POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATION
Institute of Computing Science

Laboratory report

BLOCKCHAIN IMPLEMENTATION

Aleksander Zioło, 145433
Krzysztof Michalak, 146529

POZNAŃ 2023

Project description

The blockchain application is written in Python using Flask and is controlled by the server. Clients can log in to their accounts and generate new blocks, view the chain, or assert its integrity. Our system's goal is to allow appending user input to the blockchain in the way the authenticity of the message is indisputable and accessible to any other member.

Design choices

We have decided to utilize *Proof of Work* (PoW) consensus mechanism. Each time a user wants to append a block to the chain, other hosts verify appended proof value, calculate hash of *JSON* representation of a block structure and merge it if the proof is correct. This approach ensures valid state of the chain and reject abuse attempts.

Implementation

```python
import hashlib
import json
import datetime
from random import randint

class Blockchain:

    def __init__(self):
        self.chain = []
        self.append_block(message='', author='SYSTEM', timestamp=str(datetime.datetime.now()), proof=randint(1,9999999))

    def pow_hash(self,proof1, proof2):
        return hashlib.sha256(str(proof1**2 - proof2**2).encode()).hexdigest()

    def block_hash(self, block):
        encoded_block = json.dumps(block, sort_keys=True).encode()
        return hashlib.sha256(encoded_block).hexdigest()

    def append_block(self, message, author, timestamp, proof): #block generation
        if len(self.chain)>0:
            previous_hash = self.block_hash(self.chain[-1])
            previous_proof = self.chain[-1]['proof'] #verifying that block candidate met the requirement
            hash_op = self.pow_hash(proof,previous_proof)
            if hash_op[:5] != '00000':
                return None
        else:
            previous_hash  = '0'

        block = {'index': len(self.chain) + 1,
                 'message': message,
                 'author': author,
                 'timestamp': timestamp,
                 'proof': proof,
                 'previous_hash': previous_hash}
        self.chain.append(block)
        return block
```

FIGURE 1: Class declaration and first set of functions implemented in Blockchain: constructor, *pow_hash* and *block_hash* hashing methods, as well as *append_block*.

The **Blockchain** class contains all required components for chain manipulation.

- **constructor** initializes the chain list and calls *append_block*. The genesis block has a default message of '', author as 'SYSTEM', the current timestamp, and a proof value generated randomly between 1 and 9999999 to randomize next sequences.

- **pow_hash** calculates the hash of the difference between proof1 squared and proof2 squared using the SHA-256 hashing algorithm. This is one of the most common formulas as it is not too challenging to solve.
- **block_hash** encodes the block to JSON format and calculates hash of that block.

```
def proof_of_work(self): #calculating the proof that client can add a block
    previous_proof = self.chain[-1]['proof']
    current_proof = 1
    check_proof = False
    while check_proof is False:
        hash_op = self.pow_hash(current_proof,previous_proof) #calculate until first five bytes are 0s
        if hash_op[0:5] == '00000':
            check_proof = True
        else:
            current_proof += 1
    return current_proof

def check_chain_validity(self): #recalculating of all hashes and proofs
    prev_blk = self.chain[0]
    index_blk = 1
    while index_blk < len(self.chain):
        block = self.chain[index_blk]
        if block['previous_hash'] != self.block_hash(prev_blk): #checking block integrity
            return False
        previous_proof = prev_blk['proof']
        proof = block['proof']
        hash_op = self.pow_hash(proof,previous_proof) #checking if author has actually met the requirements to append a block
        if hash_op[:5] != '00000':
            return False
        prev_blk = block
        index_blk += 1
    return True
```

FIGURE 2: Second set of functions implemented in Blockchain: *proof_of_work* and *check_chain_validity*.

- **proof_of_work** iterates over value that squared with the proof of a previous block returns hash. If the first 5 characters of that hash are zeros, the blockchain considers it as a valid proof and a condition to append a block is met.
- **check_chain_validity** method verifies the whole blockchain by iterating over it and recalculating all of the hashes and checking proofs.

Web Application

The web front-end consists of 5 pages: Login, Home, Mine, Show chain and Validate.
On the login page user is required to authenticate via login and password. Fields are validated and password hashed using SHA-256.

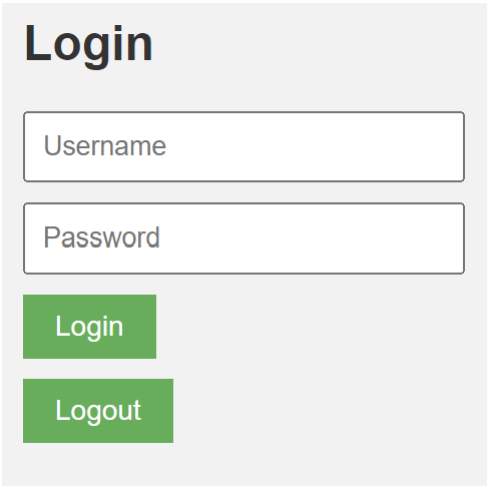

FIGURE 3: Home page

Home page has 4 buttons: *Mine new block* which allows adding new block to the blockchain, *get chain* which shows current state of the blockchain, *valid* that checks the validity of a chain and a *logout*.

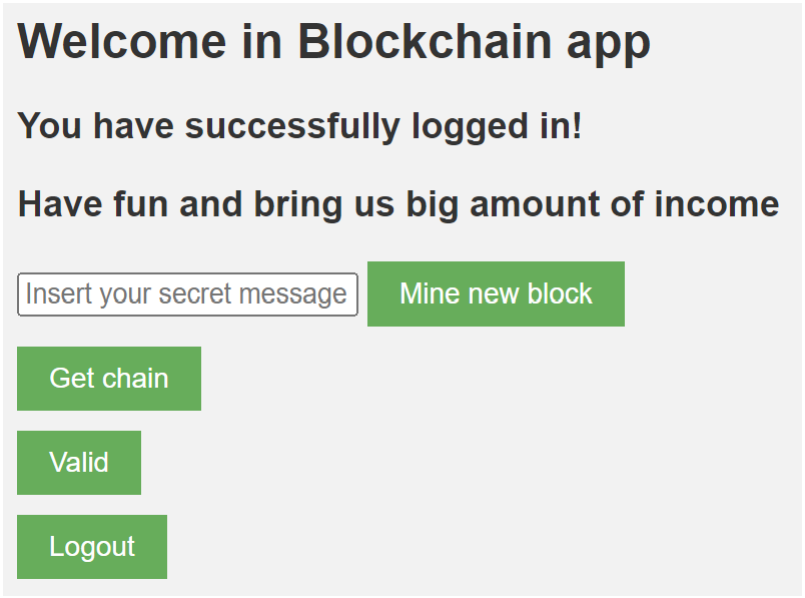

FIGURE 4: Login page

Appending the message to the new block before mining.

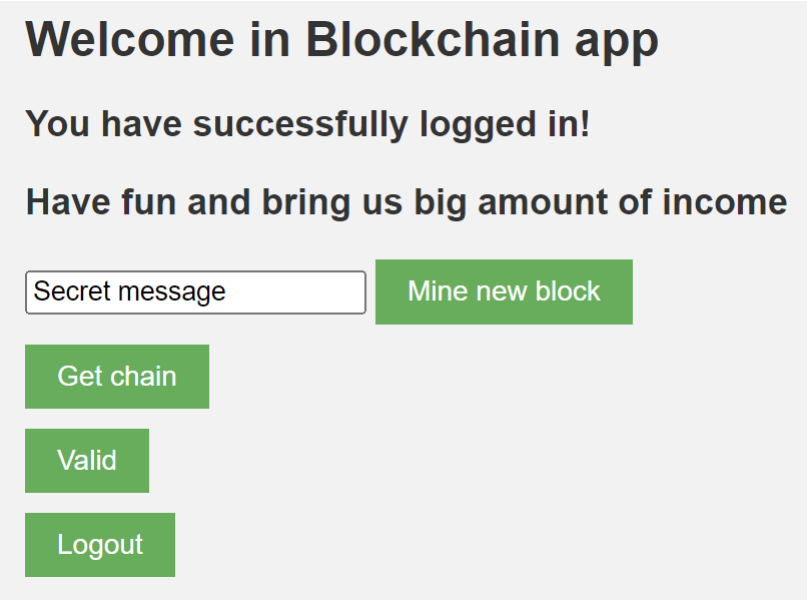

FIGURE 5: Home page with a message

After finishing calculating of the proof, a new block summary will be displayed along with a message, index of the block, timestamp, proof and hash of previous block.

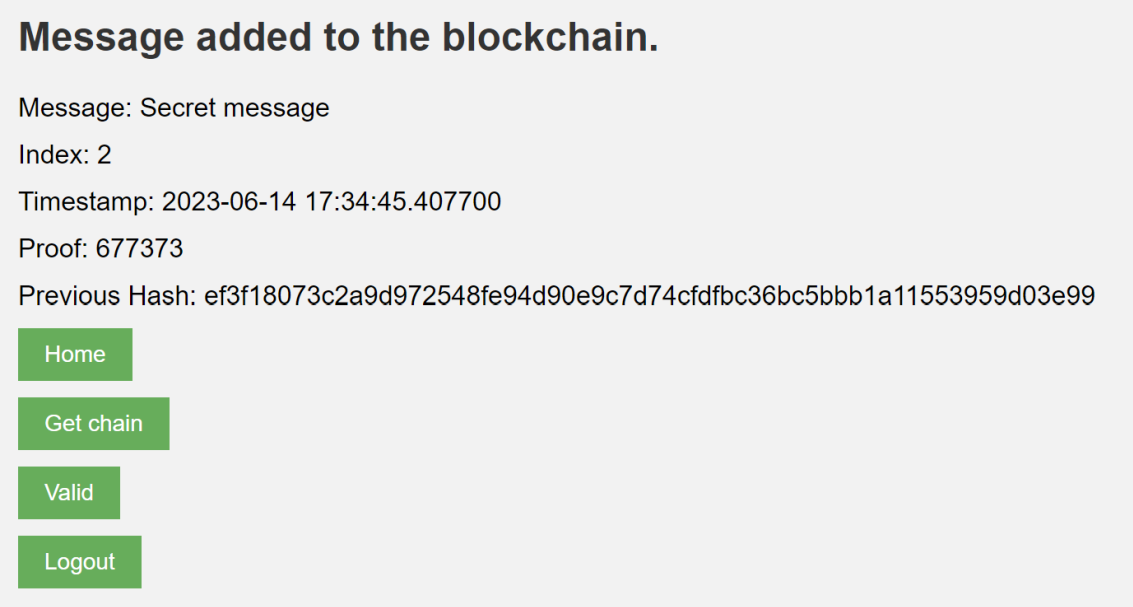

FIGURE 6: Adding a new block to blockchain

All blocks can be viewed on *Get chain* page.

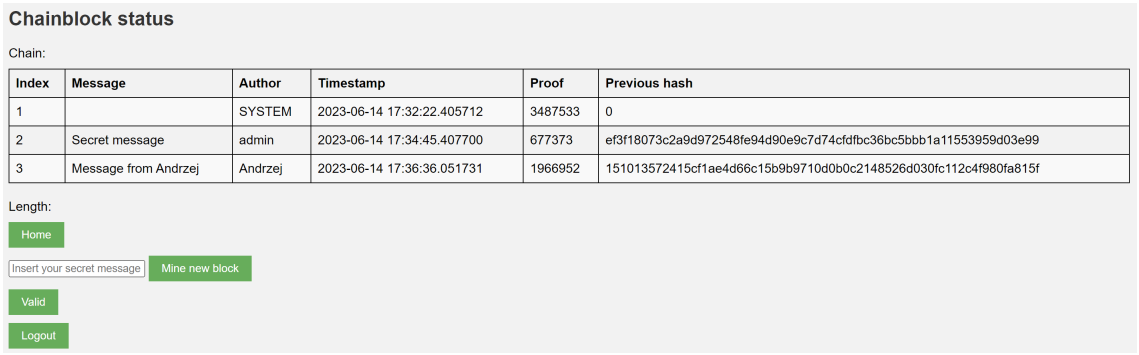

FIGURE 7: Checking all blocks in blockchain

On *Valid* page any user can request a blockchain validity check.

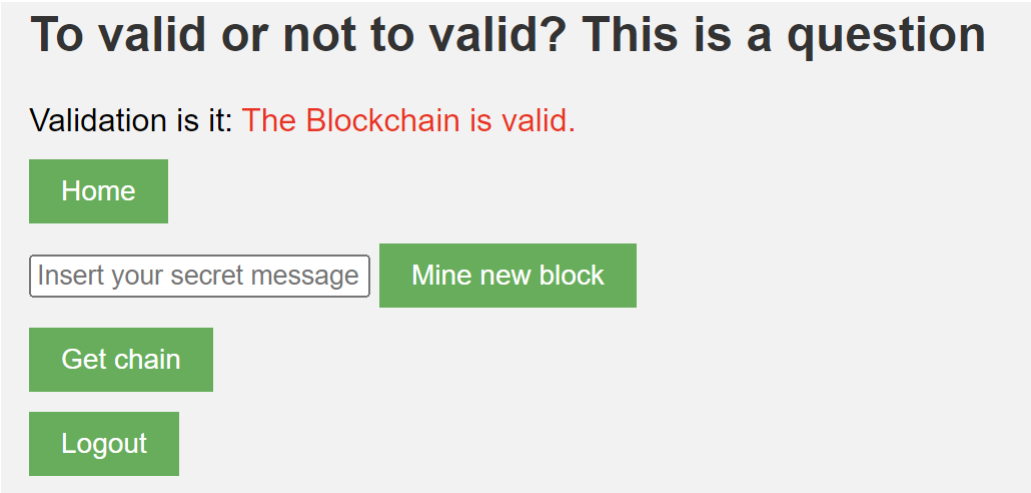

FIGURE 8: Checking validity of blockchain

Further development

The blockchain's application could be greatly expanded by implementing following additional features:

- private and public keys to authenticate and validate users;
- client-sided PoW calculation using JavaScript;
- adding more user-friendly conversation view.

Poznan University of Technology
Faculty of Computing and Telecommunication
Institute of Computing Science

Typeset using LaTeX.