СОДЕРЖАНИЕ

1 ТЕХНОЛОГИЧЕСКИЙ РАЗДЕЛ

1.1 Определения стека технологий

В современной сфере разработки программного обеспечения выбор стека технологий играет ключевую роль в успехе проекта. Стек технологий включает в себя языки программирования, фреймворки, библиотеки и среды разработки, которые совместно обеспечивают качественное создание, тестирование и развертывание приложений. При выборе стека необходимо учитывать множество факторов, таких как целевая платформа, производительность, безопасность, масштабируемость и удобство разработки.

В данном разделе приводится обоснование выбора конкретных технологий и инструментов для разработки программного комплекса организации и выдачи верифицированных цифровых дипломов.

Интегрированная среда разработки (IDE) [**?**] или редактор кода являются важнейшими инструментами для программиста. Выбор подходящей платформы может значительно повысить продуктивность и качество кода. Были рассмотрены три популярных решения: WebStorm, Visual Studio Code и NeoVim.

WebStorm [**?**], разработанный JetBrains, является мощной IDE для JavaScript, TypeScript и других смежных технологий. Она предлагает множество встроенных функций, таких как автодополнение кода, отладка, интеграция с системами контроля версий, удаленную разработку и многое другое. WebStorm обеспечивает отличную поддержку для современных фреймворков и библиотек, включая разработку умных контрактов на Solidity. Несмотря на все свои преимущества, WebStorm является ресурсоемким и платным, что делает его менее подходящим для использования.

Visual Studio Code (VS Code) [**?**], созданный Microsoft, является одним из самых популярных редакторов кода благодаря своей легковесности и расширяемости. VS Code поддерживает огромное количество плагинов, которые могут добавить функциональность, схожую с полноценной IDE. Он

обеспечивает хорошую производительность и поддерживает большинство современных языков программирования и фреймворков, однако, как и WebStorm, VS Code собирает телеметрию и требует значительные системные ресурсы при использовании множества расширений и инструментов.

NeoVim [**?**] является усовершенствованной версией классического Vim, предлагая более современные возможности и улучшенную расширяемость. Основное преимущество NeoVim заключается в его легковесности, в том числе интерфейса, и возможности создания полностью настраиваемой среды разработки. С помощью различных плагинов и конфигураций, разработчик может собрать идеальную среду для своих нужд, при этом сохраняя высокую производительность даже на слабых компьютерах. NeoVim обеспечивает поддержку автодополнения кода, синтаксического анализа, отладки и интеграции с системами контроля версий, что делает его отличным выбором для разработки программного продукта.

После определния среды для разработки необходимо выделить наиболее подходящии языки программирования и фреймворки для реализации программного комплекса. Требуется выбрать язык для написания умного контракта, программного интерфейса приложения и клиентских приложений для администратора и пользователей системы.

Для разработки сервера и API был выбран Python с библиотекой FastAPI [**?**]. Python известен своей простотой и читаемостью, что делает его идеальным для быстрой разработки и прототипирования. FastAPI, в свою очередь, предоставляет высокопроизводительный фреймворк для создания API, который поддерживает асинхронное программирование и автоматически генерирует документацию. Основные преимущества заключаются в следующем:

— Простота и читаемость кода.
— Быстрая разработка и развертывание.
— Поддержка асинхронного программирования.
— Автоматическая генерация документации API.

В случае с умным контрактом был выбран язык Solidity, который идеально подходит к задаче благодаря своей специализации на EVM (Ethereum Virtual Machine) [**?**]. Этот язык программирования был специально разработан для написания смарт-контрактов на блокчейне Ethereum и других совместимых с ним блокчейнов. Solidity обладает богатым набором функций, позволяющих эффективно работать с контрактами, и имеет широкую поддержку в сообществе разработчиков блокчейн. Его синтаксис основан на JavaScript, что сильно облегчает изучение и разработку. Выбор был сделан на основе нижеприведенных факторов:

— Оптимизирован для EVM.
— Широкая поддержка блокчейнов.
— Большое сообщество и множество доступных ресурсов.
— Хорошая интеграция с инструментами разработки и тестирования.

Для разработки клиентской части приложения был выбран TypeScript в сочетании с React [**?**]. TypeScript является строго типизированным надмножеством JavaScript, что позволяет избежать ряда ошибок на этапе компиляции и улучшает качество кода. React, как один из самых популярных фреймворков для создания пользовательских интерфейсов, обеспечивает высокую производительность и удобство разработки благодаря своей компонентной архитектуре.

Преимущества TypeScript:

— Статическая типизация снижает количество ошибок.
— Улучшенная поддержка IDE и автодополнение кода.
— Совместимость с JavaScript.

Преимущества React:

— Компонентная архитектура облегчает повторное использование кода.
— Высокая производительность и эффективность.
— Широкое сообщество и богатый экосистемой библиотек и инструментов.

1.2 Разработка компонентов системы

При разработке необходимо обращать внимание не только на стек технологий, но и на паттерны проектирования. Сейчас существует множество шаблонов проектирования и подходов, которые помогают создавать эффективные, поддерживаемые и масштабируемые приложения. В этом разделе были рассмотрены некоторые из популярных паттернов.

1.2.1 Шаблоны проектирования

Шаблоны проектирования представляют собой проверенные решения для типичных задач, с которыми сталкиваются разработчики. Они помогают структурировать код и улучшить его качество. В данном проекте были использованы следующие шаблоны проектирования:

1) *Фабричный метод (Factory Method).*
    а) Используется для создания объектов без необходимости указывать точный класс создаваемого объекта.
    б) Применяется в системе для создания различных типов токенов или объектов данных.
2) *Одиночка (Singleton).*
    а) Обеспечивает наличие единственного экземпляра класса и предоставляет глобальную точку доступа к нему.
    б) Используется для управления подключением к базе данных или другим сервисам, которые должны быть доступны из разных частей приложения.
3) *Декоратор (Decorator).*
    а) Позволяет динамически добавлять новую функциональность объектам.
    б) Применяется для расширения возможностей базовых компонентов интерфейса без изменения их кода.

1.2.2 Подходы к разработке

Разработка программного обеспечения включает различные подходы, которые помогают организовать процесс и структуру кода. В этом проекте были использованы следующие подходы:

1) *DDD (Domain-Driven Design).*
    а) DDD фокусируется на модели предметной области и использовании языка, понятного всем участникам проекта.
    б) В проекте используется DDD для создания модели данных и логики бизнес-процессов, что обеспечивает лучшее понимание требований и упрощает поддержание кода.
2) *TDD (Test-Driven Development).*
    а) TDD предполагает написание тестов перед реализацией функциональности.
    б) Этот подход помогает обеспечить высокое качество кода и уменьшить количество ошибок, возникающих в процессе разработки.

1.2.3 Архитектура программного обеспечения

Архитектура программного обеспечения описывает структуру системы, взаимодействие между компонентами и потоки данных. В этом разделе были рассмотрены архитектуры фронтенда и бэкенда.

Фронтенд системы был реализован с использованием React и TypeScript. Основные элементы архитектуры включают:

1) *Компоненты.*
    а) Приложение состоит из отдельных компонентов, каждый из которых отвечает за свою часть интерфейса.
    б) Компоненты могут быть повторно использованы или работать независимо.
2) *Состояния.*
    а) Для управления состоянием приложения используется Context

API или Redux.

б) Это обеспечивает централизованное управление состоянием и упрощает передачу данных между компонентами.

3) *Роутеры (маршрутизаторы).*

а) Используется для управления навигацией внутри приложения.

б) Позволяет создавать одностраничные приложения (SPA) с плавным переходом между страницами.

Код приложения для администратора приведен в приложении **??**, а клиентская часть для пользователей в приложении **??**.

Бэкенд системы реализован с использованием Python и FastAPI. Основные компоненты архитектуры включают:

1) *REST API.*

а) FastAPI используется для создания RESTful API, который обеспечивает взаимодействие с фронтендом.

б) API обеспечивает доступ к функциональности системы, такой как создание и управление токенами, получение данных и выполнение бизнес-логики.

2) *Слои приложения.*

а) Контроллеры. Обрабатывает входящие HTTP-запросы, вызывает соответствующие сервисы и возвращает ответы.

б) Сервисы. Содержит бизнес-логику приложения. Взаимодействует с репозиториями и внешними сервисами

в) Репозитории. Отвечает за доступ к данным, обеспечивает взаимодействие с базой данных и другими хранилищами данных.

3) *Web3 обработчик.*

а) Используется подключение к RPC-узлу для взаимодействия с блокчейн-сетью.

б) Реализованы функции для создания, одобрения и управления смарт-контрактами на основе Solidity.

1.3 Графический пользовательский интерфейс

В данном разделе были рассмотрены два основных пользовательских интерфейсов программного комплекса: веб-приложение для создания коллекции дипломов администратором и телеграм-бот для просмотра и получения дипломов пользователями. Оба компонента обеспечивают удобство использования и доступ к функциональности системы.

1.3.1 Веб-приложение для создания коллекции дипломов

В первую очередь было реализовано веб-приложение для управления. Оно позволяет администратору загружать изображения дипломов, вводить метаданные и прочие параметры коллекции. Реализация состоит из основной страницы с несколькими компонентами, обеспечивающими удобный интерфейс.

Основные функции веб-приложения:

— Загрузка изображений дипломов.
— Ввод метаданных в формате JSON.
— Указание названия, описания и автора коллекции.
— Настройка количества требуемых подписей для создания NFT.

Авторизация реализована при помощи модуля обратного прокси Caddy [**?**] и требует токен для входа. После авторизации, на главной странице, администратор может загрузить изображения или архив изображений с дипломами, внести метаданные и указать дополнительные параметры для новой коллекции дипломов. Пример интерфейса приведен на изображении 9, а код в листинге **??** из приложения **??**.

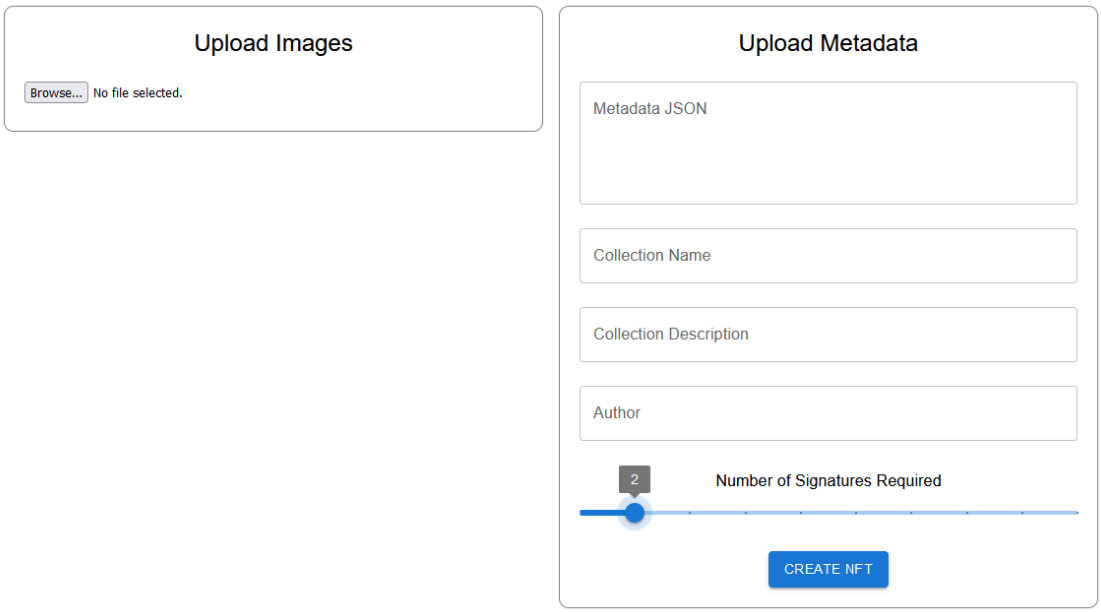

Рисунок 1 – Интерфейс администратора для создания коллекции дипломов

1.3.2 Телеграм-бот для просмотра и получения дипломов

Телеграм-бот был разработан с использованием библиотеки node-telegram-bot-api, код которого приведен в листинге **??** из приложения **??**. Он позволяет пользователям просматривать и получать свои дипломы в формате NFT, обеспечивая удобный интерфейс для взаимодействия с системой через мессенджер. Пример интерфейса приведен на рисунке 10, где изображено оба окна.

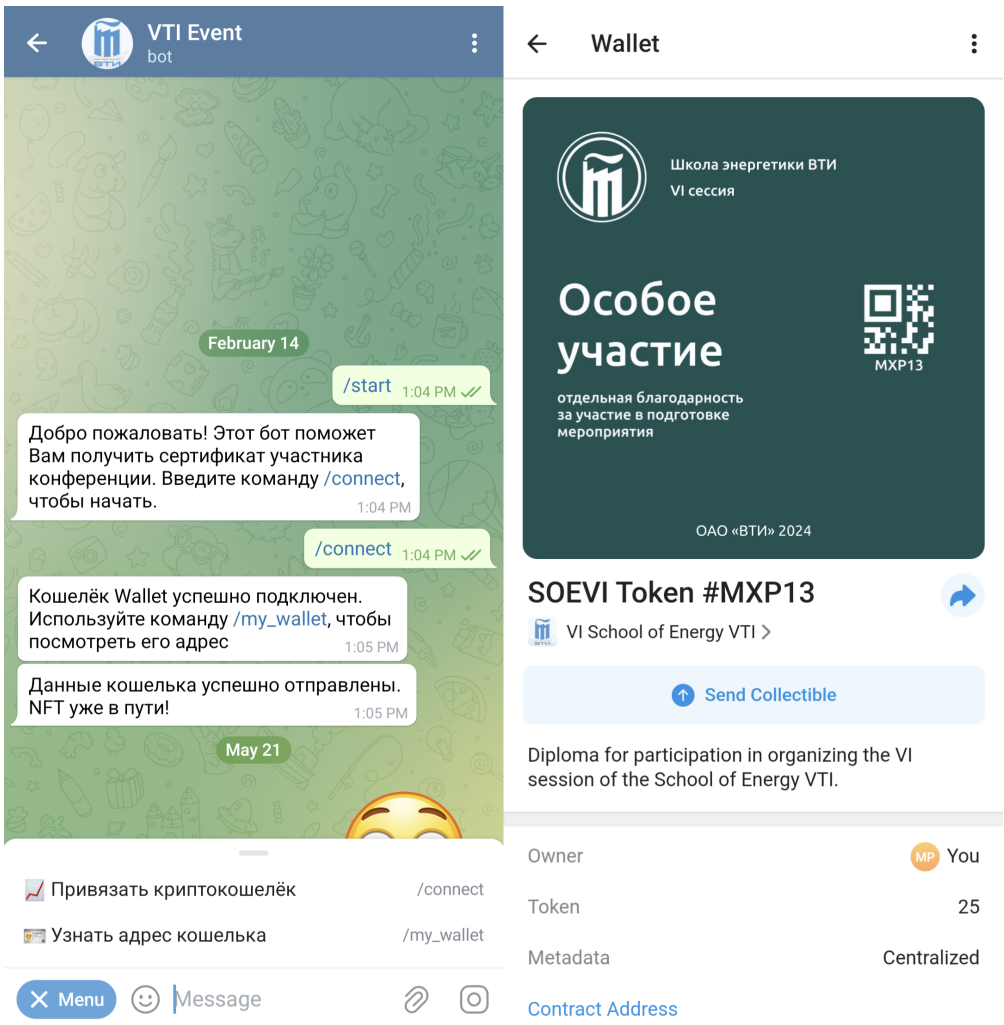

Рисунок 2 – Пользовательский бот для получения и просмотра дипломов

Основные функции телеграм-бота:

— Просмотр списка доступных дипломов.
— Получение информации о конкретном дипломе.
— Получение ссылки на NFT-диплом.

Пользователи могут взаимодействовать с ботом через простые команды, такие как /start, /connect и т.д. Бот обрабатывает команды, запрашивает данные у API и возвращает пользователям соответствующую информацию.

1.3.3 Просмотр дипломов на других платформах

Реализация умного контракта, приведенного в листинге **??** из приложения **??**, для «ПК Дипломер» предоставляет значительные преимущества, одной из которых является возможность просмотра данных дипломов на

любых платформах, которые умеют взаимодействовать с соответствующей блокчейн-сетью. Это достигается благодаря использованию стандарта ERC-721 [**?**], широко признанного и поддерживаемого многими блокчейн-платформами и приложениями.

ERC-721 является стандартом для невзаимозаменяемых токенов (NFT) в блокчейне Ethereum и совместимых с EVM (Ethereum Virtual Machine) блокчейнах. Он определяет набор правил бинарного интерфейса приложения (ABI) [**?**], которые должны быть реализованы смарт-контрактами. Это обеспечивает их совместимость с различными децентрализованными приложениями (dApps) [**?**] и платформами.

Пример прсмотра дипломов на сторонней платформе NFTScan приведен на рисунке 11.

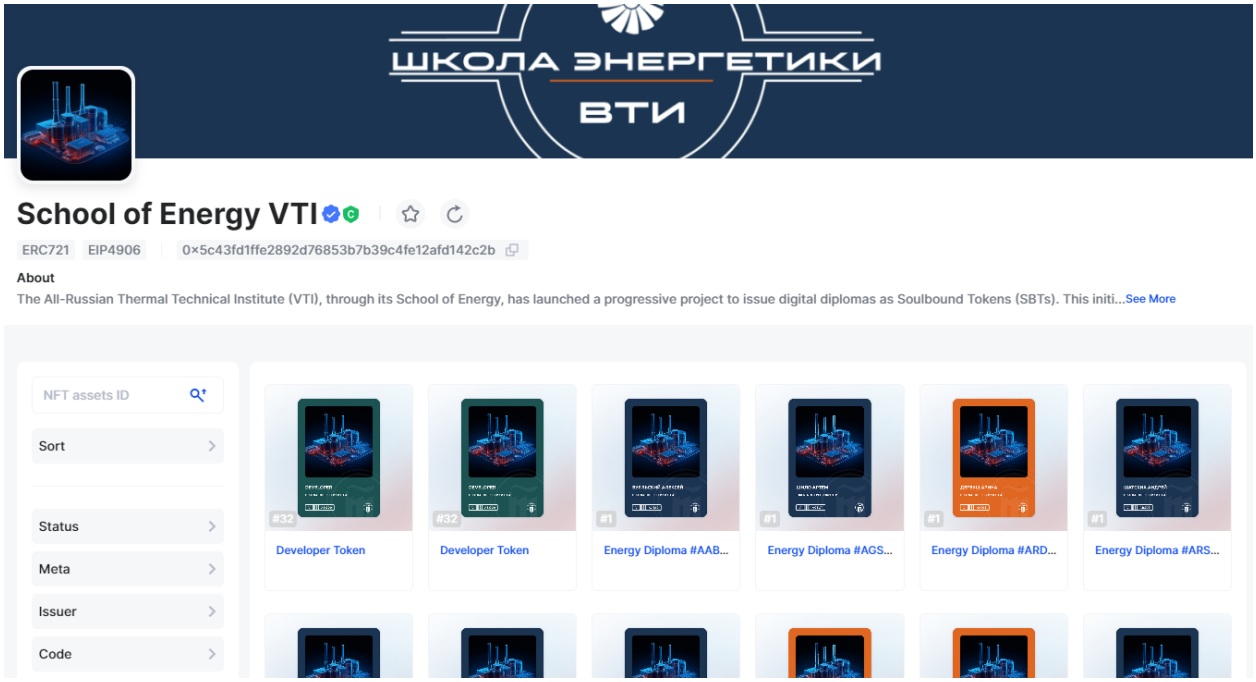

Рисунок 3 – Просмотр дипломов на сторонней платформе

Стоит отметить, что внешние платформы не могут отобразить всю информацию из контрактов, вроде нескольких подписавших, но на отображение самого диплома это не влияет.

1.4 Серверная часть приложения

Основная часть системы состоит из двух компонентов: минтера (minter) и API. Эти компоненты обеспечивают взаимодействие клиентских приложений с блокчейном и позволяют пользователям управлять своими дипломами в формате NFT.

1.4.1 Система выпуска дипломов

Минтер (minter) является основным компонентом системы, отвечающим за создание новых NFT-документов. Он реализует функциональность выпуска дипломов и взаимодействия с умным контрактом на блокчейне, обеспечивая выполнение всех операций, связанных с созданием и управлением токенами.

Сама функция выпуска диплома лаконичная и отвечает лишь за передачу подготовленных ранее данных в блокчейн. В ней указывается комиссия за транзакцию (газ) и дополнительные пользовательские данные. Код функции приведен в листинге 1.

Листинг 1 – Функция выпуска диплома

```
    def mint(to, uri):
        gas_price = w3.eth.gas_price
        nonce = get_nonce()
        tx = contract.functions.proposeMint(to, uri).build_transaction({
            'chainId': chain_id,
            'gas': 200000,
            'gasPrice': gas_price,
            'nonce': nonce,
        })
        signed_tx = account.sign_transaction(tx)
        tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
        receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
        return receipt
```

1.4.2 Программный интерфейс прилоения

API обеспечивает интерфейс для взаимодействия клиентских приложений с системой. Оно реализовано с использованием FastAPI и предоставляет RESTful сервисы для выполнения различных операций, таких как создание, одобрение и сжигание токенов, а также получение информации о пользователях и токенах.

— Создание новых токенов. API предоставляет конечные точки для создания новых токенов, что позволяет клиентским приложениям инициировать процесс создания дипломов.
— Одобрение транзакций. Конечные точки для одобрения транзакций обеспечивают возможность мультиподписи.
— Получение информации. API позволяет получать информацию о пользователях и токенах, обеспечивая необходимую функциональность для клиентских приложений.
— Сжигание токенов. Конечные точки для сжигания токенов позволяют управлять жизненным циклом дипломов.

Все существующие в системе маршруты до конечных точек (эндпоинтов) представлены в таблице 1.

Таблица 1 – Маршруты API

Маршрут	**Метод HTTP**	**Описание**
/mint	POST	Создание нового NFT-документа
/approve_mint	POST	Одобрение создания нового NFT-документа
/burn	POST	Сжигание существующего NFT-документа
/balance	GET	Получение баланса (количество NFT) для указанного адреса
/total_supply	GET	Получение общего количества выпущенных NFT
/roles/add_minter	POST	Назначение роли MINTER для указанного адреса
/roles/remove_minter	POST	Удаление роли MINTER у указанного адреса
/roles/add_multisig	POST	Назначение роли MULTISIG для указанного
/roles/remove_multisig	POST	Назначение роли MULTISIG для указанного адреса

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое интегрированная среда разработки (IDE)? // Amazon Web Services URL: https://aws.amazon.com/ru/what-is/ide (дата обращения: 19.05.2024).

2. WebStorm IDE // JetBrains URL: https://www.jetbrains.com/ru-ru/webstorm (дата обращения: 19.05.2024).

3. Visual Studio Code is a distribution of the Code // GitHub URL: https://github.com/microsoft/vscode (дата обращения: 19.05.2024).

4. Neovim — текстовый редактор с TUI интерфейсом // ALT Gnome URL: https://alt-gnome.wiki/neovim.html (дата обращения: 19.05.2024).

5. FastAPI is a modern, fast (high-performance), web framework for building APIs // Tiangolo URL: https://fastapi.tiangolo.com (дата обращения: 19.05.2024).

6. Ethereum Virtual Machine (EVM) // Ethereum Org URL: https://ethereum.org/en/developers/docs/evm (дата обращения: 19.05.2024).

7. The library for web and native user interfaces // React Dev URL: https://react.dev (дата обращения: 19.05.2024).

8. API or ABI changing // Caddy URL: https://caddyserver.com (дата обращения: 19.05.2024).

9. ERC-721: Non-Fungible Token Standard // Ethereum Org URL: https://eips.ethereum.org/EIPS/eip-721 (дата обращения: 19.05.2024).

10. API or ABI changing // ALT Linux Wiki URL: https://www.altlinux.org/API_or_ABI_changing (дата обращения: 19.05.2024).

11. Decentralized Applications (dApps) // Investopedia URL: https://www.investopedia.com/terms/d/decentralized-applications-dapps.asp (дата обращения: 19.05.2024).

ПРИЛОЖЕНИЕ А
Код программного комплекса «Дипломер»

А.1 Умный контракт для блокчейна на EVM

Листинг 2 – Смарт-контракт

```
1   pragma solidity ^0.8.7;
2
3   import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
4   import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.
        sol";
5   import "@openzeppelin/contracts/utils/Counters.sol";
6   import "@openzeppelin/contracts/access/AccessControl.sol";
7
8   /// @custom:security-contact mpanfilov@vti.ru
9   contract SoulBoundToken is ERC721URIStorage, AccessControl {
10
11      using Counters for Counters.Counter;
12
13      bytes32 public constant MINTER_ROLE = keccak256("MINTER_ROLE");
14      bytes32 public constant MULTISIG_ROLE = keccak256("MULTISIG_ROLE");
15      Counters.Counter private _tokenIdCounter;
16
17      // For multisig approvals
18      mapping(uint256 => address[]) private _mintApprovals;
19      uint256 public constant MIN_APPROVALS = 2; // Minimum number of
            approvals required
20
21      constructor() ERC721("SoulBoundToken", "SBT") {
22          _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
23          _grantRole(MINTER_ROLE, msg.sender);
24          _grantRole(MINTER_ROLE, 0x0000000000000000000000000000000000000000
                );
25      }
26
27      // @dev Function to mint tokens with multisig approval. Accepts an
            address and a token URI.
28      function proposeMint(address to, string memory uri) public onlyRole(
            MINTER_ROLE) {
29          uint256 tokenId = _tokenIdCounter.current();
30          _mintApprovals[tokenId].push(msg.sender);
```

```
31              _safeMint(to, tokenId);
32              _setTokenURI(tokenId, uri);
33          }
34
35          function approveMint(uint256 tokenId) public onlyRole(MULTISIG_ROLE) {
36              require(_exists(tokenId), "Token does not exist.");
37              require(!_isApproved(tokenId, msg.sender), "Already approved.");
38
39              _mintApprovals[tokenId].push(msg.sender);
40
41              if (_mintApprovals[tokenId].length >= MIN_APPROVALS) {
42                  _finalizeMint(tokenId);
43              }
44          }
45
46          function _isApproved(uint256 tokenId, address approver) internal view
                returns (bool) {
47              address[] memory approvals = _mintApprovals[tokenId];
48              for (uint i = 0; i < approvals.length; i++) {
49                  if (approvals[i] == approver) {
50                      return true;
51                  }
52              }
53              return false;
54          }
55
56          function _finalizeMint(uint256 tokenId) internal {
57              _tokenIdCounter.increment();
58              delete _mintApprovals[tokenId];
59          }
60
61          function burn(uint256 tokenId) external {
62              require(
63                  ownerOf(tokenId) == msg.sender || hasRole(DEFAULT_ADMIN_ROLE,
                        msg.sender),
64                  "Only owner or contract admin can burn it."
65              );
66              _burn(tokenId);
67          }
68
69          function totalSupply() public view returns (uint256) {
70              return _tokenIdCounter.current();
```

```
71          }
72
73          function _beforeTokenTransfer(address from, address to, uint256
                tokenId, uint256 batchSize) pure internal override(ERC721) {
74              require(from == address(0) || to == address(0), "SBD token can't
                    be transferred.");
75          }
76
77          function _burn(uint256 tokenId) internal override(ERC721URIStorage) {
78              super._burn(tokenId);
79          }
80
81          function supportsInterface(bytes4 interfaceId) public view virtual
                override(ERC721URIStorage, AccessControl) returns (bool) {
82              return (ERC721.supportsInterface(interfaceId) || AccessControl.
                    supportsInterface(interfaceId));
83          }
84
85          // Role management functions
86          function addMinter(address account) public onlyRole(DEFAULT_ADMIN_ROLE
                ) {
87              grantRole(MINTER_ROLE, account);
88          }
89
90          function removeMinter(address account) public onlyRole(
                DEFAULT_ADMIN_ROLE) {
91              revokeRole(MINTER_ROLE, account);
92          }
93
94          function addMultisig(address account) public onlyRole(
                DEFAULT_ADMIN_ROLE) {
95              grantRole(MULTISIG_ROLE, account);
96          }
97
98          function removeMultisig(address account) public onlyRole(
                DEFAULT_ADMIN_ROLE) {
99              revokeRole(MULTISIG_ROLE, account);
100         }
101     }
```

Листинг 3 – Тесты умного контракта

```
1   from web3 import Web3
```

```
2       from contract_data import abi
3       from dotenv import dotenv_values
4
5       config = dotenv_values(".env")
6
7       w3 = Web3(Web3.HTTPProvider(config["WEB3_PROVIDER"]))
8       chain_id = int(config["CHAIN_ID"])
9       contract_address = config["CONTRACT_ADDRESS"]
10      private_key = config["PRIVATE_KEY"]
11
12      contract = w3.eth.contract(address=contract_address, abi=abi)
13      account = w3.eth.account.from_key(private_key)
14
15      def get_nonce():
16          return w3.eth.get_transaction_count(account.address)
17
18      def mint(to, uri):
19          gas_price = w3.eth.gas_price
20          nonce = get_nonce()
21          tx = contract.functions.proposeMint(to, uri).build_transaction({
22              'chainId': chain_id,
23              'gas': 200000,
24              'gasPrice': gas_price,
25              'nonce': nonce,
26          })
27          signed_tx = account.sign_transaction(tx)
28          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
29          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
30          return receipt
31
32      def approve_mint(token_id):
33          gas_price = w3.eth.gas_price
34          nonce = get_nonce()
35          tx = contract.functions.approveMint(token_id).build_transaction({
36              'chainId': chain_id,
37              'gas': 100000,
38              'gasPrice': gas_price,
39              'nonce': nonce,
40          })
41          signed_tx = account.sign_transaction(tx)
42          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
43          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
```

```
44          return receipt
45
46      def balance(address: str = account.address):
47          return contract.functions.balanceOf(address).call()
48
49      def burn(token_id: int):
50          gas_price = w3.eth.gas_price
51          nonce = get_nonce()
52          tx = contract.functions.burn(token_id).build_transaction({
53              'chainId': chain_id,
54              'gas': 100000,
55              'gasPrice': gas_price,
56              'nonce': nonce,
57          })
58          signed_tx = account.sign_transaction(tx)
59          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
60          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
61          return receipt
62
63      def transfer(to, token_id: int):
64          gas_price = w3.eth.gas_price
65          nonce = get_nonce()
66          tx = contract.functions.transferFrom(account.address, w3.
                to_checksum_address(to), token_id).build_transaction({
67              'chainId': chain_id,
68              'gas': 100000,
69              'gasPrice': gas_price,
70              'nonce': nonce,
71          })
72          signed_tx = account.sign_transaction(tx)
73          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
74          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
75          return receipt
76
77      # Example usage:
78      if __name__ == "__main__":
79          to_address = "0xRecipientAddressHere"
80          token_uri = "ipfs://your-token-uri"
81
82          # Propose mint
83          mint_receipt = mint(to_address, token_uri)
84          print(f"Mint Proposal Transaction Receipt: {mint_receipt}")
```

```
85
86          # Approve mint (you'll need to do this from multiple accounts with
                MULTISIG_ROLE)
87          token_id = 0  # Assuming the first token ID
88          approve_receipt = approve_mint(token_id)
89          print(f"Mint Approval Transaction Receipt: {approve_receipt}")
90
91          # Check balance
92          user_balance = balance(to_address)
93          print(f"User Balance: {user_balance}")
94
95          # Burn token
96          burn_receipt = burn(token_id)
97          print(f"Burn Transaction Receipt: {burn_receipt}")
```

A.2 Модуль выпуска дипломов

Листинг 4 – Функции выпуска дипломов

```
1   from web3 import Web3
2   from contract_data import abi
3   from dotenv import dotenv_values
4
5   config = dotenv_values(".env")
6
7   w3 = Web3(Web3.HTTPProvider(config["WEB3_PROVIDER"]))
8   chain_id = int(config["CHAIN_ID"])
9   contract_address = config["CONTRACT_ADDRESS"]
10  private_key = config["PRIVATE_KEY"]
11
12  contract = w3.eth.contract(address=contract_address, abi=abi)
13  account = w3.eth.account.from_key(private_key)
14
15  def get_nonce():
16      return w3.eth.get_transaction_count(account.address)
17
18  def propose_mint(to, uri):
19      gas_price = w3.eth.gas_price
20      nonce = get_nonce()
21      tx = contract.functions.proposeMint(to, uri).build_transaction({
22          'chainId': chain_id,
23          'gas': 200000,
```

```
24              'gasPrice': gas_price,
25              'nonce': nonce,
26          })
27          signed_tx = account.sign_transaction(tx)
28          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
29          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
30          return receipt
31
32      def approve_mint(token_id):
33          gas_price = w3.eth.gas_price
34          nonce = get_nonce()
35          tx = contract.functions.approveMint(token_id).build_transaction({
36              'chainId': chain_id,
37              'gas': 100000,
38              'gasPrice': gas_price,
39              'nonce': nonce,
40          })
41          signed_tx = account.sign_transaction(tx)
42          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
43          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
44          return receipt
45
46      def balance(address: str = account.address):
47          return contract.functions.balanceOf(address).call()
48
49      def burn(token_id: int):
50          gas_price = w3.eth.gas_price
51          nonce = get_nonce()
52          tx = contract.functions.burn(token_id).build_transaction({
53              'chainId': chain_id,
54              'gas': 100000,
55              'gasPrice': gas_price,
56              'nonce': nonce,
57          })
58          signed_tx = account.sign_transaction(tx)
59          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
60          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
61          return receipt
62
63      def total_supply():
64          return contract.functions.totalSupply().call()
65
```

```
66      def add_minter(account_address):
67          gas_price = w3.eth.gas_price
68          nonce = get_nonce()
69          tx = contract.functions.addMinter(account_address).build_transaction({
70              'chainId': chain_id,
71              'gas': 100000,
72              'gasPrice': gas_price,
73              'nonce': nonce,
74          })
75          signed_tx = account.sign_transaction(tx)
76          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
77          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
78          return receipt
79
80      def remove_minter(account_address):
81          gas_price = w3.eth.gas_price
82          nonce = get_nonce()
83          tx = contract.functions.removeMinter(account_address).
                build_transaction({
84              'chainId': chain_id,
85              'gas': 100000,
86              'gasPrice': gas_price,
87              'nonce': nonce,
88          })
89          signed_tx = account.sign_transaction(tx)
90          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
91          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
92          return receipt
93
94      def add_multisig(account_address):
95          gas_price = w3.eth.gas_price
96          nonce = get_nonce()
97          tx = contract.functions.addMultisig(account_address).build_transaction
                ({
98              'chainId': chain_id,
99              'gas': 100000,
100             'gasPrice': gas_price,
101             'nonce': nonce,
102         })
103         signed_tx = account.sign_transaction(tx)
104         tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
105         receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
```

```
106         return receipt
107
108     def remove_multisig(account_address):
109         gas_price = w3.eth.gas_price
110         nonce = get_nonce()
111         tx = contract.functions.removeMultisig(account_address).
                build_transaction({
112             'chainId': chain_id,
113             'gas': 100000,
114             'gasPrice': gas_price,
115             'nonce': nonce,
116         })
117         signed_tx = account.sign_transaction(tx)
118         tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
119         receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
120         return receipt
```

Листинг 5 – Интерфейс для консольного запуска (__main__.py)

```
1   from minter import propose_mint, approve_mint, balance, burn, transfer,
        total_supply, add_minter, remove_minter, add_multisig, remove_multisig
2
3   if __name__ == "__main__":
4       to_address = "0xRecipientAddressHere"
5       token_uri = "ipfs://token-uri"
6
7       # Propose mint
8       mint_receipt = propose_mint(to_address, token_uri)
9       print(f"Mint Proposal Transaction Receipt: {mint_receipt}")
10
11      # Approve mint (you'll need to do this from multiple accounts with
            MULTISIG_ROLE)
12      token_id = 0  # Assuming the first token ID
13      approve_receipt = approve_mint(token_id)
14      print(f"Mint Approval Transaction Receipt: {approve_receipt}")
15
16      # Check balance
17      user_balance = balance(to_address)
18      print(f"User Balance: {user_balance}")
19
20      # Burn token
21      burn_receipt = burn(token_id)
22      print(f"Burn Transaction Receipt: {burn_receipt}")
```

```
23
24          # Check total supply
25          supply = total_supply()
26          print(f"Total Supply: {supply}")
27
28          # Manage roles
29          new_minter = "0xNewMinterAddressHere"
30          add_minter_receipt = add_minter(new_minter)
31          print(f"Add Minter Transaction Receipt: {add_minter_receipt}")
32
33          remove_minter_receipt = remove_minter(new_minter)
34          print(f"Remove Minter Transaction Receipt: {remove_minter_receipt}")
35
36          new_multisig = "0xNewMultisigAddressHere"
37          add_multisig_receipt = add_multisig(new_multisig)
38          print(f"Add Multisig Transaction Receipt: {add_multisig_receipt}")
39
40          remove_multisig_receipt = remove_multisig(new_multisig)
41          print(f"Remove Multisig Transaction Receipt: {remove_multisig_receipt}
                ")
```

A.3 Программный интерфейс приложения (API)

Листинг 6 – Ядро API

```
1       from fastapi import FastAPI, HTTPException
2       from mint import propose_mint, approve_mint
3       from burn import burn_token
4       from read import get_balance, get_total_supply
5       from roles import add_minter_role, remove_minter_role, add_multisig_role,
            remove_multisig_role
6
7       app = FastAPI()
8
9       @app.post("/mint/")
10      def mint(to_address: str, token_uri: str):
11          receipt = propose_mint(to_address, token_uri)
12          return {"receipt": receipt}
13
14      @app.post("/approve_mint/")
15      def approve_mint(token_id: int):
16          receipt = approve_mint(token_id)
```

```
17          return {"receipt": receipt}
18
19      @app.post("/burn/")
20      def burn(token_id: int):
21          receipt = burn_token(token_id)
22          return {"receipt": receipt}
23
24      @app.get("/balance/{address}")
25      def balance(address: str):
26          balance = get_balance(address)
27          return {"balance": balance}
28
29      @app.get("/total_supply/")
30      def total_supply():
31          supply = get_total_supply()
32          return {"total_supply": supply}
33
34      @app.post("/roles/add_minter/")
35      def add_minter(address: str):
36          receipt = add_minter_role(address)
37          return {"receipt": receipt}
38
39      @app.post("/roles/remove_minter/")
40      def remove_minter(address: str):
41          receipt = remove_minter_role(address)
42          return {"receipt": receipt}
43
44      @app.post("/roles/add_multisig/")
45      def add_multisig(address: str):
46          receipt = add_multisig_role(address)
47          return {"receipt": receipt}
48
49      @app.post("/roles/remove_multisig/")
50      def remove_multisig(address: str):
51          receipt = remove_multisig_role(address)
52          return {"receipt": receipt}
```

Листинг 7 – Упрощенная модель для базы данных

```
1       from sqlalchemy import create_engine, Column, Integer, String
2       from sqlalchemy.ext.declarative import declarative_base
3       from sqlalchemy.orm import sessionmaker
4
```

```
5       DATABASE_URL = "sqlite:///./test.db"
6
7       engine = create_engine(DATABASE_URL)
8       SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine
            )
9       Base = declarative_base()
10
11      class Token(Base):
12          __tablename__ = "tokens"
13
14          id = Column(Integer, primary_key=True, index=True)
15          owner = Column(String, index=True)
16          uri = Column(String, index=True)
```

Листинг 8 – Модуль управления ролями

```
1       from web3 import Web3
2       from contract_data import abi
3       from config import WEB3_PROVIDER, CHAIN_ID, CONTRACT_ADDRESS, PRIVATE_KEY
4
5       w3 = Web3(Web3.HTTPProvider(WEB3_PROVIDER))
6       contract = w3.eth.contract(address=CONTRACT_ADDRESS, abi=abi)
7       account = w3.eth.account.from_key(PRIVATE_KEY)
8
9       def get_nonce():
10          return w3.eth.get_transaction_count(account.address)
11
12      def add_minter_role(account_address):
13          gas_price = w3.eth.gas_price
14          nonce = get_nonce()
15          tx = contract.functions.addMinter(account_address).build_transaction({
16              'chainId': CHAIN_ID,
17              'gas': 100000,
18              'gasPrice': gas_price,
19              'nonce': nonce,
20          })
21          signed_tx = account.sign_transaction(tx)
22          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
23          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
24          return receipt
25
26      def remove_minter_role(account_address):
27          gas_price = w3.eth.gas_price
```

```
28          nonce = get_nonce()
29          tx = contract.functions.removeMinter(account_address).
                build_transaction({
30              'chainId': CHAIN_ID,
31              'gas': 100000,
32              'gasPrice': gas_price,
33              'nonce': nonce,
34          })
35          signed_tx = account.sign_transaction(tx)
36          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
37          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
38          return receipt
39
40      def add_multisig_role(account_address):
41          gas_price = w3.eth.gas_price
42          nonce = get_nonce()
43          tx = contract.functions.addMultisig(account_address).build_transaction
                ({
44              'chainId': CHAIN_ID,
45              'gas': 100000,
46              'gasPrice': gas_price,
47              'nonce': nonce,
48          })
49          signed_tx = account.sign_transaction(tx)
50          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
51          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
52          return receipt
53
54      def remove_multisig_role(account_address):
55          gas_price = w3.eth.gas_price
56          nonce = get_nonce()
57          tx = contract.functions.removeMultisig(account_address).
                build_transaction({
58              'chainId': CHAIN_ID,
59              'gas': 100000,
60              'gasPrice': gas_price,
61              'nonce': nonce,
62          })
63          signed_tx = account.sign_transaction(tx)
64          tx_hash = w3.eth.send_raw_transaction(signed_tx.rawTransaction)
65          receipt = w3.eth.wait_for_transaction_receipt(tx_hash.hex())
66          return receipt
```

A.4 Веб-приложение для создания дипломов

Листинг 9 – Интерфейс администратора для создания коллекции дипломов

```
import React, { useState } from 'react';
import { Button, TextField, Typography, Box, Slider, InputLabel,
    FormControl, Select, MenuItem, Grid } from '@mui/material';
import axios from 'axios';

const NFTForm = () => {
  const [name, setName] = useState('');
  const [description, setDescription] = useState('');
  const [author, setAuthor] = useState('');
  const [signatures, setSignatures] = useState(2);
  const [file, setFile] = useState(null);
  const [fileUrl, setFileUrl] = useState('');
  const [metadata, setMetadata] = useState('');

  const handleFileUpload = async (event) => {
    const file = event.target.files[0];
    setFile(file);

    const formData = new FormData();
    formData.append('file', file);

    try {
      const response = await axios.post('IPFS_UPLOAD_URL', formData, {
        headers: {
          'Content-Type': 'multipart/form-data',
        },
      });

      setFileUrl(response.data.Hash);
    } catch (error) {
      console.error('Error uploading file:', error);
    }
  };

  const handleSubmit = async () => {
    try {
      const response = await axios.post('/api/mint', {
        name,
```

```
            description,
            author,
            signatures,
            fileUrl,
            metadata,
          });

          console.log('Minting successful:', response.data);
        } catch (error) {
          console.error('Error minting NFT:', error);
        }
      };

      return (
        <Grid container spacing={2} sx={{ padding: '20px' }}>
          <Grid item xs={12} md={6}>
            <Box sx={{ textAlign: 'center', border: '1px solid grey',
                borderRadius: '10px', padding: '20px' }}>
              <Typography variant="h5" gutterBottom>Upload Images</Typography>
              <FormControl fullWidth margin="normal">
                <input
                  type="file"
                  id="file-upload"
                  accept=".jpg,.jpeg,.png,.svg,.gif,.webp,.mp4,.zip"
                  onChange={handleFileUpload}
                />
              </FormControl>
            </Box>
          </Grid>
          <Grid item xs={12} md={6}>
            <Box sx={{ textAlign: 'center', border: '1px solid grey',
                borderRadius: '10px', padding: '20px' }}>
              <Typography variant="h5" gutterBottom>Upload Metadata</
                  Typography>
              <TextField
                fullWidth
                label="Metadata JSON"
                variant="outlined"
                margin="normal"
                multiline
                rows={4}
                value={metadata}
```

```
77                  onChange={(e) => setMetadata(e.target.value)}
78                />
79                <TextField
80                  fullWidth
81                  label="Collection Name"
82                  variant="outlined"
83                  margin="normal"
84                  value={name}
85                  onChange={(e) => setName(e.target.value)}
86                />
87                <TextField
88                  fullWidth
89                  label="Collection Description"
90                  variant="outlined"
91                  margin="normal"
92                  value={description}
93                  onChange={(e) => setDescription(e.target.value)}
94                />
95                <TextField
96                  fullWidth
97                  label="Author"
98                  variant="outlined"
99                  margin="normal"
100                 value={author}
101                 onChange={(e) => setAuthor(e.target.value)}
102               />
103               <Box sx={{ width: '100%', marginTop: '20px' }}>
104                 <Typography gutterBottom>Number of Signatures Required</
                      Typography>
105                 <Slider
106                   value={signatures}
107                   onChange={(e, newValue) => setSignatures(newValue)}
108                   aria-labelledby="continuous-slider"
109                   valueLabelDisplay="auto"
110                   step={1}
111                   marks
112                   min={1}
113                   max={10}
114                 />
115               </Box>
116               <Button
117                 variant="contained"
```

```
                color="primary"
                onClick={handleSubmit}
                sx={{ marginTop: '20px' }}
              >
                Create NFT
              </Button>
            </Box>
          </Grid>
        </Grid>
      );
    };

    export default NFTForm;
```

A.5 Телеграм-бот для пользовательского взаимодействия

Листинг 10 – Коннектор для подключения кошелька

```
import solConnect from '@solconnect/sdk';
import { solConnectStorage } from './storage';
import * as process from 'process';

type StoredConnectorData = {
    connector: solConnect;
    timeout: ReturnType<typeof setTimeout>;
    onConnectorExpired: ((connector: solConnect) => void)[];
};

const connectors = new Map<number, StoredConnectorData>();

export function getConnector(
    chatId: number,
    onConnectorExpired?: (connector: solConnect) => void
): solConnect {
    let storedItem: StoredConnectorData;
    if (connectors.has(chatId)) {
        storedItem = connectors.get(chatId)!;
        clearTimeout(storedItem.timeout);
    } else {
        storedItem = {
            connector: new solConnect({
                manifestUrl: process.env.MANIFEST_URL,
```

```
                    storage: new solConnectStorage(chatId)
                }),
                onConnectorExpired: []
            } as unknown as StoredConnectorData;
        }

        if (onConnectorExpired) {
            storedItem.onConnectorExpired.push(onConnectorExpired);
        }

        storedItem.timeout = setTimeout(() => {
            if (connectors.has(chatId)) {
                const storedItem = connectors.get(chatId)!;
                storedItem.connector.pauseConnection();
                storedItem.onConnectorExpired.forEach(callback => callback(
                    storedItem.connector));
                connectors.delete(chatId);
            }
        }, Number(process.env.CONNECTOR_TTL_MS));

        connectors.set(chatId, storedItem);
        return storedItem.connector;
    }
```

Листинг 11 – Ядро бота

```
import dotenv from 'dotenv';
dotenv.config();

import { bot } from './bot';
import { walletMenuCallbacks } from './connect-wallet-menu';
import {
    handleConnectCommand,
    handleDisconnectCommand,
    handleSendWalletCommand,
    handleShowMyWalletCommand
} from './commands-handlers';
import { getStorage, initRedisClient } from './sol-connect/storage';
import TelegramBot from 'node-telegram-bot-api';

async function main(): Promise<void> {
    await initRedisClient();

```

```
18          const callbacks = {
19              ...walletMenuCallbacks
20          };
21
22          bot.on('callback_query', query => {
23              if (!query.data) {
24                  return;
25              }
26
27              let request: { method: string; data: string };
28
29              try {
30                  request = JSON.parse(query.data);
31              } catch {
32                  return;
33              }
34
35              if (!callbacks[request.method as keyof typeof callbacks]) {
36                  return;
37              }
38
39              callbacks[request.method as keyof typeof callbacks](query, request
                    .data);
40          });
41
42          bot.onText(/\/connect/, handleConnectCommand);
43
44          bot.onText(/\/send_wallet/, handleSendWalletCommand);
45          bot.onTextОтправить(/ ещёраз /, handleSendWalletCommand)
46
47          // bot.onText(/\/send_tx/, handleSendTXCommand);
48
49          bot.onText(/\/disconnect/, handleDisconnectCommand);
50
51          bot.onText(/\/my_wallet/, handleShowMyWalletCommand);
52
53          bot.onText(/\/start/, (msg: TelegramBot.Message) => {
54              bot.sendMessage(
55                  msg.chat.id,
56                  `Добропожаловать
57      ! ЭтотботпоможетВамполучитьсертификат . Введитекоманду /connect, чтобыначать
            .`
```

```
58                );
59
60                const storage = getStorage(msg.chat.id);
61                storage.setItem("code", msg.text?.split(" ")[1] || "")
62            });
63        }
64
65        main();
```