Lucky Train: Документация

1. Введение

2. Общая архитектура

3. Спецификация смарт-контракта

4. Use Cases

1. Введение

1.1. Общее описание проекта

***Lucky Train** — это децентрализованный DeFi-проект в экосистеме TON (The Open Network), который предоставляет пользователям простой и удобный способ стейкинга собственного токена **TrainCoin**. Ключевым элементом проекта является механика **«одноразовых билетов»** (Tickets), позволяющая реализовать **двойной burn**(двойное сжигание токенов) и поддерживать баланс интересов между участниками и командой.*

*Основная идея Lucky Train — дать держателям TrainCoin возможность **получать вознаграждение за блокировку (стейкинг) своих токенов** при условии, что пользователи покупают специальные билеты разного класса (Standard, Premium, VIP). После истечения заданного срока блокировки пользователи могут забрать свою залоченную сумму вместе с заранее определённой наградой.*

*Проект ориентирован на широкий круг пользователей, в том числе тех, кто **впервые** знакомится со стейкингом в TON. Для упрощения взаимодействия Lucky Train будет доступен через **мини-приложение в Telegram** (Telegram Mini-App), в котором можно:*

- ***Подключить кошелёк** (TonConnect или встроенный Telegram-кошелёк),*
- ***Приобрести TrainCoin** (через интеграцию с DEX),*
- ***Купить билет** и **застейкать** свои токены,*
- ***Получить награду** (claim) по истечении срока стейка.*

*Особенность «одноразовых билетов» заключается в том, что каждый билет позволяет совершить ровно один цикл стейкинга. При завершении стейка билет «исчерпывается» и удаляется, а пользователь, желая вновь участвовать, покупает новый билет. Таким образом стимулируется дополнительный спрос на TrainCoin и происходит **дефляция**(уменьшение общего количества токенов в обращении) благодаря встроенному механизму сжигания на двух этапах (при покупке билета и при входе в стейк).*

*Вся бизнес-логика (покупка билета, стейк, начисление и выплата наград, burn-транзакции) реализована внутри **смарт-контракта TrainCoin Staking**. Он защищает интересы стейкеров за счёт принципа «никогда не использовать залоченные стейковые токены в иных целях» (баланс контракта не может быть уменьшен ниже суммы всех застейканных средств). Администраторы могут контролировать параметры билетов, пополнять*

или выводить награды из пула, однако не могут нарушить защиту principal пользователей.

Таким образом, Lucky Train объединяет в себе:

- *Удобную механику **стейкинга**,*
- *Прозрачность и доверие, достигаемые **смарт-контрактом** в сети TON,*
- ***Дефляционный** эффект для TrainCoin через двойной burn,*
- ***Простой интерфейс** в формате Telegram Mini-App, доступный даже для новичков.*

1.2. Цели и задачи

*Главная цель **Lucky Train** — предложить держателям TrainCoin **простой, безопасный и выгодный способ стейкинга**, ориентированный как на опытных пользователей, так и на новичков в экосистеме TON. Для достижения этой цели проект решает ряд ключевых задач:*

1. ***Упростить вход в DeFi и стейкинг***
   - *Использование Telegram Mini-App снимает барьер для новых пользователей.*
   - *Интеграция с DEX для покупки токена «в один клик» и подключение через TonConnect или встроенный кошелёк Telegram упрощают процесс.*

2. ***Гарантировать защиту пользовательских средств***
   - *Механизм «balance ≥ totalStaked» в смарт-контракте исключает риск потери principal, даже если в пуле наград временно недостаточно токенов.*
   - *Возможность паузы контракта (pauseContract) и ручного пополнения пула наград (depositReward) позволяют гибко реагировать на непредвиденные обстоятельства.*

3. ***Создать прозрачные и фиксированные условия вознаграждений***
   - *Каждый «билет» имеет чётко заданные параметры: срок блокировки (durationDays), процент награды (rewardPercent), процент сжигания (burnStakePercent).*
   - *Пользователь заранее знает, на каких условиях и какой доход он может получить по окончании стейка.*

4. ***Обеспечить устойчивую дефляционную модель***

- ***Двойной burn*** *(при покупке билета и при входе в стейк) уменьшает общее количество токенов в обращении.*
- *Это способствует поддержанию и потенциальному росту ценности TrainCoin на рынке.*

5. ***Гибкость управления и адаптация под рынок***
   - *Админ-функции (setTicketParams, setBuyDistribution и т.д.) позволяют корректировать цены билетов, проценты наград, распределение burn/pool/team и другие параметры без перезапуска контракта.*
   - *При необходимости администратор может оперативно реагировать на изменения рыночной конъюнктуры или потребности сообщества.*

6. ***Стимулировать развитие экосистемы TON***
   - *Повышение ликвидности и спроса на TrainCoin за счёт механики стейкинга и удобного доступа к DEX.*
   - *Интеграция с TonConnect и Telegram Mini-App делает TON-экосистему более доступной для массового пользователя.*

Таким образом, Lucky Train ставит перед собой задачу ***совместить удобство и надёжность*** *для участников стейкинга с* ***дефляционной моделью****, выгодной для долгосрочных держателей TrainCoin, и при этом поддержать рост всей экосистемы TON.*

1.3. Краткое описание TrainCoin

TrainCoin — это нативный токен экосистемы Lucky Train, выпущенный в формате **Jetton** в блокчейне **TON** (The Open Network). Он служит базовым активом для всех ключевых операций в проекте (покупка билетов, стейкинг, формирование пулов наград), а также обеспечивает ряд преимуществ для его держателей:

1. **Фиксированный выпуск (supply)**
   - Эмиссия TrainCoin ограничена заранее установленным количеством.
   - Дополнительный выпуск токенов (re-mint) не предусмотрен, что снижает риск инфляции.

2. **Совместимость со стандартом Jetton**
   - TrainCoin соответствует общепринятым стандартам TON Jetton, поэтому любой пользователь, владеющий TON-кошельком, может безопасно его хранить и передавать.

- Переводы TrainCoin осуществляются через стандартные методы (`transfer`, `onJettonTransfer`) с учётом логики смарт-контрактов TON.

3. **Дефляционный механизм через «Double Burn»**
   - В смарт-контракте стейкинга Lucky Train часть токенов **сжигается** (burn) при покупке билета (buyTicket) и при входе в стейк (stakeTicket).
   - Это приводит к постепенному снижению общего количества TrainCoin в обращении, потенциально повышая его ценность.

4. **Ключевая роль в стейкинге**
   - Для участия в стейкинге (получения наград) пользователям необходим именно TrainCoin.
   - Благодаря встроенной интеграции с децентрализованной биржей (DEX) в Telegram Mini-App, любой желающий может быстро и удобно приобрести TrainCoin, чтобы начать стейк.

5. **Использование в экосистеме Lucky Train**
   - TrainCoin не только участвует в стейкинге, но и может быть применён в других будущих сервисах или партнёрских проектах экосистемы Lucky Train.
   - Планируется продолжать развивать утилитарную ценность токена, сохраняя при этом прозрачную и безопасную модель владения.

Таким образом, TrainCoin — это фундаментальный актив, лежащий в основе механики Lucky Train. За счёт фиксированного выпуска и постоянного сжигания (Double Burn) он создаёт устойчивую дефляционную среду и одновременно стимулирует пользователей к долгосрочному участию в проекте.

2. Общая архитектура

2.1. Компоненты системы

В экосистеме Lucky Train участвуют несколько ключевых компонентов, обеспечивающих совместную работу и удобство для пользователей:

1. **Смарт-контракт стейкинга (TrainCoin Staking Contract)**
   - Основной бизнес-логикой (покупка билетов, двойной burn, защита principal, начисление и выплата наград) управляет смарт-контракт на базе TON.
   - Поддерживает приём и отправку Jetton-токенов (TrainCoin) и обрабатывает специальные payload'ы (op-коды) для различного функционала (buyTicket, stakeTicket, claim, admin-операции и т.д.).

2. **TrainCoin (Jetton Minter и Jetton Wallet)**
   - Токен TrainCoin развёрнут в сети TON.
   - При транзакциях используется стандартный механизм `transfer` / `onJettonTransfer`.
   - Процесс «Double Burn» реализуется внутри стейкинг-контракта, который взаимодействует с Jetton Minter, сжигая часть токенов.

3. **Telegram Mini-App**
   - Сервис, встраиваемый в Telegram, где пользователь может покупать и стейкать токены TrainCoin, а также получать вознаграждения (claim).
   - Обеспечивает дружелюбный интерфейс для подключения кошелька (через TonConnect или встроенный Telegram-кошелёк).
   - Позволяет пользователям оперативно покупать TrainCoin (через встроенную интеграцию DEX) и выполнять все действия (buyTicket, stakeTicket, claim) в одном месте.

4. **DEX (Децентрализованная биржа)**
   - Сторонний сервис или встроенный виджет, позволяющий менять TON (или другие токены) на TrainCoin.
   - Интегрируется в Mini-App, чтобы пользователи могли приобрести или продать TrainCoin быстро и безопасно, не покидая Telegram.

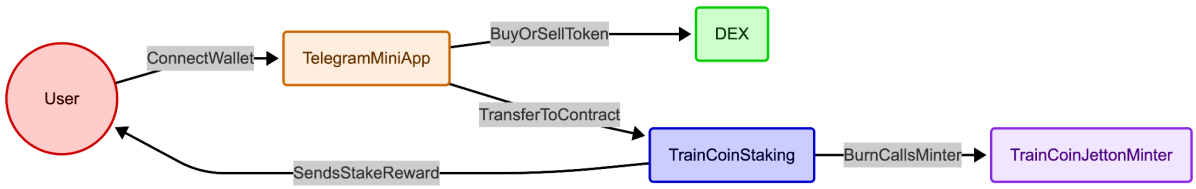
5. **Участники с различными ролями**
   - **Пользователи (Stakers)** совершают основные действия: покупают билеты, стейкают, делают claim.
   - **Owner/Администратор** управляет параметрами смарт-контракта (настройки билетов, пауза, вывод наград и т.д.).

- **Менеджеры** (Manager) имеют часть прав для оперативного управления (например, менять параметры билетов или пополнять пул), но не могут выводить награды.

Все эти компоненты работают совместно, обеспечивая надёжный стейкинг и удобный доступ к функционалу Lucky Train.

2.2. Схема взаимодействия

На высоком уровне взаимодействие компонентов выглядит так:



1. **Пользователь** открывает Telegram Mini-App и подключает кошелёк (через TonConnect или встроенный Telegram Wallet).

2. Если у пользователя нет TrainCoin, он может **купить токен на DEX**, используя TON или другой доступный актив, прямо из интерфейса Mini-App.

3. Для стейкинга пользователь совершает несколько ключевых операций:
   - **buyTicket**: отправка `price` токенов TrainCoin на смарт-контракт с payload `OP_BUY_TICKET`, что порождает одну из трёх билетов (Standard / Premium / VIP).
   - **stakeTicket**: отправка нужного количества TrainCoin на смарт-контракт с payload `OP_STAKE`, после чего часть токенов сжигается (`burnStakePercent`), а остаток уходит в `stakeAmount`.

- **claim**: вызов функции `claimTicket` (обычно через внешний метод или специальное сообщение), после чего смарт-контракт возвращает залоченную сумму и начисленную награду.

4. **Смарт-контракт** обрабатывает логику двойного burn (при покупке билета и при входе в стейк), пополняет reward-пул, отслеживает `totalStaked`, не позволяя вывести средства ниже этой суммы.

5. **Администратор** (Owner) может при необходимости:
   - Пополнить наградный пул (`depositReward`),
   - Вывести часть наград (`withdrawReward`), если это не нарушает защиту principal,
   - Менять параметры билетов и распределения (burn/pool/team),
   - Приостанавливать (pause) или возобновлять (resume) работу контракта, если возникнет необходимость.

Эта схема обеспечивает безопасное хранение стейковых токенов в самом смарт-контракте и удобный для пользователя процесс — все действия происходят внутри Telegram, без необходимости переходить на сторонние сайты.

2.3. Роли (User, Admin, Manager)

В Lucky Train предусмотрены несколько ролей, определяющих права и возможности в системе:

1. **Пользователь (User)**
   - Может покупать билеты (buyTicket), стейкать (stakeTicket) и забирать награду (claim).
   - Не имеет доступа к админ-функциям.
   - В большинстве случаев взаимодействует со смарт-контрактом через Telegram Mini-App, подтверждая транзакции в своём TON-кошельке.

2. **Администратор (Owner)**
   - Полноценный контроль над ключевыми настройками смарт-контракта:
     - Изменение параметров билетов (цены, сроки, награды, проценты burn и т.д.),
     - Управление распределением при покупке билета (setBuyDistribution),
     - Пауза/возобновление (pauseContract, resumeContract),
     - Вывод средств из пула наград (withdrawReward),

    - Назначение менеджеров, передача владельческих прав (transferOwnership).

- Имеет прямой доступ к критическим функциям, но **не может** нарушить защиту principal (баланс контракта не опустится ниже `totalStaked`).

3. **Менеджер (Manager)**
    - Может выполнять некоторые административные задачи:
        - Менять параметры билетов (setTicketParams),
        - Менять распределение цены билета (setBuyDistribution),
        - Пополнять пул наград (depositReward).

    - **Не** имеет права выводить награды (withdrawReward) или приостанавливать контракт (pause).
    - В назначении или снятии полномочий менеджера решающую роль играет владелец (Owner).

Такое разделение ролей обеспечивает баланс между гибкостью управления проектом и безопасностью средств пользователей.

3. Спецификация смарт-контракта (TrainCoin Staking)

3.1. Общие сведения

Смарт-контракт TrainCoin Staking является центральным элементом всей системы Lucky Train. Он отвечает за:

1. **Ведение учёта «билетов»** (tickets), с помощью которых пользователи участвуют в стейкинге.
2. **Приём и отправку токенов TrainCoin** по стандарту Jetton в TON (через метод `onJettonTransfer`), обрабатывая различные сценарии (покупка билета, стейк, пополнение пула наград и т.д.).
3. **Распределение токенов** при операциях buyTicket, stakeTicket, claimTicket, включая двойное сжигание (double burn).
4. **Защиту principal** застейканных токенов за счёт правила «balance контракта не может опуститься ниже totalStaked».
5. **Административные функции**, позволяющие Owner/Manager управлять параметрами билетов, распределением burn/pool/team, ставить контракт на паузу и т.д.

Чтобы реализовать эту логику, в смарт-контракте хранится ряд **ключевых переменных**:

- **`tickets`** (mapping или аналогичный механизм хранения)
  Хранит записи о каждом «билете» (ticketId). Каждая запись включает:
  - `owner` (адрес пользователя, которому принадлежит билет),
  - `classId` (тип билета: Standard, Premium, VIP и т.д.),
  - `stakeAmount` (количество токенов, реально залоченное в стейке),
  - `startTime` (время начала стейка).

- **`nextTicketId`** (счётчик)
  Используется для генерации уникальных идентификаторов билетов (ticketId) по мере их создания.

- **`totalStaked`** (целое число/uint)
  Общая сумма токенов, которые в данный момент активны в стейках. Помогает гарантировать, что баланс контракта не будет уменьшен ниже этой величины.

- **`rewardPoolBalance`** (целое число/uint)
  Логическая переменная, отражающая, сколько токенов доступно для

выплаты наград (не включая застейканный principal). Она пополняется при покупке билетов (часть цены уходит в пул) и при ручном депозите (depositReward).

- **Параметры для распределения** (например, `burnBuyPercent`, `poolBuyPercent` и т.д.)
  Задают пропорции сжигания и зачисления в пул наград/команде при покупке билета.

- **Механизмы проверки паузы**
  Состояние `paused`, указывающее, разрешены ли операции buyTicket и stakeTicket.

- **Управление ролями**
  - `owner` (адрес владельца контракта),
  - `managers` (mapping адресов, у которых часть административных прав).

Вся логика смарт-контракта сосредоточена вокруг обработки входящих Jetton-транзакций (method `onJettonTransfer`) и вызовов административных функций. В следующих подразделах мы подробнее рассмотрим, **как** именно работает каждая часть функционала: билеты, reward-пул, админ-настройки, безопасности и т.д.

3.2. Механика «одноразовых билетов» (Tickets)

В основе стейкинга Lucky Train лежит концепция «одноразовых билетов». **Билет** — это сущность (запись) в смарт-контракте, которая обеспечивает право на один цикл стейкинга: от покупки билета до финального claim. После завершения стейка билет удаляется, и для нового стейка нужно купить новый билет.

3.2.1. Создание билета: buyTicket

1. **Пользователь отправляет** ровно `price` токенов TrainCoin на смарт-контракт, добавляя payload с полем `op = OP_BUY_TICKET` и указанием `classId`.

2. Контракт проверяет:
    - Не находится ли контракт на паузе (если `paused = true`, то reject).
    - Совпадает ли фактически полученная сумма (transferedAmount) с ценой `price`, заданной для выбранного `classId`. Если нет — транзакция откатывается.

3. При успешном прохождении проверок:
    - Сжигается часть (например, `burnBuyPercent`) от `price`.
    - Часть (например, `poolBuyPercent`) зачисляется в reward-пул (`rewardPoolBalance`).
    - Остаток (например, `teamBuyPercent`) может быть переведён команде (teamAddress) «В случае необходимости администратор может изменить teamAddress через функцию setTeamAddress».

4. Создаётся новая запись в `tickets`:
    - `ticketId` = `nextTicketId`, после чего `nextTicketId` увеличивается.
    - `owner = msg.sender` (или адрес, которому принадлежит jetton-кошелёк).
    - `classId` = переданный пользователем.
    - `stakeAmount = 0` (пока не застейкано).
    - `startTime = 0` (стейк ещё не начат).

В результате пользователь становится владельцем «пустого» билета, который ещё надо активировать (застейкать токены).

3.2.2. Запуск стейка: stakeTicket

1. Пользователь снова отправляет перевод токенов TrainCoin на тот же смарт-контракт, указывая в payload `op = OP_STAKE` и `ticketId`. Количество токенов в этом переводе – это сумма, которую пользователь хочет залочить (stake).

2. Контракт проверяет:
    - Не на паузе ли контракт (`paused = false`)?
    - Существует ли указанный `ticketId` и принадлежит ли он пользователю (проверка `ticket.owner`)?

- У билета `stakeAmount` должно быть равно 0 (билет ещё не активирован).
- Не превышает ли переданная сумма лимит `maxStake`, указанный в параметрах соответствующего класса билета. Если `amount > maxStake`, то транзакция откатывается.

3. При успехе:
   - Сжигается часть токенов (`amount * burnStakePercent / 100`), остаток становится `stakeAmount`.
   - Устанавливается `ticket.stakeAmount = stakeAmount`.
   - `ticket.startTime = now()`.
   - Увеличивается `totalStaked` на `stakeAmount`.

Теперь билет **активирован**: стейк работает, пользователь ждёт окончания срока блокировки.

3.2.3. Продолжительность и завершение (claimTicket)

- У каждого класса билета (Standard, Premium, VIP) есть свой `durationDays`. После того как `now() >= startTime + durationDays*86400`, пользователь может сделать claim.

- **Claim** обычно вызывается не через onJettonTransfer, а через внешний метод (например, `claimTicket(ticketId)`) или специальное сообщение. При этом смарт-контракт:

  1. Проверяет, что срок стейка уже истёк. Если нет — reject.

  2. Вычисляет награду: `reward = (stakeAmount * rewardPercent) / 100`.

  3. Убедится, что в `rewardPoolBalance` достаточно средств для выплаты награды, и что операция не опустит баланс контракта ниже `totalStaked - stakeAmount`.

  4. Если средств достаточно, то:
     - `rewardPoolBalance -= reward`.
     - `totalStaked -= stakeAmount`.
     - Переводит пользователю `(stakeAmount + reward)`.
     - Удаляет запись о билете (`delete tickets[ticketId]`).

После этого билет считается **«исчерпанным»** (completed): он физически удаляется, и если пользователь хочет стейкать дальше, ему надо заново купить билет (`buyTicket`).

Нет механизма раннего выхода (early unstake): досрочное снятие невозможно.

3.2.4. Зачем «одноразовые билеты»?

1. **Дополнительный дефляционный эффект**
   - Каждый раз при покупке нового билета пользователь платит `price`, часть которого сжигается. Чем чаще люди стейкают, тем больше токенов выгорает.

2. **Стимулирование спроса**
   - Пользователь не может «вечным» образом стейкать один и тот же билет. Когда стейк заканчивается, для повторного участия требуется купить новый билет, тем самым поддерживая спрос на TrainCoin.

3. **Гибкая настройка классов билетов**
   - Каждый класс билета предлагает свои условия (срок, награда, burn).
   - Админ при необходимости может корректировать параметры (price, durationDays, rewardPercent, burnStakePercent), влияя на экономическую модель.

Таким образом, механика одноразовых билетов обеспечивает **цикличность**, стимулирует **дефляцию** и даёт проекту **гибкий инструмент** для регулирования стейкинговых условий.

3.3. Параметры классов билетов

В Lucky Train существует несколько классов билетов, отличающихся **ценой**, **сроком блокировки**, **процентом вознаграждения** и **процентом burn при входе в стейк**. По умолчанию предполагается три класса (Standard, Premium, VIP), однако их точные параметры и общее количество могут варьироваться.

1. **Стандартные атрибуты класса билета**:
   - **`price`**: стоимость билета в токенах TrainCoin, которую пользователь платит при `buyTicket`.
   - **`durationDays`**: срок блокировки (в сутках), в течение которого застейканные токены нельзя получить обратно без claim.

- **`rewardPercent`**: процент, по которому рассчитывается награда для пользователя по окончании стейка.
- **`burnStakePercent`**: доля, которая сжигается непосредственно при отправке токенов в стейк (`stakeTicket`).
- **`maxStake`**: максимальное количество токенов, которое пользователь может внести в этот билет при вызове stakeTicket. Если пользователь попытается застейкать больше, чем `maxStake`, транзакция будет отклонена.

2. **Пример параметров (предварительные значения)**:
   - **Standard**: price = 30 TrainCoin, durationDays = 7, rewardPercent = 10%, burnStakePercent = 2%, `maxStake = 1000`
   - **Premium**: price = 50 TrainCoin, durationDays = 14, rewardPercent = 15%, burnStakePercent = 3%, `maxStake = 5000`
   - **VIP**: price = 80 TrainCoin, durationDays = 28, rewardPercent = 25%, burnStakePercent = 5%, `maxStake = 10000`

3. **Гибкая настройка**:
   - Все параметры могут быть изменены **админ-функциями** (например, `setTicketParams(classId, newPrice, newDuration, newRewardPct, newBurnStakePct, newMaxStake)`).
   - Уже **купленные билеты** сохраняют параметры, действовавшие на момент покупки. Изменения затрагивают только **будущие** билеты, приобретаемые после смены настроек.
   - Так, если проект решит повысить награду для VIP-билетов или изменить срок блокировки для Standard, достаточно вызвать соответствующую админ-команду.

4. **Взаимодействие с логикой стейкинга**:
   - При `buyTicket`, смарт-контракт проверяет, что пользователь отправил ровно `price` в соответствии с `classId`.
   - При `stakeTicket`, именно `burnStakePercent` влияет на объём сжигаемых токенов.
   - При финальном `claim`, расчёт награды происходит по формуле:
     reward = (stakeAmount × rewardPercent) / 100
     а продолжительность блокировки определяется `durationDays`.

Таким образом, параметры билетов задают **экономическую модель** стейкинга (величину взноса, уровень вознаграждения, интенсивность burn) и могут адаптироваться под рыночные условия или стратегию развития проекта.

3.4. Reward Pool (пополнение и защита principal)

Одной из ключевых особенностей смарт-контракта Lucky Train является **разделение баланса** на две важные части:

1. **Залоченные токены (principal)**, соответствующие сумме всех текущих стейков (`totalStaked`).
2. **Пул наград (reward pool)**, откуда пользователям выплачиваются проценты (reward) при `claim`.

3.4.1. Reward Pool и его пополнение

- **Reward Pool** логически отражён переменной `rewardPoolBalance`.
- Пополняется двумя основными способами:

    1. **Часть цены билета при buyTicket**: когда пользователь покупает билет, некоторая доля от `price` (например, `poolBuyPercent`) зачисляется в пул наград.

    2. **Ручное пополнение админом** через операцию `depositReward(amount)`:

        - Администратор (или любой желающий) отправляет TrainCoin на смарт-контракт с payload `OP_DEPOSIT_REWARD`.
        - Контракт увеличивает `rewardPoolBalance` на `amount`.

- При этом сам пул наград хранится **на балансе смарт-контракта**, а не на отдельном адресе. Но мы учитываем его объём в переменной `rewardPoolBalance`, чтобы понимать, сколько средств доступно для выплат.

3.4.2. Защита principal (правило «баланс ≥ totalStaked»)

- Вся логика стейкинга опирается на гарантии, что **principal** (застейканная сумма) не может быть «задета» и всегда будет доступна пользователям для возврата.

- Для этого в смарт-контракте прописано правило:

  (текущий баланс контракта) ≥ totalStaked.

Это означает, что любая операция, которая пыталась бы вывести токены и оставить на контракте меньше, чем `totalStaked`, будет отклонена (revert).

- **Почему это важно**:

1. Если по каким-либо причинам в `rewardPoolBalance` не хватает токенов на выплату наград, пользователь всё равно может дождаться пополнения пула (админом или поступлениями от новых билетов).

2. Principal пользователя остаётся нетронутым до тех пор, пока он не сделает claim, и контракт проверяет, что действительно есть средства на возврат.

3.4.3. Выплаты из пула

- Когда пользователь делает `claim`, контракт вычисляет `reward = stakeAmount * rewardPercent / 100` и проверяет:

  rewardPoolBalance ≥ reward и (balance - reward) ≥ (totalStaked − stakeAmount).

  - Первое условие: в пуле наград достаточно средств, чтобы покрыть reward.
  - Второе условие: после выплаты пользовательского principal + reward баланс контракта не должен опуститься ниже `totalStaked - stakeAmount`.

- При успешной проверке средства отправляются пользователю, а `rewardPoolBalance` уменьшается на `reward`, `totalStaked` — на `stakeAmount`.

3.4.4. Административный вывод наград (withdrawReward)

- Аналогичные проверки действуют и при `withdrawReward(amount)`.
- Контракт не позволяет вывести больше, чем есть в пуле наград, и проверяет, не нарушит ли это правило balance ≥ totalStaked.
- Таким образом, администратор не может изъять **principal** пользователей.

3.4.5. Сценарии нехватки средств

- Если при `claim` награда (reward) не может быть выплачена (например, `rewardPoolBalance < reward`), транзакция откатывается (revert).
- Пользователь может дождаться пополнения пула (например, с помощью новых покупателей билетов или админской функции `depositReward`) и повторить `claim` позже.
- Principal при этом остаётся в контракте, не теряется.

За счёт такой архитектуры Lucky Train обеспечивает **чёткое разделение** между стейковыми средствами (principal) и пулом наград, а также гарантирует возврат основного залога даже в условиях временной нехватки средств в reward-пуле.

3.5. Админ-функции

Смарт-контракт Lucky Train поддерживает ряд специальных операций, доступных только владельцу контракта (**Owner**) или менеджерам (**Manager**) — в зависимости от уровня прав. Эти функции необходимы для гибкого управления экономикой билетов, параметрами стейкинга и пулом наград, а также для обеспечения безопасности проекта.

3.5.1. Управление параметрами билетов

1. **`setTicketParams(classId, newPrice, newDuration, newRewardPercent, newBurnStakePercent, newMaxStake)`**
   - Позволяет изменить экономические параметры существующего класса билетов (например, Standard, Premium, VIP).
   - Затрагивает **только будущие** билеты, которые будут куплены после изменения настроек.
   - Доступна **Owner** и **Manager** (это и есть основные права менеджера).

2. **`setBuyDistribution(burnPct, poolPct, teamPct)`**
   - Определяет, какая доля от цены билета (`price`) сжигается (`burnPct`), какая идёт в пул наград (`poolPct`), и какая — команде (`teamPct`).
   - Сумма трёх значений должна быть ровно 100%. Если не так — транзакция откатывается.
   - Позволяет оперативно настраивать экономику «покупки билетов».

- Доступно только **Owner**

3.5.2. Управление пулом наград

1. **`depositReward(amount)`**
   - Любой адрес может отправить TrainCoin на контракт с payload `OP_DEPOSIT_REWARD`, чтобы пополнить `rewardPoolBalance`.
   - Обычно это делает админ (Owner/Manager) или партнёрский проект.
   - Нет ограничений на использование этой функции в режиме паузы (можно пополнять пул всегда).

2. **`withdrawReward(amount)`**
   - Доступно **только Owner** (высший уровень доступа).
   - Позволяет вывести `amount` токенов из пула наград (уменьшая `rewardPoolBalance`).
   - Перед выводом контракт проверяет, не опустится ли общий баланс ниже `totalStaked`. Если попытка нарушает правило «balance ≥ totalStaked», операция откатывается.
   - Гарантирует, что вывод не затронет застейканные средства (principal).

3.5.3. Режим паузы (Pause / Resume)

- **`pauseContract()`**
  - Переводит контракт в режим «паузы».
  - Запрещает ключевые операции: `buyTicket` и `stakeTicket`. (Claim по уже застейканным билетам остаётся доступным.)
  - Используется в экстренных случаях (например, обнаружена критическая уязвимость).

- **`resumeContract()`**
  - Выводит контракт из режима паузы, восстанавливая нормальную работу.

Обе эти функции доступны **только Owner**.

3.5.4. Управление ролями (Owner / Manager)

1. **`transferOwnership(newOwner)`**
   - Только текущий `owner` может вызвать.
   - Передаёт права владельца контракта новому адресу `newOwner`.

2. **`addManager(addr)` / `removeManager(addr)`**
   - Только `owner` может назначать или отзывать права «менеджера».
   - Менеджеры обладают подмножеством прав (например, могут менять параметры билетов, но не могут делать `withdrawReward` или паузить/стартовать контракт).

3. **`setTeamAddress(newTeamAddress)`**
   - Доступно только Owner
   - Меняет адрес, на который отправляется «team-часть» при покупке билета.
   - Все новые операции `buyTicket` будут отправлять «team-часть» на этот обновлённый адрес.
   - Уже совершённые покупки не пересматриваются (teamAddress, который действовал в момент покупки, был окончательным). Применяется только к будущим операциям `buyTicket`

3.5.5. Принципы безопасности

- Несмотря на широкие полномочия Owner в части изменения параметров и вывода наград, **правило «balance ≥ totalStaked»** не даёт вывести залоченные пользовательские средства (principal).
- При режиме паузы пользователи всё ещё могут забрать стейк (claim), что гарантирует возвращение своих токенов в случае долгой «заморозки» проекта.
- Роль Manager позволяет делегировать часть административных задач, не давая полный доступ к средствам в reward-пуле.

Таким образом, админ-функции дают возможность **динамически адаптировать** экономические условия стейкинга и управлять пулом наград, сохраняя при этом **безопасность** средств стейкеров на контракте.

3.6. Технические детали onJettonTransfer (оп-коды)

Так как Lucky Train Staking реализован в виде **Jetton-смарт-контракта** (или контракта, совместимого со стандартом Jetton в TON), все операции с токенами (buyTicket, stakeTicket, depositReward и т.д.) происходят через функцию `onJettonTransfer` внутри контракта. При этом в payload (дополнительных данных) указываются **op-коды** (operation codes), которые определяют, какую именно операцию нужно выполнить.

1. **Формат вызова**
   - Когда пользователь или администратор отправляет перевод TrainCoin на адрес смарт-контракта, Jetton Minter (или Jetton Wallet) вызывает метод `onJettonTransfer(amount, sender, payload)` контракта Lucky Train Staking.
   - В поле `payload` ожидается специфический формат (например, сериализованная структура с полями `op`, `ticketId`, `classId` и т.д.).
   - Контракт внутри `onJettonTransfer` извлекает из `payload` нужные данные и решает, какую операцию (buyTicket, stakeTicket и т.д.) выполнить.

2. **Основные op-коды**
   - **`OP_BUY_TICKET`** (например, значение 1)
     - Используется при покупке билета (buyTicket).
     - Проверяется, что `amount` равен `price` выбранного класса билета, сжигается часть токенов, формируется запись о билете.

   - **`OP_STAKE`** (например, значение 2)
     - Используется для ввода токенов в стейк (stakeTicket).
     - Проверяется валидность билета (ticketId), сжигается `burnStakePercent`, остаток добавляется к `stakeAmount`, увеличивается `totalStaked`.

   - **`OP_DEPOSIT_REWARD`** (например, значение 3)
     - Пополнение пула наград (depositReward).
     - Любое количество токенов можно отправить, эти средства увеличивают `rewardPoolBalance`.

3. **Отсутствие прямого op-кода для claim**
   - Обычно `claimTicket` вызывается **не** через `onJettonTransfer`, а через внешний метод (или отдельный сообщение/вызов). Это

связано с тем, что при claim не нужно пересылать дополнительные токены на контракт.
    - Вызов `claimTicket(ticketId)` может быть оформлен как обработка внутреннего сообщения (internal message) или get-метода с последующим отсылом внешнего сообщения в блокчейн. Точный механизм зависит от реализации.

4. **Проверки и revert**
    - Внутри `onJettonTransfer` на каждом шаге выполняются обязательные проверки (pauseContract, корректность `amount`, валидность `ticketId`, совпадение суммы с `price` и т.д.).
    - При несоблюдении условий (например, неправильная сумма или несуществующий билет) вызывается `revert`, и транзакция откатывается.

5. **Гибкость и расширяемость**
    - Задав несколько op-кодов, контракт может удобно разделять логику покупки, стейка и пополнения пула.

Таким образом, работа с Lucky Train Staking основана на **стандартном процессе Jetton transfer** в TON, где **payload** определяет тип операции. Это упрощает интеграцию с кошельками и DApp-сервисами (TonConnect, Telegram Mini-App), позволяя автоматически формировать нужный payload при совершении перевода токенов.

3.7. Безопасность и Edge Cases

В смарт-контракте Lucky Train предусмотрен ряд механизмов, призванных обеспечить сохранность средств пользователей и стабильную работу в разных ситуациях.

3.7.1. Правило «баланс ≥ totalStaked»

- **Суть**: контракту запрещено любым действием опустить свой фактический баланс TrainCoin ниже суммы всех застейканных средств (`totalStaked`).
- **Значение**: гарантируется защита principal: даже если пул наград (rewardPoolBalance) окажется пуст, у пользователей остаются их застейканные токены (stakeAmount).
- **Реализация**:

- При попытке `withdrawReward`, контракт проверяет, останется ли после вывода достаточно средств для покрытия `totalStaked`. Если нет — `revert`.
- При `claimTicket`, перед выплатой reward происходит аналогичная проверка (нельзя превысить лимит, чтобы не затрагивать principal других пользователей).

3.7.2. Паузный режим (Pause)

- **Назначение**: при экстренных обстоятельствах (обнаружении бага, подозрительных транзакциях и т.д.) администратор может вызвать `pauseContract()`.

- **Последствия**:
  - Новые билеты не покупаются (запрещён `buyTicket`).
  - Невозможно начать новый стейк (`stakeTicket`).
  - Однако пользовательские **claim**-операции доступны, чтобы не блокировать возможность вывести уже застейканные средства.

- **Безопасность**:
  - Позволяет быстро «заморозить» ввод новых средств, пока команда не устранит проблему.
  - Защищает контракт от дальнейших потенциально опасных операций.

3.7.3. Проверки параметров (price, stakeAmount, durations, maxStake)

- При `buyTicket` контракт требует, чтобы сумма перевода (`amount`) **строго** совпадала с `price` класса билета. Если нет — `revert("Incorrect ticket price")`.
- При `stakeTicket` нужно, чтобы `stakeAmount` ещё не был установлен (т.е. билет не активирован) и чтобы пользователь передавал токены на **свой** билет (`owner` совпадает).
- При `stakeTicket` дополнительно проверяется, чтобы `amount <= maxStake` для данного класса билета. Если `amount > maxStake`, вызывается `revert("Stake amount exceeds maxStake")`.
- `durationDays`, `rewardPercent`, `burnStakePercent` и т.д. должны быть валидными и установленными админом заранее. Неверный

`classId` или некорректное распределение (сумма burn/pool/team ≠ 100%) приводят к откату транзакции.

3.7.4. Недостаток средств в пуле наград

- Если при `claim` выясняется, что `rewardPoolBalance < calculatedReward`, транзакция откатывается. Пользователь не может получить награду до тех пор, пока пул не будет пополнен либо не поступят новые средства от покупки билетов.
- Principal (застейканная сумма) от этого **не страдает**: пользователь может просто подождать пополнения пула и сделать `claim` повторно.

3.7.5. Уязвимости, связанные с ролями

- Только `owner` может передавать владение (`transferOwnership`), назначать/удалять менеджеров (addManager/removeManager) и выводить награды (`withdrawReward`).
- Менеджеры не имеют права выводить награды или ставить контракт на паузу.
- Если `owner` ключ скомпрометирован, злоумышленник не сможет украсть застейканные токены (т.к. действует правило «баланс ≥ totalStaked»), но может вывести награды из пула (если они не нужны для покрытой части principal).

3.7.6. Edge Cases

- **Повторный стейк в один билет**: запрещён, т.к. `stakeAmount` должен быть 0 до первого стейка и не может быть изменён после. Нужно покупать новый билет для каждого нового стейка.
- **Пауза во время стейка**: не препятствует claim, пользователь может вывести средства, когда наступит время. Но купить новый билет или застейкать заново не удастся, пока пауза не снята.
- **Дробные награды**: используется целочисленная арифметика, при вычислении `reward = (stakeAmount * rewardPercent) / 100` должно происходить округление вниз и дробные части процента теряются в пользу контракта.

Все описанные проверки и механизмы в совокупности минимизируют риск для пользователей, позволяя гарантированно вернуть их залоченные токены и избежать атаки на пул наград. При этом админ имеет достаточно инструментов, чтобы вовремя реагировать на необычные ситуации (pauseContract) и управлять экономическими параметрами (setTicketParams, setBuyDistribution).

3.8. События (Events), (Опционально)

Для прозрачности и удобства отслеживания процессов внутри смарт-контракта Lucky Train предусматривается генерация событий (Events) при ключевых операциях. В экосистеме TON это может быть реализовано через внутренние сообщения с нужными метаданными или иным способом (зависит от реализации). Ниже перечислены основные события, которые рекомендуется генерировать:

1. **TicketBought(ticketId, user, classId, price)**
   - Генерируется при успешной покупке билета (`buyTicket`).
   - Содержит:
     - `ticketId` — уникальный идентификатор только что созданного билета.
     - `user` — адрес (или TON-кошелёк), у которого теперь есть билет.
     - `classId` — класс билета (Standard, Premium, VIP…).
     - `price` — количество токенов TrainCoin, уплаченных за билет.

2. **TicketStaked(ticketId, user, stakeAmount)**
   - Генерируется при успешном входе в стейк (`stakeTicket`).
   - Содержит:
     - `ticketId` — идентификатор билета, который активировался для стейка.
     - `user` — адрес владельца билета.
     - `stakeAmount` — фактическая сумма токенов, залоченная в стейке (после вычета `burnStakePercent`).

3. **TicketClaimed(ticketId, user, rewardAmount)**
   - Генерируется при успешном выполнении `claimTicket`.
   - Содержит:
     - `ticketId` — идентификатор билета, по которому завершили стейк.

    - `user` — адрес владельца билета.
    - `rewardAmount` — количество токенов, полученных в качестве награды (не включая `stakeAmount`, который возвращается).

4. **RewardDeposited(amount, depositor)**
    - Генерируется, когда пул наград пополняется (операция `depositReward`).
    - `amount` — число токенов, добавленных в `rewardPoolBalance`.
    - `depositor` — адрес, совершивший пополнение (возможно, Owner/Manager или любой другой).

5. **RewardWithdrawn(amount, receiver)**
    - Генерируется, когда администратор (Owner) выводит токены из пула наград (`withdrawReward`).
    - `amount` — сумма выведенных токенов.
    - `receiver` — адрес, на который отправлены токены (как правило, это Owner).

6. **TicketParamsUpdated(classId, newPrice, newDuration, newRewardPercent, newBurnStakePercent)**
    - Генерируется при вызове `setTicketParams`.
    - Позволяет отслеживать изменения экономических условий для определённого класса билетов.

7. **BuyDistributionUpdated(newBurnPct, newPoolPct, newTeamPct)**
    - Генерируется при вызове `setBuyDistribution`.
    - Отражает обновление пропорций распределения цены билета.

8. **ContractPaused()** / **ContractResumed()**
    - Генерируются при вызове `pauseContract()` и `resumeContract()`.
    - Указывают на смену состояния контракта (работает / приостановлен).

9. **OwnershipTransferred(oldOwner, newOwner)**
    - Генерируется при вызове `transferOwnership`.
    - Показывает, что контроль над контрактом перешёл к новому адресу.

10. **ManagerAdded(addr)** / **ManagerRemoved(addr)**
    - Генерируются при добавлении или удалении менеджеров (addManager / removeManager).

- Позволяют отслеживать изменения в списке ответственных за часть админ-функций.

Зачем нужны события

- **Прозрачность**: любой участник рынка, биржа или аналитический сервис может подписываться на события и видеть, какие билеты куплены, какие стейки запущены и сколько наград выплачено.
- **Удобная интеграция**: сторонние приложения и боты могут реагировать на события (например, оповещать пользователя в Telegram, когда закончится срок стейка).
- **Аудит**: при проверке контракта легко увидеть в логе, какие админ-операции проводились (пауза, изменение параметров и т.д.).

Все эти события вместе дают полную картину о работе смарт-контракта, повышая уровень доверия и прозрачности для держателей TrainCoin и сообщества TON.

4. Use Cases

4.1. Пользовательские сценарии

В данном разделе описаны основные шаги, которые совершает рядовой пользователь (держатель TrainCoin), чтобы воспользоваться стейкингом в Lucky Train. Мы рассмотрим ключевые операции: покупка билета, стейк, получение награды, а также ситуацию, когда у пользователя изначально нет TrainCoin.

4.1.1. Приобретение TrainCoin (если пользователь его не имеет)

1. **Пользователь** заходит в Telegram Mini-App Lucky Train.
2. Видит опцию «Купить TrainCoin» (через встроенный DEX).
3. Выбирает, за какую валюту (TON или другой доступный токен) он хочет купить TrainCoin, указывает сумму, подтверждает сделку в своём кошельке (TonConnect или встроенный Telegram Wallet).
4. По окончании свопа TrainCoin поступает на его адрес. Теперь пользователь готов покупать билеты или стейкать.

(Если TrainCoin у пользователя уже есть, данный шаг пропускается.)

4.1.2. Покупка билета (buyTicket)

1. Пользователь в Mini-App выбирает класс билета (Standard, Premium, VIP) — ориентируясь на длительность (durationDays), процент награды (rewardPercent), стоимость (price) и др.
2. Нажимает «Купить билет», формируется транзакция **Jetton transfer** с payload `OP_BUY_TICKET` и параметром `classId`.
3. Пользователь **подтверждает** перевод `price` токенов в своём кошельке.
4. Смарт-контракт:
   - Проверяет, что сумма совпадает с `price` для данного `classId`.
   - Сжигает часть (burnBuyPercent), отправляет часть (poolBuyPercent) в rewardPoolBalance, остаток (teamBuyPercent) — по заложенной схеме.
   - Создаёт новую запись в `tickets` (ticketId), где `stakeAmount = 0`.

5. Пользователь видит, что билет успешно куплен, но пока **не активирован** для стейка.

4.1.3. Запуск стейка (stakeTicket)

1. Пользователь выбирает купленный билет (у которого `stakeAmount = 0`) и нажимает «Начать стейк».
2. Mini-App запрашивает у пользователя, сколько токенов он хочет залочить. Пользователь указывает сумму (X).
3. Происходит вторая **Jetton transfer**-транзакция с payload `OP_STAKE` и `ticketId`, где `amount = X`.
4. Контракт проверяет:
   - Не на паузе ли контракт,
   - Правильный ли `ticketId` и совпадает ли `owner` билета с тем, кто отправляет токены,
   - Что `stakeAmount` ещё не установлен (0).

5. При успехе:
   - Сжигается `(X * burnStakePercent) / 100`, остаток идёт в `stakeAmount`.
   - Записывается `startTime = now()`, `ticket.stakeAmount = stakeAmount`, увеличивается `totalStaked += stakeAmount`.

6. Пользователь видит, что билет **активирован** и начался отсчёт срока блокировки (например, 7, 14 или 28 дней — зависит от класса билета).

4.1.4. Получение награды (claim)

1. По истечении срока блокировки (durationDays) пользователь видит в Mini-App, что билет перешёл в статус «Доступен для клейма (claim)».
2. Нажимает «Claim» — Mini-App вызывает функцию `claimTicket(ticketId)` (не через onJettonTransfer, а через внешний метод или специальное сообщение).
3. Контракт проверяет:
   - `now() >= startTime + durationDays*86400`,
   - Есть ли в `rewardPoolBalance` достаточно средств, чтобы покрыть `reward = stakeAmount * rewardPercent / 100`,
   - После выплаты `(stakeAmount + reward)` баланс не опустится ниже `totalStaked - stakeAmount`.

4. Если проверка пройдена:
   - `rewardPoolBalance -= reward`,
   - `totalStaked -= stakeAmount`,

- Выплачиваются `(stakeAmount + reward)` назад на кошелёк пользователя,
- Билет **удаляется** из `tickets`.

5. Пользователь видит, что стейк завершён, а награда поступила на его адрес.

4.1.5. Повторное участие

- Если пользователь хочет продолжить стейк, то после одного полного цикла (buyTicket → stakeTicket → claim), ему нужно заново купить билет.
- Параллельно можно иметь несколько билетов (если пользователь покупает несколько раз). Каждый билет существует независимо.

4.1.6. Ситуация с паузой или нехваткой наград

- **Пауза**: Если контракт находится в режиме pause, пользователь не сможет **купить новый билет** или **начать стейк**, но всё ещё может сделать claim по уже запущенным стейкам.
- **Нехватка средств в пуле наград**: Если при claim не хватает токенов (rewardPoolBalance < reward), операция откатывается. Пользователь может подождать пополнения пула (через depositReward или покупки билетов другими участниками) и повторить claim позже. Principal при этом не потеряется.

Таким образом, **пользовательские сценарии** базируются на стандартных действиях стейкинга (покупка билета, запуск стейка, вывод награды), но обёрнуты в простое и доступное взаимодействие через Telegram Mini-App. При этом вся внутренняя бизнес-логика и безопасность обеспечиваются смарт-контрактом Lucky Train.

4.2. Админ-сценарии

Ниже описаны основные действия, которые могут совершать администраторы (Owner) и менеджеры (Manager) для управления и поддержания здоровой экономики смарт-контракта. Некоторые операции доступны только Owner, а часть функционала — как Owner, так и Manager (зависит от распределения ролей, заданного в контракте).

4.2.1. Пополнение пула наград (depositReward)

1. Администратор или любой желающий может отправить нужное количество TrainCoin на адрес контракта с payload `OP_DEPOSIT_REWARD`.
2. Контракт, получив это сообщение, увеличит `rewardPoolBalance` на величину отправленных токенов.
3. Такая операция не ограничена паузным режимом (можно пополнять пул даже если контракт приостановлен).

Зачем это нужно?

- Если пул наград исчерпывается, пользователи не могут получить вознаграждение (claim). Админ, заметив низкий уровень `rewardPoolBalance`, может пополнить его, чтобы поддержать продолжение нормальной работы стейкинга.

4.2.2. Вывод средств из пула наград (withdrawReward)

1. Только **Owner** может вызвать функцию `withdrawReward(amount)`.
2. Контракт проверяет, не нарушит ли вывод правила «баланс ≥ totalStaked». Если после вывода баланс опустится ниже `totalStaked`, операция откатывается (revert).
3. При успешной проверке `rewardPoolBalance` уменьшается на `amount`, а токены отправляются на адрес Owner.

Зачем это нужно?

- Если накопились избыточные средства в пуле наград, Owner может использовать часть из них на развитие проекта или иные нужды.
- Защита от вывода principal стейкеров остаётся неприкосновенной благодаря проверке `balance ≥ totalStaked`.

4.2.3. Изменение параметров билетов (setTicketParams)

1. Owner или Manager вызывает `setTicketParams(classId, newPrice, newDuration, newRewardPercent, newBurnStakePercent)`.
2. Контракт обновляет данные для указанного класса билетов.
3. Новые параметры действуют **только** для билетов, которые будут куплены после изменения. Уже купленные сохраняют прежние настройки.

Когда это нужно?

- Чтобы реагировать на рыночные изменения (например, увеличить награду для стимулирования стейкеров).
- Корректировать длительность, цену или burn-проценты для разных классов билетов.

4.2.4. Изменение распределения при покупке билетов (setBuyDistribution)

1. Только Owner вызывает `setBuyDistribution(burnPct, poolPct, teamPct)`.
2. Проверяется, что сумма трёх значений = 100%, иначе транзакция откатывается.
3. Контракт обновляет доли, которые будут сжигаться, уходить в reward-пул и команде при каждом `buyTicket`.

Пример:

- Было: `burnPct = 10, poolPct = 40, teamPct = 50`.
- Стало: `burnPct = 15, poolPct = 35, teamPct = 50`.

4.2.5. Пауза и возобновление (pauseContract / resumeContract)

1. Только **Owner** имеет право вызывать `pauseContract()` или `resumeContract()`.
2. При паузе запрещаются операции `buyTicket` и `stakeTicket`, но **claim** остаётся доступным.
3. При возобновлении действия восстанавливается обычный режим.

Когда это нужно?

- При обнаружении критических уязвимостей, некорректном поведении или плановом обновлении.
- Во время паузы можно устранять проблему, а пользователи смогут продолжать выводить свои застейканные средства (claim).

4.2.6. Управление ролями (Owner / Manager)

- **transferOwnership(newOwner)**: только текущий Owner может передать полный контроль новому адресу `newOwner`.
- **addManager(addr)** / **removeManager(addr)**: назначение и снятие прав менеджера. Менеджеры могут выполнять часть функций (настройка билетов, распределений, пополнение наградного пула) без доступа к выводу наград или паузе.

Дополнительные сценарии для Owner/Manager

- **Мониторинг баланса**: регулярно проверять `rewardPoolBalance` и `totalStaked`, чтобы вовремя пополнять пул или менять экономические параметры.
- **Аналитика**: отслеживать события, такие как `TicketBought`, `TicketStaked`, `TicketClaimed`, чтобы понимать активность пользователей.

Таким образом, админ-сценарии позволяют **тонко регулировать** параметры стейкинга и обеспечивают **защиту** интересов стейкеров, не давая вывести их principal, даже если у Owner есть доступ к части средств в пуле наград.