

CS4455 Ethical Hacking and Security

Blockchain Technologies and Applications

Project: Ticketing System DApp

Daniel Kennedy

22340017

23/05/2024

Design Description

There are three stakeholders whose roles are reflected in the implementation of this project:

1. The Organiser
2. The Usher (i.e. person who admits people to the event)
3. Patrons

My smart contract implements the ERC20 standard for fungible token exchange, with a few augmentations for the application of a ticketing mechanism:

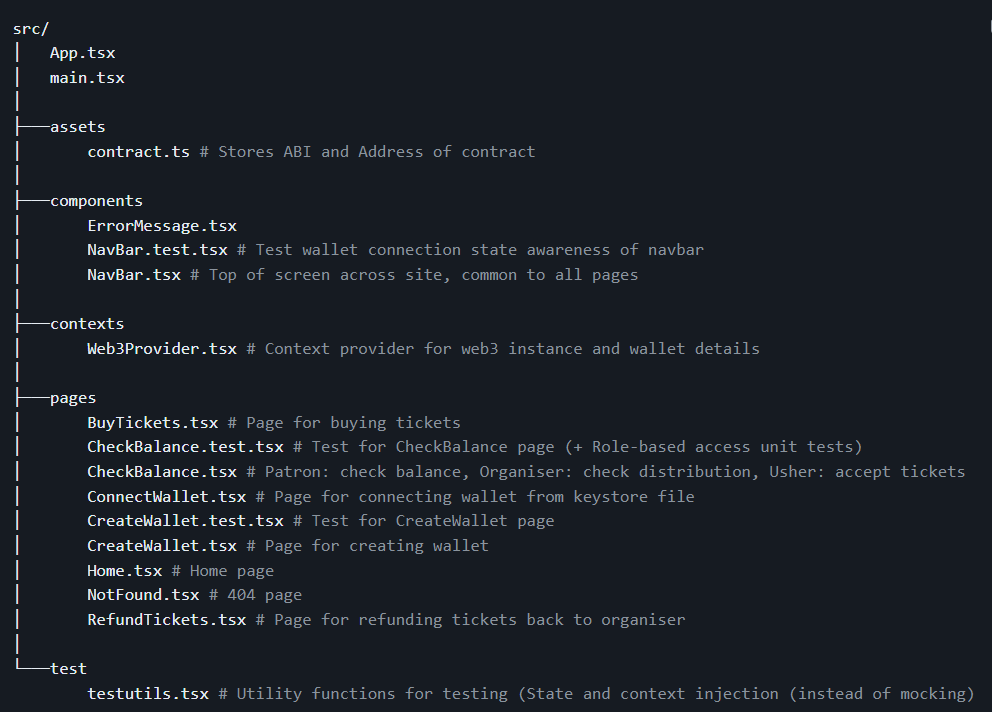
1. The total supply of ticket is sent to the organiser upon contract initialisation
2. Ticket purchases are possible, which take tickets from the organiser’s balance and put them in the patron’s.
3. The Usher, who is designated by the organiser (but should be a different wallet address) at contract initialisation is permitted to move tickets from a patron’s balance to the wallet’s balance. Once here, they cannot be moved so the contract’s address acts effectively as a burn wallet.
4. There is a “ticketsForSale” flag set to true when the contract is initialised, and this is required to conduct ticket transactions (purchase, refund). The organiser can call endSale() to take the funds from the contract address and set the flag to false so tickets can no longer be bought

For the frontend, I created a React web app. This allowed me to do some neat things with state management (wallet info, web3js instance) across multiple pages by putting common functionality into a shared context provider. I used tailwind for styling, testing library with Jest for testing and web3-react (wrapper of web3.js) for communication with the Sepolia network.

Code Overview

Repo: <https://github.com/danielkennedy1/tickets>

Below is a breakdown of the React src folder, with some irrelevant files omitted for brevity:



Note: The two files at the top (App and main) are the entry point to the react application

Running instructions (requires node, npm):

Open project root from submission folder (or clone repo).

Run the following in command line:

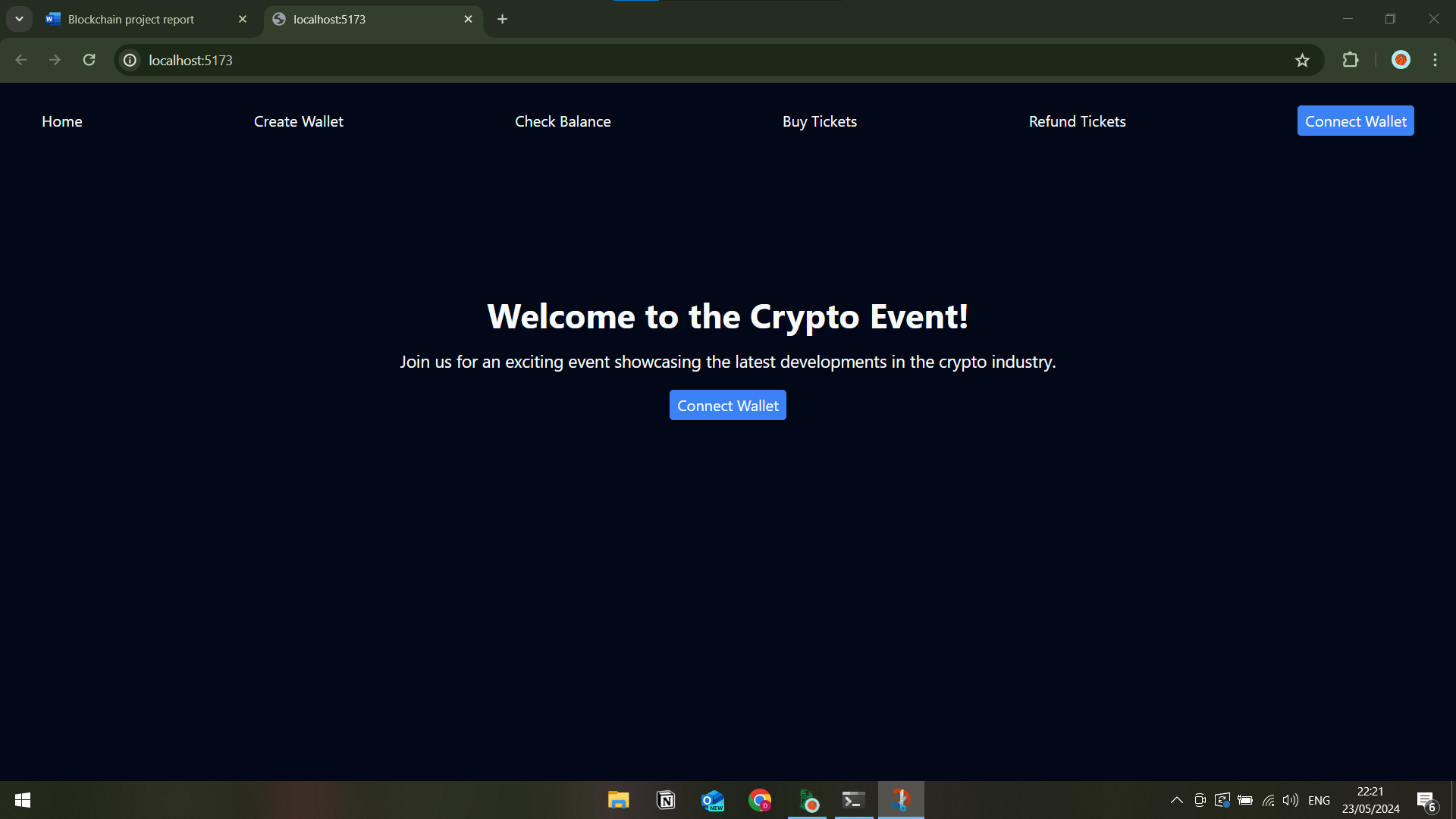
npm install

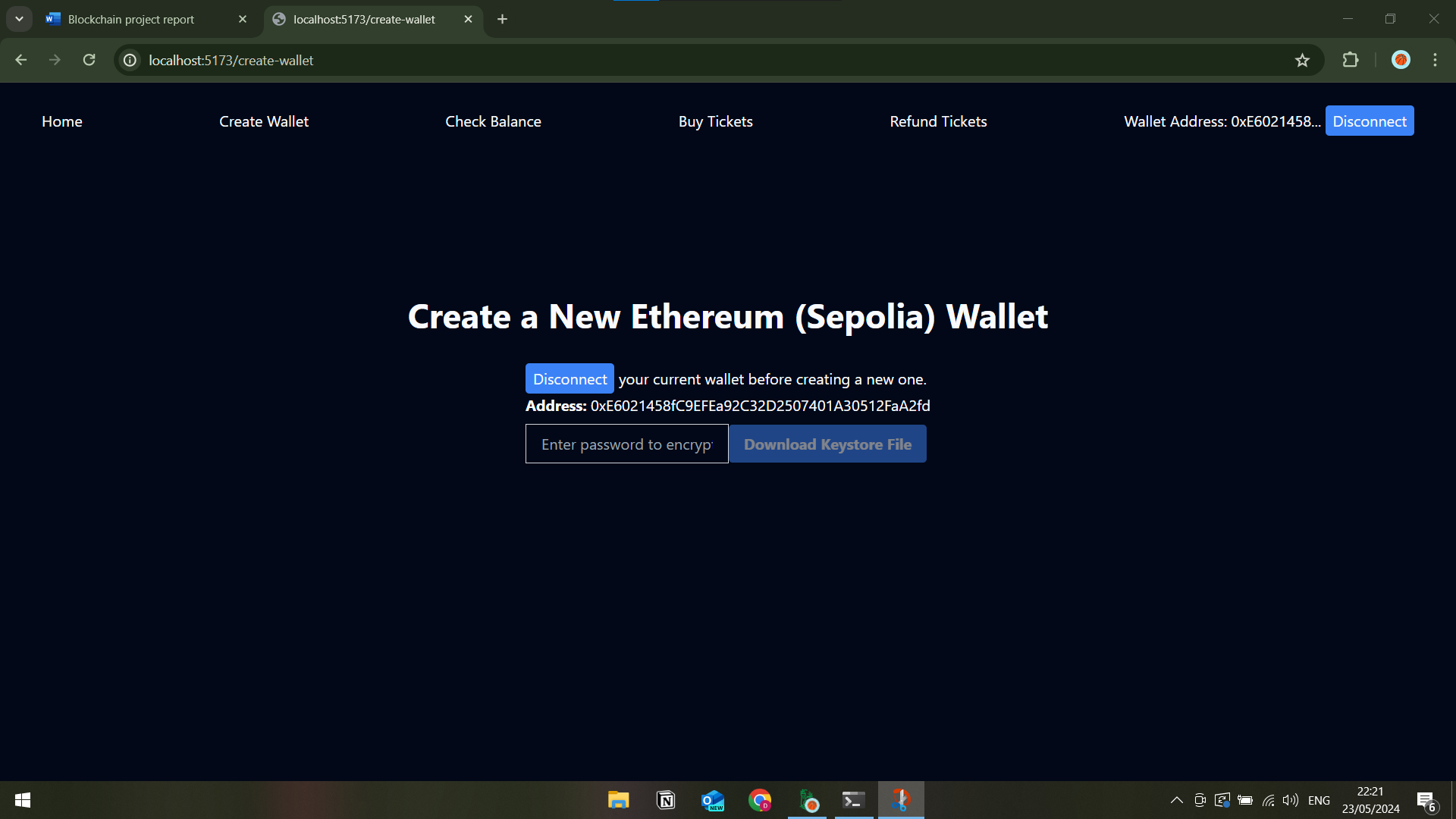
npm run dev

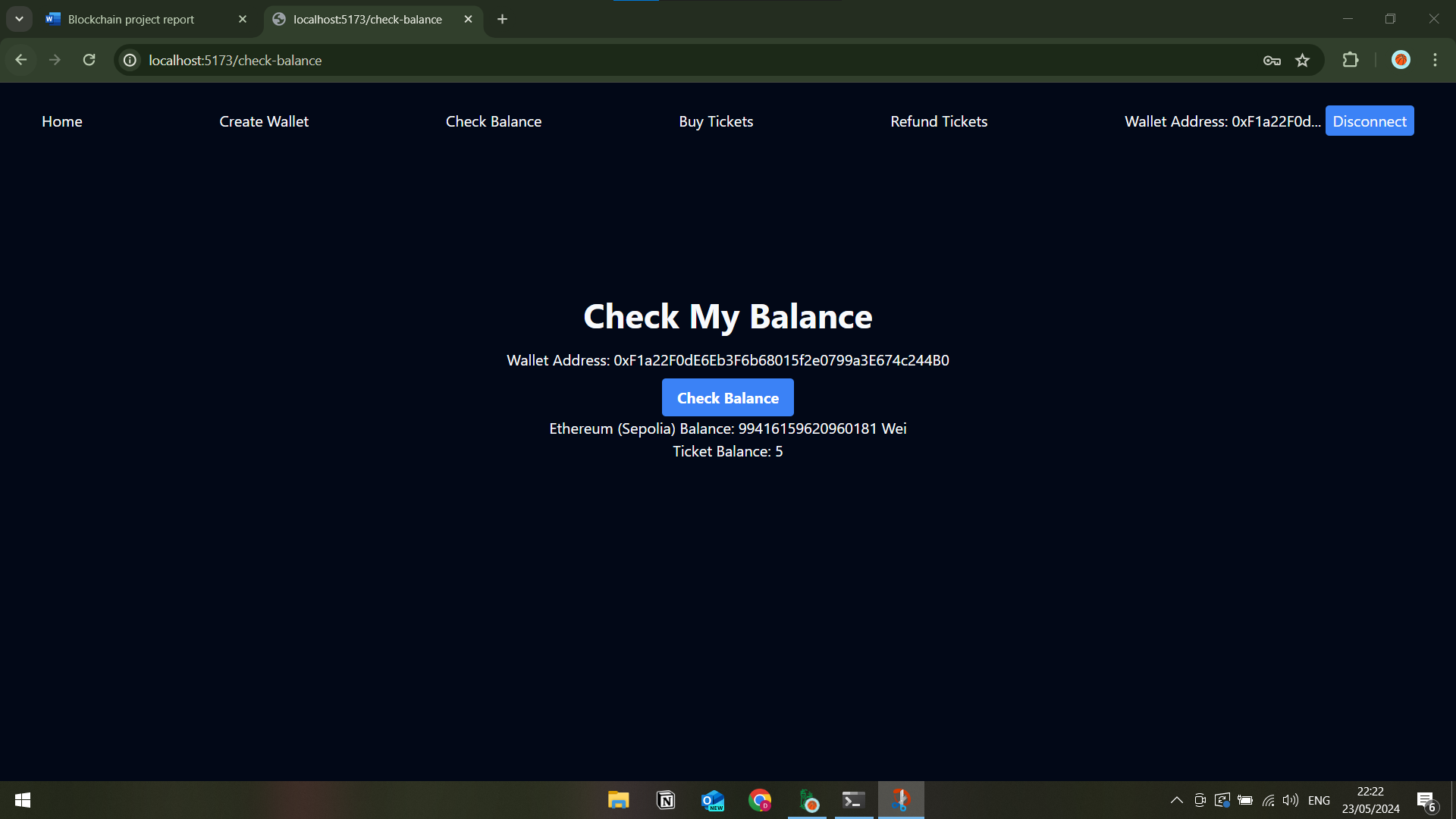
Transaction links

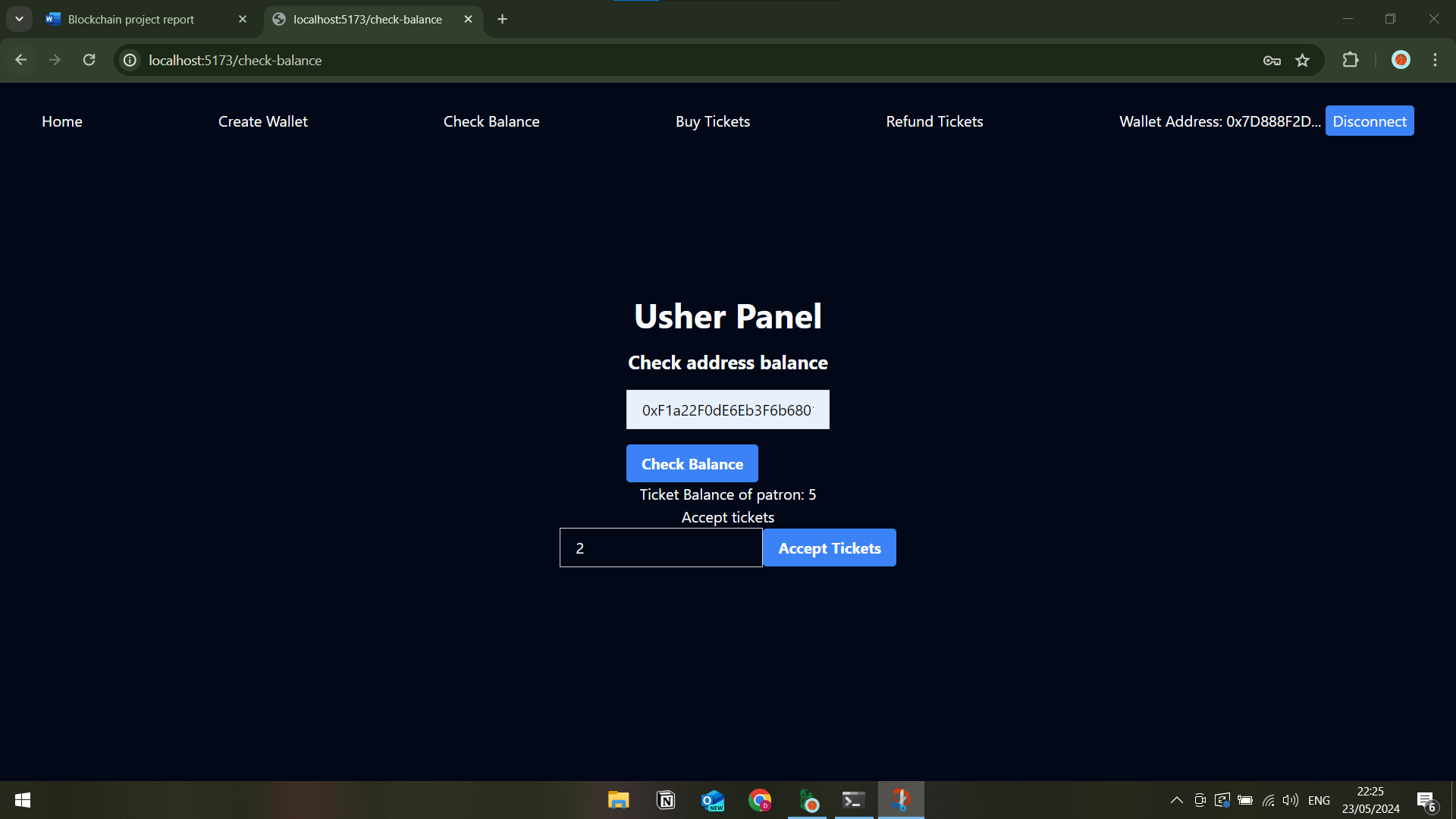
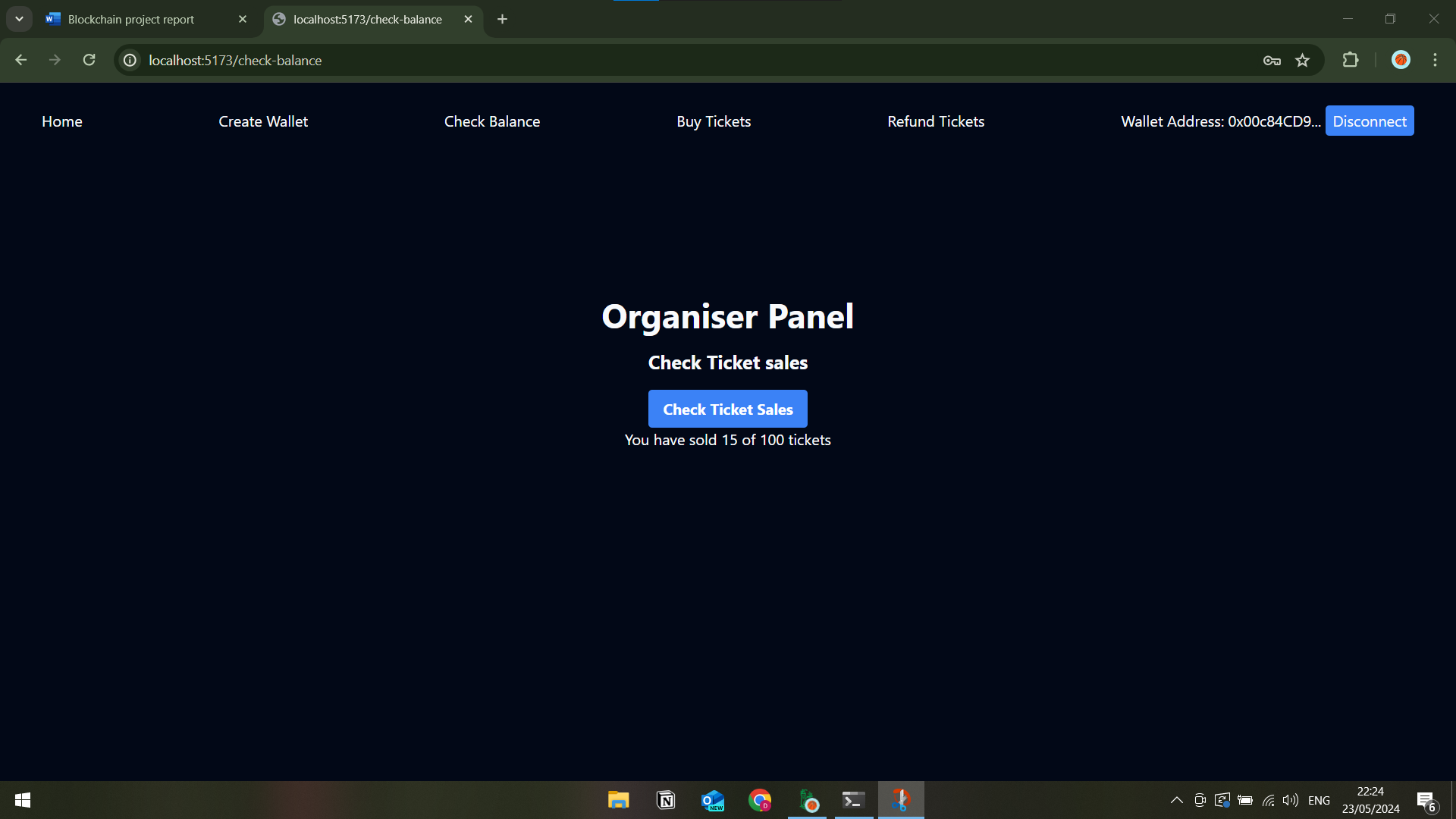
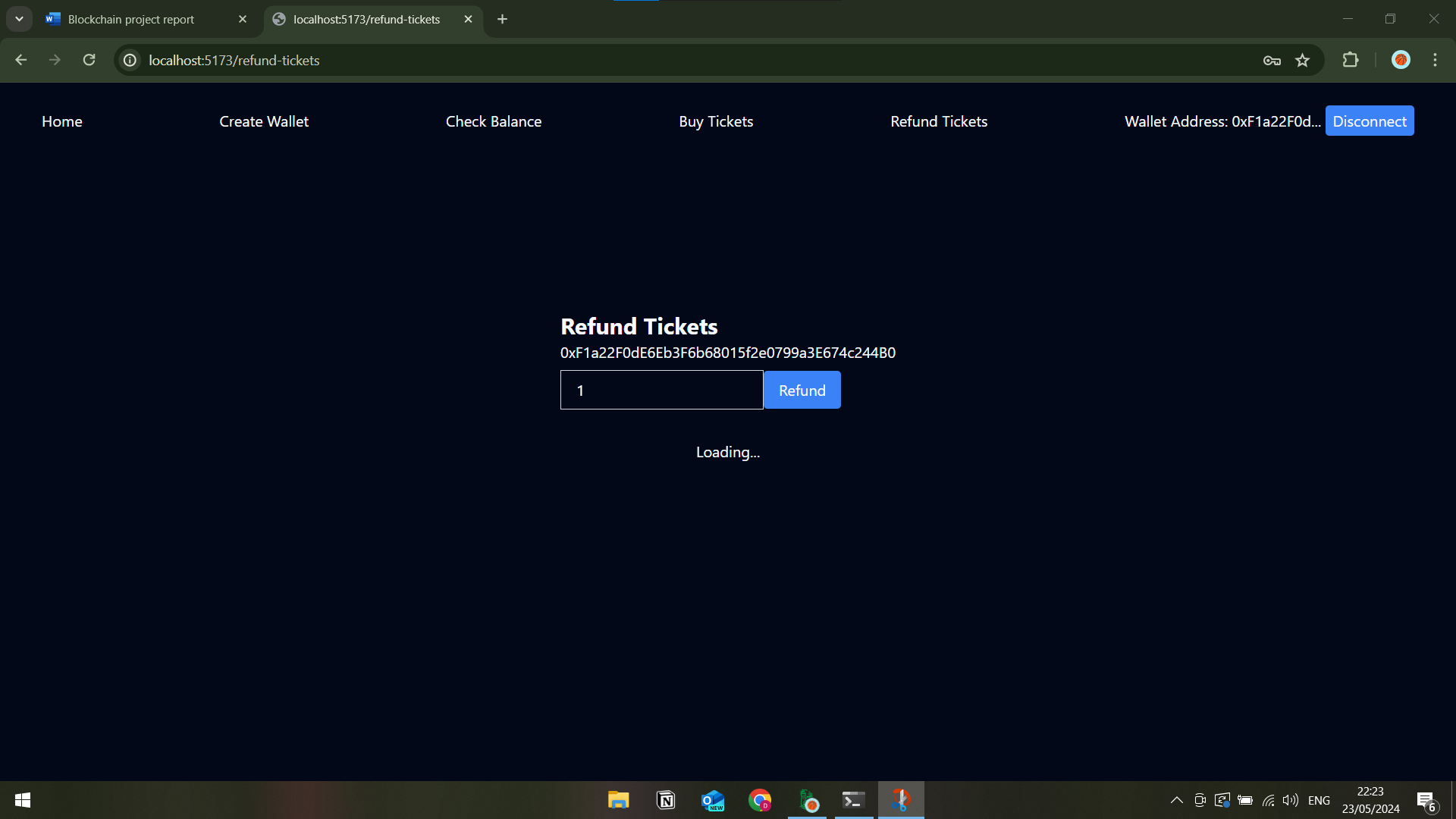
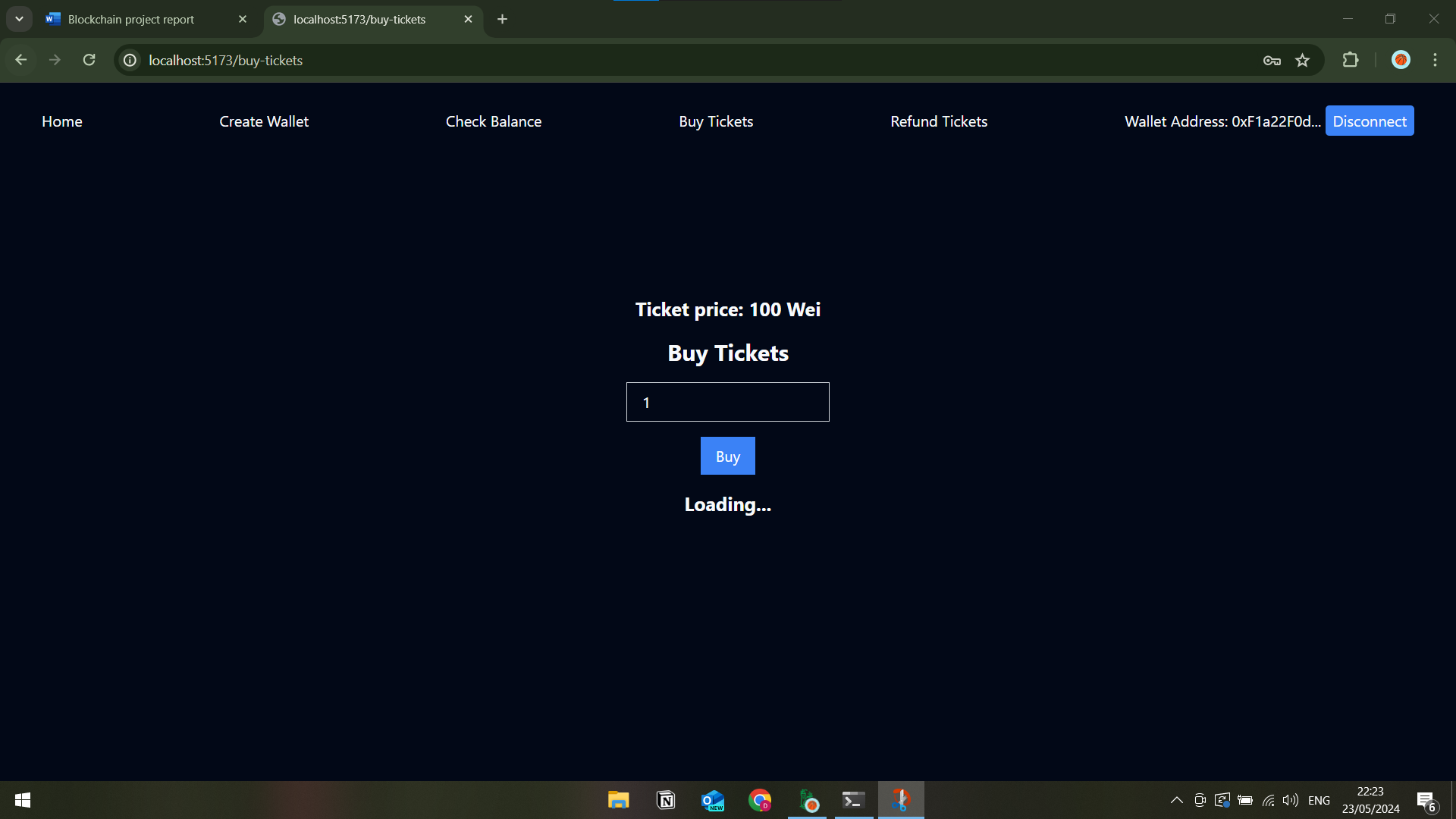
* Successful contract deployment: [Etherscan](https://sepolia.etherscan.io/tx/0x1d807a5451c5ca717a01d9f82c1deeb02dd731d000d4a23e93ff3634de8d5734)
* Successful token purchase: [Etherscan](https://sepolia.etherscan.io/tx/0xbc04a255d1da70f5e89c9cc42b52c8a4fc8cab1be9f34ef532c3c584dcf86523)
* Successful topping up of:
  + Contract creator (“organiser”): [Etherscan](https://sepolia.etherscan.io/tx/0xb4e606c229b8966f3ce2b4d18040990a371c893408f1e569275891299fba8d6e)
  + Ticket Purchaser (“patron”): [Etherscan](https://sepolia.etherscan.io/tx/0x4a6f5c2eedf9d930212a0a7510859fad43b36df97c98a5079e5d13b5ec4f4eae)
  + Vendor / Doorman (“usher”): [Etherscan](https://sepolia.etherscan.io/tx/0x0cbfeb41217a79ba314e8a61262cc2aa995e32a8c1b9c75918a3237bfab2963c)

Frontend screenshots









Reflection

If I were to start this project over, besides utilising a more versatile token standard for the event mechanism (ERC-721 for seated event or ERC-1155 for both seated and standing, etc.), I would have liked to structure the frontend differently to better separate concerns in relation to different use cases, probably by giving different “areas” to different roles. This would take further effort in pushing common functionality into shared context providers, etc., but would make for a more maintainable and extensible application.