CS6666: Blockchain and Distributed Ledger Technologies

Project: Smart Contract for Ticketing

Team 5: Suhas P(CS17B116), Samuel J(CS17B026), Harshit K(CS17B103), Arnav M(CS17B110), Vamsi KV(CS17B045)

Abstract: Normally when tickets circulate in the market, they are bought at retail price and sold at a higher amount at a later day (usually close to the day of the event). We want to eliminate this abuse by integrating it into blockchain with smart contracts. The ticket along with the transaction will be part of a contract in the blockchain. If this ticket is resold, the actual amount transferred can't be tampered with as it will be embedded in the byte code of the contract. As smart contracts are fully programmable, it allows appending outside dependencies and more logic. The ticket is digital, so can't be physically transferred.
In a nutshell, a blockchain-based event and ticketing system has the following benefits:

1. Elimination of ticket duplication and counterfeit tickets
2. Elimination of ticket touts and purchasing bots
3. Fully transparent ticketing aftermarket
4. Automatic refund at the time of cancelation

Why Blockchain?

We aren't using private server model to manage the application for ticket transfers as this application can be updated or tampered with, by someone who has access to the code. Embedding this entire application in a smart contract ensures immutability, where it is monitored by multiple decentralized system. The smart contract is also transparent to everybody. It provides a single API to both the theatres and ticket buyers/users.

Our system design:

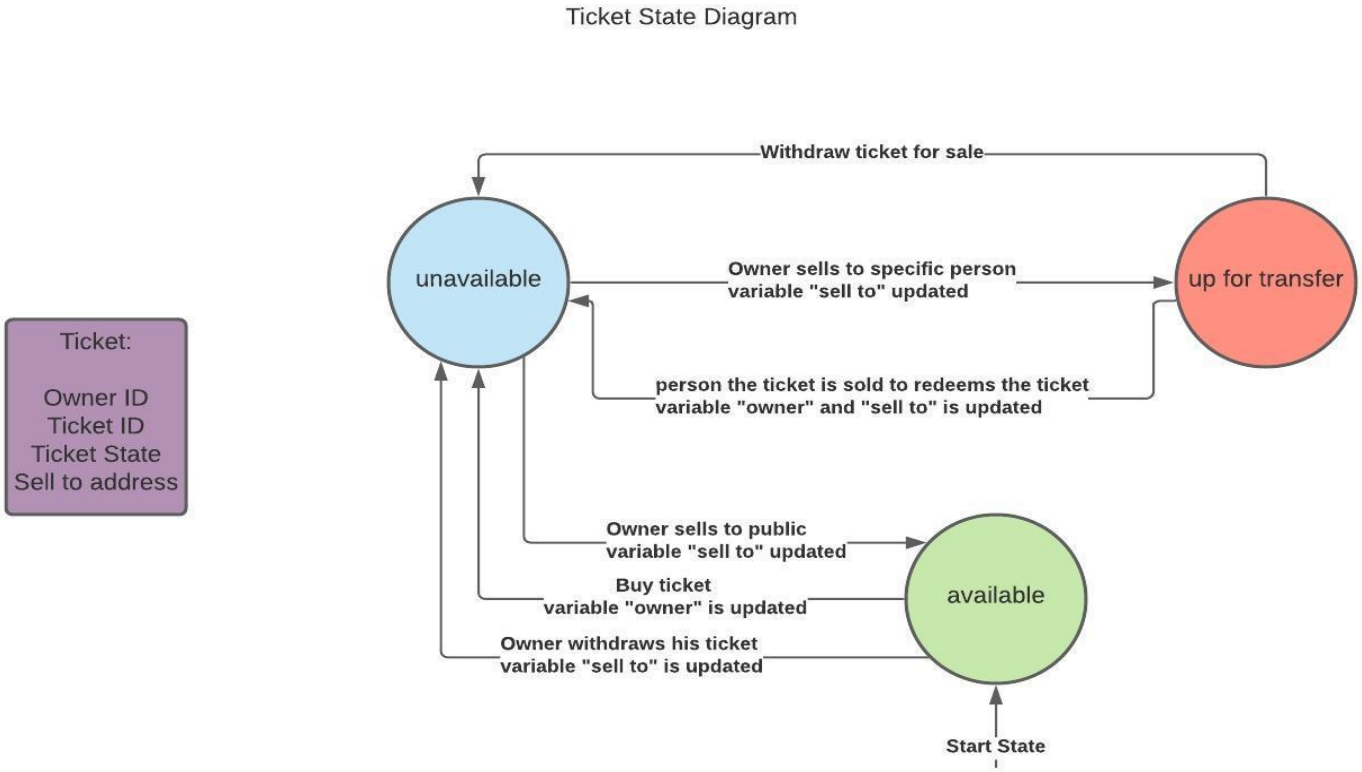
In our implementation, the smart contract will start with a set number of tickets, and these are the only tickets that'll be in circulation. Each ticket has a state which is shown in the state diagram below. Details on the states:

1. Available: This is the state that all tickets start off in. Any account can buy a ticket from this state, as long as they pay the fees.
    a. Once a ticket is bought, it moves to the "Unavailable" state and the "owner id" field is updated to the user who bought the ticket.
    b. If the ticket had an owner before, it can be withdrawn. This changes the state to unavailable
2. Unavailable: This is the state a ticket is in, when it is under the ownership of a particular account (user). Other accounts will not be able to buy or

access this ticket directly. In this state, we can transition to two other states as follows:

   a. The owner of this ticket can put this ticket for sale to the general public. This changes the state to “Available”.
   b. The owner of this ticket can put this ticket up for sale to a specific person of the owner’s choice. This changes the state to “Up for transfer”. The “sell to address” field is updated to the said account of choice.

3. Up for transfer: This is a state, where a user can accept and buy a ticket that is up for sale specifically for them. We have two transitions again:
   a. The person to whom this ticket was sent to, accepts and buys it. In this case, the state changes to “Unavailable”. The “sell to address” field is reset to null and the “owner id” field is updated to the new user.
   b. The person who put the ticket up for sale withdraws the ticket. The state changes to “Unavailable”. The “sell to address” is reset to null.



Implementation:

The tools we used:

1. Ganache (From Truffle Suite): It's a personal blockchain for Ethereum based application development. We used it to simulate 10 miners.
2. Meta Mask: It's a Chorme extension, which is a gateway to blockchain applications.
3. Truffle Suite: Development environment for blockchain dapps (decentralized applications) and smart contracts.
4. Node JS: For running server, deploying the frontend.

Solidity code overview (Modules):

1. Buy Ticket: Buy's ticket from tickets available to general public. We check whether the amount paid by the user is less than the price of the ticket, then the money is refunded and no action is performed. If the amount paid is more than the ticket price, the excess amount is refunded and the ticket is allotted if possible. If no ticket is available, the ticket price is also refunded. This ticket state is updated as shown before.
2. Redeem to Pool: If the ticket specified by the user is owned by the by the said user, then it is put up for sale to the general public and the state of the ticket is accordingly updated.
3. Sell to: Given a specific ticket and a specific user you want to sell the said ticket to, the contract checks if you own the ticket, and if you do, then it is put up for transfer to the target user.
4. Accept Ticket: The user can buy a ticket put up for sale specifically for him. We check whether the amount paid by the user is less than the price of the ticket, then the money is refunded and no action is performed. If the amount received is correct and the ticket target is the current user, then the ticket's ownership is transferred to this user.
5. Withdraw Transfer: A user can rollback the sale of a ticket. The ticket's state is updated to unavailable.

Simulation:

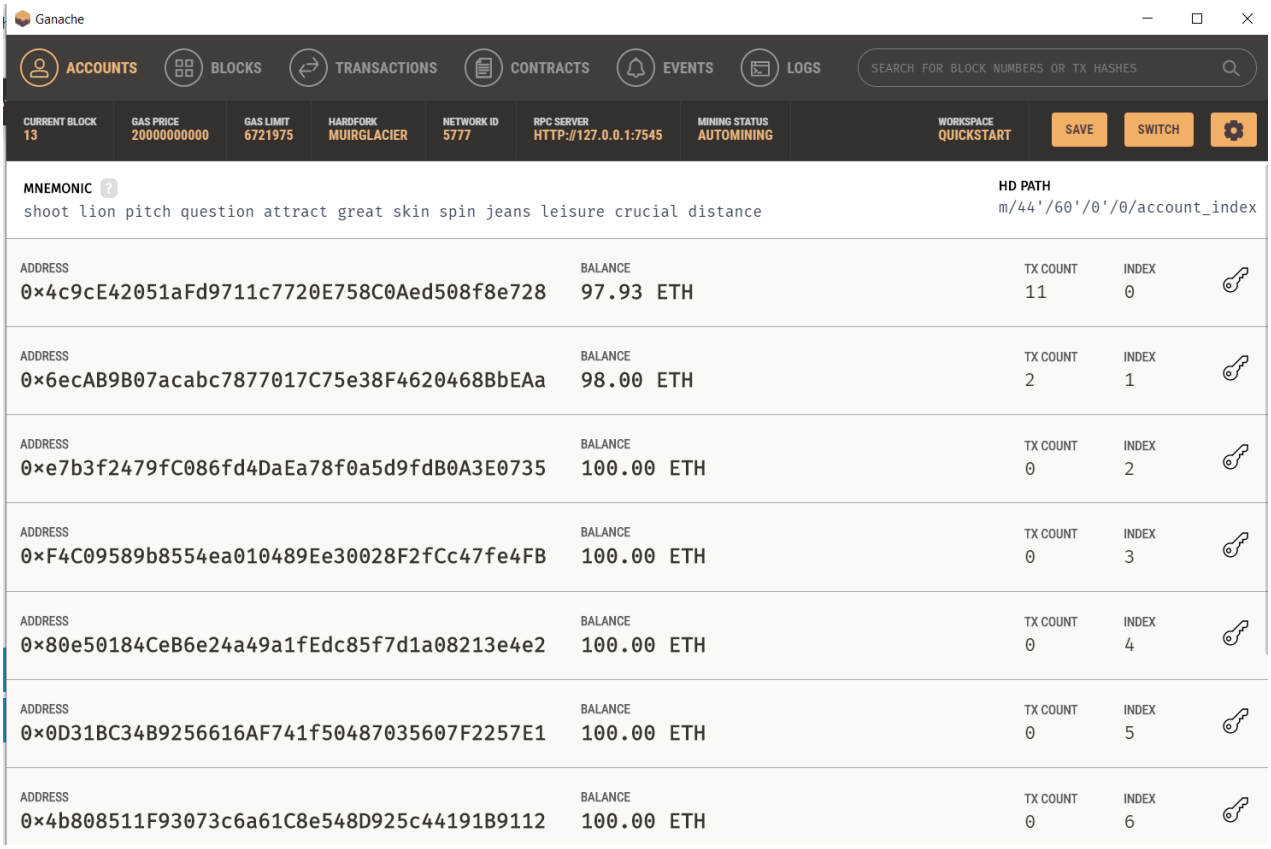


Figure 1.1: Ganache simulating 10 miners.

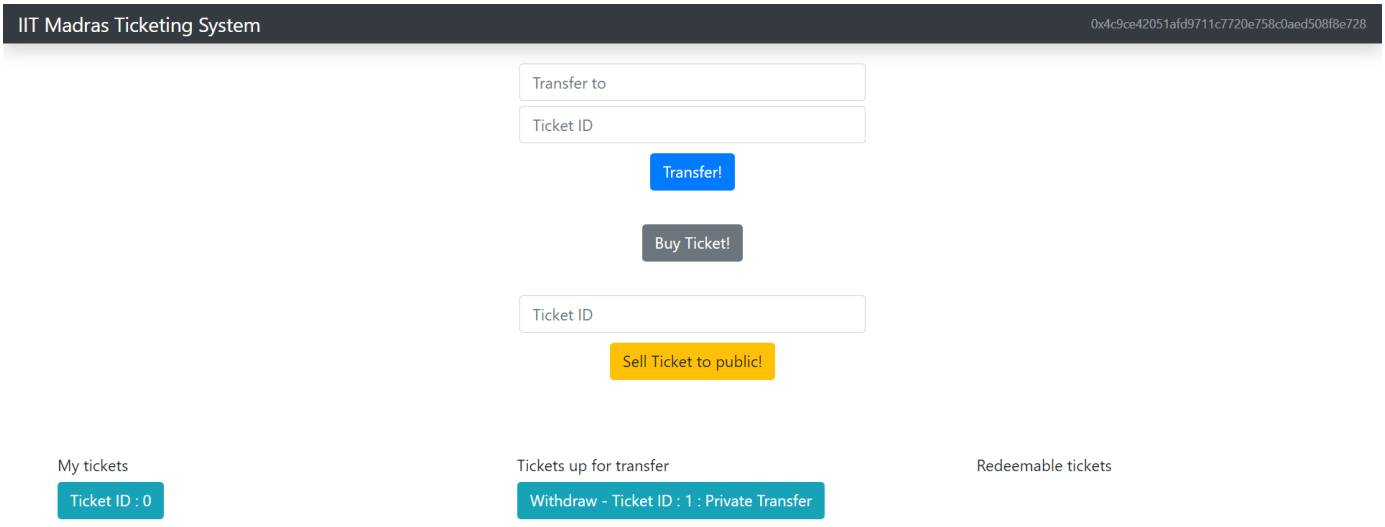


Figure 1.2: Basic UI

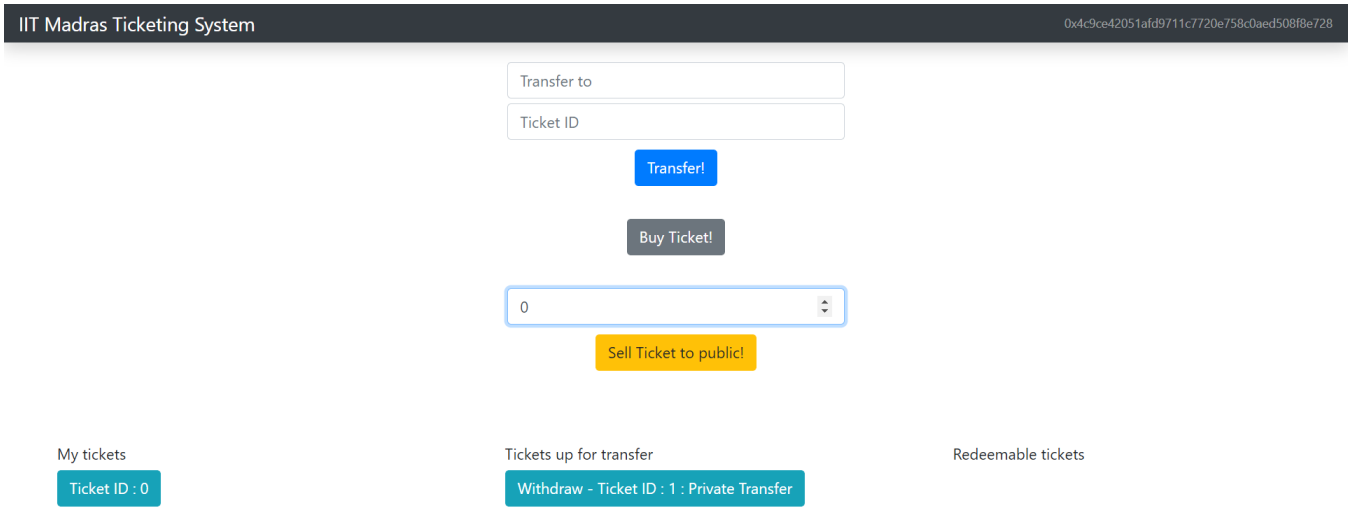


Figure 1.3: Selling Ticket 0 to Public

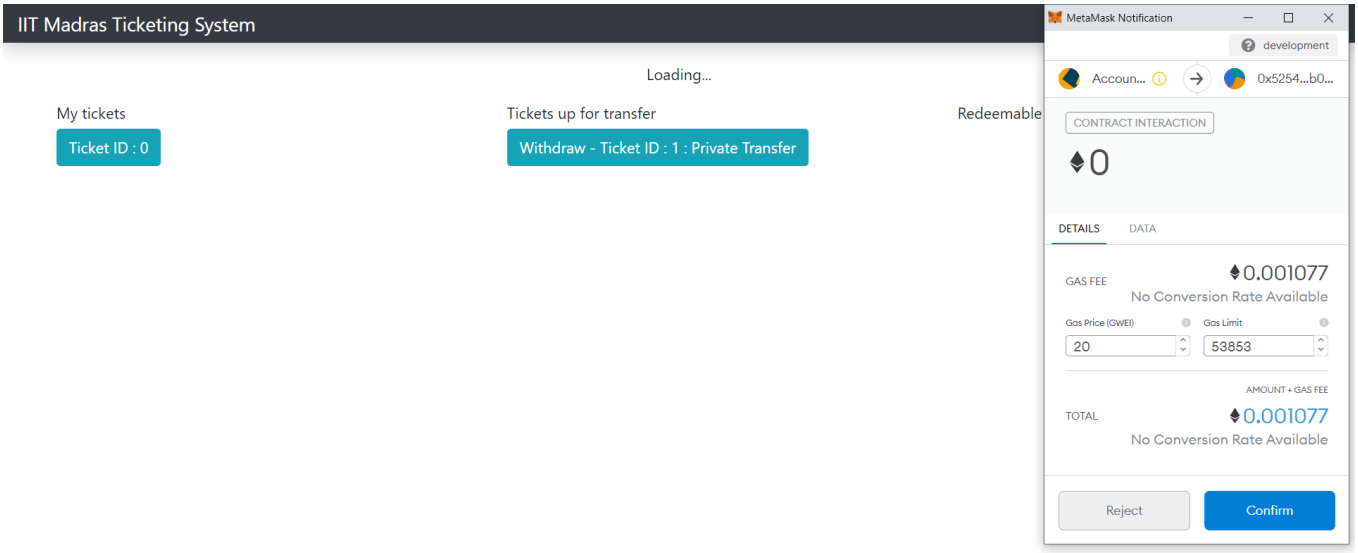


Figure 1.4: Confirming sale (Along with payment of gas)

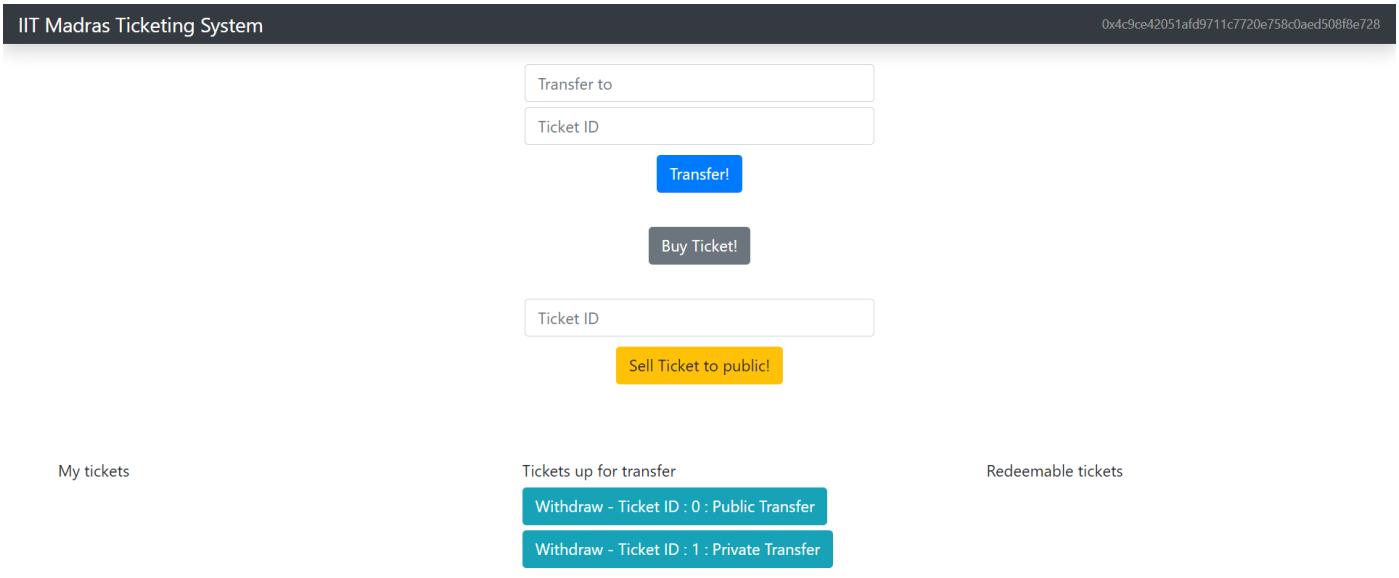


Figure 1.5: Ticket open for sale to the public

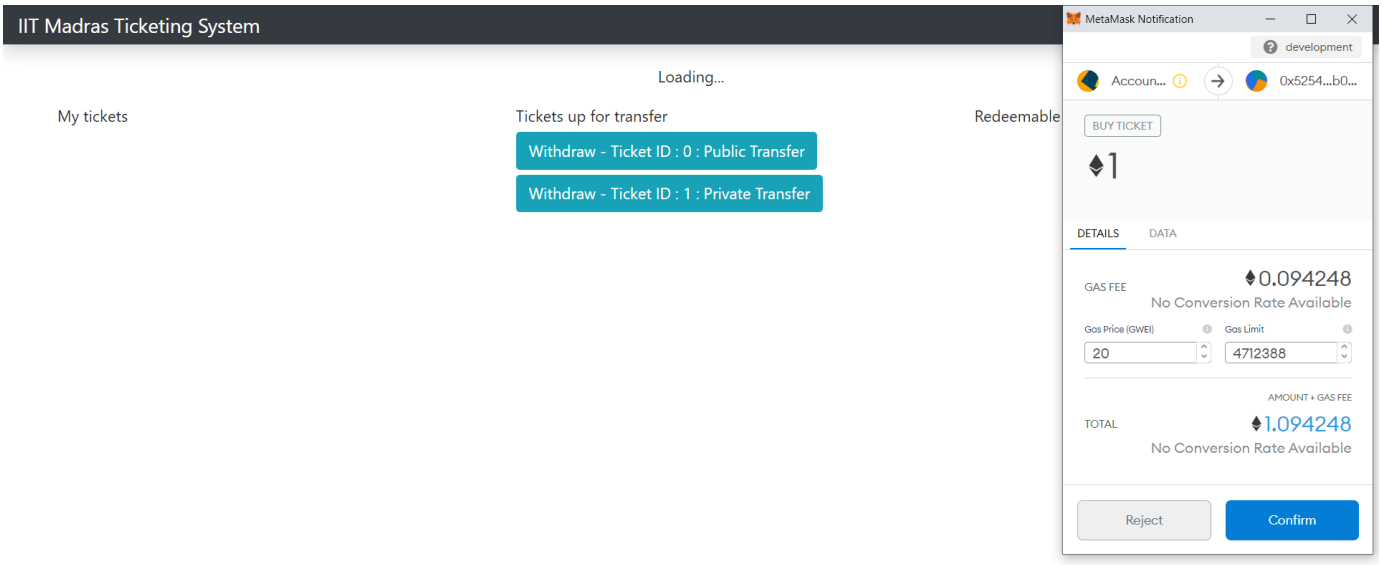


Figure 1.6: Buying back any ticket up for sale to public. Notice that we are paying 1 Ethereum along with gas price.

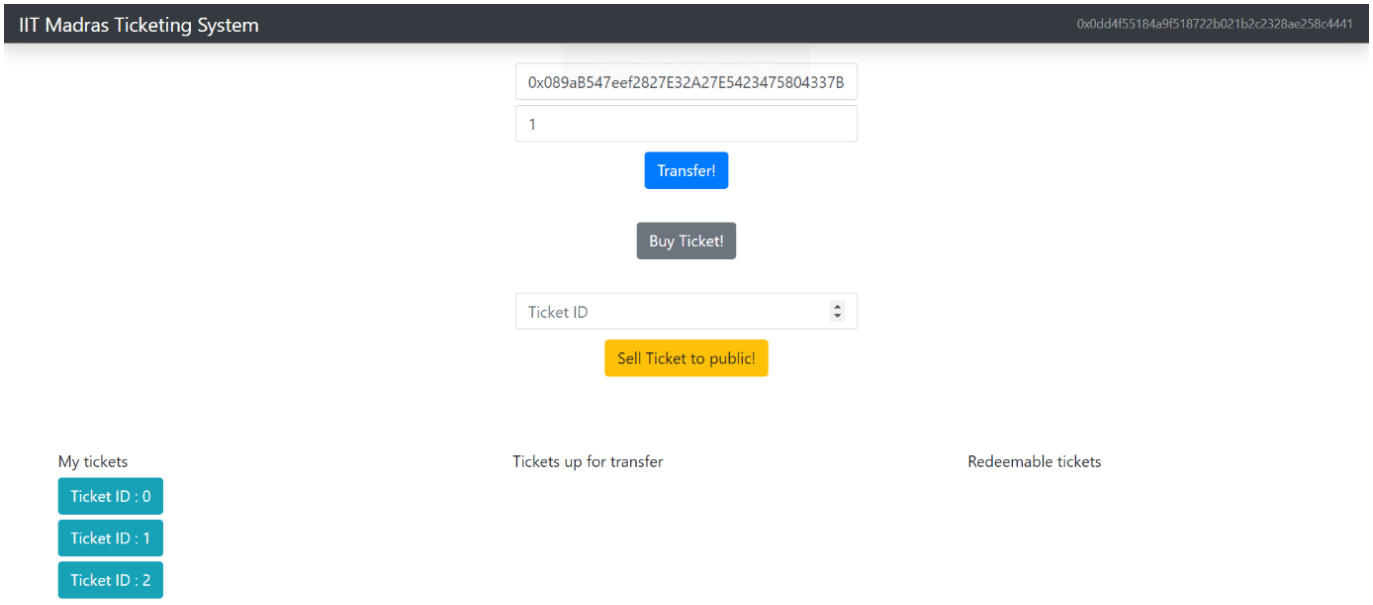


Figure 1.7: Put up Ticket ID 1 for sale for a particular address.

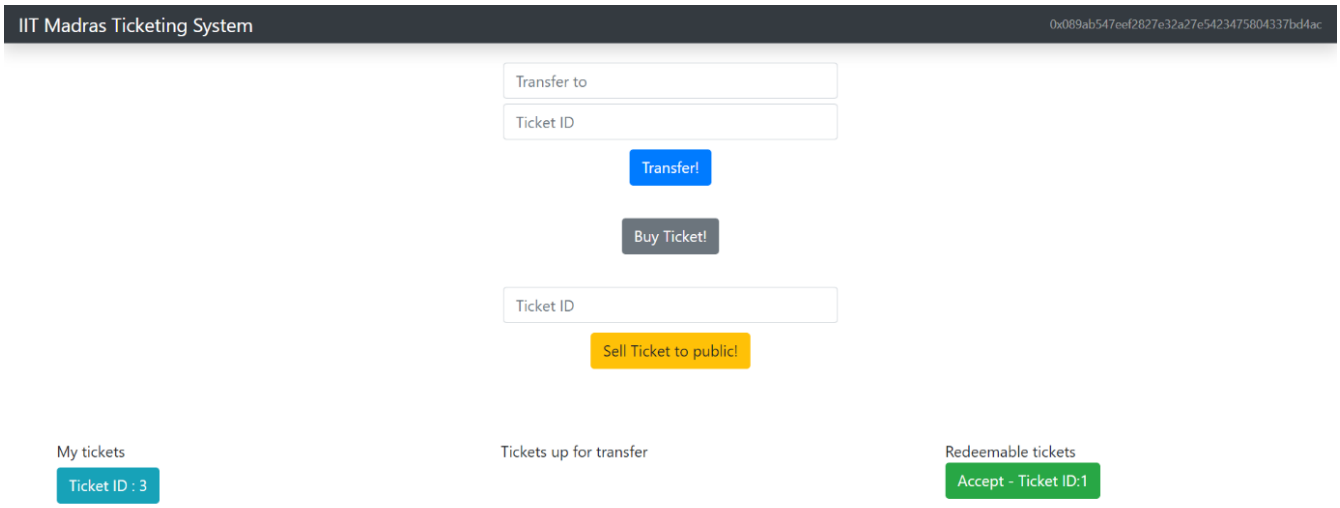


Figure 1.8: The ticket is now listed as redeemable

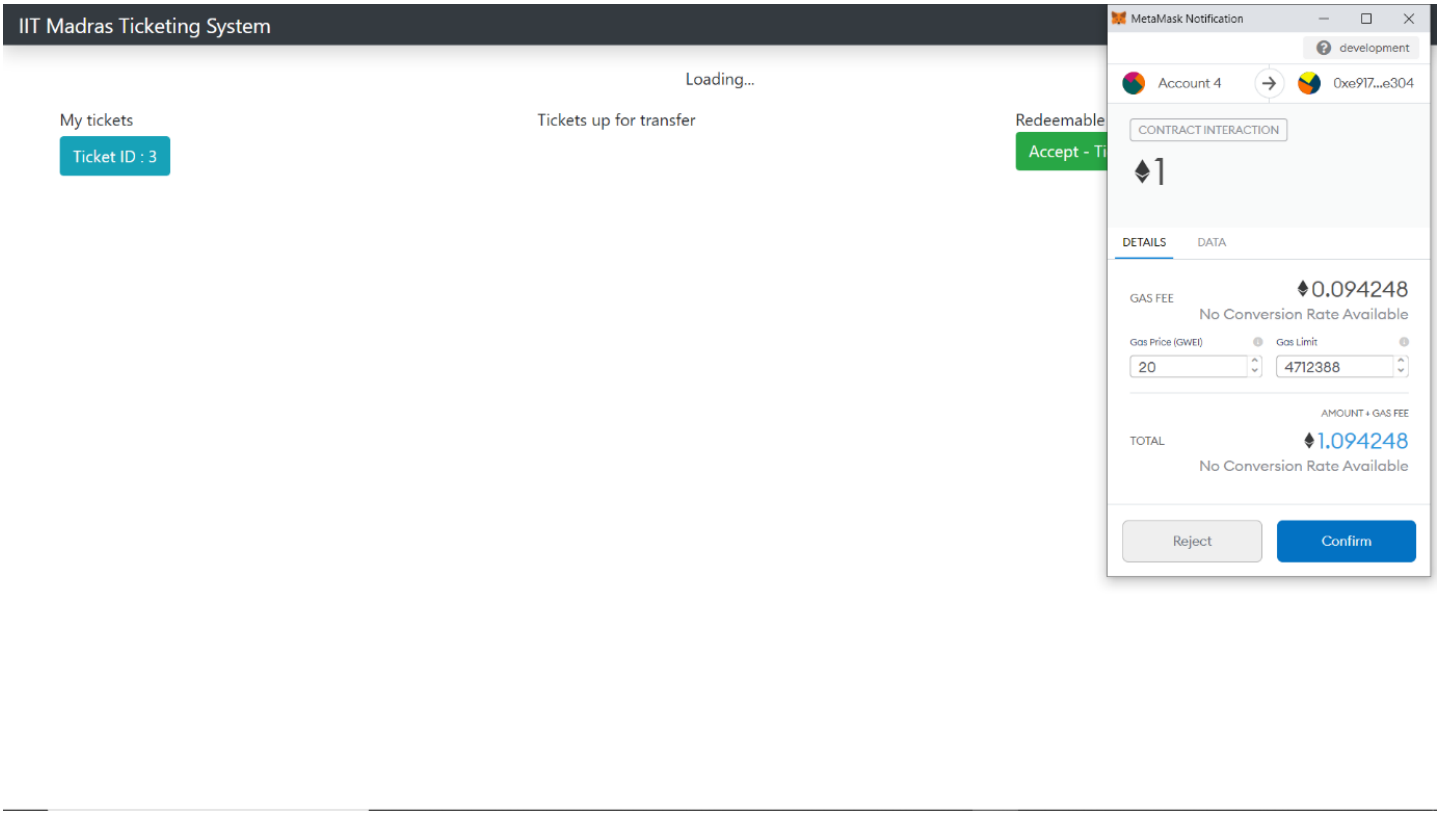


Figure 1.9: Buy the ticket with ID 1, that is up for sale for the current user, by paying 1 Ethereum and gas.

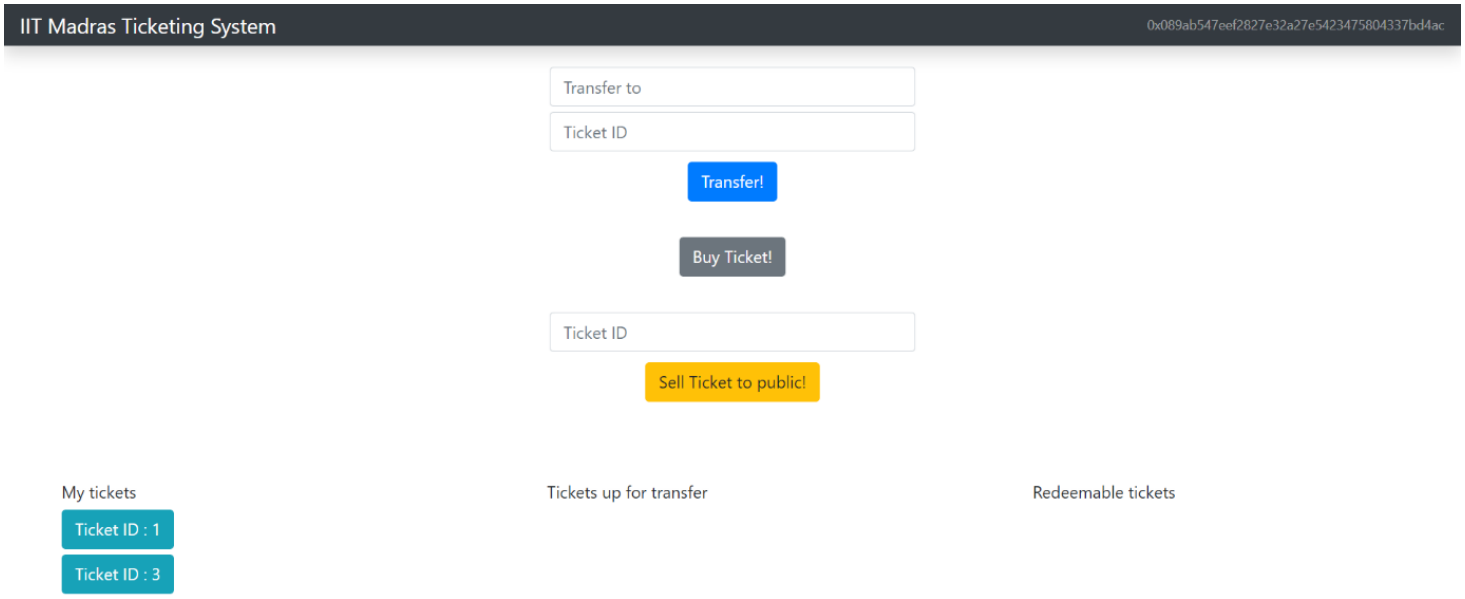


Figure 1.10: You've redeemed ticket 1.

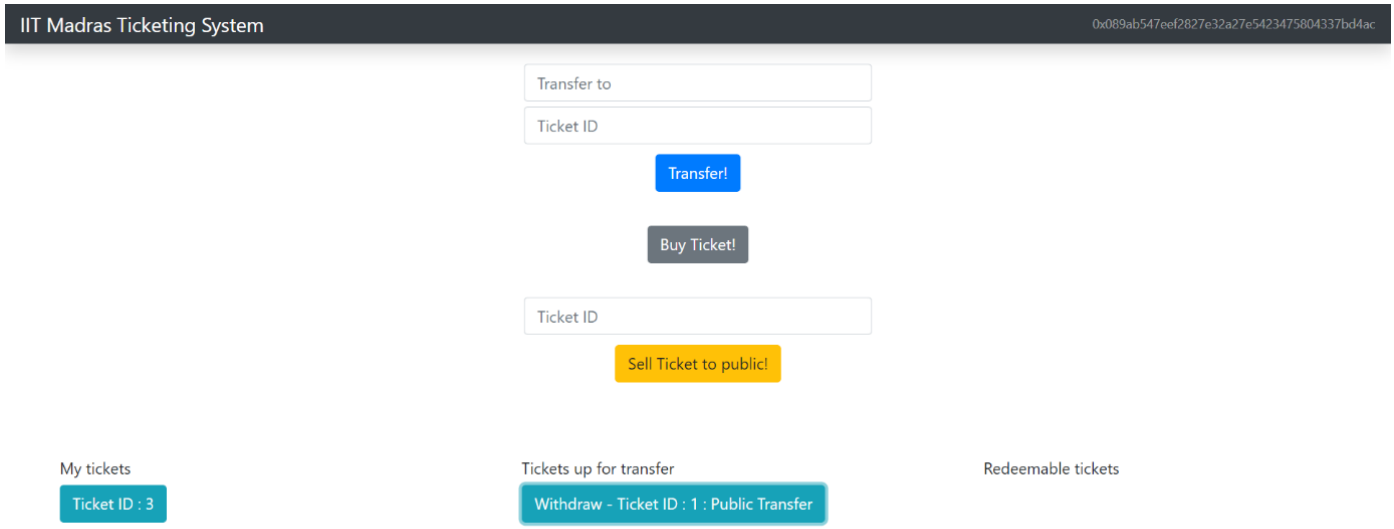


Figure 1.11: Withdrawing a ticket that was put up for sale.

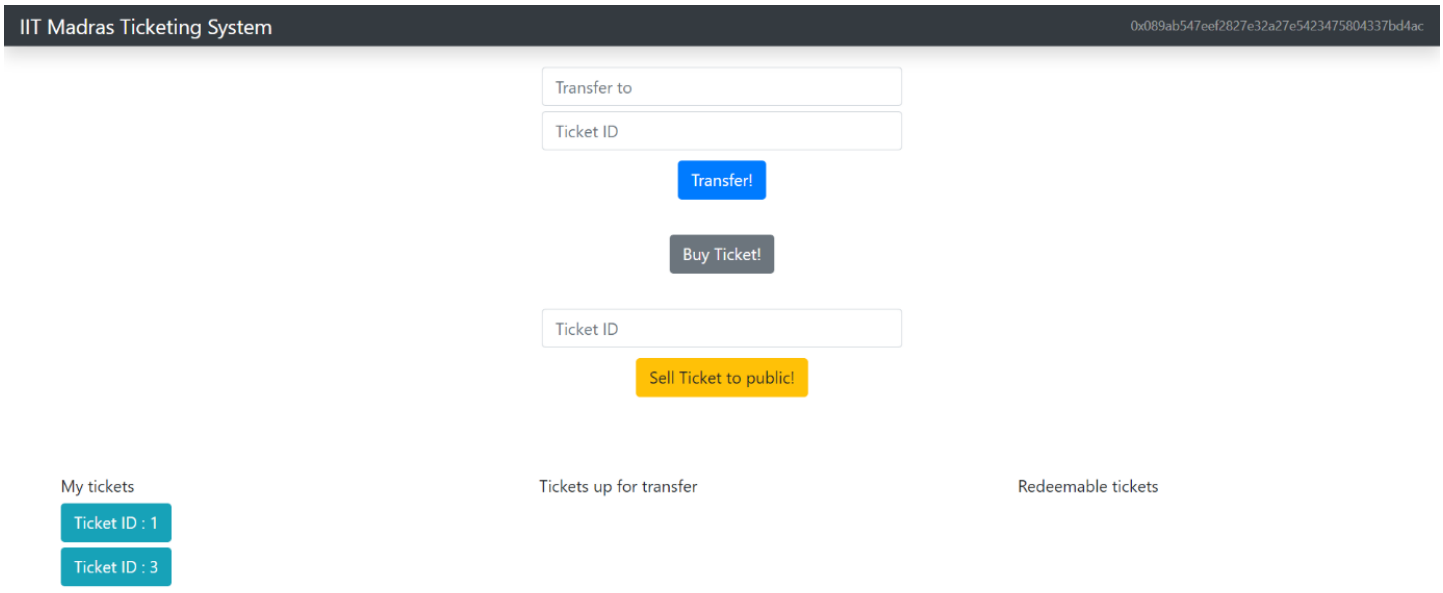


Figure 1.12: Withdrew the ticket.

Future steps:

This is a pretty robust ticketing system, that doesn't allow tampering with the code backed by the Ethereum blockchain. This can be expanded to have users not connected to a miner account directly, and they can be uniquely identified using a universally accepted identification (Eg. Passport ID). The tickets can be implemented digitally, with QR code scanning for verification. We could also integrate multiple currency support.

Credits:

1. Dr. Janakiram's lectures for theory.
2. Dapp University tutorials and To-Do List for template code.