PopTickets : A Decentralized Event Booking Application

Susmita Karyakarte
New York University
New York, USA
sk8663@nyu.edu

Abstract

PopTickets is an effort in decentralizing platforms for artists or their management agencies to connect directly to their fans and sell tickets to concerts or their events while eliminating bridging platforms like Ticketmaster. Such platforms, being centralized, frequently fail in providing availability of the platform at all times and meet demands for popular events. Also, they charge service fees and ticket transfer fees which change dynamically based on demand, giving rise to scalpers and causing inconvenience to genuine fans of the artist. Decentralization of such a platform via an application built on top of the Ethereum network can help to solve the above problems. In such an app if a fan wants to transfer their ticket to someone else they can simply enter the buyer's address and pay only the transaction fees instead of paying transfer fees to a centralized agency. Scalability of DApps is a matter of improved resource management [1]. Though under development, there are scaling techniques like sharding, state channels which can effectively allow DApps to be more scalable. A DApp using a yearly subscription business model for it's loyal customers could rule out the dynamic service fees of centralized agencies.

1 Implementation of the Proposal

The source code for the implementation can be found here. The application looks like any other website which allows a user to do the following:

1.1 Creation of the event and payment to the organizer

On choosing the "Add Your Event" option in the navigation bar on the website, the organizer can create an event by entering details such as name, venue, description and more (Start Date, End Date, Duration, Price of a single ticket, total tickets, image) of the event. On clicking "Submit", the organizer is then redirected to pay transaction fees via Metamask and confirm the transaction for creation of the event and then the event opens up for the public.

All the funds collected until the end of the event are stored in the balance of the smart contract which enables the refund and transfer features explained further. After the event the organizer can get access to the funds by clicking the "Get Funds" option. This also incurs a transaction and gas fees.

1.2 Storing of images and any changes to the event details

Images are stored as hashes on the blockchain and I have used the Pinata API service for the same. The service pins images to the IPFS network via a REST API. The options for "Change", "Delete/Cancel", "Halt" of the event are only available to the organizer and it is important for the organizer to maintain the same address which was used for creation of the event, as only this address has access to all the options related to the event. For "Change" option, the same form as creation of the event shows up and every single time the organizer submits the form a transaction will occur charging the organizer gas fees. Hence saving total transaction costs for the event is left up to the organizer. It would be better for the organizer to limit all event changes to one time.

1.3 Cancellation or Halting of the event

For any reason if the organizer wishes to halt selling of tickets to the event, the organizer can choose the "Halt" option and this would make fans unable to get any more tickets. Fans would also not be able to get tickets to the event after the total number of tickets are sold. For cancellation organizer can choose "Delete/Cancel" option and this would delete the event from the events list.

All the balance collected by selling of tickets will be refunded to the fans. Every ticket is a ERC-721 token with a unique ticket ID stored in a mapping with ticket ID mapping to it's buyer. The tickets are minted as non-fungible tokens following the ERC-721 standard using the existing smart contract provided by the OpenZeppelin library which also provides a way to burn NFTs. The tickets are burned after cancellation of the event.

1.4 Purchasing of tickets by a fan

Fan can get tickets to an event by choosing the "Current Events" option in the navigation bar which displays a list of all the events. On choosing the "Get Tickets" option a small form shows up where the fan can choose number of tickets and then confirm the transaction via Metamask. A single fan can only buy 3 tickets in total to prevent scalpers.

1.5 Refund and Transfer of tickets

Once a fan buys a ticket it will be visible to the fan after clicking the event box on the “Current Events” page, there the fan will have the option to either get “Refund” or “Transfer” tickets. After choosing “Refund” a small form shows up where the fan can choose the number of tickets to be refunded, if the fan has more than one, and then confirm the transaction via Metamask. On choosing “Transfer”, the fan can enter the address of the person to transfer the ticket to and choose a ticket either ticket “1 , 2, 3” depending on how many tickets the fan has and then confirm the transaction via Metamask.

1.6 Components/Technologies

PopTickets is implemented with HTML, MaterializeCSS, JS for the frontend, Node.js and Express for the backend, Ether.js for interacting with the Ethereum blockchain and Solidity for smart contracts. Remix IDE and Ganache client were used for development and testing. Pinata API service for connecting to the IPFS network and OpenZeppelin smart contracts which include ERC721 and it’s extension contracts used in the app [2].

2 Challenges and Performance Evaluation

As a Web2 Developer, programming in Solidity introduced me to a lot of different concepts, such as handling of various data types and their conversion to JavaScript data types, storing of images as hashes, optimizing the design of storage in order to minimize gas costs.

As can be seen in the implementation, the form which creates an event stores a lot of details related to the event to the blockchain. There are about 9 fields just related to the event and a few more variables were to be used in the contract. The contract won’t pass the SOLC due to “the stack being too deep”. The assembly command SSTORE is executed in the background when storing data to the blockchain which takes up 20,000 gas hence using a struct to couple values together can help to reduce the number of times SSTORE is used. [3]

Further variables that can be stacked together to fill a 256 bit slot are placed adjacent to each other as seen below. Simply reducing uint sizes and placing uint type variables together saved 11,000 gas. Another way to reduce gas costs as future work can be to have encode and decode functions for form fields and store them directly with the MSTORE command. The option to “Enable Optimization” also helped to reduce gas costs in Remix IDE. Storing variables as constants wherever possible also helped similarly.

```
// Everything related to Event Creation
contract Events is ERC721 {

    // Form Details

    struct FormDetails {
        string eventname;
        string venue;
        string description;
        uint64 startdate;
        uint64 enddate;
        uint64 price;
        uint32 duration;
        uint32 capacity;
    }

    FormDetails public form;
    uint64 private ticketID;
    uint32 public constant maxtix = 3;
    string public imgHash;
    bool private cancelled = false;
    address payable public OwnerofEvent;
```

Figure 1. Variables

The form uses two fields “Start date” and “End date” for storing the events. Dates in solidity are stored as uint variables, hence finding a way to convert date input in JS, was tricky. Also, when fetching form data back to display fields solidity returns uint variables as BigNumber which also has to be converted back to numbers and then strings in JS.

Using the IPFS network to store images helped to save storage space. The usual ipfs-http-client node module update introduced some breaking changes this year hence implementation was carried out with Pinata service.

The deployment of the project requires 0.02069352 ETH, so with the current price of $1222.72, the deployment would take about $25. The transactions for contract interactions are following the 21,000 gas limit with most transactions being under 20,000 gas. Form creation/change form detail transactions, as expected, consume more gas at around 39,55,373 units costing $8 for adding or changing details of an event. It is useful in a way so event organizers won’t change tickets prices or other details too often.

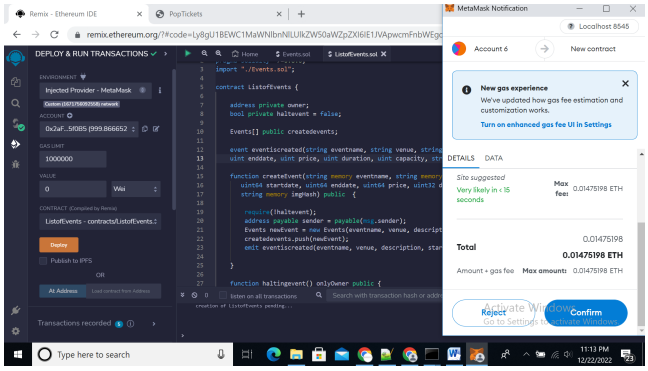

Figure 2. Contract 1 Deployment Cost

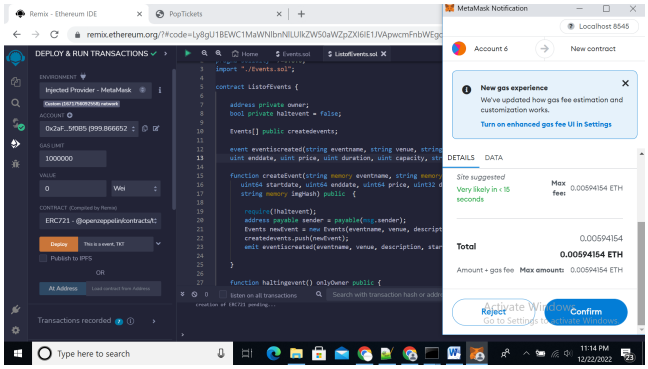

Figure 3. Contract 2 Deployment Cost

3 Conclusion/Future Work

Hence, the development phase for the app came with quite a few challenges and with the constant need of optimizing the smart contract code as and how it was being coded. An area of improvement would be the deployment cost which could be reduced further with other optimization techniques.

Future scope for the project could include creating tokens with the ERC-1155 standard. A feature that can be added to the website could be a way for users to earn fungible tokens in the form of a game or a lottery and they can use these tokens to convert them into NFTs which will serve as tickets to events. Any games or lotteries would incentivize users to keep using the platform.

References

[1] Dennis Van der Vecht and Rahul Buddhdev. 2022. *Embarrassed by the Speed of Your DApp? Use DApp Scaling.* Retrieved 22 December 2022 from https://10clouds.com/blog/defi/the-speed-of-your-dapp-use-dapp-scaling/

[2] OpenZeppelin. 2018. *OpenZeppelin Contracts.* Retrieved 22 December 2022 from https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token/ERC721

[3] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.