

Contrôle Continu

Implémentation d'une Blockchain en Python

Guinga Aly

Filière : Sciences des Données, Big Data & IA

Module : Fondamentaux de la Blockchain

Prof. Imad Sassi

Table des matières

1	Introduction	2
2	Partie I : Blockchain Mononœud	2
2.1	Architecture et Concepts Clés	2
2.2	Hachage SHA-256	2
2.3	Preuve de Travail (PoW)	2
2.4	API REST - Endpoints Partie I	2
2.5	Tests et Résultats - Partie I	2
2.5.1	Installation et Démarrage	2
2.5.2	Test 1 : Consultation de la chaîne initiale	3
2.5.3	Test 2 : Création de transaction et minage	3
2.5.4	Test 3 : Validation de la chaîne	3
3	Partie II : Décentralisation et Consensus	3
3.1	Réseau de Nœuds	3
3.2	Algorithme de Consensus	4
3.3	API REST - Endpoints Partie II	5
3.4	Tests et Résultats - Partie II	5
3.4.1	Configuration Multi-Nœuds	5
3.4.2	Test 4 : Enregistrement de noeuds	5
3.4.3	Test 5 : Création d'un conflit	5
3.4.4	Test 6 : Résolution du consensus	6
4	Conclusion	6

1 Introduction

Ce rapport présente l'implémentation d'une blockchain fonctionnelle en Python réalisée dans le cadre du Contrôle Continu. Le travail est divisé en deux parties : la création d'une blockchain mononœud (Partie I) et son extension vers un réseau décentralisé avec consensus (Partie II).

2 Partie I : Blockchain Mononœud

2.1 Architecture et Concepts Clés

Notre implémentation repose sur la classe `Blockchain` qui gère :

- La chaîne de blocs (`self.chain`)
- Les transactions en attente (`self.current_transactions`)
- Le bloc genesis créé à l'initialisation

Chaque bloc contient : index, timestamp, transactions, preuve de travail (PoW) et hash du bloc précédent.

2.2 Hachage SHA-256

Le hachage utilise `sort_keys=True` pour garantir un résultat déterministe :

Listing 1 – Fonction de hachage

```
1 @staticmethod
2 def hash(block):
3     block_string = json.dumps(block, sort_keys=True).encode()
4     return hashlib.sha256(block_string).hexdigest()
```

2.3 Preuve de Travail (PoW)

Difficulté fixée à '0000' (4 zéros). L'algorithme incrémente un nonce jusqu'à trouver un hash valide :

Listing 2 – Proof of Work

```
1 def proof_of_work(self, last_proof):
2     proof = 0
3     while self.valid_proof(last_proof, proof) is False:
4         proof += 1
5     return proof
6
7 @staticmethod
8 def valid_proof(last_proof, proof):
9     guess = f'{last_proof}{proof}'.encode()
10    guess_hash = hashlib.sha256(guess).hexdigest()
11    return guess_hash.startswith("0000")
```

2.4 API REST - Endpoints Partie I

2.5 Tests et Résultats - Partie I

2.5.1 Installation et Démarrage

Listing 3 – Installation

```
pip install flask requests
python blockchain.py
```

Endpoint	Méthode	Description
/mine	GET	Mine un nouveau bloc avec récompense
/transactions/new	POST	Ajoute une transaction
/get_chain	GET	Retourne la blockchain complète
/is_valid	GET	Vérifie l'intégrité de la chaîne

TABLE 1 – Endpoints Partie I

```
(.venv) PS C:\Users\aligu\Desktop\else\BlockChain-example> python ./blockchain.py
* Serving Flask app 'blockchain'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://100.91.147.139:5000
Press CTRL+C to quit
```

FIGURE 1 – Démarrage du serveur

2.5.2 Test 1 : Consultation de la chaîne initiale

```
Invoke-WebRequest -Uri http://localhost:5000/get_chain | Select-Object -ExpandProperty Content
```

```
PS C:\Users\aligu\Desktop\else\BlockChain-example> Invoke-WebRequest -Uri http://localhost:5000/get_chain | Select-Object -ExpandProperty Content
{"chain":[{"index":1,"previous_hash":"1","proof":100,"timestamp":1762807432.6443017,"transactions":[]},"length":1}
PS C:\Users\aligu\Desktop\else\BlockChain-example>
```

FIGURE 2 – Blockchain initiale (bloc genesis)

2.5.3 Test 2 : Création de transaction et minage

```
# Creer une transaction
$body = @{"sender="Alice"; recipient="Bob"; amount=10} | ConvertTo-Json
Invoke-WebRequest -Uri http://localhost:5000/transactions/new `

# Miner le bloc
Invoke-WebRequest -Uri http://localhost:5000/mine
```

2.5.4 Test 3 : Validation de la chaîne

```
Invoke-WebRequest -Uri http://localhost:5000/is_valid
```

3 Partie II : Décentralisation et Consensus

3.1 Réseau de Nœuds

Implémentation d'un système d'enregistrement des noeuds voisins utilisant un `set` pour garantir l'idempotence :

```

● PS C:\Users\aligu\Desktop\else\BlockChain-example> Invoke-WebRequest -Uri http://localhost:5000/nodes/resolve

StatusCode      : 200
StatusDescription : OK
Content         : {"chain":[{"index":1,"previous_hash":"1","proof":100,"timestamp":1762807432.6443017,"transactions":[]},{"index":2,"previous_hash":"c8bcf8a639e7685a747dce26d56c9b7e1759ed3c85a1f44ba2aba49707c102dc","pr...
RawContent      : HTTP/1.1 200 OK
                  Connection: close
                  Content-Length: 671
                  Content-Type: application/json
                  Date: Mon, 10 Nov 2025 21:11:17 GMT
                  Server: Werkzeug/3.1.3 Python/3.13.6

                  {"chain":[{"index":1,"previous_has...
Forms          : {}
Headers        : {[Connection, close], [Content-Length, 671], [Content-Type, application/json], [Date, Mon, 10 Nov 2025 21:11:17 GMT]...}
Images         : {}
InputFields    : {}
Links          : {}
ParsedHtml     : mshtml.HTMLDocumentClass
RawContentLength : 671

```

FIGURE 3 – Résultat du minage avec récompense

```

PS C:\Users\aligu\Desktop\else\BlockChain-example> Invoke-WebRequest -Uri http://localhost:5000/is_valid | Select-Object -ExpandProperty Content
● Object -ExpandProperty Content
{"message":"La blockchain est valide."}

○ PS C:\Users\aligu\Desktop\else\BlockChain-example>

```

FIGURE 4 – Validation de l'intégrité

Listing 4 – Enregistrement de nœuds

```

1 def register_node(self, address):
2     parsed_url = urlparse(address)
3     if parsed_url.netloc:
4         self.nodes.add(parsed_url.netloc)
5     elif parsed_url.path:
6         self.nodes.add(parsed_url.path)

```

3.2 Algorithme de Consensus

Implémentation de la règle de "la chaîne la plus longue" :

Listing 5 – Résolution des conflits

```

1 def resolve_conflicts(self):
2     neighbours = self.nodes
3     new_chain = None
4     max_length = len(self.chain)
5
6     for node in neighbours:
7         response = requests.get(f'http:///{node}/get_chain')
8         if response.status_code == 200:
9             length = response.json()['length']
10            chain = response.json()['chain']
11
12            if length > max_length and self.is_chain_valid(chain):
13                max_length = length
14                new_chain = chain
15
16            if new_chain:
17                self.chain = new_chain
18                return True
19
20        return False

```

3.3 API REST - Endpoints Partie II

Endpoint	Méthode	Description
/nodes/register	POST	Enregistre des nœuds voisins
/nodes/resolve	GET	Lance l'algorithme de consensus

TABLE 2 – Endpoints Partie II

3.4 Tests et Résultats - Partie II

3.4.1 Configuration Multi-Nœuds

Lancement de deux instances sur les ports 5000 et 5001 :

```

PS C:\Users\aligu\Desktop\else\BlockChain-example> python on blockchain_node1.py
* Serving Flask app 'blockchain_node1'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://106.91.147.139:5000
Press CTRL+C to quit

PS C:\Users\aligu\Desktop\else\BlockChain-example> python on blockchain_node2.py
* Serving Flask app 'blockchain_node2'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5001
* Running on http://106.91.147.139:5001
Press CTRL+C to quit

```

FIGURE 5 – Deux nœuds actifs

3.4.2 Test 4 : Enregistrement de nœuds

```
$body = @{
    nodes = @("http://localhost:5001")
} | ConvertTo-Json
Invoke-WebRequest -Uri http://localhost:5000/nodes/register `-
    -Method POST -Body $body -ContentType "application/json"
```

```

PS C:\Users\aligu\Desktop\else\BlockChain-example> $body = @{
    >>>     nodes = @("http://localhost:5000", "http://localhost:5001")
    >>> } | ConvertTo-Json
    >>
    >> Invoke-WebRequest -Uri http://localhost:5000/nodes/register -Method POST -Body $body -ContentType "application/json" | Select-Object -ExpandProperty Content
    {"message":"Nouveaux n\u00e9uds ajout\u00e9s avec succ\u00e9s","total_nodes":["localhost:5002","localhost:5000","localhost:5001"]}

PS C:\Users\aligu\Desktop\else\BlockChain-example>

```

FIGURE 6 – Enregistrement réussi

3.4.3 Test 5 : Cr\u00e9ation d'un conflit

Cr\u00e9ation de cha\u00eenes de longueurs diff\u00e9rentes :

- N\u00e9ud 1 (port 5000) : 3 blocs
- N\u00e9ud 2 (port 5001) : 4 blocs

```

PS C:\Users\aligu\Desktop\else\BlockChain-example> # Miner 2 blocs
>> Invoke-WebRequest -Uri http://localhost:5000/mine | Select-Object -ExpandProperty Content
>> Invoke-WebRequest -Uri http://localhost:5000/mine | Select-Object -ExpandProperty Content
>>
● >> # Vérifier la longueur
>> Invoke-WebRequest -Uri http://localhost:5000/get_chain | Select-Object -ExpandProperty Content
{"index":4,"message":"Nouveau bloc miné avec succès !","previous_hash":"86d32a0995b906c12cad08161e2fc7070e50b0532b65c06b74efb0c55f748fc7","proof":119678,"timestamp":1762810009.9831874,"transactions":[{"amount":10,"recipient":"Bob","sender":"Alice"}, {"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}

{"index":5,"message":"Nouveau bloc miné avec succès !","previous_hash":"79b23b047d85052df5c0a1cf0d6d30e341947cdac92a91e0800d18c88f4d338","proof":146502,"timestamp":1762810010.2497206,"transactions":[{"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}

{"chain":[{"index":1,"previous_hash":"1","proof":100,"timestamp":1762807432.6443017,"transactions":[]}, {"index":2,"previous_hash":"c8bcf8a639e7685a747dce26d56c9b7e1759ed3c85a1f44ba2aba49707c102dc","proof":35293,"timestamp":1762808229.7716236,"transactions":[{"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}, {"index":3,"previous_hash":"7cab988faf6f6f17a3feee9bfa0b057a3e65a5ad341cb04352e87b67769","proof":35089,"timestamp":1762808475.6822434,"transactions":[{"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}, {"index":4,"previous_hash":"86d32a0995b906c12cad08161e2fc7070e50b0532b65c06b74efb0c55f748fc7","proof":119678,"timestamp":1762810009.9831874,"transactions":[{"amount":10,"recipient":"Bob","sender":"Alice"}, {"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}, {"index":5,"previous_hash":"79b23b047d85052df5c0a1cf0d6d30e341947cdac92a91e0800d18c88f4d338","proof":146502,"timestamp":1762810010.2497206,"transactions":[{"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}], "length":5}

```

FIGURE 7 – Conflit : chaînes de longueurs différentes

3.4.4 Test 6 : Résolution du consensus

```
Invoke-WebRequest -Uri http://localhost:5000/nodes/resolve
```

```

● PS C:\Users\aligu\Desktop\else\BlockChain-example> Invoke-WebRequest -Uri http://localhost:5000/nodes/resolve | Select-Object -ExpandProperty Content
{"chain": [{"index":1,"previous_hash":"1","proof":100,"timestamp":1762807432.6443017,"transactions":[]}, {"index":2,"previous_hash":"c8bcf8a639e7685a747dce26d56c9b7e1759ed3c85a1f44ba2aba49707c102dc","proof":35293,"timestamp":1762808229.7716236,"transactions":[{"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}, {"index":3,"previous_hash":"7cab988faf6f6f17a3feee9bfa0b057a3e65a5ad341cb04352e87b67769","proof":35089,"timestamp":1762808475.6822434,"transactions":[{"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}, {"index":4,"previous_hash":"86d32a0995b906c12cad08161e2fc7070e50b0532b65c06b74efb0c55f748fc7","proof":119678,"timestamp":1762810009.9831874,"transactions":[{"amount":10,"recipient":"Bob","sender":"Alice"}, {"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}, {"index":5,"previous_hash":"79b23b047d85052df5c0a1cf0d6d30e341947cdac92a91e0800d18c88f4d338","proof":146502,"timestamp":1762810010.2497206,"transactions":[{"amount":1,"recipient":"5c35d393c43b4e85992c3c7a664281b4","sender":"0"}]}], "message": "Notre chaîne fait autorité !"}

```

FIGURE 8 – Consensus résolu : chaîne remplacée

Résultat : Le noeud 1 adopte la chaîne du noeud 2 (4 blocs). Les deux noeuds sont maintenant synchronisés.

4 Conclusion

Cette implémentation a permis d'implémenter une blockchain fonctionnelle comprenant :

Partie I :

- Structure de blocs avec hachage SHA-256
- Preuve de Travail (difficulté '0000')
- Système de transactions avec récompense de minage
- Validation de l'intégrité de la chaîne
- API REST avec 4 endpoints

Partie II :

- Système de réseau décentralisé
- Enregistrement de noeuds voisins
- Algorithme de consensus (chaîne la plus longue)
- Résolution automatique des conflits

Tous les tests ont été effectués avec succès, démontrant le bon fonctionnement de l'implémentation conforme aux exigences du sujet.