НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

ЗВІТ

з виконання лабораторної роботи №1
з дисципліни: **“Сучасні методи програмування”**
на тему: **“Розробка архітектури блокчейна”**

Виконав:
студент групи КА-22мп
ННК “ІПСА”
Муравльов А.Д.
Викладач: Соболь О.О.

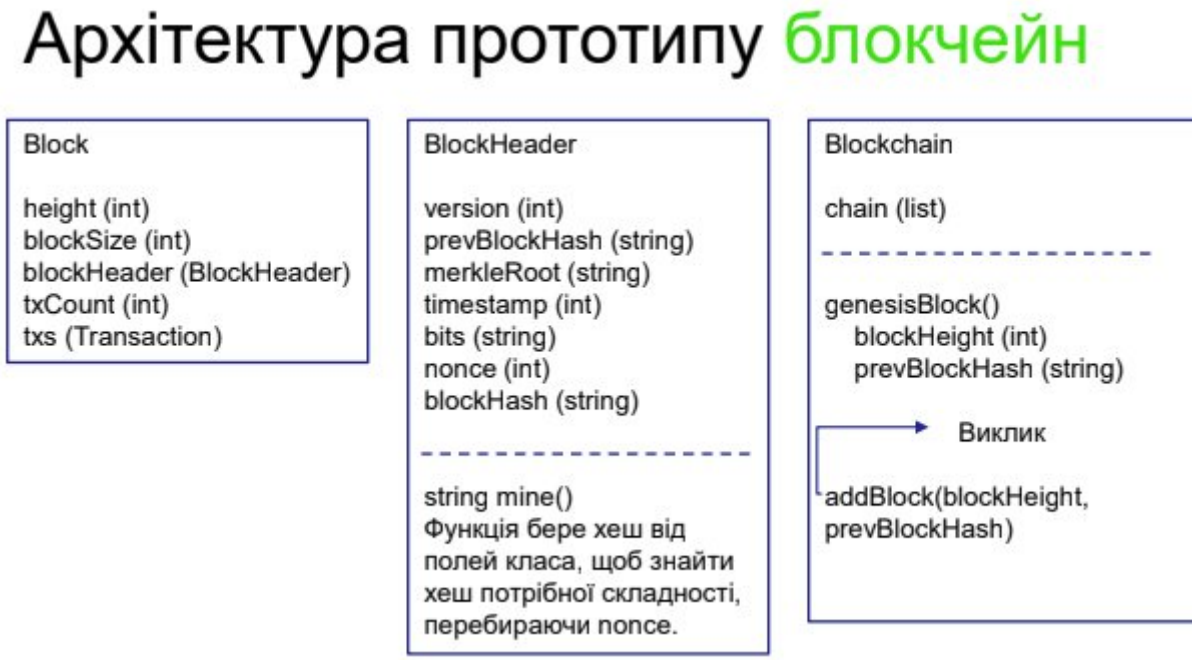
Київ — 2022

ЗМІСТ

Завдання роботи

Мета роботи: За запропонованою та описаною в матеріалах практики №2 архітектурою та бізнес-логікою створити програмний продукт, який генерує блокчейн.

Запропонована архітектура з практики №2:



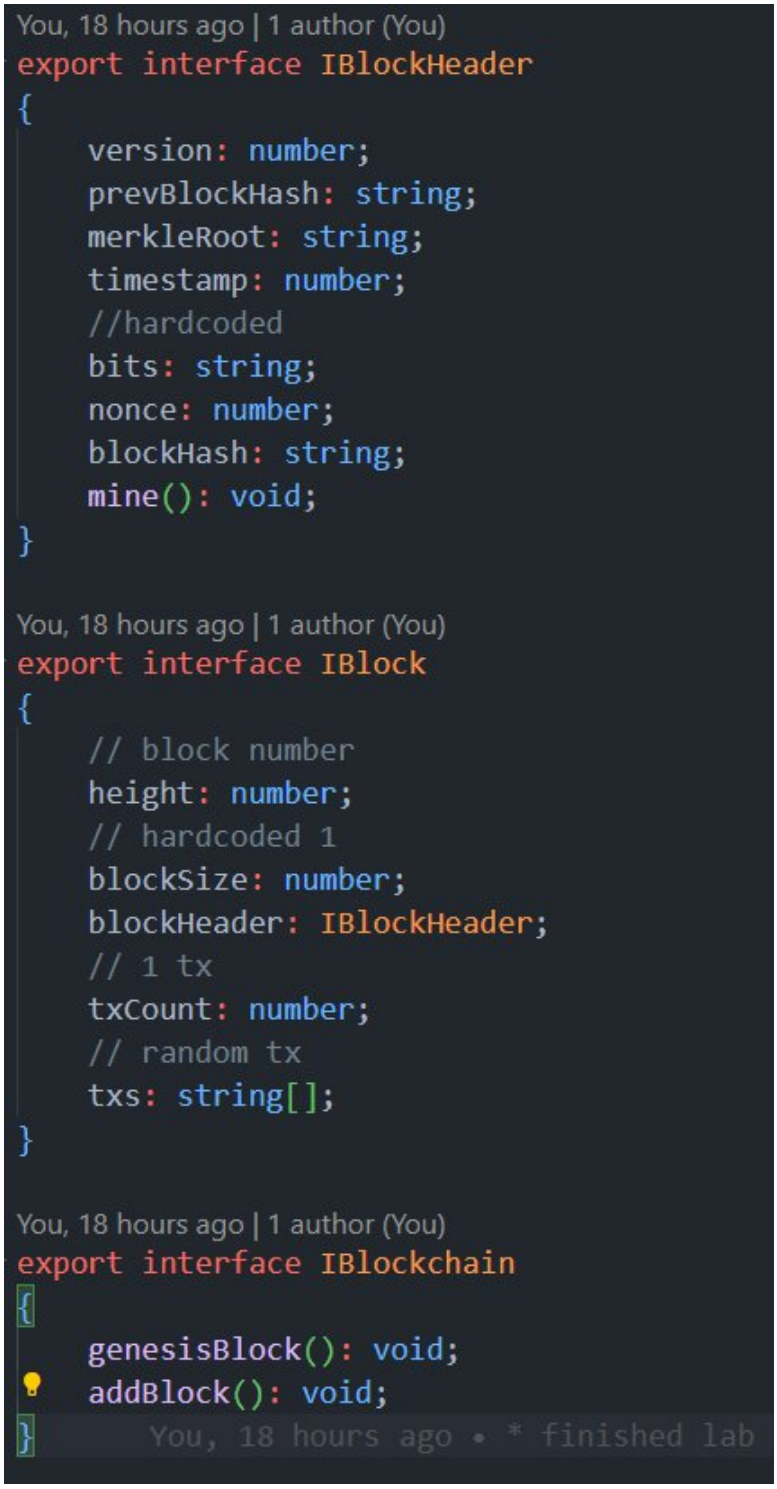
Завдання згідно практики:

1. За запропонованою архітектурою створити програмний продукт, який генерує блокчейн
2. Бізнес-логіка програмного продукту описується виконанням наступних кроків:
    0. Створити genesis block, замайнити його додати в ланцюг;
    1. Дізнатися висоту останнього блока та його хеш;
    2. Створити новий блок, замайнити його та додати в ланцюг;
3. Повернення до пункту 1;
3. Вивести в термінал отриману структуру даних;
4. Бізнес-логіка програмного продукту предбачає:
    4.1. Хешування двома раундами за алгоритмом SHA256;
    4.2. Складність майнінгу передбачає знаходження хешу з префіксом «0000»;
    4.3. Оскільки на даному етапі ми не генеруємо транзакції, то в якості тразакції використовуємо string “<Your_name> sent {BlockHeight} coins to Alice”.

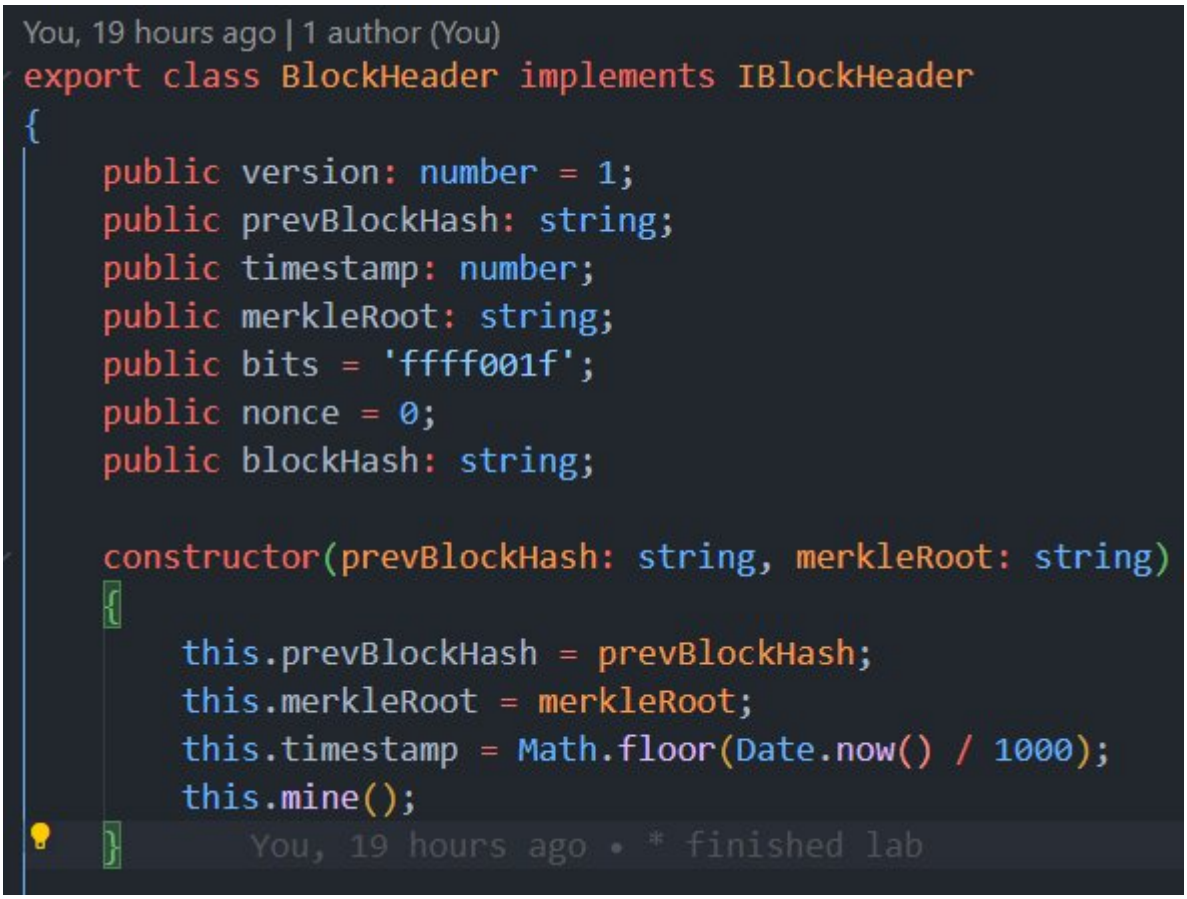
Хід виконання роботи

Для виконання обрано мову програмування TypeScript, оскільки вона є досить розповсюдженою у сфері web3 розробки (найчастіше у зв'язці з бібліотекою ethers.js). Середа розробки – Visual Studio Code

Для початку створимо інтерфейси для класів з запропонованої архітектури:



Почнемо імплементацію з найнижчого рівня імплементації запропонованої архітектури ПЗ – Block Header:



Ініціалізуємо клас з полями з інтерфейсу, оскільки згідно завдання version та target difficulty bits задано хардкодом – їх ініціалізуємо одразу при об’явленні. Хеш попереднього блоку та хеш кореня дерева хешей передаємо до конструктора, ініціалізуємо таймстемп через JS Date api та викликаємо функцію майнинга з конструктора.

Функція mine:

```
public mine()
{
    let resultHash: string = '';
    while (!resultHash.startsWith('0000'))
    {
        const concatenatedValuesLE = [
            this.version.toString(16),
            this.prevBlockHash,
            this.merkleRoot,
            this.timestamp.toString(16),
            this.bits,
            this.nonce.toString(16)
        ].map(value => value.padStart(4, '0').match(/../g)?.reverse().join('')).join('');
        resultHash = sha256Encode(sha256Encode(concatenatedValuesLE));
        this.nonce += 1;
    }
    this.blockHash = resultHash;
}       You, 19 hours ago • * finished lab
```

Для обчислення хешу з заданих значень усі поля з типом number конвертуємо в hex строку, далі для кожного значення створюємо масив бітів, інвертуємо їх порядок, трансформуємо масив кожен бітів у строку через перший join, отриманий масив бітів у little-endian форматі конкатенуємо другим join.

Отриману строку хешуємо у два раунди за алгоритмом sha256, що використовує хешування з бібліотеки crypto:

```
cryptography.ts > sha256Encode
You, 19 hours ago | 1 author (You)
import * as crypto from 'crypto';

export function sha256Encode(input: string)
{
    const hash = crypto.createHash('sha256').update(input).digest('hex');

    return hash;
}       You, 19 hours ago • * finished lab …
```

В кінці циклу отримуємо хеш та бампимо значення nonce поки хеш не буде починатися з 0000 згідно завдання.

Далі імплементуємо клас блоку, аналогічно задаємо значення з завдання:

```
You, 19 hours ago | 1 author (You)
import { IBlock, IBlockHeader, zeroHash } from './../types';
import { BlockHeader } from './block-header';
import { MerkleTree } from './merkle-tree/merkle-tree';

You, 19 hours ago | 1 author (You)
export class Block implements IBlock
{
    public height: number;
    public blockSize: number = 1;
    public blockHeader: IBlockHeader;
    public txCount: number;
    public txs: string[];

    constructor(prevBlockHash: string | null, height: number)
    {
        this.height = height;
        this.txs = [ `Andrii Muravlov sent ${ this.height } coins to Alice` ];
        this.txCount = this.txs.length;
        const merkleRoot = new MerkleTree(this.txs).RootHash;
        console.log(`Mining block with number ${ height }`);      You, 19 hou
        this.blockHeader = new BlockHeader(
            prevBlockHash || zeroHash(),
            merkleRoot
        );
    }
}
```

Згідно завдання поле blockSize хардкодимо, блок має лише одну транзакцію у представленні Name sent blockHeight coins to Alice. Звісно, можна отримати merkleRoot для такого випадку лише отримавши хеш цієї транзакції, однак для scalability можливих майбутніх завдань імплементуємо клас який буде

приймати у себе масив транзакцій та створювати дерево хешів:

```
You, 19 hours ago | 1 author (You)
export class MerkleTreeChildNode
{
    public hash: string;

    constructor(public data: string)
    {
        this.hash = sha256Encode(data);
    }
}

You, 19 hours ago | 1 author (You)
export class MerkleTreeNode
{
    public hash: string;

    constructor(public leftChild: any, public rightChild: any)
    {
        this.hash = sha256Encode(leftChild.hash + rightChild.hash);
    }
}

export function generateLevel(nodes: MerkleTreeChildNode[])
{
    const result: MerkleTreeNode[] = [];
    while (nodes.length > 1)
    {
        const first = nodes.shift();
        const second = nodes.shift();
        result.push(new MerkleTreeNode(first, second));
    }
    if (nodes.length == 1)
    {
        const last = nodes.shift();
        result.push(new MerkleTreeNode(last, undefined));      You, 19 ho
    }
    return result;
}
```

MerkleChildNode – задає хеш дочірнього елемента дерева, MerkleTreeNode - задає хеш нової ноди, функція generateNode Отримує масив усіх вузлів дерева хешів. Імплементація дерева хешів:

```
You, 19 hours ago | 1 author (You)
export class MerkleTree
{
    private root: string;

    constructor(txs: string[])
    {
        let childNodes = txs.map(tx => new MerkleTreeChildNode(tx)) as any;
        while (childNodes.length > 1)
        {
            childNodes = generateLevel(childNodes);
        }
        this.root = childNodes[ 0 ].hash;       You, 19 hours ago • * finished
    }

    public get RootHash()
    {
        return this.root;
    }
}
```

Усі отримані транзакції спочтаку хешуються а потім для кожної створюється вузол доти поки масив нод не буде містити лише хеш усіх транзакцій – він і є коренем дерева. Записуємо цей хеш у приватне поле root та створюємо гетер щоб отримати його значення.

Далі у класі блоку для поля blockHeader ініціалізуємо відповідний клас та передаємо до нього значення кореня дерева хешів та хешу попереднього блоку (для генезіс блоку це значення буде null, у такому випадку передаємо функцію що повертає нульовий хеш в 32 байти):

```
export function zeroHash()
{
    let hashStr = '';
    let counter = 0;
    while (counter < 64)
    {
        hashStr += '0';
        counter++;
    }
    return hashStr;
};
```

Далі створюємо клас блокчейну:

```
const CHAIN_LIMIT_FOR_DEMO = 5;

You, 19 hours ago | 1 author (You)
export class BlockChain implements IBlockchain
{
    private chain: Block[] = [];      You, 19 hours ago • * finished lab

    constructor()
    {
        this.genesisBlock();
        let startTs = Date.now();
        while (this.chain.length < CHAIN_LIMIT_FOR_DEMO)
        {
            this.addBlock();
        }
        console.log(`Mined ${ CHAIN_LIMIT_FOR_DEMO } blocks in ${ Math.floor((Date.now() - startTs) / 1000) } seconds`);
        console.log(this.chain);
    }

    public genesisBlock()
    {
        this.chain.push(new Block(null, this.chain.length));
    }

    public addBlock()
    {
        this.chain.push(new Block(this.chain.slice(-1)[ 0 ].blockHeader.blockHash, this.chain.length));
    }
}
```

Згідно завдання має поле chain як список блоків у блокчейні, у конструкторі викликаємо створення генезис блоку та стоврення довільної кількості блоків заданих глобальною константою. Після того як усі блоки було створено та замайнено робиться вивід даних про час їх генерації та власне сам список блоків.

У головному файлі ініціалізуємо клас блокчкейну, та транспайлимо typescript у javascript (npx tsc):

```
You, 19 hours ago | 1 author (You)
import { BlockChain } from './utils/blockchain';

const blockchain = new BlockChain();      You, 19 hours ago • * finished lab
```

Отримуємо затранспайлений бандл:

Запускаємо отриманий код за допомогою команди node .\dist\index.js:

```
PS D:\Documents\University\SMP\Blockhain> node .\dist\index.js
Mining block with number 0
Mining block with number 1
Mining block with number 2
Mining block with number 3
Mining block with number 4
Mined 5 blocks in 41 seconds
[
  Block {
    blockSize: 1,
    height: 0,
    txs: [ 'Andrii Muravlov sent 0 coins to Alice' ],
    txCount: 1,
    blockHeader: BlockHeader {
      version: 1,
      bits: 'ffff001f',
      nonce: 15046,
      prevBlockHash: '0000000000000000000000000000000000000000000000000000000000000000',
      merkleRoot: '854ec80d93fe30ede7edb67d9dbbcc83f221fc3f6b72fd831503dee6d189f4a5',
      timestamp: 1671367561,
      blockHash: '0000dc39f5231071d2f4c5ec18b61a93df26108ac5ac8c5f57071c3bb25d3ac9'
    }
  },
  Block {
    blockSize: 1,
    height: 1,
    txs: [ 'Andrii Muravlov sent 1 coins to Alice' ],
    txCount: 1,
    blockHeader: BlockHeader {
      version: 1,
      bits: 'ffff001f',
      nonce: 1048629,
      prevBlockHash: '0000dc39f5231071d2f4c5ec18b61a93df26108ac5ac8c5f57071c3bb25d3ac9',
      merkleRoot: 'b8f90c20d07bf8f0c3fdcffcb1d2ef5cdb75519e1ead80fd0a6c0b13bf4e161a',
      timestamp: 1671367561,
      blockHash: '000076c43342d1ff7077789079a9b336efb19f62a2995f8c30e7720775c2b453'
    }
  },
  Block {
    blockSize: 1,
    height: 2,
    txs: [ 'Andrii Muravlov sent 2 coins to Alice' ],
    txCount: 1,
    blockHeader: BlockHeader {
      version: 1,
      bits: 'ffff001f',
      nonce: 1164972,
      prevBlockHash: '000076c43342d1ff7077789079a9b336efb19f62a2995f8c30e7720775c2b453',
      merkleRoot: '04ebf25e10d89f5bfdc2c621efe3ba110253fef392619d2a492cfb281d2759de',
      timestamp: 1671367574,
      blockHash: '000047543cc91d5caf52e10278dc99a669732e71c7e1f8758d451f8212d6acfe'
    }
  },
  Block {
    blockSize: 1,
    height: 3,
    txs: [ 'Andrii Muravlov sent 3 coins to Alice' ],
```

ВИСНОВКИ

При виконанні лабораторної роботи ознайомився з архітектурою блокчейна, та розробив програмний продукт, що його генерує. Ознайомлено з поняттями майнінгу, target difficulty, струтури блоків та дерева хешів. Для більш наочної репрезентації можна відрефакторити дерево хешів, з реальними полями транзакцій та їх хешуванням, також додати клас ноди, який буде абстракцією над майнером та задати їм різні параметри перебору nonce.
Лістинг коду викладено в github репозиторії:
https://github.com/lakub-muravlov/blockchain-example