udp UNIVERSIDAD DIEGO PORTALES

Desarrollo de una red de Blockchain

Integrantes:
Gabriel González - Lukas Montero - Rodrigo Órdenes - Benjamín Tello

1. Motivación del Proyecto

En el contexto del mundo digital, las redes de blockchain emergen como una tecnología que introduce a un paradigma de registro y transferencia de datos en forma descentralizada y segura.
Al principio se mantuvo una gran curiosidad al mundo de las Blockchains porque las asociamos directamente a las criptomonedas, pero con el tiempo nos dimos cuenta de que abarca mucho más.
Cuando adquirimos mayor información, nuestras motivaciones para el proyecto mutaron a comprender el funcionamiento y mecanismos internos presentes de las redes de blockchain en un entorno práctico, aplicando una estructura de programación orientada a objetos, protocolos de comunicación y lógica compleja para lograr transferencia de información sincronizada.

2. Descripción de la Arquitectura de Software

La arquitectura de software de nuestra red de Blockchain abarca los siguientes alcances.

- Capa de infraestructura: La generación y comunicación entre nodos se configura con la librería “Libp2p”.
- Capa de datos: El manejo y almacenamiento de registros se aborda con programación orientada a objetos y base LevelDB.
- Capa de integración: La lógica empleada en el sistema asegura que todas las transacciones sean procesadas de manera segura y transparente.
- Capa de aplicación: Aunque el sistema no implementa directamente algoritmos de consenso avanzados, el REST API permite a los usuarios realizar transacciones y gestionar cuentas.

3. Implementación

En el presente proyecto se implementó una red de blockchain con nodos capaces de comunicarse entre sí y realizar transacciones de forma sincronizada.
La red de Blockchain está implementada principalmente en javascript con el framework NodeJS. Este sistema se estructura de los siguientes componentes.

- Modelos de Datos Utilizados:
    - Transaction: Almacena los detalles de las transacciones entre un emisor y un receptor.
    - Block: Contiene un conjunto de N transacciones.
    - Account: Incluye información de cuentas de usuario y sus respectivas billeteras.

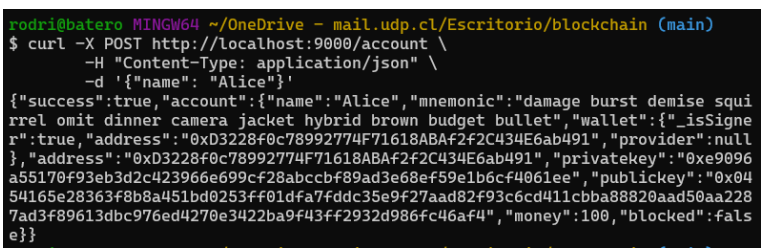
    - DBBlock y DBAccount: Estas clases manejan las operaciones de lectura y escritura en las bases de datos de bloques y cuentas, respectivamente.
        - Lectura: Recuperación de datos de bloques y cuentas, cómo obtener el último bloque o detalles de una cuenta específica.
        - Escritura: Registrar nuevas transacciones en bloques y actualizar o crear cuentas.
- Bloque de Origen
    - El sistema puede incluir un bloque de origen (genesis block) que sirve como punto de inicio de la cadena de bloques.
    - Este bloque inicial puede contener transacciones predefinidas o ajustes específicos para inicializar la red.
- Nodo: Cada nodo en la red blockchain es una instancia independiente que opera en el marco de NodeJS. Los nodos mantienen y validan la cadena de bloques.
- Protocolo de Comunicación:
    - Utiliza libp2p para la comunicación entre nodos, lo que permite una red descentralizada sin un ente central que gestione el flujo de tráfico.
    - Capacidad para realizar broadcasts de mensajes, asegurando que las transacciones y los bloques sean sincronizados entre todos los nodos.
    - Resiliente a fallos, permitiendo que la red continúe funcionando incluso si algunos nodos fallan.
- Solución de Doble Gasto
    - La red asegura que cada transacción sea única y se procese una sola vez.
    - Esto se logra mediante el bloqueo de cuentas y la verificación de cada transacción en los bloques antes de su confirmación y sincronización en toda la red. El bloqueo de cuentas es realizado mediante una variable booleana y cambia su estado cuando inicia y finaliza una transacción.

4. Pruebas de Funcionamiento

4.1 Pruebas REST API en escenario sin errores

Para comprobar una lógica correcta de funcionamiento del sistema, se realizan solicitudes hacia el servidor en un escenario en que las entradas de datos no tienen errores. Mostrando como respuesta el usuario generado con éxito. El alcance de estas pruebas abarca el resto de funcionalidades como “Mostrar Menú”, “Registrar Usuario”, “Generar Transacción”, “Obtener Cuenta por Dirección”, “Mostrar Bloques” y ”Bajar Nodo”. A continuación se muestran algunas de las pruebas.

- Registrar Cuenta:



- Generar Transacción:

```
rodri@batero MINGW64 ~/OneDrive - mail.udp.cl/Escritorio/blockchain (main)
$ curl -X POST http://localhost:9000/transaction -H "Content-Type: application
/json" -d '{"senderAddress": "0x58868a605c9a2F31Fb9D43ef95a02444b7079dAf", "re
cipientAddress": "0x2260A2374F4113Ba57d69494393652692084bb69", "amount": "2"}'
{"success":true,"transaction":{"sender":"Bob","recipient":"Alice","amount":"2"
,"signature":"7c6c4c91398b69d3fc0ed5688cbae83dc90319c1c423fcd0c746fab2f1b732fb
","valido":true,"nonce":1701304258994}}
```

4.2 Pruebas REST API con errores controlados

Para comprobar un manejo de errores adecuado, hemos implementado casos correspondientes en que las entradas en la API sean incorrectas o maliciosas. Hemos adherido las sentencias try-catch retornar códigos de respuesta erróneos definidos por HTTP. Nos hubiera gustado añadir filtros de inyecciones SQL y XSS debido a que como las funcionalidades están expuestas en una API, cuando escale a un aplicativo web podría haber cierto margen de riesgo a ataques cibernéticos. A continuación, se muestran algunas de las pruebas.

- Generar transacción con más saldo del que se tiene (Problema del doble gasto):

```
rodri@batero MINGW64 ~/OneDrive - mail.udp.cl/Escritorio/blockchain (main)
$ curl -X POST http://localhost:9000/transaction -H "Content-Type: application
/json" -d '{"senderAddress": "0x58868a605c9a2F31Fb9D43ef95a02444b7079dAf", "re
cipientAddress": "0x2260A2374F4113Ba57d69494393652692084bb69", "amount": "2000
"}'
{"error":"Insufficient funds in the sender account."}
```

- Generar transacción con saldo negativo (si es aceptado podría añadir más saldo):

```
rodri@batero MINGW64 ~/OneDrive - mail.udp.cl/Escritorio/blockchain (main)
$ curl -X POST http://localhost:9000/transaction -H "Content-Type: application
/json" -d '{"senderAddress": "0x58868a605c9a2F31Fb9D43ef95a02444b7079dAfs", "r
ecipientAddress": "0x2260A2374F4113Ba57d69494393652692084bb69", "amount": "-20
00"}'
{"error":"Internal server error"}
```

- Generar transacción con sender/recipient inválidos:

```
rodri@batero MINGW64 ~/OneDrive - mail.udp.cl/Escritorio/blockchain (main)
$ curl -X POST http://localhost:9000/transaction -H "Content-Type: application
/json" -d '{"senderAddress": "0x58868a605c9a2F31Fb9D43ef95a02444b7079dAfs", "r
ecipientAddress": "0x2260A2374F4113Ba57d69494393652692084bb69", "amount": "200
0"}'
{"error":"Internal server error"}
```

El alcance de estas pruebas abarca “Mostrar Menú”, “Registrar Usuario”, “Generar Transacción”, “Obtener Cuenta por Dirección”, “Mostrar Bloques” y ”Bajar Nodo”.

4.3 Pruebas de Nodos

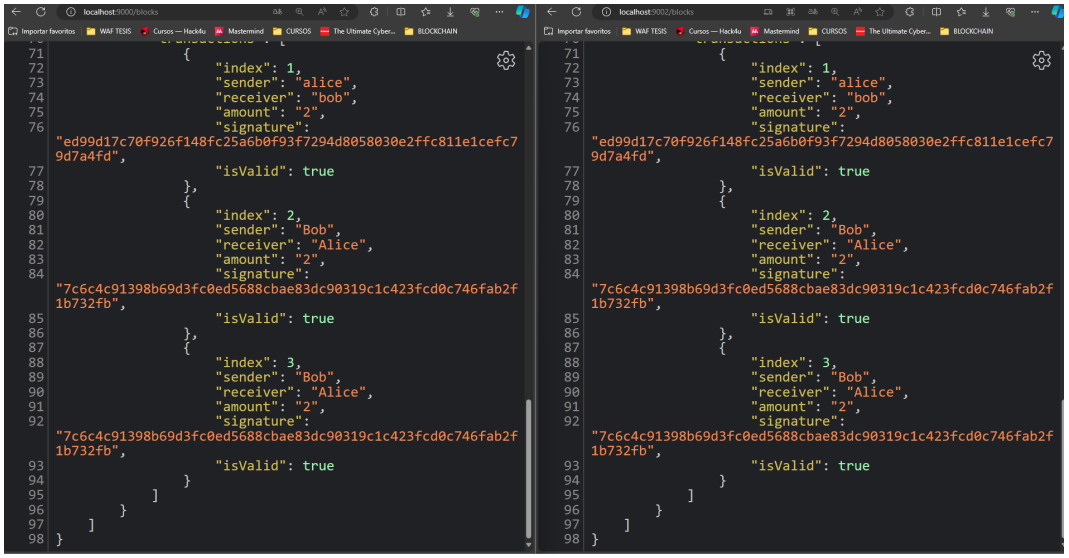
Se realizan pruebas a nivel de ejecución de la instancia del nodo.

- Ejecución del nodo sin especificar un puerto:

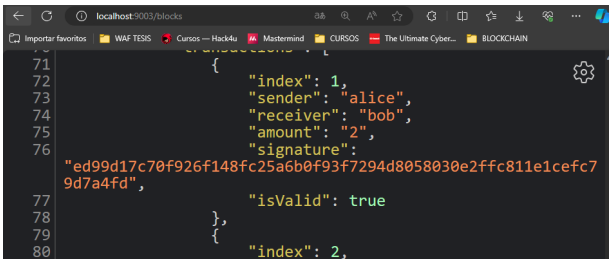
```
rodri@batero MINGW64 ~/OneDrive - mail.udp
.cl/Escritorio/blockchain/app (main)
$ node nodo1.js
Usage: node script.js <port>
```

- Funcionamiento de la red cuando un nodo se cae: Se da de baja uno de los nodos de la red, y se registra un usuario con éxito.

- Sincronización de los nodos: Al levantar el nodo se sincronizan las cuentas y la base de datos que almacenan los bloques.



- Sincronización de nuevo nodo: Al levantar un nodo nuevo, este se sincroniza con la información de los demás.



- No se pueden realizar transacciones de una misma dirección de origen al mismo tiempo (problema del doble gasto).