PDL Coursework

Lewis Gudgeon, Daniel Perez, Paul Pritz, Sam Werner

Feb 2023

1 Overview

In this coursework, you will implement an on-chain ticketing service that features three core components:

1. an NFT that represents tickets,
2. a primary marketplace that allows users to mint these tickets in exchange for an ERC20 token payment,
3. a secondary marketplace that allows holders of tickets to sell them to other users.

A code skeleton is available here: `https://gitlab.doc.ic.ac.uk/podl/2023/coursework-skeleton`

All components must be implemented as smart contracts in Solidity version 0.8.10 or higher.

1.1 The Code Skeleton

A code skeleton is provided to assist you. It contains interfaces that you must use, i.e. implement them by inheriting them in your smart contracts. You are free to add other smart contracts and as many functions as you see fit, provided the interfaces in the code skeleton are implemented. You may also implement several of the provided interfaces in the same contract.

The code skeleton also contains a contract that you should rely on in your solution, specifically an ERC20 token contract that should be used as a currency for purchasing tickets both in the primary as well as the secondary marketplace. This ERC20 token itself can be minted by sending ETH in the `mint` function of that contract. For testing, you should therefore call the `mint` function with some ETH value specified to receive ERC20 tokens.

The following sections provide a detailed specification of the components that you will need to implement.

2 Components - Smart Contracts

2.1 Ticket NFT

The ticket NFT that you should implement needs to adhere to the provided `ITicketNFT`. This interface is a reduced version of the ERC721 interface. Each ticket has the following associated metadata that should be stored in the contract:

- A unique ID for the ticket. Once a ticket ID has been allocated, this ID should not be reallocated.
- A string containing the name of the current ticket holder. It can only be changed by the holder of the ticket.
- A timestamp until which the ticket is valid. It should be set upon minting a ticket to be 10 days (i.e. $10 * 86,400$ seconds) after the time at which the ticket was minted.
- A boolean flag indicating whether the ticket has been used or not. Only the admin of the primary market contract should be able to modify this flag. Initially, it should be set to `false`. This flag should not be modifiable after the ticket has expired.

The ticket NFT interface contains getters to access this metadata, as well as setters to update the data that can be updated. All the functions that need to be implemented, as well as the ticket minting mechanism, are described in depth in the interface.

2.2 Primary Marketplace

The primary market contract should contain logic to mint a new ticket for a fixed price, fixed at $100e18$ units of the ERC20 token. Specifically, it should implement the function `purchase` that takes the initial holder's name as input and transfers ERC20 tokens from the purchaser to the admin of the contract, which should be set to the contract creator. Note that the purchaser will have to approve the amount before calling the `purchase` function. The number of tickets that can be purchased should be limited to $1,000$. There is only one kind of ticket. Only the primary market should be able to mint tickets.

2.3 Secondary Marketplace

The `SecondaryMarket` contract should implement logic to trade tickets between different users. Ticket holders should be able to list their ticket for sale using the `listTicket` function, where they specify a price denominated in the provided ERC20 token. When a ticket holder lists their ticket, this ticket should be transferred to the secondary market place contract. Bear in mind that you will need to maintain some record of who holds the listing.

Other users should then be able to purchase a listed ticket using the `purchase` function, where they provide a name and the ticket ID of the ticket they wish to purchase. The name associated with the purchased ticket should then be updated accordingly. The secondary marketplace contract should also enforce that no expired or used tickets can be sold. Should tickets expire while they are listed, they can no longer be sold. Note that the `purchase` function will need to transfer ERC20 tokens from the purchaser to the originator of the listing, which will need to be approved by the purchaser beforehand.

Furthermore, ticket holders should be able to delist their listed tickets using the `delistTicket` function. In this function, they will only need to provide the ticket ID of the listed ticket they wish to delist. The contract should ensure that only the owner of a listing can delist it.

A 5% fee should be charged on all sales made through the secondary market. This fee will be charged on the purchase price and will be paid by the user who listed the ticket, i.e., by reducing the amount they receive from a sale. The fees should go to the admin of the primary market contract.

3 Grading and Submission

You will be graded on the correct implementation of the functionality described above and on general code quality. Note that you will not be marked on the gas efficiency of your solution. All functions specified in the interfaces should be tested. Your final solution should be submitted as a zip file containing all the relevant source code and tests to CATE. This should compile. The files should respect the folder structure in the skeleton. An optional README can be added if you wish to communicate any particularities about the code.