Desentrañando los Secretos de Blockchain: De la Teoría a la Implementación

Sebastián Arroyo*, Victor Fuentes*, Matías Gastellu*, Marcelo Quiñones*
*Universidad Diego Portales, Av. Ejército Libertador 441, 8370191 Santiago, Región Metropolitana

***Abstract*—En el siguiente documento se detalla el diseño e implementación de un sistema de Blockchain, con una estructura bien definida y aplicando los principios de Blockchain y de sistemas descentralizados. El proyecto consiste en un sistema de transacciones de pequeña escala orientado al testeo de las funcionalidades principales de una cadena de bloques, construida con objetos y respaldada en una base de datos. La base de datos es implementada en LevelDB, se usa la librería Libp2p para establecer la comunicación entre los nodos participantes de la red, y además se define una capa de datos y una interfaz programada en JavaScript. Finalmente, se detallan las pruebas finales, demostrando su buen funcionamiento.**

I. Introducción

Para comprender el funcionamiento del sistema diseñado y su arquitectura, primero se deben comprender los fundamentos de Blockchain (o cadena de bloques), el modelo de capas en el que se basa, así como el modo de comunicación entre los distintos componentes de una red de Blockchain.

Una cadena de bloques, es una tecnología de registro de datos distribuida que permite a múltiples partes confiar entre sí y realizar transacciones sin la necesidad de un intermediario central. Su origen se remonta a la creación de Bitcoin en 2009, para luego ir evolucionando de la mano de los distintos sistemas que se fueron creando con el tiempo, tales como Ethereum, Solana, Cardano, etc.

En su esencia, BlockChain es una cadena de bloques de datos, en que cada bloque contiene un conjunto de transacciones. La característica distintiva de esta tecnología radica en su capacidad para crear un consenso descentralizado, donde todos los participantes de la red mantienen una copia idéntica del libro de contabilidad distribuido, y cada nueva transacción es validada y registrada mediante algoritmos de consenso.

La arquitectura Blockchain se puede dividir en 5 capas:

1) Capa de hardware: La capa de hardware en un sistema de Blockchain comprende la infraestructura física esencial para el funcionamiento de la red. Incluye elementos como nodos de la red, hardware de minería, una base de datos, dispositivos de red, etc.
2) Capa de datos: Se define la estructura de datos del sistema como una lista enlazada de bloques en las que se ordenan las transacciones. La lista consta de bloques que se entrelazan mediante punteros que apuntan a los hash de cada bloque.
3) Capa de red: Establece una comunicación distribuida de par a par entre los nodos que participan en la red, permitiendo el descubrimiento, las transacciones y la propagación de bloques. Esta capa garantiza que los nodos puedan encontrarse entre sí e interactuar, difundir y sincronizar para mantener la red blockchain en un estado legítimo.
4) Capa de consenso: Implementa un algoritmo de consenso y garantiza que los nodos de la red estén de acuerdo en el estado actual de la cadena de bloques. La capa de consenso es crucial para mantener la integridad y la coherencia del libro de contabilidad distribuido. Estas funcionalidades se omitirán en el presente proyecto.
5) Capa de aplicación: En este nivel se desarrollan y ejecutan aplicaciones específicas que aprovechan la funcionalidad de la cadena de bloques. Se encuentra el desarrollo de aplicaciones, smart contracts, interfaces de usuario y funcionalidades especificas según las necesidades de la red

Si bien algunos de los elementos mencionados, tales como la capa de consenso y algunas funcionalidades de la capa de aplicación, no se utilizan en este proyecto, a continuación se detalla el funcionamiento e implementación de nuestra criptomoneda **CheloCoin**.

II. Motivación

El proyecto surge de la necesidad de abordar los desafíos que comprende una red de Blockchain de manera práctica, implementando funcionalidades vistas a lo largo del semestre con el objetivo de crear un entorno de prueba seguro, eficiente, con información persistente y sin restricciones monetarias para participar en la red. Las motivaciones de este proyecto son diversas y se centran en realizar un sistema compuesto por una base de datos central, para probar las funcionalidades principales de una red de Blockchain.

La red implementada resuelve problemas de seguridad, transparencia, fiabilidad e inmutabilidad. Esto lo logra gracias a la implementación de un registro inmutable de transacciones, un aumento de seguridad en las transacciones a través de firmas y algoritmos criptográficos en la información tratada, un fácil acceso a los datos y transacciones inmutables y atomizadas, resolviendo el problema del doble gasto. Por otra parte, la red posee datos persistentes, de modo que si la red cae, la cadena se puede reestructurar gracias a la información almacenada en la base de datos.

La red está enfocada a un sistema de testeo para realizar transacciones entre los nodos participantes de la red y con una base de datos centralizada, dejando de lado las restricciones monetarias y algunas funcionalidades de las billeteras de cada usuario. El funcionamiento de la red es el siguiente: un usuario puede participar en la red fácilmente al crear un nodo, obteniendo su billetera y sus firmas, para luego testear cada

una de las funcionalidades implementadas, a través de una interfaz de usuario. El usuario debe ser capaz de conectarse en la red p2p, visualizar el registro de transacciones, buscar y realizar transacciones y acceder a la información de la cadena, como el bloque génesis y el ultimo de la cadena.

III. Descripción de la arquitectura de software

La arquitectura del proyecto posee un enfoque orientado a la modularidad y se puede dividir en 5 capas lógicas

A. Capa de datos

Aquí se encuentra la base de datos centralizada que almacena la información de la cadena de datos. Esta interfaz comprende de los datos de bloques, transacciones y billeteras. La base de datos corresponde a LevelDB, la cual guarda datos del tipo llave-valoren la carpeta *database* del proyecto, se implementó usando el lenguaje JavaScript y se le asignó el puerto 3000. Además, esta capa provee las funcionalidades necesarias para manejar la seguridad de las transacciones y crear las claves criptográficas necesarias para acceder y gestionar las billeteras. Incluye la función de *hash* SHA256 para las transacciones y la generación de llaves *pem* para las llaves públicas y privadas de cada billeteras, usando *spki* y *pkcs8*.

Por otra parte, se trabaja con las clases *cadena*, *Bloque*, *Transacción* y *Billetera*. En las clases se definen los datos fundamentales para crear una cadena compuesta por bloques, los cuales almacenan una transacción, asociada a dos billeteras.

Cabe destacar que cada bloque está compuesto por una transacción, con el objetivo de facilitar el testeo del sistema y obtener una mayor simplicidad en su implementación. Además facilita el seguimiento de transacciones, implica una menos complicaciones para integrar un eventual algoritmo de consenso y facilita la indexación de bloques.

B. Capa de Red

La capa de red P2P facilita la comunicación, la sincronización y la cooperación entre los nodos de la red. El modelo de conexión de par a par (P2P) permite la comunicación directa entre nodos en la red sin depender de un servidor central. De este modo se crea una topología en la cual los participantes de la red se unen como nodos y pueden escribir y leer información de la cadena mandando broadcasts para mantener la red actualizada. Esta capa se implementó usando JavaScript y para su implementación se consideró usar el protocolo echo, asignar direcciones IP a cada nodo, hacer el envío de flujos de datos usando *pipes*. Además, cada nodo se puede comunicar con sus pares conociendo las direcciones de sus pares y usando Web Sockets, además se usa un multiplexor de flujos para gestionar el envío y recepción de flujos de datos.

C. Capa de Acceso a Datos

Esta capa facilita la comunicación entre la capa de aplicación y la base de datos. Gestiona la lectura y escritura de datos en la LevelDB a través de una API. Dicha API define direcciones asociadas al puerto en donde corre el servidor, con el objetivo de obtener información y mantener actualizada la base de datos LevelDB. Provee las operaciones necesarias para almacenar datos, obtener el bloque Génesis, obtener la cadena de datos, obtener el último bloque, obtener un bloque según su ID, etc.

D. Capa de aplicación

Aquí reside la lógica de negocio central del sistema, es decir las funcionalidades de lectura y escritura de bloques, ejecución de transacciones, la creación de la Blockchain, la inicialización de los nodos que componen la red y establece la comunicación entre nodos. Esta capa está implementada en JavaScript.

Usa los puntos de acceso esenciales para interactuar con la cadena de bloques y realizar diversas operaciones, tales como obtener información sobre transacciones, acceder a los datos de billetera, enviar la cadena completa a la base de datos y acceder a datos específicos de la cadena de bloques. Además, se permite instanciar los objetos de las clases descritas en la capa de datos y se implementa la lógica de las transacciones.

Gracias a la implementación de las funcionalidades nombradas, se logra inicializar la cadena a través de la creación del bloque Génesis y la actualización de la base de datos, luego verifica si el bloque genesis ya está creado y de ser así, la cadena de datos se obtendrá directo de la carpeta de la información de la database. En caso contrario, se realiza una segunda inicialización, para luego crear billeteras, conectar nodos y realizar transacciones que nutran al sistema. Por otra parte, se inicializan los nodos y se define la comunicación entre nodos a través de mensajes asíncronos.

Por otra parte, en esta capa se implementa la lógica de las transacciones y billeteras, de modo que cuando se transfiere dinero, esto se ve reflejado en el balance de las billeteras participantes. Se establece la persistencia de las billeteras instanciadas, de modo que si el sistema cae los balances y las llaves no se ven afectadas. Además, se ingresan validadores de balance a la hora de hacer transacciones.

E. Capa de Interfaz de Usuario (UI)

En esta capa, se maneja la interacción del usuario con el sistema, de modo que cuando un nodo se conecta a la red, en su consola se despliega un menú interactivo con un listado de funcionalidades que puede realizar. Dichas funcionalidades se usan para testear el sistema y son: obtener el último bloque de la cadena, obtener el último bloque de la base de datos, obtener el bloque génesis de la base de datos, enviar una transacción genérica, obtener un bloque indicando su ID, mandar un mensaje entre un par de nodos y salir del menú.

IV. Detalles de la implementación

A continuación se explican las funcionalidades principales del sistema implementado.

A. Implementación del Nodo y Protocolo de Comunicación

Para llevar a cabo la creación y comunicación de nodos se implementó el protocolo de comunicación P2P (peer to peer) a través de una biblioteca modular llamada Libp2p,

la cual incorpora un set de protocolos de comunicación, especificaciones y herramientas orientadas a este tipo de redes.

Se usa el protocolo *echo* en su versión 1.0.0 y para mandar flujos de datos (streams) se usan pipes, que poseen una fuente (el nodo que manda los datos) y un destino (el nodo que recibe la información). La información se decodifica y el nodo de origen multiplexa los canales de información para mandar el mensaje a modo de broadcast, enviando el mensaje a todos los nodos que componen la red. Para que esto suceda los nodos poseen direcciones y una tabla de ruteo con los nodos de toda la red.

Cabe destacar que la gestión de flujos de datos se realiza de forma asíncrona, ya que las operaciones de red y E/S (entrada y salida) son operaciones no bloqueantes. Por lo cual, en este entorno asincrónico, se utiliza async y await para manejar la asincronía de las operaciones.

Para que un nodo se una a la red se le debe asignar un ID de par, una dirección IP de uso local (en el rango de ”127.0.0.0” a ”127.255.255.255”) y un puerto (sobre 7000), luego se define el método de transporte de datos por Web Sockets y usa una lista de *Bootstrapers*. Dichos *Bootstrapers* desempeñan un papel crucial en este proceso al proporcionar direcciones de nodos conocidos, ya que poseen las direcciones de todos los nodos de la red, de modo que cuando un nodo entra, este se puede conectar con el resto de los nodos. La lista de Bootstrapers se va actualizando a medida que los nodos entran y salen de la red.

Los nodos necesitan estar actualizados y conocer la información de sus pares con el objetivo de poder acceder a ellos, de modo que dicha información se va almacenando en la carpeta *nodes*, como una especie de tabla de ruteo. por cada nodo hay se almacenan dos archivos; en uno de ellos se almacena su dirección IP, su puerto, se especifica el protocolo de envío de mensajes *tcp* y la llave pública de su billetera, mientras que en el otro archivo se guarda el ID del nodo.

B. Métodos de Lectura y Escritura implementados

Con respecto a los métodos de lectura, el usuario dispone de las siguientes funcionalidades de lectura para recibir datos de la cadena y de la base de datos. Se diferencian ambas funcionalidades para demostrar que la información de la cadena coincide con la información que se guarda en la base datos. Para obtener los datos de la blockchain, se accede directamente al objeto instanciado de la clase cadena, en el cual hay un método para obtener bloques de la lista de bloques que almacena. Mientras que para leer la información de la base de datos, se accede a las direcciones de la API REST implementada, de modo que se puede acceder a los endpoints correspondientes a la consulta por bloque, consulta de la última transacción y consulta de saldo de las billeteras.

En el caso de la escritura se diferencia la funcionalidad de escritura en la blockchain y en la base de datos. Para agregar un bloque a la cadena se realiza una transacción entre billeteras, en la cual se debe indicar la llave pública del receptor y la firma digital. Luego, se verifica la información del remitente y si es válido, se realiza el proceso de minado del bloque. Una vez que el bloque es minado se agrega al final de la cadena. Finalmente, la información se actualiza en la base de datos, accediendo a la dirección de escritura de la API REST implementada. Así, la información de la blockchain se actualiza constantemente y en caso sufrir alguna caída, el sistema se pueda reconstruir.

Para agregar un bloque a la cadena se realiza una transacción entre billeteras, en la cual se debe indicar la llave pública del receptor y la firma digital. Luego, se verifica la información del remitente y si es válido, se realiza el proceso de minado del bloque. Una vez que el bloque es minado se agrega al final de la cadena.

C. Solución del Doble Gasto

El doble gasto se soluciona porque al realizar una transacción y minar un bloque, la información nueva es leída y validada por todos los nodos de la red antes de ser registrada. Por otra parte, las transacciones son atómicas, es decir todas las operaciones dentro de una transacción deben ser confirmadas como parte del bloque o ninguna de ellas. En caso de que la transacción no sea aceptada por la red, la transacción se revierte, es decir, las operaciones dentro de la transacción se deshacen, y la cadena de bloques vuelve a un estado consistente. De esta manera, una transacción nunca se registra más de una vez.

D. Modelos de Datos

A continuación, se muestra el modelo de datos utilizado, mostrando las clases con sus atributos y métodos y la base de datos con las operaciones que expone su API.

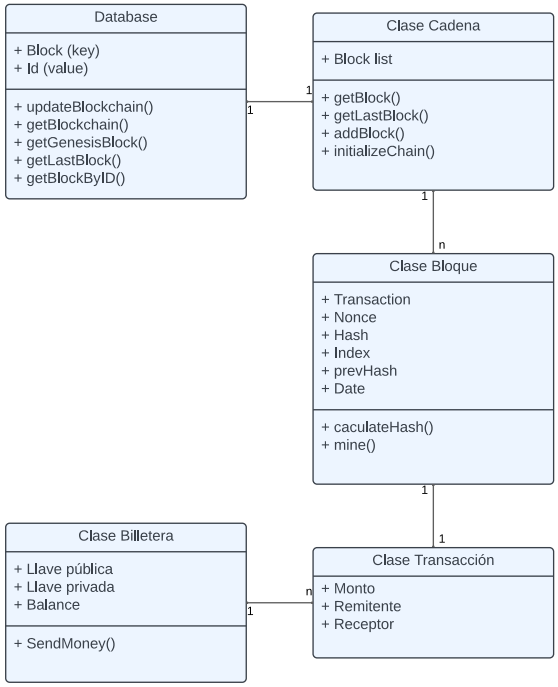

Fig. 1. Modelo de datos.

E. Bloque de Origen

El primer paso para implementar el sistema corresponde a la creación del bloque Génesis (o bloque de origen), el cual corresponde a la primera transacción realizada en el sistema y por lo tanto, es el primer bloque del sistema. El primer paso del sistema implementado es inicializar el sistema, en el cual se pueden dar dos casos. Si el bloque de origen no existe, se crea una cadena y se agrega un bloque compuesto por una transacción genérica y se actualiza la base de datos. En caso de que le bloque de origen ya exista, la cadena se recuperará completamente y directo de la base de datos.

F. Minado de bloques

El proceso de minería de bloques es esencial en una cadena de bloques para agregar nuevos bloques a la cadena de manera segura y descentralizada. Para realizarlo se implementa un método de minado en la clase *bloque*, que posteriormente es utilizado para agregar un bloque en la clase *cadena*.

El método de minado itera continuamente hasta que se encuentre un hash que cumpla con los requisitos de dificultad (igual a 4) del bloque, aumentando el valor del nonce para introducir variabilidad en el cálculo. En cada iteración se extrae un prefijo del hash actual, que tiene una longitud igual a la dificultad especificada y se compara con una cadena de ceros del mismo tamaño que la dificultad. Si el prefijo coincide con la cadena de ceros (dificultad alcanzada), se considera que el bloque se ha minado con éxito y se actualiza el hash del bloque con el valor actual. Finalmente, se imprime un mensaje indicando que el bloque ha sido minado con éxito.

V. Pruebas de Funcionamiento

Para realizar el testeo del sistema se debe iniciar la base de datos en una consola, luego para correr los nodos 1 y 2, se deben correr los comandos `npm start 1` y `npm start 2` en dos consolas distintas.

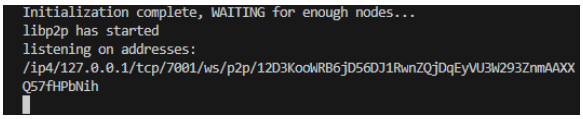

Fig. 2. Inicialización del nodo 1.

Fig. 3. Nodos ya iniciados y opciones de cada terminal.

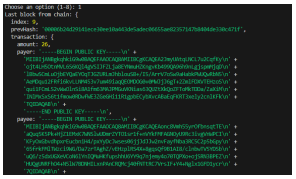

Fig. 4. Último bloque de la cadena.

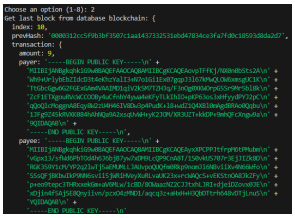

Fig. 5. Último bloque de la base de datos.

Fig. 6. Obtención del bloque Génesis de la base de datos.

Fig. 7. Envío de una transacción genérica.

Fig. 8. Envío de una transacción genérica.

Fig. 9. Envío de un mensaje a todos los nodos de la red.

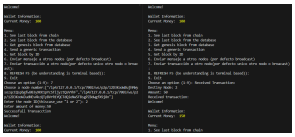

Fig. 10. Envío de fondos del nodo 1 al nodo 2, mostrando los cambios en los fondos.

VI. Conclusión

En conclusión, el proyecto ha logrado con éxito la implementación de un sistema de blockchain a pequeña escala, demostrando sus funcionalidades principales y estableciendo una base sólida para futuras mejoras y desarrollos. Las pruebas finales positivas son indicativas de la robustez del sistema en el contexto definido. A modo de resumen, se implementaron las principales funcionalidades: construcción de una blockchain con una estructura orientada a objetos, lectura de bloques en la cadena y en la base de datos, escritura y minado de bloques almacenando la información en la Level DB, un sistema de transacciones y billeteras seguro y persistente, implementación de un entorno de red con comunicación entre nodos utilizando Libp2p y una interfaz de usuario para mostrar las funcionalidades principales.