

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей  
Кафедра информатики

**ОТЧЕТ**  
к лабораторной работе  
на тему  
**ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА**

Студент  
Руководитель  
Нормоконтролер  
Рецензент

Т. А. Русакович  
Е. В. Тушинская  
Е. В. Тушинская  
Е. В. Тушинская

Минск 2023

## СОДЕРЖАНИЕ

Введение.....	3
1 Описание программного продукта .....	4
2 Тестирование приложения .....	9
3 Развертывание приложения .....	10
3.1 Диаграмма развертывания.....	10
3.2 Развертывание приложения и другие примечания.....	11
Заключение .....	12
Приложение А (обязательное) Листинг кода.....	13

## **ВВЕДЕНИЕ**

Тестирование программного продукта играет важную роль в обеспечении его качества, надежности и эффективной работы. Это процесс, позволяющий выявлять ошибки, проверять функциональность и убеждаться в соответствии программы требованиям, прежде чем она попадет к пользователям.

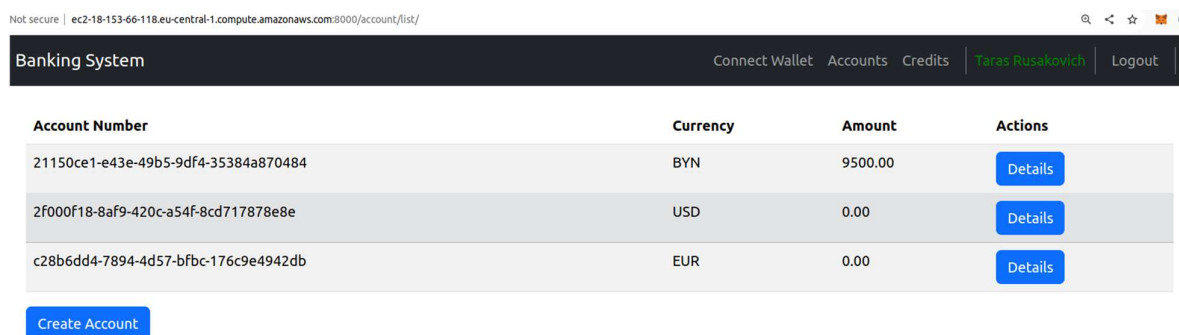
Тестирование программного продукта охватывает широкий спектр методов направленных на проверку различных аспектов программы. Это включает модульное, интеграционное и системное тестирование, а также другие виды тестирования, необходимые для обеспечения качества программного продукта.

В данной лабораторной работе будет разработано веб-приложение банка, а также написаны модульные тесты для его основных функций.

# 1 ОПИСАНИЕ ПРОГРАММНОГО ПРОДУКТА

Разработанное приложение представляет собой упрощенный вариант банковской системы, в которой пользователь может создавать счета, переводить между ними средства в различной валюте, брать и оплачивать кредит, а также покупать токен белорусского рубля в EVM-based блокчейне.

При входе на сайт пользователь должен сначала зарегистрироваться, либо войти в свой аккаунт если он уже зарегистрирован. При входе пользователю на почту отправляется шестизначный код для двухфакторной аутентификации. При успешном входе пользователя переводит на страницу со списком счетов, также изменяется внешний вид заголовка страницы. На рисунке 1.1 приведен внешний вид страницы счетов.



The screenshot shows a web browser window with the address bar displaying 'Not secure | ec2-18-153-66-118.eu-central-1.compute.amazonaws.com:8000/account/list/'. The page title is 'Banking System'. The navigation bar includes 'Connect Wallet', 'Accounts', 'Credits', and a user profile 'Taras Rusakovich' with a 'Logout' button. The main content area displays a table of accounts with columns: Account Number, Currency, Amount, and Actions. There are three accounts listed, each with a 'Details' button. Below the table is a 'Create Account' button.

Account Number	Currency	Amount	Actions
21150ce1-e43e-49b5-9df4-35384a870484	BYN	9500.00	<a href="#">Details</a>
2f000f18-8af9-420c-a54f-8cd717878e8e	USD	0.00	<a href="#">Details</a>
c28b6dd4-7894-4d57-bfbc-176c9e4942db	EUR	0.00	<a href="#">Details</a>

[Create Account](#)

Рисунок 1.1 – Страница списка счетов пользователя

Приложение включает в себя следующие разделы:

- страница входа в приложение и страница регистрации;
- страница со списком счетов пользователя;
- страница деталей по счету, которая включает функционал переводов, покупки токенов и удаления;
- страница со списком кредитов пользователя;
- страница создания кредита;
- страница деталей кредита, которая включает функционал оплаты кредита и смены счета;

Все страницы, требуют наличия авторизованного пользователя, соответственно, пользователь получает ошибку со ссылкой на странице входа при попытке неавторизованного или недоступного взаимодействия.

Заголовок базовой страницы включает в себя ссылки на страницы со списками счетов и кредитов, а также кнопку подключения кошелька MetaMask.

На рисунке 1.2 приведено изображение страницы со списком кредитов пользователя.

Credit Number	Currency	All amount to pay	Next payout	Actions
55f5ae36-3f23-4d07-84fe-841465dd602a	USD	11499.99	12/05/2024	<a href="#">Details</a>
77604c46-97e8-434a-8f09-bdeada19edaa	BYN	11400.12	01/09/2024	<a href="#">Details</a>

[Create Credit](#)

Рисунок 1.2 – Страница списка кредитов пользователя

Страница содержит список кредитов пользователя с валютой, суммой, которую осталось выплатить, датой следующей выплаты, кнопкой со ссылкой на страницу деталей кредита и кнопкой со ссылкой на страницу создания кредита.

На рисунке 1.3 приведено изображение страницы создания кредита.

Duration in month:

Payment type:

Sum of credit:

Account:

Rate percent: 0.15

[Calculate](#)

Sum to pay for the credit: 11500.0

[Take](#)

Рисунок 1.3 – Страница создания кредита

На данной странице пользователь может выбрать срок кредита, тип выплат, сумму, счет, получить информацию о процентной ставке, взять кредит. Кредит будет открыт в валюте указанного счета. Поля с типом кредита обновляются в соответствии с возможными заданными.

На рисунке 1.4 приведено изображение страницы деталей кредита.

Not secure | ec2-18-153-66-118.eu-central-1.compute.amazonaws.com:8000/credit/55f5ae36-3f23-4d07-84fe-841465dd602a/

Banking System

Connect Wallet Accounts Credits Taras Rusakovich Logout

### Credit Information

Owner:	Taras Rusakovich		
Credit Number:	55f5ae36-3f23-4d07-84fe-841465dd602a		
Currency:	USD		
Amount To Pay:	11499.99		
One Payout:	3833.33		
Next Payout:	12/05/2024		
One Current Payout:	3833.33		
Count Of Payouts:	3		
Account:	2f000f18-8af9-420c-a54f-8cd717878e8e	21150ce1-e43e-49b5-9df4-35384a870484	Change
Created At:	12/11/2023		
Updated At:	12/11/2023		

Payment:  Payment

Рисунок 1.4 – Страница деталей кредита

На данной странице пользователь получает полную информацию по выбранному кредиту. Информацию по сумме выплат рассчитанную на заданный при создании единичный период выплат, общую сумму, которую осталось выплатить, и сумму, которую осталось выплатить до следующего единичного периода выплат. Внизу страницы есть поле для ввода суммы выплат и кнопка проведения выплаты. Справа находится список счетов и кнопка смены кредитного счета.

На рисунке 1.5 представлено изображение страницы деталей по счету.

Not secure | ec2-18-153-66-118.eu-central-1.compute.amazonaws.com:8000/account/21150ce1-e43e-49b5-9df4-35384a870484/

Banking System

Connect Wallet Accounts Credits Taras Rusakovich Logout

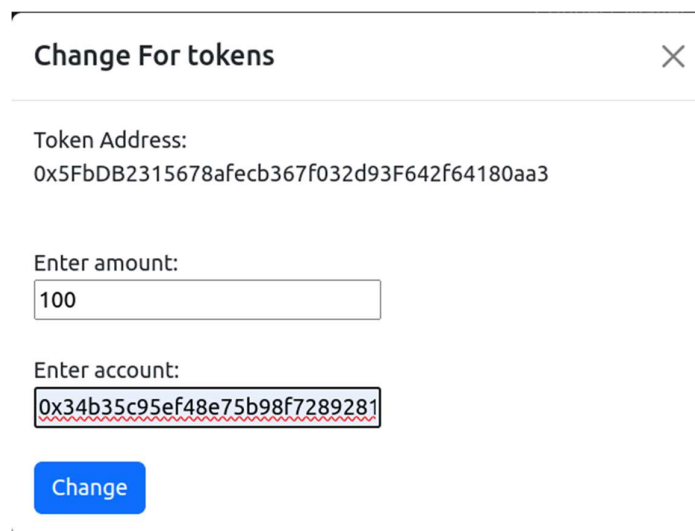
### Account Information

Owner:	Taras Rusakovich		
Account Number:	21150ce1-e43e-49b5-9df4-35384a870484		
Currency:	BYN		
Amount:	9500.00		
Created At:	12/10/2023		
Updated At:	12/10/2023		

Buy Tokens Transfer Delete

Рисунок 1.5 – Страница деталей по счету

На данной странице приведена информация о счете и сумме на нем. Присутствует кнопка для покупки токенов, окно покупки представлено на рисунке 1.6. Для покупки необходимо указать сумму, которая будет списана со счета, и адрес в блокчейне, на который будут переведены токены. Сверху указан адрес токена, который можно добавить в кошелек.



**Change For tokens** ✕

Token Address:  
0x5FbDB2315678afecb367f032d93F642f64180aa3

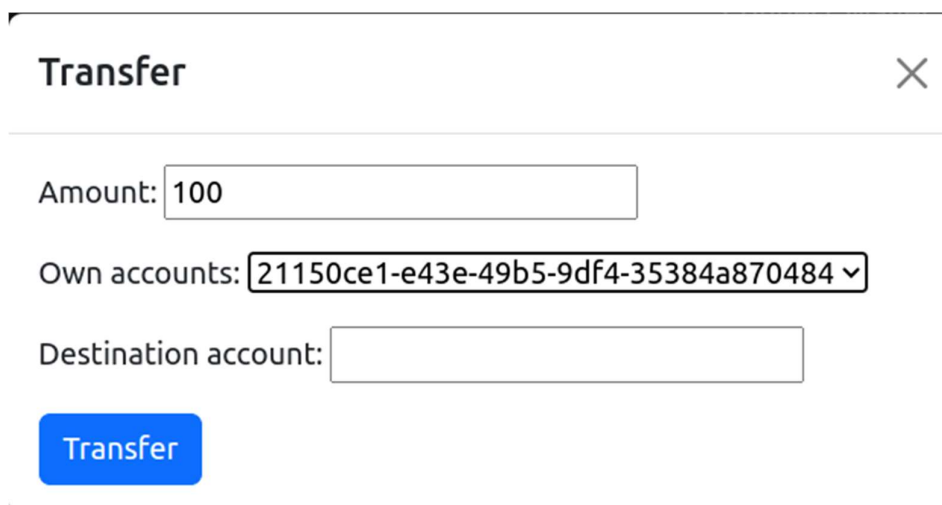
Enter amount:

Enter account:

**Change**

Рисунок 1.6 – Окно покупки токенов

Также имеется кнопка проведения трансфера, окно трансфера указано на рисунке 1.7, где необходимо указать сумму перевода, выбрать один из личных счетов или ввести любой счет самостоятельно. Снизу находится кнопка осуществления перевода.



**Transfer** ✕

Amount:

Own accounts:

Destination account:

**Transfer**

Рисунок 1.7 – Окно трансфера между счетами

На рисунке 1.8 представлено изображение подключения кошельку при нажатии на кнопку подключения.

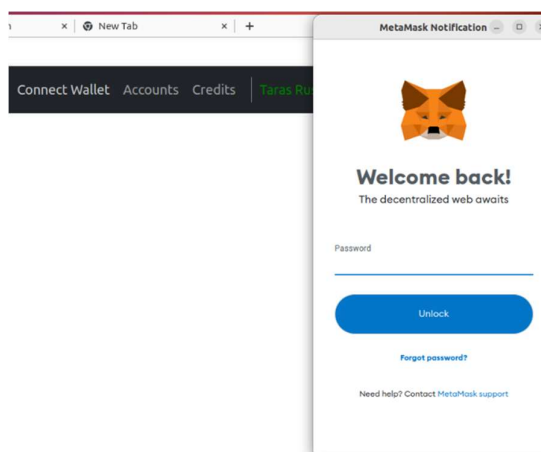


Рисунок 1.8 – Окно подключения к кошельку

Необходимо ввести пароль. В случае отсутствия кошелька пользователь будет перенаправлен на страницу загрузки. В случае, если уже подключен – будет выведено окно с текущим подключенным адресом.

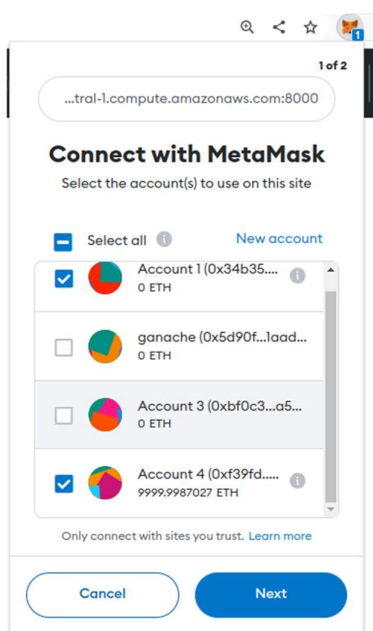


Рисунок 1.9 – Выбор адресов для подключения

Далее необходимо выбрать какие адреса подключить, окно выбора представлено на рисунке 1.9.



## 2 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Для написания тестов приложения используется фреймворк unittest. Он поддерживает автоматизацию тестирования, совместное использование кода настройки и завершения работы тестов, объединение тестов в коллекции и независимость тестов от системы отчетности.

При тестировании целью ставится покрыть тестами основные ключевые части логики приложения. Так как основная логика приложения взаимодействует с базой данных, то unit-тесты будут направлены на тестирование базы данных. Для тестирования запускаются тестовые базы данных и приложение, после проведения тестов, данные удаляются.

В первую очередь проверяются сервисы, которые отвечают за основную логику изменения и получения данных. Основное внимание уделено тестам, которые кроме изменения и получения данных также имеют логику обработки или взаимодействуют с другими сервисами, такие как кредитования, погашение кредитов и переводы. Далее проверяются отображения, которые отвечают за логику получения запросов и отправки ответов, перенаправления и валидации. Также проводятся тесты для авторизации, которые включают в себя проверку работы токена, его создания, получения и расшифровки. Проверяются функции для регистрации и входа пользователя в систему, а также логика двухфакторной аутентификации.

Результаты тестирования представлены на рисунке 2.1.

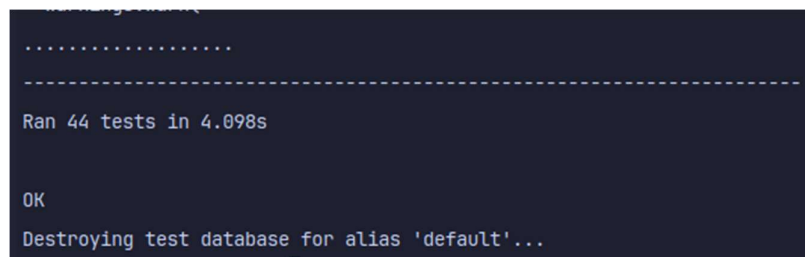


Рисунок 2.1 – Результат тестирования

## 3 РАЗВЕРТЫВАНИЕ ПРИЛОЖЕНИЯ

### 3.1 Диаграмма развертывания

Диаграмма развертывания – это тип UML-диаграммы, которая показывает архитектуру исполнения системы, включая такие узлы, как аппаратные или программные среды исполнения, а также промежуточное программное обеспечение, соединяющее их.

На рисунке 3.1 приведена диаграмма развертывания для разработанного приложения.

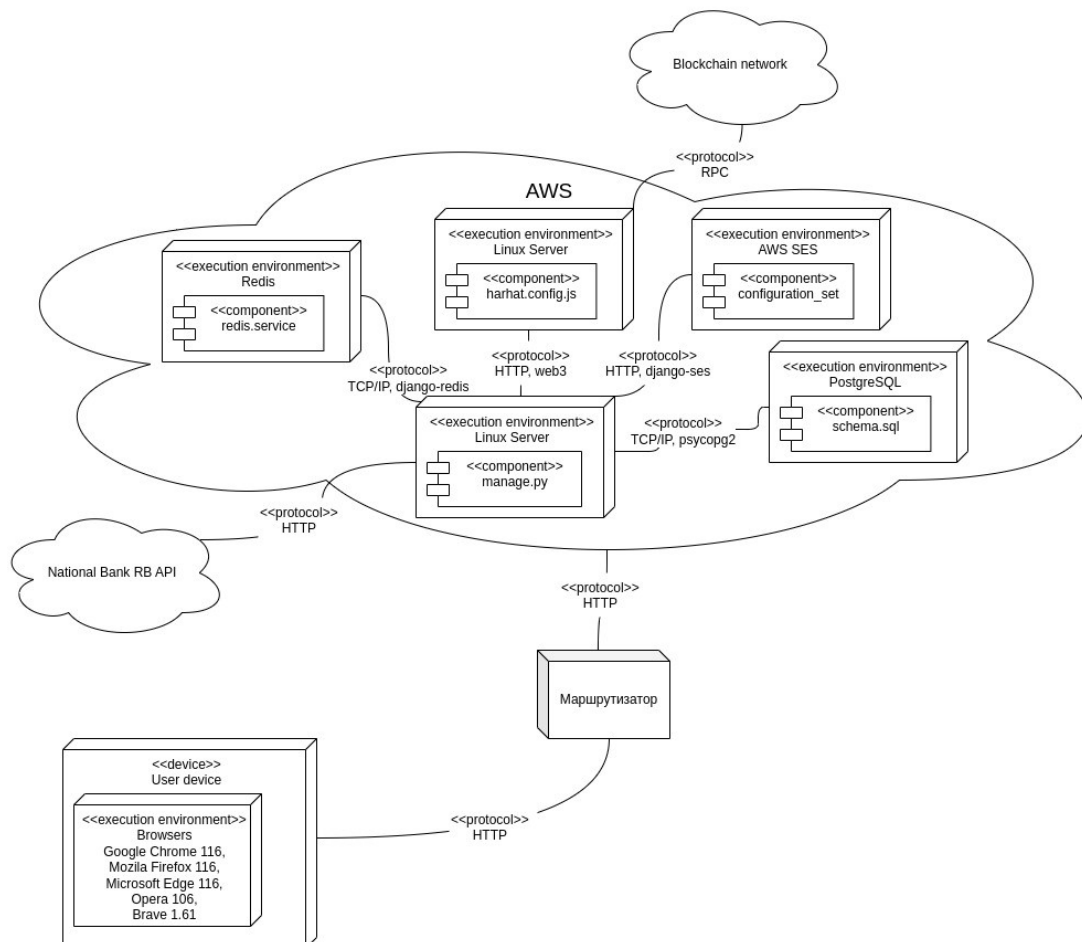


Рисунок 3.1 – Диаграмма развертывания

Диаграммы развертывания помогают моделировать аппаратную топологию системы по сравнению с другими типами UML-диаграмм, которые в основном описывают логические компоненты системы.

### 3.2 Развертывание приложения и другие примечания

Приложение разработано с помощью фреймворка Django и языка программирования Python, также смарт-контрактов использовался язык программирования Solidity, а для создания узла подключения к сети, деплоя и разработки блокчейн-части использовался фреймворк HardHat. Все части приложения для обеспечения переносимости, удобства разработки и упрощения развертывания запускаются в docker-контейнерах, которые в свою очередь запускаются одним docker-compose файлом. Для развертывания был выбран Amazon Web Services провайдер облачных технологий, который предоставляет всевозможные облачные решения.

Для развертывания необходимо создать аккаунт в AWS. Провести настройку SES (Simple Email Service), верифицировать используемые адреса электронной почты, если не получать производственный доступ, то будет возможность отправлять сообщения только верифицированным адресам, а также установить ключи доступа в веб-приложении. Далее создается экземпляр EC2 (Elastic Compute Cloud), это сервис, который предоставляет вычислительные мощности виртуальных/физических машин с заранее заданными требованиями, с возможностью в дальнейшем увеличить мощности. После запуска виртуальной машины необходимо установить на ней докер и гит, через гит получить репозиторий с кодом, а через докер запустить контейнеры. Запустив контейнеры необходимо применить миграции и развернуть контракт в сети блокчейн. После этого приложение будет доступно для использования.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения был разработан полноценный программный продукт упрощенного аналога банковской системы, а также написаны тесты для ключевой логики приложения.

Приложение развернуто в открытом доступе. А также разработана диаграмма развертывания и составлен отчет, оформленный в соответствии с общими требованиями стандарта предприятия БГУИР.

Также было представлено краткое описание программного продукта, которое показывает, как реализованы различные функции банковской системы в спроектированном веб-приложении.

## Приложение А (обязательное) Листинг кода

```
def execute_account_transaction(self, source_account_uuid, destination_account_uuid, amount):
    with transaction.atomic():
        if source_account_uuid == destination_account_uuid:
            raise CustomValueError('Using the same source and destination accounts is not allowed.')

        source_account = Account.objects.get(account_uuid=source_account_uuid)
        destination_account = Account.objects.get(account_uuid=destination_account_uuid)

        if source_account.amount - amount < 0 or amount == 0.:
            raise CustomValueError('Insufficient funds')

        amount_to_send = ExchangeRateAPI().calculate_amount(source_account.currency,
                                                             destination_account.currency, amount)
        source_account.amount -= amount
        destination_account.amount += amount_to_send

        source_account.save()
        destination_account.save()

def exchange_for_token(self, account, amount, bc_account):
    amount_to_get = ExchangeRateAPI().calculate_amount(account.currency, "BYN", amount)
    if account.amount - amount < 0 or amount_to_get == 0:
        raise ValueError('Insufficient funds')
    account.amount -= amount
    account.save()

    w3 = Web3(Web3.HTTPProvider(BC_URL))
    if not w3.is_connected():
        raise ConnectionError("Failed to connect to HTTPProvider")

    with open("web3/artifacts/contracts/BYNToken.sol/BYNToken.json") as abi_file:
        contract_abi = json.load(abi_file)["abi"]

    contract = w3.eth.contract(address=CONTRACT_ADDRESS, abi=contract_abi)
    token_amount = w3.to_wei(amount_to_get, 'ether')
    nonce = w3.eth.get_transaction_count(w3.eth.account.from_key(PRIVATE_KEY).address)

    current_amount = contract.functions.balanceOf(Web3.to_checksum_address("0xf39Fd6e51aad88F6F4ce6aB8827279cFfB92266")).call()
    if current_amount < token_amount:
        raise ValueError(f'There are {current_amount} available tokens')
    transaction = contract.functions.transfer(bc_account, token_amount).build_transaction({
        'chainId': w3.eth.chain_id,
        'gas': 200000, # Adjust the gas limit as needed
        'nonce': nonce,
```

```

    })
    signed_txn = w3.eth.account.sign_transaction(transaction,
PRIVATE_KEY)

    try:
        tx_hash = w3.eth.send_raw_transaction(signed_txn.rawTransaction)
        w3.eth.wait_for_transaction_receipt(tx_hash)
        print(f"Transaction sent! Hash: {tx_hash.hex()}")
    except Exception as e:
        account.amount += amount
        account.save()
        raise e

def credit_payout(self, credit_pk, sum_to_pay):
    credit = self.retrieve_credit_pk(credit_pk)
    account = credit.account_uuid
    if sum_to_pay <= credit.amount_to_pay and sum_to_pay <= ac-
count.amount:
        account.amount -= sum_to_pay

        credit.amount_to_pay -= sum_to_pay
        remember_count = credit.payout_count
        credit.payout_count = math.ceil(credit.amount_to_pay /
credit.one_off_payment)
        credit.one_off_current_payment = credit.amount_to_pay %
credit.one_off_payment
        diff_count = remember_count - credit.payout_count
        if credit.payout_count == 0:
            try:
                with transaction.atomic():
                    credit.delete()
                    account.save()
            except IntegrityError:
                raise IntegrityError()
        else:
            if diff_count != 0:
                if credit.description_uuid.payment_type == "OM":
                    credit.next_payout += datetime.timedelta(days=30 *
diff_count)

                    elif credit.description_uuid.payment_type == "OY":
                        credit.next_payout += datetime.timedelta(days=30 * 12
* diff_count)

            try:
                with transaction.atomic():
                    account.save()
                    credit.save()
            except IntegrityError:
                raise IntegrityError()
        else:
            raise ValidationError("Not valid sum of payment")

class ExchangeRateAPI:
    api_url = "https://api.nbrb.by/exrates/rates?periodicity=0"
    coef = decimal.Decimal(1.05)

    def get_today_rates(self):
        try:
            usd = 0.
            eur = 0.
            content = json.loads(requests.get(self.api_url).content)
            for i in content:
                if i["Cur_Abbreviation"] == "USD":

```

```

        usd = i["Cur_OfficialRate"]
        elif i["Cur_Abbreviation"] == "EUR":
            eur = i["Cur_OfficialRate"]
        if usd == 0.0 or eur == 0.0:
            raise NotFound("Can't get rate")
        return {"USD": decimal.Decimal(usd), "EUR": decimal.Decimal(eur)}
    except Exception:
        raise NotFound("Can't get rate")

    def calculate_amount(self, currency_sell, currency_buy, amount):
        if currency_sell == currency_buy:
            return amount
        elif currency_sell == "BYN":
            return round(amount / (self.get_today_rates()[currency_buy] *
self.coef), 2)
        else:
            if currency_buy == "BYN":
                return round(amount * (self.get_today_rates()[cur-
rency_sell]), 2)
            else:
                in_byn = round(amount * (self.get_today_rates()[cur-
rency_sell]), 2)
                return self.calculate_amount("BYN", currency_buy, in_byn)

G G, [11.12.2023 13:45]
class AccountTransferView(View):
    service = AccountService()

    def get_account(self, request, pk, context):
        account = self.service.retrieve_account_by_pk(pk=pk)
        if not account:
            raise NotFound("Account does not exist")
        if account.owner != request.user:
            raise AuthException()
        context["account"] = self.service.get_account_context(account)

    @logged_in
    def post(self, request, pk):
        form = AccountTransferForm(request.user, request.POST)
        context = {"account_transfer_form": form, "tokenAddress":
CONTRACT_ADDRESS,}

        if request.POST.get("destination_account", None) and re-
quest.POST.get("own_accounts", None) != '--':
            form.add_error("destination_account", "Only one destination field
must be chosen")
            self.get_account(request, pk, context)
            return render(request, template_name="account/account_de-
tail.html", context=context)

        if not request.POST.get("destination_account", None) and re-
quest.POST.get("own_accounts", None) == '--':
            form.add_error("destination_account", "Destination must be cho-
sen")
            self.get_account(request, pk, context)
            return render(request, template_name="account/account_de-
tail.html", context=context)

        if form.is_valid():
            amount = form.cleaned_data["amount"]
            amount = decimal.Decimal(amount)
            destination = form.cleaned_data["destination_account"]

```

```

        source = pk
        own_account = form.cleaned_data["own_accounts"]
        account = self.service.retrieve_account_by_pk(pk=pk)
        if not account:
            raise NotFound("Account does not exist")
        if account.owner != request.user:
            raise AuthException()
        try:
            if own_account != "--":
                self.service.execute_account_transaction(str(source),
str(own_account), amount)
            elif destination is not None:
                self.service.execute_account_transaction(str(source),
str(destination), amount)
            else:
                account = self.service.retrieve_account_by_pk(pk=pk)
                context["account"] = self.service.get_account_context(ac-
count)
                return render(request, template_name="account/account_de-
tail.html", context=context)
        except CustomValueError as e:
            form.add_error("destination_account", e.message)
            self.get_account(request, pk, context)
            return render(request, template_name="account/account_de-
tail.html", context=context)

        else:
            self.get_account(request, pk, context)
            return render(request, template_name="account/account_de-
tail.html", context=context)

        return redirect("account_list")

class AccountTokenView(View):
    service = AccountService()

    def get_account(self, request, pk, context):
        account = self.service.retrieve_account_by_pk(pk=pk)
        if not account:
            raise NotFound("Account does not exist")
        if account.owner != request.user:
            raise AuthException()
        context["account"] = self.service.get_account_context(account)

    @logged_in
    def post(self, request, pk):
        parsed_data = parse_qs(request.body.decode("utf-8"))
        data = {key: value[0] if len(value) == 1 else value for key, value in
parsed_data.items()}
        context = {"account_transfer_form": AccountTransferForm(re-
quest.user), "tokenAddress": CONTRACT_ADDRESS}
        try:
            data["amount"] = decimal.Decimal(data["amount"])
        except KeyError:
            context["token"] = True
            context["content"] = "Invalid amount"
            self.get_account(request, pk, context)
            return render(request, template_name="account/account_de-
tail.html", context=context)

```

G G, [11.12.2023 13:45]



```

if validate_decimal_value(data["amount"]):
    try:
        if len(data["bc_account"]) == 42:
            amount = data["amount"]
            bc_account = Web3.to_checksum_address(data["bc_account"])
            account = self.service.retrieve_account_by_pk(pk=pk)
            if not account:
                raise NotFound("Account does not exist")
            if account.owner != request.user:
                raise AuthException()
            else:
                self.service.exchange_for_token(account, amount,
bc_account)
                context["account"] = self.service.get_account_con-
text(account)

                return render(request, template_name="account/ac-
count_detail.html", context=context)
            else:
                self.get_account(request, pk, context)
                context["token"] = True
                context["content"] = "Invalid blockchain account"
                return render(request, template_name="account/account_de-
tail.html", context=context)
        except KeyError:
            self.get_account(request, pk, context)
            context["token"] = True
            context["content"] = "Invalid blockchain account"
        except ValueError:
            self.get_account(request, pk, context)
            context["content"] = "Insufficient funds, check account
amount"

            context["token"] = True
        except ConnectionError:
            self.get_account(request, pk, context)
            context["content"] = "Problem connecting to network, try
again later"

            context["token"] = True
        except Exception as e:
            self.get_account(request, pk, context)
            context["content"] = "Something went wrong"
            context["token"] = True
        finally:
            return render(request, template_name="account/account_de-
tail.html", context=context)
    else:
        self.get_account(request, pk, context)
        context["token"] = True
        context["content"] = "Invalid amount"
        return render(request, template_name="account/account_de-
tail.html", context=context)

```