НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Звіти до комп’ютерних практикумів з кредитного модуля “Технологія Блокчейн”

Прийняв

доцент кафедри

Яланецький В. А.

Виконав

Студент групи ІТ-02

Макаров І.С.

Київ – 2022

Комп‘ютерний практикум No 2.

Мета:

Ознайомитися з фреймворком FastAPI, реалізувати функціонал взаємодії з прототипом блокчейну.

ВИКОНАННЯ

https://github.com/kinfi4/Kin-Blockchain

Почну з того, що наведу скріншот структури проекту, що вийшов в мене в кінці кінців:

Тут одразу такой допишу дисклеймер: у вас в завданні написано, що всі методи та класи повинні починатись з ПІБ студента, що максимально дивно, в мене стоїть на PyCharm плагін, що показує ім'я того, хто написав кожен клас та метод (плагін бере цю інфу з git), так я думаю буде і комфортно читати, і у вас буде доказ, що код написаний мною.

Це по перше, по друге в якості бекенд фреймворку я використовував не Flask, а FastAPI, бо люблю його більше, структура в них +/- однакова, тому не думаю, що це є проблема.

Вся бізнес логіка в мене лежить в domain/ я думаю вам краще було б її переглянути на Github, але з минулої лаби вона майже не змінилась, тому не думаю, що є сенс її тут вставляти. Давайте подивимось на endpoints.

Спочатку ендпоінт отримання всього блокчейну

```python
@router.get('/full-blockchain', response_model=list[BlockModel])
@inject
def get_full_blockchain(
    blockchain: Blockchain = Depends(Provide[Container.blockchain]),
):
    blocks = blockchain.get_blockchain()

    return [BlockModel.from_domain(block) for block in blocks]
```

Відповідно наступний ендпоінт призначенний для створення транзакції

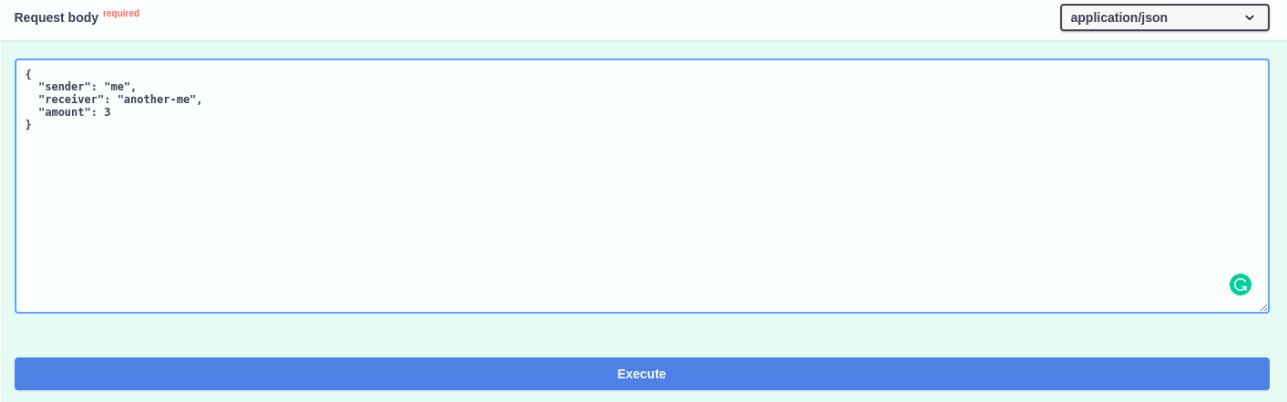
```python
Illia Makarov
@router.post('', response_model=TransactionModel, status_code=status.HTTP_201_CREATED)
@inject
def create_transaction(
    transaction: TransactionModel,
    transaction_service: TransactionService = Depends(Provide[Container.services.transaction_service])
):
    added_transaction = transaction_service.add_transaction(transaction.to_domain())

    return TransactionModel.from_domain(added_transaction)
```

І нарешті останній ендпоінт для майнінгу:

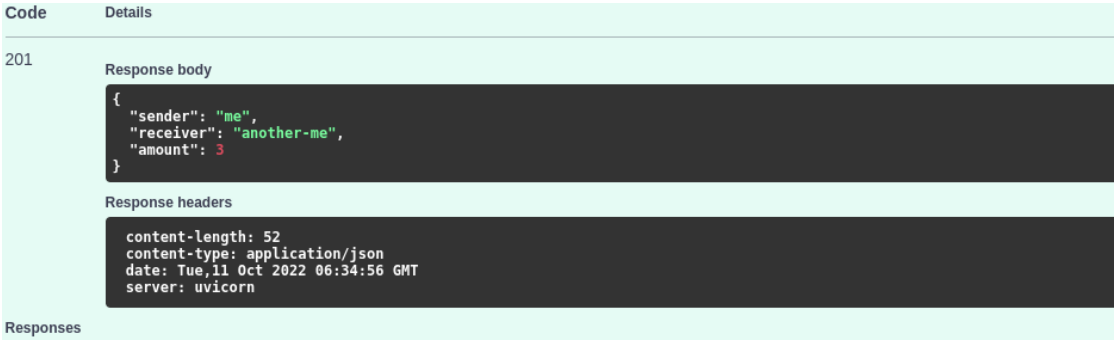
```python
Illia Makarov
@router.get('', response_model=BlockModel)
@inject
def mine_block(
    miner_address: str,
    mining_service: MiningService = Depends(Provide[Container.mining_service]),
    blockchain: Blockchain = Depends(Provide[Container.blockchain]),
    transaction_service: TransactionService = Depends(Provide[Container.services.transaction_service])
):
    transaction_reward = TransactionEntity(
        sender='0',
        receiver=miner_address,
        amount=1
    )
    transaction_service.add_transaction(transaction_reward)

    block = mining_service.mine_new_block()
    new_block = blockchain.create_block(block)

    return BlockModel.from_domain(new_block)
```

Давайте тепер подивимось на те, як це працює. FastAPI із коробки має підтримку swagger, що є дуже корисним в тестуванні апішок.

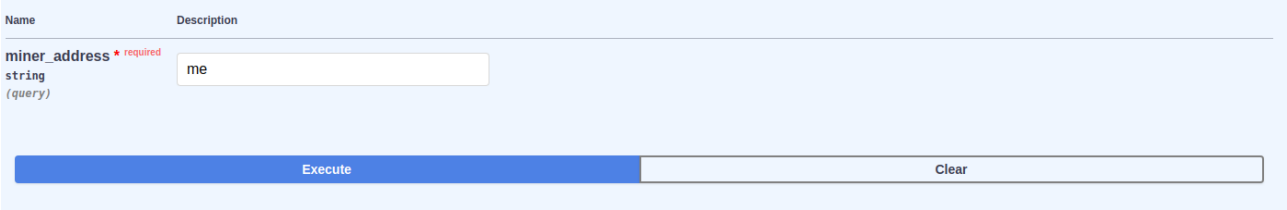
Давайте спробуємо створити транзакцію для початку. Перевірки для того, чи можлива така транзакція я поки не писав, бо це не частина цієї роботи, на скільки я розумію



відправимо запит, та отримаємо та отримаємо результат

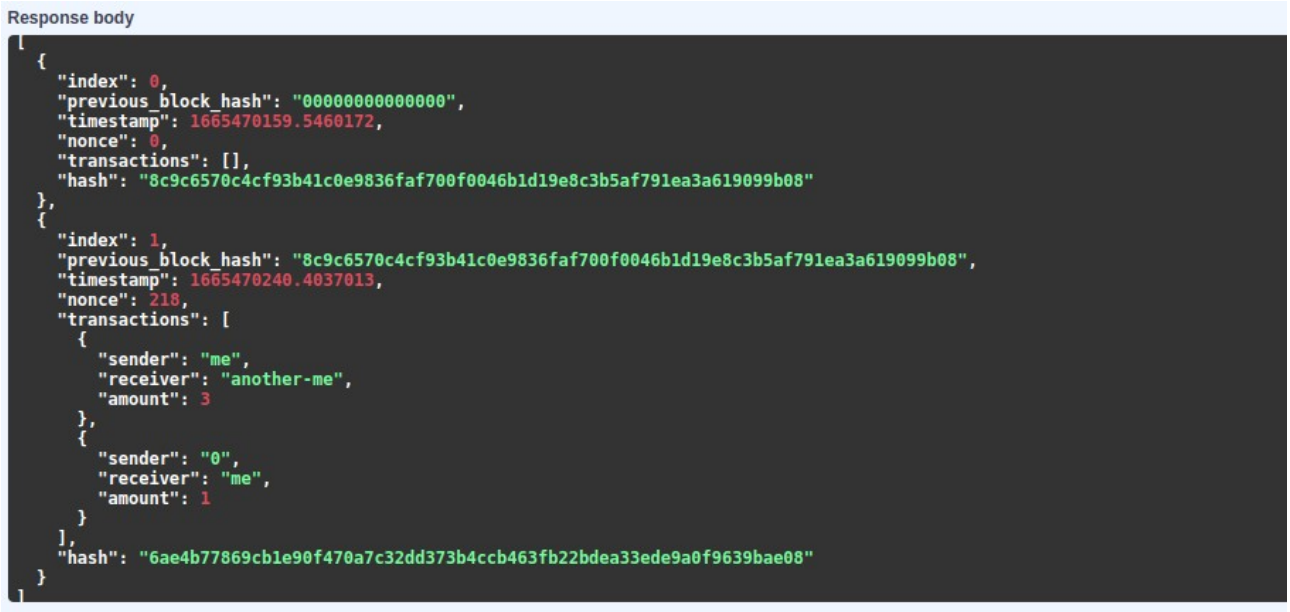


Тепер давайте змайнимо блок, ендпоінт майнінгу приймає на всід нашу адресу, куди направити нагороду.



Ось блок, що ми змайнили, як видно тут є Coinbase транзакція та ще одна, що ми створили до того

Давайте тепер подивимось, як виглядає блокчейн



тут як видно два блоки, перший Genesis Block, і другий, що ми щойно змайнили.

Нагадаю, що моє день народження це 03.08.2002, тому hash закінчуються на 08.

Дякую за увагу.

ВИСНОВОК

В даній роботі, ми додали до стелету нашого блокчейну невеличку апішку, аби була можливість взаємодіяти з нашою системою.