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

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 блокчеином NEAR-protocol под возможности discord и реализовать discord-бота, ...(тут что-то про модель).

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 the NFT smart contract (according to the NEP-171 standard), the marketplace smart contract, adjust the API for interacting with the NEAR-protocol blockchain under the capabilities of discord and implement the discord-bot, ... (here is something about the model).

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

2.2 Постановка задачи

В качестве блокчейна используется NEAR protocol[1]. NEAR Protocol работает по схеме Proof-of-Stake(Pos) [2]. 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[3].*

Определение. *DApps — это приложения, которые включают логику работы с функциями блокчейна [4].*

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 [4]. Часто сокращают до слова контракт. В некоторых протоколах называют по-другому, например в Solana - это программы[5].*

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)[6].*

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[7] и near-sdk-as[8] соответственно). В

данном проекте smart-контракты NFT и маркетплейса реализовываются на языке Rust.

Discord-бот реализуется на языке программирования TypeScript, используя near-api-js[9]. Discord-бот либо запускает «view operations», для получения метаданных аккаунта и view методов NFT, маркетплейс smart-контрактов; либо, при «change operations» создает транзакции и предоставляет url для NEAR Wallet аккаунта пользователя.

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 [10].*

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):
  –
- 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[11] работающий через Redis[12];

– Написать реализацию авторизации в Near Wallet[13] через discord-бота, который использует вышеописанный KeyStore;

– Написать реализацию создания url на подпись транзакции/транзакций(одна транзакция[1], один Action[15]; одна транзакция, несколько Action; несколько транзакций, несколько Action);

– Создание «Профиля пользователя»(Вызов осуществляется через slash-команду[16] или контекстное меню[17]);

– Реализация просмотра списка NFT, которыми владеет пользователь, которые продает пользователь, которые продаются на всем маркетплейсе(Вызовы осуществляются через контекстные меню, slash-команды, кнопки[18] в профиле пользователя. Список выглядит по-разному в зависимости от количества NFT, если NFT больше определенного количества, то будет подгружаться только часть в целях оптимизации ресурсов и листаться это множество будет через меню выбора[19]);

– 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 // пока что не сделано, но сделан метод в smart-контракте;

– Поддержать сервис с генеративно-состязательной сетью в discord-bot // 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

[1)] В данном контексте класс Transaction[14]

и маркетплейса. В 5 разделе идет описание трудностей и их решение в разработке discord-бота.

3 Обзор существующих работ и решений

3.1 Маркетплейсы

НВ данный момент существует большое количество NFT маркетплейсов: opensea[20], rarible[21], solanart[22]. Если брать маркетплейсы только на базе NEAR Protocol, тогда существуют такие примеры как: Paras[23], Mintbase[24], остановимся на них поподробнее.

3.1.1 Paras Paras является наиболее популярным и представляет следующий набор функций: посмотреть какие NFT выставлены на продажу, посмотреть купленные NFT, купить NFT, продать NFT, создать новую коллекцию NFT. На площадке представлены следующие виды NFT: пиксель-арты, иллюстрации, абстрактные картины, картины разных персонажей, фотографии. Все объекты можно отсортировать по убыванию или возрастанию цены.

Smart-контракты Paras лежат в открытом доступе[25, 26].

// Тут что-то про smart-контракты

Paras использует сервис fleek, этот


```
{
    "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"
}
```


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

3.1.2 Mintbase Mintbase является менее популярным маркетплейсом, однако он предоставляем гораздо больше категорий NFT, но все ключевые функ-

ции такие же. В качестве новых категорий выступают: 3d изображение, gif, профессиональные фотографии, аудиодорожки, произведения художников.

3.2 Генеративно-состязательные сети

4 Smart-контракты

4.1 Структура NFT smart-контракта

В данной главе я опишу строение NFT smart-контракта, написанного на языке Rust. Вся логика соответствует описанному стандарту NEP-171[27].

4.1.1 Near sdk фреймворк Опишу основные функции, структуры, декараторы, которые используются при написании smart-контрактов. Для этого необходим фреймворк near-sdk[7].

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 */
```

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 */
```

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 */
```

4.1.2 Core Functionality Для начала опишем основные структуры и функции[28], 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,                                   /*  REQUIRED (version like
↪  "nft-1.0.0") */

    pub name: String,                                   /*  REQUIRED (like "Maxim")  */

    pub symbol: String,                                 /*  REQUIRED (like like "MOCHI" or
↪  "MV3")  */

    pub icon: Option<String>,                           /*  small image associated with this
↪  contract (Data URL)  */

    pub base_uri: Option<String>,                       /*  Centralized gateway known to
↪  have reliable access to
                                                            decentralized storage assets
↪  referenced by `reference` or `media` URLs  */

    pub reference: Option<String>,                      /*  URL to a JSON file with more
↪  infoa link to a valid JSON file containing
                                                            various keys offering
↪  supplementary details on the token  */

    pub reference_hash: Option<Base64VecU8>,            /*  Base64-encoded sha256 hash of
↪  JSON from reference field. Required if `reference` is included.  */
}

#[derive(BorshDeserialize, BorshSerialize, Serialize, Deserialize)]
#[serde(crate = "near_sdk::serde")]
pub struct TokenMetadata {
    pub title: Option<String>,                          /*  ex. "Arch Nemesis: Mail Carrier"
↪  or "Parcel #5055"  */

    pub description: Option<String>,                    /*  free-form description  */

    pub media: Option<String>,                          /*  URL to associated media,
↪  preferably to decentralized, content-addressed storage  */

    pub media_hash: Option<Base64VecU8>,                /*  Base64-encoded sha256 hash of
↪  content referenced by the `media` field. Required if `media` is included.  */

    pub copies: Option<u64>,                            /*  number of copies of this set of
↪  metadata in existence when token was minted.  */

    pub issued_at: Option<u64>,                         /*  When token was issued or minted,
↪  Unix epoch in milliseconds  */

    pub expires_at: Option<u64>,                        /*  When token expires, Unix epoch
↪  in milliseconds  */

    pub starts_at: Option<u64>,                         /*  When token starts being valid,
↪  Unix epoch in milliseconds  */

    pub updated_at: Option<u64>,                        /*  When token was last updated,
↪  Unix epoch in milliseconds  */

    pub extra: Option<String>,                          /*  anything extra the NFT wants to
↪  store on-chain. Can be stringified JSON.  */

    pub reference: Option<String>,                      /*  URL to an off-chain JSON file
↪  with more info.  */
```

```
    pub reference_hash: Option<Base64VecU8>,                    /*  Base64-encoded sha256 hash of
↪   JSON from reference field. Required if `reference` is included.  */
}

#[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>
}

/* Struct that which can be requested via view call */
#[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>
}
```

Структура NFT 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:

```
#[near_bindgen]
#[derive(BorshDeserialize, BorshSerialize, PanicOnDefault)]
pub struct Contract {
    pub owner_id: AccountId,                                    /* Contract owner */
    pub tokens_per_owner: LookupMap<AccountId, UnorderedSet<TokenId>>,      /* Get all tokens by
↪   account_id */
    pub tokens_by_id: LookupMap<TokenId, Token>,                /* Token struct by
↪   token_id */
    pub token_metadata_by_id: UnorderedMap<TokenId, TokenMetadata>,         /* Token metadata by
↪   token_id */
    pub metadata: LazyOption<NFTContractMetadata>,              /* Contract metadata */
}
```

Структура контракта хранит:

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 рассказно во главе Royalties). Так как это payable функция, то пользователь должен будет внести депозит для хранения информации о добавляемом токене. Лишний депозит вернется пользвателю обратно.

Псевдокод будет выглядеть следующим образом:

```
#[payable]
pub fn nft_mint(
    &mut self,
    token_id: TokenId,
    metadata: TokenMetadata,
    receiver_id: AccountId,
    perpetual_royalties: Option<HashMap<AccountId, u32>>
) {
    /* Сохранить начальный storage_usage*/
    let initial_storage_usage = env::storage_usage();

    /* Распаковать и положить perpetual_royalties */
    royalty = AcceptRoyalties(perpetual_royalties);

    /* Создать токен */
    let token = Token {
        owner_id: receiver_id,
        approved_account_ids: Default::default(),
        next_approval_id: 0,
        royalty
    };

    /* Проверить, что такого token_id не существует */
    assert!(self.tokens_by_id.insert(&token_id, &token).is_none());

    /* Добавить токен в необходимые структуры */
    self.token_metadata_by_id.insert(&token_id, &metadata);
    self.add_token_to_owner(&token.owner_id, &token_id);

    /* Вернуть неиспользованный депозит */
    let required_storage_in_bytes = env::storage_usage() - initial_storage_usage;
    refund_deposit(required_storage_in_bytes);
}
```

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 струкуту описанную выше или None, если такого токена не существует.

```
fn nft_token(&self, token_id: TokenId) -> Option<JsonToken> {
    if let Some(token) = self.tokens_by_id.get(&token_id) {
        let metadata = self.token_metadata_by_id.get(&token_id).unwrap();
        Some(JsonToken {
            token_id,
            owner_id: token.owner_id,
            metadata,
            approved_account_ids: token.approved_account_ids,
            royalty: token.royalty
        })
    } else {
        None
    }
}
```

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:

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:

```
fn nft_on_transfer(
    &mut self,
    sender_id: AccountId,
    previous_owner_id: AccountId,
    token_id: TokenId,
    msg: String,
) -> Promise;
```

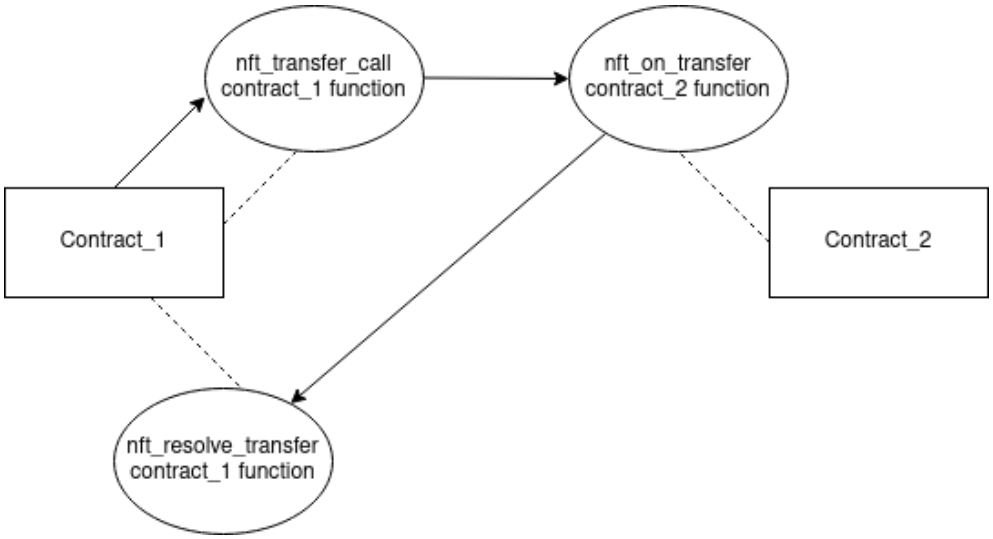

Рисунок 4.1. nft_transfer_call

```rust
#[payable]
fn nft_transfer(
    &mut self,
    receiver_id: AccountId,
    token_id: TokenId,
    approval_id: Option<u64>,
    memo: Option<String>,
) {
    let calle_id = env::predecessor_account_id();
    let prev_token = self.internal_transfer(&calle_id, &receiver_id, &token_id, approval_id, memo);
}

#[payable]
fn nft_transfer_call(
    &mut self,
    receiver_id: AccountId,
    token_id: TokenId,
    approval_id: Option<u64>,
    memo: Option<String>,
    msg: String,
) -> PromiseOrValue<bool> {

    /* Сохраняем отправителя и копию токена до отправки */
    let sender_id = env::predecessor_account_id();
    let previous_token = self.internal_transfer(&sender_id, &receiver_id, &token_id, approval_id,
↪   memo.clone());

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

    /* Вызываем nft_on_transfer на другом контракте, потом nft_resolve_transfer на своем */
    reciever_contract::nft_on_transfer(
        sender_id,
        previous_token.owner_id.clone(),
        token_id.clone(),
        msg,
        receiver_id.clone(),
```

```
        NO_DEPOSIT,
        env::prepaid_gas() - GAS_FOR_NFT_TRANSFER_CALL
    ).then(
        my_contract::nft_resolve_transfer(
            authorized_id,
            previous_token.owner_id,
            receiver_id,
            token_id,
            previous_token.approved_account_ids,
            memo,
            env::current_account_id(),
            NO_DEPOSIT,
            GAS_FOR_RESOLVE_TRANSFER
        )
    ).into()
}

#[private]
fn nft_resolve_transfer(
    &mut self,
    authorized_id: Option<String>,
    owner_id: AccountId,
    receiver_id: AccountId,
    token_id: TokenId,
    approved_account_ids: HashMap<AccountId, u64>,
    memo: Option<String>
) -> bool {

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

    /* Иначе возвращаем токен обратно владельцу */
    log!("Return token {} from {} to {}", token_id, receiver_id, owner_id);

    self.internal_remove_token_from_owner(&receiver_id.clone(), &token_id);
    self.internal_add_token_to_owner(&owner_id, &token_id);
    token.owner_id = owner_id.clone();
    refund_approved_account_ids(receiver_id.clone(), &token.approved_account_ids);
    token.approved_account_ids = approved_account_ids;
    self.tokens_by_id.insert(&token_id, &token);

    false
}
```

4.1.3 Enumeration Для удобного взаимодействия с контрактом, необходимо добавить больше view функций с pagination для просмотра NFT токенов[29]:

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[30]. 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:

```
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 могу испортить всю логику на других smart-контрактах. Подробнее такие краивые случаи описаны в стандарте[30].

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[31]. Для этого у нас есть поле royalty в структуре Token, которая отображет 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 структу токен */
    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.

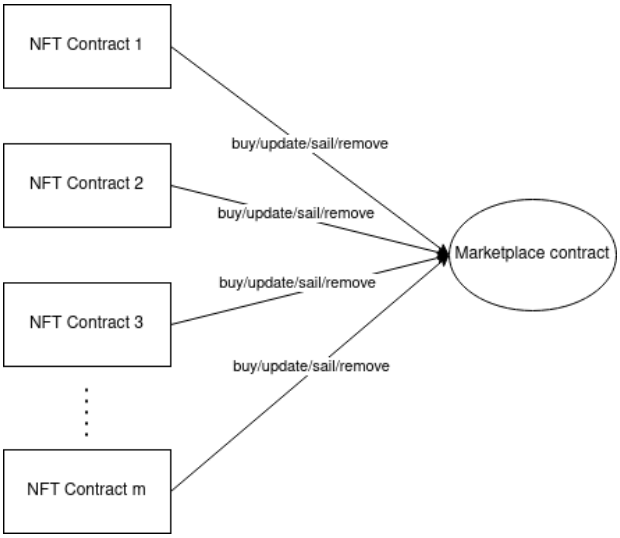

Рисунок 4.2. Marketplace core functionality

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 обосновано в главе Approval Management */
    pub approval_id: u64,

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

    /* Идендификатор 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 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.

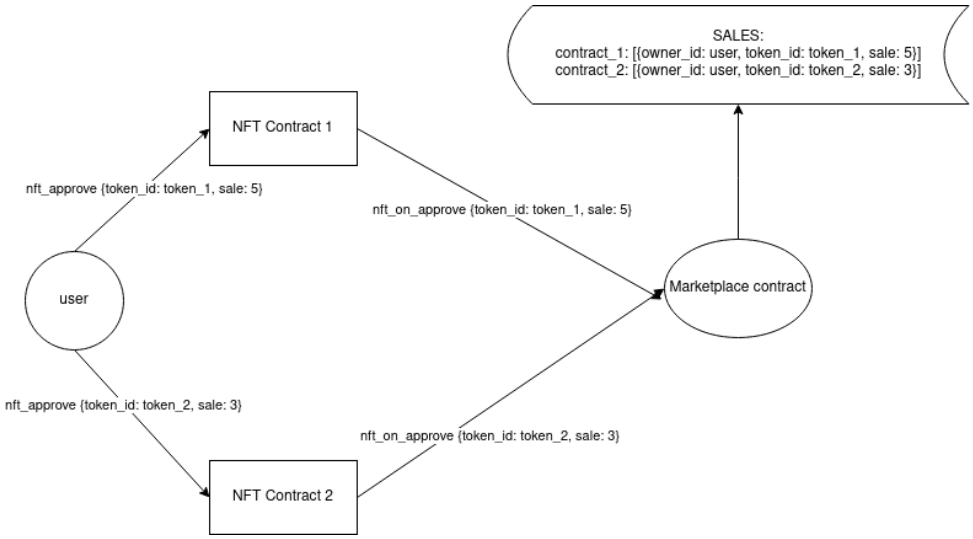

Рисунок 4.3. Marketplace contract sell

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::precessor_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::precessor_account_id());

    /* Обновим цену продажы 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 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.

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
        }
    }

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

-

6 Генеративно-состязательная сеть

7 Сервис с генеративно-состязательной сетью

8 Выводы и результаты

9 Список источников

[1] *NEAR Protocol*. 2022. URL: https://near.org/.

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

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

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

[5] Solana Foundation. *Introduction*. 2022. URL: https://spl.solana.com/.

[6] NEAR Protocol. *Transaction*. 2022. URL: https://docs.near.org/docs/concepts/transaction.

[7] NEAR Protocol. *Near/near-sdk-rs: Rust library for writing near Smart Contracts*. 2022. URL: https://github.com/near/near-sdk-rs.

[8] NEAR Protocol. *Near/near-sdk-as: AssemblyScript library for writing near Smart Contracts*. 2022. URL: https://github.com/near/near-sdk-as.

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

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

[11] NEAR Protocol. *Class KeyStore*. 2022. URL: https://near.github.io/near-api-js/classes/key_stores_keystore.keystore.html.

[12] Redis Ltd. *Redis*. 2022. URL: https://redis.io/.

[13] Roketo Labs LTD. *Near wallet*. 2022. URL: https://wallet.near.org/.

[14] NEAR Protocol. *Class transaction*. 2022. URL: https://near.github.io/near-api-js/classes/transaction.transaction-1.html.

[15] NEAR Protocol. *Class action*. 2022. URL: https://near.github.io/near-api-js/classes/transaction.action.html.

[16] discord.js. *discord.js Guide, slash commands*. 2022. URL: https://discordjs.guide/interactions/slash-commands.html#registering-slash-commands.

[17] discord.ts. *discord.ts official documentation, context menu*. 2022. URL: https://discord-ts.js.org/docs/decorators/gui/context-menu/.

[18] discord.js. *discord.js guide, buttons*. 2022. URL: https://discordjs.guide/interactions/buttons.html#building-and-sending-buttons.

[19] discord.js. *discord.js Guide*. 2022. URL: https://discordjs.guide/interactions/select-menus.html#building-and-sending-select-menus.

[20] Inc OpenSea Ozone Networks. *OpenSea, the largest NFT Marketplace*. 2022. URL: https://opensea.io/.

[21] Inc Rarible. *NFT Marketplace*. 2022. URL: https://rarible.com/.

[22] Solanart. *Solanart - discover, collect and trade nfts*. 2022. URL: https://www.solnaart.com/.

[23] Paras. *NFT Marketplace for digital collectibles on near*. 2022. URL: https://paras.id/.

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

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

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

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

[28] *core-functionality*. 2022. URL: https://nomicon.io/Standards/Tokens/NonFungibleToken/Core.

[29] *enumeration-standard*. 2022. URL: https://nomicon.io/Standards/Tokens/NonFungibleToken/Enumeration.

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

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

10 Приложения

10.1 Ссылка на репозиторий

Ссылка на Gitlab репозиторий с проектом - Gitlab