

**UNIVERSIDAD SAN FRANCISCO DE QUITO
USFQ**

Colegio de Ciencias e Ingenierías

**Digitalización y manejo seguro de procesos transaccionales
mediante la implementación del modelo Blockchain.**

Carlos David Lucero Olalla

Ingeniería en Ciencias de la Computación

Trabajo de fin de carrera presentado como requisito
para la obtención del título de
Ingeniero en Ciencias de la Computación

Quito, 13 de mayo de 2021

**UNIVERSIDAD SAN FRANCISCO DE QUITO
USFQ**

Colegio de Ciencias e Ingenierías

**HOJA DE CALIFICACIÓN
DE TRABAJO DE FIN DE CARRERA**

**Digitalización y manejo seguro de procesos transaccionales
mediante la implementación del modelo Blockchain.**

Carlos David Lucero Olalla

Nombre del profesor, Título académico

Daniel Andrés Riofrío Almeida

Quito, 13 de mayo de 2021

© DERECHOS DE AUTOR

Por medio del presente documento certifico que he leído todas las Políticas y Manuales de la Universidad San Francisco de Quito USFQ, incluyendo la Política de Propiedad Intelectual USFQ, y estoy de acuerdo con su contenido, por lo que los derechos de propiedad intelectual del presente trabajo quedan sujetos a lo dispuesto en esas Políticas.

Asimismo, autorizo a la USFQ para que realice la digitalización y publicación de este trabajo en el repositorio virtual, de conformidad a lo dispuesto en la Ley Orgánica de Educación Superior del Ecuador.

Nombres y apellidos: Carlos David Lucero Olalla

Código: 00137141

Cédula de identidad: 1722633664

Lugar y fecha: Quito, 13 de mayo de 2021

ACLARACIÓN PARA PUBLICACIÓN

Nota: El presente trabajo, en su totalidad o cualquiera de sus partes, no debe ser considerado como una publicación, incluso a pesar de estar disponible sin restricciones a través de un repositorio institucional. Esta declaración se alinea con las prácticas y recomendaciones presentadas por el Committee on Publication Ethics COPE descritas por Barbour et al. (2017) Discussion document on best practice for issues around theses publishing, disponible en <http://bit.ly/COPETHeses>.

UNPUBLISHED DOCUMENT

Note: The following capstone project is available through Universidad San Francisco de Quito USFQ institutional repository. Nonetheless, this project – in whole or in part – should not be considered a publication. This statement follows the recommendations presented by the Committee on Publication Ethics COPE described by Barbour et al. (2017) Discussion document on best practice for issues around theses publishing available on <http://bit.ly/COPETHeses>.

RESUMEN

El presente documento pretende entender los conceptos teóricos y la implementación práctica de un Blockchain básico, ejecutado localmente, y como se convierte en una alternativa para la administración de un proceso transaccional. Se explican los pasos a seguir para la construcción del mismo, su diseño, su arquitectura, su código desarrollado en Python, y los fundamentos teóricos detrás del mismo.

También se muestran los diferentes casos de uso y aplicación; se comprueba la seguridad e integridad de datos que tanto caracteriza a este modelo, y el uso de alternativas más profesionales, para el desarrollo de un Blockchain propio, mediante herramientas desarrolladas por empresas. Se simula la administración de una aplicación CRUD mediante la adaptación del código original del modelo para así evidenciar los diferentes campos en los que Blockchain puede ser aplicado.

Palabras clave: Blockchain, red, procesos, seguridad, bloques, transacción.

ABSTRACT

This document aims to understand the theoretical concepts and the practical implementation of a basic Blockchain, executed locally, and how it becomes an alternative for the administration of a transactional process. The steps to follow for its construction, its design, its architecture, its code developed in Python, and the theoretical foundations behind it are explained.

The different use and application cases are also shown; The security and integrity of the data that characterizes this model is verified, and the use of more professional alternatives for the development of its own Blockchain, using tools developed by companies. The administration of a CRUD application is simulated by adapting the original code of the model in order to show the different fields in which Blockchain can be applied.

Keywords: Blockchain, network, processes, security, blocks, transaction.

TABLA DE CONTENIDO

TABLA DE CONTENIDO	7
INTRODUCCIÓN	11
CAPÍTULO 1: MARCO TEÓRICO	13
1.1 Blockchain	13
1.2 Tipos de Blockchain	15
1.3 Bloques	17
1.3.1 Bloque Génesis	19
1.4 Algoritmos de Consenso	20
1.4.1 Proof of Work	20
1.4.2 Proof of Stake	20
1.4.3 Proof of activity	21
1.4.5 Practical byzantine fault tolerance	21
1.5 Criptografía	21
1.6 Árbol de Merkle	22
1.7 Debilidades	23
CAPÍTULO 2: DISEÑO Y ARQUITECTURA	25
2.2 Bloques	26
2.3 Proof of Work	28
2.4 Servidor/API	30
2.5 Interfaz	31
2.6 Agregar más nodos a la red	32
2.7 Ejecutar la Aplicación	34
2.8 Librerías y dependencias	35
2.9 Protocolos	35
2.10 Simulación de Aplicación CRUD	36
2.10.1 Eliminar	37
2.10.2 Actualizar	39
CAPÍTULO 3: SEGURIDAD Y FUNCIONAMIENTO	42
3.1 Agregar un bloque inválido	42
3.2 Modificar la transacción de un bloque	44
3.3 Validar cadena utilizada	44
CAPITULO 4: APLICACIONES	46
4.1 Salud	46
4.2 Comunicación	47

4.3 Votaciones Electorales	47
4.4 Ethereum	47
4.5 Deployment en IBM	48
CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES	51
5.1 Conclusiones	51
5.2 Recomendaciones	52
REFERENCIAS BIBLIOGRÁFICAS	53

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Blockchain básico de Bitcoin (Zheng, Xie, & Dai, 2018).....	14
Ilustración 2: Bloques y sus campos.....	19
Ilustración 3: Estructura árbol de Merkle (Antonopoulos, 2014).....	23
Ilustración 4: Diseño del Blockchain a implementar.....	25
Ilustración 5: Clase Block.py.....	26
Ilustración 6: Función para crear bloque Genesis en Blockchain.py.....	27
Ilustración 7: Función para agregar bloque en Blockchain.py	28
Ilustración 8: Función para minar en Blockchain.py.....	28
Ilustración 9: Función para ejecutar el prook-of-work en Blockchain.py	29
Ilustración 10: Logo del framework para Python Flask	30
Ilustración 11: Interfaz del prototipo de prueba de transacciones	32
Ilustración 12: Petición cURL para agregar nodos.....	32
Ilustración 13: Petición al nodo 8000 para adjuntar el nodo 8001 usando Postman	33
Ilustración 14: Comandos para ejecutar servidor en Flask	34
Ilustración 15: Comando para ejecutar la aplicación web	34
Ilustración 16: Campos de un bloque	37
Ilustración 17: Interfaz gráfica de la aplicación actualizada	38
Ilustración 18: Método delete() del controlador	38
Ilustración 19: Método delete_transaction() de los nodos.....	39
Ilustración 20: Cadena luego de eliminar una transacción	39
Ilustración 21: Método update() del controlador	40
Ilustración 22: Método update_transaction() de los nodos.....	41
Ilustración 23: Cadena luego de actualizar una transacción.....	41
Ilustración 24: Ejemplo de nodo corriendo en el puerto 8000.....	42
Ilustración 25: Chain del nodo 8000 con el bloque Génesis y otro bloque válido	43
Ilustración 26: Petición en Postman para agregar un bloque invalido.....	43
Ilustración 27: Cadena del nodo 8000	45
Ilustración 28: Cadena del nodo 8001	45
Ilustración 29: Logo de Ethereum	48
Ilustración 30: Interfaz de Ganache.....	48
Ilustración 31: Arquitectura general de IMB Blockchain Platform.....	49

INDICE DE TABLAS

Tabla 1: Comparación entre tipos de Blockchain (Alvarez, 2018)	17
Tabla 2: Estructura general de un bloque	17
Tabla 3: Estructura del encabezado de un bloque	18
Tabla 4: Métodos y rutas declaradas en Server.py	31

INTRODUCCIÓN

La necesidad de la implementación de nuevas tecnologías que faciliten procesos que inicialmente fueron diseñados para ser desarrollados de manera manual, y la búsqueda de sustentar los mismos en fundamentos de seguridad y eficiencia ha llevado al ser humano a crear sistemas/arquitecturas informáticas que cumplan con estos requerimientos. Esta constante digitalización de elementos físicos ha llevado al surgimiento de nuevos modelos comerciales y de gestión (Teece & Linden, 2017). Es aquí donde gana fuerza el término Blockchain, de la mano del surgimiento de la conocida moneda digital Bitcoin en 2008, en un artículo publicado bajo el seudónimo Satoshi Nakamoto, donde se menciona el uso de cadenas de bloques (Blockchain) para el manejo de transacciones dentro de una red P2P (Rodríguez, 2018).

El modo más simple de explicar el porqué de la implementación de Blockchain es mediante un ejemplo: al momento en que deseamos pagar algo físicamente, el proceso es relativamente sencillo, se intercambia una moneda de cierto valor a cambio de algo de igual valor; teniendo constancia de que yo era dueño del dinero, el cual mediante una transacción pasó a la posesión de alguien más. Si deseamos replicar el proceso de manera digital el panorama se complica, pues sin una entidad reguladora nada nos garantizaría la veracidad de la posesión del dinero, tampoco podríamos confirmar el hecho de que el dinero, pase de mi posesión a la de otra persona, sin la intervención de un tercero no autorizado, y mucho menos podríamos saber si la transacción fue corrompida (Navarro). Blockchain soluciona estos problemas debido a su sistema caracterizado por su integridad, anonimato, persistencia e incorruptibilidad, basado (a diferencia de bancos y demás entidades financieras) en la descentralización de sus datos (Antonopoulos, 2014), cuya implementación es explicada más adelante.

La popularidad y éxito que actualmente tiene Bitcoin ha dado paso a las empresas para implementar en sus procesos, aplicaciones basadas en Blockchain, mucho más allá de ser utilizado solo en criptomoneda, llegando a ser la base de distintos servicios financieros como pagos en línea y demás. Adicionalmente, Blockchain está siendo el sustento de la nueva generación de aplicaciones que interactúan con internet como lo son los smart contracts, internet of things, security systems, etc (Zheng, Xie, & Dai, 2018).

CAPÍTULO 1: MARCO TEÓRICO

1.1 Blockchain

Blockchain es una tecnología que tuvo su auge en 2008, con la aparición de Bitcoin, y es considerada como una nueva manera de almacenar y administrar datos, donde la información, en vez de encontrarse ubicada en un solo lugar, se encuentra distribuida a través de una red compuesta por nodos (Ashwini, 2018). Desde distintos puntos de vista, es considerado como un tipo específico de base de datos distribuida y compartida, que distribuye los datos a manera de cadena mediante bloques criptográficos interrelacionados (Conway, *Blockchain Explained*, 2020), los cuales son inmutables, por lo que la información contenida es irreversible. Una definición más formal considera Blockchain como una Tecnología de Libro Mayor Distribuido (DLT) basada en una arquitectura P2P (peer-to-peer), que impide a un solo usuario y/o entidad tener el control de la red; sino que éste se distribuye entre todos los participantes de la misma (Merian, 2019), brindando así una transparencia transaccional muy atractiva para las empresas.

Esta tecnología se desarrolla a partir de tres componentes principales: “una transacción, un registro de transacciones y un sistema que verifica y almacena la transacción” (BBVA, 2016) donde actúa como unidad fundamental el bloque que registra información de manera cronológica utilizando funciones criptográficas que transforman la información en una nueva serie de caracteres, denominadas hashes, sobre las transacciones suscitadas y las replica a través de todo el sistema, encadenándose uno con el otro, el cual crece a medida que nuevos bloques se vayan añadiendo. En base a esta arquitectura puede decirse que mientras más nodos contenga el Blockchain, más

fuerte será (Singh & Kim, 2018). La Ilustración 1 muestra un ejemplo básico de la estructura del Blockchain que utiliza Bitcoin.

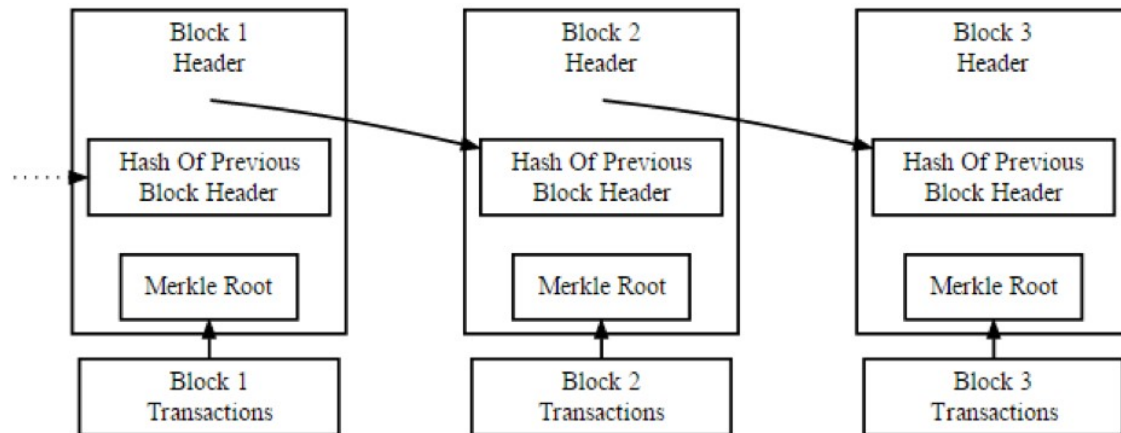


Ilustración 1: Blockchain básico de Bitcoin (Zheng, Xie, & Dai, 2018)

En esta arquitectura cada bloque contiene el hash del bloque anterior por lo que, si se desea manipular la información, todos los bloques deberán ser cambiados debido a la interrelación que tienen entre ellos (Singh & Kim, 2018); es gracias a esto que Blockchain es reconocido como un sistema impenetrable e incorruptible, cuyas principales características son las siguientes:

- No puede ser corrompida: para agregar una transacción esta debe ser verificada y validada por todos los nodos/bloques, promoviendo su transparencia (Rodríguez, 2019).
- Descentralizada: la red no se controla por una sola entidad central (Rodríguez, 2019).
- Seguridad: gracias a que su integridad está basada en la arquitectura de la red en sí, sumado al uso de mecanismos de cifrado y hash (Rodríguez, 2019).
- Distribuida: su costo computacional está distribuido entre toda la red, garantizando un mejor resultado (Rodríguez, 2019).

- Consenso: la toma de decisiones dentro de la red debe ser consensuada entre todos quienes la conforman gracias a sus algoritmos (Rodríguez, 2019).
- Acuerdos más rápidos: En comparación a los sistemas tradicionales (Rodríguez, 2019).

La arquitectura general de cualquier blockchain, al ser una red de malla descentralizada, se basa en distintos módulos encargados del correcto funcionamiento de la estructura y del manejo de datos. Esta arquitectura varía mínimamente según el autor, sin embargo, los componentes más comunes son los siguientes:

- Almacenamiento de datos: la estructura mediante la cual se administran los datos dentro de la red basándose en una secuenciación de las cadenas de bloques (Muzammal, 2019).
- Consenso: el método utilizado por Blockchain para garantizar la integridad del sistema. Esto quiere decir lo mencionado respecto a que cada participante del sistema debe validar cualquier cambio de información (Muzammal, 2019).
- Validación: el proceso por el cual Blockchain asegura un correcto sistema de transacciones verificando su estado (Muzammal, 2019).
- Red P2P: el tipo de red mediante la cual el intercambio de información se realiza de igual a igual, para luego ser replicada en todo el sistema.
- Criptografía: encargada de la seguridad y privacidad de la información dentro del sistema (Muzammal, 2019).

1.2 Tipos de Blockchain

Las arquitecturas Blockchain se clasifican principalmente en tres tipos:

- **Pública:** todos los que conforman la red pueden acceder a su información y realizar transacciones. Es susceptible a participación de usuarios no autorizados por lo que es necesaria una encriptación mayor, provocando que su expansión sea costosa, lenta y difícil (Oh & Shong, 2017).
- **Privada:** Es administrada mediante una entidad controladora, por lo que los usuarios son fácilmente reconocibles y su costo de expansión es baja, rápida y fácil (Oh & Shong, 2017).
- **Híbrida:** conocida también como Blockchain de consorcio, es una mezcla entre privada y pública donde nodos preestablecidos tienen la autoridad sobre el sistema. Es bastante eficiente, ya que mantiene la seguridad de un Blockchain privado y resuelve el tema de lentitud de un Blockchain público (Oh & Shong, 2017).

En la Tabla 1, se evidencia más fácilmente las diferencias entre los tipos de Blockchain de manera comparativa.

Tipos de Blockchain	<u>Pública</u>	<u>Híbrida</u>	<u>Privada</u>
Entidad gestora	Todos los participantes.	Participantes autorizados.	Institución central.
Gobernanza	Muy difícil cambiar la regla que se ha hecho.	Las reglas podrían cambiarse fácilmente según el acuerdo de los participantes.	Las reglas podrían cambiarse fácilmente según la institución central.
Expansión de red y velocidad de transmisión	Difícil expandir y velocidad lenta.	Fácil expandir y velocidad rápida.	Muy fácil ampliar y velocidad rápida.
Acceso	Cualquiera.	Solo usuarios autorizados.	Solo usuarios autorizados.
Identificabilidad	Semi-anónimo.	Identificable.	Identificable.
Prueba de transacción	Mediante algoritmos como PoW y PoS (proof of work y proof of stake), y no se	A través de autenticación, verificación de la transacción y generación de	Realizada por la institución central.

	puede conocer de antemano.	bloques se realizan de acuerdo con las reglas acordadas.	
Casos de uso	Bitcoin	R3CEV	Linq

Tabla 1: Comparación entre tipos de Blockchain (*Alvarez, 2018*)

1.3 Bloques

Como se mencionó anteriormente, la unidad de almacenamiento de datos principal en un Blockchain es el bloque, el cual está conformado por el peso del bloque en bytes, un encabezado que contiene un grupo de metadatos, el contador de transacciones que depende del tamaño del bloque en sí y de la transacción; y un registro de las transacciones realizadas con el mismo (Antonopoulos, 2014). La especificación y tamaño de cada campo de la estructura del bloque se muestra en la Tabla 2.

Tamaño	Campo	Descripción
4 bytes	Peso del Bloque	Peso del bloque en bytes.
80 bytes	Encabezado del Bloque	Diversos campos forman el encabezado.
1-9 bytes (VarInt)	Contador de Transacciones	Cuántas transacciones sucedieron.
Variable	Transacciones	Transacciones en el bloque.

Tabla 2: Estructura general de un bloque

El encabezado del bloque, se compone a su vez de distintos metadatos que caracterizarán e identificarán al nodo, los cuales se muestran en la Tabla 3:

Tamaño	Campo	Descripción
4 bytes	Version	Número de versión para monitorear actualizaciones.
32 bytes	Previous Block Hash	Referencia al bloque anterior en la cadena.
32 bytes	Merkle Root	Hash del root del merkle tree de las transacciones del bloque.
4 bytes	Timestamp	Tiempo de creación del bloque.
4 bytes	Difficulty Target	La dificultad utilizada por el algoritmo proof-of-work.
4 bytes	Nonce	Contador utilizado por el algoritmo proof-of-work.

Tabla 3: Estructura del encabezado de un bloque

La vinculación entre bloques se realiza en base al hash que representa a cada uno. En caso de que un nuevo bloque desee entrar al Blockchain, un nodo examinará el encabezado del bloque a ingresar en búsqueda del hash del bloque anterior y lo validará, actualizando constantemente la red (Antonopoulos, 2014).

Por otro lado, existe el término “minería” dentro de Blockchain, donde los “mineros” son los encargados de validar nuevas transacciones y registrarlas en la red. Estos mineros reciben recompensas por el esfuerzo informático que conlleva resolver los problemas matemáticos de algoritmos hash que requiere esta tarea. La acción de minar es comúnmente relacionada a Bitcoin, sin embargo, es un término que abarca cualquier Blockchain, puesto que mediante éste es posible manipular la red a nivel de nodo, es decir, agregar bloques.

1.3.1 Bloque Génesis

Se denomina como bloque génesis al primer bloque en base al cual el Blockchain desarrolla su estructura (Y., 2018). Es interesante saber que si se recorre cronológicamente todos los nodos de la red, eventualmente se llega a este nodo conformado por un código estático y que funciona a modo de raíz.

En la Ilustración 2 se muestra un ejemplo sencillo de la estructura de los bloques, los campos que contienen y como estos se encuentran encadenados.

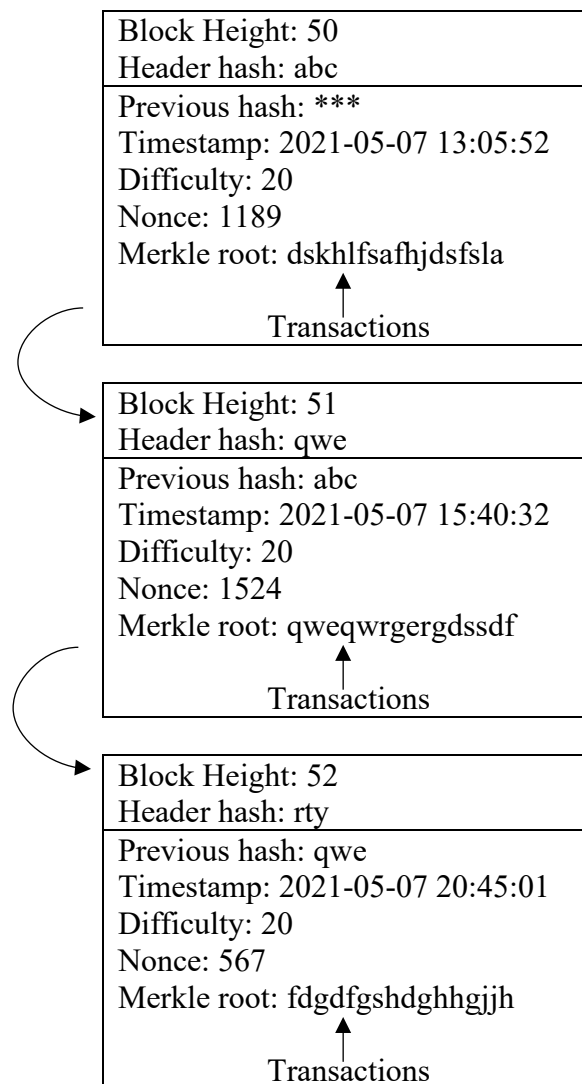


Ilustración 2: Bloques y sus campos

1.4 Algoritmos de Consenso

1.4.1 Proof of Work

Proof of Work (PoW), es la estrategia de consenso, en la cual se basa el Bitcoin. Este proceso es conocido comúnmente como “minar” y se utiliza para verificar la autenticidad de una cadena en Blockchain. Para esto, los nodos “mineros” calculan constantemente un valor hash para el bloque encabezado y una vez que uno de estos nodos obtiene un valor, posible candidato a ser considerado en la red, éste debe ser confirmado por los demás nodos. Una vez obtenida la confirmación, el conjunto de transacciones utilizadas para los cálculos es autenticada y llega a formar parte de la cadena como un nuevo bloque dentro del Blockchain. Durante este proceso, varios bloques válidos pueden ser creados simultáneamente por diferentes nodos, desarrollando así varias ramas en la cadena. El protocolo PoW, elige a la cadena más larga para ser la auténtica, parando el crecimiento de las demás (Zheng, Xie, & Dai, 2018).

PoW es el algoritmo de consenso más utilizado actualmente; sin embargo, presenta ciertas desventajas. Dos de las principales, son su alto consumo de energía debido a su alto costo computacional y su vulnerabilidad en caso de que mineros obtengan el 51% del cómputo de la red, ya que se podría manipular el Blockchain para beneficio propio (Ast, 2019).

1.4.2 Proof of Stake

Proof of Stake (PoS) surgió como una alternativa de menor consumo al protocolo PoW. Este consenso es más económico en términos de recursos computacionales, ya que el cálculo del valor hash ya no es necesario; en lugar de esto,

los nodos utilizan únicamente una firma digital. Esto, sin embargo, puede provocar que el control de la cadena se reduzca exclusivamente a unos cuantos participantes, siendo estos los más “dominantes”, por lo que se han propuesto varios mecanismos para evitar esta situación. Por ejemplo, una red puede elegir el siguiente nodo que minará la cadena basado en un valor aleatorio, o basado en la antigüedad del nodo dentro de la cadena (Zheng, Xie, & Dai, 2018).

1.4.3 Proof of activity

Proof of activity es un protocolo que nació como combinación de los dos descritos anteriormente, ya que un bloque minado debe ser verificado por un número determinado de nodos mineros antes de ser incluido en la cadena (Zheng, Xie, & Dai, 2018).

1.4.5 Practical byzantine fault tolerance

Practical byzantine fault tolerance (PBFT) es otro tipo de algoritmo de consenso en el que el nodo encargado de minar un futuro bloque es elegido por la cadena basado en parámetros específicos según el sistema. El proceso de selección puede dividirse en pre-preparación, preparación y confirmación. En cada etapa, el nodo debe obtener la confirmación de al menos dos tercios de todos los nodos para continuar a la siguiente. Es claro que para que esto sea realizable, todos los nodos deben ser visibles en la cadena (Zheng, Xie, & Dai, 2018).

1.5 Criptografía

La manera en que cada nodo se vincula entre sí y es identificado dentro de la red, y que es a la vez el núcleo de la seguridad de un Blockchain, proviene de su hash criptográfico. Para obtener este dato, Blockchain utiliza las funciones de hash SHA256.

Este valor es el resultado de aplicar el algoritmo sobre el encabezado de cada bloque dos veces. Hay que tomar en cuenta que el hash correspondiente a cada bloque no se incluye como tal en la estructura de datos de un bloque, sino que es calculado al momento en que éste ingresa a la red y se vincula al resto de nodos; sin embargo, es posible almacenar este dato con el fin de una indexación y recuperación más sencilla del bloque (Antonopoulos, 2014).

Es importante mencionar que como se pudo ver en la estructura de los bloques en la Sección 1.3, existe un valor dentro de cada bloque que representa su posición dentro del Blockchain denominado “peso del bloque”. Este también constituye una forma de poder identificar un nodo dentro de la red, sin embargo, podría no representar un identificador único ya que es posible que dos o más bloques se encuentren en la misma ubicación dentro de la red, pero éste es un caso muy excepcional.

1.6 Árbol de Merkle

Las transacciones contenidas dentro de un bloque son almacenadas dentro del mismo mediante la estructura de datos Árbol de Merkle o Árbol Hash Binario. Este permite verificar la integridad de grandes conjuntos de datos y genera una “huella digital” de todo el conjunto de transacciones. Este se construye realizando hash recursivo de abajo hacia arriba sobre pares de nodos hasta obtener un solo hash que resuma el total de todas las transacciones dentro del nodo, y es denominado raíz de merkle, basándose también en el algoritmo de SHA256 dos veces (Chumbley, 2016).

En caso de que se desee verificar si una transacción se encuentra en un bloque, el nodo debe producir hashes de arriba hacia abajo, es decir, al contrario a cuando se desea obtener la raíz de Merkle. Estos hashes por lo general son de 32 bytes

correspondientes a $\log_2(N)$ donde N es la cantidad de nodos existentes; y así conectar la transacción con la raíz de Merkle.

El Árbol de Merkle, destaca por la rapidez con la que realiza las operaciones al momento de transmitir datos y la fiabilidad que brinda (ESAN, 2019). En la Ilustración 3 se resume los pasos a seguir y como se encuentra estructurado un Árbol de Merkle.

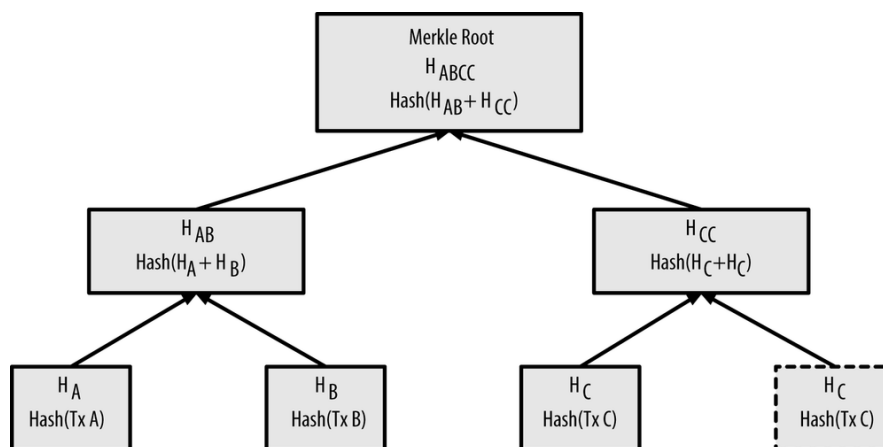


Ilustración 3: Estructura árbol de Merkle (Antonopoulos, 2014)

1.7 Debilidades

Una vez entendidos los conceptos detrás del funcionamiento del Blockchain, es importante mencionar que ningún modelo es perfecto. Entre las desventajas se encuentran la inmutabilidad de la información que en ciertos casos puede perjudicar a un usuario que realizó una transacción equivocadamente; también la dependencia de la velocidad de procesamiento de toda la red, ya que, si existe algún fallo, toda la red se vería afectada en términos de eficiencia.

Otra de sus desventajas está en la imposibilidad de acceso a la red en caso de que un usuario olvide las credenciales de acceso a su cuenta (por ejemplo); también debido a que este modelo puede reemplazar ciertos procesos, la cantidad de desempleo incrementaría gracias a que la demanda de mano de obra humana disminuiría. Como fue

mencionado, también existe la rara posibilidad de que un solo usuario controle más del 50% de la red, lo que le daría la capacidad de corromperla fácilmente (Camargo).

CAPÍTULO 2: DISEÑO Y ARQUITECTURA

2.1 Diseño y componentes

Con el fin de replicar el modelo de Blockchain descrito en el Capítulo 1, se utilizan diversas herramientas que permiten generar una versión local y elemental del mismo. Para esto, se utiliza el lenguaje de programación Python debido a su relativa sencilla aplicación, eficiencia y porque consta de librerías/paquetes que permiten la creación del Blockchain en una red. Este prototipo de arquitectura simula un proceso transaccional más simple, a modo de aplicación web de mensajería cuya capa de datos corre sobre uno o más nodos Blockchain y será desplegada en una misma red local donde estos nodos se comunicarán entre sí.

Los nodos pueden encontrarse corriendo en un mismo dispositivo en caso de que este cuente con la capacidad informática, tanto en software, como hardware suficiente para lograrlo, por lo general un ordenador básico con procesador Core i7/Ryzen 7, 8 GB RAM, y disco duro HDD cumple con los requisitos. Si lo que se busca es lanzar la red en máquinas virtuales o micro ordenadores, cada nodo deberá ubicarse en un dispositivo distinto.

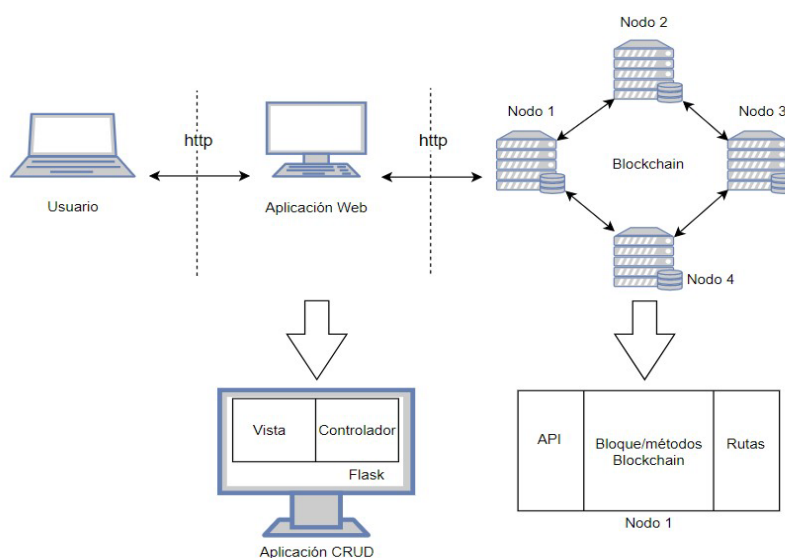


Ilustración 4: Diseño del Blockchain a implementar

En la Ilustración 4 se muestra el diseño de Blockchain a implementar.

Específicamente para esta aplicación, cada nodo se encuentra disponible localmente en un mismo ordenador a modo de servidor Flask (explicado en la Sección 2.4), cada uno en un puerto distinto. Cada bloque contiene únicamente una transacción; es importante mencionar que un Blockchain ya implementado en producción no necesariamente está construido de esta manera, pero para eventos prácticos esta aplicación si lo está.

La aplicación consta también de una interfaz gráfica (GUI) que se encuentra ejecutándose igualmente de manera local a modo de servidor Flask, en un puerto distinto a los correspondientes a cada nodo. Mediante el GUI será posible la creación de nuevos bloques vinculados a cada nodo y agregar las transacciones a los mismos.

La aplicación está desarrollada y adaptada en base al código subido por el usuario *satwikkansal* a la plataforma GitHub (satwikkansal, 2020).

2.2 Bloques

Cada bloque seguirá la estructura mencionada en la Sección 1.3 y será creado mediante la clase `Block.py` definida en la Ilustración 5:

```
class Block:
    def __init__(self, index, transactions, timestamp, previous_hash, nonce=0):
        self.index = index
        self.transactions = transactions
        self.timestamp = timestamp
        self.previous_hash = previous_hash
        self.nonce = nonce

    def compute_hash(self):
        block_hash = json.dumps(self.__dict__, sort_keys=True)
        return sha256(block_hash.encode()).hexdigest()
```

Ilustración 5: Clase `Block.py`

Esta clase contiene una función que se encarga de calcular el hash, basado en la función de encriptación SHA256, correspondiente a cada bloque y se encadena con todos los demás bloques de la red mediante el campo *previous_hash*. Como fue mencionado en la Sección 1.3, cada bloque puede contener varias transacciones agregadas mediante la interfaz gráfica que estarán almacenadas en formato JSON de la siguiente manera:

```
{
  "author": "autor",
  "content": "Contenido",
  "timestamp": "Tiempo en el que fue creado"
}
```

Es necesario tener en cuenta la creación del Bloque Génesis y establecer sus campos/variables *index* y *previous_hash* a cero. Este se crea mediante el método descrito en la Ilustración 6, dentro de la clase `Blockchain.py` y se lo encadena con toda la red.

```
def create_genesis_block(self):
    genesis_block = Block(0, [], 0, "0")
    genesis_block.hash = genesis_block.compute_hash()
    self.chain.append(genesis_block)
```

Ilustración 6: Función para crear bloque Genesis en `Blockchain.py`

Para agregar nuevos bloques a la cadena se utiliza el método de la Ilustración 7, que verifica la validez de la operación y comprueba que el campo *previous_hash* concuerde con el hash calculado del bloque al que se desea concatenar.

```

def add_block(self, block, proof):
    previous_hash = self.last_block.hash

    if previous_hash != block.previous_hash:
        return False

    if not Blockchain.is_valid_proof(block, proof):
        return False

    block.hash = proof
    self.chain.append(block)
    return True

```

Ilustración 7: Función para agregar bloque en Blockchain.py

Por último, en lo relacionado a manipular bloques, se desarrolló una función que permite agregar las transacciones pendientes del bloque y validar el *proof of work* (algoritmo explicado en la sección 2.3). La función es la siguiente, y se encuentra en la clase Blockchain.py de la Ilustración 8:

```

def mine(self):
    if not self.pending_transactions:
        return False

    last_block = self.last_block

    new_block = Block(index=last_block.index + 1,
                      transactions=self.pending_transactions,
                      timestamp=time.time(),
                      previous_hash=last_block.hash)

    proof = self.proof_of_work(new_block)
    self.add_block(new_block, proof)
    self.pending_transactions = []
    return True

```

Ilustración 8: Función para minar en Blockchain.py

2.3 Proof of Work

En base a la sección 1.4 para esta aplicación en particular se utiliza el Proof of Work. Mediante este algoritmo se garantiza que la tarea de calcular el hash en el

proceso de minado de cada bloque y los que lo preceden no sea fácil. Para esto, según la definición del como este algoritmo trabaja, creamos una función que valide cada cálculo de hash realizado, el cual trabajará sobre una variable *difficulty*, que tendrá el valor de 2. Esta variable define un objetivo para dificultar el cálculo del hash; este objetivo se traduce en una cantidad considerable de ceros al inicio de cada hash, el cual para fines prácticos se lo estableció en 2, lo que quiere decir que cada hash de esta aplicación deberá empezar con un par de ceros.

También se utiliza una variable *nonce* predefinida que afectará el resultado del hash y que al momento de minar, debe cambiar hasta satisfacer la condición del hash y que la operación sea validada por la red. El código de implementación de este algoritmo se encuentra en la clase `Blockchain.py` y se detalla en la Ilustración 9.

```
def proof_of_work(block):
    block.nonce = 0
    computed_hash = block.compute_hash()
    while not computed_hash.startswith('0' * Blockchain.difficulty):
        block.nonce += 1
        computed_hash = block.compute_hash()
    return computed_hash
```

Ilustración 9: Función para ejecutar el prook-of-work en `Blockchain.py`

A pesar de no ser el único algoritmo de consenso que se podría implementar, por lo general suele ser el más utilizado en Blockchains públicas, debido a que establece la misma dificultad informática para la resolución de su acertijo criptográfico (cálculo del hash sin conocimiento de las variables *difficulty* ni *nonce*) para los mineros, donde el primero que lo resuelva (el que mejor capacidad de cómputo tenga) será el recompensado (Ast, 2019).

2.4 Servidor/API

Una vez creadas las clases base que definen a la red es necesario definir los nodos que consumirán sus métodos y llamarán a las mismas para ser utilizadas por la interfaz y de esta manera poder agregar bloques y transacciones. al Blockchain. Para poder crear y montar la API (application programming interface), es decir, el conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones (Hat), localmente se utiliza el framework de Python, Flask (1.1.x) cuyo logo representativo es la Ilustración 10.



Ilustración 10: Logo del framework para Python Flask

Los métodos y rutas que utiliza esta Aplicación creada sobre el modelo Blockchain creado se listan en la Tabla 4 y se encuentran en el archivo Server.py.

Método	Ruta	Descripción
(POST) new_transaction	/new_transaction	Agrega nueva transacción al Blockchain.
(GET) get_chain	/chain	Devuelve la cadena del nodo.
(GET) mine_unconfirmed_transactions	/mine	Minar las transacciones pendientes.
(POST) register_new_peers	/register_node	Agrega participantes a la red.
(POST) register_with_existing_node	/register_with	Registra el nodo actual con el especificado en la instrucción.

(POST) <code>verify_and_add_block</code>	<code>/add_block</code>	El nodo verifica el bloque y es agregado a la cadena.
(GET) <code>get_pending-tx</code>	<code>/pending_tx</code>	Devuelve las transacciones pendientes aún no agregadas a ningún bloque.

Tabla 4: Métodos y rutas declaradas en `Server.py`

El método *register_new_peers*, es el que da la cualidad a la red de poder ser descentralizada, ya que con este se puede agregar diversos nodos que trabajan sobre la misma cadena y dar conciencia de la existencia de cada uno entre ellos. Una función sencilla de consenso entre nodos, que garantiza que se utilice la cadena más larga como la válida (Sección 1.4.1), sumada al método *verify_and_add_block* son los que garantizan la integridad del Blockchain y de que cada nodo tenga conocimiento sobre cualquier cambio realizado en la red, y poder funcionar/actualizarse correctamente.

2.5 Interfaz

La interfaz gráfica de la aplicación está creada bajo el patrón de diseño vista/controlador. Así, la vista está desarrollada en HTML en base a una plantilla, ésta se conecta con los nodos creados en la red local anteriormente, en la Sección 2.4, y consume las rutas de la API de cada nodo. En base a los campos requeridos de cada transacción la interfaz solicita un mensaje y el autor. El botón *Agregar transacción* agrega ésta al bloque, pero para poder validarla dentro de la red es necesario *Minar transacciones pendientes*, ya que de esta manera el servidor verifica la veracidad de la transacción y la agrega al Blockchain. El controlador de la interfaz se encuentra en el archivo `view.py`. La vista principal de la interfaz gráfica puede visualizarse en la Ilustración 11.

Blockchain

Mensaje

Tu nombre Agregar transacción

Minar transacciones pendientes Visualizar

D David
Publicado a las 01:57

Hola

Ilustración 11: Interfaz del prototipo de prueba de transacciones

2.6 Agregar más nodos a la red

La idea principal del funcionamiento del Blockchain es su sistema descentralizado. Para ello, lo ideal sería agregar varios nodos a la red y el modo de hacerlo se encuentra definido mediante el método *register_new_peers* y la ruta */register_with* tal como se detalla en la Tabla 4.

Una vez que se tiene el primer nodo corriendo, se lanza otro de la misma manera, pero en un puerto distinto (en caso de montar la red sobre un solo dispositivo). Es necesario, realizar una petición POST a la ruta */register_with* para validar la inclusión de este nuevo nodo en la red. Postman es una buena herramienta para realizar este tipo de peticiones; sin embargo, en este ejemplo se muestra la vinculación de los nodos utilizando una petición cURL de la Ilustración 12, ya que por lo general viene por defecto en los sistemas operativos, en este caso, Windows.

```
curl -X POST http://127.0.0.1:8001/register_with -H 'Content-Type: application/json'
-d '{"node_address": "http://127.0.0.1:8000"}'
```

Ilustración 12: Petición cURL para agregar nodos

De esta manera, queda registrado el nuevo nodo conectado con el primer nodo creado, logrando cumplir la arquitectura “ideal” de un sistema Blockchain. Es importante mencionar que para el correcto funcionamiento de esta aplicación, y de la red, siempre que se desee agregar un nuevo nodo se debe realizar la petición a la ruta *register_with* tanto de el/los nodos ya creados como de el/los nodos que se vayan a adjuntar, es decir, los nodos “viejos” deben saber de la existencia de los nodos “nuevos” y viceversa.

Adicionalmente, se describirá el proceso utilizando la herramienta Postman debido a que las peticiones cURL suelen fallar en determinadas ocasiones, fallo que suele presentarse principalmente debido a una mala sintaxis de la petición. En la Ilustración 13, se encuentra un ejemplo de una petición POST escrita en Json, utilizando Postman.

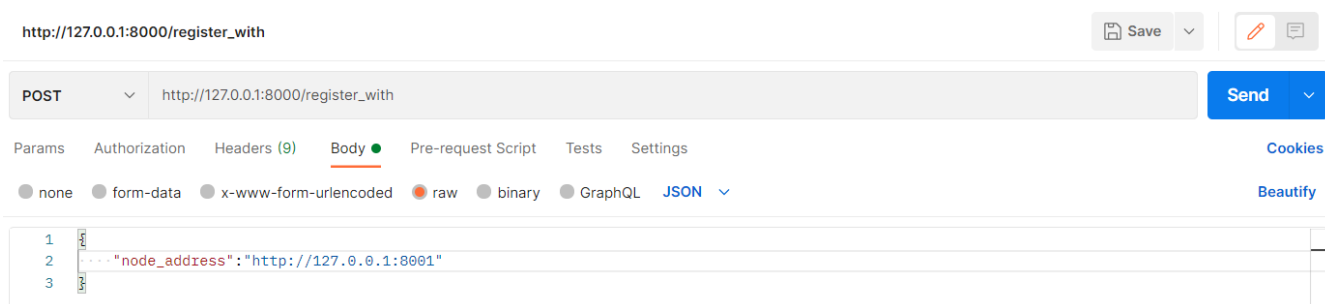


Ilustración 13: Petición al nodo 8000 para adjuntar el nodo 8001 usando Postman

Si todo se realizó correctamente, el nodo 8000 (en este caso) devolverá un `status_code` con el texto “Registration Successful”, confirmando que el nodo se agregó correctamente.

2.7 Ejecutar la Aplicación

Una vez definidas las clases base, el servidor en Flask y la aplicación, se procede a ejecutar todo en nuestra red local. En primer lugar, se ejecuta el servidor en Flask mediante los comandos descritos en la Ilustración 14:

```
>set FLASK_APP=Server.py
```

```
>flask run --port 8000
```

Ilustración 14: Comandos para ejecutar servidor en Flask

El segundo comando de la Ilustración 14 lanzará nuestro servidor en el puerto local 8000; es importante hacerlo de esta manera debido a la configuración por defecto del código de la aplicación, la cual se puede cambiar en caso de modificar los puertos y nombres especificados en el código.

Lo siguiente es ejecutar la aplicación de prueba, en este caso en red local mediante comando de la Ilustración 15:

```
>python Run.py
```

Ilustración 15: Comando para ejecutar la aplicación web

En la configuración de este ejemplo, la aplicación web se ejecuta en el puerto 5000 de la máquina; este es el puerto especificado por defecto en el código. En este momento la aplicación tiene acceso a la implementación de Blockchain que se encuentra ejecutándose localmente.

2.8 Librerías y dependencias

Para poder ejecutar la aplicación y la implementación de Blockchain que se explica en este documento, se requiere tener instalado las siguientes dependencias, independientemente de su versión:

- Json
- Datetime
- Requests
- Sha256
- Time
- Flask

2.9 Protocolos

Por lo general, una red Blockchain ya fuertemente estable se basa en su propio protocolo para la transferencia de datos, considerándose Blockchain como un protocolo en sí, muy similar al funcionamiento de TCP/IP, pero incorporando características que lo vuelven un método mucho más fuerte para el envío y recepción de información (Walker, 2018).

Específicamente para esta aplicación, el protocolo utilizado para la conexión entre la interfaz web con el servidor Blockchain y para la transferencia de datos fue HTTP debido a que la misma se encuentra disponible simplemente sobre una red local. Por el mismo motivo y con el objetivo de evidenciar más fácilmente el funcionamiento del Blockchain no se utilizó HTTPS, lo cual, de ser utilizado, brindaría una seguridad

mayor a la ya obtenida gracias a la encriptación de todas las peticiones realizadas a la API.

2.10 Simulación de Aplicación CRUD

Tal como se mencionó en la Sección 1.1, donde fueron explicados los fundamentos básicos de una Blockchain, uno de los puntos fuertes de este modelo y que lo caracterizan es la inmutabilidad de los datos, es decir, la permanencia de la información dentro de la red, por lo que la modificación o eliminación de transacciones es algo inicialmente no contemplado en la definición básica del funcionamiento de Blockchain. Es por este motivo, que en ciertas ocasiones, es necesario adaptar la implementación de un Blockchain, con el fin de satisfacer los requerimientos de distintas aplicaciones según sea el caso, como éste.

Para el desarrollo de este documento se adaptó el código del Blockchain original, con el objetivo de que el mismo soporte operaciones básicas características de una aplicación CRUD. El modelo CRUD (create, read, update and delete) es comúnmente utilizado en el desarrollo de aplicaciones web y su principio más básico es el soporte de las operaciones mencionadas anteriormente (Codecademy). El Blockchain original ya consta por defecto de las operaciones de crear y leer transacciones, por lo que para esta simulación fueron implementadas las operaciones de modificar y eliminar transacciones; cada operación se explica en las siguientes secciones.

Teniendo en cuenta la inmutabilidad del Blockchain, no es posible implementar estas dos operaciones así como tal, sino simularlas. Para explicar mejor lo mencionado, dentro de una red Blockchain no es posible borrar o modificar transacciones (a excepción de ciertos casos), ya que la idea de este tipo de red es mantener intactos todos los datos que la conforman. Dicho de manera más específica, los bloques no pueden (ni

deben) ser alterados. Como solución a esta problemática, se determinó realizar estas operaciones a nivel de interfaz gráfica respectivamente y simularlas a nivel de red/nodos. También fue necesario agregar las nuevas rutas */delete_transaction* y */update_transaction* en el archivo *Server.py*.

2.10.1 Eliminar

Como se mencionó anteriormente, los bloques y transacciones no pueden ser eliminados, por esto se agregó un campo *status* en cada bloque, el cual cambiará entre “create”, “update” o “delete” y servirá para determinar el estado en el que se encuentra el bloque, según su interacción con la interfaz gráfica. En la Ilustración 16, se puede observar los campos requeridos para la creación de un bloque una vez insertado el nuevo atributo *status*.

```
class Block:
    def __init__(self, index, transactions, timestamp, previous_hash, status, nonce=0):
        self.index = index
        self.transactions = transactions
        self.timestamp = timestamp
        self.previous_hash = previous_hash
        self.nonce = nonce
        self.status = status
```

Ilustración 16: Campos de un bloque

Inicialmente, cada bloque se creará con el atributo *status* en “create” y este cambiará mediante la interfaz gráfica, la cual se comunica con el nodo y le notifica cambiar este atributo de su/sus bloque/s según el caso mediante solicitudes POST. La modificación de la interfaz gráfica se puede visualizar en la Ilustración 17 donde el principal cambio se encuentra en la adición de los campos Eliminar y Actualizar.

Blockchain

Mensaje

D

David

Publicado a las 20:54

Hola

Ilustración 17: Interfaz gráfica de la aplicación actualizada

Como se dijo inicialmente, los cambios se realizaron principalmente a nivel de interfaz gráfica y su controlador; en el código del controlador de la vista (el archivo `view.py`) se agregó la ruta `/delete` y el método `delete()` que se encargan de comunicarse con el nodo y le envía el índice del bloque al que corresponde la transacción que se desea eliminar, como se puede ver en la Ilustración 18.

```
@app.route('/delete', methods=['POST'])
def delete():
    ind = request.form["index"]
    post_object = {
        'index': ind
    }
    new_tx_address = "{}/delete_transaction".format(CONNECTED_NODE_ADDRESS)
    requests.post(new_tx_address,
                  json=post_object,
                  headers={'Content-type': 'application/json'})
    print("view", post_object)
    return redirect('/')
```

Ilustración 18: Método `delete()` del controlador

Del lado del servidor/nodo se agregó la ruta `/delete_transaction` y el método `delete_transaction()` que se encargan de recorrer todos los bloques hasta encontrar el

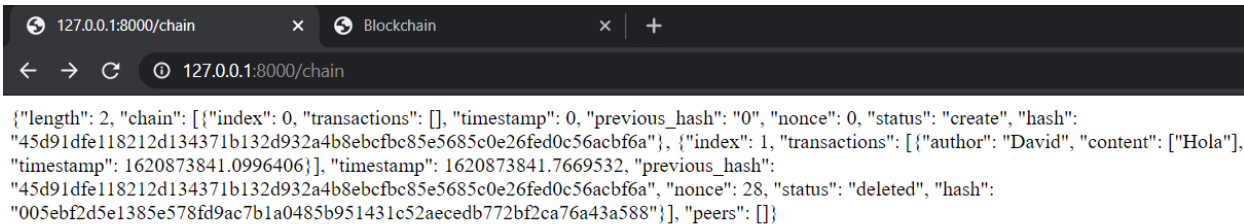
correspondiente al índice enviado desde el lado del controlador y cambiar su *status* a “deleted”, como se puede ver en la Ilustración 19.

```
@app.route('/delete_transaction', methods=['POST'])
def delete_transaction():
    tx_data = request.get_json()
    required_fields = ["index"]
    print("server", tx_data["index"])
    for field in required_fields:
        if not tx_data.get(field):
            return "Invalid transaction data", 404

    for block in blockchain.chain:
        if block.index == int(tx_data["index"]):
            block.status = "deleted"
    return "Success", 201
```

Ilustración 19: Método `delete_transaction()` de los nodos

Una vez implementados estos cambios podemos ver como se visualiza la cadena, al eliminar una transacción de un bloque en la Ilustración 20. El controlador de la interfaz gráfica, se encarga de leer el *status* de cada bloque y no mostrar los que se encuentran en “deleted”.



```
{
  "length": 2,
  "chain": [
    {
      "index": 0,
      "transactions": [],
      "timestamp": 0,
      "previous_hash": "0",
      "nonce": 0,
      "status": "create",
      "hash": "45d91dfe118212d134371b132d932a4b8ebcfbc85e5685c0e26fed0c56acbf6a"
    },
    {
      "index": 1,
      "transactions": [
        {
          "author": "David",
          "content": ["Hola"],
          "timestamp": 1620873841.0996406
        }
      ],
      "timestamp": 1620873841.7669532,
      "previous_hash": "45d91dfe118212d134371b132d932a4b8ebcfbc85e5685c0e26fed0c56acbf6a",
      "nonce": 28,
      "status": "deleted",
      "hash": "005ebf2d5e1385e578fd9ac7b1a0485b951431c52aecdb772bf2ca76a43a588"
    }
  ],
  "peers": []
}
```

Ilustración 20: Cadena luego de eliminar una transacción

2.10.2 Actualizar

Para la operación de actualizar, se utiliza el mismo campo *status* agregado en la Sección 2.10.1, pero en este caso cuando se desee actualizar una transacción cambiará su estado a “updated”. Para la implementación de esta operación, se modificó el campo *transactions* del bloque; este campo recibe de parte del controlador los *datos author*,

content y *timestamp*. Específicamente se modificó el campo *content* el cual es declarado como una lista que llevará un registro de los cambios que se realizarán a esa transacción. De igual manera, a la operación de eliminar, la interfaz gráfica es la encargada de mostrar solo el último contenido de la lista del bloque, con las transacciones actualizadas y permitirá, para fines prácticos, solo modificar el contenido de cada transacción, es decir, su mensaje (ver Ilustración 17).

Se implementó la ruta */update* y el método *update()* en el archivo *view.py* correspondiente al controlador. Estos se encargan de comunicarse con el nodo, enviándole el índice de la transacción que se desea modificar y el nuevo contenido de esta, como se puede ver en la Ilustración 21.

```
@app.route('/update', methods=['POST'])
def update():
    ind = request.form["index"]
    upd = request.form["update"]
    post_object = {
        'index': ind,
        'update': upd
    }
    new_tx_address = "{} /update_transaction".format(CONNECTED_NODE_ADDRESS)
    requests.post(new_tx_address,
                  json=post_object,
                  headers={'Content-type': 'application/json'})
    print("view", post_object)
    return redirect('/')
```

Ilustración 21: Método *update()* del controlador

Del lado del nodo, se implementó la ruta */update_transaction* y el método *update_transaction()* que se encargarán de recibir la información por parte del controlador de la vista. Recorre los bloques de la cadena hasta coincidir con el bloque correspondiente a la transacción que se desea modificar, modifica su *status* a “updated”

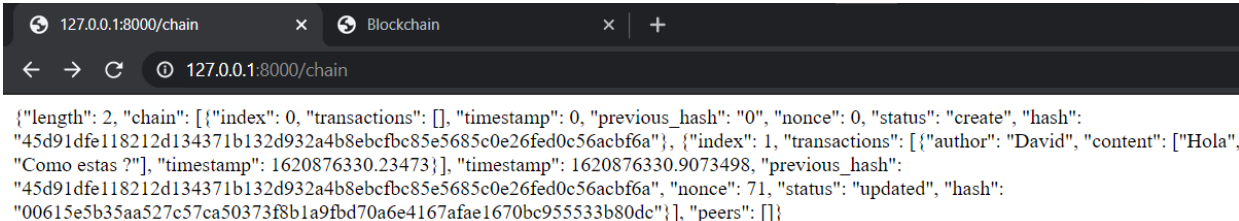
y agrega el nuevo contenido de la transacción a la lista de todos los contenidos de las transacciones del bloque, como se puede ver en la Ilustración 22.

```
@app.route('/update_transaction', methods=['POST'])
def update_transaction():
    tx_data = request.get_json()
    required_fields = ["index", "update"]
    print("server", tx_data["update"])
    for field in required_fields:
        if not tx_data.get(field):
            return "Invalid transaction data", 404

    for block in blockchain.chain:
        if block.index == int(tx_data["index"]):
            block.status = "updated"
            print(block.transactions[0])
            print(block.transactions)
            block.transactions[0]['content'].append(tx_data["update"])
    return "Success", 201
```

Ilustración 22: Método `update_transaction()` de los nodos

Podemos ver la cadena resultante, una vez que se actualiza una transacción, y como se agregan las modificaciones en la lista de contenido de cada transacción en la Ilustración 23.



```
{
  "length": 2,
  "chain": [
    {
      "index": 0,
      "transactions": [],
      "timestamp": 0,
      "previous_hash": "0",
      "nonce": 0,
      "status": "create",
      "hash": "45d91dfe118212d134371b132d932a4b8ebcfbc85e5685c0e26fed0c56acbf6a"
    },
    {
      "index": 1,
      "transactions": [
        {
          "author": "David",
          "content": ["Hola", "Como estas ?"],
          "timestamp": 1620876330.23473
        }
      ],
      "timestamp": 1620876330.9073498,
      "previous_hash": "45d91dfe118212d134371b132d932a4b8ebcfbc85e5685c0e26fed0c56acbf6a",
      "nonce": 71,
      "status": "updated",
      "hash": "00615e5b35aa527c57ca50373f8b1a9fbd70a6e4167afae1670bc955533b80dc"
    }
  ],
  "peers": []
}
```

Ilustración 23: Cadena luego de actualizar una transacción

CAPÍTULO 3: SEGURIDAD Y FUNCIONAMIENTO

La aplicación se encuentra programada, creada y funcional, tanto la interfaz gráfica como el lado del servidor, pero, ¿cómo se evidencia realmente el funcionamiento interno de la red y que tan segura es? En esta sección se tratará, en la medida de lo posible, de comprobar y comprender de mejor manera el funcionamiento que está por detrás de la implementación de este modelo Blockchain.

Para demostrar la seguridad que brinda esta implementación se inician dos nodos, uno corriendo en el puerto 8000 (Ilustración 24) y otro en el 8001. Para esta demostración no es estrictamente necesario ejecutar la interfaz gráfica. En los siguientes apartados se muestran distintos escenarios que ratifican la seguridad que brinda Blockchain.

A screenshot of a Windows command prompt window. The title bar reads 'C:\Windows\System32\cmd.exe - flask run --port 8000'. The terminal shows the following text:

```
C:\Users\Usuario\Documents\ProyectoIntegrador\Blockchain_code>set FLASK_APP=Server.py
C:\Users\Usuario\Documents\ProyectoIntegrador\Blockchain_code>flask run --port 8000
* Serving Flask app "Server.py"
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:8000/ (Press CTRL+C to quit)
```

Ilustración 24: Ejemplo de nodo corriendo en el puerto 8000

3.1 Agregar un bloque inválido

Trabajando únicamente en el nodo 8000, se puede observar que se encuentra creado el bloque Génesis, al cual corresponde su determinado hash. Luego, mediante la interfaz gráfica, se agregan dos bloques, uno válido y otro no válido.

```

{"length": 2, "chain": [{"index": 0, "transactions": [], "timestamp": 0, "previous_hash": "0", "nonce": 0,
"hash": "6dbf23122cb5046cc5c0c1b245c75f8e43c59ca8ffec292715e5078e631d0c9"}, {"index": 1,
"transactions": [{"author": "David", "content": "Hola", "timestamp": 1618275977.2530565}], "timestamp":
1618275978.2207363, "previous_hash":
"6dbf23122cb5046cc5c0c1b245c75f8e43c59ca8ffec292715e5078e631d0c9", "nonce": 52, "hash":
"001942061843927c70980e3de666691548157171d6d272f34dd61f5f6409e27e"}], "peers": []}

```

Ilustración 25: Chain del nodo 8000 con el bloque Génesis y otro bloque válido

En la Ilustración 25 se observa el encadenamiento de los bloques, nótese que el atributo *previous_hash* del bloque con *index* 1 corresponde al hash calculado del bloque Génesis. En caso de querer agregar un bloque inválido, es decir, con su campo *previous:hash* distinto al hash del bloque con *index* 1, los nodos que componen la red lo rechazarán y nos devolverá un mensaje de error tal como se muestra en la parte inferior de la Ilustración 26.

```

POST http://127.0.0.1:8000/add_block
Body
[
  1  {
  2    "index": "41",
  3    "transactions": "comoestas",
  4    "timestamp": "12042021",
  5    "previous_hash": "abcde",
  6    "nonce": "0",
  7    "hash": "qwerty"
  8  }
]
Status: 400 BAD REQUEST
1 The block was discarded by the node

```

Ilustración 26: Petición en Postman para agregar un bloque inválido

De esta manera, garantizamos que la arquitectura del Blockchain, en efecto requiere de una validación previa de las transacciones que se intenten realizar antes de permitir que la red sea modificada.

3.2 Modificar la transacción de un bloque

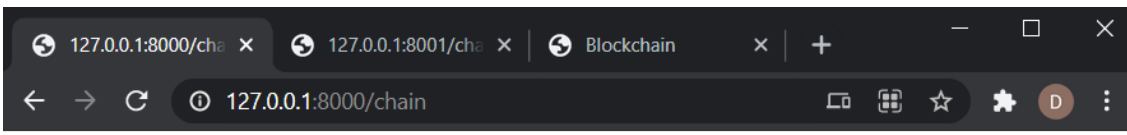
En este caso en específico, el sistema está diseñado de tal manera que manipular una transacción ya creada no es posible, debido a que no se puede acceder a las mismas, solo agregarlas. Esto depende del tipo de aplicación que se desee implementar en el Blockchain. Sin embargo, en base a lo anteriormente demostrado, si de algún modo una persona con malas intenciones logra acceder a las transacciones de un bloque y las modifica, la red se invalidaría bajo un concepto similar al anterior mostrado al querer agregar un nodo inválido, su hash cambiaría por lo que todos los nodos de la red rechazarían esa operación y mantendrían el estado actual del Blockchain, consensuado por todos los nodos.

3.3 Validar cadena utilizada

Surge una cuestión al momento de querer agregar un nuevo nodo al Blockchain, ¿qué cadena utilizará la red? Esto se da debido a que en un inicio, cada nodo individualmente tiene su propia cadena, la cual cambia a medida que se encadena con otros nodos. La cuestión se da al momento en el que, por ejemplo, una cadena ya concatenada por dos nodos ya creados trata de vincular la nueva cadena correspondiente a un nuevo nodo.

La respuesta está en la definición como tal del algoritmo de consenso utilizado, en este caso el `proof_of_work`, el cual menciona que el Blockchain, una vez que valida la unión de un nuevo nodo, valida también que nodo contiene la cadena más larga para

utilizar esta en toda la red (Sección 1.4.1). Es por esto que es necesario realizar la operación mencionada en la Sección 2.6 y ejemplificada en la Ilustración 12, de realizar el proceso de registrar el nuevo nodo con los nodos viejos y viceversa, ya que de esta manera toda la red tiene conocimiento de quienes la componen. Si esta validación de cadenas no se realiza y los nodos no tienen conocimiento de su existencia entre sí, esta aplicación funcionaría a modo de diferentes Blockchains, uno por cada nodo. La situación explicada se puede observar en las cadenas de los dos nodos creados en la Ilustración 27 y la Ilustración 28.

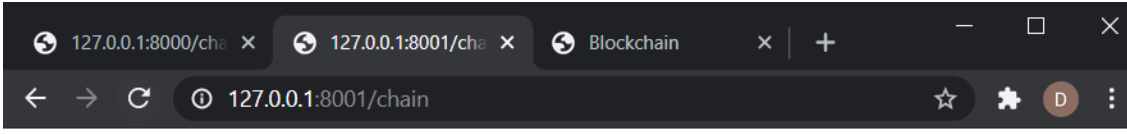


```

{"length": 1, "chain": [{"index": 0, "transactions": [], "timestamp": 0, "previous_hash": "0", "nonce": 0, "hash": "6dbf23122cb5046cc5c0c1b245c75f8e43c59ca8ffeac292715e5078e631d0c9"}], "peers": ["http://127.0.0.1:8001/", "http://127.0.0.1:8000/"]}

```

Ilustración 27: Cadena del nodo 8000



```

{"length": 1, "chain": [{"index": 0, "transactions": [], "timestamp": 0, "previous_hash": "0", "nonce": 0, "hash": "6dbf23122cb5046cc5c0c1b245c75f8e43c59ca8ffeac292715e5078e631d0c9"}], "peers": ["http://127.0.0.1:8000/", "http://127.0.0.1:8001/"]}

```

Ilustración 28: Cadena del nodo 8001

Una vez realizado ese proceso, tal como era de esperarse, tanto el nodo 8000 como el nodo 8001, ahora tienen la misma cadena, de no realizarse de esta manera la red no se actualizará, ya que solo un nodo tendrá conocimiento del otro y sus cadenas serán distintas, invalidando el Blockchain.

CAPITULO 4: APLICACIONES

Como se ha evidenciado durante el desarrollo de este documento, la implementación de Blockchain tiene grandes funcionalidades, debido al modo en el que este maneja la información y la integridad que es capaz de generar. Es por esto que este modelo se convirtió en un gran atractivo para diversas empresas/industrias alrededor del mundo. Su aplicación más famosa es en Criptomonedas, específicamente en el Bitcoin, el cual actualmente ha logrado ganar importante peso en la economía global, costando \$58.209 la unidad aproximadamente a día de hoy (Goldprice, 2021). El motivo del constante éxito del Bitcoin se basa, entre varias cosas, a que debido a su arquitectura (Blockchain) su valor no es propenso a inflaciones ni se encuentra regulado por una entidad centralizada, convirtiéndolo cada vez más en el respaldo financiero de varios países (Conway, Why Is Bitcoin's Price Rising?, 2020).

A continuación, se listarán algunas de las industrias y/o servicios donde se puede hacer uso de Blockchain, y que no se encuentran relacionados a procesos monetarios, donde claramente este modelo ha ganado más reconocimiento.

4.1 Salud

Blockchain se ha extendido más allá de transacciones financieras, impactado hasta en la industria de la salud. Dentro de ésta, está siendo utilizado a modo de base de datos, donde información general de cada persona, como edad e historial médico, se encuentra disponible en la red para que los dispositivos utilizados por médicos o quien esté autorizado acceda a estos datos, superando la limitación de almacenamiento al tratar con gran volumen de datos (Intelligence, 2020).

4.2 Comunicación

Las empresas enfocadas en los medios de comunicación implementan Blockchain, con el objetivo de reducir costos, proteger la información y los derechos de propiedad intelectual de la música. De hecho, la compañía de distribución y producción de películas MGM Studios utiliza Blockchain para la “transmisión global a la web, dispositivos móviles y TV en todas partes para audiencias de ciertas propiedades” (Intelligence, 2020).

4.3 Votaciones Electorales

Blockchain, garantiza que el proceso de votación sea más asequible y seguro por lo que es un excelente método para prevenir ataques informáticos que intenten corromper los nodos de una red de votación, y de esta manera es posible obtener una información más certera y eficiente al momento de realizar un conteo, por ejemplo (Intelligence, 2020).

4.4 Ethereum

Es posiblemente la segunda implementación más famosa de Blockchain. Ethereum es una plataforma de código abierto donde cualquiera puede programar aplicaciones descentralizadas. Es utilizado principalmente para el desarrollo de contratos inteligentes que se autoejecutan, siempre y cuando cumplan ciertas condiciones (Buterin, 2016).

Ethereum (Ilustración 29) amplía la capacidad de uso de su modelo en Blockchain más allá de limitarlo solo a una criptomoneda y que, a diferencia de Bitcoin, no limita la cantidad de Ether (su criptomoneda) que puede ser emitida (Plus500).



Ilustración 29: Logo de Ethereum

4.5 Deployment en IBM

Existen diversas maneras de realizar el deployment (lanzamiento a producción) de un Blockchain. Por ejemplo, una herramienta que permite realizar pruebas y ejecutar comandos sobre un Blockchain de Ethereum personal es Ganache; este tipo de herramientas de desarrollo facilitan la recreación de entornos de Blockchains localmente (Beyer, 2019) como podemos ver en la Ilustración 30.

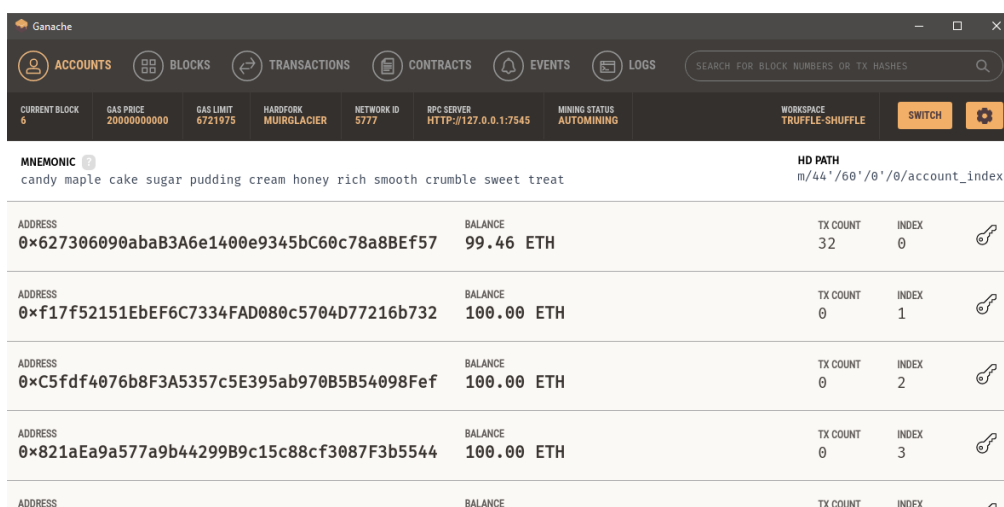


Ilustración 30: Interfaz de Ganache

Ganache es una herramienta enfocada a la manipulación de smart contracts mediante Ethereum por lo que específicamente es útil para este tipo de implementación. Por este motivo, una herramienta más exhaustiva y que puede ser utilizada para diversas

implementaciones de Blockchain, más allá de la que Ganache está enfocado, es IBM Blockchain Platform.

IBM Blockchain Platform es un servicio brindado por IBM basado en Hyperledger (plataforma de código abierto enfocado en el desarrollo e implementación de Blockchains (Hyperledger)) que permite al usuario la implementación, escalabilidad y control sobre los componentes de una red Blockchain, que se ejecuta en los propios servidores de IBM Cloud, además de constar con una alta capacidad de personalización y la posibilidad de crear aplicaciones cliente propias que puedan ejecutarse sobre el Blockchain (Maheshwari, 2020). En la Ilustración 31 la arquitectura detrás del funcionamiento de esta herramienta puede visualizarse de manera general.

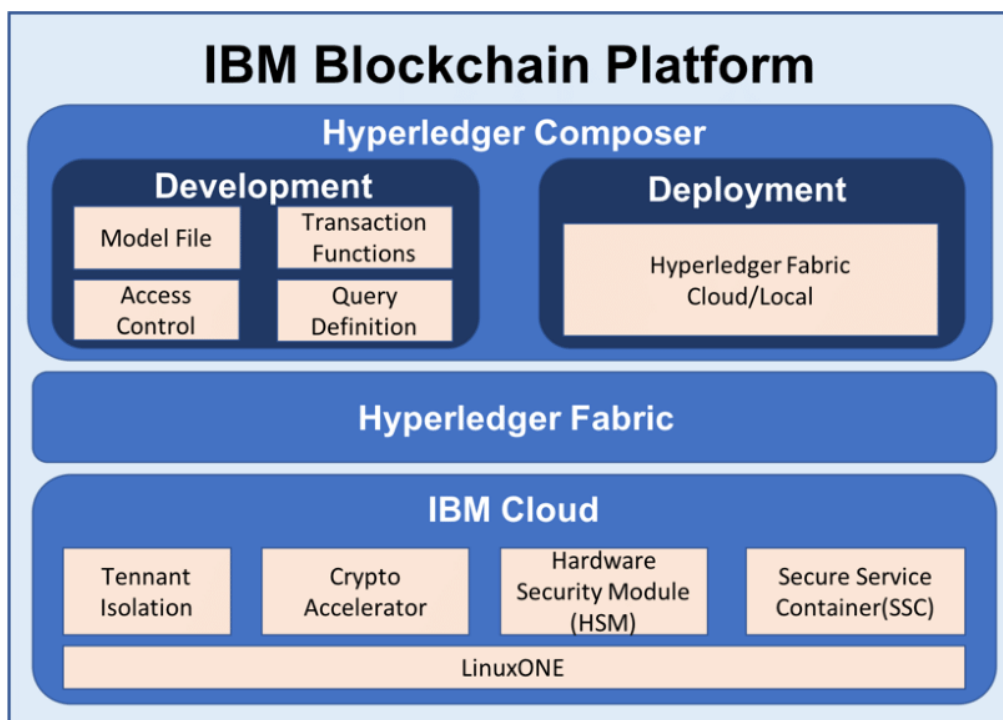


Ilustración 31: Arquitectura general de IBM Blockchain Platform

Los pasos a seguir para utilizar esta herramienta, así como documentación acerca de su uso se encuentran publicados en la página oficial de IBM Developer, específicamente en el link correspondiente a la siguiente cita y que se encuentra adjuntado en la sección de Referencias (Maheshwari, 2020).

CAPÍTULO 5: CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Blockchain, actualmente, es uno de los modelos que más seguridad e integridad brindan y no cabe duda que en un futuro será uno de los más utilizados. Su amplio campo de implementación lo convierten en una viable alternativa para la digitalización de diferentes tipos de procesos y/o transacciones, motivo por el cual muchas empresas se encuentran interesadas en su uso y desarrollo, incluso, a pesar de no ser recomendado su uso en aplicaciones CRUD, puede ser adaptado para satisfacer los diferentes tipos de necesidades según el caso tal como se pudo observar en el desarrollo de este documento.

Es evidente la seguridad que caracteriza este modelo, por lo que fácilmente puede ser utilizado para administrar información de alta importancia, lo cual es una gran ventaja teniendo en cuenta que cada día los atacantes informáticos mejoran sus herramientas y sus modos de operar. Si bien es cierto que una implementación de Blockchain más globalizada requiere de cierta dificultad tanto computacional como intelectual, una vez esta se encuentre correctamente establecida, el panorama es bastante favorecedor, solo hace falta ver el éxito actual de Bitcoin.

Por otro lado, Blockchain favorece también a la economía mundial al convertirse en una alternativa de administración de intercambios de comercios y valores, además de favorecer a diferentes campos del mercado laboral y de servicios, optimizando y mejorando la eficiencia de muchos de sus procesos, algunos de los cuales fueron explicados en este documento. Es importante también tener un conocimiento acerca del funcionamiento detrás del mismo, para así poder sacar provecho aún más de sus posibilidades.

Respecto a Ingeniería en Computación, y áreas relacionadas, este modelo explota muchos de los conocimientos aprendidos durante el transcurso de mi carrera, pues es necesario saber acerca de redes, seguridad informática, desarrollo de aplicaciones, protocolos, programación, manejo de datos, y demás. Por este motivo, la implementación de un Blockchain es una excelente manera de aplicar y desarrollar este tipo de destrezas propias de un informático.

Es importante considerar que ningún modelo es perfecto y que Blockchain como cualquier otro pueden presentar ciertas debilidades. Por esto es necesario, tener claro el objetivo que se desee en la implementación de un Blockchain, y como este se desenvolverá, para así determinar si este modelo es el apropiado para lo que se desee realizar y en caso de serlo contar con planes de contingencia en la medida de lo posible.

5.2 Recomendaciones

Primeramente, tener en cuenta que si se desea implementar este modelo, mínimo es necesario un conocimiento básico de sus conceptos y funcionalidad, así como considerar realmente si el modelo resultará eficiente para lo que se busca realizar. Por otro lado, considero que no se debe descartar este método como una alternativa de uso gracias a todas las ventajas que proporciona, así que cualquier empresa debería tener en alta consideración su implementación.

Recomiendo hacer uso de herramientas profesionales que diversas empresas brindan para la facilidad, tanto para el desarrollo como administración de un Blockchain, así como su deployment. De este modo es posible asegurar una correcta implementación del mismo, además de ahorrar muchos de los pasos correspondientes a su creación y optimizar así también el tiempo.

REFERENCIAS BIBLIOGRÁFICAS

- Alvarez, L. (Noviembre de 2018). *ANÁLISIS DE LA TECNOLOGÍA BLOCKCHAIN, SU ENTORNO Y SU IMPACTO EN MODELOS DE NEGOCIOS*. Obtenido de Repositorio USM:
<https://repositorio.usm.cl/bitstream/handle/11673/47346/3560900251199UTFSM.pdf?sequence=1&isAllowed=y>
- Antonopoulos, A. (1 de diciembre de 2014). *Mastering Bitcoin*. Obtenido de O'Reilly:
<https://learning.oreilly.com/library/view/mastering-bitcoin/9781491902639/copyright.html>
- Ashwini, A. (16 de enero de 2018). *Everything you need to know about Blockchain: Part One*. Obtenido de The Startup: <https://medium.com/swlh/everything-you-need-to-know-about-blockchain-part-one-d66552425d15>
- Ast, F. (25 de Mayo de 2019). *Entendiendo los Protocolos de Consenso de Blockchain*. Obtenido de Medium: <https://medium.com/astec/entendiendo-los-protocolos-de-consenso-de-blockchain-4858c71722d2>
- BBVA. (enero de 2016). *Tecnología BLOCKCHAIN*. Obtenido de https://www.bbva.com/wp-content/uploads/2017/10/ebook-cibbv-tecnologia_blockchain-es.pdf
- Beyer, S. (28 de noviembre de 2019). *What is Ethereum Ganache?* Obtenido de MYCRYPTOPEDIA: <https://www.mycryptopedia.com/what-is-ethereum-ganache/>
- Buterin, V. (2016). *What is Ethereum?* Obtenido de Ethereum Official webpage: https://marketing.binary.com/crypto/Binary.com_WhatisEthereum.pdf
- Camargo, F. (s.f.). *¿Cuáles son las ventajas y desventajas de Blockchain?* Obtenido de Camargo.life: <https://camargo.life/blockchain-ventajas-y-desventajas/#:~:text=Las%20principales%20desventajas%20de%20Blockchain,informaci%C3%B3n%20no%20se%20puede%20modificar.>
- Chumbley, A. (27 de abril de 2016). *Merkle Tree*. Obtenido de Brilliant: <https://brilliant.org/wiki/merkle-tree/>
- Codecademy. (s.f.). *What is CRUD?* Obtenido de codecademy: <https://www.codecademy.com/articles/what-is-crud>
- Conway, L. (17 de noviembre de 2020). *Blockchain Explained*. Obtenido de Investopedia: <https://www.investopedia.com/terms/b/blockchain.asp>
- Conway, L. (05 de noviembre de 2020). *Why Is Bitcoin's Price Rising?* Obtenido de Investopedia: <https://www.investopedia.com/tech/cryptocurrency-this-week/>
- ESAN. (4 de diciembre de 2019). *Fundamentos de Blockchain: ¿qué es un Merkle tree?* Obtenido de conexionesan: <https://www.esan.edu.pe/apuntes-empresariales/2019/12/fundamentos-de-blockchain-que-es-un-merkle-tree/>
- Goldprice. (09 de 04 de 2021). Obtenido de GOLDPRICE: <https://goldprice.org/es/cryptocurrency-price/bitcoin-price>
- Hat, R. (s.f.). *Qué son las API y para qué sirven*. Obtenido de Red Hat: <https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces#:~:text=Una%20API%20es%20un%20conjunto,de%20saber%20c%C3%B3mo%20est%C3%A1n%20implementados.>
- Hyperledger. (s.f.). *Hyperledger*. Obtenido de Hyperledger: <https://www.hyperledger.org/>

- Intelligence, I. (2 de marzo de 2020). *The growing list of applications and use cases of blockchain technology in business and life*. Obtenido de Business Insider: <https://www.businessinsider.com/blockchain-technology-applications-use-cases>
- Maheshwari, S. (10 de julio de 2020). *Learn step-by-step how to set up a basic blockchain network*. Obtenido de IBM Developer: <https://developer.ibm.com/technologies/blockchain/tutorials/quick-start-guide-for-ibm-blockchain-platform/>
- Merian, L. (29 de enero de 2019). *What is blockchain? The complete guide*. Obtenido de COMPUTERWORLD: <https://www.computerworld.com/article/3191077/what-is-blockchain-the-complete-guide.html#:~:text=Based%20on%20a%20peer%2Dto,%2C%20or%20game%2C%20the%20network.>
- Muzammal, M. (enero de 2019). *Renovating blockchain with distributed databases: An open source system*. Obtenido de ScienceDirect: <https://www.sciencedirect.com/science/article/pii/S0167739X18308732?via%3Dihub>
- Navarro, W. (s.f.). *HISTORIA DEL BLOCKCHAIN, LA SOLUCIÓN A UN PROBLEMA*. Obtenido de Addalia: <https://blog.addalia.com/historia-del-blockchain>
- Oh, J., & Shong, I. (4 de diciembre de 2017). *A case study on business model innovations using Blockchain: focusing on financial institutions*. Obtenido de emerald insight: <https://www.emerald.com/insight/content/doi/10.1108/APJIE-12-2017-038/full/html>
- Plus500. (s.f.). *¿Qué es Ethereum?* Obtenido de plus500: <https://www.plus500.es/Instruments/ETHUSD/What-is-Ethereum~1>
- Rodríguez, N. (3 de Diciembre de 2018). *Historia de la tecnología Blockchain: Guía definitiva*. Obtenido de 101 Blockchains: <https://101blockchains.com/es/historia-de-la-blockchain/>
- Rodríguez, N. (9 de enero de 2019). *6 Características clave de la tecnología blockchain que debes conocer!* Obtenido de 101 Blockchains: <https://101blockchains.com/es/caracteristicas-tecnologia-blockchain/>
- satwikkansal. (17 de Septiembre de 2020). *python_blockchain_app*. Obtenido de GitHub: https://github.com/satwikkansal/python_blockchain_app
- Singh, M., & Kim, S. (9 de noviembre de 2018). *Branch based blockchain technology in intelligent vehicle*. Obtenido de ScienceDirect: <https://www.sciencedirect.com/science/article/pii/S1389128618308399?via%3Dihub>
- Teece, D., & Linden, G. (24 de Agosto de 2017). *Business models, value capture, and the digital enterprise*. Obtenido de SpringerOpen: <https://jorgdesign.springeropen.com/articles/10.1186/s41469-017-0018-x#citeas>
- Walker, R. (1 de Noviembre de 2018). *Intro to Blockchain: How Protocols Work*. Obtenido de Medium: <https://medium.com/all-things-venture-capital/intro-to-vc-how-blockchain-protocols-work-750e9371b053#:~:text=re%2Dhash%20it%20if%20you,sending%20information%20and%20transferring%20value.>
- Y., R. (2 de febrero de 2018). *Blockchain Data Structure*. Obtenido de LinkedIn: <https://www.linkedin.com/pulse/blockchain-data-structure-ronald-chan>
- Zheng, Z., Xie, S., & Dai, H.-N. (octubre de 2018). *Blockchain challenges and opportunities: A survey*. Obtenido de ResearchGate:

https://www.researchgate.net/publication/328338366_Blockchain_challenges_and_opportunities_A_survey