Desarrollo de una aplicación descentralizada con Blockchain: Dapp para el acceso y modificación de información sensible.

Manuel Suárez Taboada
Máster en Ingeniería Informática

Félix Freitag

8 de enero de 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	Desarrollo de una aplicación descentralizada con Blockchain: Dapp para el acceso y modificación de información sensible.
Nombre del autor:	Manuel Suárez Taboada
Nombre del consultor:	Félix Freitag
Fecha de entrega (mm/aaaa):	01/2020
Área del Trabajo Final:	Sistemas Distribuidos
Titulación:	*Máster en Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

El presente trabajo tiene como objetivo el estudio de la tecnología Blockchain con el fin de asimilar los conceptos y componentes de la misma. Además, se realiza una comparativa entre diferentes soluciones de tipo Blockchain, con el propósito de seleccionar una de ellas, para llevar a cabo una prueba de concepto. Esta prueba de concepto se basa en la implementación de una Aplicación Descentralizada (Dapp) para el acceso y modificación de información sensible.

En una segunda parte del trabajo, se configura y despliega una red de Blockchain empleando la tecnología seleccionada y se desarrolla una Dapp que hace uso de la misma.

Como resultado final, obtenemos un sistema que se corresponde con una aproximación sobre el uso de Blockchain en un ámbito en el que se requiere la gestión de datos sensibles. Esto nos permite validar su uso potencial en ámbitos con este tipo de necesidades.

Abstract (in English, 250 words or less):

The present project has the purpose to study the Blockchain technology. The objective is to assimilate the concepts and the components of this technology. Besides it includes a comparison between different solutions in order to choose the most appropriate for a proof of concept. This proof of concept is based in a Decentralized Application (Dapp) development to manage sensitive information.

In the second part of the Project, I configure and deploy the Blockchain network using the Blockchain solution selected previously and I develop a Dapp that uses it.

As a result we obtain a system that corresponds to an approach on the use of Blockchain in an area where sensitive data management is required. This allows us to validate its potential use in cases with this type of needs.

Palabras clave (entre 4 y 8):

Blockchain, Hyperledger, Fabric, Aplicación descentralizada, React.js

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

Durante los últimos años son muchas las iniciativas, impulsadas tanto desde los sectores privados como públicos, alrededor de Blockchain. Estas iniciativas responden al auge de sistemas que trabajan de un modo completamente descentralizado y autónomo. Se trata de sistemas en los que no existen elementos dentro de la red que hagan las funciones de servidor. Tampoco existen jerarquías entre los nodos, cada uno de ellos mantiene una copia de los datos y de la lógica de negocio.

Un ejemplo de estas iniciativas es Alastria, la cual es una organización Española sin ánimo de lucro que nace en 2017. Alastria es impulsada por organizaciones multisectoriales con el objetivo de crear una red pública regulada basada en Blockchain.

Todos estos sistemas e iniciativas se basan en Blockchain. Esta tecnología, conocida también como "*Cadena de bloques*", se podría definir como una base de datos distribuida y segura. La información se almacena en bloques, los cuales se enlazan con su predecesor y se emplea criptografía de clave pública y funciones de tipo hash para dotar de seguridad al sistema.

De entre todos los proyectos que emplean Blockchain tenemos que mencionar Bitcoin, ya que se trata de uno de los primeros sistemas descentralizados, y a pesar de que existieron otras criptomonedas y diseños anteriores, es considerada la primera criptomoneda que

empezó a operar en 2009. Bitcoin surge a raíz de un artículo publicado en 2008 bajo el seudónimo de *Satoshi Nakamoto* y cuyo título es "*Bitcoin: A Peer-to-Peer Electronic Cash System*"[1].

Actualmente son muchas las organizaciones que están explorando el uso de Blockchain para sus intereses, lo que están dando como resultado numerosos proyectos alrededor de esta tecnología.

1.2. Objetivos del Trabajo

El objetivo del presente trabajo es el de realizar un estudio de campo sobre los elementos que componen un sistema descentralizado que emplea Blockchain. Mediante esta investigación, se pretende asimilar todos los conceptos necesarios para disponer de una perspectiva que nos permita entender el paradigma de los sistemas descentralizados y todas las posibilidades que ofrecen.

Una vez finalizado el trabajo de investigación, se desarrollará una aplicación descentralizada, *Dapp*, que nos permita el acceso y la modificación de información sensible mediante un sistema de permisos. Entendiendo como información sensible aquellos datos pertenecientes a un usuario que son privados, como informes clínicos o datos bancarios. Podemos destacar los siguientes puntos como objetivos del trabajo:

Investigación sobre la tecnología blockchain: Estudio de la tecnología Blockchain y de su funcionamiento. Se investigaran las diferentes redes existentes en la actualidad, de cara a tomar una decisión respecto a la red en la que se desplegará la Dapp a desarrollar.

Asimilar los conceptos introducidos por Blockchain: Entender el paradigma de las aplicaciones descentralizadas de modo que, nos sea

posible visualizar el abanico de posibilidades que nos ofrecen estos sistemas.

Diseño e implementación de la aplicación descentralizada: Realización del diseño y desarrollo de la Dapp con las tecnologías señaladas cubriendo un caso de uso real.

1.3. Enfoque y método seguido

El enfoque a seguir consiste en la realización de un estudio previo de las tecnologías a emplear en el proyecto, prestando especial atención a la tecnología Blockchain, ya que se trata de una tecnología muy reciente y en la que no se cuenta con experiencia previa. Por otro lado, este estudio nos permitirá seleccionar el tipo de tecnología Blockchain con el fin de alcanzar los objetivos propuestos.

Una vez seleccionada la tecnología Blockchain, se plantea partir de una solución existente y adaptar la misma a nuestros requerimientos, en lugar de realizar un desarrollo desde cero. Con este enfoque se pretende minimizar el riesgo de no contar con experiencia en este tipo de tecnologías y poder obtener un producto lo antes posible.

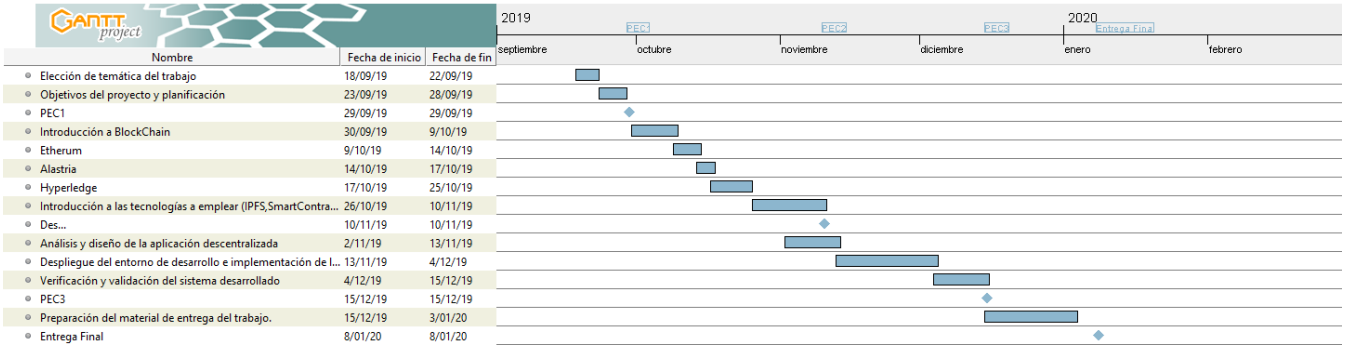
1.4. Planificación del Trabajo

Se muestran a continuación cada una de las tareas e hitos planificados:

- *Elección de la temática del trabajo*: Estudio de las diferentes alternativas para la realización del trabajo.
- *Objetivos del proyecto y planificación*: Definición de los objetivos del proyecto y de la planificación temporal de las tareas que lo componen.
- **PEC1**: Entrega parcial que incluye los objetivos del proyecto y la planificación temporal.
- *Introducción a Blockchain*: Estudio de los componentes y funcionamiento de la tecnología Blockchain.

- *Estudio de las diferentes redes públicas y privadas*: Ethereum, Alastria e Hyperledger.
- *Introducción a las tecnologías a emplear*: Estudio de las tecnologías a utilizar en el proyecto (IPFS, React.js, Smart Contracts)
- **PEC2**: Entrega parcial que consiste en un informe con los resultados de la investigación teórica.
- *Análisis y diseño de la aplicación descentralizada*: Definición de los diferentes componentes del sistema y su interacción entre ellos.
- *Despliegue del entorno de desarrollo e implementación de la aplicación descentralizada*: Instalación y configuración del entorno de trabajo y desarrollo de la Dapp.
- *Verificación y validación del sistema desarrollado*: Realización de pruebas de verificación y validación, con el objetivo de asegurar que el sistema funciona sin errores y que responde a los objetivos planteados.
- **PEC3**: Entrega parcial que incluye la Dapp desarrollada.
- *Documentación*: Preparación de los entregables del trabajo.
- **Entrega Final**: Incluye la presentación de diapositivas y la exposición en video.

A continuación se muestra el diagrama de Gantt con la temporalización del proyecto:



Nombre	Fecha de inicio	Fecha de fin
Elección de temática del trabajo	18/09/19	22/09/19
Objetivos del proyecto y planificación	23/09/19	28/09/19
PEC1	29/09/19	29/09/19
Introducción a BlockChain	30/09/19	9/10/19
Etherum	9/10/19	14/10/19
Alastria	14/10/19	17/10/19
Hyperledge	17/10/19	25/10/19
Introducción a las tecnologías a emplear (IPFS,SmartContra...	26/10/19	10/11/19
Des...	10/11/19	10/11/19
Análisis y diseño de la aplicación descentralizada	2/11/19	13/11/19
Despliegue del entorno de desarrollo e implementación de l...	13/11/19	4/12/19
Verificación y validación del sistema desarrollado	4/12/19	15/12/19
PEC3	15/12/19	15/12/19
Preparación del material de entrega del trabajo.	15/12/19	3/01/20
Entrega Final	8/01/20	8/01/20

Figura 1: *Diagrama de Gantt*

En la siguiente tabla de hitos se puede apreciar, además de las fechas de inicio y fin de cada una de las tareas, la carga de trabajo de cada una de ellas expresada en horas:

ID	Nombre	Fecha de inicio	Fecha de fin	Duración(Horas)
1	Elección de temática del trabajo	18/09/2019	22/09/2019	8
2	Objetivos del proyecto y planificación	23/09/2019	28/09/2019	12
3	PEC1	29/09/2019	29/09/2019	0
4	Introducción a Blockchain	30/09/2019	09/10/2019	20
5	Ethereum	09/10/2019	14/10/2019	22
6	Alastria	14/10/2019	17/10/2019	16
7	Hyperledger	17/10/2019	25/10/2019	22
8	Introducción a las tecnologías a emplear (IPFS,SmartContracts, React.js)	26/10/2019	10/11/2019	40
9	PEC2	10/11/2019	10/11/2019	0
10	Análisis y diseño de la aplicación descentralizada	02/11/2019	13/11/2019	30
11	Despliegue del entorno de desarrollo e implementación de la aplicación descentralizada	13/11/2019	04/12/2019	50
12	Verificación y validación del sistema desarrollado	04/12/2019	15/12/2019	40
13	PEC3	15/12/2019	15/12/2019	0
14	Preparación del material de entrega del trabajo.	15/12/2019	03/01/2020	40
15	Entrega Final	08/01/2020	08/01/2020	0
			Total	300

***Figura 2**: Tabla de Hitos*

1.5. Breve sumario de productos obtenidos

- *Memoria*: Se trata del presente documento. Engloba toda la documentación técnica y teórica del mismo, así como información acerca de la planificación.
- *PAC's de seguimiento*: Conjunto de informes de seguimiento que contienen información relativa a la evolución del proyecto a lo largo de tiempo, su avance respecto a la planificación, problemas encontrados y toma de decisiones.
- *SmartContracts*: Conjunto de programas que se encargan de modelar el comportamiento de la Blockchain.
- *Configuración de la Blockchain*: Ficheros de configuración y scripts que conforman la Blockchain.

- *Aplicación descentralizada (Dapp)*: Parte correspondiente al Front-end donde se encuentran las funcionalidades desarrolladas y que serán usadas por los usuarios finales.

1.6. Breve descripción de los otros capítulos de la memoria

A continuación se presentan los capítulos de los que consta la memoria:

- *Introducción a Blockchain:* Se realiza una introducción de la tecnología Blockchain y su origen.
- *Dapp para el acceso y modificación de información sensible:* Se presenta el análisis y diseño del sistema.
- *Desarrollo y configuración de sistema:* Se presenta los detalles del desarrollo y configuración llevados a cabo, así como los elementos del prototipo obtenido y una valoración de los mismos.
- *Conclusiones:* Apartado en el que se incluye una reflexión sobre la consecución de objetivos, los conocimientos adquiridos y la evolución de trabajo.
- *Bibliografía:* Conjunto de citas y referencias bibliográficas utilizadas a lo largo de todo el trabajo.
- *Anexos:* Apartado en el que incluye información complementaría sobre el trabajo. Se incluyen los manuales técnicos generados.

2. Introducción a Blockchain

2.1. Historia

Nos tenemos que remontar al año 1991 para encuadrar los conceptos en los que se basa la tecnología Blockchain. En el artículo "*How to Time-Stamp a Digital Document*"[2], escrito por Stuart Haber y W. Scott Stornetta, se plantea la problemática de certificar en qué momento se crea o modifica un documento por última vez, independientemente del medio en el que se almacena. El objetivo es que no sea posible, para un usuario, certificar que un documento es de una fecha anterior o posterior a la que realmente es. Como requisitos adicionales de los procedimientos definidos, Haber y Scott, plantean la necesidad de mantener la privacidad de los documentos y que no sea necesario almacenar los registros por parte de la entidad que realiza la certificación.

En el artículo se parte de una solución relativamente sencilla, "*digital safety-deposit box*", sobre la que se plantean ciertos requerimientos y problemas a los que es necesario darles solución. Esta solución consiste en la certificación por parte de un TSS (*Time Stamp Service*) de un documento remitido por un cliente. El TSS procede a registrar la fecha y hora en la que el documento es remitido y a almacenar una copia del mismo. En caso de que sea puesto en entredicho, es posible comparar el documento con la copia almacenada por el TSS.

A partir de esta solución se plantean una serie de puntos de mejora en relación a los siguientes aspectos:

- *Privacidad*: Un atacante puede realizar *eavesdrop* mientras el documento es transmitido y además, el TSS almacena una copia del mismo.

- *Ancho de banda y almacenamiento*: Los requerimientos de ancho de banda y capacidad de almacenamiento para transmitir documentos de cierto tamaño a través de la red son muy altos.
- *Incompetencia*: El contenido de los documentos se puede corromper durante la transmisión de los mismos.
- *Confianza*: No hay mecanismos que prevengan que el TSS actúe maliciosamente registrando fechas incorrectas.

Para darle solución a las cuestiones planteadas se propone el uso de funciones hash sobre los documentos para resolver las cuestiones sobre la privacidad, ancho de banda y almacenamiento. Se propone enviar al TSS el hash del documento en lugar del propio documento, de esta forma se mantiene la privacidad de la información, se reduce el ancho de banda para transmitir el documento y la necesidad de espacio de almacenamiento en el TSS.

Con el objetivo de resolver la posible incompetencia del TSS, se plantea el uso de un mecanismo de firma digital. El TSS añade la fecha y hora sobre el hash recibido y lo firma digitalmente, para posteriormente enviárselo de nuevo al cliente. De este modo, se consigue que el cliente pueda verificar que el TSS ha sido realmente el que le ha enviado la información firmada, que la información es correcta y que la fecha y hora indicada es la correcta. Además, se evita que el TSS almacene la información.

Sobre el último aspecto a solventar, la confianza, se plantean dos soluciones en el artículo. La primera consiste en enlazar la información firmada de un bloque con información del bloque inmediatamente anterior. Con esta solución conseguimos que sea posible verificar que efectivamente un bloque está correctamente firmado de forma bidireccional, ya que contiene información del bloque anterior, y en la

otra dirección porque se puede cuestionar al siguiente si dispone de la información del anterior. La segunda consiste en distribuir la información anterior entre todos los participantes, así se puede prescindir del TSS centralizado.

A día de hoy nos encontramos con múltiples proyectos basados en la tecnología Blockchain en diversos ámbitos, que van desde el sector bancario hasta el sector de la sanidad, pasando por la industria musical y por los servicios públicos y gubernamentales.

2.1.1. Bitcoin

Cuando hablamos de Blockchain debemos mencionar Bitcoin, ya que se trata de la primera red de tipo Blockchain. Surge en 2008 a partir de un artículo publicado bajo el seudónimo de Satoshi Nakamoto y cuyo título es "*Bitcoin: A Peer-to-Peer Electronic Cash System*"[1]. A pesar de que existieron otras criptomonedas y diseños anteriores, es considerada la primera criptomoneda que empezó a operar en 2009.
Bitcoin conforma un gran libro de cuentas, distribuido y público en el que se anotan todas las transacciones realizadas en el sistema de forma segura y difícilmente falsificable. Para lograr esto se basa en la tecnología Blockchain. El objetivo de Bitcoin es el de proveer de un medio digital de pago, de manera que no exista una tercera parte o intermediario de confianza en las transacciones llevadas a cabo entre nodos del sistema. Gracias a lo cual se reducen los costes de emisión de transacciones y mantenimiento de la propia red.

En Bitcoin, cada una de las monedas se define como una cadena de firmas digitales. Tal y como se aprecia en la *Figura 3*, cuando se realiza la transferencia de una moneda de un dueño a otro, se firma digitalmente un hash de la transacción anterior junto con la clave pública del próximo dueño. Finalmente se añade esta información al

final de la moneda. Así se puede verificar la cadena de propietarios de la moneda mediante las firmas digitales.

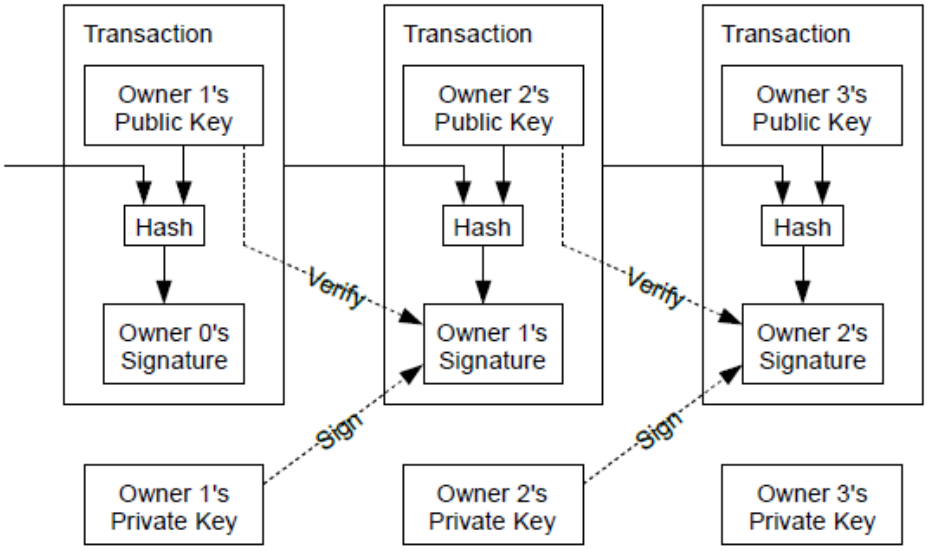


Figura 3: *Transferencia de moneda en bitcoin*[1]

Uno de los problemas principales que resuelve Bitcoin es el del doble gasto. Para solventarlo, sin necesidad de un intermediario, la confianza en una tercera parte es substituida en Bitcoin por una prueba de trabajo, *Proof of work*, realizada por N participantes de la red, sobre el orden en que las transacciones fueron realizadas. Esto implica que todas las transacciones deben ser públicas. Lo que puede parecer a simple vista que vulnera la privacidad de los nodos que las realizan, no lo es en realidad, ya que las claves públicas se mantienen anonimizadas.

La prueba de trabajo, *Proof of work*, consiste en un algoritmo de consenso en el cual cada uno de los nodos busca un problema para que resuelvan el resto de nodos de la red. En Bitcoin cada nodo recibe y almacena transacciones en un bloque, en el momento en que finaliza un bloque busca la prueba de trabajo correspondiente. Una vez encontrado el problema correspondiente envía dicha prueba al resto de

nodos que proceden a su verificación. En caso de que la mayoría de nodos la acepte, dicho bloque pasa a formar parte de la cadena de bloques y se procede con el siguiente bloque. La cadena de bloques más larga es siempre considerada la correcta, ya que es la que más gasto de CPU tiene. Así, si la mayoría de los nodos son honestos, la cadena más larga crecerá más rápido. De modo que para falsificar una cadena será necesario rehacer todo el trabajo de la cadena y adelantar a los nodos honestos en la cantidad de trabajo que realizan, lo cual tiene una probabilidad muy baja de que ocurra.

Tal como se indica en el artículo, Bitcoin está basada en el modelo que plantearon Haber y Stornetta en su artículo "*How to Time-Stamp a Digital Document*"[2]. Como indicábamos anteriormente, Haber y Stornetta, plantean varias alternativas en su artículo para resolver el problema de la confianza en la certificación de la fecha y hora de documentos. Bitcoin en su arquitectura combina las dos opciones, distribuyendo la confianza de la red sobre todos los nodos que la conforman y además aplicando el enlazado de los bloques al mismo tiempo.

2.2. Tipos de Blockchain

Dentro de los distintos tipos de cadenas de bloques que existen, podemos distinguir varios tipos atendiendo a diferentes criterios. Por un lado, las cadenas de bloques se pueden diferenciar por el tipo de acceso que se realizada a los datos, existen Blockchains públicas y privadas. Por otro lado, también se pueden diferenciar según los permisos, por lo que podemos encontrarnos Blockchain con permisos, *permissioned*, o sin permisos, *permissionless*.

2.2.1. Blockchain Públicas

Son aquellas cadenas en las que es posible la lectura de la cadena de bloques por cualquier nodo. Además está permitido el envío de bloques para que estos sean añadidos a la cadena.

Al mismo tiempo dentro de las cadenas públicas, podemos realizar distinción atendiendo a los permisos. En las cadenas de bloques sin permisos, o *permissionless*, todos los nodos pueden procesar transacciones y crear bloques. Mientras que en las que cuentan con permisos, *permissioned*, existen una serie nodos, cuya identidad es conocida, que son los responsables del procesamiento de transacciones y creación de bloques.

Como ventajas de las cadenas de bloques públicas *permissionless* podemos destacar:

- Son cadenas totalmente descentralizadas por lo que la información se distribuye a través de todos los nodos que conforman la red.
- Se mantiene en todo momento la anonimidad de los participantes.
- La transparencia, todos los participantes participan de igual forma en la cadena de bloques y tienen accesible la misma información.
- Dado que están abiertas a cualquier participante que quiera tomar parte, permiten la realización de transacciones en un entorno no seguro.

Como desventajas podemos destacar las siguientes:

- Gran consumo de recursos del algoritmo de consenso *Proof of Work*

- Existe un número de transacciones limitado que se pueden introducir en un bloque.
- El anonimato puede ser considerado como una desventaja en ámbitos en los que existen requerimientos más estrictos sobre la identidad y seguridad.

2.2.2. Blockchain privadas

Las Blockchains privadas son aquellas en las que están restringidos los nodos que pueden escribir la cadena de bloques. En este tipo de cadenas de bloques existen ciertos nodos con una identidad conocida que realizan los trabajos de verificación de bloques. Mientras que la lectura puede ser abierta a todos los nodos que permanecen en la red o estar restringida.

Como ventajas de las cadenas de bloques privadas resaltamos:

- *Rapidez*: El rendimiento de estas redes es mayor ya que existen nodos predefinidos que se encargan exclusivamente de la validación de transacciones.
- *Mayor nivel de privacidad*: El nivel de privacidad es mayor ya que la red no es abierta a cualquier participante.
- *Menor coste*: El coste de las transacciones es menor ya que no es necesario implementar el algoritmo de consenso "*Proof of Work*".

Como desventajas cabe mencionar las siguientes:

- *Puntos de fallo*: Dado que son redes que no están descentralizadas al cien por cien, los puntos de fallo son mayores.

- Es necesario confiar plenamente en los nodos validadores ya que en caso de que estos resulten *hackeados* podrían hacer un uso fraudulento de la red.

2.3. Aplicaciones descentralizadas

Las aplicaciones descentralizadas, también conocidas como Dapps, son aquellas que se ejecutan en un entorno totalmente descentralizado. Tal y como se puede apreciar en la *Figura 4*, al contrario de lo que ocurrre en las aplicaciones centralizadas, en las aplicaciones descentralizadas no existe un único nodo central.

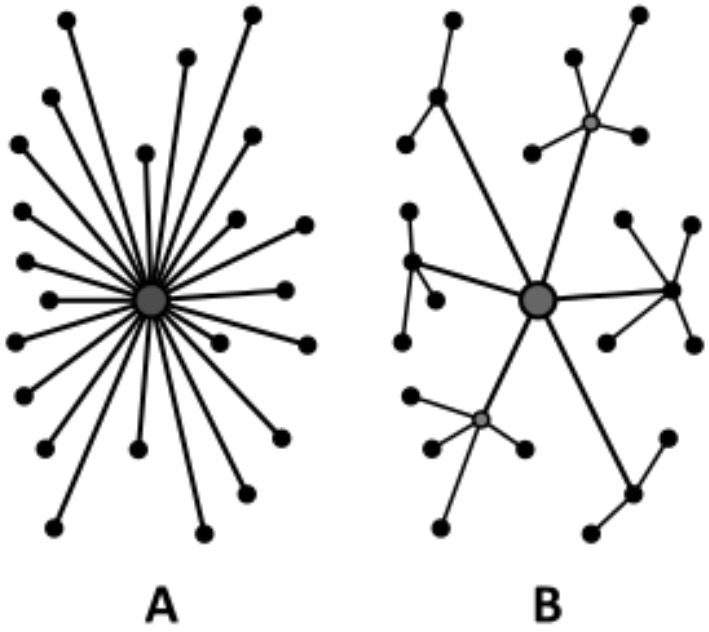


Figura 4: *(A) Sistema Centralizado – (B) Sistema descentralizado*[3]

Estas aplicaciones pueden ejecutarse desde cualquier nodo que forme parte de la red. Se han vuelto muy populares desde la aparición de Bitcoin. Una de las ventajas, de un modelo descentralizado, es su mayor tolerancia a fallos respecto a modelos centralizados, y la redundancia y alta disponibilidad de los servicios y de los datos que se obtiene. Más concretamente, en el contexto de las Blockchain, estas

Dapps tienen como ventaja la eliminación de intermediarios y la seguridad que ofrecen gracias al uso de la cadena de bloques y certificados con claves público/privadas para garantizar la integridad y autenticidad de los datos.

2.4. Ethereum

Surge en el año 2015 y es una plataforma Opensource descentralizada y basada en Blockchain. Dentro de la clasificación por tipos de las cadenas de bloques encaja como Blockchain pública *permissionless* y dispone de su propia criptomoneda, llamada Ether (ETH).

A diferencia de otras Blockchain como Bitcoin, Ethereum es una cadena bloques programable. Esto nos permite desarrollar y desplegar sobre ella aplicaciones descentralizadas o *dapps.* Estas aplicaciones descentralizadas se pueden desplegar en cualquier nodo de la red, e interactúan con la cadena de bloques mediante *Smart Contracts.* Los *Smart Contracts* son aplicaciones que permiten definir, por una o más partes, una serie de reglas que se ejecutan automáticamente cuando se cumplen las condiciones para ello. Estos *Smarts Contracts*, o contratos inteligentes no pueden ser modificados una vez están presentes en la cadena de bloques, del mismo modo, cuando se ejecutan no es posible dar marcha atrás sobre los mismos. El lenguaje empleado en Ethereum para desarrollar estos *Smart Contracts* es Solidity.

Entre las ventajas que proporcionan los Smart Contracts podemos destacar que reducen los costes debido a que desaparecen los intermediarios de las relaciones establecidas entre las partes, el tiempo de establecimiento de estos acuerdos se ve reducido respecto a los

métodos tradicionales y evitan el fraude ya que no es posible revertirlos.

Otro elemento introducido en Ethreum es el de DAO (*Digital decentralized autonomous organization*) que es un modelo de negocio y a su vez una forma de financiamiento. Permite la creación de organizaciones totalmente autónomas que se definen mediante un conjunto de Smart Contracts. Estos Smart Contracts vienen a sustituir los estatutos y reglas por las que se rigen las organizaciones tradicionales. La participación en las mismas concede derecho a voto sobre las acciones a desarrollar, lo que permite que la organización sea controlada por sus participantes y no por una persona física o un consejo como se realiza de forma tradicional.

2.5. Alastria

Organización sin ánimo de lucro que nace en 2017 a raíz de la *Red Lyra*. Alastria forma un consorcio de empresas multisectorial cuyo objetivo es el de democratizar el acceso a Blockchain en España. En general, trata de definir un marco técnico para el uso de Blockchain, desarrollando las técnicas y herramientas necesarias para favorecer el acceso, adopción y uso de la tecnología Blockchain en nuestro país. Provee de una red de Blockchain neutra e independiente en la cual es posible desarrollar productos y servicios basados en Blockchain con reconocimiento legal en España y dentro del marco Europeo.

Alastria provee de una red semipública permisionada. Existen los siguientes tipos de nodos, cada uno de los cuales cumple una función determinada dentro de la red:

- *Nodos validadores (Block Makers):* Son los nodos que se encargar de ejecutar el algoritmo de consenso IBFT y de escribir los bloques en la cadena.

- *Nodo permisionador (Boot node):* Son nodos con direcciones conocidas por todos los nodos de la red. Cuando un nodo se une por primera vez a la red, puede hacerlo a uno de este tipo y a partir de este, descubrir nuevos nodos.
- *Nodos regulares*: Son los nodos que replican los bloques de la Blockchain validados y ejecutan las transacciones que son incluidas posteriormente en la red por parte de los validadores.

Además de la propia red de Blockchain, Alastria dispone de un modelo de identidad digital, mediante el cual intenta que las transacciones tengan validez legal. Alastria se define como red agnóstica por lo que existe la posibilidad de desplegar desarrollos de diferentes proveedores. Actualmente cuenta con tres redes, dos de las cuales están en fase de pruebas:

- *Quorum Red T*: Cuenta con una arquitectura basada en la Blockchain Quorum, que a su vez está basada en Ethereum.
- *Parity*: Red en fase de pruebas. Parity es un Framework de Blockchain que está formado por un conjunto de herramientas opensource.
- *Hyperledger*: Red en fase de pruebas basada en Hyperledger Fabric de Linux Software Foundation.

2.6. Hyperlegder

En el año 2015 surge este proyecto Opensource bajo el paraguas de la Linux Foundation. En el mismo colaboran múltiples organizaciones, cuyos sectores van, desde el sector de las finanzas, pasando por el sector aeroespacial, hasta el sector de la automoción, entre otros.
El objetivo de Hyperledger es el de crear una infraestructura que albergue herramientas, frameworks, librerías e interfaces de cara a mejorar la fiabilidad y rendimiento de la tecnología Blockchain.

Podemos clasificar Hyperledger como red de Blockchain privada permisionada. Como principales diferencias con otras tecnologías Blockchain como Bitcoin o Ethereum, podemos destacar que Hyperledger no dispone de una criptomoneda propia.

Principalmente existen dos ramas que engloban los diferentes proyectos de los que consta Hyperledger. Por un lado, están los Frameworks:

- *Hyperledger Fabric:* Cadena de bloques permisionada y distribuida para el desarrollo de aplicaciones y soluciones en el ámbito empresarial.
- *Hyperledger Sawtooth:* Cadena de bloques inicialmente desarrollada por Intel y que permite su despliegue en modo *permissioned* o *permissionless*. Incluye un algoritmo de consenso llamado "*PoET*", Proof of Elapsed Time.
- *Hyperledger Iroha:* Cadena de bloques orientada a su integración en otros proyectos como Hyperledger Fabric y Hyperledger Sawtooth que cuenta con una serie de elementos preconfigurados. Además, está orientada al desarrollo de proyectos móviles. Está desarrollada en C++.

Por otro lado, Hyperledger también ofrece la posibilidad de emplear una serie de herramientas que tienen como objetivo facilitar el desarrollo y mantenimiento de proyectos basados en Blockchain:

- *Composer:* Conjunto de herramientas con el objetivo de facilitar el desarrollo de soluciones basadas en Blockchain como aplicaciones y Smart Contracts.
- *Cello:* Desarrollado inicialmente por Intel permite la implementación de una Cadena de bloques como un servicio facilitando el desarrollo, operación y gestión de los mismos.

- *Explorer*: Herramienta que permite la visualización de objetos de una cadena de bloques como bloques, transacciones, nodos y estadísticas.

2.7. Comparativa

En la tabla que se presenta a continuación se muestra una comparativa de las distintas redes presentadas anteriormente:

	Ethereum	Alastria	Hyperledger
Año de lanzamiento	2014	2017	2015
Tipo de Blockchain	Pública	Privada	Privada
Permisos	Permissionless	Permissioned	Permissioned
Criptomoneda	Ether	-	-
Smart Contracts	Si	Si	Si
Lenguaje	Solidity	Solidity	Go, node.js, java
Velocidad (transacciones por segundo)	~15	~400	~3.000-20.000
Algoritmo de Consenso	PoW	IBFT	PBFT

Figura 5: *Tabla comparativa*

Teniendo en cuenta las condiciones en las que se enmarca el presente trabajo, se consideran los siguientes factores de cara a tomar una decisión respecto al tipo de Blockchain que se empleará para desarrollo del mismo:

- *Privacidad*: Se debe mantener en todo momento la privacidad de los datos, de las operaciones que se realizan sobre los mismos y sobre las entidades que las realizan.
- *Identidad*: El acceso a datos sensibles debe ser realizado sólo por entidades autorizadas al mismo. Se debe tener la capacidad de restringir las diferentes operaciones que es posible realizar en función del tipo de usuario.

- *Trazabilidad*: El acceso a datos sensibles debe quedar registrado en todo momento. Tanto la lectura como la modificación de los mismos.

Teniendo en cuenta los aspectos anteriores se descarta el uso de Ethereum dado que es un Blockchain pública. Se considera más adecuado el uso de una Blockchain privada, como Alastria o Hyperledger, dado que se van a gestionar datos sensibles, y su diseño y arquitectura está orientado dotar el sistema de mayor confiabilidad y seguridad.

Llegados a este punto, tenemos como alternativas Alastria e Hyperledger como redes privadas. Como desventaja de Alastria respecto a Hyperledger, la velocidad de las transacciones de esta es significativamente menor que en Hyperledger.

Cabe destacar también que la Red T de Alastria está basada en Quorum, que es una Blockchain que fue creada por JP Morgan con el objetivo de disponer de una red de Blockchain orientada a las entidades financieras. Por este motivo, esta red está enfocada al ámbito de las finanzas. Mientras que Hyperledger tiene una orientación que abarca un abanico más amplio de sectores. Quorum a su vez, está basado en Ethereum, a pesar de lo cual las transacciones son solo visibles para nodos autorizados. Además, a priori Hyperledger nos proporcionará una gestión de permisos más adaptable, ya que permite desgranar los mismos por cada rol de usuario limitando las acciones que puede realizar cada uno. En cambio, con Quorum los nodos pueden ser validadores o no, pero no es posible establecer permisos con tanto detalle como en Hyperledger.

Teniendo en cuenta los puntos indicados anteriormente se selecciona Hyperledger para el desarrollo del presente trabajo, concretamente Hyperledger Fabric. Dado que es una tecnología que abarca más ámbitos aparte de las finanzas y en los que habrá que realizar una gestión de permisos y roles más fina.

Finalmente, tal como se indicaba anteriormente, Alastria trata de ser una iniciativa agnóstica. Actualmente se encuentra trabajando en el despliegue de una red basada en Hyperledger. De modo que en un futuro se podría llevar a cabo la adaptación de este trabajo en la Red de Alastria.

3. Dapp para el acceso y modificación de información sensible

Actualmente, en el ámbito de los sistemas de información sanitaria, se presentan diferentes soluciones para lograr la interoperabilidad entre las Historias Clínicas Electrónicas pertenecientes a diferentes organizaciones de la salud. Algunas de estas soluciones son sistemas de información que exportan los datos de los pacientes a un dispositivo físico, estos datos se deben importar posteriormente en el sistema destino o ser consultados directamente en el dispositivo. Estas soluciones requieren la intervención de personas autorizadas que ejecuten la exportación de la información bajo petición expresa del paciente. Existen casos en los que se ha llevado a cabo la integración entre historias clínicas de paciente mediante el desarrollo de soluciones basadas en estándares[4]. Este intercambio de información se realiza en base a un acuerdo realizado exprofeso entre las organizaciones implicadas en el intercambio.

Cabe destacar también, que la cronificación de enfermedades que surge en parte, debido al envejecimiento cada vez mayor de la población, hace surgir la necesidad de definir circuitos de telemedicina y teleasistencia, en los cuales, es posible que un paciente introduzca datos por sí mismo desde su casa, como por ejemplo una medida de tensión, o en los que un profesional, visualiza información clínica o ejecuta algún tipo de acción de forma deslocalizada, como la generación de un informe. Esto hace que cada vez sea más frecuente el requerimiento, tanto por parte de los profesionales como por parte de los pacientes, de disponer de acceso a la información clínica desde sus propios dispositivos.

En todas las soluciones planteadas anteriormente, siempre se requiere la participación de una entidad o persona autorizada para llevar a cabo el intercambio de información. El paciente no es el responsable directo de autorizar este intercambio, a pesar de que los datos intercambiados son suyos.

En el presente trabajo, se plantea el desarrollo de un sistema para el acceso a la Historia Clínica de los pacientes de una organización de la salud. Esta organización desea que el sistema de acceso a la Historia Clínica sea común tanto para los facultativos, que accederán a la misma para revisar el detalle de sus pacientes, como para los pacientes, que podrán acceder desde sus casas a su historial médico.

Como método de identificación de los usuarios, se define que estos se identifiquen mediante certificados digitales. A los facultativos se les asigna un certificado digital por parte de la propia organización en la que trabajan, mientras que los pacientes deberán presentar un certificado digital propio emitido por una entidad que controla el censo poblacional.

Se decide implementar este portal de acceso mediante tecnología Blockchain, habrá que tener en cuenta a la hora de desarrollar el sistema, la condición de información sensible que tienen los datos médicos de los pacientes y las implicaciones de seguridad que ello supone. Por último, dado que Blockchain es una tecnología reciente se plantea realizar una prueba de concepto previa para determinar la viabilidad del proyecto.

3.1. Análisis y diseño del Sistema

En el presente apartado se especifican por un lado, los requerimientos del sistema a desarrollar y sus casos de uso, y por otro lado, el detalle de su arquitectura y la especificación de cada uno de los elementos de los que está compuesto el sistema.

3.1.1. Especificación de Requisitos

En el presente apartado se definen los requisitos que debe cumplir el sistema. Por un lado distinguimos los requisitos funcionales y por otro lado, los requisitos no funcionales:

- Requisitos Funcionales:
    - El sistema almacenará en la Blockchain la información relativa a los informes. Se almacenará la siguiente información para cada informe:
        - Identificador del informe
        - Identificador de paciente
        - Validador del informe
        - Fecha y hora de creación del informe
    - En el sistema existirán dos roles de usuario diferentes, por un lado el rol paciente y por otro lado, el rol facultativo.
    - En función del tipo de rol y del usuario se tendrá permiso de acceso a la información o no.
        - Los pacientes sólo podrán acceder a aquellos documentos de los que sean propietarios.
        - Los facultativos tendrán acceso total a todos los informes del sistema.

- Todos los accesos y modificaciones sobre los informes deberán quedar registrados, de modo que se pueda determinar quien accedió a los informes, quién realizó modificaciones sobre los mismos y en qué momento se llevaron a cabo.

- Requisitos No Funcionales
  - Dado que nos encontramos diseñando un sistema distribuido, la información viajará por la red, lo que implica que los protocolos empleados para la comunicación entre los diferentes componentes del sistema deben ser seguros.
  - El sistema debe ser accesible desde cualquier punto a través de internet.
  - Esta Dapp actuará como punto de acceso para todas aquellas personas que requieran acceder al repositorio de informes médicos.

3.1.1.1.1. Casos de uso

Según los requerimientos especificados en el apartado anterior, se presentan en este apartado los diferentes casos de uso que se implementarán en el sistema:

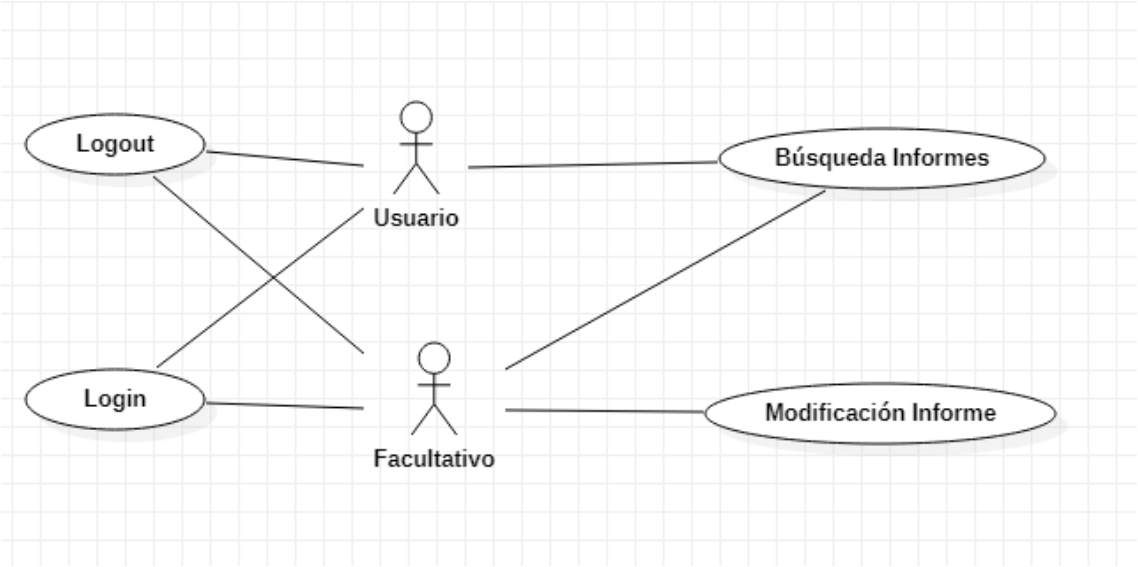


Figura 6: *Diagrama de casos de uso*

3.1.1.1.2. Caso de uso: Login de usuarios

CU-1	Login de usuario
Actores	Facultativos y pacientes
Precondiciones	• El certificado del usuario se encuentra cargado en el navegador
Descripción	• El usuario introduce el id de usuario y pulsa el botón Login • El sistema redirige al usuario a la pantalla de búsqueda de informes en caso de que la autenticación sea correcta
Postcondiciones	• El sistema almacena en sesión los datos de usuario

3.1.1.1.2.1. Diagrama de secuencia

Se muestra a continuación el diagrama de secuencia correspondiente al caso de uso de Login de Usuario:

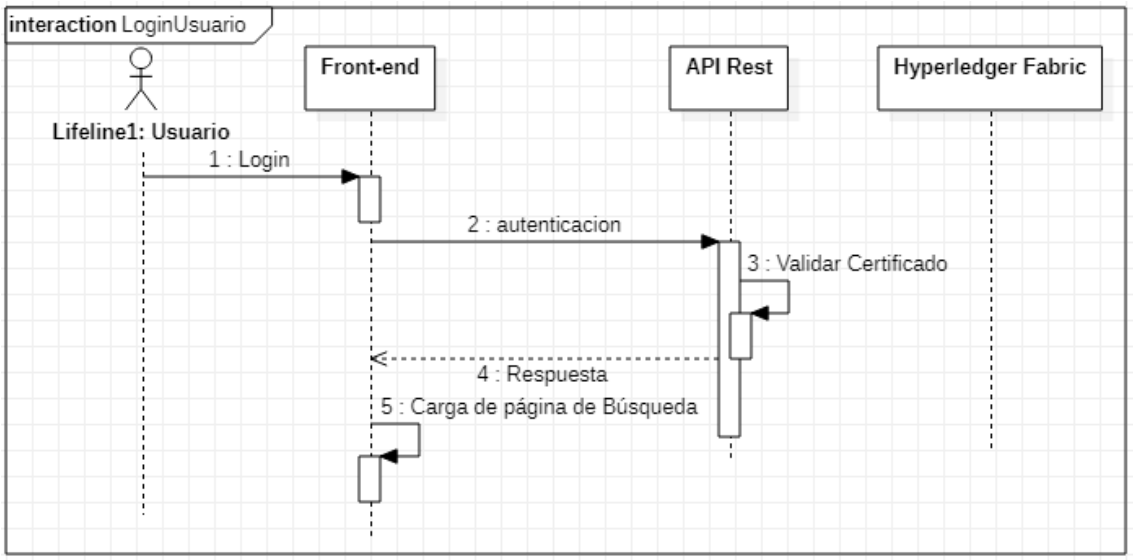


Figura 7: *Diagrama de secuencia del CU-1*

3.1.1.1.3. Caso de uso: Logout de usuarios

CU-2	Logout de usuario
Actores	Facultativos y pacientes
Precondiciones	• El usuario se encuentra logueado en el sistema • El certificado del usuario se encuentra cargado en el navegador
Descripción	• El usuario pulsa el enlace "*logout*" situado en la parte superior derecha de la pantalla • El sistema redirige al usuario a la pantalla de Login
Postcondiciones	• El sistema elimina los datos de usuario de la sesión

3.1.1.1.3.1. Diagrama de secuencia

Se muestra a continuación el diagrama de secuencia correspondiente al caso de uso de Logout de Usuario:

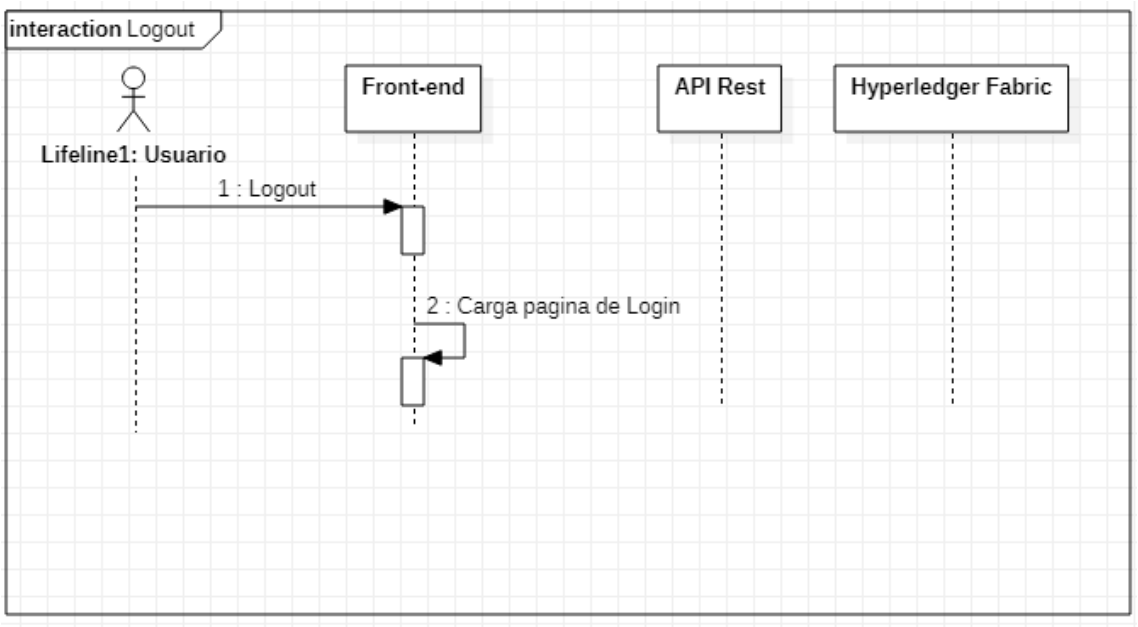


Figura 8: *Diagrama de secuencia del CU-2*

3.1.1.1.4. Caso de uso: Búsqueda de informes

CU-3	Búsqueda de informes
Actores	Facultativos y pacientes
Precondiciones	• El usuario se encuentra logueado en el sistema • El certificado del usuario se encuentra cargado en el navegador
Descripción	• El usuario introduce el id de Informe por el que quiere realizar la búsqueda y pulsa el botón de buscar • El sistema realiza una consulta a la Blockchain y obtiene los resultados
Postcondiciones	• En caso de que existan resultados coincidentes con el parámetro se mostrarán en una tabla • En caso de que no existan resultados coincidentes se indicara con un mensaje de aviso

3.1.1.1.4.1. Diagrama de secuencia

Se muestra a continuación el diagrama de secuencia correspondiente al caso de uso de Búsqueda de informes:

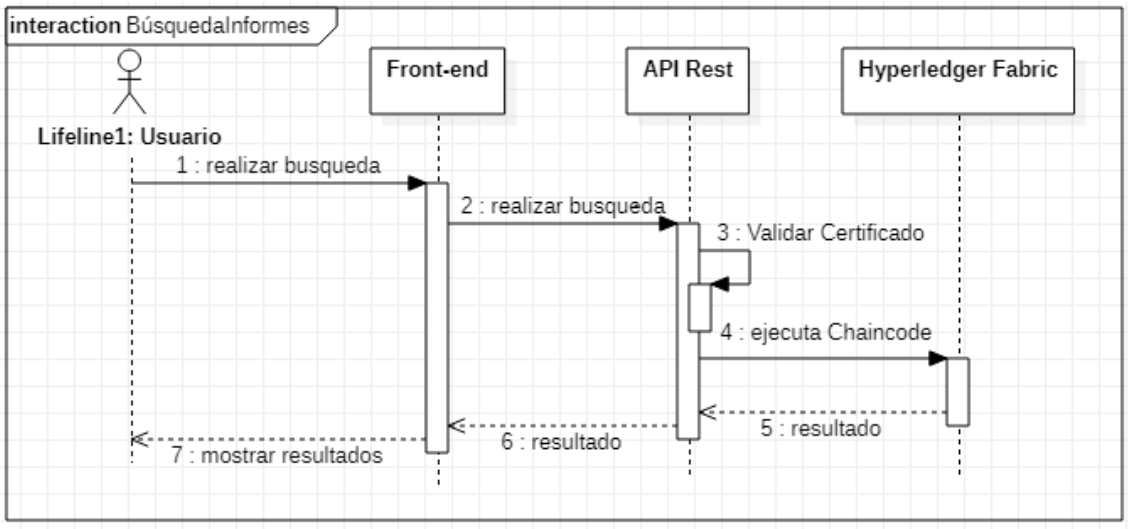


***Figura 9**: Diagrama de secuencia del CU-3*

3.1.1.1.5. Casos de uso: Modificación de informe

CU-4	Modificación de informe
Actores	Facultativos
Precondiciones	• El usuario se encuentra logueado en el sistema • El certificado del usuario se encuentra cargado en el navegador • El usuario debe de haber realizado una búsqueda previa con resultados coincidentes
Descripción	• El usuario podrá realizar la modificación del usuario validador de informe sobre la propia tabla de resultados. • Una vez guardada la modificación el sistema persiste la modificación sobre la Blockchain.
Postcondiciones	• En informe se mostrará actualizado en la tabla y en la Blockchain

3.1.1.1.5.1. Diagrama de secuencia

Se muestra a continuación el diagrama de secuencia correspondiente al caso de uso de Modificación de Informe:

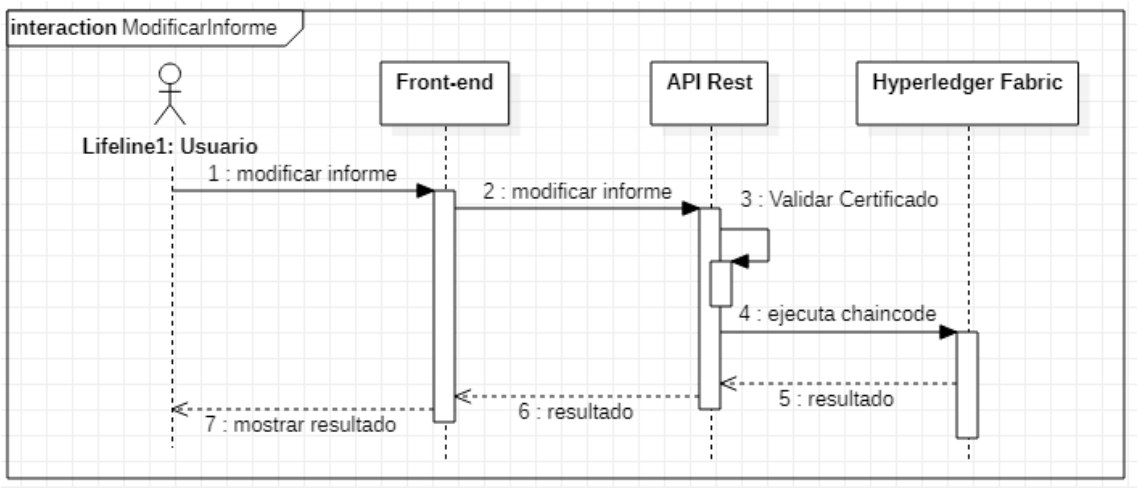


Figura 10: *Diagrama de secuencia del CU-4*

3.1.2. Arquitectura del Sistema

En el siguiente diagrama (*Figura 11*) se muestra la arquitectura del sistema:

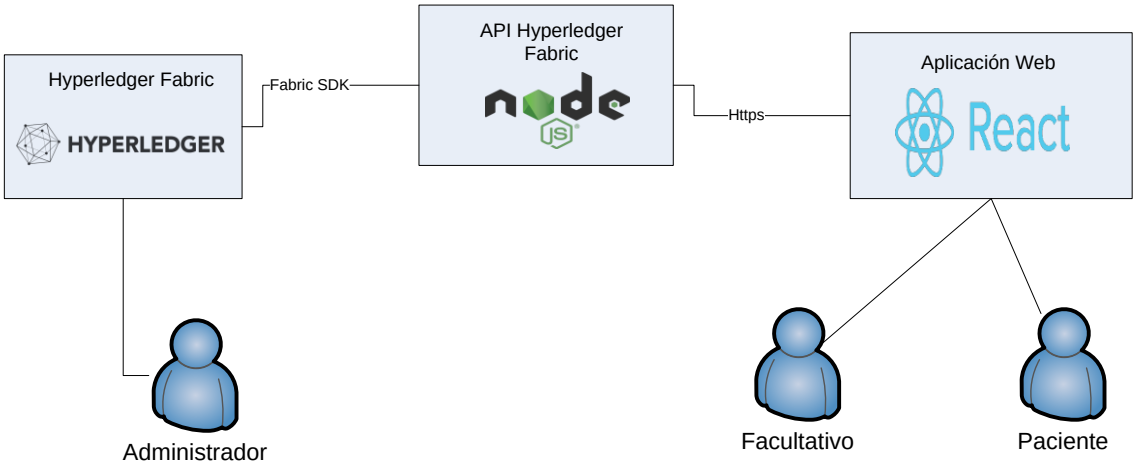


Figura 11: *Arquitectura de sistema*

- *Aplicación Web (React)*: Este componente se corresponde con el Front-end del sistema. A través de mismo realizan el acceso los facultativos y pacientes.
- *API Hyperledger Fabric (node.js):* Este componente se corresponde con una API Rest. Su objetivo es el de proveer de

una fachada de servicios Rest que interactuarán con la Blockchain Hyperledger a través de la ejecución de los chaincode.

Para establecer la comunicación entre el Front-end con la Blockchain se decide desarrollar una API Rest propia, debido a que la que provee Hyperledger no dispone de apenas documentación y referencias. En cuanto al SDK de Hyperledger Fabric, este se usará en los propios servicios Rest desarrollados para interactuar con la red Blockchain.

- *Hyperledger Fabric*: Se corresponde con la Blockchain en la cual se almacena la información de los informes y en la que se lleva el registro de accesos. Como base de la red Blockchain se parte de los tutoriales "*Building your First Network*" y "*Writing Your First Application*" que se pueden encontrar en el repositorio de documentación de Hyperledger Fabric[5]. Obtenemos los siguientes elementos después de seguir los tutoriales indicados:
    - *First Network:* Se trata de una red de Blockchain básica que cuenta con dos organizaciones y un canal de comunicación entre ellas. Sobre esta red se llevan a cabo una serie de modificaciones para adaptarla a los objetivos de nuestro sistema.
    - *Fabcar:* Aplicación descrita en el tutorial "*Writing Your First Application*" que incluye una serie de chaincodes para interaccionar con la red "*First Network*". Sobre esta aplicación se desarrollan los elementos necesarios para nuestro sistema.

En el siguiente diagrama se puede apreciar el diseño de la Blockchain implementada:

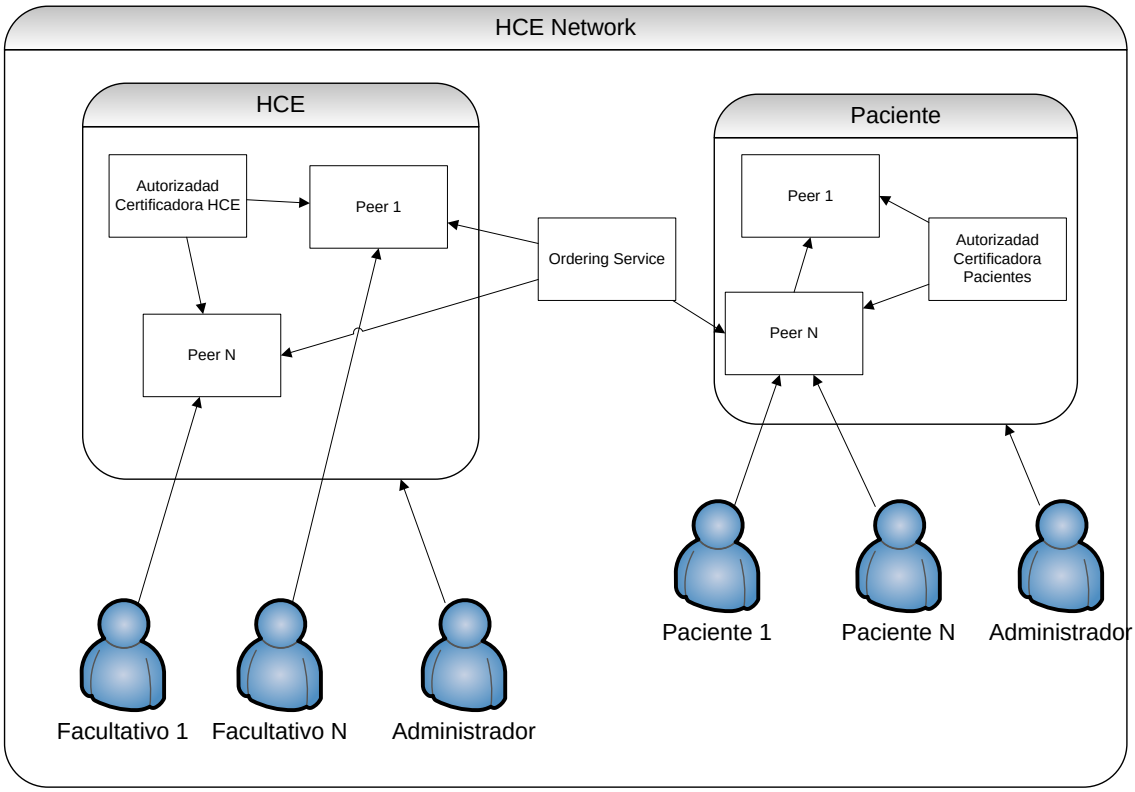


Figura 12: *Blockchain HCE Network*

Según se puede apreciar en el diagrama anterior (*Figura 12*), la red de Blockchain creada se llama HCE Network y representa una Historia Clínica Electrónica (HCE). A continuación se detallan los componentes que forman parte de la misma:

- *HCE*: Es la organización que representa la Historia Clínica Electrónica de la organización de la salud. A su vez está formada por una autoridad certificadora propia cuya responsabilidad es la de otorgar identidades a los usuarios que forman parte de la misma, en este caso, a los facultativos. Además, la red está formada por una serie de nodos (peers) que la conforman. En este caso existen dos nodos, pero en un caso de uso real podría existir un nodo por centro hospitalario.
- *Paciente*: Se corresponde con una organización que controla un censo poblacional y a la que pertenecen los pacientes. Dispone de una autoridad certificadora que se encarga de otorgar

identidades a los pacientes. Además, cuenta con un conjunto de nodos (peers) que forman parte de la red.

- *Ordering Service*: Es el nodo que se encarga de mantener la consistencia entre los diferentes nodos de la red mediante el algoritmo de consenso y asegurando el orden de las transacciones. Por simplicidad, dado que se trata de un trabajo en el que se desarrolla un *Proof of Concept*, tan sólo se despliega un orderer (algoritmo *solo*). En un entorno de producción no debe haber un solo nodo *orderer*.

3.1.3. Gestión de información sensible

Cuando hablamos de información sensible debemos hacer referencia al Reglamento General de Protección de Datos. Este es el reglamento europeo para el tratamiento de datos personales (RGPD).

En lo que se refiere a datos de salud, estos son consideramos datos a los que hay que tratar con un nivel de protección alto. En el contexto del trabajo en el que nos encontramos, esto implica lo siguiente cuando se trata este tipo de información en un sistema:

- Estos datos deben transmitirse cifrados con el fin de evitar que sean interceptados por un tercero y este tenga acceso a los mismos sin consentimiento.
- Debe existir un registro de acceso a los datos en el que se registre quien accede a los datos, en que momento y si este tenía autorización para hacerlo o no.

Existen otras consideraciones de seguridad respecto a estos datos, como el mantenimiento de copias de seguridad en una ubicación diferente, pero estos quedan fuera del alcance del presente trabajo ya que afectan a temas de manteniemiento e infraestructura generales.

3.1.4. Hyperledger Fabric

Es necesario tener en cuenta diversos aspectos a la hora de diseñar un sistema que gestiona información sensible en Hyperledger Fabric. Por un lado, teniendo en cuenta que se trata de un sistema distribuido, debemos asegurar que la información sensible se transmite en todo momento a través de canales seguros. Desde un punto de vista funcional, debemos asegurar también que cada uno de los usuarios que acceden al sistema, acceden tan sólo a la información a la que tienen acceso según el rol que tiene asignado cada uno. Respecto a este último punto, es importante también que el sistema, tenga la capacidad de identificar a cada uno de los usuarios de forma univoca.

Se hará uso de las siguientes opciones para asegurar que el tratamiento de la información sensible es adecuado:

- Hyperledger Fabric
    - La comunicación entre los diferentes nodos que toman parte en la blockchain se realizará en todo momento mediante mutual TLS (Transport Layer Security). De este modo la información se transmitirá cifrada en todo momento y se verificará la identidad de las partes de forma bidireccional mediante certificados.
    - El acceso a la información se restringirá en la propia implementación de los chaincodes. En el código se contralará el acceso a la información en función de información asociada a los certificados, como el rol del usuario o su identificador. Se empleará la librería CID (Client Users Identity) para llevar a cabo este control.

    - Los usuarios dispondrán de un certificado de clave pública que será empleado por la Blockchain para validar las transacciones.
- Dapp
    - La comunicación con la API Rest se realizará mediante protocolo Https, es decir, que la información viajará cifrada entre las diferentes entidades que toman parte en la comunicación.
    - El certificado de los usuarios se incluirá de modo que se realizará la validación del certificado cliente en el servidor. Así conseguimos asegurar que la identidad del cliente es válida.

Como opciones disponibles que se podrían emplear en el contexto de trabajo en el que nos encontramos, Hyperledger Fabric también provee de una funcionalidad llamada "*Data Collections*", para la gestión de información privada en un canal de comunicación entre organizaciones. Esta funcionalidad permite definir un subset de información, de modo que sólo sea accesible a un subconjunto de organizaciones dentro del canal de comunicación. Se descarta su uso en el presente trabajo debido a que la red que se construye, dispone de dos organizaciones sobre las que no es necesario establecer datos privados.

Hyperledeger Fabric también dispone de la opción de configurar "Access Control Lists" (ACL). La definición de ACL's nos permite controlar el acceso a los recursos de la Blockchain mediante la asignación de políticas a las diferentes identidades. Estas políticas se asocian a nivel de recurso, donde un recurso puede ser una chaincode de usuario, una chaincode de sistema o un "event resource". Para este trabajo queda fuera del alcance el uso de las ACL's, ya que si bien es posible restringir el uso de los recursos, estas restricciones se

establecen sobre los peers de las organizaciones y no sobre los usuarios, que es lo que se requiere en este caso.

4. Desarrollo y configuración del Sistema

En el presente apartado se presentan los detalles técnicos de implementación de sistema. Se comienza con la presentación de las características del entorno de desarrollo, se continúa con la introducción a las distintas tecnologías empleadas, se presentan los diferentes componentes de los que consta el back-end y el front-end del sistema, y para terminar, se realiza una valoración del prototipo obtenido.

4.1. Entorno de desarrollo

Como entorno de desarrollo de desarrollo para la implementación del sistema se cuenta con una máquina virtual Ubuntu 18.04 LTS, sobre la que se despliega la red blockchain de Hyperledger Fabric.
Para acometer el desarrollo del Front-end se empleó el propio equipo y se hizo uso del editor de código Visual Code. Del mismo modo, se empleó el mismo editor de código para el desarrollo de los Chaincode de la Blockchain y de la API Rest. Finalmente como servidor de la capa API Rest de la Blockchain se hace uso de un servidor node.js.

4.2. Tecnologías empleadas

4.2.1. Hyperledger Fabric

Hyperledger Fabric es uno de los Frameworks de Blockchain que provee Linux Software Foundation en el marco de su iniciativa Hyperledger. Es una Blockchain privada permisionada que cuenta con un diseño modular y versátil que permite su uso en un amplio abanico de sectores.

A diferencia de otras Blockchain del mercado, como Ethereum, Hyperledger Fabric no cuenta con una criptomoneda propia. Permite el desarrollo de Smart Contracts, que son conocidos como Chaincode y que pueden ser programados en Go, node.js o Java.

Dado su carácter de red privada permisionada, Hyperledger Fabric resulta adecuada para el desarrollo de soluciones basadas en Blockchain en aquellos ámbitos donde la confidencialidad de las transacciones sea importante, como en los casos en los que se gestionan datos de carácter sensible. Además, resulta adecuada cuando es necesario establecer diferentes permisos para llevar a cabo ciertas acciones dentro de la red, ya que permite establecer un sistema de permisos para que cada rol de usuario tan solo pueda llevar a cabo una serie de acciones determinada. Para lograr esto último, todos los nodos que toman parte en la red deben ser conocidos dentro de la red.

Existen los siguientes tipos de nodos en Hyperledger Fabric:

- *Orderer Node:* Se encarga de la distribución de los bloques en la red, de asegurar su orden y de la ejecución del algoritmo de consenso.
- *Peer node:* Mantienen una copia de los datos del ledger sincronizados con la red. Además en función del rol que cumpla dentro del conjunto de Peer nodes podrá validar las transacciones.
- *Client Node:* Son aquellos nodos que envían transacciones. Deben de conocer de forma previa los nodos a los que enviar las transacciones para su validación.

4.2.2. Smart Contracts

Los Smart Contracts son programas que se encuentran distribuidos en la red de Blockchain y hacen uso de su confianza, seguridad y consenso entre los peers. Dotan a la Blockchain de la capacidad de ejecutar y hacer cumplir los acuerdos establecidos por dos o más partes cuando se cumplen las condiciones para su ejecución. Mediante el protocolo de consenso se validan y realizan nuevas transacciones que son propagadas posteriormente a todos los nodos, para ser ejecutados por los mismos de forma secuencial.

En Hyperledger Fabric se conocen como "*chaincode*". Estos tan sólo se podían desarrollar en Go en las primeras versiones, mientras que en la actualidad existe la posibilidad de emplear javascript (node.js) o java. Para nuestra implementación emplearemos node.js. Junto con el ledger, los Smart Contracts forma el core de Hyperledger Fabric. En el ledger se almacena el histórico de estados por el cual ha pasado la Blockchain, mientras que los Smart Contracts, definen las interacciones que se pueden llevar a cabo entre los diferentes nodos que conforman la red.

4.2.3. React.js

React.js es una librería open-source de Javascript cuya utilidad es la de construir interfaces de usuario de una sola página para App móviles y Web. Permite también el desarrollo de componentes UI reutilizables. Como características principales podemos destacar las siguientes:

- *Declarativo*: Al contrario que otros lenguajes, como jQuery, React.js es declarativo lo que permite que mediante la modificación de las propiedades de los componentes se produzca un cambio en la funcionalidad.

- *Basado en Componentes*: React.js está basado en componentes. Estos son representados como clases que heredan de *Component*. Todos ellos tienen como requerimiento definir un método *render*() que se encarga de establecer el contenido del componente. Además, está permitido el uso de JSX, que es una extensión que nos permite emplear etiquetas Html dentro de javascript, logrando así aumentar la expresividad del lenguaje.
- *Propiedades*: Son los atributos de configuración de cada componente. También se conocen como "*props*" y permiten que los componentes sean dinámicos, configurables y reutilizables.
- *Estado*: Se trata de la definición del propio componente en un instante determinado de tiempo. Existen dos tipos de componentes, con estado y sin estado.
- *DOM*: react.js mantiene su propio DOM. De modo que almacena la versión anterior del DOM y la actual, comparándola para actualizar de la mejor forma el DOM del navegador.

4.2.4. Node.js

Node.js es un entorno de ejecución de javascript construido con el motor de javascript V8 de Chrome. Está tecnología es usada en el presente trabajo con el objetivo de desarrollar la API Rest de Hyperledger Fabric.

4.3. Back-end

El Back-end del sistema cuenta por un lado con la configuración de la red de Blockchain, y por otro lado, con la API Rest que permite la ejecución de los Chaincode desde el Front-end. Respecto a la

Blockchain, esta se configura mediante Hyperledger Fabric. Se indican a continuación los elementos de configuración más importantes:

- *crypto-config.yaml:* Este fichero contiene la definición de topología de la red y a partir del mismo, se generan el conjunto de certificados para cada una de las organizaciones de la red y sus elementos. Se emplea la herramienta *cryptogen,* que se encarga de generar los certificados para cada una de las entidades y miembros de la red.
- *configtx.yaml*: Define los elementos que se emplearán en la red y que serán pasados a *configtxgen* con el objetivo de crear el bloque génesis y los channels. Configura los artefactos como el Orderer y el channel, y los MSP (Membership Service Providers) de las organizaciones.
- *base/peer-base.yaml:* Contienen la configuración común a todos los nodos que conforman la red.
- *base/docker-compose-base.yaml*: Especifica la configuración específica de cada uno de los nodos que conforman la red.
- *docker-compose-cli.yaml*: Fichero que extiende la configuración del fichero anterior y define la configuración del contenedor *cli.* Contenedor encargado de gestionar la comunicación por comandos con la red de Blockchain.
- *docker-compose-couch.yaml*: Especifica la definición de almacenamiento para cada uno de los nodos de cada organización.
- *startFabric.sh*: Script que se encarga de arrancar todos los elementos de la red y de instalar el chaincode.

La red de Blockchain consta, además de esta configuración, de una serie de métodos que conforman un Smart Contract o chaincode de la Blockchain. Se muestra a continuación los métodos implementados en el Chaincode:

- *initLedger()*: Método que se ejecuta en el proceso de arranque de la red de Blockchain y que inicializa el Ledger. La estructura del Ledger cuenta con una estructura de almacenamiento tipo *clave-valor*, donde se almacenan los datos correspondientes a los informes.
- *queryInforme()*: Método para consultar un informe de un determinado paciente en la Blockchain, especificando como parámetro el identificador de informe.
- *createInforme()*: Método que inserta un informe en la Blockchain. Este método se ha empleado llevar a cabo la validación del sistema. Por lo que no se emplea desde el front-end.
- *queryAllInformes()*: Método para la consulta del conjunto total de informes de un determinado paciente.
- *actualizarValidador()*: Método que tiene como objetivo la actualización del facultativo validador de un informe determinado. Recibe como parámetros el ID del informe a modificar y el nuevo validador.

Para finalizar, se desarrolla una API Rest para gestionar la interacción del Front-end con la red de Blockchain. Esta API consta de dos servicios Post:

- */api/buscarInformes*: Este servicio se encarga de ejecutar las búsquedas, tanto parametrizadas como sin parametrizar, en la Blockchain.
- */api/buscarValidador*: Este servicio se encarga de ejecutar la modificación de validador de informe en la Blockchain.

Indicar que esta API se publica a través de Https y realiza verificación del certificado cliente, de modo que se implementa Mutual TLS. Se

desarrolla empleando *express.js* que se trata de una infraestructura de aplicaciones Web *Node.js.*

Por otro lado, los servicios publicados en la API, cuenta con los siguientes scripts de Node.js que nos permiten realizar operaciones de mantenimiento y administración de la blockchain:

- *enrollAdmin.js*: Registro de usuario administrador en la organización HCE. Se emplea para realizar la configuración inicial de la Blockchain por lo que no se encuentra publicado a través de la API Rest.
- *Registeruser.js*: Registro de usuario en la organización HCE. Se emplea para realizar la configuración inicial de la Blockchain por lo que no se encuentra publicado a través de la API Rest.
- *enrollAdmintPacient.js*: Registro de administrador en la organización Paciente. Se emplea para realizar la configuración inicial de la Blockchain por lo que no se encuentra publicado a través de la API Rest.
- *registerFacultativo.js*: Registro de usuario en la organización Paciente. Se emplea para realizar la configuración inicial de la Blockchain por lo que no se encuentra publicado a través de la API Rest.
- *inserciones.js*: Permite la inserción de un nuevo informe en la Blockchain. Se emplea para realiza pruebas de validación por lo que no se encuentra publicado en la API.
- *consultas.js*: Incluye las operaciones de búsqueda con y sin parámetros en la Blockchain.
- *Actualizaciones.js*: Incluye la operación de actualización de validador de los informes de la Blockchain.

4.4. Front-end

El Front-end se desarrolla mediante React.js. Se ha llevado a cabo mediante el desarrollo de diferentes componentes cada uno de los cuales se emplea bajo diferentes situaciones. En la siguiente figura podemos observar la pantalla de Login:

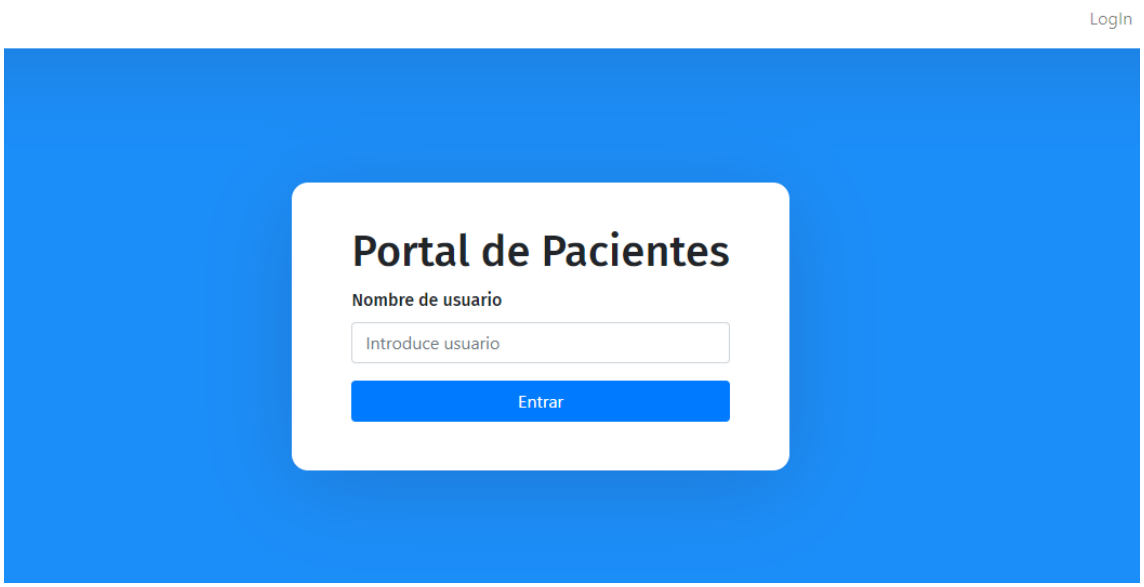


Figura 13: *Login del Portal de Pacientes*

Una vez realizado el Login, se muestra en la parte superior derecha el usuario que se encuentra logueado en el sistema. Además, tal y como se muestra en la siguiente figura, se presenta una tabla sin resultados cargados y un formulario para realizar búsquedas:

Figura 14: *Formulario de búsqueda*

En la siguiente figura, podemos ver la pantalla correspondiente a una búsqueda, realizada por un Facultativo, sin especificar ningún filtro donde se muestran todos los informes independientemente del paciente al que pertenezcan:

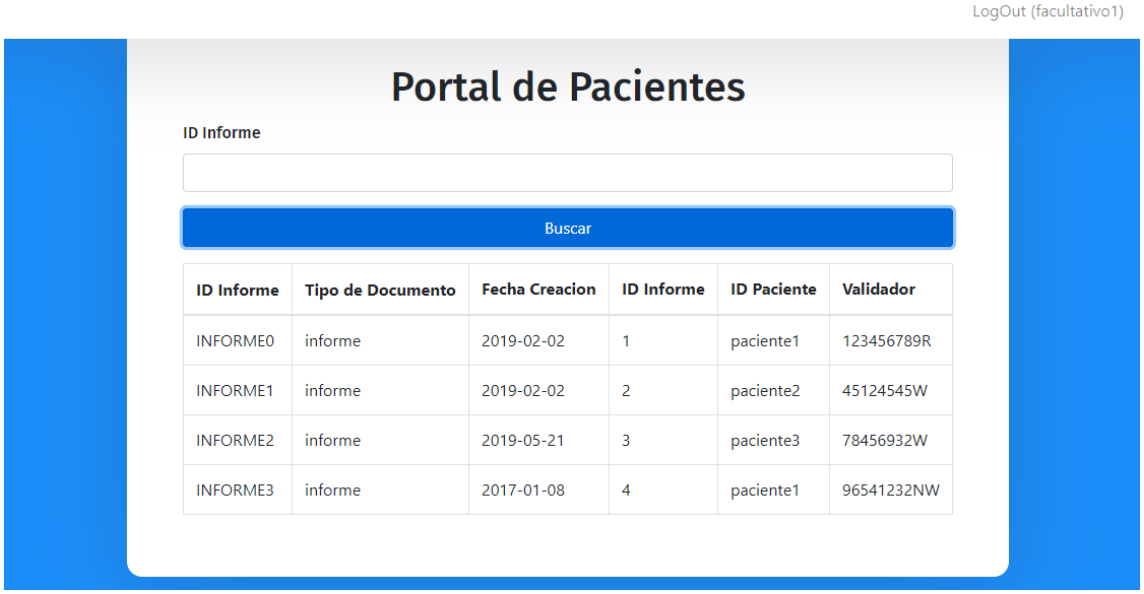


Figura 15: *Resultados de búsqueda de Facultativo*

Por último, en la siguiente figura se muestra los resultados de una búsqueda realizada por el Paciente1, y que muestra tan sólo sus informes:

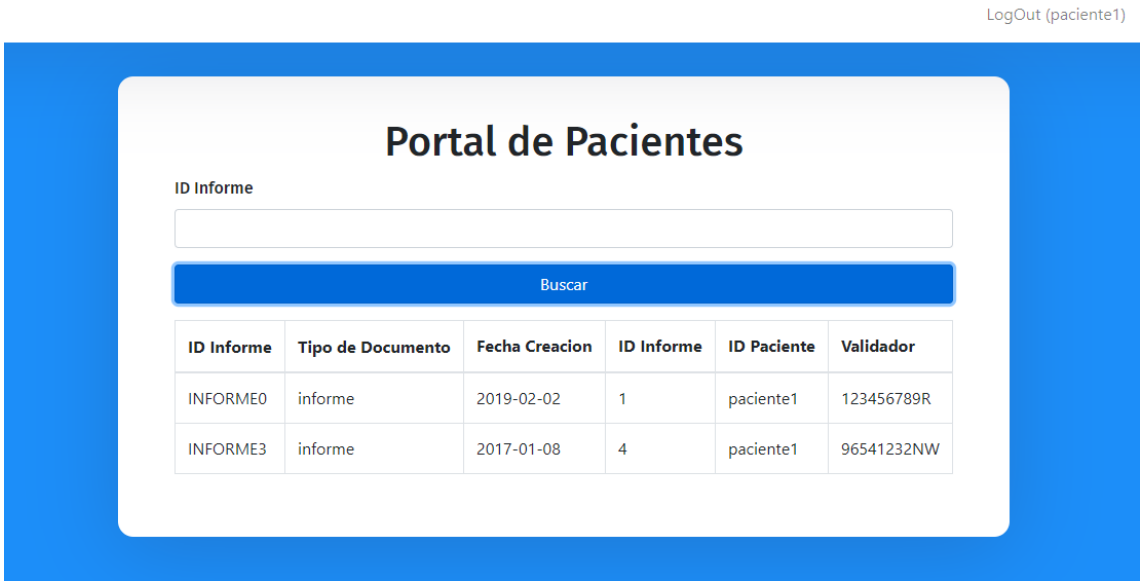


Figura 16: *Resultado de búsqueda de Paciente*

4.5. Valoración del prototipo obtenido

El presente trabajo arroja resultados positivos, ya que se demuestra que es posible emplear la tecnología Blockchain en un ámbito en el que es necesario gestionar información sensible. Por un lado, Blockchain soporta las tecnologías necesarias, como Mutual TLS, para el uso de esta información de forma segura, y por otro lado, dispone de los elementos necesarios, como el registros de transacciones y la posibilidad de gestionar permisos en función del usuario, para cumplir con los requerimientos de la RGPD.

Con el objetivo de que este prototipo disponga de una funcionalidad completa, se requiere la ampliación del sistema de cara a disponer de un repositorio documental en el que almacenar los documentos. El almacenamiento de los documentos en la propia Blockchain es inviable debido al impacto que esto tendría en el rendimiento de la misma.

En cuanto a las limitaciones, indicar que la implementación del prototipo actual, usando reactjs, no permite la autenticación del cliente mediante el certificado instalado en el navegador, debido a que reactjs no lo soporta de forma nativa. Para solventar este problema, sería necesario emplear otra tecnología en el front-end que si que lo soporte, o desarrollar una libería javascript que se encargue de gestionar el envío del certificado cliente en las llamadas a la API Rest. En todo caso, el servidor API Rest desarrollado si que soporta la validación de los clientes por certificado. Esta funcionalidad ha sido validada mediante la herramienta Postman.

Otra limitación del sistema desarrollado, es que no soporta una integración nativa con Hyperledger Fabric por lo que no hay una interacción directa desde el Front-end con la Blockchain. Se ha tenido que crear una API Rest para realizar la interacción con Hyperledger Fabric. La solución de emplear una API Rest es correcta y encaja dentro de la línea de tecnologías actuales para comunicarse con el Front-end, pero una integración directa sería una solución más elegante.

Por último, en cuanto a Hyperledger Fabric como Blockchain, destacar que la metodología de desarrollo propuesta es mejor que Hyperledger Composer, ya que nos permite conocer mejor los detalles de implementación y configuración de la Blockchain. Esto posibilita una mejor adaptación de la red de Blockchain a nuestras necesidades concretas.

5. Conclusiones

El presente trabajo me ha permitido obtener conocimientos teóricos acerca de la tecnología Blockchain, además de conocimientos técnicos sobre una implementación concreta de la misma, como es Hyperledger Fabric. Mediante el desarrollo del sistema he podido experimentar con esta tecnología y sobre su posible aplicación a un caso de uso real. He podido trabajar con otras tecnologías como son reactjs y node.js con las que no había trabajado nunca.

Considero que se han logrado todos los objetivos propuestos, si bien es cierto, que se ha tenido que modificar el alcance del sistema propuesto inicialmente debido a la falta de tiempo, que a su vez tuvo como origen la falta de experiencia en las tecnologías empleadas, por lo que se requirió más tiempo del planificado para el desarrollo. En cuanto a la planificación se han cumplido todos los hitos especificados.

Respecto a la tecnología Blockchain empleada resaltar que puede ser una tecnología válida para su uso en un entorno real. A pesar de ello, antes acometer una solución de este tipo habría que realizar un estudio para determinar si el sistema cuenta con la escalabilidad necesaria para soportar la carga asociada a una solución de este tipo.

Por lo que se refiere al propio desarrollo del sistema, destacar que se trata de un entorno con múltiples elementos distribuidos y que tanto su integración como la depuración de errores requirieron de más tiempo del esperado. En este sentido, el hecho de haber tomado como punto de partida una solución existente en la documentación de Hyperledger como base, ha minimizado este impacto en nuestro sistema.

5.1. Trabajo futuro

Una de las líneas de trabajo futuro podría ser el desarrollo de una librería javascript para la interacción con la red de Hyperledger Fabric directamente. Por hacer una analogía, se trataría de desarrollar una libería como Web3.js, que permite la interacción directamente con una red Blockchain Ethereum.

Otras de las posibilidades de trabajo futuro, podría ser extender el sistema construido de modo que se permitiera el acceso a diferentes repositorios para los mismos usuarios, así, un facultativo de una organización podría acceder a los informes de un paciente de otro repositorio. Esto sería una forma de establecer un mecanismo de segunda opinión entre profesionales de diferentes organizaciones y además, se lograría interconexión entre diferentes repositorios (Historias Clínicas Electrónicas). Está línea de trabajo debería explorar la posibilidad de que los pacientes fueran los dueños de sus datos clínicos desde el punto de vista técnico, y por lo tanto, pudieran conceder acceso a toda aquella organización o profesional que ellos decidieran sin necesidad de intervención de una tercena parte.

6. Bibliografía

[1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System | Satoshi Nakamoto Institute,” 2008.

[2] S. Haber and W. Scott Stornetta, “How to time-stamp a digital document,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1991.

[3] “Sistemas descentralizados.” [Online]. Available: https://en.wikipedia.org/wiki/Decentralised_system#/media/File:Decentralization_diagram.svg. [Accessed: 07-Dec-2019].

[4] M.-H. Kuo, A. W. Kushniruk, and E. Borycki, “A comparison of national health data interoperability approaches in Taiwan, Denmark and Canada,” *Electron. Healthc.*, vol. 10, no. 2, pp. 18–29, 2011.

[5] “Hyperledger Docs.” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/. [Accessed: 15-Nov-2019].

[6] “Wikipedia - NPM.” [Online]. Available: https://en.wikipedia.org/wiki/Npm_(software). [Accessed: 03-Dec-2019].

[7] “Wikipedia - Hash Function.” [Online]. Available: https://en.wikipedia.org/wiki/Hash_function. [Accessed: 04-Oct-2019].

[8] “Wikipedia - Docker.” [Online]. Available: https://es.wikipedia.org/wiki/Docker_(software). [Accessed: 09-Nov-2019].

[9] “Wikipedia - gRPC.” [Online]. Available: https://es.wikipedia.org/wiki/GRPC. [Accessed: 06-Dec-2019].

[10] “Wikipedia - nodejs.” [Online]. Available: https://en.wikipedia.org/wiki/Node.js. [Accessed: 01-Dec-2019].

[11] “Expressjs.” [Online]. Available: https://expressjs.com/es/. [Accessed: 01-Dec-2019].

[12] “RGPD.” [Online]. Available: https://es.wikipedia.org/wiki/Reglamento_General_de_Protección_de_Datos.

[13] “Reactjs.” [Online]. Available: https://es.reactjs.org/docs/getting-started.html. [Accessed: 05-Dec-2019].

[14] “Wikipedia - Python.” [Online]. Available: https://es.wikipedia.org/wiki/Python. [Accessed: 10-Dec-2019].

[15] “Client Certificates.” [Online]. Available: https://medium.com/@sevcsik/authentication-using-https-client-certificates-3c9d270e8326. [Accessed: 16-Dec-2019].

[16] “IPFS.” [Online]. Available: https://es.wikipedia.org/wiki/Sistema_de_archivos_interplanetarios. [Accessed: 21-Oct-2019].

[17] “Wikipedia - Ethereum.” [Online]. Available: https://en.wikipedia.org/wiki/Ethereum. [Accessed: 03-Oct-2019].

[18] “Hyperledger Composer.” [Online]. Available: https://hyperledger.github.io/composer/latest/. [Accessed: 20-Oct-2019].

[19] “Wikipedia - Bitcoin.” [Online]. Available: https://en.wikipedia.org/wiki/Bitcoin. [Accessed: 02-Oct-2019].

[20] “Wikipedia - Blockchain.” [Online]. Available: https://en.wikipedia.org/wiki/Blockchain. [Accessed: 02-Oct-2019].

[21] “React bootstraptable2.” [Online]. Available: https://react-bootstrap-table.github.io/react-bootstrap-table2/docs/about.html.

[22] “Hyperledger Docs.” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/. [Accessed: 15-Nov-2019].

[23] “Hyperledger Fabric.” [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/release-1.4/. [Accessed: 21-Oct-2019].

[24] “Ethereum.” [Online]. Available: https://www.ethereum.org. [Accessed: 07-Oct-2019].

[25] “Alastria.” [Online]. Available: https://alastria.io/. [Accessed: 14-Oct-2019].

[26] “Aplicación descentralizada.” [Online]. Available: https://www.computerhope.com/jargon/d/decentral.htm. [Accessed: 07-Dec-2019].

7. Anexos

7.1. Instalación y configuración de Hyperledger Fabric

7.1.1. Entorno

Hyperledger Fabric será configurado en una máquina con una distribución Ubuntu 18.04. Esta a su vez estará alojada en una máquina virtual mediante Oracle VirtualBox v.6.0.

7.1.2. Prerrequisitos

Antes iniciar la instalación de Hyperledger Fabric debemos realizar la instalación de los siguientes componentes:

- CURL: Proyecto software que contiene una biblioteca y un intérprete de comandos orientado a la transferencia de archivos. Se realiza su instalación mediante el siguiente comando:

```
sudo apt-get install curl
```

- NVM (Node Version Manager): Nos permite mantener instaladas diferentes versiones de Node en el mismo sistema pudiéndolas cambiar bajo demanda.

```
sudo apt-get install curl curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.0/install.sh | bash

nvm install 12.13/8.9.4
```

- Docker: Proyecto de código abierto que permite automatizar el despliegue de aplicaciones dentro de contenedores software.

Esto proporciona una capa de abstracción más y nos permite lograr la virtualización de aplicaciones en diferentes sistemas operativos.

```
sudo apt install docker.io
```

- NPM (Node Package Manager): Se trata de un sistema de gestión de paquetes para node.js y un entorno de ejecución javascript.

```
npm install npm@5.6.0 –g
```

- build-essentials: Paquete que contiene los elementos necesarios para generar paquetes para distribuciones GNU Linux.

```
sudo apt install build-essential
```

- gRPC: Sistema de llamada a procedimiento remoto (RPC) de código abierto inicialmente desarrollado por Google.

```
npm install grpc 1.24.2
```

- docker compose: Herramienta para la definición y ejecución de aplicaciones multi-contenedores.

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o
/usr/local/bin/docker-compose

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

- Python (v.2.7.15): Lenguaje de programación multiparadigma administrado por la Python Software Foundation. Cuenta con una licencia de código abierto.

```
sudo apt-get install python
```

- Go: Lenguaje de programación desarrollado por Google concurrente y compilado que está inspirado en la sintaxis de C.

```
wget https://dl.google.com/go/go1.11.linux-amd64.tar.gz

sudo tar -xvf go1.11.linux-amd64.tar.gz
```

7.1.3. Instalación de Hyperledger Fabric

Es necesario ejecutar los siguientes es comandos para llevar a cabo la instalación de Hyperledger Fabric:

```
curl -sS https://raw.githubusercontent.com/hyperledger/fabric/master/scripts/bootstrap.sh

chmod +x ./scripts/bootstrap.sh

./scripts/bootstrap.sh
```

7.1.4. Script de arranque de la Blockchain HCE_Networork

```
#!/bin/bash
#
# Copyright IBM Corp All Rights Reserved
#
# SPDX-License-Identifier: Apache-2.0
#
# Exit on first error
set -e
```

```
# don't rewrite paths for Windows Git Bash users
export MSYS_NO_PATHCONV=1
starttime=$(date +%s)
CC_SRC_LANGUAGE=${1:-"javascript"}
CC_SRC_LANGUAGE=`echo "$CC_SRC_LANGUAGE" | tr [:upper:] [:lower:]`

CC_RUNTIME_LANGUAGE=node # chaincode runtime language is node.js
CC_SRC_PATH=/opt/gopath/src/github.com/chaincode/portal/javascript

# clean the keystore
rm -rf ./hfc-key-store

# launch network; create channel and join peer to channel
cd ../HCE_Network



CONFIG_ROOT=/opt/gopath/src/github.com/hyperledger/fabric/peer
ORG1_MSPCONFIGPATH=${CONFIG_ROOT}/crypto/peerOrganizations/HCE.exa
mple.com/users/Admin@HCE.example.com/msp
ORG1_TLS_ROOTCERT_FILE=${CONFIG_ROOT}/crypto/peerOrganizations/HCE.e
xample.com/peers/peer0.HCE.example.com/tls/ca.crt
ORG2_MSPCONFIGPATH=${CONFIG_ROOT}/crypto/peerOrganizations/paciente.
example.com/users/Admin@paciente.example.com/msp
ORG2_TLS_ROOTCERT_FILE=${CONFIG_ROOT}/crypto/peerOrganizations/pacie
nte.example.com/peers/peer0.paciente.example.com/tls/ca.crt
ORDERER_TLS_ROOTCERT_FILE=${CONFIG_ROOT}/crypto/ordererOrganizations
/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-
cert.pem

set -x

echo "Instantiating smart contract on mychannel"
docker exec \
  -e CORE_PEER_LOCALMSPID=HCEMSP \
  -e CORE_PEER_MSPCONFIGPATH=${ORG1_MSPCONFIGPATH} \
  cli \
  peer chaincode instantiate \
    -o orderer.example.com:7050 \
```

```
    -C mychannel \
    -n portal \
    -l "$CC_RUNTIME_LANGUAGE" \
    -v 1.0 \
    -c '{"function":"initLedger","Args":[]}'  \
    -P "AND('HCEMSP.member','PacienteMSP.member')" \
    --tls \
    --cafile ${ORDERER_TLS_ROOTCERT_FILE} \
    --peerAddresses peer0.HCE.example.com:7051 \
    --tlsRootCertFiles ${ORG1_TLS_ROOTCERT_FILE}

echo "Waiting for instantiation request to be committed ..."
sleep 10



echo "Submitting initLedger transaction to smart contract on mychannel"
echo "The transaction is sent to all of the peers so that chaincode is built before
receiving the following requests"
docker exec \
  -e CORE_PEER_LOCALMSPID=HCEMSP \
  -e CORE_PEER_MSPCONFIGPATH=${ORG1_MSPCONFIGPATH} \
  cli \
  peer chaincode invoke \
    -o orderer.example.com:7050 \
    -C mychannel \
    -n portal \
    -c '{"function":"initLedger","Args":[]}' \
    --waitForEvent \
    --tls \
    --cafile ${ORDERER_TLS_ROOTCERT_FILE} \
    --peerAddresses peer0.HCE.example.com:7051 \
    --peerAddresses peer1.HCE.example.com:8051 \
    --peerAddresses peer0.paciente.example.com:9051 \
    --peerAddresses peer1.paciente.example.com:10051 \
    --tlsRootCertFiles ${ORG1_TLS_ROOTCERT_FILE} \
    --tlsRootCertFiles ${ORG1_TLS_ROOTCERT_FILE} \
    --tlsRootCertFiles ${ORG2_TLS_ROOTCERT_FILE} \
    --tlsRootCertFiles ${ORG2_TLS_ROOTCERT_FILE}
```

```
set +x

cat <<EOF

Total setup execution time : $(($(date +%s) - starttime)) secs ...

EOF
```

7.2. Instalación de entorno de desarrollo React.js

Para la instalación del entorno para el desarrollo del Front-end se lleva a cabo la configuración de un proyecto reactjs siguiendo los siguientes pasos:

1. Instalación del paquete *create-react-app* y dependencias:

```
npm install -g create-react-app

npm install body-parser

npm install concurrently
```

2. *Visual Code*: Es un editor de código fuente. Se puede obtener el instalador en la Web del propio editor.

   https://code.visualstudio.com/

7.3. Smart Contracts

Se muestra a continuación el código correspondiente al Chaincode *portal.js:*

```
/*
 * SPDX-License-Identifier: Apache-2.0
```

```
 */

'use strict';

const { Contract } = require('fabric-contract-api');
const ClientIdentity = require('fabric-shim').ClientIdentity;

class Portal extends Contract {

constructor() {
    super();
 }

 async instantiate(ctx) {
    // No implementation required with this example
    // It could be where data migration is performed, if necessary
    console.log('Instantiate the contract');
 }

   async initLedger(ctx) {
     console.info('============= START : Initialize Ledger
===========');
     const informes = [
                    {
          id_informe: '1',
          fechaCreacionInforme: '2019-02-02',
          validadorInforme: '123456789R',
          paciente: 'paciente1',
        },
        {
          id_informe: '2',
          fechaCreacionInforme: '2019-02-02',
          validadorInforme: '45124545W',
          paciente: 'paciente2',
        },
                    {
          id_informe: '3',
          fechaCreacionInforme: '2019-05-21',
```

```
            validadorInforme: '78456932W',
            paciente: 'paciente3',
        },
                    {
            id_informe: '4',
            fechaCreacionInforme: '2017-01-08',
            validadorInforme: '96541232NW',
            paciente: 'paciente1',
        },

    ];

     for (let i = 0; i < informes.length; i++) {
        informes[i].docType = 'informe';
        await ctx.stub.putState('INFORME' + i, Buffer.from(JSON.stringify(informes[i])));
        console.info('Added <--> ', informes[i]);
     }
     console.info('============ END : Initialize Ledger ==========');
  }

  async queryInforme(ctx, idInforme) {

     const informeAsBytes = await ctx.stub.getState(idInforme); // get informe del chaincode state
          const allResults = [];
     if (!informeAsBytes || informeAsBytes.length === 0) {
        throw new Error(`${idInforme} does not exist`);
     }

          const Key = idInforme;
     let Record;
     try {
        let cid = new ClientIdentity(ctx.stub);
        if (cid.assertAttributeValue('rolUsuario', 'facultativo')) {
            Record = JSON.parse(informeAsBytes.toString('utf8'));
            allResults.push({ Key, Record });
```

```
        }else{
                let informe = JSON.parse(informeAsBytes.toString('utf8'));

            if(cid.assertAttributeValue('idUsuario', informe.paciente.toString())){
                Record = JSON.parse(informeAsBytes.toString('utf8'));
                allResults.push({ Key, Record });
            }
        }

    } catch (err) {
        console.log(err);
        Record = informeAsBytes.toString('utf8');
    }

      return JSON.stringify(allResults);
  }

  async createInforme(ctx, idInforme, fechaCreacionInforme, validadorInforme,
paciente) {
    console.info('============= START : Crear Informe
===========');

    const informe = {
        idInforme,
        docType: 'informe',
        fechaCreacionInforme,
        validadorInforme,
        paciente,
    };

    await ctx.stub.putState(idInforme, Buffer.from(JSON.stringify(informe)));
    console.info('============= END : Crear Informe ===========');
  }

  async queryAllInformes(ctx) {
    const startKey = 'INFORME0';
    const endKey = 'INFORME999';
```

```
        const iterator = await ctx.stub.getStateByRange(startKey, endKey);

        const allResults = [];
        while (true) {
            const res = await iterator.next();

            if (res.value && res.value.value.toString()) {
                console.log(res.value.value.toString('utf8'));

                const Key = res.value.key;
                let Record;
                try {
                    let cid = new ClientIdentity(ctx.stub);
                    if (cid.assertAttributeValue('rolUsuario', 'facultativo')) {
                        Record = JSON.parse(res.value.value.toString('utf8'));
                        allResults.push({ Key, Record });
                    }else{
                        let informe = JSON.parse(res.value.value.toString('utf8'));
                        if(cid.assertAttributeValue('idUsuario', informe.paciente)){
                            Record = JSON.parse(res.value.value.toString('utf8'));
                            allResults.push({ Key, Record });
                        }
                    }
                } catch (err) {
                    console.log(err);
                    Record = res.value.value.toString('utf8');
                }

            }
            if (res.done) {
                console.log('end of data');
                await iterator.close();
                console.info(allResults);
                return JSON.stringify(allResults);
            }
        }
    }
```

```
    async actualizarValidadorInforme(ctx, idInforme, nuevoValidador) {
        console.info('============= START : actualizarValidadorInforme
===========');

        const informeAsBytes = await ctx.stub.getState(idInforme); // get informe
de la chaincode state
        if (!informeAsBytes || informeAsBytes.length === 0) {
            throw new Error(`${idInforme} does not exist`);
        }
        const informe = JSON.parse(informeAsBytes.toString());
        let cid = new ClientIdentity(ctx.stub);
        if (cid.assertAttributeValue('rolUsuario', 'facultativo')) {
            informe.validadorInforme = nuevoValidador;
        }

        await ctx.stub.putState(idInforme, Buffer.from(JSON.stringify(informe)));
        console.info('============= END : actualizarValidadorInforme
===========');
    }

}

module.exports = Portal;
```

7.4. API Rest de Hyperledger

Se muestra en el presente apartado el código correspondiente a la API Rest de Hyperledger desarrollada:

```
'use strict';

const funcion = require('./consultas.js');
const funcion2 = require('./actualizaciones.js');
const express = require('express');
const bodyParser = require('body-parser');
var fs = require('fs')
```

```
var https = require('https')

const app = express();
const port = process.env.PORT || 5000;



app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

app.get('/api/test', async (req, res) => {
  var respuesta = await funcion.consultaInformes('facultativo1');
  res.setHeader('Content-Type', 'application/json');
  res.json(respuesta);
  res.send({ express: `I received your POST request. This is what you sent me: ${respuesta}` });
});

app.get('/api/test2', async (req, res) => {

  const cert = req.connection.getPeerCertificate();
  if (req.client.authorized) {
        res.send(`Hello ${cert.subject.CN}, your certificate was issued by ${cert.issuer.CN}!`);
  }
  else if (cert.subject) {
        res.status(403).send(`Sorry ${cert.subject.CN}, certificates from ${cert.issuer.CN} are not welcome here.`);
  } else {
        res.status(401).send(`Sorry, but you need to provide a client certificate to continue.`);
  }

  res.setHeader('Content-Type', 'application/json');
  res.json(respuesta);
  res.send({ express: `I received your POST request. This is what you sent me: ${respuesta}` });
});
```

```
app.post('/api/buscarInformes', async (req, res) => {

  const cert = req.connection.getPeerCertificate();
  if (req.client.authorized) {

    var usuario = req.body.usuario;
    var idInforme = req.body.post;
    if(idInforme == null || idInforme == ''){
        var respuesta = await funcion.consultaInformes(usuario);
    }
    else{
        var respuesta = await funcion.consultaInformesPaciente(usuario,idInforme);
    }

    res.setHeader('Content-Type', 'application/json');
    res.json(respuesta);
    res.status(200);
    res.send({ express: `I received your POST request. This is what you sent me:`
});

  }
  else if (cert.subject) {
    console.log("Autorizado:111");
    res.status(403).send(`Sorry ${cert.subject.CN}, certificates from
${cert.issuer.CN} are not welcome here.`);
  } else {
    console.log("Autorizado:222");
    res.status(401).send(`Sorry, but you need to provide a client certificate to
continue.`);
  }

});



app.post('/api/modificarValidador', async (req, res) => {
  if (req.client.authorized) {
```

```
    var usuario = req.body.usuario;
    var idInforme = req.body.post;
    var nuevoValidador = req.body.nuevoValidadorInforme;
    var respuesta = await funcion2.actualizarValidadorInforme(usuario,idInforme,nuevoValidador);

    res.setHeader('Content-Type', 'application/json');
    res.json(respuesta);
    res.status(200);
    res.send(`I received your POST request. This is what you sent me: ${req.body.post}`,
    );
  }else if (cert.subject){
    res.status(403).send(`Sorry ${cert.subject.CN}, certificates from ${cert.issuer.CN} are not welcome here.`);
  } else {
    res.status(401).send(`Sorry, but you need to provide a client certificate to continue.`);
  }

});

https.createServer({
  key: fs.readFileSync('server_key.pem'),
  cert: fs.readFileSync('server_cert.pem'),
  requestCert: true,
  rejectUnauthorized: false,
  ca: [ fs.readFileSync('server_cert.pem') ]
}, app)
.listen(5000, function () {
  console.log('Example app listening on port 5000! Go to https://localhost:5000/')
})
```