Report

To Run:

Use live server within VS Code to run

To run tests have `xvfb` installed and run `python3 tests/test_e2e.py`

Code Overview

I used simple HTML and JavaScript to make this project. I used the pico.css framework for my css and styling.

I used remix to compile, deploy and do initial testing for my smart contract.

For testing, I used python and selenium for full end to end testing of each feature of my website.

This fully tests the code by emulating a user using the frontend and verify the results that are shown on the website

```
.
├── README.md
├── assets
│       ├── Krakoa3.ttf
│   ├── Green_Lagoon.webp
│   ├── Mary_Janes.jpeg
│   ├── MojoverseLive.png
│   ├── dazzler.jpeg
│   └── greenLagoon.jpeg
├── js
│   ├── balances.js
│   ├── contractInfo.js
│   ├── errorModal.js
│   ├── makeWallet.js
│   └── shop.js
├── balances.html
├── index.html
├── makeWallet.html
├── shop.html
├── font.css
├── tests
│   └── test_e2e.py
├── wallets
│   └── 0x648242eD89cdfa84Bf880729338d6b29221aa1de.json
├── .github
│   └── workflows
│         └── tests.yml
└── LagoonTicket.sol
```

Design Description

Code

There are 4 pages

Index.html

Displays a home page and links to the other pages

makeWallet.html

A user can input a password, the click "Make" to create a wallet and then display the information on the webpage. There is also the option to download the wallet json file

Balances.html

There are 3 drop downs here for each actor.

The Attendee can input their wallet address and the site will display their crypto balance, ticket balance and additional ticket information.

The Venue can press the check button. Their crypto balance and ticket balance will appear. Along with distribution information like how many tickets have been sold and the total supply

The Doorman can input a wallet address. If there are tickets, a green box with "Let In" will appear, along with how many tickets they have. If there are no tickets, a red box with "Don't Let In" will appear.

Shop.html

This has two sections.

The Load Wallet section lets users input their wallet file, password and load their wallet. If they want to see extra wallet information, they can click "See your Info" and a modal will pop up. Additionally their ticket and crypto balance will appear in the "Your Balance" section for each of use.

The Shop section lets users buy however many tickets they want for 0.00001 ETH or sell however many tickets for the same price. When either of there operations is done, a loading modal will appear until the transaction is complete. Then a transaction result modal will appear with all the transaction information.

For each of these pages, their corresponding javascript file handles the logic and dealing with the Web3 functionally

Contract

I implemented the ERC-20 standard for my smart contract. I extended the example we were given buy adding two functions, buyTickets() and returnTokens().

`buyTickets()` uses the value sent with the message, and transfers the equivalent amount of tickers from the vendor to the caller

`returnTokens(uint256 amount)` takes in the amount if tickets to be returned. It transfers that amount from the caller to the vendor and then calculates the amount of eth to be returned bases on ticket price. This is then transferred from the contract to the caller.

This was complied and deployed on Remix.

Tests

I made test cases for each operation on my website, this includes:

Test Name	Page	Verifies
`test_title`	index.html	title of the webpage is correct
`test_makeWallet`	makeWallet.html	once wallet is made, wallet info is shown on page
`test_UserCheck`	balances.html	once address is given, token & balance info appears
`test_VenueCheck`	balances.html	once button is pressed, token info appears
`test_DoorManCheckOut`	balances.html	address with no token given, "Don't Let In" is shown
`test_DoorManCheckIn`	balances.html	address with token given, "Let In" is shown
`test_BuyTicket`	shop.html	once token bought, token balance has gone up 1
`test_RetunTicket`	shop.html	once token return, token balance has gone down 1

Using Github Actions I set up a pipeline, that when on push or pull, the code automatically run the website and run the test cases to verify there are no breaking changes.

Transaction Links

Actor	Type	Link
Attendee	Account	https://sepolia.etherscan.io/address/0x648242eD89cdfa84Bf880729338d6b29221aa1de
Doorman	Account	https://sepolia.etherscan.io/address/0x26c3f76cB5b81827fb17Da5D2775C5eC62Dc12B7
Venue	Account	https://sepolia.etherscan.io/address/0xC881d45D2FE2F23A4346e0B39211059081ceFFDF
Contract	Account	https://sepolia.etherscan.io/address/0x644a7c1c7c694512c9d8bed7a17c5f5b36178716
Venue	Contract Creation	https://sepolia.etherscan.io/tx/0xa417a21e7a8d8b639c17b8487f4aadb90a93ffff5227118892e8f34aca732d15

Token		https://sepolia.etherscan.io/token/0x644a7c1c7c694512c9d8bed7a17c5f5b36178716
Attendee	Buying Ticket	https://sepolia.etherscan.io/tx/0xb1af6ee023fa56af6c3cc03e1d47c62ccc536aa458e0f84857483e8c2e4bf618
Attendee	Returning Ticket	https://sepolia.etherscan.io/tx/0xdcb8784402da035b4b8844a14d71d7cfa8552b0cd3f9fee6d35ebc102161ff18
Contract Creator/ Venue	Topping Up	https://sepolia.etherscan.io/tx/0x67ba7d633d6a03b61986109873a44769acbba4789e22b81c04a6edd5ffa6c604
Attendee	Topping Up	https://sepolia.etherscan.io/tx/0xa1f687cf489ffa319940d8c4ae2edfe75d23ed0b5303b54620ff60536a01dcdf
Doorman	Topping Up	https://sepolia.etherscan.io/tx/0x5bd96eb04f4a61474a51437c2946453b893d37864be75fac22db92c16bd46b33