PTI práctica 2: BLOCKCHAIN

Joan Llonch Majó

Luis Jesús Valverde Zavaleta

Entorno y configuración de la práctica

El entorno que hemos utilizado para la realización de la práctica es una maquina virtual proporcionado por la UPC que es la Ubuntu22v3r1 y que la hemos corrido en VirtualBox.

Para configurar la maquina y tenerla preparada para la realización de la práctica, seguimos los comandos especificados en el enunciado de la asignatura. Nuestra maquina ya tenia de forma predeterminada la versión Python 3.10.12, simplemente ejecutamos los siguientes comandos en la terminal y tuvimos el entorno listo para comenzar a trabajar:

- sudo add-apt-repository ppa:deadsnakes/ppa
- sudo apt-get install python3-pip
- python3 -m pip install Flask requests
- python3 -m pip install Werkzeug
- python3 -m venv prbc
- source prbc/bin/actívate
- pip3 install flask requests werkzeug

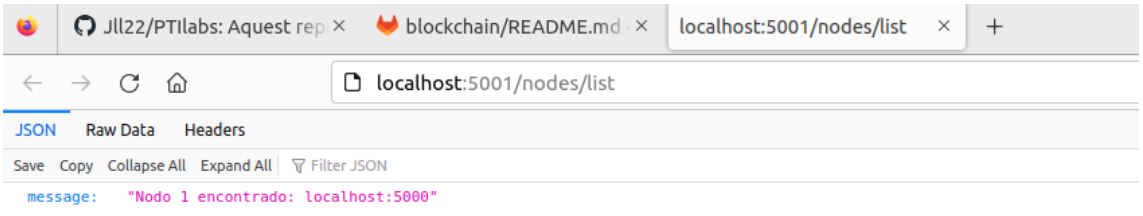
Realización de la práctica

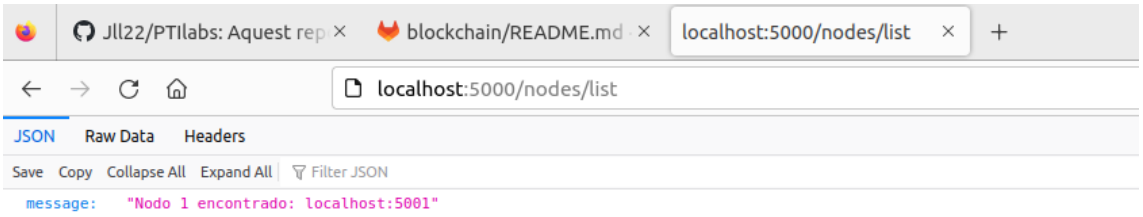
Lo primero que hicimos para la realización de la práctica fue la creación de un script llamado solo.sh con todos los comandos que hay especificados en el enunciado para correr los dos nodos con los que trabajaremos, para la escriptura de transacciones, la minería de bloques, etc. Después cambiamos los permisos para la ejecución del script.

El primer método que tuvimos que implementar fue el siguiente: “@app.route('/nodes/list', methods=['GET'])”, este método nos permite ver los nodos que están registrados en un nodo. La forma en la que lo implementamos fue la siguiente:

```
@app.route('/nodes/list', methods=['GET'])
def list_nodes():
    x = 1
    for node in blockchain.nodes:
        response = {
            'message': f'Nodo {x} encontrado: {node}',
        }
        x += 1
        return jsonify(response), 200
```

Cuando ejecutamos el script y nos redirigimos a las direcciones http://localhost:5001/nodes/list y http://localhost:5000/nodes/list , tuvimos el siguiente resultado:

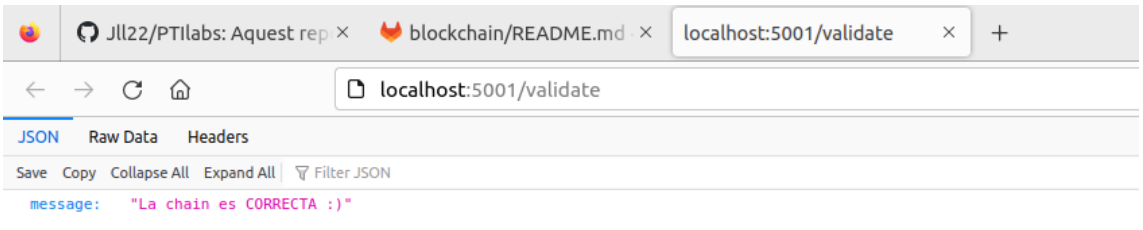


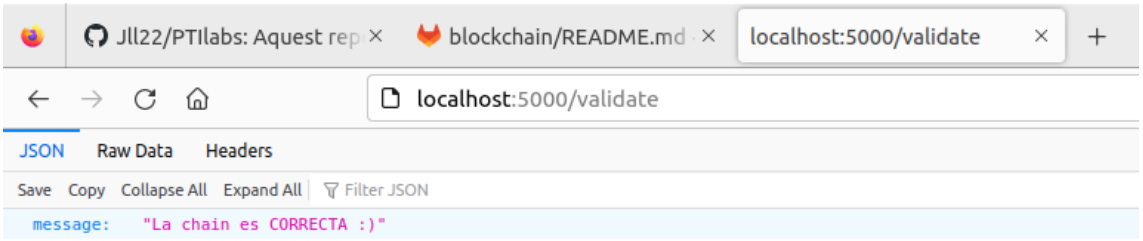


Después implementamos el segundo método propuesto por el enunciado, "@app.route('/validate', methods=['GET'])", este método se encargaba de validar la cadena de hashes de la blockchain almacenada en el nodo y mostraba un mensaje diciendo si era válida o inválida, el código programado fue el siguiente:

```
@app.route('/validate', methods=['GET'])
def validacio():
    if(blockchain.valid_chain(blockchain.chain)):
        response = {
            'message': 'La chain es CORRECTA :)'
        }
        return jsonify(response), 200
    else:
        response = {
            'message': 'La chain es INCORRECTA :('
        }
        return jsonify(response), 200
```

El resultado que obtuvimos fueron los siguientes, obviamente las chain que había en cada nodo eran válidas ya que el siguiente punto eran invalidarlas





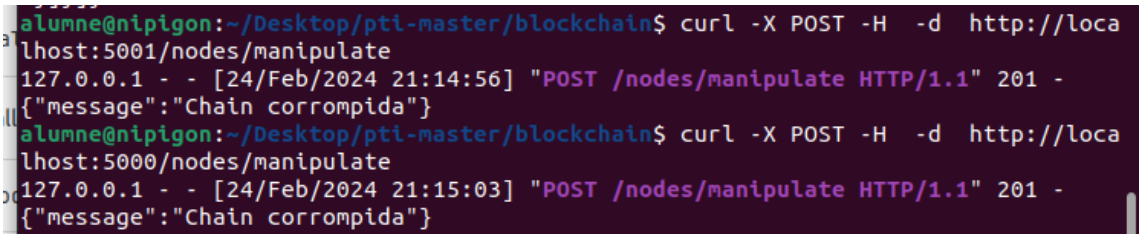
El último método que nos pidieron implementar fue el siguiente: "@app.route('/nodes/manipulate', methods=['POST'])", este método se encargaba de manipular la cadena de un nodo y así invalidarla y cuando llamemos al método programado anteriormente o yendo a la dirección http://localhost:5001/nodes/list ver que es incorrecta. El código programado de este método fue el siguiente:

```python
@app.route('/nodes/manipulate', methods=['POST'])
def manipulacio():

    # Cogemos el primer bloque, el bloque genesis
    bloque_troya = blockchain.chain[0]
    proof = blockchain.proof_of_work(bloque_troya)

    # Transacción que añadimos al siguiente bloque
    blockchain.new_transaction(
        sender="0",
        recipient=node_identifier,
        amount=1,
        order=0,
    )

    # Aqui es donde invalidamos la cadena
    # ya que le pasamos el hash y proof del bloque genesis
    # en vez del ultimo bloque de la secuencia
    previous_hash = blockchain.hash(bloque_troya)
    block = blockchain.new_block(proof, previous_hash)
    response = {
        'message': 'Chain corrompida'
    }
    return jsonify(response), 201
```

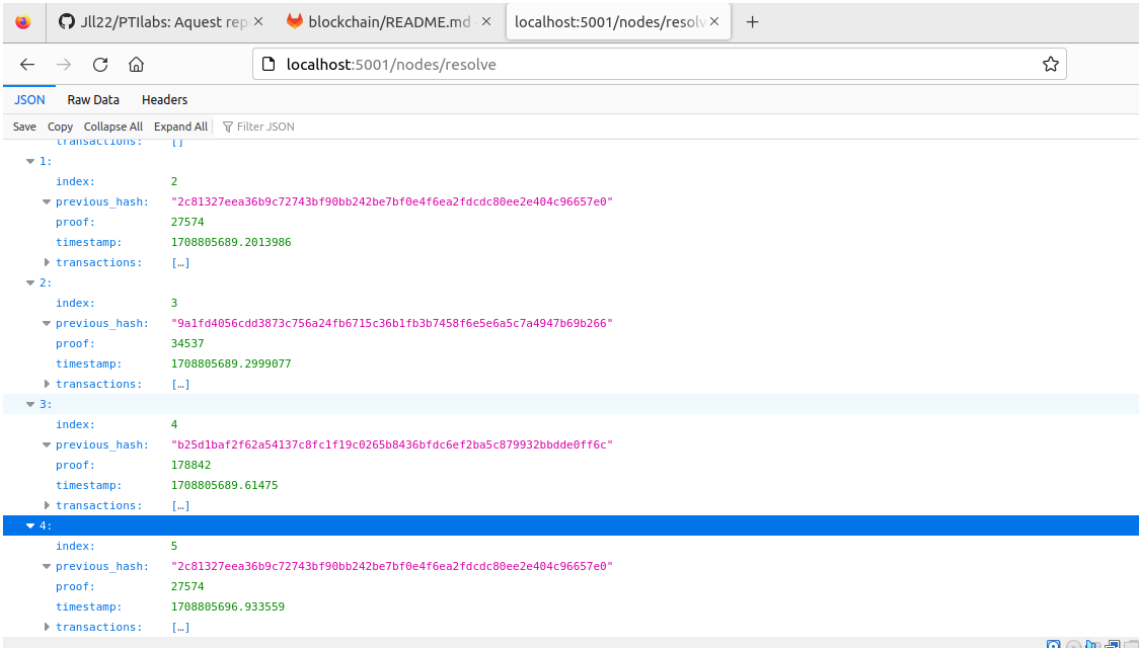
Nuestra forma de manipular la cadena fue añadiendo bloques a la cadena enviándole como previous_hash -> el hash del primer bloque y el proof del primer bloque también, en vez de enviar el previous_hash y el proof del ultimo bloque de la secuencia como tal. Para ejecutarlo hicimos los siguientes pasos:

1. Ejecutar el script solo.sh y tener las cadenas válidas en ambos nodos:

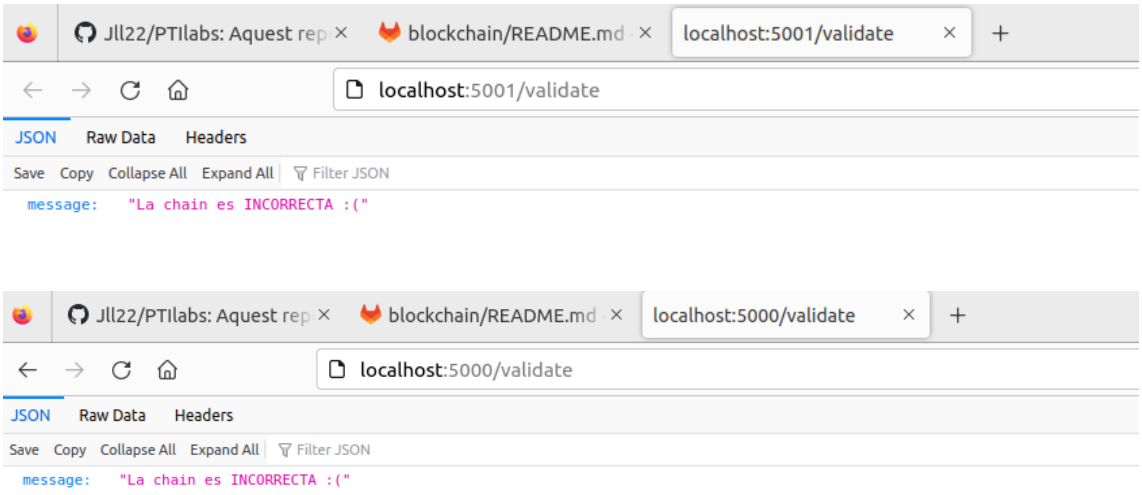
2. Ejecutar por terminal este comando para ambos nodos



3. Podemos observar que se han añadido un bloque nuevo con el previous_hash y proof que no deberían ya que son iguales a los del bloque con index = 2, en resumen el bloque 1 y 4 tienen ambos parámetros iguales.



4. El último paso fue comprobar que la cadena en ambos nodos era incorrecta:



Preguntas adicionales

Think of an alternative to the used proof-of-work. What options have you found/considered? Can this alternative be integrated in our code? If possible, make the modification needed for this alternative and test it.

Una alternativa que he encontrado es la de Proof-of-Stake, su objetivo es el mismo que PoW y es crear consenso entre todas las partes que integran la red. El sistema se basa en que cada nodo se llama validador y la decisión sobre que nodo valida un bloque se realiza teniendo en cuenta una serie de criterios, el principal criterio en el que se basa es en la cantidad de monedas que posees, ya que el proceso de elección es aleatorio como tal pero tus probabilidades de ser seleccionado aumentan en función de la cantidad de monedas que estén bajo tu posesión ya que esto se traduce como que tienes mayor participación en la red, pero el hecho de tener más cantidad no garantiza que seas elegido sino una probabilidad mayor. Una vez se haya hecho el proceso de elección, los elegidos tienen la capacidad de realizar unas tareas que se les permite, al realizar estas tareas se reciben unos incentivos y ganas peso en la participación proporcionalmente con el nivel de la tarea. Las principales ventajas respecto al proof-of-work son las siguientes:

- Mayor escalabilidad y velocidad, ya que la forma de minar en PoS es en forma de elección en función de tu participación en el sistema y en cambio en PoW todos compiten entre si requiriendo una gran inversión en poder computacional y en tiempo.
- A nivel de consumo energético es mucho menor, ya que PoW requiere de un gran poder de cómputo, que por lo general provienen de maquinas con un consumo elevado de electricidad, y en cambio PoS cambia el proceso de minería por un proceso de participación reflejada en la tenencia de monedas o en el tiempo dentro de la red.
- En PoW la acumulación de poder de minería se centra en unos grupos mineros concretos, consiguiendo que se centralice la red en manos de unos pocos, en cambio con PoS se busca un sistema más diversificado y democratizado, premiando a la participación de los usuarios otorgándoles diferentes tareas a realizar en la red.
- Un sistema más sólido ante los ataques del 51%, ya que en PoW con que reúnas el 51% de capacidad de computo de la red ya se puede llevara a cabo, en cambio con PoS solo

es posible si tienes bajo tu control el 51% de todas las monedas del sistema y si fuera el caso y el atacante realizara un ataque de este tipo, el valor de la moneda tiende a caer. Lo que conlleva a pérdidas económicas muy grandes para el atacante. Esta situación sirve de disuasivo para evitar estos ataques, manteniendo al mismo tiempo la seguridad de la red.

Se podría implementar pero requeriría de hacer varios cambios en el código y dedicar un tiempo considerable, la forma en la que cambiamos nuestro código fue la siguiente:

```python
import random

class ProofOfStake:
    def __init__(self):
        self.stakeholders = {}

    def register_stakeholder(self, address, balance):
        self.stakeholders[address] = balance

    def choose_block_creator(self):

        total_stake = sum(self.stakeholders.values())
        random_number = random.uniform(0, total_stake)
        current_sum = 0
        for address, balance in self.stakeholders.items():
            current_sum += balance
            if current_sum > random_number:
                return address
```

```python
class Blockchain:
    def __init__(self, proof_of_stake):
        self.current_transactions = []
        self.chain = []
        self.nodes = set()
        self.proof_of_stake = proof_of_stake

        # Create the genesis block
        self.new_block(previous_hash='1', proof=100)



    def add_block(self):
        block_creator = self.proof_of_stake.choose_block_creator()
        previous_hash = blockchain.hash(last_block)
        new_block = self.new_block(block_creator,previous_hash)
        return new_block
```

In our exercise, "everybody" can write a new transaction. Think about mechanisms to protect the execution of the transaction method (to answer in the report).

Un mecanismo para evitar que cualquiera pueda escribir una nueva transacción seria implementando el uso de contraseñas o algún mecanismo de validación en el que si lo pasas puedes escribir pero si no eres válido te impide realizar esta función.

Does our excercise store a state in the blockchain, e.g the amounts A and B have after having done a transaction? If not, how would you add it to the code (to answer in the report).

No, en nuestro sistema no guardamos el estado en la blockchain y una forma en la que se podría implantar seria añadiendo un atributo que indique el estado.

In our exercise, are the data of the transactions private? How is the openness of the data related with the validation of the blocks? Is there any technique/technology (search in Internet) that you could apply to our exercise to increase the privacy of the data of the transaction while still allowing the validation of the blockchain?

En nuestro ejercicio los datos de las transacciones no son privados.

La apertura de los datos está relacionada con la validación de los bloques, en el sentido de que todos los nodos de la red deben estar de acuerdo en la validez de las transacciones y los bloques. La transparencia de los datos permite a todos los nodos verificar de forma independiente las transacciones y llegar a un consenso sobre el estado de la cadena de bloques.

Para aumentar la privacidad de los datos de las transacciones y seguir permitiendo la validación de la blockchain, una técnica que podría ser aplicada es la de la encriptación. Un ejemplo de tecnología que podríamos utilizar sería la de ZKPs (“zero-knowledge proofs”), que nos permitiría a un nodo demostrar a otro nodo que la informaciñon es verdadera sin revelar ninguna información más allá de la validez. ZKPs podría usarse para validar transacciones sin revelar los datos sensibles como el sender, recipient o el amount, esto aseguraría privacidad mientras mantenemos la integridad y la validez de la blockchain. Para la implementación de ZKPs usaríamos protocolos criptográficos como zk-SNARKs o zk_STARKs.