**UNIVERSIDAD NACIONAL DE LA PLATA**

**MAESTRÍA EN INGENIERÍA DE SOFTWARE**

**Trabajo Practico Final**

**Asignatura: Técnicas y Herramientas**

**Año: 2023**

**Título: BlockChain**

Autores: Gomes de Oliveira Neto, Luiz

Mascelloni, Mariela

Docente: Prof. Dr. Matías Urbieta

**DECISIONES DE DISEÑO:**

En la resolución del problema planteado, hemos considerado los siguientes puntos:

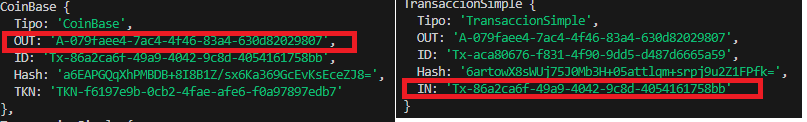
1. Se han creado dos nodos desde el inicio para abordar la situación descripta en el enunciado, donde un nodo está interconectado con otros nodos y el nuevo bloque se replica entre ellos.
2. Entre los distintos nodos, no se establece una conexión a través del hash, tal como ocurre con los bloques dentro de la blockchain local. Esto también se refleja en el diagrama.
3. Para realizar pruebas, hemos seguido los siguientes pasos:

* Inicialmente, se ha creado una única transacción Coinbase.
* Luego, se han iterado todas las transacciones simples (o normales) requeridas.
* Finalmente, se han creado transacciones compuestas. Para las transacciones compuestas, hemos realizado pruebas con conjuntos de transacciones, algunos de los cuales generan un nivel superior a 2. Esto se ha hecho para fines de validación.

1. Hemos asumido que las validaciones de las transacciones se basan en la correcta formación de las mismas. Por ejemplo, se ha verificado que los atributos como "OUT" y "ID" sigan el formato adecuado, como "A-<valor>" para "OUT", donde el valor debe ser un UUID válido.
2. Para actualizar los niveles hemos aplicado el patrón “Composite”.



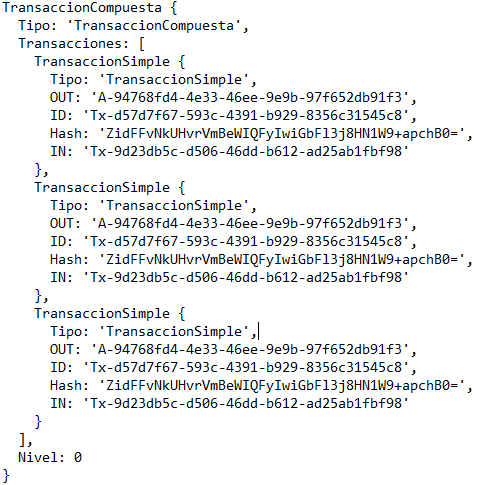
A continuación, se presenta un ejemplo de cómo se crea una transacción simple:



Se puede observar que en la transacción Coinbase, el atributo "OUT" coincide con el "OUT" de la última transacción. Además, en la transacción Simple, el atributo "IN" corresponde al "ID" de la Coinbase, que representa al propietario del token. Estas relaciones están diseñadas de manera que reflejen adecuadamente la transferencia de activos y la propiedad del token a lo largo de las transacciones.

A continuación, se presenta un ejemplo de cómo se crea unas transacciones compuestas:

Esta es una prueba de una transacción compuesta que involucra tres transacciones simples. En su interior, esta transacción compuesta contiene cuatro transacciones en total: la propia transacción compuesta y tres transacciones simples. Como resultado, el nivel de anidación de esta transacción compuesta es 0, ya que se encuentra en el nivel superior de la estructura.



Esta es una prueba de una transacción compuesta que incluye un total de 10 transacciones: 3 transacciones compuestas y 7 transacciones simples. Esta estructura presenta un nivel de anidación de 1.

**[compuesta]= [compuesta1]**

**Simple1**

**Simple2**

**Simple3**

**=[compuesta1]**

**Simple1**

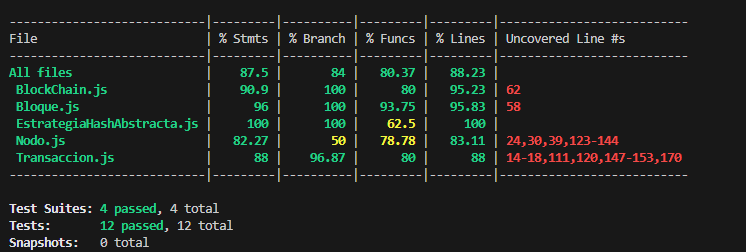
**Simple2**

**Simple3**

**= Simple 1**



A continuación, en la figura siguiente, presentamos los resultados obtenidos de las pruebas realizadas. Estos resultados ofrecen una visión integral de cómo nuestro sistema responde ante diferentes escenarios y condiciones, lo que nos permite evaluar su rendimiento, estabilidad y comportamiento en diversas situaciones.



**DOCUMENTACIÓN DEL PROYECTO**

Nuestra solución ha sido documentada siguiendo las siguientes prácticas con el fin de asegurar la comprensión, mantenibilidad y escalabilidad del código.

1. **Comentarios en el Código:**

Hemos implementado comentarios en el código para explicar secciones importantes, lógica compleja, decisiones de diseño y otra información relevante. Esto ayuda a los desarrolladores a comprender el propósito de diferentes partes del código y la lógica subyacente.

1. **Documentación de Funciones y Métodos:**

Cada función y método implementado ha sido documentado con comentarios explicativos que describen el propósito, los parámetros esperados y lo que se devuelve. Esto facilita la comprensión de funcionalidades específicas y cómo pueden ser utilizadas.

1. **Documentación de Clases:**

Cada clase en el código ha sido documentada detalladamente para explicar su responsabilidad, propiedades y métodos. Esto ayuda a los desarrolladores a entender la estructura general del proyecto y cómo interactúan las diferentes clases entre sí.

1. **Documentación de Módulos:**

Cada módulo del proyecto ha sido documentado para proporcionar una visión general de lo que hace el módulo y qué clases y funcionalidades contiene. Esto ayuda a organizar el código en módulos cohesivos y permite a los desarrolladores encontrar rápidamente las partes relevantes del código.

La documentación se encuentra en los propios archivos del código fuente, adjunta a los bloques de código relevantes. Esto permite que los desarrolladores accedan a la información directamente mientras exploran el código. La documentación es importante porque:

* **Facilita el Mantenimiento:** Comentarios detallados y documentación de clases permiten que otros desarrolladores comprendan rápidamente el propósito de diferentes partes del código, lo que hace que el mantenimiento sea más eficiente.
* **Acelera la Colaboración:** La documentación facilita que un equipo colabore en un proyecto, ya que todos tienen acceso a la información necesaria para contribuir de manera efectiva.
* **Reduce el Tiempo de Aprendizaje:** Los nuevos miembros del equipo pueden comprender rápidamente el proyecto, sus estructuras y funcionalidades a través de la documentación.
* **Evita Errores:** La comprensión clara del código a través de la documentación reduce la probabilidad de introducir errores durante el desarrollo o el mantenimiento.
* **Ayuda en la Escalabilidad:** A medida que el proyecto crece, la documentación permite que los desarrolladores encuentren rápidamente las partes relevantes del código, incluso en proyectos complejos.

**PATRONES UTILIZADOS EN EL PROYECTO**

Los patrones de diseño son soluciones probadas y estandarizadas para problemas comunes en el diseño de software. Estos patrones proporcionan pautas y estructuras reutilizables que pueden ser aplicadas en diversas situaciones para resolver problemas similares. Los patrones permiten a los diseñadores y desarrolladores de software abordar desafíos de manera eficiente y coherente, siguiendo prácticas que han demostrado ser efectivas en la industria.

En nuestro proyecto, utilizamos patrones de diseño para mejorar la calidad del código, facilitar el mantenimiento, promover la modularidad y la reusabilidad, y permitir la escalabilidad y extensibilidad del sistema. Algunas de las razones clave por las que utilizamos patrones de diseño son:

* **Reusabilidad:** Los patrones de diseño encapsulan soluciones probadas para problemas comunes. Al aplicar estos patrones, evitamos reinventar la rueda y utilizamos soluciones que han sido exitosas en otros contextos.
* **Legibilidad y Claridad:** Los patrones de diseño establecen una estructura clara y reconocible en el código. Esto hace que el código sea más legible para los desarrolladores, ya que siguen convenciones bien conocidas.
* **Mantenibilidad:** Los patrones de diseño promueven la separación de preocupaciones y la organización del código en componentes cohesivos. Esto facilita la identificación y resolución de problemas y la realización de cambios sin afectar otras partes del sistema.
* **Modularidad:** Los patrones de diseño ayudan a dividir el sistema en componentes independientes y reemplazables. Esto facilita la creación y gestión de módulos de manera aislada.
* **Escalabilidad y Extensibilidad:** Al aplicar patrones de diseño, creamos un diseño flexible que puede adaptarse a cambios futuros y nuevas funcionalidades sin alterar toda la estructura del sistema.
* **Buenas Prácticas:** Los patrones de diseño representan las mejores prácticas recopiladas a lo largo del tiempo por la comunidad de desarrollo de software. Utilizar estos patrones nos permite beneficiarnos de la sabiduría acumulada de expertos en el campo.

**PATRÓN DE DISEÑO**

En nuestro proyecto, hemos utilizado tanto el patrón de diseño "Strategy" como el "Composite" para gestionar diferentes estrategias de generación de hash en el proceso de creación de bloques. Creamos una clase abstracta llamada **EstrategiaHashAbstracta** que define el método **generarHash()**. Luego, implementamos las clases **MD5** y **SHA256** que heredan de esta clase abstracta y sobrescriben el método **generarHash()** para utilizar los algoritmos **MD5 y SHA256**, respectivamente.

Elegimos el patrón "Composite" porque es una estrategia efectiva que se refleja en la implementación de una transacción compuesta. Este patrón permite construir objetos complejos a partir de objetos más simples, lo que es útil en el contexto de transacciones compuestas, donde una transacción puede contener subtransacciones. Al utilizar el patrón "Composite", podemos tratar tanto a las transacciones individuales como a las compuestas de la misma manera, lo que simplifica la gestión y manipulación de transacciones en el sistema.

Este patrón es importante porque nos permite desacoplar el cálculo del hash del resto de la lógica de bloque y transacción. Esto facilita la extensibilidad ya que podemos agregar fácilmente nuevas estrategias de hash en el futuro sin afectar otras partes del sistema. Además, mejora la legibilidad del código al separar las diferentes implementaciones de hash en clases individuales y cohesivas.

El patrón Strategy se encuentra en las clases **EstrategiaHashAbstracta, MD5 y SHA256**. Encontrará este patrón en el manejo de cálculo de hash en la creación de bloques y transacciones, y su implementación se basa en la selección de algoritmo hash en función de la estrategia elegida.

Este uso de patrón Strategy demuestra nuestra preocupación por el modularidad y la extensibilidad en el diseño del proyecto, lo que facilita la adición y modificación de funcionalidades en el futuro sin causar impactos no deseados en otras partes del sistema.

En resumen, los patrones de diseño son herramientas valiosas para crear software de alta calidad, confiable y mantenible. En nuestro proyecto, utilizamos patrones de diseño para abordar desafíos específicos, como la gestión de estrategias de hash, y para asegurarnos de que nuestro código siga estándares reconocidos y prácticas recomendadas en la industria del desarrollo de software.