

# Blockchain

Alumnos:

- Jordi Solé García (Grupo 11)
- Víctor Gallego Izquierdo (Grupo 11)

## Objetivos de la práctica

El objetivo de esta práctica es añadir funcionalidades a una Web API creada en el lenguaje Python y con la ayuda de las librerías `Flask` para el uso del cliente HTTP para realizar operaciones con este. Además, seguiremos usando Curl.

A partir de una funciones ya dadas, tenemos que completar esta API con tres nuevas funcionalidades:

- **/nodes/list:** Permite ver los otros nodos registrados los cuales se han registrado en un nodo.
- **/validate:** Permite validar la cadena de hashes de la blockchain almacenada en un nodo.
- **/nodes/manipulate:** Permite manipular la cadena de un nodo, haciendo que si se valida, hará incorrecta.

## Tareas a realizar

### Configuración del entorno

El primer paso es configurar una serie de librerías de Python, y este si no lo tenemos instalado. Primero comprobamos si Python está instalado, para ello ejecutamos `python3 --version`. En nuestro caso ya está instalado, por lo que podemos continuar con los siguientes pasos. Ahora tenemos que añadir el programa que permite instalar las librerías, `pip` e instalamos las librerías pertinentes. Para ello ejecutamos lo siguiente:

```
sudo apt-get install python3-pip  
python3.6 -m pip install Flask requests  
python3.6 -m pip install Werkzeug==0.16
```

Una vez tenemos ejecutado esto, hacemos una ejecución del programa para ver que funcione correctamente con el comando `python3 blockchain.py -p 5000`.

# Añadir nuevas funcionalidades

## /nodes/list

En esta función hay que mostrar quienes son los nodos registrados en el nodo en el que se ejecuta la función.

Para ello, añadimos esta función al código:

```
@app.route('/nodes/list', methods=['GET'])
def listar():
    response = {
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 200
```

Esta función mostrará la lista de nodos que tiene la blockchain, previamente registrados con la función `register_nodes()`. Se creará una respuesta en formato JSON que se mostrará en el navegador.

## /validate

En esta función tenemos que validar la cadena de hashes que tiene la blockchain guardada en un nodo concreto. Para hacerlo, añadimos la siguiente función.

```
@app.route('/validate', methods=['GET'])
def validar():
    if blockchain.valid_chain(blockchain.chain):
        response = {
            'message': 'Chain OK',
        }
    else:
        response = {
            'message': 'Error al validar'
        }
    return jsonify(response), 200
```

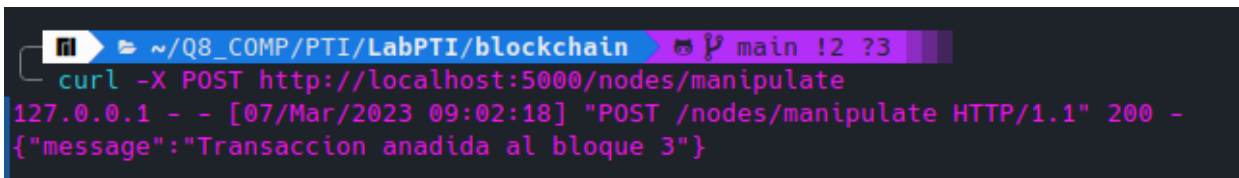
Esta función usa `valid_chain(blockchain.chain)` que viene ya creada en la práctica para validar la cadena. Esta devuelve un Boolean. En caso de ser o no correcta, creamos una respuesta en JSON con el correspondiente mensaje en cada caso.

## /nodes/manipulate

La última función consiste en manipular el chain de un nodo y cuando este esté validado, con la función `validar()`, a partir de este momento será incorrecto. Añadimos el siguiente código:

```
@app.route('/nodes/manipulate', methods=['POST'])
def manipular():
    del blockchain.chain[-1]
    block = blockchain.new_transaction('D', 'Víctor', '2023',
    '23')
    response = {
        'message': f'Transaccion añadida al bloque
    {block}'
    }
    return jsonify(response), 200
```

Para comprobar que esto funciona correctamente, podemos usar el comando `curl -X POST http://localhost:5000/nodes/manipulate` y nos mostrará el siguiente mensaje en el terminal:

A terminal window with a dark background. The prompt shows the current directory as ~/Q8\_COMP/PTI/LabPTI/blockchain and the shell as main !2 ?3. The command curl -X POST http://localhost:5000/nodes/manipulate has been executed. The output shows a successful HTTP response from 127.0.0.1 with status 200 and a JSON body: {"message": "Transaccion anadida al bloque 3"}.

```
~/Q8_COMP/PTI/LabPTI/blockchain main !2 ?3
curl -X POST http://localhost:5000/nodes/manipulate
127.0.0.1 - - [07/Mar/2023 09:02:18] "POST /nodes/manipulate HTTP/1.1" 200 -
{"message": "Transaccion anadida al bloque 3"}
```

## Preguntas adicionales

1. **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.**

Para que un sistema de Proof of Work sea realmente útil, no puede colapsar la red, por lo que las transacciones tienen que tener una carga de trabajo alta y difícil de resolver pero que la validación por parte de los demás sea una tarea sencilla pero que no comprometa la seguridad de las transacciones.

Existen diferentes tipos de Proof of Work:

- HashCash:

Referencias:

- [bit2me](#)
- [Academia Finanzas](#)

2. **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).**

Existen cuatro principios fundamentales que una blockchain tiene que seguir para ser segura en las transacciones ([Portal del comerciante](#)) :

1. **Principio de autenticidad:** que la persona o empresa que dice estar al otro lado de la red es quién dice ser.
2. **Principio de integridad:** que lo transmitido a través de la red no haya sido modificado.
3. **Principio de intimidad:** que los datos transmitidos no hayan sido vistos durante el trasiego telemático.
4. **Principio de no repudio:** que lo transmitido no pueda ser repudiado o rechazado.

3. **Does our exercise 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 se almacena ningún tipo de estado. Solo se almacenan los siguientes datos: "amount", "order", "recipient" y "sender". Para poder guardar el estado, solo tendríamos que añadir un nuevo parametro "state" y en la función `new_transaction` añadir este parámetro e iniciarlo correctamente.