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:
Лущ 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 7
2.3 Этапы проекта 9
2.4 Структура работы 11
3 Обзор существующих работ и решений 11
3.1 Маркетплейсы 11
3.2 Генеративно-состязательные сети 15
4 Smart-контракты 15
4.1 Структура NFT smart-контракта 17
4.2 Структура маркетплейс smart-контракта 25
5 Discord-бот 32
5.1 Взаимодействие с блокчейнами 32
5.2 Пользовательский интерфейс 37
5.3 Развертывание бота 41
6 Генеративно-состязательная сеть 41
7 Сервис с генеративно-состязательной сетью 42
7.1 Устройство сервиса 42
7.2 Устройство случайного получения NFT 43
8 Результаты и дальнейшие планы 43
9 Приложения 44
9.1 Ссылка на репозиторий 44
Список источников 44

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-состязательную сеть. Для этого необходимо было реализовать smart-конракт 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-состязательная 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.

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[2].*

Почему же NEAR Protocol является актуальным в нынешнее время? NEAR сеть обработала более 150 миллионов транзакций на текущий момент (28.05.2022 год). Для сравнения во время начала 2022 года количество обработанных транзакций оценивалось числом в 60 миллионов. Активных аккаунтов в NEAR насчитывается более 12 миллионов штук (28.05.2022). На момент начала курсовой работы их было всего лишь 3 миллиона штук.

Несомненно выбор NEAR Protocol в качестве блокчейна является актуальным, потому что за менее чем 2 года он достиг таких цифр [3].

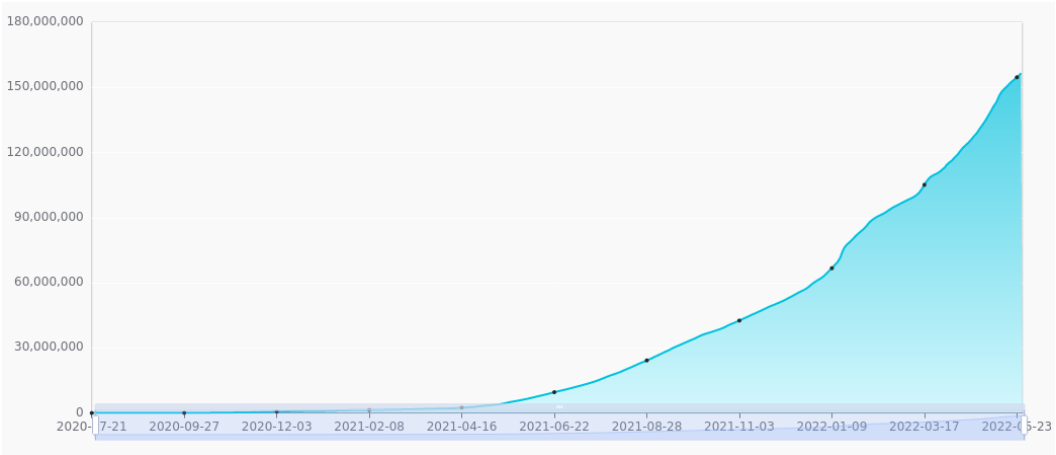


Рисунок 2.1. Количество транзакций обработанных в NEAR Network

Рисунок 2.2. Количество созданных кошельков в NEAR Network

Объяснение актуальности выбора самого приложения на базе блокчейна опирается на исследование Mapping the NFT revolution[4]. 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 в сравнении 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[5]. NEAR Protocol работает по схеме Proof-of-Stake (Pos) [6]. Отличительные черты относительно других блокчейнов - улучшенная масштабируемость, производительность, а также простота реализации приложений.

Определение. *DApps — это приложения, которые включают логику работы с функциями блокчейна[7].*

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[7]. Часто сокращают до слова контракт. В некоторых протоколах называют по-другому, например в Solana - это программы[8].*

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)[9].*

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[10] и near-sdk-as[11] соответственно). В данном проекте smart-контракты NFT и маркетплейса реализовываются на языке Rust.

Discord-бот реализуется на языке программирования TypeScript, используя near-api-js[12]. Discord-бот либо запускает «view operations» smart-контрактов, для получения метаданных привязанных к аккаунту; либо, при «change operations» создает транзакции с переданными аргументами пользователя и предоставляет URL для подписания транзакции пользователю. Discord API представляет множественный функционал для общения пользователя с ботом: slash-команды[13], контекстные меню[14], меню выбора[15], кнопки[13], модалы[16].

Замечание. *Каждый smart-контракт в NEAR (написанный на Rust/Assembly Script) 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 [17].*

Для создания сервиса генерации NFT, необходимо разработать алгоритм машинного обучения, основанный на комбинации из двух нейронных сетей: генератора и дискриминатора, настроенных на работу друг против друга. Основной задачей генератора является генерация новых изображений, подобных тем, что имеются в тренировочном наборе данных. Однако основной целью дискриминатора является проверка изображения на подлинность.

// Тут Арсен Токкожин напишет определения

Для связи дискорд бота с сервисом генерации NFT с Discord-ботом необходимо разработать сервис, который будет возвращать URL для подписания транзакции на получение NFT. Нужно реализовать функционал, который позволяет получить случайный NFT, который не будет просматриваться в smart-контракте, чтобы пользователь не знал, какое NFT досталось ему до подписания транзакции. Сервис реализуется на языке Python с использованием фреймворка fast-api. Обращение к сервису реализуется с помощью http - запросов.

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 соотношение 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 покупку продаваемых на маркетплейсе NFT токенов;
  - Покрыть основную часть функционала маркетплейс smart-контракта тестами;

- Разработать discord-бота (Лущ):
  - Изучить Javascript/Typescript;
  - Изучить основы работы с браузером через Javascript (сессионное/локальное хранилище браузера, класс window);
  - Изучить near-api-js и его кода для дальнейшего его переписывания под функциональность discord;
  - Реализовать KeyStore[18] работающий через Redis[19];
  - Написать реализацию авторизации в NEAR Wallet[20] через discord-бота, который использует вышеописанный KeyStore;
  - Написать реализацию создания URL на подпись транзакции/транзакций (одна транзакция[1], один Action[22]; одна транзакция, несколько Action; несколько транзакций, несколько Action);
  - Создание «Профиля пользователя». Вызов осуществляется через slash-команду[23] 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;
  - 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;
  - Поддержка изменения цены NFT;
  - Поддержать сервис с генеративно-состязательной сетью в discord-боте;
  - Сделать docker образ для удобного деплоя discord-бота;
  - Деплой discord-бота на облачный сервис (Кусиденов);
- Реализовать генеративно-состязательную сеть (Токкожин):
  - Сбор тренировочного набора данных;
  - Написать архитектуру модели генератора;
  - Написать архитектуру модели дискриминатора;
  - Обучение моделей;

[1)] В данном контексте класс Transaction[21]

– Улучшение качества модели путем изменения гиперпараметров нейронных сетей;

– Написание скрипта для простой генерации картинки, способствующей созданию сервиса;

– Добавление возможности влиять на параметры генерируемого изображения, с целью генерации картинки с заданной конфигурацией;

- Реализовать сервис с генеративно-состязательной сетью (Кусиденов):

  – Изучить как связать бота с сервисом и обсудить требования;

  – Продумать как скрыть сгенерированное NFT для пользователя маркетплейса;

  – Реализовать сервис, который по запросу бота будет отдавать NFT с его характеристиками;

  – Реализовать контракт, который скрывает какое NFT достанется пользователю.

2.4 Структура работы

Работа организована следующим образом. В разделе 3 дается обзор существующих на сегодняшний день маркетплейсов на NEAR Protocol и про генеративно-состязательную сеть. Раздел 4 описывает устройство и реализацию smart-контрактов NFT и маркетплейса. В 5 разделе идет описание трудностей и их решения в разработке discord-бота. В 7 разделе описывается сервис генерации NFT и его взаимодействие с discord-ботом. Конечные результаты и планы на дальнейшую работу можно найти в разделе 8.

3 Обзор существующих работ и решений

3.1 Маркетплейсы

На данный момент существует большое количество NFT маркетплейсов: opensea[24], rarible[25], solanart[26]. Если брать маркетплейсы только на базе NEAR Protocol, тогда существуют такие примеры как: Paras[27], Mintbase[28]. 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 предоставить 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 токену.
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 (картинки) - 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[29, 30].

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 [31] (см. 4.1). Дополнительно они привязывают каждый токен к какой-то конкретной коллекции и не позволяют создавать токен без привязки к коллекции (Рисунок 3.1).

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[32] (см. 5.1.5). IPFS предоставляется сервисом fleek[33]. В качестве ссылки на медиа-файл и метаданные они хранят CID, а не полный URL, это связанно с тем, что минт 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 3.3), а после CID подставляется в URL этого шлю-

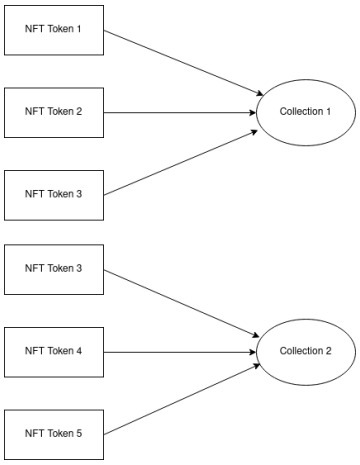


Рисунок 3.1. NFT токены и колекции в paras

за (Листинг 3.1), 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 3.1: Подстановка CID в URL у NEAR Wallet[34]

```
{
    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 3.2: Структура при вызове «nft_metadata» у NFT контракта

Давайте рассмотрим структуру метаданных NFT (Листинг 3.3). 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 3.3: Структура метаданных NFT в Paras

3.1.2 Mintbase Mintbase является менее популярным маркетплейсом, однако он предоставляем 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)[35]. С высокоуровневой точки зрения контракты Mintbase обладают совершенно другой архитектурой в отличие от Paras, так как в них вводится новый уровень абстракции. Mintbase поддерживает структуру состающую из следующих взаимосвязанных сущностей: Store, Thing, Token.

1. Store - NFT smart-контракт (децентрализованное API).
2. Thing - тип NFT (коллекция токенов).
3. Token - определенная NFT с уникальным владельцем.

Касательно Thing, Token - все понятно, это просто коллекция и токен из конкретной колекции, а вот Store - обозначает магазин, который может выпускать колекции с токенами. Чтобы пользователю начать создавать/продавать NFT ему нужно сначала создать свой Store smart-контракт. Для этого mintbase написала Factory библиотеку для создания своих Store. После этого владелец от лица своего Store сможет создавать новые NFT токены и коллекции токенов.

Резюмируя, получается что пользователь создает свой маркетплейс контракт и использует его для покупки/продажи/создания с другими маркетплейс контрактами пользователей. Paras в свою очередь дает один уникальный маркетплейс контракт с которым взаимодействуют пользователи.

Mintbase хранит метаданные NFT в Arweave. Как уже говорилось, Mintbase очень сильно отошел от стандарта, что проявляется и выдаваемой структуре. Разницу можно заметить в полях «metadata», «royalty» (см. 4.1.5) и «split_owners» (Листинг 3.4), также почти все метаданные NFT хранятся в распределенном хранилище. «royalty» и «split_owners» нужны для «Forever Royalties» и «Split Revenues». «Forever Royalties» это вечные проценты, которые нельзя поменять, после создания NFT, когда как «Split Revenues» это процентные доходы при продажи, которые слетают каждую продажу. Если взглянуть на структуру метаданных в распределенном хранилище (Листинг 3.5), то можно заметить, что «royalty» дублируется (не понятно для чего), существуют стандартные поля описания, названия, атрибутов NFT, ссылки на социальные сети, ссылка на медиа-объект, а также тип NFT - стандарт NFT.

Определение. *Arweave[36] - это протокол для постоянного хранения данных. Данный протокол связывает людей, у которых есть место на жестком диске, с теми, кому нужны данные ресурсы. Над Arweave построена децентрализованная сеть, которая называется permaweb[37]. С точки зрения функционала, данное хранилище, но запрашивание метаданных и самого объекта происходит через GraphQL[38]. GraphQL - инструмент для создания API.*

3.2 Генеративно-состязательные сети

// Текст будет написан Токкожиным Арсеном

4 Smart-контракты

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

```
{
    token_id: '194',
    owner_id: 'puma_zul.near',
    approved_account_ids: { 'market.mintbase1.near': 75 },
    metadata: {
        < Все поля как в стандарте NEP-171 null >
        extra: 'art',
        reference: 'smHkXU8rCq1Ggft1SPqTOpFw9sQxCwRg9kSNM7RatZ4',
    },
    royalty: {
        split_between: {
        'house_of_zul_nft_auctio.near': { numerator: 2650 },
        'polytechnic.near': { numerator: 2450 },
        'puma_zul.near': { numerator: 2500 },
        'genieve_dawkins.near': { numerator: 2400 }
        },
        percentage: { numerator: 1000 }
    },
    split_owners: {
        split_between: {
        'polytechnic.near': { numerator: 2250 },
        'house_of_zul_nft_auctio.near': { numerator: 7750 }
        }
    },
    minter: 'puma_zul.near',
    loan: null,
    composeable_stats: { local_depth: 0, cross_contract_children: 0 },
    origin_key: null
}
```

Листинг 3.4: Структура NFT в Mintbase

```
{
    "category":"art",
    "description":"...",
    "copies":5,
    "media_hash":"jvnFdtx-XIOoRPJMeS_JhHG_zMcP_PzRvPgMOOcLgMU",
    "lock":null,
    "visibility":"safe",
    "youtube_url":null,
    "animation_url":"...",
    "animation_hash":"...",
    "document":null,
    "document_hash":null,
    "royalty":{
        "genieve_dawkins.near":2400,
        "house_of_zul_nft_auctio.near":2650,
        "puma_zul.near":2500,
        "polytechnic.near":2450
    },
    "royalty_perc":0.1,
    "split_revenue":{
        "polytechnic.near":2250,
        "house_of_zul_nft_auctio.near":7750
    },
    "tags":["oil","canvas","zul","azul "],
    "media":"https://arweave.net/jvnFdtx-XIOoRPJMeS_JhHG_zMcP_PzRvPgMOOcLgMU",
    "extra":[
        {"trait_type":"Start Date","value":1652943600,"display_type":"date"},
        ...
    ],
    "title":"...",
    "store":"eremod.mintbase1.near",
    "external_url":"htttps://eremod.com",
    "type":"NEP171"
}
```

Листинг 3.5: Структура метеданных NFT в распределенном хранилище в Mintbase

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[31].

4.1.1 NEAR SDK Введем основные функции, структуры, декараторы, которые используются при написании smart-контрактов. Для этого необходим фреймворк near-sdk[10].

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 4.1: Атрибуты NEAR SDK фреймворк

Структуры:

```
use near_sdk::collections::{LazyOption, LookupMap, UnorderedMap, UnorderedSet};
LookupMap       /* Неитерируемый 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 4.2: Структуры NEAR SDK фреймворк

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 4.3: Функции NEAR SDK фреймворк

```rust
#[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 4.5: NFT контракт структура TokenMetadata

4.1.2 Core Functionality Опишем основные структуры и функции[39], которые используются в NFT контрактах.

```rust
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 4.4: Метаданные NFT контракта

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.

Структура метаданных токена (Листинг 4.5) 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.

```rust
#[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 4.6: NFT контракт структура Token и JsonToken

Структура NFT токена (Листинг 4.6) 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 4.7: 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 (см. 4.1.5). Так как это payable функция, то пользователь должен будет внести депозит для хранения информации о добавляемом токене. Лишний депозит вернется пользователю обратно. Также в отличие от Paras мы не обязуем пользователя привязывать токен

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

    /* Распаковать и положить royalties */
    royalty = AcceptRoyalties(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 4.8: NFT token mint

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

Листинг 4.9: NFT nft_token

к конкретной коллекции чем с одной стороны дали больше свободы пользователю, а с другой стороны упростили реализацию.

Псевдокод создание токена описан в листинге 4.8.

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 4.9).

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 выполниться 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:

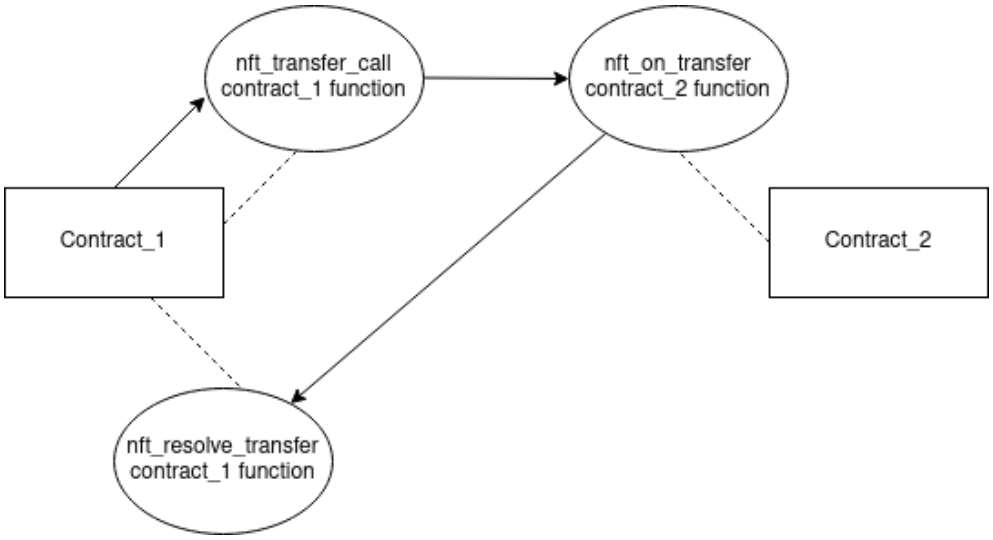


Рисунок 4.1. nft_transfer_call

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 описан в листинге 4.10.

4.1.3 Enumeration Для удобного взаимодействия с контрактом, необходимо добавить больше view функций с pagination для просмотра NFT токенов[40]:

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 вышеописанных функций можно найти в листинге 4.11.

```
fn 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 */
    return 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() {
        return 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);

    return false
}
```

Листинг 4.10: NFT контракт transfer

4.1.4 Approval Management Необходимо добавить функционал передачи своего токена другим аккаунтом от своего имени[41]. Для этого будет хранить 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, которая описана в листинге 4.12.

```
pub fn nft_total_supply() -> U128 {
    return length(own_contract.token_metadata_by_id)
}

pub fn nft_tokens(from_index, limit) -> Vec<JsonToken> {
    return own_contract.token_metadata_by_id.keys()
        .skip(from_index)
        .take(limit)
        .map(|token_id| self.nft_token(token_id))
        .collect()
}

pub fn nft_supply_for_owner(account_id) -> U128 {
    if Exist(account_id) {
        return length(own_contract.tokens_per_owner.get(account_id))
    } else {
        return 0
    }
}

pub fn nft_tokens_for_owner(
    account_id,
    from_index,
    limit
) -> Vec<JsonToken> {
    if Exist(account_id) {
        return tokens.iter()
            .skip(from_index)
            .take(limit)
            .map(|token_id| self.nft_token(token_id))
            .collect()
    } else {
        return vec![];
    }
}
```

Листинг 4.11: NFT контракт enumeration

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

Листинг 4.12: NFT контракт approval management

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

Листинг 4.13: NFT контракт approval management

Если он решит добавить account_2, account_3, как доверенные тогда структура примет вид как в листинге 4.13.

Счетчик next_approval_id необходим, чтобы не случилось случая, когда новый владелец токена решил добавить доверенный аккаунт, который был до

```
fn nft_payout(token_id, balance) -> Payout {
    /* Проверить, что токен существует */
    assert(ExistToken(token_id))

    /* Достать структуру токен */
    token = own_contract.tokens_by_id.get(token_id);
    result = PayoutConstruct();

    /* Посчитать доли других аккаунтов */
    current_sum = 0;
    for (key, value) in Iter(token.royalty) {
        if key == token.owner_id {
            continue;
        }
        result.payout.insert(
            key, CalcPayout(value, balance)
        );
        current_sum += value;
    }
    /* Посчитать свою долю */
    result.payout.insert(token.owner_id, CalcPayout(10000 - current_sum, balance));
    return result
}
```

Листинг 4.14: NFT контракт nft_payout

этого. Такие случаи могут испортить всю логику на других smart-контрактах. Подробнее краевые случаи описаны в стандарте[41].

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[42]. Для этого у нас есть поле royalty в структуре Token, которая отображает 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 данных функций можно найти в листингах 4.14 и 4.15.

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

```
#[payable]
fn nft_transfer_payout(
    receiver_id, token_id,
    approval_id, balance,
) -> Payout {
    /* Отправить токен */
    sender_id = env::predecessor_account_id();
    prev_token = own_contract.internal_transfer(sender_id, receiver_id, token_id, approval_id);

    result = PayoutConstruct();

    /* Посчитать доли других аккаунтов */
    current_sum = 0;
    for (key, value) in Iter(prev_token.royalty) {
        if key == prev_token.owner_id {
            continue;
        }
        result.payout.insert(key, CalcPayout(value, balance));
        current_sum += value;
    }
    /* Посчитать свою долю */
    result.payout.insert(prev_token.owner_id, CalcPayout(10000 - current_sum, balance));
    return result
}
```

Листинг 4.15: NFT контракт nft_transfer_payout

реализован разными способами. Мы придерживались архитектуры, которая была описана у Paras(3.1.1), то есть один маркетплейс smart-контракт с которым будут взаимодействовать пользователи.

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.

Структура маркетплейс контракта описана в листинге 4.16.

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 он является доверенным аккаунтом для продаваемого токена.

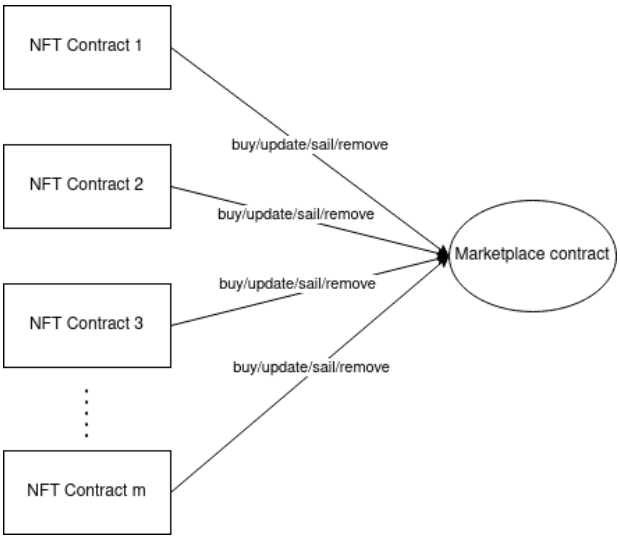


Рисунок 4.2. Маркетплейс контракт core functionality

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

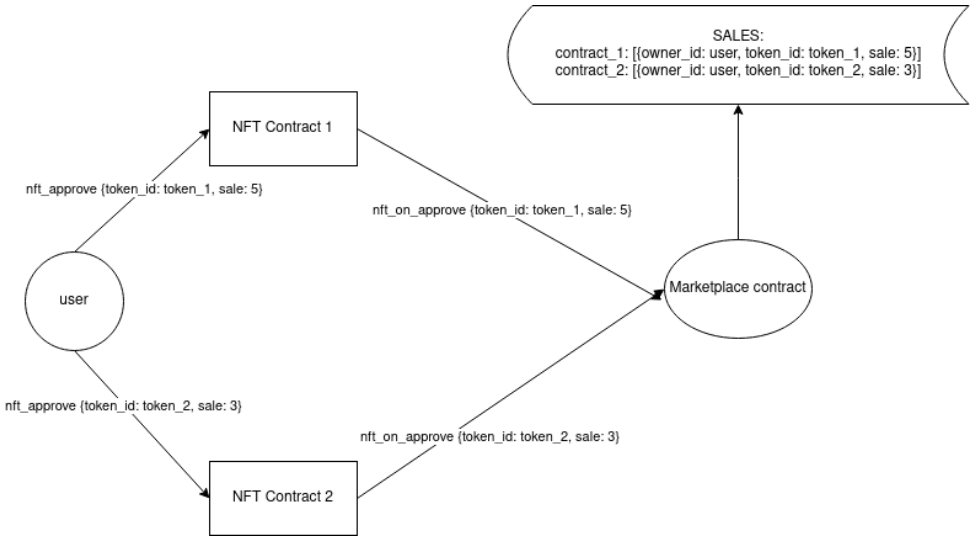


Рисунок 4.3. Выставление на продажу в маркетплейс контракте

Псевдокод вышеописанного функционала можно увидеть в листинге 4.17.

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. Для внесения near под хранение используется функция storage_deposit, а для

```rust
/* 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>,
}
```

Листинг 4.16: Marketplace contract struct

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, функционал можно увидеть в коротком псевдокоде (Листинг 4.18).

Покупка осуществляется следующим образом: пользователь дергает nft_offer команду у маркетплейс контракта, тот в свою очередь вызывает nft_transfer_payout на NFT контракте, а потом завершает или отменяет покупку используя resolve_purchase. Функция resolve_purchase при удачной покупке разделит сумму продажи относительно долей royalties. Данные величины нам возвращает функция nft_transfer_payout.

Схема покупки выглядит следующим образом (рис. 4.4), а с псевдокодом функций покупки можно ознакомиться в Листинге 4.19.

```
fn nft_on_approve(token_id, owner_id, approval_id, sale_price) {
    /* NFT контракт с которого был вызвана продажа */
    nft_contract_id = env::predecessor_account_id();

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

    assert(owner_id == signer_id)

    /* Считаем сколько нужно на хранилище и сколько внесено */
    paid_storage = own_contract.storage_deposits.getPaidStorage(signer_id);
    required_storage = CalcRequiredStorage();
    assert(paid_storage > required_storage);

    /* Добавляем покупку в необходимые структуры */
    contract_and_token_id = nft_contract_id + '.' + token_id;
    own_contract.sales.InsertNewSale(
        contract_and_token_id, owner_id, approval_id, nft_contract_id, token_id, sale_price
    );

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

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

Листинг 4.17: Маркетплейс контракт nft_on_approve

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

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

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

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

Листинг 4.18: Маркетплейс контракт изменение цены/отмена продажи

4.2.2 Enumeration Для того чтобы удобно взаимодействовать с маркетплейс 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.

```
#[payable]
pub fn offer(nft_contract_id, token_id) {
    /* Смотрим сколько пользователь внес депозита */
    deposit = env::attached_deposit();

    contract_token_id: String = contract_id + '.' + token_id;

    /* Получаем структуру продажи для token_id из нужного NFT контракта */
    offer_sale = own_contract.sales.get(contract_token_id);
    buyer_id = env::predecessor_account_id();

    /* Проверяем, что внесенный депозит больше цены и что покупатель не является владельцем */
    assert(offer_sale.owner_id != buyer_id);
    assert(deposit >= offer_sale.sale_conditions);

    own_contract.process_purchase(contract_id, token_id, deposit,buyer_id);
}

#[private]
pub fn process_purchase(nft_contract_id, token_id, price, buyer_id) -> Promise {
    /* Удаляем структуру продажи из соответствующих структур */
    purchased_sale = own_contract.internal_remove_sale(nft_contract_id, token_id);

    /* Вызываем nft_transfer_payout на NFT smart-контракте и завершаем покупку в resolve_purchase */
    nft_contract::nft_transfer_payout(
        buyer_id,
        token_id,
        purchased_sale.approval_id,
        price
    ).then(own_contract::resolve_purchase(
        buyer_id,
        price
    ))
}

#[private]
pub fn resolve_purchase(
    buyer_id,
    price,
) -> U128 {
    /* Если операция nft_transfer_payout на контракте NFT была выполнена успешно */
    if IsSuccess() {
        /* Получаем payout структуру из nft_transfer_payout */
        payout = PromisResult();

        /* Отправляем каждому royalti аккаунту соответствующую долю */
        for (receiver_id, amount) in payout {
            Transfer(receiver_id, amount);
        }
        return price;
    } else {
        return None;
    }
}
```

Листинг 4.19: Маркетплейс контракт покупка токена

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 описан в листинге 4.20.

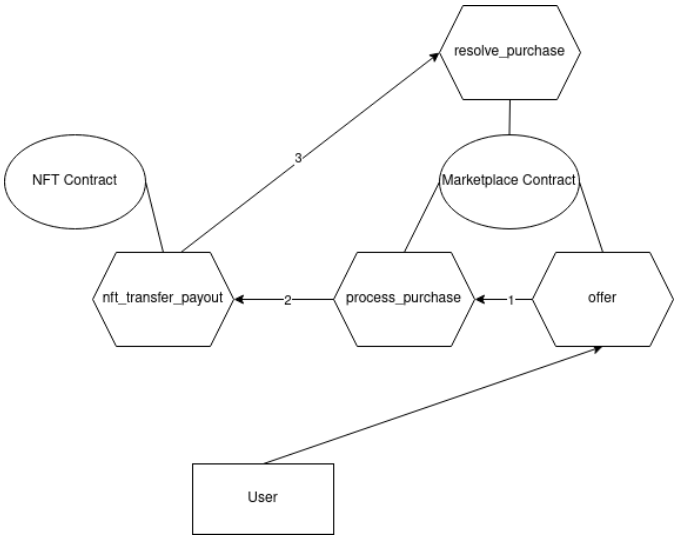


Рисунок 4.4. Покупка токена

```
pub fn get_supply_sales() -> u64 {
    return length(own_contract.sales);
}

pub fn get_supply_by_owner_id(account_id) -> u64 {
    owner_id = own_contract.by_owner_id.get(account_id);
    return length(owner_id);
}

pub fn get_sales_by_owner_id(account_id, from_index, limit) -> Vec<Sale> {
    sales = owner_contract.by_owner_id.get(account_id);

    sales.iter()
        .skip(from_index)
        .take(limit)
        .map(|token_id| owner_contract.sales.get(token_id))
        .collect()
}

pub fn get_supply_by_nft_contract_id(nft_contract_id) -> u64 {
    nft_contract_id = own_contract.by_nft_contract_id.get(nft_contract_id);
    return length(nft_contract_id);
}

pub fn get_sales_by_nft_contract_id(nft_contract_id, from_index, limit) -> Vec<Sale> {
    let sales = own_contract.by_nft_contract_id.get(nft_contract_id);

    sales.iter()
        .skip(from_index)
        .take(limit)
        .map(|token_id| own_contract.sales.get(nft_contract_id, '.', token_id))
        .collect()
}

pub fn get_sale(nft_contract_token) -> Option<Sale> {
    own_contract.sales.get(nft_contract_token)
}
```

Листинг 4.20: Маркетплейс контракт enumeration

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[43] 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 подаккаунты, которые по своему функционалу ничем не отличаются от обычного аккаунта. Данные подаккаунты решают проблему развертывания контрактов: на один 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 ненадобности.

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 на 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

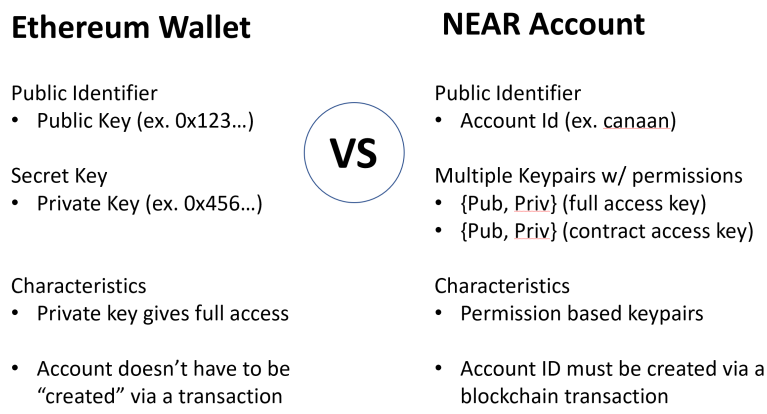


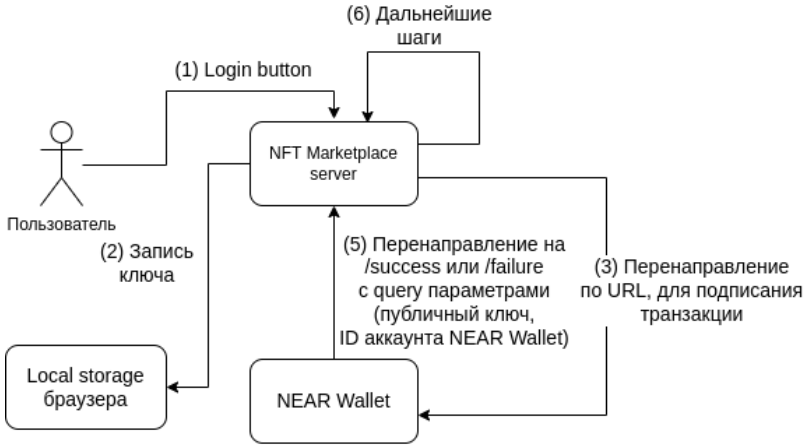
Рисунок 5.1. Сравнение аккаунтов Ethereum и NEAR

нам не потребуется, так как у нас либо «view operations» в контрактах, либо «payable change operations».

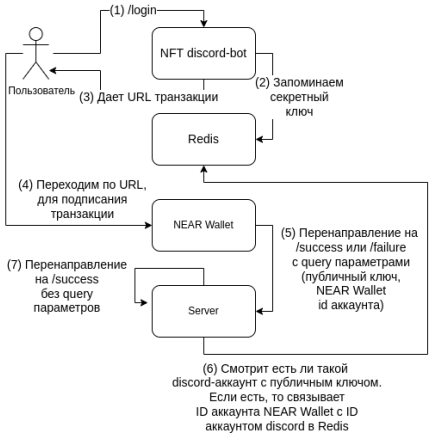
На маркетплейсах в виде сайта вся авторизация происходит на client-side стороне: создается пара ключей (access key) - публичный и секретный; секретный ключ сохраняется в local storage браузера; подписывается транзакция и в последствии этот ключ будет использован сайтом, для подтверждения авторизации (Рисунок 5.2a). Сценарий авторизации в discord-бот различается, ведь 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 базы данных было принято использовать Redis. Был написан KeyStore, который взаимодействует с Redis и функционал, который возвращает URL, а не перенаправляет 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[44]. В веб API есть два объекта, которые используют интерфейс storage: session storage, local storage[45]. Session storage хранит данные до закрытия браузера, когда как данные в local storage не имеют ограничения по времени и могу быть удалены только намерено.*

Определение. *KeyStore[18] — это класс, который хранит ключи, для подписей транзакций. Их 4 вида: BrowserLocalStorageKeyStore, InMemoryKeyStore, MergeKeyStore, UnencryptedFileSystemKeyStore. BrowserLocalStorageKeyStore*



(a) Авторизация в NEAR Wallet в near-api-js



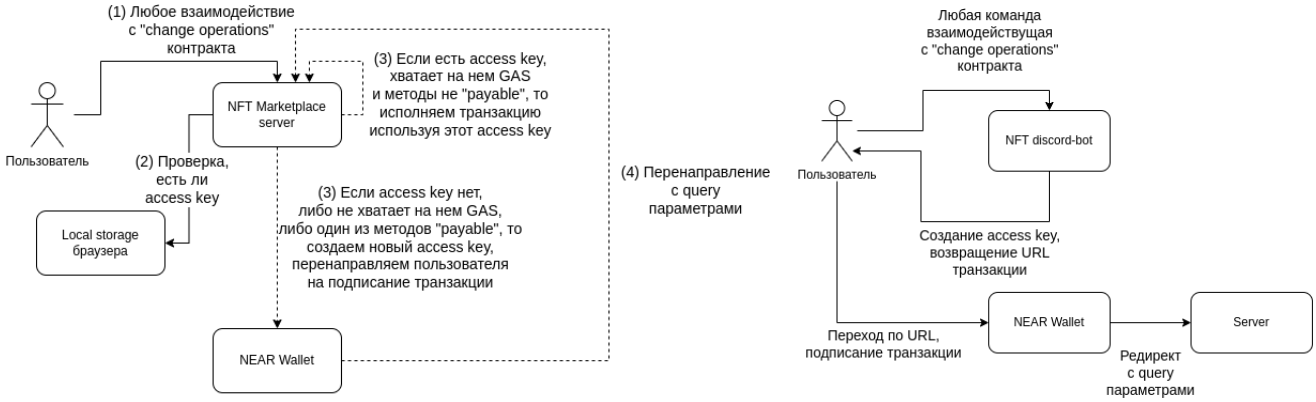
(b) Авторизация в NEAR Wallet в discord-боте

Рисунок 5.2

пользуется локальным хранилищем браузера для записи, изменения, просмотра значений по ключу. InMemoryKeyStore хранит все в оперативной памяти, используется для тестирования. MergeKeyStore используется для объединения множества KeyStore. UnencryptedFileSystemKeyStore хранит все на диске в виде JSON файла, используется в near cli[46].

5.1.3 Вызовы методов у контрактов Вызовы методов у smart-контрактов — это то, чем бот занимается большую часть времени. Как и в случае авторизации, на меркетплейсах в виде сайте подписывание транзакций происходит на client-side стороне (Рисунок 5.3a): проверка существования access key в local storage браузера, при существовании и еще нескольких условиях идет подписание транзакции без участия пользователя, в ином случае создается URL по которому пользователь подписывает транзакцию. В случае discord-бота (Рисунок 5.3b) просто выброшена та часть, которую возможно исполнить на client-side стороне и проверка на access key, из-за ненадобности, так как у нас только «payable change operations».

near-api-js не предоставляет хороших оберток для подписания транзакций, в которых несколько Action и Transaction. В нашем случае такие сценарии нужны при отмене продажи NFT и покупки NFT. В случае выставлении на продажу требовалось вызывать методы в таком порядке: «storage_deposit» у контракта маркетплейса для вложения депозита хранения, «nft_approve» у контракта NFT для утверждения маркетплейса в NFT контракте и «storage_withdraw» у контракта маркетплейса для возвращения



(a) Подпись транзакций в NEAR Wallet в near-api-js

(b) Подпись транзакций в NEAR Wallet в discord-боте

Рисунок 5.3

неиспользуемых NEAR для хранения. В случае отмены продажи: «offer» и «storage_withdraw» у контракта маркетплейса. Были написаны данные обертки.

Определение. *Есть несколько видов Action: CreateAccount, DeployContract, FunctionCall, Transfer, Stake, AddKey, DeleteKey, DeleteAccount. Из этого списка discord-бот использует только FunctionCall, который нужен для вызова методов у smart-контракта. FunctionCall образуется из названия метода, аргументов метода, вносимого депозита и вносимых GAS. Transaction это набор Action, и название контракта, то есть в одном Transaction можно подписывать только методы одного smart-контракта. При подписании транзакции может быть несколько Transaction, что нам позволяет подписывать за раз исполнение разных методов на разных smart-контрактах.*

5.1.4 Структура получаемого NFT и его метаданных На данный момент, при создании NFT в структуре (Листинг 5.1) 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). ID токена при создании формируется следующим образом:

```
{
    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://bafybeiahhurffoxjubs42l7bl3jjc5zk5vrafiijcxkhex2ukjm3zsbbti.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://bafybeiahhurffoxjubs42l7bl3jjc5zk5vrafiijcxkhex2ukjm3zsbbti.ipfs.dweb.link/m',
        reference_hash: null
    }
}
```

Листинг 5.1: Структура получаемого NFT

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

Листинг 5.2: Структура метаданных NFT в децентрализованном хранилище

ID NEAR аккаунта, который минтит NFT, и текущее время. Такая структура формирования ID токена позволяет искоренить все коллизии.

Структура метаданных (Листинг 5.2) 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 ненадобности.

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 развертывании 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[47] — бесплатное децентрализованное хранилище NFT на IPFS[32] и Filecoin[48]. 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.*

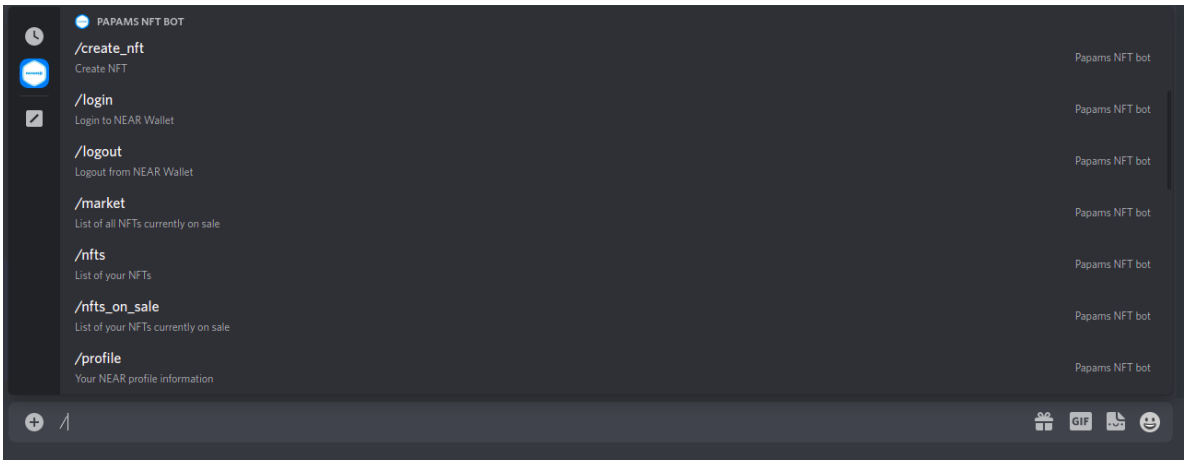
Определение. *IPFS(InterPlanetary File System) - это протокол распределенной системы обмена файлов. При добавлении файла в IPFS, он делится на маленькие куски, криптографически хэшируется и отдается уникальный фингерпринтр, который называется CID(Content identifier) [32].*

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, взаимодополняющие 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.*

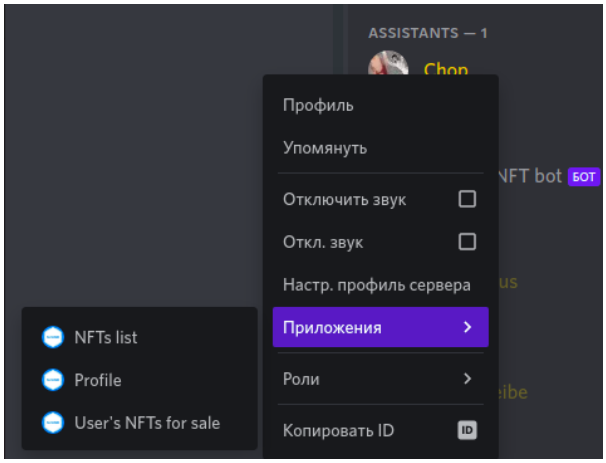
Замечание. *Изначально использовался сервис web3storage[49] для хранения, но он отличился медленной загрузкой, получением файлов, из-за чего появилась необходимость в смене на NFT.Storage.*

5.2 Пользовательский интерфейс

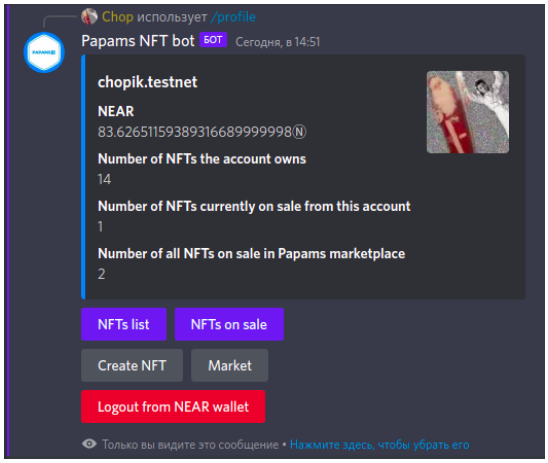
У пользователя, есть несколько мест откуда он может запускать команды бота: slash-команды (Рисунок 5.4a), контекстные меню по пользователю (Рисунок 5.4b) и сами профиля пользователей (Рисунок 5.4c, Рисунок 5.6a). Со своего профиля и с помощью slash-команд, все взаимодействия идут в отношении себя, когда как, при выборе профиля и использовании на нем функционала из контекстного меню операции совершаются в отношении этого пользователя.



(a) Slash-команды бота

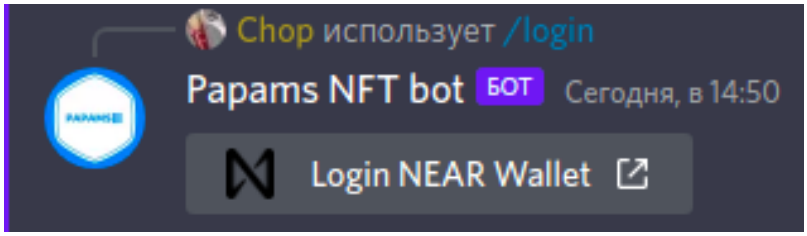


(b) Контекстные меню бота

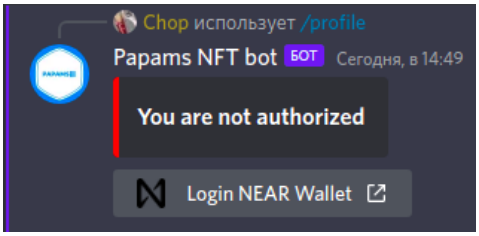


(c) Свой профиль пользователя

Рисунок 5.4. Точки входа пользователя



(a) Намеренная авторизация

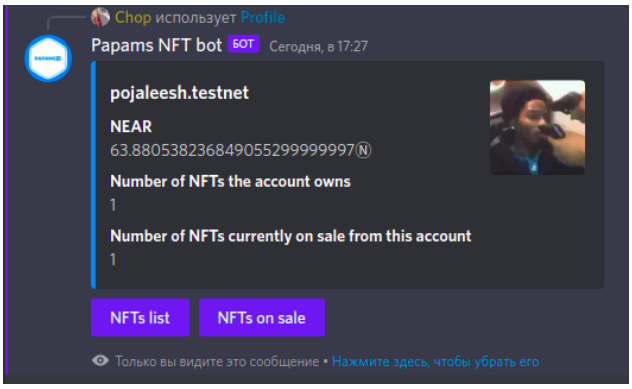


(b) Предложение об авторизации

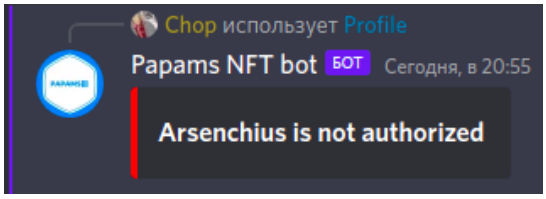
Рисунок 5.5. Авторизация пользователя

Авторизация пользователя происходит через соответствующую slash-команду (Рисунок 5.5a) или при взаимодействии бота в отношении своего профиля, если пользователь не авторизован (Рисунок 5.5b).

С помощью контекстных меню (Рисунок 5.4b), можно взаимодействовать с другими пользователями: мы можем получить профиль пользователя (Рисунок 5.6a), его список NFT, его список продаваемых NFT (Рисунок 5.7b).

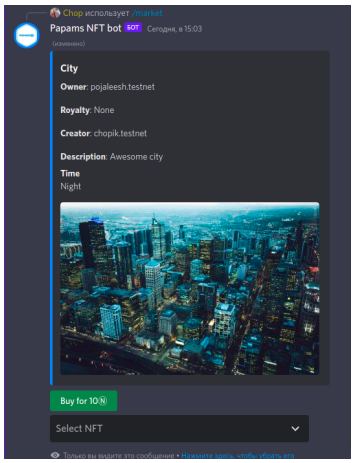


(a) Чужой профиль пользователя

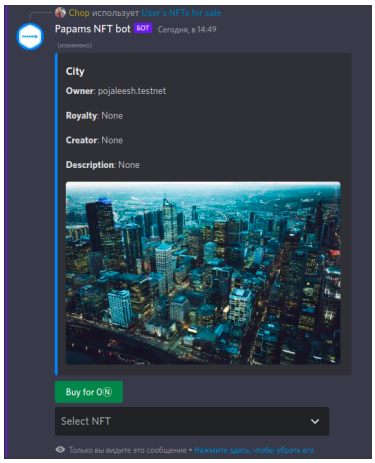


(b) Взаимодействие с пользователем, если он не авторизован

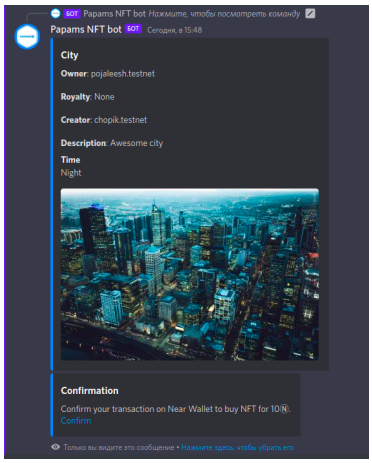
Рисунок 5.6. Взаимодействия с другими пользователями



(a) Список NFT на маркеплейсе



(b) Список продаваемых NFT у пользователя



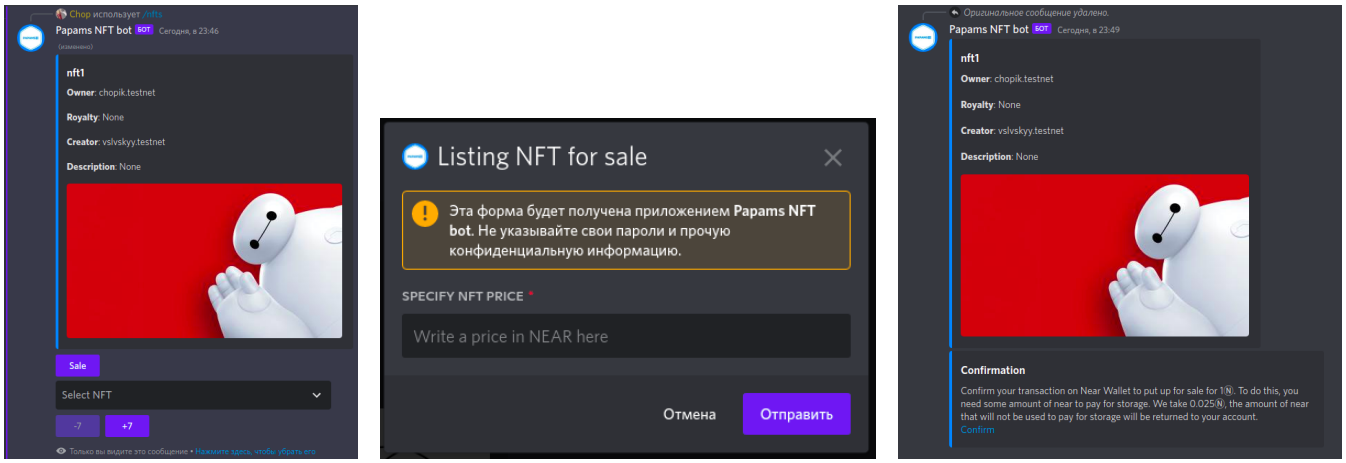
(c) Подтверждение покупки

Рисунок 5.7. Покупка NFT

Замечание. *Если в контекстном меню взаимодействовать с собой, то эффект будет такой-же как от slash-команд, то есть фактически мы можем не использовать slash-команды вообще и взаимодействовать только через GUI(Graphical User Interface, Графический Пользовательский Интерфейс).*

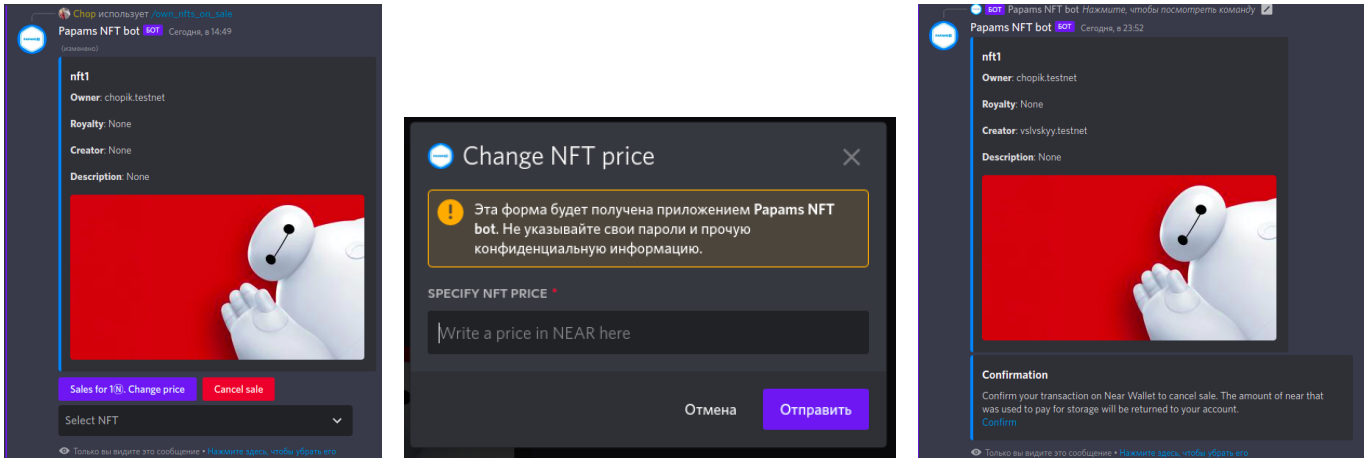
При взаимодействии с пользователем мы можем купить у него NFT, если таковые имеются (Рисунок 5.7b). В своем общем списке NFT, в своем списке продаваемых NFT или в общем списке продаваемых NFT (Рисунок 5.9a) можно отменить продажу или сменить цену.

Со своего профиля (Рисунок 5.4c) или с помощью slash-команды «/create_nft» можно запустить процесс создания NFT (Рисунок 5.10), он пе-



(a) Свой общий список NFT *(b) Указание цены, при продаже* *(c) Подтверждение продажи*

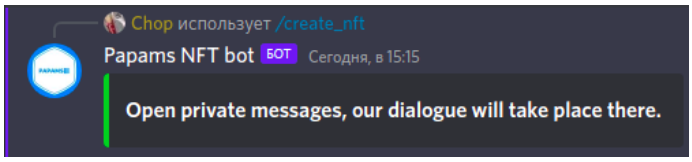
Рисунок 5.8. Выставление NFT на продажу



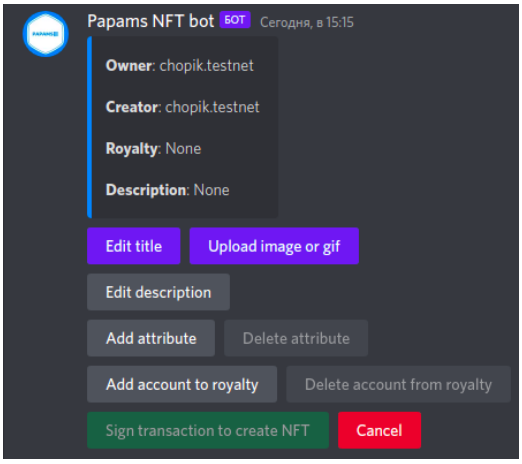
(a) Интерфейс продажи: свой список продаваемых NFT, общий список продаваемых NFT, свой общий список NFT *(b) Указание цены, при ее смене* *(c) Подтверждение отмены продажи*

Рисунок 5.9. Отмена продажи и смена цены NFT

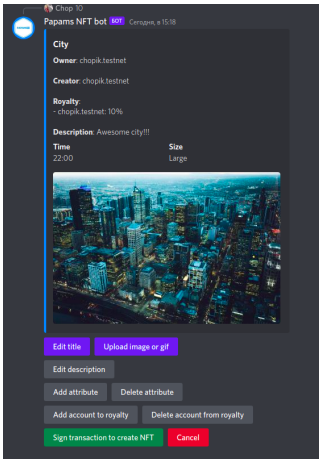
реводит пользователя в личные сообщение, так как от пользователя нужен будет медиа-объект, для загрузки которого, пока что нет функционала не через сообщения. NFT собирается в виде конструктора - NFT строится при добавлении нового элемента, для текстовых заполнений используются модалы, когда как для загрузки медиа-объекта, бот отправляет сообщение, в котором говорит, что пользователь должен ответить на это сообщение сообщением с медиа-объектом (если пользователь отвечает сообщением без медиа-файла, то



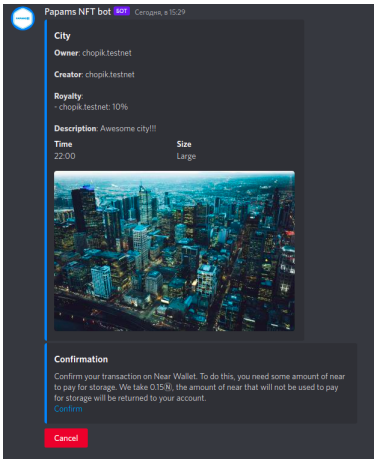
(a) Сообщение о переходе в личные сообщения



(b) Конструктор NFT



(c) Законченное NFT



(d) Подтверждение минта

Рисунок 5.10. Минт NFT

выведется сообщение с ошибкой, на которое нужно будет ответить, чтобы загрузить медиа-файл).

Замечание. *Визуальный вид списка NFT (Рисунок 5.9a, Рисунок 5.8a) меняется от их количества: если NFT меньше 7, то не будет кнопок «+7», «-7», которые будут подгружать следующие/предыдущие 7 NFT. Это сделано для того, чтобы в один момент не подгружать все NFT с блокчейна.*

5.3 Развертывание бота

Чтобы развернуть наш проект в интернете используется Alibaba cloud ecs (Elastic Compute Service). Для этого используется проксирование с помощью nginx (Рисунок 5.11).

6 Генеративно-состязательная сеть

// Текст будет написан Токкожиным Арсеном

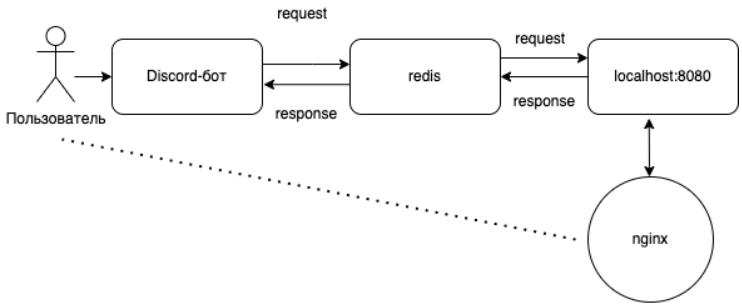


Рисунок 5.11. Схема развертывания бота

7 Сервис с генеративно-состязательной сетью

В данной главе будет описываться сервис для создания NFT с помощью генеративно-состязательной сети и описание его контракта.

7.1 Устройство сервиса

Сервис представляет собой http-сервис, которому на вход подается GET-запрос от discord-бота и он на своей стороне генерирует картинку, характеристики к ней и формирует URL транзакции.

Архитектура взаимодействия бота и сервера проиллюстрирована на Рисунке 7.1

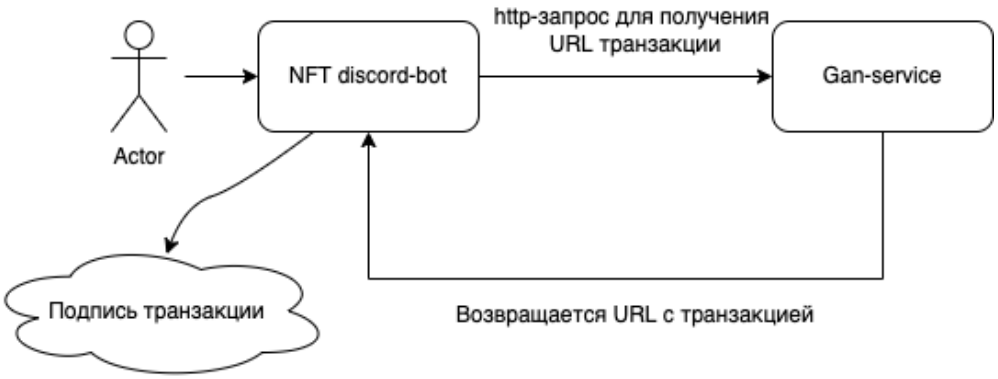


Рисунок 7.1. Marketplace contract sell

Замечание. *Создание нового NFT удовлетворяет следующему свойству: Пользователь при генерации NFT увидит саму картинку только подписания транзакции.*

7.2 Устройство случайного получения NFT

Для того чтобы пользователь не видел картинку которую получит до подписания транзакции мы используем следующую схему: Формируем контракт в котором будут N сгенерированных NFT, которые хранятся в динамическом массиве, в этот массив добавляется (N + 1)’ый NFT, но при этом пользователю будет возвращаться случайный NFT из этого массива. Таким образом пользователь не сможет угадать, какая именно nft ему выпадет.

8 Результаты и дальнейшие планы

// Будет написано, когда Токкожин свою часть напишет

9 Приложения

9.1 Ссылка на репозиторий

Ссылка на Gitlab репозиторий с проектом - Gitlab

Список источников

[1] Theodosis 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] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: http://www.bitcoin.org/bitcoin.pdf.

[3] NEAR Inc. *NEAR Explorer | Stats*. URL: https://explorer.near.org/stats.

[4] 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.

[5] *NEAR Protocol*. URL: https://near.org/.

[6] ILLIA POLOSUKHIN. *Thresholded proof of stake*. апр. 2019. URL: https://near.org/blog/thresholded-proof-of-stake/.

[7] Bina Ramamurthy. *Blockchain in Action*. S.l: Manning Publications, 2020. ISBN: 9781617296338.

[8] Solana Foundation. *Introduction*. URL: https://spl.solana.com/.

[9] NEAR Protocol. *Transaction*. URL: https://docs.near.org/docs/concepts/transaction.

[10] NEAR Protocol. *Near/near-sdk-rs: Rust library for writing near Smart Contracts*. URL: https://github.com/near/near-sdk-rs.

[11] NEAR Protocol. *Near/near-sdk-as: AssemblyScript library for writing near Smart Contracts*. URL: https://github.com/near/near-sdk-as.

[12] NEAR Protocol. *near-api-js (JavaScript library)*. URL: https://docs.near.org/docs/api/javascript-library.

[13] discord.js. *discord.js guide, buttons*. URL: https://discordjs.guide/interactions/buttons.html#building-and-sending-buttons.

[14] discord.ts. *discord.ts official documentation, context menu*. URL: https://discord-ts.js.org/docs/decorators/gui/context-menu/.

[15] discord.js. *discord.js Guide, select menus*. URL: https://discordjs.guide/interactions/select-menus.html#building-and-sending-select-menus.

[16] discord.js. *discord.js guide, modals*. URL: https://discordjs.guide/interactions/modals.html#building-and-responding-with-modals.

[17] NEAR Protocol. *Introduction, Thinking in gas*. URL: https://docs.near.org/docs/concepts/gas#thinking-in-gas.

[18] NEAR Protocol. *Class KeyStore*. URL: https://near.github.io/near-api-js/classes/key_stores_keystore.keystore.html.

[19] Redis Ltd. *Redis*. URL: https://redis.io/.

[20] Roketo Labs LTD. *Near wallet*. URL: https://wallet.near.org/.

[21] NEAR Protocol. *Class transaction*. URL: https://near.github.io/near-api-js/classes/transaction.transaction-1.html.

[22] NEAR Protocol. *Class action*. URL: https://near.github.io/near-api-js/classes/transaction.action.html.

[23] discord.js. *discord.js Guide, slash commands*. URL: https://discordjs.guide/interactions/slash-commands.html#registering-slash-commands.

[24] Inc OpenSea Ozone Networks. *OpenSea, the largest NFT Marketplace*. URL: https://opensea.io/.

[25] Inc Rarible. *NFT Marketplace*. URL: https://rarible.com/.

[26] Solanart. *Solanart - discover, collect and trade nfts*. URL: https://www.solnaart.com/.

[27] Paras. *NFT Marketplace for digital collectibles on near*. URL: https://paras.id/.

[28] Mintbase. *NFT Marketplace; Toolkit*. URL: https://www.mintbase.io/.

[29] ParasHQ. *paras-nft-contract*. URL: https://github.com/ParasHQ/paras-nft-contract.

[30] ParasHQ. *paras-nft-contract*. URL: https://github.com/ParasHQ/paras-marketplace-contract.

[31] *NEAR NFT Standarts*. 2022. URL: https://nomicon.io/Standards/Tokens/NonFungibleToken/Core.

[32] Protocol Labs. *IPFs powers the distributed web*. URL: https://ipfs.io/.

[33] fleek. URL: https://fleek.co/.

[34] 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.

[35] Mintbase. *Mintbase/mintbase-core: Mintbase core NFT store factory contracts*. URL: https://github.com/Mintbase/mintbase-core.

[36] Minimum Spanning Technologies. *Store data, permanently*. URL: https://www.arweave.org/.

[37] URL: https://arwiki.wiki/#/en/the-permaweb.

[38] The GraphQL Foundation. URL: https://graphql.org/.

[39] *core-functionality*. URL: https://nomicon.io/Standards/Tokens/NonFungibleToken/Core.

[40] *enumeration-standard*. URL: https://nomicon.io/Standards/Tokens/NonFungibleToken/Enumeration.

[41] *approval-standard*. 2022. URL: https://nomicon.io/Standards/Tokens/NonFungibleToken/ApprovalManagement.

[42] *royalty-standard*. 2022. URL: https://nomicon.io/Standards/Tokens/NonFungibleToken/Payout.

[43] NEAR Protocol. *Account*. URL: https://docs.near.org/docs/concepts/account.

[44] *Storage - интерфейсы веб API: MDN*. URL: https://developer.mozilla.org/ru/docs/Web/API/Storage.

[45] *Window.localstorage - Интерфейсы веб API: MDN*. URL: https://developer.mozilla.org/ru/docs/Web/API/Window/localStorage.

[46] NEAR Protocol. *Near cli*. URL: https://docs.near.org/docs/tools/near-cli.

[47] *Free decentralized storage and bandwidth for nfts on IPFS and Filecoin.* URL: https://nft.storage/.

[48] Filecoin. *A decentralized storage network for humanity's most important information*. URL: https://filecoin.io/.

[49] *Web3 storage - simple file storage with IPFS and Filecoin*. URL: https://web3.storage/.