

**Федеральное государственное автономное образовательное
учреждение высшего образования**

**Национальный исследовательский университет
«Высшая школа экономики»**

Факультет компьютерных наук

**Основная образовательная программа
Прикладная математика и информатика**

ГРУППОВАЯ КУРСОВАЯ РАБОТА

Программный проект

на тему

«Реализация NFT маркетплейса на базе Discord API»

Выполнили студенты:

Луц Иван Сергеевич, группа 195, 3 курс,

Басалаев Максим Александрович, группа 195, 3 курс

Токкожин Арсен Ардакович, группа 194, 3 курс

Кусиденов Адильхан Маратович, группа 195, 3 курс

Руководитель КР:

Внешний руководитель Рыжиков Никита Ильич

МОСКВА 2022

СОДЕРЖАНИЕ

1	Аннотация	3
1.1	Аннотация	3
1.2	Abstract	3
1.3	Список ключевых слов	4
2	Введение	5
2.1	Актуальность и значимость	5
2.2	Постановка задачи	6
2.3	Этапы проекта	8
2.4	Структура работы	10
3	Обзор существующих работ и решений	10
3.1	Маркетплейсы	10
3.2	Генеративно-состязательные сети	13
4	Smart-контракты	13
4.1	Структура NFT smart-контракта	13
4.2	Структура маркетплейс smart-контракта	24
5	Discord-бот	29
5.1	Взаимодействие с блокчейнами	29
5.2	Пользовательский интерфейс	34
6	Генеративно-состязательная сеть	34
7	Сервис с генеративно-состязательной сетью	35
8	Выводы и результаты	35
9	Список источников	36
10	Приложения	39
10.1	Ссылка на репозиторий	39

1 Аннотация

1.1 Аннотация

В настоящее время все чаще популяризируется концепция блокчейна. В связи с этим растет количество разных приложений взаимозависимых с данной концепцией. Один из самых популярных объектов является NFT(non-fungible token, невзаимозаменяемый токен). На этой идее существует большое количество протоколов на разных блокчейнах, которые позволяют обмениваться NFT на торговых площадках. Целью данного командного проекта является реализация discord-бота с функционалом NFT маркетплейса в новом и быстроразвивающемся блокчейне NEAR Protocol и сервисом генерации NFT, используя генеративно-сопоставительную сеть. Для этого необходимо было реализовать smart-контракт NFT(согласно стандарту NEP-171), smart-контракт маркетплейса, подстроить API для взаимодействия с блокчейном NEAR-protocol под возможности discord, реализовать discord-бота и написать сервис генерации NFT, в основе которого лежит генеративно-сопоставительная сеть.

1.2 Abstract

Currently, the concept of blockchain is increasingly popularized. In this regard, the number of different applications interdependent with this concept is growing. One of the most popular objects is NFT (non-fungible token). On this idea, there are a large number of protocols on different blockchains that allow you to exchange NFTs on trading floors. The goal of this team project is to implement a discord bot with NFT marketplace functionality on the new and rapidly growing NEAR Protocol blockchain and an NFT generation service using a generative adversarial network. To do this, it was necessary to implement an NFT smart contract (according to the NEP-171 standard), a marketplace smart contract, adjust the API for interacting with the blocked NEAR protocol under the capabilities of discord, implement a discord bot and write an NFT generation service based on generative adversarial network.

1.3 Список ключевых слов

Блокчейн, near, smart-контракты, non-fungible token, генеративно-состязательная сеть, discord-бот, маркетплейс.

2 Введение

2.1 Актуальность и значимость

В целом если рассматривать приложения, которые взаимодействуют с блокчейном, то в последние несколько лет они несомненно являются актуальными и значимыми[1]. В такой сфере мне кажется вопрос об актуальности и значимости лучше делегировать на выбор блокчейна.

Почему же NEAR Protocol является актуальным в нынешнее время? NEAR сеть обработала более 1.5 миллиона транзакций на момент начала 2021 года. Активных аккаунтов в этот же промежуток NEAR насчитывала более 50 тысяч. Несомненно выбор NEAR Protocol в качестве блокчейна является актуальным, потому что за менее чем 7 месяцев он достиг таких цифр.



Рисунок 2.1. Количество транзакций обработанных в NEAR Network

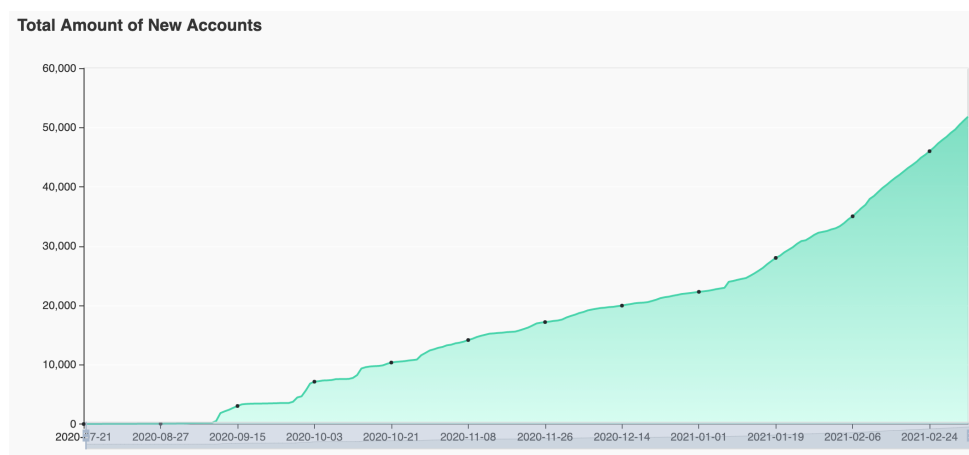


Рисунок 2.2. Количество созданных кошельков в NEAR Network

Объяснение актуальности выбора самого приложения на базе блокчейна опирается на исследование Mapping the NFT revolution[2]. Авторы исследовали тренды 6.1 миллиона обменов в котором участвовало 4.7 миллиона NFT между 23 июнем 2017 и 27 апреля 2021. В заключении они утверждают, что: "В целом, NFT новый инструмент, который удовлетворяет некоторые потребности создателей, пользователей и коллекционеров большого класса цифровых моделей. Как таковые, они, по крайней мере, и останутся или представляют собой начальный шаг к новым инструментам для работы с цифровой собственностью".

Почему именно наша реализация будет актуальной по сравнению с уже существующими решениями:

1. Не существует еще ни одного полноценного NFT маркетплейса в Discord. Все существующие прототипы - это взаимодействие с API браузерных NFT маркетплейсов, который позволяют просто просматривать NFT токены, но не позволяют их создавать или обменивать, или продавать и так далее. Так как Discord - очень популярное приложение, API для построения бота, которого предоставляет очень широкий функционал, то выбор именно Discord по сравнению с аналогами Telegram или Slack.

2. На данный момент не существует ни единого NFT маркетплейса, который встроил бы в себя функцию генерации NFT токена, используя генеративно-состязательную сеть. Мы хотим предоставить пользователю такую возможность, чтобы сэкономить время на придумывание NFT.

Определение. *Discord - популярное приложение для группового чата, изначально было создано для того, чтобы дать геймерам место для создания сообществ и общения.*

Из вышеперечисленного утверждается, что NFT маркетплейс в блокчейне NEAR, предоставляющий интерфейс взаимодействия через Discord бота - соответствует нынешним трендам.

2.2 Постановка задачи

В качестве блокчейна используется NEAR protocol[3]. NEAR Protocol работает по схеме Proof-of-Stake(Pos) [4]. Отличительные черты относительно других блокчейнов - улучшенная масштабируемость, производительность, а также простота реализации приложений.

Определение. Блокчейн — децентрализованная база данных, которая содержит информацию о всех операциях произведенных в ней. Информация об операциях хранится в виде цепочки блоков. Удалить или изменить цепочку блоков невозможно, все это защищено криптографическими методами. Самым первым блокчейном является Bitcoin[5].

Определение. DApps — это приложения, которые включают логику работы с функциями блокчейна[6].

Самой значимой частью реализации DApp являются Smart-контракты. Копии Smart-контрактов разворачивается с помощью специальной транзакции на всех узлах-участниках и исполняются в сети блокчейна.

Определение. Smart-контракт — это неизменяемый исполняемый код, представляющий логику DApp, работающий в блокчейне[6]. Часто сокращают до слова контракт. В некоторых протоколах называют по-другому, например в Solana - это программы[7].

Определение. Транзакция — это наименьшая единица работы, которая может быть назначена сети блокчейна. Работа в данном случае означает вычисление(выполнение функции) или хранение(чтение/запись данных)[8].

Определение. Узлы-участники/валидаторы — множество машин, которое обрабатывает транзакции в блокчейне.

Для написания smart-контрактов Near protocol предоставляет sdk на языках Rust и AssemblyScript(near-sdk-rs[9] и near-sdk-as[10] соответственно). В данном проекте smart-контракты NFT и маркетплейса реализовываются на языке Rust.

Discord-бот реализуется на языке программирования TypeScript, используя near-api-js[11]. Discord-бот либо запускает «view operations», для получения метаданных аккаунта и view методов smart-контрактов; либо, при «change operations» создает транзакции и предоставляет URL для NEAR Wallet аккаунта пользователя. Discord API представляет множественный функционал для общения пользователя с ботом: slash-команды[12], контекстные меню[13], меню выбора[14], кнопки[12], модальности[15](новый функционал, который нужно будет поддержать в ближайшем будущем).

Замечание. Каждый smart-контракт в Near(написанный на Rust/AssemblyScript) переводится в WebAssembly(Wasm), который исполняет виртуальная

машина на участвующем узле(валидаторе) блокчейна. У smart-контракта, есть два вида функций: которые меняют состояние блокчейна - «change operations» и «view operations» - не меняют состояние блокчейна. Каждая транзакция имеет некоторое денежное обложение, которое измеряется в «Gas». Gas - это сборы на исполнение транзакции, данные единицы - детерминированы, то есть одна и та же транзакция всегда имеет одинаковое обложение в Gas. Стоимость Gas пересчитывается в зависимости от загруженности сети в блокчейне [16].

2.3 Этапы проекта

В рамках групповой курсовой работы была поставлена цель реализации discord-бота с функционалом NFT маркетплейса в NEAR protocol и сервисом генерации NFT, используя генеративно-состязательную сеть. Для реализации данной цели были выделены следующие этапы:

- Изучить теоретический базис связанный с NEAR Protocol(Лущ, Басалаев, Токкожин, Кусиденов)
- Реализовать smart-контракты(Басалаев):
 - Изучить язык Rust для написания smart-контрактов.
 - Изучить стандарт NFT токенов.
 - Реализовать core функционал NFT токенов - mint(создание) NFT и отправка между пользователями.
 - Реализовать enumeration функции - получение списка токенов, используя pagination.
 - Реализовать Approval Management внутри структуры NFT.
 - Поддержать Royalties - распределение доходов от продажи NFT среди нескольких аккаунтов в соотношении с долями.
 - Покрыть основную часть функционала NFT smart-контракта тестами.
 - Изучить существующие решения маркетплейс контрактов, подчеркнуть из них самое полезное.
 - Организовать функции покупки хранилища под продажу NFT токенов.
 - Написать функцию возвращения NEAR за неиспользуемое хранилище для продажи NFT токенов.

- Реализовать функцию выставления на продажу в маркетплейс smart-контракте.
- Добавить enumeration функции - получение списка продаваемых токенов, используя pagination.
- Реализовать изменение цены/отмену продажи для выставленного на маркетплейс NFT токена.
- Добавить возможность покупки продаваемых на маркетплейсе NFT токенов.
- Покрыть основную часть функционала маркетплейс smart-контракта тестами.
- Разработать discord-бота(Луц):
 - Изучить Javascript/Typescript;
 - Изучить основы работы с браузером через Javascript(сессионное/локальное хранилище браузера, класс window);
 - Изучить near-api-js и его код для дальнейшего его переписывания под функциональность discord;
 - Реализовать KeyStore[17] работающий через Redis[18];
 - Написать реализацию авторизации в Near Wallet[19] через discord-бота, который использует вышеописанный KeyStore;
 - Написать реализацию создания URL на подпись транзакции/транзакций(одна транзакция¹, один Action[21]; одна транзакция, несколько Action; несколько транзакций, несколько Action);
 - Создание «Профиля пользователя». Вызов осуществляется через slash-команду[22] или контекстное меню;
 - Реализация просмотра списка NFT, которыми владеет пользователь, которые продает пользователь, которые продаются на всем маркетплейсе. Вызовы осуществляются через контекстные меню, slash-команды, кнопки в профиле пользователя. Список выглядит по-разному в зависимости от количества NFT;
 - Поддержка покупки, продажи, отмены продажи NFT. Вызовы в виде кнопок при просмотре NFT списка;
 - Изучение децентрализованных распределенных хранилищ;
 - Реализация mint(создания) NFT с использованием децентрализованных распределенных хранилища;

¹) В данном контексте класс Transaction[20]

- Поддержка изменения цены NFT // пока что не сделано, но сделан метод в smart-контракте;
- Поддержать сервис с генеративно-состязательной сетью в discord-боте // пока что не сделано;
- Сделать docker образ для удобного деплоя discord-бота;
- Деплой discord-бота на облачный сервис(Кусиденов);
- Реализовать генеративно-состязательную сеть(Токкожин):
- Реализовать сервис с генеративно-состязательной сетью(Кусиденов):

2.4 Структура работы

Работа организована следующим образом. В разделе 3 дается обзор существующих на сегодняшний день маркетплейсов на NEAR Protocol и (что-то про ган). Раздел 4 описывает устройство и реализацию smart-контрактов NFT и маркетплейса. В 5 разделе идет описание трудностей и их решения в разработке discord-бота.

3 Обзор существующих работ и решений

3.1 Маркетплейсы

На данный момент существует большое количество NFT маркетплейсов: opensea[23], rarible[24], solanart[25]. Если брать маркетплейсы только на базе NEAR Protocol, тогда существуют такие примеры как: Paras[26], Mintbase[27]. Остановимся на них поподробнее.

3.1.1 Paras Paras является наиболее популярным, интерфейс взаимодействия представлен пользователю в веб-браузере по адресу paras.id. Для того, чтобы авторизоваться нужно использовать предоставит свой NEAR кошелек. Paras предоставляет огромное количество функций:

1. Создать NFT токен.
2. Выставить на продажу NFT токен.
3. Обновить цену выставленному на продажу NFT токenu.
4. Убрать с продажи выставленной NFT токен.
5. Уничтожить свой NFT токен.

6. Получить продаваемые NFT токены со следующей фильтрацией:

- (a) Фильтрация по содержимому токена(картинке) - пиксель арт, фотографии, иллюстрации и так далее.
- (b) Фильтрация по времени создания.
- (c) Фильтрация по максимальной цене.
- (d) Фильтрация по минимальной цене.

7. Выставить оффер на непродаваемый токен.

Smart-контракты Paras лежат в открытом доступе[28, 29].

Замечание. Обычно *smart-контракты DApps* принято выкладывать в открытый доступ, чтобы любой пользователь мог их посмотреть и полностью доверять сервису.

С точки зрения написания smart-контрактов Paras имеет абсолютно такую же структуру NFT smart-контракта, потому что они придерживаются стандарта [30](см. 4.1). Дополнительно они привязывают каждый токен к какой-то конкретной коллекции и не позволяют создавать токен без привязки к коллекции. Схема выглядит таким образом:

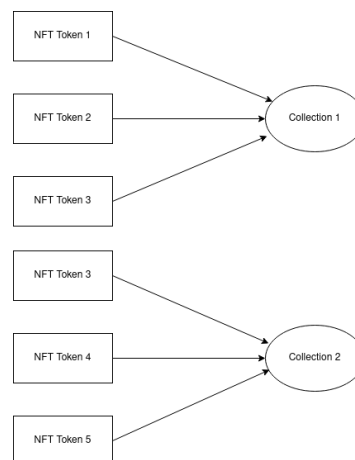


Рисунок 3.1. NFT токены и коллекции в paras

Smart-контракт маркетплейса paras предоставляет дополнительную функцию, как выставление оффера(предложения о покупке) на любой NFT токен. Эту функцию мы планируем позаимствовать в ближайшем будущем.

Paras, как и большинство маркетплейсов хранит медиа-файл и метаданные NFT на IPFS[31](см. 5.1.5). IPFS предоставляется сервисом fleek[32]. В качестве ссылки на медиа-файл и метаданные они хранят CID, а не полный URL, это связано с тем, что мINT NFT таким образом будет гораздо дешевле,

ведь хранение в NEAR, довольно дорогое(см. 5.1.4). URL восстанавливается с помощью вызова метода «nft_metadata» у NFT контракта для получения нужного шлюза(Листинг 3), а после CID подставляется в URL этого шлюза(Листинг 1), где при неуказании берется шлюз от cloudflare, который по опыту слабодоступен.

```
function buildMediaUrl(media, base_uri) {
  if (!media || media.includes('/://') || media.startsWith('data:image')) {
    return media;
  }
  if (base_uri) {
    return `${base_uri}/${media}`;
  }
  return `https://cloudflare-ipfs.com/ipfs/${media}`;
}
```

Листинг 1: Подстановка CID в URL у NEAR Wallet[33]

```
{
  spec: 'nft-1.0.0',
  name: 'Paras Collectibles',
  symbol: 'PARAS',
  icon: "data:image/svg+xml,%3Csvg width='1080' height='1080' viewBox='0 0 1080 1080' fill='none'
  ↪ xmlns='http://www.w3.org/2000/svg'%3E%3Crect width='1080' height='1080' rx='10'
  ↪ fill='%230000BA'/%3E%3Cpath fill-rule='evenodd' clip-rule='evenodd' d='M335.238 896.881L240
  ↪ 184L642.381 255.288C659.486 259.781 675.323 263.392 689.906 266.718C744.744 279.224 781.843
  ↪ 287.684 801.905 323.725C827.302 369.032 840 424.795 840 491.014C840 557.55 827.302 613.471
  ↪ 801.905 658.779C776.508 704.087 723.333 726.74 642.381 726.74H468.095L501.429
  ↪ 896.881H335.238ZM387.619 331.329L604.777 369.407C614.008 371.807 622.555 373.736 630.426
  ↪ 375.513C660.02 382.193 680.042 386.712 690.869 405.963C704.575 430.164 711.428 459.95
  ↪ 711.428 495.321C711.428 530.861 704.575 560.731 690.869 584.932C677.163 609.133 648.466
  ↪ 621.234 604.777 621.234H505.578L445.798 616.481L387.619 331.329Z' fill='white'/%3E%3C/svg%3E",
  base_uri: 'https://ipfs.fleek.co/ipfs',
  reference: null,
  reference_hash: null
}
```

Листинг 2: Структура при вызове «nft_metadata» у NFT контракта

Давайте рассмотрим структуру метаданных NFT (Листинг 3). Paras, хоть и поддерживает по стандарту NEP-171 поле «description», но хранит описание в метаданных NFT токена. Это аналогично тем же причинам, что и при хранении CID, а не полного URL в полях на медиа-файл и метаданные. Также они хранят название и идентификатор коллекции, создателя NFT, атрибуты и тип файла. Во многом наши метаданные будут подражать этой структуре.

```
{
  "description": "Proof of Attendance to events hosted by NEAR Gang Couture.",
  "collection": "Haute Gang - Collaborations",
  "collection_id": "haute-gang-collaborations-by-neargangcouturenear",
  "creator_id": "neargangcouture.near",
  "attributes": [
    { "trait_type": "Rarity", "value": "No Star" },
    { "trait_type": "Type", "value": "Mask" }
  ],
  "blurhash": "UqFtJxPWpdYDGJ${t2V[?[ICMyenPCxVobae",
  "mime_type": "image/jpeg"
}
```

Листинг 3: Структура метаданных NFT в Paras

3.1.2 Mintbase Mintbase является менее популярным маркетплейсом, однако он предоставляет гораздо больше категорий NFT, но все ключевые функции такие же. В качестве новых категорий выступают: 3D изображение, gif, профессиональные фотографии, аудиодорожки, произведения художников.

Smart-контракты Mintbase на половину открыты (некоторые в открытом доступе, некоторые нет)[34].

// TODO контракты

3.2 Генеративно-состязательные сети

4 Smart-контракты

В данной главе будет описываться строение и реализация smart-контрактов.

4.1 Структура NFT smart-контракта

В данной подглаве будет описываться строение NFT smart-контракта, написанного на языке Rust. Вся логика соответствует описанному стандарту NEP-171[30].

4.1.1 NEAR sdk Введем основные функции, структуры, декораторы, которые используются при написании smart-контрактов. Для этого необходим фреймворк near-sdk[9].

Атрибуты:

```
#[near_bindgen]                                /* генерирует smart-контракт,
↳ совместимый с блокчейном NEAR */
#[derive(BorshDeserialize, BorshSerialize)]    /* запоминает состояние контракта */
#[derive(PanicOnDefault)]                      /* не позволяет инициализировать
↳ контракт дефолтными значениями, нужен метод new с декоратором init */
#[payable]                                     /* помечает метод, который может
↳ принимать депозит */
```

Листинг 4: Near sdk framework attributes

Структуры:

```
use near_sdk::collections::{LazyOption, LookupMap, UnorderedMap, UnorderedSet};
LookupMap      /* Неупорядоченный словарь, который хранит свои значения в боре */
UnorderedMap   /* Итерируемый словарь, который хранит свои значения в боре */
UnorderedSet   /* Итерируемое множество объектов, которые хранятся в боре */
LazyOption     /* Структура, которая лениво инициализируется */
```

Листинг 5: Near sdk framework structures

Функции:

```
env::storage_byte_cost() /* стоимость хранения одного байта */
env::attached_deposit()  /* внесенный депозит */
env::predecessor_account_id() /* предыдущий аккаунт от которого прилетел cross-contract call
↳ или это мы сами, если мы первые в цепочке */
env::log_str()           /* написать лог */
env::prepaid_gas()       /* количество gas предоставленного для call другой функции */
```

Листинг 6: Near sdk framework functions

4.1.2 Core Functionality Опишем основные структуры и функции[35], которые используются в NFT контрактах.

```
pub type TokenId = String;

#[derive(BorshDeserialize, BorshSerialize, Serialize, Deserialize, Clone)]
#[serde(crate = "near_sdk::serde")]
pub struct NFTContractMetadata {
    pub spec: String,
    pub name: String,
    pub symbol: String,
    pub icon: Option<String>,
    pub base_uri: Option<String>,
    pub reference: Option<String>,
    pub reference_hash: Option<Base64VecU8>,
}
```

Листинг 7: NFT contract metadata

Структура контракта представляет из себя следующие поля:

1. spec - версия, является обязательным полем.
2. name - название контракта, является обязательным полем.
3. symbol - краткое название, является обязательным полем.
4. icon - иконка, которая будет отображаться вместе с контрактом(url).
5. base_uri - url, который ведет на надежное централизованное хранилище данных в reference.
6. reference - url на json с дополнительными данными(json должен располагаться в децентрализованном хранилище).
7. reference_hash - sha256 хэш от json на который ведет url в поле reference.

```
#[derive(BorshDeserialize, BorshSerialize, Serialize, Deserialize)]
#[serde(crate = "near_sdk::serde")]
pub struct TokenMetadata {
    pub title: Option<String>,
    pub description: Option<String>,
    pub media: Option<String>,
    pub media_hash: Option<Base64VecU8>,
    pub copies: Option<u64>,
    pub issued_at: Option<u64>,
    pub expires_at: Option<u64>,
    pub starts_at: Option<u64>,
    pub updated_at: Option<u64>,
    pub extra: Option<String>,
    pub reference: Option<String>,
    pub reference_hash: Option<Base64VecU8>,
}
```

Листинг 8: NFT contract token metadata

Структура метаданных токена состоит из следующих полей:

1. title - название NFT токена.
2. description - описание NFT токена.
3. media - ссылка на содержимое NFT токена, желательно, чтобы эта ссылка вела на децентрализованное хранилище.
4. media_hash - хэш от содержимого NFT токена, на которое ведет поле media.
5. copies - количество копий NFT токена.
6. issued_at - время, когда NFT токен был создан.
7. expires_at - время, когда время жизни NFT токена истекает.
8. starts_at - время, когда токен начал быть валидным.
9. extra - любые дополнительные данные.
10. reference - ссылка на json с дополнительной информацией о JSON.
11. reference_hash - sha256 хэш от содержимого на которое ведет ссылка в поле reference.

```
#[derive(BorshDeserialize, BorshSerialize)]
pub struct Token {
    pub owner_id: AccountId,
    pub next_approval_id: u64,
    pub approved_account_ids: HashMap<AccountId, u64>,
    pub royalty: HashMap<AccountId, u32>
}

#[derive(Serialize, Deserialize)]
#[serde(crate = "near_sdk::serde")]
pub struct JsonToken {
    pub token_id: TokenId,
    pub owner_id: AccountId,
    pub metadata: TokenMetadata,
    pub approved_account_ids: HashMap<AccountId, u64>,
    pub royalty: HashMap<AccountId, u32>
}
```

Листинг 9: NFT contract token structs

Структура NFT токена представляет из себя 3 связанные структуры:

1. TokenMetadata - метаданные токена, где каждое из полей является опциональным.
2. Token - для каждого токена образуется связь:
 - (a) owner_id - аккаунт владельца токена.
 - (b) approved_accounts_ids - словарь из доверенных аккаунтов, где значения является счетчик версий.
 - (c) next_approval_id - текущая версия токена.

(d) royalty - доля других аккаунтов, на получение денег с продажи токена.

3. JsonToken - endpoint структура, которая возвращается при работе с контрактом извне.

Теперь опишем структуру класса контракта, в котором хранятся созданные токены:

```
#[near_bindgen]
#[derive(BorshDeserialize, BorshSerialize, PanicOnDefault)]
pub struct Contract {
    pub owner_id: AccountId,
    pub tokens_per_owner: LookupMap<AccountId, UnorderedSet<TokenId>>,
    pub tokens_by_id: LookupMap<TokenId, Token>,
    pub token_metadata_by_id: UnorderedMap<TokenId, TokenMetadata>,
    pub metadata: LazyOption<NFTContractMetadata>,
}
```

Листинг 10: NFT contract struct

1. owner_id - владелец контракта, которые задается единственный раз при инициализации.

2. metadata - метаданные контракта, которые задаются единственный раз при инициализации.

3. tokens_per_owner - позволяет по аккаунту получить все токены, которыми владеет.

4. tokens_by_id - позволяет по TokenId получить структуру Token описанную выше.

5. tokens_metadata_by_id - позволяет по TokenId получить структуру TokenMetadata описанную выше.

Следующая функция из core functionality без которой нельзя осуществить никакой продажи - создание или mint NFT токена. Функция nft_mint принимает token_id, метаданные, владельца и royalties(про них будет рассказано во главе Royalties). Так как это payable функция, то пользователь должен будет внести депозит для хранения информации о добавляемом токене. Лишний депозит вернется пользователю обратно.

Псевдокод будет выглядеть следующим образом (Листинг 11):

```
#[payable]
func nft_mint(token_id, metadata, receiver_id, royalties) {
    /* Сохранить начальный storage_usage */
    start_storage_usage = env::storage_usage();

    /* Распаковать и положить royalties */
    royalty = АceptarRoyalties(royalties);

    /* Создать токен */
    token = CreateToken(metadata, royalties);

    /* Проверить, что такого token_id не существует */
    assert(!Exist(token_id));

    /* Добавить токен в необходимые структуры */
    contract.tokens_by_id.Insert(token_id, token);
    contract.token_metadata_by_id.Insert(token_id, metadata);
    add_token_to_owner(token.owner_id, token_id);

    /* Вернуть неиспользованный депозит */
    refund(env::storage_usage() - start_storage_usage);
}
```

Листинг 11: NFT token mint

Каждый пользователь может запросить на просмотр любой NFT токен с помощью view функции nft_token, указав в параметрах token_id. В качестве результата пользователь получит JsonToken структуру описанную выше или None, если такого токена не существует (Листинг 12).

```
func nft_token(token_id) -> Option<JsonToken> {
    if !Exist(token_id) {
        return None;
    } else {
        return ConstructJsonToken(token_id);
    }
}
```

Листинг 12: NFT nft_token

Последние функции из core functionality отвечают за передачу nft токена:

1. nft_transfer - отправить токен другому аккаунту.
2. nft_transfer_call - отправить токен другому аккаунту для выполнения какой-то услуги, то есть должна будет выполняться какая-то дополнительная логика на другом smart-контракте.
3. nft_on_transfer - дополнительная логика, которая должна исполниться в другом контракте после nft_transfer_call.

4. `nft_resolve_transfer` - функция, которая определяет нужно ли возвращать токен обратно или нет после `nft_on_transfer`.

С первой функция все ясно, она просто отправляет токен, а со второй лучше привести иллюстрацию:

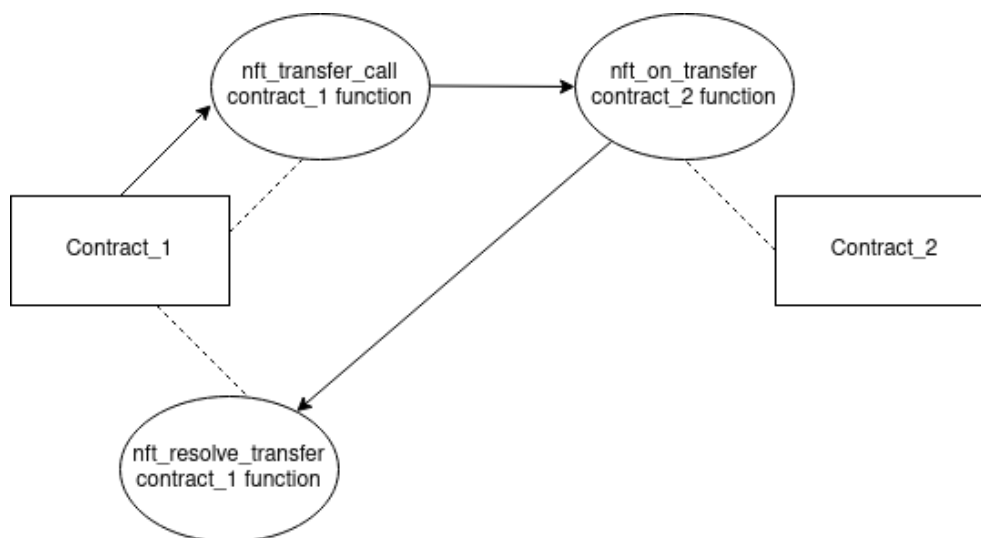


Рисунок 4.1. *nft_transfer_call*

Смоделируем пример, где мы хотим отправить из `contract_1` свой токен в другой контракт `contract_2` и выполнить в нем дополнительную сервисную логику (например `contract_2` это контракт маркетплейса, который должен будет выставить что-то на продажу). Тогда сервисная логика должна будет реализована в `nft_on_transfer`. Вызывать ее должен будет `nft_transfer_call` и завершать всю эту цепочку должна будет функция `nft_resolve_transfer`. Псевдокод функций будет выглядеть следующим образом:

```

func nft_on_transfer(sender_id, previous_owner_id, token_id, msg) -> Promise;

#[payable]
fn nft_transfer_call(receiver_id, token_id, approval_id, msg) -> PromiseOrValue<bool> {

    /* Сохраняем отправителя и копию токена до отправки */
    sender_id = env::predecessor_account_id();
    previous_token = own_contract.internal_transfer(sender_id, receiver_id, token_id, approval_id);

    /* Если отправитель не владелец, значит мы ему доверили наш токен, подробнее в главе approval
    ↪ managements */
    authorized_id = None;
    if sender_id != previous_token.owner_id {
        authorized_id = sender_id;
    }

    /* Вызываем nft_on_transfer на другом контракте, потом nft_resolve_transfer на своем */
    reciever_contract::nft_on_transfer(
        sender_id,
        previous_token.owner_id,
        token_id,
        msg,
        receiver_id,
    ).then(
        own_contract::nft_resolve_transfer(
            authorized_id,
            previous_token.owner_id,
            receiver_id,
            token_id,
            previous_token.approved_account_ids,
        )
    )
}

#[private]
fn nft_resolve_transfer(
    authorized_id,
    owner_id, receiver_id,
    token_id, approved_account_ids,
) -> bool {

    /* Передача произошла успешно */
    if IsSuccesfull() {
        true
    }

    /* Иначе возвращаем токен обратно владельцу */
    own_contract.internal_remove_token_from_owner(receiver_id, token_id);
    own_contract.internal_add_token_to_owner(owner_id, token_id);
    token.owner_id = owner_id;
    refund_approved_account_ids(receiver_id, token.approved_account_ids);
    token.approved_account_ids = approved_account_ids;
    own_contract.tokens_by_id.insert(token_id, token);

    false
}

```

Листинг 13: NFT nft_token

4.1.3 Enumeration Для удобного взаимодействия с контрактом, необходимо добавить больше view функций с pagination для просмотра NFT токенов[36]:

1. `nft_total_supply` - получить общее количество существующих токенов.
2. `nft_tokens` - получить существующие токены, используя pagination.
3. `nft_supply_for_owner` - получить общее количество существующих токенов для конкретного аккаунта.
4. `nft_token_for_owner` - получить существующие токены для конкретного аккаунта, используя pagination.

Псевдокод будет выглядеть следующим образом:

```
pub fn nft_total_supply(&self) -> U128 {
    self.token_metadata_by_id.len()
}

pub fn nft_tokens(
    &self, from_index: Option<U128>,
    limit: Option<u64>
) -> Vec<JsonToken> {
    self.token_metadata_by_id.keys()
        .skip(from_index as usize)
        .take(limit.unwrap_or(15) as usize)
        .map(|token_id| self.nft_token(token_id.clone()).unwrap())
        .collect()
}

pub fn nft_supply_for_owner(
    &self,
    account_id: AccountId,
) -> U128 {
    if Exist(account_id) {
        self.tokens_per_owner.get(&account_id).len()
    } else {
        0
    }
}

pub fn nft_tokens_for_owner(
    &self,
    account_id: AccountId,
    from_index: Option<U128>,
    limit: Option<u64>,
) -> Vec<JsonToken> {
    if Exist(account_id) {
        tokens.iter()
            .skip(from_index as usize)
            .take(limit.unwrap_or(15) as usize)
            .map(|token_id| self.nft_token(token_id.clone()).unwrap())
            .collect()
    } else {
        return vec![];
    }
}
```

4.1.4 Approval Management Необходимо добавить функционал передачи своего токена другим аккаунтом от своего имени[37]. Для этого будет храниться список доверенных аккаунтов(`approved_account_ids`). Также структура токена хранит `next_approval_id`, который изначально равен 0 и увеличивается на единицу при каждом новом добавленном доверенном аккаунте.

Рассмотрим пример, где `account_1` решил создать токен, тогда у него будет следующая структура:

```
Token: {
  owner_id: account_1
  approved_accounts_ids: {}
  next_approval_id: 0
}
```

Если он решит добавить `account_2`, `account_3`, как доверенные тогда структура станет следующей:

```
Token: {
  owner_id: account_1
  approved_accounts_ids: {
    account_2: 0,
    account_3: 1
  }
  next_approval_id: 2
}
```

Счетчик `next_approval_id` необходим, чтобы не случилось случая, когда новый владелец токена решил добавить доверенный аккаунт, который был до этого. Такие случаи могут испортить всю логику на других smart-контрактах. Подробнее такие крайние случаи описаны в стандарте[37].

Approval Management не добавляет новых внешних view функций или payable функций, а просто вносит некоторую дополнительную логику проверки в существующие функции из секции Core Functionality.

4.1.5 Royalties Последнее чего требует стандарт - распределение прибыли от продажи NFT или от любой другой логики, которая будет возвращать NEAR среди нескольких аккаунтов в зависимости от долей[38]. Для этого у нас есть поле `royalty` в структуре Token, которая отобразит пары в соответствующие доли. Сумма всех долей должна быть равна 10.000.

Также добавятся две новые функции:

1. `nft_payout` - получить распределение баланса в зависимости от долей для конкретного `token_id`.
2. `nft_transfer_payout` - совершить перевод токена и вернуть распределение баланса от долей.

Псевдокод будет выглядеть следующим образом:

```
fn nft_payout(&self, token_id: TokenId, balance: u128, max_len_payout: u32) -> Payout {
    /* Проверить, что токен существует */
    assert(ExistToken(token_id))

    /* Достать структуру токена */
    let token = self.tokens_by_id.get(&token_id);
    let mut current_sum = 0;
    let mut res = Payout {
        payout: HashMap::new()
    };

    /* Посчитать доли других аккаунтов */
    for (k, v) in token.royalty.iter() {
        let key = k.clone();
        if key == token.owner_id {
            continue;
        }
        res.payout.insert(
            key,
            calc_payout(*v, balance)
        );
        current_sum += *v;
    }
    /* Посчитать свою долю */
    res.payout.insert(
        token.owner_id,
        calc_payout(10000 - current_sum, balance)
    );
    res
}

#[payable]
fn nft_transfer_payout(
    &mut self,
    receiver_id: AccountId,
    token_id: TokenId,
    approval_id: u64,
    memo: Option<String>,
    balance: u128,
    max_len_payout: u32,
) -> Payout {
    /* Отправить токен */
    let sender_id = env::predecessor_account_id();
    let prev_token = self.internal_transfer(sender_id, receiver_id, token_id, approval_id, memo);

    let mut current_sum = 0;
    let mut result = Payout {
        payout: HashMap::new()
    };

    /* Посчитать доли других аккаунтов */
    for (k, v) in prev_token.royalty.iter() {
        let key = k.clone();
        if key == prev_token.owner_id {
            continue;
        }
        result.payout.insert(key, calc_payout(*v, balance));
        current_sum += *v;
    }
    /* Посчитать свою долю */
    result.payout.insert(prev_token.owner_id, calc_payout(10000 - current_sum, balance));
}
```

```

    result
}

```

4.2 Структура маркетплейс smart-контракта

В данной главе будет описано строение маркетплейс smart-контракта. Контракт маркетплейса уже не подчиняется никакому стандарту и может быть реализован разными способами.

4.2.1 Core functionality Начнем с функций, которые должны быть доступны пользователю:

1. Выставить NFT токен на продажу.
2. Обновить цену своего выставленного на продажу NFT токена.
3. Убрать с продажи свой выставленный до этого NFT токен.
4. Получить список выставленных на продажу NFT токенов.
5. Купить выставленный на продажу NFT токен.

Заметим, что на маркетплейс должны уметь выставлять токены нескольких NFT контрактов, потому что все они стандартизированны. То есть пользователи могут покупать/продавать токены абсолютно разных NFT контрактов.

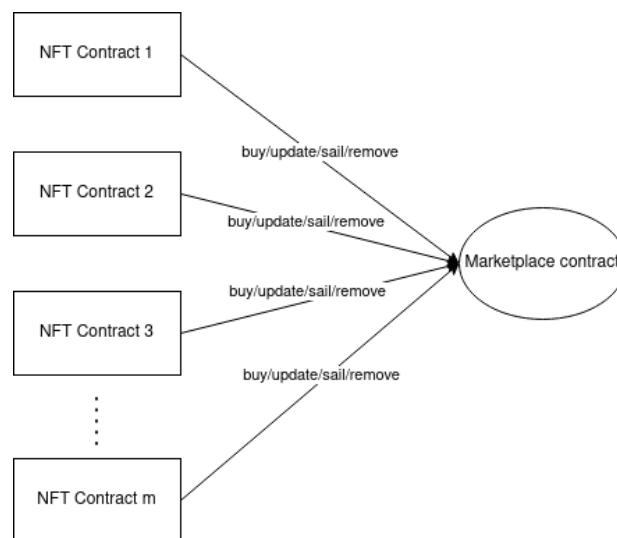


Рисунок 4.2. Marketplace core functionality

Опишем структуру маркетплейс контракта:

```

/* Так как выставить токен могут с разных контрактов, удобно будет соединить их в одной строке */
/* ContractAndTokenId = contract ID + DELIMITER + token ID */
pub type ContractAndTokenId = String;

/* Цена токенов будет в YoctoNear */

```



```

pub type SalePriceInYoctoNear = U128;

pub type TokenId = String;

/* Структура NFT токена выставленного на продажу */
#[derive(BorshDeserialize, BorshSerialize, Serialize, Deserialize)]
#[serde(crate = "near_sdk::serde")]
pub struct Sale {
    /* Владелец NFT токена */
    pub owner_id: AccountId,

    /* Значение этого поля обоснован в главе Approval Management */
    pub approval_id: u64,

    /* nft_contract_id с которого был выставлен NFT токен */
    pub nft_contract_id: String,

    /* Идентификатор выставленного токена */
    pub token_id: String,

    /* Цена */
    pub sale_conditions: SalePriceInYoctoNear,
}

/* Структура контракта */
#[near_bindgen]
#[derive(BorshDeserialize, BorshSerialize, PanicOnDefault)]
pub struct Contract {
    /* Владелец контракта */
    pub owner_id: AccountId,

    /* Выставленные на продажу токены по ContractAndTokenId */
    pub sales: UnorderedMap<ContractAndTokenId, Sale>,

    /* Выставленные на продажу ContractAndTokenId по конкретному аккаунту */
    pub by_owner_id: LookupMap<AccountId, UnorderedSet<ContractAndTokenId>>,

    /* Выставленные на продажу токены по конкретному аккаунту */
    pub by_nft_contract_id: LookupMap<AccountId, UnorderedSet<TokenId>>,

    /* Внесенная сумма на хранение nft токена */
    /* Смысл данной структуры будет обоснован позже */
    pub storage_deposits: LookupMap<AccountId, Balance>,
}

```

Когда пользователь хочет выставить на продажу NFT токен, он должен вызвать `nft_approve` у своего NFT контракта, чтобы добавить аккаунт маркетплейса в доверенные аккаунты, тогда на контракте маркетплейса вызовется метод `nft_on_approve`, который добавит токен на продажу. В результате, когда другой пользователь захочет купить токен, то маркетплейс сможет легко перевести его новому владельцу, потому что он является доверенным аккаунтом для продаваемого токена.

На иллюстрации будет приведен пример, где пользователь выставляет на продажу два токена с двух разных NFT контрактов на одном маркетплейс контракте.

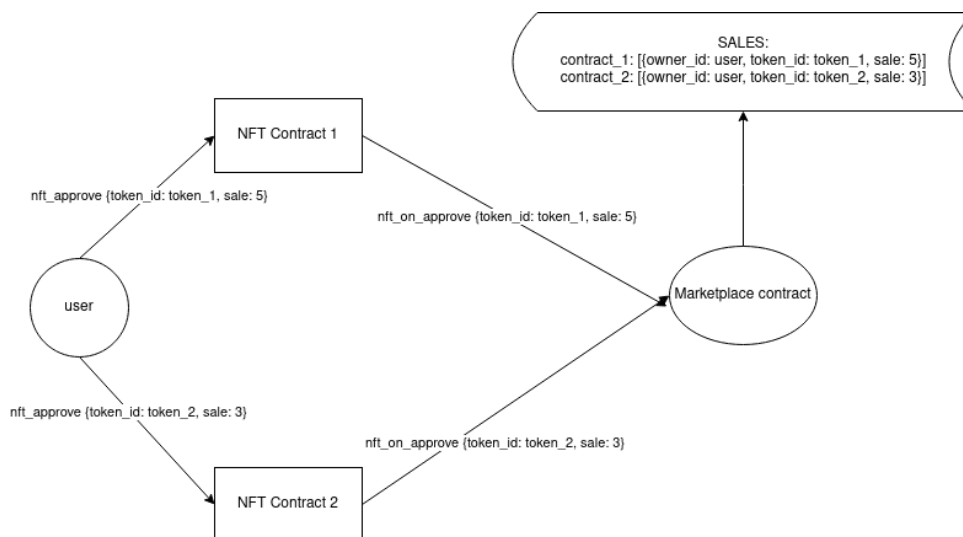


Рисунок 4.3. Marketplace contract sell

Псевдокод будет выглядеть следующим образом:

```

fn nft_on_approve(
    &mut self,
    token_id: TokenId,
    owner_id: AccountId,
    approval_id: u64,
    msg: String,
) {
    /* NFT контракт с которого был вызвана продажа */
    let nft_contract_id: AccountId = env::predecessor_account_id();

    /* Аккаунт пользователя, который подписал контракт */
    let signer_id: AccountId = env::signer_account_id();

    assert(owner_id == signer_id)

    /* Считаем сколько нужно на хранилище и сколько внесено */
    let paid_storage: u128 = self.storage_deposits.getPaidStorage(signer_id);
    let required_storage: u128 = CalcRequiredStorage();

    /* Проверяем, что денег на хранение достаточно */
    assert(paid_storage > required_storage);

    /* Sale conditions take from msg filed, if user don't fill msg it will panic */
    let sale_price = GetSalePrice(msg);

    /* Добавляем покупку в необходимые структуры */
    let contract_and_token_id: String = nft_contract_id + '!' + token_id;
    self.sales.InsertNewSale(
        contract_and_token_id, owner_id, approval_id, nft_contract_id, token_id, sale_price
    );

    self.by_owner_id.InsertNewSale(
        contract_and_token_id, owner_id, approval_id, nft_contract_id, token_id, sale_price
    );

    self.by_nft_contract_id.InsertNewSale(
        owner_id, approval_id, nft_contract_id, token_id, sale_price
    );
}
  
```

Так как мы делаем cross-contract call между двумя контрактами, тогда определить необходимые средства на хранения продаваемого NFT токена выглядит проблематичным. Поэтому пользователь должен будет сам покупать хранилище и сам его освобождать, когда его токены продались и место освободилось. Именно для этого необходимо поле `storage_deposits` в контракте. Чтобы внести near под хранение используется функция `storage_deposit`, а для вывода near за неиспользуемое место `storage_withdraw`. Логика их кажется тривиальной, поэтому псевдокод приводиться не будет.

Изменение цены и отмена продажи, тоже выглядят достаточно тривиальными, напишем короткий псевдокод:

```
#[payable]
pub fn remove_sale(&mut self, nft_contract_id: AccountId, token_id: String) {
    /* Проверим, что владелец токена пытается его убрать с продажи */
    assert (self.sales.OwnerByToken(token_id) == env::predecessor_account_id());

    /* Удаляем продажу из структур контракта */
    contract_and_token_id = nft_contract_id + '.' + token_id;
    self.sales.RemoveByToken(token_id);
    self.by_owner_id.RemoveByContractAndToken(owner_id, contract_and_token_id);
    self.by_nft_contract_id.RemoveByContractAndToken(contract_and_token_id, token_id);
}

#[payable]
pub fn update_price(
    &mut self,
    nft_contract_id: AccountId,
    token_id: String,
    price: U128,
) {
    /* Проверим, что владелец токена пытается его убрать с продажи */
    assert (self.sales.OwnerByToken(token_id) == env::predecessor_account_id());

    /* Обновим цену продажи в структурах контракта */
    contract_and_token_id = nft_contract_id + '.' + token_id;
    self.sales.UpdateByToken(token_id, price);
}
```

Последний пункт это покупка NFT токена.

4.2.2 Enumeration Для того, чтобы удобно взаимодействовать с маркетплейсом контрактом были добавлены несколько view функций, которые позволяют выгружать продаваемые NFT.

1. `get_supply_sales` - получить суммарное количество выставленных токенов.
2. `get_supply_by_owner_id` - получить суммарное количество выставленных токенов за определенным пользователем.

3. `get_supply_by_nft_contract_id` - получить суммарное количество выставленных токенов за определенным NFT контрактом.

4. `get_sales_by_nft_contract_id` - получить выставленные на продажу токены за определенным NFT контрактом, используя pagination.

5. `get_sales_by_owner_id` - получить выставленные на продажу токены за определенным пользователем, используя pagination.

6. `get_sale` - получить определенный продаваемый токен.

Псевдокод данных функций будет выглядеть следующим образом:

```
pub fn get_supply_sales(
    &self,
) -> u64 {
    self.sales.len()
}

pub fn get_supply_by_owner_id(
    &self,
    account_id: AccountId,
) -> u64 {
    let owner_id = self.by_owner_id.get(&account_id);

    if let Some(owner_id) = owner_id {
        owner_id.len()
    } else {
        0
    }
}

pub fn get_sales_by_owner_id(
    &self,
    account_id: AccountId,
    from_index: Option<u128>,
    limit: Option<u64>,
) -> Vec<Sale> {
    let owner_id = self.by_owner_id.get(&account_id);

    let sales = if let Some(owner_id) = owner_id {
        owner_id
    } else {
        return vec![];
    };

    sales.iter()
        .skip(from_index)
        .take(limit)
        .map(|token_id| self.sales.get(&token_id).unwrap())
        .collect()
}

pub fn get_supply_by_nft_contract_id(
    &self,
    nft_contract_id: AccountId,
) -> u64 {
    let nft_contract_id = self.by_nft_contract_id.get(&nft_contract_id);

    if let Some(nft_contract_id) = nft_contract_id {
        nft_contract_id.len()
    } else {
        0
    }
}
```

```

    0
  }
}

pub fn get_sales_by_nft_contract_id(
  &self,
  nft_contract_id: AccountId,
  from_index: Option<u128>,
  limit: Option<u64>,
) -> Vec<Sale> {
  let by_nft_contract_id = self.by_nft_contract_id.get(&nft_contract_id);

  let sales = if let Some(by_nft_contract_id) = by_nft_contract_id {
    by_nft_contract_id
  } else {
    return vec![];
  };

  sales.iter()
    .skip(from_index)
    .take(limit)
    .map(|token_id| self.sales.get(&format!("{}", nft_contract_id, '.', token_id)).unwrap())
    .collect()
}

pub fn get_sale(&self, nft_contract_token: ContractAndTokenId) -> Option<Sale> {
  self.sales.get(&nft_contract_token)
}

```

5 Discord-бот

5.1 Взаимодействие с блокчейнами

В данной главе описано ядровое устройство discord-бота: описание работы near-api-js и его переписывание под устройство Discord, устройство метаданных NFT-токена, работа с децентрализованным распределенным хранилищем.

5.1.1 Аккаунты и access keys в NEAR Для понимания взаимодействия требуются минимальные знания об аккаунтах и access keys.

Аккаунты в NEAR[39] устроены так, что они имеют человеко-читаемый ID в отличие от большинства других блокчейнов, где обычно используется некоторый hash(Рисунок 5.1). Длина логина от 2 до 64 символов и содержит в конце суффикс обозначающий сеть блокчейна. Аккаунт может создавать под-аккаунты, которые по своему функционалу ничем не отличаются от обычного аккаунта. Данные подаккаунты решают проблему деплоя контрактов: на один

Ethereum Wallet

Public Identifier

- Public Key (ex. 0x123...)

Secret Key

- Private Key (ex. 0x456...)

Characteristics

- Private key gives full access
- Account doesn't have to be "created" via a transaction

**NEAR Account**

Public Identifier

- Account Id (ex. canaan)

Multiple Keypairs w/ permissions

- {Pub, Priv} (full access key)
- {Pub, Priv} (contract access key)

Characteristics

- Permission based keypairs
- Account ID must be created via a blockchain transaction

Рисунок 5.1. Сравнение аккаунтов Ethereum и NEAR

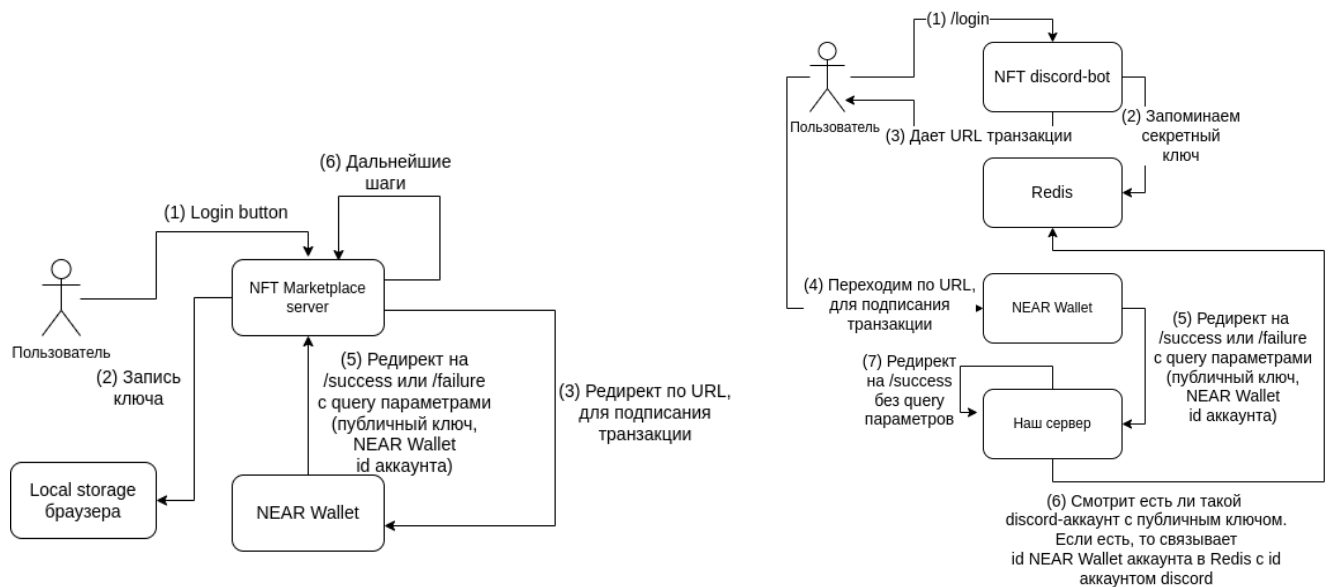
аккаунт можно развернуть только один smart-контракт, id аккаунта и будет значить, какой smart-контракт требуется.

Определение. В NEAR, как и во всех блокчейнах есть несколько сетей: *mainnet*, *testnet* и так далее. *Mainnet* - главная(продакшн) сеть. *Testnet* используется для тестирования сервисов.

Каждый аккаунт имеет создавать множество, которые в NEAR называются access keys. Существует два типа access keys: FullAccess и FunctionCall. Первый дает полный доступ, второй вид ключа уникален и дает разрешения только на подписание функций контрактов. В нашем сервисе не будет использоваться FullAccess access key из-за ненужности.

5.1.2 Авторизация в NEAR Wallet Авторизация в NEAR Wallet играет роль связывания аккаунта кошелька и пользователя, то есть фактически нам говорит, что у этого пользователя есть этот аккаунт. Наверное, стоит отметить, что, если пользователь авторизовался в NEAR Wallet в нашем сервисе, то от его лица можно вызывать методы контракта, на котором была подписана транзакция(пока не закончатся GAS, которые были указаны при подписании транзакции), к которым не нужно вложение депозита. Данный функционал нам не потребуется, так как у нас либо «view operations» в контрактах, либо «payable change operations».

На маркетплейсах в виде сайта вся авторизация происходит на client-side стороне: создается пара ключей(access key) - публичный и секретный; секретный ключ сохраняется в local storage браузера; подписывается транзакция и в последствии этот ключ будет использован сайтом, для подтверждения авторизации(Рисунок 5.2а). Сценарий авторизации в discord-бот различается, ведь



(a) Авторизация в NEAR Wallet в near-api-js (b) Авторизация в NEAR Wallet в discord-боте

Рисунок 5.2

ключ должен храниться на стороне сервера (Рисунок 5.2b). Для этого были необходимо реализовать KeyStore, который взаимодействует с какой-нибудь базой данных на стороне сервера. На роль БД было принято использовать Redis. Был написан KeyStore, который взаимодействует с Redis и функционал, который возвращает URL, а не редеректит пользователя. При любом взаимодействии с ботом, проверяется авторизован ли пользователь, если он не авторизован, то ему предлагается кнопка с URL на подписание транзакции авторизации.

Определение. *Storage* — интерфейс веб API, который позволяет добавлять, изменять, удалять элементы данных, которые представляются в виде ключа и значения[40]. В веб API есть два объекта, которые используют интерфейс storage: session storage, local storage[41]. Session storage хранит данные до закрытия браузера, когда как данные в local storage не имеют ограничения по времени и могут быть удалены только намерено.

Определение. *KeyStore*[17] — это класс, который хранит ключи, для подписей транзакций. Их 4 вида: BrowserLocalStorageKeyStore, InMemoryKeyStore, MergeKeyStore, UnencryptedFileSystemKeyStore. BrowserLocalStorageKeyStore пользуется локальным хранилищем браузера для записи, изменения, просмотра значений по ключу. InMemoryKeyStore хранит все в оперативной памяти, используется для тестирования. MergeKeyStore используется для объедине-

```

{
  token_id: 'chopik.testnet.1652636744470',
  owner_id: 'chopik.testnet',
  approved_account_ids: { 'papamsmarket.pojaleesh.testnet': 2 },
  royalty: { 'chopik.testnet': 10000 },
  metadata: {
    title: 'City',
    description: null,
    media:
      ↪ 'https://bafybeiahhurffoxjubs42l7bl3jjc5zk5vrafiiijcxkhex2ukjm3zsbbti.ipfs.dweb.link/f',
    media_hash: null,
    copies: null,
    issued_at: null,
    expires_at: null,
    starts_at: null,
    updated_at: null,
    extra: null,
    reference:
      ↪ 'https://bafybeiahhurffoxjubs42l7bl3jjc5zk5vrafiiijcxkhex2ukjm3zsbbti.ipfs.dweb.link/m',
    reference_hash: null
  }
}

```

Листинг 14: Структура получаемого NFT

ния множества *KeyStore*. *UnencryptedFileSystemKeyStore* хранит все на диске в виде JSON файла, используется в *near cli*[42].

5.1.3 Вызовы методов у контрактов Вызовы методов у smart-контрактов — это то, чем бот занимается большую часть времени.

5.1.4 Структура получаемого NFT и его метаданных На данный момент, при создании NFT в структуре(Листинг 14) используются следующие поля: «token_id», «owner_id», «royalty», «metadata.title», «metadata.media», «metadata.reference». «metadata.description» не используется, все описание NFT хранится в метаданных в децентрализованном хранилище в целях экономии оплаты за хранение. В «metadata.media», «metadata.reference» хранятся полные URL до медиа-файла и метаданных. В ближайшее время, планируется хранить только CID и только в поле media, так как медиа-файл и метаданные хранятся в одной директории, то для их определения достаточно одного CID(см. 5.1.5).

Структура метаданных(Листинг 15) полностью аналогично структуре метаданных в Paras, так как эта структура оправдана своей функциональностью. Только в отличие от Paras нет полей: «collection», «collection_id», «blurhash», «mime_type». «collection», «collection_id» нет, потому что на данный момент мы не поддерживаем коллекции NFT. «mime_type», «blurhash» из-за ненужности.


```

{
  "description": "Cool city",
  "creator_id": "chopik.testnet",
  "attributes": [
    { "trait_type": "Time", "value": "Night" },
    { "trait_type": "Color", "value": "Blue" }
  ]
}

```

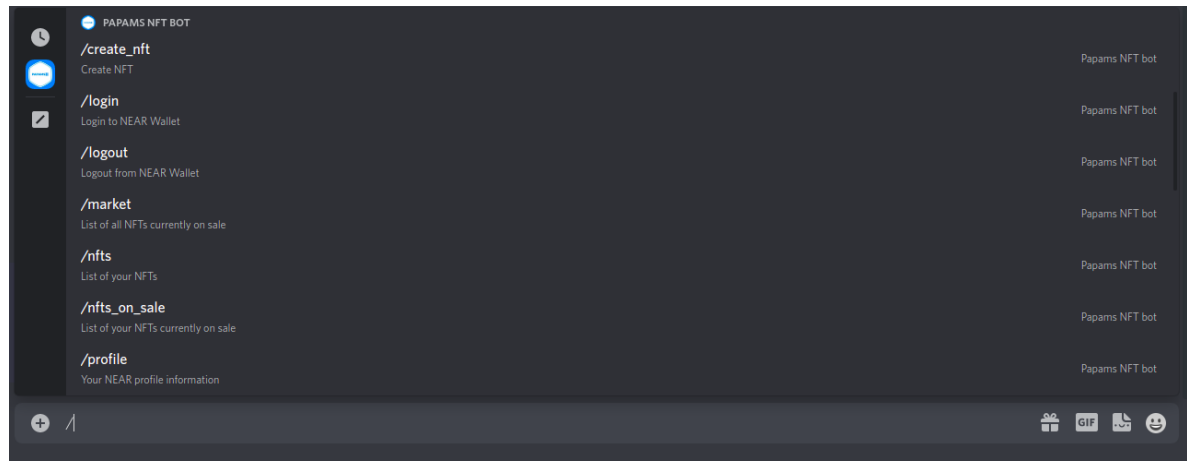
Листинг 15: Структура метаданных NFT в децентрализованном хранилище

5.1.5 Децентрализованное хранилище данных Обычно для хранения метаданных и медиа-объекта используется другой блокчейн специализированный под хранение. Связанно это с тем, что при деплое smart-контракта пользователь платит в NEAR за хранение байт, которые хранит контракт, используя механизм, который называется storage staking. На основе storage stacking есть и атаки на smart-контракты, например «million cheap data additions» - злоумышленник добавляет огромное количество бесполезных данных при вызове методов, из-за этого в наших контрактах данное обложение в контрактах оплачивает пользователь, которому хочется минимизировать свои растраты на операции. Из-за этого лучше стратегий для хранения объемных файлов(в основном таковыми являются медиа-файлы) это хранить данные в off-chain. Популярным решением является IPFS, при котором любой набор данных представляется удобным адресом - CID. В роли сервиса предоставляющего хранилище был выбран NFT.Storage[43] — бесплатное децентрализованное хранилище NFT на IPFS[31] и Filecoin[44]. NFT.Storage предоставляет HTTP API для взаимодействия с хранилищем и обертку на Javascript.

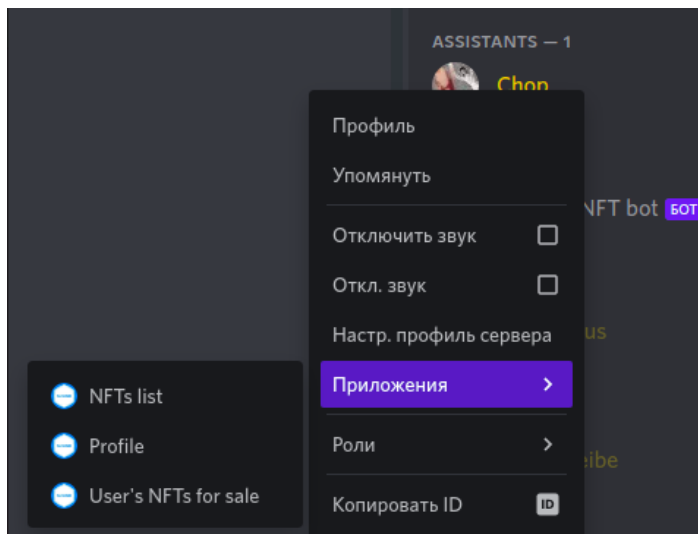
Замечание. Цена на storage staking устанавливается сетью блокчейна, на данный момент это 1 NEAR за 100 КБ.

Определение. IPFS(InterPlanetary File System) - это протокол распределенной системы обмена файлов. При добавлении файла в IPFS, он делится на маленькие куски, криптографически хэшируется и отдается уникальный фингерпринт, который называется CID(Content identifier) [31].

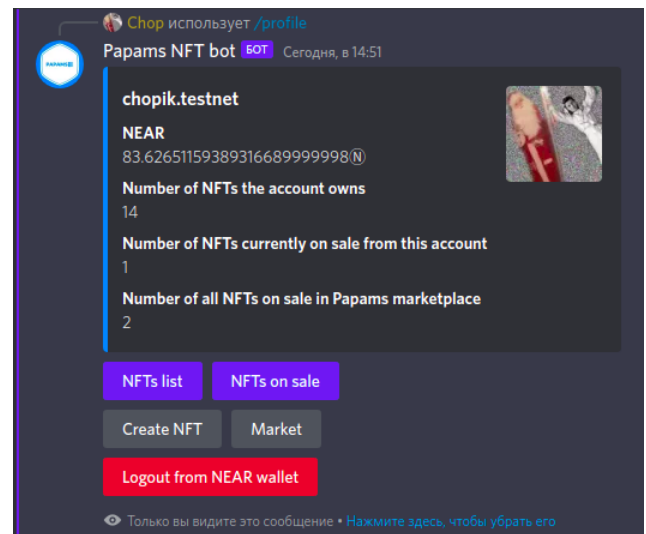
Замечание. Для того, чтобы построить стимулирующий слой для сохранения данных в IPFS существует Filecoin. Filecoin - это децентрализованное сетевое хранилище. Filecoin и IPFS - это два разных протокола, взаимодействующие друг друга. Когда как IPFS позволяет пользователям хранить, запрашивать и передавать друг другу данные, в то время, как Filecoin предназначен для обеспечения системы постоянного хранения.



(a) Slash-команды бота



(b) Контекстные меню бота



(c) Профиль пользователя

Рисунок 5.3. Точки входа пользователя

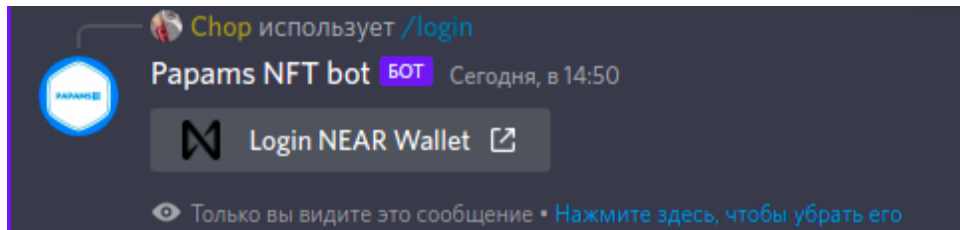
Замечание. Изначально использовался сервис *web3storage*[45] для хранения, но он отличился медленной загрузкой, получением файлов, из-за чего появилась необходимость в смене на *NFT.Storage*.

5.2 Пользовательский интерфейс

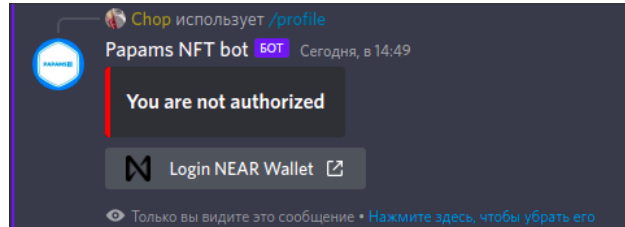
5.2.1 Точки входа

6 Генеративно-состязательная сеть

// Текст будет написан Токкожиным Арсеном



(a) Намеренная авторизация



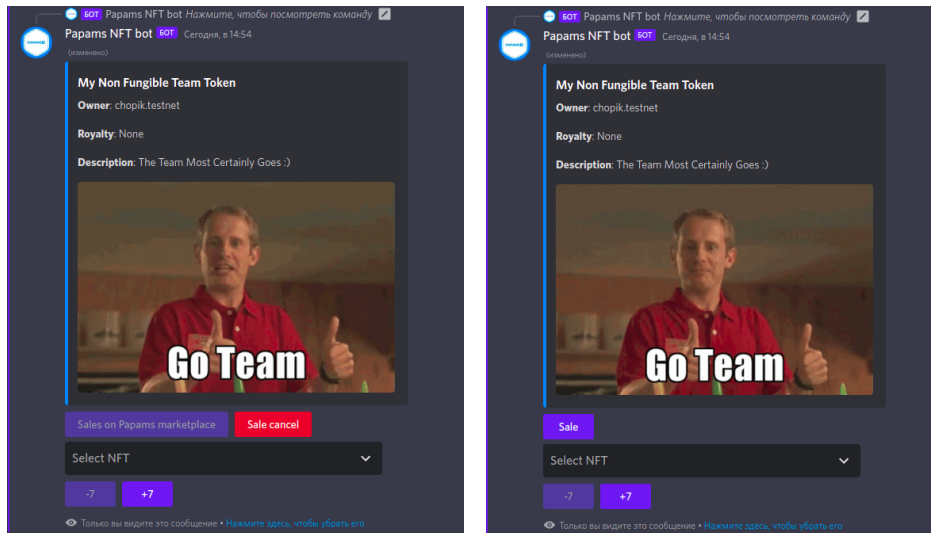
(b) Предложение об авторизации

Рисунок 5.4. Авторизация пользователя

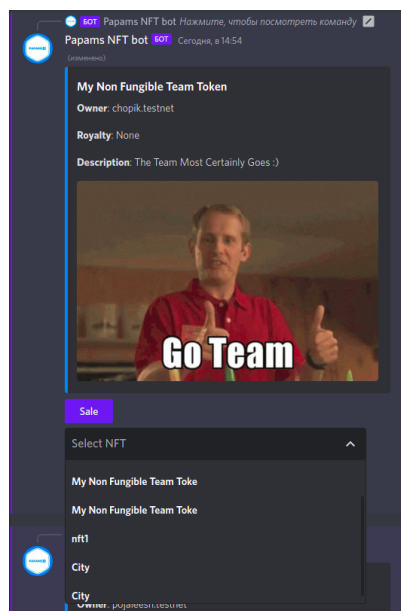
7 Сервис с генеративно-состязательной сетью

// Текст будет написан Кусиденовым Адильханом

8 Выводы и результаты



(a) Slash-команды бота (b) Контекстные меню бота



(c) Профиль пользователя

Рисунок 5.5. Точки входа пользователя

9 Список источников

- [1] Theodosios Mourouzis и Chrysostomos Filipou. “The Blockchain Revolution: Insights from Top-Management”. в: *CoRR* abs/1712.04649 (2017). arXiv: 1712.04649. URL: <http://arxiv.org/abs/1712.04649>.
- [2] Matthieu Nadini и др. “Mapping the NFT revolution: market trends, trade networks, and visual features”. в: *Scientific Reports* 11.1 (окт. 2021). ISSN: 2045-2322. DOI: 10.1038/s41598-021-00053-8. URL: <http://dx.doi.org/10.1038/s41598-021-00053-8>.

- [3] *NEAR Protocol*. URL: <https://near.org/>.
- [4] ILLIA POLOSUKHIN. *Thresholded proof of stake*. апр. 2019. URL: <https://near.org/blog/thresholded-proof-of-stake/>.
- [5] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: <http://www.bitcoin.org/bitcoin.pdf>.
- [6] Bina Ramamurthy. *Blockchain in Action*. S.l: Manning Publications, 2020. ISBN: 9781617296338.
- [7] Solana Foundation. *Introduction*. URL: <https://spl.solana.com/>.
- [8] NEAR Protocol. *Transaction*. URL: <https://docs.near.org/docs/concepts/transaction>.
- [9] NEAR Protocol. *Near/near-sdk-rs: Rust library for writing near Smart Contracts*. URL: <https://github.com/near/near-sdk-rs>.
- [10] NEAR Protocol. *Near/near-sdk-as: AssemblyScript library for writing near Smart Contracts*. URL: <https://github.com/near/near-sdk-as>.
- [11] NEAR Protocol. *near-api-js (JavaScript library)*. URL: <https://docs.near.org/docs/api/javascript-library>.
- [12] discord.js. *discord.js guide, buttons*. URL: <https://discordjs.guide/interactions/buttons.html#building-and-sending-buttons>.
- [13] discord.ts. *discord.ts official documentation, context menu*. URL: <https://discord-ts.js.org/docs/decorators/gui/context-menu/>.
- [14] discord.js. *discord.js Guide, select menus*. URL: <https://discordjs.guide/interactions/select-menus.html#building-and-sending-select-menus>.
- [15] discord.js. *discord.js guide, modals*. URL: <https://discordjs.guide/interactions/modals.html#building-and-responding-with-modals>.
- [16] NEAR Protocol. *Introduction, Thinking in gas*. URL: <https://docs.near.org/docs/concepts/gas#thinking-in-gas>.
- [17] NEAR Protocol. *Class KeyStore*. URL: https://near.github.io/near-api-js/classes/key_stores_keystore.keystore.html.
- [18] Redis Ltd. *Redis*. URL: <https://redis.io/>.
- [19] Roketo Labs LTD. *Near wallet*. URL: <https://wallet.near.org/>.

- [20] NEAR Protocol. *Class transaction*. URL: <https://near.github.io/near-api-js/classes/transaction.transaction-1.html>.
- [21] NEAR Protocol. *Class action*. URL: <https://near.github.io/near-api-js/classes/transaction.action.html>.
- [22] discord.js. *discord.js Guide, slash commands*. URL: <https://discordjs.guide/interactions/slash-commands.html#registering-slash-commands>.
- [23] Inc OpenSea Ozone Networks. *OpenSea, the largest NFT Marketplace*. URL: <https://opensea.io/>.
- [24] Inc Rarible. *NFT Marketplace*. URL: <https://rarible.com/>.
- [25] Solanart. *Solanart - discover, collect and trade nfts*. URL: <https://www.solnart.com/>.
- [26] Paras. *NFT Marketplace for digital collectibles on near*. URL: <https://paras.id/>.
- [27] Mintbase. *NFT Marketplace; Toolkit*. URL: <https://www.mintbase.io/>.
- [28] ParasHQ. *paras-nft-contract*. URL: <https://github.com/ParasHQ/paras-nft-contract>.
- [29] ParasHQ. *paras-nft-contract*. URL: <https://github.com/ParasHQ/paras-marketplace-contract>.
- [30] NEAR NFT Standarts. 2022. URL: <https://nomicon.io/Standards/Tokens/NonFungibleToken/Core>.
- [31] Protocol Labs. *IPFs powers the distributed web*. URL: <https://ipfs.io/>.
- [32] fleek. URL: <https://fleek.co/>.
- [33] NEAR Wallet. *Near-wallet/nonfungibletokens.js at 22b76a96b2ac71d1b1ca4f5acb85e79643cd8ef7 · near/near-wallet*. URL: <https://github.com/near/near-wallet/blob/22b76a96b2ac71d1b1ca4f5acb85e79643cd8ef7/packages/frontend/src/services/NonFungibleTokens.js#L101>.
- [34] Mintbase. *Mintbase/mintbase-core: Mintbase core NFT store factory contracts*. URL: <https://github.com/Mintbase/mintbase-core>.
- [35] *core-functionality*. URL: <https://nomicon.io/Standards/Tokens/NonFungibleToken/Core>.

- [36] *enumeration-standard*. URL: <https://nomicon.io/Standards/Tokens/NonFungibleToken/Enumeration>.
- [37] *approval-standard*. 2022. URL: <https://nomicon.io/Standards/Tokens/NonFungibleToken/ApprovalManagement>.
- [38] *royalty-standard*. 2022. URL: <https://nomicon.io/Standards/Tokens/NonFungibleToken/Payout>.
- [39] NEAR Protocol. *Account*. URL: <https://docs.near.org/docs/concepts/account>.
- [40] *Storage - интерфейсы веб API: MDN*. URL: <https://developer.mozilla.org/ru/docs/Web/API/Storage>.
- [41] *Window.localStorage - Интерфейсы веб API: MDN*. URL: <https://developer.mozilla.org/ru/docs/Web/API/Window/localStorage>.
- [42] NEAR Protocol. *Near cli*. URL: <https://docs.near.org/docs/tools/near-cli>.
- [43] *Free decentralized storage and bandwidth for nfts on IPFS and Filecoin*. URL: <https://nft.storage/>.
- [44] Filecoin. *A decentralized storage network for humanity's most important information*. URL: <https://filecoin.io/>.
- [45] *Web3 storage - simple file storage with IPFS and Filecoin*. URL: <https://web3.storage/>.

10 Приложения

10.1 Ссылка на репозиторий

Ссылка на Gitlab репозиторий с проектом - [Gitlab](#)