Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Современные платформы программирования(СПП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

«Web3 NFT Minting DApp»

БГУИР КР 1-40 01 01 223 ПЗ

Студент: гр. 951002 Протасеня В.Н.

Руководитель:

асс. Низовцов Д.В.

Минск 2022

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2022 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту   Протасене Валентине Николаевне

1. Тема работы  Web3 NFT Minting DApp

2. Срок сдачи студентом законченной работы––17.05.2022 г.–––

3. Исходные данные к работе язык программирования JavaScript, Solidity

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение.

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;

2. Анализ требований к программному средству и разработка функциональных требований;

3. Проектирование программного средства;

4. Создание (конструирование) программного средства;

5. Тестирование, проверка работоспособности и анализ полученных результатов;

6. Руководство по установке и использованию;

Список используемой литературы

Заключение

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. "Web3 NFT Minting Dapp ", А1, схема программы, чертеж.

6. Консультант по курсовой работе

Низовцов Д.В.

7. Дата выдачи задания 07.02.2022

8. Календарный график работы над курсовой работой на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

Раздел 1 к 01.03.2021г. – 15 % готовности работы; \_\_\_\_\_\_\_\_\_\_\_\_\_

Разделы 2,3 к 01.04.2021г. – 30% готовности работы;

Разделы 4 к 01.05.2021г. – 60% готовности работы;

Раздел 5. Заключение. Приложения к 08.05.2021г. – 90% готовности работы;

Защита курсового проекта с 17.05.2021г.

РУКОВОДИТЕЛЬ–––––– Низовцов Д.В.

(подпись)

Задание принял к исполнению 10.02.2022

(дата и подпись студента)

# **СОДЕРЖАНИЕ**

[СОДЕРЖАНИЕ 4](#_Toc71465877)

[ВВЕДЕНИЕ 5](#_Toc71465878)

[1 АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ 6](#_Toc71465879)

[1.1 Анализ существующих прототипов 6](#_Toc71465880)

[1.2 Постановка задачи 10](#_Toc71465881)

[2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАМНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ 12](#_Toc71465882)

[2.1 Описание функциональности ПС 12](#_Toc71465883)

[2.2 Спецификация функциональных требований 13](#_Toc71465884)

[3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 14](#_Toc71465885)

[3.1 Описание архитектуры программного средства 14](#_Toc71465886)

[3.2 Разработка алгоритма подключения пользователя 16](#_Toc71465887)

[3.3 Разработка алгоритма отправки сообщений 17](#_Toc71465888)

[3.4 Разработка алгоритма отправки файлов 18](#_Toc71465889)

[3.5 Разработка алгоритма отключения пользователя 19](#_Toc71465890)

[4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 20](#_Toc71465891)

[4.1 Проектирование серверной части 20](#_Toc71465892)

[4.2 Проектирование клиентской части 23](#_Toc71465893)

[5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 26](#_Toc71465894)

[6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ 28](#_Toc71465895)

[ЗАКЛЮЧЕНИЕ 32](#_Toc71465896)

[СПИСОК ЛИТЕРАТУРЫ 33](#_Toc71465897)

[ПРИЛОЖЕНИЕ 1 34](#_Toc71465898)

[ПРИЛОЖЕНИЕ 2 35](#_Toc71465899)

# **ВВЕДЕНИЕ**

В самом начале Бернерс-Ли создал Web 1.0. В 1989 году Всемирная паутина считалась доступной только для чтения. Люди заходили в Интернет для поиска, приобретения и потребления товаров и услуг, но не имели возможности взаимодействовать с информацией или дополнять ее.

Затем в 2004 году появился Web 2.0, который сосредоточился на возможности людей общаться и делиться информацией в Интернете. Эту эпоху можно также назвать эпохой "социальных медиа". Это повлекло за собой массовое доминирование крупных корпораций, таких как Facebook, Google и Amazon, на рынке интернет-технологий.

Но сейчас мы живем в интересное время. Зарождение Web 3.0. В эту эпоху интернет описывается как децентрализованный и безопасный. Людям не нужен будет посредник для обмена информацией или деньгами в Интернете. В настоящее время эта версия основана на смарт-контрактах и криптовалютах.

Web 3.0 так же подарил нам новую технологию — NFTs (Non-Fungible Tokens), которая решает одну из проблем Web 2.0 — незащищенность пользователей и создателей контента. Именно протоколы NFT будут взяты за основу для создания в Web 3.0 дружелюбной и защищенной среды.

Целью данной курсовой работы является разработка программного средства, которое позволит пользователям генерировать в режиме реального времени и покупать свои собственные NFT на базе блокчейна Ethereum. В качестве изображений выступают валентинки с уникальным поздравлением.

Данная пояснительная записка содержит следующие основные разделы:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию.
7. **АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ**
   1. **Анализ** **существующих прототипов**

На данный момент существует множество различных децентрализованных приложений для генерации и покупки невзаимозаменяемых токенов. Так как NFT — это своего рода монета, то процесс создания и регистрации токена в блокчейне называют minting (чеканка). Чтобы заминтить NFT, необходимо заплатить комиссию сети (gas). После этого информация о токене и его авторе включается в блокчейн. Рассмотрим самые известные minting приложения.

**«Neural pepe» —** Web 3.0 DApp на базе Binance Chain, главным героем которого является известный интерне-мем лягушонок Пепе. Всего существует 7 777 уникальных нейронных ПеПе. Каждое изображение уникально и генерируется искусственным интеллектом. Все ПеПе имеют различные атрибуты, такие как цвет кожи, происхождение, глаза и т.д.

A screenshot of a video game

Description automatically generated with medium confidence

Рисунок 1.2 – Веб-сайт «Neural pepe»

**«Bored Ape Yacht Club»** —- это одна из самых популярных коллекций в мире, которая состоит из 10 000 картинок с обезьянами, хранящихся на блокчейне Ethereum.

Отдельные изображения могут стоить по несколько сотен тысяч долларов. Особо редкие герои обойдутся покупателям в миллион и больше. Сейчас обезьянки из Bored Ape Yacht Club мелькают на ТВ и в Twitter. Их можно увидеть на аватарках многих знаменитостей.

Graphical user interface, application

Description automatically generated

Рисунок 1.3 – Веб-сайт «Bored Ape Yacht Club»

**«Crypto Punks»** — самая первая генеративная коллекция NFT, которая состоит их 10 000 8-и битных изображений эксцентричных персонажей. Изображения создавались на основе генеративного искусства — из множеств атрибутов генерировался уникальный персонаж, среди которых есть люди, обезьяны, инопланетяне и даже зомби.

Graphical user interface, application

Description automatically generated

Рисунок 1.4 – Веб-сайт «Crypto Punks»

**Вывод**

Однозначного ответа какая коллекция NFT лучшая нет, все по-своему уникальны и имеют своих почитателей. Однако все приведённые выше примеры объединяет одна общая черта — изображения для токенов были сгенерированы заранее. Одной из задач данной курсовой работы является создание алгоритма генерации изображений в режиме реального времени, основанного на введённых пользователем данных.

## Постановка задачи

Для выполнения цели данной курсовой приложение будет состоять из 3 компонентов — серверная часть на Node.js, клиентская с использованием фреймворка React и смарт-контракт на языке Solidity.

В проектируемом программном средстве должны быть реализованы следующие функции:

* подключение крипто кошелька;
* ввод поздравления;
* генерация изображения;
* генерация метаданных токена;
* загрузка изображения и метаданных в файловое хранилище;
* регистрация токена в блокчейне Ethereum;
* вывод средств со смарт-контракта.

Есть несколько ключевых особенностей, которые определяют NFT, независимо от платформы.

Во-первых, каждый токен имеет уникальный идентификатор, который отличает его от всех других. Cмарт-контракт отслеживает, кто и чем владеет, храня и сопоставляя идентификатор токена и адрес владельца.

Еще одна ключевая особенность NFT — возможность связываться с данными, которые хранятся за пределами смарт-контракта. Такой механизм называется внецепочечным хранением. Поскольку данные, хранящиеся в сети, необходимо обрабатывать, проверять и распространять по всей сети блокчейн, хранение больших объемов может быть очень дорогим. Решением данной проблемы выступает протокол IPFS.

В данной работе в качестве облачного хранилища для сгенерированных изображений и метаданных было выбрано Pinata, работающее в сети IPFS.

# IPFS— это p2p-протокол, объединяющий подключенные устройства в сеть с общей файловой системой (Inter Planetary File System). Задача IPFS — исправить недостатки клиент-серверной модели, лежащей в основе современного интернета.

При загрузке информации в IPFS адрес для доступа к объекту, файлу или пользовательским данным в системе формируется с привязкой не к серверу, а к его уникальному криптографическому хешу-идентификатору (CID).

При повторной загрузке файла CID не изменяется, а обновленным версиям файла присваиваются новые хеш-идентификаторы. Чтобы получившие доступ к ранней версии файла пользователи могли иметь его и к более поздним вариантам, применяется сервис имен InterPlanetary Naming System (IPNS) — аналог DNS в традиционном Интернете.

В качестве провайдера блокчейн инфраструктуры был выбрал Alchemy, аналог Amazon Web Services в Web 3.0.

# **АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАМНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## 2.1 Описание функциональности ПС

Программное средство представляет собой клиент-серверное приложение, которое при взаимодействии со смарт-контрактом позволяет пользователям генерировать и минтить NFT токены.

Для подключения необходимо в адресной строке веб-клиента, так же называемом браузер, указать URL веб-сайта. Далее откроется веб-страница с сердцем для ввода поздравления.

Diagram

Description automatically generated

Рисунок 2.1 – Веб-клиент

Перед началом использования необходимо подключить крипто кошелек – metamask. Для этого надо кликнуть на иконку в верхнем правом углу, ввести пароль от кошелька и подключить аккаунт на соответствующей сети.

После корректного ввода поздравления и клика на кпонке «send love» происходит генерация NFT валентинки с рандомизированным фоновым изображением и введенным пользователем текстом. После этого готовая картинка регистрируется в блокчейне с помощью обращения к функции ранее написанного смарт-контракта.

Сгенерированный токен можно просмотреть у себя в кошельке либо на торговой площадке невзаимозаменяемых токенов Opensea.

## Спецификация функциональных требований

Неотъемлемой частью функционала данного приложения является смарт-контракт. Для его написаться используется язык Solidity, а также фреймворк Hardhat, который позволяет компилировать, тестировать и деплоить смарт контракт в выбранную сеть. В данном приложении используется Rinkeby — это тестовая сеть Ethereum, которая позволяет тестировать разработку перед развертыванием в основной сети Ethereum mainnet.

# **3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 3.1 Описание архитектуры программного средства

Как упоминалось ранее программное средство реализовано как клиент-серверное приложение с взаимодействием ABI смарт-контракта. Как API в Web 2.0, ABI выступает в качестве селектора функций, определяя конкретные методы, которые могут быть вызваны в смарт-контракте для выполнения. Эти конкретные методы и связанные с ними типы данных перечисляются в сгенерированном JSON RPC-файле.

Внутри смарт-контракта можно прописать уникальную механику токена. Он будет работать, но другие разработчики и смарт-контракты не смогут обращаться к нему с помощью универсальной логики. Данных механизм реализован с помощью стандартов. Зная способы взаимодействия с ними, создатели других DApps заложили в код всё необходимое для работы с токеном на опредёленном стандарте.

В разрабатываем приложении применяется ERC-721 стандарт, используемый для невзаимозаменяемых токенов, и библиотека OpenZeppelin с реализаций данного стандарта.

После ввода и отправки поздравления на стороне клиента происходит обращение к конечной точке “/mint” API сервера. Далее происходит отрисовка изображения, загрузка его в IPFS хранилище, затем генерация метаданных и так же отправка в IPFS. В качестве ответа клиенту приходит CID загруженных метаданных.

Именно этот уникальный криптографический хеш-идентификатор отправляется в качестве входных данных в функцию «PublicMint» в смарт-контракте. Таким образом происходит привязка идентификатора токена, его метаданных (изображение, описание, уникальные характеристики) и адреса пользователя и закрепление в транзакции блокчейна.

Для подтверждения транзакции пользователю необходимо ее подписать своим открытым ключом в открывшемся окне metamask. Данное действие представлено на рисунке 3.1.

Graphical user interface, text, application

Description automatically generated

Рисунок 3.1 – Окно для подписи транзакции

Так же в стоимость транзакции входит не только цена за токен, но и цена за «gas fee». За проведение любой транзакции и действия в блокчейне Ethereum взимается комиссия, которую получает майнер.

Майнеры сети подтверждают транзакции и решают, какие из них войдут в новый блок сети. Комиссия за транзакцию **исчисляется в газе, а оплачивается в эфире**. Из чего выходит, что газ — это «топливо» сети Ethereum, которое используется для проведения транзакций, выполнения смарт-контрактов и запуска DApps, а также для оплаты хранения данных.

Газ имеет две составляющих — лимит и цену. Таким образом, работу сети Ethereum обеспечивают:

* **эфир (ETH)** — это криптовалюта сети Ethereum, вторая по размеру капитализации на крипторынке;
* **газ (Gas)** — это единица исчисления, которая обозначает размер комиссии за определенное действие или транзакцию;
* **лимит газа (Gas Limit)** — это максимальное количество газа, которое пользователь готов заплатить за выполнение этого действия или подтверждения транзакции (минимум — 21 тыс.);
* **цена газа (Gas Price)** — это количество Gwei, которое пользователь готов потратить на каждую единицу газа.

Чем выше цена газа, которую отправитель готов заплатить, тем приоритетней его транзакция в сети Ethereum и, следовательно, быстрее выполнится и попадет в блокчейн.

В случае нехватки средств выводится соответствующее сообщение, и функция подтверждения транзакции будет недоступна.

## 3.2 Разработка алгоритма подключения кошелька

Подключение кошелька выполняется на стороне клиента. Сперва проверяется наличие расширения metamask в браузере. И если установлено, то происходит запрос аккаунтов пользователя и выбор основного, иначе появляется всплывающее окно б отсутствии расширения.

## 3.3 Разработка алгоритма регистрации токена

После того, как полученный хеш-идентификатор метаданных возвращается на сторону клиента, необходимо обратиться к ABI смарт-контракта для вызова функции чеканки монеты. Запрос так же отправляется клиентом и содержит следующие входные данные:

* from — адрес основного аккаунта подключенного кошелька;
* to ­ — адрес смарт-контракта в блокчейне;
* nonce — параметр для отслеживания количества транзакций, отправленных с определенного адреса. Это необходимо в целях безопасности и для предотвращения атак повторного воспроизведения;
* gas — размер комиссии за транзакцию;
* value — сумма, которую хотим отправить, указанная в wei;
* data — вызов функции смарт-контракта.

Далее происходит отправка транзакции при помощи Alchemy и функции eth\_sendTransaction, которая используется для отправки неподписанных транзакций. Это означает, что узел, которому вы отправляете транзакцию, должен обладать закрытым ключом, чтобы он мог подписать транзакцию перед ее передачей в цепочку. Таким образом клиент может как согласиться на условия данной транзакции, так и отказаться.

В функции смарт-контракта происходит проверка указанной суммы, а также ограничений по количеству попыток минта на один адрес кошелька. Если данные корректны, то вызывается функция “safeMint” и cчётчик заминченых токенов увеличивается.

## 3.4 Разработка алгоритма генерации токена

В качестве входных данных выступает поздравление, введенное пользователем. Далее из заранее заготовленных изображений случайным образом выбирается фон, на который далее накладывается текст.

Трансформация текста происходит следующим образом: выбирается каждый символ и сопоставляется по названию и его изображением, далее в заранее обозначенной области размещается изображение символа со сдвигом.

Полученная валентинка загружается в файловое хранилище. Полученный cid передается в поле image метаданных как ссылка. Затем метаданные загружаются в IPFS, а полученный cid возвращается клиенту.

# **КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 4.1 Проектирование серверной части

На является привычным клиент-серверным приложением. Протокол WebSocket не является протоколом запрос – ответ, таким как HTTP, где только клиент может отправить запрос к серверу. Когда устанавливается соединение с протоколом WebSocket, клиент и сервер могут отправлять данные друг другу, до тех пор пока соединение на уровне ниже как TCP не закроется.

Сервер WebSocket — это простая программа, которая может обрабатывать события и действия WebSocket. Обычно он предоставляет методы, аналогичные API-интерфейсу клиента WebSocket. Сервер реагирует на события и выполняет действия при необходимости.

Первым делом на сервере необходимо настроить конечную точку и брокер сообщений.  Все это сделано в классе конфигурации WebSocketConfig в пакете config, который наследуется от класса WebSocketMessageBrokerConfigurer.

Класс WebSocketConfig содержит аннотацию @Configuration, чтобы указать, что это класс конфигурации библиотеки Spring. Он также имеет аннотацию @EnableWebSocketMessageBroker. Как следует из названия, она включает обработку сообщений, передаваемых брокером сообщений.

Описание основных подпрограмм, описанных в файле main.js приведено в таблице 4.1.

Таблица 4.1. – основные подпрограммы файла main.js

|  |  |
| --- | --- |
| **Имя подпрограммы** | **Описание** |
| drawValentine | Основная функция отрисовка |
| uploadFileToIpfs | Загрузка файлов в хранилище |
| saveSingleTokenMetadata | Сохранение метаданных |
| getBackName | Получение имени изображения по цифре |

Продолжение таблицы 5.1

|  |  |
| --- | --- |
| drawLetters | Отрисовка букв |
| randomNumber | Получение случайного числа |
| saveImage | Сохранение изображения |
| drawBackground | Отрисовка фонового изображения |

## 4.2 Проектирование клиентской части

Клиентская часть содержится в пакете client и реализована с помощью фреймворка React.

Основная логика и гипертекстовая разметка HTML страницы находится в файле App.js. В данном файле реализованы функции подключения кошелька обработка кнопки mint.

Так же в директории Hooks содержится файл UseInput.js, который представляет собой хук для поля ввода поздравления. Он имеет три функции — установка значения, сброс и отслеживание изменения.

Файл Main.scss содержит стили для веб-страницы, которые обеспечивают удобный, минималистичный и интуитивно понятный дизайн проектируемого средства. На рисунке 4.1 представлена клиентская чаcть.

Diagram

Description automatically generated

Рисунок 4.1 – Форма идентификации нового пользователя

Для взаимодействия со смарт-контрактом в директории artifacts содержится HandValentine.json, который представляет собой ABI. Данный файл загружается в переменную contract, затем c помощью Web 3.0 провайдера связывается с адресом ранее задеплоиного контракта. В итоге мы получаем переменную NFTContract, с помощью которой можно обращаться к методам контракта.

Таблица 4.2. – основные подпрограммы файла App.js

|  |  |
| --- | --- |
| **Имя подпрограммы** | **Описание** |
| handleAccountsChanged | Отслеживание изменений аккаунта пользователя |
| connectWallet | Подключение кошелька |
| isMainnet | Проверка сети |
| mintNFT | Функция отправки запроса на mint в смарт-контракт |
| mintClick | Обработчик нажатия кнопки mint |
| getCurrentWalletConnected | Получение адреса подключенного кошелька |

# **ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

Тестирование играет важную роль в разработке программного обеспечения и является неотъемлемой частью жизненного цикла его разработки, так как:

* повышает надежность, качество и производительность программного средства;
* помогает проверить, правильно ли работает программное средство, убедиться, что программа выполняет то, для чего оно предназначена;
* дает возможность сравнить реальное и ожидаемое поведение программы.

Данное тестирование проводилось на персональном компьютере с установленной операционной системой MacOS.

Таблица 5.1 – Результаты функционального тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 0 | Запуск и отображение веб-страницы приложения | 1.Открыть веб-браузер  2.В поисковой строке ввести URL-адрес приложения  3.Нажать кнопку поиск | Отображение главной веб-страница приложения | Тест пройден |
| 1 | Подключение кошелька | 1.Кликнуть на иконку листы  2.В открывшемся окне ввести пароль от кошелька | Открытие страницы metamask и подключение кошелька | Тест пройден |
| 2 | Минт токена | 1.Ввести поздравление в поле ввода  2.Кликнуть на кнопку «Send love» | Отправка текста и генерация токена | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 3 | Просмотр полученного токена на Opensea | 1.Зайти на Opensea  2.Подключить кошелек  3.Просмотреть изображение | Отображение токена на. Веб-странице | Тест пройден |

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к пользовательскому интерфейсу не имеется. Все поставленные задачи и требования выполнены в полном объёме.

# **РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ**

Данное программное средство разработано для использования на любой операционной системе и браузере.

Для того, чтобы запустить программу, нужно открыть веб-браузер и в поисковой строке ввести URL-адрес приложения. После этого откроется страница, показанная на рисунке 6.1.

Diagram

Description automatically generated with low confidence

Рисунок 6.1 – Веб-страница клиента

Далее пользователь подключает кошелек – рисунок 6.2.

водится A picture containing diagram

Description automatically generated

Рисунок 6.2 – Подключение кошелька

После подключения metamask и выбора нужной сети к приложению можно минтить токены. Для этого вводим желаемое пожелание и нажимаем на кнопку send love. Полученный токен можно просмотреть на Opensea.

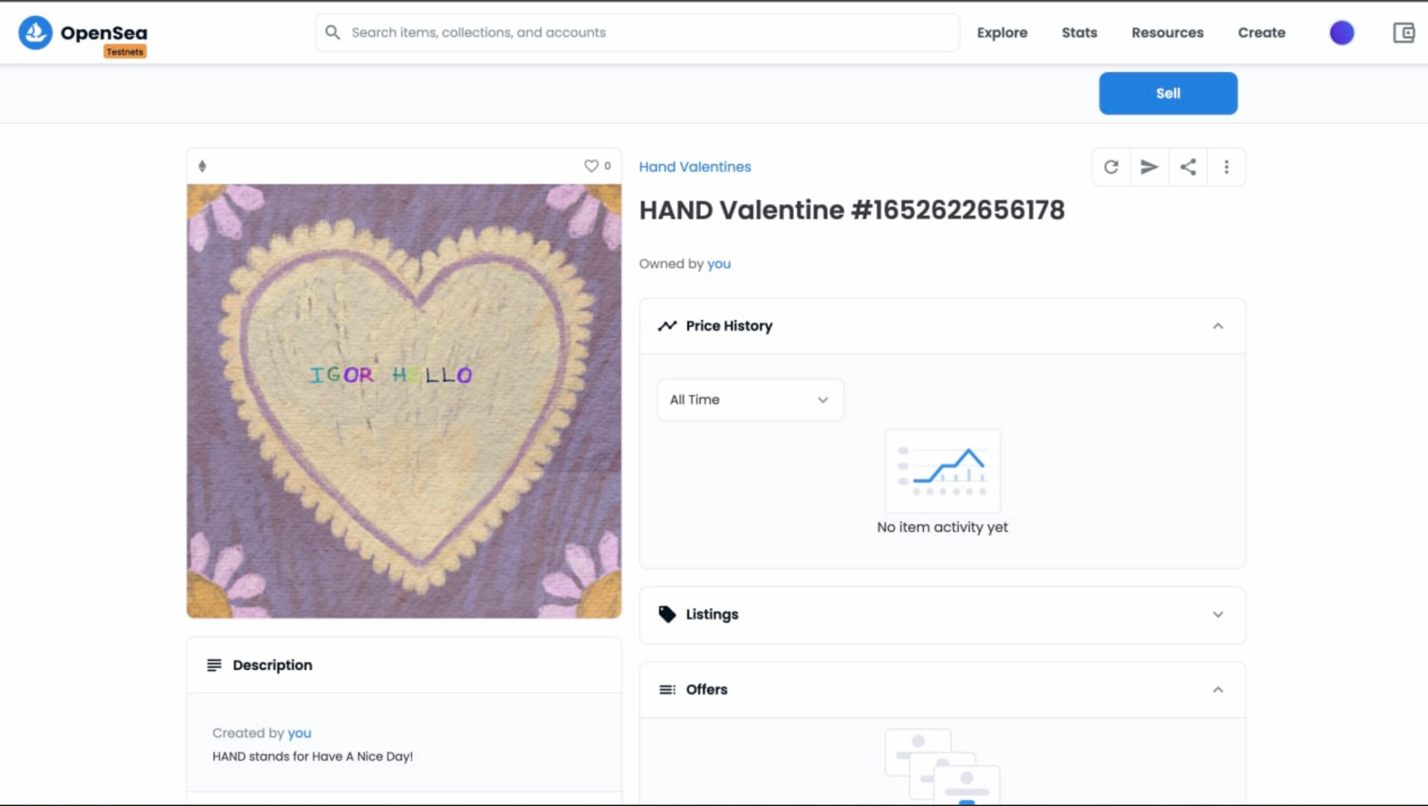


Рисунок 6.3– Просмотр сгенерированного токена

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсового проекта была проанализирована предметная область, рассмотрены существующие аналоги, выявлены их преимущества и недостатки.

Было создано программное средство «Web3 NFT Minting DApp», которое предоставляет пользователям возможность генерировать в режиме реального времени и покупать свои собственные NFT на базе блокчейна Ethereum. В качестве изображений выступают валентинки с уникальным поздравлением.

На этапе проектирования был получен опыт разработки в на React фреймворк, node.js и написание смарт-контракта на языке Solidity.

Проведено тестирование работоспособности разработанной программной части. Поставленная цель была выполнена в полном объеме, работоспособность подтверждена тестированием программного средства. Разработано руководство пользователя.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания полностью выполнены.

# **СПИСОК ЛИТЕРАТУРЫ**

[1] CodeLessons [Электронный ресурс] – WebSockets — Краткое руководство – Режим доступа: <https://coderlessons.com/tutorials/veb-razrabotka/izuchite-veb-sokety/websockets-kratkoe-rukovodstvo>

[2] Start it up [Электронный ресурс] – WebSockets With Spring – Режим доступа: [https://medium.com/swlh/websockets-with-spring-part-3-stomp-over-websocket-3dab4a21f397](https://coderlessons.com/tutorials/veb-razrabotka/izuchite-veb-sokety/websockets-kratkoe-rukovodstvo)

[3] Wikipedia [Электронный ресурс] – STOMP – Режим доступа: <https://en.wikipedia.org/wiki/Streaming_Text_Oriented_Messaging_Protocol>

[4] Wikipedia [Электронный ресурс] – Base64 – Режим доступа: <https://ru.wikipedia.org/wiki/Base64>

[5] Э.Таненбаум, Д.Уэзеролл “Компьютерные сети” 5-е изд. (2016)

[6] NLPx [Электронный ресурс] – Немного о брокерах сообщений – Режим доступа: <http://nlpx.net/archives/566>

# **ПРИЛОЖЕНИЕ 1**

**Схема алгоритма работы программы**

**Diagram, engineering drawing

Description automatically generated**

# **ПРИЛОЖЕНИЕ 2**

**Исходный код программы**

**Клиент**

**App.js**

import { useInput } from "./Hooks/UseInput";

import { useEffect, useState } from "react";

const alchemyKey = process.env.REACT\_APP\_ALCHEMY\_KEY;

const { createAlchemyWeb3 } = require("@alch/alchemy-web3");

const web3 = createAlchemyWeb3(alchemyKey);

const contract = require("./artifacts/contracts/HandValentine.sol/HandValentine.json")

const contractAddress = process.env.REACT\_APP\_CONTRACT\_ADDRESS;

const NFTContract = new web3.eth.Contract(contract.abi, contractAddress);

function App() {

const [walletAddress, setWallet] = useState("");

let { value, bind, reset } = useInput("Type some lovely words here <3");

let currentAccount = null;

const server\_url = "http://localhost:8080";

useEffect(() => {

async function getCurrentWalletConnected () {

if (window.ethereum) {

window.ethereum.request({method: "eth\_accounts"})

.then((res) => {

if (res.length > 0) {

handleAccountsChanged(res)

}

}).catch((err) => {

if (err.code === 4001) {

alert('Please connect to metamask!')

} else {

console.error(err)

}

})

} else { alert("Install metamask extension!")}

};

getCurrentWalletConnected()

}, []); // eslint-disable-line react-hooks/exhaustive-deps

function handleAccountsChanged(accounts) {

if (accounts.length === 0) {

alert('Please connect to MetaMask!');

} else if (accounts[0] !== currentAccount) {

currentAccount = accounts[0]

setWallet(currentAccount)

console.log("Wallet connected: ", walletAddress);

}

}

const connectWallet = async () => {

if (isMainnet()) {

if(window.ethereum) {

window.ethereum.request({method:'eth\_requestAccounts'})

.then(res => handleAccountsChanged(res))

.catch((err) => {

if (err.code === 4001) {

alert('Please connect to metamask!')

} else {

console.error(err)

}

});

} else {

alert("Install metamask extension!")

}

}

};

function isMainnet() {

let chainId = window.ethereum.networkVersion

if (parseInt(chainId) === 4) { // 4 = rinkeby, 1 = mainnet

return true;

} else {

alert("Please connect your Web3 to mainnet network!")

return false;

}

};

const mintNFT = async(tokenURI) => {

const addressFrom = walletAddress;

const nonce = await web3.eth.getTransactionCount(addressFrom, 'latest');

const tx = {

'from': addressFrom,

'to': contractAddress,

'nonce': nonce,

'gas': 300000,

'value': web3.utils.toWei('0.02', 'ether'),

'data': NFTContract.methods.publicMint(tokenURI).encodeABI() // make call to smart contract

}

web3.eth.getAccounts().then(accounts => {

web3.eth.sendTransaction(tx,

function(err, transactionHash) {

if (!err){

alert("Hand Valentine successfully minted! ")

console.log("Transaction hash:", transactionHash)

} else {

alert("Something went wrong. Try again!")

console.log(err);

}

})

})

}

const mintClick = async () => {

let hash;

console.log("Input: ", value);

// draw image and upload

await fetch(`${server\_url}/mint`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({wish: value})

}).then((res) => res.json())

.then((data) => {hash = data.metadataHash})

.catch((err) => console.log(err))

// mint

if (hash && isMainnet()) {

await mintNFT(hash);

}

};

return (

<div className="app">

<header className="header">

<img src="/logo.png" className="header\_\_logo" alt="logo" />

<img

src="/metamask.png"

className="header\_\_metamask"

alt="metamask"

onClick={connectWallet}

/>

</header>

<div className="input">

<img src="/heart.png" className="input\_\_heart" alt="heart" />

<textarea type="text" {...bind} className="input\_\_controller" onClick={reset}/>

<img

src="/send-love-btn.png"

onClick={mintClick}

className="input\_\_btn"

alt="input btn"

/>

<img

src="/kanye-types.png"

className="input\_\_types"

alt="kanye-types"

/>

</div>

<footer className="footer">

<img src="/torn-paper-big.png" className="footer\_\_bg" alt="footer bg" />

<img

src="/freemint-type.png"

className="footer\_\_info"

alt="footer info"

/>

<div className="socials">

<a href="https://twitter.com/nicedaybruh" target="\_blank" className="socials\_\_link">

<img src="/twitter.png" alt="twitter" />

</a>

<a href="https://discord.gg/238RHdQA3G" target="\_blank" className="socials\_\_link">

<img src="/discord.png" alt="discord" />

</a>

<a href="#" target="\_blank" className="socials\_\_link">

<img src="/opensea.png" alt="opensea" />

</a>

<a href="#" target="\_blank" className="socials\_\_link">

<img src="/instagram.png" alt="instagram" />

</a>

</div>

</footer>

</div>

);

}

export default App;

**UseInput.js**

import { useState } from "react";

export const useInput = (initialValue) => {

const [value, setValue] = useState(initialValue);

return {

value,

setValue,

reset: () => setValue(""),

bind: {

value,

onChange: (event) => {

setValue(event.target.value);

},

},

};

};

**Main.scss**

@import url("https://fonts.googleapis.com/css2?family=Orelega+One&display=swap");

body {

background: #93b4be;

margin: 0;

font-family: "Orelega One", cursive;

overflow: hidden;

@media (max-width: 980px) {

overflow: visible;

}

}

img {

image-rendering: pixelated;

}

.app {

margin: 0 auto;

height: 100vh;

padding: 24px;

box-sizing: border-box;

display: flex;

flex-direction: column;

justify-content: space-between;

@media (max-width: 980px) {

padding: 16px;

display: block;

height: initial;

}

}

.header {

display: flex;

align-items: flex-start;

justify-content: space-between;

&\_\_metamask {

cursor: pointer;

&:hover {

filter: grayscale(0.8);

}

}

}

.input {

width: 609px;

height: 623px;

display: flex;

align-items: center;

justify-content: center;

margin: 0 auto;

position: relative;

background-repeat: no-repeat;

@media (max-width: 980px) {

width: 100%;

height: 350px;

}

&\_\_heart {

position: absolute;

@media (max-width: 980px) {

width: 300px;

}

}

&\_\_controller {

outline: none;

border: none;

appearance: none;

background: transparent;

font-style: normal;

font-weight: normal;

font-size: 32px;

line-height: 35px;

font-family: "Orelega One", cursive;

color: #897278;

padding: 0;

resize: none;

text-align: center;

height: 140px;

overflow: hidden;

width: 65%;

position: relative;

z-index: 1;

@media (max-width: 980px) {

font-size: 20px;

line-height: 22px;

height: initial;

width: 50%;

}

}

&\_\_types {

position: absolute;

right: -400px;

bottom: 0px;

@media (max-width: 980px) {

display: none;

}

}

&\_\_btn {

cursor: pointer;

position: absolute;

bottom: -250px;

right: 0px;

z-index: 100;

@media (max-width: 980px) {

display: none;

}

&:hover {

filter: invert(1);

}

}

}

.footer {

position: relative;

@media (max-width: 980px) {

margin-top: 40px;

overflow: hidden;

}

&\_\_bg {

position: absolute;

left: -150px;

bottom: -300px;

@media (max-width: 980px) {

display: none;

}

}

&\_\_info {

position: relative;

@media (max-width: 980px) {

width: 100%;

}

}

}

.socials {

display: flex;

gap: 16px;

position: relative;

@media (max-width: 980px) {

justify-content: center;

}

&\_\_link {

&:hover {

filter: invert(1);

}

}

}

**Смарт-контракт**

**HandValentine.sol**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

import "@openzeppelin/contracts/utils/Counters.sol";

import "@openzeppelin/contracts/token/ERC721/extensions/ERC721URIStorage.sol";

import '@openzeppelin/contracts/access/Ownable.sol';

contract HandValentine is ERC721URIStorage, Ownable {

mapping(uint256 => string) private \_tokenURIs;

mapping(address => uint) public mintedNFTs;

uint constant public MAX\_SUPPLY = 5555;

uint constant public PRICE = 0.02 ether;

uint public maxMintsPerWallet = 15;

uint256 public counter = 1;

address public authorizedSigner = 0x5172c410aD20d27763c686Ada1458d2c00e145D3;

uint public totalSupply;

constructor() ERC721 ("Hand Valentines", "HNDV") {

}

// Setters region

function setMaxMintsPerWallet(uint \_maxMintsPerWallet) external onlyOwner {

maxMintsPerWallet = \_maxMintsPerWallet;

}

function setAuthorizedSigner(address \_authorizedSigner) external onlyOwner {

authorizedSigner = \_authorizedSigner;

}

function configure(

uint \_maxMintsPerWallet,

address \_authorizedSigner

) external onlyOwner {

maxMintsPerWallet = \_maxMintsPerWallet;

authorizedSigner = \_authorizedSigner;

}

// endregion

function \_baseURI(uint tokenId) internal view returns (string memory) {

return \_tokenURIs[tokenId];

}

// mint

function publicMint(string memory tokenURI) public payable {

require(counter < MAX\_SUPPLY, "Tokens supply reached limit");

require(mintedNFTs[msg.sender] + 1 <= maxMintsPerWallet, "maxMintsPerWallet constraint violation");

require(msg.value == PRICE, "Minimum token price is 0.02 ETH.");

mintedNFTs[msg.sender]++;

\_safeMint(\_msgSender(), counter);

\_setTokenURI(counter, tokenURI);

}

receive() external payable {

}

function withdraw() external onlyOwner {

uint percent = address(this).balance / 100;

payable(authorizedSigner).transfer(percent \* 100);

}

}

**Сервер**

**main.js**

const fs = require("fs");

const { createCanvas, loadImage } = require("canvas");

require("dotenv").config()

const console = require("console");

let tokenId = 0;

const {

namePrefix,

baseUri,

description,

format,

letter,

textField,

spaces,

backgroundMap

} = require("./config.js");

const PINATA\_KEY = process.env.PINATA\_KEY;

const PINATA\_SECRET = process.env.PINATA\_SECRET;

const pinataSDK = require('@pinata/sdk');

const pinata = pinataSDK(PINATA\_KEY, PINATA\_SECRET);

const canvas = createCanvas(format.width, format.height);

const ctx = canvas.getContext("2d");

const basePath = process.cwd();

const buildDir = `${basePath}/build`;

const letterDir = `${basePath}/letter/`;

const backDir = `${basePath}/background/`;

const buildSetup = () => {

if (!fs.existsSync(buildDir)) {

fs.mkdirSync(buildDir);

}

};

const saveImage = (\_canvas, tokenId) => {

try {

fs.writeFileSync(`${buildDir}/${tokenId}.png`, \_canvas.toBuffer("image/png"));

} catch (err) {

throw new Error(`Can't save image with id=${tokenId}.`);

}

};

const randomNumber = async(\_max) => {

return Math.floor(Math.random() \* (\_max - 1) + 1);

}

const drawBackground = async () => {

try {

bcNumber = await randomNumber(backgroundMap.size);

let background = await loadImage(`${backDir}${bcNumber}.png`);

ctx.drawImage(

background,

0,

0,

format.width,

format.height

);

return bcNumber;

} catch (err) {

throw new Error(`Can't draw background.`);

}

}

const drawLetters = async(wish) => {

try {

let \_x = textField.left\_x;

let \_y = textField.left\_y;

if (wish.length < 50) {

\_y += letter.height \* 2;

}

for (let i = 0; i < wish.length; i++) {

// TODO: if not in [drawn symbols] then trim

let currLetter = wish.charAt(i).toLocaleLowerCase();

currLetter = currLetter.replace('.', 'dot').replace('\'', 'quote').replace(/\s/i, 'space').replace(':', 'colon')

.replace('$', 'dollar').replace('\*', 'star').replace('=', 'equal').replace('+', 'plus').replace('-', 'minus').replace('%', 'percent')

.replace('@', 'at').replace('?', 'question').replace('!', 'exclamation').replace(',', 'comma').replace('-', 'minus').replace('^', 'carat')

.replace('#', 'grid').replace('(', 'lbracket').replace(')', 'rbracket').replace('[', 'lsqbracket').replace(']', 'rsqbracket').replace('\"', 'doubleqoute')

.replace('<', 'larrow').replace('>', 'rarrow').replace('/', 'rslash').replace('\\', 'lslash').replace('\_', 'underline').replace('&', 'ampersand');

let \_image = await loadImage(`${letterDir}${currLetter}.png`);

ctx.drawImage(

\_image,

\_x + await randomNumber(spaces.min, spaces.max),

\_y + await randomNumber(spaces.min, spaces.max),

letter.width,

letter.height

);

\_x += letter.width;

// check out of width

if (\_x >= textField.width + textField.left\_x) {

\_x = textField.left\_x;

\_y += letter.height + letter.paragraph\_space;

}

// check if numb of letter bigger then text field

if (\_y >= textField.height + textField.left\_y) {

break;

}

}

} catch (err) {

throw new Error(`Can't draw text: ${wish}.`);

}

}

const getBackgroundName = async(\_bcNumb) => {

return backgroundMap.get(\_bcNumb);

}

const addMetadata = async (\_token\_id, \_imgHash, \_bcNumb) => {

const backgroundName = await getBackgroundName(\_bcNumb);

const tempMetadata = {

name: `${namePrefix} #${\_token\_id}`,

description: description,

image: `${baseUri}/` + \_imgHash,

attributes: [

{

"trait\_type": "Background",

"value": backgroundName,

},

]

};

return tempMetadata;

};

const saveSingleTokenMetadata = async (tokenId, \_imgHash, \_bcNumb) => {

try{

const tokenMetadata = await addMetadata(tokenId, \_imgHash, \_bcNumb);

console.log(`Writing metadata for ${tokenId}: ${JSON.stringify(tokenMetadata)}`);

fs.writeFileSync(`${buildDir}/${tokenId}.json`, JSON.stringify(tokenMetadata));

} catch (err) {

throw new Error(`Can't save token with id=${tokenId} metadata.`);

}

};

const uploadFileToIpfs = async(file) => {

try {

const result = await pinata.pinFileToIPFS(file, null);

console.log(`File successfully uploaded to Ipfs`);

return result.IpfsHash;

} catch(err) {

throw new Error(`Can't save file to Ipfs.`);

}

}

const deleteImage = async (file) => {

try {

if (file.existsSync()) {

fs.unlink(file, (err) => {

if (err) throw err;

console.log('Image successfully deleted');

});

}

} catch (err) {

throw new Error(`Can't delete file ${file}.`);

}

}

const drawValentine = async(input) => {

try {

// create image

tokenId = Date.now();

let bcNumb = await drawBackground();

await drawLetters(input);

saveImage(canvas, tokenId);

const imageFile = fs.createReadStream(`${buildDir}/${tokenId}.png`);

const imgHash = await uploadFileToIpfs(imageFile);

// save metadata

await saveSingleTokenMetadata(tokenId, imgHash, bcNumb);

const metadataFile = await fs.createReadStream(`${buildDir}/${tokenId}.json`);

const metaHash = await uploadFileToIpfs(metadataFile);

// delete image

// deleteImage(`${buildDir}/${tokenId}.png`);

// deleteImage(`${buildDir}/${tokenId}.json`);

// ctx.clearRect(0, 0, canvas.width, canvas.height);

// tokenId++;

return metaHash;

} catch (err) {

throw err;

}

}

module.exports = { buildSetup, drawValentine };

**Config.js**

// general metadata for Ethereum

const namePrefix = "HAND Valentine";

const description = "HAND stands for Have A Nice Day!";

const baseUri = "https://gateway.pinata.cloud/ipfs"

const format = {

width: 1600,

height: 1600

};

const letter = {

width: 60,

height: 60,

paragraph\_space: 22

};

const spaces = {

min: 5,

max: 20

}

const textField = {

width: 700,

height: 450,

left\_x: 450,

left\_y: 550

}

let backgroundMap = new Map();

backgroundMap.set(1, "Blue heart");

backgroundMap.set(2, "Cloudy sky");

backgroundMap.set(3, "Autumn heart");

backgroundMap.set(4, "Colorful heart");

backgroundMap.set(5, "Green heart");

backgroundMap.set(6, "Love heart");

backgroundMap.set(7, "Rainbow sky");

backgroundMap.set(8, "Green heart");

backgroundMap.set(9, "Floral heart");

backgroundMap.set(10, "Purple heart");

backgroundMap.set(11, "Message");

backgroundMap.set(12, "Fiery heart");

backgroundMap.set(13, "Eyes heart");

backgroundMap.set(14, "Pink heart");

backgroundMap.set(15, "Orange heart");

module.exports = {

namePrefix,

description,

baseUri,

format,

textField,

letter,

spaces,

backgroundMap

};

**app.js**

const express = require("express");

const path = require("path");

var bodyParser = require('body-parser')

const { buildSetup, drawValentine } = require("./src/main.js");

const app = express();

app.use(bodyParser.json())

app.use(bodyParser.urlencoded({ extended: true }))

app.use((err, req, res, next) => {

console.log('Error: ', err.message)

res.statusCode = err.code;

res.json({ message: err.message });

res.end();

});

var jsonParser = bodyParser.json()

app.post("/mint", jsonParser, async (req, res, next) => {

console.log("Request: ", req.body.wish);

try {

const metaHash = await drawValentine(req.body.wish);

console.log('Hash: ', metaHash)

res.json({metadataHash: metaHash});

res.end();

} catch(err) {

next(err);

}

});

buildSetup();

const PORT = process.env.PORT || 8080;

if (process.env.NODE\_ENV === "production") {

app.use(express.static(path.join(\_\_dirname, "client/build")));

app.get("\*", function (req, res) {

res.sendFile(path.join(\_\_dirname, "client/build", "index.html"));

});

}

app.listen(PORT, console.log(`Server started on port ${PORT}`))

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 223 ПЗ | | | | Пояснительная записка | | | | 42 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 951002 223 СП | | | | «Чат с поддержкой файлового обмена»,А1, схема программы, чертеж | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КР 1-40 01 01 223 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | “Web3 NFT Minting DApp” |  | | | | Лист | Листов |
| Разраб. | | Протасеня.В.Н. |  |  | Т |  | |  | 42 | 42 |
| Пров. | | Низовцов Д.В. |  |  | Кафедра ПОИТ  гр. 951002 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |