Blockchain report

Katarzyna Badio 145306, Julia Chabora 145218

14 czerwca 2023

1 The goal of the project

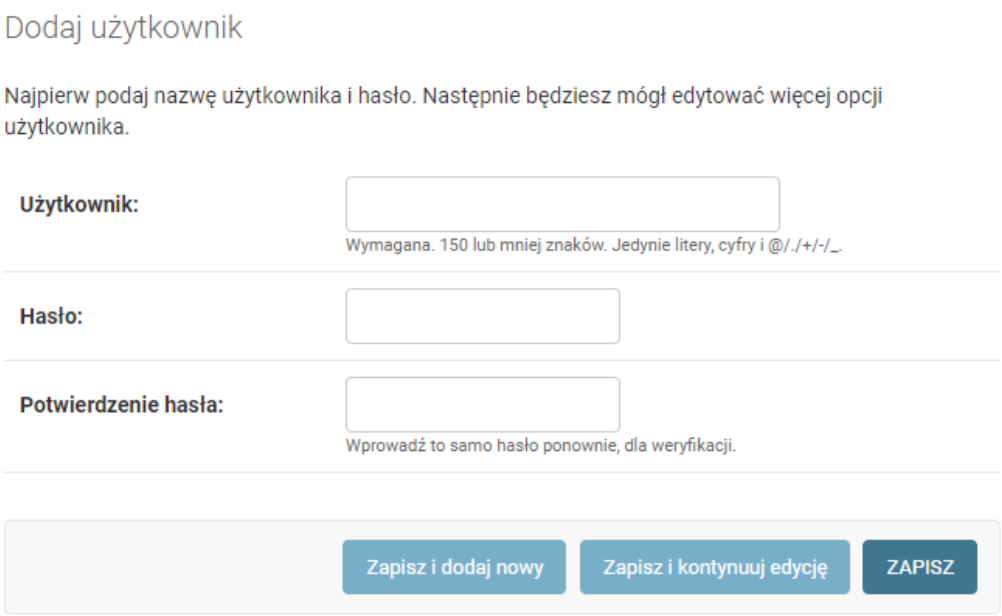
The objective of this project is to implement a basic blockchain structure that can be used to store and verify transactions. The blockchain should include the necessary components such as blocks, transactions and cryptographic mechanisms to ensure data integrity and security.

2 Implementation of security features and measures to prevent unauthorised access

To launch a project, one must create users by clicking on an administration page (1) and entering username and password (2). To these users transactions will be sent. To add private key and public key for the user, select option for profile adding. Key generation comes from page *https://travistidwell.com*. From this point, users can add transactions and mine blocks. First block is always added as a default in a project.

Rysunek 1: Main administration page

There is authentication of a user and access control. *Django* itself allows for native login. For example, when a user is not logged in and wants to go



Rysunek 2: Adding user

to a subpage *add_transaction*, user will be redirected to the login page again. In code, it is ensured by adding *@login_required* at the top of the function.

3 Implementation details and design decisions

First, a user wants to add a transaction (3) and a function *add_transaction* loads html page. A function to manage transaction, *add_transaction_manage* (5), has a POST method. A new instance of a transaction is created, a sender is a user from a request method. Recipient is a person who is chosen in an html form.

3.1 Effective use of cryptographic mechanisms to ensure the validity of the blocks

Transactions and blocks must be hashed. First, the whole message is taken, so an sender id, recipient id, amount and a timestamp. The proper hashing function is named *hash_transaction* (4) and the algorithm used is SHA256. Then, to the instance, which will be hashing, the SHA algorithm is passed.

Use of hashing satisfies the property of integrity of the message. If someone had modified the contents, the hash of the message would be different.

Message is being encoded with utf-8. After this, *sign_transaction* function is performed. It ensures that sender is authenticated with a private key, so a sender won't be impersonated. This is assigned as a signature. Then, to check if a transaction is correct, we would check if a signature is correct, based on a public key. Transaction is saved.

```
38  @login_required
39  def add_transaction(request):
40      if request.method == 'GET':
41          user_list = User.objects.order_by("username")
42          return render(request, 'project/transaction_add.html',
43                        {"users": user_list})
44
```

Rysunek 3: Adding transaction

```
77      def hash_tansaction(self):
78          message = self.get_transaction_message()
79          algorithm = hashes.SHA256()
80          hasher = hashes.Hash(algorithm)
81          hasher.update(message.encode('utf-8'))
82          hash_bytes = hasher.finalize()
83          hash_hex = binascii.hexlify(hash_bytes).decode('utf-8')
84          return hash_hex
```

Rysunek 4: Hash transaction

Then, the blocks are created with one or more transactions in each of them (5). Blocks are mined and it means that the nonce for a block is being recalculated. To the mined block, no new transactions can be added. Block has a reference to the previous blocks, because it uses a hash of previous block to create its own hash.

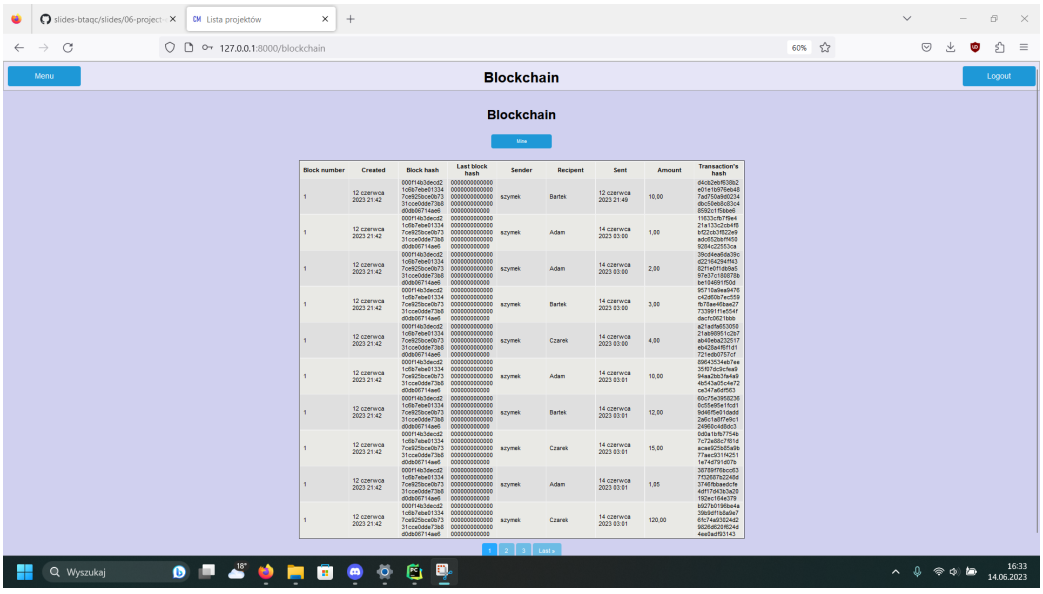
For a block, there is a place for ten transactions and before adding one, it is checked if block is full. In the picture number 6, blocks are mined. First of blocks which has not been mined yet is chosen as the next one. For each row in the table, there is a timestamp, a block hash, sender, recipient and transaction hash.

```python
@login_required
def add_transaction_manage(request):
    if request.method == 'POST':
        if request.POST.get('user_id') and request.POST.get('amount'):
            transaction = Transaction()
            transaction.sender = request.user
            transaction.recipient = User.objects.get(id=request.POST.get('user_id'))
            transaction.amount = request.POST.get('amount')
            transaction.transaction_hash = transaction.hash_tansaction()
            transaction.sign_transaction()
            transaction.save()

            current_block = Block.objects.latest('id')
            if current_block.is_full():
                # Create a new block
                new_block = Block(previous_block=current_block, nonce=current_block.generate_new_nonce(),
                                  block_number=current_block.block_number+1,)
                new_block.save()
                current_block = new_block
            current_block.transactions.add(transaction)

            messages.add_message(request, messages.SUCCESS, 'Transaction will be added to the blockchain!')
            return redirect(add_transaction)
        else:
            messages.add_message(request, messages.ERROR, 'Error! Some fields are empty!')
            return redirect(add_transaction)
    return redirect(add_transaction)
```

Rysunek 5: Managing adding transaction



Rysunek 6: Blocks mining

At first, new nonce is generated and it is used in a hashing function (7) in a while loop. Normally, it would use a graphic card for mining, and here it creates the nonce for the calculation part. The target is reached when a hash has a number of zeroes at the beginning, which is a set up difficulty. After this, the hash is assigned as a new one and the block is marked as mined.

```
145     def mine_block(self):
146         target = "0" * DIFFICULTY
147 
148         while True:
149             self.nonce = random.randint(0, 2 ** 32 - 1)
150             block_hash = self.hash_function()
151 
152             if block_hash.startswith(target):
153                 self.block_hash = block_hash
154                 break
```

Rysunek 7: Mine block function

3.2 Proper implementation of consensus mechanisms to ensure the validity of blocks

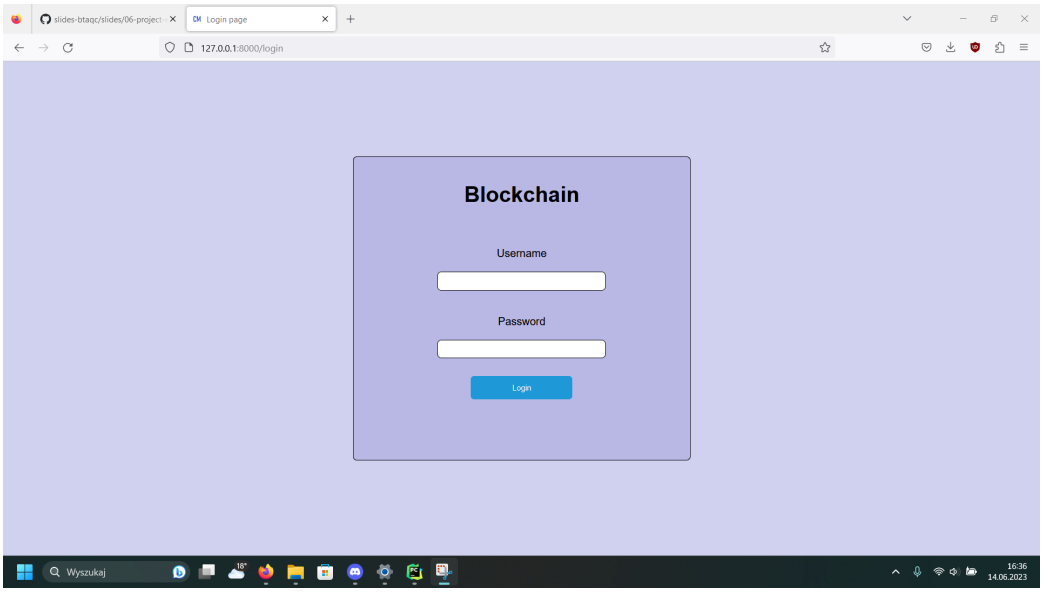
In the implementation, proof of work is checked. So, after the block is mined, it must be checked if other users agree to the fact and that the whole hashing is properly done.

Each user votes for his own block and creates a vote for it, because he certainly wants it for his block to be true and it is added to the database. Then other users will vote. With each new login to the system, the function for checking all votes is run. Validation function has subfunctions *validate_block*, *validate_hash* and *validate_transactions*.

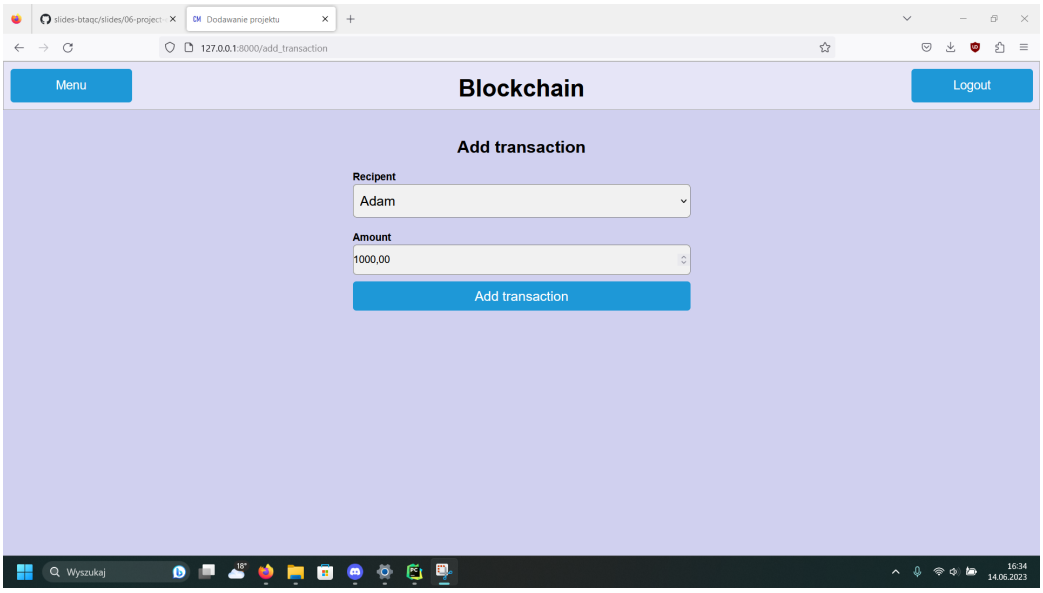
If all users who have logged in between three days, have voted, then we can check how many votes are for yes and how many for no. If yes is greater or equal or greater than no, the block is validated. Otherwise, it is deleted with all the transactions and links to the blocks which have been built on it.

3.3 Design and usability of user interface

User interface is simple and intuitive, created for desktop. The language for an application is english. The actions for each page are described at the top of the forms, as seen in the picture 8 and 9.



Rysunek 8: Login page



Rysunek 9: Add transaction page

4 Test results

Django and database used allows for a storage of many users, blocks and transactions. While testing the solution, with the provided data, no latency has been noticed and all properties for the blockchain mechanisms like hashing and consensus mechanism are provided. If a block is not correct, a validation method is run, so that all blocks are correctly linked to each other.