Створюємо Blockchain з нуля на Python

05.04.2019 Cinex Блокчейн 0

Найшвидший спосіб дізнатися, як працює блокчейн - це створити його.

ПОШУК

ПОШУК ...

ПОЗНАЧКИ

APACHE AUTORUN BASH BITCOIN BLOCKCHAIN COMPOSER CRYPTOCURRENCY CSS DATABASE DJANGO DOCKER ELOQUENT ENGLISH ENUM GAMES GIMP GIT HTML JAVA JAVASCRIPT JQUERY LARAVEL MERCURIAL PARSING PHOTOSHOP PHP PHPSTORM

Ви тут, тому що, як і я, трохи схиблені на криптовалютах. Крім цього, ви явно хочете дізнатися, як працює блокчейн - фундаментальна технологія, пов'язана з криптовалютами.

Однак, розуміння блокчейн - не найпростіша справа, принаймні, так було в моєму випадку. Я пройшов через тонни відеороликів, вивчав керівництва і постійно розчаровувався через занадто маленьку кількість прикладів.

Я люблю вчитися на практиці. Це найкраще показує суть справи на рівні коду, що запам'ятовується найкраще. Якщо у вас схожа думка, то в кінці статті ви отримаєте робочий блокчейн з твердим розумінням того, як воно працює.

Перед тим, як почати...

Пам'ятайте, що блокчейн — це незмінний, послідовний ланцюжок записів, кожна частина цього ланцюжка називається блоками. Вони можуть містити транзакції, файли або будь-який вид даних, який вам завгодно. Однак важливий момент полягає в тому, що вони пов'язані разом хешами.

На кого націлена дана стаття?

В цілому, ви вже повинні більш-менш вільно читати і писати основи Python, поряд з розумінням роботи запитів HTTP, так як ми будемо говорити про блокчейн на HTTP.

Що нам потрібно? Переконайтеся, що у вас встановлений Python 3.6+ (а також pip). Вам також потрібно буде встановити Flask і чудову бібліотеку Requests:

```
Shell
1 pip install Flask==0.12.2 requests==2.18.4
```

Вам також потрібно буде HTTP клієнт, такий як Postman або cURL. В цілому, що-небудь підійде.

PLAYONLINUX PLUGINS
PROBLEM SOLVING PYTHON REGEXP
SEO SOFTWARE ENGINEERING SPRING
SQL TELEGRAM TIPS AND TRICKS
TIPS AND TRICKS UNIT TESTING
WORDPRESS ВІРШІ ШПАРГАЛКИ

ОСТАННІ НОТАТКИ

20 Laravel Eloquent порад і трюків
15.12.2019

Як зробити скріншот сайту по URL на PHP
04.08.2019

Docker Cheat Sheet
05.07.2019

Гарячі клавіші Ubuntu Linux
01.07.2019

Git happens! 6 типових помилок Git і як їх виправити
22.06.2019

НЕДАВНІ КОМЕНТАРІ

Вихідний код статті

Ось тут ви можете ознайомитися з кодами: https://github.com/dvf/blockchain

Крок 1: Створення блокчейна

Відкрийте свій улюблений редактор тексту, або IDE, особисто я віддаю перевагу PyCharm. Створіть новий файл під назвою **blockchain.py**. Ми використовуємо тільки один файл, але якщо ви заплутаєтеся, ви завжди можете пройтися по вихідному коду.

Скелет блокчейна

Ми створимо клас блокчейна, чий конструктор створює початковий порожній лист (для зберігання нашого блокчейна), і ще один - для зберігання транзакцій. Ось креслення нашого класу:

```python
class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []

    def new_block(self):
        # Створює новий блок і вносить його в ланцюг
        pass

    def new_transaction(self):
        # Вносить нову транзакцію в список транзакцій
        pass

    @staticmethod
    def hash(block):
        # Хешує блок
        pass

    @property
```

```
    def last_block(self):
        # Повертає останній блок в ланцюжку
        pass
```

Наш клас Blockchain відповідає за управління ланцюгом. Він буде зберігати транзакції, а також мати кілька допоміжних методів для внесення нових блоків в ланцюг. Почнімо з роботи з декількома методами.

Як виглядає блок?

Кожен блок містить індекс, тимчасовий штамп (час unix), список транзакцій, доказ (про це пізніше) і хеш попереднього блока.

Ось приклад того, як виглядає один блок:

```python
block = {
    'index': 1,
    'timestamp': 1506057125.900785,
    'transactions': [
        {
            'sender': "8527147fe1f5426f9dd545de4b27ee00",
            'recipient': "a77f5cdfa2934df3954a5c7c7da5df1f",
            'amount': 5,
        }
    ],
    'proof': 324984774000,
    'previous_hash': "2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824"
}
```

З цього моменту, розуміння ланцюга повинно бути роздільним - кожен новий блок містить у собі хеш попереднього блоку. Це принципово важливо, тому що цим забезпечується незмінність блокчейна: якщо зловмисник зламає попередній блок, то всі інші блоки будуть містити неправильні хеші.

Чи є в цьому сенс? Якщо ні - то вам потрібно приділити час, щоб зрозуміти і усвідомити це, оскільки ми говоримо про фундаментальні принципи роботи блокчейна.

Внесення транзакцій в блок

Нам потрібен буде спосіб внесення транзакцій в блок. Наш метод **new_transaction()** відповідає за це, і він досить прямолінійний:

```python
class Blockchain(object):
    ...

    def new_transaction(self, sender, recipient, amount):
        """
        Направляє нову транзакцію в наступний блок

        :param sender: <str> Адреса відправника
        :param recipient: <str> Адреса отримувача
        :param amount: <int> Сума
        :return: <int> Індекс блоку, який буде зберігати цю транзакцію
        """

        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
        })

        return self.last_block['index'] + 1
```

Після того, як **new_transaction()** внесе транзакцію в список, він поверне індекс блоку, в якому повинно бути внесена транзакція - а саме наступна. У майбутньому, це буде корисно для користувача, що відправляє транзакцію.

Створення нових блоків

Після того, як ми отримали примірник блокчейна, нам потрібно посадити в нього блок генезису - перший блок без попередників. Нам також потрібно внести "пруф" в наш блок генезису, який являє собою результат Майнінг (докази проведеної роботи). Ми розглянемо майнінг пізніше.

В доповнення до створення блоку генезису в конструкторі, ми також викотимо методи для **new_block**(), **new_transaction**() и **hash**():

```python
import hashlib
import json
from time import time


class Blockchain(object):
    def __init__(self):
        self.current_transactions = []
        self.chain = []

        # Створення блоку генезису
        self.new_block(previous_hash=1, proof=100)

    def new_block(self, proof, previous_hash=None):
        """
        Створення нового блоку в блокчейні

        :param proof: <int> Докази проведеної роботи
        :param previous_hash: (Опціонально) хеш попереднього блоку
        :return: <dict> Новий блок
        """

        block = {
            'index': len(self.chain) + 1,
            'timestamp': time(),
            'transactions': self.current_transactions,
            'proof': proof,
            'previous_hash': previous_hash or self.hash(self.chain[-1]),
        }

        # Перезавантаження поточного списку транзакцій
        self.current_transactions = []

```

```
        self.chain.append(block)
        return block

    def new_transaction(self, sender, recipient, amount):
        """
        Направляє нову транзакцію в наступний блок

        :param sender: <str> Адреса відправника
        :param recipient: <str> Адреса одержувача
        :param amount: <int> Сума
        :return: <int> Індекс блоку, який буде зберігати цю транзакцію
        """
        self.current_transactions.append({
            'sender': sender,
            'recipient': recipient,
            'amount': amount,
        })

        return self.last_block['index'] + 1

    @property
    def last_block(self):
        return self.chain[-1]

    @staticmethod
    def hash(block):
        """
        Створює хеш SHA-256 блоку

        :param block: <dict> Блок
        :return: <str>
        """

        # Ми повинні переконатися в тому, що словник впорядкований, інакше у нас будуть
        block_string = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(block_string).hexdigest()
```

Код вище повинен бути досить ясним - я вніс кілька коментарів і документацію, щоб все було зрозуміло. Структура даних буде в json. Ми майже закінчили з скелетом нашого блокчейна. Однак на даний момент, вам напевно цікаво, як створюються нові блоки?

Розуміння підтвердження роботи

Алгоритм пруфа роботи (**Proof of Work, PoW**) — це те, як нові блоки створені або майнятся в блокчейні. Мета PoW — це знайти число, яке вирішує проблему. Число повинно бути таким, щоб його важко було знайти, але легко підтвердити (говорячи про обчислення) ким завгодно в Інтернеті. Це головне завдання алгоритму.

Розглянемо простий приклад, щоб отримати краще уявлення.

Скажімо, що хеш того чи іншого числа, помноженого на інше число повинен закінчуватися нулем. Таким чином, **hash(x * y) = ac23dc...0**. Для цього спрощеного прикладу, уявімо що x = 5. Як це працює в Python:

```python
from hashlib import sha256

x = 5
y = 0  # Ми ще не знаємо, чому дорівнює y...
while sha256(f'{x*y}'.encode()).hexdigest()[-1] != "0":
    y += 1

print(f'The solution is y = {y}')
```

Рішення тут наступне: **y = 21**, так як створений хеш закінчується нулем:

```python
hash(5 * 21) = 1253e9373e...5e3600155e860
```

У біткоїні, такий алгоритм називається **Hashcash**. І він особливо не відрізняється від наведеного вище прикладу. Це алгоритм, який намагається вирішити покоління Майнерів (читай, окрема раса), щоб створити новий блок. В цілому, складність визначається кількістю символом, які розглядаються в рядку. Майнер непогано винагороджуються за вирішення завдання отриманням coin в транзакції.

Реалізація базового PoW

Давайте реалізуємо аналогічний алгоритм для нашого блокчейна. Наше правило буде аналогічно зазначеному раніше:

> *Знайдіть число «p«, яке хеширується з попереднім створеним рішенням блоку з хешем містить 4 заголовних нуля.*

```python
import hashlib
import json

from time import time
from uuid import uuid4


class Blockchain(object):
    ...

    def proof_of_work(self, last_proof):
        """
        Проста перевірка алгоритму:
         - Пошуку числа p`, так як hash (pp`) містить 4 заголовних нуля, де p - попередн
         - p є попереднім доказом, а p` - новим

        :param last_proof: <int>
        :return: <int>
        """

        proof = 0
        while self.valid_proof(last_proof, proof) is False:
            proof += 1

        return proof

    @staticmethod
    def valid_proof(last_proof, proof):
        """
        Підтвердження доказу: Чи містить hash (last_proof, proof) 4 великих нуля?

        :param last_proof: <int> Попереднє підтвердження
        :param proof: <int> Поточний доказ
        :return: <bool> True, якщо правильно, False, якщо ні.
```

```
        """

        guess = f'{last_proof}{proof}'.encode()
        guess_hash = hashlib.sha256(guess).hexdigest()
        return guess_hash[:4] == "0000"
```

Щоб скорегувати складність алгоритму, ми можемо змінити кількість заголовних нулів. У нашому випадку, 4-достатньо. Ви дізнаєтеся, що внесення одного провідного нуля створює колосальну різницю в часі, необхідному для пошуку рішення (Майнінг).

Наш клас практично готовий, так що ми можемо почати взаємодіяти з ним через HTTP запити.

Крок 2: Блокчейн як API

Тут ми задіємо фреймворк під назвою Flask. Це макро-фреймворк, який помітно спрощує зіставлення кінцевих точок з функціями Python. Це дозволяє нам взаємодіяти з нашим блокчейном в інтернеті за допомогою HTTP-запитів.

Ми створимо три метода:

- **/transactions/new** для створення нової транзакції в блоці;
- /**mine**, щоб вказати серверу, що потрібно майнити новий блок;
- /**chain** для повернення всього блокчейну

Настройка Flask

Наш "сервер" сформує єдиний вузол в нашій мережі блокчейну. Давайте створимо шаблонний код:

```python
import hashlib
import json
from textwrap import dedent
```

```python
from time import time
from uuid import uuid4

from flask import Flask


class Blockchain(object):
    ...


# Створюємо примірник вузла
app = Flask(__name__)

# Генеруємо унікальну на глобальному рівні адресу для цього вузла
node_identifier = str(uuid4()).replace('-', '')

# Створюємо примірник блокчейну
blockchain = Blockchain()


@app.route('/mine', methods=['GET'])
def mine():
    return "We'll mine a new Block"

@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    return "We'll add a new transaction"

@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Коротке пояснення того, що ми тільки що додали:

- **Рядок 15**: Створення екземпляра вузла. Можете більше дізнатися про Flask тут;
- **Рядок 18**: Створення випадкового імені нашого вузла;

- **Рядок 21**: Створення примірника **класу Blockchain**;
- **Рядок 24-26**: Створення кінцевої точки **/mine**, яка є GET-запитом;
- **Рядок 28-30**: Створення кінцевої точки **/transactions/new**, яка является POST-запитом, так як ми будемо відправляти туди дані;
- **Рядок 32-38**: Створення кінцевої точки **/chain**, яка повертає весь блокчейн;
- **Рядок 40-41**: Запускає сервер на порті: 5000.

Кінцева точка транзакцій

Ось так запит транзакції повинен виглядати. Це те, що користувач відправляє на сервер:

```python
{
    "sender": "my address",
    "recipient": "someone else's address",
    "amount": 5
}
```

Так як ми вже володіємо методом класу для **додавання транзакцій в блок**, справа залишилася за малим. Давайте напишемо функцію для внесення транзакцій:

```python
import hashlib
import json
from textwrap import dedent
from time import time
from uuid import uuid4

from flask import Flask, jsonify, request

...

@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    values = request.get_json()

```

```
    # Переконайтеся в тому, що необхідні поля знаходяться серед в POST-даних
    required = ['sender', 'recipient', 'amount']
    if not all(k in values for k in required):
        return 'Missing values', 400

    # Створення нової транзакції
    index = blockchain.new_transaction(values['sender'], values['recipient'], values['am

    response = {'message': f'Transaction will be added to Block {index}'}
    return jsonify(response), 201
```

Кінцева точка майнінгу

Кінцева точка майнінгу - це частина, де відбувається магія, і це просто! Для цього потрібно зробити три речі:

- Підрахувати PoW;
- Нагородити Майнера (нас), додавши транзакцію, що дає нам 1 коін;
- Зліпити наступний блок, внісши його у ланцюг.

```python
import hashlib
import json

from time import time
from uuid import uuid4

from flask import Flask, jsonify, request

...

@app.route('/mine', methods=['GET'])
def mine():
    # Ми запускаємо алгоритм підтвердження роботи, щоб отримати наступне підтвердження
    last_block = blockchain.last_block
    last_proof = last_block['proof']
    proof = blockchain.proof_of_work(last_proof)

    # Ми повинні отримати винагороду за знайдене підтвердження
```

```python
    # Відправник "0" означає, що вузол заробив крипто-монету
    blockchain.new_transaction(
        sender="0",
        recipient=node_identifier,
        amount=1,
    )

    # Створюємо новий блок, шляхом внесення його в ланцюг
    previous_hash = blockchain.hash(last_block)
    block = blockchain.new_block(proof, previous_hash)

    response = {
        'message': "New Block Forged",
        'index': block['index'],
        'transactions': block['transactions'],
        'proof': block['proof'],
        'previous_hash': block['previous_hash'],
    }
    return jsonify(response), 200
```

Зверніть увагу на те, що одержувач замайненого блоку - це адреса нашого вузла. Велика частина того, що ми тут зробили, це просто взаємодія з методами в нашому класі Blockchain. З цього моменту, ми закінчили, і можемо почати взаємодіяти з нашим blockchain на Python.

Крок 3: Взаємодія з нашим блокчейном

Ви можете використовувати старий добрий **cURL** або **Postman** для взаємодії з нашим API в мережі.

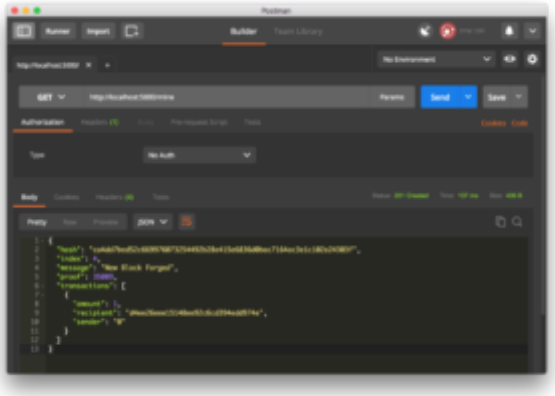
Запускаємо сервер:

```shell
$ python blockchain.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

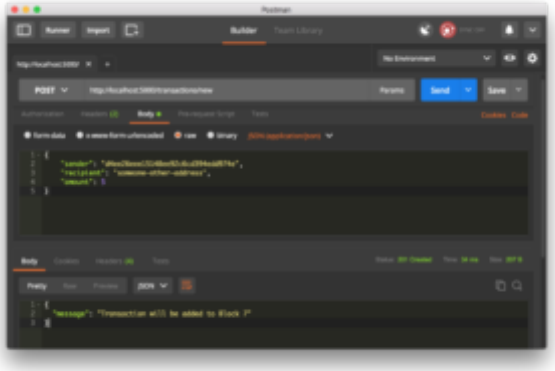
Давайте спробуємо майнити блок, створивши GET-запит до вузла **http://localhost:5000/mine**:

```python
```

```
curl http://localhost:5000/mine
```



Тепер, давайте **створимо нову транзакцію**, відправивши POST-запит до вузла **http://localhost: 5000/transactions/new** з тілом, що містить структуру нашої транзакції:



Якщо ви не користуєтеся **Postman**, тоді ви можете створити аналогічний запит за допомогою cURL:

```
Python
$ curl -X POST -H "Content-Type: application/json" -d '{
 "sender": "d4ee26eee15148ee92c6cd394edd974e",
```

```
 "recipient": "someone-other-address",
 "amount": 5
}' "http://localhost:5000/transactions/new"
```

Я перезапустив свій сервер і замайнив два блоки, разом їх кількість 3. Давайте перевіримо весь ланцюжок, виконавши запит до вузла **http://localhost:5000/chain**:

```python
{
  "chain": [
    {
      "index": 1,
      "previous_hash": 1,
      "proof": 100,
      "timestamp": 1506280650.770839,
      "transactions": []
    },
    {
      "index": 2,
      "previous_hash": "c099bc...bfb7",
      "proof": 35293,
      "timestamp": 1506280664.717925,
      "transactions": [
        {
          "amount": 1,
          "recipient": "8bbcb347e0634905b0cac7955bae152b",
          "sender": "0"
        }
      ]
    },
    {
      "index": 3,
      "previous_hash": "eff91a...10f2",
      "proof": 35089,
      "timestamp": 1506280666.1086972,
      "transactions": [
        {
          "amount": 1,
          "recipient": "8bbcb347e0634905b0cac7955bae152b",
          "sender": "0"
        }
      ]
    }
```

```
  ],
  "length": 3
}
```

Крок 4: Консенсус

Поки все йде дуже добре. У нас є базовий blockchain, який приймає транзакції і дає можливість майнити нові блоки. Але вся суть блокчейна в тому, що вони повинні бути децентралізованими. А якщо вони децентралізовані, яким чином ми можемо гарантувати, всі вони відображають один ланцюжок?

Це називається проблемою Консенсусу, так що нам потрібно реалізувати **алгоритм Консенсусу**, якщо нам потрібно більше одного вузла в нашому ланцюзі.

Реєстрація нових вузлів

Щоб ми змогли реалізувати **алгоритм Консенсусу**, нам потрібно знайти спосіб дати вузлу знати про існування сусідніх вузлів в ланцюзі. Кожен вузол в нашій ланцюга повинен містити регістр інших вузлів в ланцюзі. Отже, нам знадобитися більше кінцевих точок:

- **/nodes/register** для прийняття списоку нових вузлів в формі URL-ів;
- **/nodes/resolve** для реалізації нашого алгоритму Консенсусу, який вирішує будь-які конфлікти, пов'язані з підтвердженням того, що вузол перебуває в своєму ланцюзі.

Нам потрібно буде змінити конструктор нашого **Blockchain** і привнести метод для реєстрації вузлів:

```python
...
from urllib.parse import urlparse
...


```

```python
class Blockchain(object):
    def __init__(self):
        ...
        self.nodes = set()
        ...

    def register_node(self, address):
        """
        Вносимо новий вузол в список вузлів

        :param address: <str> адреса вузла , другими словами: 'http://192.168.0.5:5000'
        :return: None
        """

        parsed_url = urlparse(address)
        self.nodes.add(parsed_url.netloc)
```

Зверніть увагу на те, що ми використовували **set()** для **зберігання списку вузлів**. Це легкий спосіб переконатися в тому, що внесення нових вузлів є ідемпотентна - це означає, що незалежно від того, скільки разів ми внесемо певний вузол, він виникне тільки один раз.

Реалізація алгоритму Консенсусу

Як ми вже знаємо, конфлікт полягає в тому, що **один вузол має інший ланцюг**, пов'язаний з іншим вузлом. Щоб вирішити це, ми введемо правило, де найдовша і валідна ціна є авторитетною. Іншими словами, довгий ланцюг мережі де-факто є єдиним. Використовуючи цей алгоритм, ми досягнемо Консенсусу серед вузлів в нашій мережі.

```python
...
import requests


class Blockchain(object)
    ...

    def valid_chain(self, chain):
        """
        Перевіряємо, чи є внесений в блок хеш коректним
```

```
11
12          :param chain: <list> blockchain
13          :return: <bool> True якщо вона є дійсною, False, якщо ні
14          """
15
16          last_block = chain[0]
17          current_index = 1
18
19          while current_index < len(chain):
20              block = chain[current_index]
21              print(f'{last_block}')
22              print(f'{block}')
23              print("\n-----------\n")
24              # Перевірте правильність хешу блоку
25              if block['previous_hash'] != self.hash(last_block):
26                  return False
27
28              # Перевіряємо, чи є підтвердження роботи коректним
29              if not self.valid_proof(last_block['proof'], block['proof']):
30                  return False
31
32              last_block = block
33              current_index += 1
34
35          return True
36
37      def resolve_conflicts(self):
38          """
39          Це наш алгоритм Консенсусу, він вирішує конфлікти,
40          замінюючи наш ланцюг на найдовший в ланцюг
41
42          :return: <bool> True, якщо б наша ланцюг була замінена, False, якщо ні
43  .
44          """
45
46          neighbours = self.nodes
47          new_chain = None
48
49          # Шукаємо тільки ланцюги, довше нашого
50          max_length = len(self.chain)
51
52          # Захоплюємо і перевіряємо всі ланцюги з усіх вузлів мережі
53          for node in neighbours:
54              response = requests.get(f'http://{node}/chain')
55
```

```
            if response.status_code == 200:
                length = response.json()['length']
                chain = response.json()['chain']

                # Перевіряємо, чи є довжина найдовшою, а ланцюг - валідним
                if length > max_length and self.valid_chain(chain):
                    max_length = length
                    new_chain = chain

        # Замінюємо наш ланцюг, якщо знайдемо інший валідний і довший
        if new_chain:
            self.chain = new_chain
            return True

        return False
```

Перший метод **valid_chain()** відповідає за перевірку того, чи є ланцюг валідним, запустивши цикл через кожен блок і проводячи верифікацію як хешу, так і proof.

Метод **resolve_conflicts()**, який запускає цикл через всі наші сусідні вузли, завантажує їх ланцюги і проводить перевірку, як і в попередньому методі. **Якщо валідний ланцюг, довжина якого більше, ніж наш, ми замінюємо наш.**

Давайте зареєструємо дві кінцеві точки нашого API, одну для внесення сусідніх вузлів, а другу - для вирішення конфліктів:

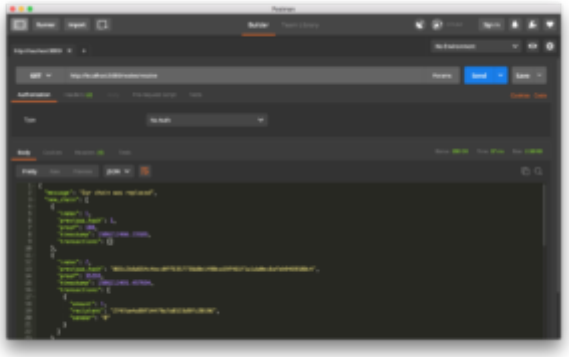
Python

```
@app.route('/nodes/register', methods=['POST'])
def register_nodes():
    values = request.get_json()

    nodes = values.get('nodes')
    if nodes is None:
        return "Error: Please supply a valid list of nodes", 400

    for node in nodes:
        blockchain.register_node(node)

    response = {
        'message': 'New nodes have been added',
```

```
        'total_nodes': list(blockchain.nodes),
    }
    return jsonify(response), 201


@app.route('/nodes/resolve', methods=['GET'])
def consensus():
    replaced = blockchain.resolve_conflicts()

    if replaced:
        response = {
            'message': 'Our chain was replaced',
            'new_chain': blockchain.chain
        }
    else:
        response = {
            'message': 'Our chain is authoritative',
            'chain': blockchain.chain
        }

    return jsonify(response), 200
```

Тепер ви можете сісти за інший комп'ютер (якщо хочете), і розгорнути різні вузли в своїй мережі. Також ви можете розгорнути процеси за допомогою різних портів на одному і тому ж комп'ютері. Я розгорнув ще один вузол на своєму комп'ютері, але на іншому порті і зареєстрував його за допомогою мого поточного вузла. Таким чином, я отримав два вузла: **http://localhost:5000** і **http://localhost:5001**.

Після цього я отримав два нові блоки в вузлі 2, щоб переконатися в тому, що ланцюг був довшим. Після цього, я викликав GET **/nodes/resolve** в вузлі 1, де ланцюг був замінений нашим алгоритмом **консенсусу**:



І це була обгортка ... Знайдіть друзів і разом спробуйте протестувати ваш блокчейн!

Підведемо підсумки

Сподіваюся, ця стаття надихнула вас на що-небудь нове. Я в захваті від криптовалют, так як я вірю, що блокчейн радикально змінить наше уявлення про економіку, уряд і облікові записи!

BITCOIN BLOCKCHAIN CRYPTOCURRENCY PYTHON SOFTWARE ENGINEERING

« **ПОПЕРЕДНІЙ**
25 цитат

ДАЛІ »
Spring Data JPA. JUnit тести для Services. Частина 3

ЗАЛИШТЕ ПЕРШИЙ КОМЕНТАР

Залишити коментар

Вашу адресу електронної пошти не буде опубліковано

Коментувати

Ім'я*

Email *

Збережіть моє ім'я, електронну пошту у цьому веб-переглядачі наступного разу, коли я коментуватиму.

ОПУБЛІКУВАТИ КОМЕНТАР