



UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

Registros Médicos Permisados y Distribuidos a través de Hyperledger Fabric e InterPlanetary Filesystem

Trabajo Especial de Grado presentado ante la ilustre
Universidad Central de Venezuela por
Br. Alexander Yammine
Para optar al título de Licenciado en Computación

Tutores:
Prof. Antonio Rueda
Profa. Mercy Ospina

Caracas, Octubre 2018

UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

ACTA DE VEREDICTO

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación, para examinar el Trabajo Especial de Grado titulado **"Registros Médicos Permisados y Distribuidos a través de Hyperledger Fabric e InterPlanetary Filesystem"** y presentado por el bachiller: Br. Alexander Yammine Al Halabi C.I.: 23529986, a los fines de optar al título de Licenciado en Computación, dejamos constancia de lo siguiente:

Leído como fue dicho trabajo, por cada uno de los miembros del jurado, se fijó el día 11 de octubre del 2018, a las 4:00pm, para que el autor lo defendiera en forma pública, lo que se hizo en la sala de conferencias del Centro de Computación, mediante una presentación oral de su contenido, luego de lo cual respondieron a las preguntas formuladas. El prof. Antonio Rueda estuvo presente a través de una videoconferencia debido a que se encontraba fuera del país y los jurados presentes estuvieron de acuerdo. Una vez finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobar con la nota de 20 puntos.

En fe de lo cual se levanta la presente Acta, en Caracas el día 11 de Octubre del 2018.

px: M.H. Prof. Antonio Rueda (Tutor) Prof. Antonio Russoniello (Jurado Principal)	REPUBLICA DE VENEZUELA Facultad de Ciencias Escuela de Computación Universidad Central - Caracas	Profa. Mercy Ospina (Tutora) Prof. Miguel Astor (Jurado Principal)
--	--	--

Agradecimientos

Le agradezco a Dios por haberme dado la sabiduría y la fortaleza para que fuera posible alcanzar este triunfo.

A mis padres, mi padre Roberto Yammine, por apoyarme siempre y estar junto a mí cuando lo necesito, por ser un excelente padre, mi madre Jeanette Al Halabi, por ser la mejor madre del mundo, a mi hermana Lamia, por brindarme mucho apoyo.

A la ilustre Universidad Central de Venezuela y la Facultad de Ciencias, por permitirme desarrollar esta carrera universitaria. A todos los profesores, que me enseñaron y formaron a lo largo de cada período académico. A mis tutores Antonio Rueda y Mercy Ospina, por su gran apoyo y paciencia. A mis jurados, Antonio Russoniello y Miguel Astor, por el interés, apoyo y crítica necesarios para la culminación de este trabajo.

**Universidad Central de Venezuela.
Facultad de Ciencias
Escuela de Computación**

**Registros Médicos Permisados y Distribuidos a través de Hyperledger Fabric e InterPlanetary
Filesystem**

Autor: Br. Alexander R. Yammine Al Halabi

Tutores: Prof. Antonio Rueda

Profa. Mercy Ospina

Fecha: Octubre del 2018

Resumen

Un historial médico es una colección de información médica de un paciente, acumulada durante un periodo de tiempo. El tener acceso a esta información en el momento oportuno hace una gran diferencia para el tratamiento de un paciente, especialmente en una situación de emergencia. Este historial típicamente se almacena en medios impresos, exigiendo al médico que tenga espacio donde almacenarlos y pudiendo deteriorarse con el transcurso del tiempo, y aunque la digitalización es una posible solución puede generar problemas de accesos no autorizados y/o alteración de la información, además de generar registros digitales de tamaño considerable que pudieran ser complejos de manejar. Utilizando la tecnología de cadena de bloques, llamada *Blockchain*, podemos almacenar registros médicos donde es importante por un lado su disponibilidad inmediata, como es el caso del tipo de sangre o alergias, información de primera mano en caso de emergencias, y por otro lado su inmutabilidad. Como aplicación de blockchain, se utiliza Hyperledger Fabric, proyecto de código abierto para blockchain empresariales, la cual garantiza seguridad y control de acceso. Para el almacenamiento y transferencia de registros médicos de gran tamaño, como lo es la imagenología médica, se utiliza InterPlanetary Filesystem (IPFS) a consecuencia de los problemas de escalabilidad de la blockchain para el almacenamiento de archivos de gran tamaño. El objetivo de este trabajo es desarrollar una interfaz de programación de aplicaciones que sirva de intermediario entre ambas aplicaciones para crear, modificar y acceder al historial médico de un paciente, limitando el acceso del mismo por los profesionales tratantes del paciente, utilizando una metodología de desarrollo basado en componentes.

Palabras Claves: Blockchain, InterPlanetary Filesystem, Historial médico, Registros médicos, API

Índice de contenido

INTRODUCCIÓN	6
CAPÍTULO 1: PROBLEMA DE INVESTIGACIÓN	7
1.1 Planteamiento del problema	7
1.2 Objetivo general	8
1.3 Objetivos específicos	8
1.4 Justificación	8
1.5 Alcance	9
CAPÍTULO 2: MARCO TEÓRICO	10
2.1 Interplanetary Filesystem	10
2.1.1 Características de IPFS	10
2.1.2 Diseño de IPFS	13
2.2 Cadena de Bloques	16
2.2.1 Transacciones y firmas digitales	16
2.2.2 Consenso de nodos	17
2.2.3 Incentivo	18
2.2.4 Contratos Inteligentes	19
2.3 Hyperledger Fabric	20
2.3.1 Contratos Inteligentes	21
2.3.2 Seguridad	21
2.3.3 Arquitectura del sistema	22
2.3.3.1 Transacciones	22
2.3.3.2 Estructura de Datos	22
2.3.3.3 Tipos de nodos	23
2.3.3.4 Prueba de consenso	24
2.3.4 Hyperledger Explorer	24
2.4 Hyperledger Composer	25
2.4.1 Características de Composer	25
2.4.2 Composer Rest Server	27
2.5 Apache Kafka	28
2.5.1 Características	28
2.5.2 Arquitectura del sistema	28
2.6 Restful	29
2.7 Privacidad de la información médica de un paciente	31
2.7.1 Ley de ejercicio de medicina en Venezuela	31
2.7.2 HIPAA	32
CAPÍTULO 3: MARCO METODOLÓGICO	33

3.1 Ingeniería de software basada en componentes	33
3.2 Componentes y modelos de componentes	34
3.2.1 Características de un componente	34
3.2.2 Modelos de componentes	35
3.3 Composición de componentes	37
CAPÍTULO 4: MARCO APLICATIVO	38
4.1 Definición de los requerimientos	38
4.2 Diseño técnico	38
4.2.1 Diagrama de componentes	44
4.2.2 Diagrama de Secuencia	45
4.3 Definición de las herramientas	52
4.3.1 Node.js	52
4.3.2 Node package manager (npm)	52
4.3.3 Hyperledger Composer CLI	52
4.3.4 Hyperledger Explorer	53
4.3.4 Hyperledger Composer Playground	53
4.3.5 Postman	53
4.4 Entorno de trabajo	53
4.5 Desarrollo del componente	54
4.5.1 Interfaz HTTP	55
4.5.2 Resultados	59
4.5.2.1 Administrador	59
4.5.2.2 Perfil Personal	63
4.5.2.3 Perfil Público	66
4.5.2.4 Historial Médico	69
4.5.2.5 Paciente	72
4.5.2.6 Profesional	75
4.5.2.7 Transacciones	77
4.6 Pruebas Unitarias	82
ANEXO A - DIAGRAMAS DE SECUENCIA	85
ANEXO B - CONFIGURACIÓN DEL SERVICIO	96
B1. Instalar InterPlanetary Filesystem	96
B2. Instalar Hyperledger Fabric	97
B3. Instalar Hyperledger Composer	99
B3. Instalar la API	100
CONCLUSIONES	102
TRABAJO A FUTURO	103
BIBLIOGRAFÍA	104

INTRODUCCIÓN

El historial médico es una colección de documentos clínicos que proporciona información de la evolución del paciente (Ley 41/2002, 2018), esta colección permite el monitoreo y seguimiento de parte de un médico tratante, para el cual su obtención de forma inmediata resulta esencial. Además de proveer disponibilidad de la información, se debe garantizar la confidencialidad de los datos del paciente permitiendo su acceso solo por parte de entes permitidos, como por ejemplo, su médico tratante.

Gracias a la digitalización de estos registros es necesario utilizar mecanismos de protección de acceso a la información, ya que esta se va a encontrar distribuida en la red. Una posible solución a este problema está en la tecnología de cadena de bloques permisiva. En una blockchain permisiva es posible seleccionar a los usuarios que puedan tener acceso a la información dentro de la red y las operaciones que puedan realizar sobre ellas, por ejemplo, la lectura y la escritura. Además, en las blockchain permisivas se registra el historial de todas las operaciones de modificación del registro que se han realizado en la red.

El presente documento se encuentra estructurado en cuatro capítulos de la siguiente manera: Capítulo 1, se abarca el problema de investigación que nos llevó a elaborar este trabajo especial de grado y los objetivos para resolverlo. En el capítulo 2, marco teórico, donde se define cada uno de los conceptos, tecnologías y herramientas involucrados con el proyecto. En el capítulo 3, la metodología utilizada, buscando la solución utilizando el desarrollo de software basado en componentes y finalmente en el Capítulo 4, el desarrollo de la aplicación, donde se describe con mayor detalle la aplicación que se desarrolló en el presente trabajo especial de grado.

CAPÍTULO 1: PROBLEMA DE INVESTIGACIÓN

En este capítulo se plantea el problema a solventar y se explicará el contexto actual. También, se especifican los objetivos, la justificación y el alcance definidos para esta investigación.

1.1 Planteamiento del problema

Anualmente se generan millones de registros médicos, esta información típicamente se almacena en medios impresos, exigiendo al médico que tenga espacio donde almacenarlos y pudiendo deteriorarse estos con el transcurso del tiempo (Estrada, 2017). El uso regular de los registros médicos por distintos actores hace que se conviertan en redundantes y posiblemente conflictivos; por ejemplo, si el paciente cambia de médico tratante el registro vuelve a generarse o se tiene que emitir una copia del registro almacenado.

Los registros médicos digitales son una solución al problema de deterioro, debido a la característica de copia “perfecta” que se logra en las reproducciones digitales. A su vez la digitalización de los documentos permite la visualización y procesamiento de los datos y su acceso por parte de médicos tratantes. Sin embargo, la característica de copia perfecta de los registros digitales facilita su uso y propagación no autorizado, aspecto que resulta problemático debido a la discrecionalidad con la que deben manejarse los registros médicos (Gutiérrez, Núñez, Aguirre, y Delgado, 2014). La tecnología de contratos inteligentes en blockchain es una solución propuesta al problema de manejo discrecional de los datos en registros médicos digitales. En esencia, la tecnología blockchain corresponde a una lista de transacciones enlazadas por hashing, que no pueden ser modificadas sin dejar registro del cambio. Los contratos inteligentes son programas que se ejecutan en la blockchain, igualmente dejando registro de su actividad (Tar, 2017). A través de los contratos inteligentes se puede lograr un acceso permisado y discrecional de información médica sensible. Sin embargo, para el almacenamiento de estos registros digitales de tamaño considerable, como la imagenología médica, es necesario utilizar métodos distintos que blockchain. Aunque blockchain ha demostrado ser una tecnología que garantiza la integridad y seguridad en la información, al permitir que la *blockchain* almacene archivos de gran tamaño, se aumenta la latencia de los nodos que entran a la red y se reduce la velocidad en que se procesan las transacciones (Croman et al., 2016). Se han planteado

ideas complementarias y superiores a la blockchain para almacenar archivos de gran tamaño en la cadena de bloques, entre estos métodos destaca el InterPlanetary File System (IPFS). IPFS es un sistema similar al protocolo Bittorrent, ideado para la transferencia de información de tamaño considerable, como lo son los registros médicos (Benet, 2017).

1.2 Objetivo general

Desarrollar una aplicación de registros médicos distribuidos utilizando blockchain de Hyperledger Fabric e Interplanetary File System (IPFS) como red de entrega de contenidos para garantizar la disponibilidad de la información y su acceso permisado.

1.3 Objetivos específicos

1. Definir los requerimientos para el desarrollo de la aplicación
2. Implementar un sistema para la obtención de registros médicos en las plataformas de IPFS
3. Implementar un sistema para la asignación de permisos de acceso a los registros médicos en Hyperledger Fabric
4. Desarrollar una interfaz de programación de aplicaciones para el acceso a la plataforma de IPFS y Hyperledger Fabric
5. Realizar pruebas unitarias y funcionales de la API

1.4 Justificación

Los sistemas *peer-to-peer* (**P2P**) minimizan los puntos de fallo a través de la redundancia, es decir, si un nodo se encuentra caído, el sistema puede seguir prestando sus servicios, este tipo de sistemas no utiliza el esquema clásico cliente-servidor.

Para una red de hospitales, institutos o entes dedicados a prestar servicios médicos es muy importante tener acceso a la información más reciente de un paciente, dado que en un momento de emergencia (en el cual se pueden presentar complicaciones adicionales como la inconsciencia o la ignorancia del paciente), un profesional pueda tener acceso a la información médica del paciente que se haya hecho previamente, así sea en alguna entidad médica distinta a donde se presenta la emergencia. En esta red, las entidades tendrían acceso a toda la información pública de un paciente, como su tipo de sangre o las alergias y

manteniendo el acceso permitido a la información más delicada, como exámenes de sangre, resultados de pruebas de VIH o radiografías.

Esta información médica tiene la garantía de ser legítima, actualizada y válida, ya que utilizando tecnologías como blockchain e IPFS se almacenan los registros de todas las modificaciones que se realicen sobre la información médica.

1.5 Alcance

En este trabajo se desarrolla una interfaz de programación de aplicaciones (application programming interface, **API**) que sirve de intermediario entre el software de una entidad médica y los sistemas de IPFS y la blockchain. Esta aplicación tiene la funcionalidad de comunicarse con la red de IPFS al momento de introducir o extraer archivos y de comunicarse con el contrato inteligente instanciado en la blockchain de Hyperledger Fabric, para esto es necesario tener configurada una red de nodos de Hyperledger Fabric e IPFS.

CAPÍTULO 2: MARCO TEÓRICO

En este capítulo se explicarán los conceptos, las herramientas y las tecnologías que se utilizaron para el desarrollo de este proyecto. Primero se explicará qué es InterPlanetary Filesystem, cómo funciona y su diseño. De segundo, se introducirá la tecnología de cadena de bloques llamada Blockchain para dar paso a la explicación de Hyperledger Fabric, su diseño y características. De tercero, se explicará la herramienta llamada Hyperledger Composer, utilizada para creación y administración de Contratos inteligentes en Hyperledger Fabric. Seguidamente, se introduce la definición de una API Rest y por último, se mencionan y describen las leyes de privacidad de un paciente, tanto la HIPAA como la Ley de ejercicio de Medicina en Venezuela.

2.1 Interplanetary Filesystem

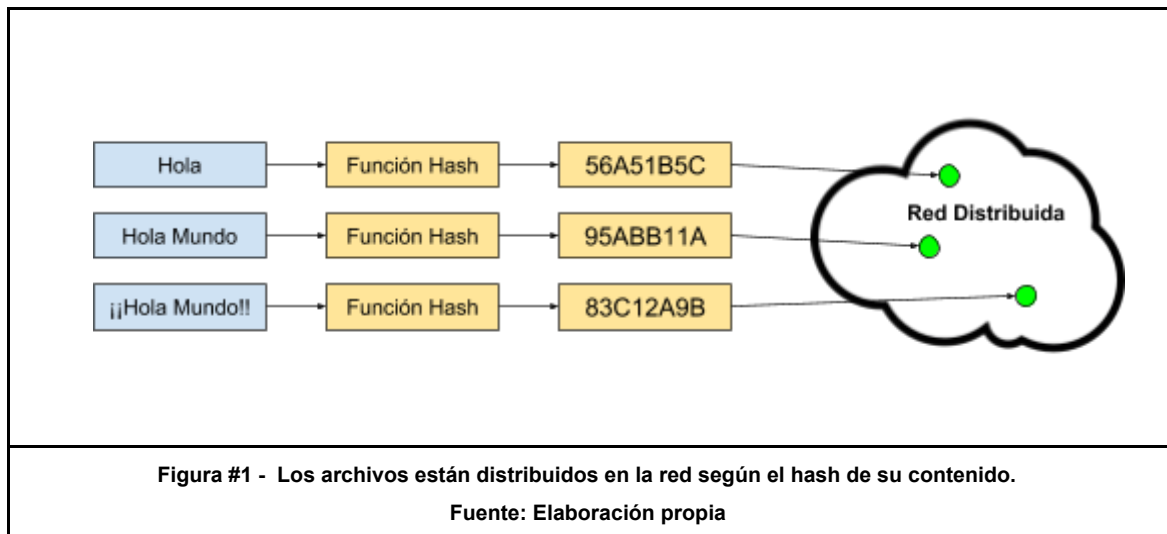
InterPlanetary File System (**IPFS**) es un sistema distribuido de archivos *peer to peer*. IPFS proporciona un modelo direccionado de almacenamiento de bloques con hiperenlaces dirigidos al contenido de alto rendimiento. Esto forma un *Merkle Directed Acyclic Graph* (grafo acíclico dirigido) generalizado, que es una estructura de datos sobre la cual se pueden construir sistemas de archivos versionados, cadenas de bloques e incluso una web permanente, en la que toda versión del contenido que alguna vez haya sido cargado persista. IPFS combina una tabla hash distribuida, un sistema incentivado de intercambio de bloques y un espacio de nombres autocertificados. IPFS es una red robusta, carente de puntos únicos de falla y los nodos presentes en ella no necesitan confiar entre sí (Benet, 2017)

2.1.1 Características de IPFS

Tablas de Hash Distribuidas

Una tabla *Hash* es una estructura de datos que almacena pares de clave y valor que permite consultar el valor asociado a cada clave. Una *Distributed Hash Table* es una extensión multinodo a las tablas hash que puede determinar el nodo responsable de almacenar los datos asociados a una clave dada. Cada nodo mantiene información (por ejemplo, la dirección IP) de un pequeño número de nodos (“vecinos”) en el sistema,

formando una red superpuesta con mensajes de enrutamiento para almacenar y recuperar claves, (Balakrishnan, Kaahoek, Karger, Morris, y Stoica, 2003) como se muestra en la **figura #1**.



Intercambio de bloques - *Bittorrent*

BitTorrent es un sistema de intercambio de archivos entre pares que coordina redes de nodos para cooperar entre sí y lograr la distribución de fragmentos de archivos. Las características más importantes de BitTorrent que se tomaron en cuenta en el desarrollo de IPFS son:

- El protocolo de intercambio de datos de *BitTorrent*, este utiliza una estrategia *tit-for-tat* que recompensa a los nodos que se cooperan entre sí y castiga a los nodos que solo descarguen de los demás.
- Los pares de *BitTorrent* rastrean la disponibilidad del archivo, priorizando el envío de piezas más raras primero, dejando las más comunes al final. Con esta estrategia, se reduce la probabilidad de que se pierdan fragmentos del archivo (Cohen, 2003).

Sistema de Control de Versiones - Git

Un Sistema de Control de Versiones proporciona registros sobre los cambios que se realicen sobre un archivo o un conjunto de archivos a lo largo del tiempo. El sistema de control de versiones Git proporciona un poderoso modelo del árbol Merkle DAG que captura

los cambios en un árbol del sistema de archivos de una manera amigable y distribuida (Benet, 2017).

- Los objetos inmutables representan Archivos (blob), Directorios (árbol) y Cambios (commit).
- Los objetos están dirigidos al contenido, mediante el hash criptográfico de sus contenidos.
- Los enlaces a otros objetos están incrustados, formando un Merkle DAG. Esto proporciona muchas propiedades útiles de integridad y flujo de trabajo.
- La mayoría de los metadatos de control de versiones (ramas, etiquetas, etc.) son simplemente referencias de punteros y, por lo tanto, de bajo costo para crear y actualizar.
- Los cambios de versión solo actualizan referencias o agregan objetos.
- Distribuir cambios de versiones de archivos a otros usuarios es simplemente transferir objetos y actualizar referencias remotas (Benet, 2017)

Sistema de archivos autocertificados - SFS

El *Self-Certifying File System (SFS)* es un sistema de archivos de red seguro y descentralizado para Internet. SFS proporciona un espacio de nombres para todos los archivos del mundo. Los usuarios puede acceder a sus archivos almacenados del nodo que más confían y que pueda estar en cualquier parte del mundo, además con saber el nombre del archivo, un usuario podrá compartirlo con cualquier otro. SFS introdujo una técnica para construir sistemas de archivos autocertificados, direccionando los sistemas de archivos remotos usando el siguiente esquema:

/sfs/Location:HostID

Location: Información que le dice al cliente donde buscar el servidor del sistema de archivos, ésta puede ser una dirección IP.

HostID: Información que le dice al cliente cómo certificar un canal seguro para ese servidor.

Para lograr una comunicación segura, todos los servidores SFS tienen una clave pública. El *HostID* es un hash criptográfico de la clave pública con el campo *Location*. El *HostID* permite a los clientes preguntar a los servidores por sus claves públicas y verificar la autenticidad de la respuesta. Conocer la clave pública de un servidor permite que un cliente se comunique de forma segura con ella. Por lo tanto, el nombre de un sistema de archivos SFS certifica su servidor (Mazieres y Kaashoek, 2000).

2.1.2 Diseño de IPFS

IPFS es un sistema *peer-to-peer* que une todas las ideas anteriormente mencionadas. IPFS presenta una nueva plataforma para desplegar aplicaciones web, y un nuevo sistema para distribuir y versionar datos de cualquier tamaño. Los nodos de IPFS almacenan objetos en el almacenamiento local y se conectan entre sí con otros nodos de la red para transferir estos objetos, dichos objetos representan archivos y otras estructuras de datos.

El protocolo IPFS se divide en una pila de sub-protocolos responsables de diferentes funcionalidades:

Identidades

Los nodos son identificados por su id, que es el hash de su clave pública. Los nodos almacenan su ID, clave pública y clave privada encriptadas por una firma digital, como se muestra en el **Código #1** en lenguaje Go.

```
type NodeId Multihash // El formato multihash almacena un encabezado
corto que especifica la función hash utilizada y la longitud en bytes del
hash
type Multihash []byte
type PublicKey []byte
type PrivateKey []byte
type Node struct {
    NodeId NodeId
    PubKey PublicKey
    PriKey PrivateKey
}
// Algoritmo para generar el ID del nodo
difficulty = <integer parameter>
n = Node{}
do {
    n.PubKey, n.PrivKey = PKI.genKeyPair()
    n.NodeId = hash(n.PubKey)
    p = count_preceding_zero_bits(hash(n.NodeId))
} while (p < difficulty)
```

Código #1 - Algoritmo para la generación de la clave pública (Mazieres y Kaashoek, 2000)

Para que haya una conexión exitosa entre pares, estos tienen que verificar su identidad, ambos pares intercambian sus claves públicas para comprobar su validez. Este procedimiento solo se aplica al conectarse por primera vez (Benet, 2017).

Enrutador

Los nodos de IPFS necesitan un sistema de enrutamiento para encontrar las direcciones de los demás nodos en la red y para encontrar a los demás nodos que pueden tener los objetos que necesitamos. IPFS logra esto usando un DHT basado en *S/Kademlia* y *Coral*. Los valores pequeños (igual o inferior a 1 KB) se almacenan directamente en el DHT. Para valores más grandes, el DHT almacena referencias, que son los Node IDs de pares que pueden engarce de almacenar el bloque (Benet, 2017).

Intercambio de Bloques - Protocolo *Bitswap*

En IPFS, la distribución de datos ocurre intercambiando bloques con sus pares utilizando un protocolo inspirado en *BitTorrent*: *BitSwap*. Los nodos de *Bitswap* buscan obtener un conjunto de bloques (objetos que se quiere descargar) y tienen otro conjunto de bloques que ofrecer (objetos que se quiere compartir). Bitswap opera como un mercado donde los nodos pueden obtener bloques de archivos que necesiten, independientemente si no se tratase del mismo archivo, es decir pueden estar intercambiando bloques de distintos archivos.

En muchos casos, la distribución de bloques no es complementaria, es decir que un nodo puede no tener un bloque que otro quiera. Para solucionar este caso, cuando un nodo no tenga piezas que le hayan sido solicitadas, comienza a buscar las piezas que sus pares necesiten, motivando a los nodos a difundir piezas raras, incluso si no están interesados en ellas (Benet, 2017).

Object Merkle DAG

DHT y Bitswap hacen que IPFS formen un robusto y rápido sistema Peer-to-Peer para la distribución y el almacenamiento de bloques. Adicionalmente, IPFS utiliza una estructura de datos llamada Merkle *DAG*, un grafo acíclico dirigido en donde los enlaces entre objetos son hashes criptográficos de los objetivos.

Esta es una generalización de la estructura de datos de Git. Merkle DAG proporciona a IPFS muchas propiedades bastante útiles, como:

- Direccionamiento de contenidos: Todo el contenido se identifica de manera única por su *multihash checksum*, incluyendo los enlaces.
- Resistencia a la manipulación: Todo contenido puede ser verificado con el *checksum*. Si un contenido es modificado o se corrompe, IPFS lo sabrá.
- Sin duplicados: Al igual que Bittorrent, los objetos están identificados por su contenido y no por su nombre, es decir, si varios objetos tienen el mismo contenido, solo se almacena una vez.

El Merkle DAG de IPFS es una forma flexible de almacenar datos. Los objetos de IPFS pueden navegarse con una API añadiendo una ruta, estas rutas funcionan de la misma manera que lo hace tradicionalmente UNIX y la web (Benet, 2017). La arquitectura de enlace del Merkle DAG permiten navegación única, por ejemplo una ruta completa en IPFS puede ser de la siguiente forma:

/ipfs/<hash-del-objeto>/<nombre-de-la-ruta-del-objeto>

Ejemplo:

/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt

Archivos

IPFS define un conjunto de objetos para modelar un sistema de archivos versionado sobre Merkle DAG (Benet, 2017). Este modelo de objeto es similar al de Git:

- Bloque: un bloque de datos de tamaño variable.
- Lista: una colección de bloques u otras listas.
- Árbol: una colección de bloques, listas u otros árboles.
- Commit: registra los cambios de versiones de un árbol.

Con IPFS es posible resolver el problema de escalabilidad de la cadena de bloques utilizándola para la transferencia y almacenamiento de archivos de gran tamaño. En el siguiente punto se explicará que es una Blockchain, cómo está diseñada y sus principales características que son importantes para el desarrollo de este trabajo.

2.2 Cadena de Bloques

Una cadena de bloques (conocida como Blockchain) es una base de datos distribuida, que almacena registros de transacciones realizadas. La cadena está compuesta por bloques enlazados, cuando se genera un nuevo bloque, este se enlaza al anterior. Este sistema está diseñado para que no se pueda realizar ninguna modificación sin traza, ya que cada bloque contiene almacenado el hash del bloque anterior, ya de querer modificar alguna transacción, esta modificación se verá reflejada en los bloques posteriores. Así mismo, se garantiza integridad de información entre los pares con un sistema de broadcast: cuando se realiza una transacción, esta es enviada a todos los nodos en la red para que sea registrada en todos (Champagne, 2014, p.11).

2.2.1 Transacciones y firmas digitales

Una firma digital es un mecanismo por el cual el receptor pueda verificar el remitente del mensaje. Para que esto sea posible, se utiliza un cifrado asimétrico basado en el algoritmo RSA, que genera una clave pública y una clave privada. La clave privada es confidencial, y se utiliza para cifrar el mensaje, mientras que la clave pública es utilizada para descifrar el mensaje (Stallings, 2005, p.256).

Un ejemplo de una firma digital asimétrica:

1. Carlos redacta un correo para María.
2. Carlos firma digitalmente el mensaje con su **clave privada**.
3. Carlos envía el mensaje firmado digitalmente a María a través de internet, por correo electrónico.
4. María recibe el mensaje firmado digitalmente y comprueba su autenticidad usando la **clave pública** de Carlos para descifrar el correo.
5. María ya puede leer el mensaje con total seguridad ya que ha sido Carlos el que escribió el correo.

En la *blockchain* de Bitcoin, las transacciones se generan cuando el dueño de una moneda desea transferirlo a otra persona, donde el dueño tendrá que firmar un mensaje que está compuesto por la clave pública del destinatario y el hash de la transacción anterior. El sistema puede comprobar que fué el dueño de la moneda quien firmó el mensaje utilizando su clave pública (Nakamoto, 2009).

2.2.2 Consenso de nodos

En las Blockchain públicas como Bitcoin, cualquier nodo puede entrar a la red y participar. Para decidir de forma segura quién será el nodo en agregar el siguiente bloque a la cadena, se tiene que pasar por un consenso para garantizar que todos los nodos en la red estén de acuerdo con el nuevo bloque que se va a agregar en la cadena, con esto se puede evitar comportamientos indeseados por si algún nodo desea dañar la cadena. Los dos algoritmos más utilizados para el consenso de nodos, son la prueba de trabajo (**PoW**) y la prueba de participación (**PoS**).

Prueba de Trabajo

Para este sistema se requiere que cada nodo realice un tipo trabajo de alto costo y que pueda ser verificado fácilmente. La prueba de trabajo es utilizada en la cadena de bloques de Bitcoin. Lo que se busca es que cada nodo encuentre un hash que cumpla un requisito, y es que este hash tenga una cierta cantidad de ceros a su izquierda, esta cantidad de ceros viene dada por la dificultad del algoritmo y se ajusta cada 2016 bloques para que el hash pueda ser resuelto en aproximadamente 9 minutos. El proceso de crear nuevos bloques a la cadena utilizando este algoritmo de consenso es llamado “minería”. Si en un mismo instante, se generan dos bloques, los nodos tendrán que almacenar el segundo bloque que les llegue, el sistema continuará normalmente hasta que llegue el bloque del “desempate”, es decir, el siguiente bloque definirá cuál de los dos anteriores bloques se quedará en la cadena, con esto se crea la premisa de “la cadena más larga, es la correcta”.

La prueba de trabajo también es considerada un sistema de votación determinando una nueva representación en cuanto a decisión por mayoría, si la mayoría estuviese basado por un voto por IP, podría ser alterada por un ente que genere varias IPs. En la prueba de trabajo se puede considerar un CPU como equivalente a un voto. Ya que la decisión de la mayoría, es representada por la cadena más larga. Si el mayor poder de cómputo es controlado por nodos honestos, la cadena crecerá más rápido y superará en extensión a cualquier cadena que esté compitiendo con ella. Como se mencionó anteriormente, para poder modificar un bloque anterior, un atacante tendrá que rehacer la prueba de trabajo del bloque y los posteriores a ese hasta superar la cadena formada por los nodos honestos. La probabilidad para que esto ocurra, se reduce exponencialmente por cada nuevo bloque generado en la cadena honesta (Nakamoto, 2009).

Prueba de Participación

Para la prueba de participación se aplica para cadenas de bloques que tengan una criptomoneda de forma nativa, el algoritmo selecciona pseudo-aleatoriamente a un nodo (validador) basado en la cantidad de monedas que tenga, mientras más monedas tenga retenido el nodo, mayor es la probabilidad de que sea seleccionado para agregar el siguiente bloque a la cadena (Ray, 2015).

Existen muchas variantes de este algoritmo, dos de ellas son la prueba de depósito y prueba de participación híbrida usada por *Peercoin*.

Para el consenso de la prueba de depósito, el algoritmo seleccionará a un validador cada intervalo de tiempo (Por ejemplo, cada 10 segundos), uno de los factores que se toma en cuenta antes de escoger al validador es la cantidad de monedas que tenga congelado en el momento, estas monedas congeladas están bloqueadas y no pueden ser usadas hasta que se generen cierta cantidad de bloques. Hay variantes de este algoritmo que aplica un tiempo de vejez, donde un depósito realizado hace un largo periodo de tiempo puede no tener el mismo impacto que los depósitos recientes. Por ejemplo, un tiempo de vejez donde después de 90 días, el depósito no tendrá validez (Narayanan, Bonneau, Felten, Miller y Goldfeder, 2016).

La prueba de participación usada por *Peercoin*, es una mezcla de la prueba de participación y la prueba de trabajo. Donde la complejidad del algoritmo de descifrado del hash, es inversamente proporcional a la cantidad de monedas que tenga retenido el validador. Es decir que mientras más monedas tenga un validador, más fácil será encontrar el hash vencedor de la prueba de trabajo, por ejemplo, si con 200 monedas pudiese encontrarse el hash en 2 días, con 400 monedas pudiese encontrarse en 1 día (King y Nadal, 2012).

2.2.3 Incentivo

Una de las características más importantes en una blockchain pública, es el beneficio que obtienen los nodos al momento de generar un nuevo bloque para la cadena. Este incentivo es comúnmente la generación de una nueva moneda y la obtención la comisión de las transacciones almacenadas en el nuevo bloque. Esto significa un incentivo a los nodos para que dediquen recursos de cómputo a la validación de transacciones y el mantenimiento de la red.

Un atacante pudiese reunir suficiente potencia de CPU para forzar a que los nodos honestos devuelvan una transacción que se había hecho anteriormente o agregar más monedas a su monedero, pero es más rentable apoyar la cadena honestamente que cometer un fraude, ya que atacando la cadena fácilmente podría desvalorizar su inversión en la criptomoneda y desperdiciar todo el dinero en hardware y energía eléctrica invertidos para realizar el ataque (Cohen, 2003).

2.2.4 Contratos Inteligentes

Varias cadenas de bloques tienen la capacidad de ejecutar códigos de programación. Estos scripts, llamados “contratos inteligentes” son protocolos especiales que ayudan a regular los contratos, están destinados a contribuir, verificar o implementar la negociación o el cumplimiento de los mismos. Los contratos inteligentes permiten realizar transacciones creíbles sin la necesidad de tener terceros intermediarios. El contrato inteligente contiene toda la información de los términos y condiciones, y ejecuta automáticamente las sentencias previstas. Habitualmente dentro de estos contratos inteligentes se definen activos y las operaciones sobre ellos, como por ejemplo un contrato para la venta de automóviles, donde el activo sea un automóvil y una de las operaciones sea el cambio de propietario (Tar, 2017).

Las propiedades de la Blockchain garantizan inmutabilidad de la información y registran todos los cambios del mismo en la cadena, aunque a pesar de estos beneficios no se puede permitir la libre entrada de nodos ni permitir que la información sea pública sino que se restrinja únicamente para las organizaciones participantes. A causa de esto, se propuso el uso de una Blockchain permisada, donde los nodos de la red se conocen y la información que se genere en la misma sea privada, donde únicamente los nodos de la red pueden tener acceso a ella.

2.3 Hyperledger Fabric

Hyperledger es un proyecto para avanzar con la tecnología blockchain en la industria, fundado por *Linux Foundation* en el 2015. Para no declarar un solo estándar de blockchain, se creó un enfoque colaborativo para desarrollar distintas tecnologías blockchain a través de un proceso comunitario, de código abierto y para su adopción como estándares a lo largo del tiempo (What is Hyperledger Fabric?,2017).

Hyperledger Fabric es uno de los proyectos pertenecientes a Hyperledger. Fabric es un sistema de código abierto y extensible para implementar y operar blockchain permisados, este tiene características en común con otras tecnologías blockchain, tiene un libro mayor (o de contabilidad, aquí es donde se almacenan las transacciones), utiliza contratos inteligentes y usa un sistema donde los participantes administran sus transacciones. Fabric es verdaderamente extensible para ejecutar aplicaciones distribuidas, admite protocolos de consenso modulares, lo que permite que el sistema se adapte a casos de uso particulares, los mecanismos de consenso se pueden activar o desactivar, los datos del libro mayor se pueden almacenar en múltiples formatos. Fabric es el primer sistema de cadena de bloques que ejecuta aplicaciones distribuidas (Contratos inteligentes) escritas en cualquier lenguaje de programación estándar y de uso general (Como GO, Java, Javascript, entre otros) y sin la dependencia de una criptomoneda nativa, como Ethereum (What is Hyperledger Fabric?,2017).

En lo que Fabric se diferencia de las otras tecnologías blockchain, es que este es privado y permisado, en lugar donde un sistema abierto permitiría que cualquier nodo con identidades desconocidas participen en la red, los miembros de una red en Hyperledger Fabric se inscriben a través de un proveedor de servicios de membresía (MSP), es decir que se necesitan permisos para entrar a una red en Fabric. Fabric tiene la capacidad de crear distintos canales, lo cual permite que un grupo de participantes cree un libro mayor separado, es decir, se tiene una cadena de bloques distinta para cada canal, esta opción es importante ya que algunos participantes pueden ser competidores y no se desea que tengan acceso a las transacciones que se hayan realizado con otros participantes, por ejemplo un precio especial que se realicen a otros participantes. Si dos participantes forman un canal, entonces únicamente estos tienen las copias del libro mayor de ese canal (What is Hyperledger Fabric?,2017).

Hyperledger Fabric tiene un subsistema que está compuesta por dos componentes, el estado del libro de contabilidad y el registro de las transacciones. El primer componente, define el estado del libro de contabilidad en un punto dado del tiempo, es la base de datos del libro mayor. Mientras el componente del registro de las transacciones, contiene los registros de todas las transacciones que dan como resultado el valor actual del estado del libro de contabilidad, es el historial de actualizaciones para este. Entonces el libro mayor está conformada por una base de datos que contiene el estado actual del libro y el historial de transacciones que se han realizado. Por defecto, Fabric utiliza LevelDB como base de datos donde almacena objetos clave-valor, aunque para ambientes de producción se recomienda utilizar la base de datos CouchDB (What is Hyperledger Fabric?,2017).

2.3.1 Contratos Inteligentes

En *Fabric* los contratos inteligentes son llamados “*chaincode*”, estos son piezas de código escritos en unos lenguajes permitidos, actualmente con Java o Go, estos se instalan en una red (previamente creada e instalada) y mediante un SDK (Kit de desarrollo de software) o CLI (línea de comandos de consola) se crea una interfaz para la interacción con el mismo. En el momento que se invocan estas aplicaciones, son ejecutadas de forma segura en contenedores “Docker” y en la mayoría de los casos, el chaincode interactúa es con la base de datos (por ejemplo, para consultarlo), y no con el registro de transacciones (Androulaki et al., 2018).

2.3.2 Seguridad

Dependiendo de las necesidades de la red, los participantes dentro de una red B2B (Empresa a empresa) pueden ser extremadamente sensibles sobre la cantidad de información que comparten. Fabric ofrece varias formas para garantizar la privacidad de los datos y el control de acceso. La primera, es separar la red en distintos canales, en donde dentro cada canal contenga los participantes que estén autorizados a ver los datos de los chaincodes que se implementen en ese canal. De segunda forma, dentro de un canal se pueden restringir los datos de entrada del chaincode solo al conjunto de nodos endosantes (estos nodos tienen la función de aprobar una transacción antes de que se ejecute) utilizando una configuración de visibilidad que determinara si los datos de entrada y salida del chaincode se incluirán en la transacción a enviar, pudiendo solo enviar los datos de

salida. De tercero, se pueden generar claves hash o cifrar los datos de entrada, si se generan claves hash, se deberá proporcionar otra entrada para compartir estos datos de entrada, y si se cifran, se deberá proporcionar un medio para compartir la clave de descifrado. De cuarto, se puede restringir el acceso de los datos a ciertos roles dentro de la organización, creando control de acceso en la lógica del chaincode. Por último, los datos del libro mayor se pueden cifrar a través del cifrado del sistema de archivos del nodo, y los datos en tránsito se cifran a través de TLS (Hyperledger Fabric FAQ, 2018).

2.3.3 Arquitectura del sistema

2.3.3.1 Transacciones

En Fabric existen dos tipos de transacciones, la primera, es la instanciación de un chaincode, esta suele tomar unos parámetros de entrada y su correcta instalación dentro de una red, esta se registra en la cadena de bloques. La segunda, son las transacciones de invocación, estas transacciones son operaciones de un chaincode previamente instanciada, estas operaciones son llamadas que corresponden a funciones implementadas en el chaincode. En Fabric, las transacciones de escritura son las que se registran en la cadena de bloques, las que no modifican el estado del libro mayor, no tienen porque registrarse, por ejemplo las operaciones de lectura (Architecture Explained, 2018).

2.3.3.2 Estructura de Datos

Estado de la cadena de bloques

Los estados de la cadena de bloques se modelan en la base de datos mediante objetos clave-valor versionado (**KVS**), en donde las claves son nombres y los valores son blobs (Binary Large Objects), estos son archivos de gran tamaño. Estas entradas son manipuladas por las aplicaciones (chaincodes) que se ejecutan en la cadena de bloques a través de las operaciones put y get. El estado se almacena persistentemente y se registran las actualizaciones del estado (Architecture Explained, 2018).

Libro Mayor

El libro mayor proporciona un historial de registros verificables de todos los cambios exitosos (transacciones válidas) y los intentos fallidos de cambiar el estado del mismo (transacciones no válidas) que ocurren durante las operaciones del sistema. El libro mayor es construido por “nodos orderer”, que como se mencionó anteriormente, estos bloques de transacciones están enlazados por el bloque anterior almacenando su hash, cada bloque contiene una matriz de transacciones válidas o no válidas totalmente ordenadas. Cada nodo (peer) contiene una copia de la cadena de bloques y también opcionalmente un subconjunto de nodos orderer (Architecture Explained, 2018).

2.3.3.3 Tipos de nodos

Los nodos en *Fabric* son entidades de comunicación en la cadena de bloques, cada uno cumple un rol dentro del sistema, estos pueden ejecutarse en el mismo servidor físico. Existen 3 tipos de nodos, el nodo cliente, el nodo *peer* y el nodo *orderer*.

Nodo cliente

Este representa el cliente final, y es el que se encarga de realizar las transacciones a los endosantes. Debe conectarse a cualquier nodo peer dentro de la red para comunicarse con la cadena de bloques, esto puede ser de varias maneras, utilizando un SDK para comunicarse con algún peer o usando la línea de comandos de consola (**CLI**).

Nodo peer

Cada peer mantiene una copia de la cadena de bloques, se encarga de confirmar las transacciones y actualizar el estado de la base de datos y la cadena de bloques, estas transacciones llegan de los nodos orderer en forma de bloques ordenados, además, los peer pueden tener un rol especial de peer endosante o simplemente ser un endosante. Cada peer tiene que validar una transacción antes de registrarla en la cadena de bloques, y Fabric permite crear políticas de endoso, para que una transacción sea validada correctamente. Las políticas de endoso definen las condiciones necesarias y suficientes para un endoso exitoso de una transacción (como un conjunto de firmas de endosantes), por ejemplo, para registrar una transacción de una chaincode, 2 de los 3 peers tienen que validarlo, o dar más privilegio a un peer de un conjunto, en donde si este valida la transacción, es condición suficiente para que se registre en la cadena de bloques.

Nodo orderer

Los nodos orderer forman un servicio de pedidos que proporciona garantías de entrega, este suministra un canal de comunicación compartido a nodos clientes y nodos peer que ofrece un servicio de difusión para mensajes que contienen transacciones. Los clientes se conectan al canal y pueden transmitir mensajes en él, que luego se entregan a todos los compañeros.

Este canal de comunicación tiene la cualidad de dar garantía en la entrega de todos los mensajes que se transmitan, denominada emisión atómica. En otras palabras el canal emite los mismos mensajes a todos los pares conectados y se los envía a todos en el mismo orden lógico. Los mensajes comunicados son las transacciones candidatas para su inclusión en la cadena de bloques (Architecture Explained, 2018). Para ambientes de producción, es necesario utilizar Apache Kafka para el intercambio de mensajes entre un conjunto de nodos orderer.

2.3.3.4 Prueba de consenso

El algoritmo de consenso de Fabric está dividido en 3 fases: Aprobación, Ordenamiento, y Validación. La fase de Aprobación está basada en la política de aprobación, donde los participantes tienen que votar si es válida o no la transacción, esta política puede estar sujeta al contrato inteligente, donde menciona como tiene que ser la política de aprobación, por ejemplo si dos de tres participantes aprueba la transacción, entonces esta válida, de lo contrario no será válida. La fase de ordenamiento o de pedidos, recibe las transacciones válidas de la fase de aprobación y da la orden de escribir las transacciones en la blockchain y de enviar a la red las transacciones realizadas. Y la última fase, se recibe el bloque con las transacciones validadas, donde se ejecutan y se verifican cada una de ellas, actualizando el estado de la base de datos y almacenando los registros en la blockchain (*Hyperledger Architecture, Volume 1*, 2017).

2.3.4 Hyperledger Explorer

Hyperledger Explorer es un módulo de blockchain y uno de los proyectos de Hyperledger hospedados por *The Linux Foundation*. Diseñado para crear una aplicación web fácil de usar, Hyperledger Explorer puede ver, invocar, implementar o consultar bloques, transacciones y datos asociados, información de red (nombre, estado, lista de nodos), códigos de cadena y familias de transacciones, así como cualquier otra información relevante almacenada en el libro mayor (Hyperledger Explorer , 2018).

Hyperledger Fabric demuestra ser una tecnología que cumple con los requisitos para el desarrollo de este proyecto, ya que mantiene los beneficios de una cadena de bloques con las características empresariales que le agregan aún más privacidad y seguridad que una cadena de bloques pública. En el siguiente punto, se explicará la herramienta para el desarrollo de contratos inteligentes para Fabric, llamado Hyperledger Composer. Esta herramienta permite definir los participantes involucrados, los activos del sistema, reglas de acceso entre los participantes y los activos, entre otras cosas.

2.4 Hyperledger Composer

Hyperledger Composer es un conjunto de herramientas y un framework de desarrollo para facilitar el desarrollo de aplicaciones blockchain. Composer facilita la integración de aplicaciones blockchain con el sistema de Hyperledger Fabric. Composer le permite modelar una red comercial e integrar sistemas y datos existentes con aplicaciones de blockchain.

Hyperledger Composer admite la infraestructura de la cadena de bloques de Hyperledger Fabric, admite protocolos de consenso de blockchain conectables para garantizar que las transacciones sean validadas de acuerdo con la política por los participantes de la red comercial designados (Welcome to Hyperledger Composer, 2018).

2.4.1 Características de Composer

En Composer el archivo donde se encuentra la aplicación se llama *Business Network Archive* (.bna), que son conjuntos de archivos que conforman dicha aplicación. Estos archivos son:

Model Files (.cto)

Estos archivos definen los recursos del sistema y se escriben con el lenguaje de modelado CTO, estos pueden ser:

- Activos, participantes, transacciones y eventos
 - Los activos o *assets* son objetos o valores dentro del modelo, por ejemplo: Automóviles, Títulos, etc.

- Los participantes son las personas que interactúan con los activos por medio de las transacciones, por ejemplo: Médicos, ingenieros, empresarios, dueños, etc.
 - Las transacciones son las acciones que un participante puede realizar, habitualmente suelen ser para modificar algún activo en el sistema, por ejemplo: “el participante Pedro vende un automóvil”
 - Los eventos son notificaciones que envía la *blockchain* y son consumidas por aplicaciones externas, por ejemplo: Se envía un mensaje cada vez que se venda un automóvil, cuando llegue un cargamento, etc (Hyperledger Composer Modeling Language, 2018)
- Tipo enumerado
 - Son un tipo de dato que asigna valores predeterminados a cadenas de strings, véase el código #2 como ejemplo.

```
enum colores {
  o AMARILLO
  o AZUL
  o ROJO
  o VERDE
}
```

Código #2 - Ejemplo del enum type
Fuente: Elaboración propia

- Conceptos
 - Son clases abstractas que no representan activos, participantes ni transacciones, estos pueden ser direcciones, leyes o información adicional. (Hyperledger Composer Modeling Language, 2018).

Script File

Los scripts contienen transacciones implementadas que están previamente definidas en el *model file*, estas funciones son invocadas utilizando la API de hyperledger composer.

Access Control Rules

Es un archivo que contiene los permisos en el sistema, que se definen en un lenguaje de control de acceso (ACL) que proporciona control de acceso declarativo sobre los elementos del modelo de dominio. Al definir las reglas de ACL, puede determinar qué usuarios/roles tienen permiso para crear, leer, actualizar o eliminar elementos/activos en el modelo de dominio de una red comercial (Hyperledger Composer Access Control Language, 2018). Por ejemplo, en un supermercado, los carniceros sólo pueden servir las carnes.

Query File

Las consultas en Hyperledger Composer están escritas en un lenguaje de consulta personalizado. Las consultas se definen en un único archivo de consulta llamado (queries.qry) dentro de una definición de red empresarial (Hyperledger Composer Query Language, 2018).

2.4.2 Composer Rest Server

Composer provee un servidor API rest para la fácil comunicación con la BNA instalada en la *Blockchain*, dando soporte a la autenticación de usuarios y TLS.

En el siguiente punto se describe un poco del funcionamiento de Apache Kafka, aplicación que es fundamental para el funcionamiento de un ambiente distribuido de nodos orderer en Hyperledger Fabric.

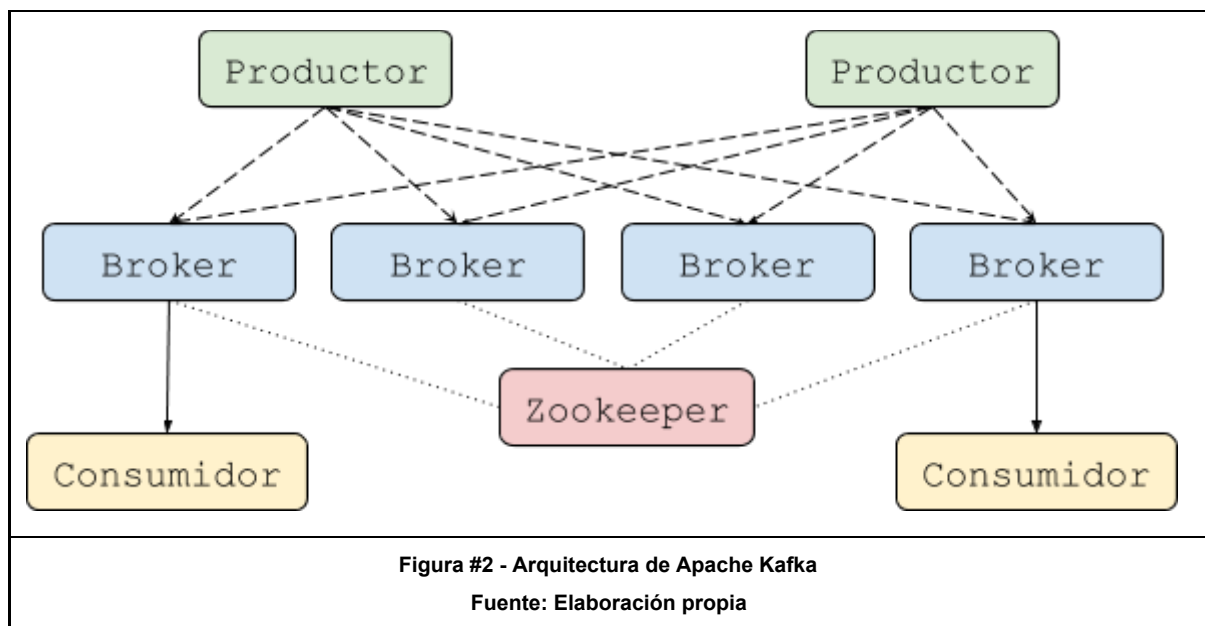
2.5 Apache Kafka

Kafka es un servicio de registro distribuido, particionado y replicado, desarrollado por LinkedIn en 2011 y de código abierto. Opera como una cola de publicación masivamente escalable, diseñada como un registro de transacciones distribuidas. Fue creado para proporcionar "una plataforma unificada para manejar todos los feeds de datos en tiempo real que una compañía grande podría tener" (Mouzakitis, 2016).

2.5.1 Características

- Como se mencionó anteriormente, es un servicio de registro replicado.
- No usa ningún otro protocolo preexistente para la comunicación. En cambio, usa un protocolo binario TCP personalizado.
- Tiene fuertes semánticas de ordenamiento y garantías de durabilidad.

2.5.2 Arquitectura del sistema



Los productores publican mensajes a temas (*topics*) y los consumidores consumen los mensajes de los *topics*. Los productores y los consumidores operan de manera push / pull, con los productores enviando mensajes a los *brokers* (Kafka) y los consumidores encuestando a estos para obtener nuevos mensajes. Los intermediarios son nodos de

Kafka , almacenando los mensajes publicados para que los consumidores obtengan su propia tarifa. Eso se muestra en la figura 2. Esto significa que los agentes de Kafka son *stateless* (protocolo sin estado), es decir no rastrean el consumo, dejando la eliminación de mensajes en una política de retención configurable (Mouzakitis, 2016).

Los mensajes consisten en una carga útil de bytes sin formato, con información del tema (*topic*) y de partición codificada. Kafka agrupa los mensajes por temas y los consumidores se suscriben a los temas que necesitan. Los mensajes en Kafka están ordenados por marca de tiempo y son inmutables, con la lectura como la única operación permitida (Mouzakitis, 2016).

En la siguiente sección se explica uno de los estándares más famosos para crear una interfaz utilizando el protocolo http para la obtención de información o ejecución de operaciones en un servidor. Aunque la aplicación desarrollada no utiliza esta arquitectura en su totalidad, su interfaz desarrollada hereda muchas características de esta.

2.6 Restful

REST (*representational state transfer*) es un estilo de arquitectura para crear interfaces utilizando el protocolo HTTP para la obtención de datos o indicar la ejecución de operaciones sobre los datos en un servidor (Richardson y Ruby, 2007, pp.15-16). REST nos permite crear aplicaciones o servicios que pueden ser utilizadas por cualquier cliente que use HTTP.

REST afirma que la Web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

- **Un protocolo cliente/servidor sin estado:** cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)
- **Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información:** HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se

equiparan a las operaciones CRUD en bases de datos que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.

- **Una sintaxis universal para identificar los recursos:** En un sistema REST, cada recurso es direccionable únicamente a través de su URI, es decir el único punto de entrada que debe tener una API rest es la URL, la solicitud de los recursos o la ejecución de las operaciones se encuentran en la URN (*Uniform Resource Name*), utilizando un sistema parecido al directorio de rutas en UNIX. Véase la tabla #1.
- **El uso de hiper-medios:** tanto para la información de la aplicación como para las transiciones de estado de la aplicación, la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

Tabla #1 - Acceso a los recursos por medio la URL Fuente: Elaboración propia	
/api/	Punto de entrada de la API
/api/col	Punto de entrada de todos los elementos de la colección
/api/col/:id	Punto de entrada un elemento en específico de la colección
/api/col/:id/subcol	Punto de entrada de una subcolección dentro de la colección identificada por un id
/api/col/:id/subcol/:subid	Punto de entrada de un elemento en específico dentro de la subcolección

Por último, en la siguiente sección se mencionan las leyes que protegen la privacidad de la información médica del paciente. Para caso de este proyecto, se tomaron en cuenta la ley HIPAA y la Ley de ejercicio de la medicina en Venezuela.

2.7 Privacidad de la información médica de un paciente

La privacidad de un paciente es un derecho que tiene la persona en decidir cuando, como y hasta donde se puede compartir la información médica del mismo con otras personas, incluye el mantener la confidencialidad y el compartir datos identificatorios. Esta información solamente estará disponible para el médico tratante, proveedores de servicios médicos o algún otro profesional relacionado para poder tratar al paciente.

2.7.1 Ley de ejercicio de medicina en Venezuela

En Venezuela existen las leyes que regulan el ejercicio del médico o médica y se detallan en la ley del ejercicio del médico, capítulo VI, artículos 46 al 53. En dicho capítulo explica detalladamente cada aspecto que debe seguir el profesional para la protección del paciente, el amparo y salvaguarda su honor y la dignidad de la ciencia. En otras palabras, el secreto es inviolable y el profesional está en la obligación de guardarlo. Sin embargo, no se viola el secreto médico solo en esos casos: primero, cuando la revelación del secreto se hace por mandato de ley. Segundo, si el paciente autoriza al médico o médica para que lo revele. Tercero, cuando el médico o médica, en su calidad de experto o experta de una empresa o institución y, previo consentimiento por escrito del paciente, rinde su informe sobre las personas sometidas a exámenes al departamento médico de aquella. Cuarto, cuando el médico o médica ha sido encargado o encargada, por la autoridad competente, para dictaminar sobre el estado físico o mental de una persona. Quinto, cuando actúa en el desempeño de sus funciones como médico o médica forense, o médico o médica legista. Sexto, cuando denuncia ante las autoridades sanitarias los casos de enfermedades de notificación obligatoria de que tenga conocimiento. Séptimo, cuando expide un certificado de nacimiento o de defunción, o cualquiera otro relacionado con un hecho vital, destinado a las autoridades judiciales, sanitarias, de estadísticas o del registro civil. Octavo, cuando los representantes legales del niño, niña y adolescente exijan por escrito al médico o médica la revelación del secreto. Sin embargo, el médico o médica podrá, en interés del niño, niña y adolescente, abstenerse de dicha revelación. Noveno, cuando se trate de salvar la vida o el honor de las personas. Décimo, cuando se trate de impedir la condena de un o una inocente. Onceavo, cuando se informe a los órganos gremiales médicos de asuntos relacionados con la salud de la comunidad, en cuanto atañe al ejercicio de la medicina. El secreto se puede compartir entre médicos que llevan el mismo caso y el paciente siempre debe saber la verdad aunque el caso sea grave (Gaceta Oficial N° 39.823 2011).

2.7.2 HIPAA

La HIPAA es la Ley de Responsabilidad y Transferibilidad de Seguros Médicos (Health Insurance Portability and Accountability Act por sus siglas en inglés). La Ley consiste en cinco Títulos. El Título I de HIPAA protege la cobertura de seguro de salud para los trabajadores y sus familias cuando cambian o pierden sus trabajos. El Título II de HIPAA, conocido como las disposiciones de Simplificación Administrativa, requiere el establecimiento de estándares nacionales para transacciones electrónicas de atención médica e identificadores nacionales para proveedores, planes de seguro de salud y empleadores. El Título III establece las pautas para las cuentas de gastos médicos antes de impuestos, el Título IV establece las pautas para los planes de salud grupales, y el Título V rige las pólizas de seguro de vida propiedad de la compañía.

La Regla de Privacidad de HIPAA regula el uso y la divulgación de información médica protegida en poder de "entidad médica" (generalmente, cámaras de compensación de salud, planes de salud patrocinados por el empleador, aseguradores de salud y proveedores de servicios médicos que participan en determinadas transacciones). La información médica protegida es cualquier información que posee una entidad cubierta sobre el estado de salud, la provisión de atención médica o el pago de atención médica que puede vincularse con cualquier persona, esto incluye cualquier parte de la historia clínica o historial de pagos de un individuo. Las entidades médicas están en la obligación de informar al paciente hasta 30 días previos o divulgar la información médica protegida cuando lo exija la ley o si hay sospechas de abuso infantil.

Estas entidades pueden divulgar la información de salud protegida a los funcionarios encargados de hacer cumplir la ley para fines de aplicación de la ley según lo exija la ley (incluidas órdenes judiciales, órdenes judiciales, citaciones judiciales) y solicitudes administrativas; o para identificar o localizar a un sospechoso, un fugitivo, un testigo material o una persona desaparecida.

Una entidad puede divulgar la PHI (*Protected Health Information*) a ciertas partes para facilitar el tratamiento, el pago o las operaciones de atención médica sin la autorización expresa por escrito del paciente. Cualquier otra divulgación de PHI requiere que la entidad cubierta obtenga autorización escrita del individuo para la divulgación. En cualquier caso, cuando una entidad divulga cualquier PHI, debe hacer un esfuerzo para divulgar sólo la información mínima necesaria requerida para lograr su propósito (Health Information Privacy, 2017).

CAPÍTULO 3: MARCO METODOLÓGICO

En esta sección, se define toda la metodología de desarrollo del componente:

3.1 Ingeniería de software basada en componentes

Esta metodología surgió a finales de la década de 1990 como un enfoque al desarrollo de sistemas de software basado en la reutilización de componentes de software motivado a las frustraciones por la poca reutilización del mismo.

Los componentes son abstracciones de alto nivel y se definen mediante sus interfaces, todos los detalles de su implementación se ocultan a otros componentes. La CBSE es el proceso de definir, implementar e integrar o componer los componentes independientes e imprecisos en los sistemas (**Sommerville, 2011**). Los fundamentos de la ingeniería de software basada en componentes son:

- Componentes independientes que se especifican por completo mediante sus interfaces. La interfaz del componente y su implementación tienen que ser independientes y separables, ya que debe permitirse la sustitución de componentes sin cambiar o perjudicar otras partes del sistema.
- Estándares de componentes que facilitan la integración de éstos. Se definen estándares para especificar las interfaces de los componentes y cómo estos se comunican entre sí. Para lograr esto, las interfaces deben implementarse por todos los componentes integrantes, ya que si todos los componentes utilizan los mismos estándares, entonces su ejecución es independiente de su lenguaje de programación, por lo tanto los componentes escritos en diferentes lenguajes pueden integrarse en un mismo sistema.
- Middleware que brinda soporte de software para integración de componentes. Para que los componentes independientes y distribuidos trabajen bien en conjunto, se necesita de aplicaciones que manejen las comunicaciones de los componentes. Un middleware para soporte de componentes puede brindar apoyo para la asignación de recursos, la gestión de transacciones, la seguridad y concurrencia.
- Un proceso de desarrollo que se engrana con la ingeniería de software basada en componentes. Usted necesita un proceso de desarrollo que permita la evolución de requerimientos, dependiendo de la funcionalidad de los componentes disponibles (**Sommerville, 2011**).

3.2 Componentes y modelos de componentes

Un componente es una unidad de software independiente que puede organizarse con otros componentes para crear un sistema de software.

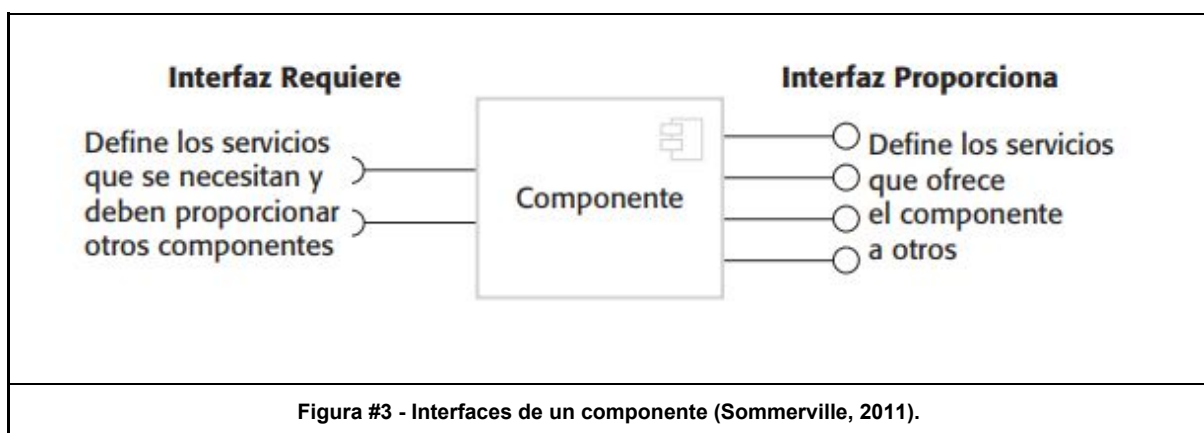
3.2.1 Características de un componente

En la **tabla #2** muestra las características más esenciales de un componente (**Sommerville, 2011**).

Tabla #2 - Características de un componente Fuente: Elaboración propia	
Características del componente	Descripción
Estandarizado	Un componente utilizado durante un proceso CBSE se ajusta a un modelo de componentes estándar. Este modelo puede definir interfaces de componentes, metadatos de componentes, documentación, composición e implementación.
Independiente	Debe ser factible componerlo e implementarlo sin usar otros componentes específicos.
Componible	Todas las interacciones externas deben tener lugar mediante interfaces definidas públicamente.
Implementable	Un componente debe estar autocontenido. Debe ser capaz de ejecutarse como entidad independiente en una plataforma de componente que permita una implementación del modelo de componentes, esto significa que el componente es binario y no tiene que compilarse antes de su implementación.
Documentado	Los componentes deben implementarse por completo, para que los usuarios potenciales puedan decidir si los componentes cumplen o no sus necesidades. Debe especificarse

	la sintaxis y, de manera ideal, la semántica de todas las interfaces de componente.
--	---

Los componentes tienen dos interfaces relacionadas, como se muestra en la **figura #3**. Estas interfaces reflejan los servicios que proveen los componentes y los servicios que el componente requiere para ejecutarse correctamente. La interfaz “proporciona” define los servicios que ofrece el componente. Esta interfaz es el componente API. Define los métodos que puede solicitar el usuario del componente. La interfaz “requiere” especifica qué servicios deben ofrecer otros componentes en el sistema para que un componente opere correctamente. Si no están disponibles, entonces el componente no funcionará. Esto no compromete la independencia o el carácter implementable de un componente, porque la interfaz “requiere” no define cómo deben proporcionarse dichos servicios (**Sommerville, 2011**).

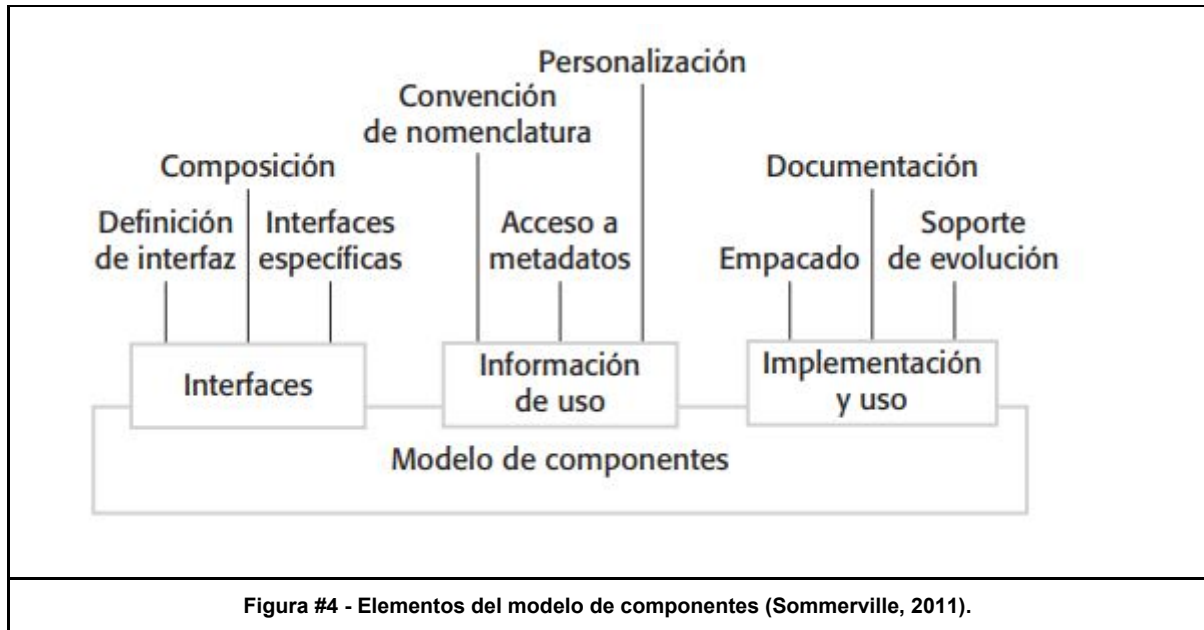


3.2.2 Modelos de componentes

Un modelo de componentes es una definición de estándares para implementación, documentación y despliegue de componentes. Estos estándares se establecen para que los desarrolladores de componentes se aseguren de que éstos pueden interoperar. En la figura #4 se resumen esos elementos de modelo (**Sommerville, 2011**). Los elementos básicos pueden clasificarse en tres grupos según su relación:

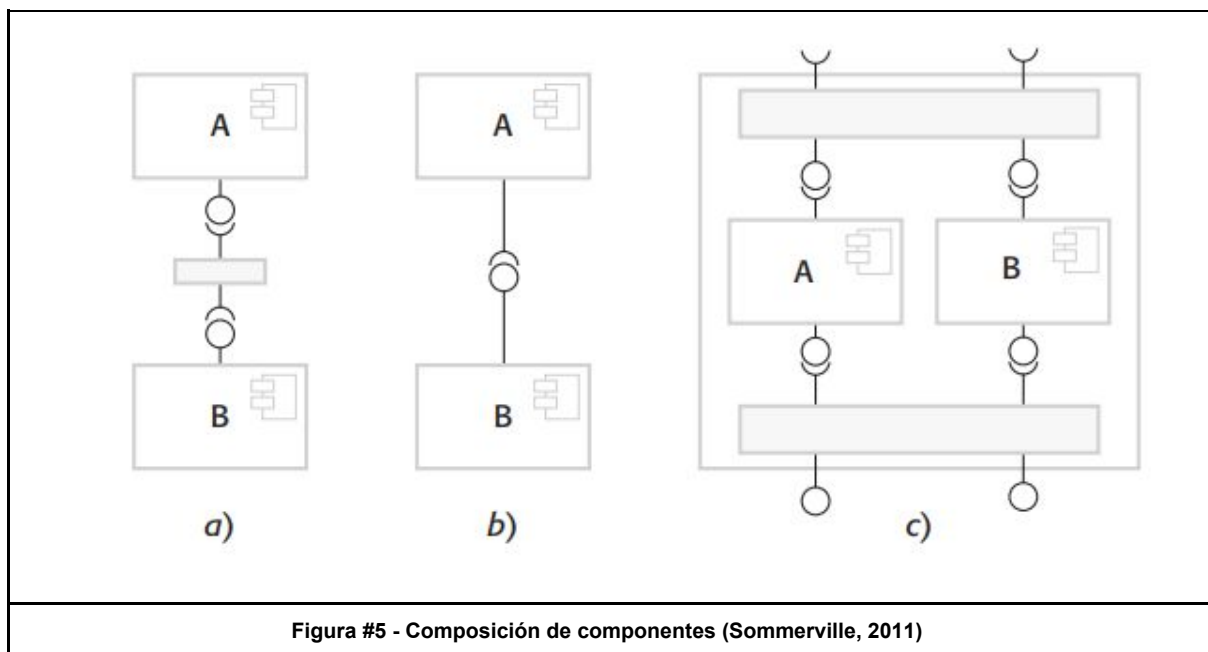
- Interfaces: Los componentes se definen al especificar sus interfaces, por lo tanto en el modelo de componentes se especifica en cómo tiene que definirse las interfaces y los elementos, como por ejemplo, los nombres de las operaciones, los parámetros, las excepciones, etc.

- Información de uso: Para que los componentes se distribuyan y se acceda a ellos de manera remota, deben tener un nombre único asociado. Los metadatos del componente tienen que tener información acerca de sus interfaces y atributos, ya que permiten a los usuarios del componente determinar qué servicios se proporcionan y requieren. Los componentes deben de poder configurarse para ajustarse en un sistema de aplicación.
- Implementación: El modelo de componentes tiene que incluir una especificación de como están empacados los componentes para su implementación como entidades independientes, ya que los componentes deben empacarse con todo el software de soporte que no proporcione la infraestructura de componente. El modelo de componentes debe incluir reglas que rijan cuándo y cómo se permite la sustitución de componentes, ya que conforme surjan nuevos requerimientos, los componentes deberán cambiarse o sustituirse. Finalmente, el modelo de componentes puede definir la documentación de componentes que deba producirse (**Sommerville, 2011**).



3.3 Composición de componentes

La composición de componentes es el proceso de unir varios componentes entre sí con “código pegamento” especialmente escrito para crear un sistema u otro componente. Existen distintas formas en las que se pueden integrar componentes, como se muestra en la figura #5. De izquierda a derecha, dichos diagramas ilustran la composición secuencial, la composición jerárquica y la composición aditiva (**Sommerville, 2011**).



Los componentes que se desarrollen especialmente para una composición, se deberá diseñar las interfaces de dichos componentes, de manera que todos los componentes dentro de la composición sean compatibles en el sistema (**Sommerville, 2011**).

CAPÍTULO 4: MARCO APLICATIVO

En esta sección se explicará el desarrollo del componente para la comunicación con IPFS y Hyperledger fabric utilizando la metodología descrita en el capítulo anterior, así como también las herramientas utilizadas en este proyecto.

4.1 Definición de los requerimientos

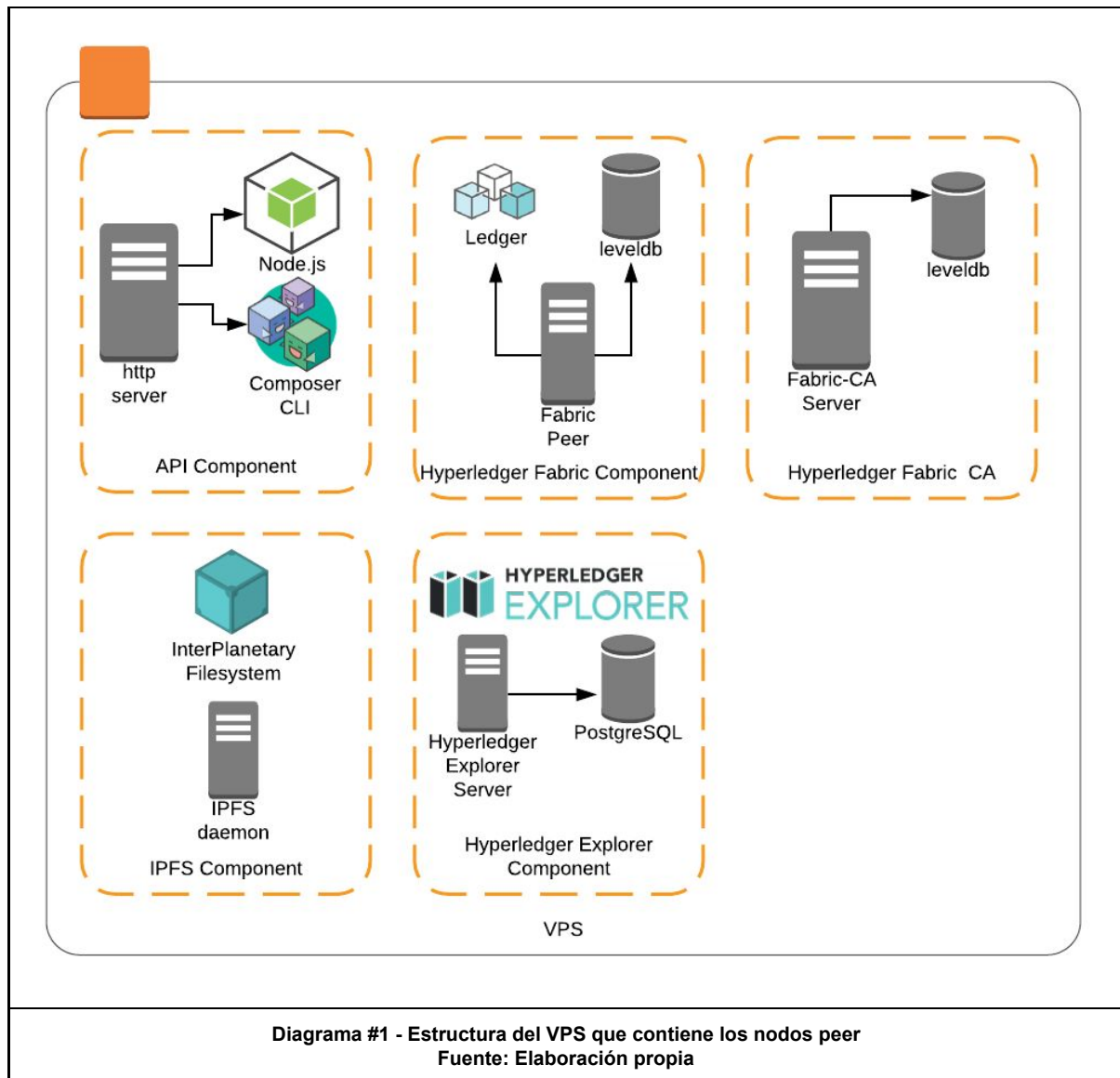
El requerimiento principal es desarrollar una interfaz de programación de aplicaciones capaz de servir como intermediario entre IPFS y el contrato inteligente instanciado en la red de Hyperledger Fabric. Para lograr cumplir este requerimiento es necesario cumplir otros requisitos:

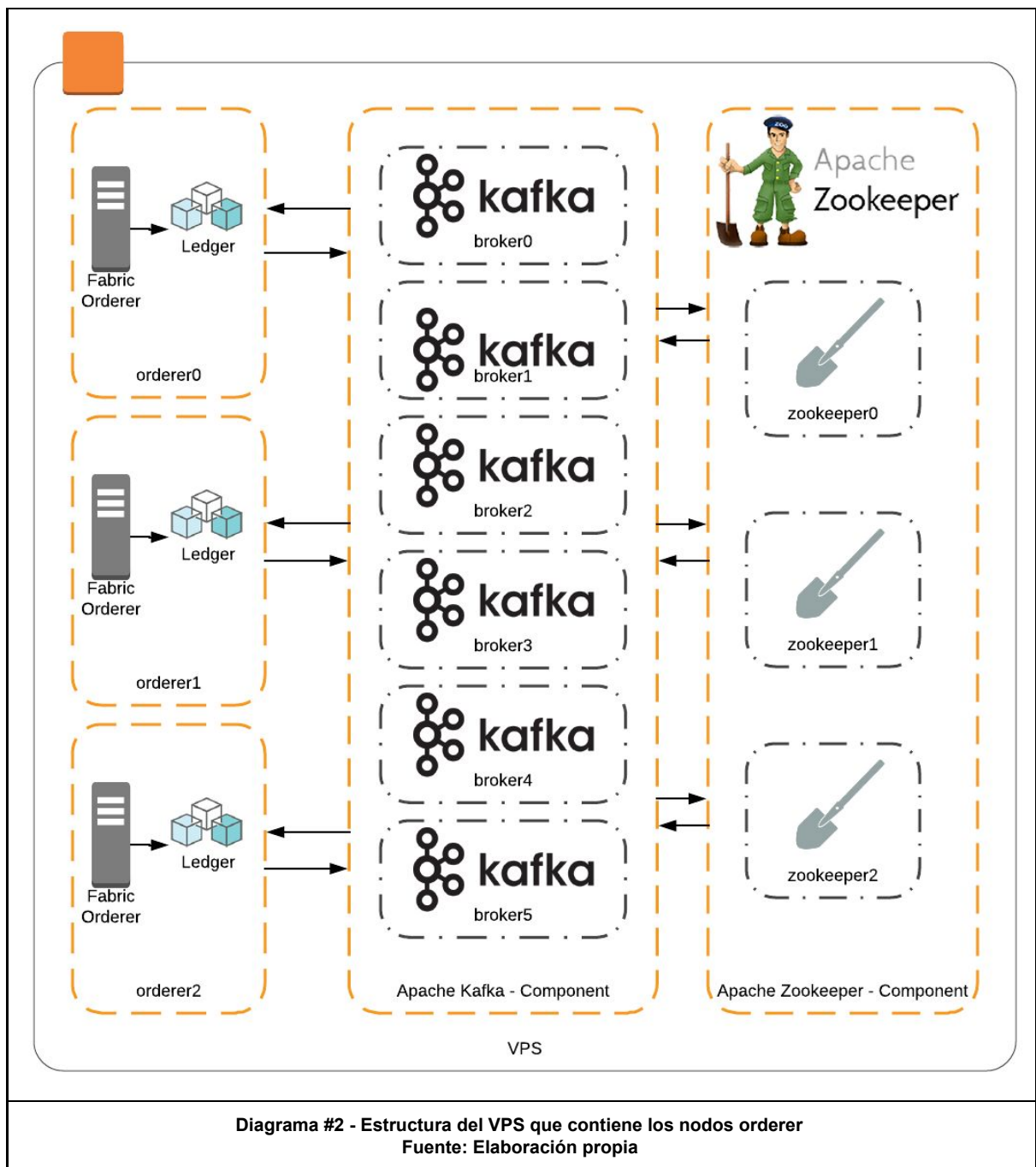
- Levantar una red de IPFS privada, en donde solo se conecten los nodos que queramos que estén en la red.
- Levantar un ambiente de Hyperledger Fabric, compuesto por varias organizaciones, en donde cada una tenga su propio servidor.
- Diseñar e implementar un contrato inteligente, capaz de cubrir los requerimientos necesarios para el almacenamiento y la obtención del historial médico de un paciente.
- Instanciar el contrato previamente diseñado en la red de Hyperledger Fabric, utilizando la herramienta Hyperledger Composer, en donde se crearán los administradores del sistema y las identificaciones de los usuarios (Pacientes, doctores, etc).

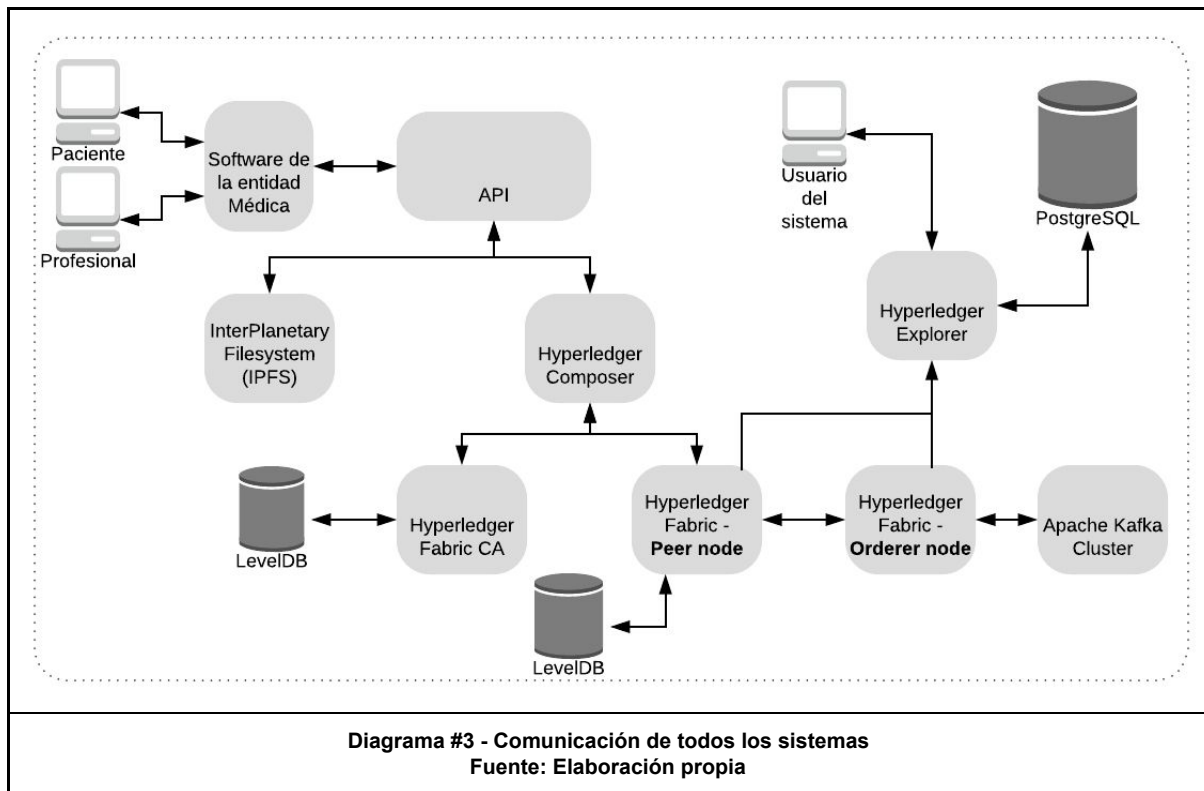
4.2 Diseño técnico

Como se mencionó en el punto anterior, para desarrollar la aplicación fue necesario crear un ambiente en donde esté en ejecución los sistemas de IPFS y Hyperledger Fabric. Se tuvo que crear 3 instancias de un VPS (Servidor virtual privado) en donde cada una representará una organización. Cada instancia va a tener instalado un nodo de IPFS, un servidor de Hyperledger Fabric CA (para el manejo de las identidades), un nodo peer de Hyperledger Fabric y un explorador de bloques llamado Hyperledger Explorer, véase la diagrama #1. Para los nodos orderer de Hyperledger Fabric se tuvo que crear una nueva instancia VPS con mejores características para ejecutar Apache Kafka, dicha instancia tiene

instalado 3 nodos orderer, 3 nodos de Apache Zookeeper y 6 Brokers de Apache Kafka, véase la diagrama #2. La comunicación entre los distintos sistemas que participan en la red se puede visualizar en el diagrama #3.







Para el diseño del modelo del contrato inteligente, se definió de la siguiente manera, como participantes, se encuentran los pacientes, los profesionales (médicos, enfermeros, etc) y los administradores, en la tabla #3 se muestran los datos que describen a cada participante. Para los activos, se definieron los siguientes: HistorialMedico, PerfilPublico y PerfilPersonal, mostrados en la tabla #4. Por último, las transacciones que son: AsignarPermisos, ModificarRegistroMedico y RevocarAccesoMedico, pudiéndose visualizar en la tabla #5. Dado que Hyperledger Composer ya contiene las transacciones de crear, modificar, leer y eliminar, éstas no serán implementadas.

Tabla #3 - Definición de los participantes Fuente: Elaboración propia		
Participante		
Tipo de Participante	Dato	Descripción
Paciente	String ID	ID del paciente
	String[] Autorizados	Arreglo de string que contiene los id de los profesionales tratantes del paciente
	PerfilPersonal	Objeto que contiene la

		información personal del paciente
Profesional	String ID	ID del profesional
	String[] Pacientes	Arreglo de string que contiene los id de los pacientes que trata el profesional
	PerfilPublico	Objeto que contiene la información pública del profesional
Administrador	String ID	ID del administrador
	String Nombre	Nombre del administrador
	String Apellido	Apellido del administrador

Tabla #4 - Definición de los activos Fuente: Elaboración propia		
Activos		
Tipo de Activo	Dato	Descripción
PerfilPersonal	String ID	ID del perfil
	String Nombre	Nombre del paciente
	String Apellido	Apellido del paciente
	String Email	Email del paciente
	Integer dob	Fecha de nacimiento del paciente
	String Dirección	Dirección del paciente
	String TipoSangre	Tipo de sangre del paciente
	String Alergia	Alergias del paciente
	String Donante	Si el paciente es donante
PerfilPublico	String ID	ID del perfil
	String Nombre	Nombre del profesional
	String Apellido	Apellido del profesional
	String email	Email del profesional

	Integer dob	Fecha de nacimiento del profesional
	String dirección	Dirección del profesional
	String licencia	Licencia del profesional
HistorialMedico	String ID	ID del historial médico
	String hash	Hash del historial almacenado en IPFS
	String[] permisos	Arreglo de ID de profesionales que se les concedió acceso al historial médico del paciente
	Double version	Versión del historial
	Paciente	Paciente al que le pertenece el historial médico

Tabla #5 - Transacciones del modelo Fuente: Elaboración propia		
Transacciones		
Transacción	Parámetros	Descripción
AsignarPermisos	HistorialMedico	ID de historial médico
	Profesional	ID del profesional a asignarle el acceso al historial médico
ModificarRegistroMédico	HistorialMedico	ID del historial médico
	String newHash	Nuevo hash generado por IPFS por la modificación del registro médico
	Double version	Nueva versión del registro médico
RevocarAcceso	HistorialMedico	ID del historial médico
	Profesional	ID del profesional el cual se le quiere revocar los acceso al historial médico del paciente

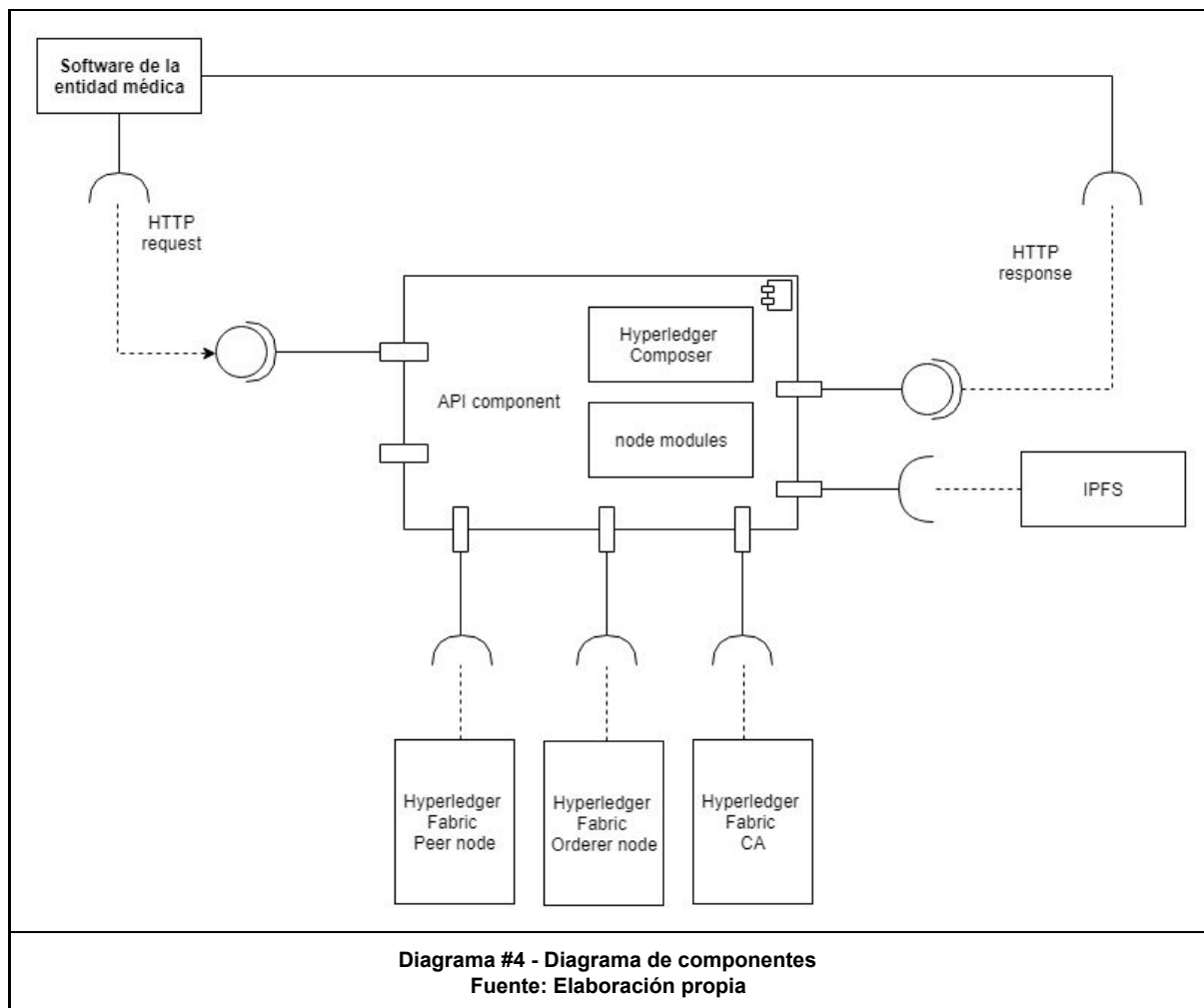
Gracias a las reglas de acceso de Hyperledger Composer, es posible restringir el acceso de los recursos a los participantes, como por ejemplo un médico no puede tener acceso a un historial médico de un paciente del cual no se le ha otorgado acceso, o un administrador no podrá modificar el historial médico de un paciente. Estas reglas están definidas de la siguiente manera:

- Administradores:
 - Crear, modificar o leer pacientes
 - Crear, modificar o leer profesionales
 - Crear, modificar o leer perfiles personales
 - Crear, modificar o leer perfiles públicos
- Pacientes:
 - Leer su perfil personal
 - Leer su historial médico
 - Otorgar permiso a un profesional de su historial médico
 - Revocar permiso a un profesional de su historial médico
- Profesionales:
 - Leer su perfil público
 - Leer historiales médicos de sus pacientes
 - Modificar historiales médicos de sus pacientes
- Administradores de la red:
 - Crear, modificar, leer o eliminar cualquier recurso (activo) o participante

Cabe destacar que los administradores de la red son participantes especiales que son creados en el momento de instanciar el contrato inteligente, uno por cada organización y su función es administrar la aplicación.

4.2.1 Diagrama de componentes

El diagrama de componentes se diseñó de la manera más simple y entendible, véase el diagrama #5. La interfaz será por medio de una API escrita en Javascript, esta se encargará de llamar a las transacciones del contrato inteligente, introducir o extraer archivos de IPFS y por supuesto enviar las respuestas al cliente.



4.2.2 Diagrama de Secuencia

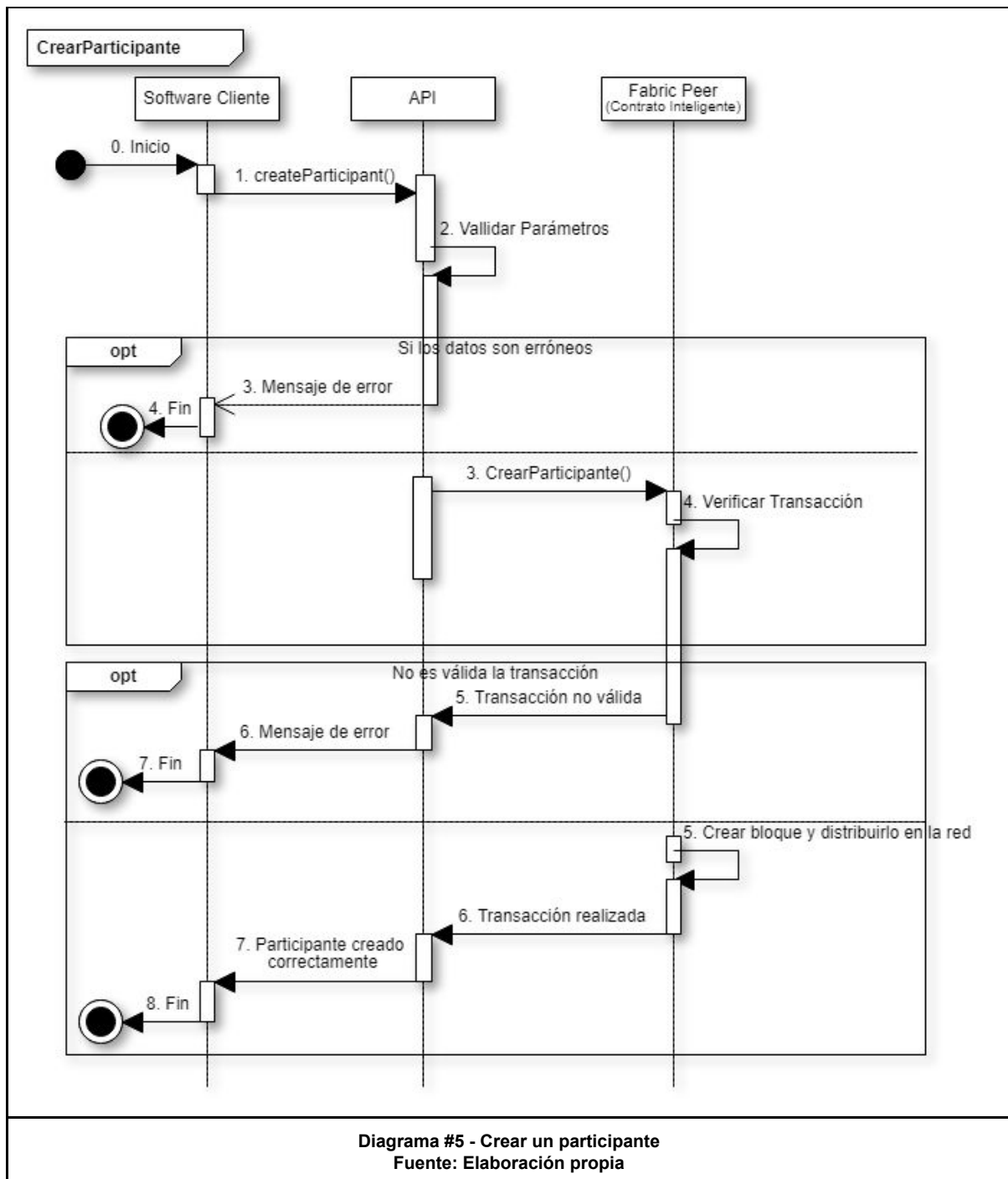
Para el desarrollo de la aplicación se tomaron en cuenta diferentes diseños para la API, al final se optó por un servicio sin estado, en donde la API no almacene información del cliente, sino que la petición que se le realice a este, tenga la suficiente información para realizar la operación y devolverle una respuesta.

Toda operación que se realice sobre la API, debe de tener adjunto la tarjeta de identificación del participante, es decir, para que un paciente pueda descargar su historial médico tiene que tener su tarjeta de identificación para realizar la consulta. Esta tarjeta de identificación está conformada por los datos de los servidores de Hyperledger (Peer, Orderer, Fabric CA) y las credenciales del usuario (Clave privada, y clave pública).

Dado que son demasiados diagramas, y muchos de ellos repiten las mismas operaciones, se explicaran los más importantes o esenciales y los demás estarán en la sección de **ANEXO A** del documento, después de este capítulo.

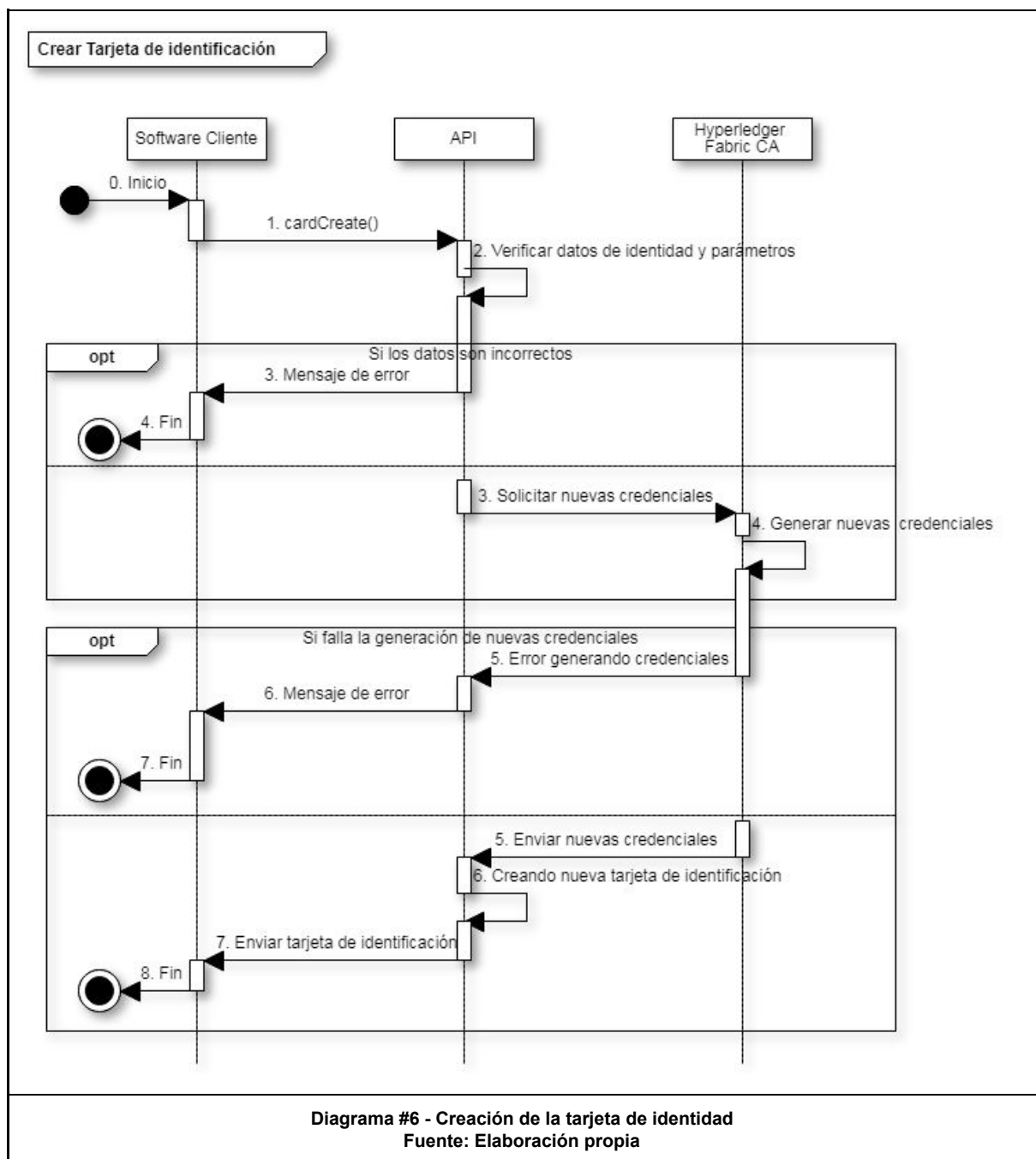
- **Crear Participante**

En el diseño del contrato inteligente, se definieron distintos tipos de participantes, como los pacientes, los profesionales y los administrador, para que estos puedan tener activos dentro del sistema, primero hay que crearlos. El diagrama de secuencia correspondiente a esta operación puede visualizarse en el diagrama #5



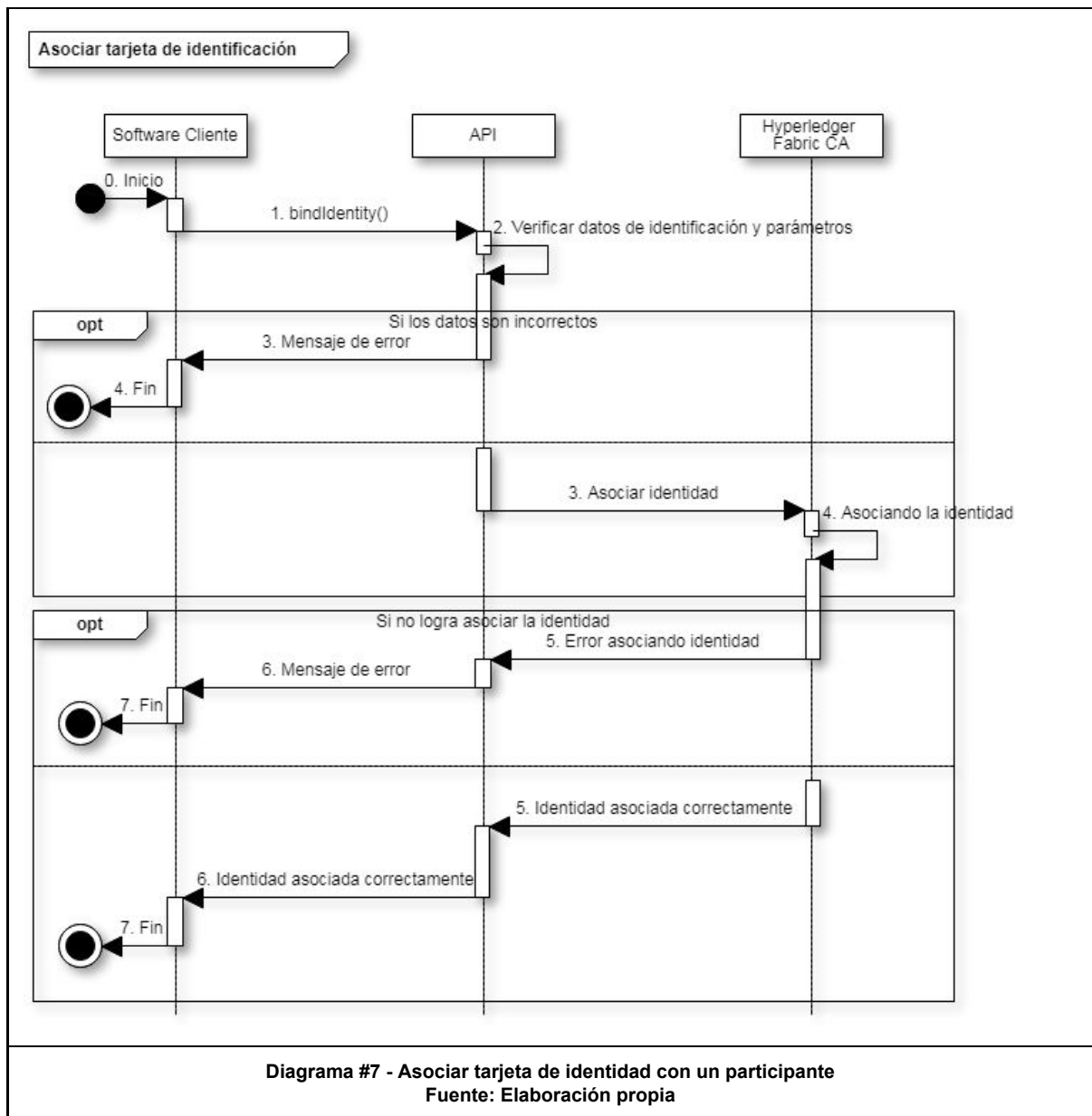
- **Crear tarjeta de identidad**

Para que el usuario creado en el paso anterior pueda interactuar con el sistema, primero hay que crearle una tarjeta de identidad y luego asociarlo con su ID. En diagrama #6 se puede visualizar el proceso. Cabe destacar que este proceso es una de las ventajas que ofrece la herramienta Hyperledger Composer, el uso de las identidades y la posibilidad de asociarlos a una tarjeta.



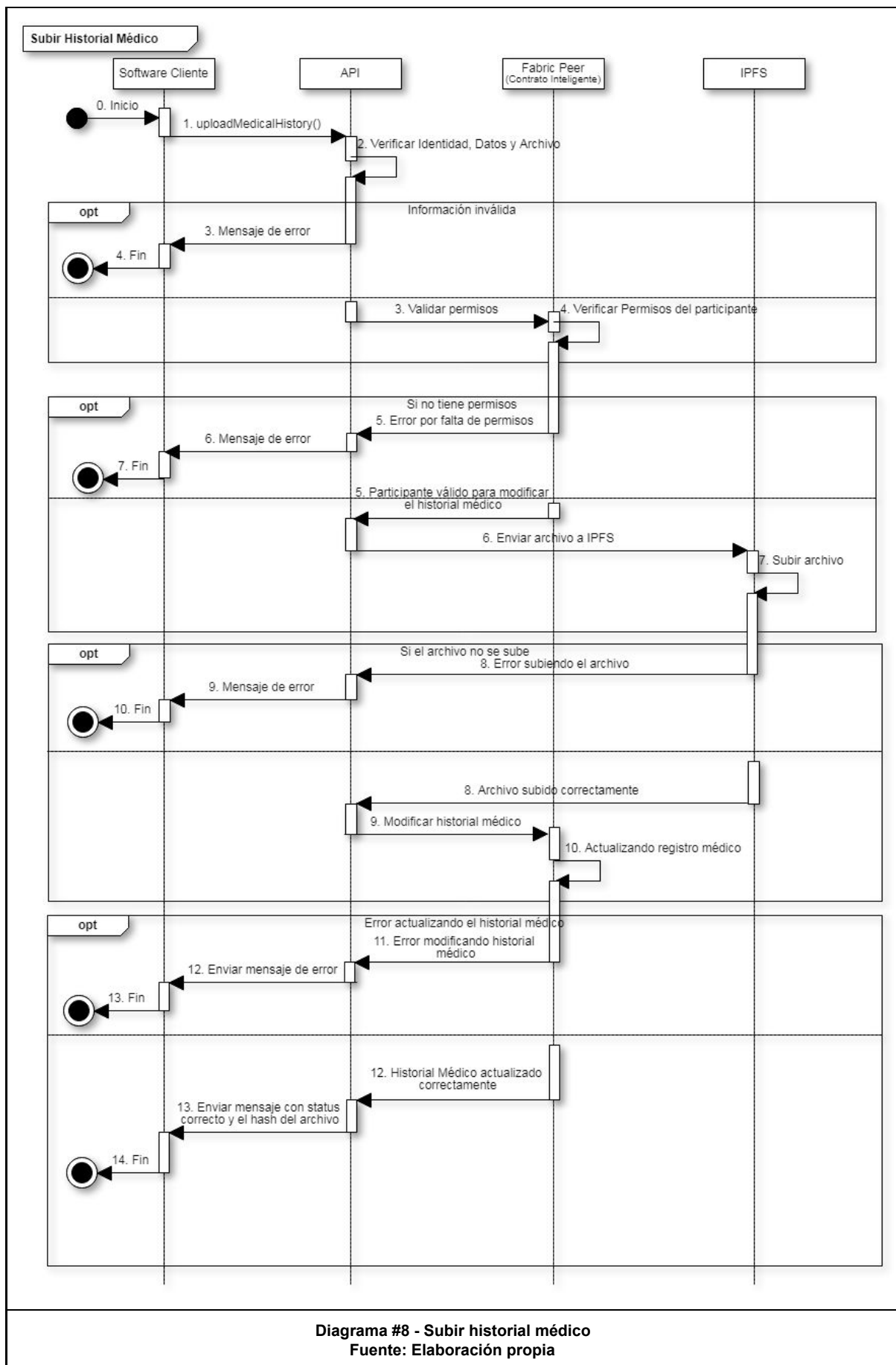
- **Asociar tarjeta de identidad con un participante**

Después de crear la tarjeta de identidad y el participante, lo ideal es asociar la tarjeta al participante, de esta forma el participante podrá interactuar con la red. Véase en el diagrama #7 el proceso para asociar una identidad con un participante.



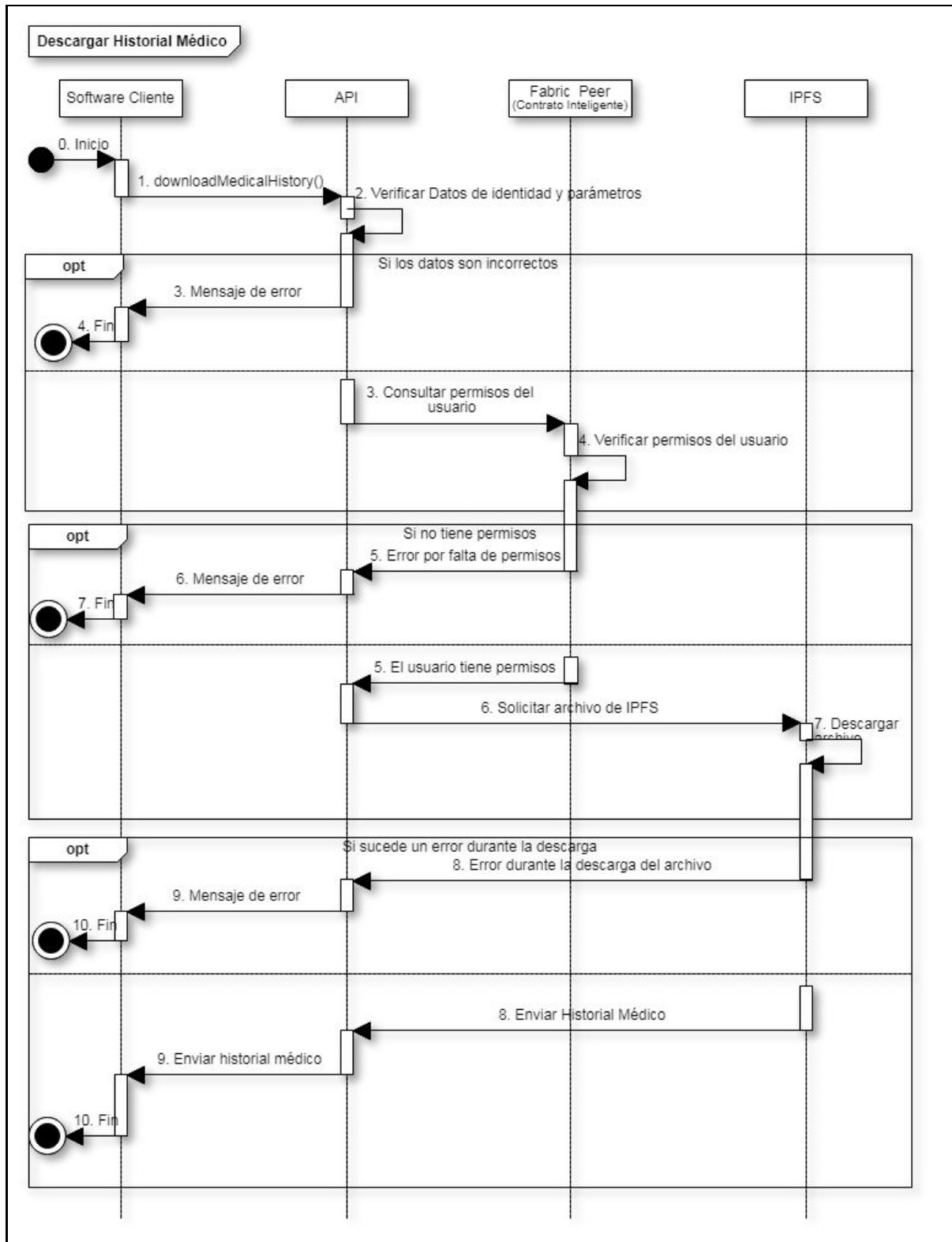
- **Subir Historial Médico de un Paciente**

Para subir un historial médico, previamente tendría que estar creado, el proceso de creación del historial médico, como el de perfil personal o perfil privado se encontrarán en la sección de anexos del documento, pues estos procesos son muy idénticos. En el diagrama #8 puede visualizarse el proceso para subir un historial médico. Este historial médico tiene que subirse en formato de compresión TAR para que pueda ser procesado por el API.



- **Descargar Historial Médico**

El proceso para descargar un historial médico creado en el paso anterior se puede visualizar en el diagrama #9. Cabe destacar que salida será un archivo comprimido con la información médica del paciente.



4.3 Definición de las herramientas

A continuación se indicarán cada una de las herramientas utilizadas para el desarrollo de la API:

4.3.1 Node.js

Node.js es un entorno de ejecución multiplataforma, de código abierto y asíncrono para ejecutar aplicaciones programadas en Javascript. Nodejs funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y salidas asíncronas las cuales pueden ejecutarse concurrentemente en un número de hasta cientos de miles sin incurrir en costos asociados al cambio de contexto. Javascript fue el lenguaje utilizado para el desarrollo de la aplicación, dado la facilidad y la portabilidad de la aplicación (Patel, 2018).

4.3.2 Node package manager (npm)

Npm es el manejador de paquetes por defecto de Node.js, utilizada para instalar módulos nuevos para la aplicación, o instalar las dependencias de la misma para que pueda ser ejecutada. También es utilizada para la ejecución de scripts, como por ejemplo “npm test”, en donde ejecuta el comando asociado a la etiqueta “test” (Patel, 2018).

4.3.3 Hyperledger Composer CLI

Es un conjunto de herramientas que permite la facilidad de implementar e instanciar contratos inteligentes en una red de nodos de Hyperledger Fabric. Composer ofrece abstracciones centradas en el negocio, así como aplicaciones de muestra con procesos devops fáciles de probar para crear soluciones robustas de blockchain que impulsan la alineación entre los requisitos del negocio con el desarrollo técnico, en la sección de Hyperledger Composer del marco teórico podrá encontrar información más detallada de esta herramienta.

4.3.4 Hyperledger Explorer

Hyperledger Explorer es un módulo de la cadena de bloques utilizada para visualizar de una manera muy amigable, el estado de la blockchain. Aunque no sea una herramienta, este ha servido bastante para las pruebas de la aplicación. Muestra la cantidad de bloques, hash de las transacciones, el estado de los nodos (si están en ejecución o apagados), etc. En la sección de Hyperledger Composer del marco teórico podrá encontrar información más detallada de esta herramienta.

4.3.4 Hyperledger Composer Playground

Composer Playground es una aplicación que sirve para dar una interfaz al contrato inteligente, donde permite la modificación del mismo, aplicar pruebas y ver el estado de la red. Gracias a esta herramienta fue posible realizar pruebas sobre los permisos de los participantes, permitiendo la rápida y fácil creación de participantes y activos por su interfaz gráfica. En la sección de Hyperledger Composer del marco teórico podrá encontrar información más detallada de esta herramienta

4.3.5 Postman

Postman es una aplicación de escritorio que sirve para realizar pruebas en APIs. Dicha aplicación permite realizar peticiones personalizadas, agregando parámetros a la cabecera HTTP, seleccionar el tipo de petición (Post, get, etc.), entre otras opciones. Esta herramienta es utilizada para mostrar los resultados de la aplicación, gracias a su interfaz que permite la visualización de las respuestas de la API (Farmer, 2017).

4.4 Entorno de trabajo

La aplicación se desarrolló sobre una computadora con las siguientes características:

- Procesador Intel® Core™ i3-3240 3.40 Ghz 2 núcleos (4 subprocesos)
- Sistema Operativo Ubuntu 16.04 Lts (Xenial) de 64bits
- 8 GB de RAM DDR3 1333 mhz
- 500 GB de HDD (Unidad de disco duro)

Aunque el entorno de pruebas se realizó en unos servidores virtuales, en donde se encuentran instalados los servidores de Hyperledger Fabric (Peer), IPFS, Hyperledger

Fabric CA y Hyperledger Explorer, estos servidores cumplen con las siguientes características:

- Procesador virtual de 1 núcleo
- 2048 MB de RAM
- 40 GB de SSD (Unidad de estado sólido)

Dado que los requisitos de Apache Kafka y Apache Zookeeper superaban las características anteriormente descritas, se decidió crear un nuevo servidor virtual que almacena los servidores de Hyperledger Fabric (orderer), Apache Kafka y Apache Zookeeper. Este servidor virtual cumple con las siguientes características:

- Procesador virtual de 2 núcleos
- 4 GB de RAM
- 60 GB de SSD (Unidad de estado sólido)

4.5 Desarrollo del componente

Para el desarrollo de la API se utilizó una metodología ad hoc, siguiendo lineamientos propios. Se tomó como base la interfaz de línea de comandos de Hyperledger Composer que da el acceso a los participantes, los recursos y las transacciones del contrato inteligente dada una identificación. Como el diseño principal no espera que el servidor almacene estados (stateless), por cada petición se tendría que recibir una tarjeta de identificación en la cual sirve como autenticador, donde se pueden validar las credenciales del participante y realizar la operación en el mismo instante. Un ejemplo de esta credencial puede visualizarse en la figura #6, la figura contiene dos imágenes, la imagen de la derecha es el contenido de la carpeta “credentials” de la imagen izquierda.

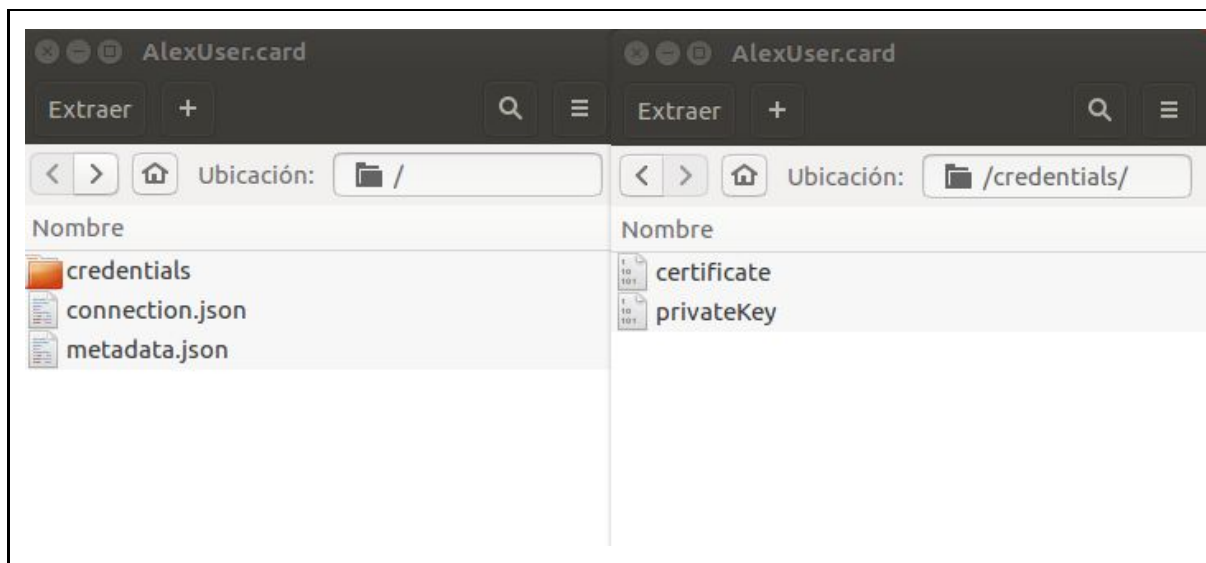


Figura #6 - Tarjeta de identificación
Fuente: Elaboración propia

4.5.1 Interfaz HTTP

Dado que las especificaciones para una API REST es necesario utilizar los formatos json o xml para la mayoría de las consultas, como por ejemplo las de escritura, y no permite la posibilidad de adjuntar archivos, como imágenes, sino que tienen que estar dentro del json/xml, se decidió implementar una variante de esta arquitectura.

Todas las peticiones, tanto de escritura como de lectura, tienen que realizarse utilizando el verbo http POST, en REST, las operaciones de lectura se hacen utilizando el verbo GET.

El contenido del cuerpo del mensaje http, ya no será un objeto json sino un objeto de tipo formulario, esto permite almacenar registros del tipo clave-valor en el cuerpo del mensaje y la posibilidad de adjuntar varios archivos. Todas las peticiones tienen que tener en la cabecera HTTP el siguiente parámetro: "Content-Type: multipart/form-data;".

La URL para tener acceso a los recursos del tipo "participante" puede verse en la tabla #6.

Tabla #6 - URL para el acceso a los participantes del sistema Fuente: Elaboración propia			
Recurso	URL	Parámetros	Descripción
Paciente	/participant/createPatient	String idPatient String idRegistry	Crear un nuevo paciente
	/participant/getPatients		Obtener la información de todos los pacientes registrados
	/participant/getPatientsById	String id	Obtener la información de un paciente por su id
	/participant/updatePatient	String idPatient String idRegistry	Modificar la información de un paciente
	/participant/deletePatient	String id	Eliminar un paciente del sistema
	/participant/createAdmin	String id String name String lastname	Crear un nuevo administrador
	/participant/getAdmins		Obtener la información de

Administrador			todos los administradores registrados
	/participant/getAdminsById	String id	Obtener la información de un administrador por su id
	/participant/updateAdmin	String id String name String lastname	Modificar la información de un administrador
	/participant/deleteAdmin	String id	Eliminar un administrador del sistema
Profesional	/participant/createDoctor	String idDoctor String idRegistry	Crear un nuevo profesional
	/participant/getDoctors		Obtener todos los profesionales registrados en el sistema
	/participant/getDoctorsById	String id	Obtener la información de un profesional por su id
	/participant/updateDoctor	String idDoctor String idRegistry	Modificar un profesional del sistema
	/participant/deleteDoctor	String id	Eliminar un profesional del sistema

La URL para tener acceso a los recursos del tipo “assets” (activos) puede verse en la tabla #7.

Tabla #7 - URL para el acceso a los activos del sistema Fuente: Elaboración propia			
Recurso	URL	Parámetros	Descripción
Perfil	/asset/createPerfilPersonal	String id String name String lastname String email String direction Int date String allergy String donor String bloodtype	Crear perfil personal, que luego se asocia con el paciente creado. Un perfil personal contiene toda la información personal del paciente.
	/asset/getPerfilPersonal		Obtiene todos los perfiles personales de los pacientes

Personal	/asset/getPerfilPersonalById	String id	Obtiene el perfil personal de un paciente identificado por su id
	/asset/updatePerfilPersonal	String id String name String lastname String email String direction Int date String allergy String donor String bloodtype	Modifica los datos de un perfil personal previamente creado
	/asset/deletePerfilPersonal	String id	Elimina un perfil personal de un paciente identificado por su id
Perfil Público	/asset/createPerfilPublico	String id String name String lastname String email String direction Int date String license	Crear perfil público, que luego se asocia con el profesional creado. Un perfil público contiene toda la información pública del profesional
	/asset/getPerfilPublico		Obtiene todos los perfiles públicos de los profesionales
	/asset/getPerfilPublicoById	String id	Obtiene el perfil público de un profesional identificado por su id
	/asset/updatePerfilPublico	String id String name String lastname String email String direction Int date String license	Modifica los datos de un perfil público previamente creado
	/asset/deletePerfilPublico	String id	Elimina un perfil público de un profesional identificado por su id
	/asset/createHistorialMedico	String idPatient String idRegistry	Crea un nuevo historial médico que se asocia a un participante

Historial Médico			previamente creado
	/asset/getHistorialMedico		Obtiene la información de todos los historiales médicos creados
	/asset/getHistorialMedicoById	String id	Obtiene la información de un historial médico identificado por su id
	/asset/deleteHistorialMedico	String id	Elimina un historial médico

La URL para tener acceso a las transacciones del sistema, puede verse en la tabla #8.

Tabla #8 - URL para el acceso a las transacciones del sistema Fuente: Elaboración propia			
Transacción	URL	Parámetros	Descripción
Ping network	/ping		Consultar el estado actual del sistema
Crear carta	/cardCreate	String name	Crear una tarjeta de identificación, para que posteriormente sea asociada a una identidad.
Asociar Identidad	/bindIdentity	String id String type	Asocia una identidad a una tarjeta previamente creada. Para esto, tiene que adjuntarse una tarjeta que no tenga una identificación asociada.
Dar acceso a un historial médico	/transaction/giveAccess	String idRegistry String idDoctor	Concede acceso de un historial médico a un profesional
Revocar acceso a un historial médico	/transaction/revokeAccess	String idRegistry String idDoctor	Revoca el acceso a un historial médico de un profesional
Subir un	/uploadRegistry	String idRegistry Float version Tar file	Actualiza el historial médico de un paciente. Esta operación recibe un

historial médico de un paciente			segundo dato adjunto, que es el archivo comprimido en formato TAR del historial médico
Descargar un historial médico de un paciente	/downloadRegistry	String idRegistry	Descarga el historial médico de un paciente

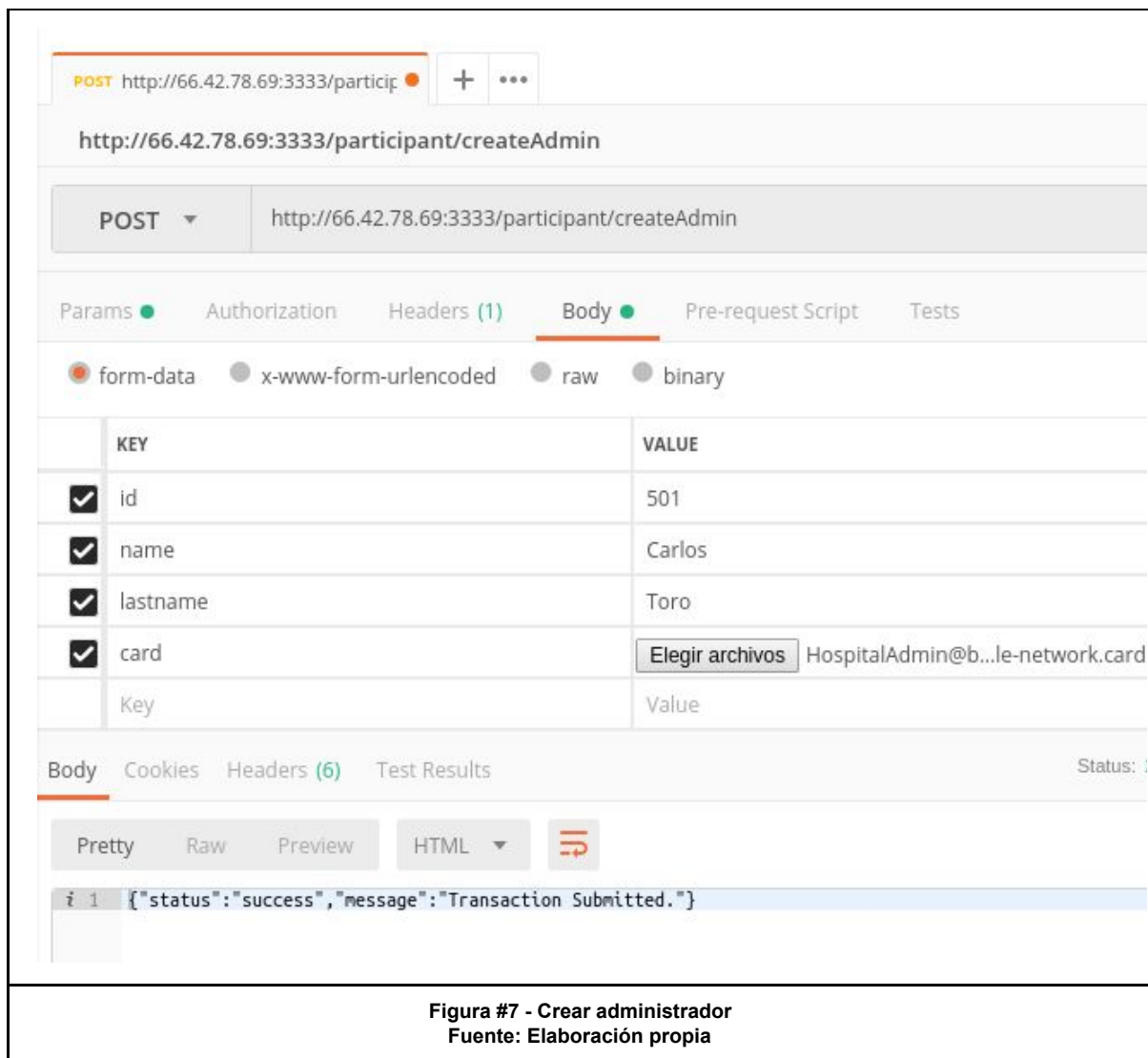
Es necesario aclarar que absolutamente todas las consultas con la API es necesario tener adjunto la tarjeta de identificación en el campo “card”.

4.5.2 Resultados

En esta sección se mostrarán los resultados exitosos de todas las llamadas a la API desarrollada. Estas llamadas se realizaron a través de la aplicación Postman, explicada anteriormente en la sección de Definición de herramientas de este mismo capítulo.

4.5.2.1 Administrador

En la figura #7 se muestra la operación crear administrador. Esta operación recibe el ID que tendrá el administrador, su nombre y su apellido. Esta operación, al igual que todas, necesitará una tarjeta de identificación con los permisos adecuados para poder realizar esta operación.



En la figura #8 se muestra la operación para consultar todos los administradores. Con esta operación es posible obtener la información de todos los participantes del tipo administrador.

En la figura #9 se muestra la operación para consultar un administrador dado su id. Esta operación a diferencia de la anterior, obtiene la información de un administrador en específico.

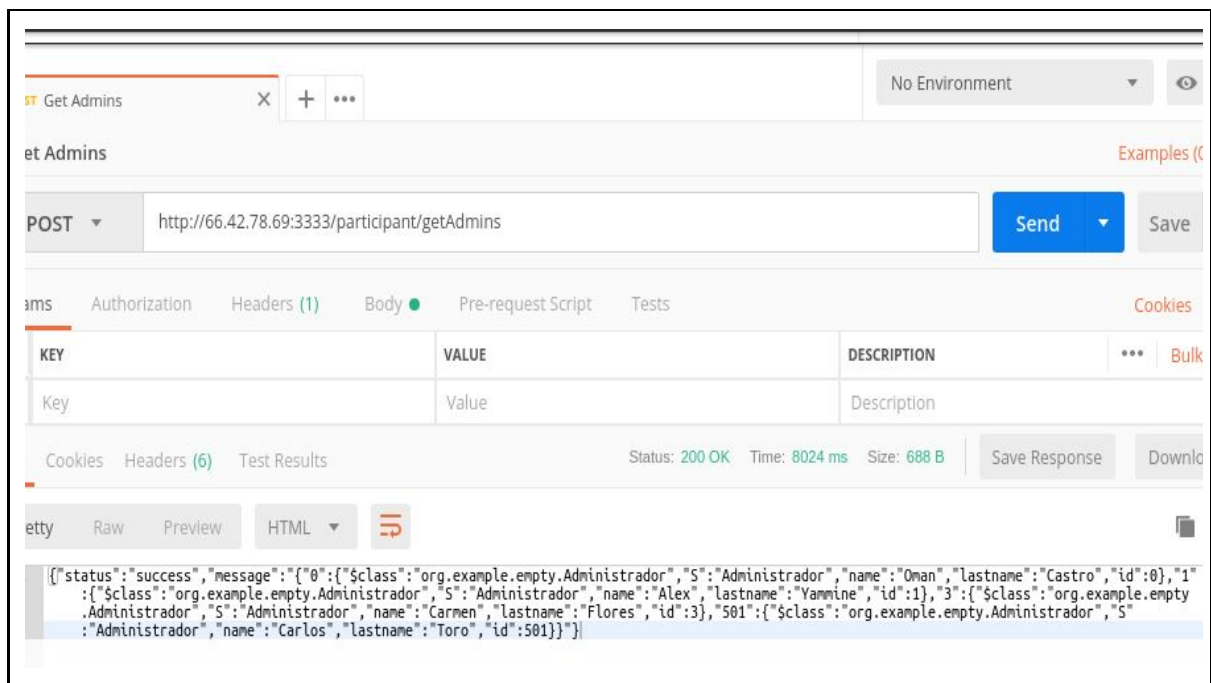


Figura #8 - Consultar administradores
Fuente: Elaboración propia

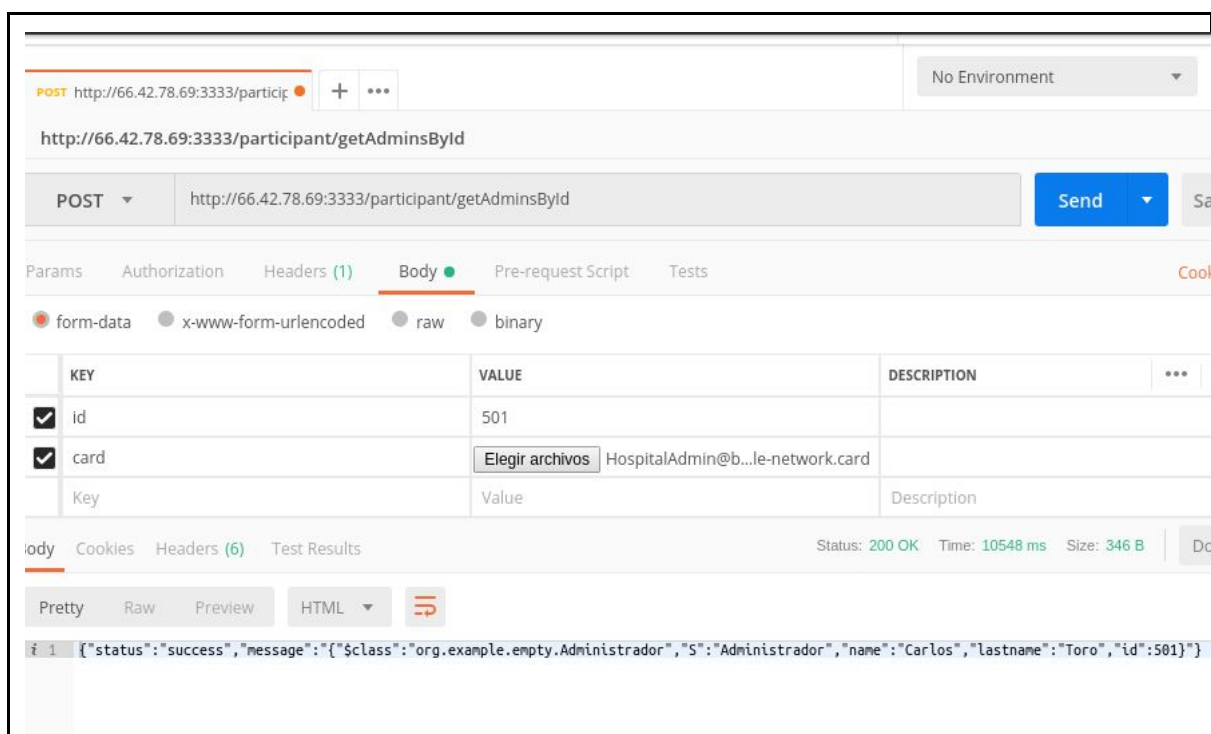


Figura #9 - Consultar un administrador por su id
Fuente: Elaboración propia

En la figura #10 se muestra la operación para modificar un administrador. Esta operación solicita los mismos datos que la operación para crear el administrador, con la diferencia de que el ID es para identificar al administrador que se va a modificar.

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://66.42.78.69:3333/participant/updateAdmin
- Body Type:** form-data
- Form Data:**

KEY	VALUE	DESCRIPTION
card	Elegir archivos HospitalAdmin@b...le-network.card	
name	Carlos	
lastname	Toro	
id	501	
- Status:** 200 OK
- Time:** 9969 ms
- Size:** 260 B
- Response Body:** {"status": "success", "message": "Transaction Submitted."}

Figura #10 - Modificar un administrador
Fuente: Elaboración propia

En la figura #11 se muestra la operación para eliminar un administrador previamente creado. Esta operación, al igual que todas las siguientes operaciones de eliminar participantes o activos, únicamente modifican el estado de la blockchain, ya que los registros de las transacciones evidencian de la existencia del administrador eliminado y estos registros no pueden ser eliminados de la blockchain.

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://66.42.78.69:3333/participant/deleteAdmin
- Body Type:** form-data
- Form Data:**

KEY	VALUE	DESCRIPTION
card	Elegir archivos HospitalAdmin@b...le-network.card	
id	500	
- Status:** 200 OK
- Time:** 11798 ms
- Size:** 260 B
- Response Body:** {"status": "success", "message": "Transaction Submitted."}

Figura #11 - Eliminar un administrador
Fuente: Elaboración propia

4.5.2.2 Perfil Personal

En la figura #12 se muestra la operación crear perfil personal. El activo perfil personal contiene la información básica de un paciente y que en caso de exponerse, no va a infringir una ninguna ley de privacidad pues los registros almacenados son públicos, como el tipo de sangre o alergias.

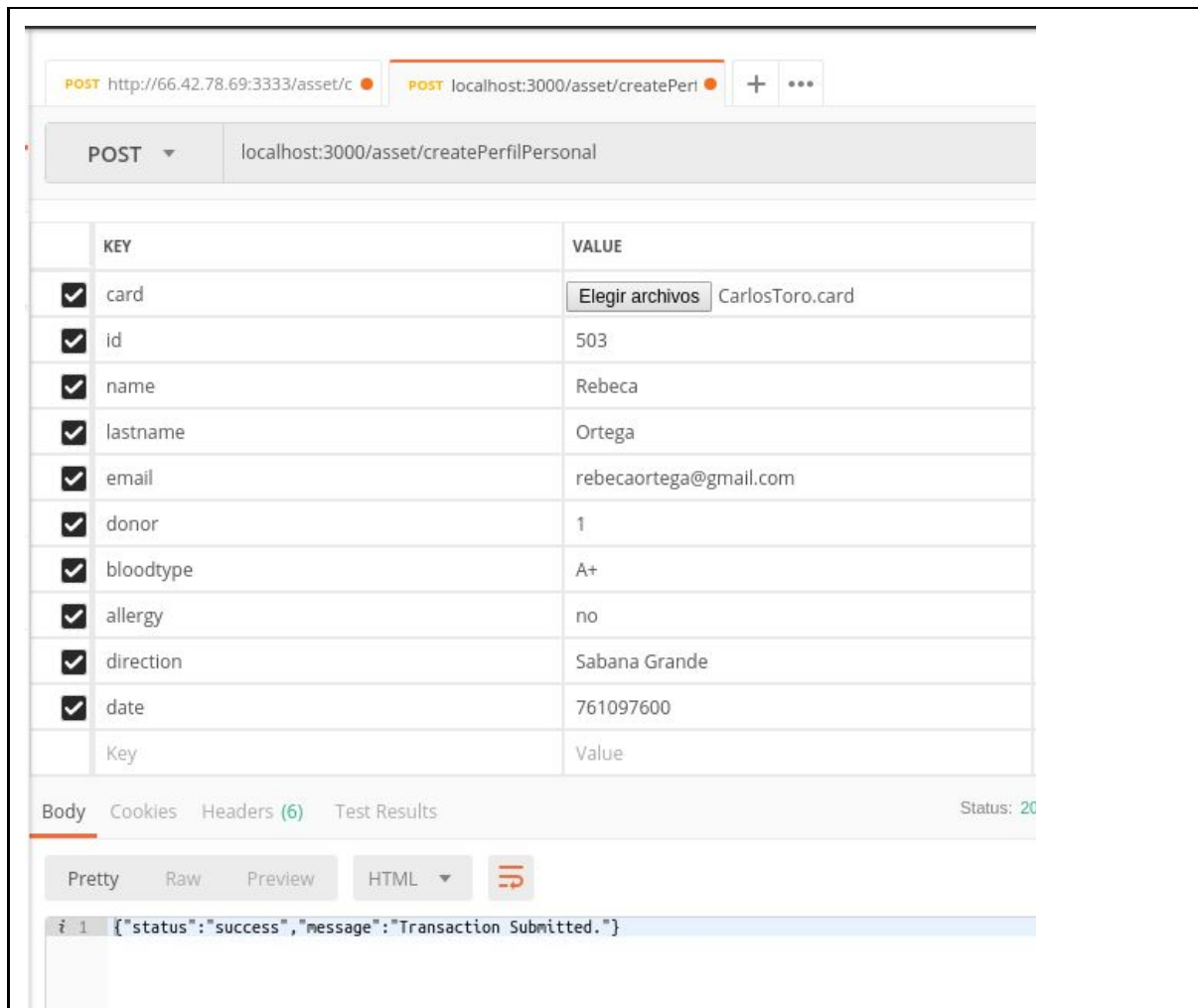


Figura #12 - Crear perfil personal
Fuente: Elaboración propia

En la figura #13 se muestra la operación para consultar todos los perfiles personales. Con esta operación es posible obtener la información de todos los activos del tipo perfil personal.

En la figura #14 se muestra la operación para consultar un perfil personal dado su id. Esta operación a diferencia de la anterior, obtiene la información de un perfil personal en específico.

The screenshot shows a REST client interface with a POST request to `http://66.42.78.69:3333/asset/getPerfilPersonal`. The response is a JSON array of personal profiles. The response status is 200 OK, with a time of 8490 ms and a size of 1.96 KB.

Figura #13 - Consultar perfiles personales
Fuente: Elaboración propia

En la figura #15 se muestra la operación para modificar un perfil personal. Para esta operación se exigen los mismos datos para la creación de un perfil personal.

POST

http://66.42.78.69:3333/asset/g

+

...

No Environment

http://66.42.78.69:3333/asset/getPerfilPersonalById

POST

http://66.42.78.69:3333/asset/getPerfilPersonalById

Send

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

	KEY	VALUE	DESCRIPTION
☒	card	Elegir archivos CarlosToro.card	
☒	id	503	
	Key	Value	Description

Body

Cookies

Headers (6)

Test Results

Status: 200 OK

Time: 7877 ms

Size: 454 B

Pretty

Raw

Preview

HTML

```

1 [{"status":"success","message":{"$class":"org.example.empty.PerfilPersonal","id":503,"nombre":"Rebeca","apellido":"Ortega","email":"rebecaortega@gmail.com","dob":761097600,"direccion":"Sabana Grande","TipoSangre":"A+","Alergias":"no","Donante":1}}]

```

Figura #14 - Consultar perfil personal dado un id

Fuente: Elaboración propia

POST

http://66.42.78.69:3333/asset/u

+

...

No Environment

POST

http://66.42.78.69:3333/asset/updatePerfilPersonal

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

	KEY	VALUE	DESCRIPTION
☒	card	Elegir archivos CarlosToro.card	
☒	id	503	
☒	name	Rebeca	
☒	lastname	Ortega	
☒	email	rebecaortega@gmail.com	
☒	date	761097600	
☒	donor	no	
☒	allergy	no	
☒	direction	Santa Ines	
☒	bloodtype	A+	
	Key	Value	Description

Body

Cookies

Headers (6)

Test Results

Status: 200 OK

Time: 15122 ms

Size: 260 B

Pretty

Raw

Preview

HTML

```

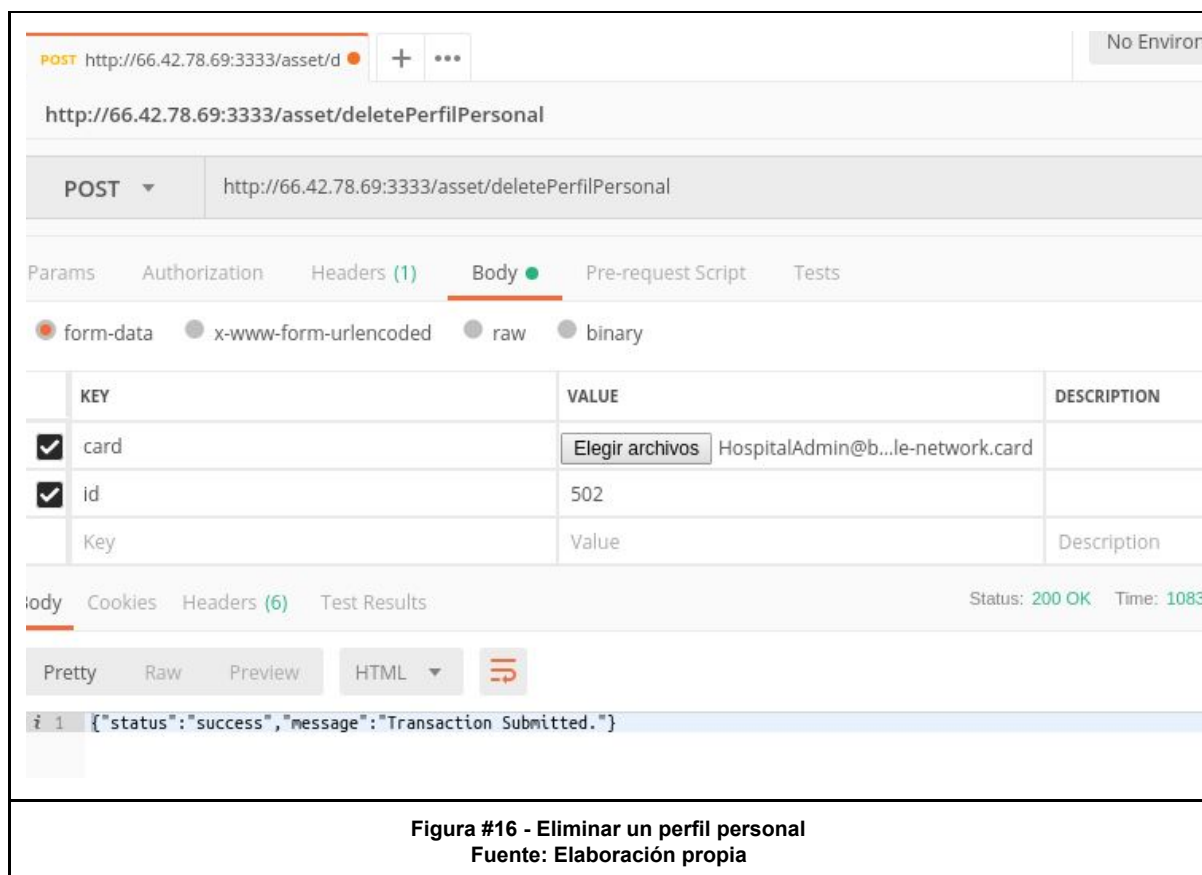
1 [{"status":"success","message":"Transaction Submitted."}]

```

Figura #15 - Modificar un perfil personal

Fuente: Elaboración propia

En la figura #16 se muestra la operación para eliminar un perfil personal dado su ID.



4.5.2.3 Perfil Público

En la figura #17 se muestra la operación para crear un perfil público. Un perfil público contiene toda la información necesaria para describir al participante “profesional”, esta incluye su información de contacto, su licencia, etc.

En la figura #18 se muestra la operación para consultar todos los perfiles públicos. Con esta operación es posible obtener la información de todos los activos del tipo perfil público.

En la figura #19 se muestra la operación para consultar un perfil público dado un id. Esta operación a diferencia de la anterior, obtiene la información de un perfil personal en específico.

POST http://66.42.78.69:3333/asset/c

POST http://66.42.78.69:3333/asset/createPerfilPublico

Params Authorization Headers (1) Body Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

KEY	VALUE	DESCRIPTION
card	Elegir archivos CarlosToro.card	
id	504	
name	Rafael	
lastname	Ramirez	
direction	Santa Fe	
date	29808033	
license	D93A56GH0	
email	rafael.ramirez@hotmail.com	
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time

Pretty Raw Preview HTML

```
1 [{"status": "success", "message": "Transaction Submitted."}]
```

Figura #17 - Crear un perfil público
Fuente: Elaboración propia

POST http://66.42.78.69:3333/asset/g

No Environment

http://66.42.78.69:3333/asset/getPerfilPublico

POST http://66.42.78.69:3333/asset/getPerfilPublico Send Save

Params Authorization Headers (1) Body Pre-request Script Tests Cookies

form-data x-www-form-urlencoded raw binary

KEY	VALUE	DESCRIPTION
card	Elegir archivos CarlosToro.card	
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 8075 ms Size: 1 KB Download

Pretty Raw Preview HTML

```
1 [{"status": "success", "message": [{"504": {"$class": "org.example.empty.PerfilPublico", "id": 504, "nombre": "Rafael", "apellido": "Ramirez", "email": "rafael.ramirez@hotmail.com", "dob": 29808033, "direccion": "Santa Fe", "licencia": "D93A56GH0"}, "2000": {"$class": "org.example.empty.PerfilPublico", "id": 2000, "nombre": "Alejandro", "apellido": "Barone", "email": "AlejandroBarone", "dob": 11110, "direccion": "123213123", "licencia": "123123123123", "9052964": {"$class": "org.example.empty.PerfilPublico", "id": 9052964, "nombre": "Roberto", "apellido": "Yammine", "email": "Roberto.yammine@gmail.com", "dob": 11110, "direccion": "SRL BARUTA", "licencia": "213123122", "14260918": {"$class": "org.example.empty.PerfilPublico", "id": 14260918, "nombre": "Jeanette", "apellido": "Yammine", "email": "Jeanetteyammine@hotmail.com", "dob": 0, "direccion": "SRL BARUTA", "licencia": "126116515"}}]}
```

Figura #18 - Consultar todos los perfiles públicos
Fuente: Elaboración propia

POST Get PerfilPublico

POST http://66.42.78.69:3333/asset/getPerfilPublicoById

Params Authorization Headers (1) **Body** Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

KEY	VALUE	DESCRIPTION
☒ card	Elegir archivos CarlosToro.card	
☒ id	504	
Key	Value	Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 11521 ms Size: 429 B Save Response

Pretty Raw Preview HTML

```
1 [{"status": "success", "message": {"$class": "org.example.empty.PerfilPublico", "id": 504, "nombre": "Rafael", "apellido": "Ramirez", "email": "rafael.ramirez@hotmail.com", "dob": 29808033, "direccion": "Santa Fe", "licencia": "D93A56GH0"}}]
```

Figura #19 - Consultar un perfil público por su id
Fuente: Elaboración propia

En la figura #20 se muestra la operación para modificar un perfil público. Esta operación solicita los mismos datos para crear un perfil público.

POST http://66.42.78.69:3333/asset/u

POST http://66.42.78.69:3333/asset/updatePerfilPublico

Params Headers (6) **Body** Pre-request Script Tests

form-data x-www-form-urlencoded raw binary

KEY	VALUE	DESCRIPTION
☒ card	Elegir archivos CarlosToro.card	
☒ id	504	
☒ name	Rafael	
☒ lastname	Ramirez	
☒ email	rafaelramirez@hotmail.com	
☒ direction	Santa fe	
☒ license	D93A56GH0	
☒ date	29808033	
Key	Value	Description

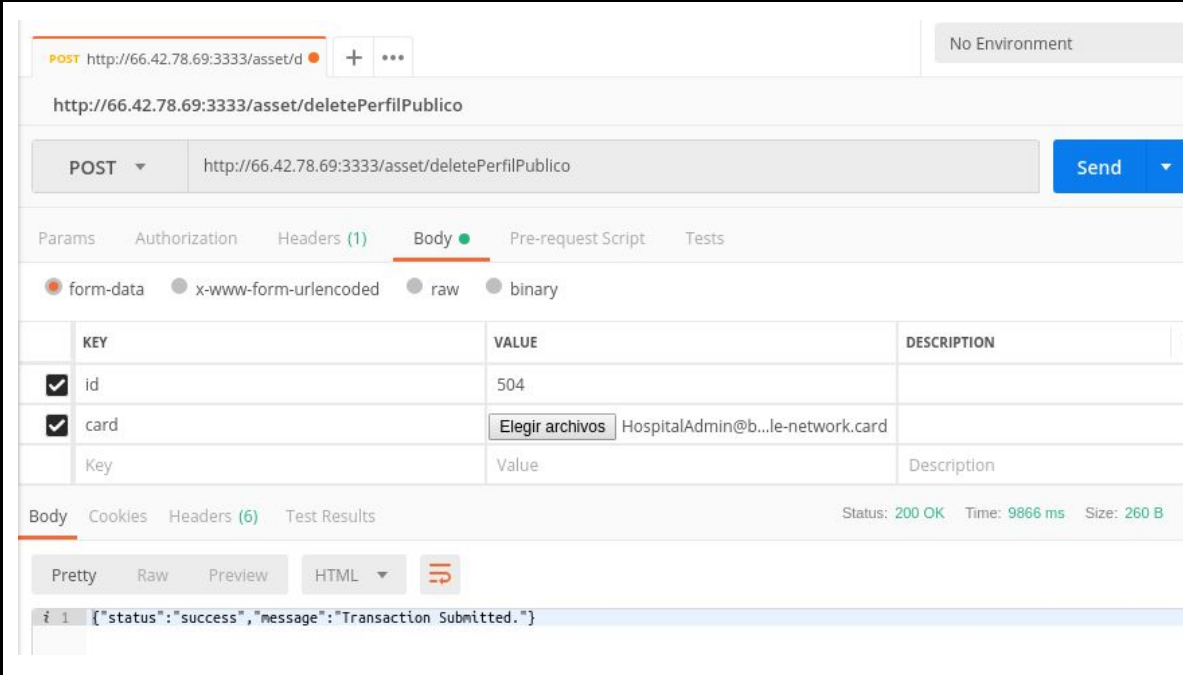
Body Cookies Headers (6) Test Results Status: 200 OK Time: 9857 ms Size: 260 B

Pretty Raw Preview HTML

```
1 [{"status": "success", "message": "Transaction Submitted."}]
```

Figura #20 - Modificar un perfil público
Fuente: Elaboración propia

En la figura #21 se muestra la operación para eliminar un perfil público dado su ID.



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://66.42.78.69:3333/asset/deletePerfilPublico
- Body Type:** form-data
- Form Data:**

KEY	VALUE	DESCRIPTION
☒ id	504	
☒ card	Elegir archivos HospitalAdmin@b...le-network.card	
Key	Value	Description
- Status:** 200 OK
- Time:** 9866 ms
- Size:** 260 B
- Response Body (JSON):**

```
{
  "status": "success",
  "message": "Transaction Submitted."
}
```

Figura #21 - Eliminar un perfil público
Fuente: Elaboración propia

4.5.2.4 Historial Médico

En la figura #22 se muestra la operación para crear un historial médico. Un historial médico contiene la información vital para dirección el archivo en IPFS que contiene toda la información médica del paciente. De modificar mal este registro, se podría poner en riesgo la información médica del paciente almacenada en IPFS.

En la figura # 23 se muestra la operación para consultar todos los historiales médicos. Esta operación solamente retorna los “metadatos” del historial médico, es decir, el hash del archivo en IPFS, la versión del mismo, y a que paciente pertenecen. Para obtener la información médica se tiene que realizar la operación “/downloadRegistry”.

En la figura #24 se muestra la operación para consultar un historial médico dado su id. Al igual que la operación anterior, esta operación retornará la información “metadata” de un historial médico dado su ID.

POST

http://66.42.78.69:3333/asset/c

+

...

No Environment

http://66.42.78.69:3333/asset/createHistorialMedico

POST

http://66.42.78.69:3333/asset/createHistorialMedico

Send

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

form-data

x-www-form-urlencoded

raw

binary

	KEY	VALUE	DESCRIPTION
☒	card	Elegir archivos CarlosToro.card	
☒	idPatient	503	
☒	idRegistry	503	
	Key	Value	Description

Body

Cookies

Headers (6)

Test Results

Status: 200 OK

Time: 14269 ms

Size: 1.02 KB

Pretty

Raw

Preview

HTML

Send

```

1 [{"status":"success","message":"Transaction Submitted."}]

```

Figura #22 - Crear historial médico

Fuente: Elaboración propia

POST

http://66.42.78.69:3333/asset/g

+

...

No Environment

http://66.42.78.69:3333/asset/getHistorialMedico

POST

http://66.42.78.69:3333/asset/getHistorialMedico

Send

Save

Params

Authorization

Headers (1)

Body

Pre-request Script

Tests

Cookies

form-data

x-www-form-urlencoded

raw

binary

	KEY	VALUE	DESCRIPTION	...	Bul
☒	card	Elegir archivos HospitalAdmin@b...le-network.card			
	Key	Value	Description		

Body

Cookies

Headers (6)

Test Results

Status: 200 OK

Time: 7945 ms

Size: 1.02 KB

Download

Pretty

Raw

Preview

HTML

Send

```

1 [{"status":"success","message":{"503":{"$class":"org.example.empty.HistorialMedico","id":503,"hash":null,"permisos":["(empty array)","version":0,"paciente":"resource:org.example.empty.Paciente#503"},"1000":{"$class":"org.example.empty.HistorialMedico","id":1000,"hash":"QmbF842yEAvBw9EcaA71zPzQjBSaseuJ08wchm4JoACHWS","permisos":["2000","version":1,"paciente":"resource:org.example.empty.Paciente#1000"},"2000":{"$class":"org.example.empty.HistorialMedico","id":2000,"hash":"QmbF842yEAvBw9EcaA71zPzQjBSaseuJ08wchm4JoACHWS","permisos":["2000","version":1,"paciente":"resource:org.example.empty.Paciente#2000"},"15600158":{"$class":"org.example.empty.HistorialMedico","id":15600158,"hash":"QmbF842yEAvBw9EcaA71zPzQjBSaseuJ08wchm4JoACHWS","permisos":["14260918,9052964","version":3,"paciente":"resource:org.example.empty.Paciente#15600158"]}}}]

```

Figura #23 - Consultar todos los historiales médicos

Fuente: Elaboración propia

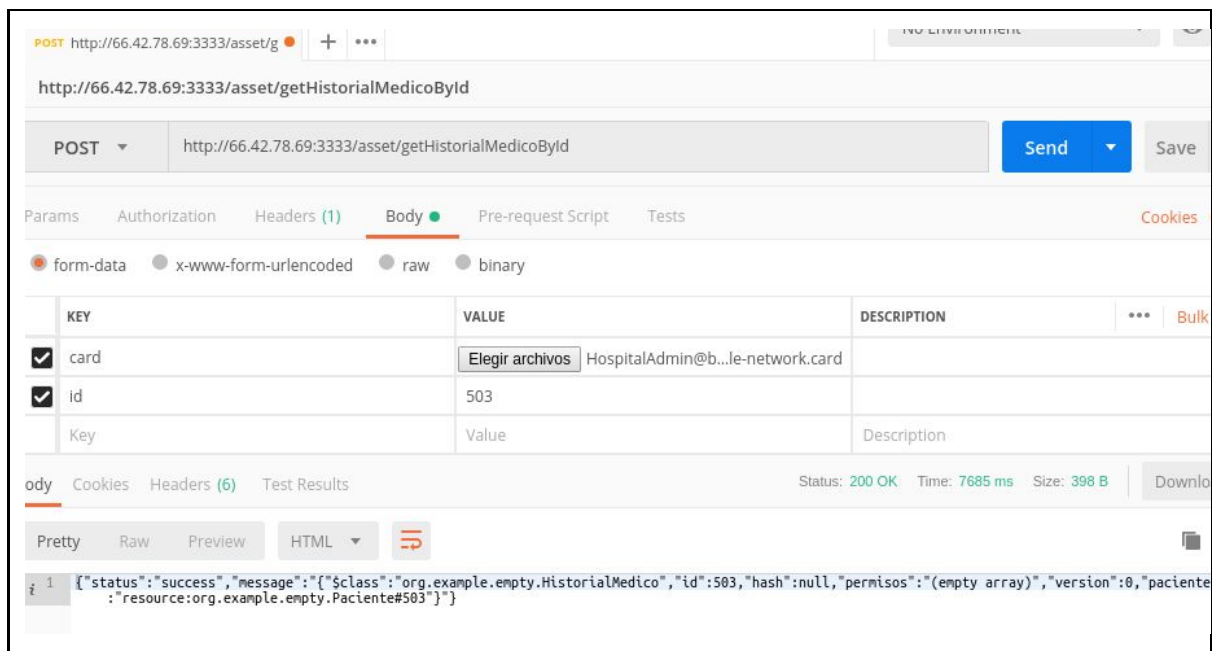


Figura #24 - Consultar un historial médico dado un id
Fuente: Elaboración propia

En la figura #25 se muestra la operación para eliminar un historial médico.

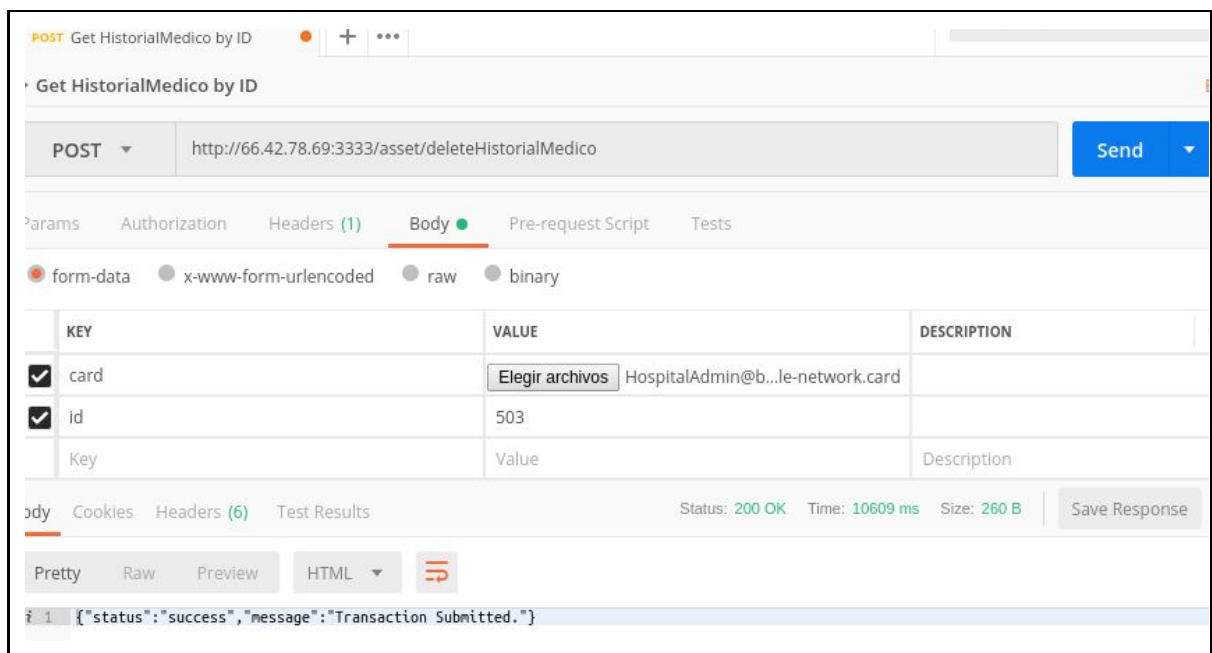
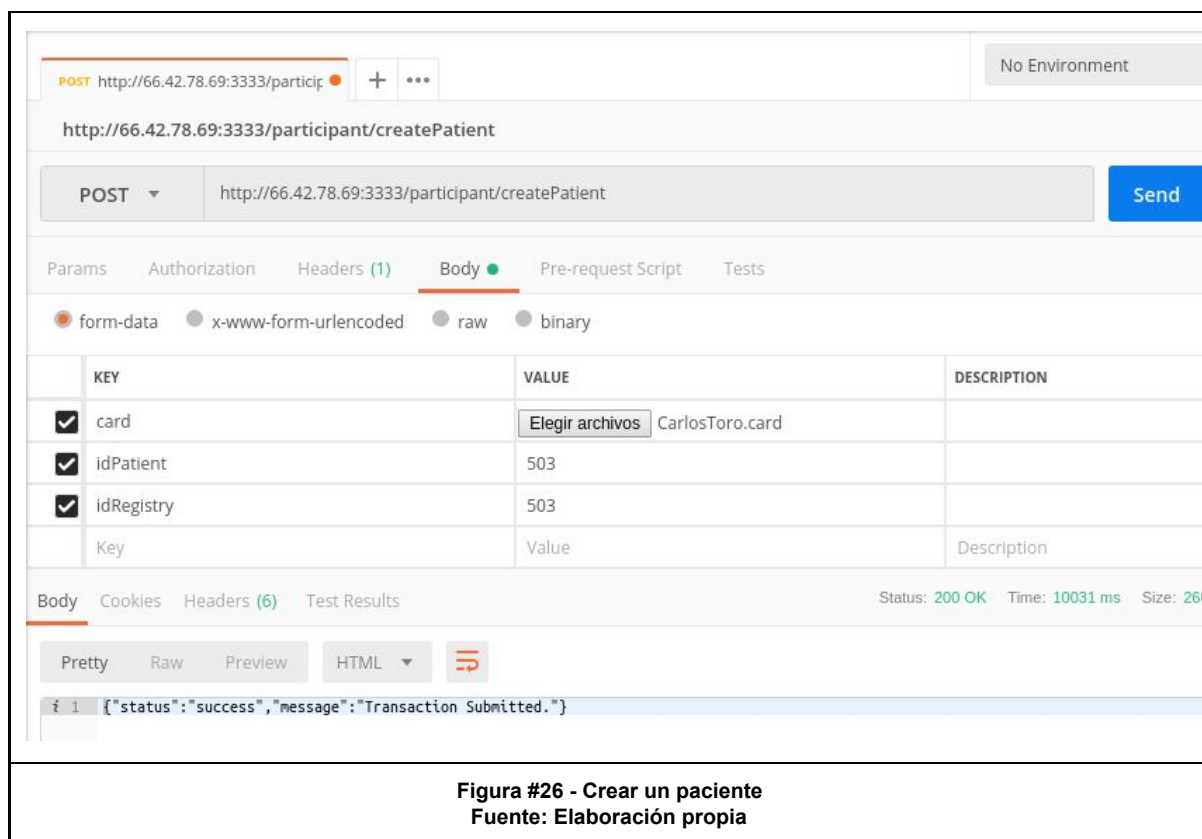


Figura #25 - Eliminar un historial médico
Fuente: Elaboración propia

4.5.2.5 Paciente

En la figura #26 se muestra la operación para crear un paciente. Esta operación crea un participante del tipo paciente, en este caso para mejor comodidad se le asignó el mismo ID del perfil personal, de manera que sea más fácil la búsqueda y la identificación de los registros.



En la figura #27 se muestra la operación para consultar todos los pacientes. Esta operación solo va a mostrar la información suministrada durante la creación del participante y un arreglo que contiene los profesionales tratante del mismo.

En la figura #28 se muestra la operación para consultar un paciente dado su id. Al igual que la operación anterior, nada más que mostrará la información de un solo paciente.

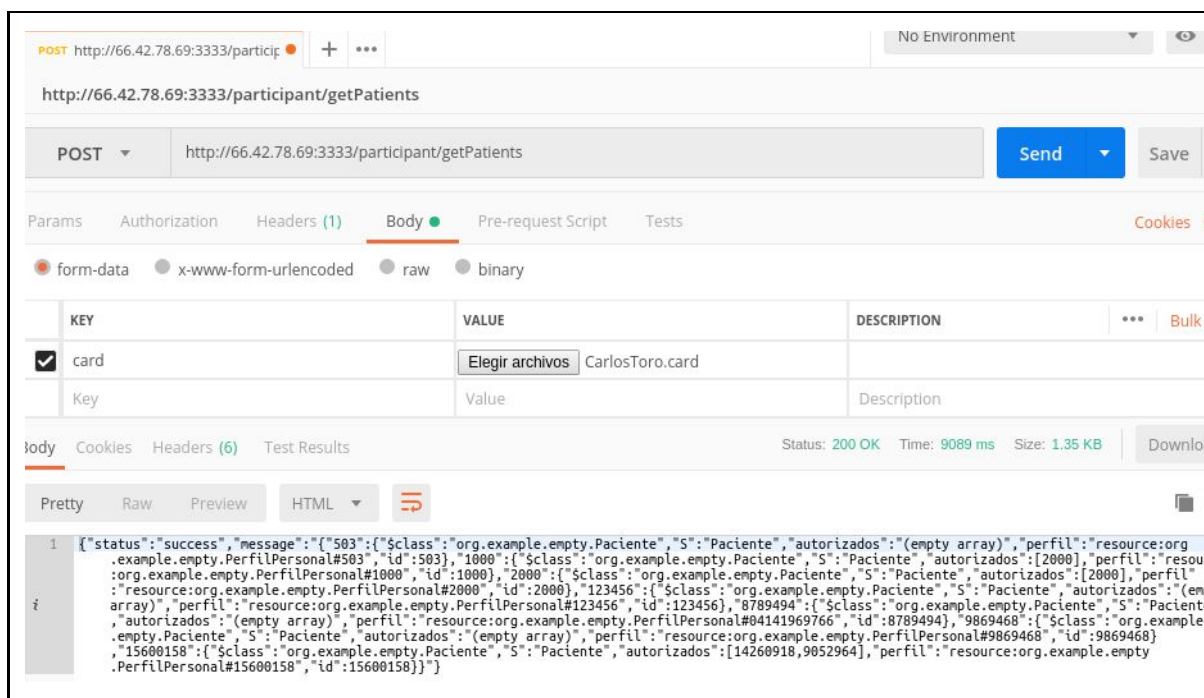


Figura #27 - Consultar todos los pacientes
Fuente: Elaboración propia

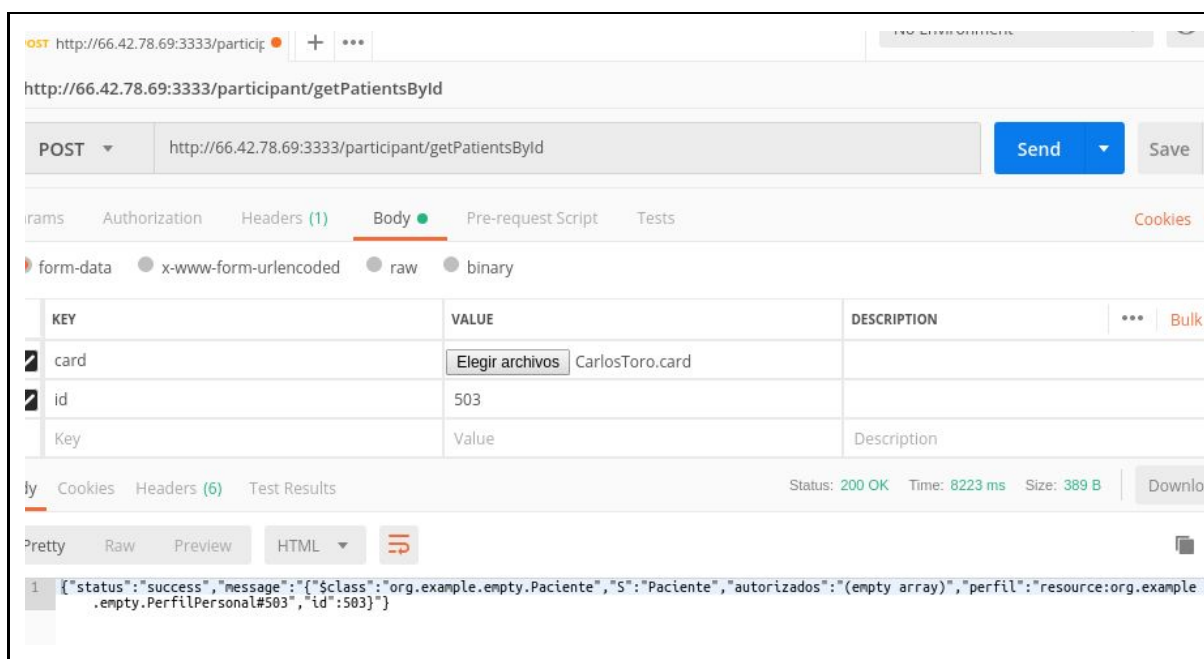
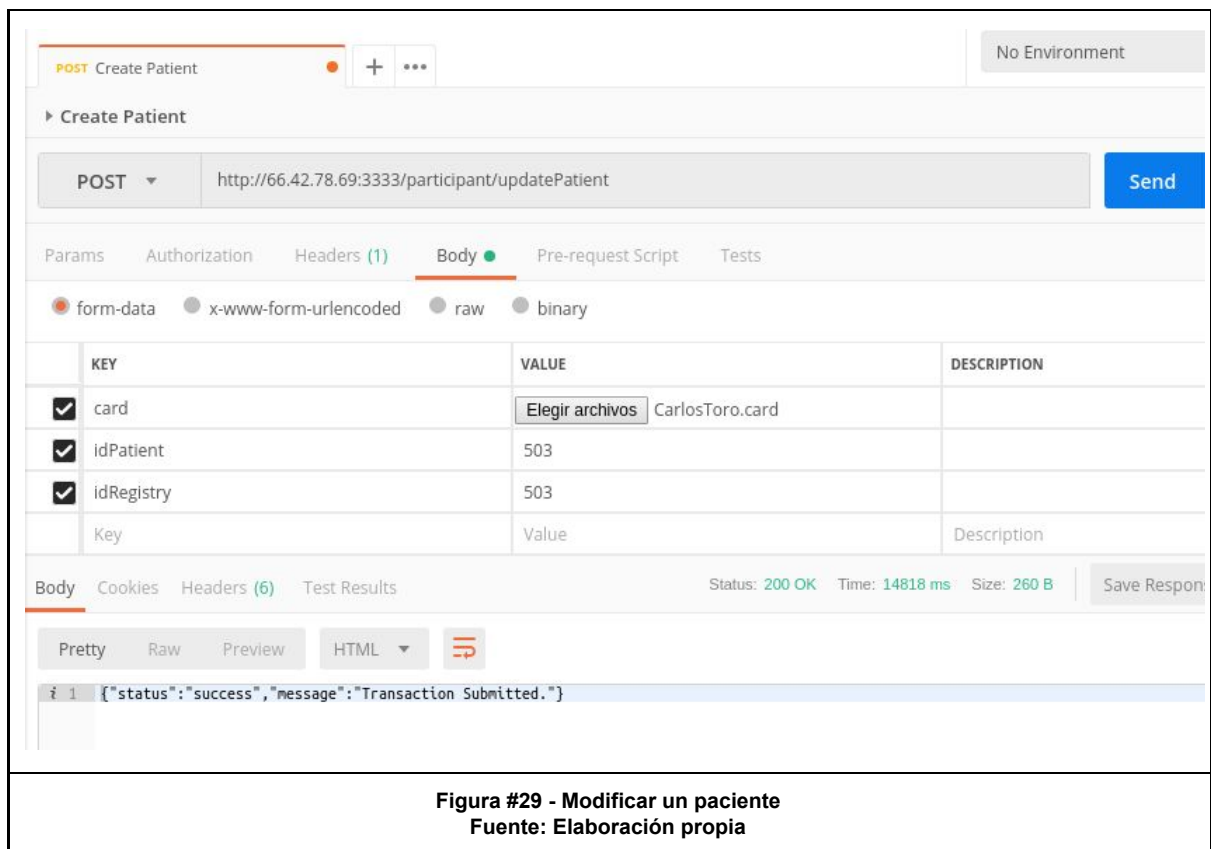
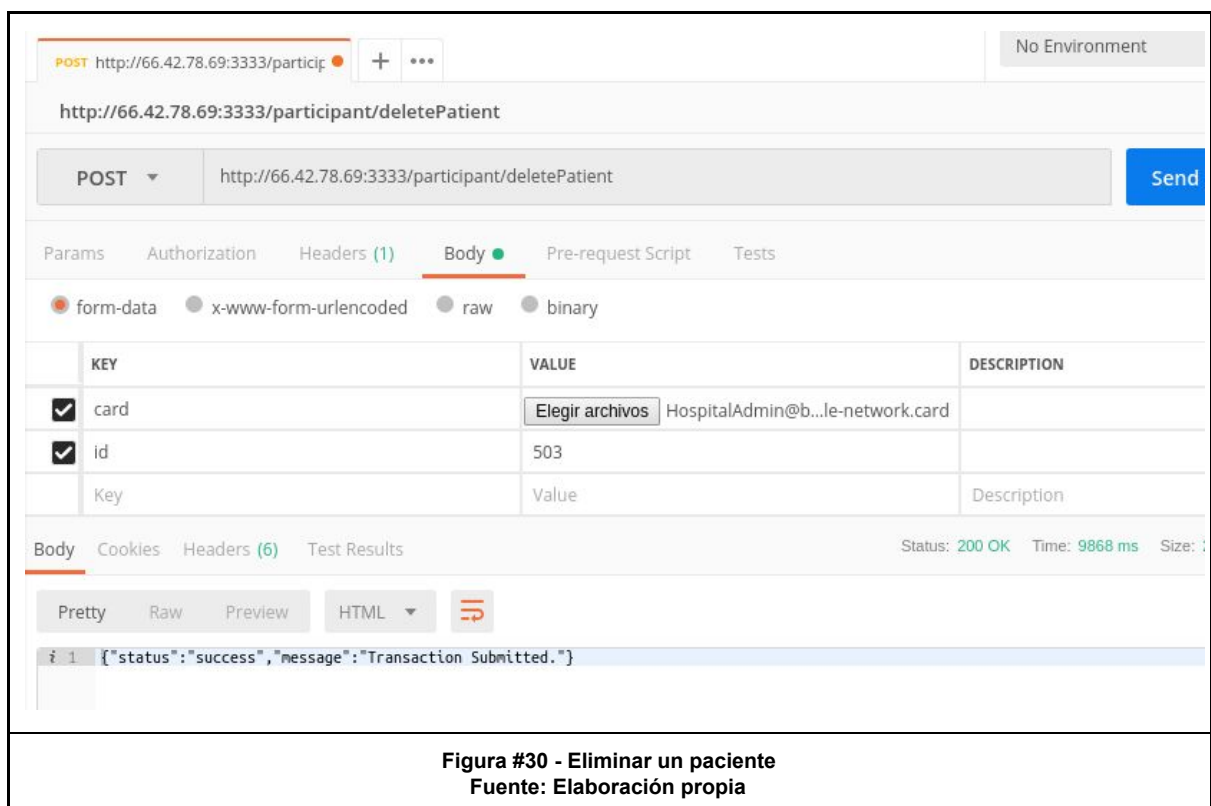


Figura #28 - Consultar un paciente por su id
Fuente: Elaboración propia

En la figura #29 se muestra la operación para modificar la información de un paciente. Esta operación es delicada, el único cambio que permite es el id del perfil personal asignado en el momento de la creación del paciente.



En la figura #30 se muestra la operación para eliminar a un paciente. Es importante recalcar que debería eliminarse después el perfil personal del paciente.



4.5.2.6 Profesional

En la figura #31 se muestra la operación para crear a un profesional.

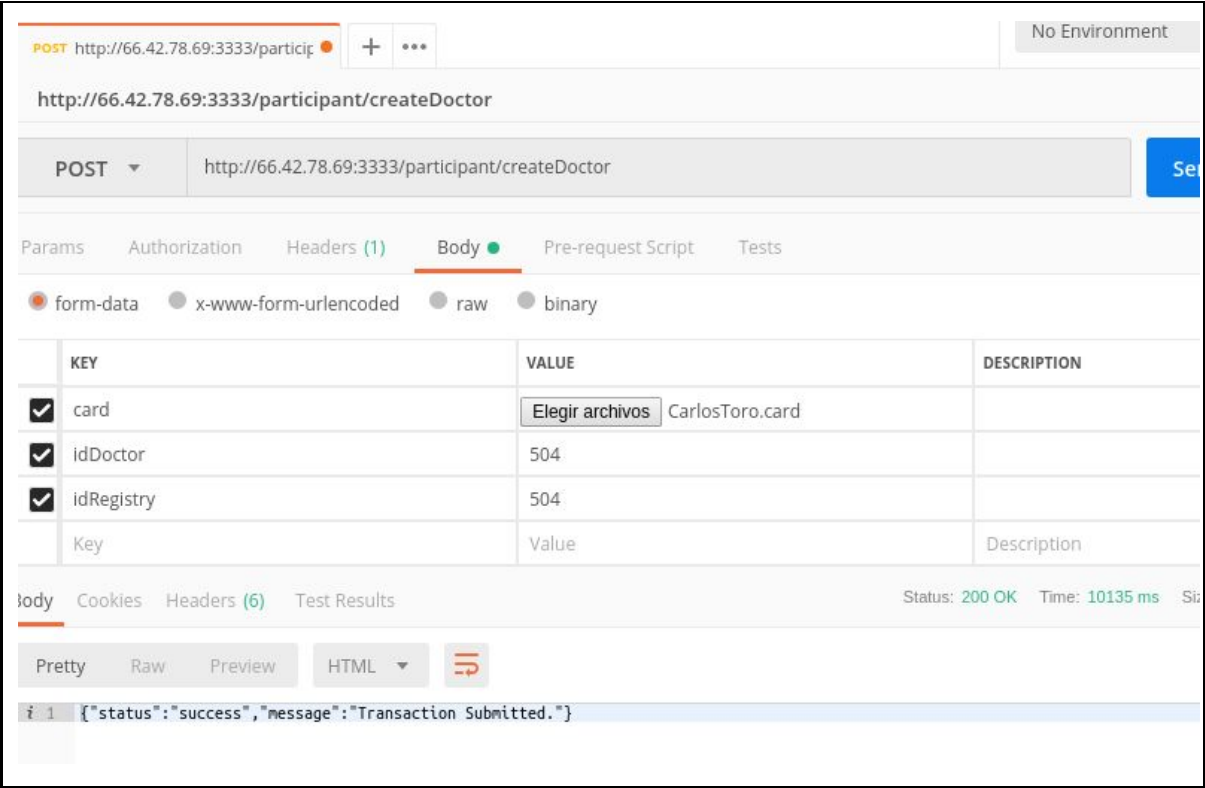


Figura #31 - Crear a un profesional
Fuente: Elaboración propia

En la figura #32 se muestra la operación para consultar todos los profesionales.

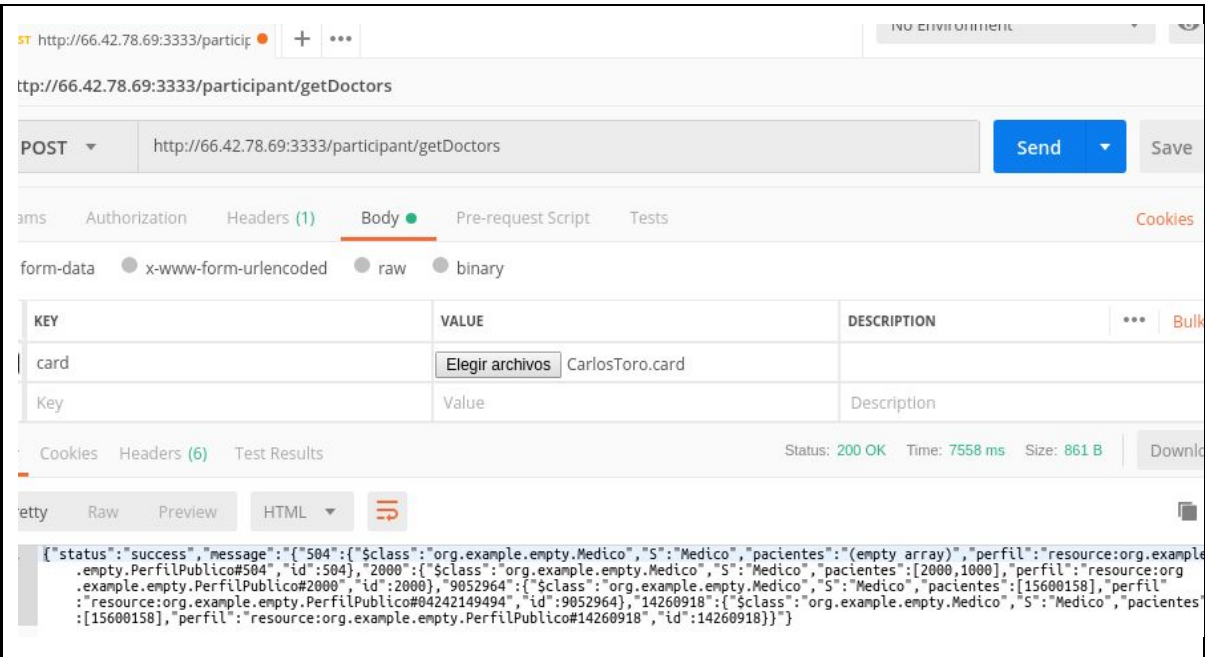
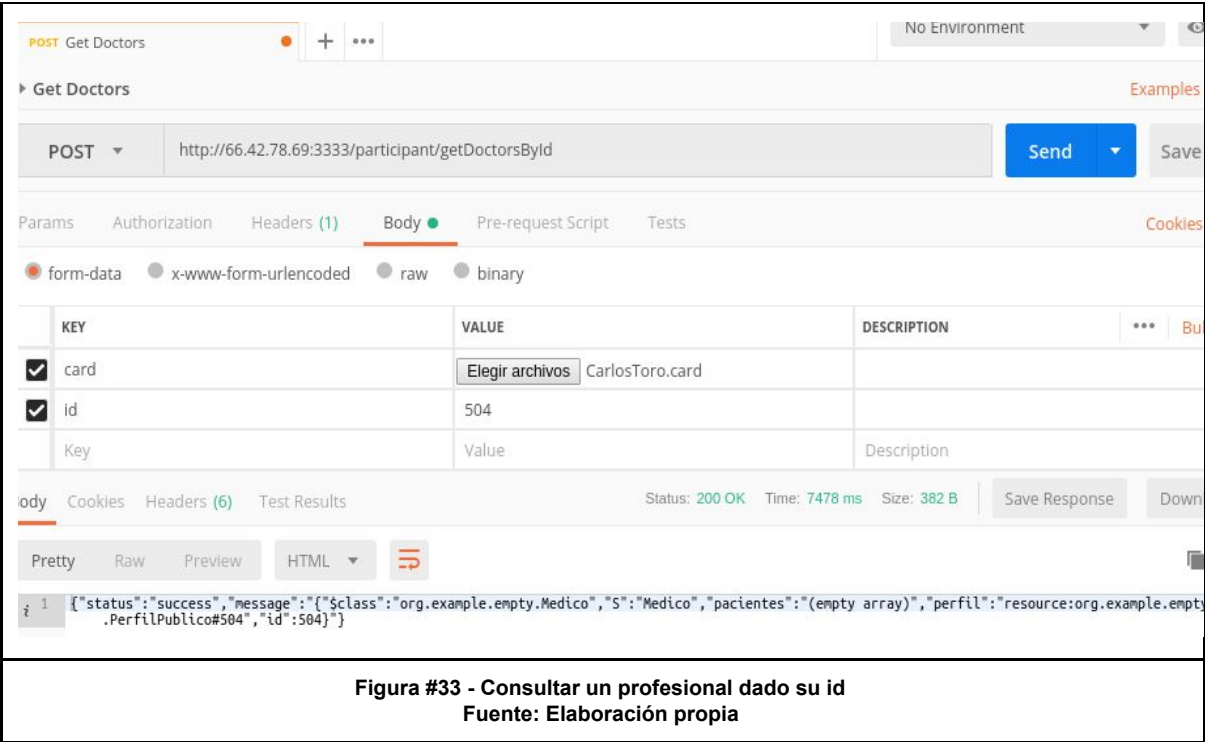
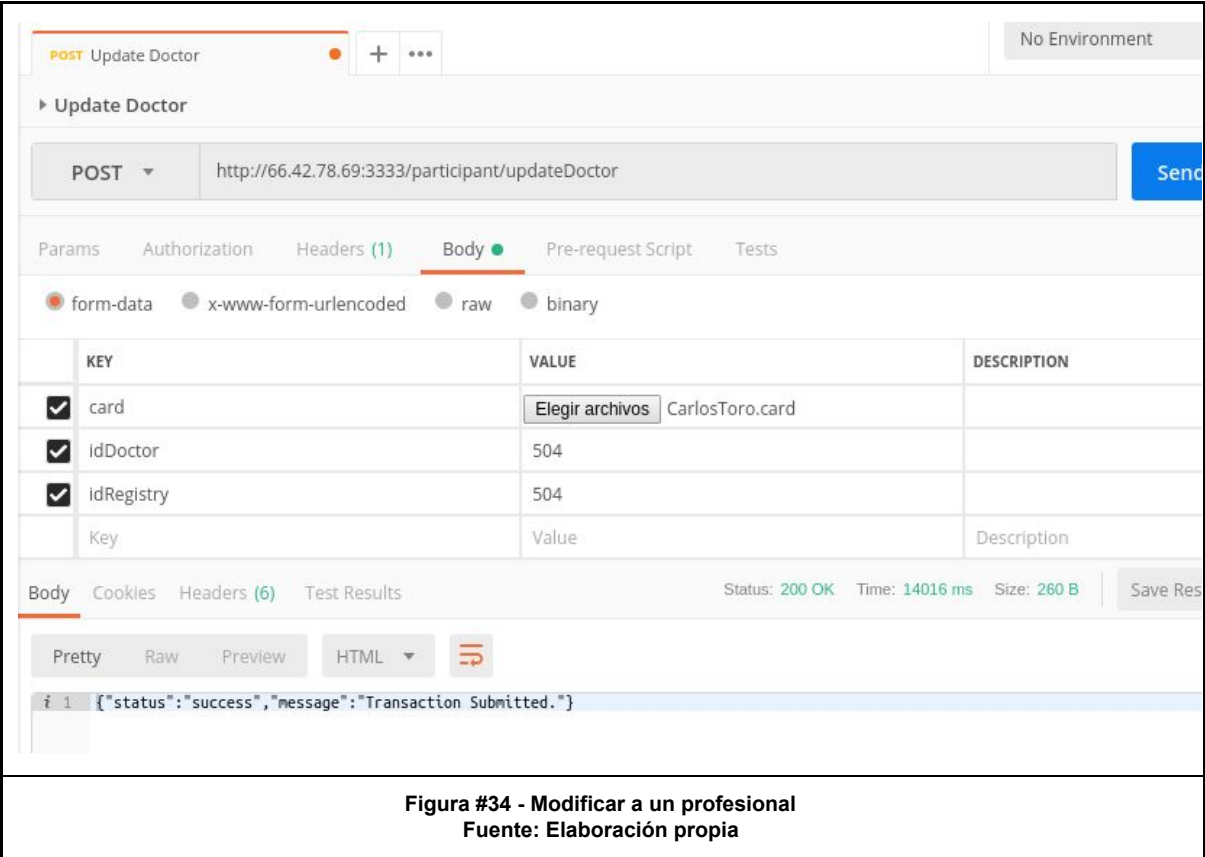


Figura #32 - Consultar todos los profesionales
Fuente: Elaboración propia

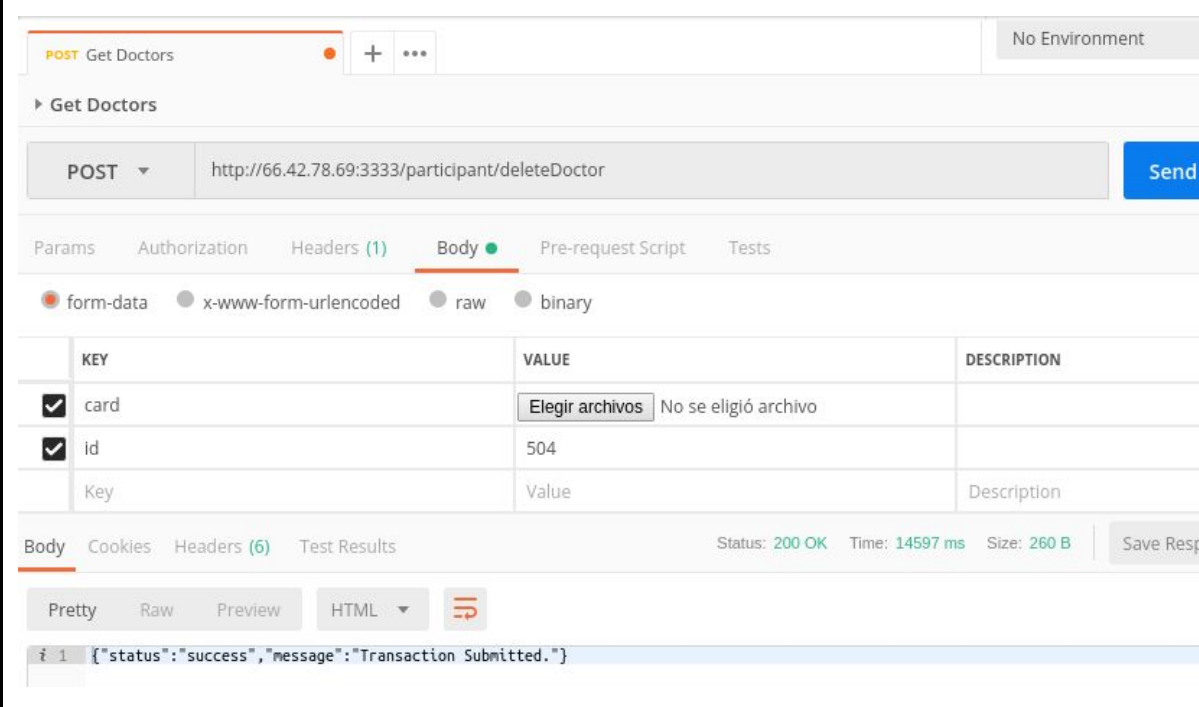
En la figura #33 se muestra la operación para obtener la información de un profesional dado su id.



En la figura #34 se muestra la operación para modificar a un profesional.



En la figura #35 se muestra la operación para eliminar a un profesional.



The screenshot shows a REST client interface with a POST request to `http://66.42.78.69:3333/participant/deleteDoctor`. The request body is form-data with the following fields:

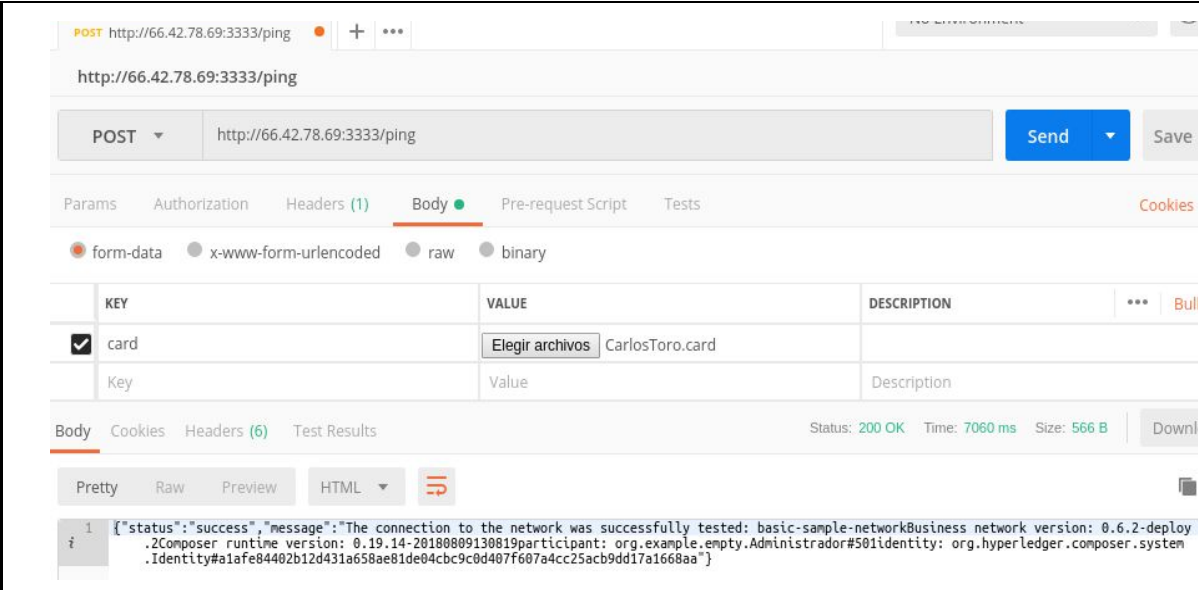
KEY	VALUE	DESCRIPTION
☒ card	Elegir archivos No se eligió archivo	
☒ id	504	

The response is a JSON object: `{ "status": "success", "message": "Transaction Submitted." }`. The status is 200 OK, time is 14597 ms, and size is 260 B.

Figura #35 - Eliminar a un profesional
Fuente: Elaboración propia

4.5.2.7 Transacciones

En la figura #36 se muestra la operación para consultar el estado de la red.



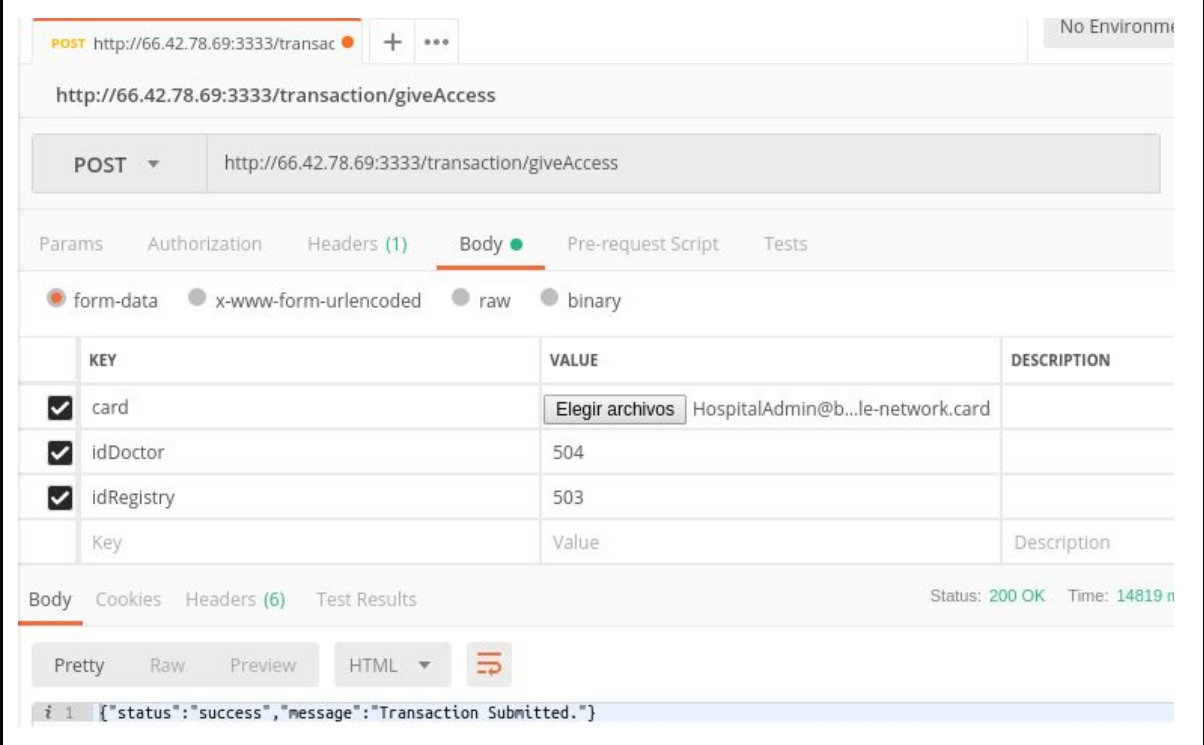
The screenshot shows a REST client interface with a POST request to `http://66.42.78.69:3333/ping`. The request body is form-data with the following fields:

KEY	VALUE	DESCRIPTION
☒ card	Elegir archivos CarlosToro.card	

The response is a JSON object: `{ "status": "success", "message": "The connection to the network was successfully tested: basic-sample-network&Business network version: 0.6.2-deploy...2Composer runtime version: 0.19.14-20180809130819participant: org.example.empty.Administrador#501identity: org.hyperledger.composer.system.Identity#a1afe84402b12d431a658ae81de04cbc9c0d407f607a4cc25acb9dd17a1668aa" }`. The status is 200 OK, time is 7060 ms, and size is 566 B.

Figura #36 - Consultar estado de la red
Fuente: Elaboración propia

En la figura #37 se muestra la operación para otorgar acceso de un historial médico a un profesional. Al asignarle permisos al médico, este podrá acceder o modificar el historial médico.



The screenshot shows a REST client interface with the following details:

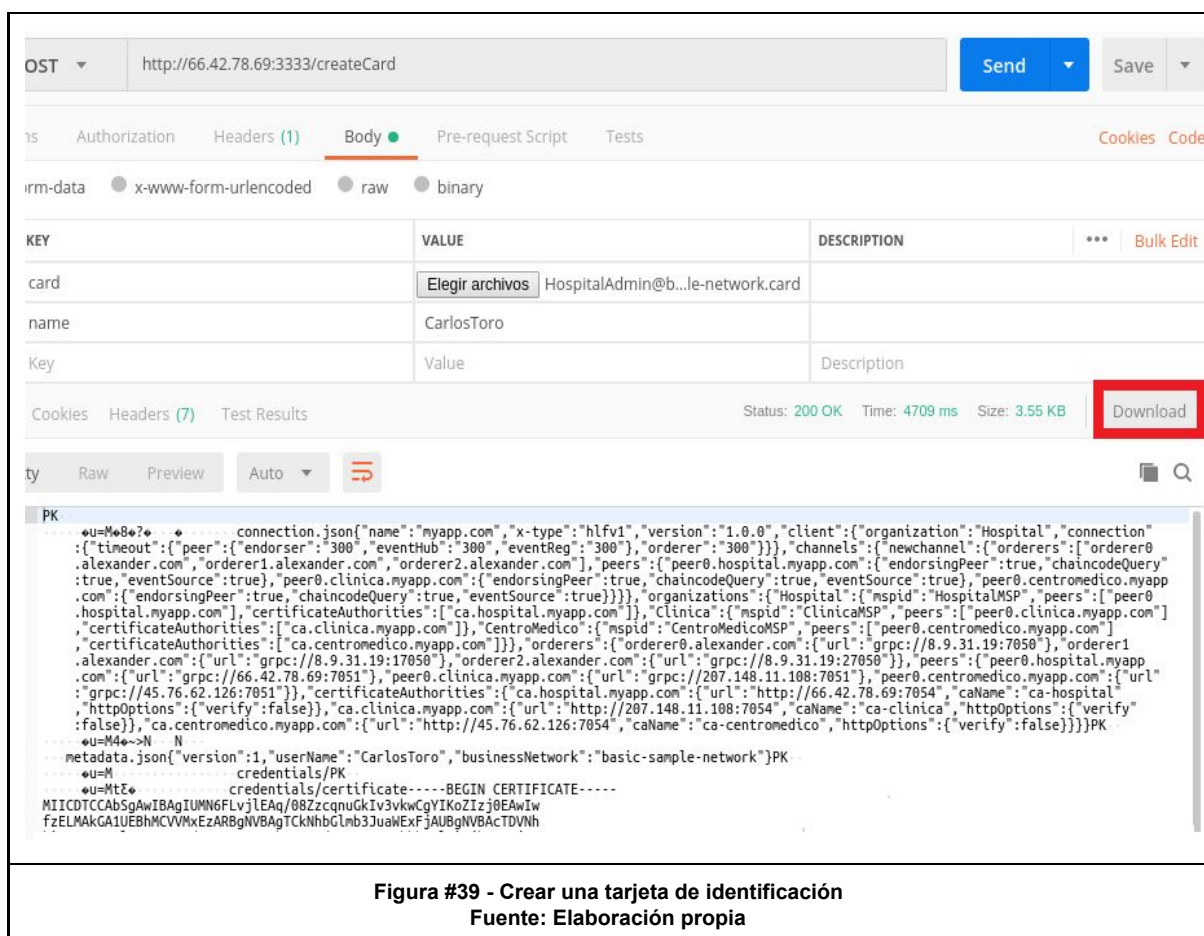
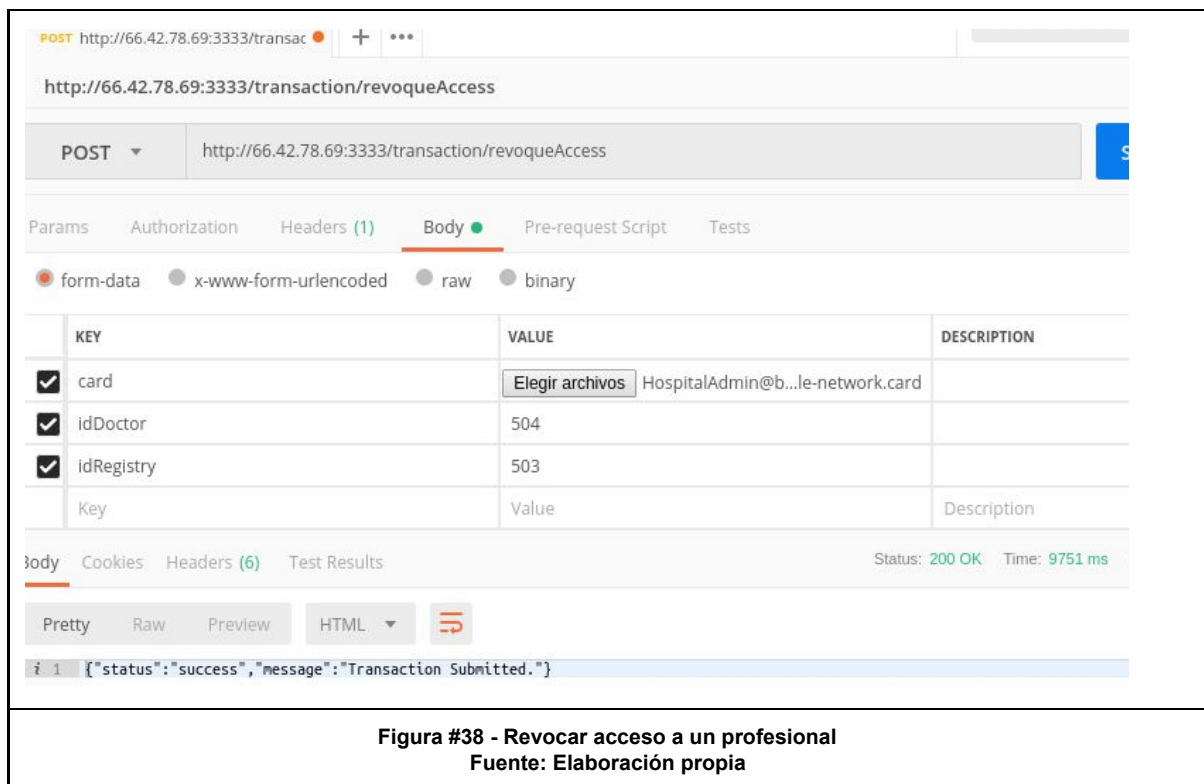
- Method:** POST
- URL:** http://66.42.78.69:3333/transaction/giveAccess
- Body Type:** form-data
- Body Fields:**

KEY	VALUE	DESCRIPTION
card	Elegir archivos HospitalAdmin@b...le-network.card	
idDoctor	504	
idRegistry	503	
Key	Value	Description
- Status:** 200 OK
- Time:** 14819 n
- Response Body:** [{"status": "success", "message": "Transaction Submitted."}]

Figura #37 - Otorgar acceso a un profesional
Fuente: Elaboración propia

En la figura #38 se muestra la operación para revocar el acceso a un profesional de un historial médico. Al revocar el acceso a un profesional de un historial médico, este perderá el permiso de tener acceso sea para consultar o modificar la información médica del paciente.

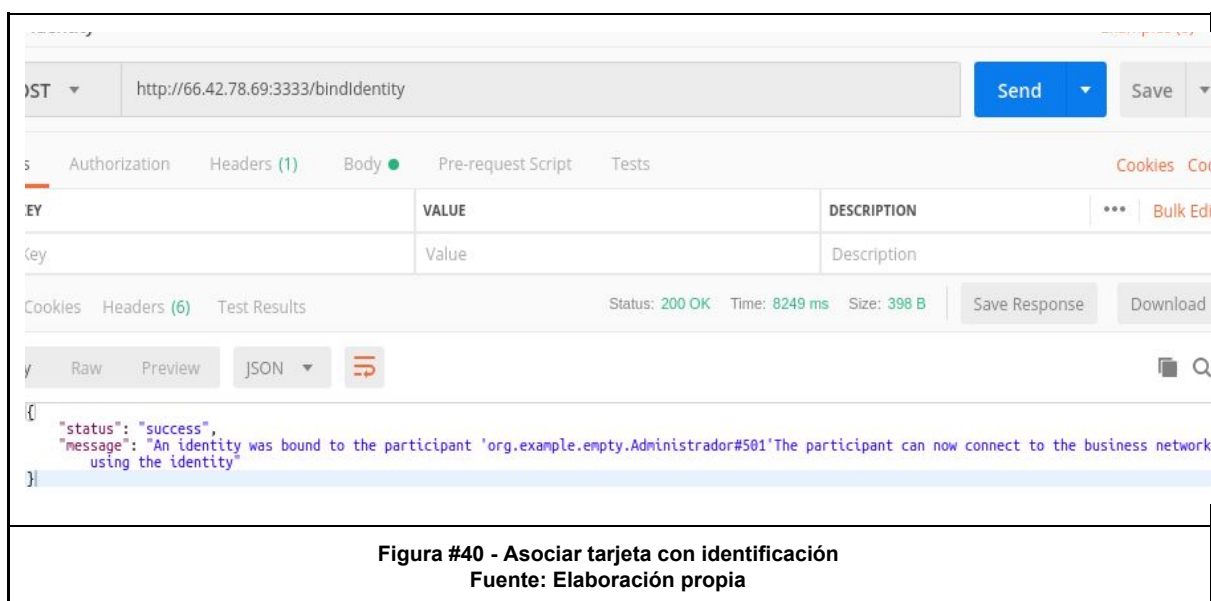
En la figura #39 se muestra la operación para crear una tarjeta de identificación, el resultado de esta operación va a dar como respuesta un archivo para descargar. La tarjeta de identificación es única, dentro se almacena la clave privada del usuario, de perderse, no habría forma alguna de generarse de nuevo, pues la seguridad del sistema, es gracias a que el usuario firme su transacción con la clave privada para validar que la operación la esté haciendo el mismo y no otro usuario que esté usurpando su identidad.



En la figura #40 se muestra la operación para asociar una tarjeta previamente creada con una identidad. Cabe aclarar que la tarjeta que se adjunta es la que se creó en el paso anterior y no una tarjeta con una identificación.

En la figura #41 se muestra la operación para subir un historial médico, para realizar esta operación tiene que adjuntarse el historial médico en formato TAR, este formato es un estándar de compresión.

En la figura #42 se muestra la operación para descargar un historial médico previamente subido. La respuesta de esta petición, de ser correcta será un archivo comprimido en formato TAR que contendrá el historial médico.



4.6 Pruebas Unitarias

Las pruebas unitarias se utilizan para comprobar el correcto funcionamiento de una unidad de código. En este caso, las pruebas unitarias se realizaron para cada una de las operaciones mencionadas en la sección anterior (4.5.1 Interfaz HTTP). Para lograr esto, se utilizaron dos módulos de javascript llamadas Mocha y Chai. En las figuras #43, #44 y #45 se pueden visualizar los resultados de las pruebas unitarias.

```
Admin
  Create an Admin
    ✓ Should create a new Admin (10632ms)
  Get Admins
    ✓ Should get all Admins (7774ms)
  Get an Admin by id
    ✓ Should get an Admin by id (8235ms)
  Update Admin
    ✓ Should update an Admin (15162ms)
  Delete Admin
    ✓ Should delete and Admin (9809ms)

PerfilPersonal
  Create PerfilPersonal
    ✓ Should create new PerfilPersonal (10006ms)
  Get all PerfilPersonal
    ✓ Should get all PerfilPersonal (8172ms)
  Get one PerfilPersonal by id
    ✓ Should get one PerfilPersonal by id (8230ms)
  Update PerfilPersonal
    ✓ Should update PerfilPersonal (9873ms)
  Delete PerfilPersonal
    ✓ Should delete PerfilPersonal (14807ms)

PerfilPublico
  Create PerfilPublico
    ✓ Should create new PerfilPublico (9868ms)
  Get all PerfilPublico
    ✓ Should get all PerfilPublico (8203ms)
  Get one PerfilPublico by id
    ✓ Should get one PerfilPublico by id (8619ms)
  Update PerfilPublico
    ✓ Should update PerfilPublico (10135ms)
  Delete PerfilPublico
    ✓ Should delete PerfilPublico (11390ms)

HistorialMedico
  Create HistorialMedico
    ✓ Should create new HistorialMedico (14990ms)
  Get all HistorialMedico
    ✓ Should get all HistorialMedico (7856ms)
  Get one HistorialMedico by id
```

Figura #43 - Pruebas unitarias (1)
Fuente: Elaboración propia

```
HistorialMedico
  Create HistorialMedico
    ✓ Should create new HistorialMedico (14990ms)
  Get all HistorialMedico
    ✓ Should get all HistorialMedico (7856ms)
  Get one HistorialMedico by id
    ✓ Should get one HistorialMedico by id (7797ms)
  Delete HistorialMedico
    ✓ Should delete HistorialMedico (10045ms)

Patient
  Create Patient
    ✓ Should create a new Patient (9759ms)
  Get Patients
    ✓ Should get all Patients (7700ms)
  Get Patient by id
    ✓ Should get Patient by id (14011ms)
  Update Patient
    ✓ Should update Patient (9844ms)
  Delete Patient
    ✓ Should delete Patient (10325ms)

Doctor
  Create Doctor
    ✓ Should create a new Doctor (9627ms)
  Get Doctors
    ✓ Should get all Doctors (7681ms)
  Get Doctor by id
    ✓ Should get Doctor by id (8275ms)
  Update Doctor
    ✓ Should update Doctor (15588ms)
  Delete Doctor
    ✓ Should delete Doctor (9476ms)
```

Figura #44 - Pruebas unitarias (2)
Fuente: Elaboración propia

```
Transaction operation
  Give access to a Doctor
    ✓ Should give access to a Doctor (10132ms)
  Revoke access to a Doctor
    ✓ Should revoke access to a Doctor (10289ms)

Registry operation
  Upload Registry
    ✓ Should upload registry (24422ms)
  Download Registry
    ✓ Should download registry (10459ms)
  Create card
    ✓ Should create card (3745ms)
  Ping the network
    ✓ Should ping the network (7149ms)

35 passing (6m)
```

Figura #45 - Pruebas unitarias (3)
Fuente: Elaboración propia

ANEXO A - DIAGRAMAS DE SECUENCIA

Dentro de este anexo se va continuar la descripción de los diagramas de secuencias del capítulo 4 del presente documento. Ya que muchos de estos diagramas se tornan repetitivos, se decidió anexarlos a parte para no interrumpir el flujo de dicho capítulo.

- **Diagrama de secuencia para operaciones de participante**

En el diagrama #10 se encuentra visualizada la operación de modificar participante. Las modificaciones dentro del sistema siempre quedarán registradas en la blockchain.

En el diagrama #11 se encuentra visualizada la operación de consultar participante. A diferencia de las operaciones de modificación, las operaciones de solo lectura no generan transacciones para la blockchain, pues no modifican el estado del mismo.

En el diagrama #12 se encuentra visualizada la operación de eliminar participante. Al eliminar un participante que tenga una tarjeta de identidad, esta quedará inútil y no se podrá realizar operaciones a la API con la tarjeta de identificación del participante eliminado.

- **Diagrama de secuencia para operaciones de perfil público/personal**

En el diagrama #13 se encuentra visualizada la operación crear perfil público/personal. Dado que estas operaciones realizan el mismo proceso con la diferencia de los datos por parámetros, el diagrama aplica para ambos activos.

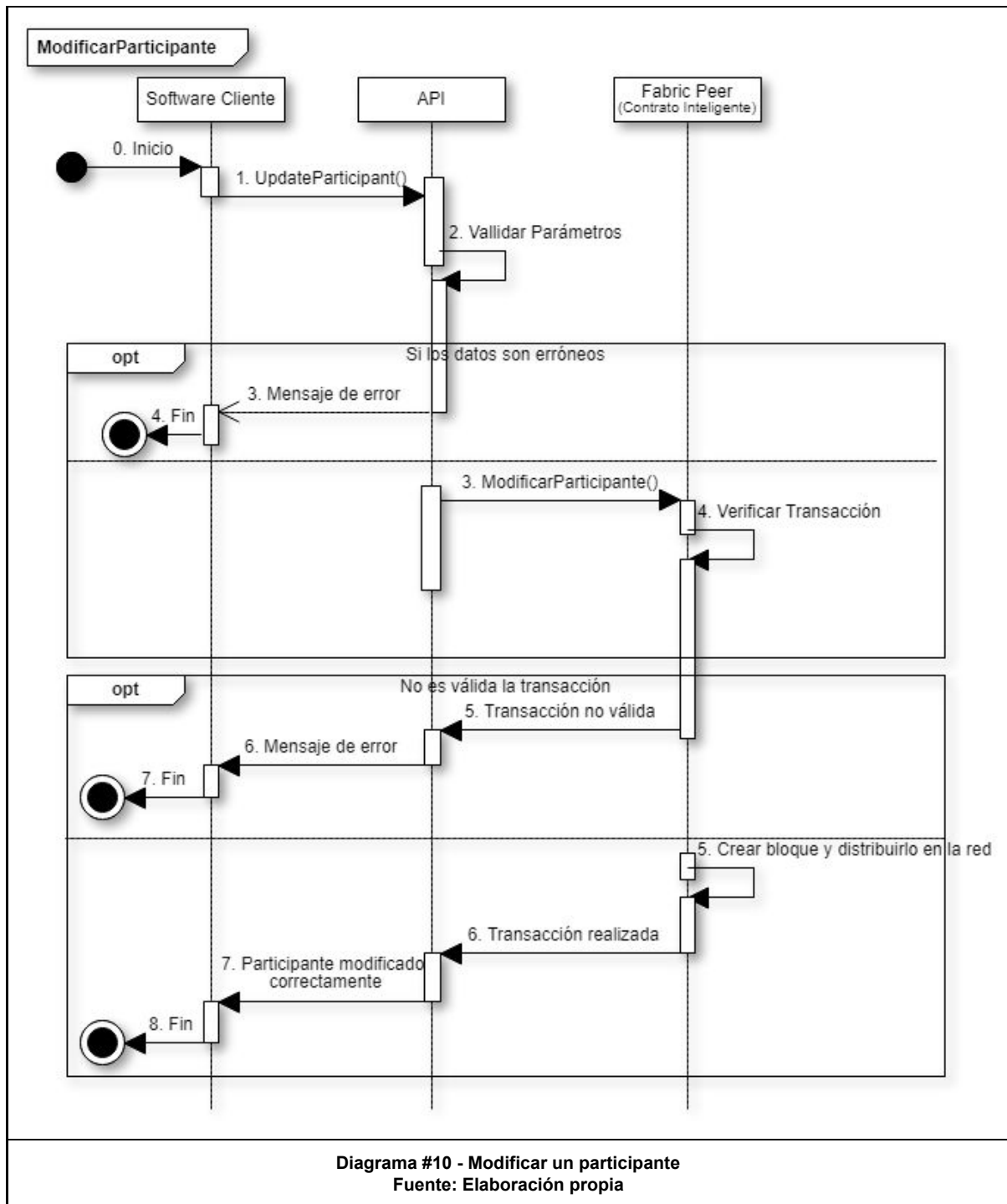
En el diagrama #14 se encuentra visualizada la operación de modificar perfil público/personal.

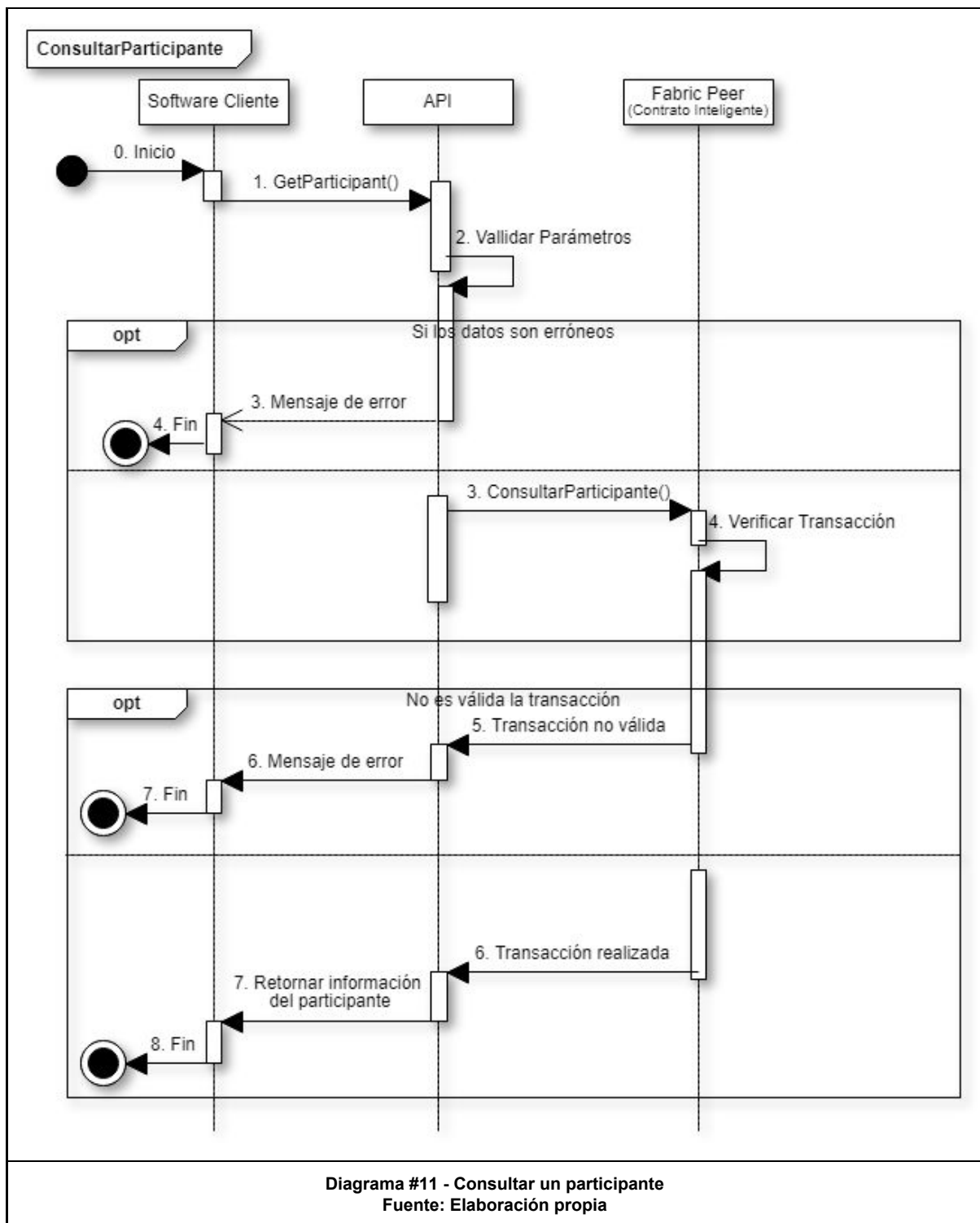
En el diagrama #15 se encuentra visualizada la operación de consultar el perfil público/personal.

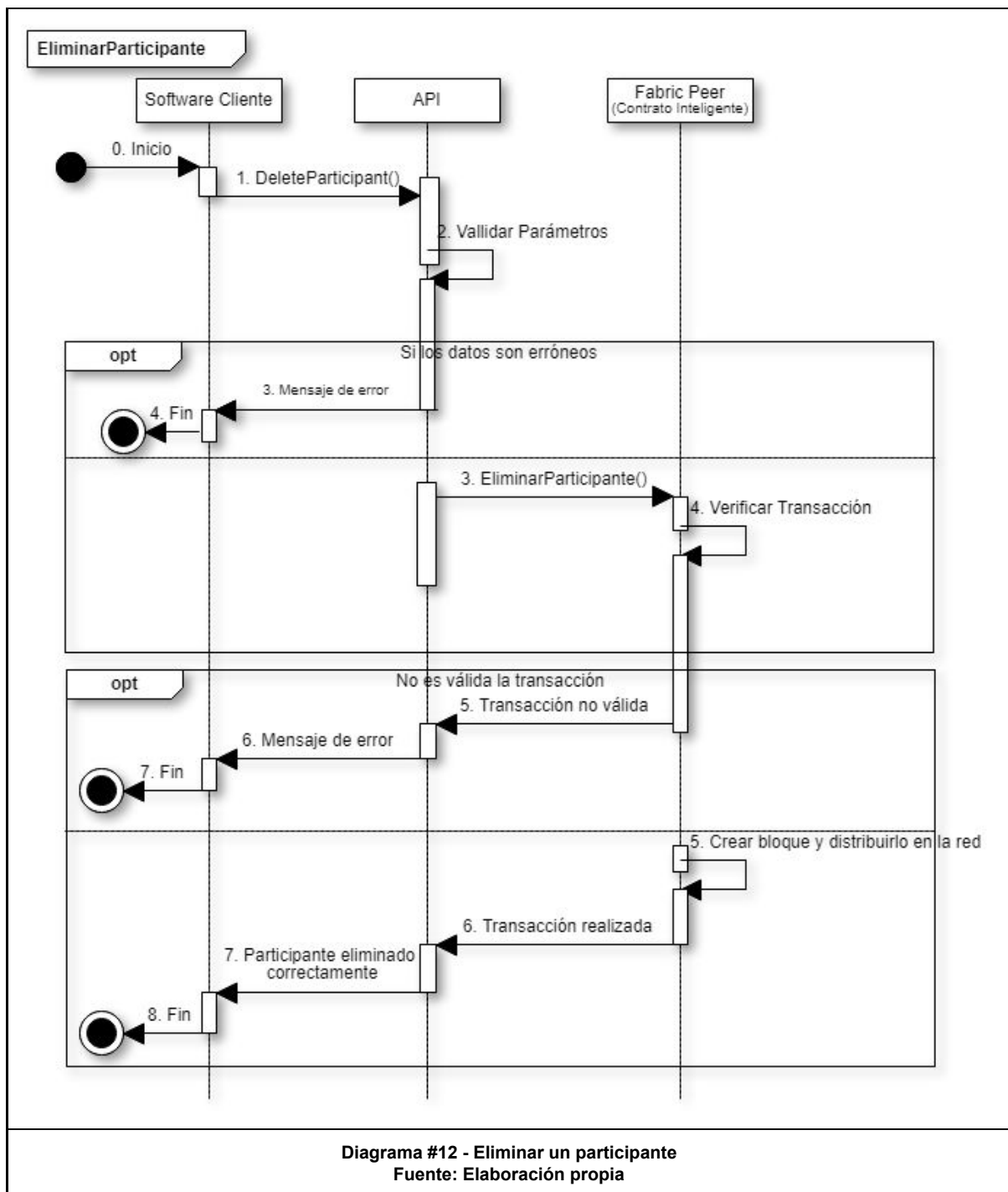
En el diagrama #16 se encuentra visualizada la operación de eliminar un perfil público/personal.

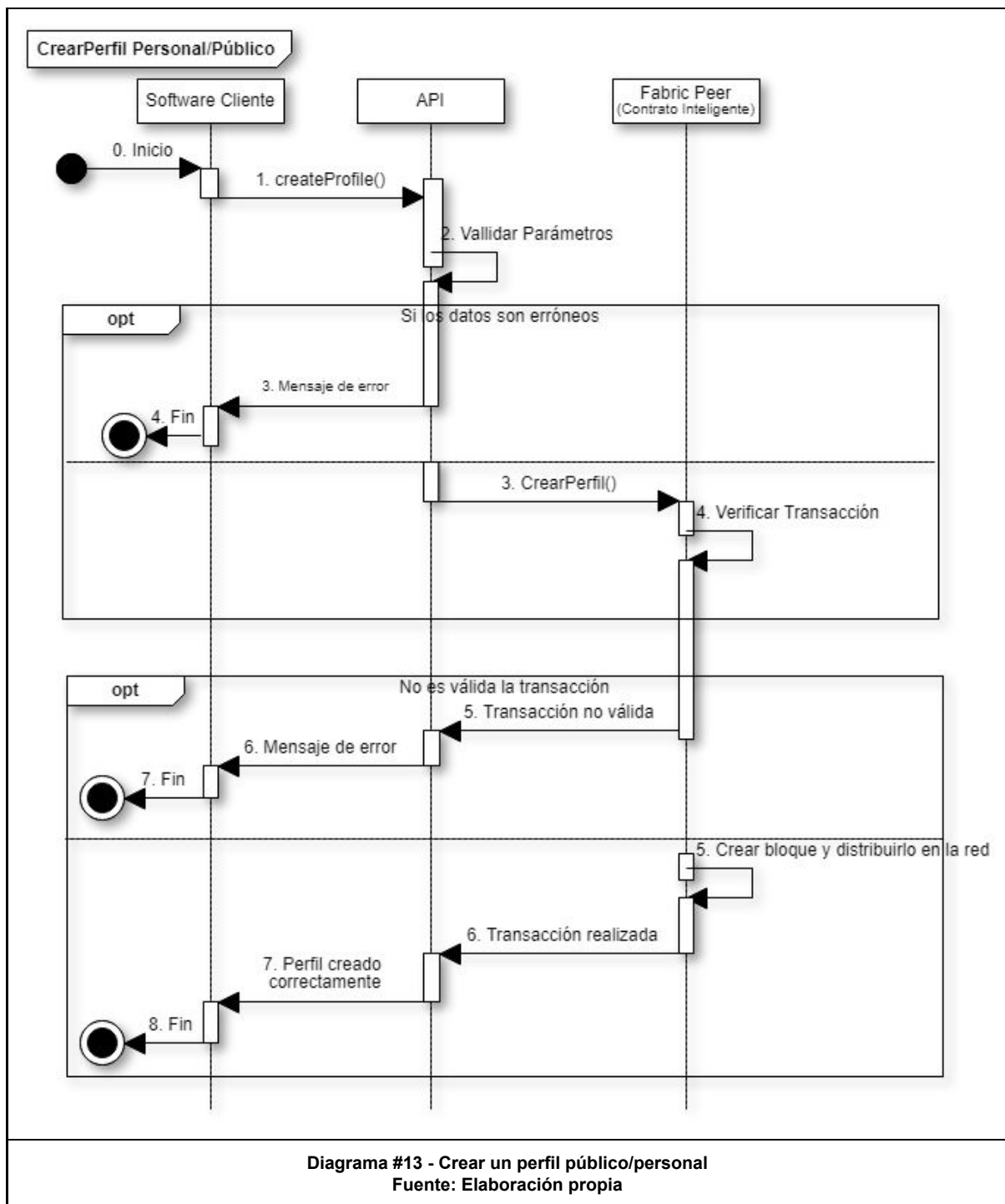
- **Diagrama de secuencia para consultar el estado de la red**

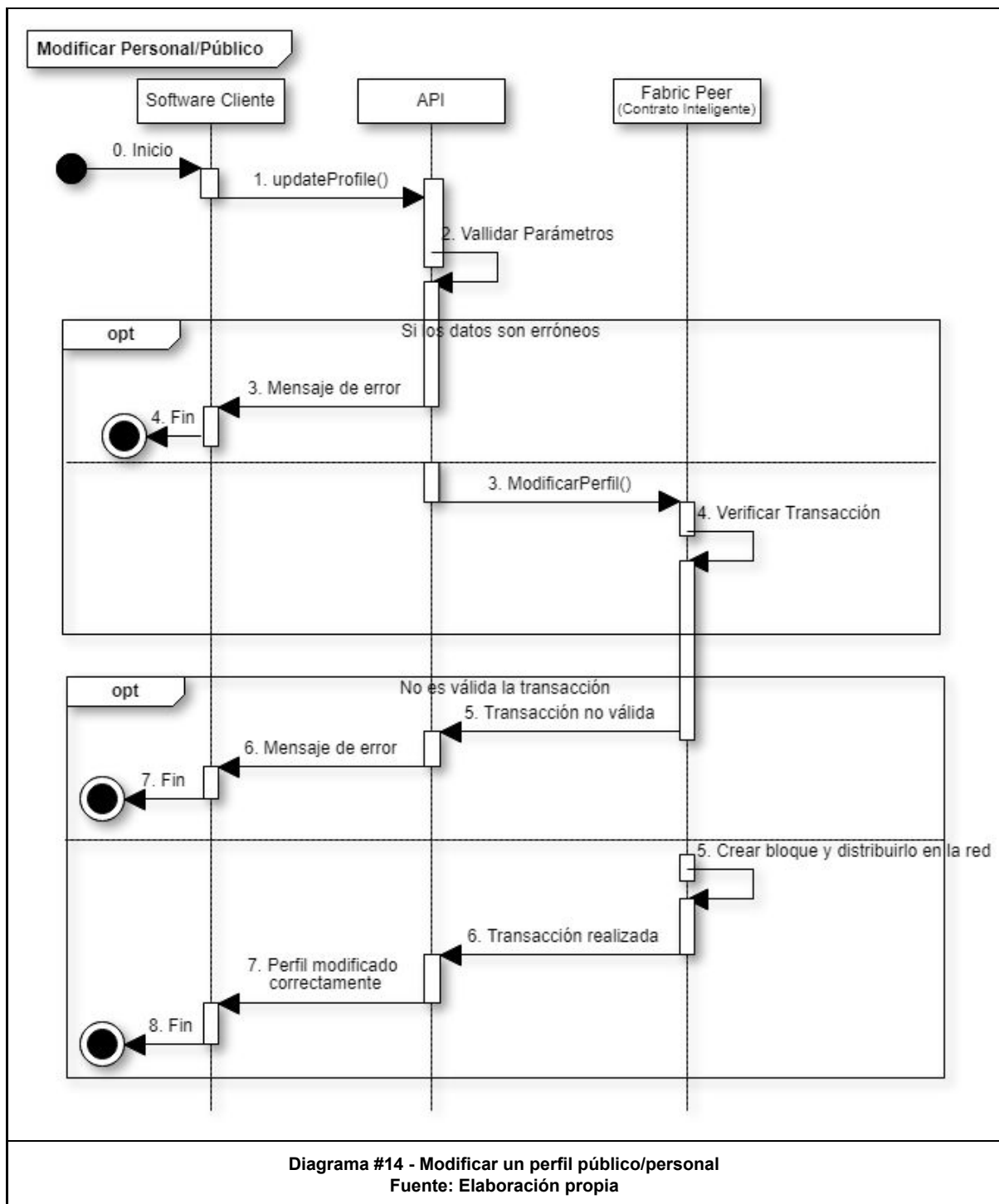
En el diagrama #17 se encuentra visualizada la operación “ping”, con ella se puede consultar el estado de la red. Esta operación es de solo lectura, por lo tanto no modifica el estado de la cadena de bloques.

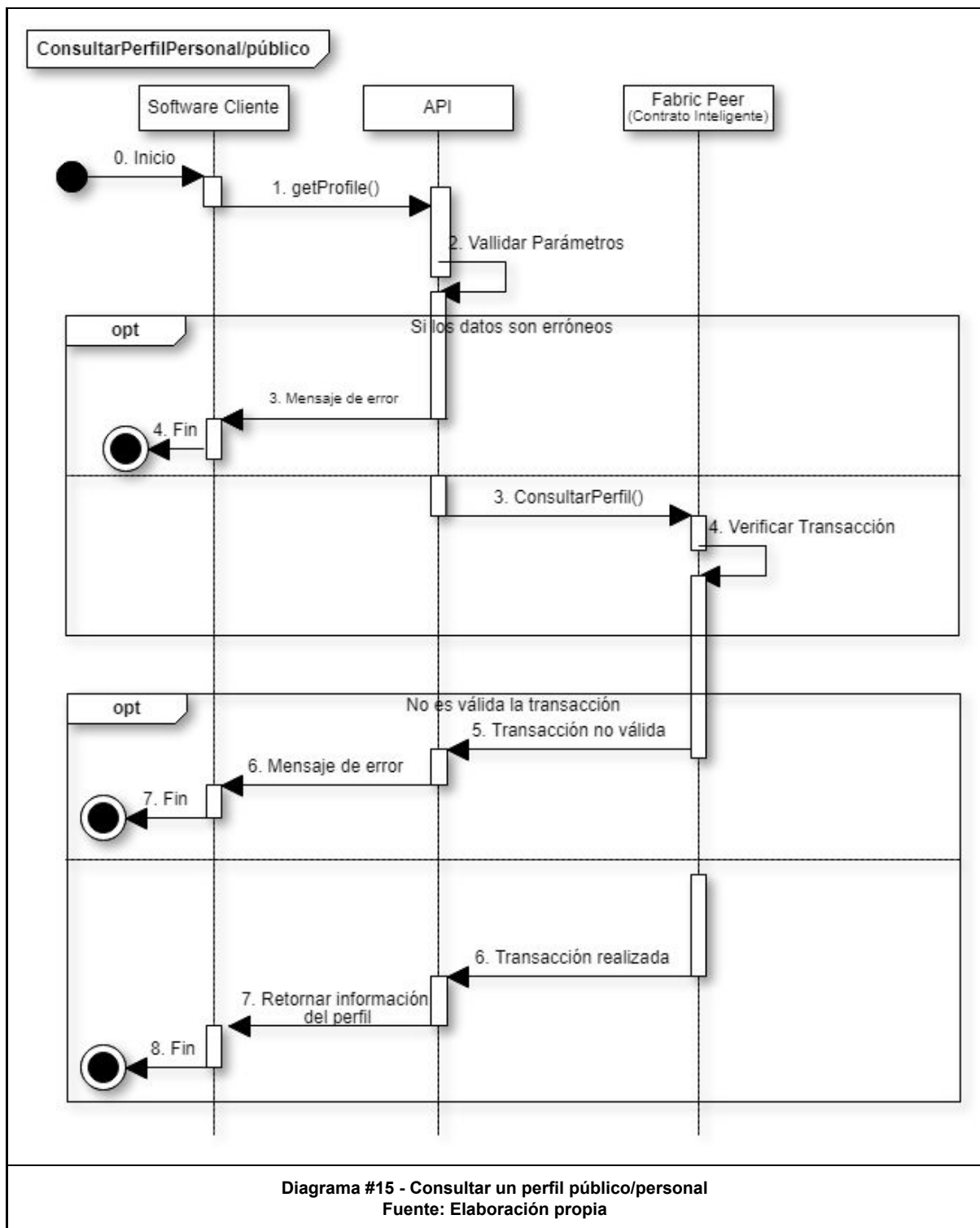


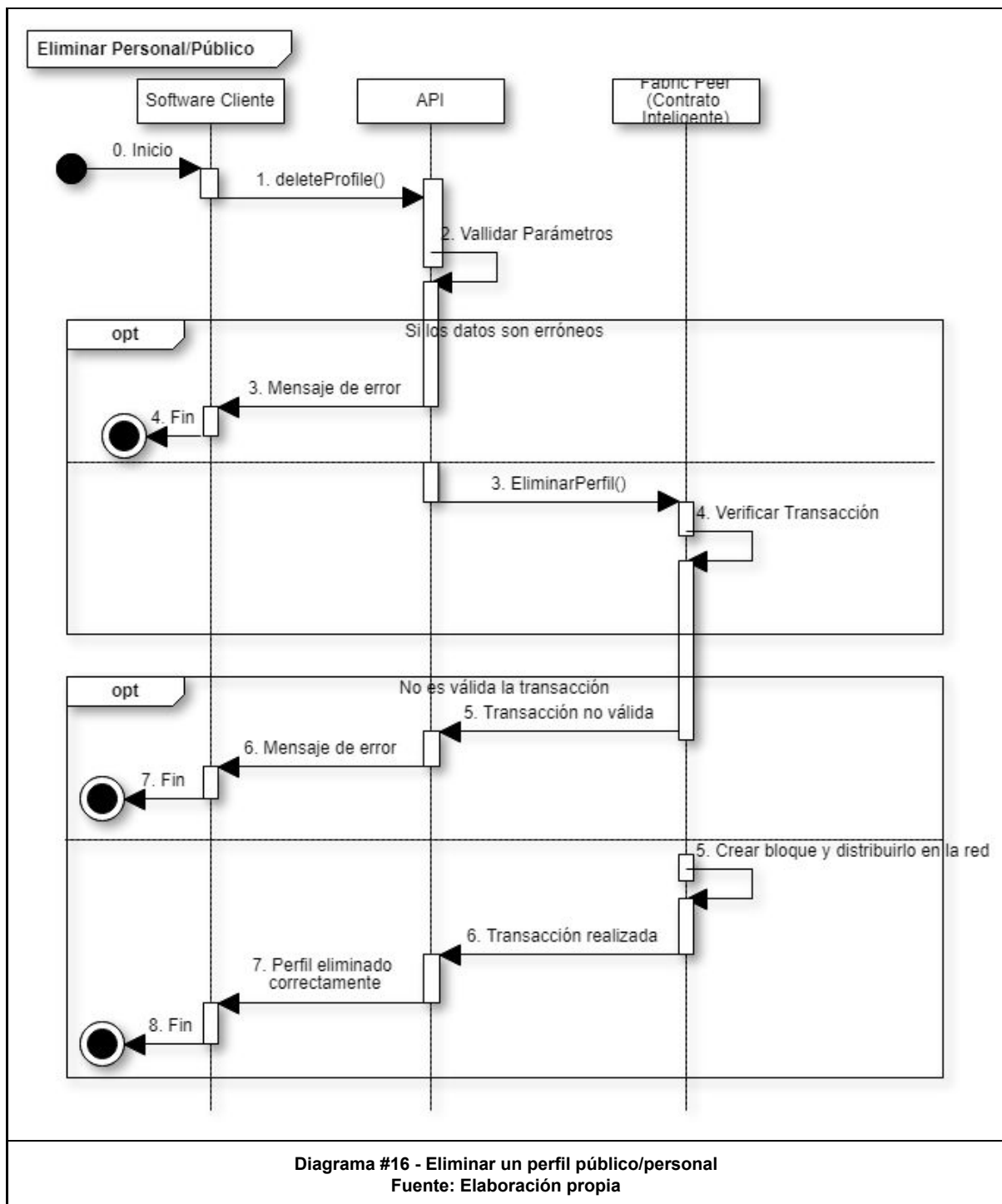


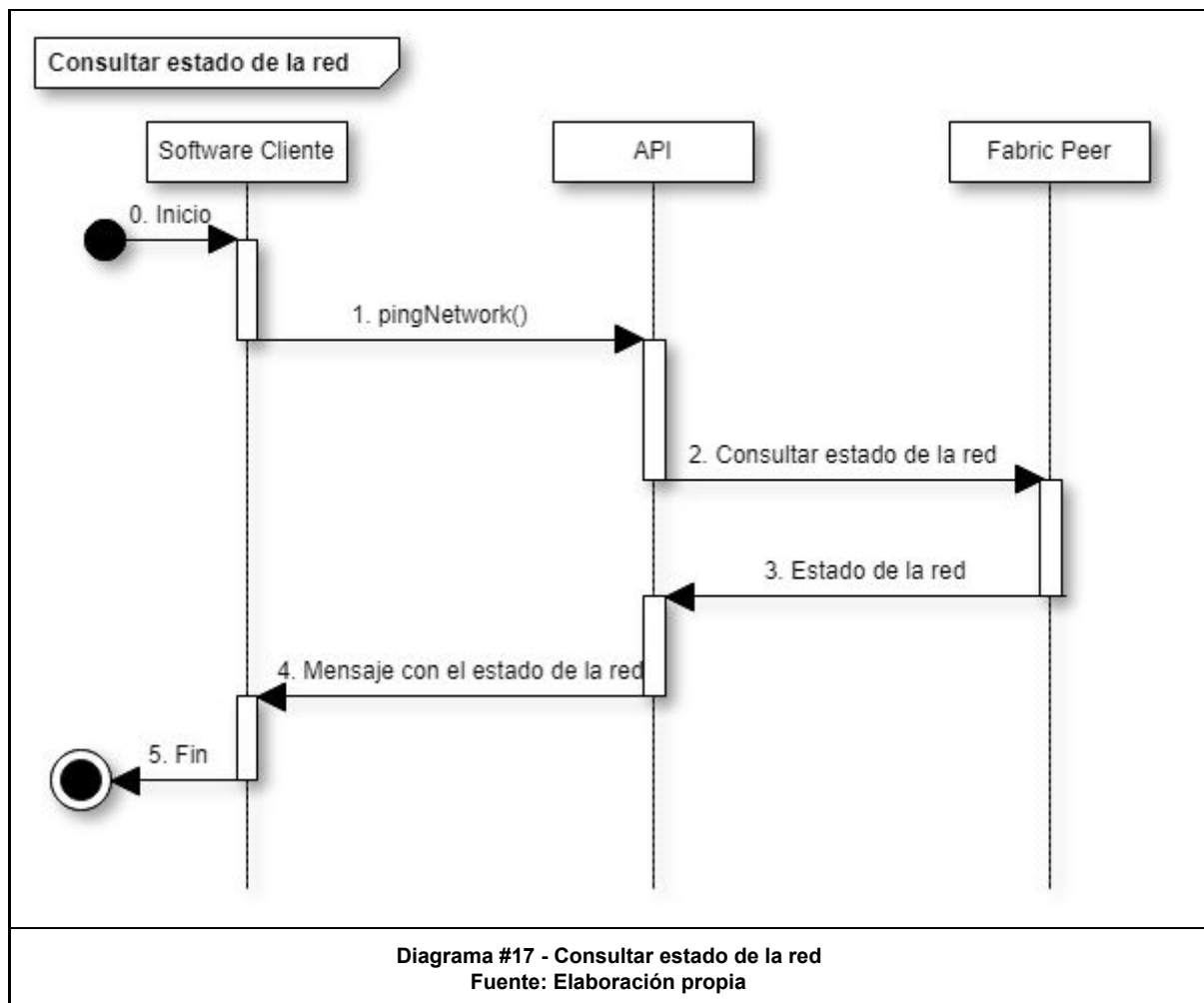








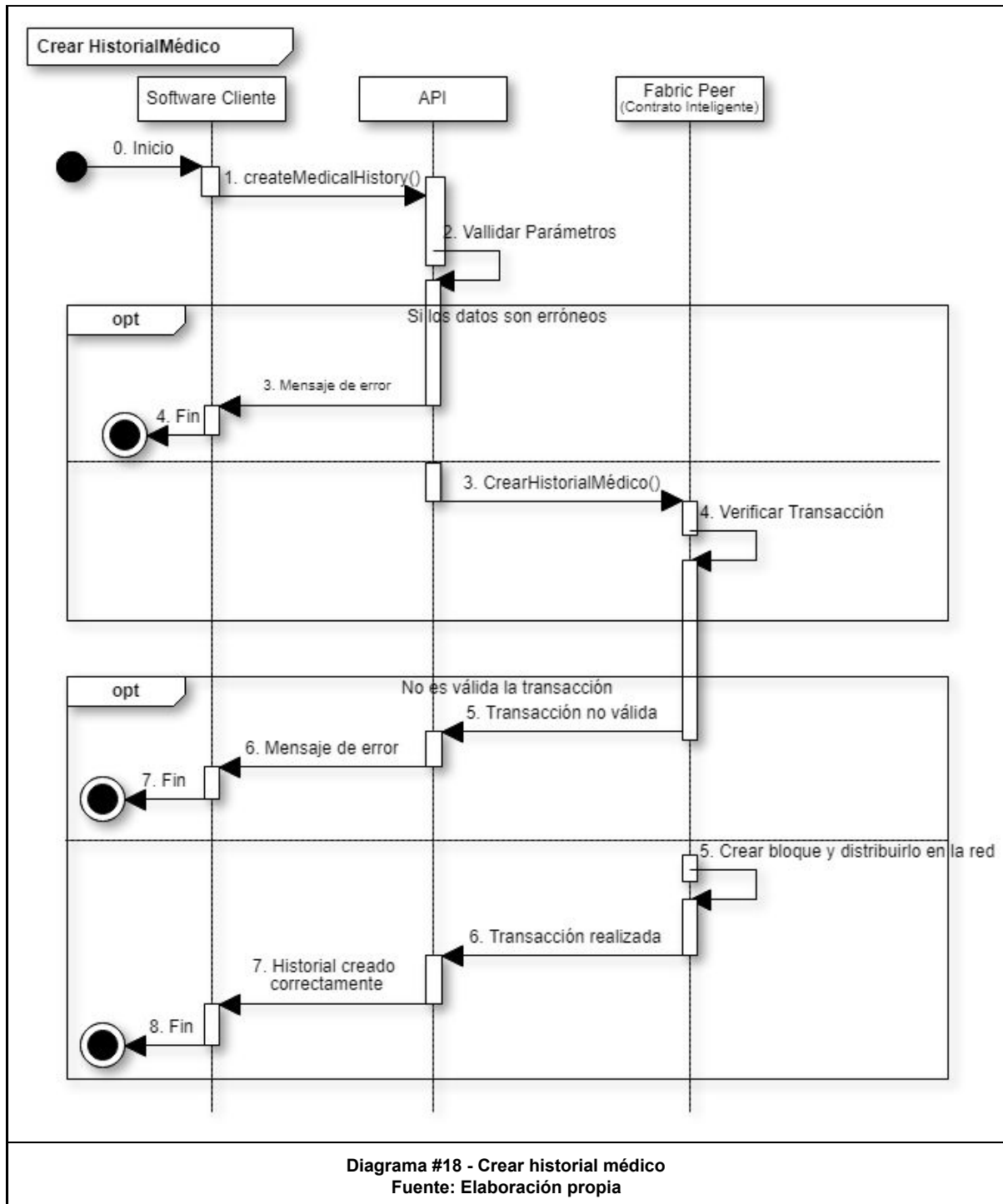


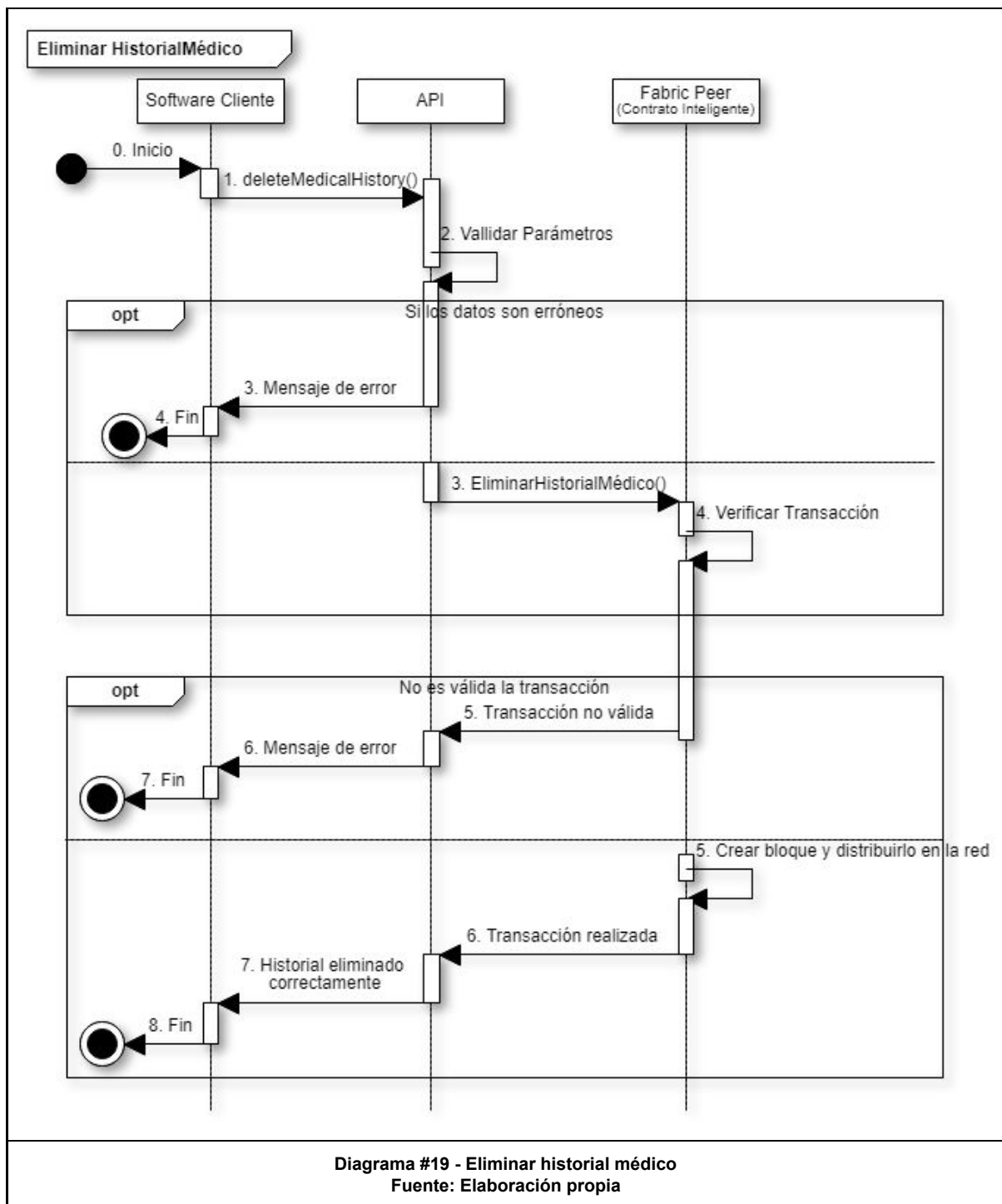


- **Diagrama de secuencia para operaciones del historial médico**

En el diagrama #18 se encuentra visualizado el proceso para crear un historial médico. Durante la creación de este activo no se interactúa con la red de IPFS, pues la operación inicial es solo definir los metadatos del activo (Historial Médico) para que luego pueda subirse la información médica.

En el diagrama #19 se encuentra visualizado el proceso para eliminar un historial médico. Dentro del proceso puede fijarse que no interactúa con la red de IPFS, es decir solamente se elimina el activo dentro de la cadena de bloques. Esto se debe a que IPFS no puede eliminar archivos que se encuentren distribuidos en la red, sino que solo elimina en el nodo local.





ANEXO B - CONFIGURACIÓN DEL SERVICIO

Dentro de este anexo se encontrarán las instrucciones para la instalación y configuración de la API. Pero antes de llegar a dichos procesos, es necesario la instalación de varias herramientas y un ambiente de pruebas para el testeo de la misma. Tanto en la parte de desarrollo como en la de pruebas, se utilizó el sistema operativo Ubuntu Xenial 16.04 LTS de 64 bits.

B1. Instalar InterPlanetary Filesystem

En esta sección se mostrarán los pasos para instalar y configurar la aplicación IPFS.

1. Descargar IPFS desde el siguiente link: <https://dist.ipfs.io/#go-ipfs> , para el desarrollo de la aplicación se utilizó la versión **v0.4.17**
2. Descomprimir el archivo descargado y ejecutar el script para instalar la aplicación.

```
$ tar xvfz go-ipfs.tar.gz
```

```
$ cd go-ipfs
```

```
$ ./install.sh
```

3. Instalar la aplicación git, para clonar repositorios de la página github

```
$ sudo apt-get install git -y
```

4. Clonar el siguiente repositorio, este repositorio contiene el código fuente en el lenguaje GO de una aplicación que genera claves de 32 bytes que se utilizará para configurar la red privada de IPFS.

```
$ git clone https://github.com/Kubuxu/go-ipfs-swarm-key-gen
```

5. Instalar el lenguaje de programación GO (Versión 1.8.7). GO se puede descargar del siguiente link: <https://golang.org/dl/>

6. Descomprimir e instalar

```
$ tar -C /usr/local -xzf go1.8.7.linux-amd64.tar.gz
```

```
$ export PATH=$PATH:$PWD/go.1.8.7.linux-amd64/go/bin
```

7. Compilar ejecutar el código anteriormente clonado

```
$ go ipfs-swarm-key-gen/main.go > ~/.ipfs/swarm.key
```

8. Avisar a IPFS que se va a ejecutar dentro de una red privada

```
$ export LIBP2P_FORCE_PNET=1
```

9. Eliminar los bootstrap node del servidor de ipfs, para ejecutar este comando, el demonio de ipfs tiene que estar ejecutándose. Los bootstrap node son las conexiones que utiliza IPFS para comunicarse con la red (Si no hay ninguno, el nodo local de IPFS estará desconectada de la red).

```
$ ipfs init
```

```
$ ipfs daemon
```

```
$ ipfs bootstrap rm --all
```

10. Ahora hay que agregar los nodos de nuestra red privada. Esto se realiza de la siguiente manera:

```
$ ipfs bootstrap add <multiaddr>
```

Multiaddr es la dirección de un nodo IPFS y se estructura de la siguiente manera:

```
/verProtocolo/IP/protocolo/puerto/ipfs/nodeID
```

Un ejemplo es el siguiente:

```
/ip4/104.236.76.40/tcp/4001/ipfs/QmSoLV4Bbm51jM9C4gDYZQ9Cy3U6aXM  
JDAbzgu2fzaDs64
```

11. Ahora, para que la red funcione correctamente, los nodos tienen que compartir la misma llave generada en el paso 7, tener la misma variable de entorno asignado en el paso 8, y tener la dirección de los nodos que compondrán la red privada.

B2. Instalar Hyperledger Fabric

En esta sección se muestran las instrucciones para instalar y configurar un ambiente de desarrollo de Hyperledger Fabric.

1. Para instalar Hyperledger Fabric, primero hay que instalar sus dependencias, la lista es la siguiente:
 - a. Docker Engine: Versión 17.03 o una más reciente
 - b. Docker-composer: Versión 1.8 o una más reciente
 - c. Nodejs 8.9.x
 - d. npm 5.x
 - e. git 2.9 (Instalado en la sección B1 de este mismo capítulo)
 - f. Python 2.7.x (Por defecto, ya viene instalado en el sistema operativo)
2. Desinstalar versiones anteriores de docker

```
$ sudo apt-get remove docker docker-engine docker.io
```

3. Descargar docker del siguiente link: <https://download.docker.com/linux/ubuntu/dists/xenial/pool/stable/amd64/> , la versión utilizada para la API fue la siguiente: docker-ce_17.03.3~ce-0~ubuntu-xenial_amd64

4. Instalar docker

```
$ sudo dpkg -i docker-ce_17.03.3~ce-0~ubuntu-xenial_amd64.deb
```

5. Descargar e instalar docker-composer

```
$ sudo curl -L  
"https://github.com/docker/compose/releases/download/1.22.0/docker-  
compose-$(uname -s)-$(uname -m)" -o  
/usr/local/bin/docker-compose
```

6. Otorgar permisos a la aplicación de docker composer.

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

7. Descargar e instalar NVM (*Nodejs version manager*). Con esta aplicación se puede descargar y configurar fácilmente las aplicaciones de Nodejs y npm

```
$ curl -o-  
https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh |  
bash
```

8. Modificar el script .profile que se encuentra en la dirección \$HOME y agregarle las siguientes líneas:

```
export NVM_DIR="${XDG_CONFIG_HOME:-$HOME/.}nvm"  
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

9. Para finalizar la instalación de nvm, se ejecuta el siguiente comando:

```
$ command -v nvm
```

10. Instalar Nodejs y npm con NVM.

```
$ nvm install 8.9.4
```

```
$ nvm use 8.9.4
```

11. Instalar Hyperledger Fabric. Creamos una carpeta y cambiamos de directorio a la carpeta creada.

```
$ mkdir ~/fabric-dev-servers && cd ~/fabric-dev-servers
```

12. Luego, estando dentro de la carpeta creada en el paso anterior, ejecutamos los siguientes comandos para descargar el ambiente de desarrollo para Hyperledger Fabric.

```
$ curl -O
https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz
$ tar -xvf fabric-dev-servers.tar.gz

$ export FABRIC_VERSION=hlfv11
$ ./downloadFabric.sh
```

B3. Instalar Hyperledger Composer

1. Instalar Hyperledger Composer. Los siguientes comandos son para descargar e instalar todas las herramientas de composer.

```
$ npm install -g composer-cli@0.19.14-20180809130819

$ npm install -g composer-playground@0.19.14-20180809130819
```

2. Dirigirse a la carpeta que creamos en la sección anterior B2, estando dentro de ella ejecutamos los siguientes comandos:

```
$ cd ~/fabric-dev-servers
$ export FABRIC_VERSION=hlfv11
$ ./startFabric.sh
$ ./createPeerAdminCard.sh
```

Los pasos anteriormente ejecutados, levantan un ambiente de desarrollo Hyperledger Fabric, crea e importa la tarjeta de identificación del administrador de la red.

B3. Instalar la API

En esta sección se va a configurar la API para que pueda funcionar correctamente.

1. Antes que nada, se tiene que instanciar el contrato inteligente, dentro del directorio de la API, se encontrará otro directorio llamado medical-chain, dentro se encontrará el contrato inteligente llamado basic-sample-network.bna e importar la tarjeta de identificación del administrador de la red que se crea al instanciar el contrato inteligente.

```
$ composer network install --archiveFile basic-sample-network.bna --card  
PeerAdmin@fabric-network
```

```
$ composer network start --networkName basic-sample-network  
--networkVersion 0.6.1-deploy.1 --networkAdmin admin  
--networkAdminEnrollSecret adminpw --card PeerAdmin@fabric-network --file  
businessAdmin.card
```

```
$ composer card import -c businessAdmin -f businessAdmin.card
```

2. Ahora hay que instalar las dependencias de la API. Para esto, hay que dirigirse al directorio raíz de la misma y ejecutar el siguiente comando:

```
$ npm install  
$ mkdir cards  
$ mkdir identityFiles  
$ mkdir lib/tmp
```

3. Si todo salió correctamente, ahora hay que configurar la API. Para esto, abrimos con nuestro editor de preferencia el archivo llamado appconfig.json, este archivo de configuración se encuentra en la carpeta raíz de la API. Los valores de dicho archivo JSON, son los siguientes:
 - a. host: Dirección IPv4 del servidor, por defecto es localhost
 - b. port: Puerto donde va a escuchar el servidor, por defecto es 3333
 - c. uploadDir: Directorio donde se van a guardar los archivos que se suban al servidor, debería dejarse como está
 - d. ipfs.host: Dirección IPv4 de la API de ipfs, si no se ha modificado, por defecto es localhost
 - e. ipfs.port: Puerto de la API, por defecto es 5001
 - f. ipfs.protocol: Protocolo que utiliza la API de ipfs para la transferencia de archivos, por defecto es http.
 - g. adminCard: Tarjeta de identificación del administrador de la red, este usuario se creo en el paso 1 de esta sección, y se puede exportar su tarjeta de la siguiente manera.

```
$ composer card export -c businessAdmin -f businessAdmin.card
```

- h. profile: Archivo json que contiene la información de la red de Hyperledger Fabric, para conseguir este archivo hay varias maneras, la primera es descomprimiendo cualquier tarjeta de identificación, dentro de ella se puede encontrar. Y la segunda, es dentro de la carpeta que se creó en la sección anterior (B2) ~/fabric-dev-servers

- 4. Ahora, lo que queda es ejecutar la API.

```
$ npm start
```

- 5. De ser necesario, puede ejecutarse las pruebas unitarias para verificar que la API se instaló y se esté ejecutando correctamente.

```
$ npm test
```

CONCLUSIONES

El objetivo general planteado en el desarrollo de este proyecto es la creación de una aplicación que sirva como intermediario entre un contrato inteligente instanciado en la blockchain de Hyperledger Fabric y una red privada en IPFS para el manejo de historiales médicos permitidos.

Para el análisis de los requerimientos, se definieron las funcionalidades que debía tener el sistema, como tendría que estar diseñado la aplicación y el contrato inteligente para que este funcione y pueda limitar los accesos a los usuarios. Gracias al InterPlanetary Filesystem, que se utilizó para el almacenamiento y transferencia de los historiales médicos, se logró que historiales médicos de tamaño considerable puedan ser compartidos entre distintas entidades médicas reduciendo el ancho de banda y el espacio de almacenamiento. Con Hyperledger Fabric y su herramienta Hyperledger Composer, se logró otorgar permisos de acceso a los historiales médicos gracias a la funcionalidad de permisología que ofrecen, permitiendo que un profesional acceda al historial médico de un paciente si este le ha otorgado el acceso al mismo. Gracias a esto, se desarrolló una interfaz de programación de aplicaciones, en donde el software de una entidad médica pudiese conectarse a la red de IPFS y al contrato inteligente instanciado en una red de Hyperledger Fabric para el almacenamiento de historiales médicos y el acceso permitido a los mismos. Al final, se realizaron las pruebas funcionales y unitarias de la API, dando resultados positivos.

Por último es importante destacar la utilización de la blockchain para la ejecución de contratos inteligentes, ya que este es un método que demuestra la posibilidad de garantizar la transparencia y la inmutabilidad de los datos (sin antes mostrar registros sobre sus cambios), otorgando seguridad y confianza entre partes de forma descentralizada. Por otro lado, aunque el uso de Hyperledger Fabric es muy complicado por la escasez de documentación por parte de la comunidad, como lo son los tutoriales, es importante motivar la investigación de esta tecnología por los beneficios que puede traer a futuro, llegando definitivamente a reemplazar los sistemas transaccionales centralizados.

TRABAJO A FUTURO

La aplicación logra restricciones de acceso a los historiales médicos distribuidos, pero existen más métodos posibles de los que se hablaron en este documento, como por ejemplo:

- La posibilidad de que un paciente pueda compartir su historial médico a otro paciente.
- La posibilidad de que un profesional tratante pueda compartir el historial médico de su paciente con otro profesional que deba intervenir en un tratamiento.
- La posibilidad de que el paciente pueda otorgar accesos a otro usuario en caso de que éste se encuentre inhabilitado, por ejemplo, que el paciente esté en estado de coma a mitad de un tratamiento y alguien tenga que tener los accesos del historial médico.
- Que el paciente pueda descargar su historial médico anonimizado, para que pueda compartirlo con otros usuarios.

Un siguiente paso sería la extensión de la API con las opciones anteriormente mencionadas. Una segunda recomendación, sería utilizar procesamiento a gran escala sobre los registros en IPFS, ya que este último se volvería en un gran centro de almacenamiento de datos, y el procesamiento de los mismos puede generar bastantes beneficios para las investigaciones, como por ejemplo estudiar los crecimientos de tumores con el tiempo, predecir ciertas enfermedades, entre otros, utilizando los datos almacenados de los pacientes.

Por último, una tercera recomendación sería agregar información de los tratamientos de los pacientes, como las recetas médicas, terapias, etc.

BIBLIOGRAFÍA

Artículo 3 Ley 41/2002. (2018). Recuperado de:

<https://www.boe.es/buscar/act.php?id=BOE-A-2002-22188>

Estrada, M. (2017). *Conservación preventiva en Archivos y Bibliotecas primera parte*. Recuperado de:

<http://www.infotecarios.com/conservacion-preventiva-archivos-bibliotecas-primera-parte/#.W9CMqUtKgdV>

Gutiérrez, J. & Núñez, M. & Aguirre, H. & Delgado, R. (2014). *Implementación de la seguridad en el manejo de las imágenes médicas*. Recuperado de: <http://www.medigraphic.com/pdfs/invdiss/ir-2014/ir144d.pdf>

Croman, K. & Decker, C. & Eyal, I. & Gencer, A. E. & Juels, A. & Kosba, A. & Miller, A. & Saxena, P. & Shi, E. & Gün Sirer, E. & Song, D. & Wattenhofer, R. (2016). On Scaling Decentralized Blockchains (A Position Paper).

Benet, J. (2017). *IPFS - content addressed, versioned, P2P file system (whitepaper)* (3rd ed.)

Balakrishnan, Hari & M Kaashoek, Frans & Karger, David & Morris, Robert & Stoica, Ion. (2003). Looking up data in P2P systems. Communications of the ACM. 10.1145/606272.606299.

Champagne, Phil. El Libro de Satoshi. (2014). (Edición BlockchainEspana.com): Descargable desde: <http://www.libroblockchain.com/satoshi/> (pp. 11)

Cohen, Bram. (2003). Incentives build robustness in BitTorrent. Workshop on Economics of Peer to Peer systems. 6.

Mazieres, David & Kaashoek, M. Frans. (2000). *Self-certifying file system*. Massachusetts Institute of Technology.

Buterin, V. (2013) Ethereum: a next generation smart contract and decentralized application platform.

Stallings, William. (2005). *Cryptography and network security - principles and practice* (3. ed.). (pp. 256)

Nakamoto, Satoshi. (2009). *Bitcoin: A Peer-to-Peer Electronic Cash System*.

Ray, James. (2015). *Proof of Stake FAQs*. Recuperado de: <https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs>

Tar, Andrew. (2017). *Smart Contracts, Explained*. Recuperado de: <https://cointelegraph.com/explained/smart-contracts-explained>

King, Sunny & Nadal, Scott. (2012). *PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake*.

Narayanan, Arvind & Bonneau, Joseph & Felten, Edward & Miller, Andrew & Goldfeder, Steven. (2016). *Bitcoin and Cryptocurrency Technologies*. (pp. 190-212)

What is Hyperledger Fabric?. (2017). Recuperado de: <http://hyperledger-fabric.readthedocs.io/en/release-1.0/blockchain.html#what-is-hyperledger-fabric>

Androulaki, Elli & Barger, Artem & Bortnikov, Vita & Cachin, Christian & Christidis, Konstantinos & De Caro, Angelo & Enyeart, David & Ferris, Christopher & Laventman, Gennady & Manevich, Yacov & Muralidharan, Srinivasan & Murthy, Chet & Nguyen, Binh & Sethi, Manish & Singh, Gari & Smith, Keith & Sorniotti, Alessandro & Stathakopoulou, Chrysoula & Vukolic, Marko & Yellick, Jason. (2018). *Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains*.

Hyperledger Fabric FAQ. (2018). Recuperado de: <http://hyperledger-fabric.readthedocs.io/en/release-1.1/Fabric-FAQ.html#security-access-control>

Architecture Explained. (2018). Recuperado de: <http://hyperledger-fabric.readthedocs.io/en/release-1.1/arch-deep-dive.html#system-architecture>

Hyperledger Architecture, Volume 1. (2017). Recuperado de: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf

Hyperledger Explorer. (2018). Recuperado de: <https://www.hyperledger.org/projects/explorer>

Welcome to Hyperledger Composer (2018). Recuperado de: <https://hyperledger.github.io/composer/v0.19/introduction/introduction.html>

Hyperledger Composer Modeling Language. (2018). Recuperado de: https://hyperledger.github.io/composer/v0.19/reference/cto_language

Hyperledger Composer Access Control Language. (2018). Recuperado de: https://hyperledger.github.io/composer/v0.19/reference/acl_language

Hyperledger Composer Query Language. (2018) Recuperado de: <https://hyperledger.github.io/composer/v0.19/reference/query-language>

Mouzakitis, E. (2016). *Monitoring Kafka performance metrics.* Recuperado de: <https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/>

Richardson, Leonard & Ruby, Sam. (2007). *RESTful Web Services.* 978-0-596-52926-0. (pp. 15-16)

Gaceta Oficial N° 39.823. Ley de ejercicio de la medicina. República Bolivariana de Venezuela. 19 de diciembre de 2011

Health Information Privacy. (2017). Recuperado de: <https://www.hhs.gov/hipaa/index.html>

Sommerville, I. (2011). *Software engineering* (9th edition).

Patel, P. (2018). *What exactly is Node.js?*. Recuperado de:
<https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5>

Farmer, K. (2017). *Student Blog: What is Postman, and why use it?*. Recuperado de:
<https://www.digitalcrafts.com/blog/student-blog-what-postman-and-why-use-it>