



ESTADOS UNIDOS MEXICANOS  
**EDUCACIÓN**

SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO  
NACIONAL DE MÉXICO

Instituto Tecnológico de Orizaba

---

---

---

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

OPCIÓN I.- TESIS

TRABAJO PROFESIONAL

“DESARROLLO DE UNA APLICACIÓN  
DESCENTRALIZADA BASADA  
EN BLOCKCHAIN”

QUE PARA OBTENER EL GRADO DE:  
MAESTRO EN SISTEMAS  
COMPUTACIONALES

PRESENTA:

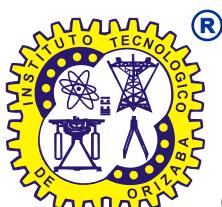
*I.E. José Luis Balderas Rosas*

DIRECTOR DE TESIS:

*Dr. Ulises Juárez Martínez*

CODIRECTOR DE TESIS:

*Dra. Lisbeth Rodríguez Mazahua*



ORIZABA, VERACRUZ, MÉXICO.

MAYO 2021



Instituto Tecnológico de Orizaba  
División de Estudios de Posgrado e Investigación

Orizaba, Ver., 18/mayo/2021  
Dependencia: División de Estudios de  
Posgrado e Investigación  
Asunto: Autorización de Impresión  
OPCIÓN: I

**C. JOSÉ LUIS BALDERAS ROSAS**

Candidato a Grado de Maestro en:  
**SISTEMAS COMPUTACIONALES**  
**P R E S E N T E.-**

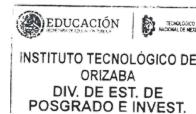
De acuerdo con el Reglamento de Titulación vigente de los Centros de Enseñanza Técnica Superior, dependiente de la Dirección General de Institutos Tecnológicos de la Secretaría de Educación Pública y habiendo cumplido con todas las indicaciones que la Comisión Revisora le hizo respecto a su Trabajo Profesional titulado:

**" Desarrollo de una aplicación descentralizada basada en blockchain"**

Comunico a Usted que este Departamento concede su autorización para que proceda a la impresión del mismo.

**A T E N T A M E N T E**

Excelencia en Educación Tecnológica®  
CIENCIA – TÉCNICA - CULTURA®



**DR. MARIO LEONCIO ARRIJOA RODRÍGUEZ**  
**JEFE DE LA DIVISIÓN DE ESTUDIOS**  
**DE POSGRADO E INVESTIGACIÓN**

Avenida Oriente 9 No. 852  
Col. Emiliano Zapata, C.P. 94320  
Orizaba, Veracruz, México.  
Teléfono: 272-110-53-60  
Email: [cyd.orizaba@tecnm.mx](mailto:cyd.orizaba@tecnm.mx)  
[www.orizaba.tecnm.mx](http://www.orizaba.tecnm.mx)





Instituto Tecnológico de Orizaba  
División de Estudios de Posgrado e Investigación

Orizaba, Ver., 15/marzo/2021

Asunto: Revisión del Trabajo Escrito

**C. MARIO LEONCIO ARRIOJA RODRIGUEZ**  
**JEFE DE LA DIVISION DE ESTUDIOS**  
**DE POSGRADO E INVESTIGACION**  
**P R E S E N T E.**

Los que suscriben miembros del jurado, han realizado la revisión de la Tesis del (la) C.:

**JOSÉ LUIS BALDERAS ROSAS**  
**No. DE CONTROL: M13360684**

la cual lleva el título de:

**“Desarrollo de una aplicación descentralizada basada en blockchain”**

y concluyen que se acepta.

**ATENTAMENTE**

PRESIDENTE: DR. ULISES JÚAREZ MARTÍNEZ

SECRETARIO: DRA. LISBETH RODRÍGUEZ MAZAHUA

VOCAL: M.C.E. BEATRIZ ALEJANDRA OLIVARES ZEPAHUA

VOCAL SUP.: DR. ADOLFO CENTENO TÉLLEZ

EGRESADO (A) DE LA **MAESTRIA EN SISTEMAS COMPUTACIONALES**

OPCIÓN: I Thesis



Avenida Oriente 9 No. 852  
Col. Emiliano Zapata, C.P. 94320  
Orizaba, Veracruz, México.  
Teléfono: 272-110-53-60  
Email: depi\_orizaba@tecnm.mx  
www.orizaba.tecnm.mx



# Contenido

<b>Abreviaturas</b>	<b>x</b>
<b>Resumen</b>	<b>xii</b>
<b>Abstract</b>	<b>xiii</b>
<b>Introducción</b>	<b>xiv</b>
<b>1. Antecedentes</b>	<b>1</b>
1.1. Marco Teórico . . . . .	1
1.1.1. <i>Blockchain</i> . . . . .	1
1.1.2. Tecnologías utilizadas en <i>blockchain</i> . . . . .	2
1.1.3. Nonce . . . . .	7
1.1.4. Marca de tiempo . . . . .	7
1.1.5. Funcionamiento de <i>blockchain</i> . . . . .	8
1.1.6. Elementos <i>blockchain</i> . . . . .	8
1.1.7. Estructura de <i>blockchain</i> . . . . .	9
1.1.8. Categorización de la tecnología <i>blockchain</i> . . . . .	10
1.1.9. Criptomonedas . . . . .	11
1.1.10. <i>Bitcoin</i> . . . . .	11
1.1.11. <i>Ethereum</i> . . . . .	11
1.1.12. <i>Hyperledger</i> . . . . .	12
1.1.13. <i>Hyperledger Fabric</i> . . . . .	12
1.1.14. Transacciones . . . . .	12
1.1.15. Protocolo . . . . .	13
1.1.16. Nodo . . . . .	13
1.1.17. <i>Smart Contract</i> . . . . .	13
1.1.18. Aplicación descentralizada (DApp) . . . . .	13
1.1.19. <i>Ledger</i> distribuido . . . . .	13
1.2. Planteamiento del problema . . . . .	14
1.3. Objetivos . . . . .	14
1.3.1. Objetivo general . . . . .	14
1.3.2. Objetivos específicos . . . . .	15
1.4. Justificación . . . . .	15

<b>2. Estado de la práctica</b>	<b>17</b>
2.1. Trabajos relacionados . . . . .	17
2.2. Análisis del estado de la práctica . . . . .	29
2.3. Solución propuesta . . . . .	38
2.3.1. Justificación de la solución seleccionada . . . . .	38
<b>3. Aplicación de la metodología</b>	<b>40</b>
3.1. Descripción de la solución . . . . .	40
3.2. Análisis de artefactos . . . . .	41
3.3. Metodología . . . . .	47
3.4. Aplicación de la metodología . . . . .	49
3.4.1. Aplicación de las fases de la metodología para el desarrollo de la herramienta . . . . .	49
3.4.2. Pruebas con la plataforma . . . . .	93
3.4.3. Codificación y pruebas de contratos inteligentes ( <i>chaincodes</i> ) . .	95
3.4.4. Codificación y pruebas con la aplicación externa . . . . .	113
<b>4. Resultados</b>	<b>116</b>
4.1. Acceso al sistema . . . . .	116
4.2. Roles en el sistema . . . . .	117
4.3. Comprobación de la descentralización en la aplicación . . . . .	126
<b>5. Conclusiones y recomendaciones</b>	<b>130</b>
5.1. Conclusiones . . . . .	130
5.2. Trabajos a futuro . . . . .	131
<b>A. Pruebas con la plataforma</b>	<b>132</b>

# Índice de figuras

1.1. Estructura de un árbol Merkle [7] . . . . .	5
1.2. Arquitectura de un árbol Merkle en <i>blockchain</i> [7] . . . . .	6
1.3. Estructura de <i>blockchain</i> [7] . . . . .	10
3.1. Ejemplo del uso de ícono “ <i>chain</i> ” en el diagrama clases. . . . .	42
3.2. Ejemplo de modelo ER, entre Base de datos relacional y <i>blockchain</i> . . . . .	44
3.3. Ejemplo de notación de carril para <i>blockchain</i> . . . . .	45
3.4. Proceso de trabajo para el desarrollo de la aplicación descentralizada. . . . .	48
3.5. Diagramas de caso de uso . . . . .	51
3.6. Diagrama de secuencia (CDU-01) . . . . .	71
3.7. Diagrama de secuencia (CDU-03) . . . . .	72
3.7. Diagrama de secuencia (CDU-03) . . . . .	73
3.8. Diagrama de secuencia (CDU-04) . . . . .	74
3.8. Diagrama de secuencia (CDU-04) . . . . .	75
3.9. Diagrama de secuencia (CDU-05) . . . . .	76
3.9. Diagrama de secuencia (CDU-05) . . . . .	77
3.10. Diagrama de secuencia (CDU-06) . . . . .	78
3.10. Diagrama de secuencia (CDU-06) . . . . .	79
3.11. Diagrama de secuencia (CDU-07) . . . . .	80
3.12. Diagrama de secuencia (CDU-08) . . . . .	81
3.13. Diagrama de secuencia (CDU-09) . . . . .	82
3.14. Diagrama de secuencia (CDU-10) . . . . .	83
3.15. Diagrama de secuencia (CDU-02) . . . . .	83
3.16. Arquitectura del sistema. . . . .	84
3.17. Diagrama de componentes. . . . .	88
3.18. Interfaz de registro. . . . .	90
3.19. Interfaz iniciar sesión. . . . .	90
3.20. Interfaz de autobuses. . . . .	91
3.21. Interfaz agregar autobús. . . . .	91
3.22. Interfaz agregar personal. . . . .	92
3.23. Interfaz de rutas. . . . .	93
3.24. Interfaz modificar ruta. . . . .	93
3.25. Generación de artefactos de red. . . . .	95
3.26. Activación de la red. . . . .	96
3.27. Generación de red finalizada . . . . .	97

3.28. Diseño de la red <i>blockchain</i> de trabajo . . . . .	98
3.29. Estructura de la red <i>blockchain</i> de trabajo. . . . .	98
3.30. Creación de la red <i>blockchain</i> . . . . .	100
3.31. Verificación de la creación de la red . . . . .	100
3.32. Creación del canal autobuses. . . . .	101
3.33. Canal de autobuses creado. . . . .	101
3.34. Comando para unir la <code>Org1</code> al canal autobuses. . . . .	102
3.35. Canal unido. . . . .	102
3.36. Estado mundial, <i>World state</i> . . . . .	103
3.37. <i>World state</i> replicado en todos los nodos. . . . .	103
3.38. Variables de entorno para el ciclo de vida <i>chaincode</i> . . . . .	104
3.39. Empaquetado del <i>chaincode</i> . . . . .	104
3.40. Instalación del <i>chaincode</i> . . . . .	105
3.41. Instalación del <i>chaincode</i> finalizada. . . . .	105
3.42. Políticas de endorsamiento aprobadas en la <code>org1</code> y <code>org2</code> . . . . .	105
3.43. <i>commit</i> del <i>chaincode</i> . . . . .	106
3.44. Confirmación del <i>commit</i> . . . . .	106
3.45. Intefaz <i>invoke</i> . . . . .	107
3.46. Intefaz <i>query</i> . . . . .	107
3.47. Lectura del elemento <code>id:1</code> en el <i>world state</i> . . . . .	108
3.48. <i>Chaincode buscontrol</i> . . . . .	109
3.49. Continuación <i>chaincode buscontrol</i> . . . . .	109
3.50. Método <i>get</i> para consulta autobuses. . . . .	111
3.51. Método <i>post</i> para crear un autobús. . . . .	111
3.52. Método <i>post</i> para modificar un autobús. . . . .	112
3.53. Método <i>post</i> para eliminar un autobús. . . . .	112
3.54. Método <i>get</i> para consultar todos los autobuses. . . . .	112
3.55. Petición <i>post</i> realizada desde <i>Postman</i> . . . . .	113
3.56. Petición <i>get</i> realizada desde <i>Postman</i> . . . . .	113
3.57. Componente transportes (Angular). . . . .	114
3.58. Servicios(Angular). . . . .	114
3.59. <code>global.ts</code> . . . . .	115
3.60. Servicio para gestionar autobuses. . . . .	115
4.1. Inicio o home. . . . .	117
4.2. Acceso al sistema. . . . .	117
4.3. Listado de autobuses. . . . .	118
4.4. Agregar autobús. . . . .	119
4.5. Modificar autobús. . . . .	119
4.6. Calendarización autobús. . . . .	120
4.7. Formulario para la calendarización del autobús. . . . .	120
4.8. Listado de conductores. . . . .	121
4.9. Agregar conductor. . . . .	121
4.10. Modificar conductor. . . . .	122
4.11. Lista de rutas. . . . .	123

4.12. Modificar ruta y agregar parada. . . . .	123
4.13. Registrar usuario. . . . .	124
4.14. Credenciales del usuario. . . . .	124
4.15. Gestión de cobros. . . . .	125
4.16. Tarifas. . . . .	125
4.17. Cobros. . . . .	126
4.18. Bases de datos. . . . .	127
4.19. couchDB organización 1. . . . .	127
4.20. couchDB organización 2. . . . .	128
4.21. couchDB organización 3. . . . .	128
4.22. base de datos <code>autobuses_payscontrol</code> nodo 2. . . . .	128
4.23. base de datos <code>autobuses_payscontrol</code> nodo 1. . . . .	129
4.24. Consulta de pagos mediante la API . . . . .	129
A.1. Crear instancia. . . . .	132
A.2. Instancia creada. . . . .	133
A.3. Inicio de máquina virtual CentOS 7. . . . .	133
A.4. Actualización de CentOS 7. . . . .	134
A.5. Paquetes del sistema. . . . .	135
A.6. Instalación de paquetes. . . . .	136
A.7. Instalación de Docker. . . . .	137
A.8. Comando para agregar usuario al grupo. . . . .	137
A.9. Prueba de instalación docker-compose. . . . .	138
A.10. Instalación de WGET. . . . .	139
A.11. Código <code>?hello world?</code> en Go. . . . .	141
A.12. Ejecución de <code>?hello?</code> en Go. . . . .	141
A.13. Verificación de versión Git. . . . .	141

# Índice de tablas

2.1. Cuadro comparativo de trabajos relacionados. . . . .	30
3.1. Estereotipos para representar contratos inteligentes en diagrama de clases UML. . . . .	42
3.2. Estereotipos para el diagrama de secuencia UML. . . . .	43
3.3. Patrones de diseño orientados a software <i>blockchain</i> . . . . .	45
3.4. Especificación del Caso de Uso Iniciar Sesión. . . . .	51
3.5. Especificación del Caso de Uso Cerrar Sesión. . . . .	52
3.6. Especificación del Caso de Uso Gestionar Conductores. . . . .	53
3.7. Especificación del Caso de Uso Gestionar Autobuses. . . . .	57
3.8. Especificación del caso de Uso Gestionar Rutas. . . . .	60
3.9. Especificación del Caso de Uso Gestionar Paradas. . . . .	63
3.10. Especificación del Caso de Uso Monitorear Autobús. . . . .	66
3.11. Especificación del Caso de Uso Actualizar Ruta. . . . .	67
3.12. Especificación del Caso de Uso Calendarizar Rutas. . . . .	68
3.13. Especificación del Caso de Uso Visualizar Calendarización. . . . .	69

# Abreviaturas

- API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones).
- BBSE (*Blockchain Based Software Engineering*, Ingeniería de Software Basada en *Blockchain*).
- BCS (*Blockchain Services*, Servicios *Blockchain*).
- BOS (*Blockchain Oriented Software*, Software Orientado a *Blockchain*).
- BOSE (*Blockchain Oriented Software Engineering*, Ingeniería de Software Orientada a *Blockchain*).
- BPMN (*Business Process Model and Notation*, Modelo y Notación de Procesos de Negocio).
- CI (*Continuous Integration*, Integración Continua).
- CSS (*Cascading Style Sheets*, Hoja de Estilos en Cascada)
- DApp (*Decentralized Application*, Aplicación Descentralizada).
- DCS (*Distributed Control System*, Sistema de Control Distribuido).
- DLT (*Distributed Ledger Technology*, Tecnología de Registro Distribuido).
- DOM (*Document Object Model*, Modelo de Objetos del Documento).
- DSL (*Domain Specific Languages*, Lenguaje de Dominio Específico).
- E-R (*Entity Relation*, Entidad Relación).
- EVM (*Ethereum Virtual Machine*, Máquina Virtual de Ethereum).
- HTTP (*Hypertext Transfer Protocol*, Protocolo de Transferencia de Hipertexto).
- HTML (*HyperText Markup Language*, Lenguaje de Marcas de Hipertexto)
- ICO (*Initial Coin Offering*, Oferta Inicial de Monedas).
- IDE (*Integrated Development Environment*, Entorno de Desarrollo Integrado)
- IoT (*Internet of Things*, Internet de las Cosas).

- JVM (*Java Virtual Machine*, Máquina Virtual de Java).
- NFC (*Near Field Communication*, Comunicación de Campo Cercano).
- OSS (*Operational Support System*, Sistema de Soporte Operacional).
- POW (*Proof Of Work*, Prueba de Trabajo).
- POS (*Proof Of Stake*, Prueba de Participación).
- Red P2P (*Red Peer to Peer*, Red Entre Pares).
- SC (*Smart Contract*, Contrato Inteligente).
- SDK (*Software Development Kit*, Kit de desarrollo de software).
- SE (*Software Engineering*, Ingeniería de Software).
- SWEBOK (*Software Engineering Body of Knowledge*, Conjunto de Conocimientos de Ingeniería de Software).
- UML (*Unified Modeling Language*, Lenguaje de Modelo Unificado).
- XP (*eXtreme Programming*, Programación Extrema).

# Resumen

La tecnología *blockchain* ha ganado un impulso masivo en los últimos años. Actualmente esta tecnología aún no está en una completa madurez, sin embargo, ha mostrado robustez en el desarrollo de aplicaciones descentralizadas. El desarrollo de software orientado a *blockchain* ha crecido de manera exponencial, debido a esto no se han integrado artefactos de modelación para este nuevo paradigma de software, siendo una parte fundamental el modelado de software en general para el desarrollo de cualquier tipo de software. No realizar un modelado o no modelar de manera correcta, conlleva a problemas futuros durante la planificación o el desarrollo del software, por ello yace la necesidad de realizar un modelado adecuado de software orientado a *blockchain*.

Teniendo en cuenta la problemática, el objeto de este proyecto es estudiar la tecnología *blockchain* y llevar a cabo una investigación de metodologías y artefactos reportados para software orientado a *blockchain*. Asimismo se realizó un estudio profundo de estas metodologías y artefactos, identificando los que mejor se adaptan a este nuevo paradigma de software mediante la identificación de fortalezas y debilidades de estas técnicas. Con lo anterior se representa de forma adecuada la fragmentación de la lógica de negocio entre contratos inteligentes y componentes convencionales, y se realizó el desarrollo de una aplicación descentralizada eliminando problemas y conflictos durante el desarrollo, mediante un modelado acorde.

# **Abstract**

blockchain technology has gained massive momentum in recent years. Today, this technology is not yet fully mature, but it has shown robustness in the development of decentralized applications. The development of blockchain oriented software has grown exponentially because of this no modeling artifacts have been integrated for this new software paradigm. Being a key part of the software modeling in general for the development of any software. Not modeling or not modeling correctly leads to future problems during software development or planning, hence the need for proper blockchain-oriented software modeling.

In view of the problem, the aim of this project is to study blockchain technology and conduct research on reported methodologies and artifacts for blockchain oriented software. We also conducted a thorough study of these methodologies and artifacts, identifying those that best adapt to this new software paradigm by identifying the strengths and weaknesses of these techniques. This adequately represents the fragmentation of business logic between intelligent contracts and conventional components. The development of the decentralized application was performed by eliminating problems and conflicts during the development, using a suitable modeling.

# Introducción

*Blockchain* se describió en el año 1991 bajo el nombre *Digital Timestamps*, sin embargo, pasó desapercibida hasta 2008 cuando se lanzó un *white paper* denominado *Bitcoin: A Peer-to-Peer Electronic Cash System* y posteriormente en 2009 apareció *bitcoin* la primera criptomoneda creada bajo el pseudónimo de Satoshi Nakamoto, llevando a una revolución no solo en el ámbito económico, sino también en el tecnológico con su protocolo *blockchain* donde se ha podido comprobar su seguridad y robustez. La tecnología *blockchain* es una de las tecnologías con mayor potencial disruptivo en los últimos años, permite implementar una base de datos distribuida, pública e immutable basada en una secuencia creciente de bloques. Los nodos en la cadena de bloques cuentan con múltiples transacciones y réplicas de los datos, por lo que se ve como un registro de transacciones ordenado, eficaz y seguro. Con la aparición de Ethereum y el uso de contratos inteligentes se dieron cuenta que *blockchain* se podía usar como una computadora descentralizada, ejecutando contratos inteligentes, programas que se usan como base para la ejecución contractual automatizada.

La revolución tecnológica de *blockchain* en los últimos años ha provocado su expansión a ritmos sin precedentes y el desarrollo ha crecido de manera exponencial, sin embargo, el desarrollo rápido de diversas aplicaciones orientadas a *blockchain*, conlleva a un desarrollo de software apresurado y sin control. Un escenario que no garantiza la modelación de software, ni la calidad del software. El primer paso para desarrollar un sistema de software utilizando prácticas sólidas de ingeniería de software es tener un proceso de desarrollo claro y diseñar prácticas y notaciones útiles para el propósito [1]. Con base en esto, se pueden utilizar prácticas específicas de desarrollo, prueba,

implementación y evaluación de seguridad.

Para ofrecer una mejor visión de esta tesis, el presente documento está constituido de la siguiente forma:

Capítulo uno, tiene como objetivo proveer los conceptos básicos del dominio del problema (marco teórico), de igual forma el planteamiento del problema, objetivo general, objetivos específicos y la justificación del proyecto; en el capítulo dos se aborda una revisión de investigaciones y publicaciones que se han llevado a cabo enfocadas en la implementación de la tecnología *blockchain* (estado de la práctica); en el capítulo tres, se describe la manera en que se desarrolló la DApp; en el capítulo cuatro se presentan los resultados obtenidos con la DApp en un caso de estudio. Por último se presentan las conclusiones.

# Capítulo 1

## Antecedentes

En el presente capítulo se explican los conceptos más importantes para el desarrollo del presente proyecto, se da a conocer la problemática y su posible solución, asimismo el objetivo general y los objetivos específicos para comprender cómo se afronta dicho proyecto, y una justificación la cual apoya y respalda la realización del proyecto.

### 1.1. Marco Teórico

A continuación se muestran los conceptos más importantes relacionados con el presente proyecto.

#### 1.1.1. *Blockchain*

Existen numerosas definiciones de *blockchain* de diferentes autores, y no se reporta una definición única internacionalmente acordada [2]. Por lo cual se citan varias definiciones.

- *Blockchain* es un tipo de libro mayor distribuido que se crea como una cadena de criptografía, bloques vinculados con transacciones y envía todos los datos a todos los nodos en su red. Tama et al. [3] son de opinión similar, describiendo *blockchain* como ”parte de la capa de implementación de un sistema de software distribuido”, cuyo propósito es garantizar la integridad de los datos [2].

- *Blockchain* es una estructura de datos caracterizada por los siguientes elementos clave: redundancia de datos, verificación de los requisitos de transacción antes de la validación, registro de transacciones en bloques ordenados secuencialmente, transacciones basadas en criptografía de clave pública y un lenguaje de *script* de transacción [4].
- *Blockchain* es una red descentralizada y de igual a igual aplicable para eliminar la interferencia o participación de terceros en las actividades comerciales y de intercambio de datos. Tiene una base de datos distribuida que se replica entre todos los pares de la red. Crea un entorno digno de confianza y libre de intromisiones entre pares mutuamente no confiables. Sirve como un libro de transacciones comerciales, un sistema para compartir datos, activos y registros valiosos, mediante el uso de prueba de trabajo u otro mecanismo de consenso para crear un entorno confiable, transparente y responsable en lugar de depender de un tercero. Dado que replican los datos entre todos los pares, permite a los usuarios verificar y auditar las transacciones [5].

### 1.1.2. Tecnologías utilizadas en *blockchain*

*Blockchain* es un concepto complejo, que consiste en una combinación de diferentes tecnologías, cada una de las cuales es un pilar fundamental para hacer de *blockchain* una tecnología importante.

#### Red P2P

Una red P2P es una red descentralizada e interconectada que comparte tareas o cargas de trabajo (como potencia de procesamiento o almacenamiento de datos) entre todos los participantes por igual. Los participantes crean los datos, los almacenan o transfieren y están disponibles para todos en la red [6].

## Criptografía

Es la ciencia de codificar y decodificar mensajes para mantenerlos seguros, esto significa que la información solo es vista por los destinatarios previstos y nadie más. El método implica tomar datos no encriptados como un fragmento de texto, y encriptarlos utilizando un algoritmo matemático, conocido como cifrado [6].

En *blockchain*, la criptografía se usa principalmente para dos propósitos:

1. Asegurar la identidad del remitente de las transacciones.
2. Asegurar que los registros pasados no puedan manipularse.

## Cifrado

Es un algoritmo matemático utilizado para cifrar o descifrar datos de forma segura o hacer ambas cosas [6].

## Criptografía de clave pública

La criptografía de clave pública, también conocida como criptografía asimétrica, representa una mejora en la criptografía de clave simétrica estándar, ya que permite que la información se transfiera a través de una clave pública que se puede compartir con cualquier persona.

En lugar de usar una sola clave para el cifrado y descifrado, como es el caso de la criptografía de clave simétrica, se utilizan claves separadas (una clave pública y una clave privada) [6].

## Clave pública

Utilizada como una dirección para transferir activos e información en la cadena de bloques, la clave pública descifra los mensajes destinados a un destinatario en particular [6].

## Clave privada

Es la información que aparece en una cadena de números y letras al azar, su valor no es muy diferente de una contraseña y, como tal, debe mantenerse oculto para cualquier persona que no sea el propietario y siempre debe tener una copia de seguridad, ya que no se puede recuperar una vez perdido [6].

## Firma digital

Las firmas digitales son uno de los principales aspectos para garantizar la seguridad e integridad de los datos que se registran en una cadena de bloques. Son una parte estándar de los protocolos de *blockchain*, utilizados principalmente para asegurar transacciones y bloques de transacciones, transferencia de información, gestión de contratos y cualquier otro caso en el que sea importante detectar y prevenir cualquier manipulación externa. Las firmas digitales utilizan criptografía asimétrica, lo que significa que la información se puede compartir con cualquier persona, mediante el uso de una clave pública [6].

## ***Hashing***

Es el proceso de convertir una entrada de cualquier longitud en una cadena de texto de tamaño fijo usando una función matemática, esto significa que un texto sin importar que tan largo sea, se puede convertir en una secuencia de números y letras por medio de un algoritmo, el mensaje de entrada se llama *input*, el algoritmo que realiza el proceso se llama función *hash* y la salida se llama el valor del *hash* [6].

## ***Hashing blockchain***

En *blockchain*, los *hash* se usan para representar el estado actual de *blockchain*. Como tal, la entrada representa todo lo que ha sucedido en *blockchain*, por lo que cada transacción hasta ese punto se combina con los nuevos datos que se están agregando. Lo que esto significa es que la salida se basa y está determinada por todas las transacciones anteriores que han ocurrido en *blockchain* [6].

## Estructuras de datos

Las estructuras de datos son una forma especializada de almacenar datos. Los dos objetos *hashing* más importantes que llevan a cabo esta función son punteros y listas enlazadas. Los punteros almacenan direcciones como variables y, como tal, apuntan a las ubicaciones de otras variables. Las listas enlazadas son una secuencia de bloques conectados entre sí a través de punteros [6].

## Árboles Merkle

Un árbol merkle, también conocido como árbol de *hash*, es una estructura de datos de *hashes* utilizada para registrar datos en una cadena de bloques de manera segura y eficiente. Es especialmente útil, ya que permite a cualquiera confirmar la validez de una transacción individual sin tener que descargar una cadena de bloques completa. Los árboles Merkle son un componente clave para permitir que la tecnología *blockchain* funcione al tiempo que proporcionan seguridad, integridad e irrefutabilidad y, junto con los protocolos de consenso, son dos de los aspectos más importantes de la tecnología *blockchain* [6]. En la figura 1.1 se muestra la estructura de un árbol merkle.

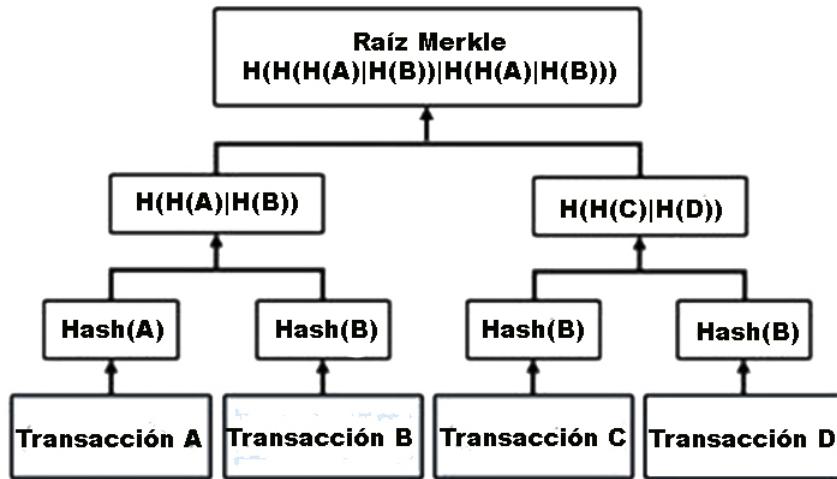


Figura 1.1: Estructura de un árbol Merkle [7].

Como ya se mencionó anteriormente los árboles merkle son un componente clave y fundamental para la tecnología *blockchain*. Un bloque tiene la raíz de merkle, el cual

proviene de una transacción de *hash*, empareja dos transacciones con *hash* y genera el nodo de árbol de nivel superior. Posteriormente, se obtendrá un *hash* para almacenar, el cual es determinista en función de los *hash* de todas las transacciones subyacentes. Este *hash* es la raíz merkle y cada bloque en *blockchain* tiene una raíz merkle almacenada en la parte superior o encabezado del bloque. A continuación se muestra en la figura 1.2 la arquitectura de un árbol merkle en *blockchain*.

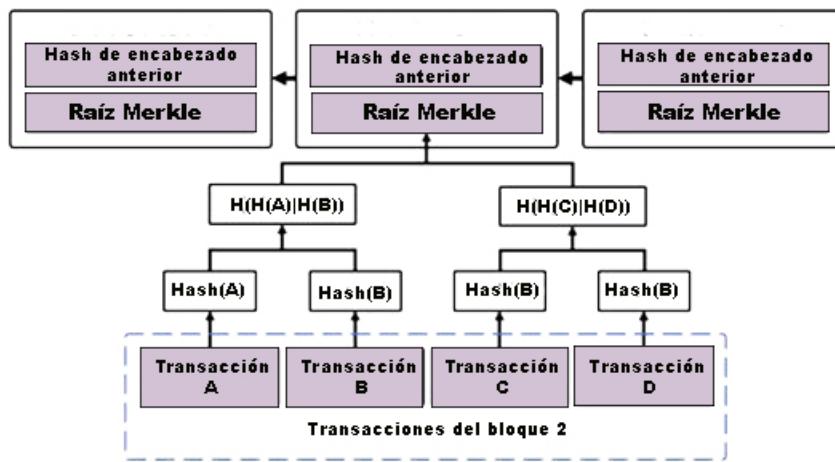


Figura 1.2: Arquitectura de un árbol Merkle en *blockchain* [7].

## Algoritmo de consenso

Los algoritmos de consenso son protocolos que crean un sistema irrefutable de acuerdo entre varios dispositivos a través de una red distribuida, al tiempo que evitan la explotación del sistema. Los protocolos de consenso de *blockchain* son los que mantienen todos los nodos de una red sincronizados entre sí [6].

### ***Proof of work (PoW)***

PoW es el mecanismo de consenso más utilizado para la tecnología *blockchain*, resuelve acertijos<sup>1</sup> complejos del nuevo bloque antes de aprobar el bloque en el libro mayor. Una vez resuelto, la solución se reenvía a los mineros<sup>2</sup> restantes en la red, los mineros

<sup>1</sup>Acertijos: problemas que requieren de cálculo matemático para resolver los problemas.

<sup>2</sup>Mineros: nodos que recopilan transacciones y trabajan para organizar las transacciones en bloques.

verifican la solución y se acepta en sus respectivas copias del libro mayor. Después de la verificación, las transacciones aprobadas finalizan. Si algún usuario intenta hacer una transacción duplicada, será visible en la red y esa transacción nunca será aceptada [1].

### ***Proof of Stake (POS)***

PoS es un enfoque alternativo para PoW que requiere algunos cálculos de CPU para la minería<sup>3</sup>. En comparación con PoW, el proceso de PoS es bastante diferente. El protocolo de consenso de prueba de participación es un sistema robusto que cumple de manera efectiva y eficiente su propósito previsto, una ventaja es que requiere menor poder computacional [1].

#### **1.1.3. Nonce**

*Nonce* es un elemento que funciona en conjunto con el *hash*, su función principal es evitar la manipulación de información de los bloques. Es un número arbitrario que se emplea en criptografía dentro de los protocolos de autenticación, este número garantiza que los *hash* utilizados no se pueden volver a utilizar y es destinado a la autenticación de transferencia de datos entre dos o más partes. Normalmente el *nonce* implementa un *timestamp* o marca de tiempo.

#### **1.1.4. Marca de tiempo**

Es un tipo de dato almacenado en los bloques, es único y su función principal es determinar el momento, es decir la hora y la fecha en que el bloque se minó y se validó por la red *blockchain*. Implementar la marca de tiempo hace que el bloque no se repita en un futuro, ya que este contiene la hora y la fecha de creación del bloque, por lo cual, no existe la posibilidad de que se repita el mismo *hash* que se asignó previamente.

---

<sup>3</sup>Minería: La minería de criptomonedas permite que la red funcione como una red descentralizada de igual a igual sin la necesidad de una autoridad central de terceros.

### 1.1.5. Funcionamiento de *blockchain*

*Blockahin* debe su nombre a cómo funciona y a la forma en que almacena sus datos, la información se empaqueta en bloques y se vinculan para formar una cadena con otros bloques de información similar. Cada bloque contiene tres elementos: información del bloque, *hash* del bloque y *hash* del bloque anterior, a excepción del primer bloque llamado génesis, este bloque no tiene *hash* del bloque anterior. El *hash* del bloque anterior vincula los bloques entre sí y evita que cualquier bloque se altere o que se inserte un bloque entre dos bloques existentes. De esta manera, el método hace que la cadena de bloques sea a prueba de manipulaciones, presentando el atributo clave de inmutabilidad. Las transacciones se almacenan en una red descentralizada, por lo cual, las transacciones nuevas son emitidas a todos los nodos y cuando un nodo encuentra la prueba de trabajo emite el bloque a todos los nodos, los nodos validan el bloque y aceptan el bloque si todas las transacciones en el bloque son válidas [8].

### 1.1.6. Elementos *blockchain*

A continuación, se presenta una definición de los elementos clave que conforman la tecnología *blockchain* y permiten el funcionamiento de la misma y así ofrecer los beneficios de esta tecnología, los elementos son los siguientes: bloque, cadena y red.

- **Bloque.** *Blockchain* consta de bloques, y cada bloque contiene encabezado y cuerpo. El cuerpo acomoda los datos, en el caso de *Bitcoin*, la colección incluye los datos transaccionales y el encabezado consta de metadatos (*hash* del bloque, *hash* del bloque anterior, prueba de trabajo y una marca de tiempo). Los bloques están limitados por el tiempo en una recopilación de datos, lo que significa que después de un tiempo específico, estos bloques se convertirán en una parte permanente de la cadena de bloques y sus datos no serán retráctiles ni modificables.
- **Cadena.** *Blockchain* consiste en una cadena que conecta bloques, mantiene todos los bloques y crea confianza entre ellos. El *hash* del bloque anterior en cada bloque lo conecta con dicho bloque, que se remonta al primer bloque, el bloque génesis

(primer bloque de *blockchain*). Si se logra modificar el bloque, esto también cambiará el *hash* de ese bloque, lo que significa que todos los bloques posteriores se vuelven inválidos porque ya no almacenan el *hash* válido del bloque anterior y, por lo tanto, la cadena se rompe.

- **Red.** La red es la última y una parte importante de cualquier *blockchain*. Es una colección o un grupo de computadoras comúnmente conocido como un nodo. Cuando ocurre cualquier transacción, o ingresa al sistema la transacción, todos los nodos la verifican primero y una vez que se llega a un consenso entre la mayoría de los nodos, esa transacción se convierte en la parte permanente de la cadena de bloques. Los nodos obtienen la mano de obra para ejecutar el protocolo con la parte de la criptomoneda porque consume su energía eléctrica. Para hacer que la cadena de bloques sea más segura, la red debe ser más grande y más distribuida [9].

### 1.1.7. Estructura de *blockchain*

La estructura de datos de *blockchain* se define como un registro ordenado de enlaces retrospectivos de bloques de transacciones. El bloque se compone de dos partes: los datos principales y el encabezado. Los datos principales contienen una lista de transacciones, mientras que el encabezado incluye un *hash* del bloque anterior y actual, raíz *merkle*, marca de tiempo, *nonce* y otra información. La figura 1.3 muestra la estructura de *blockchain*.

1. Descentralizado: la estructura básica de *blockchain*, la red está descentralizada, lo que significa que no necesita depender de ningún servidor o nodo. Los datos se registran, almacenan y actualizan por un grupo de nodos.
2. Transparencia: cuando los datos se transmiten en la cadena de bloques, los registros en cada nodo son abiertos y transparentes; esta es la razón por la que se confía en *blockchain*.

3. Código abierto: los registros de los sistemas *blockchain* son públicamente verificables para cualquier usuario, y el usuario también puede usar el sistema *blockchain* para desarrollar cualquier aplicación.
4. Autonomía: según el mecanismo de consenso, cada nodo en la cadena de bloques transmite o actualiza los datos entre sí en una situación segura. Esta idea es de una sola entidad a todo el sistema para que nadie interfiera con ella.
5. Inmutable: cualquier registro siempre se mantendrá, almacenará y no se alterará a menos que los nodos restantes tengan un registro en el que se cambie más del 51 % del registro.
6. Anonimato: la tecnología *blockchain* resuelve el problema de la confianza en el nodo a nodo, por lo que la transmisión de datos o la transacción pueden ocultarse, y solo cuando se conoce la dirección *blockchain* del comerciante quedará expuesta.

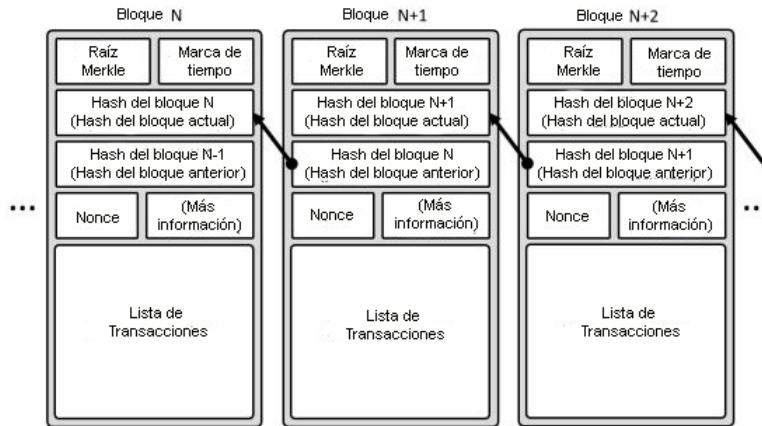


Figura 1.3: Estructura de *blockchain* [7].

### 1.1.8. Categorización de la tecnología *blockchain*

#### *Blockchain* pública

La red pública de *blockchain* es una red abierta y cualquiera puede participar sin obtener permiso. Los participantes ejecutan el protocolo de consenso y mantienen el libro público compartido. Es más seguro y mantiene una baja privacidad [1].

### ***Blockchain* privada**

Se requiere una invitación para participar en una *blockchain* privada. Una *blockchain* privada pone restricciones a los participantes, definitivamente aumenta la privacidad y se requiere menos esfuerzo para lograr el consenso. La *blockchain* privada es menos segura<sup>4</sup> que la *blockchain* pública [1].

#### **1.1.9. Criptomonedas**

Las criptomonedas son monedas digitales, introducidas por primera vez con la creación de *Bitcoin*, desarrollada bajo el pseudónimo de Satoshi Nakamoto. Una criptomoneda es un sistema de dinero electrónico entre pares puramente descentralizado, y es la primera tecnología que supera con éxito el requisito de que una parte centralizada valide las transacciones. La arquitectura de criptomonedas proporciona varias características combinadas que incluyen validación criptográfica para todas las transacciones, dinero descentralizado y transacciones, todo almacenado en libros públicos dentro de un marco quasi anónimo [10].

#### **1.1.10. *Bitcoin***

*Bitcoin*, la primera criptomoneda del mundo, permite realizar transacciones mediante una red *peer to peer*, la cual elimina a un intermediario central, irrumpiendo en el sector bancario [5] debido a su medio de pago virtual basado en la tecnología *blockchain*.

#### **1.1.11. *Ethereum***

*Ethereum* es una plataforma global de código abierto para aplicaciones descentralizadas, donde se escribe código que controle el valor digital del ether (criptomoneda de *Ethereum*), se ejecuta y es accesible desde cualquier parte del mundo [11].

---

<sup>4</sup>Seguridad *blockchain* privada: con menos nodos, es mucho más fácil para un ‘mal actor’ ganar el control de la red y poner en peligro toda la red. Una red privada es mucho más vulnerable a riesgos de *hacks* y la manipulación de datos.

### 1.1.12. *Hyperledger*

*Hyperledger* es un proyecto de código abierto; se clasifica como una *blokchain* privada permisionada, su infraestructura alberga herramientas, marcos de trabajo, bibliotecas e interfaces para mejorar la fiabilidad y rendimiento de la tecnología *blockchain*, una característica a destacar es que no maneja una *criptomoneda*.

### 1.1.13. *Hyperledger Fabric*

Cadena de bloques permisionada y distribuida para el desarrollo de aplicaciones y soluciones en el ámbito empresarial.

### 1.1.14. Transacciones

Las transacciones en [12] se definen como un mensaje entre los participantes y consta de tres segmentos:

#### Firma

La firma digital del participante de origen, cifrada con su clave privada del emisor para que otros nodos de la red puedan verificar que el mensaje realmente vino del participante de origen.

#### Entradas

Es una referencia a una salida de otra transacción existente. Cada transacción tiene una o múltiples entradas, sumándose los valores de cada salida. El valor total de las entradas es la suma máxima disponible para las salidas.

#### Salidas

Es una lista de cómo deben distribuirse los fondos en las entradas. Todos los fondos en las entradas deben redistribuirse en las salidas, por lo que el emisor pagará al destinatario la cantidad requerida y devolverá el resto como cambio.

### 1.1.15. Protocolo

Conjunto de reglas que describen cómo debería funcionar la comunicación y la transmisión de datos entre dispositivos electrónicos. Estas reglas deben definirse antes de enviar cualquier dato, detallando cómo se estructura la información y cómo cada dispositivo la enviará o recibirá [6].

### 1.1.16. Nodo

Nodo es simplemente una computadora personal o cualquier otro sistema que opera en la red. Definir nodos en Internet significa cualquier cosa con dirección IP.

### 1.1.17. *Smart Contract*

Es un código de computadora integrado en la cadena de bloques que se ejecuta automáticamente, permitiendo el intercambio de cualquier cosa de valor (dinero, propiedad, contenido, entre otros) cuando se cumplen ciertas condiciones, a saber, las del contrato [5].

### 1.1.18. Aplicación descentralizada (DApp)

Una DApp es completamente de código abierto, opera de forma autónoma, sin ninguna entidad que controle sus tokens, la aplicación genera tokens mediante un algoritmo estándar o un conjunto de criterios y distribuye sus tokens al comienzo de su operación. Sus datos y registros de operaciones se almacenan criptográficamente en una cadena de bloques [13].

### 1.1.19. *Ledger* distribuido

El *Ledger* distribuido o libro mayor distribuido se refiere a la capacidad de los usuarios para almacenar y acceder información o registros relacionados con activos y tenencias en una base de datos compartida. Un libro mayor distribuido está descentralizado para eliminar la necesidad de una autoridad central o intermediario para procesar,

validar o autenticar transacciones [14].

## 1.2. Planteamiento del problema

*Blockchain* es una tecnología relativamente nueva que ha mostrado robustez para el desarrollo de aplicaciones basadas en libros mayores distribuidos y seguridad en el manejo de datos, generando una demanda exponencial de aplicaciones descentralizadas sin tener en cuenta la ingeniería de software. Aunado al potencial disruptivo de esta tecnología, se observa la necesidad de aplicar metodologías y artefactos adecuados para el desarrollo de software, especialmente en el caso de *Ethereum* y *blockchain* similares que ofrecen el concepto de Contratos Inteligentes. La lógica de negocio en aplicaciones descentralizadas difiere en la implementación tradicional, donde una parte se encapsula en los contratos inteligentes; además se integra la transferencia de las unidades de valor o incentivos para la ejecución o activación de los contratos. Derivado de lo anterior se observa la necesidad de representar en forma adecuada la fragmentación de la lógica de negocio entre contratos inteligentes y componentes convencionales. Con base en lo reportado en la literatura es necesario conocer y aplicar los artefactos adecuados para modelar aplicaciones descentralizadas adecuadamente.

## 1.3. Objetivos

A continuación, se muestra el objetivo general y los objetivos específicos planteados para el presente proyecto.

### 1.3.1. Objetivo general

Desarrollar una aplicación descentralizada (DApp) aplicando una metodología y artefactos adecuados de ingeniería de software basadas en *blockchain*, mediante el estudio de las metodologías y artefactos reportados, para la implementación de inmutabilidad y transparencia de datos.

### 1.3.2. Objetivos específicos

1. Revisar los fundamentos de *blockchain* en particular y de *ledgers* distribuidos en general para conocer las bases de las aplicaciones descentralizadas.
2. Identificar y estudiar las metodologías reportadas de ingeniería de software basadas en *blockchain* para comprender los elementos disruptivos del desarrollo de aplicaciones descentralizadas
3. Analizar las características de las plataformas de *blockchain* que soportan DApps para la selección adecuada de tecnologías.
4. Desarrollar una DApp a partir de un caso de estudio utilizando las metodologías y tecnologías previamente seleccionadas.

## 1.4. Justificación

*Blockchain* es una tecnología P2P para aplicaciones descentralizadas que utilizan libros mayores transparentes, ofreciendo seguridad en los datos de punto a punto. Mas allá de las transacciones monetarias y financieras en general, la tecnología de *blockchain* se utiliza para rastrear y verificar cualquier tipo de intercambio digital como registros médicos y cuidado de la salud, cadenas de suministro y control por medio de NFC, IoT, música, deportes, regulación de la tierra, identificación personal, protección de datos, entre otros. Aunado a lo anterior, el aumento de aplicaciones móviles con seguridad mejorada y mayor calidad ha permitido el surgimiento de aplicaciones orientadas a *blockchain*. Las características clave de los sistemas de software orientado a *blockchain* (BOS) son las siguientes:

- Replicación de datos: los datos se replican y almacenan en muchos sistemas, lo que garantiza la seguridad de los datos. Cada nodo tiene una copia del código de *blockchain*.
- Verificaciones de requisitos: BOS verifican los requisitos de la transacción antes de procesar la validación.

- Registro de transacciones: BOS registran las transacciones en un registro secuencial de bloques interconectados que se crean mediante un algoritmo de consenso.
- Criptografía de clave pública: las transacciones se basan en la criptografía de clave pública.

Estas características hacen que el software basado en *blockchain* sea un gran beneficio para garantizar la seguridad de los datos. Conocer las metodologías y artefactos adecuados para modelar BOS es necesario para garantizar productividad en el desarrollo, reducir errores de codificación, proponer arquitecturas adecuadas y mejorar la documentación.

# **Capítulo 2**

## **Estado de la práctica**

En el presente capítulo se muestran algunos trabajos relacionados con el presente tema de tesis, asimismo se hace un análisis comparativo de estos, concentrando así la información más relevante de ellos.

### **2.1. Trabajos relacionados**

A continuación, se muestran 15 artículos seleccionados con base en el proyecto, de los cuales se realiza un resumen por cada artículo en donde se observa la similitud que tienen dichos artículos con el presente proyecto.

En [15], se establecieron los problemas actuales y las nuevas direcciones para BOSE y se investigó la necesidad de nuevas prácticas especializadas de ingeniería de software para el sector *blockchain*, para ello se identificaron los desafíos más relevantes para BOSE orientada al estado de la práctica; se destacaron las peculiaridades de algunos proyectos BOS y se propusieron nuevas direcciones de investigación para BOSE. Para identificar los desafíos y problemas que se originan en BOSE, se identificaron los elementos que conforman una cadena de bloques y se proporcionaron extractos de SWEBOK para enmarcar adecuadamente los problemas relacionados, se realizó un estudio exploratorio para definir nuevas direcciones de investigación para BOSE y se definió un proyecto BOS como un proyecto de software que contribuye a la realización de un proyecto *block-*

*chain*, con ello se extrajo información sobre popularidad, lenguajes de programación, participación de la comunidad y edad. Se encontró que existen pocos repositorios de BOS codificados en Java respecto al total de repositorios de BOS, asimismo se encontró que los 10 repositorios principales de BOS se crearon hace cuatro años en promedio y la mayoría tienen problemas, por lo anterior se sugieren nuevas direcciones de investigación para BOSE como: colaboración entre grandes equipos, actividades de prueba y herramientas especializadas para la creación de contratos inteligentes.

En [4], se analizó cómo la tecnología *blockchain* revoluciona a la ingeniería de software bajo el término BBSE. Al resolver dos problemas principales de SE como servicios de CI y *package managers*<sup>1</sup>, y además resolver los diversos desafíos dentro de *blockchain* como la verificación de contratos, la detección de vulnerabilidad e interoperabilidad. Para los servicios de CI, se propuso un sistema de integración continua basada en *blockchain* (BCI), en el cual después de que un desarrollador ingresa una compilación y el precio de recompensa en el *mempool* (área de espera para las transacciones de *Bitcoin*), se transmite a la red BCI, los interesados ejecutan la compilación e informan los resultados entre ellos, si se encuentra un consenso, se agrega la compilación a *blockchain* y para el repositorio de paquetes basado en *blockchain* se encapsula un marco de prueba de regresión impulsado por la comunidad para verificar los repositorios de paquetes. Al igual que BCI, cada participante en el repositorio es capaz de elegir un nuevo candidato de lanzamiento del *mempool*, verificar que el lanzamiento funcione según lo previsto y no rompa la compatibilidad con los clientes posteriores. Por lo anterior se determinó que una integración continua basada en *blockchain* permite a los desarrolladores especificar un alto costo de transacción en esta compilación; los nodos dan prioridad a la ejecución de este trabajo de construcción debido a su alto margen de beneficio; los desarrolladores son más flexibles, pero solo pagan por lo que necesitan en ese momento. Una BCI sirve como una prueba de trabajo viable de codificar y por medio de la firma de salida de registro de compilación. Al desarrollar un sistema distribuido que se ejecuta en *blockchain*, ya no hay un solo punto de falla, y abre el mercado para el poder

---

<sup>1</sup> *Package managers*: sistema de gestión de paquetes o gestor de paquetes.

de la computación en el que todos tienen la posibilidad de participar y ganar dinero con recursos inactivos. Esto a su vez también significa que las reglas económicas de la demanda y la oferta regulan los precios de construcción en más de una compañía.

Marchesi et al. [1] propusieron y probaron un proceso de diseño y desarrollo para aplicaciones *blockchain* basadas en contratos inteligentes que hace frente a problemas, ya que no se tienen en cuenta conceptos básicos de ingeniería de software, ni se asegura la calidad de software, por ello se propone el proceso de desarrollo de software para cumplir con los requisitos: analizar, diseñar, codificar, probar e implantar aplicaciones *blockchain*, haciendo uso de artefactos tradicionales como diagramas UML. El proceso general se basó principalmente en los principios del manifiesto ágil, complementado con algunas notaciones y prácticas específicas. *Blockchain* se usa también como una computadora descentralizada, ejecutando *Smart Contracts*, programas que se usan como base para la ejecución contractual automatizada. La idea principal detrás del enfoque propuesto surge de la observación de que un *Smart Contract* es un programa de software que se ejecuta en todos los nodos de una *blockchain* y cuyas salidas y estado deben ser los mismos en todos los nodos. A pesar del enorme esfuerzo actualmente en curso en el desarrollo de DApps, las prácticas de ingeniería de software todavía se aplican mal en el desarrollo de software de BOS. Las herramientas y técnicas de la ingeniería de software tradicional aún no se han adaptado y modificado para adherirse a este nuevo paradigma de software. Por ello se proporcionó un modelo completo de interacciones entre el software tradicional y el entorno *blockchain*, incluyendo diagramas UML y diagramas de contratos inteligentes, todo para BOSE, así como un esquema general para gestionar los procesos de desarrollo BOS, con esto se ayuda a la SE dándole un enfoque sólido que proporciona a los desarrolladores instrumentos similares a los que se usan en la ingeniería de software tradicional.

Rocha y Ducasse [16] explicaron que mediante el uso de contratos inteligentes es posible desarrollar software orientado a *blockchain* (BOS) que implemente parte de la lógica de negocios *blockchain*. Dado que el modelo es una parte importante del diseño de un software, es posible que los desarrolladores presenten dificultades para planificar el

desarrollo de BOS. Por ello se presentaron tres enfoques de modelado complementarios para BOS basado en los estándares de modelado: Modelo E-R , UML y BPMN a través de un ejemplo simple de un BOS, una tienda en línea que crea un programa de puntos de fidelidad basado en *blockchain*, los requisitos de puntos de fidelidad son similares al ejemplo de criptomoneda para contratos inteligentes. Cada modelo se centra en un aspecto particular: basado en datos, estructurado y proceso. Para los datos impulsados, se crea un modelo E-R que se utiliza principalmente para especificar datos en bases de datos relacionales. Para la estructura dirigida, se selecciona el diagrama de clase como el modelo entre los seis diagramas de estructura UML. Para el proceso impulsado, se utiliza la notación BPMN. Cada enfoque tiene sus puntos fuertes y débiles. El modelo E-R se utiliza para especificar los datos en BOS, pero es insuficiente para diseñar todos los aspectos de un BOS; el diagrama de clase modela la estructura interna de contratos inteligentes, pero llega a engañar a los desarrolladores sobre cómo implementar la interacción con la cadena de bloques y codificar objetos superfluos en la cadena de bloques; BPMN no modela adecuadamente los datos complejos o la estructura interna de los artefactos BOS. Sin embargo, el diagrama de carril<sup>2</sup> es útil para modelar la interacción entre la aplicación y la cadena de bloques. Aunque en [16] no se dio una solución general, se creó conciencia sobre la falta de anotaciones de modelado para BOS.

Destefanis et al. [10] abogaron por la necesidad de una disciplina de ingeniería de software *blockchain*, ya que los contratos inteligentes han generado problemas o conflictos, por lo que una disciplina de programación de contratos inteligentes y *blockchain* con mejores prácticas, ayudará a resolver los problemas y conflictos, por lo anterior surge la necesidad de una disciplina de ingeniería de software *blockchain*. Se consideró un caso de estudio, el ataque a Parity Wallet, en el que un desarrollador sin experiencia congeló varias cuentas administradas por la aplicación Parity Wallet, que causó la congelación de 500k *Ether*. Este caso de estudio representa un ejemplo pragmático de problemas que actualmente ocurren en el desarrollo de contratos inteligentes y software relacionado con *blockchain* en general. Dichos problemas están asociados a la

---

<sup>2</sup>Diagrama de carriles: es un diagrama de BPMN, es un tipo de diagrama de flujo que delimita quién hace qué en un proceso.

falta de mejores prácticas adecuadamente estandarizadas para la ingeniería de software *blockchain*. Malos comportamientos perjudiciales del software, que causaron pérdidas monetarias considerables y divisiones en la comunidad, han planteado el problema del diseño, validación y ejecución correctos de contratos inteligentes. Los problemas planteados mostraron claramente la necesidad de una ingeniería de software basada en *blockchain*, para definir nuevas direcciones y permitir un desarrollo de software efectivo. Se necesitan nuevos roles profesionales, seguridad y confiabilidad mejorada, marcos de modelado, verificación y métricas especializadas para llevar las aplicaciones *blockchain* al siguiente nivel. Tres áreas principales para comenzar a abordar han sido destacadas por el análisis del caso de estudio, mejores prácticas, patrones de diseño y pruebas. BO-SE crea un puente entre la ingeniería de software tradicional y el desarrollo de software con *blockchain*, definiendo nuevas metodologías, análisis de fallas, patrones, métricas de calidad, estrategias de seguridad y enfoques de prueba capaces de soportar un área novedosa y disciplina de ingeniería de software. Con todo lo anterior se descubrió que la vulnerabilidad de la biblioteca se debía principalmente a una actividad de programación negligente y no a un problema en el lenguaje *Solidity*.

Existen especulaciones sobre miles de criptomonedas disponibles y numerosas estafas de oferta inicial de monedas (ICO) que también han provocado debates sobre la tecnología *blockchain*. En [5], se investigó y se rastreó el desarrollo de sistemas *blockchain* para ver la importancia de las DApps y el futuro de *blockchain* para cumplir con las características deseables de las DApps. Una DApp desplegada no necesita mantenimiento ni gobierno de los desarrolladores originales, en otras palabras, una aplicación o servicio ideal de *blockchain* funciona sin intervención humana, lo que forma una Organización Autónoma Descentralizada. *Ethereum* ha alojado diferentes categorías de DApps, que incluyen intercambio, energía, finanzas, salud, identidad, seguros y medios de comunicación. Sin embargo, muchas DApps de última generación son de hecho solo parcialmente descentralizadas. Las futuras DApps exigen una plataforma *blockchain* que cumpla con baja latencia, habilitar transacciones *offline*, costo monetario razonable, mantenimiento flexible y gestión de identidad más simple; diferentes implementaciones de *blockchain*

con diferencias sutiles en áreas técnicas clave surgen constantemente para abordar diferentes deficiencias en los sistemas existentes. Al seleccionar una tecnología potencial de *blockchain*, se desea tener una implementación que sea estable y que esté dispuesta a ser flexible cuando sea necesario. Dependiendo de la DApp, se busca una tecnología *blockchain* que admita contratos inteligentes y alguna forma de pagos escalables. *Ethereum* está diseñado para ser una plataforma para facilitar contratos inteligentes descentralizados a través de su propia criptomoneda, el *Ether*. El contrato inteligente se refiere a la idea de que se notarian y se ejecutan automáticamente los contratos legales, equipados con Solidity, un lenguaje de programación Turing completo (un lenguaje con esta característica se aplica para resolver cualquier problema computacional e implementar estructuras complejas). A los desarrolladores de *Ethereum* les es posible implementar una serie de contratos inteligentes, que son programas ejecutables escritos en bloques debido a la naturaleza inmutable de los bloques.

En [2], se resumió una descripción general de la tecnología *blockchain*, la arquitectura, seguridad y desafíos técnicos. Las transacciones en *Bitcoin* nunca dependen de un tercero porque tiene una estructura de almacenamiento especialmente diseñada, debido a que *bitcoin* utiliza la tecnología *blockchain* para desarrollar. *Blockchain* es similar a un libro de contabilidad y las transacciones se almacenan en forma de bloques, sus características principales son la descentralización, persistencia, anonimato y auditabilidad, con esto reduce costos y se mejora la eficiencia de la aplicación, utiliza una red P2P para validar las transacciones sin una autorización central, para ello se utiliza un algoritmo de consenso, el cual permite que el nodo ofrezca una transacción válida y evite la adición de bloques y transacciones incorrectas. Hay dos principales algoritmos de consenso, *proof of work* y *proof of stake*. Se dice que *blockchain* es inmutable porque cada registro de la transacción una vez agregado no se es posible modificar ni destruir. La seguridad de *blockchain* se basa en que no hay una base de datos centralizada, los datos se descentralizan en diferentes nodos disponibles en la red. Se está tratando de ampliar la funcionalidad de la tecnología *blockchain*, para que esta sea sobresaliente, pero se necesitan importantes desarrollos técnicos. El nuevo desarrollo tecnológico en

la industria es la tecnología *blockchain* aunque originalmente se utilizó en criptomonedas, se está utilizando en otras áreas como: Internet de las Cosas, uso de contratos inteligentes y transmisión de contenido.

La validación de licencias de software ha crecido en complejidad debido a una combinación de desarrollo de tecnología y economía, las ventas y distribución de software se han vuelto más complejas y el software también se está volviendo más complejo conforme el alcance de uso aumenta, se han encontrado varias medidas prácticas para disuadir o prevenir la copia no autorizada, sin embargo, la viabilidad de estas depende de varios factores, como el costo de medida versus el valor del software. Por lo cual en [12], se utilizó una criptomoneda *blockchain* similar a la de *Bitcoin*, en la cual se proporcionó un método para software descentralizado y *peer to peer*, se hizo una descripción general de las funciones de la cadena de bloques de criptomonedas y el beneficio de un par descentralizado y la arquitectura *peer to peer*. Se necesitó de un mecanismo que generara valores únicos y que no se regeneraran, pero que pudieran verificarse fácilmente con el motor de origen en cualquier momento; para cumplir con las premisas para la validación de la licencia de software, se propuso un *blockchain* de criptomonedas para crear un método novedoso para el mecanismo de validación de licencia de software. A través de la arquitectura descentralizada de *blockchain peer to peer*, cualquier desarrollador o proveedor de software es capaz de asignar licencias a los usuarios de manera fácil y rentable. Dos métodos principales para utilizar una cadena de bloques para la validación de la licencia de software son el modelo maestro de *Bitcoin* y el modelo a la medida. El Modelo maestro de *Bitcoin* se usa para proporcionar una prueba de propiedad no repudiable de un *Bitcoin* que se originó en una dirección específica, con lo que confiere el derecho de la licencia de software al usuario. Sin embargo, la aplicación de software debe tener la capacidad de leer la cadena de bloques para establecer la cadena de título para el usuario. El modelo a la medida utiliza una especificación de transacción de *blockchain* personalizada que incluye campos adicionales adaptados a los requisitos de un esquema de validación de licencia de software flexible. Esto proporciona el alcance necesario para la amplia gama de usuarios y modelos de licencia en el entorno

tecnológico moderno. También proporciona varios mecanismos útiles usando *blockchain* como base para la validación de la licencia, actualización de licencia, transferencia de propiedad y software, la validación de la licencia se logra fácilmente usando la cadena de bloques, y a través del mismo mecanismo, se agregan protecciones adicionales de integridad y seguridad.

La comunicación distribuida ha sido durante mucho tiempo un problema no resuelto en el campo del control distribuido y la robótica cooperativa. Se ha informado sobre fallas en la red y ataques de intrusos de control distribuido y sistemas multi-robóticos. En [9] se creó una imagen global de la tecnología *blockchain* sobre su principio de funcionamiento y elementos clave en el lenguaje de control y robótica, En general, los DCS enfrentan cuatro tipos de problemas de seguridad: intercepción, interrupción, modificación y fabricación. En el campo de los robots cooperativos los investigadores enfrentan problemas de seguridad, toma de decisiones distribuidas y variación en el comportamiento; *blockchain* con su naturaleza descentralizada y segura proporciona seguridad, además de que facilita la solución a los diversos problemas en DCS y robots cooperativos. La descentralización es una característica clave de *blockchain*, las amenazas de seguridad como la intercepción e interrupción se abordan mediante el uso de *blockchain* en los DCS, la descentralización replica los datos a todos los nodos conectados al sistema, con esto cuando intentan alterar una copia de los datos, el resto permanece intacto. En los robots la descentralización hace que los robots sean más seguros, ya que los vuelven independientes de la intervención de terceros, el *hashing* (algoritmo) de *blockchain* hace que los datos sean seguros porque si alguien intenta invadir la privacidad o alterar los datos, el *hash* del bloque cambiará, lo que resulta en la ruptura de la cadena y que la solicitud sea inválida. *Blockchain* ayuda en la toma de decisiones distribuida, debido a que cuando se propone alguna decisión se publica en algún bloque con una dirección particular para cada opción, el resto de los nodos votará por las decisiones accediendo a sus direcciones, la decisión con el máximo de votos entrará en acción. *Blockchain* revolucionará el campo de los DCS y los robots, y les proporcionará un entorno más seguro e infalible.

En [17] se examinaron casos de uso en fuentes existentes, como tesis, artículos y blogs de expertos de la industria; se hizo una descripción principal de los primordiales aspectos tecnológicos y principios de funcionamiento de *blockchain*, con esto se evaluaron los casos de uso; se realizó un análisis básico de la viabilidad de la solución propuesta en los artículos, porque con frecuencia un artículo o incluso un artículo científico propone solo una descripción teórica sin una idea clara de implementación física. *Blockchain* resuelve muchos problemas, sin embargo, existen varios desafíos técnicos como la escalabilidad y la integridad de los participantes en la red, existen muchas comparaciones de ventajas y desventajas del *blockchain*. Esto resalta la necesidad de una investigación y evaluación adicional de diferentes opciones de aplicación *blockchain*, a partir de los casos de uso, quedó claro que la mayoría de las aplicaciones *blockchain* se relacionan con la gestión y verificación de datos, principalmente desarrolladas en el sector financiero, otros sectores como las instituciones gubernamentales, son menos flexibles y, por lo tanto, la adaptación de *blockchain* es significativamente más lenta. Cabe mencionar que la investigación en el modelo de evaluación de *blockchain* está en una etapa muy temprana, y es necesaria una mayor investigación, con la evaluación correcta de la necesidad de *blockchain*, se ahorra una gran cantidad de recursos en costos de desarrollo y mantenimiento de software.

El desarrollo de software en *blockchain* está en su infancia en consideración de la línea de tiempo, sin embargo, la alta demanda provocó su expansión a un ritmo sin precedentes, los desarrolladores están incentivados para crear aplicaciones *blockchain*, más de 3000 desarrolladores contribuyen regularmente a alrededor de 2087 repositorios de *blockchain* alojados en Github<sup>3</sup>, sin embargo, con el ecosistema que cambia rápidamente y los cronogramas de implementación ajustados, es raro diseñar una arquitectura robusta, revisar códigos, probar la funcionalidad, el rendimiento, la escalabilidad por parte de los propios desarrolladores o expertos externos. Por ello, en [18], se estableció como objetivo llevar a cabo una encuesta formal para explorar las prácticas de ingeniería de software que incluyen análisis de requisitos, asignación de tareas, pruebas y verificación.

---

<sup>3</sup>GitHub es una forja para alojar proyectos utilizando el sistema de control de versiones Git.

Específicamente, se interesaron en la opinión de los desarrolladores sobre las prácticas de desarrollo en *blockchain*. La encuesta fue un instrumento ideal para el estudio, ya que los desarrolladores tienen experiencias de sus desafíos y necesidades, se adoptó un enfoque sistemático de análisis cualitativo para construir un esquema de codificación, utilizando un software de análisis cualitativo, múltiples codificadores asignaron códigos de forma independiente a cada respuesta, logrando una confiabilidad sustancial, el estudio encontró algunas similitudes de aplicaciones BCS con proyectos generales de OSS. Con respecto a la práctica de ingeniería de software, la revisión de código y las pruebas unitarias son las dos más populares entre los desarrolladores de BCS, y apenas prefieren la programación de pares, que es muy popular entre los desarrolladores de OSS. En conclusión, los proyectos en *blockchain* están aumentando considerablemente, y la capitalización de mercado se ha vuelto sorprendentemente alta, mientras que la investigación empírica de SE que explora esta área es bastante inadecuada. En consecuencia, las prácticas de ingeniería de software no se estudian sustancialmente y la limitación de las prácticas no se aborda. Este estudio ha cerrado esta brecha y muestra los detalles de las opiniones de los propios desarrolladores y se sugirió que se sigan las prácticas de SE en el caso de BCS de manera efectiva, mientras que la escasez de soporte se establece en algunas pruebas críticas y requisitos de análisis de seguridad.

En [19], se propuso un conjunto de patrones para el diseño de aplicaciones basadas en *blockchain*, muchas empresas y gobiernos están explorando aplicaciones basadas en *blockchain* en áreas tan diversas, pero a pesar del interés, no se reporta una visión sistemática y holística cuando se aplica *blockchain* al diseño de aplicaciones de software dado que las tecnologías *blockchain* se encuentran en una etapa temprana. Se recopiló una lista de patrones para aplicaciones basadas en *blockchain*, la colección de patrones se clasifica en cuatro tipos: patrones del mundo externo, patrones de gestión de datos, patrones de seguridad y patrones estructurales contractuales, dichos patrones están diseñados teniendo en cuenta la naturaleza de *blockchain* y cómo es posible introducir los patrones específicamente en aplicaciones del mundo real. La adopción de un patrón de diseño provoca compensaciones entre los atributos de calidad. La colección de patrones

incluye tres patrones sobre la interacción entre *blockchain* y el mundo externo, cuatro patrones de gestión de datos, tres patrones de seguridad y cinco patrones estructurales de contrato inteligente. La colección de patrones proporciona una guía arquitectónica para que los desarrolladores creen aplicaciones en *blockchain*, para aprovechar mejor las propiedades positivas de *blockchain* y evitar limitaciones.

Se piensa que *blockchain* terminará modificando una serie de aplicaciones importantes, desde la atención médica, hasta el gobierno. La tecnología *blockchain* aparentemente es simple, pero tiene inmensas aplicaciones que se extienden a través de los negocios globales y la civilización, en un futuro los consumidores se beneficiarán con las características de *blockchain* sin siquiera considerarlo. En [20], se estudió qué es la tecnología *blockchain*, cómo funciona, ventajas y desventajas e implementación de *blockchain* utilizando Java. La tecnología detrás de *Bitcoin* es *blockchain*, comúnmente consideran *Bitcoin* y *blockchain* de manera similar. Pero las aplicaciones de *blockchain* son más que el *Bitcoin*, *blockchain* es muy versátil. La base de datos de *blockchain* puede contener datos de automóviles, registros médicos y mucho más, en el caso que se necesite mantener los datos en condiciones que no se modifiquen, deben ser seguros, abiertos al usuario y descentralizados, entonces *blockchain* es útil. Los datos que se almacenan utilizando una estructura de datos de *blockchain* son resistentes a los cambios, ya que después de desarrollar un bloque, será necesario modificar todos los bloques después de él. Revisando la implementación de *blockchain* usando Java, la cadena de bloques basada en Java se mantuvo de forma voluntaria y, por lo tanto, desde el punto de vista de los usuarios, solo tiene la función de enviar mensajes y verlos en la cadena de bloques, bastante similar a una sala de chat pública. *Blockchain* garantiza resolver problemas y permite a los desarrolladores poner en práctica fácilmente la tecnología *blockchain* utilizando el lenguaje de programación Java.

En [21] se analizó el estado del arte, centrándose en *blockchains* privados y se estudiaron sistemas de producción y de investigación en cuatro dimensiones: libro mayor distribuido, criptografía, protocolo de consenso y contrato inteligente. Se presentó Blockbench, el cual es un marco de referencia para comprender el rendimiento de *block-*

*chains* privados frente a cargas de trabajo de procesamiento de datos. Es importante y desafiante tener una comprensión firme de lo que las tecnologías centrales tienen que ofrecer, especialmente con respecto a sus capacidades de procesamiento de datos, en ese contexto *blockchain* es una solución para la gestión de transacciones distribuidas: los nodos mantienen réplicas de los datos y acuerdan un orden de ejecución de las transacciones. Si bien es un desafío, es importante tener una idea clara de lo que la tecnología es capaz de hacer y de no hacer. Usando Blockbench, se realizó una evaluación integral de tres cadenas de bloques principales: *Ethereum*, Parity y *Hyperledger*. De los tres sistemas, *Ethereum* es más maduro tanto en términos de su base de código, base de usuarios y comunidad de desarrolladores. Otro problema de usabilidad que se encuentra es la transferencia de contratos inteligentes de un sistema a otro, debido a sus diferentes modelos de programación. Los resultados demostraron varias compensaciones en el espacio de diseño, así como grandes brechas de rendimiento entre *blockchain* y sistemas de bases de datos. A partir de los principios de diseño de los sistemas de bases de datos, se discutieron varias instrucciones de investigación para acercar el rendimiento de *blockchain* al ámbito de las bases de datos y mejorar futuras cadenas de bloques.

Las aplicaciones basadas en *blockchain* son capaces de aprovechar propiedades como la inmutabilidad de los datos, la integridad, el acceso justo, la transparencia y el no repudio de las transacciones. Sin embargo, *blockchain* tiene limitaciones técnicas y los arquitectos que diseñan aplicaciones basadas en *blockchain* tienen dificultades, ya que la tecnología tiene muchas configuraciones y variantes, dado que *blockchain* está en una etapa temprana, hay pocos datos de productos o evaluaciones tecnológicas confiables y disponibles para comparar diferentes *blockchains*. Por ello en [22], se propuso una taxonomía de *blockchains* y sistemas basados en *blockchain* que ayuden con el diseño y la evaluación de su impacto en las arquitecturas de software, al crear aplicaciones basadas en *blockchain*, se deben considerar sistemáticamente las características y configuraciones de *blockchain* y evaluar su impacto en los atributos de calidad para los sistemas en general, por ello en esta taxonomía se capturan las principales características arquitectónicas relevantes de varias cadenas de bloques e indica su apoyo a diversos atributos

de calidad. Con ello se pretende hacer una base para la arquitectura de sistemas basados en *blockchain*. En el diseño, la taxonomía resalta las compensaciones derivadas de las decisiones de diseño relacionadas con las plataformas *blockchain*, esta taxonomía se basó en productos industriales, foros técnicos, literatura académica y propia experiencia en el uso de *blockchain* y el desarrollo de prototipos. La taxonomía se utiliza durante el proceso de arquitectura de sistemas de software para guiar el diseño del sistema y ayudar a los arquitectos de software a evaluar y comparar *blockchains*, y permitir la investigación en marcos de toma de decisiones arquitectónicas para *blockchains* y sistemas basados en *blockchain*.

## 2.2. Análisis del estado de la práctica

La tabla comparativa 2.1 muestra el análisis de 15 artículos de trabajos relacionados con el tema a desarrollar, tomando en cuenta el problema, contribución, tecnologías, resultado y estado.

Tabla 2.1: Cuadro comparativo de trabajos relacionados.

<b>Artículo</b>	<b>Problemas</b>	<b>Contribución</b>	<b>Tecnología</b>	<b>Resultado</b>	<b>Estado</b>
[15]	Nuevos roles profesionales, seguridad y fiabilidad, arquitectura de software, lenguajes de modelado y métrica.	A la ingeniería de software orientada a <i>blockchain</i> .	<i>Blockchain</i> .	Se encontraron muchos problemas por lo que, surgieron nuevas direcciones de investigación para BOSE como: colaboración entre grandes equipos, actividades de prueba y herramientas especializadas para la creación de contratos inteligentes.	Concluido

Tabla 2.1 Cuadro comparativo de trabajos relacionados.

Artículo	Problemas	Contribución	Tecnología	Resultado	Estado
[4]	SE: servicios de integración continua y <i>package managers</i> . <i>Blockchain</i> : verificación de contratos, detección de vulnerabilidad e interoperabilidad.	Revolución a la ingeniería de software bajo el concepto BBSE.	<i>Blockchain</i> .	Con BCI: Los desarrolladores son más flexibles, solo pagan por lo que necesitan en ese momento. Una BCI sirve como una <i>proof of work</i> viable de codificar y por medio de la firma de salida de registro de compilación. Implementado un sistema distribuido que se ejecuta en <i>blockchain</i> , ya no hay un solo punto de falla, abre el mercado para el poder de la computación, las reglas económicas de la demanda y la oferta regulan los precios de construcción.	En desarrollo.

Tabla 2.1 Cuadro comparativo de trabajos relacionados.

<b>Artículo</b>	<b>Problemas</b>	<b>Contribución</b>	<b>Tecnología</b>	<b>Resultado</b>	<b>Estado</b>
[1]	No se tienen en cuenta conceptos básicos de ingeniería de software y se aplica mal en el desarrollo de software BOS. No es segura la calidad de software.	A la SE dándole un enfoque sólido y a los desarrolladores proporcionando instrumentos similares a la SE tradicional.	Modelado UML, <i>Blockchain</i> .	Se proporciona un modelo completo de interacciones entre el software tradicional y el entorno <i>blockchain</i> , incluyendo diagramas UML y diagramas de contratos inteligentes, todo para BOSE. Con la cual se obtiene una solidez para el desarrollo de BOS.	Concluido.
[16]	No hay un estándar de diseño para modelar BOS.	Facilidad al desarrollar y planificar BOS.	Solidity, Modelos E-R, UML y BPMN.	Mediante el ejemplo práctico se observó que los modelos E-R, UML y BPMN tienen sus puntos fuertes y débiles, y se necesita una notación especializada para BOS para diseñarlo adecuadamente.	En desarrollo.

Tabla 2.1 Cuadro comparativo de trabajos relacionados.

<b>Artículo</b>	<b>Problemas</b>	<b>Contribución</b>	<b>Tecnología</b>	<b>Resultado</b>	<b>Estado</b>
[10]	Generación de conflictos por contratos inteligentes, debido a la necesidad de ingeniería de software orientada a <i>blockchain</i> .	Análisis de fallas, patrones, métricas de calidad, estrategias de seguridad y enfoques de prueba capaces de soportar un área novedosa y disciplina de ingeniería de software.	<i>Blockchain</i> .	Se logró entender la necesidad de una ingeniería basada en <i>blockchain</i> , se descubrió que la vulnerabilidad se debe principalmente a una programación negligente (falta de BOSE).	Concluido.
[5]	No se sabe qué tan importante es una DApp.	Entender la importancia, así como las características de una DApp	<i>Blockchain</i> . <i>Solidity</i> .	Se logró entender el funcionamiento de una DApp, sus características y las categorías que existen hasta este momento. Asimismo la necesidad de una plataforma <i>blockchain</i> que admite contratos inteligentes.	Concluido.
[2]	Desafíos técnicos y falta de conocimiento sobre <i>blockchain</i> .	Ayudar a entender la tecnología <i>blockchain</i> , su arquitectura y seguridad.	No se menciona.	Se entendieron las características de <i>blockchain</i> y cómo funciona su arquitectura y los desafíos que conlleva como la seguridad.	En desarrollo.

Tabla 2.1 Cuadro comparativo de trabajos relacionados.

<b>Artículo</b>	<b>Problemas</b>	<b>Contribución</b>	<b>Tecnología</b>	<b>Resultado</b>	<b>Estado</b>
[12]	Copia no autorizada (piratería).	A la validación de licencia de software.	<i>Blockchain.</i>	Mecanismos útiles, utilizando <i>blockchain</i> como base para la validación de licencia y actualización. Protección de integridad y seguridad.	Concluido.
[9]	Fallas en la red y ataques de intrusos de control distribuido y sistemas multi-robóticos.	A los DCS y a los robots enjambre, para que tengan un entorno más seguro e infalible.	<i>Blockchain.</i>	<i>Blockchain</i> facilita diversos problemas en DCS y robots cooperativos.	Concluido.
[17]	La necesidad de una investigación y evaluación adicional de diferentes opciones de aplicación <i>blockchain</i> .	No se especifica.	No se especifica.	Quedó claro que la mayoría de las aplicaciones <i>blockchain</i> se relacionan con la gestión y verificaciones de datos, principalmente desarrolladas en el sector financiero.	Concluido.

Tabla 2.1 Cuadro comparativo de trabajos relacionados.

<b>Artículo</b>	<b>Problemas</b>	<b>Contribución</b>	<b>Tecnología</b>	<b>Resultado</b>	<b>Estado</b>
[18]	Investigación empírica de SE por lo que las prácticas de ingeniería de software no se estudian sustancialmente.	No se especifica.	Software de análisis cualitativo.	Mediante la opinión de los desarrolladores, se sugirió llevarse a cabo las prácticas de SE en el caso de BCS de manera efectiva, mientras que la escasez de soporte se establece en algunas pruebas críticas y requisitos de análisis de seguridad.	Concluido.
[19]	Se reporta que no hay una visión sistemática y holística cuando se aplica <i>blockchain</i> al diseño de aplicaciones de software.	A mejorar las propiedades de <i>blockchain</i> y evitar limitaciones.	No se especifica.	La colección de patrones proporcionó una guía arquitectónica para que los desarrolladores creen aplicaciones en <i>blockchain</i> .	En desarrollo.

Tabla 2.1 Cuadro comparativo de trabajos relacionados.

<b>Artículo</b>	<b>Problemas</b>	<b>Contribución</b>	<b>Tecnología</b>	<b>Resultado</b>	<b>Estado</b>
[20]	<i>Bitcoin</i> y <i>blockchain</i> se consideran iguales y la tecnología <i>blockchain</i> se ve aparentemente simple.	Entender que la tecnología detrás de <i>Bitcoin</i> es <i>blockchain</i> y entender las vastas características que ofrece <i>blockchain</i> .	Java. <i>Block-chain</i> .	La tecnología <i>blockchain</i> garantiza resolver diversos problemas y poner a <i>blockchain</i> en práctica fácilmente utilizando Java.	Concluido.
[21]	No se tiene una idea clara de lo que la tecnología <i>blockchain</i> puede y no puede hacer.	A <i>blockchain</i> para acercar su rendimiento al ámbito de las bases de datos.	<i>Blockchains</i> privados.	Los resultados demuestran varias compensaciones en el espacio de diseño, así como grandes brechas de rendimiento entre <i>blockchain</i> y sistemas de bases de datos. Pero a partir y utilizando los principios de diseño es posible mejorar futuras cadenas de bloques.	En desarrollo.

Tabla 2.1 Cuadro comparativo de trabajos relacionados.

<b>Artículo</b>	<b>Problemas</b>	<b>Contribución</b>	<b>Tecnología</b>	<b>Resultado</b>	<b>Estado</b>
[22]	Muchas configuraciones y variantes dentro de <i>blockchain</i> .	Ayudar con importantes consideraciones arquitectónicas sobre el rendimiento y los atributos de calidad (disponibilidad, seguridad y rendimiento) de los sistemas basados en <i>blockchain</i> .	<i>Blockchain</i> .	La taxonomía se utiliza durante el proceso de arquitectura de sistemas de software y permite la investigación en marcos de toma de decisiones arquitectónicas para <i>blockchains</i> y sistemas basados en <i>blockchain</i> .	Concluido.

Después de haber analizado los artículos de la tabla 2.1, se concluye que en algunos de estos artículos se habla de la ingeniería de software en el ámbito de *blockchain*, sin embargo, la mayoría de aplicaciones *blockchain* se crean de manera apresurada, sin aplicar ingeniería de software y aún aplicando SE en [15], se afirmó que la mayoría de los repositorios tienen problemas abiertos, es por ello que se sugieren nuevas direcciones de investigación para BOSE y se propone la implementación de ingeniería de software basada en *blockchain* o ingeniería de software orientada a *blockchain*. En [1], se realizó una prueba al proceso de diseño y desarrollo de aplicaciones *blockchain*, haciendo frente a problemas de ingeniería de software y a la falta de calidad de software mediante el uso de UML. Además del uso de UML, en [16] se describió un modelo complementario para BOS basado en estándares de modelado: modelo E-R, UML y BPMN, sin embargo, aun hay mucho trabajo por hacer para aplicar herramientas y técnicas de ingeniería de software de manera adecuada en el desarrollo de software orientado a *blockchain*, por ello en [10] se hace mención de la necesidad de una disciplina estandarizada de ingeniería de software *blockchain*, ya que esta disciplina con mejores prácticas, ayudará a resolver los problemas y conflictos. Realizando una comparativa con base en lo anterior, en gran parte de los trabajos relacionados se hacen propuestas de implementar ingeniería de software basada u orientada a *blockchain*, mientras que en el presente proyecto, se busca el desarrollo de una DApp aplicando una metodología y artefactos, mediante un estudio profundo de metodologías y artefactos reportados, para llevar a cabo un modelado correcto del software a desarrollar.

## 2.3. Solución propuesta

En esta sección se presenta la propuesta de la solución que ofrece mayores ventajas para dar solución a la problemática que plantea.

### 2.3.1. Justificación de la solución seleccionada

La elección de la solución se basa en el desempeño y compatibilidad de las tecnologías, así como las cualidades ofrecidas por la metodología de desarrollo de software.

En la solución propuesta se contempla el uso de la plataforma *Hyperledger*, la cual es un tipo de *blockchain* con permiso, tiene un alto rendimiento de transacciones y tiene mejor protección contra perturbaciones externas. Ademas *Hyperledger* cuenta con múltiples marcos de trabajo por lo cual se optó por *Hyperledger Fabric* debido a que es muy versátil y es modular, es decir, no se especializa en un solo sector, por otra parte *Hyperledger Fabric* tiene soporte para los lenguajes Java, Node.js y Go. Para el desarrollo de *Smart contracts* se optó por el lenguaje de programación Go (*Back-end*) y Node.js para el *front-end*. La metodología propuesta es una metodología híbrida tomando como base dos metodologías ágiles aprovechando las características que *Scrum* y XP ofrecen. *Scrum* ofrece la planificación y el manejo de iteraciones que permitirá un desarrollo eficiente del proyecto; XP ofrece la capacidad para enfrentar cambios y una comunicación fluida, ademas de ser la metodología ágil más específica respecto a prácticas de ingeniería. Con el uso de estas dos metodologías se obtiene un desarrollo de software completo.

# **Capítulo 3**

## **Aplicación de la metodología**

En el presente capítulo se describen las actividades que se realizaron para el desarrollo de la tesis.

### **3.1. Descripción de la solución**

Con base en el planteamiento del problema y en función de la disponibilidad de los recursos tecnológicos para la construcción del producto de este proyecto académico, se desarrolló una aplicación descentralizada tomando en cuenta la necesidad de aplicar metodologías y artefactos adecuados para el desarrollo de software orientado a *blockchain*, para representar la lógica de negocio entre contratos inteligentes y componentes convencionales. La solución propuesta se enfoca en desarrollar una aplicación descentralizada, la cual está basada en la tecnología *blockchain*, esta tecnología proporciona seguridad, debido a que la cadena de bloques es una base de datos compartida P2P (*peer to peer*, igual a igual) que permite almacenar información de manera inmutable y ordenada. Dicha aplicación se desarrolló con base en un conjunto de artefactos identificados en trabajos relacionados, obteniendo de esta manera el modelado y la aplicación de ingeniería de software basada en *blockchain*. Asimismo se hace uso de una metodología híbrida (SXP), tomando como base dos metodologías ágiles: *eXtreme programming* y SCRUM.

### 3.2. Análisis de artefactos

El modelado es una parte importante del diseño de un software y al no realizar un modelado correcto, los desarrolladores tienen dificultades para planificar su software orientado a *blockchain*, tomando en cuenta lo anterior mediante la identificación de artefactos con base en sus ventajas y desventajas, se seleccionaron una serie de artefactos, para evitar problemas y conflictos durante el desarrollo de la DApp. A continuación, se mencionan los artefactos identificados, para ayudar a modelar BOS, se identificó el uso de diagramas UML, Modelo ER, y Notación BPMN. Sin embargo, para algunos diagramas se introducen nuevos conceptos como estereotipos UML. Los artefactos para modelar Software orientado a *blockchain*:

- **Diagrama de casos de uso.**

No presenta cambios, ya que este diagrama representa la forma en que el cliente opera con el sistema.

- **Diagrama de clases.**

Representa la estructura y las relaciones de los contratos inteligentes, se introdujeron varios estereotipos en este tipo de diagrama, que se muestran en la tabla 3.1. Además, tiene la ventaja de modelar atributos y funciones. De igual manera se propone el uso de un ícono llamado “*chain*” para la representación gráfica del contrato como una clase, como se observa en la figura 3.1. Por otro lado, un diagrama de clases no es adecuado para modelar todas las relaciones de datos en un BOS.

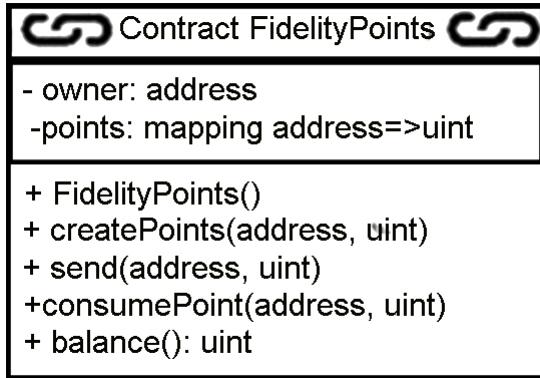


Figura 3.1: Ejemplo del uso de ícono “chain” en el diagrama clases.

Tabla 3.1: Estereotipos para representar contratos inteligentes en diagrama de clases UML.

Estereotipo	Descripción
«contract»	Denota un contrato inteligente.
«library contract»	Representa un contrato tomado de alguna biblioteca (estándar).
«struct»	Representa una estructura que contiene datos pero ninguna operación definida y utilizada en la estructura de datos de un contrato.
«enum»	Una enumeración, que contiene solo una lista de posibles valores.
«interface»	Un contrato que contiene solo declaraciones de funciones.
«modifier»	Un tipo particular de función, definida en <i>Solidity</i> .
«array»	La relación 1: n se implementa utilizando una matriz.
«map»	La relación 1: n se implementa mediante un mapeo.
«map[uint]»	La relación 1: n se implementa utilizando un mapeo de entero al valor.

- **Diagrama de estado.**

Representa los diversos estados de un contrato inteligente; este diagrama no necesita ningún concepto nuevo.

- **Diagrama de secuencia.**

Representa los mensajes enviados a un contrato inteligente o de un contrato inteligente a otro contrato inteligente. Si se hace uso de la plataforma *Ethereum*, el diagrama de secuencia necesita un nuevo tipo de mensaje: la transferencia de ethers. Se introdujeron una serie de estereotipos, mostrados en la tabla 3.2.

Tabla 3.2: Estereotipos para el diagrama de secuencia UML.

Estereotipo	Descripción
«person»	Un rol humano, para enviar mensajes usando una billetera u otra aplicación.
«system»	Un sistema externo, capaz de enviar mensajes a <i>blockchain</i> .
«device»	Un dispositivo (generalmente IoT), capaz de enviar mensajes.
«contract»	Representa un contrato inteligente.
«oracle»	Un tipo particular de contrato inteligente, cuyas fechas están escritas por un tercero confiable y permiten acceder a información sobre el mundo externo. Tiene solo declaraciones de funciones.
«account»	Una cuenta de <i>Ethereum</i> , solo con <i>Ethers</i> . Solo puede recibir <i>Ethers</i> , o enviar <i>Ethers</i> a otra cuenta o contrato inteligente si el propietario activa la transferencia.

- **Modelo ER.**

El modelo ER es adecuado para capturar los datos y su relación, el modelo ER podría ser la única notación adecuada para especificar la relación de datos entre

*blockchain* y una base de datos relacional. Dado que los datos en *blockchain* son de acceso público, es importante no exponer datos sensibles en contratos inteligentes, por ello se verifica lo que se coloca en la cadena de bloques, para luego vincular los datos del contrato con la base de datos relacional. En la figura 3.2 se presenta un ejemplo del modelo ER, la dirección de la cadena de bloques de los clientes y su cantidad de puntos están expuestos. Sin embargo, no es posible adquirir el nombre y la dirección particular de los clientes sin acceder a la base de datos relacional. Tanto los datos de *blockchain* como los de la base de datos relacional están vinculados por la relación entre las entidades *Point* y *Client* de acuerdo a la figura 3.2.

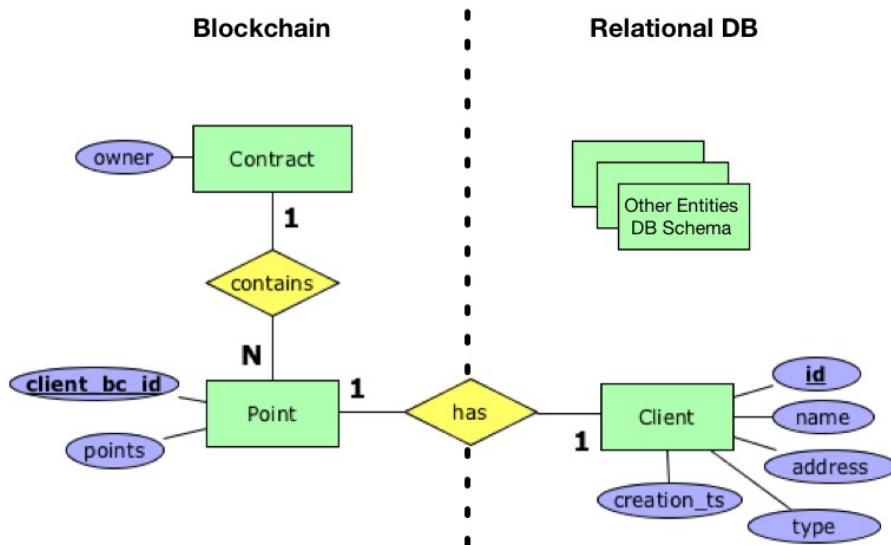
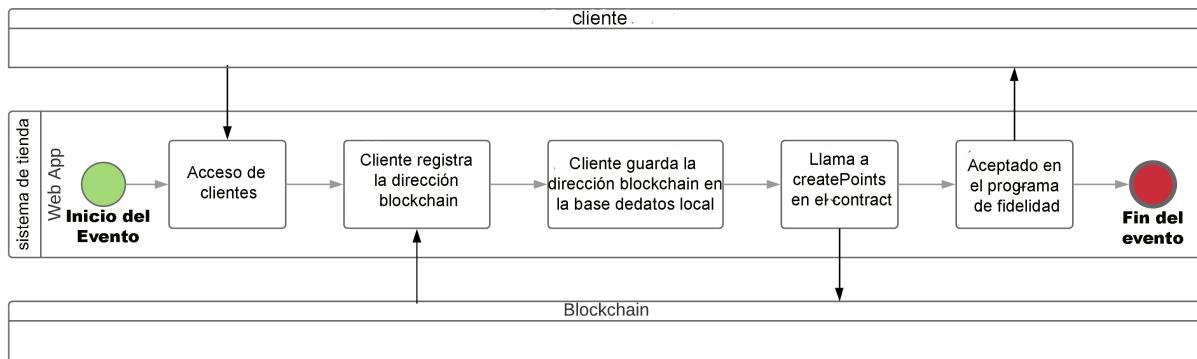


Figura 3.2: Ejemplo de modelo ER, entre Base de datos relacional y *blockchain*

#### ■ Notación BPMN.

Esta notación es la más adecuada para especificar el proceso empresarial. Dado que el comportamiento del proceso en sí, en un BOS, podría requerir una especificación de diseño más detallada. BPMN no es capaz de modelar adecuadamente datos complejos o la estructura interna de artefactos BOS. Sin embargo, la notación de carril es útil para modelar la interacción entre la aplicación y la cadena de bloques. A continuación en la figura 3.3 se muestra un ejemplo.

Figura 3.3: Ejemplo de notación de carril para *blockchain*

De igual manera, se identificaron algunos patrones de diseño que se clasifican en cuatro tipos: Interacción de patrones y el mundo externo, gestión de datos, seguridad y patrones estructurales del contrato. Algunos patrones están diseñados teniendo en cuenta la naturaleza de *blockchain* y cómo se pueden introducir específicamente en aplicaciones del mundo real. Otros son variantes de patrones de diseño existentes aplicados en el contexto de aplicaciones basadas en *blockchain* y contratos inteligentes. En la tabla 3.3 se muestra el nombre del patrón con una descripción del mismo.

Tabla 3.3: Patrones de diseño orientados a software *blockchain*.

Categoría	Nombre	Resumen
Interacción con el mundo externo	<i>Oracle</i>	Introduce el estado de los sistemas externos en el entorno de ejecución de <i>blockchain</i> cerrado.
	<i>Reverse Oracle</i>	Los componentes fuera de la cadena de un sistema existente dependen de contratos inteligentes que se ejecutan en una cadena de bloques para suministrar los datos solicitados y verificar las condiciones requeridas.
	<i>Legal and Smart contract pair</i>	Establece un enlace bidireccional entre un acuerdo legal y un contrato inteligente correspondiente.

Tabla 3.3 Patrones de diseño orientados a software *blockchain*.

Categoría	Nombre	Resumen
Gestión de datos	<i>Encrypting on-chain data</i>	Asegura la confidencialidad de los datos almacenados en <i>blockchain</i> cifrándolos.
	<i>Tokenisation</i>	Usar <i>tokens</i> en <i>blockchain</i> para representar activos y servicios digitales o físicos transferibles.
	<i>Off-chain data storage</i>	Utiliza el <i>hash</i> para garantizar la integridad de conjuntos de datos arbitrariamente grandes que pueden no encajar directamente en la cadena de bloques.
	<i>State channel</i>	Las transacciones que tienen un valor demasiado pequeño en relación con una tarifa de transacción de cadena de bloques o que requieren una latencia mucho más corta que la que puede proporcionar una cadena de bloques, se realizan fuera de la cadena con el registro periódico de las liquidaciones de transacciones netas en cadena.
Seguridad	<i>Multiple authorization</i>	Un conjunto de direcciones de <i>blockchain</i> que puede autorizar una transacción está predefinido. Solo se requiere un subconjunto de las direcciones para autorizar transacciones.
	<i>Off-chain secret enable dynamic authorization</i>	Usar un <i>hash</i> creado fuera de la cadena para vincular dinámicamente la autoridad para una transacción.

Tabla 3.3 Patrones de diseño orientados a software *blockchain*.

Categoría	Nombre	Resumen
Patrones estructurales de contrato	<i>X-confirmation</i>	Espera una cantidad suficiente de bloques como confirmaciones para garantizar que una transacción agregada a <i>blockchain</i> sea inmutable con alta probabilidad.
	<i>Contract registry</i>	Antes de invocar un contrato inteligente, la dirección de la última versión del contrato inteligente se encuentra buscando su nombre en un registro de contrato.
	<i>Embedder permission</i>	Los contratos inteligentes utilizan el control de permisos incorporado para restringir el acceso a la invocación de las funciones definidas en los contratos inteligentes.
	<i>Data contract</i>	Almacena datos en un contrato inteligente por separado.
	<i>Factory Contract</i>	Un contrato de plantilla en cadena se usa como una fábrica que genera instancias de contrato a partir de la plantilla.
	<i>Incentive execution</i>	Se proporciona una recompensa al invocador de una función de contrato por invocarla.

### 3.3. Metodología

Como metodología se eligió una metodología híbrida (SXP), la cual tiene como base dos metodologías ágiles: Scrum, ofrece planificación y el manejo de iteraciones que permite un desarrollo eficiente del proyecto; XP ofrece la capacidad para enfrentar cambios, además de ser la metodología ágil más específica respecto a prácticas de ingeniería [23]. SXP, permite actualizar los procesos de software para el mejoramiento de la actividad

productiva; promoviendo la creatividad, aceptación de cambios y exige un alto nivel de responsabilidad [23]. Con el uso de esta metodología se consideran las fases que se muestran en la figura 3.4, de cada fase se despliegan flujos de trabajo.

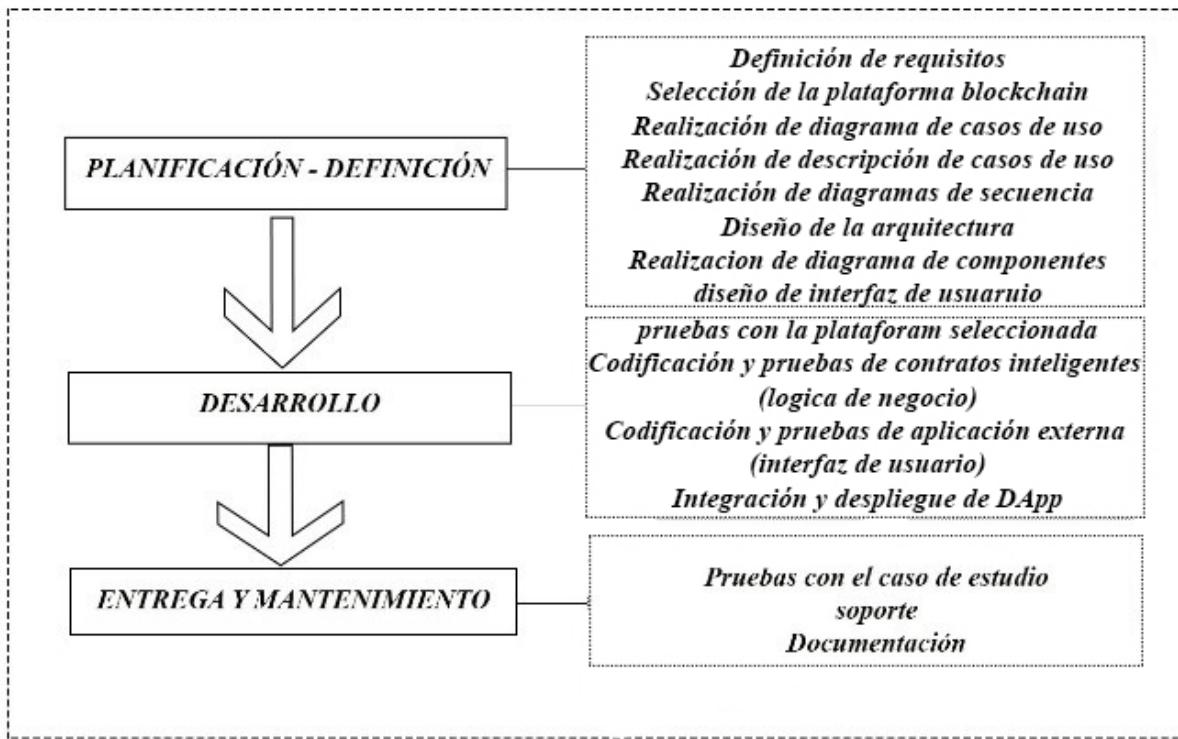


Figura 3.4: Proceso de trabajo para el desarrollo de la aplicación descentralizada.

- Planificación – Definición.

Durante esta fase se establece la visión del proyecto, elaborando la ficha técnica del proyecto, se obtienen los requisitos, se realiza el modelado y la arquitectura de la solución del proyecto.

- Desarrollo.

Se implementa el producto hasta que cumpla con todas las funcionalidades definidas y prescritas en la planificación, codificando y realizando pruebas unitarias de la lógica de negocio e interfaz de usuario. Para posteriormente integrar y desplegar la DApp.

- Entrega y mantenimientos.

Puesta en marcha del producto, durante esta etapa o fase se da soporte al sistema, con base en los requisitos definidos.

### 3.4. Aplicación de la metodología

En esta sección se describe el diseño del sistema conforme a la metodología SXP, obteniendo el modelado del sistema, representando la información que el software transforma, la arquitectura y las funciones que permiten que esto ocurra, las características que desean los usuarios y el comportamiento del sistema. El modelo cumple estos objetivos en diferentes niveles de abstracción. En el trabajo de ingeniería de software hay dos tipos de modelos [24] :

1. Modelo de requerimientos: Representa los requerimientos del cliente.
2. Modelo de diseño: representa características del software que ayudan a la construcción del software, como: arquitectura, interfaz de usuario y detalle en el nivel de componente.

El modelado incluye el análisis, el diseño y describe una representación más detallada del software. El objetivo del modelado es afirmar el entendimiento del trabajo que se va a hacer y dar una guía de la implementación.

El modelado se basa en los tipos de modelos mencionados anteriormente y en los artefactos identificados. En las siguientes secciones se da una explicación detallada de cada una de las partes que conforman la metodología.

#### 3.4.1. Aplicación de las fases de la metodología para el desarrollo de la herramienta

A continuación se describen las fases de la metodología híbrida SXP: Planificación - Definición, Desarrollo, y Entrega y Mantenimiento.

## Planificación - Definición

Para esta fase se realiza:

1. Modelado de requerimientos.
  - Modelo basado en el escenario.
    - Diagrama de casos de uso.
    - Descripción de casos de uso.
  - Modelo de comportamiento.
    - Diagrama de secuencia.
2. Modelado de diseño.
  - Arquitectura.
  - Diagrama de componentes.
  - Interfaz de usuario.

### Modelado de requerimientos

Antes de modelar se necesita la ingeniería de requerimientos, esta disciplina es importante dentro de la ingeniería de software en donde se obtienen todos los elementos de la aplicación.

Para el modelado de requerimientos, se consideran los siguientes modelos:

**Modelos basados en el escenario** Los elementos basados en el escenario ilustran cómo interactúa el usuario con el sistema y la secuencia específica de actividades que ocurren cuando se utiliza el software. Se hace uso del diagrama de casos de uso para la representación del modelo basado en el escenario.

**Diagrama de casos de uso** Alistair Cockburn [25] afirma que “un caso de uso conlleva a hacer un contrato”, describe el comportamiento del sistema en distintas condiciones en las que el sistema responde a una petición de alguno de sus participantes.

En esencia, un caso de uso narra una historia estilizada sobre cómo interactúa un usuario final con el sistema en circunstancias específicas.

Se identifican los requisitos funcionales y los actores que interactuarán con el sistema, el sistema consta de dos actores: administrador y conductores y de una serie de casos de uso, los cuales se muestran en la Figura 3.5.

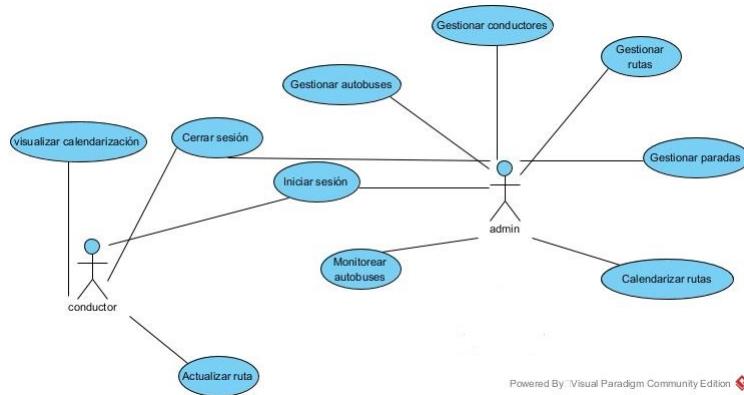


Figura 3.5: Diagramas de caso de uso

A continuación, se muestra una descripción de cada caso de uso, en el cual se describe de manera específica las actividades que realiza el actor para llevar a cabo un proceso dentro del sistema.

Caso de uso Iniciar Sesión. La tabla 3.4 muestra su especificación.

Tabla 3.4: Especificación del Caso de Uso Iniciar Sesión.

CDU-01	Iniciar sesión	
Actor	Administrador y conductores.	
Objetivo	Iniciar sesión en el sistema.	
Pre-condición	El certificado del usuario se encuentra activo.	
	Actor	Sistema
	1.- Captura los datos de inicio de sesión (usuario y contraseña) y se autentica.	

Tabla 3.4 Especificación del Caso de Uso Iniciar Sesión.

CDU-01	Iniciar sesión			
		2.- El sistema verifica y valida el certificado del usuario, si la autenticación es correcta, redirige a la pantalla inicial; si no, ver curso de excepción 1.		
Curso básico	Post-condición			
El sistema almacena en sesión los datos del administrador o conductor.				
Curso excepción 1				
Actor	Sistema			
	1.- El sistema indica que los datos ingresados son incorrectos.			
	2. Termina caso de uso.			

Caso de uso Cerrar Sesión. La tabla 3.5 muestra su especificación.

Tabla 3.5: Especificación del Caso de Uso Cerrar Sesión.

CDU-02	Cerrar sesión	
Actor	Administrador y conductores.	
Objetivo	Cerrar sesión en el sistema.	
Pre-condición	El usuario ha iniciado sesión en el sistema.	
Curso básico	Actor	Sistema
	1.- El usuario hace clic en cerrar sesión.	
		2.- El sistema redirige al usuario a la pantalla de inicio de sesión.
		3.- Termina caso de uso.

Tabla 3.5 Especificación del Caso de Uso Cerrar Sesión.

CDU-02	Cerrar sesión
Post-condición	El sistema almacena en sesión los datos del administrador o conductor.
Curso excepción 1	
Actor	Sistema
	1.- El sistema indica que los datos ingresados son incorrectos.
	2. Termina caso de uso.

Caso de uso Gestionar Conductores. La tabla 3.6 muestra su especificación.

Tabla 3.6: Especificación del Caso de Uso Gestionar Conductores.

CDU-03	Gestionar conductores	
Actor	Administrador.	
Objetivo	Registrar, eliminar y modificar los datos de un conductor, y consultar conductores.	
Pre-condición	El usuario ha iniciado sesión Ingresar al sistema como Administrador. Estar en la sección de Conductores	
	Actor	Sistema
	1.- El usuario selecciona una acción.	
		2.- Si el usuario seleccionó registrar conductor, el sistema redirige al formulario de registrar conductor, si no, ver curso alterno 1.

Tabla 3.6 Especificación del Caso de Uso Gestionar Conductores.

CDU-03	Gestionar conductores	
	<p>3.- El usuario captura los datos del conductor (Foto del conductor, nombre, apellidos, RFC (Registro Federal de Contribuyentes), CURP (Clave Única de Registro de Población) , NSS (Número de seguridad social), correo, salario, dirección, clave del INE (Instituto Nacional Electoral)</p> <p>4.- El usuario guarda el formulario.</p>	
Curso básico		<p>4.- El sistema verifica hayan sido indicados todos los campos obligatorios y el:</p> <ul style="list-style-type: none"> <li>a) Formato del RFC</li> <li>b) Formato de la CURP</li> <li>c) Formato del NSS</li> </ul> <p>En caso contrario, ver curso de excepción 1.</p> <p>4.- El sistema verifica que no exista un conductor con el mismo RFC o NSS, de lo contrario, ver curso de excepción 2.</p> <p>5.- Guarda los datos del conductor.</p>

Tabla 3.6 Especificación del Caso de Uso Gestionar Conductores.

CDU-03	Gestionar conductores	
Curso alterno 1		
	Actor	Sistema
		1.- Si el usuario seleccionó buscar conductor, el sistema redirige y muestra los conductores registrados.
2.- El usuario selecciona un conductor. 3.- El usuario selecciona una acción sobre el conductor.		
		4.- Si el usuario seleccionó modificar conductor, el sistema redirige al formulario modificar conductor, de lo contrario, ver curso alterno 2.
5.- El usuario modifica los campos necesarios. 6.- El usuario guarda los cambios		
		7.-El sistema verifica los campos modificados, si son correctos, guarda los cambios. Si no, ver curso de excepción 3.
Curso alterno 2		
	Actor	sistema
		1.- Si el usuario seleccionó eliminar, el sistema muestra un diálogo de confirmación.
2.- El usuario selecciona una acción.		

Tabla 3.6 Especificación del Caso de Uso Gestionar Conductores.

CDU-03	Gestionar conductores
	3.- Si el usuario selecciona “sí”, se elimina al conductor, de lo contrario, ver curso de excepción 4.
Curso de excepción 1	
Actor	Sistema
	<ol style="list-style-type: none"> <li>1. El sistema indica que faltan datos y/o que son incorrectos.</li> <li>2. Termina caso de uso.</li> </ol>
Curso de excepción 2	
Actor	Sistema
	<ol style="list-style-type: none"> <li>1. El sistema indica que ya existe otro conductor registrado con el mismo RFC y/o NSS.</li> <li>2. Termina caso de uso.</li> </ol>
Curso de excepción 3	
Actor	Sistema
	<ol style="list-style-type: none"> <li>1. El sistema indica que los datos son incorrectos.</li> <li>2. Termina caso de uso.</li> </ol>
Curso de excepción 4	
Actor	Sistema
	<ol style="list-style-type: none"> <li>1. El sistema redirige a la búsqueda de los conductores.</li> <li>2. Termina caso de uso.</li> </ol>

Caso de uso Gestionar Autobuses. La tabla 3.7 muestra su especificación.

Tabla 3.7: Especificación del Caso de Uso Gestionar Autobuses.

CDU-04	Gestionar autobuses	
Actor	Administrador.	
Objetivo	Registrar, eliminar y modificar los datos de un autobús, y consultar autobuses.	
Pre-condición	El usuario ha iniciado sesión Ingresar al sistema como Administrador. El usuario se encuentra en la pantalla de autobuses.	
	Actor	Sistema
	1.- El usuario selecciona una acción.	2.- Si el usuario seleccionó registrar autobuses, el sistema redirige al formulario de registrar autobuses, si no, ver curso alterno 1.
	3.- El usuario captura los datos del autobús (Foto del autobús, ID autobús, ID conductor, descripción, ciudad, zona y placa del autobús).  4.- El usuario guarda el formulario.	
		5.- El sistema verifica que hayan sido indicados todos los campos obligatorios y a) Id del conductor b) Formato de la placa

Tabla 3.7 Especificación del Caso de Uso Gestionar Autobuses.

CDU-04	Gestionar autobuses	
Curso básico	<p>En caso contrario, ver curso de excepción 1.</p> <p>6.- El sistema verifica que no exista un autobús con el mismo Id o placas, de lo contrario, ver curso de excepción 2.</p> <p>7.- Guarda los datos del autobús.</p>	
Curso alterno 1		
Actor	Sistema	
	<p>1.- Si el usuario seleccionó buscar autobús, el sistema redirige y muestra los autobuses registrados.</p>	
2.- El usuario selecciona un autobús. 3.- El usuario selecciona una acción sobre el autobús.		
	<p>4.- Si el usuario seleccionó modificar autobús, el sistema redirige al formulario modificar autobús, de lo contrario, ver curso alterno 2.</p>	
5.- El usuario modifica los campos necesarios. 6.- El usuario guarda los cambios		
	<p>7.- El sistema verifica los campos modificados, si son correctos, guarda los cambios. Si no, ver curso de excepción 3.</p>	
Curso alterno 2		

Tabla 3.7 Especificación del Caso de Uso Gestionar Autobuses.

CDU-04	Gestionar autobuses	
	Actor	Sistema
		1.- Si el usuario seleccionó eliminar autobús, el sistema muestra un diálogo de confirmación.
2.- El usuario selecciona una acción.		3.- Si el usuario selecciona “sí”, se elimina al autobús, de lo contrario, ver curso de excepción 4.
Curso de excepción 1		
	Actor	Sistema
		1.- El sistema indica que faltan datos y/o que son incorrectos.
		2.- Termina caso de uso.
Curso de excepción 2		
	Actor	Sistema
		1.- El sistema indica que ya existe otro autobús registrado con el mismo Id y/o placas.
		2.- Termina caso de uso.
Curso de excepción 3		
	Actor	Sistema
		1.- El sistema indica que los datos son incorrectos.
		2.- Termina caso de uso.
Curso de excepción 4		
	Actor	Sistema

Tabla 3.7 Especificación del Caso de Uso Gestionar Autobuses.

CDU-04	Gestionar autobuses
	1.- El sistema redirige a la búsqueda de los autobuses.
	2.- Termina caso de uso.

Caso de uso Gestionar Rutas. La tabla 3.8 muestra su especificación.

Tabla 3.8: Especificación del caso de Uso Gestionar Rutas.

CDU-05	Gestionar rutas	
Actor	Administrador.	
Objetivo	Registrar, eliminar y modificar los datos de una ruta, y consultar rutas.	
Pre-condición	El usuario ha iniciado sesión Ingresar al sistema como Administrador. El usuario se encuentra en la sección de rutas.	
	Actor	Sistema
	1.- El usuario selecciona una acción sobre las rutas.	
		2.- Si el usuario seleccionó registrar rutas, el sistema redirige al formulario de registrar rutas, si no, ver curso alterno 1.
	3.- El usuario captura los datos de la ruta (lugar de salida, lugar de llegada, nombre, clave, duración y estado).	

Tabla 3.8 Especificación del Caso de Uso Gestionar Rutas.

CDU-05	Gestionar rutas	
	4.- El usuario guarda el formulario.	
Curso básico		<p>5.- El sistema verifica que hayan sido indicados todos los campos obligatorios y que el lugar de llegada no sea el mismo que el de salida, caso contrario, ver curso de excepción 1.</p> <p>6.- El sistema verifica que no exista una ruta igual, de lo contrario, ver curso de excepción 2.</p> <p>7.- Guarda los datos de la ruta.</p>
Curso alterno 1		
Actor	Sistema	
	<p>1.- Si el usuario seleccionó buscar ruta, el sistema redirige y muestra las rutas registradas.</p>	
2.- El usuario selecciona una ruta. 3.- El usuario selecciona una acción sobre la ruta.		
	<p>4.- Si el usuario seleccionó modificar ruta, el sistema redirige al formulario de modificar la ruta, de lo contrario, ver curso alterno 2.</p>	

Tabla 3.8 Especificación del Caso de Uso Gestionar Rutas.

CDU-05	Gestionar rutas
5.- El usuario modifica los campos necesarios. 6.- El usuario guarda los cambios	7.-El sistema verifica los campos modificados, si son correctos, guarda los cambios. Si no, ver curso de excepción 3.
Curso alterno 2	
Actor	Sistema
	1.- Si el usuario seleccionó eliminar ruta, el sistema muestra un diálogo de confirmación.
2.- El usuario selecciona una acción.	3.- Si el usuario selecciona “sí”, se elimina la ruta, de lo contrario, ver curso de excepción 4.
Curso de excepción 1	
Actor	Sistema
	1.- El sistema indica que faltan datos y/o que son incorrectos. 2.- Termina caso de uso.
Curso de excepción 2	
Actor	Sistema
	1.- El sistema indica que ya existe otra ruta registrada con el mismo destino y origen. 2.- Termina caso de uso.

Tabla 3.8 Especificación del Caso de Uso Gestionar Rutas.

CDU-05	Gestionar rutas	
Curso de excepción 3		
Actor	Sistema	
	1.- El sistema indica que los datos son incorrectos. 2.- Termina caso de uso.	
Curso de excepción 4		
Actor	Sistema	
	1.- El sistema redirige a la búsqueda de las rutas. 2.- Termina caso de uso.	

Caso de uso Gestionar Paradas. La tabla 3.9 muestra su especificación.

Tabla 3.9: Especificación del Caso de Uso Gestionar Paradas.

CDU-06	Gestionar paradas	
Actor	Administrador.	
Objetivo	Registrar, eliminar y modificar los datos de una parada, y consultar paradas.	
Pre-condición	El usuario ha iniciado sesión Ingresar al sistema como Administrador. El usuario deberá estar en la sección de paradas.	
	Actor	Sistema
	1.- El usuario selecciona una acción.	

Tabla 3.9 Especificación del Caso de Uso Gestionar Paradas.

CDU-06	Gestionar paradas	
Curso básico		2.- Si el usuario seleccionó registrar parada, el sistema redirige al formulario de registrar parada, si no, ver curso alterno 1.
	3.- El usuario selecciona en el mapa el punto donde se situará la parada.	
		4.- El sistema manda la latitud y longitud del punto seleccionado. 5.- Muestra el formulario.
	6.- El usuario ingresa el nombre de la parada y un color que identifique la línea de autobuses.	
	7.- Guarda el formulario.	
		8.- El sistema verifica que todos los campos obligatorios hayan sido indicados, si no, ver curso de excepción 1. 9.- El sistema verifica que no exista una parada con el mismo nombre, caso contrario ver curso de excepción 2.
	Curso alterno 1	
	Actor	sistema

Tabla 3.9 Especificación del Caso de Uso Gestionar Paradas.

CDU-06	Gestionar paradas
	1.- Si el usuario seleccionó buscar paradas, el sistema redirige y muestra las paradas registradas.
2.- El usuario selecciona una parada. 3.- El usuario selecciona una acción sobre la parada.	
	4.- Si el usuario seleccionó modificar parada, el sistema redirige al formulario de modificar la parada, de lo contrario, ver curso alterno 2.
5.- El usuario modifica los campos necesarios. 6.- El usuario guarda los cambios	
	7.-El sistema verifica los campos modificados, si son correctos, guarda los cambios. Si no, ver curso de excepción 3.
Curso alterno 2	
Actor	sistema
	1.- Si el usuario seleccionó eliminar parada, el sistema muestra un diálogo de confirmación.
2.- El usuario selecciona una acción.	
	3.- Si el usuario selecciona “sí”, se elimina la parada, de lo contrario, ver curso de excepción 4.
Curso de excepción 1	

Tabla 3.9 Especificación del Caso de Uso Gestionar Paradas.

CDU-06	Gestionar paradas	
	Actor	Sistema
	1.- El sistema indica que faltan datos.	
	2.- Termina caso de uso.	
	Curso de excepción 2	
	Actor	Sistema
	1.- El sistema indica que ya existe otra parada registrada con el mismo nombre.	
	2.- Termina caso de uso.	
	Curso de excepción 3	
	Actor	Sistema
	1.-El sistema indica que los datos son incorrectos.	
	2.- Termina caso de uso.	
	Curso de excepción 4	
	Actor	Sistema
	1.- El sistema redirige a la búsqueda de las paradas.	
	4.- Termina caso de uso.	

Caso de uso Monitorear Autobús. La tabla 3.10 muestra su especificación.

Tabla 3.10: Especificación del Caso de Uso Monitorear Autobús.

CDU-07	Monitorear autobús
Actor	Administrador.
Objetivo	Monitorear la ubicación del autobús en tiempo real y registrar la ubicación constante dentro de la red <i>blockchain</i> .
	El usuario ha iniciado sesión.

Tabla 3.10 Especificación del Caso de Uso Monitorear Autobús.

CDU-07	Monitorear autobús	
Pre-condición	Ingresar al sistema como administrador. El usuario se encuentra en la pantalla de monitoreo.	
	Actor	Sistema
	1.- El usuario selecciona un autobús.	
		2.- El sistema muestra los autobuses en ruta.
	3.- El usuario selecciona un autobús a monitorear.	
		4.- El sistema muestra mediante un mapa la ubicación actual del autobús.
Curso básico	4.- El usuario selecciona guardar ubicación automáticamente del autobús.	
		5.- Se guarda la ubicación del autobús en la <i>blockchain</i> y en la base de datos.
Post-condiciones	El informe se mostrará en una tabla (hora y ubicación).	

Caso de uso Actualizar Ruta. La tabla 3.11 muestra su especificación.

Tabla 3.11: Especificación del Caso de Uso Actualizar Ruta.

CDU-08	Actualizar ruta
Actor	Conductor.
Objetivo	Actualizar su ruta al inicio de su jornada, para obtener un control de sus rutas y de su jornada laboral.
	El usuario ha iniciado sesión

Tabla 3.11 Especificación del Caso de Uso Actualizar Ruta.

CDU-08	Actualizar ruta	
Pre-condición	Ingresar al sistema como empleado (conductor). El usuario deberá estar en la pantalla de actualizar ruta.	
Curso básico	Actor	Sistema
	1.- El usuario selecciona ingresar ruta.	
		2.- El sistema muestra el formulario.
	3.- El usuario ingresa los datos en el formulario (id ruta, hora de salida) y activar monitoreo.	
	4.- El usuario guarda el formulario.	
		4.- El sistema guarda la información.  5.- El sistema guarda de forma constante la ubicación del conductor en la base de datos y la <i>blockchain</i> .

Caso de uso Calendarizar Rutas. La tabla 3.12 muestra su especificación.

Tabla 3.12: Especificación del Caso de Uso Calendarizar Rutas.

CDU-09	Calendarizar rutas
Actor	Administrador.
Objetivo	Calendarizar las rutas semanales de los conductores.
Pre-condición	El usuario ha iniciado sesión. Ingresar al sistema como administrador. El usuario deberá estar en la pantalla de calendarizar rutas.

Tabla 3.12 Especificación del Caso de Uso Calendarizar Rutas.

CDU-09	Calendarizar rutas	
	Actor	Sistema
Curso básico	1.- El usuario selecciona un conductor.	
		2.- El sistema muestra un formulario (día, ruta, hora entrada y hora de salida).
	3.- El usuario completará el formulario, seleccionando un día, la ruta, hora de entrada y hora de salida.	3.- El sistema mostrará los días disponibles y laborables del conductor, si un día ya fue registrado, no se mostrará.
	4.- El usuario guardará el formulario.	5.- El sistema guarda la información. 6.- Termina caso de uso.

Caso de uso Visualizar Calendarización. La tabla 3.13 muestra su especificación.

Tabla 3.13: Especificación del Caso de Uso Visualizar Calendarización.

CDU-10	Visualizar calendarización
Actor	Conductor.
Objetivo	Visualizar la calendarización asignada por el administrador.
Pre-condición	El usuario ha iniciado sesión. Ingresar al sistema como empleado (conductor).

Table 3.13 Especificación del Caso de Uso Visualizar Calendarización.

CDU-10	Visualizar calendarización	
	Actor	Sistema
Curso básico	1.- El usuario selecciona visualizar calendarización.	
		2.- El sistema muestra en forma de tabla su calendarización (hora entrada, hora salida, ruta y días laborables).
	3.- El usuario visualiza su calendarización.	
	4.- Termina caso de uso	

### Modelo de comportamiento

Los elementos del comportamiento ilustran la forma en la que los eventos externos cambian el estado del sistema o las clases que residen dentro de éste.

### Diagrama de secuencia

Indica la forma en la que los eventos provocan transiciones de un objeto a otro. Una vez identificados los objetos por medio del análisis del sistema, se crea el diagrama de secuencia representando el modo en el que los eventos causan el flujo de uno a otro como función del tiempo. A continuación se muestran los diagramas de secuencia del sistema.

La figura 3.6 muestra el diagrama de secuencia del Caso de Uso Iniciar Sesión.

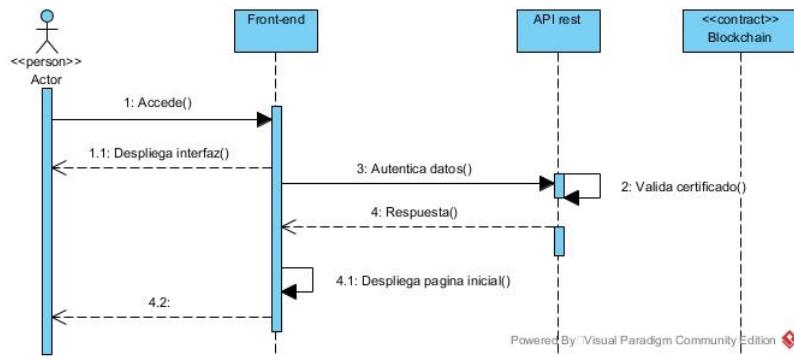
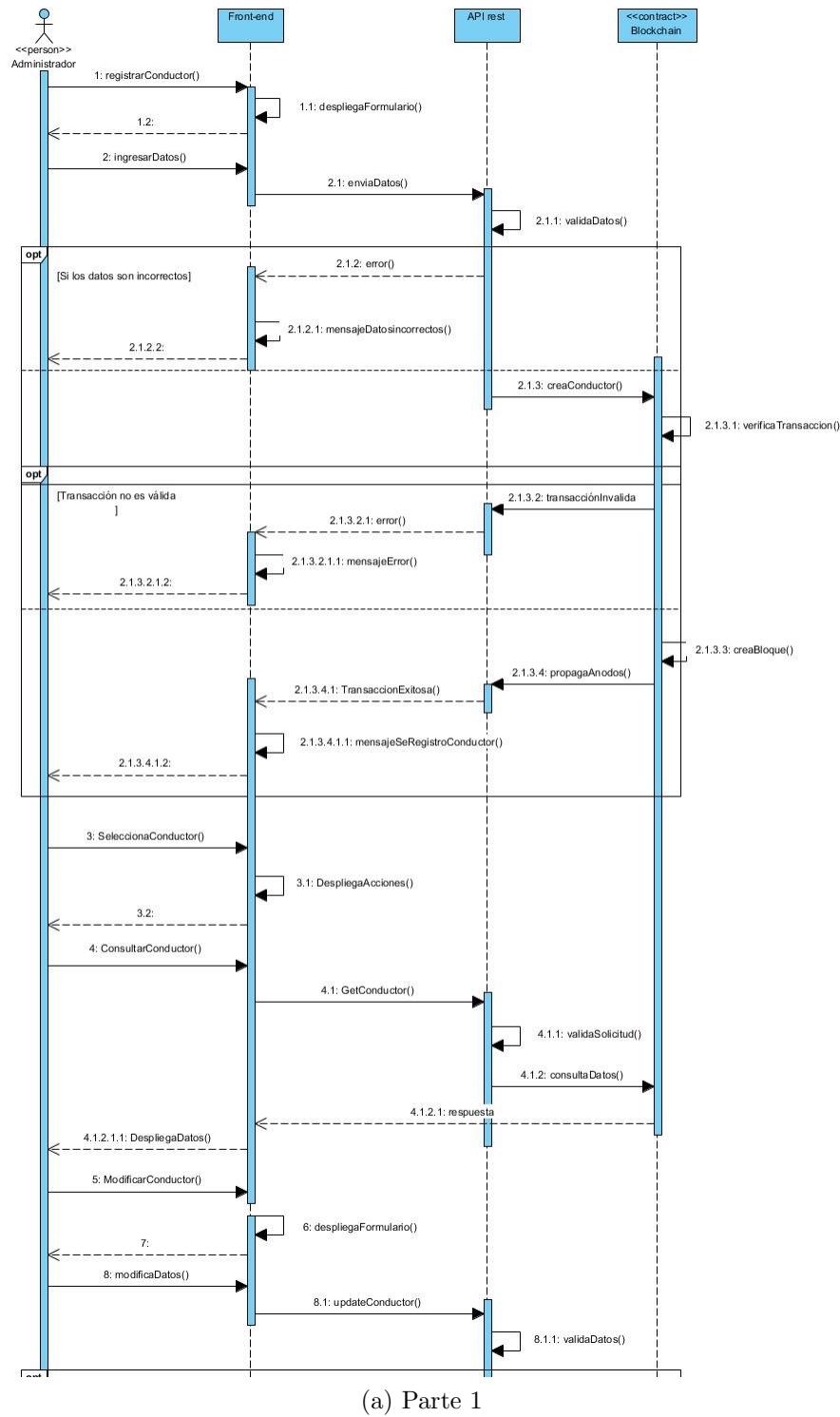


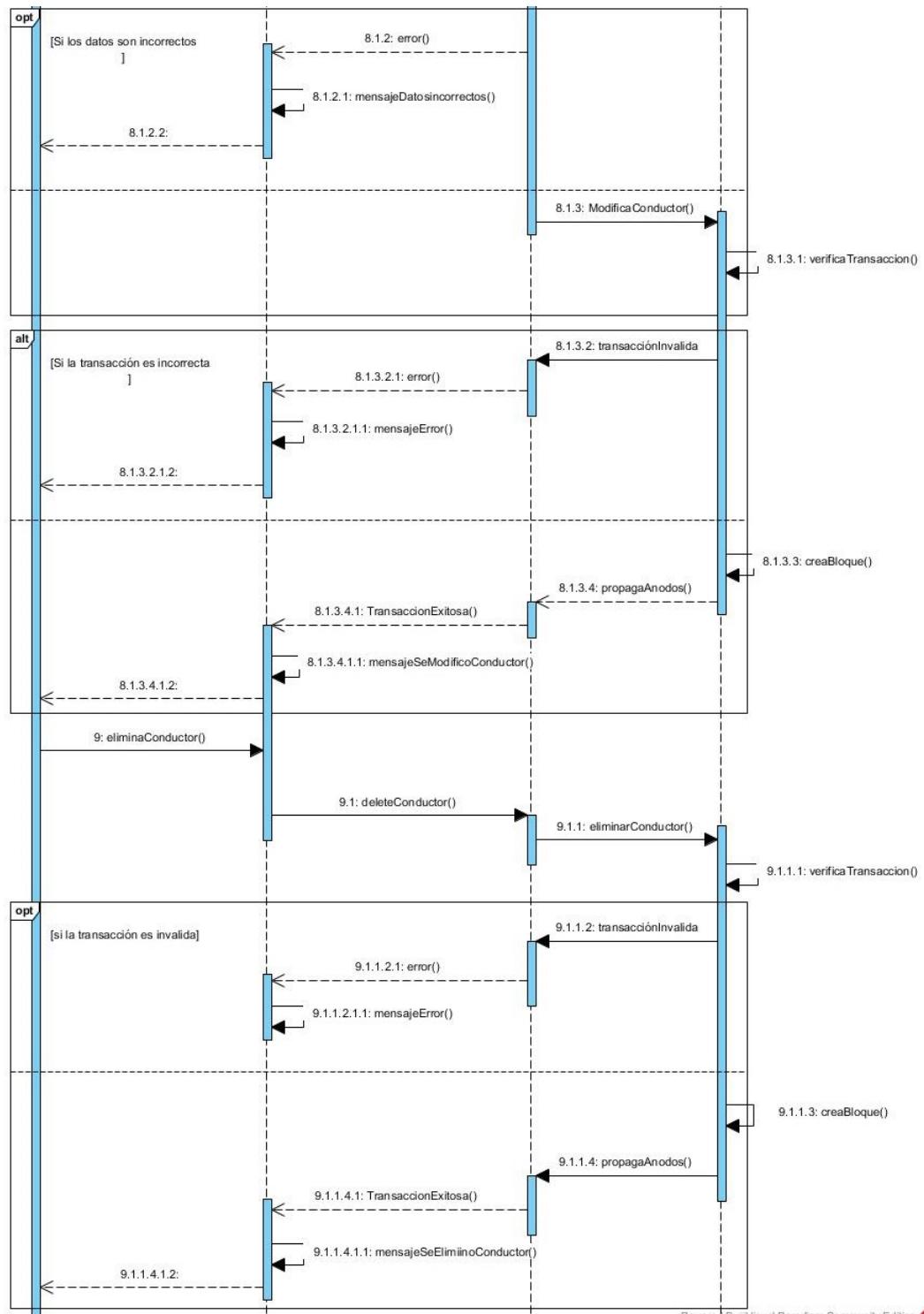
Figura 3.6: Diagrama de secuencia (CDU-01)

La figura 3.7 muestra el diagrama de secuencia del Caso de Uso Gestionar Conductores.



(a) Parte 1

Figura 3.7: Diagrama de secuencia (CDU-03)



(b) Parte 2

Figura 3.7: Diagrama de secuencia (CDU-03)

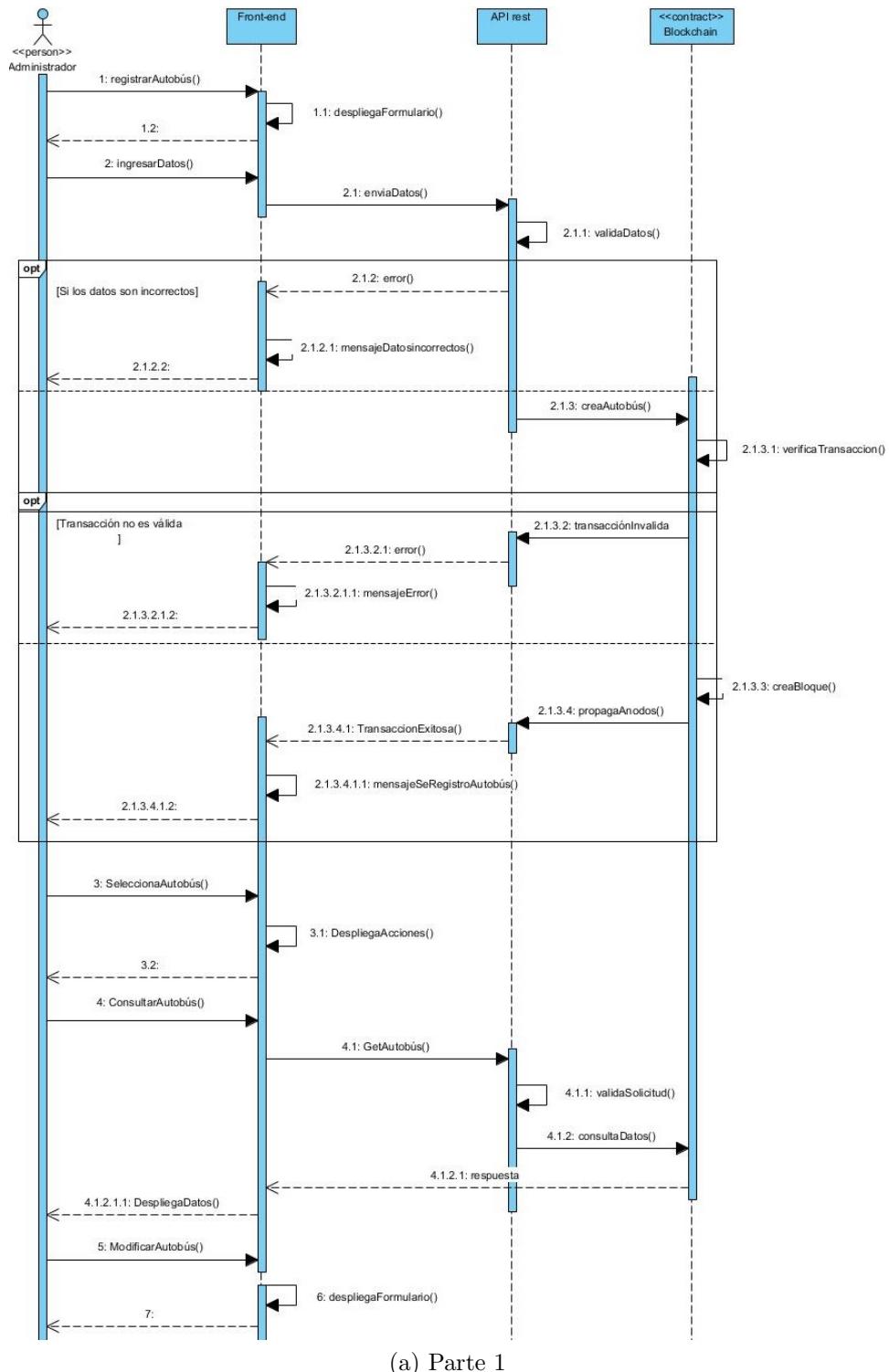
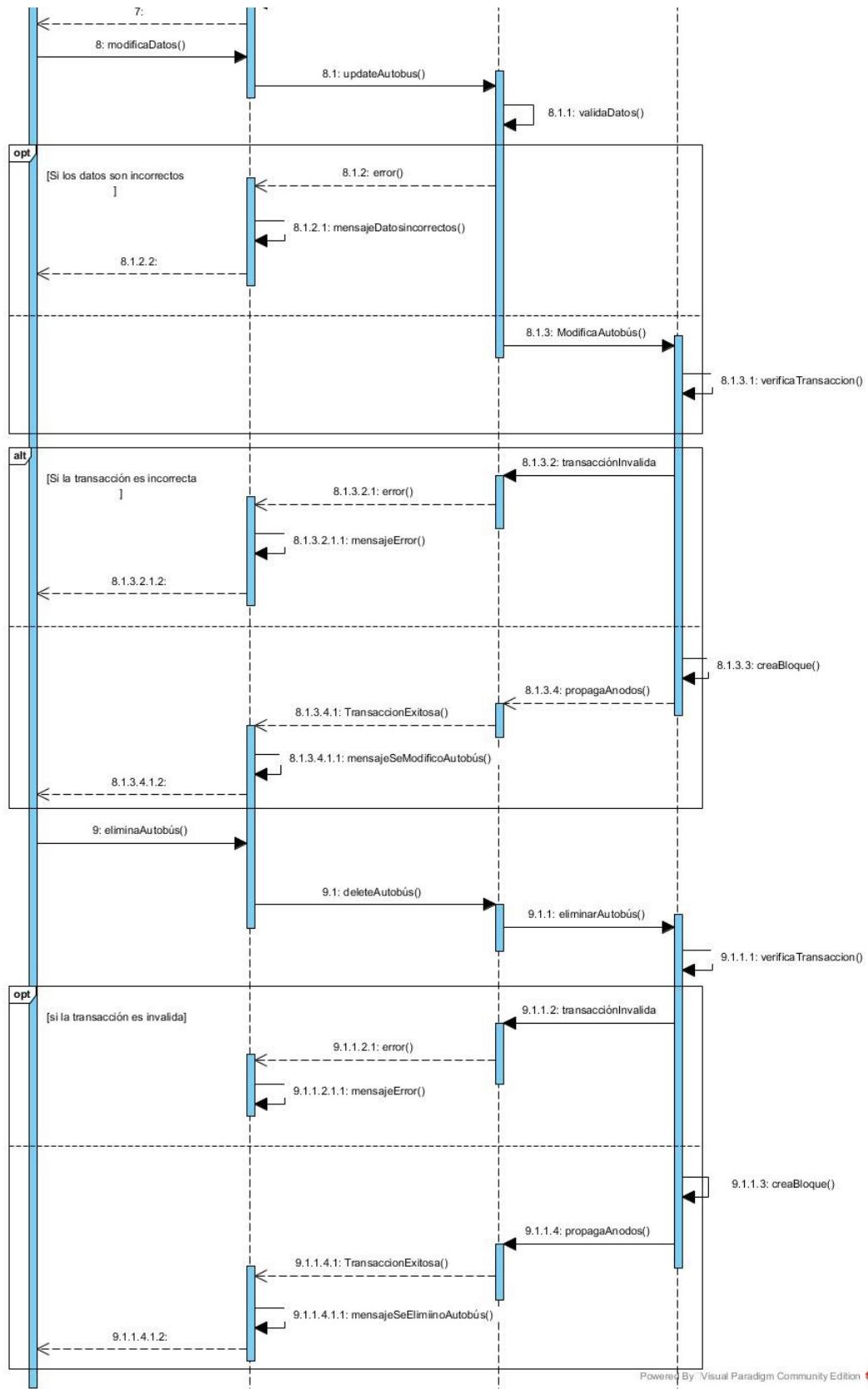


Figura 3.8: Diagrama de secuencia (CDU-04)

La figura 3.8 muestra el diagrama de secuencia del Caso de Uso Gestionar Autobuses.



(b) Parte 2

Figura 3.8: Diagrama de secuencia (CDU-04)

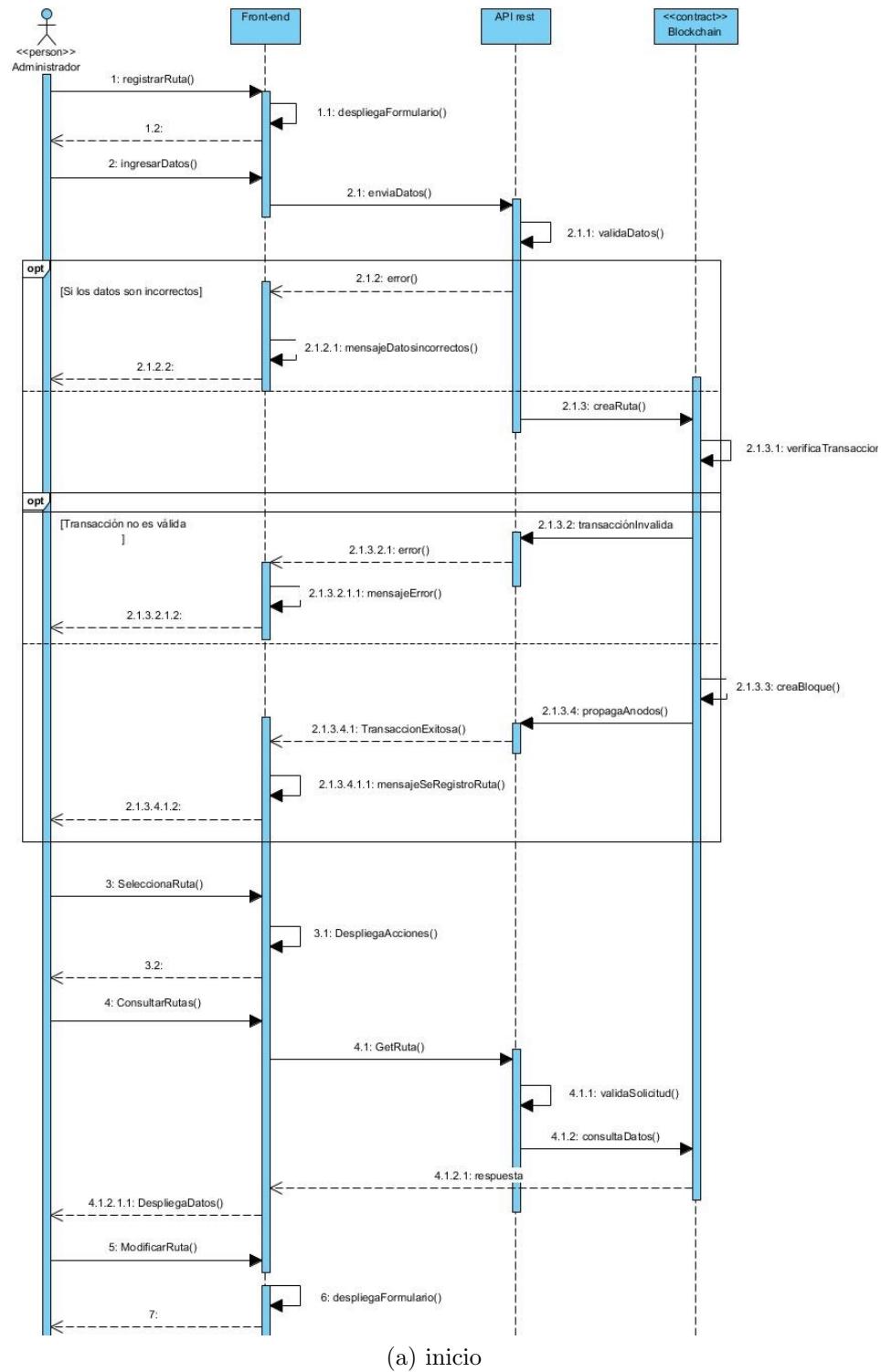
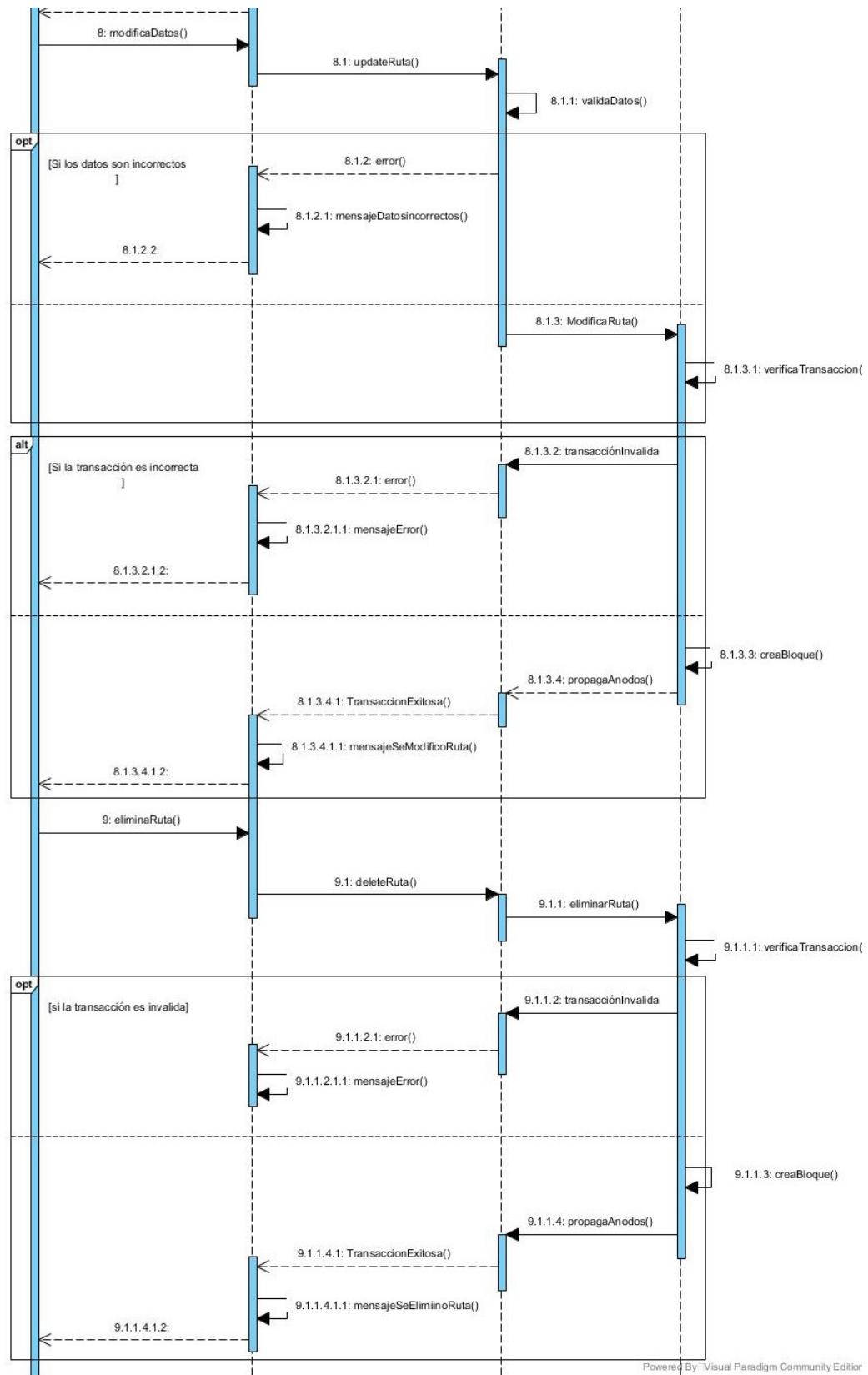


Figura 3.9: Diagrama de secuencia (CDU-05)

La figura 3.9 muestra el diagrama de secuencia del Caso de Uso Gestionar Rutas.



(b) continuación

Figura 3.9: Diagrama de secuencia (CDU-05)

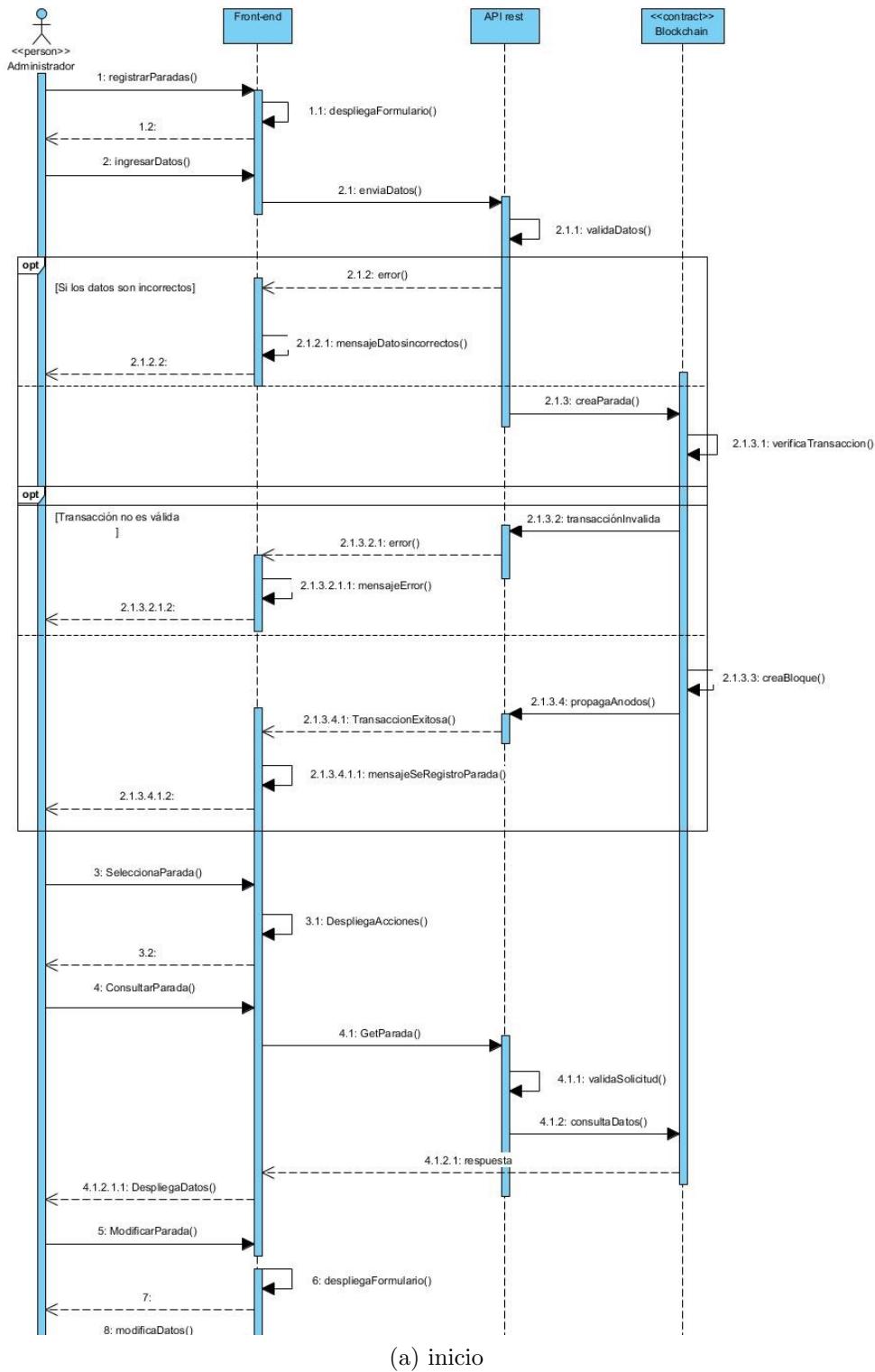
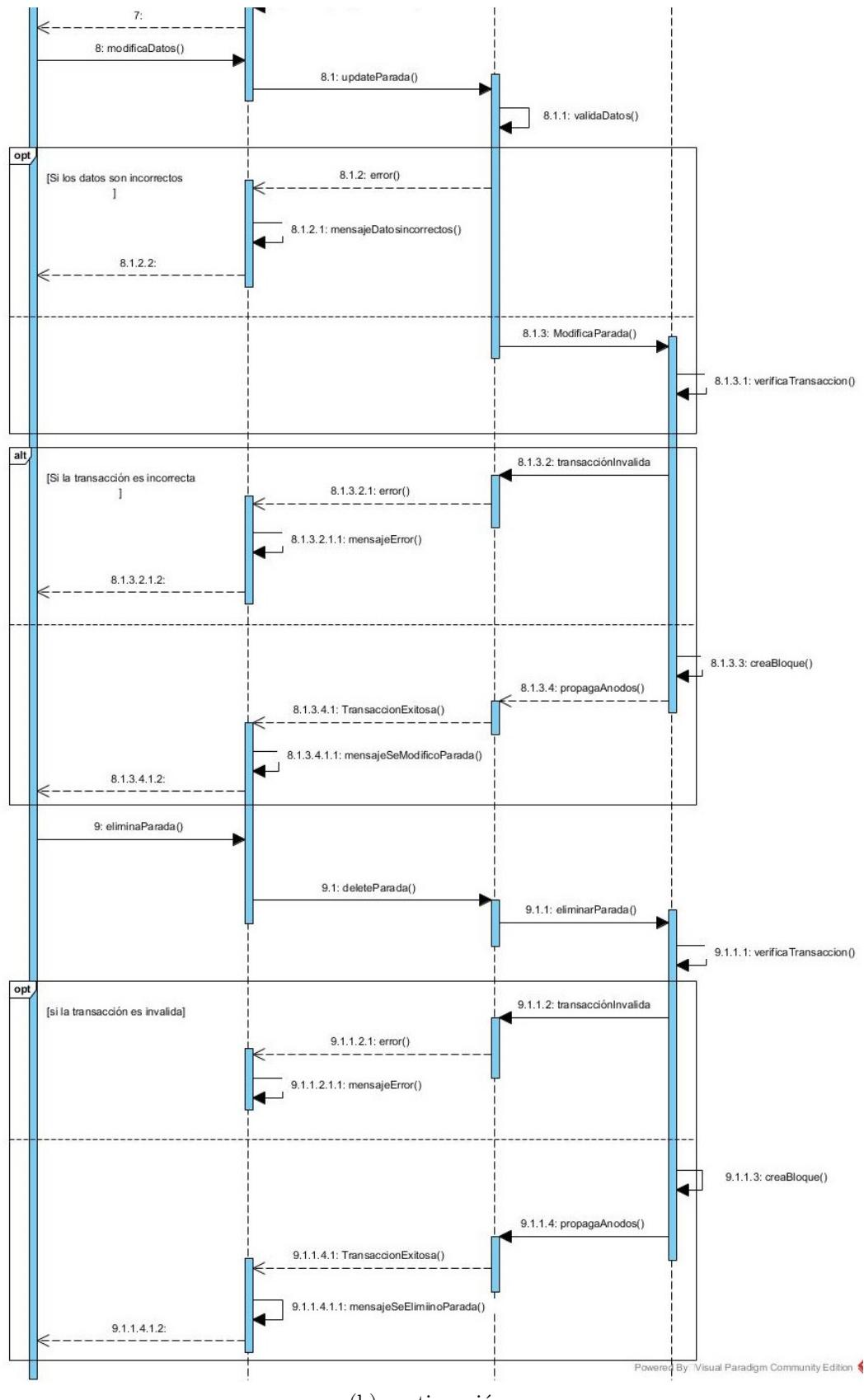


Figura 3.10: Diagrama de secuencia (CDU-06)

La figura 3.10 muestra el diagrama de secuencia del Caso de Uso Gestionar Paradas.



(b) continuación

Figura 3.10: Diagrama de secuencia (CDU-06)

La figura 3.11 muestra el diagrama de secuencia del Caso de Uso Monitorear Autobuses.

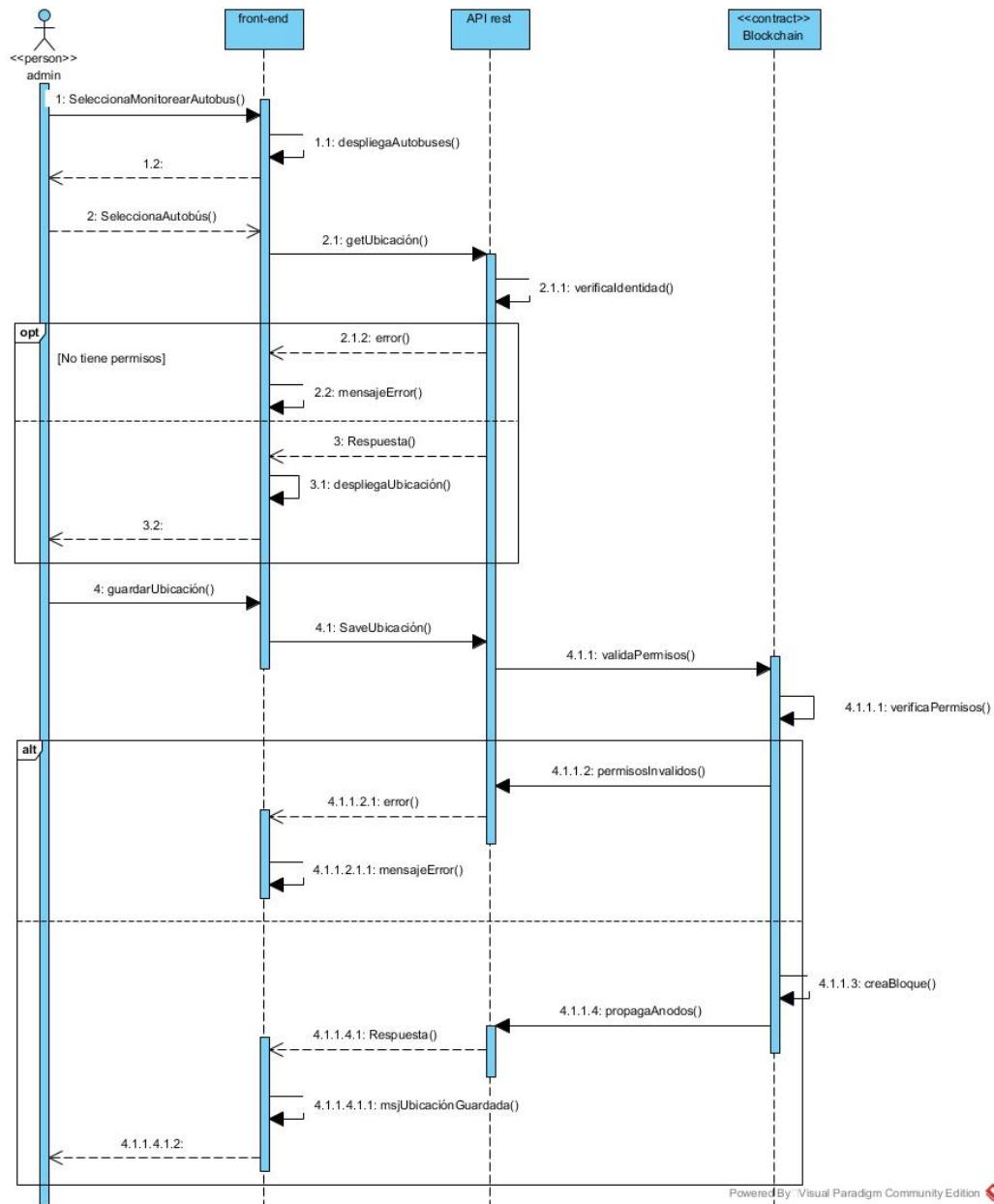


Figura 3.11: Diagrama de secuencia (CDU-07)

La figura 3.12 muestra el diagrama de secuencia del Caso de Uso Actualizar Ruta de Conductor.

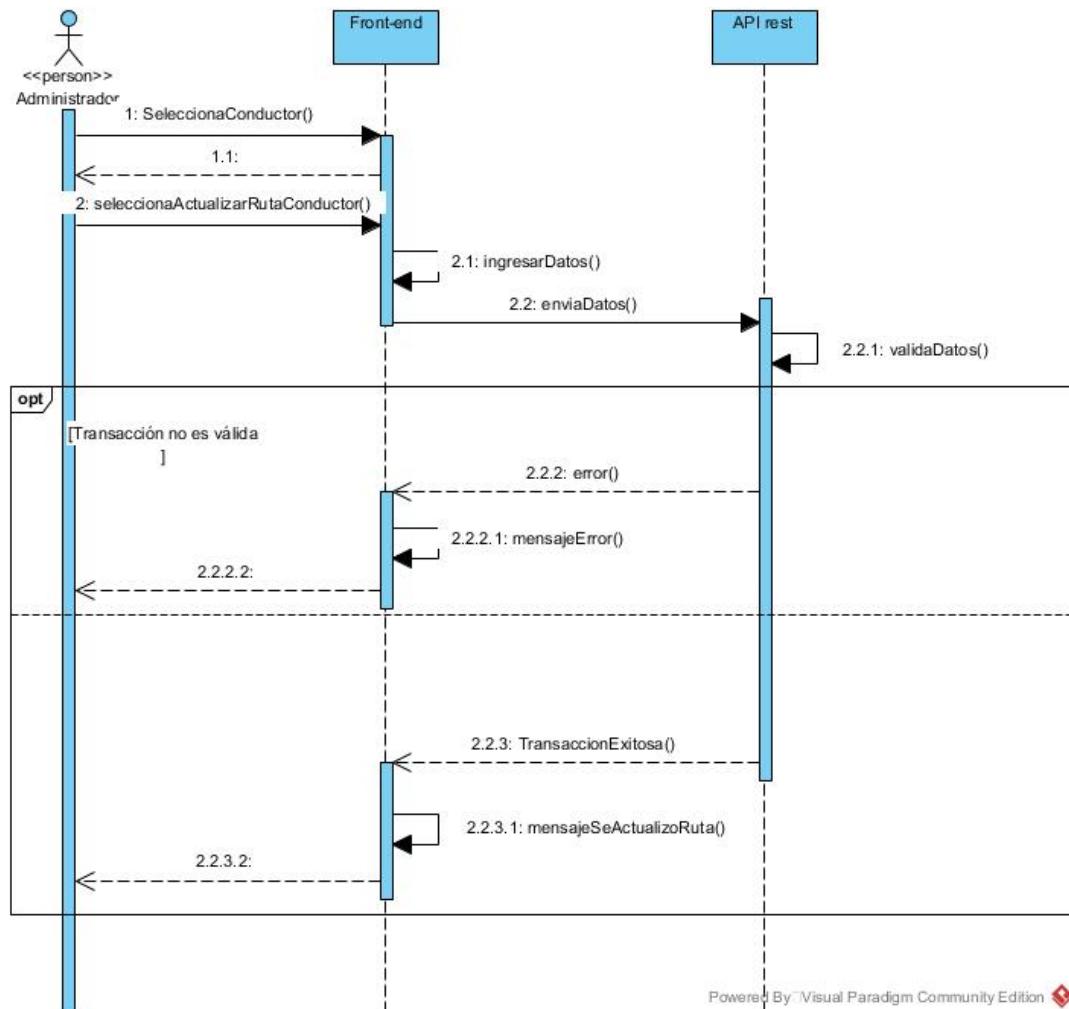


Figura 3.12: Diagrama de secuencia (CDU-08)

La figura 3.13 muestra el diagrama de secuencia del Caso de Uso Calendarizar Rutas.

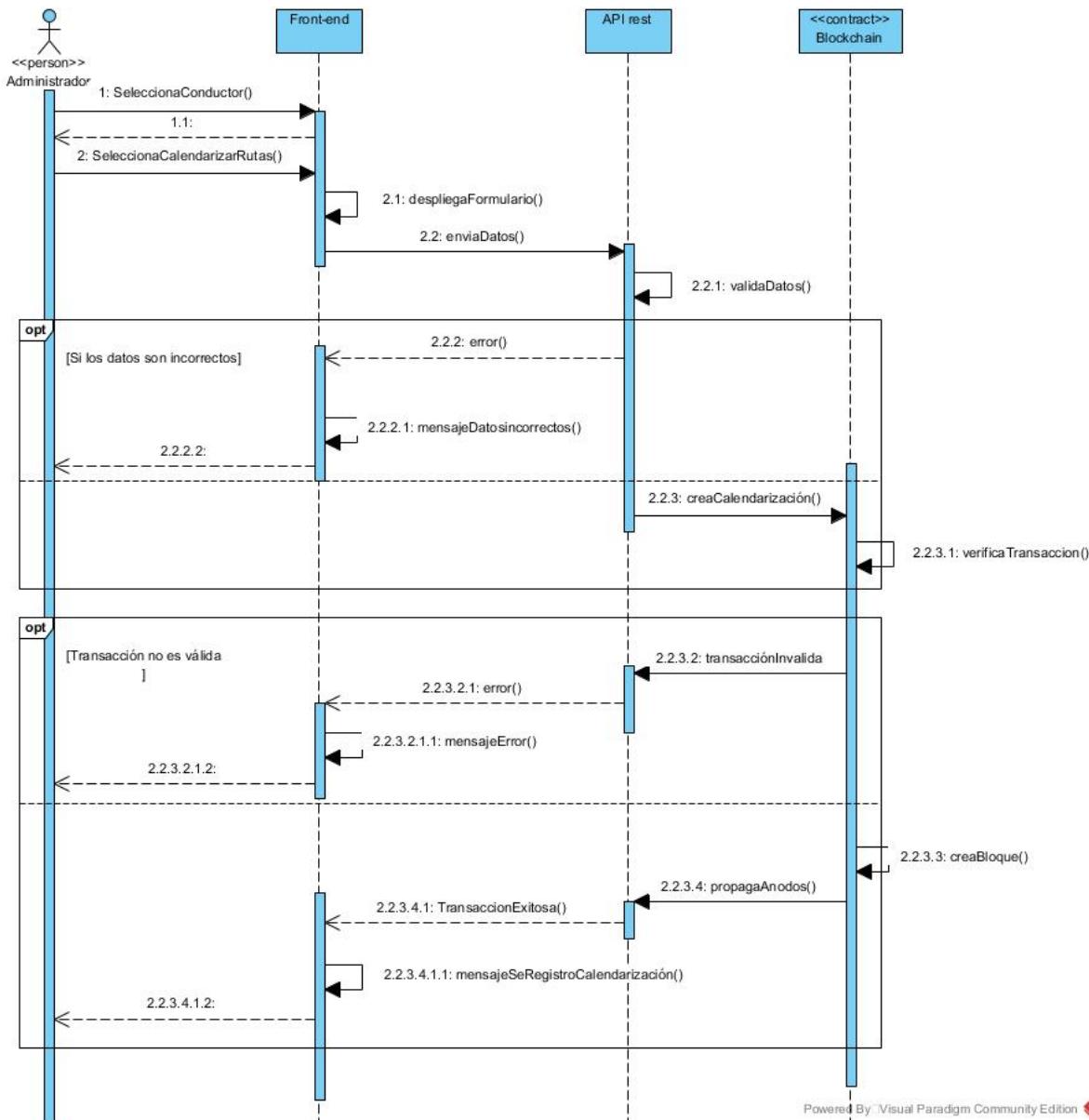


Figura 3.13: Diagrama de secuencia (CDU-09)

La figura 3.14 muestra el diagrama de secuencia del Caso de Uso Visualizar Calendarización.

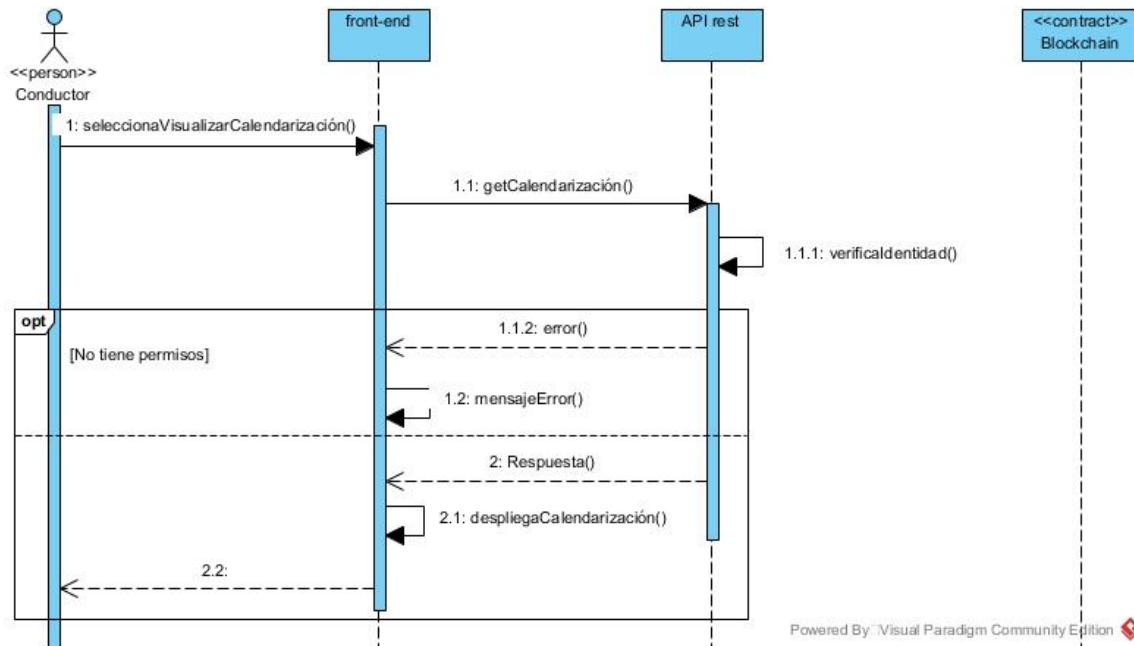


Figura 3.14: Diagrama de secuencia (CDU-10)

La figura 3.15 muestra el diagrama de secuencia del Caso de Uso Cerrar Sesión.

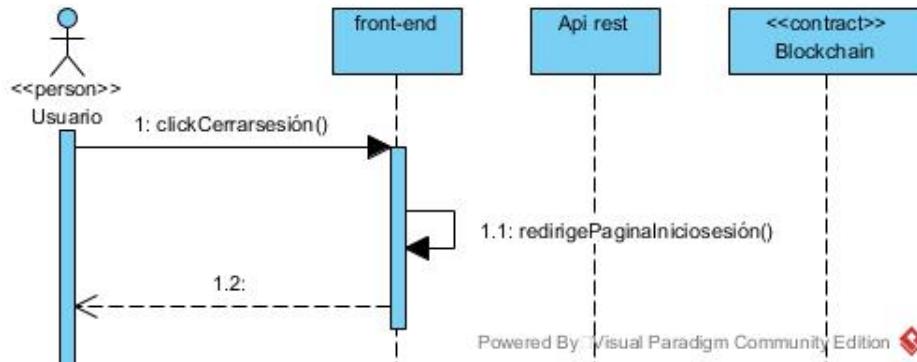


Figura 3.15: Diagrama de secuencia (CDU-02)

## Modelado de diseño

El modelado de diseño proporciona detalles sobre arquitectura del software, interfaces de usuario y componentes que se necesitan para implementar el sistema.

## Arquitectura

Pressman [24], menciona que un diagrama de arquitectura ayuda a plantear una vista completa del sistema que se construirá. Muestra la estructura y organización de los componentes de software. Para el diseño de la arquitectura de la DApp se hace uso del patrón arquitectónico en capas, el cual está constituido por cinco capas: capa de aplicación, capa de contratos inteligentes (lógica de negocio), capa de libro mayor distribuido, capa de base de datos y capa de seguridad, representadas en la figura 3.16.

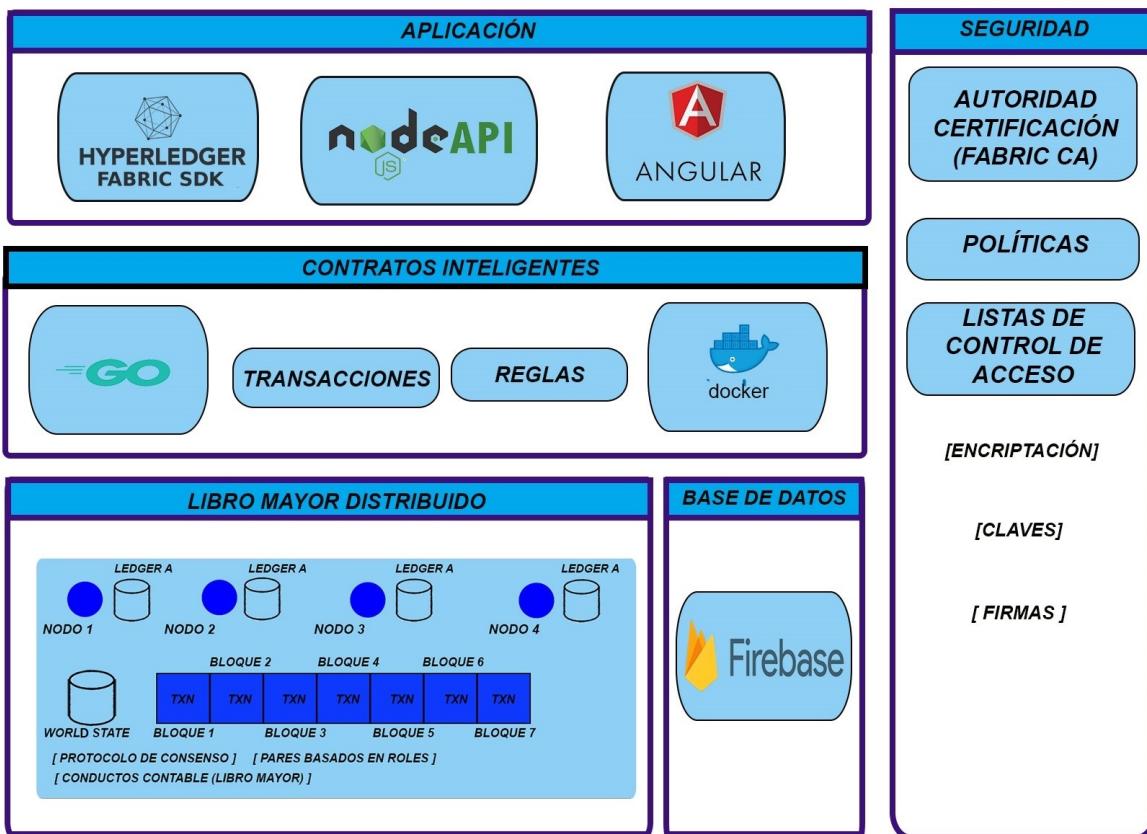


Figura 3.16: Arquitectura del sistema.

**Capa de aplicación** La capa de aplicación hace referencia a la interacción entre el usuario y el software.

La interfaz de usuario del lado del cliente se implementa utilizando AngularJS y la API en Node.js.

Angular: componente que corresponde con el *front-end* del sistema. A través de este

componente los clientes realizan el acceso al sistema, el *front-end* consultará datos a la API y a su vez permitirá hacer peticiones *POST* para enviar transacciones a la cadena de bloques.

El SDK de Fabric: ayuda a establecer la comunicación entre el *front-end* y el *back-end*, mediante el SDK se ejecuta el *chaincode* del usuario y se realizan transacciones en la red. Para la ejecución de una transacción, el cliente se conecta a la red de *Hyperledger Fabric* a través del SDK, el cliente crea una transacción y la envía al par de aprobación. El SDK de *Hyperledger Fabric* se usará en los servicios *REST* (*Representational State Transfer*; Transferencia de Estado Representacional) desarrollados para interactuar con la red *blockchain*.

API *REST*: provee de una fachada de servicios *REST* que interactuarán con la *blockchain Hyperledger* a través de la ejecución de los *chaincode*. La API *REST* utiliza HTTP para peticiones de datos *get*, *put*, *post* y *delete*. Un escenario de petición HTTP funciona así:

1. Un cliente envía una petición HTTP a un servidor.
2. El servidor devuelve una respuesta HTTP.

**Contrato inteligente** Un contrato inteligente en *Hyperledger Fabric* se llama *chaincode*. Es un programa, codificado en *Go*, para este caso. *Chaincode* se ejecuta en un contenedor *Docker* aislado. Esto brinda la capacidad de utilizar la API existente y facilitar la migración. El propósito de *chaincode* es ser la capa empresarial en el desarrollo de software. Al implementar un nuevo *chaincode*, es posible asignar una política, de la cual los *peers* deben firmar las transacciones que ejecutan este *chaincode*. *Chaincode* se puede implementar mediante la API *fabric-contract-api*.

**Libro mayor** El libro mayor consta de dos componentes:

- Registro de transacciones.
- *World state* (estado mundial)

Las transacciones cambian el estado mediante el uso de *chaincode*. La implementación de un nuevo *chaincode* se considera una transacción. El *chaincode* se firma y el sistema crea un paquete inmutable. Se crea una imagen de *Docker* separada y se ejecuta como una máquina separada.

*Hyperledger Fabric* depende de certificados, por lo que se proporciona un servicio separado llamado *Fabric CA* (*Certificate Authority*, Autoridad certificada) para generar dinámicamente certificados para los usuarios.

**Seguridad** Una de las partes fundamentales que proporciona la tecnología *blockchain* es la seguridad. Para esto la aplicación necesita también lo que se denomina tarjetas o *business network cards*, que contienen las credenciales, las claves, los certificados y un perfil de conexión, necesario para conectar a las aplicaciones y componentes de la infraestructura de *Hyperledger*. Los usuarios dispondrán de un certificado de clave pública que será empleado por la *blockchain* para validar las transacciones, el certificado de los usuarios se incluirá de modo que se realizará la validación del certificado cliente en el servidor.

Componente *Fabric CA*: asigna certificados, donde el usuario hace una petición para obtener el certificado, la autoridad de registro valida la identidad del participante y manda la petición a la autoridad de certificación, que asigna el certificado; la autoridad de certificación remite el certificado al participante que lo había solicitado. Una vez que el participante inicia una transacción, un componente de la infraestructura de *Hyperledger Fabric* valida la autenticidad de la transacción comprobando la validez del certificado con la autoridad de validación.

*Hyperledger Fabric* también dispone de la opción de configurar ACL (*Access Control List*, Lista de Control de Acceso), permite controlar el acceso a los recursos de la *blockchain* mediante la asignación de políticas a las diferentes identidades. Estas políticas se asocian a nivel de recurso, donde un recurso es un *chaincode* de usuario, un *chaincode* de sistema o un *event resource*.

**Estructura de Base de Datos** Para obtener escalabilidad, integridad y confiabilidad de los datos, se hace uso de una base de datos No SQL (*Structured Query Language*, lenguaje de consulta estructurada). Esta base de datos aporta mayor flexibilidad, velocidad y manejabilidad, características necesarias para la creación del *blockchain*. El motor de base de datos seleccionado es *Firebase*.

### Diagrama de componentes

El diagrama de componentes mostrado en la figura 3.17, proporciona una vista más simple y entendible de cómo se comunican los componentes del sistema, la red de *Hyperledger Fabric* (*blockchain*) funciona como *back-end* con una aplicación *front-end* para comunicarse con la red. El SDK de *Hyperledger Fabric* contiene una API (NodeJS) que ayuda a establecer la comunicación entre el *front-end* (Angular) y el *back-end*, el SDK proporciona una forma de ejecutar el *chaincode* del usuario, de realizar transacciones en la red, de supervisar eventos. Con la SDK API, el cliente es capaz de crear una transacción y la enviarla al par de aprobación.

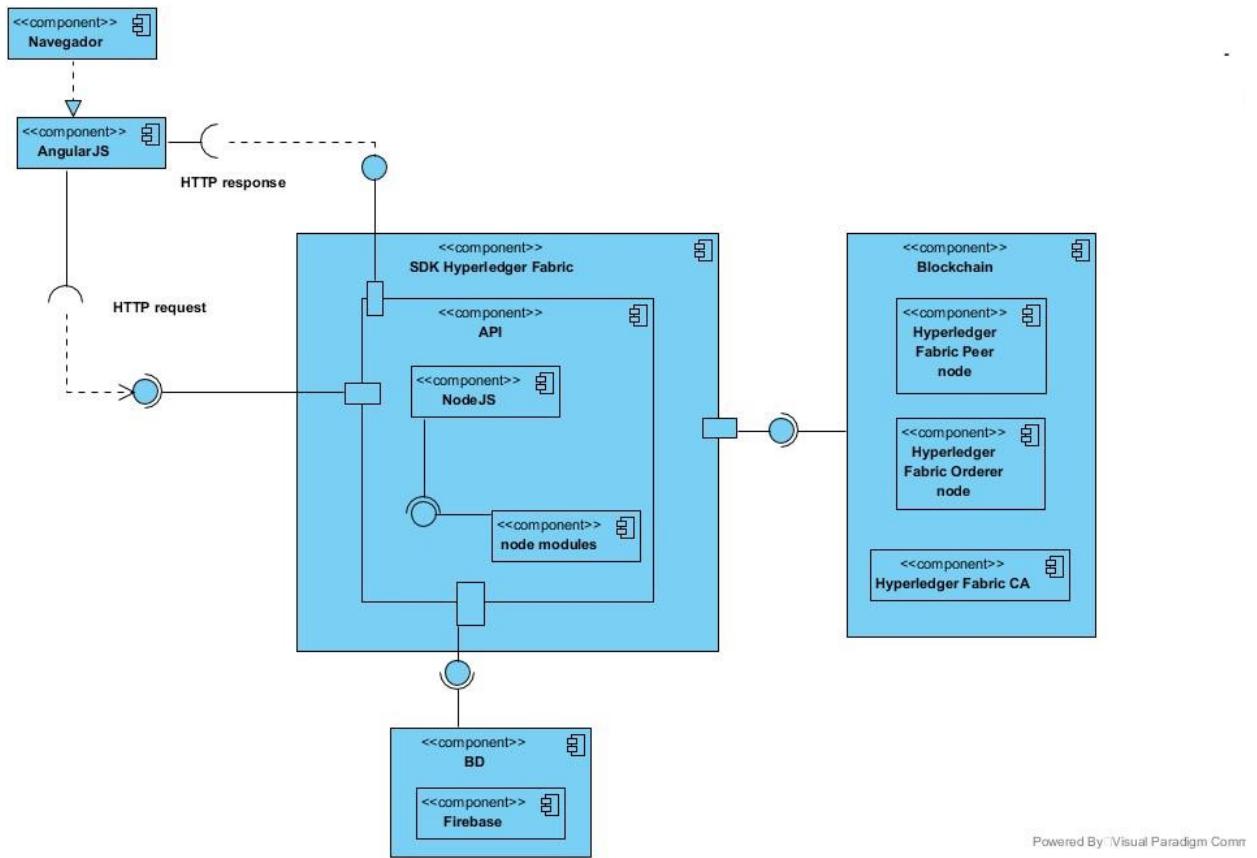


Figura 3.17: Diagrama de componentes.

## Interfaz de usuario

La interfaz de usuario es presumiblemente el elemento más importante de un sistema o producto basado en computadora. Si la interfaz estuviera mal diseñada, afectaría mucho la capacidad del usuario de aprovechar el contenido de información de una aplicación. En resumen, una interfaz defectuosa haría que fallara incluso una aplicación bien diseñada y con buena implementación.

Hay tres principios importantes que guían el diseño de interfaces eficaces:

1. Dar el control al usuario.
2. Reducir la memorización del usuario.
3. Hacer que la interfaz sea consistente.

Para lograr que una interfaz cumpla estos principios, debe llevarse a cabo un proceso de diseño bien organizado.

La interfaz de usuario se compone del diseño estructural y de comportamiento. Cada interfaz o vista presentada a continuación provee una idea preliminar de cómo se representará la información del modelo del dominio de la aplicación proporcionando una percepción que sirve como modelo a seguir cuando se llegue a la fase de desarrollo. Con base en el diagrama de casos de uso, se identifican las siguientes tareas de los actores, las cuales se describen a continuación.

**Interfaz de registro** En la Figura 3.18 se muestra la interfaz de registro, representa la vista para realizar el registro de los usuarios. Cuenta con el correo, una contraseña y un botón para iniciar o ingresar.

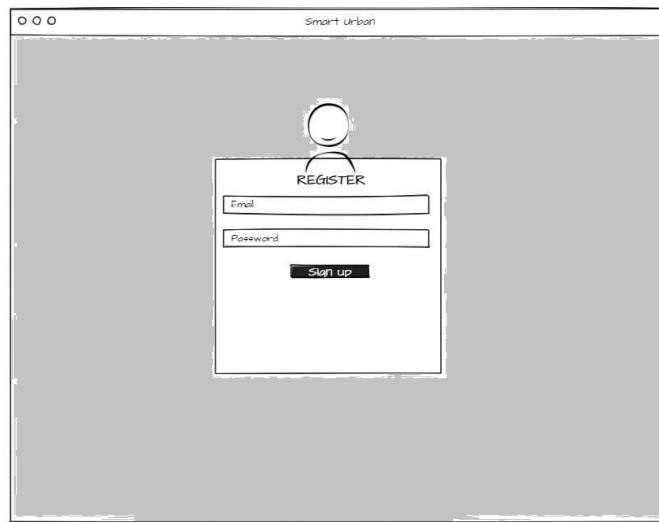


Figura 3.18: Interfaz de registro.

**Interfaz iniciar sesión** En la Figura 3.19 se muestra la interfaz de iniciar sesión, representa la vista para iniciar sesión como administrador o conductor dentro del sistema. Cuenta con los campos correo electrónico y contraseña, y un botón para acceder al sistema.

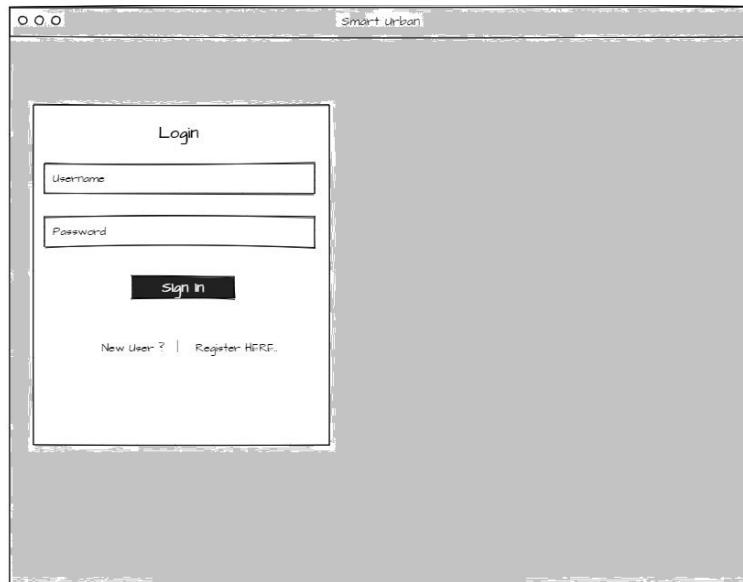


Figura 3.19: Interfaz iniciar sesión.

**Interfaz de autobuses.** En la figura 3.20 se muestra la interfaz de autobuses, representa la vista para realizar acciones como: agregar, eliminar, modificar y visualizar conductores. Si no se tienen autobuses agregados solo muestra el botón agregar autobús, si ya se tienen autobuses dados de alta, se muestra una lista de todos los autobuses.

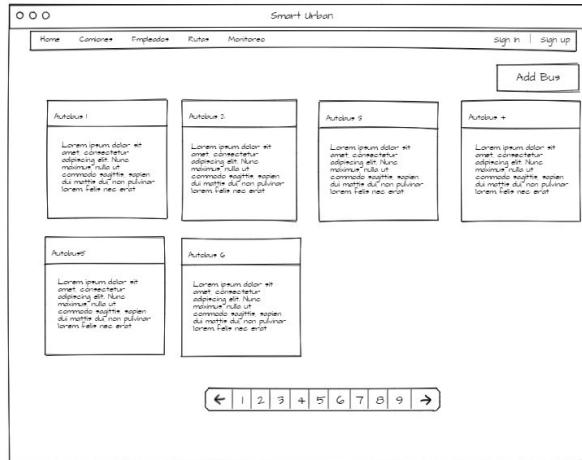


Figura 3.20: Interfaz de autobuses.

**Interfaz agregar autobús** En la figura 3.21 se muestra la interfaz para agregar autobús, representa la vista para agregar un autobús. Cuenta con los campos id conductor, id de la unidad, descripción, placas, ciudad, zona y dos botones: cancelar y añadir.

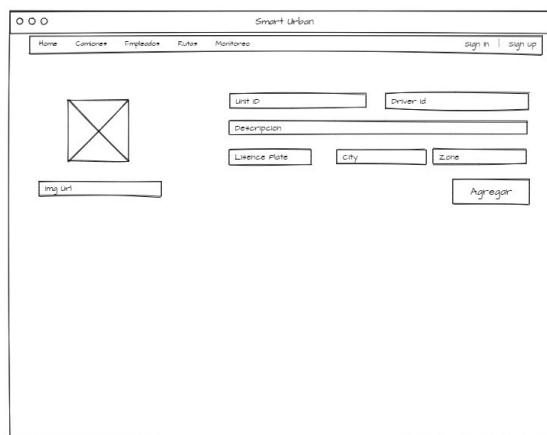


Figura 3.21: Interfaz agregar autobús.

**Interfaz agregar personal** En la figura 3.22 se muestra la interfaz para agregar personal (conductores), representa la vista para agregar un personal. Cuenta con diversos campos entre los principales: nombre, apellido, RFC, CURP, NSS, y correo. y un botón para añadir.

Figura 3.22: Interfaz agregar personal.

**Interfaz de rutas.** En la figura 3.23 se muestra la interfaz de las rutas, representa la vista para realizar diferentes acciones de las rutas. Muestra las rutas agregadas, las rutas existentes se pueden modificar y eliminar, de igual manera la interfaz cuenta con un botón para añadir nuevas rutas.

**Interfaz modificar ruta** En la figura 3.24 se muestra la interfaz para modificar una ruta, representa la vista para modificar una ruta de las que se encuentran agregadas. Cuenta con los campos lugar salida, lugar de llegada, duración y el estado. Cuenta con un botón para actualizar la ruta y muestra un mapa donde muestra la ruta asignada. Si se desea agregar paradas, se selecciona el mapa y se agrega la parada.

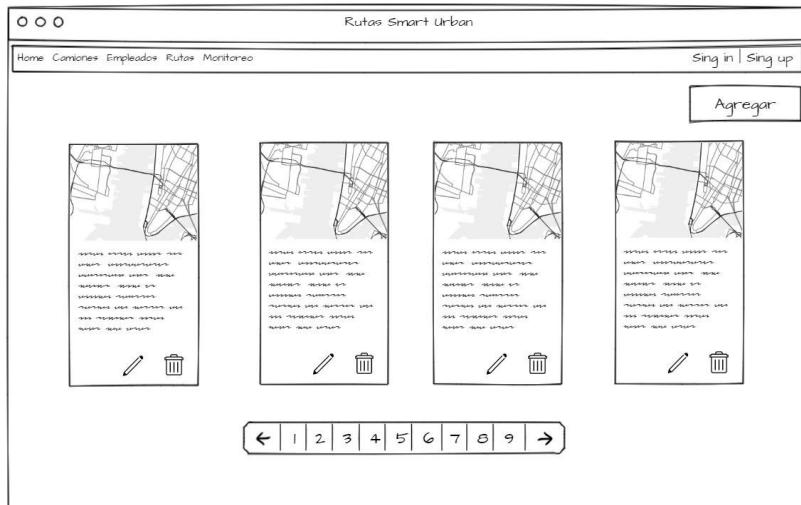


Figura 3.23: Interfaz de rutas.

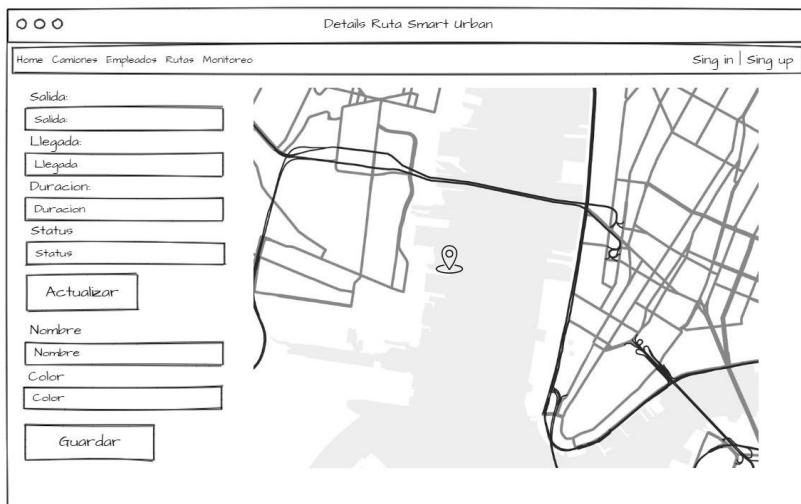


Figura 3.24: Interfaz modificar ruta.

### 3.4.2. Pruebas con la plataforma

Como ya se mencionó anteriormente se hizo uso de la plataforma *Hyperledger Fabric*. *Hyperledger Fabric* es una plataforma de código abierto de nivel empresarial mantenida por IBM y Linux Foundation. A diferencia de Bitcoin y Ethereum, *Hyperledger Fabric* no tiene ninguna moneda digital (criptomoneda), donde el acceso a la red está restringido solo a los miembros de la red, y nadie puede unirse a la red sin autorización. El mecanismo utilizado para validar las transacciones y crear bloques en *Hyperledger Fabric* es PBFT (*Practical Byzantine Fault Tolerance*, tolerancia práctica a fallas bi-

zantinas). Las transacciones se controlan en *Hyperledger Fabric* utilizando *chaincode* (contrato inteligente). Una transacción es una invocación o una solicitud de instancia que el par envía para su pedido y validación. La solicitud de instancia inicializa un *chaincode* en un canal en particular, mientras que las transacciones de invocación ejecutan una operación de lectura / escritura en el libro mayor.

Las pruebas realizadas se hicieron de manera local y en la infraestructura de la nube de Google, utilizando el servicio *compute Engine*, con el objetivo de realizar pruebas con la plataforma de *Hyperledger Fabric*, instalar todos los requisitos y comprobar que todo funciona correctamente mediante la construcción y generación de una red de prueba. Se detalla la instalación de los requisitos para utilizar *Hyperledger Fabric* en el apartado de apéndice A. A continuación se muestra la creación de red de prueba *blockchain*, para posteriormente crear la red de trabajo e iniciar la codificación de contratos inteligentes.

## Construcción de la Red

1. Abrir el subdirectorio *first-network*

**Comando:** `cd fabric-samples/first-network`

2. Generar artefactos de red, figura 3.25.

**Comando:** `./byfn.sh generate`

3. Activar la red figura 3.26.

**Comando:** `./byfn.sh up`

Los registros continuarán y lanzará todos los contenedores, luego generará un escenario completo de aplicación de *peer to peer*.

4. Dar de baja la red, figura 3.27.

Lo siguiente matará los contenedores, eliminará el material criptográfico y artefactos, y eliminará las imágenes de *chaincode* del Registro *Docker*.

**Comando:** `./byfn.sh down`

```
[wilybalderas@blockchain1 fabric-samples]$ cd first-network
[wilybalderas@blockchain1 first-network]$ ./byfn.sh generate
Generating certs and genesis block for channel 'mychannel' with CLI timeout of '10' seconds and CLI delay of '3'
seconds
Continue? [Y/n] y
proceeding ...
/home/wilybalderas/go/src/hello/fabric-samples/bin/cryptogen
#####
##### Generate certificates using cryptogen tool #####
#####
+ cryptogen generate --config=./crypto-config.yaml
org1.example.com
org2.example.com
+ res=0
+ set +x

Generate CCP files for Org1 and Org2
/home/wilybalderas/go/src/hello/fabric-samples/bin/configtxgen
#####
##### Generating Orderer Genesis block #####
#####
2020-02-06 18:10:35.149 UTC [common.tools.configtxgen] main > INFO 001 Loading configuration
2020-02-06 18:10:35.190 UTC [common.tools.configtxgen.localconfig] completeInitialization -> INFO 002 orderer type: etcDraft
2020-02-06 18:10:35.190 UTC [common.tools.configtxgen.localconfig] completeInitialization -> INFO 003 Orderer.EtcdRaft.Options unset, setting to tick_interval:"500ms" election_tick:10 heartbeat_tick:1 max_inflight_blocks:5 snapshot_interval_size:16777216
2020-02-06 18:10:35.190 UTC [common.tools.configtxgen.localconfig] Load -> INFO 004 Loaded configuration: /home/wilybalderas/go/src/hello/fabric-samples/first-network/configtx.yaml
2020-02-06 18:10:35.193 UTC [common.tools.configtxgen] doOutputBlock -> INFO 005 Generating genesis block
2020-02-06 18:10:35.193 UTC [common.tools.configtxgen] doOutputBlock -> INFO 006 Writing genesis block
#####
##### Generating channel configuration transaction 'channel.tx' #####
#####
+ configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./channel-artifacts/channel.tx -channelID mychannel
2020-02-06 18:10:35.228 UTC [common.tools.configtxgen] main > INFO 001 Loading configuration
2020-02-06 18:10:35.268 UTC [common.tools.configtxgen.localconfig] Load -> INFO 002 Loaded configuration: /home/wilybalderas/go/src/hello/fabric-samples/first-network/configtx.yaml
2020-02-06 18:10:35.268 UTC [common.tools.configtxgen] doOutputChannelCreateTx -> INFO 003 Generating new channel configtx
```

Figura 3.25: Generación de artefactos de red.

### 3.4.3. Codificación y pruebas de contratos inteligentes (*chaincodes*)

A partir de este apartado se presentan los detalles técnicos de la implementación del sistema, y se presentan los diferentes componentes de los que consta el *back-end* y el *front-end* del sistema.

#### Entorno de desarrollo

Como entorno de desarrollo para la implementación del sistema se cuenta con una máquina virtual Ubuntu 18.04 LTS, sobre la cual se despliega la red *blockchain* de *Hyperledger Fabric*. Para el desarrollo del *front-end* se empleó el editor de código visual studio code, del mismo modo, para el desarrollo de los *chaincode* o contratos inteligentes y la API Rest, como servidor de la API Rest de la *blockchain* se hace uso de un servidor web en nodeJS

Iniciar un libro mayor con activos, consultar activos, crear activos nuevos, entre otras, implica tres componentes:

1. La Red: para interactuar entre los nodos y la *blockchain*.

```
wilybalderas@blockchain1:~/go/src/hello/fabric-samples/first-network - Mozilla Firefox
https://ssh.cloud.google.com/projects/resolute-winter-253604/zones/us-central1-a/instances/blockchain1?authuser=...
go: downloading golang.org/x/net v0.0.0-2019081314303-74dc4d720e7
go: downloading golang.org/x/sys v0.0.0-20190813064441-fde4db37ae7a
go: downloading golang.org/x/text v0.3.2
go: extracting golang.org/x/sys v0.0.0-20190813064441-fde4db37ae7a
go: extracting google.golang.org/genproto v0.0.0-20190819201941-24fa4b261c55
go: extracting golang.org/x/text v0.3.2
~/go/src/hello/fabric-samples/first-network
Finished vendoring Go dependencies

START
Build your first network (BYFN) end-to-end test

Channel name : mychannel
Creating channel...
+ peer channel create -o orderer.example.com:7050 -c mychannel -f ./channel-artifacts/channel.tx --tls true --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem
+ rescan
+ peer tx
2020-02-06 18:12:24.550 UTC [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2020-02-06 18:12:24.577 UTC [channelCmd] InitCmdFactory -> INFO 002 Expect block, but got status: &{NOT_FOUND}
2020-02-06 18:12:24.581 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and orderer connections initialized
2020-02-06 18:12:24.782 UTC [cli.common] readBlock -> INFO 004 Expect block, but got status: &{SERVICE_UNAVAILABLE}
LE)
2020-02-06 18:12:24.786 UTC [channelCmd] InitCmdFactory -> INFO 005 Endorser and orderer connections initialized
2020-02-06 18:12:24.987 UTC [cli.common] readBlock -> INFO 006 Expect block, but got status: &{SERVICE_UNAVAILABLE}
LE)
2020-02-06 18:12:24.990 UTC [channelCmd] InitCmdFactory -> INFO 007 Endorser and orderer connections initialized
2020-02-06 18:12:25.192 UTC [cli.common] readBlock -> INFO 008 Expect block, but got status: &{SERVICE_UNAVAILABLE}
LE)
2020-02-06 18:12:25.195 UTC [channelCmd] InitCmdFactory -> INFO 009 Endorser and orderer connections initialized
2020-02-06 18:12:25.196 UTC [cli.common] readBlock -> INFO 009 Expect block, but got status: &{SERVICE_UNAVAILABLE}
LE)
2020-02-06 18:12:25.403 UTC [channelCmd] InitCmdFactory -> INFO 00b Endorser and orderer connections initialized
2020-02-06 18:12:25.604 UTC [cli.common] readBlock -> INFO 00c Expect block, but got status: &{SERVICE_UNAVAILABLE}
LE)
2020-02-06 18:12:25.608 UTC [channelCmd] InitCmdFactory -> INFO 00d Endorser and orderer connections initialized
```

Figura 3.26: Activación de la red.

2. Aplicación (Nodejs): que realiza llamadas a la red blockchain, invocando transacciones implementadas en el *chaincode* (contrato inteligente).
3. El contrato inteligente (Go): en sí mismo, implementa las transacciones que involucran interacciones con el libro mayor.

Para ello se realizan tres pasos principales:

1. Configuración de un entorno de desarrollo: La aplicación necesita una red con la cual interactuar, por lo que se implementa una red para el contrato inteligente y la aplicación.
2. Despliegue del contrato inteligente: El contrato inteligente permite realizar las transacciones y mediante la aplicación poder consultar y actualizar el libro mayor.
3. Interactuar con el contrato inteligente y la aplicación: La aplicación utilizará el contrato inteligente para crear, consultar y actualizar activos en el libro mayor.

### Configuración de un entorno de desarrollo

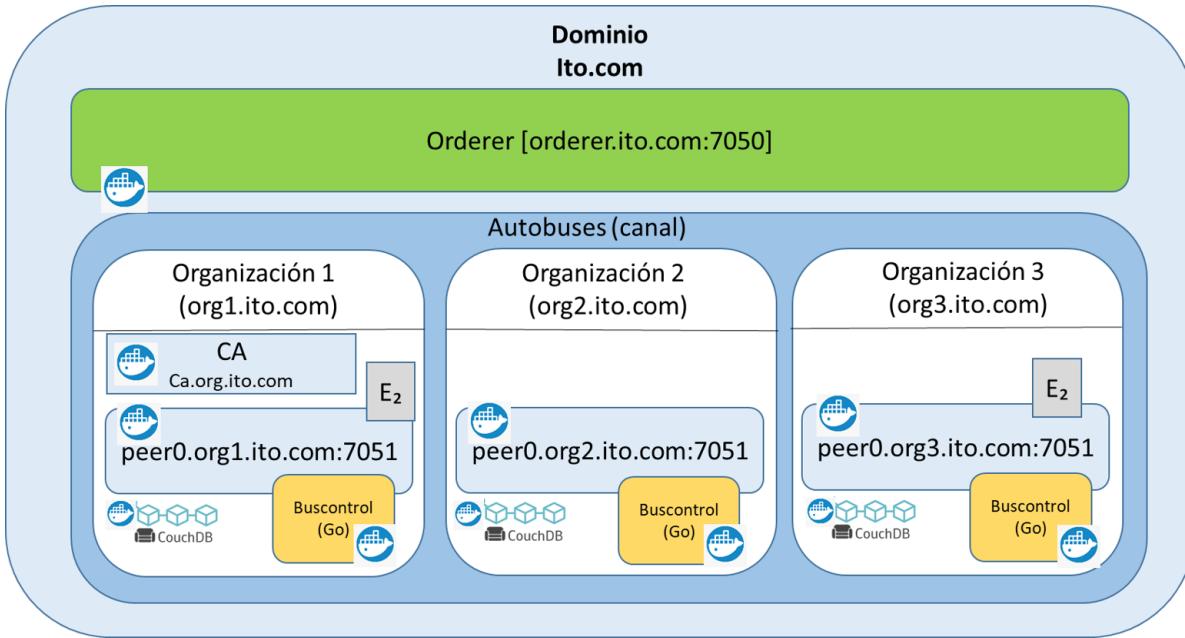
En la figura 3.28 se presenta el diseño de la red a utilizar, a continuación se describe dicha red y sus componentes.

```
wilybalderas@blockchain1:~/go/src/hello/fabric-samples/first-network - Mozilla Firefox
https://ssh.cloud.google.com/projects/resolute-winter-253604/zones/us-central1-a/instances/blockchain1?authuser=...
Invoke transaction successful on peer0.org1 peer0.org2 on channel 'mychannel'

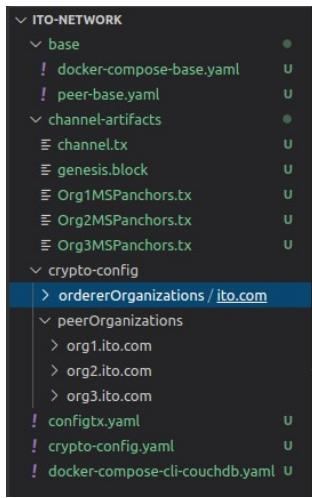
Querying chaincode on peer0.org1...
Querying on peer0.org1 on channel 'mychannel'...
Attempting to Query peer0.org1 ... 3 secs
+ peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
+ res=0
+ set +x
90
Query successful on peer0.org1 on channel 'mychannel'
Installing chaincode on peer1.org2...
+ peer lifecycle chaincode install mycc.tar.gz
+ res=0
+ set +x
90
Query successful on peer1.org2 on channel 'mychannel'
Attempting to Query peer1.org2 ... 3 secs
+ peer chaincode query -C mychannel -n mycc -c '{"Args":["query","a"]}'
+ res=0
+ set +x
90
Query successful on peer1.org2 on channel 'mychannel'
----- All GOOD, BYFN execution completed -----
```

Figura 3.27: Generación de red finalizada

- La red consta de tres participantes u organizaciones.
- Al dominio de la red se le dio el nombre de *ito.com*.
- Las tres organizaciones constan de un dominio.
- En la red se tiene el servicio de ordenamiento denominado *Orderer*: este ayuda a transaccionar y ordenas las solicitudes, peticiones o transacciones que envían las organizaciones.
- Se cuenta con un único canal denominado autobuses: es un canal privado donde solo estas tres organizaciones pueden ver los datos o lo que ocurre en el canal.
- Cada organización consta de un solo nodo.
- La red consta de una CA (certificado de autorización) en la Org1: este ayuda a controlar a los usuarios y aplicaciones que se enrolan en la red.
- Políticas de endosamiento: aprobación de transacciones, indica que solo la primera y tercera organización pueden aprobar transacciones, la segunda no puede, solo tiene rol de lectura.

Figura 3.28: Diseño de la red *blockchain* de trabajo

A continuación se muestra la estructura de la red en la figura 3.29 previamente diseñada. Esta estructura forma parte del *back-end*, el *back-end* del sistema cuenta por un lado con la configuración de la red *blockchain* y por otro, con la API Rest que permite la ejecución de los *chaincode* desde el *front-end*.

Figura 3.29: Estructura de la red *blockchain* de trabajo.

La red *blockchain* se configura mediante *Hyperledger Fabric*; se configuran los archivos *yaml* para posteriormente mediante la herramienta *cryptogen* de *Hyperledger Fa-*

bric generar las organizaciones, certificados, bloques, entre otros, dentro de la carpeta `crypto-config`.

A continuación se definen los elementos de configuración más importantes.

- `crypto-config.yaml`: Este archivo contiene la definición de topología de la red y a partir del mismo, se generan el conjunto de certificados para cada una de las organizaciones de la red y sus elementos.
- `configtx.yaml`: Define los elementos que se emplearán en la red y que serán pasados a `configtxgen` con el objetivo de crear el bloque génesis y los canales. Configura los artefactos como el `orderer` y el `channel`.
- `base/peer-base.yaml`: Contiene la configuración común a todos los nodos que conforman la red.
- `base/docker-compose-base.yaml`: Especifica la configuración de cada uno de los nodos que conforman la red.
- `docker-compose-cli-couch.yaml`: Especifica la definición de almacenamiento para cada uno de los nodos de cada organización y define la configuración del contenedor `cli`. Contenedor encargado de gestionar la comunicación por comandos con la red de *blockchain*.

Para crear la red, es necesaria la configuración anteriormente explicada y mediante el comando mostrado en la figura 3.30 se crea la red, posteriormente se puede observar que los contenedores se levantaron mediante `docker ps` (figura 3.31).

Para gestionar los contenedores `docker`, se optó utilizar una herramienta más visual llamada `portainer` en la cual es posible ver los contenedores anteriores. `Portainer` permite gestionar de forma muy fácil e intuitiva los contenedores `docker` a través de una interfaz gráfica.

```

File Edit Selection View Go Run Terminal Help
File Explorer Terminal Welcome
OPEN EDITORS
CURSOHF
chaincode
ito-network
base
! docker-compose-ba...
! peer-base.yaml
chaincode
channel-artifacts
! crypto-config
! configtx.yaml
! crypto-config.yaml
! docker-compose-cli-c...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Recent
Digest: sha256:f8c2b0a9ca640edf508a8a0830cf1963a1e0d2fd9936a64194b3f658e120b868
Status: Downloaded newer image for portainer/portainer:latest
916b5b1a277f9ee772825e539532070f9e4e23c43927874a20de407ccb66d03a
jose@jose-VirtualBox:~/fabric-samples/cursohF/ito-network$ export CHANNEL_NAME=autobuses
jose@jose-VirtualBox:~/fabric-samples/cursohF/ito-network$ export VERBOSE=false
jose@jose-VirtualBox:~/fabric-samples/cursohF/ito-network$ export FABRIC_CFG_PATH=$PWD
jose@jose-VirtualBox:~/fabric-samples/cursohF/ito-network$ CHANNEL_NAME=$CHANNEL_NAME docker-compose -f docker-compose-cli-co
urndb.yaml up -d
Creating network "ito-network_basic" with the default driver
Creating ca.org1.ito.com ... done
Creating couchdb2 ... done
Creating orderer.ito.com ... done
Creating couchdb1 ... done
Creating couchdb0 ... done
Creating peer0.org1.ito.com ... done
Creating peer0.org2.ito.com ... done
Creating peer0.org3.ito.com ... done
Creating cli ... done
jose@jose-VirtualBox:~/fabric-samples/cursohF/ito-network$ 
  
```

Figura 3.30: Creación de la red *blockchain*

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
18e05aa03c45	hyperledger/fabric-tools:2.2	"./bin/bash"		2 minutes ago	Up 2 minutes	
f839ca5c7e22	hyperledger/fabric-peer:2.2.0	"peer node start"		2 minutes ago	Up 2 minutes	0.0.0.0:
9051->7051/tcp	0.0.0.0:9053->7053/tcp	peer0.org3.ito.com		2 minutes ago	Up 2 minutes	0.0.0.0:
0d4ac09b9eac	hyperledger/fabric-peer:2.2.0	"peer node start"		2 minutes ago	Up 2 minutes	0.0.0.0:
8051->7051/tcp	0.0.0.0:8053->7053/tcp	peer0.org2.ito.com		2 minutes ago	Up 2 minutes	0.0.0.0:
8bf355a6bf6d	hyperledger/fabric-peer:2.2.0	"peer node start"		2 minutes ago	Up 2 minutes	0.0.0.0:
7051->7051/tcp	0.0.0.0:7053->7053/tcp	peer0.org1.ito.com		2 minutes ago	Up 2 minutes	0.0.0.0:
436df9f857691	couchdb:3.1	"tini -- /docker-entrypt.scr"l		2 minutes ago	Up 2 minutes	4369/tcp
9100/tcp	0.0.0.0:5984->5984/tcp	couchdb0		2 minutes ago	Up 2 minutes	4369/tcp
28000->5984b	couchdb0	"tini -- /docker-entrypt.scr"l		2 minutes ago	Up 2 minutes	4369/tcp
9100/tcp	0.0.0.0:5985->5985/tcp	couchdb1		2 minutes ago	Up 2 minutes	4369/tcp
a660154448	couchdb:3.1	"tini -- /docker-entrypt.scr"l		2 minutes ago	Up 2 minutes	4369/tcp
9100/tcp	0.0.0.0:5986->5984/tcp	couchdb2		2 minutes ago	Up 2 minutes	4369/tcp
ff9366cd3b8	hyperledger/fabric-ca:1.4.8	"sh -c 'fabric-ca-se"l		2 minutes ago	Up 2 minutes	0.0.0.0:
7054->7054/tcp	ca.org1.ito.com					

Figura 3.31: Verificación de la creación de la red

Para crear el canal previamente mencionado, denominado “autobuses”, se hace uso de *portainer* y de la consola del contenedor *cli*, en las figuras 3.32 y 3.33 se muestra la creación del canal.

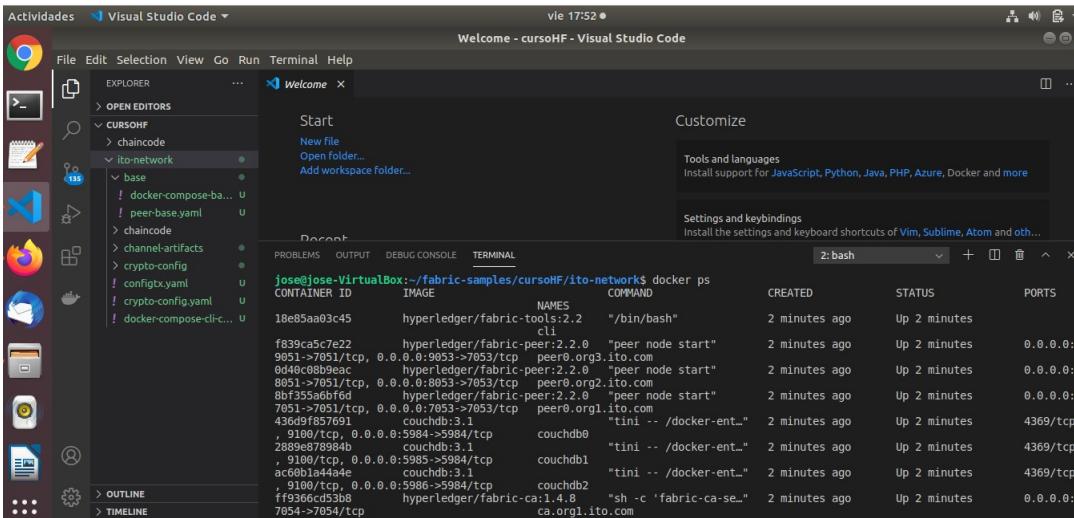


Figura 3.32: Creación del canal autobuses.

```
ge: 0xc000007fda0, {READY <nnil>}
2020-10-06 22:56:41.236 UTC [grpc] Info -> DEBU 039 Channel Connectivity change to READY
2020-10-06 22:56:41.292 UTC [msp.identity] Sign -> DEBU 03a Sign: plaintext: 0ADD060A1508051A0608A9ECF3FB05
2...86033CDA77AC12080A021A0012021A00
2020-10-06 22:56:41.292 UTC [msp.identity] Sign -> DEBU 03b Sign: digest: 3848385621DDDF7ABFB614F02B74C18B8
AF5EC8596C2AF9BFAE14389A9F248A1
2020-10-06 22:56:41.343 UTC [cli.common] readBlock -> INFO 03c Received block: 0
bash-5.0# ls
autobuses.block    channel-artifacts  crypto
bash-5.0#
```

Figura 3.33: Canal de autobuses creado.

El canal es una “subred” privada de comunicación entre dos o más miembros de la red, con el propósito de realizar transacciones privadas y confidenciales. Un canal lo definen los miembros (organizaciones), los **anchor peers**, el libro mayor compartido, los *chaincodes* y los nodos de servicio de pedidos (*ordering service node(s)*). Cada transacción en la red se ejecuta en un canal, donde cada parte debe estar autenticada y autorizada para realizar transacciones en ese canal. Por lo cual se deben unir todas las organizaciones, en la figura 3.34 se muestra el comando para unir la organización uno en la red y en la figura 3.35 se muestra el mensaje que se unió el canal.

```
autobuses.block  channel-artifacts  crypto
bash-5.0# peer channel join -b autobuses.block
2020-10-07 03:48:05.528 UTC [bccsp] GetDefault -> DEBU 001 Before using BCCSP, please call InitFactories().
Falling back to bootBCCSP.
2020-10-07 03:48:05.569 UTC [bccsp] GetDefault -> DEBU 002 Before using BCCSP, please call InitFactories().
Falling back to bootBCCSP.
2020-10-07 03:48:05.583 UTC [bccsp_sw] openKeyStore -> DEBU 003 KeyStore opened at [/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.ito.com/users/Admin@org1.ito.com/msp/keystore]...
done
2020-10-07 03:48:05.584 UTC [msp] getPemMaterialFromDir -> DEBU 004 Reading directory /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.ito.com/users/Admin@org1.ito.com/msp/signcerts
2020-10-07 03:48:05.584 UTC [msp] getPemMaterialFromDir -> DEBU 005 Inspecting file /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.ito.com/users/Admin@org1.ito.com/msp/signcerts/Admin@org1.ito.com-cert.pem
```

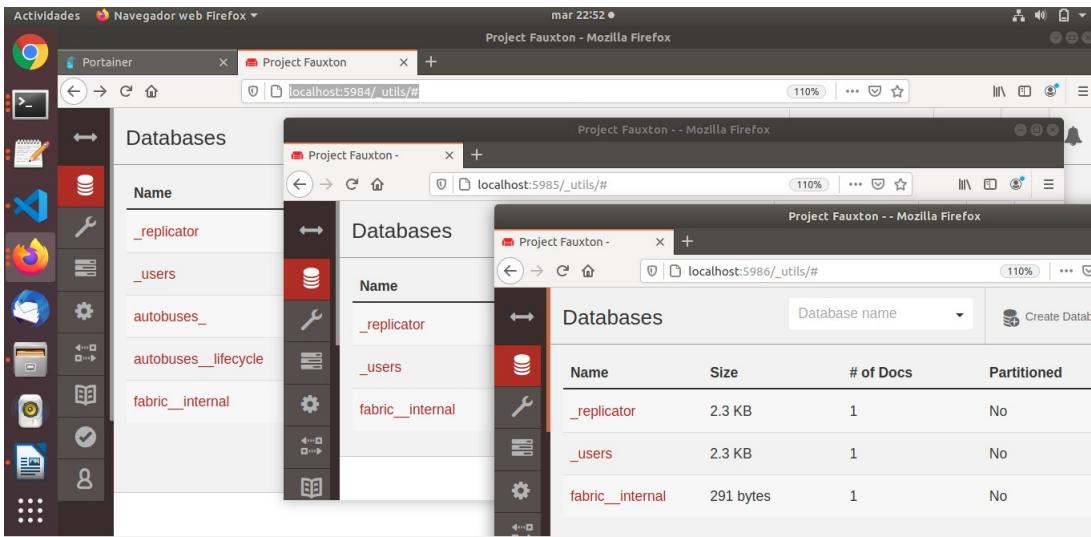
Figura 3.34: Comando para unir la Org1 al canal autobuses.

Lo anterior se realiza para cada organización en la red, para este caso en particular se tienen tres organizaciones.

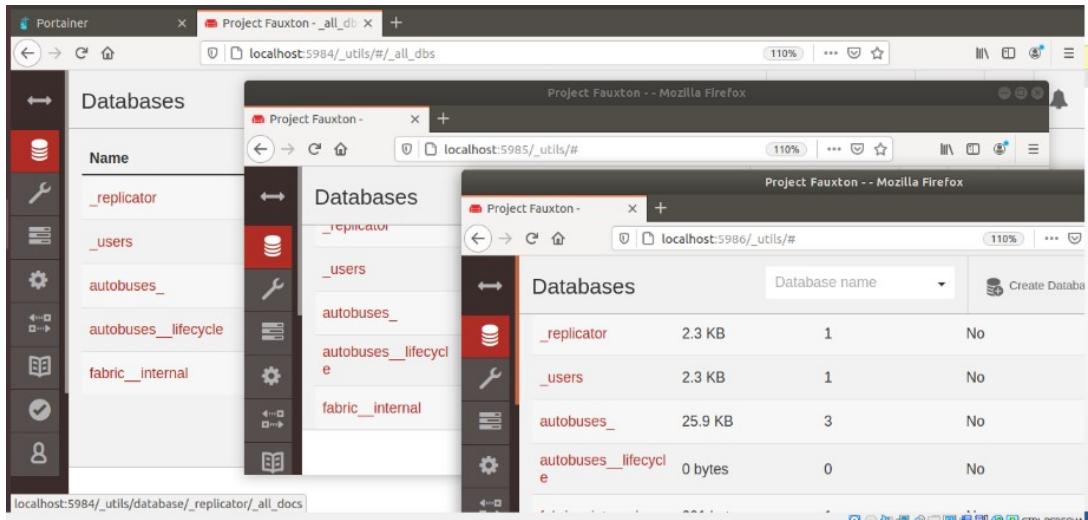
```
2020-10-07 03:48:05.625 UTC [grpc] Infof -> DEBU 026 Channel Connectivity change to READY
2020-10-07 03:48:05.625 UTC [channelCmd] InitCmdFactory -> INFO 027 Endorser and orderer connections initialized
2020-10-07 03:48:05.629 UTC [msp.identity] Sign -> DEBU 028 Sign: plaintext: 0AA4070A5C08011A0C08F5F4FB05
10...1070F4571A0A0A000A000A000A000A00
2020-10-07 03:48:05.630 UTC [msp.identity] Sign -> DEBU 029 Sign: digest: 81F8FDD8C2E54AA12A3500847DE9BB03C
84B37C5AE82FC00202D2A43FFF061CB
2020-10-07 03:48:06.289 UTC [channelCmd] executeJoin -> INFO 02a Successfully submitted proposal to join channel
bash-5.0#
```

Figura 3.35: Canal unido.

Al unirse la organización se actualiza el estado mundial: base de datos que contiene los valores actuales de un conjunto de estados del libro mayor. El estado mundial facilita la obtención del valor actual de estos estados, en lugar de tener que calcularlos recorriendo todo el registro de transacciones. Los estados del libro mayor se expresan, de forma predeterminada, como pares clave-valor. En el *couchDB* de la *Org1* se crea un nuevo estado mundial denominado *autobuses*. Sin embargo, como las demás organizaciones no se han unido no se observa algún cambio, figura 3.36.

Figura 3.36: Estado mundial, *World state*.

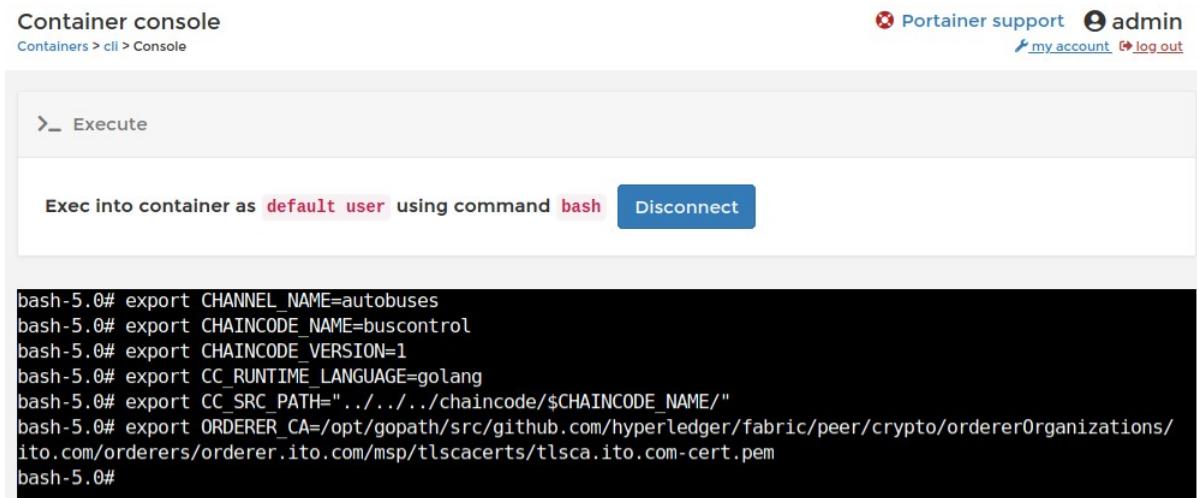
Dado que es una red descentralizada, al unir más organizaciones al canal, el estado mundial se replica en todas las organizaciones, figura 3.37.

Figura 3.37: *World state* replicado en todos los nodos.

Se desarrolla el *chaincode* en Go, el cual dota de la capacidad de ejecutar y hacer cumplir las condiciones establecidas en el mismo. Mediante el protocolo de consenso se validan y realizan nuevas transacciones que se propagan posteriormente a todos los nodos, para ser ejecutadas. Los *chaincodes* y el *world state* forman el *core* del *ledger* de *Hyperledger Fabric*. En el *world state* se almacena el histórico de estados por el cual ha pasado la *blockchain*, mientras que los *chaincodes* o *smart contracts*, definen las

interacciones que se pueden llevar a cabo entre los diferentes nodos que conforman la red. Ahora se asegura que sea más descentralizado y se cumplan algunas confirmaciones que permitan que el resto de organizaciones o nodos que son parte de la red estén de acuerdo que se despliegue el *chaincode* en la red, solo si existe el acuerdo se da paso a que sea parte de la red, para ello se usa el ciclo de vida de *chaincode*.

Para ello, se configuran algunas variables de entorno para desplegar el *chaincode*, figura 3.38, y se hace uso del ciclo de vida del *chaincode* para empaquetar este *chaincode* en un archivo `tar.gz`, figura 3.39.



The screenshot shows a web-based container console interface. At the top, there are navigation links: 'Container console' (highlighted), 'Containers > cli > Console'. On the right, there are user account links: 'Portainer support' (with a red exclamation icon), 'admin', 'my account', and 'log out'. Below the header, a sub-header reads 'Execute'. Underneath, a button says 'Exec into container as default user using command bash' with a 'Disconnect' button next to it. The main terminal window displays the following shell session:

```

bash-5.0# export CHANNEL_NAME=autobuses
bash-5.0# export CHAINCODE_NAME=buscontrol
bash-5.0# export CHAINCODE_VERSION=1
bash-5.0# export CC_RUNTIME_LANGUAGE=golang
bash-5.0# export CC_SRC_PATH="../../../../chaincode/${CHAINCODE_NAME}/"
bash-5.0# export ORDERER_CA=/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/ito.com/orderers/orderer.ito.com/msp/tlscacerts/tlsca.ito.com-cert.pem
bash-5.0#

```

Figura 3.38: Variables de entorno para el ciclo de vida *chaincode*.



The screenshot shows a terminal window with the following command and its output:

```

bash-5.0# peer lifecycle chaincode package ${CHAINCODE_NAME}.tar.gz --path ${CC_SRC_PATH} --lang ${CC_RUNTIME_LANGUAGE} --label ${CHAINCODE_NAME}_${CHAINCODE_VERSION} >&log.txt
bash-5.0# ls
autobuses.block    buscontrol.tar.gz  channel-artifacts  crypto                  log.txt
bash-5.0#

```

Figura 3.39: Empaquetado del *chaincode*.

El empaquetado se debe distribuir e instalar en cada organización, figura 3.40, el cual al finalizar provee un identificador que posteriormente se utiliza en las políticas de endorsamiento, figura 3.41.

```

bash-5.0# peer lifecycle chaincode install buscontrol.tar.gz
2020-10-07 19:55:58.770 UTC [bccsp] GetDefault -> DEBU 001 Before using BCCSP, please call InitFactories().
Falling back to bootBCCSP.
2020-10-07 19:55:58.788 UTC [bccsp] GetDefault -> DEBU 002 Before using BCCSP, please call InitFactories().
Falling back to bootBCCSP.
2020-10-07 19:55:58.792 UTC [bccsp_sw] openKeyStore -> DEBU 003 KeyStore opened at [/opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.ito.com/users/Admin@org1.ito.com/msp/keystore]...
done
2020-10-07 19:55:58.793 UTC [msp] getPemMaterialFromDir -> DEBU 004 Reading directory /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.ito.com/users/Admin@org1.ito.com/msp/signcerts
2020-10-07 19:55:58.793 UTC [msp] getPemMaterialFromDir -> DEBU 005 Inspecting file /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.ito.com/users/Admin@org1.ito.com/msp/signcerts/Admin@org1.ito.com-cert.pem
2020-10-07 19:55:58.793 UTC [msp] getPemMaterialFromDir -> DEBU 006 Reading directory /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.ito.com/users/Admin@org1.ito.com/msp/cacerts
2020-10-07 19:55:58.793 UTC [msp] getPemMaterialFromDir -> DEBU 007 Inspecting file /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org1.ito.com/users/Admin@org1.ito.com/msp/cacerts/ca.org1.ito.com-cert.pem

```

Figura 3.40: Instalación del *chaincode*.

```

2020-10-07 19:55:58.842 UTC [grpc] Infof -> DEBU 032 Channel Connectivity change to READY
2020-10-07 19:55:58.844 UTC [msp.identity] Sign -> DEBU 033 Sign: plaintext: 0AAA070A6208031A0C08CEBAF8FB05
10...FDFE5B000000FFFF7F01337C002E0000
2020-10-07 19:55:58.844 UTC [msp.identity] Sign -> DEBU 034 Sign: digest: 8CF74323B9F6529CD1060580044EF481F
88225A3BCA93A8251D30229B9669188
2020-10-07 19:56:31.884 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 035 Installed remotely:
response:<status:200 payload:"\nMbuscontrol_1:8cd7bfcd66e40a59e3574155ab6a9bb80cc6cdea27c4883aa16d38cf8
f59b\022\014buscontrol_1" >
2020-10-07 19:56:31.885 UTC [cli.lifecycle.chaincode] submitInstallProposal -> INFO 036 Chaincode code pack
age identifier: buscontrol_1:8cd7bfcd66e40a59e3574155ab6a9bb80cc6cdea27c4883aa16d38cf8f59b
bash-5.0#

```

Figura 3.41: Instalación del *chaincode* finalizada.

Se realiza la definición de las políticas de endorsamiento solo en las organizaciones deseadas. Las políticas de endorsamiento o de aprobación solo se definen para el *chaincode*, para este *chaincode* en particular solo la *org1* y *org3* son capaces de firmar transacciones de recibir transacciones, solo estas organizaciones tienen permisos de escritura en este *chaincode*, figura 3.42. Las políticas de endorsamiento son específicas del *chaincode* en el canal.

```

ge: 0xc0004931e0, {READY <nil>}
2020-10-07 20:33:06.292 UTC [grpc] Infof -> DEBU 032 Channel Connectivity change to READY
2020-10-07 20:33:06.294 UTC [msp.identity] Sign -> DEBU 033 Sign: plaintext: 0AB5070A6D08031A0C0882CCF8FB05
10...031A00120B0A074F7267334D53501003
2020-10-07 20:33:06.295 UTC [msp.identity] Sign -> DEBU 034 Sign: digest: 93F58B2E080C7AF0170884AC4CBE88CFF
F51645D3B30DCCD36D4B3B93F52FD47
{
    "approvals": {
        "Org1MSP": true,
        "Org2MSP": false,
        "Org3MSP": true
    }
}
bash-5.0#

```

Figura 3.42: Políticas de endorsamiento aprobadas en la *org1* y *org2*.

por último se compromete (*commit*) y se pone en marcha el *chaincode*, se hace uso de lifecycle y se especifica la **url** del servicio de ordenamiento, la **cafile** del servicio y se indica para qué *peer* se compromete (*peer0* de la *org1* y *peer0* de la *org3*), se especifica el canal de este *chaincode*, el nombre del *chaincode*, la versión y las políticas, figura 3.43. Se confirma mediante un *transaction id* para el primer nodo de la *Org1* y un *transaction id* para el *Org3*, figura 3.44.

```
bash-5.0# peer lifecycle chaincode commit -o orderer.acme.com:7050 --tls --cafile $ORDERER_CA --peerAddress es peer0.org1.acme.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/pe erOrganizations/org1.acme.com/peers/peer0.org1.acme.com/tls/ca.crt --peerAddresses peer0.org3.acme.com:7051 --tlsRootCertFiles /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/peerOrganizations/org3.acme.c om/peers/peer0.org3.acme.com/tls/ca.crt --channelID $CHANNEL_NAME --name $CHAINCODE_NAME --version $CHAINCO DE_VERSION --sequence 1 --signature-policy "OR ('Org1MSP.peer','Org3MSP.peer')"
```

Figura 3.43: commit del *chaincode*

```
6CB6162C97AFCE7A5311ECB6FE1F311
2020-10-07 20:45:13.698 UTC [msp.identity] Sign -> DEBU 05c Sign: plaintext: 0ADD060A1508051A0608D9D1F8FB05
22...00120D1A0B08FFFFFFFFFFFFF01
2020-10-07 20:45:13.698 UTC [msp.identity] Sign -> DEBU 05d Sign: digest: AF5B909BAAD53DF42B7E7602A43546667
EFE94F897A918495E73D0FB9B22AC78
2020-10-07 20:45:13.699 UTC [msp.identity] Sign -> DEBU 05e Sign: plaintext: 0ADD060A1508051A0608D9D1F8FB05
22...00120D1A0B08FFFFFFFFFFFFF01
2020-10-07 20:45:13.699 UTC [msp.identity] Sign -> DEBU 05f Sign: digest: 2DD800251C3C120C7A65D783F7621CDD7
DFB286F5499AA6262C6C33D821C325
2020-10-07 20:45:15.929 UTC [chaincodeCmd] ClientWait -> INFO 060 txid [21969a5f4d2e4e89dfa248e6e852d04640efd0631b8eb8dcba4d2fc6561103f8] committed with status (VALID) at peer0.org3.ito.com:7051
2020-10-07 20:45:15.950 UTC [chaincodeCmd] ClientWait -> INFO 061 txid [21969a5f4d2e4e89dfa248e6e852d04640efd0631b8eb8dcba4d2fc6561103f8] committed with status (VALID) at peer0.org1.ito.com:7051
bash-5.0#
```

Figura 3.44: Confirmación del *commit*

## Pruebas del *chaincode*

Finalmente ya que se instaló el *chaincode*, se crea una instancia a través del CLI en la red de *Hyperledger Fabric*, lo que permite la interacción con el libro mayor compartido de la red, para la interacción con el libro mayor se utilizan las interfaces de *chaincode*:

- **Init()**: Se llama a **init** cuando implementa por primera vez. Como su nombre lo indica, esta función se utiliza para realizar cualquier especialización que necesite el *chaincode*.

- **Invoke()**: `Invoke` se llama cuando se desea llamar a funciones de *chaincode* para hacer una acción(es decir, leer o escribir en el libro mayor). Las invocaciones se capturan como transacciones, que se agrupan en bloques en la cadena.
- **Query()**: `Query` se llama cuando se desea llamar una función de consulta, es decir, leer en el libro mayor.

En dichas interfaces se llaman las funciones desarrolladas en cada *chaincode*, por ejemplo se hace uso de la función `set` utilizando la interfaz `invoke`, el endpoint del servicio de ordenamiento, se usa el protocolo seguro, la ruta de archivo `cafile`, el nombre del canal, el nombre del *chaincode* y los argumentos de ejecución, figura 3.45.

```
bash-5.0# peer chaincode invoke -o orderer.ito.com:7050 --tls --cafile $ORDERER_CA -C $CHANNEL_NAME -n $CHAINCODE_NAME -c '{"Args":["Set","id:1","Jose Luis","XTL-0670", "Orizaba-Mendoza"]}'  
swCQYD\nVQQGEwJVUzETMBEGA1UECBMkQ2FsaWZvcn5pYTEwMBQGA1UEBxMNU2FuIEZyYw5j\naXNjbzENMAsGA1UECxMEcgVlcjEbMBkGA  
1UEAxMScGVlcjAub3JnM55pdG8uY29t\nMFkwEwYHKoZIzj0CAQIKoZIzj0DAQcDgAEvnsP1kp82tJREI3LL/WtCU70qUVh\nb/pY2eFZ  
VH+KvuyXyRinYOrfHhLSndkLGShhuVen9p7lFALFcGqNZkx8KNMMEsw\nDgYDVR0PAQH/BAQDAgeMAwGA1UdEwEB/wQMAwKwYDVR0JB  
CQwIoAg0GhaeHEE\nRb0a9PBjgo6J\mdx5HHX08xRU6VPMuQA5u8wCgYIKoZIzj0EAwIDRwAwRAIgGd3r\nGVTcpyk7cBbi2XGxPEqnCtXv  
aukAHICkL5TAK4CIDx3n4L7f5XunHrmS82oa0Uz\n9ecGRqIez3l5GoXPJKR2\n-----END CERTIFICATE-----\n signature:"0\\002 o|37642x\\326\\333\\327\\013\\333qt(^\\242\\326\\353\\322;\\354\\233\\267\\373\\317\\031cg\\363\\335\\002 \\025\\202\\334\\276\\023\\217\\361[\\231\\276\\345\\2063jK\\327\\251d\\023Ax:\\177*\\374\\036,@\\221\\352\\307J" >  
2020-10-07 20:53:01.835 UTC [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 045 Chaincode invoke successful.  
result: status:200  
bash-5.0#
```

Figura 3.45: Intefaz invoke.

Para la consulta del elemento que se escribió anteriormente se hace uso la interfaz `Query`, utilizando la función `Query` del *chaincode*, se especifica el nombre del canal, el nombre del *chaincode* e indica el argumento *id*, figura 3.46. De igual forma si se revisa la base de datos, dado que es una red descentralizada se replican los datos en todos los nodos. El estado o *ledger* es un libro distribuido de clave valor (consta de un *id*, una llave y el valor), el valor se guarda en un `Json` que contiene la información, figura 3.47.

```
bash-5.0# peer chaincode query -C $CHANNEL_NAME -n $CHAINCODE_NAME -c '{"Args":["Query","id:1"]}'  
2020-10-07 21:05:09.818 UTC [grpc] Infof -> DEBU 031 Channel Connectivity change to READY  
2020-10-07 21:05:09.819 UTC [msp] GetDefaultSigningIdentity -> DEBU 032 Obtaining default signing identity  
2020-10-07 21:05:09.819 UTC [msp.identity] Sign -> DEBU 033 Sign: plaintext: 0AB5070A6D08031A0C0885DBF8FB05  
10...6C1A0D0A0551756572790A0469643A31  
2020-10-07 21:05:09.819 UTC [msp.identity] Sign -> DEBU 034 Sign: digest: 1E2AB7729D7F375350B99F92EC77F731F  
99F4858A142FAD4492452269E404E1A  
{"conductor":"Wily","placa":"XTL-0670","ruta":"Orizaba-Mendoza"}  
bash-5.0#
```

Figura 3.46: Intefaz query.

The screenshot shows the Project Fauxton interface for a database named 'autobuses\_buscontrol'. On the left, there's a sidebar with icons for database management. The main area shows a table with one row, 'id:1', under the 'Metadata' tab. Below the table, a modal window is open for document 'id:1'. The modal has a 'Save Changes' button and a 'Cancel' button. The JSON content of the document is displayed:

```

1 - {
2 -   "_id": "id:1",
3 -   "_rev": "1-6e26cf18944f067dbb1751d4aed809a0",
4 -   "conductor": "Jose Luts",
5 -   "placa": "XTL-0670",
6 -   "ruta": "Orizaba-Mendoza",
7 -   "~version": "CgM8BgA="
8 - }

```

Figura 3.47: Lectura del elemento `id:1` en el *world state*.

Para cada *chaincode* se debe hacer uso del (ciclo de vida,*lifecycle*), para desplegar el *chaincode*.

Dado el caso de estudio se desarrollaron cinco *chaincodes*:

1. **buscontrol**: *chaincode* para realizar la gestión de los autobuses.
2. **drivercontrol**: *chaincode* para realizar la gestión de los conductores.
3. **payroutescontrol**: *chaincode* para realizar la gestión de las tarifas.
4. **payscontrol**: *chaincode* para realizar la gestión de los pagos o cobros.
5. **routecontrol**: *chaincode* para realizar la gestión de las rutas.

Cada *chaincode* consta de una serie de métodos que conforman al *smart contract* o *chaincode* de la *blockchain*, figura 3.48 y 3.49. Se describen a continuación los métodos implementados en el *chaincode* buscontrol.

● buscontrol.go - buscontrol - Visual Studio Code

Go Run Terminal Help

```
buscontrol.go ●
buscontrol.go
package main
import (
    "encoding/json"
    "fmt"
    "github.com/hyperledger/fabric-contract-api-go/contractapi"
)
type SmartContract struct {
    contractapi.Contract
}
type Bus struct {
    IdConductor string `json:"idconductor"`
    Descripcion string `json:"descripcion"`
    Placa string `json:"placa"`
    Ciudad string `json:"ciudad"`
    Ruta string `json:"ruta"`
    IdRuta string `json:"idRuta"`
    FInicio string `json:"fInicio"`
    FTermino string `json:"fTermino"`
    HoraSalida string `json:"horaSalida"`
    Horalegada string `json:"horaLegada"`
    Comentarios string `json:"comentarios"`
}
type QueryResult struct {
    Key string `json:"Key"`
    Buses *Bus
}
func (s *SmartContract) Set(ctx contractapi.TransactionContextInterface, busId string, idconductor string, descripcion string, placa string, ciudad string, ruta string, idRuta string, finicio string, fTermino string, horaSalida string, horaLegada string, comentarios string) error {
    exists, err := s.BusExists(ctx, busId)
    if err != nil {
        return err
    }
    if exists {
```

Figura 3.48: *Chaincode buscontrol*

```
Go Run Terminal Help

buscontrol.go ●
buscontrol.go
64
65 func (s *SmartContract) UpdateBus(ctx contractapi.TransactionContextInterface, busId string, idconductor string, descripcion string,
66 placas string, ciudad string, ruta string, idRuta string, finicio string, ftermimo string, horaSalida string, horaLegada string,
67 comentarios string) error {
68 exists, err := s.BusExists(ctx, busId)
69 if err != nil {
70     return err
71 }
72 if !exists {
73     return fmt.Errorf("el autobus %s no existe", busId)
74 }
75
76 bus := Bus{
77     IdConductor: idconductor,
78     Descripcion: descripcion,
79     Placa: placas,
80     Ciudad: ciudad,
81     Ruta: ruta,
82     IdRuta: idRuta,
83     FInicio: finicio,
84     FTermimo: ftermimo,
85     HoraSalida: horaSalida,
86     HoraLegada: horaLegada,
87     Comentarios: comentarios,
88 }
89 assetJSON, err := json.Marshal(bus)
90 if err != nil {
91     return err
92 }
93
94 return ctx.GetStub().PutState(busId, assetJSON)
95 }
96
97 func (s *SmartContract) Query(ctx contractapi.TransactionContextInterface, busId string) (*Bus, error) {
98 busAsBytes, err := ctx.GetStub().GetState(busId)
99
100 if err != nil {
101     return nil, fmt.Errorf("No se pudo leer en el worldstate. %s", err.Error())
102 }
```

Figura 3.49: Continuación *chaincode buscontrol*

Métodos implementados en el *chaincode* buscontrol:

- **set()**: Método para insertar los datos de un autobús en la *blockchain*. Recibe todos los parámetros definidos en la estructura del activo, es decir, todos los valores que constituyen al autobus.
- **updateBus()**: Método que tiene como objetivo la actualización de un autobús. Recibe como parámetros el *id* del autobús y los parámetros a modificar.
- **Query**: Método para consultar un único autobús, recibe como parámetro el *id* del autobús a consultar.
- **QueryAllBuses**: Método para consultar el conjunto total de autobuses.
- **DeleteBuses**: Método para eliminar del estado mundial el activo. Recibe como parámetro el *id* del autobús a eliminar.
- **BusExists**: Método para verificar si ya ha sido creado o existe el activo. Recibe como parámetro el **id** del activo.

Para la interacción entre el *front-end* y el *back-end*, se hace uso de *Hyperledger Fabric* SDK para nodeJS el cual permite que las aplicaciones interactúen con una red *blockchain* de *Fabric*. Se hace uso de NodeJs con el objetivo de desarrollar una API para enviar transacciones al *ledger* o consultar su contenido. Para esta API se utiliza el marco de trabajo *express* para crear el servidor y consta de diversos métodos de petición. A continuación se describen estos métodos para la interacción con el *chaincode* **buscontrol** previamente explicado y posteriormente realizar peticiones desde el *front-end*.

Métodos de la API:

- **app.get(/ConsultarAutobus)**: Método *get* para consultar un único autobús, recibe como parámetro el *id* del autobús y se hace uso de la función **Query** del *chaincode*, figura 3.50.

```

103  **** Consultar por id ****
104
105  app.get('/ConsultarAutobus/:id?', (req, res) => {
106    var projectID = req.params.id;
107    let result = contract.evaluateTransaction('Query', projectID);
108    let data = initPromise(result).then(function (result1) {
109      console.log(prettyJSONString(result1.toString()));
110      res.send(prettyJSONString(result1.toString()));
111    });
112  });
113

```

Figura 3.50: Método *get* para consulta autobuses.

- `app.post(/crearAutobus)`: Método *post* para crear un autobús nuevo, recibe diversos parámetros y utiliza las funciones del *chaincodebusExists* para no duplicar autobuses y `set` para crear o dar de alta el nuevo autobús, figura 3.51.

```

150 **** Crear ****
151
152  app.post('/crearAutobus', (req, res) => {
153    var params = req.body;
154    var busId = params.id;
155    var idConductor = params.idconductor;
156    var descripcion = params.descripcion;
157    var placa = params.placa;
158    var ciudad = params.ciudad;
159    var ruta = params.ruta;
160    result1 = contract.evaluateTransaction('BusExists', busId);
161    let data = initPromise(result1).then(function (result) {
162      console.log(prettyJSONString(result.toString()));
163      if (result == "true") {
164        return res.status(404).send({ message: 'El autobus ya existe' });
165      } else {
166        contract.submitTransaction('Set', busId, idConductor, descripcion, placa, ciudad, ruta);
167        return res.status(200).send({
168          param: params
169        })
170      }
171    });
172  });
173

```

Figura 3.51: Método *post* para crear un autobús.

- `app.post(/modificarAutobus)`: Método *post* para modificar un autobús, recibe diversos parámetros y utiliza las funciones del *chaincode busExists* para comprobar que ya exista el autobús con el *id* especificado y `UpdateBus` para modificar los parámetros especificados, figura 3.52.
- `app.post(/EliminarAutobus)`: Método *post* para eliminar un autobús, recibe el parámetro *id* del autobús y utiliza las funciones del *chaincode busExists* para comprobar que sí exista el autobús especificado y `DeleteBus` para eliminar el autobús especificado, figura 3.53.
- `app.get(/ConsultarAutobuses)`: Método *get* para consultar el conjunto de autobuses, se hace uso de la función `QueryAllBuses` del *chaincode*, figura 3.54.

```

283 //***** Modificar *****/
284
285     app.post('/modificarAutobus', (req, res) => {
286         var params = req.body;
287         var busId = params.id;
288         var idConductor = params.idconductor;
289         var descripcion = params.descripcion;
290         var placa = params.placa;
291         var ciudad = params.ciudad;
292         var ruta = params.ruta;
293         var idRuta = params.idRuta;
294         var fInicio= params.fInicio;
295         var fTermino= params.fTermino;
296         var horaSalida=params.horaSalida;
297         var horaLlegada=params.horaLlegada;
298         var comentarios=params.comentarios;
299         result1 = contract.evaluateTransaction('BusExists', busId);
300         let data = initPromise(result1).then(function (result) {
301             console.log(prettyJSONString(result.toString()));
302             if (result == "false") {
303                 return res.status(404).send({ message: 'El autobus no existe' });
304             } else {
305                 contract.submitTransaction('UpdateBus', busId, idConductor, descripcion, placa, ciudad, ruta, idRuta,
306                 return res.status(200).send({
307                     param: params
308                 })
309             }
310         });
311     });
312 });
313

```

Figura 3.52: Método *post* para modificar un autobús.

```

401 //***** Eliminar *****/
402
403     app.post('/EliminarAutobus', (req, res) => {
404         var params = req.body;
405         var busId = params.id;
406         result1 = contract.evaluateTransaction('BusExists', busId);
407         let data = initPromise(result1).then(function (result) {
408             console.log(prettyJSONString(result.toString()));
409             if (result == "false") {
410                 return res.status(404).send({ message: 'El autobus no existe' });
411             } else {
412                 contract.submitTransaction('DeleteBus', busId);
413                 return res.status(200).send({
414                     param: params
415                 })
416             }
417         });
418     });
419 });
420

```

Figura 3.53: Método *post* para eliminar un autobús.

```

489 //***** Consulta todos *****/
490
491     app.get('/ConsultarAutobuses', (req, res) => {
492
493         let result = contract.evaluateTransaction('QueryAllBuses');
494         let data = initPromise(result).then(function (result1) {
495             console.log(prettyJSONString(result1.toString()));
496             res.send(prettyJSONString(result1.toString()));
497         })
498     });
499

```

Figura 3.54: Método *get* para consultar todos los autobuses.

Para probar la API desarrollada se hace uso de la herramienta *Postman*, esta herramienta permite crear peticiones sobre APIs de una forma sencilla y poder, de esta manera, probar las APIs como se muestra en los siguientes ejemplos, figuras 3.55 y 3.56.

The screenshot shows the Postman interface with an 'Untitled Request' titled 'POST http://localhost:3000/CrearAutobus'. The 'Body' tab is selected, showing a table with four rows: 'id' (value: 2), 'nombre' (value: jose), 'placa' (value: XTV-20), and 'ruta' (value: orizaba-maltrata). Other tabs like 'Params', 'Headers', 'Tests', and 'Settings' are visible at the top.

Figura 3.55: Petición *post* realizada desde *Postman*.

The screenshot shows the Postman interface with an 'Untitled Request' titled 'GET http://localhost:3000/ConsultarAutobus/id:2'. The 'Body' tab is selected, showing a table with three rows: 'conductor' (value: jose), 'placas' (value: XTV-20), and 'ruta' (value: orizaba-maltrata). Below the table, the status is shown as 200 OK with a response size of 304 B. The 'Pretty' tab is selected in the preview area.

Figura 3.56: Petición *get* realizada desde *Postman*.

### 3.4.4. Codificación y pruebas con la aplicación externa

La aplicación externa hace referencia al *front-end* el cual fue desarrollado mediante el uso de AngularJS. Se ha llevado a cabo mediante el desarrollo de diferentes componentes, dado el caso de estudio a continuación se describe cada componente.

Componentes principales:

- **transportes**: componente para gestionar autobuses, consta de cuatro componentes **add-car**, **calendar**, **car-details**, **car-list**, figura 3.57. Cada componente se enfoca en una acción a realizar, por ejemplo, agregar un autobús, calendarizar, listar todos los autobuses, entre otros.
- **routes**: componente para gestionar las rutas

- **payRoutes**: componente para gestionar las tarifas.
- **employee**: componente para gestionar los conductores.
- **autobuses**: componente para gestionar los pagos de los autobuses.

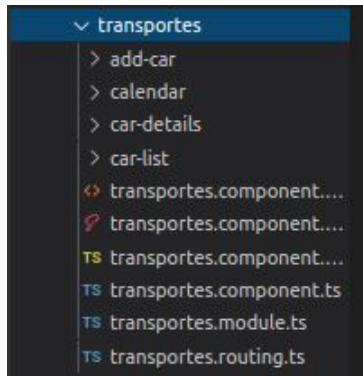


Figura 3.57: Componente transportes (Angular).

Para cada componente se creó un servicio, figura 3.58 , para realizar peticiones a la API creada y explicada anteriormente. Los servicios son clases que se encargarán de acceder a los datos para entregarlos a los componentes.

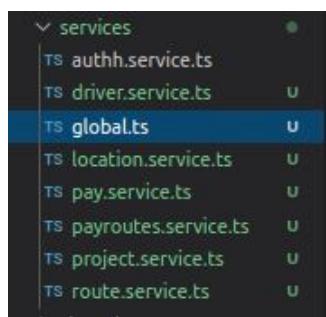


Figura 3.58: Servicios(Angular).

Se creó un **archivo.ts** global, este **archivo.ts** consta de tres simples líneas de código en donde se llama al servidor previamente creado en *express*, figura 3.59. Este archivo lleva por nombre **global.ts** y sera reutilizado en todos los servicios para acceder a la API.

Llamadas HTTP a la API usando **HttpClient**.

Para usar **HttpClient** de Angular en cualquier parte, se importa el módulo **HttpClientModule**, en la sección **imports** del **app.module.ts**:

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... calendar.component.html modal-calendar.component.html car-details.component.ts
OPEN EDITORS 2 UNSAVED
SMARTURBAN > sale
src > app > services > ts global.ts ...
1 export var Global = {
2   url:'http://localhost:3000/'
3 };

```

Figura 3.59: global.ts

- import { HttpClientModule } from "@angular/common/http";

Se debe importar e injectar en los constructores de los servicios desde los que van a realizar las llamadas HTTP. Por ejemplo, en este servicio con el cual se va a trabajar, se ha creado para listar los autobuses, consulta un autobús, actualizar un autobús o eliminar un autobús que vienen desde la API, con eso listo se realizan llamadas HTTP, figura 3.60.

```

ts project.service.ts •
src > app > services > ts project.service.ts > ProjectService > url
1 import { Injectable } from '@angular/core';
2 import { HttpClient, HttpHeaders } from '@angular/common/http';
3 import { Observable } from 'rxjs/Observable';
4 import { Global } from './global';
5 import { Bus } from '../models/bus';
6
7 @Injectable({
8   providedIn: "root"
9 })
10 export class ProjectService{
11   public url:string;
12   public opcion: string;
13   constructor(
14     private _http: HttpClient
15   ){
16     this.url = Global.url;
17   }
18   testService(){
19     return 'probando el servicio de angular';
20   }
21   getBus(opcion): Observable<any>{
22     let headers = new HttpHeaders().set('Content-Type', 'application/json');
23     return this._http.get(this.url+'ConsultarAutobus/'+opcion, {headers:headers});
24   }
25   getBuses(): Observable<any>{
26     let headers = new HttpHeaders().set('Content-Type', 'application/json');
27     return this._http.get(this.url+'ConsultarAutobuses', {headers:headers});
28   }
29   saveBus(bus: Bus): Observable<any>{
30     let params=JSON.stringify(bus);
31     let headers = new HttpHeaders().set('Content-Type', 'application/json');
32     return this._http.post(this.url+'crearAutobus', params, {headers:headers});
33   }
34   updateBus(bus: Bus): Observable<any>{
35     let params=JSON.stringify(bus);
36     let headers = new HttpHeaders().set('Content-Type', 'application/json');
37     return this._http.post(this.url+'modificarAutobus', params, {headers:headers});
38   }
39   deleteBus(opcion): Observable<any>{
40     let params=JSON.stringify(opcion);

```

Figura 3.60: Servicio para gestionar autobuses.

# Capítulo 4

## Resultados

Este capítulo tiene como objetivo mostrar los resultados del producto desarrollado, así como la aplicación del caso de estudio. El caso de estudio comprende un sistema para la gestión de autobuses, conductores, rutas, tarifas y pagos para un sistema de autobuses urbanos.

El procedimiento para la gestión de una empresa de este tipo debe ser muy ordenado y siempre se debe contar con información precisa y se debe de garantizar la seguridad de esta información por lo cual se hace uso para la gestión de toda la información la tecnología de *blockchain* y para la autenticación *firebase* y una *wallet* para guardar los certificados, para esto se crea un servicio de autenticación.

Mediante esta aplicación descentralizada, se puede tener un orden en el manejo de datos, así como también un resguardo de los datos muy seguro, ya que esta es una de las principales preocupaciones de las empresas y es una de las principales características que ofrece la tecnología *blockchain*.

### 4.1. Acceso al sistema

El sistema cuenta con inicio o *home*, figura 4.1, el cual cuenta con un botón para acceder al sistema, el acceso al sistema tiene un diseño práctico y agradable a la vista, figura 4.2 , cuenta con sus validaciones de formulario y un servicio de autenticación, esto garantiza la integridad de la información.

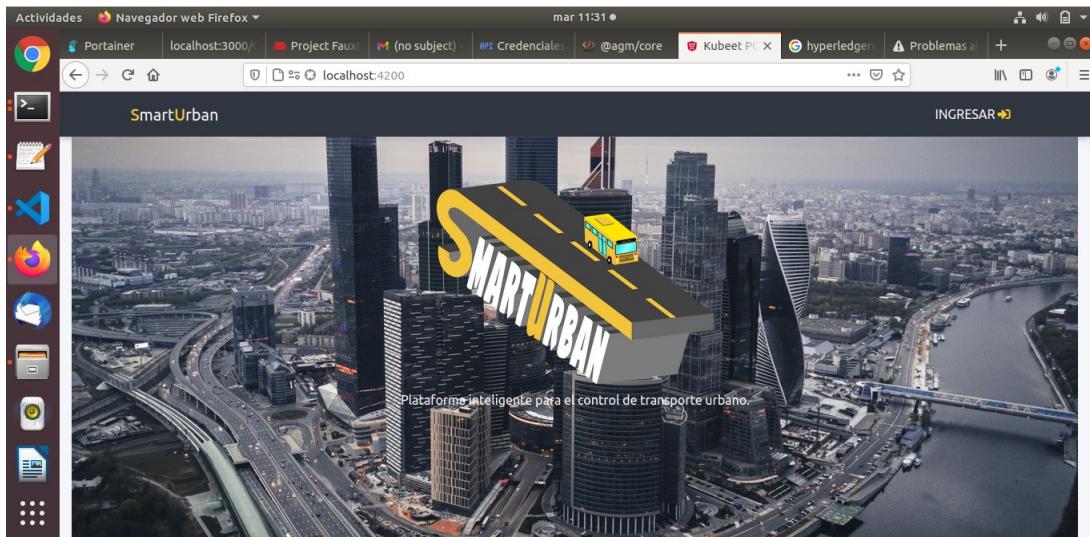


Figura 4.1: Inicio o home.

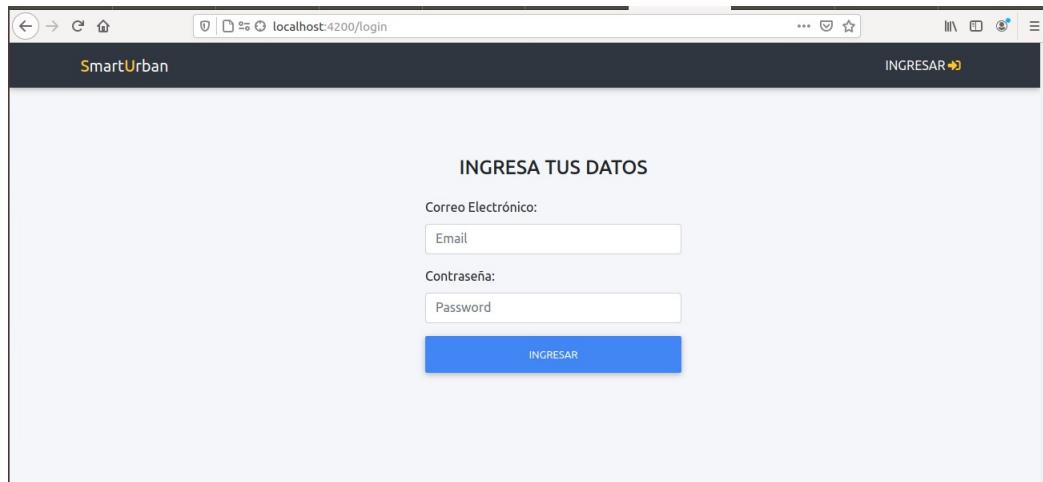


Figura 4.2: Acceso al sistema.

## 4.2. Roles en el sistema

Se definieron dos roles, uno para el administrador: este usuario podrá realizar todas las modificaciones que se requieran siempre y cuando las acciones sean válidas en el respectivo *chaincode*, dependiendo de la acción realizada se hace uso del *chaincode* y de sus reglas o funciones.

Para el rol de administrador se cuentan con seis funciones en el menú:

1. **Autobuses:** Módulo para realizar la gestión de autobuses, este módulo cuenta inicialmente con una interfaz por medio de tarjetas, figura 4.3, en la cual se listan todos los autobuses, de igual forma cuenta con las funciones de agregar autobús, eliminar y editar.

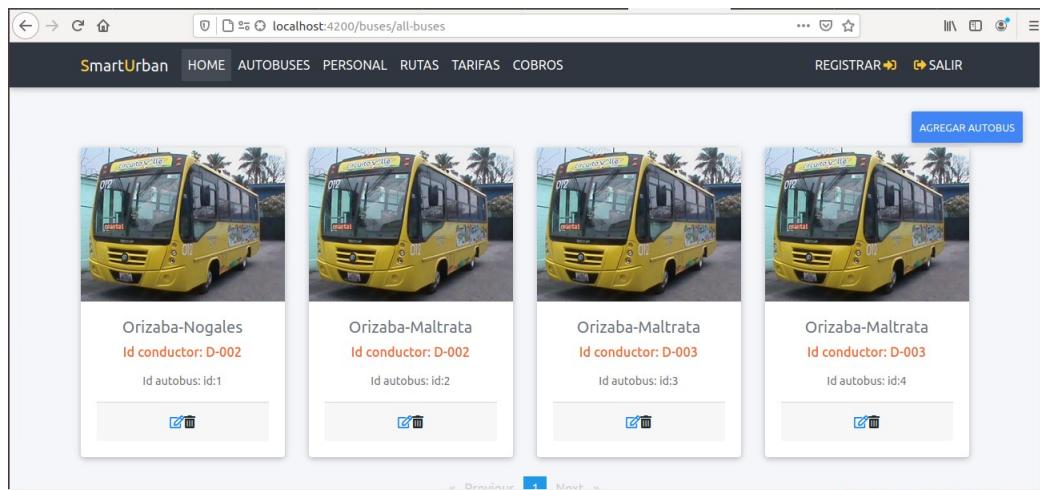


Figura 4.3: Listado de autobuses.

- **Agregar autobús:** Función para agregar un autobús, se selecciona el botón agregar autobús en la interfaz de autobuses, figura 4.3, redirigirá al formulario de autobuses, figura 4.4, este formulario cuenta con sus respectivas validaciones y el botón se encuentra deshabilitado, debido a que todos los campos son obligatorios. Los campos **id conductor** y **ruta** muestran solo los conductores y rutas que se encuentran dados de alta en el sistema. Para realizar esta acción se hace uso del *chaincode* **buscontrol** y utiliza dos métodos, uno para validar que no exista un autobús con el mismo *id* en el *ledger* y si no se encuentra un autobús con el mismo *id*, se hace uso de otra función para dar de alta el autobús en el *ledger*.
- **Modificar autobús:** Esta función muestra los detalles del autobús seleccionado, figura 4.5, si se desea modificar un dato, se modifica y se selecciona el botón actualizar, esta acción hace uso del *chaincode* y la función **busExist**

para corroborar que existe el autobús que se desea modificar y la función `UpdateBus` para actualizar los datos a modificar. También esta interfaz cuenta con un ícono para la calendarización de los autobuses .

Llena el formulario para añadir un nuevo autobús

ID de la unidad:

ID del conductor:

Descripción:

Placas del autobús:

Ciudad:

Ruta:

Figura 4.4: Agregar autobús.

Detalles del autobús

ID autobús:

Ruta:

ID conductor:

id Ruta:

Placas:

Descripción:

Figura 4.5: Modificar autobús.

Al seleccionar el ícono para la calendarización, muestra la ruta que previamente se definió para el autobús, figura 4.6, y cuenta con un botón para calendarizar.

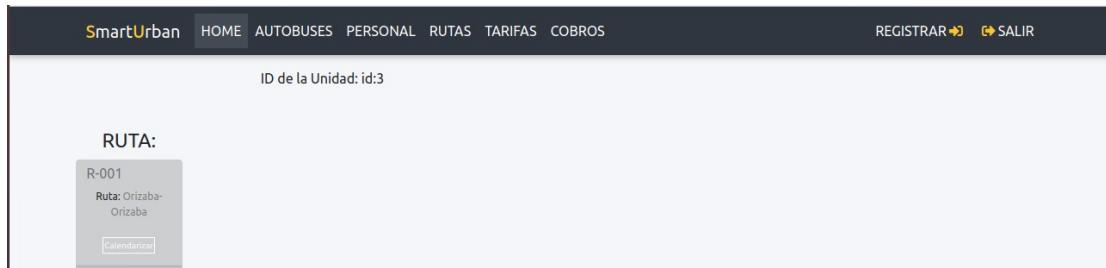


Figura 4.6: Calendarización autobús.

Para la calendarización se muestra un formulario, el cual cuenta con seis campos, figura 4.7, día de inicio, día de término, conductor, hora de salida, hora de llegada y comentarios. Esto con el objetivo de llevar un mejor control de los autobuses y conductores.

Figura 4.7: Formulario para la calendarización del autobús.

2. **Personal:** Módulo para realizar la gestión del personal (conductores), este módulo cuenta inicialmente con una interfaz en donde se listan todos los conductores, figura 4.8, de igual forma cuenta con las funciones de agregar conductor, eliminar conductor y editar conductor.

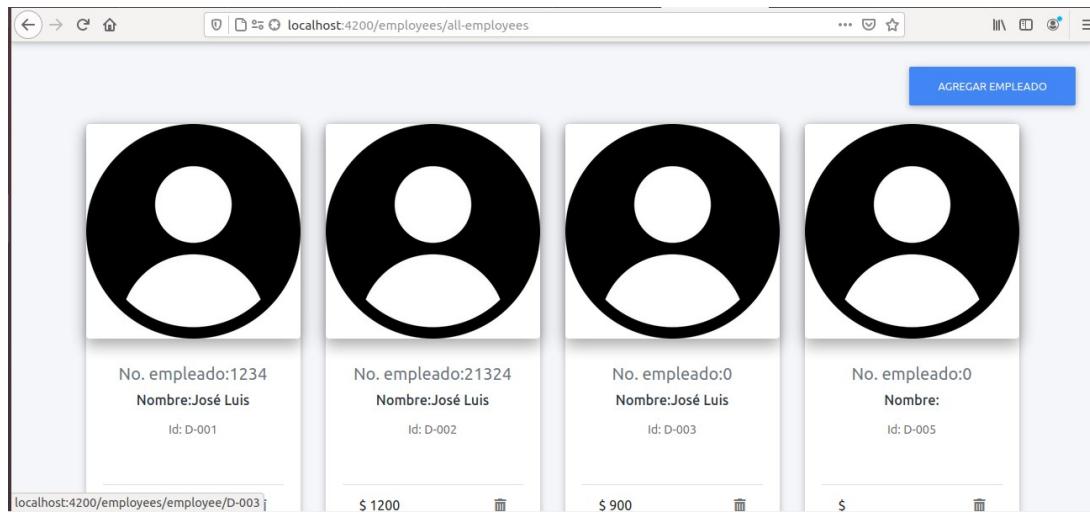


Figura 4.8: Listado de conductores.

- **Agregar empleado:** Para agregar un empleado o conductor, se selecciona el botón agregar empleado en la interfaz, figura 4.8, redirigirá al formulario para agregar un empleado, figura 4.9, este formulario cuenta con sus respectivas validaciones y el botón añadir empleado, al igual que en el formulario de autobuses se encuentra deshabilitado, debido a que todos los campos son obligatorios. Para realizar esta acción se hace uso del *chaincode drivercontrol* y utiliza dos métodos, uno para validar que no exista un conductor con el mismo *id* en el *ledger* y si no se encuentra un autobús con el mismo *id*, se hace uso de la función para dar de alta el autobús en el *ledger*, figura 4.9.

Figura 4.9: Agregar conductor.

- **Modificar conductor:** Para realizar la modificación de un conductor o empleado se debe seleccionar el nombre del conductor, esto redirige y muestra los detalles del conductor, figura 4.10, de igual manera se pueden modificar los datos del conductor y posteriormente mediante el botón actualizar empleado, se hace uso del *chaincode drivercontrol* para actualizar el conductor.

Modificar conductor	
Id	D-002
Nombre	José Luis
Apellidos	Balderas Rosas
RFC	RFCBARL950204AT3
CURP	BARL950204HVZLSS06FGG
Número de seguridad social	234321
Correo electrónico	jl_balderas@outlook.co
Departamento	Conductores
Empleador	Wily
Número de empleado	Regimen de contratación

Figura 4.10: Modificar conductor.

- **Eliminar conductor:** Para la eliminación del conductor simplemente se selecciona el ícono con forma de bote de basura, figura 4.8, este enviará al *chaincode* el *id* del conductor y validará primero las credenciales del administrador y posteriormente hace uso de las funciones **DriverExist** y **DeleteDriver**.

**3. Rutas:** Módulo para realizar la gestión de las rutas, este módulo cuenta inicialmente con una interfaz en donde se listan todas las rutas, figura 4.11, de igual forma cuenta con las funciones de agregar ruta, eliminar ruta y editar ruta.

La principal diferencia de este módulo es el uso de *google-maps* en la interfaz de editar, figura 4.12, en este componente se agregan las paradas para la ruta seleccionada, al seleccionar algún punto del mapa, agrega un marcador y muestra un campo donde se especifica el nombre de la parada y un botón para agregar el punto o la parada, al agregarlo, dentro del mapa se agrega la parada y muestra un ícono señalando el punto de la parada mediante el icono.

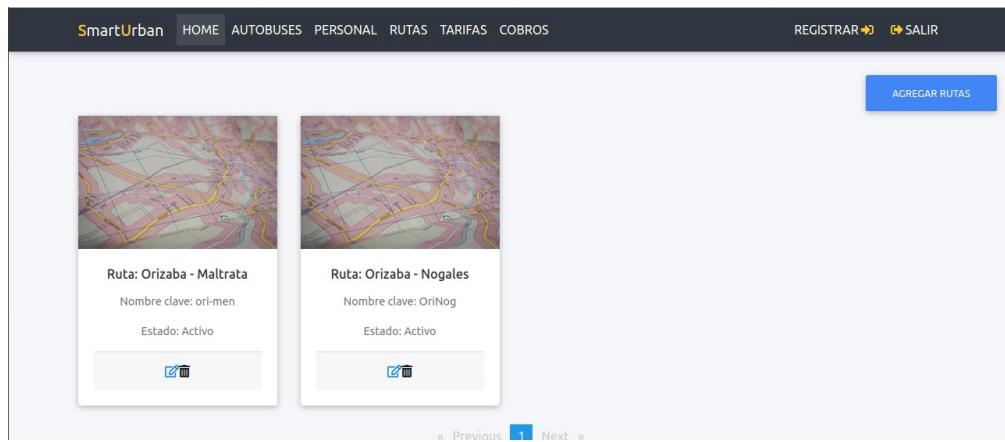


Figura 4.11: Lista de rutas.

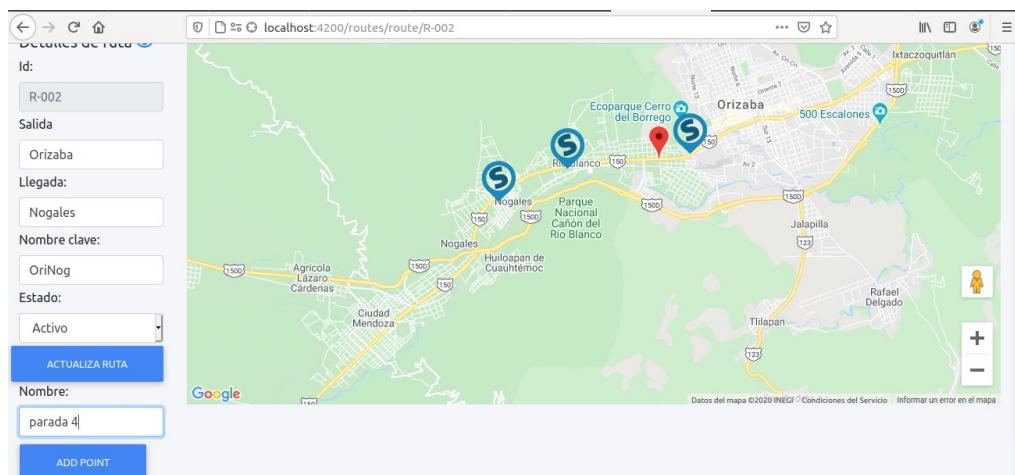


Figura 4.12: Modificar ruta y agregar parada.

4. Registro: El módulo de registro muestra un formulario en donde se debe de ingresar un usuario con formato de correo electrónico por ejemplo: `Administrador@chyc.com`, una contraseña y deberá elegir el tipo de usuario, es decir, el rol que tendrá el usuario, figura 4.13.

Al registrar el usuario, independientemente del registro que se hace en *firebase*, genera los CA, certificados de autoridad. genera un archivo con los certificados y una llave privada, para que posteriormente el usuario creado tenga la autorización de realizar acciones en la red *blockchain*. Más específicamente, estos servicios se relacionan con la inscripción de usuarios, las transacciones invocadas en la *blockchain* y las conexiones entre usuarios o componentes de la *blockchain*, figura

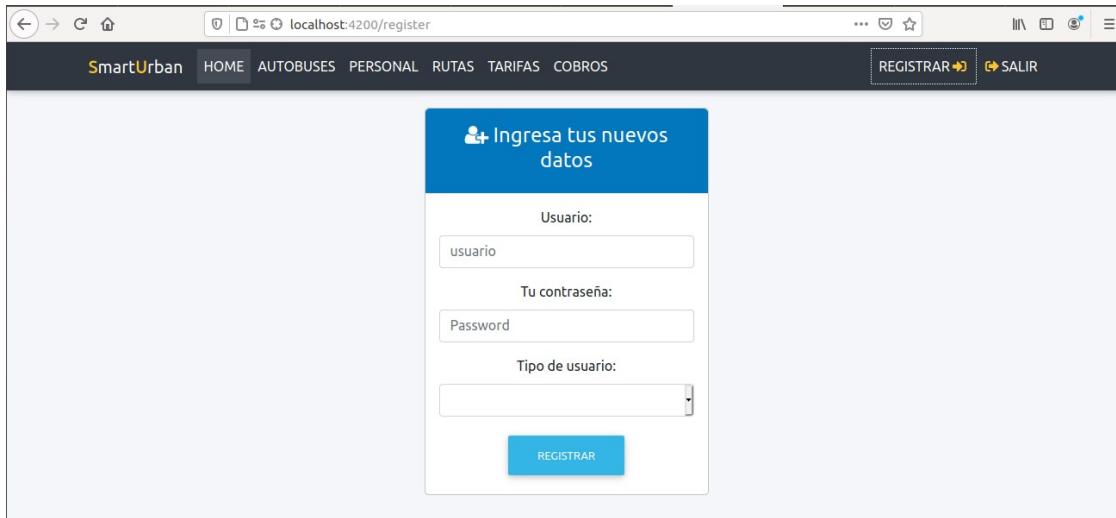


Figura 4.13: Registrar usuario.

## 4.14.



Figura 4.14: Credenciales del usuario.

5. Cobros: Módulo para realizar la gestión de los cobros realizados, este módulo cuenta con una interfaz en donde se listan los cobros por autobús, se cuenta con una lista de autobuses, para seleccionar el *id* de un autobús y muestra mediante una tabla los cobros realizados por el mismo. La tabla cuenta con un número de folio, el *id* del autobús, la ruta y el cobro, figura 4.15. Para esta acción se realiza una petición *post* y se hace uso del *chaincode payscontrol* y *busescontrol* para obtener un único autobús y todos sus cobros realizados.

6. Tarifas: Módulo para realizar la gestión de tarifas, este módulo cuenta inicialmente con una interfaz en donde se listan todas las tarifas, figura 4.16, de igual forma cuenta con las funciones de agregar tarifa, eliminar tarifa y editar tarifa,

Folio	Id Autobus	Ruta	Cobro
011	id:4	Orizaba-Nogales	13
013	id:4	Orizaba-Río blanco	12
09	id:4	Orizaba-Río blanco	12

Figura 4.15: Gestión de cobros.

las tarifas serán utilizadas posteriormente con el rol de conductor en el módulo de pagos o cobros para realizar los cobros con base en estas tarifas predefinidas.

Orizaba-Nogales Costo: \$13	Orizaba-Río blanco Costo: \$12	Orizaba-Mendoza Costo: \$15	Orizaba-Maltrata Costo: \$17
<b>Id pago de la ruta:</b> D-004	<b>Id pago de la ruta:</b> PR-001	<b>Id pago de la ruta:</b> PR-002	<b>Id pago de la ruta:</b> PR-003
Orizaba-Orizaba Costo: \$10			
<b>Id pago de la ruta:</b> PR-004			

Figura 4.16: Tarifas.

Para el rol de conductor se cuenta con una única acción en el menú:

1. **Cobros:** Este módulo como su nombre lo indica es para realizar cobros, el conductor deberá acceder al sistema, seleccionar cobros y para esto, solo se debe seleccionar la ruta, es decir, de dónde a dónde viajará el usuario, se obtendrá el costo y se realiza el pago. Como se muestra en la figura 4.17. Los cobros se enviaran al *ledger*, para que posteriormente sean gestionados por el administrador.

The screenshot shows a user interface for a payment system. At the top, there's a navigation bar with the logo 'SmartUrban' and links for 'HOME' and 'COBROS'. On the right side of the header is a 'SALIR' (Logout) button. The main area is titled 'COBROS'. It contains several input fields: 'Ruta:' with the value 'Orizaba-Nogales', 'Pago:' with the value '\$13', 'Folio:' with the value '014', and 'ID del autobus:' with the value 'id:1'. Below these fields is a blue rectangular button labeled 'PAGAR'.

Figura 4.17: Cobros.

### 4.3. Comprobación de la descentralización en la aplicación

El libro mayor almacena información fáctica importante; tanto el valor actual de los atributos de los objetos como el historial de transacciones que dieron como resultado estos valores actuales. Como anteriormente ya se había mencionado el *ledger* o libro mayor consta de dos partes: un estado mundial y una cadena de bloques. Cada uno de ellos representa un conjunto de hechos sobre un conjunto de objetos.

- Estado mundial: una base de datos que contiene los valores actuales de un conjunto de estados del libro mayor. Dado que es una red descentralizada, en cada nodo se tiene la réplica de cada base de datos, es decir, en cada nodo se tienen cinco bases de datos, figura 4.18, para realizar la gestión del sistema de autobuses urbanos.
  - `autobuses_buscontrol`.
  - `autobuses_drivercontrol`.
  - `autobuses_payoutescontrol`.
  - `autobuses_payscontrol`.
  - `autobuses_routecontrol`.

Databases	Database name	Create Database	{ } JSON	Book
autobuses__recyclebin_impl cit_org_\$.org3\$m\$\$p	38.1 KB	90	No	
autobuses__lifecycle\$p_impl cit_org_\$.org1\$m\$\$p	37.3 KB	90	No	
autobuses_buscontrol	25.8 KB	4 1	No	
autobuses_drivercontrol	15.3 KB	7	No	
autobuses_lscc	0 bytes	0	No	
autobuses_payoutescontrol	2.1 KB	5	No	
autobuses_payscontrol	3.3 KB	5	No	
autobuses_routecontrol	2.3 KB	2	No	
fabric__internal	291 bytes	1	No	

Figura 4.18: Bases de datos.

Como ya se ha mencionado dado que nuestra red se configuró con tres nodos, en cada nodo debemos tener las bases de datos replicadas, para verificar esto se accede al *couchDB* de cada nodo ubicados en `localhost:5984`, `localhost:5985` y `localhost:5986` respectivamente, figuras 4.19, 4.20 y 4.21.

Databases	Database name	Create Database	{ } JSON	Book
autobuses__recyclebin_impl cit_org_\$.org3\$m\$\$p	38.1 KB	90	No	
autobuses__lifecycle\$p_impl cit_org_\$.org1\$m\$\$p	37.3 KB	90	No	
autobuses_buscontrol	25.8 KB	4 1	No	
autobuses_drivercontrol	15.3 KB	7	No	
autobuses_lscc	0 bytes	0	No	
autobuses_payoutescontrol	2.1 KB	5	No	
autobuses_payscontrol	3.3 KB	5	No	
autobuses_routecontrol	2.3 KB	2	No	
fabric__internal	291 bytes	1	No	

Figura 4.19: couchDB organización 1.

Al tener un sistema descentralizado se pueden obtener grandes ventajas y una de ellas es: si algún nodo no estuviera activo, es decir, para este caso se tiene la Org1, la Org2 y la Org3, y si la Org3 no está activa, seguiríamos teniendo acceso a los datos, de igual manera se ejemplifica un caso o se probó eliminar una serie de datos en el nodo dos desde *couchDB*, figura 4.22, como se logra observar solo se cuenta con un activo.

Database name	Size	Document Count	Status
cit_org_<redacted>	0 bytes	0	No
autobuses.lifecycle\$h.impli	38.1 KB	90	No
autobuses.lifecycle\$p.impli	25.8 KB	4	No
autobuses_buscontrol	15.3 KB	7	No
autobuses_drivercontrol	0 bytes	0	No
autobuses_lscc	2.1 KB	5	No
autobuses_payoutescontrol	3.2 KB	1	No
autobuses_payscontrol	2.3 KB	2	No

Figura 4.20: couchDB organización 2.

Database name	Size	Document Count	Status
autobuses.lifecycle\$h.impli	38.2 KB	90	No
autobuses.lifecycle\$p.impli	35.3 KB	90	No
autobuses_buscontrol	25.8 KB	4	No
autobuses_drivercontrol	15.3 KB	7	No
autobuses_lscc	0 bytes	0	No
autobuses_payoutescontrol	2.1 KB	5	No
autobuses_payscontrol	2.5 KB	5	No
autobuses_routecontrol	2.3 KB	2	No
fabric_internal	291 bytes	1	No

Figura 4.21: couchDB organización 3.

id	key	value
01	01	{ "rev": "1-541aeac6dbbd5b90cf55..."}

Figura 4.22: base de datos autobuses\_payscontrol nodo 2.

Sin embargo, como no hubo un consenso, es decir, un acuerdo por parte de todas las organizaciones o nodos, en el resto de las organizaciones siguen todos los activos, figura

## 4.23.

The screenshot shows the CouchDB interface at `localhost:5984/_utils/#database/autobuses_payscontrol/_all_docs`. The left sidebar has sections for 'All Documents', 'Run A Query with Mango', 'Permissions', 'Changes', and 'Design Documents'. The main area displays a table with four documents:

	id	key	value
<input type="checkbox"/>	01	01	{ "rev": "1-541aeac6dbbd5b90cf55...
<input checked="" type="checkbox"/>	1	1	{ "rev": "1-97938a9122b000fe3890...
<input type="checkbox"/>	2	2	{ "rev": "1-0234c7cb914119164c1d...
<input type="checkbox"/>	3	3	{ "rev": "1-bee0063b0b34705c59b...
<input type="checkbox"/>	4	4	{ "rev": "4-50d6a5056b336fd75ecb...

Figura 4.23: base de datos autobuses\_payscontrol nodo 1.

De igual manera si se consultan todos los pagos desde el servidor de la API, se observa que devuelve todos los datos, figura 4.24.

The screenshot shows a browser window at `localhost:3000/ConsultarCobros`. The page displays a JSON array of payment records:

```
[ { "Key": "01", "Pay": { "idBus": "id-1", "clave": "", "pago": "$13" } }, { "Key": "1", "Pay": { "idBus": "id-1", "clave": "Orizaba-Rio blanco", "pago": "10" } }, { "Key": "2", "Pay": { "idBus": "id-2", "clave": "Orizaba-Nogales", "pago": "$12" } }, { "Key": "3", "Pay": { "idBus": "id-1", "clave": "", "pago": "1" } }, { "Key": "4", "Pay": { "idBus": "id-1", "clave": "Orizaba-Nogales", "pago": "$12" } } ]
```

Figura 4.24: Consulta de pagos mediante la API

Como se observa una aplicación descentralizada ofrece la ventaja de tener los datos en "N" nodos, para este caso se cuentan con tres nodos y las bases se replican en todos los nodos, de igual manera si eliminaran datos directamente en *CouchDB* en uno o más nodos, con el hecho de que un nodo tenga los datos, se podrá seguir teniendo acceso a los datos debido a su descentralización, entre más nodos haya más descentralizada será la aplicación.

# **Capítulo 5**

## **Conclusiones y recomendaciones**

En este capítulo se muestran las conclusiones y recomendaciones para este trabajo de tesis.

### **5.1. Conclusiones**

El desarrollo del presente trabajo de tesis se basó en la tecnología *blockchain*. Sin embargo, se hace uso de diversas tecnologías con el fin de obtener una aplicación descentralizada robusta y completa, logrando la seguridad necesaria y la transparencia de datos.

En función de la problemática a solucionar, se llevó a cabo la búsqueda de trabajos relacionados con el fin de conocer sus alcances y limitaciones de los artefactos para el desarrollo de software orientado a *blockchain* (BOS). Esto ayudó a tener una mejor visión para el desarrollo del producto requerido, permitiendo obtener un panorama más amplio mediante el modelado, haciendo uso de los artefactos que mejor se adaptan a este nuevo paradigma de software.

La aplicación descentralizada basada en *blockchain* desarrollada conlleva tres fases: planificación-definición, desarrollo, entrega-mantenimiento. Cada fase es de suma importancia, dado que es un sistema que conlleva el uso de diversas tecnologías y pasos

para la obtención completa del sistema, desde el modelado, la configuración de la red, la generación de artefactos para la red *blockchain*, la codificación de los *chaincodes* en *Go*, la creación del canal, la instalación del *chaincode*, la codificación de la API, la codificación del *front-end* en Angular, hasta la integración de todo el sistema.

En conclusión, la importancia de tener un proceso de desarrollo y saber o conocer qué artefactos ayudarán o aportarán ventajas para el desarrollo es de suma importancia, ya que con esto se asegura una mayor productividad en el desarrollo y provoca la reducción de errores de codificación. Asimismo se garantiza un sistema de calidad y robusto, asegurando la seguridad de los datos, siendo la característica principal por la cual se usa la tecnología *blockchain*.

## 5.2. Trabajos a futuro

Respecto a la tecnología *blockchain* empleada, cabe resaltar que es una tecnología válida para su uso en diversos entornos reales. Por lo cual se sugiere el uso de *blockchain Hyperledger Fabric* en un entorno 100 % real y activo, es decir, un sistema desplegado y trabajando en algún entorno y sobre este sistema llevar a cabo la integración de *blockchain*. De igual manera, otra de las posibilidades de trabajo a futuro sería crear un mecanismo para la evaluación de la seguridad de un sistema *blockchain* desplegado y trabajando en un entorno real.

# Apéndice A

## Pruebas con la plataforma

**Crear instancia** Para iniciar la instalación de los requisitos y dar de alta una red de *Hyperledger Fabric* se crea una instancia de máquina virtual de Linux en *Compute Engine* con Google *Cloud Platform Console*, figura A.1.

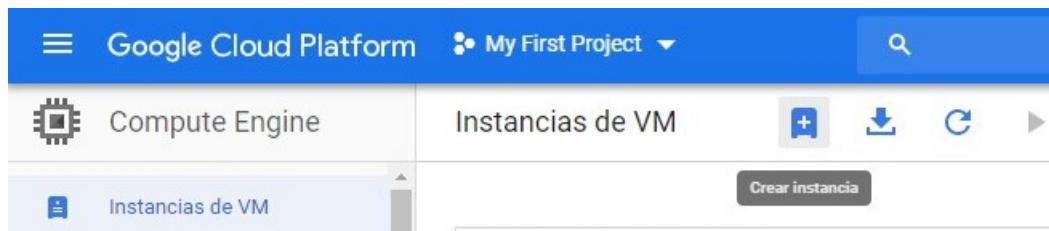


Figura A.1: Crear instancia.

Para la creación de la instancia se debe configurar el disco de arranque, seleccionando el sistema operativo a utilizar, cantidad de memoria en disco, entre otros. Se inicia el proceso de creación, figura A.2, y finalmente se crea e inicia la máquina virtual, figura A.3.



Filtrar las instancias de VM							Columns ▾
<input type="checkbox"/>	Nombre ^	Zona	Recomendación	Usada por	IP interna	IP externa	Conectar
<input checked="" type="checkbox"/>	blockchain	us-central1-a			10.128.0.2 (nic0)	35.202.90.133	SSH ▾
	blockchain1				Ninguna		⋮

Figura A.2: Instancia creada.

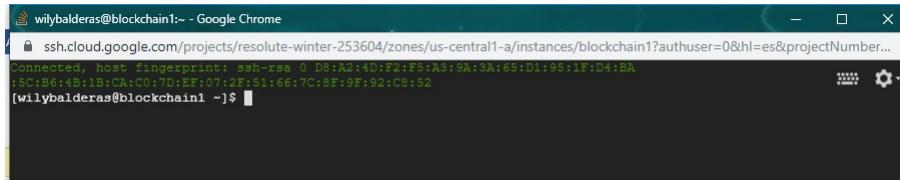


Figura A.3: Inicio de máquina virtual CentOS 7.

**Actualización de CentOS.** Aunque el paquete *Docker* está disponible en el repositorio oficial de CentOS 7, que no siempre es la última versión. El enfoque recomendado es instalar *Docker* desde los repositorios de *Docker*.

1. La figura A.4 muestra la actualización de los paquetes del sistema y se instalan las dependencias requeridas:

**Comando actualización de CentOS: sudo yum update**

2. El sistema proporciona una lista de los paquetes que se descargarán, figura A.5, así como el espacio en disco requerido. El sistema pide una confirmación para descargar los paquetes, el sistema los descarga y realiza la instalación.

3. La figura A.6 muestra la instalación de una serie de paquetes previos.

**Comando: Sudo yum install -y yum-utils device-mapper-persistent-data lvm2.**

```
wilybalderas@blockchain1:~ - Google Chrome
ssh.cloud.google.com/projects/resolute-winter-253604/zones/us-central1-a/instances/blockchain1?authuser=0&hl=es&projectNumber...
[wilybalderas@blockchain1 ~]$ sudo yum update
Loaded plugins: fastestmirror
Determining fastest mirrors
epel/x86_64/metalink
 * base: yum.tamu.edu
 * epel: d2lzk17pfhg30w.cloudfront.net
 * extras: mirror.vcu.edu
 * updates: mirror.oss.ou.edu
base
epel
Installing for dependencies:
bind-export-libs           x86_64      32:9.11.4-9.P2.el7          base          1.1 M
geoipupdate                x86_64      2.5.0-1.el7                  base          35 k
Transaction Summary
=====
Install   1 Package  (+2 Dependent packages)
Upgrade  160 Packages

Total download size: 303 M
Is this ok [y/d/N]:
```

Figura A.4: Actualización de CentOS 7.

4. Se utilizan repositorios propios de *Docker*.

**Comando:** `sudo yum-config-manager –add-repo https://download.docker.com/linux/centos/docker-ce.repo`. Esto hará que aparezca un archivo con los repositorios añadidos en `/etc/yum.repos.d/docker-ce.repo`

**Docker CE** Es una aplicación que permite crear contenedores ligeros y portables (contienen elementos necesarios para que una aplicación execute), para que las aplicaciones de software sean capaces de ejecutarse en cualquier máquina con *Docker* instalado, independientemente del sistema operativo, facilitando los despliegues.

### Instalación de *Docker CE*

A continuación se muestran los pasos para la instalación de docker CE.

1. La figura A.7 muestra la instalación de la última versión de Docker CE (Community Edition).

**Comando:** `Sudo yum install -y docker-ce`

```
wilybalderas@blockchain1:~ - Google Chrome
ssh.cloud.google.com/projects/project-id-winter-253604/zones/us-central1-a/instances/blockchain1?authuser=0&hl=en

plymouth-core.x86_64 0:0.0.9-0.32.20140113.e17.centos
plymouth-scripts.x86_64 0:0.0.9-0.32.20140113.e17.centos
policycoreutils.x86_64 0:2.5-33.el7
policycoreutils-python.x86_64 0:2.5-33.el7
polkit.x86_64 0:0.112-22.e17_7.1
polkit-gnome.x86_64 0:0.112-22.e17_7.1
python.x86_64 0:2.7.5-86.el7
python-chardet.noarch 0:2.1.2-3.el7
python-firewall.noarch 0:0.6.3-2.el7_7.1
python-langs.x86_64 0:2.7.5-86.el7
python-linux-protocols.noarch 0:0.4.11-4.el7
python-perf.x86_64 0:3.13.1-252.el7_1.2.el7
python-pip.x86_64 0:9.0.3-0.el7_5-1
python-setuptools.x86_64 0:1.10.2-7.el7
gemu-quest-agent.x86_64 10:2.12.0-3.el7
readline.x86_64 0:6.2-11.el7
rpm.x86_64 0:4.11.3-40.el7
rpm-build-libs.x86_64 0:4.11.3-40.el7
rpm.x86_64 0:4.11.3-40.el7
rsync.x86_64 0:3.1.1-34.el9
rsync.pylog.x86_64 0:8.24.0-41.el7_7
selinux-policy.noarch 0:3.13.1-252.el7.1
selinux-policy-targeted.noarch 0:3.13.1-252.el7.1
sg3_utils.x86_64 0:1.37-18.el7_7.1
sg3_utils-libs.x86_64 0:1.37-18.el7_7.1
shadow-utils.x86_64 2:4.6-5.el7
smrsh.x86_64 0:1.2-1.el7
systemd.x86_64 0:219-67.el7_7.1
systemd-libs.x86_64 0:219-67.el7_7.1
systemd-sysv.x86_64 0:219-67.el7_7.1
teamd.x86_64 0:1.27-9.el7
tuned.noarch 0:11.11.0-5.el7_7.1
tzdata.noarch 0:2019c-1.el7
util-linux.x86_64 2:3.19-1.el7_37
vconsole-common.x86_64 2:7.4.629-6.el7
vim-enhanced.x86_64 2:7.4.629-6.el7
vim-fsmonitor.x86_64 2:7.4.629-6.el7
vim-minimal.x86_64 2:7.4.629-6.el7
xfsprogs.x86_64 0:4.5.0-20.el7
yum.noarch 0:3.4.3-163.el7.centos
yum-cron.noarch 0:3.4.3-163.el7.centos
yum-plugin-fastestmirror.noarch 0:1.1.31-52.el7

Complete! [wilybalderas@blockchain1 ~]$
```

Figura A.5: Paquetes del sistema.

## 2. Administrar Docker como usuario no root.

*The Docker daemon* se une a un *socket* Unix en lugar de un puerto TCP. Por defecto, el *socket* de Unix es propiedad del usuario *root* y otros usuarios solo pueden acceder a él usando *sudo*. *The Docker daemon* siempre se ejecuta como el *root* usuario.

Si no se desea introducir el *Docker* comando como prefacio sudo, se crea un grupo Unix llamado *Docker* y se agregan los usuarios. Cuando se inicia *the Docker daemon*, crea un *socket* Unix accesible para los miembros del grupo *Docker* [26].

**Nota:** El *Docker* grupo otorga privilegios equivalentes al *root* usuario.

3. Para crear el grupo *Docker* y agregar su usuario:

Comando crear el grupo *Docker*: sudo groupadd docker.

#### 4. Agregar usuario al *Docker* grupo.

Comando: sudo usermod -aG docker \$USER. Figura A.8

**Control Docker con systemd** Algunas distribuciones de Linux usan systemd para iniciar *the Docker daemon*. Se muestran algunos ejemplos de cómo personalizar la configuración de *Docker* [26].

```
[wilybalderas@blockchain1 ~]$ sudo yum install -y yum-utils device-mapper-persistent-data lvm2
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: yum.tamu.edu
 * epel: d2lzk17pfhq30w.cloudfront.net
 * extras: mirror.vcu.edu
 * updates: mirror.os.ou.edu
Resolving Dependencies
--> Running transaction check
--> Package device-mapper-persistent-data.x86_64 0:0.8.5-1.el7 will be installed
--> Processing Dependency: libaio.so.1(LIBAIO_0.4) (64bit) for package: device-mapper-persistent-data-0.8.5-1.el7.x86_64
--> Processing Dependency: libaio.so.1(LIBAIO_0.1) (64bit) for package: device-mapper-persistent-data-0.8.5-1.el7.x86_64
--> Processing Dependency: libaio.so.1() (64bit) for package: device-mapper-persistent-data-0.8.5-1.el7.x86_64
--> Package lvm2.x86_64 7:2.02.185-2.el7 will be installed
--> Processing Dependency: lvm2-libs = 7:2.02.185-2.el7 for package: 7:lvm2-2.02.185-2.el7.x86_64
--> Processing Dependency: liblvm2app.so.2.2(Base) (64bit) for package: 7:lvm2-2.02.185-2.el7.x86_64
--> Processing Dependency: libdevmapper-event.so.1.02(Base) (64bit) for package: 7:lvm2-2.02.185-2.el7.x86_64
--> Processing Dependency: liblvm2app.so.2.2() (64bit) for package: 7:lvm2-2.02.185-2.el7.x86_64
--> Processing Dependency: libdevmapper-event.so.1.02() (64bit) for package: 7:lvm2-2.02.185-2.el7.x86_64
--> Package yum-utils.noarch 0:1.1.31-52.el7 will be installed
--> Processing Dependency: python-kitchen for package: yum-utils-1.1.31-52.el7.noarch
--> Processing Dependency: libxml2-python for package: yum-utils-1.1.31-52.el7.noarch
--> Running transaction check
--> Package device-mapper-event-libs.x86_64 7:1.02.158-2.el7 will be installed
--> Package libaio.x86_64 0:0.3.109-13.el7 will be installed
--> Package libxml2-python.x86_64 0:2.9.1-6.el7_2.3 will be installed
--> Package lvm2-libs.x86_64 7:2.02.185-2.el7 will be installed
--> Processing Dependency: device-mapper-event = 7:1.02.158-2.el7 for package: 7:lvm2-libs-2.02.185-2.el7.x86_64
--> Package python-kitchen.noarch 0:1.1.1-5.el7 will be installed
--> Running transaction check
--> Package device-mapper-event.x86_64 7:1.02.158-2.el7 will be installed
--> Finished Dependency Resolution
Dependencies Resolved
```

Figura A.6: Instalación de paquetes.

5. Iniciar *the Docker daemon* Una vez que *Docker* está instalado, se debe iniciar *the Docker daemon*. La mayoría de las distribuciones de Linux usan el comando *systemctl* para iniciar servicios. Si no tiene *systemctl*, se usa el comando *service* [26].

#### Comandos:

*systemctl*: sudo systemctl start docker

*service*: sudo service docker start

6. Verifica que se puede ejecutar comandos *Docker* sin el comando *sudo*.

#### Comando: docker run hello-world

Se descarga una imagen de prueba y se ejecuta en un contenedor. Cuando se ejecuta el contenedor, imprime un mensaje.

Si inicialmente se ejecutaron los comandos de *Docker CLI* usando *sudo* antes de agregar el usuario al grupo *Docker*, se genera el siguiente error, que indica que el */.docker/* directorio se creó con permisos incorrectos debido a que se utilizaron comandos *sudo*.

**WARNING: Error loading config file: /home/user/.docker/config.json-stat/  
home/user/.docker/config.json: permission denied**

```

Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
* base: yum.tamu.edu
* epel: dlitzk17phg30w.cloudfront.net
* extras: mirror.vcu.edu
* updates: mirror.oss.cu.edu
docker-ce-stable
(1/2) docker-ce-stable/x86_64/updateinfo | 3.5 kB 00:00:00
(2/2) docker-ce-stable/x86_64/primary_db | 55 B 00:00:00
| 34 kB 00:00:00
Resolving Dependencies
--> Running transaction check
-> Package docker-ce.x86_64 3:19.03.2-3.el7 will be installed
-> Processing Dependency: container-selinux >= 2:2.74 for package: 3:docker-ce-19.03.2-3.el7.x86_64
-> Processing Dependency: containerd.io >= 1.2.2-3 for package: 3:docker-ce-19.03.2-3.el7.x86_64
-> Processing Dependency: docker-ce-cli for package: 3:docker-ce-19.03.2-3.el7.x86_64
-> Running transaction check
--> Package container-selinux.noarch 2:2.107-3.el7 will be installed
--> Package containerd.io.x86_64 0:1.2.6-3.3.el7 will be installed
--> Package docker-ce-cli.x86_64 1:19.03.2-3.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

-----
Package           Arch      Version            Repository      Size
-----
Installing:
docker-ce          x86_64   3:19.03.2-3.el7      docker-ce-stable 24 M
Installing for dependencies:
container-selinux    noarch   2:2.107-3.el7        extras          39 k
containerd.io       x86_64   1.2.6-3.3.el7      docker-ce-stable 26 M
docker-ce-cli       x86_64   1:19.03.2-3.el7      docker-ce-stable 39 M
-----
Transaction Summary
-----
Install 1 Package (+3 Dependent packages)

Total download size: 90 M
Installed size: 368 M
Running transaction
  Installing : 3:container-selinux-2.107-3.el7.noarch
  Installing : 3:containerd.io-1.2.6-3.3.el7.x86_64
  Installing : 3:docker-ce-19.03.2-3.el7.x86_64
  Verifying  : 3:container-selinux-2.107-3.el7.noarch
  Verifying  : 3:containerd.io-1.2.6-3.3.el7.x86_64
  Verifying  : 3:docker-ce-19.03.2-3.el7.x86_64
  1/4
  2/4
  3/4
  4/4
  1/4
  2/4
  3/4
  4/4
Installed:
  docker-ce.x86_64 3:19.03.2-3.el7

Dependency Installed:
  container-selinux.noarch 2:2.107-3.el7
                                         containerd.io.x86_64 0:1.2.6-3.3.el7
                                         docker-ce-cli.x86_64 1:19.03.2-3.el7

Complete!

```

Figura A.7: Instalación de Docker.

```

[willybalderas@blockchain1 ~]$ sudo usermod -aG docker willybalderas
[willybalderas@blockchain1 ~]$ 

```

Figura A.8: Comando para agregar usuario al grupo.

Para solucionar este problema, eliminar el `/.docker/` directorio (se vuelve a crear automáticamente, pero se pierde la configuración personalizada) o cambiar la propiedad y permisos.

### Comandos:

```

sudo chown "$USER":"$USER/home/"$USER/.docker -R
sudo chmod g+rwx "$HOME/.docker" R

```

Para que la membresía de grupo sea reevaluada se cierra sesión e inicia nuevamente.

### Instalación de *Compose*

Se necesitan los siguientes paquetes de dependencia: py-pip, python-dev, libffi-dev,

openssl-dev, gcc, libc-dev, y make.

1. descargar la versión estable actual de *Docker Compose*.

**Comando:** `sudo curl ?L "https://github.com/docker/compose/releases/download/1.23.2/docker-compose-$(uname -s)-$(uname -m)o /usr/local/bin/docker-compose`

2. Aplicar permisos ejecutables al binario.

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

**Nota:** Si el comando *docker-compose* falla después de la instalación, verificar la ruta.

También se puede crear un enlace simbólico en el directorio /usr/bin o cualquier otro directorio en la ruta.

**Comando:** `sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose`

Opcionalmente, instalar finalización del comando para el *bash* y *zshconcha*. Colocar el *Script* de finalización en el directorio /etc/bash\_completion.d/

**Comando:** `sudo curl -L https://raw.githubusercontent.com/docker/compose/1.24.1/contrib/completion/bash/docker-compose -o /etc/bash_completion.d/docker-compose.`

3. Probar la instalación, figura A.9.

**Comando:** `docker-compose-version`

```
[wilybalderas@blockchain1 ~]$ docker-compose --version
docker-compose version 1.23.2, build 1110ad01
[wilybalderas@blockchain1 ~]$
```

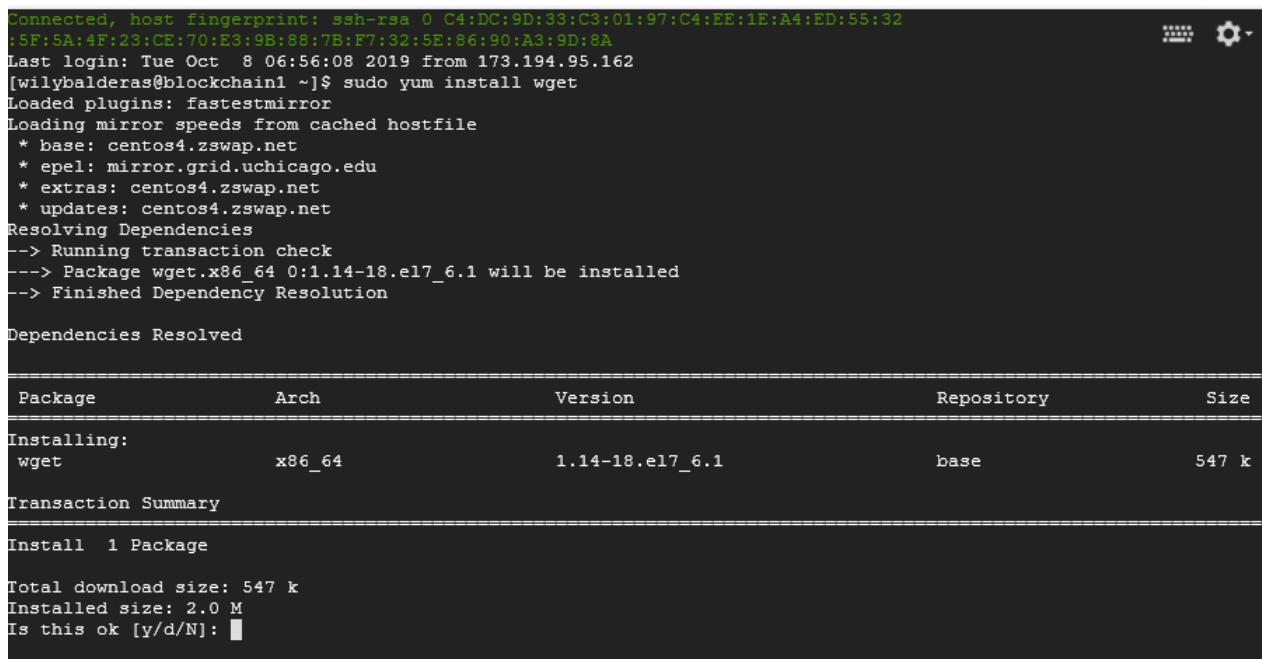
Figura A.9: Prueba de instalación docker-compose.

**WGET** WGET es una utilidad de línea de comandos para descargar archivos de la web. Con WGET, se descargan archivos usando los protocolos HTTP, HTTPS y FTP.

WGET viene preinstalado en la mayoría de las distribuciones de Linux, para verificar si el WGET está instalado en el sistema:

1. Abrir consola.
2. Escribir WGET.
3. Si se tiene instalado WGET, el sistema imprimirá WGET: *missing URL*, de lo contrario, imprimirá WGET *command not found*.
4. Si WGET no está instalado, se deberá instalar el administrador de paquetes de la distribución, figura A.10.

**Comando: sudo yum install wget**



```

Connected, host fingerprint: ssh-rsa 0 C4:DC:9D:33:C3:01:97:C4:EE:1E:A4:ED:55:32
:5F:5A:4F:23:CE:70:E3:9B:88:7B:F7:32:5F:86:90:A3:9D:8A
Last login: Tue Oct  8 06:56:08 2019 from 173.194.95.162
[wilybalderas@blockchain1 ~]$ sudo yum install wget
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: centos4.zswap.net
 * epel: mirror.grid.uchicago.edu
 * extras: centos4.zswap.net
 * updates: centos4.zswap.net
Resolving Dependencies
--> Running transaction check
--> Package wget.x86_64 0:1.14-18.el7_6.1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
| Package           | Arch      | Version        | Repository | Size   |
|:-----|:-----|:-----|:-----|:-----|
| Installing:      |          |               |            |        |
| wget              | x86_64   | 1.14-18.el7_6.1 | base      | 547 k |
| Transaction Summary |          |               |            |        |
| Install 1 Package |          |               |            |        |
| Total download size: 547 k |          |               |            |        |
| Installed size: 2.0 M |          |               |            |        |
| Is this ok [y/d/N]: |          |               |            |        |
=====
```

Figura A.10: Instalación de WGET.

## Instalación de *Go*

A continuación se muestran los pasos instalar *Go*.

1. Descargar el binario de Go . Comando: wget <https://dl.google.com/go/go1.13.linux-amd64.tar.gz>

2. Extraer el tarball en el directorio usr/local mediante el comando tar.

**Comando:** `sudo tar -C /usr/local -xzf go1.12.9.linux-amd64.tar.gz`

3. Ajustar la variable de ruta. Para indicarle al sistema dónde encontrar los binarios ejecutables de *Go* ajustando la variable de entorno.

**Comando:** `vi .bas_profile`

`PATH=$PATH:$HOME/.local/bin:$HOME/bin:/usr/local/go/bin`

Probar instalación de *Go*.

1. Se crea un directorio para el espacio de trabajo.

**Commandos:**

- `cd`
- `mkdir go`
- `cd go`
- `mkdir src`
- `cd src`
- `mkdir hello`
- `cd hello`

2. Se crea un archivo `.go`

**comando:** `vi hello.go`

3. Se realiza un **hello world!** mediante el siguiente código mostrado en la figura A.11.

```
# .bash_profile
# .bash_profile
package main
package main
import "fmt"

func main() {
    fmt.Printf("hello go world ! \n")
}
```

Figura A.11: Código ?hello world? en Go.

4. Se compila el programa

**Comando:** go build

5. se ejecuta el programa, figura A.12 **Comando:** ./hello

```
[wilybalderas@blockchain1 hello]$ go build
[wilybalderas@blockchain1 hello]$ ./hello
hello go world !
[wilybalderas@blockchain1 hello]$ █
```

Figura A.12: Ejecución de ?hello? en Go.

**Instalación de *Fabric Samples*** A continuación se muestran los pasos para instalar *Fabric Samples* y posteriormente construir la red.

1. Se necesita de un sistema de control de versiones por lo cual se instala *Git*.

**Comando:** sudo yum install git.

2. Se autoriza la descarga y se instalan los archivos correspondientes.

3. Se verifica la versión de *Git*, figura A.13.

**Comando:** git --version

```
[wilybalderas@blockchain1 hello]$ git --version
git version 1.8.3.1
[wilybalderas@blockchain1 hello]$ █
```

Figura A.13: Verificación de versión Git.

4. Si es necesario, se clona el repositorio hyperledger/fabric-samples

**Comando:** `git clone https://github.com/hyperledger/fabric-samples.git`

5. Se ingresa al directorio *fabric-samples*.

**Comando:** `cd fabric-samples`

6. En el directorio donde se instalarán los *Fabric samples* y los binarios, desplegar los archivos binarios e imágenes. Nota: Si se desea la última versión de producción, se omiten todos los identificadores de versión.

**Comando:** `curl -sSL http://bit.ly/2ysbOFE — bash -s`

El comando anterior descarga y ejecuta un *Script* de *bash* que descarga y extrae todos los binarios específicos de la plataforma que necesita para configurar la red y colocarlos en el repositorio clonado que creó anteriormente. Recupera los siguientes binarios específicos de la plataforma:

- configtxgen,
- configtxlator,
- cryptogen,
- discover,
- idemixgen
- orderer,
- peery
- fabric-ca-client

Los coloca en el subdirectorio bin del directorio de trabajo actual.

7. Si se desea configurar la variable del entorno se puede realizar mediante el siguiente

**Comando:** `vi .bash_profile`

```
PATH=$PATH:$HOME/.local/bin:$HOME/bin:/usr/local/go/bin:/home/adsoft_research/fabric-samples/bin}
```

# Bibliografía

- [1] M. Marchesi, L. Marchesi, y R. Tonelli. An Agile Software Engineering Method to Design Blockchain Applications. Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia on ZZZ - CEE-SECR'18, Moscow, Russian Federation, pages 1-8, 2018
- [2] L. V. Kiran, R. B. Dinakar, y P. S. Prasad. Blockchain Technology - A Sturdy Protective Shield. International Journal of Recent Technology and Engineering (IJR-TE), pages 1-4, 2018
- [3] B. A. Tama, B. J. Kweka, Y. Park, y K.-H. Rhee. A critical review of blockchain and its current applications. International Conference on Electrical Engineering and Computer Science (ICECOS), pp. 109-113, 2017.
- [4] M. Beller y J. Hejderup. Blockchain-based Software Engineering. Proceedings of the 41st International Conference on Software Engineering: New Ideas and Emerging Results, Piscataway, NJ, USA, pp. 53–56, 2019.
- [5] W. Cai, Z. Wang, J. B. Ernst, Z. Hong, C. Feng, y V. C. M. Leung. Decentralized Applications: The Blockchain-Empowered Software System, IEEE Access, 2018.
- [6] Lisk Documentation — Getting Started. ”<https://lisk.io/documentation/start>. [Accedido: 01-oct-2019].
- [7] Y.-C. Chen, Y.-P. Chou, y Y.-C. Chou. An Image Authentication Scheme Using Merkle Tree Mechanisms, Future Internet, jul. 2019.
- [8] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, p. 9, 2008.

- [9] A. T. Khan, X. Cao, y S. Li. A Survey on Blockchain Technology and Its Potential Applications in Distributed Control and Cooperative Robots. ArXiv181205452 Cs, nov. 2018.
- [10] G. Destefanis, M. Marchesi, M. Ortù, R. Tonelli, A. Bracciali, y R. Hierons. Smart contracts vulnerabilities: a call for blockchain software engineering. International Workshop on Blockchain Oriented Software Engineering (IWBOSE), Campobasso, pp. 19-25, 2018.
- [11] Home — Ethereum, ethereum.org. Disponible en: "[https:ethereum.org](https://ethereum.org)" [Accedido: 04-oct-2019].
- [12] J. Herbert y A. Litchfield. A Novel Method for Decentralised Peer-to-Peer Software License. Computer Science, vol. 159, p. 9, 2015.
- [13] D. Johnston et al. The General Theory of Decentralized Applications, DApps. p. 12.
- [14] W. Ruttenberg. Andrea Pinna Distributed ledger technologies in. p. 35, 2016.
- [15] S. Porru, A. Pinna, M. Marchesi, y R. Tonelli. Blockchain-Oriented Software Engineering: Challenges and New Directions. IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, Argentina, pp. 169-171, 2017.
- [16] H. Rocha y S. Ducasse. Preliminary steps towards modeling blockchain oriented software. Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain - WETSEB '18, Gothenburg, Sweden, pp. 52-57, 2018.
- [17] K. Zile y R. Strazdina. Blockchain Use Cases and Their Feasibility. Applied Computer Systems, vol. 23, No. 1, pp. 12-20, may 2018.
- [18] P. Chakraborty, R. Shahriyar, A. Iqbal, y A. Bosu. Understanding the software development practices of blockchain projects: a survey. Proceedings of the 12th

- ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '18, Oulu, Finland, pp. 1-10, 2018.
- [19] X. Xu, C. Pautasso, L. Zhu, Q. Lu, y I. Weber. A Pattern Collection for Blockchain-based Applications. Proceedings of the 23rd European Conference on Pattern Languages of Programs - EuroPLoP '18, Irsee, Germany, pp. 1-20, 2018.
- [20] R. M. Ramanathaiah. Blockchain Technology - a Technology for Future Sustainable Development. *Blockchain Technol.*, vol. 8, No. 1, p. 8, 2019.
- [21] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, y J. Wang. Untangling Blockchain: A Data Processing View of Blockchain Systems. *IEEE Trans. Knowl. Data Eng.*, vol. 30, n.o 7, pp. 1366-1385, jul. 2018.
- [22] X. Xu et al. A Taxonomy of Blockchain-Based Systems for Architecture Design. IEEE International Conference on Software Architecture (ICSA). Gothenburg, Sweden, pp. 243-252, 2017.
- [23] L. Alvarado, «SXP, metodología ágil para el desarrollo de software», p. 12.
- [24] R. S. Pressman, Software engineering: a practitioner's approach, 7th ed. New York: McGraw-Hill Higher Education, 2010.
- [25] R. S. Pressman, Ingeniería del software: un enfoque práctico. 2013
- [26] «Docker Documentation», Docker Documentation, oct. 07, 2019.  
<https://docs.docker.com/> (accedido oct. 08, 2019).