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 1.

Скелетон Блокчейну

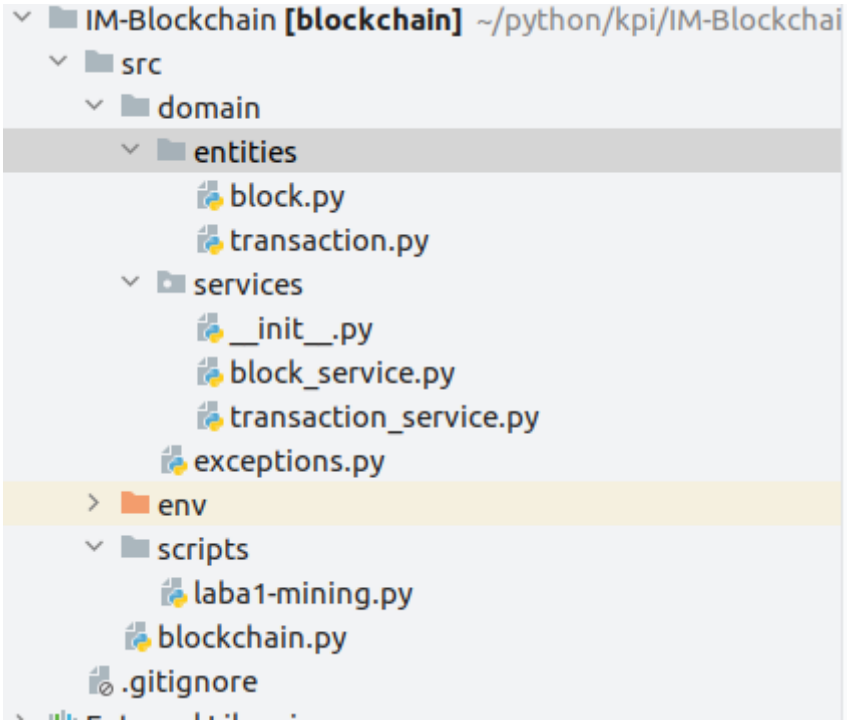
Мета:

Отримати уявлення про високорівневе функціонування примитивного блокчейну.

ВИКОНАННЯ

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

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



Основний файл тут це blockchain.py, що містить клас Blockchain, що в свою чергу символізує, хто би міг подумати, мій блокчейн.

Сам по собі клас це скоріше такий собі фасад навколо сервісів `BlockService`

та `TransactionService`, як не складно догадатись, перший в мене відповідає за створення та збереження блоків в нашій ципочці, а другий за створення та збереження транзакцій.

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.

```
class Blockchain:
    Illia Makarov
    def __init__(self, block_service: BlockService, transaction_service: TransactionService):
        self._tr_service = transaction_service
        self._bl_service = block_service

    Illia Makarov
    def add_transaction(self, transaction: Transaction) -> BlockIndex:
        if not transaction.is_valid():
            raise TransactionInvalid('Passed transaction is not valid')

        self._tr_service.add_transaction(transaction)

        return BlockIndex(self._bl_service.last_block.index + 1)

    Illia Makarov
    def create_block(self, proof: int) -> Block:
        if not self.validate_proof(proof):
            raise ProofValidationFailed(f'Could not create a block with {proof=}, proof did not pass validation')

        transactions = self._tr_service.flush_transactions()

        return self._bl_service.add_block(proof, transactions)

    Illia Makarov *
    def validate_proof(self, proof: int) -> bool:
        hash_result = get_block_hash(self._bl_service.last_block, proof)

        return hash_result[-2:] == "08"

    Illia Makarov
    def get_blockchain(self) -> list[Block]:
        return self._bl_service.get_blockchain()
```

Сам по собі клас не дуже цікавий, він слугує як proxy для сервісів, описаних вище, та має додатковий метод для валідації nonce.

```
Illia Makarov
class TransactionService:
    Illia Makarov
    def __init__(self, transaction_list: list[Transaction] = None) -> None:
        self._transactions = transaction_list if transaction_list else []

    Illia Makarov
    def add_transaction(self, transaction: Transaction) -> None:
        self._transactions.append(transaction)

    Illia Makarov
    def flush_transactions(self) -> list[Transaction]:
        if not self._transactions:
            raise RuntimeError(f'Sorry but there are no transaction in the list!')

        transactions_to_return = self._transactions
        self._transactions = []

        return transactions_to_return
```

```python
Illia Makarov
class BlockService:
    Illia Makarov
    def __init__(self, blocks_list: list[Block] = None) -> None:
        self._block_list = blocks_list if blocks_list else []

    Illia Makarov
    def add_block(self, proof: int, transactions: list[Transaction]) -> Block:
        new_block_index = BlockIndex(len(self._block_list) + 1)
        block = Block(
            index=new_block_index,
            timestamp=time(),
            transactions=transactions,
            proof=proof,
            previous_block_hash=self.last_block.get_hash()
        )

        self._block_list.append(block)

        return block

    Illia Makarov
    @property
    def last_block(self) -> Block:
        return self._block_list[-1]

    Illia Makarov
    def get_block(self, block_idx: BlockIndex) -> Block:
        return self._block_list[block_idx]

    Illia Makarov
    def get_blockchain(self) -> list[Block]:
        return self._block_list
```

Класи сервісів знаходяться в каталозі domain/services, логіка тут доволі примітивна, не бачу сенсу на ній зупинятись.

Тепер давайте подивимось, щож в мене таке блок та транзація.

Для створення моделей, що відображають дані поняття, я написав два невеликих dataclasses, в собі вони мають необхідні поля, а також методи для отримання хешу.

```python
BlockIndex = NewType('BlockIndex', int)


Illia Makarov
@dataclass(frozen=True)
class Block:
    index: BlockIndex
    previous_block_hash: str
    timestamp: float
    proof: int
    transactions: list[Transaction]

    Illia Makarov
    def get_hash(self) -> str:
        transaction_str = json.dumps(self.to_dict())
        return hashlib.sha224(transaction_str.encode()).hexdigest()

    Illia Makarov
    def to_dict(self) -> dict:
        return {
            "index": self.index,
            "previous_block_hash": self.previous_block_hash,
            "timestamp": self.timestamp,
            "proof": self.proof,
            "transactions": [transaction.to_dict() for transaction in self.transactions]
        }
```

```python
Illia Makarov
@dataclass(frozen=True)
class Transaction:
    sender: str
    receiver: str
    amount: float

    Illia Makarov
    def get_hash(self) -> str:
        transaction_str = json.dumps(self.to_dict())
        return hashlib.sha224(transaction_str.encode()).hexdigest()

    Illia Makarov
    def to_dict(self) -> dict:
        return {
            "sender": self.sender,
            "receiver": self.receiver,
            "amount": self.amount,
        }

    Illia Makarov
    def is_valid(self):
        return all([
            self.sender != '',
            self.receiver != '',
            self.amount > 0,
        ])
```

Ну щож залишилось написати скріпт, для додавання транзакцій в блокчейн, такі штуки я буду закидувати в папку scripts. Вставляти сюди купу скрінів не хочу, бо воно доволі велике, код краще подивитись ось тут: *https://github.com/kinfi4/Kin-Blockchain/blob/master/src/scripts/laba1-mining.py*

А сюди я вставлю скріншот роботи скріпту:

```
Block(index=0, previous_block_hash='makarov', timestamp=1664863592.185687, transactions=[], proof=None, block_hash='18fa28c15ea394e969604e833786f756ba9db4684db305fdaf1e2508')
Block(index=2, previous_block_hash='18fa28c15ea394e969604e833786f756ba9db4684db305fdaf1e2508', timestamp=1664863592.1908798, transactions=[], proof=3082035,
  block_hash='872f5b874eb40a29eaac008164ea279b2605fa9a129c13708e49d208')
Block(index=3, previous_block_hash='872f5b874eb40a29eaac008164ea279b2605fa9a129c13708e49d208', timestamp=1664863592.2015011, transactions=[], proof=3082080,
  block_hash='69cda7822a95f8f954cf769ae8a80222066cf5e57b19876113db0208')
Block(index=4, previous_block_hash='69cda7822a95f8f954cf769ae8a80222066cf5e57b19876113db0208', timestamp=1664863592.2689593, transactions=[], proof=3082545, block_hash=None)
...
```

Я створив 3 блоки (разом з genesis блоком вийшло 4) і вивів як виглядає наш блокчейн.

Висновок:

Ну щож, работа була цікавою, я написав дуже примитивний inmemory блокчейн, що може додавати транзакції собі у блоки, і блоки збирати у цепочку. Написав дуже примітивний скрипт майнінгу на основі PoW.