

UNIVERSIDAD NACIONAL DE CÓRDOBA



---

PROYECTO INTEGRADOR  
INGENIERÍA EN COMPUTACIÓN

---

# Prototipo de un sistema de validación de información académica en Blockchain

---

*Autor:* Elisabeth LEONHARDT  
*Matrícula:* 135108916

*Director:*  
Ing. Gustavo WOLFMANN  
*Codirector:*  
Ing. Matias CUENCA DEL REY

Facultad de Ciencias Exáctas, Físicas y Naturales

14 de octubre de 2019

## Resumen

Con el avance de la tecnología y crecimiento exponencial de Internet, cada vez más servicios se ofrecen de forma online, contagiando con esa tendencia también procesos administrativos del sector público. El sistema de gestión académica de la UNC, Guaraní, por ejemplo, posibilita entre otras funcionalidades la administración de notas. Pero en el momento actual, seguridad e integridad de la información no puede ser asegurada a través del Guaraní, lo cual obliga a la Universidad mantener todas sus actas de exámenes en papel. Certificaciones como historias académicas tienen que ser verificados a mano contra las actas originales, lo cual implica mayor cantidad de personal y mayor demora comparando con una solución digitalizada.

Un Blockchain es una base de datos distribuida de solo escritura y lectura. Una serie de mecanismos criptográficos aseguran que la información almacenada no puede ser modificada una vez que se insertó al sistema y un algoritmo de consenso se encarga de mantener la consistencia de datos, de modo que la misma información es replicada en todos los nodos. Las propiedades descritas llevan a que un Blockchain también es llamado “una única fuente de la verdad”.

El presente proyecto integrador, en primer lugar, investiga la tecnología Blockchain e implementaciones existentes de la misma. Luego evalúa cada una de ellas en base a una serie de requerimientos que se plantearon para el caso de uso de la universidad. A continuación, se elige una de las plataformas analizadas para la implementación y prueba de un prototipo. Por último, se presentan una evaluación y distintas ideas para mejorar el prototipo realizado.

# Índice general

<b>Contenido</b>	<b>II</b>
<b>Lista de Figuras</b>	<b>V</b>
<b>Lista de Tablas</b>	<b>VII</b>
<b>1. Introducción</b>	<b>1</b>
1.1. Introducción . . . . .	1
1.2. Motivación . . . . .	2
1.3. Marco institucional . . . . .	2
1.4. Objetivos . . . . .	2
1.4.1. Objetivos particulares . . . . .	3
1.5. Estructura del texto . . . . .	3
<b>2. Investigación de plataformas Blockchain</b>	<b>5</b>
2.1. Bitcoin . . . . .	5
2.1.1. Transacciones y scripts . . . . .	5
2.1.2. La red <i>peer to peer</i> . . . . .	6
2.1.3. Consenso y minado . . . . .	6
2.2. Primeras especificaciones para el prototipo . . . . .	7
2.3. Ethereum . . . . .	10
2.3.1. Funcionamiento . . . . .	10
2.4. Multichain . . . . .	11
2.4.1. Funcionamiento . . . . .	12
2.5. EOSio . . . . .	13
2.5.1. Funcionamiento . . . . .	13
2.6. IOTA . . . . .	15
2.6.1. Funcionamiento . . . . .	16
2.7. Lisk . . . . .	18
2.7.1. Funcionamiento . . . . .	18
2.8. Hyperledger . . . . .	20
2.8.1. Hyperledger Fabric . . . . .	20
2.8.2. Hyperledger Sawtooth . . . . .	21
2.8.3. Otros proyectos de Hyperledger . . . . .	23
2.9. Conclusiones . . . . .	23
<b>3. Hyperledger Fabric</b>	<b>25</b>
3.1. Composición de la red Blockchain . . . . .	25

3.2.	La existencia de varios <i>ledgers</i> . . . . .	26
3.3.	Aplicaciones y la red Blockchain . . . . .	27
3.4.	Canales . . . . .	29
3.5.	Agrupación de la red en organizaciones . . . . .	30
3.6.	La Identidad de los <i>peers</i> en la red Blockchain . . . . .	30
3.7.	Los <i>Ledgers</i> y el <i>World State</i> . . . . .	32
3.8.	El servicio para ordenar transacciones . . . . .	33
3.9.	Conclusión . . . . .	34
<b>4.</b>	<b>Análisis y Diseño del Sistema</b>	<b>35</b>
4.1.	Entrevista . . . . .	35
4.2.	Casos de uso de Blockchain . . . . .	37
4.3.	Funcionalidades . . . . .	39
4.4.	Definición de los Componentes . . . . .	40
4.5.	Definición de los Requerimientos . . . . .	43
4.6.	Definición del Comportamiento . . . . .	45
4.7.	Riesgos . . . . .	48
4.7.1.	Criterios . . . . .	48
4.7.2.	Identificación de Riesgos . . . . .	49
4.7.3.	Análisis de Riesgos . . . . .	50
4.7.4.	Planificación de Riesgos . . . . .	50
4.8.	Control de Versiones . . . . .	50
<b>5.</b>	<b>Implementación</b>	<b>54</b>
5.1.	Implementación Local . . . . .	54
5.1.1.	Configuración de Hyperledger Fabric . . . . .	55
5.1.2.	Automatización con un bash script . . . . .	63
5.1.3.	Implementación de la Aplicación con API . . . . .	63
5.1.3.1.	Desarrollo de la función <i>add_register</i> . . . . .	66
5.1.3.2.	Desarrollo de la función <i>query</i> . . . . .	68
5.1.3.3.	Desarrollo de la función <i>check_validity</i> . . . . .	69
5.1.3.4.	Desarrollo de la función <i>verify_history</i> . . . . .	70
5.1.3.5.	Desarrollo de la función <i>add_revoked</i> . . . . .	70
5.1.3.6.	Desarrollo de la función <i>query_rectified</i> . . . . .	74
5.1.4.	Implementación del Frontend con Angular e Ionic . . . . .	75
<b>6.</b>	<b>Pruebas y Validación</b>	<b>77</b>
6.1.	Validación de los requerimientos de usuario . . . . .	77
6.2.	Matriz de trazabilidad . . . . .	84
<b>7.</b>	<b>Conclusión</b>	<b>85</b>
7.1.	Problemas y Limitaciones . . . . .	86
7.2.	Trabajos futuros . . . . .	86
7.3.	Aporte personal . . . . .	87
<b>A.</b>	<b>Configuración del Ambiente de Desarrollo</b>	<b>88</b>

A.1. Instalación y Configuración del Entorno para Hyperledger Fabric . . . . .	88
A.2. Instalación y Configuración del Entorno para el Desarrollo de la API . . . .	91
A.3. Instalación y Configuración de las herramientas de frontend . . . . .	91

<b>Bibliografía</b>	<b>93</b>
---------------------	-----------

# Índice de figuras

2.1. Funcionamiento del Tangle de [1]. . . . .	17
2.2. Proyectos en Hyperledger de [2]. . . . .	20
3.1. Una red Blockchain con 3 <i>peers</i> : Cada <i>peer</i> corre una instancia idéntica del <i>chaincode</i> y del <i>ledger</i> . De [3]. . . . .	26
3.2. Un <i>peer</i> que aloja 2 <i>ledgers</i> y 3 diferentes <i>chaincodes</i> . De [3]. . . . .	26
3.3. Una red con 2 <i>ledgers</i> diferentes: <i>peers</i> P2 y P3 no saben de la existencia del <i>ledger</i> L2, sin embargo, todos los <i>peers</i> comparten información en el <i>ledger</i> L1. . . . .	27
3.4. Flujos de consulta y actualización del <i>ledger</i> . De [3]. . . . .	28
3.5. Visualización del concepto de canal. De [3]. . . . .	29
3.6. Una red Blockchain conformada por 4 organizaciones. . . . .	30
3.7. Una red compuesta de dos organizaciones, sus respectivas autoridades de certificación y sus MSP. Cada <i>peer</i> tiene una identidad digital asociada, <i>peer</i> P1 por ejemplo se identifica en el canal con la identidad D5. De [3]. . . . .	31
3.8. La estructura del Blockchain en Hyperledger Fabric. De [4]. . . . .	32
3.9. <i>Ledger</i> , <i>World State</i> y Blockchain en contexto. De [4]. . . . .	33
4.1. Diagrama de flujo para determinar un caso de uso Blockchain. De [5]. . . . .	38
4.2. Diagrama de la red Blockchain a implementar. . . . .	41
4.3. Diagrama general de la arquitectura junto con las tecnologías. . . . .	42
4.4. El rol de Ionic en el desarrollo <i>frontend</i> . De [6]. . . . .	43
4.5. Diagrama de casos de uso. . . . .	44
4.6. Diagrama de componentes. . . . .	45
4.7. Diagrama de secuencia para lecturas del <i>ledger</i> . . . . .	48
4.8. Diagrama de secuencia para escrituras del <i>ledger</i> . . . . .	48
5.1. Características técnicas de la computadora de desarrollo. . . . .	54
5.2. Directorios y archivos de la carpeta <i>crypto-config</i> . . . . .	57
5.3. Artefactos generados por <i>configtxgen</i> . . . . .	59
5.4. La red de Hyperledger Fabric desplegada con contenedores Docker . . . . .	60
5.5. Mensajes del log de un <i>peer</i> indicando que <i>ledger</i> y canal se crearon de forma exitosa. . . . .	61
5.6. Mensajes del log de peer0.fcq.pi.elisabeth.com indicando todos los <i>peers</i> conocidos. . . . .	61
5.7. Mensajes del log de peer0.famaf.pi.elisabeth.com indicando la instalación correcta del <i>chaincode</i> . . . . .	62
5.8. Respuesta de la llamada a la función <i>queryRegister</i> . . . . .	63
5.9. Prueba de la función GET con Postman . . . . .	65

5.10. Diagrama de actividad de la función <code>add_register</code> . . . . .	67
5.11. Subida exitosa de un acta a través de la API. . . . .	68
5.12. Consulta de un acta por Postman. . . . .	68
5.13. Diagrama de actividad para el caso de validación de una nota. . . . .	69
5.14. Validación exitosa de una nota. . . . .	69
5.15. Validación de una historia académica. . . . .	70
5.16. Diagrama de actividad para revocar un acta. . . . .	71
5.17. Creación de un acta rectificativa. . . . .	72
5.18. Consulta de un acta inválida. . . . .	73
5.20. Consulta por todas las actas rectificativas. . . . .	74
5.19. Validación de una historia académica involucrando un acta rectificativa. . . . .	74
5.21. <i>ion-grid</i> con los 6 casos de uso. . . . .	75
5.22. Un <i>ion-toast</i> que indica que la entrada de usuario no se encuentra en formato JSON. . . . .	76
5.23. Alertas de notificación sobre el éxito o fracaso de una operación. . . . .	76
6.1. Alertas de notificación sobre el éxito o fracaso de una operación. . . . .	78
6.2. Alertas de notificación sobre el éxito o fracaso de una operación. . . . .	79
6.3. Caso de una historia académica inválida. . . . .	80
6.4. Los contenedores que deben estar presentes para cumplir con los requeri- mientos RF-BC-01 y RF-BC-03. . . . .	82
6.5. Prueba de cambio de idioma en la página principal. . . . .	83
A.1. Características técnicas de la computadora de desarrollo. . . . .	88
A.2. Prueba de la instalación exitosa de Docker . . . . .	89
A.3. Prueba de la instalación exitosa de Docker Compose . . . . .	90
A.4. Prueba de la instalación exitosa Hyperledger Fabric y sus requerimientos . . . . .	91
A.5. Inicio exitoso de una aplicación web mínima . . . . .	92

# Índice de cuadros

4.1. Requerimientos de usuario. . . . .	44
4.2. Requerimientos funcionales de la red de Hyperledger Fabric. . . . .	45
4.3. Requerimientos funcionales de la API. . . . .	46
4.4. Requerimientos funcionales de la interfaz gráfica de usuario. ( <i>Graphical User Interface</i> ) . . . . .	47
4.5. Requerimientos no funcionales del sistema. . . . .	47
4.6. Probabilidad de riesgos vs. su efecto . . . . .	49
4.7. Riesgos identificados para el proyecto . . . . .	50
4.8. Riesgos clasificados según su probabilidad y efecto. . . . .	51
4.9. Estrategias de manejo de riesgos parte uno . . . . .	52
4.10. Estrategias de manejo de riesgos parte dos . . . . .	53
6.1. Caso de prueba TC-01 . . . . .	77
6.2. Caso de prueba TC-02 . . . . .	78
6.3. Caso de prueba TC-03 . . . . .	79
6.4. Caso de prueba TC-04 . . . . .	80
6.5. Caso de prueba TC-05 . . . . .	81
6.6. Caso de prueba TC-06 . . . . .	81
6.7. Caso de prueba TC-07 . . . . .	82
6.8. Caso de prueba TC-08 . . . . .	83
6.9. Caso de prueba TC-09 . . . . .	84
6.10. Matriz de trazabilidad . . . . .	84



# Capítulo 1

## Introducción

### 1.1. Introducción

En la Universidad Nacional de Córdoba se generan las llamadas actas de exámenes para dejar constancia de los resultados de las evaluaciones en los turnos de examen. Un acta consta de un encabezado con información sobre la materia a la cual corresponde, la fecha y otros datos necesarios y luego renglones con información sobre los alumnos inscritos junto con la nota correspondiente de cada alumno. Luego de que se completó y controló el acta, es cerrado y archivado. Además de quedar asentados en papel, las notas también son ingresadas al sistema Guaraní, un sistema de gestión de alumnos utilizado por la UNC.

Como el sistema Guaraní almacena la información ingresada en una base de datos, cualquier persona con los accesos y permisos suficientes puede ser, por ejemplo, un administrador de bases de datos, tiene la posibilidad de modificar la información almacenada. En la actualidad, la integridad de la información en el sistema Guaraní no puede ser garantizada, por lo cual certificados de historias académicas tienen que ser validadas contra las actas en papel. El proceso de validación suele llevar 10 días hábiles.

Una posible forma de automatizar el proceso podría implementarse mediante archivos en formatos legibles para computadoras, como por ejemplo JSON, en conjunto con una firma digital, para garantizar la integridad. Dicha solución, sin embargo, presenta dos problemas: en un ataque contra la disponibilidad de los datos, un administrador podría eliminar una parte de la información sin que esto sea detectado. Eso es especialmente un problema en el caso de actas rectificadas ya que no es posible modificar un acta cerrada, así las actas rectificadas digitales mantienen una firma válida y pueden volver a ser consideradas válidas en el caso de desaparición de su acta rectificativa.

Un Blockchain es una base de datos distribuida de solo escritura que hace uso de mecanismos criptográficos para que la información almacenada sea inmutable y segura. La distribución se logra a través de una red *peer to peer*, a la cual se pueden unir tantos nodos como se desea. Los datos son ordenados de forma cronológica y la automatización se vuelve posible al usar *smart contracts*, programas que son accionados por eventos

y pueden provocar cambios en el estado de los datos. De este modo, los problemas de integridad y disponibilidad podrían resolverse.

En respuesta a los problemas mencionados, la presente propuesta de proyecto integrador consiste en el desarrollo de un sistema de validación de información académica en Blockchain como prueba de concepto.

## 1.2. Motivación

Las motivaciones de llevar a cabo el trabajo integrador presente fueron las siguientes:

- La posibilidad de estudiar a una tecnología nueva.
- La oportunidad de participar en un proyecto de investigación y potencialmente realizar publicaciones científicas sobre lo estudiado.
- Si bien la tecnología Blockchain se conoce en el ambiente de las criptomonedas, su uso en otros ambientes no está establecido, por lo cual el desarrollo de un prototipo para un ambiente académico resulta de especial interés.

## 1.3. Marco institucional

El presente proyecto integrador forma parte del proyecto de investigación de “Tecnologías Blockchain” que cuenta con la participación de la Universidad Nacional de Córdoba y el Laboratorio de Redes y Computadoras. El equipo multidisciplinario de trabajo está compuesto por el Ing. José Daniel Britos, la Ing. Dra. Laura Cecilia Diaz Dávila y varios estudiantes de Ingeniería en Computación.

El objetivo general del proyecto de investigación es el de explorar la tecnología disruptiva “Blockchain” y sus posibles implementaciones más allá de su implementación original: “las criptomonedas” que dieron origen al concepto de cadena de bloques.

El rol del presente proyecto integrador, dentro del proyecto de investigación, es el de analizar, estudiar y emplear la tecnología Blockchain en un sistema de validación de información académica. Actualmente existe un proyecto semejante de validación de certificados con Blockchain en la Prosecretaría Informática de la UNC.[7]

## 1.4. Objetivos

El objetivo principal del presente trabajo es adquirir conocimientos sobre Blockchain y aplicar dichos conocimientos al desarrollar un prototipo para la validación de actas académicas y evaluar ventajas y desventajas de este caso.

### 1.4.1. Objetivos particulares

- Investigar la tecnología Blockchain, su funcionamiento y sus casos de uso a fondo.
- Investigar y estudiar diferentes implementaciones y plataformas existentes.
- Evaluar en base a los requerimientos, cuál es la implementación o plataforma más adecuada para realizar un prototipo.
- Configurar un Blockchain localmente en un ambiente de desarrollo e implementar *smart contracts* que permiten validar actas académicas.
- Complementar al Blockchain con una API y una interfaz gráfica para mejorar su usabilidad.

## 1.5. Estructura del texto

A continuación se describe el contenido de cada uno de los capítulos:

- **Capítulo 1 - Introducción:** El presente capítulo describe el trabajo en palabras generales, junto con motivaciones y objetivos del mismo.
- **Capítulo 2 - Marco Teórico:** El capítulo consta de tres partes: En primer lugar, se explica el funcionamiento y las características únicas de la tecnología Blockchain tomando la criptomoneda Bitcoin como ejemplo, seguido por una primera discusión sobre posibles requerimientos del prototipo. En segundo lugar, se analizan implementaciones alternativas detallando diferencias y similitudes con Bitcoin. Por último, se llega a una conclusión sobre qué plataforma resulta ser la más adecuada para la implementación de un prototipo.
- **Capítulo 3 - Hyperledger Fabric:** El capítulo describe con mayor detalle los componentes importantes y el funcionamiento de la plataforma que se eligió en el capítulo anterior.
- **Capítulo 4 - Análisis y Diseño:** Luego del resumen de una entrevista con el director de Prosecretaría de Informática de la UNC sobre el presente proyecto, se procede a describir las herramientas más importantes a utilizar y a definir los requerimientos del trabajo con mayor detalle.
- **Capítulo 5 - Implementación:** En primer lugar, se describen los pasos de configuración del Blockchain con Hyperledger Fabric junto con el desarrollo de los *smart contracts*. Además, se describe la implementación de la API como también la implementación de la interfaz gráfica de usuario, junto con primeras pruebas del sistema en un ambiente local.
- **Capítulo 6 - Pruebas y Validación:** En este capítulo se plantean los casos de prueba que validan los requerimientos especificados junto con los procedimientos empleados y los resultados correspondientes.

- **Capítulo 7 - Conclusiones:** En el capítulo final, se presentan limitaciones del prototipo desarrollado, posibles mejoras y conclusiones generales con respecto al trabajo.

## Capítulo 2

# Investigación de plataformas Blockchain

El presente capítulo introduce conceptos generales sobre Blockchain. Luego, se plantean los primeros requerimientos tentativos en base al conocimiento adquirido. Por último, se estudian implementaciones actuales de Blockchain y se analiza su utilidad para el proyecto integrador.

### 2.1. Bitcoin

La mejor forma de comprender la tecnología Blockchain y el potencial que tiene es estudiando su primera implementación: Bitcoin. En el *paper*, “Research Perspectives and Challenges for Bitcoin and Cryptocurrencies” [8], los autores extraen 3 componentes principales de la tecnología novedosa:

#### 2.1.1. Transacciones y scripts

En Bitcoin, una cuenta de usuario es representada por un conjunto de clave pública y privada. El *hash* de la clave pública funciona como número de cuenta a la cual se puede enviar dinero. Las transferencias de Bitcoin, de forma simplificada, consisten en un conjunto de entradas y un conjunto de salidas de monedas asociadas a los números de cuenta mencionados. El movimiento de dinero se puede entender como flujos, que se dividen o unen con cada transferencia realizada. Las salidas como entradas están vinculados a *scripts* que se tienen que ejecutar de forma correcta para que la operación sea declarada válida. El contenido de dichos *scripts* puede ser complejo, pero en transacciones comunes suele contener solamente comandos de validación de firma.

### 2.1.2. La red *peer to peer*

Cualquier PC con acceso a Internet puede unirse a la red Bitcoin. El participante nuevo establece conexiones con una cantidad aleatoria de otros participantes. Luego, información como bloques nuevos o transacciones pendientes son distribuidos vía *broadcast* e inundación. Además, el protocolo Bitcoin contiene varios mecanismos de seguridad: Cada nodo reenvía paquetes una sola vez para evitar bucles infinitos debido a la inundación. Transferencias de montos muy pequeños pueden ser usados para *penny-flooding attacks*, por lo cual nodos limitan la retransmisión de dichas transferencias por segundo.

### 2.1.3. Consenso y minado

Cuando un participante de la red realiza una transacción y la transmite a la red vía *broadcast*, esta tiene que ser validada y almacenada. Para lograr eso sin necesidad de un ente centralizado, Bitcoin usa varios mecanismos en conjunto.

- El Blockchain: Nodos especiales llamados mineros agrupan transacciones realizadas en bloques de un cierto tamaño, por ejemplo de 1 Megabyte en Bitcoin. Luego agregan el *hash* del bloque anterior a su bloque, enlazando así la información y formando una cadena de bloques con transacciones.
- Un mecanismo de consenso: El Blockchain es el registro de todas las transferencias realizadas de toda la red ordenadas en el tiempo. Cada nodo de la red tiene una copia idéntica almacenada. Por ende, todos los participantes tienen que estar de acuerdo sobre cuál es la versión válida y actual. No puede haber divisiones - si mineros publican 2 bloques al mismo tiempo y por latencias en la red hay nodos que agregan el bloque A y otros que agregan el bloque B, se produce lo que se llama *fork*. El protocolo está diseñado para que, después de un tiempo, todos los nodos opten por la misma rama.
- *Proof of Work*: Es el elemento esencial gracias al cual el Blockchain se vuelve inmutable y se llega al consenso descrito en el punto 2. Antes de poder publicar un bloque para que forme parte del Blockchain, cada minero realiza un cálculo computacional complejo, cuyo resultado es agregado al bloque y resulta fácil de verificar por otros. La dificultad del cálculo está definida en el código para que un bloque sea publicado cada 10 minutos en promedio. En el caso de una división, la red decide por la rama con el mayor *Proof of Work* acumulado. El mecanismo también resuelve otros problemas: como la publicación de un bloque es costosa en términos de capacidad de procesamiento, se evita que un nodo malintencionado produzca un *spam* de bloques. La secuencia forzada y el abandono de ramas debido a *forks* logra mantener transacciones coherentes en el Blockchain. Por último, asegura la inmutabilidad del Blockchain: si un minero quiere insertar una transacción fraudulenta en el bloque 4 de una cadena de 6 bloques, es necesario volver a computar el *Proof of Work* de los bloques 4, 5 y 6 antes de que otro minero adjunte el bloque nro. 7 a la cadena original.

Una de las desventajas principales de *Proof of Work* consiste en la competencia entre los mineros: En un instante dado, todos los mineros están trabajando para obtener resultado del cálculo complejo, pero solo el primero en obtenerlo puede adjuntar el bloque nuevo al Blockchain. Los otros tienen que abortar su cálculo, ensamblar un bloque nuevo y volver a empezar. Eso ha llevado a críticas por el consumo de energía eléctrica de la red Bitcoin: En Agosto del 2018, el consumo del año 2018 fue estimado en 73,12 TWh - la cantidad de energía que necesita la población completa de Austria durante un año.[9].

Una vez que Bitcoin ganó popularidad, se dieron dos tendencias: por un lado, muchos programadores lanzaron otras criptomonedas modificando el código fuente de Bitcoin. Las criptomonedas alternativas recibieron el nombre *altcoin* y en Agosto del 2018 se listan 1855 *altcoins* en total[10]. Por otro lado, desarrolladores vieron posibilidades más allá de las criptomonedas e implementaron productos con tecnología Blockchain cuyos enfoques no se limitan a monedas digitales. Más adelante se van a analizar varios de esos productos.

## 2.2. Primeras especificaciones para el prototipo

Luego de analizar el funcionamiento de Bitcoin, Blockchain no aparenta ser una solución conveniente para los problemas planteados en el capítulo anterior. Una arquitectura que se basa en la ausencia de confianza y una base de datos que requiere de la competencia de mineros para volverse inmutable, no parece adecuarse a un sistema estructurado con autoridades centrales como la universidad.

Si bien implementar un prototipo de un sistema de distribución de información académica en Bitcoin no aporta mucho valor - dado desventajas evidentes como la posible participación de cualquier persona en el mundo - implementar dicho prototipo en un Blockchain tiene sentido, siempre y cuando las necesidades de la universidad concuerdan con la arquitectura del sistema.

Por eso, a continuación, se detallan las limitaciones del Blockchain de Bitcoin junto con las modificaciones correspondientes que deben tenerse en cuenta para que la elección de un Blockchain como solución resulta ser conveniente:

### ■ Control de Acceso

Para formar parte de la red Bitcoin, sólo es necesario instalar un software en una computadora y crear una cuenta. No existen restricciones de acceso, la participación es anónima y cualquier nodo puede inspeccionar el historial completo de las transferencias. Para el caso del prototipo planteado, dichas características son indeseadas: el acceso de entidades ajenas debe estar restringido y la información debe ser visible únicamente por las partes autorizadas. Eso implica que es necesario conocer identidades y controlar sus accesos al Blockchain. Existen 2 modelos para el caso descrito: Blockchains privados y Blockchains de consorcio.

Un Blockchain totalmente privado es un Blockchain donde los permisos de escritura se mantienen centralizados en una organización. Los permisos de lectura pueden ser públicos o restringidos en una extensión arbitraria.[11]

Un Blockchain de consorcio es un Blockchain donde el proceso de consenso es controlado por un conjunto de nodos preseleccionados.[11]

Como la información debe ser escrita y accedida desde diferentes facultades o eventualmente universidades, un Blockchain de consorcio es la opción más conveniente para el prototipo.

#### ■ Privacidad de la información

En el caso de Blockchains de consorcio o Blockchains privados, el acceso está limitado a las partes autorizadas. Sin embargo, todas las partes autorizadas tienen acceso equitativo a toda la información. Si se desea compartir ciertos datos y mantener otros en privado, se necesita un sistema que otorga derechos de lectura y escritura individuales para cada participante.

#### ■ Simplificación del consenso

El proceso de minado en conjunto con el mecanismo criptográfico *Proof of Work* establece confianza entre partes desconocidas y logra la inmutabilidad de las transferencias en Bitcoin. Dicho trabajo es realizado por nodos mineros que compiten entre ellos para validar transacciones, calcular *Proof of Work* y ganar recompensas monetarias.

Para un Blockchain de consorcio que almacena información académica, dicho mecanismo presenta una solución ineficiente: ya existe un cierto grado de confianza entre los participantes, por lo cual una competencia entre nodos mineros se vuelve innecesaria. El alto consumo energético que eso implica puede resultar un impedimento para la adaptación de la tecnología.

Una alternativa adopción más conservadora con el uso de recursos computacionales es el mecanismo *Proof of Stake*:

Protocolos de criptomonedas que intentan evitar el desperdicio de recursos físicos escasos comúnmente confían en *proof of stake*, es decir, en mecanismos que dan poder de decisión con respecto a la continuación del *ledger* a entidades que poseen monedas dentro del sistema.[12]

Es decir, el creador de un bloque nuevo es elegido de forma aleatoria. Para poder distinguir ambos mecanismos, se dice que bloques nuevos son forjados y no minados. Dependiendo de la implementación, diferentes parámetros como antigüedad o riqueza pueden favorecer en el momento de la elección del nodo forjador.

#### ■ Reducción de costos transaccionales

Mineros en Bitcoin son recompensados por asegurar al Blockchain con sus recursos: Por cada bloque publicado, un minero recibe tarifas de las transacciones que validó y una recompensa adicional establecida por el protocolo llamada *block reward*.



El principal gasto que debe pagar un Blockchain es la seguridad. El Blockchain debe pagar a los mineros o validadores para que participen económicamente en su protocolo de consenso, ya sea prueba de trabajo o prueba de participación, y esto inevitablemente tiene algún costo. Hay dos formas de pagar este costo: inflación y tarifas de transacción.[13]

Ambos costos tienen desventajas para empresas: Transacciones tarifadas significan un gasto adicional para una empresa que se eleva con la cantidad de transacciones realizadas. En su peor caso pueden inhibir el uso de la tecnología.

El empleo de un *block reward* significa que se crean criptomonedas con cada bloque nuevo que se agrega al Blockchain. De esta forma, la cantidad total de monedas sube constantemente. Una empresa tiene que tener en cuenta el impacto que va a tener dicha inflación en el futuro.

Debido a las complicaciones mencionadas, se busca una solución que logre asegurar el Blockchain con métodos alternativos. Para el caso del prototipo a implementar, la mejor opción es usar un Blockchain sin criptomoneda.

#### ■ Automatización

Los *Smart contracts* fueron implementados por primera vez en Ethereum, la implementación de Blockchain que siguió a Bitcoin: se trata de código que se ejecuta automáticamente cuando se cumplen ciertas condiciones. Son usados por ejemplo por la aplicación Crypto Sportz que automatiza así el pago de apuestas deportivas:

Cuando se ingresen los resultados del juego en el contrato, [este] reconocerá los boletos con la predicción correcta como boletos ganadores y dividirá su saldo ETH por el número de boletos ganadores.[14]

Aplicaciones que funcionan con *smart contracts*, forman una capa de abstracción “por encima” del Blockchain, similar a la capa de aplicación en TCP/IP. Los *smart contracts* reciben parámetros y escriben transacciones en el Blockchain. Así se pueden automatizar operaciones específicas. Si bien la lógica de negocios suele estar embebida en la aplicación, *smart contracts* pueden ayudar a autorizar accesos de lectura y escritura o controlar la validez de información a escribir. Por sus posibilidades de automatización, es deseado que el prototipo a implementar permita escribir *smart contracts*.

Dado que en el presente proyecto integrador se quiere emplear una aplicación usando Blockchain y no construir un Blockchain nuevo, es conveniente partir de un código *open source* que se pueda modificar o utilizar un SDK que permite configurar el Blockchain para que cumpla con los requisitos mencionados. Para encontrar una opción conveniente, se estudiaron algunas implementaciones existentes que ofrecen características más allá de una criptomoneda: El criterio de selección de las plataformas a estudiar fue que la idea del desarrollo no se enfoque principalmente en la criptomoneda, sino en funcionalidades adicionales que podrían ser de utilidad para el prototipo a implementar. A continuación se describe cada una de las opciones junto con sus ventajas y desventajas para la implementación del prototipo.

## 2.3. Ethereum

Ethereum es una plataforma creada por el programador Vitalik Buterin, miembro activo en la comunidad Bitcoin desde 2011.[15] Buterin vio la posibilidad de utilizar la tecnología Blockchain para propósitos más allá de las criptomonedas y desarrolló Ethereum. La idea consistía en crear un Blockchain que además de soportar transacciones monetarias también sea capaz de ejecutar código escrito por sus usuarios.

Con un lenguaje de programación completo, las aplicaciones potenciales del Blockchain de Ethereum están limitadas solo por la creatividad de un desarrollador.[16], explica Buterin en una entrevista.

### 2.3.1. Funcionamiento

Como Bitcoin, Ethereum permite comparar e intercambiar criptomonedas, en este caso con el nombre Ether. El Blockchain es público: cualquier usuario puede crearse una cuenta, realizar transferencias y recibir Ethers. Las cuentas de usuarios se llaman “cuentas externas” y se accede con un conjunto de clave pública y privada.

Otro tipo de cuenta en Ethereum son las cuentas de contrato: Ejecutan código escrito por el usuario llamado *smart contract* cada vez que reciben un mensaje en forma de una transferencia firmada. Durante la ejecución es posible enviar un mensaje de respuesta, pedir la ejecución de otro *smart contract* o crear uno nuevo.

Dichos conceptos dieron lugar a la definición de una DAO (*decentralized autonomous organization*), una organización cuyas reglas de funcionamiento están escritas en forma de *smart contracts*. Así, las decisiones se toman automáticamente, sin necesidad de gerentes o ejecutivos. Para resolver casos imprevistos en el código, se recurre a votaciones entre los miembros de la organización.

En Ethereum, el minado funciona muy parecido a Bitcoin: Nodos mineros validan transacciones, las agrupan en bloques y compiten por ser los primeros en calcular el correspondiente *Proof of Work*. Un bloque nuevo es publicado aproximadamente cada 14 segundos.[17] El minero ganador cobra una recompensa fija y las tarifas de todas las transacciones incluidas en el bloque. Sin embargo, Ethereum anunció que en un futuro planea modificar el protocolo de consenso a una versión de *Proof of Stake* llamada Casper.

Junto con la validación de transacciones, los mineros también procesan los *smart contracts*. Estos son ejecutados en un entorno de *sandboxing* llamado EVM (*Ethereum Virtual Machine*). El trabajo computacional de cada operación del *smart contract* es calculado en la unidad *gas*. El usuario interesado en la ejecución del código tiene que especificar una cantidad de *gas* con su pedido. El minero ejecuta el código de forma exitosa si la cantidad de *gas* fue suficiente. Los resultados de cualquier ejecución - exitosa o no - son incluidos en un bloque y publicados en el Blockchain.

*Gas* tiene las siguientes funcionalidades en Ethereum: Protege los mineros de bucles infinitos en los *smart contracts*, ya que la ejecución se termina con una cantidad de *gas* insuficiente. Además sirve como mecanismo de pago por capacidad computacional: *gas* tiene un equivalente en Ether y la cantidad de *gas* usada por la ejecución multiplicada por su valor en Ether forman la recompensa del minero. Por último, la relación Ether por *gas* especificada por el usuario determina la velocidad con la cual la transacción es procesada: un valor alto implica mayor ganancia, lo cual significa una validación más rápida.

Por más que la plataforma es pionera en *smart contracts*, Ethereum cuenta con una serie de inconvenientes para el proyecto integrador:

- Es un Blockchain público y no es posible realizar transacciones privadas. Todas las transacciones se almacenan juntas con transacciones de otras aplicaciones en el mismo Blockchain y son visibles para todas las participantes.
- La fecha estimada para el cambio de protocolo de consenso es el 3 de enero 2020[18] y ya se estableció anteriormente que *Proof of Work* es un protocolo inadecuado para el prototipo a implementar.
- La compra de Ether es imprescindible para el uso de la plataforma.
- Existe la posibilidad de clonar el repositorio de Ethereum y modificar los aspectos requeridos del código, pero según la herramienta GitHub Gloc, el proyecto tiene aproximadamente 1.072.000 líneas de código y resulta ser muy grande para ser adaptado por una sola persona.

## 2.4. Multichain

Como ya se analizó, Blockchain y criptomonedas alternativas no presentan una solución conveniente para transacciones entre empresas, dada la falta de privacidad y el costo que implican las transferencias. Con el objetivo de resolver dichos problemas, Dr. Gideon Greenspan y Dr. Michael Rozantsev adoptaron el código de Bitcoin y crearon Multichain.

[Se trata de] una plataforma independiente para la creación y el despliegue de Blockchains privados, dentro o entre organizaciones. Su objetivo es superar un obstáculo clave para el despliegue de la tecnología Blockchain en el sector financiero institucional, proporcionando la privacidad y el control requeridos en un paquete fácil de usar.[19]

Es decir, con Multichain es posible crear un Blockchain propio adaptado al caso de uso deseado con una criptomoneda nativa.

### 2.4.1. Funcionamiento

Para adaptar Bitcoin a las necesidades de empresas, Multichain mejora tres aspectos: establece un sistema de acceso, modifica el algoritmo de minado y posibilita eliminar los costos de transferencias.

- **Restricción de Acceso**

Bitcoin utiliza una infraestructura de clave pública para sus cuentas. Multichain agrega listas de accesos a dicho mecanismo: Para que un nodo pueda conectarse con la red, es necesario que se autentique. En el caso de una autenticación inválida, la conexión se rechaza. Las listas de acceso también se usan para administrar lectura, transacciones y minado. Los permisos son otorgados por los administradores del Blockchain.

- **Minado** El concepto del minado en Multichain es igual al de Bitcoin, con la diferencia que Multichain evita la competencia entre los mineros forzando un turno rotatorio con una variable de *spacing*. Esta define la cantidad de bloques que un minero tiene que esperar hasta poder minar nuevamente. Si se toma por ejemplo el valor 7, el minero tiene que esperar que los demás agreguen 6 bloques luego del suyo. Sino el bloque será declarado inválido.

Al igual que Bitcoin, Multichain utiliza *Proof of Work*, pero el minado en turno rotatorio evita la competencia y el alto consumo energético asociado. Según el *white paper*, es posible que el mecanismo sea reemplazado por *Proof of Stake* en un futuro.

- **Costo de transacciones** “En una cadena de bloques Multichain, las tarifas de transacción y las recompensas por bloque son cero por defecto.” [19] Eso se debe a que en un ambiente privado, en el cual existen controles de acceso y confianza entre participantes, no se requiere el mismo incentivo económico que en Bitcoin para lograr seguridad e inmutabilidad del Blockchain. Sin embargo, es posible adaptar costos de transacciones y minado: Se puede configurar la dificultad de *Proof of Work*, los costos de transacciones y recompensas por bloques.

Un aspecto interesante de Multichain es que los desarrolladores apuntan a mantener la compatibilidad con Bitcoin, para que en un futuro sea posible mover activos entre Blockchains privados de empresas y el Blockchain de Bitcoin.

Para el proyecto integrador presente, Multichain cumple con varios requisitos previamente establecidos:

- Permite implementar un Blockchain privado de forma sencilla con CLI.
- Tiene un mecanismo de control de acceso para participantes.
- El proceso de minado requiere pocos recursos.
- Transacciones pueden ser configuradas como gratuitas.

- Es posible establecer controles de lectura con las claves públicas.

Sin embargo, es una plataforma pensada para el sector financiero. No soporta *smart contracts* y la implementación de los mismos tampoco figura en el *roadmap* del producto.

## 2.5. EOSio

EOSio es un producto de block.one, una empresa desarrolladora de software liderada por CEO Brendan Blumer y CTO Daniel Larimer. Su idea se originó por los altos costos de uso de implementaciones existentes como Bitcoin o Ethereum, como describen en su *whitepaper*:

Las plataformas existentes de Blockchain cobran tarifas altas para transferencias y su capacidad computacional limitada impide la adopción generalizada de Blockchain.[20]

Pero también mencionan la dificultad de aprendizaje como un impedimento para la adaptación de la tecnología. EOSio fue implementado con la idea de proveer una plataforma de *smart contracts* que fuera de uso gratuito, aprendizaje fácil y que soporte millones de transacciones con baja latencia.

### 2.5.1. Funcionamiento

#### ■ Sistema de accesos

Similar a Ethereum, todas las transacciones en el Blockchain EOSio son públicas y se guardan en un único Blockchain. La criptomoneda de la implementación se llama EOS.

El concepto de cuentas a cambio es diferente: Varios usuarios pueden ser asociados a una cuenta y ejecutar diferentes funciones dependiendo de sus privilegios. Un ejemplo puede ser un blog descentralizado, donde Roberto y Sara son administradores. Una transacción que modifica la configuración de la cuenta puede requerir la aprobación de ambos, mientras la publicación de un artículo nuevo puede realizarse por uno solo.[20]

También es posible definir un peso para confirmar transacciones: Si cambiar configuraciones de cuenta requiere dos confirmaciones pero la decisión de Roberto tiene un peso de 2, Roberto solo podría realizar dicha transacción.

La cantidad de niveles de permisos, los pesos y las acciones permitidas son libremente configurables por los administradores de la cuenta.

#### ■ Creación de bloques

EOSio utiliza un algoritmo de consenso llamado *Delegated Proof of Stake*, una optimización de *Proof of Stake*. En vez de elegir un nodo forjador de forma aleatoria

para cada bloque, se define una cantidad fija de forjadores llamados delegados a través de un sistema de votos.

En EOSio hay 21 delegados que forjan un bloque cada 0,5 segundos en turno rotatorio. Cada 126 bloques (6 bloques forjados por cada delegado), los votos para los delegados son analizados y se produce un cambio si un no delegado obtuvo más votos que uno de los delegados.

La recompensa por forjar bloques se suele compartir con los votantes y un delegado que se comporta de manera egoísta perderá votos. Delegados fuera de línea que no forjaron bloques por más que 24 horas son reemplazados.

#### ■ Costo de transacciones

En EOSio, los *smart contracts* se llaman acciones y estas pueden implicar transferencias de tokens entre cuentas. Los forjadores ejecutan el código de las aplicaciones y validan las transacciones resultantes. Como no existen tarifas de transacciones, los forjadores ganan únicamente recompensas en forma de *block reward*, lo cual se traduce en una inflación anual del 5 % [20], una decisión de diseño de los creadores.

El modelo de transacciones gratuitas implica que el desarrollador puede ofrecer aplicaciones gratuitas o *freemium*. Usuarios pueden probar un servicio y luego decidir si quieren o no pagarlo.

La capacidad de procesamiento y el ancho de banda de la red se gestionan en base al uso de la misma y en base a la riqueza de una cuenta. Es decir, con mayor monto de EOS en una cuenta, se permite una mayor cantidad de transacciones por unidad de tiempo. De esa forma se evitan sobrecargas de la red y ataques de *spamming*.

#### ■ Otros aspectos

Junto con las características mencionadas, EOSio cuenta con una serie de funcionalidades adicionales que mejoran la performance del software y la experiencia para desarrolladores como usuarios finales:

- Acciones con retraso obligatorio:

El software EOS.IO permite a los desarrolladores indicar que ciertas acciones deben esperar un período de tiempo mínimo antes de poder aplicarse. Durante este tiempo, pueden ser cancelados.[20]

- La posibilidad de recuperar una cuenta por más que se perdieron las claves privadas para acceder a ella. En Bitcoin y muchas otras criptomonedas, los fondos son asociados solamente a un conjunto de clave pública y clave privada. Las Transacciones son aprobadas si se puede verificar que fueron firmadas con la clave privada. Si ésta se pierde, el dueño también pierde la posibilidad de usar sus fondos. EOSio posibilita recuperar una cuenta en el caso del extravío de la clave privada si antes de la pérdida se definió un socio de recuperación.
- Los desarrolladores de EOSio consideran que existen situaciones donde es necesario llegar a un acuerdo sobre “asuntos subjetivos de acción colectiva que no pueden ser capturados completamente por algoritmos de software.” [20] Por esa razón, los delegados forjadores de bloques también tienen una función de

gobierno: disponen de la autoridad para congelar cuentas por comportamientos inesperados debidos a *bugs* o ataques, y pueden realizar actualizaciones en el software de EOSio.

EOSio cumple con una variedad de requerimientos para el proyecto integrador:

- El algoritmo de consenso junto con un gobierno se adapta a las necesidades de un Blockchain privado con una gestión central.
- Las transacciones no tienen costo en forma de tarifa.
- Las cuentas tienen funciones que pueden ser invocadas por diferentes personas con diferentes privilegios, lo cual permite un control de acceso de grado fino.
- *Smart contracts* están implementados en forma de acciones y se programan en C++.

Sin embargo EOSio, también cuenta con inconvenientes:

- Las transferencias no son privadas y los datos son visibles para todos los integrantes del Blockchain.
- EOSio es un Blockchain público y los desarrolladores de aplicaciones tienen que disponer de EOS en su cuenta para que sus transacciones sean procesadas.

Una posibilidad es modificar el código fuente de EOSio.

Personas interesados en construir su propia cadena de bloques derivados de nuestro software EOSio pueden bifurcar nuestro repositorio y personalizarlo para su uso.<sup>[21]</sup>

Según GitHub Gloc el repositorio de EOSio cuenta con aproximadamente 624.000 líneas de código, por lo cual, como en el caso de Ethereum, la adaptación del código fuente se escapa del alcance de un proyecto integrador.

## 2.6. IOTA

IOTA es un proyecto de la IOTA Foundation dirigido por David Sønstebø y Dominik Schiener. Cuenta con un equipo de más de 50 personas compuesto por matemáticos, ingenieros en software, especialistas en IoT y otros. En su página web, describen dos problemáticas que llevaron a la creación de IOTA:

1. El crecimiento de la cantidad de dispositivos inteligentes resulta en un crecimiento de tráfico IP, ya que todos estos dispositivos consumen y producen datos. El ancho de banda crece más lento y a largo plazo la conexión entre dispositivos IoT con nubes centralizadas va a resultar problemática. Será necesario recurrir a *fog computing* en sistemas distribuidos que almacenan y procesan información.

2. Tecnología Blockchain como base de datos distribuida parece ser una solución atractiva para computación de niebla, pero sufre de problemas de escalamiento:

No es difícil ver que una cadena compuesta por bloques de tamaño finito, que solo se pueden producir cada intervalo de tiempo, produce un cuello de botella de rendimiento y conduce a altas tarifas de transacción que deben pagarse a los mineros.[22]

Con su producto, la IOTA Foundation plantea una solución para ambos problemas: Se trata de tecnología distribuida cuya capacidad de procesamiento escala con el tamaño de la red, mientras mecanismos criptográficos aseguran de la inmutabilidad de la información.

### 2.6.1. Funcionamiento

En IOTA, las transacciones no se agrupan en bloques de datos, no se forma una cadena ni tampoco hay nodos mineros o forjadores que validan las transacciones. Los diseñadores consideraron dichos aspectos responsables por los problemas de escalabilidad de la tecnología:

Si se quisieran eliminar las tarifas y permitir que el sistema escalara, la idea natural sería eliminar el cuello de botella y los mineros.[22]

La solución que presentan consiste en escribir las transacciones en un grafo acíclico dirigido. Las transacciones representan vértices. Los nodos representan aprobaciones. Cada vez que un participante publica una transacción nueva, también valida dos transacciones ya existentes, de modo que con mayor cantidad de transacciones agregadas, también hay mayor actividad de validación. Su funcionamiento se puede ver en la Figura 2.1: nodos grises son transacciones nuevas sin validar, nodos violetas son transacciones pocas validadas y nodos celestes representan transacciones validadas muchas veces. Dicha estructura recibió el nombre Tangle, que es administrado por la IOTA Foundation y actualmente es posible transferir tokens de la criptomoneda IOTA en él.

#### ■ Cuentas

A diferencia de otras criptomonedas, IOTA no usa un conjunto de clave pública con clave privada para definir una cuenta. Cada usuario posee una semilla, que es un número aleatorio y secreto. Con la semilla se crea una clave pública y una clave privada. La clave pública es usada como destino para fondos. Si el usuario desea gastar los fondos asociados a dicha clave, firma la transferencia con la clave privada correspondiente. Luego, el conjunto de clave pública y privada es descartado y se genera uno nuevo con la semilla secreta. Ese mecanismo se llama esquema de firma única, ya que cada transacción es firmada con una única clave que no se reutiliza.

#### ■ Validación

Para transferir tokens IOTA de una cuenta a otra, hay que seguir los siguientes pasos:



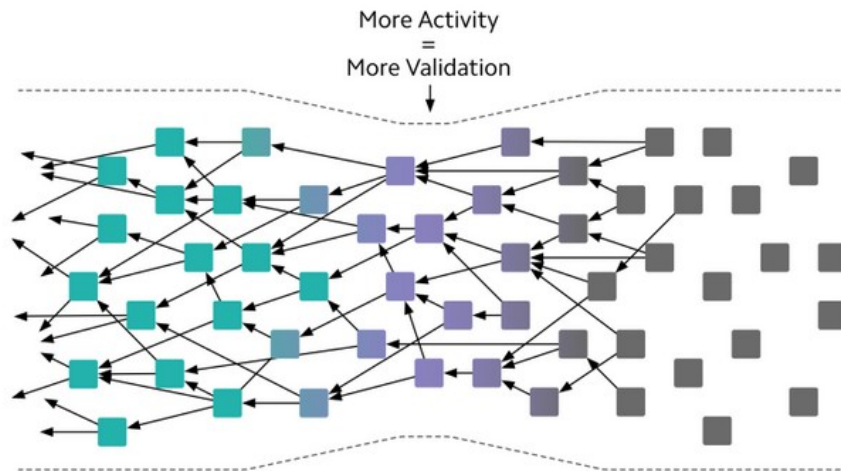


FIGURA 2.1: Funcionamiento del Tangle de [1].

1. La transacción tiene que ser firmada con claves que todavía no han sido utilizadas.
2. Es necesario elegir y validar dos *tips*: *Tips* son transacciones cuya historia todavía no fue comprobada y se verifica que no existen contradicciones entre ambas transacciones.
3. Se calcula un *Proof of Work* para cada transacción que se publica. De esa forma se evita *spam* en la red.

Luego de completar los pasos descritos, la transacción nueva es publicada como *tip* y otros nodos pueden elegirla para la validación.

#### ■ Costo de transacciones

Ya que no existen mineros ni forjadores, tampoco existen tarifas de transacciones. El costo de una transacción consiste en el esfuerzo computacional que requiere validar dos *tips* y calcular el *Proof of Work*.

IOTA resulta ser una implementación prometedora ya que desafía la arquitectura tradicional del Blockchain e introduce mayores cambios en su funcionamiento. Todavía se encuentra en las etapas tempranas de desarrollo y queda por ver si las mejoras introducidas producen el impacto esperado.

Para la aplicación a implementar, la arquitectura de IOTA presenta los siguientes inconvenientes:

- El esquema de firma única no permite asignar una identidad única a los participantes de la red.
- La solución “Qubic” para que IOTA soporte *smart contracts* todavía está en desarrollo y no existe una fecha estimativa de lanzamiento. [23]

## 2.7. Lisk

Lisk es un producto de la Lisk Foundation: CEO Max Kordek y CTO Oliver Beddows quieren implementar un *framework* que ayuda con la creación de Blockchains nuevos - públicos o privados - para una adaptación más simple y rápida de la tecnología. En mayo del 2016, se lanzó su Blockchain que actualmente permite el intercambio de la criptomoneda LSK y la implementación de aplicaciones descentralizadas, parecido al caso de Ethereum y EOSio. El *framework* que facilita la construcción de un Blockchain propio está en versión alpha y todavía se encuentra en desarrollo.

### 2.7.1. Funcionamiento

- **Cuentas**

Para poder empezar a interactuar con la red de Lisk, es necesario instalar Lisk Hub.

Lisk Hub es una solución todo en uno para administrar el ID de Lisk, acceder a los tokens de Lisk, enviarlos así como también para votar a los delegados. Combina la funcionalidad de la antigua billetera y un explorador de Blockchain.[24]

A partir de una frase de contraseña secreta se genera un Lisk-Id, un identificador único de la cuenta de usuario. Con dicho Lisk-Id es posible recibir y transferir fondos. Las transacciones tienen que ser firmadas con la frase de contraseña secreta para ser procesadas.

- **Validación**

Lisk utiliza *delegated proof of stake* para forjar bloques nuevos.[25] Existen 101 delegados que crean bloques en turno rotatorio y se genera un bloque nuevo cada 10 segundos. Cada bloque incluye 25 transacciones, de modo que Lisk tiene un rendimiento de 2,5 transacciones por segundo. Los delegados cobran un monto de 5 LSK por cada bloque creado y adicionalmente una tarifa por cada transacción incluida en el bloque. Los participantes de la red deben votar a los delegados y pueden recibir un porcentaje de la ganancia del delegado como incentivo de voto.

- **Costo de transacciones**

Las transacciones en la red Lisk tienen un costo fijo de 0,1LSK, sin importar el monto que se está transfiriendo.[26] Existe una propuesta para la implementación de tarifas dinámicas pero todavía no se encuentra considerada en el *roadmap* de la implementación.

Para utilizar el *framework*, los futuros implementadores Blockchain pueden descargar las herramientas Lisk Commander, Lisk Core y Lisk Elements, que forma un conjunto de librerías y herramientas que permiten configurar parámetros de un Blockchain propio. El lenguaje principal es Javascript, de esa manera, programadores que provienen de un ambiente de desarrollo web pueden empezar a trabajar con Lisk sin tener que aprender un

lenguaje nuevo. Será posible ajustar el mecanismo de consenso, tarifas de transacciones, derechos de accesos, rendimiento como también codificar la aplicación.

Sin embargo, no es la visión de la fundación Lisk que en un futuro haya una multitud de Blockchains aislados entre sí: su objetivo es la interacción entre “Sidechains”.

Sidechains fueron descritos por primera vez en el whitepaper “Enabling Blockchain Innovations with pegged Sidechains” donde los autores definen el concepto de la forma siguiente:

Proponemos una nueva tecnología, los Sidechains vinculados, que permite transferir Bitcoins y otros activos contables entre múltiples Blockchains.[27]

Así, un Sidechain es un Blockchain que implementó un mecanismo para poder intercambiar activos con otros Blockchains. Construir aplicaciones en Sidechains tiene varias ventajas:

1. Es posible definir el mecanismo de consenso y forjado más conveniente para una aplicación.
2. Problemas como errores de código o ataques solamente afectan a un Sidechain.
3. Bienes pueden ser intercambiados entre diferentes Sidechains sin ser convertidos: un Bitcoin en un Sidechain sigue siendo un Bitcoin.

Max Kordek describe su idea para los Sidechains de Lisk de la siguiente manera:

Es posible usar las funcionalidades del Sidechain de otras personas dentro de un Sidechain propio. Por ejemplo, si hay un servicio para enviar mensajes y otro que permite subir imágenes, se pueden aprovechar estas dos aplicaciones para construir una aplicación como una red social, sin volver a implementar la subida de imágenes y el servicio de mensajería. Simplemente se usan otras aplicaciones de Blockchain para eso. Es como un cerebro que evoluciona con cada Sidechain que está conectada al sistema.[28]

Según el CEO del proyecto, una vez que el *framework* está implementado, será posible implementar un Blockchain privado, adaptar parámetros como costos de transacción y el mecanismo de consenso, e implementar la aplicación que funcione con dicho Blockchain. El *framework* tendría la ventaja para los desarrolladores que mientras ellos se pueden concentrar en escribir la aplicación, Lisk se encargaría de mejorar el funcionamiento de la tecnología Blockchain subyacente. El proyecto tiene aspectos prometedores para el futuro y si su desarrollo estuviese más avanzado, sería considerado como una opción para este proyecto integrador, ya que cumple con los requisitos establecidos.

## 2.8. Hyperledger

Hyperledger es un proyecto de la Linux Foundation que fue iniciado en diciembre de 2015.

El proyecto va a desarrollar un *framework* de código abierto y grado empresarial para *ledgers* distribuidos, con el fin de que desarrolladores se enfoquen en construir aplicaciones robustas y específicas de su industria.[29]

Los términos *ledger* y Blockchain en este contexto describen el mismo concepto, y más adelante se va a aclarar la diferencia entre ambos.

Una variedad de empresas apoya la iniciativa: en septiembre de 2018 figuran más que 250 miembros en la página web, entre otros Cisco, Hitachi y J.P.Morgan. IBM e Intel aportaron código de trabajos relacionados con Blockchain, lo cual formó la base para los dos proyectos más avanzados: Hyperledger Fabric y Hyperledger Sawtooth.

Desde los inicios de Hyperledger se sumaron una multitud de proyectos. En total hay 5 *frameworks* para diferentes tipos de Blockchains y 5 herramientas que facilitan la interacción y el mantenimiento. En la figura 2.2 se ve el enfoque de cada uno.

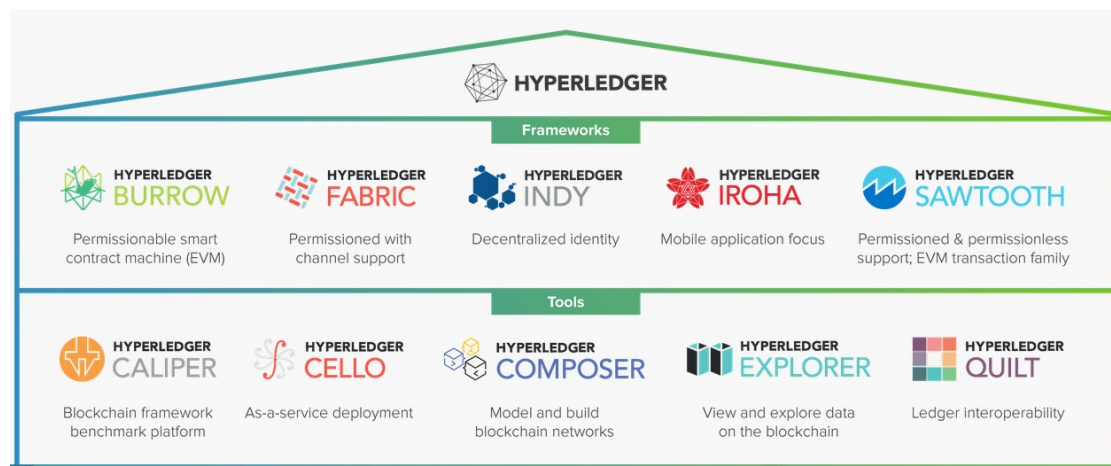


FIGURA 2.2: Proyectos en Hyperledger de [2].

La mayoría de los proyectos que figuran en 2.2 se encuentran en la etapa de incubación, por lo cual su código y su documentación se encuentran incompletos. A continuación se van a describir con mayor detalle los dos proyectos más avanzados, que cuentan con una versión 1.0 y pueden ser usados en producción.

### 2.8.1. Hyperledger Fabric

Hyperledger Fabric fue el primero de los proyectos Hyperledger y su versión 1.0 se lanzó el 11.7.2017.[30] Es un *framework* para implementar Blockchains de uso empresarial. Fue diseñado con los siguientes requerimientos:[31]

- Los participantes tienen que ser identificables.
- El acceso a la red debe ser autorizado.
- El sistema necesita alto rendimiento para procesar transacciones.
- Las transacciones se tienen que confirmar con baja latencia.
- El sistema debe proveer privacidad y confidencialidad para las transacciones y sus datos relacionados.

Los requerimientos especificados se implementaron de la siguiente manera:

■ **Cuentas**

Organizaciones que participan en el Blockchain hacen uso de una autoridad de certificación - puede ser propia o tercera - para otorgar identidades digitales a sus usuarios. Hyperledger Fabric cuenta con un *Membership Service Provider* y listas de control de acceso: cada identidad digital es verificada con respecto a su validez y sus autorizaciones cuando se conecta a la red.

■ **Validación**

En Hyperledger Fabric, los nodos que validan transacciones no son los mismos que los nodos que crean bloques nuevos. Así, la carga de procesamiento en la red se distribuye mejor. Además, la división permite la ejecución paralela de transacciones no relacionadas, mejorando así el rendimiento del Blockchain.

■ **Costo de Transacciones**

Si bien es posible definir una criptomoneda nativa en Hyperledger Fabric, no es necesario y el protocolo de consenso no lo requiere.

Fabric puede utilizar protocolos de consenso que no requieren de una criptomoneda nativa para incentivar el minado costoso o para impulsar la ejecución de *smart contracts*.<sup>[31]</sup>

Por ende, las transacciones pueden ser ejecutadas sin pagar una tarifa.

### 2.8.2. Hyperledger Sawtooth

El 30.1.2018, Hyperledger lanzó la versión 1.0 de un segundo proyecto llamado Hyperledger Sawtooth.<sup>[32]</sup> El código inicial fue aportado por Intel y fue diseñado con los siguientes objetivos:<sup>[33]</sup>

- Construir una plataforma para *ledgers* distribuidos que sean privados o públicos.
- Lograr una configuración fina de permisos de accesos y transacciones.
- Asegurar que *smart contracts* sean seguros.

- Obtener una separación estricta de la lógica de negocios con la implementación subyacente y facilitar así el desarrollo.

El software funciona de la forma siguiente:

- **Cuentas**

Con la ayuda de identidades criptográficas se crean roles con determinados permisos. Los clientes interactúan a través de una REST-API con una red de validadores, que verifican los permisos de los clientes y rechazan peticiones si los permisos no son suficientes. Si la petición de una transacción se considera válida, se la envía a un nodo procesador de transacciones para su ejecución. Dichos nodos son los que actualizan el estado del Blockchain.

- **Validación**

Como en Hyperledger Fabric, la validación de transacciones y la creación de bloques ocurren en nodos diferentes. Hyperledger Sawtooth ofrece un mecanismo de consenso nuevo llamado *Proof of Elapsed Time*, que aprovecha un ambiente seguro de ejecución en los procesadores Intel. Sin embargo, una característica de Hyperledger Sawtooth es el “mecanismo de consenso intercambiable”: La forma de llegar al consenso puede ser cambiada por una alternativa proponiendo un cambio a través de una transacción.

- **Costo de Transacciones**

Al igual que en Hyperledger Fabric, Hyperledger Sawtooth no requiere de una criptomoneda para asegurar transacciones y consenso, por lo cual el costo de las transacciones se reduce al costo de procesamiento en la red que requiere cada transacción.

Junto con los ya mencionados, Hyperledger Sawtooth provee una cantidad de mecanismos adicionales que distinguen la implementación de Hyperledger Fabric.

Las transacciones se pueden enviar en lotes: se deben ejecutar todas las transacciones de forma exitosa o ninguna. Así se consigue implementar transacciones atómicas como en bases de datos tradicionales.

Los nodos procesadores de transacciones son capaces de aprovechar paralelismo de transacciones no relacionados, mejorando así el rendimiento del Blockchain.

Hyperledger Sawtooth es compatible con Solidity, es decir *smart contracts* escritos para Ethereum pueden ser ejecutados en Hyperledger Sawtooth con la ayuda de la herramienta Seth. Además, pueden ser testeados en *sandboxes*, ambientes seguros para evitar comportamientos inesperados.

La configuración del Blockchain está guardada en forma de transacciones en el mismo Blockchain, lo cual tiene la ventaja que nodos que son agregados adicionalmente se pueden auto configurar. Además, posibilita modificar el funcionamiento de un Blockchain que ya está en uso: se proponen cambios en forma de transacciones que son aprobados o rechazados en base a un sistema de votos integrado en Hyperledger Sawtooth.

### 2.8.3. Otros proyectos de Hyperledger

**Hyperledger Iroha:** Una implementación con un esquema de consenso nuevo que es de alta performance y orientado a dispositivos móviles que se conectan y desconectan con frecuencia de la red. El proyecto todavía está en la versión beta.

**Hyperledger Burrow:** Es un cliente Blockchain con la máquina virtual de Ethereum integrada. Está diseñado para ejecutar *smart contracts* escritos en Solidity y, para ser usado en un ambiente donde diferentes Blockchains están conectadas entre ellos. Se encuentra en la etapa de incubación.

**Hyperledger Indy:** Un *framework* para un Blockchain diseñado para la gestión de identidades digitales. Como Hyperledger Burrow, está en la etapa de incubación.

Otros productos de Hyperledger están pensados para facilitar el desarrollo, como por ejemplo **Hyperledger Cello**, una herramienta para el despliegue en *clusters* con el fin de ofrecer Blockchain como servicio. **Hyperledger Caliper** permite realizar *benchmarks* de Blockchains e **Hyperledger Explorer** posibilita administrar a un Blockchain a través de una interfaz web. Todas las herramientas de Hyperledger se encuentran en etapa de incubación.

Los dos proyectos que se detallaron en esta sección cuentan con las características especificadas para el prototipo. 2.2. Son *frameworks* completamente desarrollados que permiten implementar un Blockchain con las funcionalidades requeridas. De los dos, Hyperledger Fabric se destaca como la opción más conveniente debido a que está específicamente destinado a Blockchains privados y para el prototipo a emplear no se necesita la compatibilidad con Ethereum. Además, emplear *Proof of Elapsed Time* implica una limitación con respecto al *hardware*, ya que el algoritmo solamente funciona en conjunto con arquitectura Intel, por lo cual la mecánica de consenso de Hyperledger Fabric resulta más adecuada.

## 2.9. Conclusiones

En el mercado existen muchas soluciones para implementar aplicaciones en Blockchain, algunas más desarrolladas que otras. Cada solución tiene su propio enfoque y en el momento de la redacción del informe, la mayoría requieren de un medio de pago en forma criptomonedas. Sin embargo, el código de dichas plataformas suele ser libre y *open source*, por lo cual un equipo de programadores experimentados puede implementar un Blockchain propio haciendo las modificaciones deseadas en un código ya existente.

Para un despliegue de mayor velocidad con un *framework* ya existente, se ofrecen las soluciones de Hyperledger, donde especialmente Hyperledger Fabric e Hyperledger Sawtooth cuentan con un código testado para ser empleado en ambientes de producción, el soporte de varios lenguajes de programación para aplicaciones y una amplia cantidad de documentación con tutoriales.

Hyperledger Fabric es una solución de código libre que cumple con las siguientes características:

- Es una solución pensada para Blockchains privados y Blockchains de Consorcio.
- Los permisos de acceso son configurables a través de identidades digitales, que también permiten la implementación de una lista negra para casos de identidades perdidas o robadas. En cada momento, es posible controlar los accesos de lectura y escritura de cualquier entidad.
- Si bien el uso de una criptomoneda es posible, el protocolo de consenso no requiere de un incentivo en forma de una criptomoneda.
- Los *smart contracts* son fácilmente programables en el lenguaje de programación Golang.

Dado que las características mencionadas cumplen con los requerimientos planteados para una solución Blockchain, elaborados en la sección 2.2, se eligió Hyperledger Fabric como *framework* para la implementación del prototipo.



## Capítulo 3

# Hyperledger Fabric

Para poder proceder con el diseño de un prototipo resulta necesario conocer los detalles de la implementación de la plataforma elegida. Por eso, el objetivo del presente capítulo es adquirir un profundo conocimiento de terminología, diseño y funcionamiento de Hyperledger Fabric.

### 3.1. Composición de la red Blockchain

Como también en otras implementaciones ya estudiadas, la red se compone por un conjunto de nodos participantes o *peers*, que intercambian mensajes sobre el estado del Blockchain. *Peers* son los elementos más fundamentales de Hyperledger Fabric, ya que almacenan el *ledger*, o el Blockchain. Los términos *ledger* o Blockchain en este contexto describen el mismo concepto y se van a utilizar ambos términos de forma intercambiable, hasta aclarar la diferencia en la sección 3.7. Además, los *peers* ejecutan los *smart contracts*, que en Hyperledger Fabric se llaman *chaincodes*. Existen dos operaciones sobre el Blockchain: la de lectura y la de escritura. Ambas operaciones tienen que ser implementadas en el *chaincode*: la existencia de una instancia del *ledger* solo no permite leer o modificar información.

Una colección de *peers* cumple con las siguientes características:

- Cada *peer* que pertenece a la red, almacena una instancia del *ledger* y una instancia del *chaincode*.
- Al existir instancias iguales en todos los nodos, se evita un único punto de fallo: Cuando el *ledger* es modificado, cada *peer* actualiza su instancia.
- Ya que cada *peer* es capaz de acceder y modificar el Blockchain, una aplicación que necesita leer o modificar el *ledger*, lo tiene que hacer a través del *peer*.

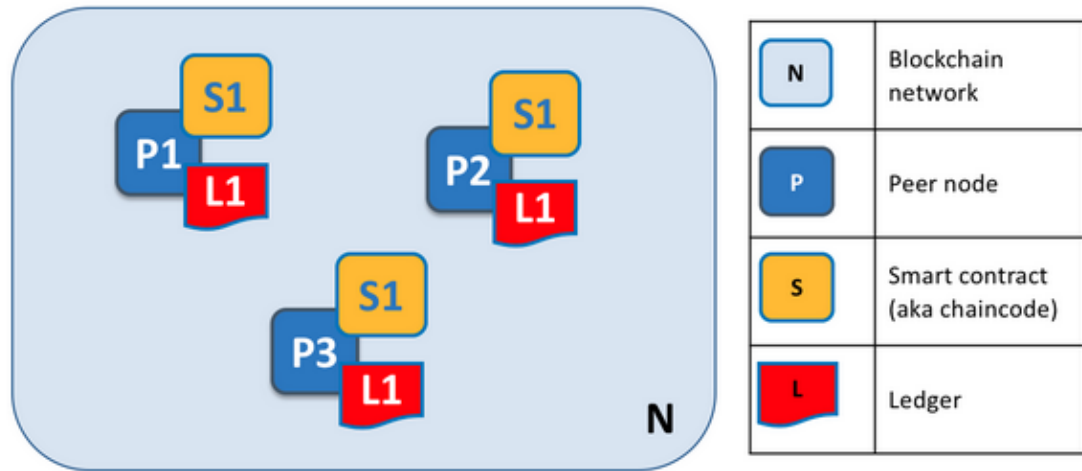


FIGURA 3.1: Una red Blockchain con 3 *peers*: Cada *peer* corre una instancia idéntica del *chaincode* y del *ledger*. De [3].

La figura 3.1 ilustra lo descrito: En una red se encuentran 3 *peers*, donde cada uno tiene una copia del *ledger* y del *chaincode*. *Peers* pueden ser agregados o eliminados de la red de forma dinámica, según las necesidades de los administradores. No existen limitaciones con respecto a la cantidad de *peers* que pueden conformar una red.

### 3.2. La existencia de varios *ledgers*

En la sección anterior se explicó que cada *peer* almacena su instancia del *ledger* y del *chaincode* y que los datos pueden ser actualizados solamente a través del *chaincode*. Una analogía sería permitir la lectura y escritura de una base de datos relacional solamente a través de llamadas a una API, en vez de ejecutar instrucciones SQL directamente.

Una particularidad de Hyperledger Fabric consiste en posibilitar la existencia de varios *ledgers*: Un *peer* puede almacenar diferentes Blockchains como también diferentes *chaincodes* que acceden al Blockchain correspondiente. La figura 3.2 visualiza la situación descrita: El *peer* P1 aloja *ledger* L1, al cual puede acceder con los *chaincodes* S1 y S2. A su vez, aloja *ledger* L2, accesible a través de S1 y S3.

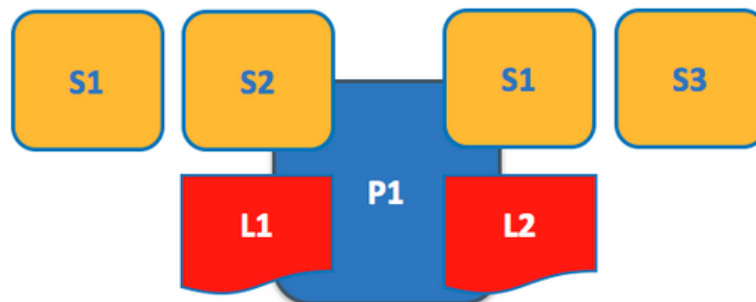


FIGURA 3.2: Un *peer* que aloja 2 *ledgers* y 3 diferentes *chaincodes*. De [3].

No existen límites con respecto a esa arquitectura: un peer puede alojar tantos *ledgers* y *chaincodes* como se desea. Tampoco hay una relación requerida entre la cantidad de *ledgers* y *chaincode*: un peer teóricamente puede alojar un *ledger* sin *chaincode* que acceda a él. Cada *ledger* almacena información independientemente de otros *ledgers* y funciona como un Blockchain diferente.

En un ambiente de Blockchains privados, el paradigma descrito es de mucha utilidad: Un conjunto de empresas puede decidir compartir información en un *ledger* al cual acceden todos los participantes, pero también pueden existir *ledgers* entre un subconjunto de participantes, permitiendo un grado de privacidad mayor. En la figura 3.3 se muestra el escenario descrito.

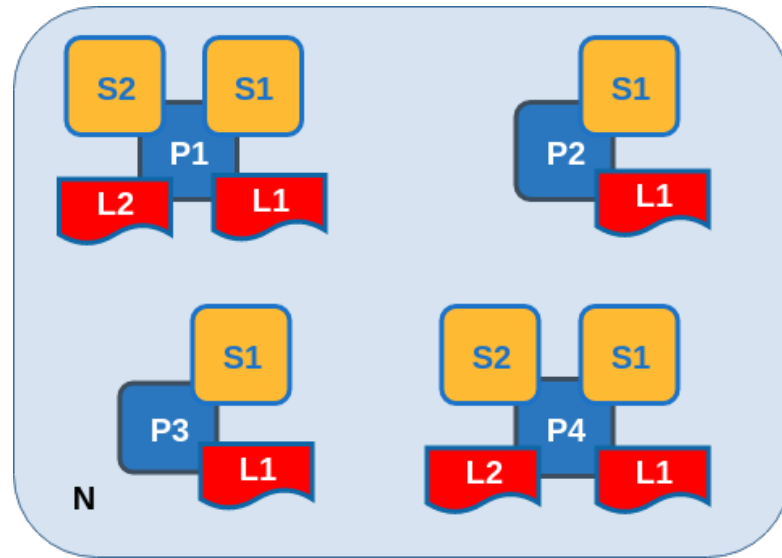


FIGURA 3.3: Una red con 2 *ledgers* diferentes: *peers* P2 y P3 no saben de la existencia del *ledger* L2, sin embargo, todos los *peers* comparten información en el *ledger* L1.

Recordando las implementaciones analizadas en el capítulo 2, por cada red de participantes existía un sólo Blockchain que almacenaba todas las transacciones efectuadas de la red: La arquitectura de Hyperledger Fabric permite mantener una multitud de Blockchains en una misma red, donde los *peers* solamente tienen conocimiento de los *ledgers* que alojan.

### 3.3. Aplicaciones y la red Blockchain

Como se mencionó en la sección 3.1, es necesario que aplicaciones accedan al Blockchain estableciendo una conexión con los *peers*. Para lograr eso, Hyperledger Fabric provee SDKs en los lenguajes Java, Python, Go y Nodejs. Programadores pueden utilizar las funciones del SDK para conectarse a la API expuesta por cada *peer* e invocar *chaincode*.

Una aplicación se puede conectar a un *peer* para consultar información existente en el Blockchain o para actualizar su estado. El flujo de cada una de las operaciones se ve detallado en la figura 3.4.

Para el caso de una consulta se requieren 3 pasos:

1. La aplicación se conecta al *peer* correspondiente. Qué aplicación se conecta a qué *peer* de la red es indiferente, siempre y cuando el *peer* tenga conocimiento del *ledger* y *chaincode* que se está por acceder.
2. Luego de una conexión exitosa, la aplicación envía una propuesta al *peer*, que invoca el *chaincode* correspondiente.
3. El *peer* espera la respuesta del *chaincode* invocado y la retorna al *peer*.

Como el *peer* mantiene una instancia completa y actualizada del *ledger* y del *chaincode*, no es necesario que verifique sus resultados con otros participantes de la red: puede satisfacer la consulta solo y de forma inmediata. La característica descrita garantiza la alta disponibilidad del Blockchain: La carga de muchas operaciones de lectura puede ser distribuida entre una multitud de *peers* y la caída de un *peer* no afecta la disponibilidad de la información.

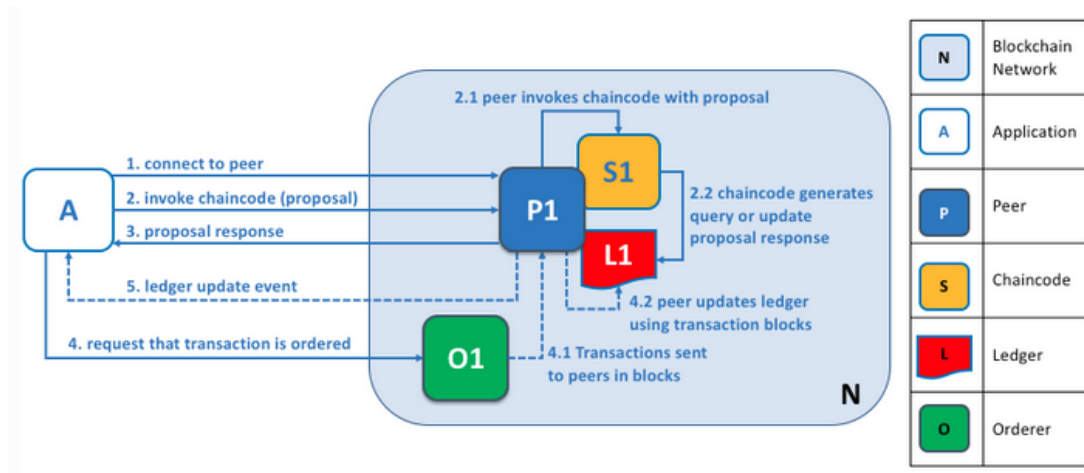


FIGURA 3.4: Flujos de consulta y actualización del *ledger*. De [3].

Una actualización del *ledger* requiere el consenso de una multitud de *peers*, por lo cual se requieren más pasos:

1. Cada *chaincode* posee su propia política de aprobación que especifica cuantos *peers* tienen que aprobar la transacción para que ésta puede ser agregada al Blockchain. Cuando una aplicación quiere escribir datos o cambios, envía una propuesta a todos los *peers* necesarios para que la transacción pueda ser aprobada.
2. Cada *peer* determina que la propuesta sea válida, que la misma propuesta no se ha hecho anteriormente (protección contra un ataques de repetición) y que la aplicación tenga los permisos suficientes para realizar una escritura en el *ledger*. Si se cumplen todas las condiciones, el *chaincode* es ejecutado por todos los *peers* que lo recibieron y los resultados se envían en forma de una respuesta a la aplicación. Es importante entender que en el momento que la aplicación recibe las respuestas, todavía no se cambió el estado del Blockchain.

3. La aplicación recibe las respuestas, verifica que todas sean iguales y si es el caso, manda su propuesta junto con las respuestas obtenidas al “servicio para ordenar transacciones” (*ordering service*) para que la transacción sea agregada al *ledger*.
4. Un *peer* especial llamado *orderer* cumple con la función de recibir diferentes transacciones, ordenarlas cronológicamente en bloques de datos y publicarlas en la red.
5. Una vez que se emitió el bloque nuevo a todos los *peers* suscritos al *ledger* correspondiente, los *peers* verifican que la transacción en sí es válida y que se cumplió con la política de aprobación requerida. Luego, la transacción se declara como válida o inválida.
6. Al completar el proceso descrito, la aplicación cliente es informada sobre la validez de la transacción y sobre su inclusión en el Blockchain. Debido al procesamiento requerido, la respuesta puede demorar varios segundos.

Vale aclarar que una “actualización” en este contexto no sobre escribe valores existentes: como en todas las implementaciones Blockchain, el *ledger* es inmutable y no soporta operaciones de borrado o sobre escritura. “Actualizar” al Blockchain entonces significa simplemente agregar información nueva.

### 3.4. Canales

En la sección anterior, se explicó que un bloque nuevo es emitido a todos los *peers* suscritos a un *ledger*. A continuación se va a explicar cómo funciona dicha subscripción. En Hyperledger Fabric se usa un concepto llamado “canales”.

Un canal describe un conjunto de *peers*, un servicio de pedidos y un *ledger* con su respectivo *chaincode*. Todos los *peers*, los nodos que se encargan del servicio de pedidos y las aplicaciones que se unen a un canal van a tener acceso a copias idénticas del mismo *ledger*. Así, para que puedan existir varios *ledgers* diferentes en una red, tienen que existir varios canales y los participantes que quieren acceder a los *ledgers* en cuestión tienen que formar parte del canal correspondiente.

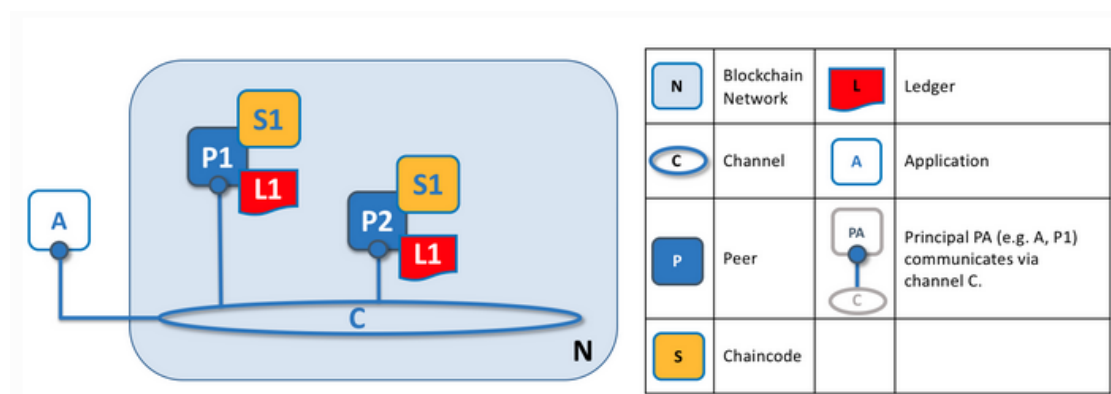


FIGURA 3.5: Visualización del concepto de canal. De [3].

### 3.5. Agrupación de la red en organizaciones

Tal como en Hyperledger Fabric se agrupan en canales los miembros de la red que acceden a un mismo *ledger*, también se agrupan los *peers* según la organización a la que pertenecen.

El concepto de organización en Hyperledger Fabric describe una entidad que participa en la red y que conecta sus *peers* a ella. La idea es que equivale a una organización en el mundo real: En el contexto del proyecto integrador, las organizaciones podrían ser las diferentes facultades que comparten información en una red Blockchain de la UNC.

Una organización puede aportar tantos *peers* a la red como desee. En la figura 3.6 la red está conformada por cuatro organizaciones y cada una tiene como mínimo un *peer* conectado a un canal en común. También se puede observar que cada organización tiene su propia aplicación para acceder a los *peers*: El desacoplado entre aplicación de usuario y *chaincode* permite la implementación de una multitud de aplicaciones que consumen los datos compartidos de un mismo Blockchain.

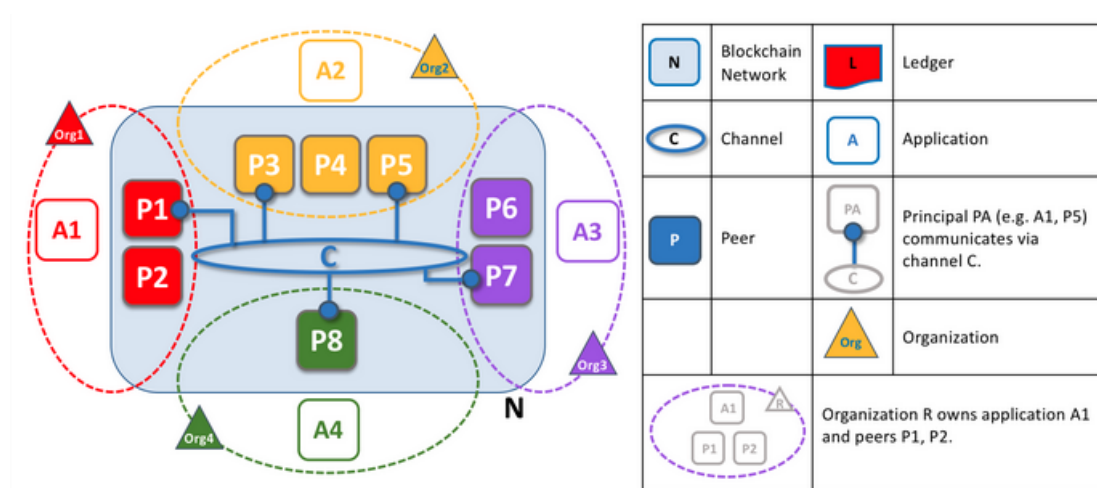


FIGURA 3.6: Una red Blockchain conformada por 4 organizaciones.

### 3.6. La Identidad de los *peers* en la red Blockchain

¿Cómo reconoce la red qué *peer* pertenece a qué organización? En Hyperledger Fabric, cada organización administra su propia Autoridad de Certificación (CA por sus siglas en inglés), que provee una identidad digital en forma de un certificado X.509 para cada *peer* que se une a la red. La figura 3.7 describe un esquema de dos organizaciones con sus autoridades de certificación correspondientes.

Pero los certificados no solamente son para *peers*: cualquier entidad que se quiere conectar a la red, también las aplicaciones, necesitan una identidad digital autorizada, sino su conexión será rechazada.

Si bien un *peer* puede tener acceso a varios *ledgers* compartidos entre diferentes organizaciones, siempre pertenece a una sola organización, la cual se determina con su identidad digital. Un *peer* no puede ser compartido entre múltiples organizaciones.

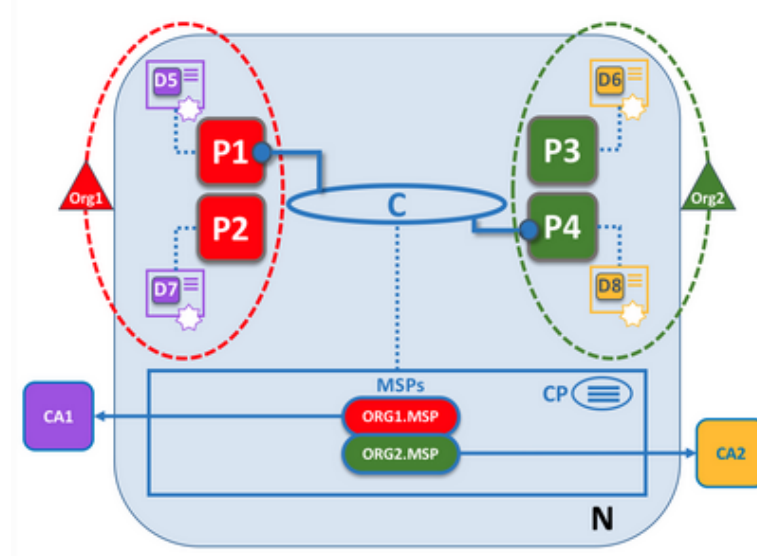


FIGURA 3.7: Una red compuesta de dos organizaciones, sus respectivas autoridades de certificación y sus MSP. Cada *peer* tiene una identidad digital asociada, *peer* P1 por ejemplo se identifica en el canal con la identidad D5. De [3].

Los flujos de autenticación y autorización en Hyperledger Fabric son manejados por un *Membership Service Provider* o MSP. Éste trabaja a dos niveles:

- El *Membership Service Provider* a nivel de canal identifica a qué organización pertenece cada *peer* y controla que la participación y la administración del Blockchain solamente puede realizarse por los *peers* con los derechos correspondientes.
- El *Membership Service Provider* a nivel local está funcionando adentro de cada *peer*: Su objetivo es verificar los derechos de acceso de lectura y escritura de los clientes o aplicaciones que se conectan a él con el objetivo de invocar *chaincode*.

La comunicación entre los diferentes *peers* funciona a través de un protocolo *gossip*, pero para que todos los participantes de un canal empiecen a compartir información, es necesario que cada miembro tenga conocimiento de los demás miembros de la red. Para el descubrimiento de *peers* en un canal se define un mínimo de un *anchor peer* por canal: *peers* de todas las organizaciones notifican al *anchor peer* sobre los miembros que pertenecen a su organización y reciben información sobre otros *peers* de otras organizaciones como respuesta. Luego de la etapa de descubrimiento, todos los *peers* del canal pueden comunicarse entre ellos y el *anchor peer* solamente se va a volver a necesitar cuando se agreguen nodos a la red o se apagan. Se recomienda que cada organización tenga su conjunto de *anchor peers* así se evita un único punto de fallos.

### 3.7. Los *Ledgers* y el *World State*

En la presente sección, se van a definir con mayor detalle los términos Blockchain y *ledger* bajo el contexto de Hyperledger Fabric.

*Ledger* es un término del inglés cuya traducción directa es “libro contable”. Describe un registro que contiene el estado actual de un negocio junto con un historial de transacciones que llevaron a dicho estado.

En un *ledger* se almacenan datos respecto a un cierto negocio. Los datos se pueden ver como objetos del negocio, aunque solamente son abstracciones de dichos objetos. En el caso de uso presentado, un objeto sería un acta de una facultad y el estado del acta puede cambiar de válido a inválido en el caso de ser revocado.

En Hyperledger Fabric, el registro histórico que lleva al estado actual es el Blockchain. Está estructurado como un registro secuencial de bloques interconectados, donde cada bloque contiene una secuencia de transacciones, cada transacción representa una consulta o actualización del estado de los objetos almacenados. La figura 3.8 representa la estructura en forma gráfica: se puede observar que no existen diferencias fundamentales con otras implementaciones como Bitcoin.

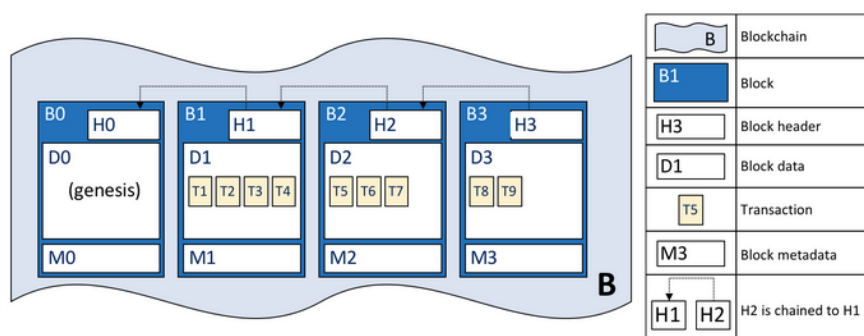
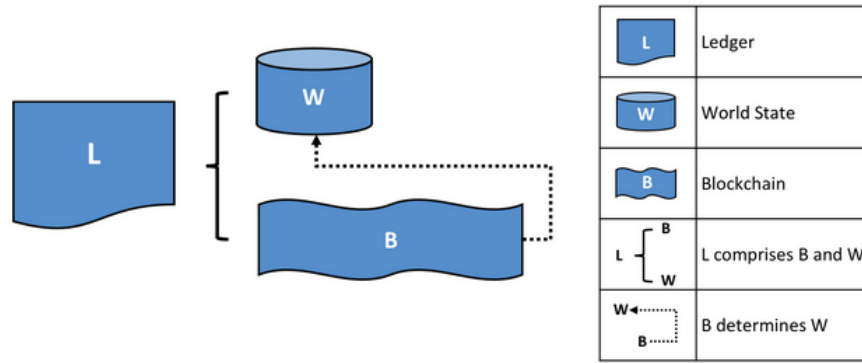


FIGURA 3.8: La estructura del Blockchain en Hyperledger Fabric. De [4].

El Blockchain se almacena como un solo archivo y la operación más frecuente que se efectúa sobre éste es adjuntar bloques adicionales. Como el archivo no se encuentra indexado, consultas sobre el estado actual implican iterar de forma secuencial sobre los bloques almacenados, lo cual resulta ineficiente. Para resolver ese problema, Hyperledger Fabric extrae el estado actual del Blockchain y lo guarda en una base de datos. El conjunto de los estados actuales de todos los objetos lleva el nombre *world state*.

Para el *world state*, Hyperledger Fabric ofrece dos opciones de base de datos diferentes: La primera es LevelDB, un simple almacenamiento para datos en el formato clave-valor. Para almacenar objetos JSON con mayor complejidad, se recomienda CouchDB. Consultas en CouchDB se pueden hacer sobre claves y valores, no solamente sobre claves como en LevelDB. Además, CouchDB permite actualizar solamente partes del objeto JSON, mientras en LevelDB la actualización afecta al objeto entero.



FIGURA 3.9: *Ledger*, *World State* y *Blockchain* en contexto. De [4].

En el contexto de Hyperledger Fabric, el conjunto de *world state* y *Blockchain* reciben el nombre *ledger*, como se ve en el gráfico 3.9. Al extraer el *world state* del *Blockchain* y almacenarlo en una base de datos, se permite la indexación del estado actual y un alto rendimiento a la hora de realizar consultas.

### 3.8. El servicio para ordenar transacciones

En el capítulo 2 se explicó la importancia de un algoritmo de consenso y el funcionamiento de *Proof of Work*, el algoritmo utilizado por una gran cantidad de implementaciones. Luego, junto con otras plataformas, se investigaron alternativas que surgieron con el objetivo de disminuir el consumo energético del algoritmo de consenso o de mejorar su rendimiento.

Los algoritmos de consenso, en su mayoría, tienen en común la ejecución de *smart contracts*, la validación de transacciones y, la formación de bloques ejecutados por un mismo *peer*. Esto significa un cuello de botella para el procesamiento de transacciones.

Hyperledger Fabric evita el problema descrito al distribuir ejecución, validación y formación de bloques entre diferentes participantes de la red: La ejecución del *chaincode* se realiza en los *peers*, la formación de bloques nuevos es realizada por *peers* especiales llamados *orderer nodes* o nodos de ordenamiento y la validación es efectuada nuevamente por los *peers*, una vez que el bloque nuevo se distribuyó en la red.

Como los *peers*, los *orderer nodes* pertenecen a una organización y obtienen sus credenciales criptográficas de una autoridad de certificación de dicha organización. Para la creación de un bloque nuevo, es posible establecer 2 criterios diferentes:

1. El primer criterio es el tiempo: Los *orderer nodes* pueden ser configurados para generar un bloque nuevo cada intervalo de tiempo, siempre y cuando haya aunque sea una transacción esperando a ser procesada.
2. El segundo criterio es el tamaño del bloque: Un bloque nuevo se agrega cuando se acumuló una cierta cantidad de transacciones que llena un bloque completo.

3. Es posible combinar ambos criterios: Así, en momentos donde se efectúan pocas transacciones, se generan bloques más chicos cada cierto intervalo de tiempo y en momentos donde las cantidades de transacciones son altas, bloques nuevos son generados con mayor frecuencia y un tamaño máximo.

Es posible modificar los parámetros mencionados cuando ya se generó un *ledger*, para optimizar la performance de las escrituras.

Hyperledger Fabric provee tres servicios diferentes para la generación y el ordenamiento de bloques: con cualquiera de los tres, el Blockchain va a formarse de la misma forma, pero difieren en su trabajo de configuración y su funcionamiento interno.

- **Solo:** Consiste de un solo *peer* generador de bloques. No provee redundancia o protección a fallos. Su uso está pensado para tests o pruebas de conceptos.
- **Raft:** Un servicio de ordenamiento compuesto por un conjunto de *orderer nodes* tolerante a fallos que implementa el protocolo Raft de la herramienta etcd.[\[34\]](#)
- **Kafka:** Un servicio similar a Raft, tolerante a fallos, que utiliza Zookeeper para la gestión del *cluster*. Su configuración y administración suelen ser más dificultosas que con Raft.

### 3.9. Conclusión

Las propiedades detalladas de Hyperledger Fabric reflejan las necesidades planteadas en el capítulo 2 para una solución que consta de un Blockchain privado:

- Los *peers* y los *ledgers* proveen el carácter distribuido también conocido de otras implementaciones.
- La posibilidad de la existencia de diferentes *ledgers* permite mayor privacidad entre los participantes que si existiera un sólo *ledger*. Su gestión es facilitada con canales.
- *Peers* pertenecen a organizaciones y sus accesos se establecen a través de identidades digitales, lo cual facilita la administración de permisos de lectura y escritura.
- La división entre *ledger* y *world state* mejora el rendimiento de lecturas: el hecho de que el *world state* se encuentra indexado, elimina la necesidad de iterar sobre los bloques del *ledger* para consultar el último estado de la información.

## Capítulo 4

# Análisis y Diseño del Sistema

Luego de haber adquirido una amplia cantidad de conocimiento sobre la tecnología Blockchain en general y sobre Hyperledger Fabric en particular, el presente capítulo investiga los problemas que tiene la UNC con la gestión y validación de información académica. Para eso se realizó una entrevista con el Ingeniero Miguel Montes, Prosecretario de Informática de la UNC. Luego se procede a analizar si las soluciones propuestas a las cuales se llegaron en la entrevista representan un caso de uso de Blockchain. Por último, se especifican Componentes, Requerimientos y Comportamiento del sistema a implementar.

### 4.1. Entrevista

La entrevista fue conducida con el objetivo de validar las primeras especificaciones para el prototipo planteado en la sección 2.2 y para lograr una perspicacia mejor de los problemas del sistema de gestión de alumnos Guaraní que podrían ser resueltos a través de un Blockchain. Con respecto al sistema Guaraní, el Prosecretario Ing. Montes planteó los problemas siguientes:

- El primer problema es un problema de digitalización: Las actas actualmente se imprimen y se firman en papel por los profesores autorizados, luego es necesario almacenarlas. Se podría implementar un equivalente digital que se firma digitalmente. Eso presenta el problema de la firma en sí: Cada profesor debe tener su clave, por ejemplo en forma de un token de hardware, lo cual involucra varios problemas: En primer lugar, significa un alto costo de adquisición y luego se requiere administrar dicho recurso, ya que en el caso de daño o de pérdida, el token tiene que ser reemplazado. A su vez, no debe ser posible aceptar firmas realizadas con el token anterior, es decir, se debe mantener una lista negra para claves extraviadas. Además, los profesores deben ser capacitados para poder realizar las firmas correctamente.
- El segundo problema afecta la integridad y la disponibilidad de las actas: Actualmente, en el momento de un examen, el profesor es el responsable de otorgar una

nota correspondiente al desempeño del alumno. Eso queda asentado, ya que ambos, alumnos como docentes, firman el acta. Sin embargo, no existe la misma responsabilidad en el caso del administrador de la base de datos: El encargado de subir la nota a la base de datos del sistema Guaraní podría modificar la información sin levantar sospecha. Un acta digitalmente firmada no podría modificarse sin que la firma pierda su validez, pero si un acta tiene un error y es rectificado, permanece con una firma válida. Aquí es donde aparece el problema de disponibilidad: Frente a la desaparición del acta rectificadora, queda el acta rectificada con una firma válida y no sería posible detectar si el acta fue rectificada o no.

- En el caso del tercer problema se trata de interoperabilidad: Al principio, la adopción puede ser difícil, ya que todas las entidades que trabajan con los documentos en cuestión, tienen que cambiar a un sistema digital. Y si la distribución de dichos documentos se hiciera a través de un Blockchain, se originaría un problema legal - la información que se genera entre Universidad y alumno por ley no puede ser difundida a terceros.

Las soluciones propuestas en la entrevista fueron las siguientes:

- Una de las propiedades características del Blockchain es su carácter inmutable cuando es lo suficientemente distribuido y bajo de control de una multitud de organismos. En el caso de utilizar Blockchain, los ataques contra la integridad y disponibilidad de las actas descritas anteriormente no serían posibles, ya que quedaría un registro inmutable de todas las actas y las actas rectificadoras.
- Para resolver el problema de la privacidad de la información, se podría aplicar una función *hash* a los datos en cuestión. Así, el Blockchain no sería un mecanismo de distribución de información, pero se podrían implementar *smart contracts*, que validen certificados otorgados contra el Blockchain.
- Siempre y cuando los demás organismos acepten los certificados emitidos y sean capaces de validarlos contra el Blockchain, se resuelve el problema de interoperabilidad. El usuario, que en este caso es el alumno, mantiene control sobre sus datos: él es responsable de presentar lo que corresponda ante el organismo en cuestión, y dicho organismo puede verificar la validez de lo presentado inmediatamente a través del Blockchain, sin embargo, sin los datos originales, no tiene acceso a ningún tipo de información privada.
- El problema de la digitalización tiene que ser resuelto por cada facultad: Una opción propuesta por Montes consta de firmar las actas en papel en el momento del examen, para que luego sean escaneadas por una persona dedicada que certifica con una firma digital que las notas del acta original coinciden con las notas del acta digital. Un *hash* del acta completo se agregaría al Blockchain. De este modo, la facultad evita la compra de una gran cantidad de tokens, ya que se requiere uno solo para la persona que escanea las actas. Además, se evita cambiar el modo de trabajo de muchos profesores a la hora de evaluar un examen.

## 4.2. Casos de uso de Blockchain

Desde el éxito de Bitcoin y el nacimiento de numerosas otras criptomonedas y plataformas Blockchain, existe mucha especulación sobre el impacto que va a tener la tecnología una vez que esté madura y adoptada. Para nombrar solo un ejemplo, *The Economist* publicó una edición el 31 de octubre de 2015 en cuya portada figuraba el título “La máquina de la confianza - cómo la tecnología detrás de Bitcoin podría cambiar al mundo.” Gideon Greenspan, Fundador de la plataforma Multichain, describe el 22 de Noviembre del mismo año en un artículo con el nombre “Evitar el proyecto Blockchain inútil” que “Estamos viendo un número cada vez mayor de compañías que crean pruebas de concepto en nuestra plataforma y / o piden nuestra ayuda.”

Para aclarar la confusión insiste en que no todos los casos de uso son adecuados para Blockchain: “Si los requisitos se cumplen con usar las bases de datos relacionales de hoy, estarías loco si usas un Blockchain.” Con el argumento que bases de datos relacionales tienen décadas de desarrollo y soportan miles de consultas por segundo, explica una serie de criterios con los cuales identificar un caso de uso Blockchain.

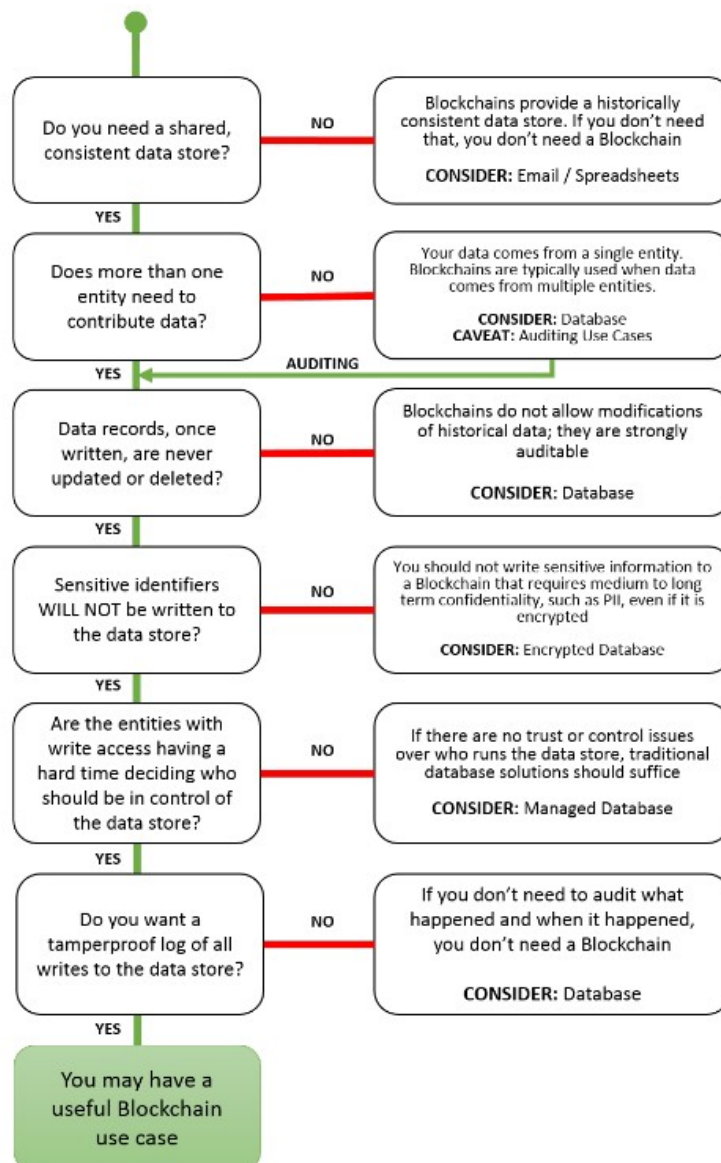


FIGURA 4.1: Diagrama de flujo para determinar un caso de uso Blockchain. De [5].

Debido a las confusiones sobre la utilidad de la tecnología Blockchain en una gran cantidad de escenarios, existe una variedad de literatura que describe como determinar un caso de uso Blockchain. Entre otros, el Instituto Nacional de Estándares y Tecnología de Estados Unidos elaboró un diagrama de flujo que ayuda a descartar fácilmente ideas que son mejor implementadas en bases de datos tradicionales.

A continuación se va a analizar el caso de uso planteado con la ayuda de la figura 4.1 para determinar si la implementación de un sistema de validación de información académica se beneficia de utilizar tecnología Blockchain o no.

1. **¿Se necesita almacenar información de forma compartida y consistente?**  
Sí, ya que el prototipo apunta a permitir la validación de credenciales académicas por una variedad de entidades, que pueden ser diferentes facultades, Universidades

o cualquier otro organismo que necesita determinar que una nota o una historia académica sea válida.

2. **¿Es necesario que más que una entidad aporte datos?** Planteando el caso que la UNC reemplaza actas comunes por actas digitales, todas las facultades estarían aportando información. También es posible extender el caso de uso a varias universidades: La validación de credenciales de alumnos que cambian de universidad o quieren realizar un postgrado es facilitada si una multitud de universidades agrega datos.
3. **¿No es necesario modificar o eliminar información una vez que fue escrita?** Para el caso del prototipo planteado, la modificación o eliminación de información no es deseada. En su lugar, se desea un historial completo inmutable de lo ocurrido: al revocar un acta, por ejemplo, se mantiene el acta original junto con su acta rectificadora.
4. **¿No se va a escribir información sensible o personal en el ledger?** A toda información se va a aplicar un *hash* antes de ser agregada al Blockchain, por ende no se va a escribir nunca información en texto plano.
5. **¿Las entidades con acceso de escritura tienen dificultades para decidir quién debe tener el control del almacén de datos?** No existe ningún proyecto para la implementación de un sistema de validación de credenciales académicas centralizado, pero la arquitectura distribuida del Blockchain implica que cada facultad (o en un futuro, Universidad) podría sumarse con pocos recursos y en el momento que sea conveniente.
6. **¿Desea un registro inviolable de todas las escrituras en el almacén de datos?** Sí, es crucial que la información sea imposible de alterar.

Luego del análisis efectuado, es posible concluir que el uso de Blockchain para el caso planteado tiene mayores beneficios sobre una base de datos tradicional.

### 4.3. Funcionalidades

Luego de la entrevista y de la evaluación del caso de uso, se consideran necesarias las siguientes funcionalidades para el prototipo:

- El prototipo debe consistir de un Blockchain con un mínimo de dos organizaciones con dos *peers* cada uno, que comparten información a través de un canal.
- Por ende, todos los *peers* deben poder acceder al mismo conjunto de información y el mismo *chaincode*.
- El prototipo debe ser capaz que recibir credenciales académicas en texto plano, aplicar una función *hash* a los datos e invocar el *chaincode* correspondiente para que dicha información sea agregada al Blockchain.

- Se debe proporcionar la funcionalidad de verificar la existencia de una cierta porción de información: Para validar una nota por ejemplo, se debe poder ingresar la información correspondiente, el prototipo debe aplicar una función *hash* a ella y buscar una coincidencia en el Blockchain para confirmar la validez.
- Es necesario proveer la posibilidad de corregir información: Si se requiere revocar un acta, tiene que ser posible agregar su acta rectificativa. Luego el acta original debe ser considerada inválida y ya no debe ser utilizada para validaciones.

#### 4.4. Definición de los Componentes

Con el fin de implementar las funcionalidades descritas en la sección 4.3, en primer lugar se definen los componentes del prototipo en su totalidad.

1. **La red Blockchain:** Se requiere una red de Hyperledger Fabric, compuesta por organizaciones que representan facultades y *peers* que comparten información a través de un canal. El prototipo va a simular tres facultades generando tres organizaciones, cada una con dos *peers*. De esa forma existe redundancia para cada organización: si un *peer* falla, el otro sigue disponible. Además, cada organización tiene que tener una Autoridad de Certificación para otorgar identidades digitales. Va a existir un solo canal al cual las tres organizaciones tienen accesos y por ende un solo *ledger*.
2. **La base de datos para el *world state*:** El prototipo a implementar maneja datos a los cuales se aplicó una función de *hash* antes de su almacenamiento. Utilizar el idioma de CouchDB para realizar consultas complejas sobre el estado del *ledger* por ende no es posible. Por eso se eligió LevelDB como la base de datos para almacenar el *world state*.
3. **El *ordering service*:** Dada la complejidad de configurar Raft o Kafka como servicio de ordenamiento para Hyperledger Fabric se optó por Solo, el servicio de ordenamiento por defecto propuesto para test o pruebas de concepto. Si bien no es recomendado su uso en entornos de producción, cumple con todo lo requerido para el prototipo en cuestión.
4. **Elección del lenguaje de programación para el Chaincode:** Las posibilidades para escribir *chaincode* en Hyperledger Fabric son Go, Node.js o Java. Luego de varias pruebas con los 3 lenguajes, la decisión fue tomada a favor de Go: Con Node.js como Java la instanciación del *chaincode* provocaba repetidos errores de tiempo de espera debido a *bugs* en la implementación de Hyperledger Fabric.
5. **Elección del SDK:** Hyperledger Fabric provee SDKs en los lenguajes Node.js, Java, Python y Go, donde los últimos dos todavía no fueron lanzados oficialmente.<sup>[35]</sup> Para el proyecto se utilizó el SDK en Node.js, ya que la documentación del mismo está muy completa y debido a que el código desarrollado puede ser fácilmente agregado a un *framework* para aplicaciones web.



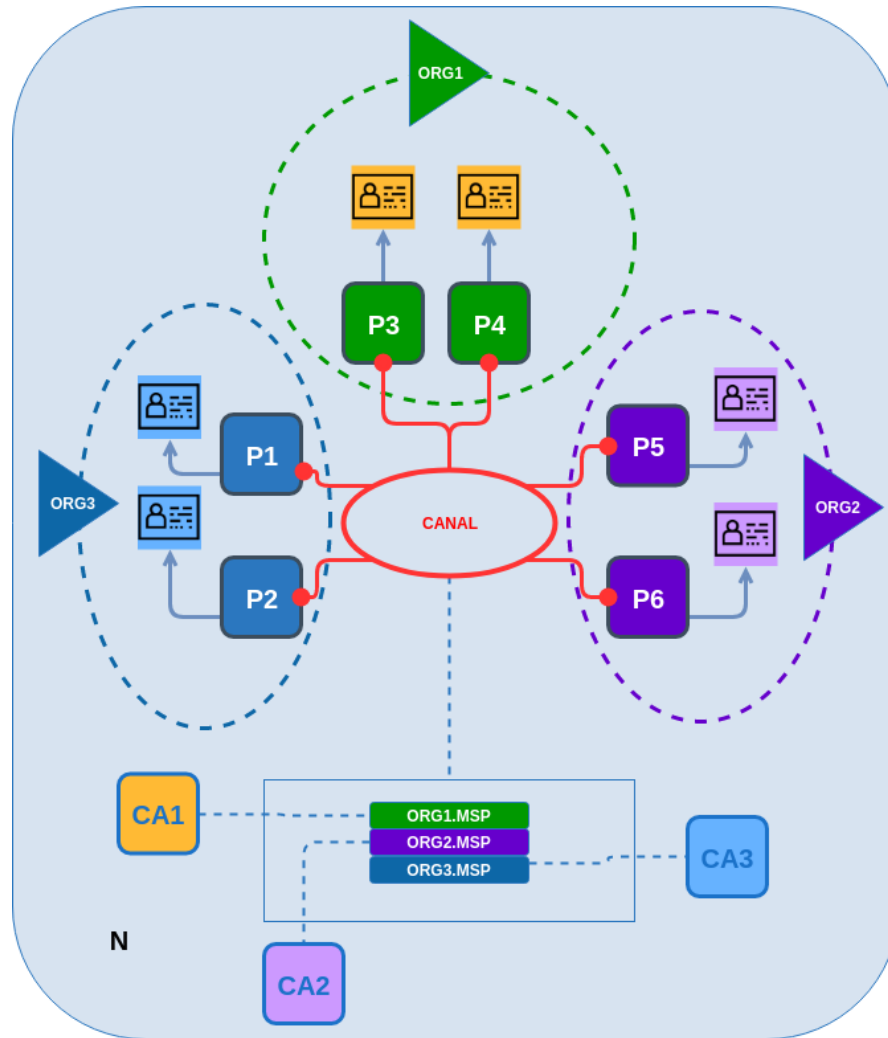


FIGURA 4.2: Diagrama de la red Blockchain a implementar.

6. **Desarrollo de una API:** El código escrito en Node.js es el código cliente que invoca al *chaincode* a través de los *peers*. Para que sea accesible a través de Internet es necesario el desarrollo de una API REST, que permita llamar las funciones con métodos HTML. La herramienta elegida para esa tarea es Express, un *framework* minimalista y flexible para aplicaciones web y APIs. Está escrito en Javascript, de modo que la integración de las funciones escritas con el SDK se puede realizar sin inconvenientes.

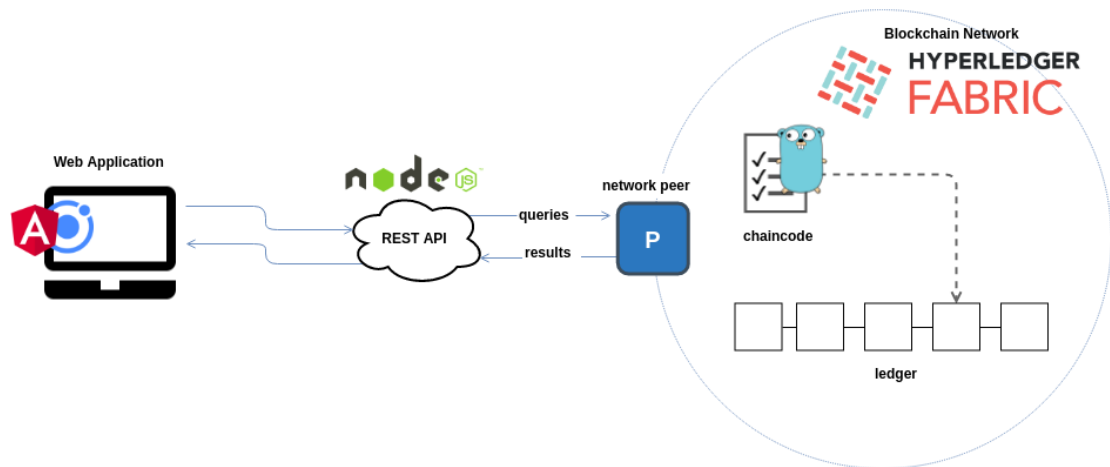


FIGURA 4.3: Diagrama general de la arquitectura junto con las tecnologías.

7. **Desarrollo de una GUI:** La API públicamente expuesta a Internet necesita un *frontend*, o una interfaz web, que acceda a ella. Su funcionalidad consiste en obtener información de la API y mostrarla de forma entendible para el usuario, como también obtener datos ingresados por el usuario y enviarlos para su procesamiento a la API.

Para lograr eso, se decidió utilizar Angular y Ionic: Angular es un *framework* mantenido por Google para el desarrollo de *Single Application Pages (SPAs)*, páginas que adaptan su contenido de forma dinámica a medida que reciben entradas del usuario, en vez de descargar páginas nuevas de un servidor.

Ionic permite agregar componentes web comúnmente usados a una aplicación escrita en Angular y agiliza así el desarrollo. A su vez viene con librerías como Apache Cordova, que permiten interactuar con dispositivos móviles. De esa forma, una misma aplicación puede ser compilada para un entorno web o un entorno móvil, dependiendo de las necesidades del desarrollador. La figura 4.4 posiciona a Ionic en el contexto de las herramientas para el desarrollo *frontend*.

FIGURA 4.4: El rol de Ionic en el desarrollo *frontend*. De [6].

## 4.5. Definición de los Requerimientos

Para especificar el funcionamiento general del sistema, se va a hacer uso del lenguaje de modelado UML. Si bien se trata del desarrollo de un sistema distribuido con una variedad de protocolos y tecnologías, el prototipo final es un producto de *software*, por lo cual un modelado en SysML no se consideró adecuado.

Para poder plantear los requerimientos, primero es necesario definir un diagrama de casos de uso, como se ve en la figura 4.5. En el diagrama se puede observar que el objetivo principal es brindar al personal administrativo un sistema de validación de información académica con las funcionalidades de agregar actas, revocarlas y validar la información contenida en las actas cargadas.

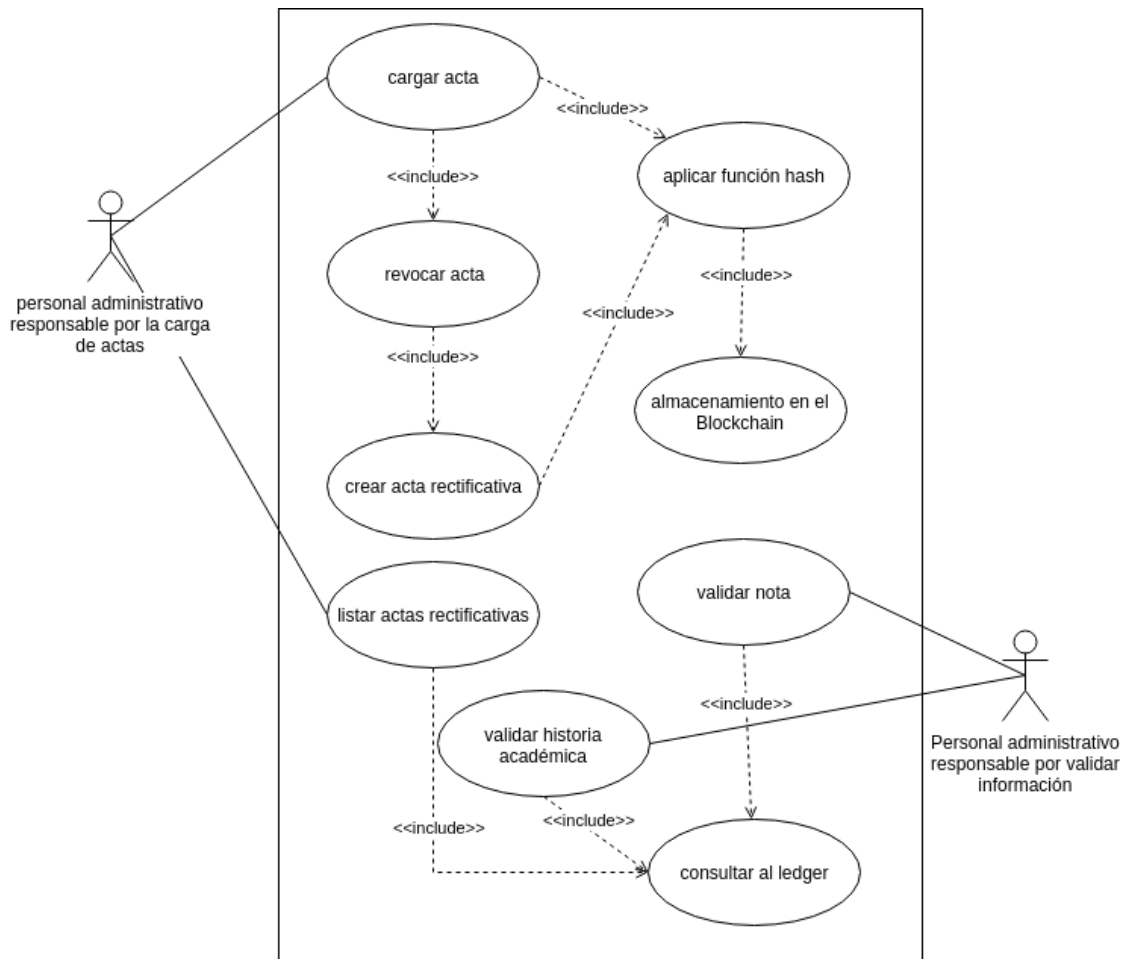


FIGURA 4.5: Diagrama de casos de uso.

Con el diagrama de caso de uso elaborado se pueden definir los siguientes requerimientos de usuario:

R-01	El personal administrativo debe poder cargar la información de un acta nueva al sistema.
R-02	Para poder verificar la presencia del acta cargada, debe ser posible consultar por el mismo a través de su id.
R-03	En caso de que se haya cometido un error, debe ser posible revocar el acta y crear su acta rectificativa.
R-04	Debe ser posible visualizar todas las actas rectificadas.
R-05	Debe ser posible validar una nota con el sistema: Se debe poder ingresar la información sobre la nota con la nota en texto plano y el sistema debe avisar si la nota se encontró en el sistema o no.
R-06	Debe ser posible validar la historia académica completa de un alumno de la misma forma que se puede validar una sola nota.

CUADRO 4.1: Requerimientos de usuario.

Luego es posible definir los requerimientos funcionales para cada uno de los componentes descritos anteriormente: la red Blockchain, la API y la interfaz gráfica. Se encuentran

en los cuadros 4.2 - 4.4. Los requerimientos no funcionales se definieron en la tabla 4.5.

RF-BC-01	La red debe ser compuesta por 3 organizaciones y dos <i>peers</i> por organización.
RF-BC-02	La red debe contar con un nodo que provee el servicio de ordenamiento.
RF-BC-03	Las organizaciones deben contar con una autoridad de certificación cada una para poder agregar usuarios de forma dinámica.
RF-BC-04	Los <i>peers</i> deben ser configurados para poder ejecutar <i>chaincode</i> en el lenguaje Golang.
RF-BC-05	El <i>chaincode</i> debe permitir escribir un objeto JSON nuevo en el <i>ledger</i> .
RF-BC-06	El <i>chaincode</i> debe permitir recuperar un objeto JSON teniendo la clave.

CUADRO 4.2: Requerimientos funcionales de la red de Hyperledger Fabric.

Para poder comprender mejor cuál es la relación estructural entre las diferentes tecnologías empleados en el prototipo, se ofrece un diagrama de componentes, realizado en la figura 4.6. Se distinguen 3 componentes principales: El *frontend*, la aplicación y la red de Hyperledger Fabric. La aplicación se divide en el código funcional de la aplicación en sí, que convierte, por ejemplo, las actas en texto plano a un *hash*, y el SDK de Hyperledger Fabric que brinda las librerías necesarias para invocar *chaincode* en los diferentes *peers* a través de una interfaz *shim*.

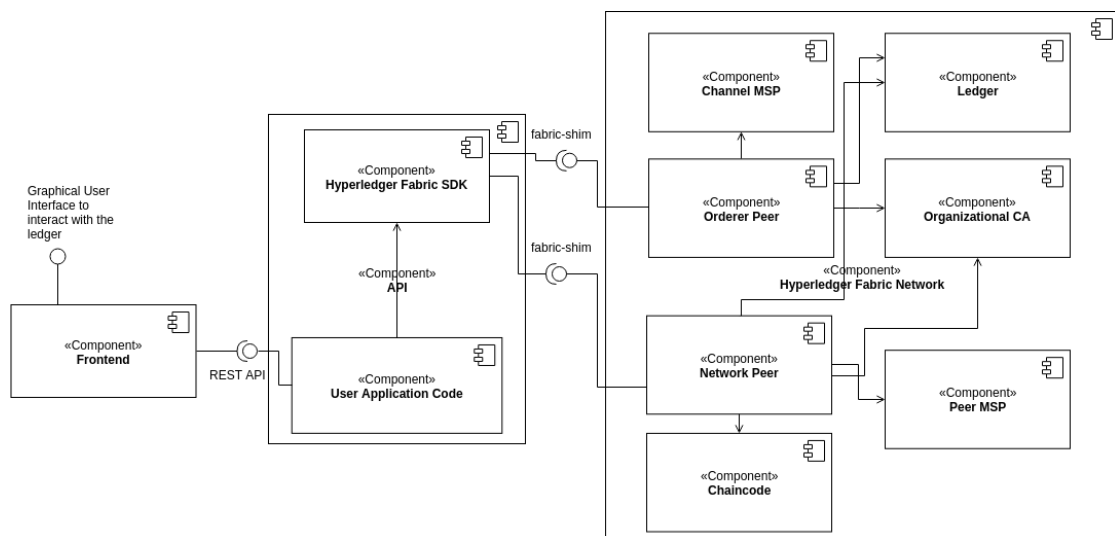


FIGURA 4.6: Diagrama de componentes.

## 4.6. Definición del Comportamiento

Para poder entender cómo interactúan los componentes diagramados, se emplearon dos diagramas de secuencia. El primero, la figura 4.7, muestra el flujo de mensajes en el caso de una lectura del *ledger*, por ejemplo para verificar la validez de un acta. En el diagrama se puede observar lo que ya se describió en la sección 3.3: Para realizar una lectura, es necesario interactuar con un solo *peer* de la red. Éste puede ejecutar el *chaincode* sobre su copia del *ledger* local y proveer resultados sin tener que involucrar otros participantes.

RF-API-01	<p>La API debe proveer un método que permita la subida de un acta. Dicho método debe:</p> <ul style="list-style-type: none"> <li>■ Recibir un objeto JSON con los datos completos del acta en texto plano.</li> <li>■ Aplicar un <i>hash</i> a la cabecera del acta y a cada uno de sus renglones.</li> <li>■ Formar un objeto JSON que consta del <i>hash</i> de la cabecera, el nombre del usuario, un indicador de la validez del acta como el conjunto de todos los renglones del acta en formato <i>hash</i>.</li> <li>■ Invocar el <i>chaincode</i> mencionado en RF-BC-06 que permita escribir dicho objeto en el Blockchain.</li> </ul>
RF-API-02	<p>La API debe proveer un método que permita recuperar el acta como fue escrito en el Blockchain, haciendo uso del <i>chaincode</i> descrito en el requerimiento RF-BC-06.</p>
RF-API-03	<p>La API debe contar con un método que permita revocar a un acta. Para eso debe</p> <ul style="list-style-type: none"> <li>■ Verificar que el acta realmente puede ser rectificada.</li> <li>■ Marcar el acta a revocar como inválida.</li> <li>■ Generar un acta nueva con los renglones corregidos.</li> </ul>
RF-API-04	<p>La API debe proveer una función que permita indicar la validez o invalidez de una nota. Para cumplir con esa funcionalidad, debe</p> <ul style="list-style-type: none"> <li>■ Recibir el id del acta en texto plano como también la información de la nota que se encuentra en el renglón correspondiente del acta.</li> <li>■ Luego debe aplicar la función <i>hash</i> al renglón y buscar el acta indicada en el Blockchain.</li> <li>■ Por último debe verificar si existe una coincidencia entre uno de los renglones recuperados y el calculado.</li> </ul>
RF-API-05	<p>La API debe poder indicar si una historia académica es válida o no, recibiendo la información necesaria y procediendo igual que en los pasos del requerimiento RF-API-04.</p>
RF-API-06	<p>La API debe notificar cualquier tipo de error: En el caso que un acta no fue encontrado o en el caso de un problema de conexión o autenticación con el Blockchain.</p>

CUADRO 4.3: Requerimientos funcionales de la API.

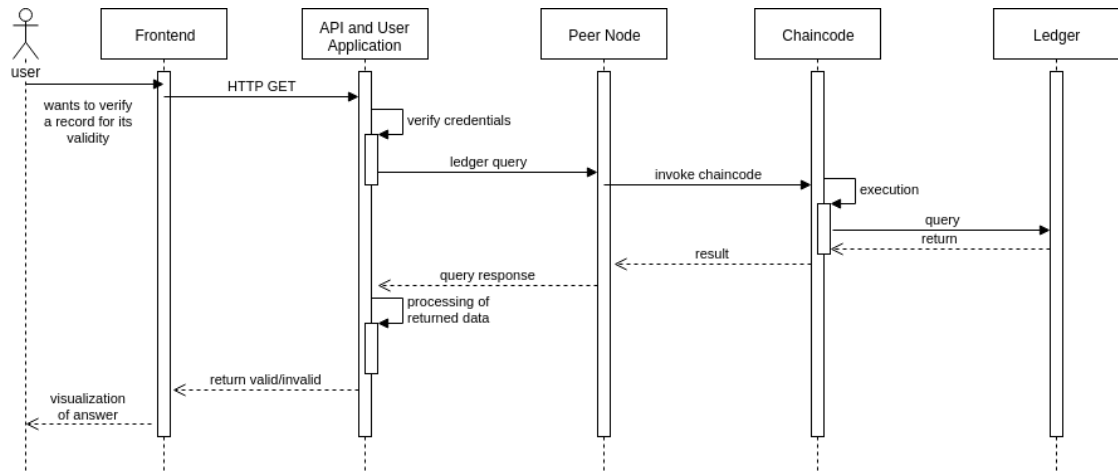
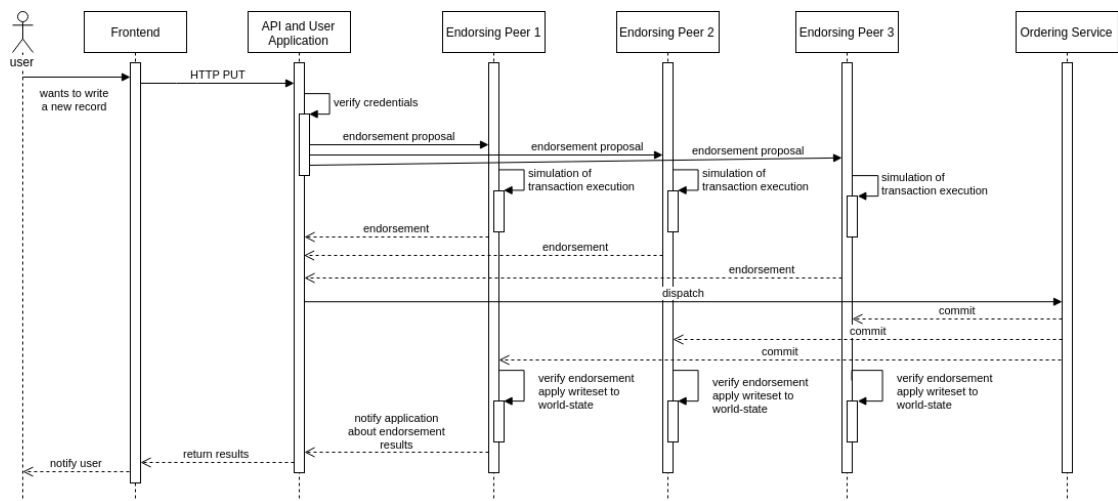
RF-GUI-01	La GUI debe contar con una página que permite la selección y ejecución de las siguientes tareas: <ol style="list-style-type: none"> <li>1. Cargar un acta nueva al sistema.</li> <li>2. Buscar un acta con su id.</li> <li>3. Validar una nota.</li> <li>4. Validar una historia académica.</li> <li>5. Revocar un acta.</li> <li>6. Consultar todos los actas rectificadas.</li> </ol>
RF-GUI-02	Para poder cumplir con cada tarea, la GUI debe proveer campos que permitan al usuario ingresar los datos necesarios para efectuar la tarea.
RF-GUI-03	La GUI debe estar en los idiomas inglés y español.
RF-GUI-04	Durante el tiempo de ejecución de consultas o escrituras en el Blockchain, la GUI debe indicar esto mismo al usuario.
RF-GUI-05	La GUI debe notificar el éxito o fracaso de cada operación debidamente al usuario.

CUADRO 4.4: Requerimientos funcionales de la interfaz gráfica de usuario. (*Graphical User Interface*)

RNF-01	El sistema debe contar con pruebas y tests integrales de los seis requerimientos de usuario.
RNF-02	El código del <i>chaincode</i> debe estar escrito en el lenguaje Golang, la API debe desarrollarse en Node.js y para la Interfaz Gráfica se debe usar Typescript.
RNF-03	La GUI debe indicar éxito o fracaso de una operación en un tiempo no mayor a los diez segundos.
RNF-04	La GUI debe contar con un instructivo para que un usuario sin conocimiento previo puede aprender a utilizar el sistema.

CUADRO 4.5: Requerimientos no funcionales del sistema.

En la figura 4.8 se puede observar el diagrama de secuencia para caso de escritura del *leger*: La aplicación envía una propuesta de aprobación de la transacción (*endorsement proposal*) a una cantidad definida de *peers*. En el diagrama se ve que la propuesta se envía a tres *peers*, pero pueden ser más o menos según la política de aprobación establecida por las organizaciones. Cada *peer* ejecuta el *chaincode* correspondiente y retorna los resultados a la aplicación, cuya obligación es verificar que coinciden. Luego, esta envía un mensaje con todas las respuestas al servicio que ordena las transacciones. Cuando un bloque nuevo es creado, las transacciones en cuestión van a estar incluidas en él y todos los *peers* controlan que se encuentran válidas y que se respetaron las políticas de aprobación. Una vez que terminaron, un *peer* notifica la aplicación para que el usuario pueda saber si la escritura en el Blockchain fue exitosa o no.

FIGURA 4.7: Diagrama de secuencia para lecturas del *ledger*.FIGURA 4.8: Diagrama de secuencia para escrituras del *ledger*.

## 4.7. Riesgos

La evaluación de riesgos de un proyecto busca posibles eventos no deseados que pueden perjudicar o amenazar al proyecto con el objetivo de minimizarlos o, en el caso de que ocurran, controlar su impacto sobre el proyecto.

El objetivo de la presente sección es establecer criterios, identificar posibles riesgos, asignar prioridades a los mismos, evaluar su probabilidad y encontrar estrategias que permiten resolver los mismos o minimizar sus efectos.

### 4.7.1. Criterios

Los riesgos se clasifican según dos criterios: su probabilidad y la gravedad en el caso de su ocurrencia. A continuación figuran las siguientes probabilidades y gravedades que se



tuvieron en cuenta según [36]:

1. Riesgo muy bajo (<10 %)
2. Riesgo bajo (10 - 25 %)
3. Riesgo moderado (25 - 50 %)
4. Riesgo alto (50 - 75 %)
5. Riesgo muy alto (>75 %)

1. Efectos insignificantes
2. Efectos tolerables
3. Efectos graves
4. Efectos catastróficos

La combinación de ambos criterios permite priorizar los riesgos teniendo en cuenta ambos factores. En base a la combinación realizada, se consideran los riesgos que se encuentran

Probabilidad/Efecto	Insignificante	Tolerable	Grave	Catastrófico
Muy baja				
Baja				
Moderada				
Alta				
Muy alta				

CUADRO 4.6: Probabilidad de riesgos vs. su efecto

dentro del área verde como riesgos de menor prioridad, los riesgos en el área amarilla como riesgos de prioridad intermedia y los riesgos en el área roja como riesgos de prioridad alta.

#### 4.7.2. Identificación de Riesgos

En la primera etapa de la gestión de riesgos, se buscan los posibles riesgos relacionados al proyecto, distinguiendo las 5 categorías siguientes:

- **Riesgos de tecnología:** Riesgos relacionados a hardware o *software* con el que se está desarrollando el proyecto.
- **Riesgos personales:** Relacionados con el personal en el equipo de desarrollo.
- **Riesgos de requerimientos:** Surgen de modificaciones de los requerimientos.
- **Riesgos de estimación:** relacionados a la estimación de recursos requeridos para el desarrollo del proyecto.

Los riesgos que se identificaron para el presente proyecto integrador figuran en la tabla 4.7.

Código	Descripción	Tipo de riesgo
Riesgo-01	Problemas de funcionamiento o avería de la PC de desarrollo durante el tiempo de desenvolvimiento.	Tecnológico
Riesgo-02	Incompatibilidad de versiones o librerías instaladas en la máquina de desarrollo con los <i>frameworks</i> usados.	Tecnológico
Riesgo-03	Funcionalidades sin implementar en el <i>framework</i> de Hyperledger Fabric.	Requerimientos
Riesgo-04	Demoras durante el desarrollo del proyecto debido a trabajo u otros inconvenientes personales.	Personales y Estimación
Riesgo-05	Abandono del proyecto.	Personal
Riesgo-06	Errores en la implementación de los <i>frameworks</i> elegidos.	Tecnológico, Requerimientos y Estimación
Riesgo-07	Falta de conocimiento para el uso de los <i>frameworks</i> , del <i>software</i> o los lenguajes de programación elegidos.	Estimación
Riesgo-08	Falta de documentación o soporte de los <i>frameworks</i> elegidos.	Estimación
Riesgo-09	Rechazo de la implementación por parte del usuario, debido a desconfianza que genera una nueva tecnología.	Estimación

CUADRO 4.7: Riesgos identificados para el proyecto

### 4.7.3. Análisis de Riesgos

En el cuadro 4.8 se ordenaron los riesgos según su importancia luego de estimar probabilidad e indicar el efecto.

### 4.7.4. Planificación de Riesgos

En la presente sección se describen las estrategias para manejar cada riesgo identificado. Los cuadros 4.9 y 4.10 detallan los pasos a seguir en el caso de la ocurrencia de cualquier riesgo.

## 4.8. Control de Versiones

Para el control de versiones, se utilizó un repositorio de Gitlab disponible bajo la URL <https://gitlab.com/ELeonhardt/final-project-codebase>

Código	Riesgo	Probabilidad	Efecto	Importancia
Riesgo-02	Incompatibilidad de versiones o librerías instaladas en la máquina de desarrollo con los <i>frameworks</i> usados.	Moderada	Insignificante	Prioridad baja
Riesgo-01	Problemas de funcionamiento o avería de la PC de desarrollo durante el tiempo de desenvolvimiento.	Baja	Grave	Prioridad Intermedia
Riesgo-03	Funcionalidades sin implementar en el <i>framework</i> de Hyperledger Fabric.	Alta	Tolerable	Prioridad Intermedia
Riesgo-05	Abandono del proyecto.	Muy baja	Catastrófico	Prioridad Intermedia
Riesgo-06	Errores en la implementación de los <i>frameworks</i> elegidos.	Alta	Tolerable	Prioridad intermedia
Riesgo-07	Falta de conocimiento para el uso de los <i>frameworks</i> , del <i>software</i> o los lenguajes de programación elegidos.	Alta	Tolerable	Prioridad intermedia
Riesgo-08	Falta de documentación o soporte de los <i>frameworks</i> elegidos.	Moderada	Grave	Prioridad intermedia
Riesgo-04	Demoras durante el desarrollo del proyecto debido a trabajo u otros inconvenientes personales.	Alta	Grave	Prioridad alta
Riesgo-09	Rechazo de la implementación por parte del usuario, debido a desconfianza que genera una nueva tecnología.	Alta	Grave	Prioridad alta

CUADRO 4.8: Riesgos clasificados según su probabilidad y efecto.

Código	Riesgo	Estrategias de Manejo del riesgo
Riesgo-01	Problemas de funcionamiento o avería de la PC de desarrollo durante el tiempo de desenvolvimiento.	<ul style="list-style-type: none"> <li>■ Implementación de una estrategia de respaldo de información en la nube con repositorios de Gitlab.</li> <li>■ Realización de la operación <i>push</i> al repositorio una o más veces por día.</li> <li>■ Arreglo o reemplazo de la máquina de desarrollo en el caso de una avería.</li> </ul>
Riesgo-02	Incompatibilidad de versiones o librerías instaladas en la máquina de desarrollo con los <i>frameworks</i> usados.	<ul style="list-style-type: none"> <li>■ Lectura cuidadosa de tutoriales de instalación y prerequisites.</li> <li>■ Uso de Docker para ganar mayor flexibilidad con versiones de librerías o lenguajes de programación.</li> </ul>
Riesgo-03	Funcionalidades sin implementar en el framework de Hyperledger Fabric.	Adaptación de los requerimientos a las posibilidades del <i>framework</i> .
Riesgo-04	Demoras durante el desarrollo del proyecto debido a trabajo u otros inconvenientes personales.	<ul style="list-style-type: none"> <li>■ Comunicación fluida con director y codirector del trabajo.</li> <li>■ Reducción de obligaciones laborales y extracurriculares a lo mínimo e indispensable.</li> </ul>
Riesgo-05	Abandono del proyecto	<ul style="list-style-type: none"> <li>■ Comunicación fluida con director y codirector del trabajo.</li> <li>■ Definición clara del alcance del proyecto para minimizar las posibilidades de desborde en términos de tiempo y trabajo.</li> </ul>

CUADRO 4.9: Estrategias de manejo de riesgos parte uno

Si bien Git es una herramienta para para mejorar la colaboración entre varios programadores, también se justifica su uso cuando una sola persona está desarrollando el proyecto. Usar un repositorio permite tener acceso remoto al proyecto desde cualquier equipo, teniendo así una copia de seguridad del proyecto completo en la nube. A su vez, se mantiene un historial de versiones y en caso de ser necesario, es posible volver a una versión anterior o comparar las diferencias entre dos versiones distintas.

Riesgo-06	Errores en la implementación de los <i>frameworks</i> elegidos.	<ul style="list-style-type: none"> <li>■ Implementación de <i>workarounds</i> descritos en la documentación o páginas de seguimientos de <i>issues</i> como Jira.</li> <li>■ Modificación de la implementación para poder cumplir con el requerimiento.</li> <li>■ Adaptación del requerimiento.</li> </ul>
Riesgo-07	Falta de conocimiento para el uso de los <i>frameworks</i> , del <i>software</i> o los lenguajes de programación elegidos.	<ul style="list-style-type: none"> <li>■ Lectura de tutoriales y realización de cursos de tiempo corto para lograr una introducción en temas cuyo desconocimiento impide el avance del proyecto.</li> <li>■ Empleo y modificación de ejemplos previstos para el aprendizaje.</li> <li>■ Consultas a programadores experimentados en los campos en cuestión.</li> </ul>
Riesgo-08	Falta de documentación o soporte de los <i>frameworks</i> elegidos.	<ul style="list-style-type: none"> <li>■ Búsqueda de explicaciones adicionales en foros de usuarios como Stackoverflow.</li> <li>■ Consultas en canales de chat de desarrolladores como Slack o Rocketchat.</li> <li>■ Modificación de la implementación para poder cumplir con el requerimiento.</li> <li>■ Adaptación del requerimiento.</li> </ul>
Riesgo-09	Rechazo de la implementación por parte del usuario, debido a desconfianza que genera una nueva tecnología.	<ul style="list-style-type: none"> <li>■ Explicación de la tecnología y capacitación del personal.</li> <li>■ Prueba de piloto para identificar mejoras en cuanto a la usabilidad.</li> </ul>

CUADRO 4.10: Estrategias de manejo de riesgos parte dos

## Capítulo 5

# Implementación

El presente capítulo describe los detalles de la implementación de los requerimientos explicados en el capítulo 4. Se desarrollará el sistema completo localmente en una computadora de uso común. Para explicar el tratamiento que recibe un acta en la aplicación, se usan diagramas de actividad detalladas y se realizan pruebas sobre cada etapa antes de proceder con la siguiente.

### 5.1. Implementación Local

La implementación local del proyecto fue realizada en una Notebook MSI PL62 7RC, el sistema operativo empleado fue Ubuntu 16.04 LTS. Las características de la máquina se detallan en la figura 5.1

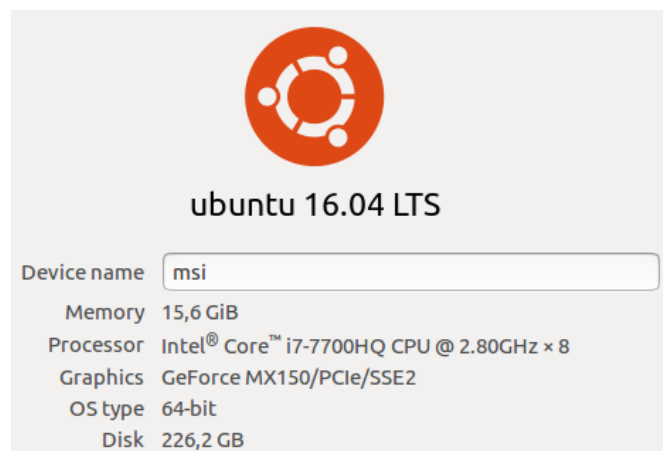


FIGURA 5.1: Características técnicas de la computadora de desarrollo.

### 5.1.1. Configuración de Hyperledger Fabric

Antes de iniciar la implementación de la aplicación en sí, es necesario configurar una red y un Blockchain con Hyperledger Fabric. Esta sección describe los pasos necesarios para cumplir con dicho objetivo.

1. **Generación de certificados con la herramienta cryptogen:** Como descrito en la sección 3.6, cada integrante de la red establece su identidad a través de un certificado X.509. Hyperledger Fabric provee una herramienta llamada cryptogen que genera un conjunto de certificados y claves, así una red puede ser iniciada fácilmente con todo el material criptográfico necesario.

Cryptogen lee la arquitectura deseada de la red desde un archivo con formato *yaml* y genera los certificados acorde a dicho archivo. Por ende, el primer paso para la configuración de la red fue expresar su topología deseada de la siguiente forma:

```
1 OrdererOrgs:
2   - Name: orderer
3     Domain: pi.elisabeth.com
4     Specs:
5       - Hostname: orderer
6
7 PeerOrgs:
8   - Name: fcefyn
9     Domain: fcefyn.pi.elisabeth.com
10    #EnableNodeOUs: true
11    Template:
12      Count: 2
13    Users:
14      Count: 1
15
16   - Name: famaf
17     Domain: famaf.pi.elisabeth.com
18    #EnableNodeOUs: true
19    Template:
20      Count: 2
21    Users:
22      Count: 1
23
24   - Name: fcq
25     Domain: fcq.pi.elisabeth.com
26    #EnableNodeOUs: true
27    Template:
28      Count: 2
29    Users:
30      Count: 1
```

Las primeras cinco líneas definen una organización para el servicio que ordena las transacciones, con su nombre y dominio. Se eligió `pi.elisabeth.com`, donde `pi` es la abreviación de proyecto integrador. El dominio del cualquier *peer* está compuesto por su *hostname* y el dominio de la organización, por lo cual se accede como `orderer.pi.elisabeth.com`. Como se va a emplear el servicio *solo*, se emplea a un solo host para dicha tarea.

En la segunda sección, línea 7 a 30, se definen las organizaciones de los *peers*. Para eso, se eligieron 3 facultades de la UNC y se definieron 2 *peers* por cada facultad, indicados por las líneas 12, 20 y 28. La propiedad *Users* está pensada para definir de forma anticipada la cantidad de usuarios que van a utilizar el Blockchain. En el caso del prototipo a desarrollar, este aspecto se va a definir de forma dinámica, por lo cual el archivo indica 1 usuario de modo que se cree solo un usuario administrador.

Una vez que se pudo establecer una organización lógica de las organizaciones y *peers*, es necesario generar el material criptográfico correspondiente. Para eso, se ejecuta la herramienta `cryptogen` en la *shell* con el siguiente comando:

```
$ ./cryptogen generate --config=./crypto-config.yaml
```

Durante la ejecución se crea una carpeta en el mismo directorio que lleva el nombre *crypto-config*. Dicha carpeta contiene las identidades digitales de todos los participantes de la red. Una parte de su estructura y contenidos se muestra en la figura 5.2

2. **Generación del bloque de génesis y configuración del canal:** En el capítulo 2 se explicó la arquitectura de un Blockchain: Información se almacena en bloques que contienen el *hash* del bloque anterior y forman así una cadena de información. El primer bloque, llamado bloque de génesis, tiene la particularidad de almacenar solamente su propio *hash*, ya que no tiene *hash* de ningún bloque anterior que almacenar.

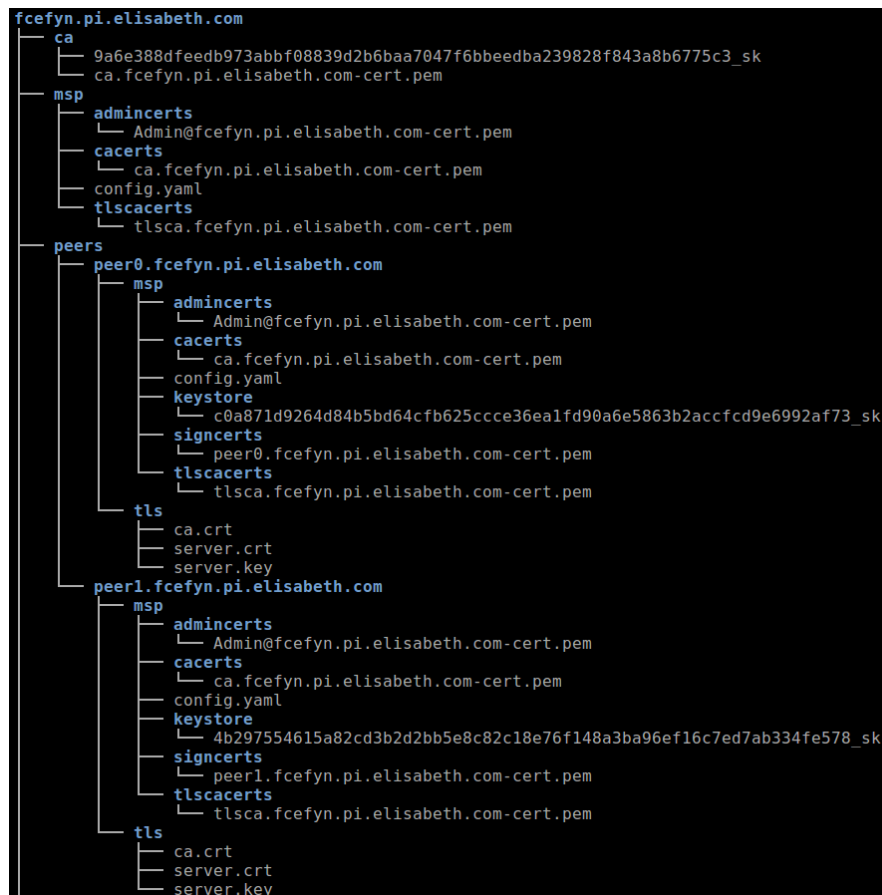
En Hyperledger Fabric, el bloque de génesis cumple con un rol particular: Almacena la configuración del servicio que ordena transacciones y además información sobre las organizaciones y *peers* que van a formar parte de la red. Un conjunto de organizaciones en este contexto se llama consorcio.

Junto con el bloque de génesis se necesita la configuración del canal, que en Hyperledger Fabric describe el consorcio que va a tener acceso a un mismo *ledger*.

El bloque de génesis y la configuración del canal son llamados artefactos y son generados con la herramienta *configtxgen*. Como *cryptogen*, consume un archivo que contiene descripciones de los miembros de la red como del servicio de ordenamiento de transacciones. A continuación se van a describir las dos secciones más importantes del archivo de configuración:

```
97 Orderer: &OrdererDefaults
98     OrdererType: solo
99     Addresses:
```



FIGURA 5.2: Directorios y archivos de la carpeta *crypto-config*

```

100         - orderer.pi.elisabeth.com:7050
101     BatchTimeout: 2s
102     BatchSize:
103         MaxMessageCount: 10
104         AbsoluteMaxBytes: 99 MB
105         PreferredMaxBytes: 512 KB

```

En las líneas 97 a 105 se especifica el funcionamiento del servicio de ordenamiento de transacciones: Como tipo se eligió *solo* (con las alternativas Raft y Kafka detalladas en la sección 3.8) y se especificó un tiempo de espera de lote (*BatchTimeout*) de dos segundos junto con un tamaño de lote (*BatchSize*) de 512KB y un máximo de 99MB. Es decir, un bloque nuevo se genera cada dos segundos, sino se llegó antes al tamaño de lote definido.

```

139 Profiles:
140     OrgsOrdererGenesis:
141         <<: *ChannelDefaults
142     Orderer:
143         <<: *OrdererDefaults
144     Organizations:

```

```

145         - *OrdererOrg
146     Capabilities:
147         <<: *OrdererCapabilities
148     Consortiums:
149         PiConsortium:
150             Organizations:
151                 - *fcefyn
152                 - *famaf
153                 - *fcq
154     OrgsChannel:
155         Consortium: PiConsortium
156         Application:
157             <<: *ApplicationDefaults
158             Organizations:
159                 - *fcefyn
160                 - *famaf
161                 - *fcq
162             Capabilities:
163                 <<: *ApplicationCapabilities

```

En las líneas 139 - 153 se define la configuración que se guarda en el bloque de génesis: Se compone por un consorcio, es decir, las tres organizaciones ya identificadas en el archivo `crypto-config.yaml` y la organización responsable de ordenar transacciones, aquí llamada `OrdererOrg`.

En las líneas 154 - 163 se especifican las organizaciones que van a tener acceso al mismo canal.

El bloque de génesis se genera con el siguiente comando:

```

$./configtxgen -profile OrgsOrdererGenesis -channelID
↪ sys-channel -outputBlock ./channel-artifacts/genesis.block

```

Luego se crea la transacción de configuración del canal con

```

$./configtxgen -profile OrgsChannel -outputCreateChannelTx
↪ ./channel-artifacts/channel.tx -channelID pichannel

```

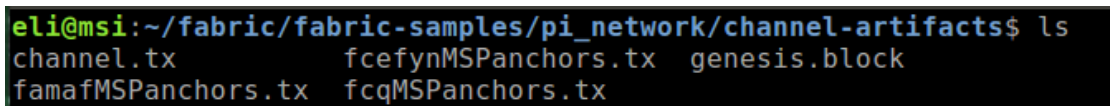
Para que *peers* puedan comunicarse entre organizaciones, al menos uno por organización necesita tener el estado de *anchor peer*. Dicha configuración se realiza con los comandos siguientes:

```

$./configtxgen -profile OrgsChannel -outputAnchorPeersUpdate
↪ ./channel-artifacts/famafMSPanchors.tx -channelID pichannel
↪ -asOrg famafMSP
$./configtxgen -profile OrgsChannel -outputAnchorPeersUpdate
↪ ./channel-artifacts/fcefynMSPanchors.tx -channelID
↪ pichannel -asOrg fcefynMSP
$./configtxgen -profile OrgsChannel -outputAnchorPeersUpdate
↪ ./channel-artifacts/fcqMSPanchors.tx -channelID pichannel
↪ -asOrg fcqMSP

```

Se puede comprobar que en la carpeta `channel-artifacts` se encuentran los dos artefactos generados.



```

eli@msi:~/fabric/fabric-samples/pi_network/channel-artifacts$ ls
channel.tx          fcefynMSPanchors.tx  genesis.block
famafMSPanchors.tx fcqMSPanchors.tx

```

FIGURA 5.3: Artefactos generados por *configtxgen*

3. **Iniciar los integrantes de la red con docker-compose:** Luego de la creación de los artefactos, el siguiente paso es iniciar todos los participantes de la red y compartirles los archivos generados. De esa forma pueden conocer su identidad y empezar a comunicarse con los demás participantes de la red. Para simular la cantidad de hosts requerida, se utilizaron las herramientas Docker y Docker Compose.

La configuración de los *peers*, autoridades de certificación y nodos de ordenamiento de transacciones se realizó en 3 archivos de docker-compose diferentes. El primero tiene el nombre *peer-base.yaml* y define un *peer* genérico con variables de entorno que aplican a todos los *peers*, independiente de su organización. El segundo archivo, *docker-compose-base.yaml*, describe todos los *peers* que van a formar parte de la red con sus variables de entorno específicas. Un ejemplo provee la declaración del primer *peer* de fcefyn:

```

75  peer0.fcefyn.pi.elisabeth.com:
76      container_name: peer0.fcefyn.pi.elisabeth.com
77      extends:
78          file: peer-base.yaml
79          service: peer-base
80      environment:
81          - CORE_PEER_ID=peer0.fcefyn.pi.elisabeth.com
82          - CORE_PEER_ADDRESS=peer0.fcefyn.pi.elisabeth.com:7051
83          - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.fcefyn.pi.elisabeth.com:7051
84          - CORE_PEER_GOSSIP_BOOTSTRAP=peer1.fcefyn.pi.elisabeth.com:7051
85          - CORE_PEER_LOCALMSPID=fcefynMSP
86          - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/fabric/msp

```

```

87     - CORE_CHAINCODE_DEPLOYTIMEOUT=300s
88     - CORE_CHAINCODE_STARTUPTIMEOUT=300s
89     volumes:
90         - /var/run/:/host/var/run/
91         - ../channel-artifacts:/etc/hyperledger/configtx
92         - ../crypto-config/peerOrganizations/fcefyn.pi.elisabeth.
          ↪ com/peers/peer0.fcefyn.pi.elisabeth.com/msp:/etc/hype
          ↪ rledger/fabric/msp
93         - peer0.fcefyn.pi.elisabeth.com:/var/hyperledger/producti
          ↪ on
94     ports:
95         - 9051:7051
96         - 9053:7053

```

Líneas 78 y 79 muestran que el servicio extiende las propiedades descritas en *peer-base.yaml*. En la sección *environment* se declaran variables de entorno específicos del *peer0* de la organización *fcefyn* y en la sección volúmenes se montan en primer lugar la transacción de configuración del canal y luego el material criptográfico generado con *crypto-config*.

El tercer archivo, con el nombre *docker-compose.yaml*, declara la topología completa de la red, extendiendo a *docker-compose-base.yaml*. En una red llamada *pi*, se definen 3 autoridades de certificación, (una por organización), un total de 6 *peers*, un *orderer* y un contenedor auxiliar con el nombre *cli*.

Para iniciar todos los contenedores, es necesario ejecutar el siguiente comando:

```
$ docker-compose up -d
```

y luego se pueden ver los contenedores iniciados con el comando

```
$ docker ps
```

```

eli@msi:~$ docker ps --format "table {{.ID}}\t{{.Image}}\t{{.Status}}\t{{.Names}}"
CONTAINER ID        IMAGE                                     STATUS              NAMES
299bdba30727        hyperledger/fabric-tools:1.4           Up 2 hours         cli
c2d50ad990c9        hyperledger/fabric-orderer:1.4         Up 2 hours         orderer.pi.elisabeth.com
3d9041101d39        hyperledger/fabric-peer:1.4            Up 2 hours         peer1.fcq.pi.elisabeth.com
9ed6cec058e8        hyperledger/fabric-peer:1.4            Up 2 hours         peer0.fcq.pi.elisabeth.com
17dda2151241        hyperledger/fabric-peer:1.4            Up 2 hours         peer1.famaf.pi.elisabeth.com
f1fc161b31b1        hyperledger/fabric-peer:1.4            Up 2 hours         peer1.fcefyn.pi.elisabeth.com
dd2c8903e421        hyperledger/fabric-ca:1.4              Up 2 hours         ca_famaf
fa72c4570207        hyperledger/fabric-ca:1.4              Up 2 hours         ca_fcefyn
ffc515ffa74a        hyperledger/fabric-peer:1.4            Up 2 hours         peer0.famaf.pi.elisabeth.com
76a00b35c6d0        hyperledger/fabric-ca:1.4              Up 2 hours         ca_fcq
09bed2955c63        hyperledger/fabric-peer:1.4            Up 2 hours         peer0.fcefyn.pi.elisabeth.com
eli@msi:~$

```

FIGURA 5.4: La red de Hyperledger Fabric desplegada con contenedores Docker

En la figura 5.4 se puede ver una captura en la cual figuran los contenedores previamente declarados. Se aplicó un filtro al comando para mejorar la visibilidad del resultado.

4. **Creación del canal y configuración de los peers:** Una vez que todos los contenedores se encuentran iniciados, es necesario crear el canal e indicar a todos los *peers* que se unan al mismo. Para completar dicha tarea, existe el contenedor con el nombre *cli*, que permite ejecutar comandos en los *peers* a través de una API.

Primero, se accede a la *shell* del contenedor:

```
$ docker exec -it cli bash
```

Luego, se ejecuta el siguiente comando para la creación del canal, haciendo uso del artefacto *channel.tx* generado en el paso 2:

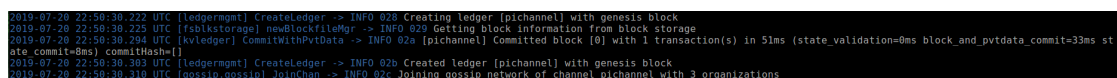
```
$ peer channel create -o orderer.pi.elisabeth.com:7050 -c pichannel
↪ -f ./channel-artifacts/channel.tx
```

Para unir los demás *peers* al canal, se ejecuta

```
$ peer channel join -b pichannel.block
```

en cada *peer*.

Con el fin de asegurar que el *ledger* y el canal se crearon de forma exitosa, se puede inspeccionar el log de cualquier *peer*:



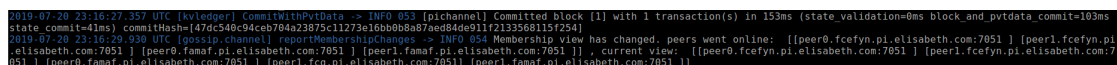
```
2019-07-20 22:50:30.222 UTC [ledgermgmt] CreateLedger -> INFO 028 Creating ledger [pichannel] with genesis block
2019-07-20 22:50:30.223 UTC [fabricstorage] newBlockfileMgr -> INFO 029 Getting block information from block storage
2019-07-20 22:50:30.294 UTC [kvledger] CommitWithPvtData -> INFO 02a [pichannel] committed block [0] with 1 transaction(s) in 51ms (state_validation=0ms block_and_pvtdata_commit=33ms state_commit=8ms) commitHash=
2019-07-20 22:50:30.383 UTC [ledgermgmt] CreateLedger -> INFO 02b Created ledger [pichannel] with genesis block
2019-07-20 22:50:30.319 UTC [gossip.gossip] JoinChan -> INFO 02c Joining gossip network of channel pichannel with 3 organizations
```

FIGURA 5.5: Mensajes del log de un *peer* indicando que *ledger* y canal se crearon de forma exitosa.

Por último, se aplican los artefactos que precisan los *anchor peers*. Aquí se ve el comando para la definición del *anchor peer* de la organización *fcq*:

```
$ peer channel update -o orderer.pi.elisabeth.com:7050 -c pichannel
↪ -f ./channel-artifacts/fcqMSPanchors.tx
```

En la figura 5.6 se puede ver que todos los *peers* se unieron al canal y que el *peer0* de la organización *fcq* tiene conocimiento de todos los demás *peers* de la red.



```
2019-07-20 22:16:27.252 UTC [kvledger] CommitWithPvtData -> INFO 033 [pichannel] Committed block [1] with 1 transaction(s) in 153ms (state_validation=0ms block_and_pvtdata_commit=103ms state_commit=41ms) commitHash=[47dc540c94ceb704a23875c11273e16bb8b8a87aed84de911f2133568115f254]
2019-07-20 22:16:29.930 UTC [gossip.channel] reportMembershipChanges -> INFO 034 Membership view has changed. peers went online: [[peer0.fcq.pi.elisabeth.com:7051] [peer1.fcq.pi.elisabeth.com:7051] [peer0.famaf.pi.elisabeth.com:7051] [peer1.famaf.pi.elisabeth.com:7051] [peer0.fcq.pi.elisabeth.com:7051] [peer1.fcq.pi.elisabeth.com:7051] [peer0.famaf.pi.elisabeth.com:7051] [peer1.famaf.pi.elisabeth.com:7051]]
```

FIGURA 5.6: Mensajes del log de *peer0.fcq.pi.elisabeth.com* indicando todos los *peers* conocidos.

5. **Instalación y Uso del Chaincode:** Ya se logró configurar la red con todos sus participantes, pero todavía no es posible almacenar la información en el *ledger*. Antes es necesario escribir e instalar el *chaincode*.

El *chaincode* en Hyperledger Fabric es el equivalente de los *smart contracts* en otras implementaciones, como por ejemplo Ethereum. En el contexto de este proyecto, se escribieron tres chaincodes: *addRegister* para agregar un acta al *ledger*, *queryRegister* para consultar por un acta y *queryAllRectified* para obtener todas las actas rectificadas.

El *chaincode* invoca la API presente en los *peers*. A continuación se puede ver la función *addRegister*:

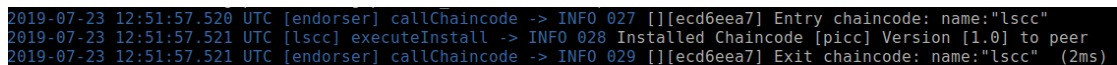
```
func (s *SmartContract) addRegister(APIstub
    ↪ shim.ChaincodeStubInterface, args []string) sc.Response {

    var buffer bytes.Buffer
    APIstub.PutState(args[0], []byte(args[1]))
    return shim.Success(buffer.Bytes())
}
```

Luego de su redacción, es necesario instalar el *chaincode* en todos los *peers* que lo van a utilizar. Como en el caso de la configuración del canal, eso también se hace a través del contenedor cli. El comando para la instalación es el siguiente:

```
$ peer chaincode install -n picc -v 1.0 -p github.com/chaincode/
```

Los mensajes correspondientes en los *logs* del *peer* confirman la instalación correcta, visibles en la figura 5.7.



```
2019-07-23 12:51:57.520 UTC [endorser] callChaincode -> INFO 027 [[ecd6eea7] Entry chaincode: name:"lscc"
2019-07-23 12:51:57.521 UTC [lscc] executeInstall -> INFO 028 Installed Chaincode [picc] Version [1.0] to peer
2019-07-23 12:51:57.521 UTC [endorser] callChaincode -> INFO 029 [[ecd6eea7] Exit chaincode: name:"lscc" (2ms)
```

FIGURA 5.7: Mensajes del log de peer0.famaf.pi.elisabeth.com indicando la instalación correcta del *chaincode*.

Luego es necesario instanciar al *chaincode*, lo cual significa que se inicializa un contenedor Docker adicional en la misma red, en el cual el *chaincode* se va a ejecutar. Mientras la instalación es necesaria en todos los *peers*, la instanciación se requiere una sola vez.

```
$ docker exec cli peer chaincode instantiate -o
    ↪ orderer.pi.elisabeth.com:7050 -C pichannel -n picc -v 1.0 -c
    ↪ '{"Args":[]}' -P "OutOf (2, 'famafMSP.peer', 'fcefynMSP.peer',
    ↪ 'fcqMSP.peer')"
```

El último argumento indica la política de aprobación: Se definió que dos *peers* de diferentes organizaciones tienen que haber ejecutado el *chaincode* y haber llegado al mismo resultado, caso contrario la operación no va a ser agregada al *ledger*.

Por último, se verifica el funcionamiento correcto de la red invocando el *chaincode* instalado. Éste tiene 3 funciones: *addRegister* para agregar un acta, *queryRegister* para consultar por un acta y *queryAllRectified* para obtener todos los actas rectificadas.

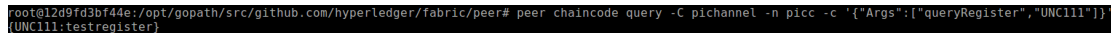
Desde el contenedor cli, se ejecutó primero el comando siguiente:

```
$ peer chaincode invoke -o orderer.pi.elisabeth.com:7050 -C
    ↪ pichannel -n picc --peerAddresses
    ↪ peer0.famaf.pi.elisabeth.com:7051 --peerAddresses
    ↪ peer0.fcefyn.pi.elisabeth.com:7051 -c '{"Args":["addRegister",
    ↪ "UNC111", {"UNC111": "testregister"}]}'
```

Para poder cumplir con la política de aprobación, la petición se envía a dos *peers* de organizaciones diferentes. La información a escribir es un *string* de prueba en formato JSON. Luego se puede consultar por el acta “UNC111” con la función *queryRegister*:

```
$ peer chaincode query -C pichannel -n picc -c
↪ '{"Args":["queryRegister","UNC111"]}'
```

con el resultado visible en la figura 5.8



```
root@12d9fd3bf44e:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode query -C pichannel -n picc -c '{"Args":["queryRegister","UNC111"]}'
{"UNC111":testregister}
```

FIGURA 5.8: Respuesta de la llamada a la función *queryRegister*

Con los pasos descritos quedan concluidas la configuración y las pruebas de la red de Hyperledger Fabric en un ambiente local.

### 5.1.2. Automatización con un bash script

Dada la numerosa cantidad de pasos a ejecutar, la importancia de escribir los parámetros y los variables de entorno correctamente y la necesidad de ejecutar pruebas con una red nueva, se automatizó el proceso descrito en la sección anterior mediante un *script* en el lenguaje *bash*. Al ejecutarlo,

- Se eliminan todos los contenedores, volúmenes, redes y material criptográfico que pueden existir de una ejecución anterior.
- Se generan nuevamente los artefactos.
- Se inician todos los contenedores especificados en el archivo `docker-compose.yaml`.
- Se configuran canal y *anchor peers*.
- Se instala el *chaincode* en todos los *peers* y se instancia.

Luego de la ejecución, es posible invocar al *chaincode* directamente. El *bash script* posibilita el avance rápido con los otros elementos del proyecto, ya que en caso de ocurrir errores, simplemente se genera una red con material criptográfico nuevo y sin datos previamente almacenados.

### 5.1.3. Implementación de la Aplicación con API

Al desarrollar la aplicación en Javascript, se partió con las funciones definidas en el ejemplo de Hyperledger Fabric disponible en <https://github.com/hyperledger/fabric-samples/tree/release/fabcar>. Luego de descargar los archivos *enrollAdmin.js*, *registerUser.js*, *query.js* e *invoke.js* junto con el respectivo *package.json*, se ejecutó el comando

```
$ npm install
```

Para iniciar el ambiente e instalar todos los módulos requeridos de node.js. Con el fin de que la aplicación se reinicie automáticamente cuando ocurran cambios de código, se instaló nodemon:

```
$ npm install nodemon --save
```

La aplicación necesita información sobre la topología de la red a la cual conectarse. Se define a través del archivo *connection.json*, que indica canales, organizaciones, *peers* y autoridades de certificación. A continuación, se ve una parte del archivo en cuestión:

```
16     "channels": {
17         "pichannel": {
18             "orderers": [
19                 "orderer.pi.elisabeth.com"
20             ],
21             "peers": {
22                 "peer0.famaf.pi.elisabeth.com": {},
23                 "peer1.famaf.pi.elisabeth.com": {},
24                 "peer0.fcefyn.pi.elisabeth.com": {},
25                 "peer1.fcefyn.pi.elisabeth.com": {},
26                 "peer0.fcq.pi.elisabeth.com": {},
27                 "peer1.fcq.pi.elisabeth.com": {}
28             }
29         }
30     },
31     "organizations": {
32         "fcefyn": {
33             "mspid": "fcefynMSP",
34             "peers": [
35                 "peer0.fcefyn.pi.elisabeth.com",
36                 "peer1.fcefyn.pi.elisabeth.com"
37             ],
38             "certificateAuthorities": [
39                 "ca-fcefyn"
40             ]
41         }
42     }
```

Las funciones separadas que se descargaron son útiles para comprobar el funcionamiento correcto del SDK, pero no es un formato conveniente para que sean invocadas desde una aplicación web. Por ende, los pasos a seguir consisten en:

1. Implementar una API con el *framework* Express.



2. Embeber las funciones descargadas para que puedan acceder al Blockchain cuando se llame la API.
3. Desarrollar y agregar las demás funciones necesarias para cumplir con los requerimientos definidos en el capítulo 4.

## 1. Implementación de una API REST con el framework Express

Se empezó con el código de un servidor web que responde a una sola consulta:

```
1 var express = require("express");
2 var bodyParser = require("body-parser");
3 var app = express();
4
5 app.use(bodyParser.json());
6 app.use(bodyParser.urlencoded({ extended: true }));
7
8 app.get("/", function (req, res) {
9     res.status(200).send({ message: 'Welcome to the Blockchain
10     ↪ restful API' });
11 });
12
13 var server = app.listen(8000, function () {
14     console.log("app running on port.", server.address().port);
15 });
```

Luego se usó el *software* Postman para comprobar el correcto funcionamiento del programa.

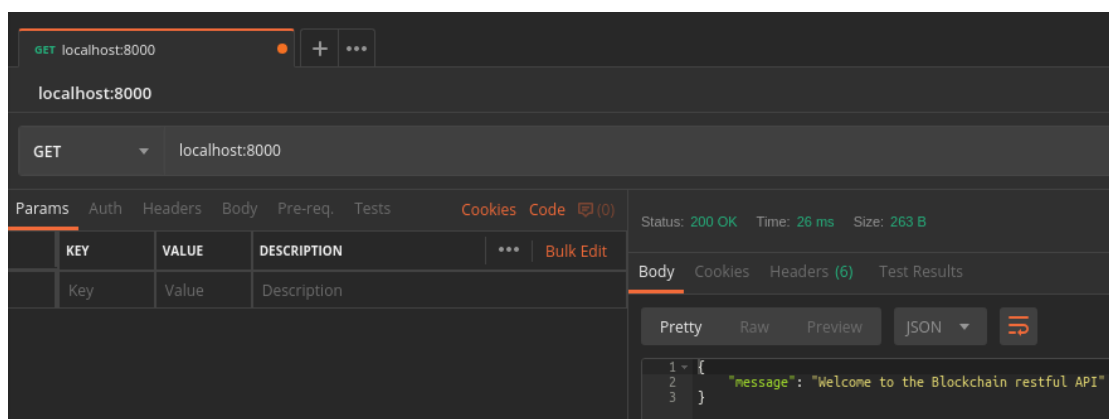


FIGURA 5.9: Prueba de la función GET con Postman

En la figura 5.9 se puede observar que la aplicación está funcionando correctamente, ya que devuelve el mensaje “Welcome to the Blockchain restful API”.

## 2. Ampliación de la API para registrar usuarios

En primer lugar, es necesario inscribir al administrador y luego registrar al usuario

que va a invocar al *chaincode*. En el contexto del proyecto integrador, se va a crear un solo usuario con el nombre *user1* para la organización *fcefyn*, sin embargo es posible ampliar la cantidad de usuarios fácilmente con la ayuda de la función ya escrita. Como el foco de la aplicación es cargar actas y validar sus entradas, la API controla de forma automática que *admin* y *user1* estén registrados:

```

1  app.listen(3000, () => {
2      console.log("Server running on port 3000");
3      enroll_admin.enrollAdmin().then(
4          success => {
5              setTimeout(() => register_user.registerUser(),
6                  ↪ 1000)
7          })
8      });

```

#### 5.1.3.1. Desarrollo de la función *add\_register*

La primera función a desarrollar agrega actas al Blockchain, ya que se necesitan datos presentes en el *ledger* para poder programar funciones futuras que, por ejemplo, validen dichos datos.

Un acta en el contexto de este proyecto tiene formato JSON. En una cabecera se describen los atributos del acta en sí y una cantidad arbitraria de renglones especifica los datos de los alumnos. Un ejemplo con dos renglones sigue a continuación:

```

1  {
2      "UNC201200017":{
3          "Acta Nr":"201200017",
4          "Año academico":"2016",
5          "Actividad":"(1001) Análisis matemático 1a",
6          "Fecha Examen":"05/10/2016",
7          "Turno": "Octubre 2016",
8          "Ubicación": "Sede Av Santa Fe",
9          "Mesa": "A",
10         "Llamado": "Llamado del Turno Octubre 2016"
11     },
12
13     "Renglones":{
14         "1":{
15             "Apellido y Nombre":"Acevedo Mauricio German",
16             "Identificación":"DNI 4260181",
17             "Instancia": "Regular",
18             "Fecha": "05/10/2017",
19             "Nota": 2,
20             "Letras": "dos",

```

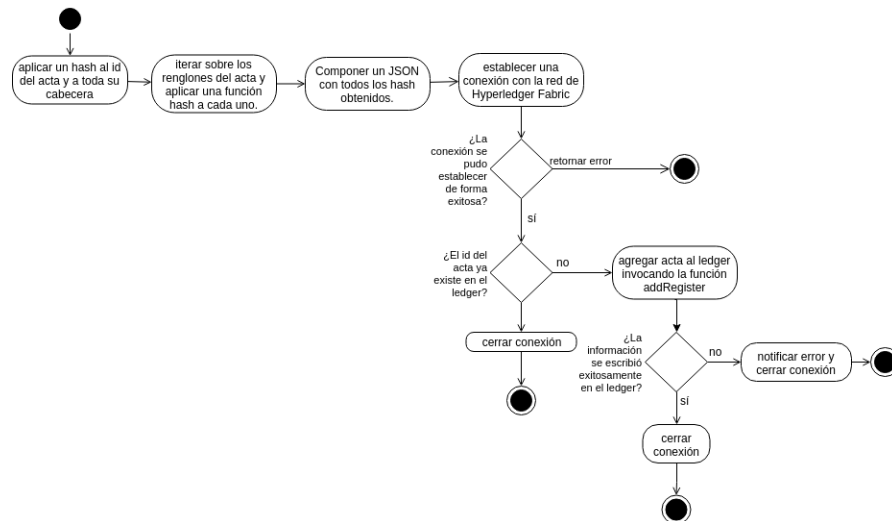


FIGURA 5.10: Diagrama de actividad de la función add\_register

```

21     "Resultado": "reprobado"},
22
23     "2":{
24     "Apellido y Nombre":"Alaniz Roxana Aide",
25     "Identificación":"DNI 9609175",
26     "Instancia": "Regular",
27     "Fecha": "05/10/2017",
28     "Nota": 8,
29     "Letras": "ocho",
30     "Resultado": "aprobado"}
31   }
32 }

```

La figura 5.10 muestra la lógica de la función implementada: Se aplican funciones *hash* al id, a la cabecera y a cada uno de los renglones y luego se genera un JSON nuevo con dicha información que tiene el siguiente aspecto:

```

1  {
2    "id": "d7ce098a17e4bb0cc49041382beb8577",
3    "header": "8bb106442e641e7af465bd40cf54375e",
4    "validity": "valid",
5    "creator": "user1",
6    "lines":
7    { "1": "81116f6d2bdf1728e17d0258d54a74e9",
8      "2": "b1af6ddd90993bdd9763e45e280cbb6b" }
9  }

```

Luego, se establece una conexión con la red de Hyperledger Fabric y se invoca el *chaincode queryRegister* con el id del acta como parámetro. Dicha consulta se hace para verificar que el acta todavía no fue agregada al Blockchain. Si se encuentra una entrada con el mismo id, la conexión se cierra y la API devuelve un mensaje de error especificando que el acta ya existe. Caso contrario se invoca a la función *addRegister* con los parámetros “UNC201200017” y el JSON con los *hash*. Si la operación de escritura fue exitosa, la API devuelve el mensaje “El acta se subió exitosamente” como se ve en la figura 5.11.

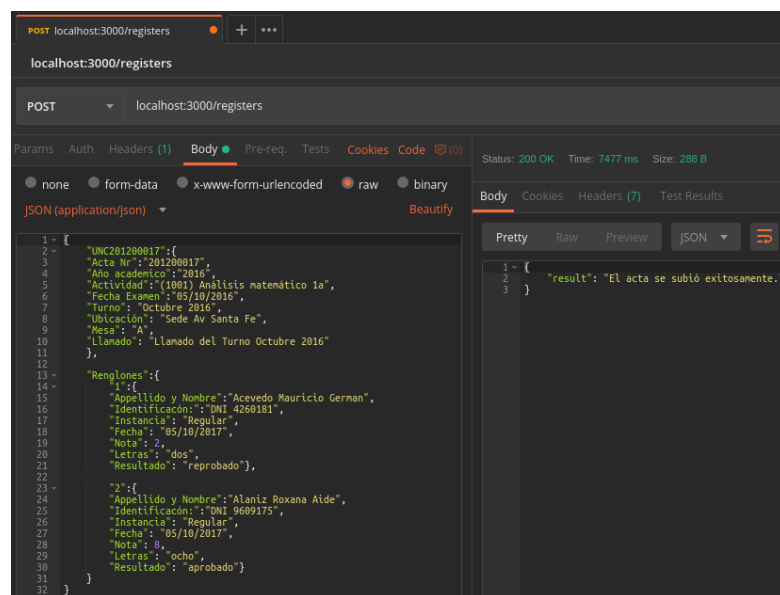


FIGURA 5.11: Subida exitosa de un acta a través de la API.

### 5.1.3.2. Desarrollo de la función *query*

La función *query* está pensada para poder comprobar el funcionamiento correcto del sistema, no para el uso del personal administrativo. Su objetivo es recuperar el acta como se agregó al Blockchain haciendo uso del *chaincode queryRegister*. En la figura 5.12 es posible ver la llamada al Blockchain junto a su respuesta constando de la información previamente subida.

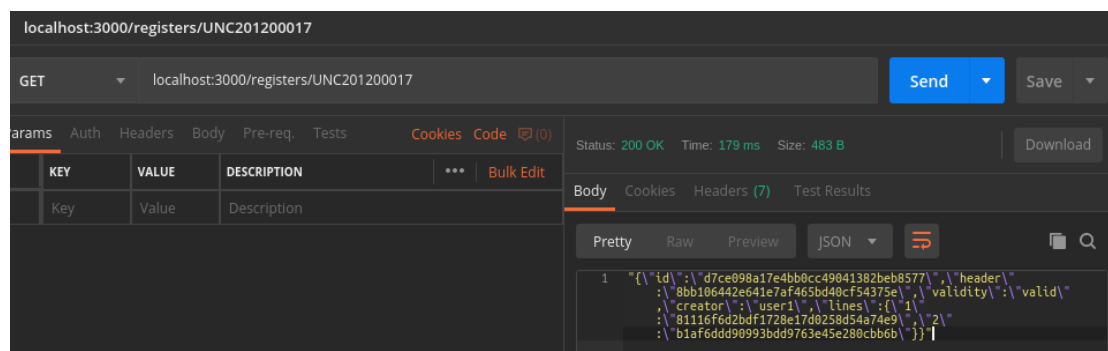


FIGURA 5.12: Consulta de un acta por Postman.

En el caso de consultar por un id inexistente, la API responde con el mensaje “Acta no encontrada.”

### 5.1.3.3. Desarrollo de la función *check\_validity*

Luego de haber subido un acta al Blockchain, ya es posible implementar la función que verifique la validez de una nota. La secuencia de pasos a implementar se puede ver en la figura 5.13.

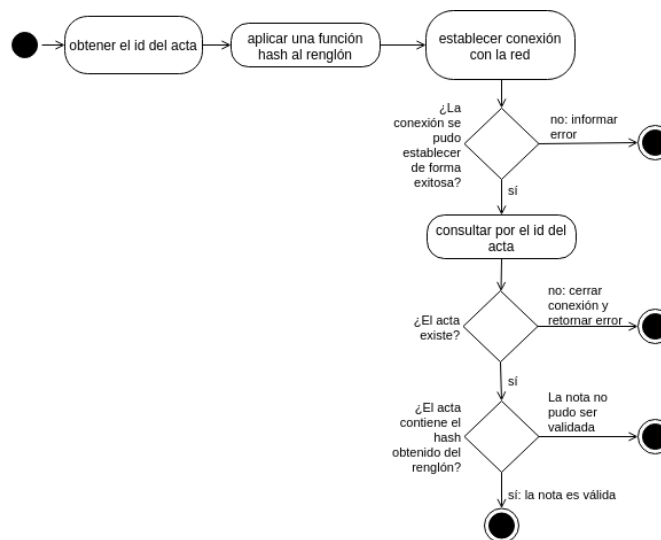


FIGURA 5.13: Diagrama de actividad para el caso de validación de una nota.

En primer lugar, se envía un objeto JSON a la API que contiene el id del acta junto al renglón que se quiere validar. La API aplica la misma función *hash* al renglón que fue aplicada en el momento de subir el acta. Luego se establece una conexión con la red y se consulta por el acta en cuestión. Si éste existe, se busca el *hash* recientemente calculado entre los renglones del acta almacenada en el Blockchain. Si existe una coincidencia, la nota es válida, en caso contrario, la nota se declara inválida. En la figura 5.14, se ve el caso de la validación exitosa de la nota de un alumno ficticio con el nombre Mauricio German Acevedo.

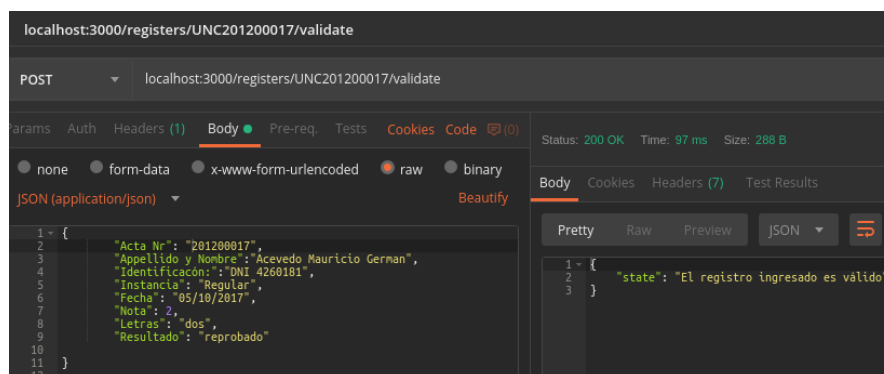


FIGURA 5.14: Validación exitosa de una nota.

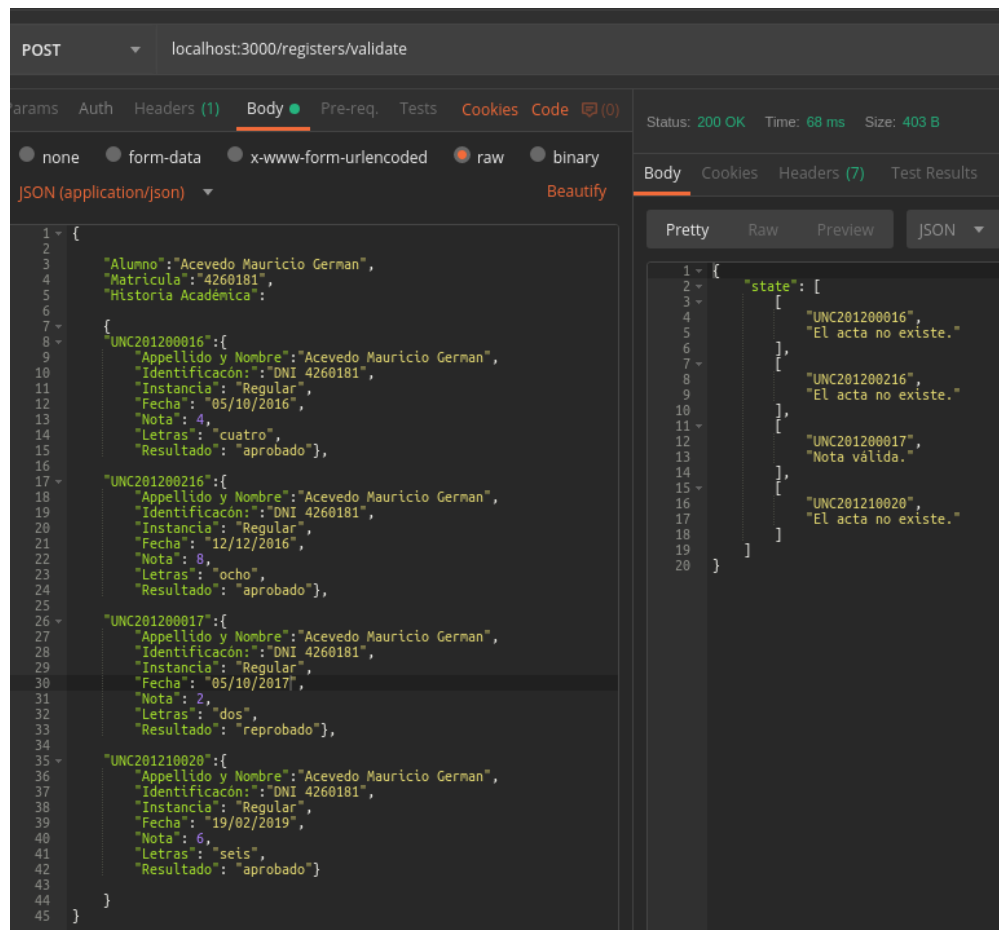


FIGURA 5.15: Validación de una historia académica.

#### 5.1.3.4. Desarrollo de la función *verify\_history*

La función que verifica la validez de una historia académica funciona de forma muy parecida a la que verifica la validez de una sola nota: La API itera sobre los renglones, aplica la función *hash* a cada uno y busca una coincidencia con los *hash* presentes en las actas obtenidos. La figura 5.15 muestra la consulta realizada junto con el resultado que la API entrega: Como hasta el momento solo se agregó un acta al Blockchain, tres de los cuatro actas no fueron encontradas. La historia académica completa solamente puede ser considerada válida si la totalidad de respuestas indica notas válidas.

#### 5.1.3.5. Desarrollo de la función *add\_revoked*

Ahora que ya es posible subir actas y validar notas, falta la implementación del requerimiento que permite revocar un acta. El diagrama de actividad para dicha función se puede ver en la figura 5.16. Para demostrar el flujo con un ejemplo, se va a asumir que ocurrió un error al reprobado el alumno Mauricio German Acevedo en el acta 201200017 y que es necesario corregir la nota por un cuatro. La corrección afectaría a un solo renglón de un total de 2 renglones del acta. Un objeto JSON muestra la información que se envía a la API para invocar las funciones correspondientes.

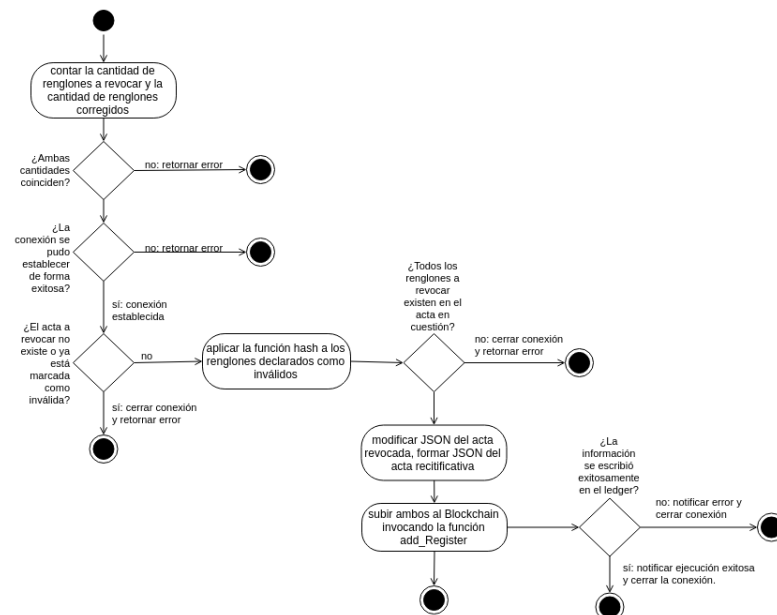


FIGURA 5.16: Diagrama de actividad para revocar un acta.

```

1  {   "Acta Nr":"201200017",
2
3     "revocar":{
4
5         "1":{
6             "Apellido y Nombre":"Acevedo Mauricio German",
7             "Identificación":"DNI 4260181",
8             "Instancia": "Regular",
9             "Fecha": "05/10/2017",
10            "Nota": 2,
11            "Letras": "dos",
12            "Resultado": "reprobado"}
13    },
14    "corregido":{
15        "1":{
16            "Apellido y Nombre":"Acevedo Mauricio German",
17            "Identificación":"DNI 4260181",
18            "Instancia": "Regular",
19            "Fecha": "05/10/2017",
20            "Nota": 4,
21            "Letras": "cuatro",
22            "Resultado": "aprobado"}
23    }
24 }

```

Los pasos que efectúa la API son los siguientes:

1. Se verifica que la cantidad de renglones a revocar coincida con la cantidad de renglones corregidos, caso contrario quedarían renglones revocados sin corregir o se corregirían renglones que no fueron revocados.
2. Después de que la API pudo establecer una conexión con la red, verifica que el acta a revocar existe y es válida. Si no sería posible crear un acta rectificativa para un acta inexistente o una que ya tiene acta rectificativa, en cuyo caso existirían dos actas rectificativas para una sola acta revocada. En el contexto del proyecto presente, solamente es posible corregir un acta una sola vez.
3. Si es posible revocar el acta, se aplica la función *hash* a todos los renglones: los que se van a revocar y los que van a corregir los revocados. Se verifica que los renglones a revocar realmente existen en el acta en cuestión, ya que si no fuera el caso, se estaría revocando información inexistente.
4. Si todas las verificaciones anteriores resultaran exitosas, se genera el acta nueva reemplazando los *hash* de los renglones revocados por los *hash* de los renglones corregidos. El id de cualquier acta rectificativa inicia con las letras RECT, para poder distinguirlo de actas que no fueron corregidas.

Por otro lado, se modifica el acta revocada, cambiando el valor de la clave “*validity*” por “*invalid*” y agregando una clave “*revoked lines*” que lleva como valor los *hash* de las líneas que fueron revocadas. Es necesario recordar que en el caso de una consulta al *ledger* con la función *query*, el *peer* consulta al *world state*, que siempre indica el valor más actual de un objeto almacenado. Una vez que se revocó el acta 201200017, una consulta siempre va recuperar el acta con estado inválido. Si se consulta por el historial de la misma acta, va a haber dos entradas: el original y luego el revocado.

5. Por último, se actualiza el acta revocada y se agrega el acta rectificativa al Block-chain.

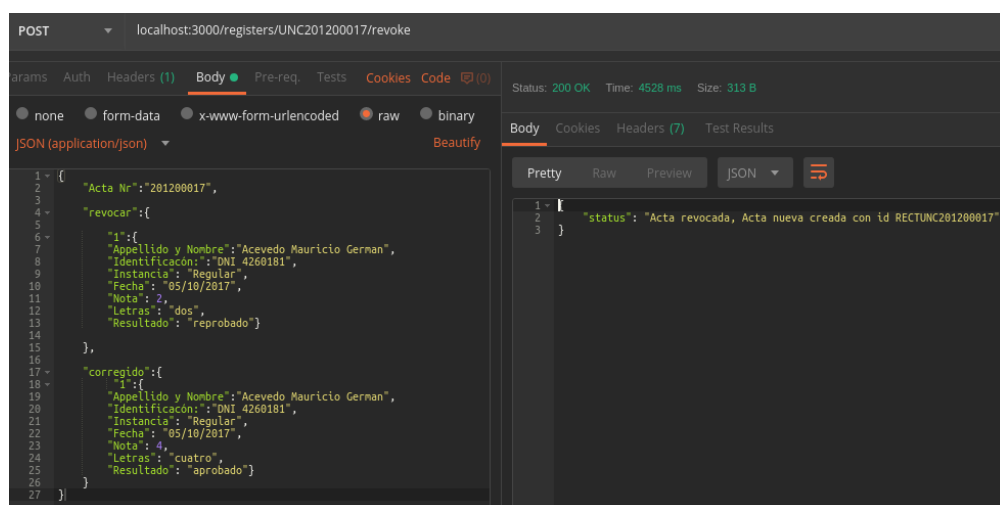


FIGURA 5.17: Creación de un acta rectificativa.



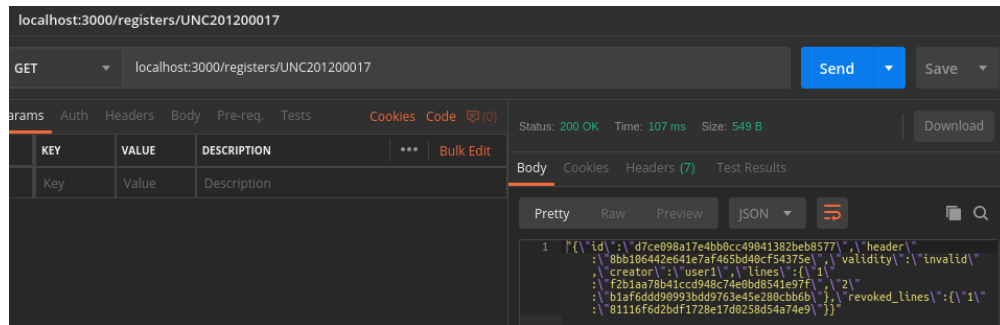


FIGURA 5.18: Consulta de un acta inválida.

En la figura 5.17 se puede ver la respuesta de la API en el caso de una corrección exitosa. Se indica el id del acta rectificativa. Si ahora se hace una consulta a `localhost:3000/registers/UNC201200017` (figura 5.18), se puede ver que el estado es inválido y se agregó la clave “revoked\_lines”, de la cual se habló anteriormente.

Como última prueba es necesario corroborar que las actas revocadas no se tienen en cuenta a la hora de validar una historia académica: En la figura 5.19 se puede observar la consulta por la validez de la historia académica de Mauricio German Acevedo con el aplazo corregido en el acta 201200017. La nota aparece válida, lo cual indica que el sistema está teniendo en cuenta de manera correcta las actas revocadas y sus actas rectificativas.

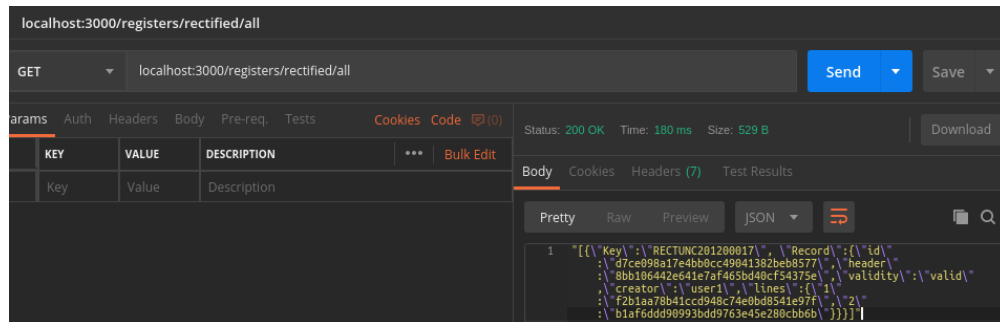


FIGURA 5.20: Consulta por todas las actas rectificativas.

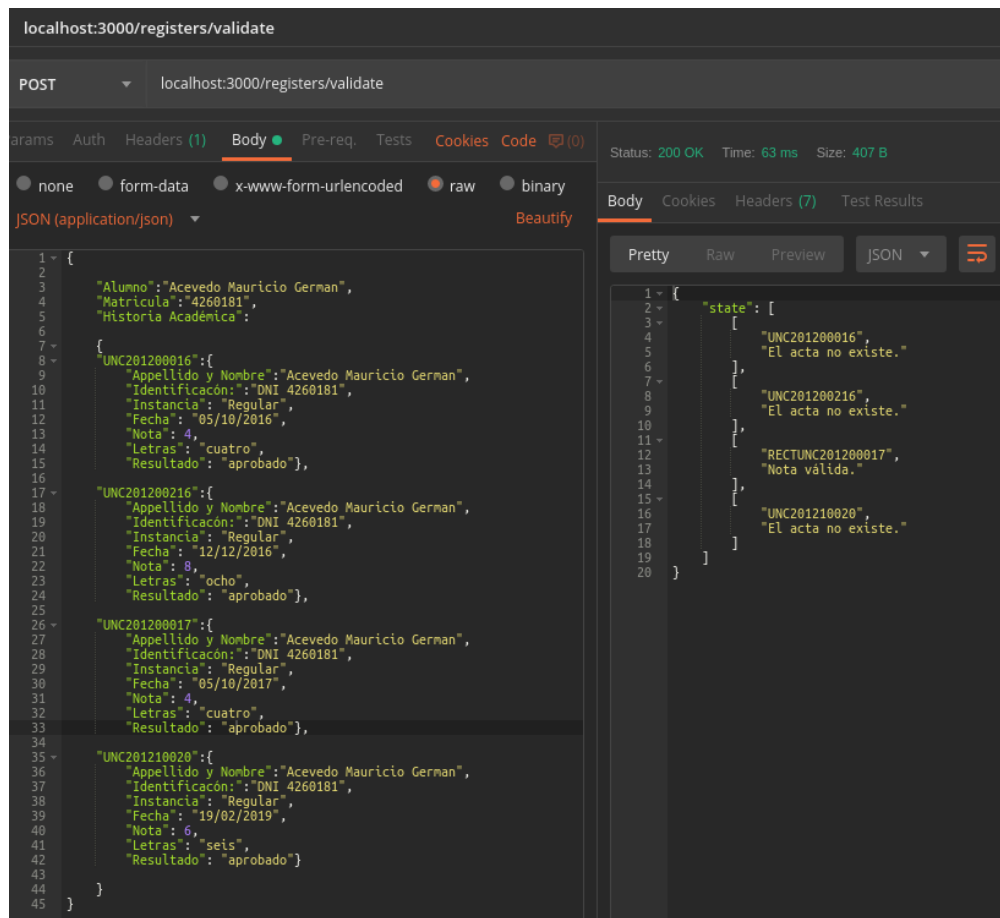


FIGURA 5.19: Validación de una historia académica involucrando un acta rectificativa.

### 5.1.3.6. Desarrollo de la función *query\_rectified*

La función es implementada de la misma forma que *query*, pero invoca al *chaincode queryAllRectified*, el cual tiene la particularidad de filtrar todas las actas con el prefijo “RECT”. En la captura 5.20, aparece solo un acta, porque en el momento de la prueba no se habían cargado y revocado más actas, sin embargo siempre figurará la cantidad correspondiente a las actas rectificadas.

#### 5.1.4. Implementación del Frontend con Angular e Ionic

Luego de las instalaciones de Angular e Ionic descritas en la sección A.3 se procedió a eliminar el contenido por defecto que lleva la aplicación de prueba. Luego, se siguieron los siguientes pasos de desarrollo:

- Con la ayuda del comando

```
$ ionic generate page <pagename>
```

se generaron en total 6 páginas, una por cada caso de uso.

- El objetivo de la página principal, *home*, es proveer una visión general de las funcionalidades de la implementación. Para eso se utilizó la estructura *ion-grid*, donde cada cuadrícula presenta uno de los casos de uso descritos. Se agregaron imágenes descriptivas para facilitar la navegación por la página. En la figura 5.21, se puede ver dicha implementación.

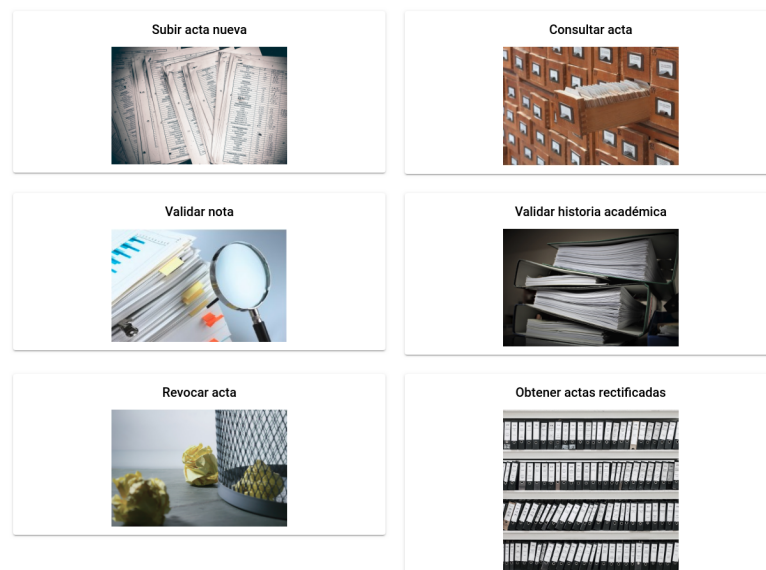


FIGURA 5.21: *ion-grid* con los 6 casos de uso.

- Al hacer clic en cada una de las cuadrículas, el usuario navega hacia la página correspondiente al caso de uso elegido. Ahí es presentado con un campo para una entrada de usuario, implementada mediante el componente *ion-input*. Como en la mayoría de los casos la entrada del usuario tiene que tener formato JSON para ser enviada a la API, se verifica que los datos aportados realmente tengan dicho formato. En caso contrario, indica un error mediante el componente *ion-toast* y la API no es invocada. Dicho caso se puede ver en la figura 5.22

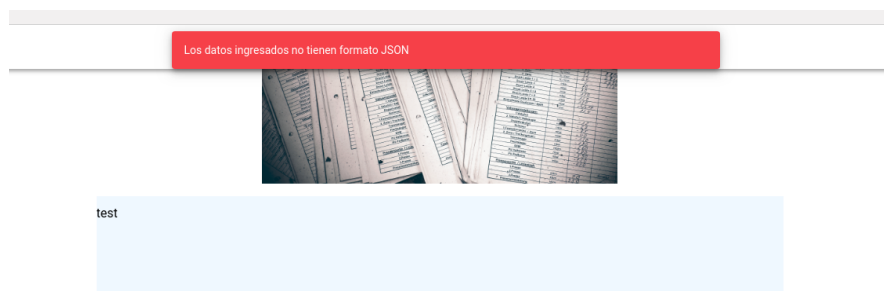


FIGURA 5.22: Un *ion-toast* que indica que la entrada de usuario no se encuentra en formato JSON.

- Para notificar al usuario si los datos ingresados se escribieron exitosamente en el Blockchain, se empleó el componente *ion-alert*. Dicho elemento abre una pequeña ventana que indica el éxito o fracaso de una operación y provee un mensaje de error si es necesario. En la figura 5.23 se muestran dos posibles alertas: en el primer caso, un acta se agregó exitosamente al Blockchain. Al querer subir la misma acta nuevamente, se produce el error visible a la derecha: si bien se pudo acceder al Blockchain, no fue posible escribir en él ya que el acta existía anteriormente.

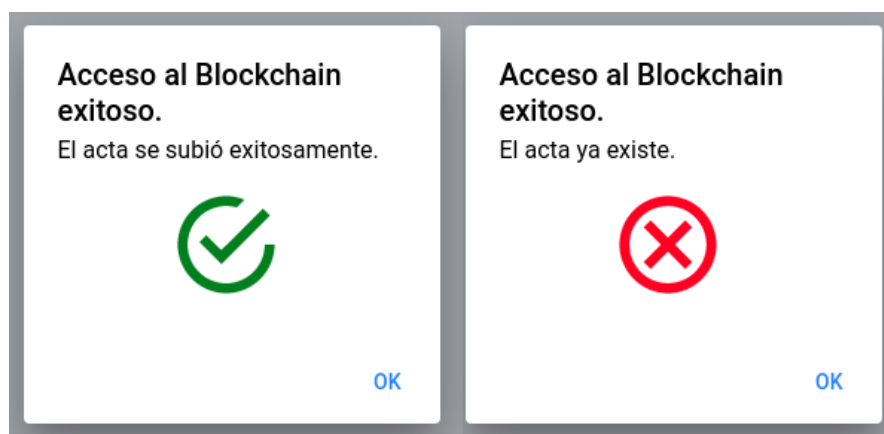


FIGURA 5.23: Alertas de notificación sobre el éxito o fracaso de una operación.

- Para que los usuarios que desconocen el sistema puedan utilizar el mismo, se generó una página adicional de ayuda, que provee instrucciones y ejemplos.
- Por último, se agregó la funcionalidad de traducción de la página web con el módulo `@ngx-translate/core`. En la parte superior se encuentran la bandera española junto a la bandera estado-unidense. Seleccionar la bandera a elección modifica el idioma en el que aparece la página.

## Capítulo 6

# Pruebas y Validación

En el presente capítulo se describen los procedimientos de prueba para los requerimientos descritos en el cuadro 4.1. Cada uno de los cuadros visibles a continuación describe el objetivo de la prueba, el procedimiento, el resultado esperado y el estado final, que indican si la prueba se realizó de forma exitosa o no. Por último, en el capítulo se encuentra una matriz de trazabilidad que permite relacionar los requerimientos de usuario con los requerimientos funcionales y asociar cada uno con su caso de prueba correspondiente.

### 6.1. Validación de los requerimientos de usuario

Id	TC-01
Título	Creación de un acta nueva en el Blockchain
Objetivo	Demostrar el funcionamiento correcto de la carga de actas al Blockchain.
Procedimiento	<ol style="list-style-type: none"><li>1. El usuario debe dirigirse hacia la página y hacer clic en el cuadro con el título “Subir acta nueva”.</li><li>2. Luego debe ingresar el acta en el campo de texto previsto y hacer clic en el botón “Subir acta”.</li><li>3. Después de un tiempo de espera una alerta indicará si el acta pudo ser subida exitosamente o no.</li></ol>
Resultados esperados	Luego de un tiempo de espera, un cartel indicará el éxito de la operación. En el caso que el acta fue subida anteriormente y ya existe, indicará con un mensaje de error que el acta no pudo ser agregada.
Estado	Aprobado

CUADRO 6.1: Caso de prueba TC-01

Id	TC-02
Título	Consulta de un acta en el Blockchain
Objetivo	Recuperar el acta del Blockchain con sus renglones en formato de <i>hash</i> , tal como se subió.
Procedimiento	<ol style="list-style-type: none"><li>1. El usuario debe dirigirse hacia la página y hacer clic en el cuadro con el título “Consultar acta”.</li><li>2. Luego debe ingresar el id del acta en el campo de texto previsto y hacer clic en el botón “Obtener”.</li></ol>
Resultados esperados	En el caso de que el acta exista, el contenido de la misma aparece debajo del botón, indicando los campos de la cabecera y los renglones. En el caso de no existir el acta, una alerta proporciona dicha información al usuario. Capturas de ambos casos se encuentran en las figuras 6.1 y 6.2.
Estado	Aprobado

CUADRO 6.2: Caso de prueba TC-02

UNC201200017

OBTENER

Cabecera

Hash del Id: d7ce098a17e4bb0cc49041382beb8577

Hash de la cabecera: 8bb106442e641e7af465bd40cf54375e

Validez: invalid

Creador: user1

Renglones

0: f2b1aa78b41ccd948c74e0bd8541e97f

1: b1af6ddd90993bdd9763e45e280cbb6b

FIGURA 6.1: Alertas de notificación sobre el éxito o fracaso de una operación.

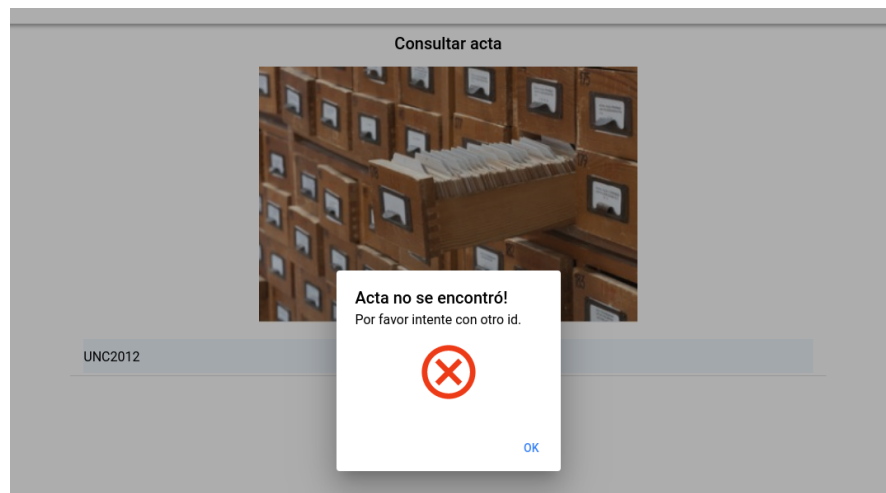


FIGURA 6.2: Alertas de notificación sobre el éxito o fracaso de una operación.

Id	TC-03
Título	Validación de una nota con el Blockchain
Objetivo	Obtener información sobre la validez o invalidez de una nota.
Procedimiento	<ol style="list-style-type: none"> <li>1. El usuario debe dirigirse hacia la página y hacer clic en el cuadro con el título “Validar nota”.</li> <li>2. Luego debe ingresar toda la información correspondiente en el campo de texto previsto y hacer clic en “Validar”.</li> <li>3. Después de un tiempo de espera una alerta indicará si la nota es válida o no.</li> </ol>
Resultados esperados	Un cartel indica si la nota es válida, si no es válida o que el acta en el cual se tenía que encontrar la nota no existe.
Estado	Aprobado

CUADRO 6.3: Caso de prueba TC-03

Id	TC-04
Título	Validación de una historia académica
Objetivo	Obtener información sobre la validez o invalidez de una historia académica.
Procedimiento	<ol style="list-style-type: none"> <li>1. El usuario debe dirigirse hacia la página y hacer clic en el cuadro con el título “Validar historia académica”.</li> <li>2. Luego debe ingresar toda la información correspondiente en el campo de texto previsto y hacer clic en “Validar”.</li> <li>3. Después de un tiempo de espera, la página indicará si la historia académica es válida o no.</li> </ol>
Resultados Esperados	Como en el caso de la validación de una sola nota, un cartel indica la validez o invalidez de la historia académica. Por debajo del botón “Validar” aparece una lista con los ids de las actas que se consultó y el resultado por cada acta. En la figura 6.3 se puede ver un ejemplo donde no se logró validar una historia académica, junto con el detalle de los errores de las notas que no se lograron validar.
Estado	Aprobado

CUADRO 6.4: Caso de prueba TC-04

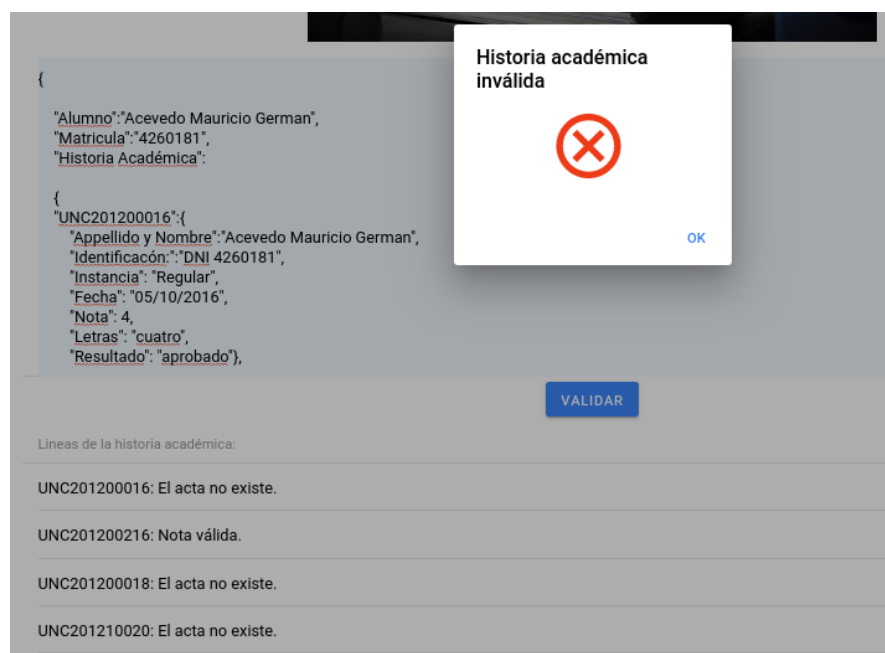


FIGURA 6.3: Caso de una historia académica inválida.



Id	TC-05
Título	Revocar un acta
Objetivo	Corroborar el funcionamiento correcto del sistema al revocar una acta.
Procedimiento	<ol style="list-style-type: none"> <li>1. El usuario debe dirigirse hacia la página y hacer clic en el cuadro con el título “Revocar acta”.</li> <li>2. Luego debe ingresar toda la información correspondiente en el campo de texto previsto y hacer clic en el botón “Revocar acta”.</li> <li>3. Después de un tiempo de espera, la página indicará si el acta pudo ser revocada o no.</li> </ol>
Resultados Esperados	En el caso de una revocación inválida, es decir, en el caso que el acta a revocar no existe o ya es inválida, el sistema devuelve un error. Si se puede revocar al acta, el sistema avisa después de un tiempo de espera que el acta pudo ser revocada correctamente.
Estado	Aprobado

CUADRO 6.5: Caso de prueba TC-05

Id	TC-06
Título	Consulta de las actas rectificadas
Objetivo	Recuperar información sobre todas las actas rectificadas existentes en el Blockchain.
Procedimiento	<ol style="list-style-type: none"> <li>1. El usuario debe dirigirse hacia la página y hacer clic en el cuadro con el título “Obtener actas rectificadas”.</li> <li>2. En la página, debe hacer clic en el botón “Obtener actas rectificadas”.</li> </ol>
Resultados esperados	En el caso de no existir ningún acta rectificada, la página muestra un texto indicando lo mismo. Si existen actas rectificadas, en una lista se detallan id, <i>hash</i> del id, <i>hash</i> de la cabecera, la validez del acta como el usuario que lo creó.
Estado	Aprobado

CUADRO 6.6: Caso de prueba TC-06

Id	TC-07
Título	Presencia de Organizaciones y CA's
Objetivo	Verificar que existen los <i>peers</i> y autoridades de certificación especificados en los requerimientos.
Procedimiento	<ol style="list-style-type: none"> <li>1. En la máquina de desarrollo local, ejecutar el comando  <code>\$ docker ps</code></li> <li>2. En el cluster de Kubernetes, ejecutar el comando  <code>\$ kubectl get pods</code></li> </ol>
Resultados esperados	Los contenedores o <i>pods</i> que están presentes se pueden ver también en la figura 6.4: Se pueden encontrar <i>peer0</i> como <i>peer1</i> de cada organización y los tres contenedores que funcionan como una autoridad de certificación.
Estado	Aprobado

CUADRO 6.7: Caso de prueba TC-07

```
eli@msi:~$ docker ps --format "table {{.Image}}\t{{.Status}}\t{{.Names}}"
IMAGE                                STATUS      NAMES
hyperledger/fabric-tools:1.4        Up 2 minutes    cli
hyperledger/fabric-orderer:1.4      Up 2 minutes    orderer.pi.elisabeth.com
hyperledger/fabric-peer:1.4         Up 2 minutes    peer0.famaf.pi.elisabeth.com
hyperledger/fabric-peer:1.4         Up 2 minutes    peer1.fcq.pi.elisabeth.com
hyperledger/fabric-peer:1.4         Up 2 minutes    peer0.fcq.pi.elisabeth.com
hyperledger/fabric-peer:1.4         Up 2 minutes    peer1.famaf.pi.elisabeth.com
hyperledger/fabric-ca:1.4           Up 2 minutes    ca-famaf
hyperledger/fabric-peer:1.4         Up 2 minutes    peer1.fcefyn.pi.elisabeth.com
hyperledger/fabric-ca:1.4           Up 2 minutes    ca-fcefyn
hyperledger/fabric-ca:1.4           Up 2 minutes    ca-fcq
hyperledger/fabric-peer:1.4         Up 2 minutes    peer0.fcefyn.pi.elisabeth.com
```

FIGURA 6.4: Los contenedores que deben estar presentes para cumplir con los requerimientos RF-BC-01 y RF-BC-03.

Id	TC-08
Título	Traducción de la página al inglés
Objetivo	Verificar que el <i>frontend</i> está disponible en los idiomas inglés y español.
Procedimiento	<ol style="list-style-type: none"> <li>1. En la máquina de desarrollo local, ingresar la URL donde está funcionando la aplicación de Angular, en este caso, <i>http://localhost:8100/home</i>.</li> <li>2. En la esquina superior derecha de la página, se encuentran una bandera española y una bandera estadounidense. El idioma por defecto es español: Para cambiar al inglés, es necesario hacer clic en la bandera estadounidense.</li> </ol>
Resultados esperados	El idioma de toda la página cambia del español al inglés y vice versa al elegir la bandera correspondiente. En la figura 6.5 puede ver el cambio para el caso del título en la página principal.
Estado	Aprobado

CUADRO 6.8: Caso de prueba TC-08



FIGURA 6.5: Prueba de cambio de idioma en la página principal.

Id	TC-09
Título	Prueba de la página de ayuda
Objetivo	Comprobar que un usuario sin conocimiento previo del uso de la página sea capaz de utilizarla.
Procedimiento	<ol style="list-style-type: none"> <li>1. Guiar al usuario a la página de ayuda y pedir que lea cuidadosamente las instrucciones.</li> <li>2. Pedir al usuario que agregue un acta nueva según las instrucciones y que luego valide una nota de dicha acta.</li> </ol>
Resultados Esperados	Suponiendo el correcto funcionamiento de frontend, API y Blockchain, el usuario no debe tener ningún inconveniente en realizar lo pedido. Para el caso de personas que no tienen ningún conocimiento previo de programación, se anticipan confusiones con la redacción del acta en formato JSON.
Estado	Aprobado

CUADRO 6.9: Caso de prueba TC-09

## 6.2. Matriz de trazabilidad

En la matriz visible a continuación se relacionan todos los requerimientos planteados con los casos de test descritos en la sección anterior, con la finalidad de obtener una visión general que todos los requerimientos fueron testeados exitosamente. Los requerimientos no funcionales RNF-01 (El sistema debe contar con pruebas y tests integrales de los seis requerimientos de usuario.) y RNF-02 (El código del *chaincode* debe estar escrito en Golang, la API debe desarrollarse en Node.js y para la interfaz gráfica se debe usar Typescript.) no figuran en la tabla, ya que establecen metodologías y herramientas de trabajo.

	RF-BC-01	RF-BC-02	RF-BC-03	RF-BC-04	RF-BC-05	RF-BC-06	RF-API-01	RF-API-02	RF-API-03	RF-API-04	RF-API-05	RF-API-06	RF-GUI-01	RF-GUI-02	RF-GUI-03	RF-GUI-04	RF-GUI-05	RNF-03	RNF-04
TC-01		✓		✓	✓	✓	✓					✓	✓	✓		✓	✓	✓	
TC-02				✓		✓		✓				✓	✓	✓		✓	✓	✓	
TC-03				✓		✓			✓			✓	✓	✓		✓	✓	✓	
TC-04				✓		✓				✓		✓	✓	✓		✓	✓	✓	
TC-05		✓		✓	✓	✓					✓	✓	✓	✓		✓	✓	✓	
TC-06				✓		✓		✓				✓	✓	✓		✓	✓	✓	
TC-07	✓		✓																
TC-08															✓				
TC-09																			✓

CUADRO 6.10: Matriz de trazabilidad

## Capítulo 7

# Conclusión

En el presente proyecto, se realizó un estudio profundo de la tecnología Blockchain, sus características más importantes como también sus aplicaciones y casos de uso. Se comparó una multitud de plataformas Blockchain diferentes. Luego, se eligió una plataforma de las analizadas para probar su funcionamiento en un caso práctico que no involucrara la necesidad de una criptomoneda. Se investigó la utilidad de un sistema de validación de actas académicas y se llegó a la conclusión que un sistema con arquitectura Blockchain puede superar en utilidad a un sistema de base de datos tradicional.

Después de la etapa de investigación se procedió a desarrollar un prototipo con el fin de evaluar la implementabilidad y utilidad de la propuesta. Para eso, el proyecto se dividió en tres partes:

1. La primera parte consiste en la configuración de la red Blockchain con el framework elegido. Se configuraron 3 organizaciones diferentes con 2 nodos cada una, que comparten el mismo canal y usan el mecanismo de aprobación de transacciones *solo*. Ambos nodos por cada organización funcionan como nodos de aprobación de transacciones y un nodo por cada organización se configuró adicionalmente como *anchor node* para posibilitar el descubrimiento de nodos de otras organizaciones. El desarrollo del *chaincode* permite agregar y modificar información al Blockchain, una vez que toda la red fue iniciada.
2. La segunda parte consiste en la programación de la API con la ayuda del SDK previsto por Hyperledger Fabric. Se implementaron los diferentes *endpoints* que invocan al *chaincode* correspondiente, el cual consulta por información existente o agrega información nueva. Se implementaron las siguientes funcionalidades:
  - a) Subir un acta nueva.
  - b) Consultar por un acta subida.
  - c) Validar una nota.
  - d) Validar una historia académica.
  - e) Rectificar un acta.

f) Consultar por todas las actas rectificadas.

3. En la última parte, se desarrolló una aplicación web para que el usuario pueda interactuar con el sistema a través de una interfaz gráfica. Con Angular, se crearon diferentes páginas que interactúan con la API y dan una retroalimentación visual al usuario para indicar si su operación fue exitosa o fracasó.

## 7.1. Problemas y Limitaciones

A continuación, se elaboran algunos problemas y limitaciones que se presentaron durante el desarrollo del proyecto.

- En el momento de la investigación, resultó difícil encontrar bibliografía. Dado que la primera implementación de una arquitectura Blockchain data en el 2009, es un tema reciente, y no existen libros que discuten el tema con la profundidad requerida. Por otro lado, especialmente gracias a las criptomonedas, es un tema polémico que atrae mucha atención, y se encuentran muchas fuentes con información errónea. Por esa razón, se aplicaron criterios rigurosos para su elección: la mayoría de la información fue extraída de papers científicos, blogs de fundadores de implementaciones Blockchain y de la documentación de dichas implementaciones.
- Una vez elegido el *framework* Hyperledger Fabric para la implementación del trabajo, se presentaron una multitud de problemas y dudas que solo se lograron resolver en parte: Si bien se encontraron preguntas relacionadas en foros como Stackoverflow, para muchas existen pocas respuestas o ninguna. Para esos casos, fue necesario asumir o experimentar con ejemplos hasta llegar a una conclusión satisfactoria.
- Todos los proyectos de Hyperledger son gestionados por la Linux Foundation y son proyectos de código libre, donde se modifican y agregan funcionalidades constantemente. *Bugs* en versiones que se usaron en este proyecto se resolvieron al migrar a versiones posteriores, las cuales, sin embargo, introducían funcionalidades desconocidas que todavía no contaban con la documentación necesaria y a veces introducían otros problemas o incompatibilidades imprevistas.

## 7.2. Trabajos futuros

A continuación, figuran algunas mejoras y extensiones al presente proyecto, que no se implementaron por limitaciones de tiempo o porque su desarrollo se desviaba del objetivo inicial del proyecto integrador.

Mejoras propuestas en la parte de interfaz gráfica y API:

- Agregar autenticación de usuarios a la aplicación web, para que solamente personal autorizado pueda agregar y revocar actas.

- Junto con la autenticación, implementar una limitación para que un acta solamente puede ser rectificado por el mismo usuario que lo creó originalmente.
- Implementar que cada acta del sistema sea firmada digitalmente por su creador.
- Desarrollar una planilla en la interfaz gráfica que permita ingresar los datos requeridos a través de un formulario en vez de usar el formato JSON. Para un usuario poco experimentado con la computación, respetar el formato JSON puede resultar engorroso.
- Para una adopción más fácil de la tecnología y mitigar el riesgo de un usuario vacilante, también es posible automatizar la obtención de los datos del alumno, de tal manera que la aplicación utilice una API del sistema Guaraní para obtener la información en texto plano, en vez de requerir su ingreso manual en formato JSON. Sin embargo, es primordial tener en cuenta que el alumno antes debe autorizar dicho acceso, ya que su información no debe ser difundida sin su consentimiento.

Mejoras propuestas en la parte de la red Blockchain:

- Desplegar la red con más que un nodo de aprobación de transacciones para lograr redundancia usando Raft o Kafka.
- Desplegar una mayor cantidad de nodos por organización: Para ambientes de producción, Hyperledger Fabric recomienda un mínimo de tres nodos por organización, de los cuales dos deben estar configurados como *anchor peers*.
- Generar las identidades digitales a través de las CA en vez de utilizar la herramienta cryptogen, ya que ésta predefine la topología de la red y una modificación posterior resulta complicada.
- Realizar un análisis de costos de un despliegue en producción.

### 7.3. Aporte personal

El conocimiento adquirido a lo largo de la carrera, especialmente en las materias de los últimos años como Bases de Datos, Redes de Computadoras y Sistemas Operativos, formó una base de conocimiento imprescindible para el desarrollo del proyecto. Si bien los paradigmas en cuestión recién cumplen 10 años, los problemas que se plantearon pudieron ser abarcados gracias a la preparación recibida en la facultad junto con paciencia, perseverancia y curiosidad.

## Apéndice A

# Configuración del Ambiente de Desarrollo

La instalación de la totalidad de las herramientas se efectuó en el sistema operativo Ubuntu 16.04LTS. Las características de la máquina con la que se trabajó durante el desarrollo local figuran en [A.1](#)

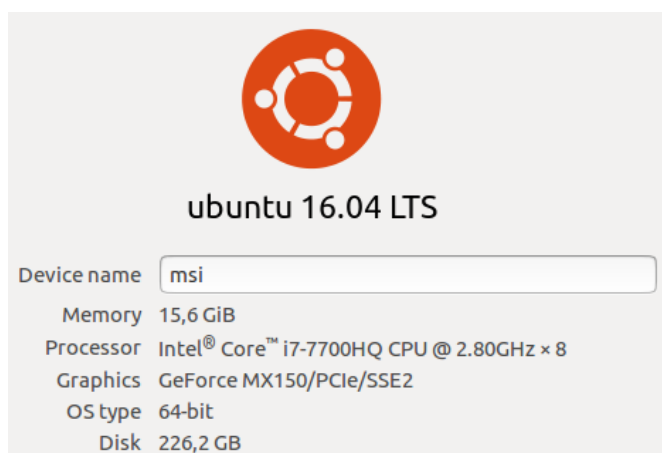


FIGURA A.1: Características técnicas de la computadora de desarrollo.

### A.1. Instalación y Configuración del Entorno para Hyperledger Fabric

Para configurar el ambiente de desarrollo de Hyperledger Fabric, se siguieron las indicaciones de la página web <https://hyperledger-fabric.readthedocs.io/en/release-1.4/prereqs.html>.

- **Instalación de curL**

Para la instalación de curL es necesario ejecutar los comandos siguientes en la consola de linux:



```
$ sudo apt-get update
$ sudo apt-get install curl
```

La versión instalada es 7.47.0.

#### ■ Instalación de Docker y Docker Compose

Para la instalación de Docker y Docker Compose se siguieron las instrucciones de la página web oficial <https://docs.docker.com/install/>.

Se optó por una instalación a través del repositorio de Docker. La configuración se realiza con los siguientes comandos:

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates
    gnupg-agent software-properties-common
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
↪ apt-key add -
$ sudo add-apt-repository "deb [arch=amd64]
↪ https://download.docker.com/linux/ubuntu $(lsb_release
↪ -cs)         stable"
```

Una vez que se agregó el repositorio, se puede proceder con la instalación:

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

Luego de completar la instalación, se ejecuta el comando de prueba para asegurar que la instalación se efectuó de forma exitosa. La figura A.2 indica que Docker está funcionando correctamente.

```
$ sudo docker run hello-world
```

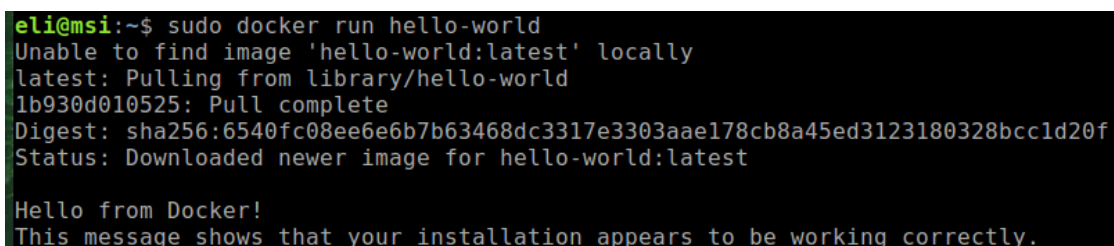
A terminal window with a black background and green text. The prompt is 'eli@msi:~\$'. The command 'sudo docker run hello-world' is entered. The output shows that Docker is pulling the 'hello-world:latest' image from the library. It displays the image ID '1b930d010525', the pull status 'Pull complete', the digest 'sha256:6540fc08ee6e6b7b63468dc3317e3303aae178cb8a45ed3123180328bcc1d20f', and the status 'Downloaded newer image for hello-world:latest'. Finally, it prints 'Hello from Docker!' and a message: 'This message shows that your installation appears to be working correctly.'

FIGURA A.2: Prueba de la instalación exitosa de Docker

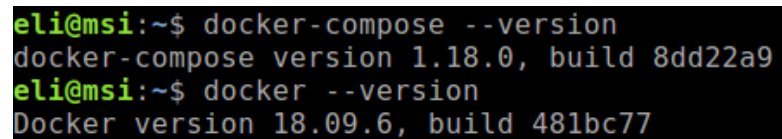
Para no tener que ejecutar todos los comandos de Docker con sudo, se creó el grupo docker y se agregó el usuario “eli” al mismo:

```
$ sudo groupadd docker
$ sudo usermod -aG docker eli
```

Por último, se ejecutaron los siguientes comandos con el fin de instalar Docker Compose según la guía de instalación de la página oficial <https://docs.docker.com/compose/install/>:

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.18.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
```

Por último se verificó la configuración correcta de ambas herramientas consultando por sus respectivas versiones. La figura A.3 indica que todo se configuró correctamente.



```
eli@msi:~$ docker-compose --version
docker-compose version 1.18.0, build 8dd22a9
eli@msi:~$ docker --version
Docker version 18.09.6, build 481bc77
```

FIGURA A.3: Prueba de la instalación exitosa de Docker Compose

#### ■ Instalación de Golang

Según la documentación, Hyperledger Fabric requiere la versión 1.11.x de Go. Para su instalación, se siguió el tutorial de la página oficial <https://golang.org/doc/install>. En primer lugar, se descargó el archivo `go1.11.12.linux-amd64.tar.gz` de la página <https://golang.org/dl/> y se descomprimió con el siguiente comando en la consola:

```
$ tar -C /usr/local -xzf go1.11.12.linux-amd64.tar.gz
```

Para el funcionamiento correcto del chaincode, se agregaron las siguientes variables de entorno al archivo `.bashrc`:

```
$ export GOROOT=/usr/local/go
$ export GOPATH=/home/eli/.go
$ export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
```

#### ■ Instalación de NodeJS y NPM

Para la instalación se siguieron las instrucciones en la página web oficial: <https://github.com/nodesource/distributions/blob/master/README.md#debinstall>, la cual indica ejecutar los siguientes comandos:

```
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
$ sudo apt-get install -y nodejs
```

Se optó por la versión 8.9.4 ya que la documentación de Hyperledger Fabric especifica dicha versión como una versión compatible con su SDK. Junto con NodeJS también se instala el software `npm` (*node package manager*) el gestor de paquetes de NodeJs.

#### ■ Instalación de Hyperledger Fabric

Para obtener las herramientas y los ejemplos de Hyperledger Fabric, es necesario clonar el repositorio y luego ejecutar el script `bootstrap.sh`, que se encuentra en la carpeta `scripts` del repositorio colonado.

```
$ git clone https://github.com/hyperledger/fabric-samples.git
$ cd fabric-samples/scripts && ./bootstrap.sh
```

Al ser un software *open source* con una multitud de colaboradores, se producen muchos cambios de versión a versión, que a veces dejan obsoletos conceptos ya aprendidos o que requieren el aprendizaje de características nuevas de la herramienta. Aquí, se optó por la versión 1.4.0, ya que tiene varias mejoras con respecto a la versión 1.0, que fue la que se investigó en un principio, pero no incluye tantas modificaciones que el conocimiento adquirido sobre la versión anterior se volvió obsoleto.

Una manera de verificar que todo está funcionando correctamente, es ejecutando el *script* de prueba `byfn.sh` (por las siglas *build your first network*). El *script* crea una red de prueba, instala *chaincode* en él y efectúa transacciones. En la figura A.4 se puede observar que el ambiente local de desarrollo para Hyperledger Fabric está funcionando de forma correcta.

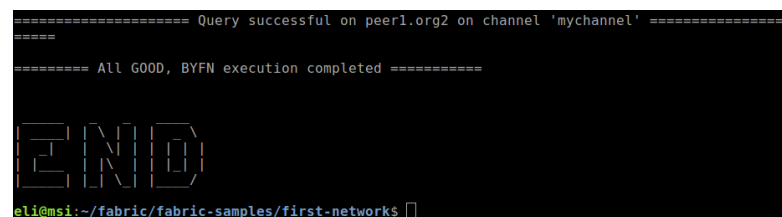


FIGURA A.4: Prueba de la instalación exitosa Hyperledger Fabric y sus requerimientos

## A.2. Instalación y Configuración del Entorno para el Desarrollo de la API

Debido a que NodeJs ya se instaló en la sección anterior, la única herramienta necesaria para esta etapa es Postman. Es un *software* que ayuda con el desarrollo de una API REST. En el contexto del proyecto integrador, se va a utilizar para realizar llamadas a la API y comprobar que las respuestas sean correctas. La herramienta se instala a través del gestor de paquetes *snap* con el comando

```
$ snap install postman
```

## A.3. Instalación y Configuración de las herramientas de frontend

Ambas herramientas, Ionic como Angular, se instalan con npm, el gestor de paquetes de NodeJs.

```
$ npm install -g @angular/cli
$ npm install -g ionic
```

Con los siguientes comandos se inicia una aplicación web nueva:

```
$ ionic start piApp  
$ ionic serve
```

El último comando abre el navegador donde se muestra una aplicación web ejemplo (figura A.5), con la cual se puede empezar a desarrollar.

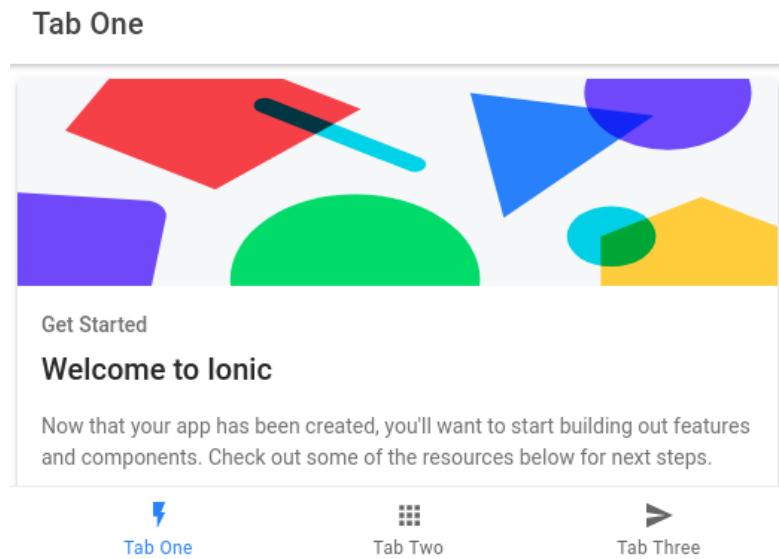


FIGURA A.5: Inicio exitoso de una aplicación web mínima

# Bibliografía

- [1] Illustration of the tangle. [Accessed: Jul. 19, 2018]. [Online]. Available: <https://www.iota.org/get-started/what-is-iota>
- [2] Hyperledger projects. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://www.hyperledger.org/projects>
- [3] Hyperledger fabric illustrations. [Accessed: August 8, 2019]. [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/peers/peers.html>
- [4] Hyperledger fabric illustrations. [Accessed: August 12, 2019]. [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.4/ledger/ledger.html>
- [5] Dhs science and technology directorate flowchart. [Accessed: December 14, 2018]. [Online]. Available: <https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf>
- [6] Ionic stack. [Accessed: August 29, 2019]. [Online]. Available: <https://ionicframework.com/what-is-ionic>
- [7] M. Montes, “Verificación de actas y certificados usando una red blockchain,” in *6to CONGRESO NACIONAL DE INGENIERÍA EN INFORMÁTICA/SISTEMAS DE INFORMACIÓN*, vol. Publicación on line - ISSN 2347-0372, 2018, [Accessed: Oct. 14, 2019]. [Online]. Available: <https://www.conaisi2018mdp.org/memorias/memorias.html#>
- [8] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, “Sok: Research perspectives and challenges for bitcoin and cryptocurrencies,” in *2015 IEEE Symposium on Security and Privacy*, May 2015, pp. 104–121.
- [9] Bitcoin energy consumption index. *Digiconomist*. [Accessed: Jul. 19, 2018]. [Online]. Available: <https://digiconomist.net/bitcoin-energy-consumption#assumptions>
- [10] Coinmarketcap. *Coinmarketcap*. [Accessed: Aug. 19, 2018]. [Online]. Available: <https://coinmarketcap.com/>
- [11] V. Buterin. (2015, August) On public and private blockchains. *Ethereum Blog*. [Accessed: Jul. 18, 2018]. [Online]. Available: <https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/>.
- [12] I. Bentov, A. Gabizon, and A. Mizrahi, “Cryptocurrencies without proof of work,” *CoRR*, vol. abs/1406.5694, 2014, [Accessed: Jul. 19, 2018]. [Online]. Available: <http://arxiv.org/abs/1406.5694>

- [13] V. Buterin. (2016, July) On inflation, transaction fees and cryptocurrency monetary policy. *Ethereum Blog*. [Accessed: Jul. 18, 2018]. [Online]. Available: <https://blog.ethereum.org/2016/07/27/inflation-transaction-fees-cryptocurrency-monetary-policy/>
- [14] (2018, February) Building a decentralized betting platform. *Medium*. [Accessed: Jul. 19, 2018]. [Online]. Available: <https://medium.com/@cryptosportz/building-a-decentralized-betting-platform-development-update-feb-1-2018-clfda8bbda2d>
- [15] *Bitcoin Magazine*. [Accessed: Jul. 19, 2018]. [Online]. Available: <https://bitcoinmagazine.com/authors/vitalik-buterin>
- [16] K. Torpey. (2016, April) Vitalik buterin on his long-term goals for ethereum. *Bitcoin Magazine*. [Accessed: Jul. 22, 2018]. [Online]. Available: <https://bitcoinmagazine.com/articles/vitalik-buterin-on-his-long-term-goals-for-ethereum-1462381147/>
- [17] Ethereum blocktime history. *Etherscan*. [Accessed: Jul. 22, 2018]. [Online]. Available: <https://etherscan.io/chart/blocktime>
- [18] Ethereum 2.0's phase zero scheduled to launch on january 3. *Cointelegraph*. [Accessed: Oct. 8, 2019]. [Online]. Available: <https://cointelegraph.com/news/ethereum-20s-phase-zero-scheduled-to-launch-on-january-3-2020-devs>
- [19] D. G. Greenspan. (2015, July) Multichain private blockchain — white paper. *Coin Sciences Ltd*. [Accessed: Jul. 25, 2018]. [Online]. Available: <https://www.multichain.com/white-paper/>
- [20] D. Larimer. (2018, March) Eos.io technical white paper v2. *block.one*. [Accessed: Aug. 28, 2018]. [Online]. Available: <https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md>
- [21] Eos.io faq. *block.one*. [Accessed: Aug. 28, 2018]. [Online]. Available: <https://eos.io/faq>
- [22] S. Popov. (2018, June) On the tangle, white papers, proofs, airplanes, and local modifiers. *IOTA Blog*. [Accessed: Sept. 6, 2018]. [Online]. Available: <https://blog.iota.org/on-the-tangle-white-papers-proofs-airplanes-and-local-modifiers-44683aff8fea>
- [23] Qubic roadmap. *IOTA Foundation*. [Accessed: Oct. 8, 2019]. [Online]. Available: <https://qubic.iota.org/roadmap>
- [24] Our suite of tools. *Lisk Foundation*. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://lisk.io/products>
- [25] Lisk's consensus algorithm. *Lisk Foundation*. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://lisk.io/productshttps://lisk.io/products>
- [26] Lisk transactions. *Lisk Foundation*. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://lisk.io/documentation/lisk-protocol/transactions>

- [27] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, and J. Timón, “Enabling blockchain innovations with pegged sidechains,” 2014, [Accessed: Sept. 10, 2018]. [Online]. Available: <https://pdfs.semanticscholar.org/1b23/cd2050d5000c05e1da3c9997b308ad5b7903.pdf>
- [28] Ivan on Tech. (2018, April) *What is Lisk? Max Kordek Exclusive Interview*. Youtube. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://www.youtube.com/watch?v=G4zc41v1JWU>
- [29] (2015, December) Linux foundation unites industry leaders to advance blockchain technology. *The Linux Foundation*. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://www.linuxfoundation.org/press-release/2015/12/linux-foundation-unites-industry-leaders-to-advance-blockchain-technology/#.WZ8FmCiG>
- [30] (2017, July) Hyperledger fabric 1.0 is released! *The Linux Foundation Projects*. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://www.hyperledger.org/blog/2017/07/11/hyperledger-fabric-1-0-is-released>
- [31] Hyperledger fabric documentation: Introduction. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://hyperledger-fabric.readthedocs.io/en/release-1.2/whatis.html>
- [32] (2018, January) Hyperledger releases hyperledger sawtooth 1.0. *The Linux Foundation Projects*. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://www.hyperledger.org/announcements/2018/01/30/hyperledger-releases-hyperledger-sawtooth-1-0>
- [33] Hyperledger sawtooth documentation: Introduction. [Accessed: Sept. 17, 2018]. [Online]. Available: <https://sawtooth.hyperledger.org/docs/core/releases/1.0/introduction.html>
- [34] The raft consensus algorithm. [Accessed: Oct. 10, 2019]. [Online]. Available: <https://raft.github.io/>
- [35] Hyperledger fabric sdks. *Hyperledger Fabric Documentation*. [Accessed: July 7, 2019]. [Online]. Available: [https://hyperledger-fabric.readthedocs.io/en/latest/getting\\_started.html](https://hyperledger-fabric.readthedocs.io/en/latest/getting_started.html)
- [36] I. Sommerville, *Software Engineering*. Addison-Wesley, 2011.