

Mini-Rapport

Blockchain Practice

Tarek ATBI

02 Février 2024

Introduction

Dans le cadre de notre unité Blockchain, j'ai réalisé plusieurs exercices visant à exploiter l'API Blockstream (**Esplora**) pour interroger des données de la blockchain Bitcoin. Ces exercices nous ont permis de mettre en œuvre des concepts clés tels que la récupération de transactions, la gestion de la pagination, l'analyse d'intervalles de blocs, et la vérification d'une preuve de Merkle.

L'objectif de ce rapport est de présenter de manière détaillée les solutions techniques mises en place pour chaque exercice.

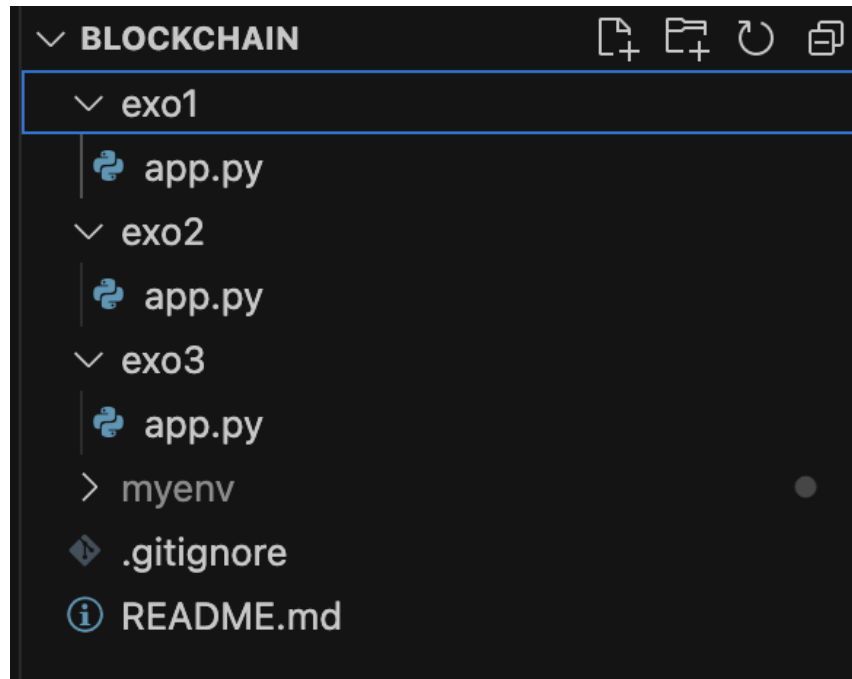


Figure -1- : Architecture de développement mise en place

1. Exercice 1 : Récupération de la transaction avec le plus gros montant échangé

Énoncé

L'exercice consiste à écrire un programme qui récupère le contenu d'un bloc (dont le hash est passé en argument) via l'API Blockstream, puis qui identifie et renvoie la transaction ayant le plus gros montant échangé (calculé comme la somme des valeurs des sorties de la transaction).

Méthodologie et Approche

- Endpoints utilisés :
 - `GET /block/<block_hash>/txs` Récupère les 25 premières transactions du bloc.
 - `GET /block/<block_hash>/txs[:start_index]` Pour récupérer les transactions d'un bloc (pagination).
- Étapes principales :
 - Récupérer toutes les transactions du bloc.
 - Calculer, pour chaque transaction, la somme des valeurs des sorties (`vout`).
 - Identifier la transaction avec le montant total le plus élevé.

Code et Explications

Ce code récupère de manière paginée toutes les transactions d'un bloc en appelant l'endpoint sans index pour la première page, puis en utilisant l'index de départ pour les pages suivantes. Il incrémente l'indice en fonction du nombre de transactions récupérées jusqu'à atteindre le total indiqué par `tx_count`. Ce mécanisme garantit que toutes les transactions du bloc sont obtenues pour une analyse complète.

```

1 while start_index < tx_count:
2     if start_index == 0:
3         url = f"{base_url}/block/{block_hash}/txs"
4     else:
5         url = f"{base_url}/block/{block_hash}/txs/{start_index}"
6
7     print(f"Fetching page {page} (start_index = {start_index})")
8     try:
9         r = session.get(url, timeout=10)
10        r.raise_for_status()
11    except requests.exceptions.RequestException as e:
12        print(f"Erreur lors de la récupération de la page {page}: {e}")
13        break
14
15    txs = r.json()
16    count = len(txs)
17    print(f"Page {page}: {count} transactions récupérées")
18    if count == 0:
19        break
20
21    transactions.extend(txs)
22    start_index += count
23    page += 1

```

Figure -2- : Code Exercice 1

Résultats et Analyse

```
▼ TERMINAL
```

```
(myenv) tarekatbi@MacBook-Pro-de-ATBI blockchain % cd exo1  
(myenv) tarekatbi@MacBook-Pro-de-ATBI exo1 % python app.py 00000000000000000000f27178b53398a352501c87ac9b507e26a5994cae2ce0  
Récupération des transactions pour le bloc : 00000000000000000000f27178b53398a352501c87ac9b507e26a5994cae2ce0  
Le bloc 00000000000000000000f27178b53398a352501c87ac9b507e26a5994cae2ce0 contient 93 transactions.  
Fetching page 1 (start_index = 0)  
Page 1: 25 transactions récupérées  
Fetching page 2 (start_index = 25)  
Page 2: 25 transactions récupérées  
Fetching page 3 (start_index = 50)  
Page 3: 25 transactions récupérées  
Fetching page 4 (start_index = 75)  
Page 4: 18 transactions récupérées  
Total des transactions récupérées: 93  
Analyse des transactions pour déterminer celle avec le plus gros montant échangé...  
  
Transaction avec le plus gros montant échangé :  
TXID          : f27afaff51c2ffc16e8896225cac433ca2073aef0169b03a79108e22cc99bf08  
Valeur totale échangée : 1581987909 satoshis  
(myenv) tarekatbi@MacBook-Pro-de-ATBI exo1 %
```

Figure -3- : Résultat Exercice 1

Ce résultat démontre que le script récupère correctement les transactions et identifie celle qui a le plus gros montant échangé.

2. Exercice 2 : Analyse d'un intervalle de blocs

Énoncé

Écrire un programme qui prend en argument deux entiers $h1$ et $h2$ (avec $h1 \leq h2$) représentant les hauteurs de deux blocs. Le programme doit renvoyer l'adresse Bitcoin qui apparaît le plus souvent dans les transactions des blocs compris entre les deux, ainsi que le nombre d'apparitions.

Méthodologie et Approche

- Endpoints utilisés :
 - `GET /block-height/<height>` pour obtenir le hash d'un bloc à partir de sa hauteur.
 - `GET /block/<block_hash>/txs[:start_index]` pour récupérer les transactions du bloc.
- Étapes principales :
 - Pour chaque hauteur, récupérer le hash du bloc.
 - Extraire toutes les transactions du bloc.
 - Pour chaque transaction, parcourir les sorties (`vout`) et extraire le champ `"scriptpubkey_address"`.
 - Compter les apparitions de chaque adresse et déterminer l'adresse la plus fréquente dans l'intervalle.

Code et Explications

```
1 # Dictionnaire pour compter les apparitions de chaque adresse
2 address_counts = defaultdict(int)
3
4 for height in range(h1, h2 + 1):
5     print(f"\nTraitement du bloc de hauteur {height}...")
6     block_hash = get_block_hash(height)
7     if not block_hash:
8         print(f" Bloc à la hauteur {height} introuvable.")
9         continue
10    transactions = get_block_transactions(block_hash)
11    print(f" Nombre de transactions dans le bloc {height} : {len(transactions)}")
12    for tx in transactions:
13        vouts = tx.get("vout", [])
14        for vout in vouts:
15            address = vout.get("scriptpubkey_address")
16            if address:
17                address_counts[address] += 1
```

Ce code parcourt chaque bloc de l'intervalle [h1, h2], récupère le hash et les transactions de chaque bloc, puis itère sur les sorties de chaque transaction pour extraire l'adresse Bitcoin via le champ "`scriptpubkey_address`". Pour chaque adresse trouvée, il incrémente un compteur dans un dictionnaire. Ainsi, il permet de compter le nombre d'apparitions de chaque adresse dans l'intervalle spécifié.

Résultats et Analyse

```
✓ TERMINAL
• (myenv) tarekatbi@MacBook-Pro-de-ATBI blockchain % cd exo2
• (myenv) tarekatbi@MacBook-Pro-de-ATBI exo2 % python app.py 40000 40010

Traitement du bloc de hauteur 40000...
  Récupération de la page 1 (start_index = 0)
  Nombre de transactions dans le bloc 40000 : 1

Traitement du bloc de hauteur 40001...
  Récupération de la page 1 (start_index = 0)
  Nombre de transactions dans le bloc 40001 : 1

Traitement du bloc de hauteur 40002...
  Récupération de la page 1 (start_index = 0)
  Nombre de transactions dans le bloc 40002 : 1

Adresse Bitcoin la plus fréquente dans l'intervalle :
Adresse : 157yLVKhHvfdpWDFUHPwtNch2aEeKJgM8K
Nombre d'apparitions : 1
• (myenv) tarekatbi@MacBook-Pro-de-ATBI exo2 %
```

Figure -5- : Résultat Exercice 2

3. Exercice 3 : Vérification de la preuve de Merkle

Énoncé

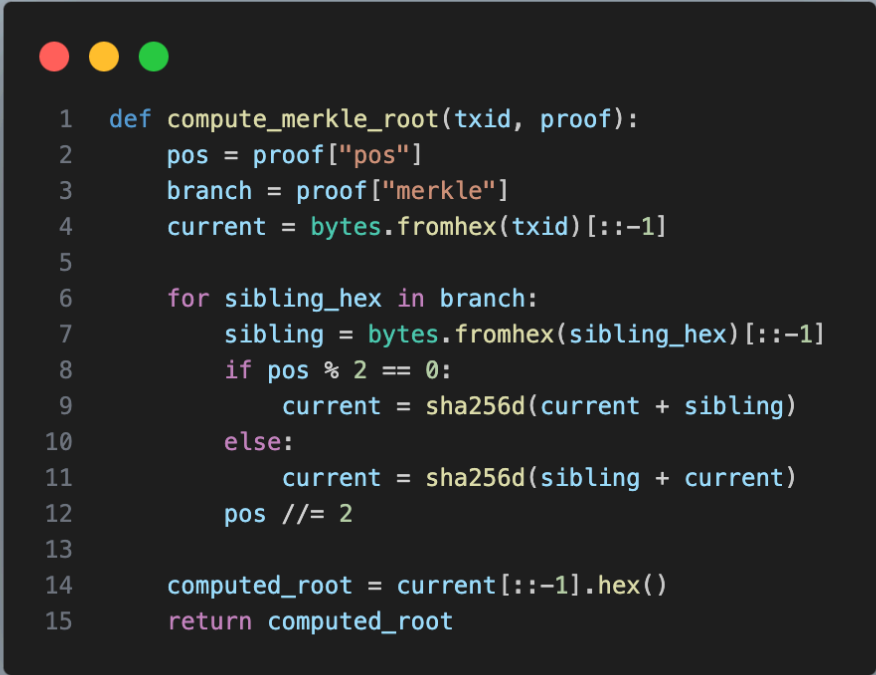
Écrire un programme qui récupère la racine de l'arbre de Merkle d'un bloc et la preuve de Merkle pour la présence d'une transaction donnée. Le programme doit ensuite vérifier que la preuve de Merkle est correcte en calculant la racine à partir du txid et de la branche fournie, puis en la comparant à la racine annoncée dans le bloc.

Méthodologie et Approche

- Endpoints utilisés :
 - `GET /block/<block_hash>` pour récupérer la racine de l'arbre de Merkle (champ "`merkle_root`").
 - `GET /tx/<txid>/merkle-proof` pour récupérer la preuve de Merkle d'une transaction.
- Étapes principales :
 - Récupérer la racine Merkle du bloc.

- Récupérer la preuve de Merkle pour le txid donné.
- Recalculer la racine en appliquant la fonction double-SHA256 sur le txid et les hash frères (branch) en respectant l'ordre déterminé par la position.
- Comparer la racine calculée à celle du bloc

Code et Explications



```
1 def compute_merkle_root(txid, proof):
2     pos = proof["pos"]
3     branch = proof["merkle"]
4     current = bytes.fromhex(txid)[::-1]
5
6     for sibling_hex in branch:
7         sibling = bytes.fromhex(sibling_hex)[::-1]
8         if pos % 2 == 0:
9             current = sha256d(current + sibling)
10        else:
11            current = sha256d(sibling + current)
12        pos //= 2
13
14    computed_root = current[::-1].hex()
15    return computed_root
```

Figure -6- : Code Exercice 3

Ce code calcule la racine de l'arbre de Merkle à partir du txid d'une transaction et de sa preuve de Merkle. Il convertit le txid en little-endian, puis combine successivement ce hash avec chaque hash frère de la preuve en appliquant une double-SHA256, en respectant l'ordre défini par la position. Enfin, le résultat est reconverti en big-endian pour obtenir la racine Merkle calculée.

Résultats et Analyse

Par exemple, pour un bloc et un txid donnés, le programme affiche :

