar con la red a través de la API.

  Actualmente, existen soluciones para integrar estos DIDs con Hyperledger Fabric. Este es el caso de TrustID, un módulo para Hyperledger desarrollado por el departamento de blockchain de Telefónica [44].

- **Configuración de la red**: otra posible mejora sería añadir más peers a cada organización. A pesar de que con un peer puede mostrarse el funcionamiento de la red blockchain, sería interesante estudiar y comprobar más en profundidad la interacción y comunicación entre los peers de una misma organización.

- **Pruebas de rendimiento**: una vez comprobado el funcionamiento de las transacciones en la red, habría sido interesante diseñar un plan de pruebas en el que se estudie el rendimiento y comportamiento de la red ante un número considerable de solicitudes de transacción simultáneas.

- **Interfaz gráfica de usuario**: se buscaba desarrollar una interfaz sencilla, ya que no era el objetivo principal del proyecto. Aun así, al ser una herramienta que permite al usuario interactuar con la red, tiene bastante importancia. La interfaz actual puede ser poco intuitiva, por lo que podrían realizarse mejoras de diseño.

REFERENCIAS

[1] «Historia clínica digital: ¿Qué es, beneficios y qué información contiene?,» 2021. [En línea]. Available: https://sentinel-monitoring.com/1403-2/.

[2] «Derecho a la Portabilidad,» 2022. [En línea]. Available: https://ayudaleyprotecciondatos.es/derecho-portabilidad/.

[3] «La Sanidad pública y el derecho a la portabilidad,» [En línea]. Available: https://www.redaccionmedica.com/secciones/privada/la-sanidad-publica-trunca-el-derecho-a-la-portabilidad-de-historia-clinica-1713.

[4] «Características tecnología blockchain,» 2019. [En línea]. Available: https://101blockchains.com/es/caracteristicas-tecnologia-blockchain/.

[5] «Metodología Scrum para el desarrollo de software ágil,» [En línea]. Available: https://www.eniun.com/metodologia-scrum-desarrollo-software-agil/.

[6] «¿Qué es Hyperledger Fabric?,» [En línea]. Available: https://aws.amazon.com/es/blockchain/what-is-hyperledger-fabric/.

[7] «¿Qué es un algoritmo de consenso en blockchain?,» [En línea]. Available: https://academy.binance.com/es/articles/what-is-a-blockchain-consensus-algorithm.

[8] «Los tipos de blockchain,» 2019. [En línea]. Available: https://abancainnova.com/opinion/los-tipos-de-blockchain-publica-privada-o-consorcio-explicados/.

[9] «FHIR, ese gran desconocido,» [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/fhir-ese-gran-desconocido.html.

[10] «¿Qué es HL7 FHIR?,» [En línea]. Available: https://www.tibco.com/es/reference-center/what-is-hl7-fhir.

[11] «Documentación de FHIR,» [En línea]. Available: https://hl7.org/fhir/.

[12] «Recurso AuditEvent,» [En línea]. Available: https://www.hl7.org/fhir/auditevent.html.

[13] «FHIR, tipo receive,» [En línea]. Available: https://terminology.hl7.org/3.1.0/CodeSystem-iso-21089-lifecycle.html#iso-21089-lifecycle-receive .

[14] «FHIR, acción execute,» [En línea]. Available: https://www.hl7.org/fhir/codesystem-audit-event-action.html#audit-event-action-E .

[15] «FHIR, tipo de datos instante,» [En línea]. Available: https://www.hl7.org/fhir/datatypes.html#instant .

[16] «FHIR, propósito de uso,» [En línea]. Available: https://terminology.hl7.org/3.1.0/ValueSet-v3-PurposeOfUse.html .

[17] «FHIR, rol PROV,» [En línea]. Available: https://terminology.hl7.org/3.1.0/CodeSystem-v3-RoleClass.html#v3-RoleClass-PROV .

[18] «FHIR, tipo de participación AUT,» [En línea]. Available: https://terminology.hl7.org/3.1.0/CodeSystem-v3-ParticipationType.html#v3-ParticipationType-AUT.

[19] «FHIR, tipo de participación IRCP,» [En línea]. Available: https://terminology.hl7.org/3.1.0/CodeSystem-v3-ParticipationType.html#v3-ParticipationType-IRCP.

[20] «Recurso Organization,» [En línea]. Available: https://www.hl7.org/fhir/organization.html .

[21] «Recurso Patient,» [En línea]. Available: https://www.hl7.org/fhir/patient.html .

[22] «Guía para pacientes y usuarios de la sanidad,» 2019. [En línea]. Available: https://www.aepd.es/sites/default/files/2019-12/guia-pacientes-usuarios-sanidad.pdf.

[23] «Reglamento General de Protección de datos, BOE,» 2016. [En línea]. Available: https://www.boe.es/doue/2016/119/L00001-00088.pdf.

[24] «Derechos RGPD: ¿Cuáles son?,» [En línea]. Available: https://protecciondatos-lopd.com/empresas/derechos-rgpd/.

[25] «Documentación Hyperledger Fabric - ¿Qué es Hyperledger Fabric?,» [En línea]. Available: https://hyperledger-fabric.readthedocs.io/es/latest/whatis.html.

[26] «Funcionamiento de RAFT,» [En línea]. Available: http://thesecretlivesofdata.com/raft/.

[27] «Documentación Hyperledger Fabric - RAFT,» [En línea]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/orderer/ordering_service.html#raft.

[28] «Documentación Hyperledger Fabric - Flujo de transacciones,» [En línea]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/txflow.html.

[29] «JWT,» [En línea]. Available: https://jwt.io/introduction .

[30] «Docker, ¿qué son los contenedores?,» [En línea]. Available: https://www.docker.com/resources/what-container/.

[31] «Docker compose,» [En línea]. Available: https://docs.docker.com/compose/ .

[32] «Docker swarm,» [En línea]. Available: https://www.nubersia.com/es/blog/kubernetes-vs-docker-swarm/ .

[33] «Node.JS,» [En línea]. Available: https://desarrolloweb.com/home/nodejs .

[34] «¿Qué es Express?,» [En línea]. Available: https://kinsta.com/es/base-de-conocimiento/que-es-express/ .

[35] «¿Qué es Go?,» [En línea]. Available: https://openwebinars.net/blog/que-es-go/ .

[36] «¿Qué es Postman?,» [En línea]. Available: https://www.encora.com/es/blog/como-realizar-pruebas-automatizadas-con-postman .

[37] «¿Qué es React?,» [En línea]. Available: https://tech.tribalyte.eu/blog-que-es-react .

[38] «Librería Axios: cliente HTTP para Javascript,» [En línea]. Available: https://desarrolloweb.com/articulos/axios-ajax-cliente-http-javascript.html.

[39] «What is CouchDB?,» [En línea]. Available: https://www.ibm.com/topics/couchdb.

[40] «How you can use Node.js as an application developer,» [En línea]. Available: https://github.com/samlinux/htsc/blob/master/meetup-120221/index.md.

[41] «Web client for FabCar Blockchain Network,» [En línea]. Available: https://github.com/chhaileng/fabcar-client.

[42] «React MultiPage Website,» [En línea]. Available: https://github.com/techomoro/ReactMultiPageWebsite.

[43] «La privada aboga por compartir datos de salud para mejorar la asistencia.,» [En línea]. Available: https://www.redaccionmedica.com/secciones/privada/la-privada-aboga-por-compartir-datos-de-salud-para-mejorar-la-asistencia-7563 .

[44] «TrustID, a new approach to fabric user identity management.,» [En línea]. Available: https://www.hyperledger.org/blog/2020/04/21/trustid-a-new-approach-to-fabric-user-identity-management.

[45] «Hyperledger Fabric v2.2.x setup,» [En línea]. Available: https://github.com/samlinux/htsc/blob/master/meetup-061020/index.md.

GLOSARIO

Assets: objeto digital formado por un conjunto de pares clave-valor que representa un valor u objeto del mundo real.

Blockchain: registro de transacciones formado por bloques de transacciones enlazados por hash.

Canal: capa de la red utilizada para la confidencialidad y aislamiento de los datos entre un conjunto de peers. Todos los peers de un canal tiene una copia de la misma ledger.

Chaincode: código que gestiona el acceso y la modificación de conjuntos de pares clave-valor del world state.

Gateway: intermediario entre las aplicaciones y las redes de Hyperledger para facilitar y hacer más segura la interacción con la red.

Hash: sucesión alfanumérica de longitud fija, que identifica a un conjunto de datos y que es generada mediante un algoritmo matemático.

Ledger: registro de estado y de cambios, formado por la blockchain y el world state.

MSP (Membership Service Provider): componente de la red que proporciona credenciales a los clientes y peers. Los clientes utilizan estas credenciales para autenticar sus transacciones y los peers para aprobar estas transacciones.

Organización: entidad de la red que posee y gestiona uno o varios peers y posee un MSP.

Peer: entidad de la red que mantiene una copia de la ledger y ejecuta chaincodes para realizar operaciones de lectura/escritura en la ledger.

Red peer-to-peer: red en la que no hay un punto central de conexión o control, y donde las partes actúan de forma autónoma respondiendo a un protocolo de comunicaciones y consenso común.

SDK: conjunto de librerías y para desarrollar aplicaciones de chaincode. Proporciona APIs para procesar transacciones y gestionar servicios de membresía.

Servicio de ordenamiento: conjunto de nodos que ordenan transacciones en bloques y los distribuyen a los peers conectados para su validación y confirmación.

World state: representa el estado actual, los últimos valores registrados en la blockchain.

ANEXO A: MANUAL DE INSTALACIÓN Y DESPLIEGUE CON DOCKER COMPOSE

En esta sección se explicará qué prerrequisitos de instalación se deben cumplir para desplegar el proyecto en un solo equipo con Docker Compose. Además, se incluirán las instrucciones para desplegar la red usando los scripts incluidos en el código.

Cabe destacar que, para este proyecto, se ha utilizado el sistema operativo Ubuntu en su versión 20.0.4 LTS.

A.1. Prerrequisitos

A continuación, se indican las herramientas necesarias para la ejecución y despliegue del proyecto [45].

A.1.1. Instalar Docker

```
# configurar el repositorio
sudo apt install \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  software-properties-common

# añadir la clave GPG oficial de Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -


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

# instalar el motor de Docker
apt update
apt install docker-ce docker-ce-cli containerd.io

# comporbar la version de Docker
docker --version
```

En este proyecto se ha instalado la versión 20.10.14.

A.1.2. Instalar Docker Compose

```
# instalar Docker-compose
curl -L "https://github.com/docker/compose/releases/download/1.27.4/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# dar permiso de ejecución al binario
sudo chmod +x /usr/local/bin/docker-compose

# comprobar la version de Docker-compose
docker-compose --version
```

En este proyecto se ha instalado la versión 1.27.4.

A.1.3. Instalar Golang

```
# descargar y descomprimir go
wget -c https://dl.google.com/go/go1.14.9.linux-amd64.tar.gz -O - | tar -xz -C
/usr/local

# añadir el binario de go a la ruta
echo 'export PATH="$PATH:/usr/local/go/bin:/root/fabric/fabric-samples/bin"' >>
$HOME/.profile

# hacer que la vble de entorno GOPATH apunte al directorio base de fabric
echo 'export GOPATH="$HOME/fabric"' >> $HOME/.profile

# recargar el perfil
source $HOME/.profile

# comprobar la version de go
go version

# comprobar las variables
printenv | grep PATH
```

En este proyecto se ha instalado la versión 1.14.9.

A.1.4. Instalar Node.js

```
# añadir PPA desde NodeSource
curl -sL https://deb.nodesource.com/setup_16.x -o nodesource_setup.sh

# ejecutar el script de instalación
. nodesource_setup.sh

# instalar Node.js
apt-get install -y nodejs

# comprobar la versión
node -v
```

En este proyecto se ha instalado la versión 16.4.2.

A.1.5. Instalar Hyperledger Fabric

Con estos comandos, se clona el repositorio de Hyperledger Fabric, se instalan los binarios y se descargan las imágenes Docker.

```
curl -sSL https://bit.ly/2ysbOFE | bash -s -- 2.4.3

# comprobar las imágenes descargadas
docker images

# comprobar el comando peer y la versión de Hyperledger Fabric
peer version
```

En este proyecto se ha instalado la versión 2.4.3.

A.2. Despliegue

```
#crear la red Docker
docker network create mynetwork
# crear Cas, peers y canal (desde el directorio donde se ubique network.sh)
./network.sh up createChannel -s couchdb -ca
# desplegar el chaincode
./network.sh deployCC -c mychannel -ccn testqueries -ccp ../chaincode-go -ccl go
# lanzar el servidor de la API
cd ../myapp # Directorio donde se ubique el Dockerfile
./build.sh
docker-compose up -d
docker exec -it <nombre_contenedor_api> node createWallet.js
# crear admins y usuarios y añadir a los wallets
docker exec -it <nombre_contenedor_api> node addToWallet.js
# arrancar la interfaz web
cd ../mynetwork-ui
npm start
# para detener la red
./network.sh down
docker system prune –volumes -f
```

ANEXO B: MANUAL DE DESPLIEGUE CON DOCKER COMPOSE

En esta sección se incluirán las instrucciones para desplegar la red con Docker Swarm usando los scripts incluidos en el código.

Cabe destacar que se parte un escenario como el de la *Figura 3-42. Nombres, direcciones IP y contenedores de los equipos del escenario.* Por tanto, para realizar el despliegue en un escenario distinto, se deben adaptar los scripts en función de los elementos que se deseen desplegar en cada equipo.

B.1. Despliegue

```
# 1º: CREAR SWARM: crea el swarm y la red mynetwork (Desde equipo manager)
./networkGeneral.sh createSwarm

# 2º: UNIR EQUIPOS AL SWARM (Desde equipos worker1 y worker2)
docker swarm join --token <token_obtenido_al_crear_el_swarm> 172.16.17.11:2377

# 3º: CREAR CAs: despliega los contenedores de las CAs (Se realiza desde manager, cada
# CA será desplegada en el equipo configurado en los ficheros yaml)
./networkGeneral.sh createCAOrderer
./networkGeneral.sh createCASas
./networkGeneral.sh createCAQuiron
./networkGeneral.sh createCAHla
./networkGeneral.sh createCAViamed

# 4º REGISTER AND ENROLL: registra e inscribe un usuario y administrador para cada
# organización
# Desde equipo manager
./networkSas.sh registerEnroll
./networkOrderer.sh registerEnroll

# Desde equipo worker1
./networkQuiron.sh registerEnroll
./networkHla.sh registerEnroll

# Desde equipo worker2
./networkViamed.sh registerEnroll

# 5º COPIAR MATERIAL CRIPTOGRÁFICO: Es necesario que el material criptográfico de las
# organizaciones esté en el equipo desde el que se vaya a crear el bloque génesis, por
# lo que se copia al equipo manager desde los equipos worker1 y worker2. Se ha escogido
# el protocolo SFTP para transferir este material (Desde equipo manager)
    cd organizations/fabric-ca
    sftp marta@worker1
    get -Pr /home/marta/mynetwork/mnw/organizations/fabric-ca/quiron
    get -Pr /home/marta/mynetwork/mnw/organizations/fabric-ca/hla
    exit

    cd ../peerOrganizations
    sftp marta@worker1
```

```
    get -Pr /home/marta/mynetwork/mnw/organizations/peerOrganizations/quiron.org.com
    get -Pr /home/marta/mynetwork/mnw/organizations/peerOrganizations/hla.org.com
    exit

    cd ../fabric-ca
    sftp marta@worker2
    get -Pr /home/marta/mynetwork/mnw/organizations/fabric-ca/viamed
    exit

    cd ../peerOrganizations
    sftp marta@worker2
    get -Pr /home/marta/mynetwork/mnw/organizations/peerOrganizations/viamed.org.com
    exit

    cd ../..
# Además, es necesario que los equipos workers tengan el material criptográfico de los
# nodos orderers para poder desplegar en ellos los contenedores de estos nodos
# Desde equipos worker1 y worker2
    cd organizations
    sftp marta@manager
    get -Pr /home/marta/mynetwork/mnw/organizations/ordererOrganizations
    exit

    cd fabric-ca
    sftp marta@manager
    get -Pr /home/marta/mynetwork/mnw/organizations/fabric-ca/ordererOrg
    exit

    cd ../..

# 6º GENERAR CCPs: Los perfiles de conexión se generan en el manager y se almacenan en
# el directorio de la aplicación, ya que serán necesarios para crear los wallets y las
# identidades de los usuarios (Desde equipo manager)
./networkGeneral.sh generateCCP

# 7º DESPLEGAR SERVICIOS DE PEERS Y COUCHDBS (Se realiza desde manager, cada contenedor
# será desplegado en el equipo configurado en los ficheros yaml)
./networkGeneral.sh deployComponentsOrderer
./networkGeneral.sh deployComponentsSas
./networkGeneral.sh deployComponentsQuiron
./networkGeneral.sh deployComponentsHla
./networkGeneral.sh deployComponentsViamed

# 8º CREAR BLOQUE GÉNESIS (Desde equipo manager, ya que es desde donde se creará el #
# canal)
./networkGeneral.sh createGenesis

# 9º CREAR CANAL (Desde equipo manager)
./networkGeneral.sh createChannel

# 10º UNIR PEERS AL CANAL (Desde equipo manager, ya que es donde está desplegado el
# contenedor cli)
./networkGeneral.sh joinChannelSas
./networkGeneral.sh joinChannelQuiron
./networkGeneral.sh joinChannelHla
./networkGeneral.sh joinChannelViamed
```

```
# Cambiar línea 83 de createChannel.sh en equipo manager ya que el sufijo
# del servicio varía en cada ejecución (ejecutar docker ps, coger peer_sas_cli)

# 11º ESTABLECER ANCHOR PEERS (Desde equipo manager)
./networkGeneral.sh setAnchorPeerSas
./networkGeneral.sh setAnchorPeerQuiron
./networkGeneral.sh setAnchorPeerHla
./networkGeneral.sh setAnchorPeerViamed

# 12º EMPAQUETAR CHAINCODE (Desde equipo manager)
./networkGeneral.sh vendorAndPackageCC

# Además, es censario que todas las organizaciones tengan el paquete del chaincode
# para poder desplegarlo
# Desde equipos workers
    sftp marta@manager
    get -P /home/marta/mynetwork/mnw/smartcontract.tar.gz
    exit

# 13º INSTALAR CHAINCODE
# Desde equipo Manager
./networkSas.sh installCC

# Desde equipo worker1
./networkQuiron.sh installCC
./networkHla.sh installCC

# Desde equipo worker2
./networkViamed.sh installCC

# 14º COMPROBAR Y APROBAR INSTALACIÓN (Desde equipo Manager)
./networkGeneral.sh queryInstalledApproveSas
./networkGeneral.sh queryInstalledApproveQuiron
./networkGeneral.sh queryInstalledApproveHla
./networkGeneral.sh queryInstalledApproveViamed

# 15º COMPROBAR QUE EL CHAINCODE ESTÁ LISTO PARA CONFIRMAR checkReadiness (Desde equipo
# Manager)
./networkGeneral.sh checkReadinessSas
./networkGeneral.sh checkReadinessQuiron
./networkGeneral.sh checkReadinessHla
./networkGeneral.sh checkReadinessViamed

# 16º CONFIRMAR LA DEFINICIÓN DEL CHAINCODE commitDefinition (Desde equipo Manager)
./networkGeneral.sh commitDefinition

# 17º COMPROBAR LA CONFIRMACIÓN DEL CHAINCODE querycommited (Desde equipo Manager)
./networkGeneral.sh queryCommitted

# 18º DESPLEGAR api (Desde equipo Manager)
cd ../myapp
# Construir y arrancar el servidor
./build.sh
./run.sh
# Crear wallets e identidades de administradores y usuarios (ejecutar dentro del
# contenedor)
docker exec -it api_sdk_api node createWallet.js
docker exec -it api_sdk_api node addToWalletAdmin.js
docker exec -it api_sdk_api node addToWalletUser.js
```

```
# 19º Arrancar interfaz
cd ../mynetwork-ui
npm start

# PARA DETENER LA RED
# Desde equipo Manager
./networkSas.sh networkDown
./networkOrderer.sh networkDown

# Desde equipo worker1
./networkQuiron.sh networkDown
./networkHla.sh networkDown

# Desde equipo worker2
./networkViamed.sh networkDown
```