D-TICKETMASTER ASSIGNMENT

uniblock

Created: Feb 2025, by David Liu
Last Edited: Feb 2025, by David Liu

-David Liu, Co-Founder and CTO of Uniblock.

D-TICKETMASTER

The goal of this assignment is to design and implement a decentralized ticketing system using blockchain technology. The system should allow users to create events, sell tickets, and verify ownership of tickets in a decentralized manner, reducing fraud and eliminating the need for intermediaries.

Submission Guidelines:

1. Submit a GitHub repository containing:
   - Smart contract code (Solidity) + Tests (at least 95% hardhat test coverage)
   - Frontend source code
   - Deployment info (Testnet used, contract address, etc.)
   - README.md report as outlined in the documentation section
2. Provide a demo video (around 15 minutes) showcasing functionality, code, tests and test coverage walkthrough

PROJECT REQUIREMENTS

1. Smart Contract Development (40 marks)
   - Implement marketplace smart contract in Solidity that allows event organizers to create and manage tickets, including specifying details such as name, date, time, location, description, ticket price and total ticket supply on sale.
   - The contract should enable the primary sales and secondary sales of tickets with % royalty distribution to the event organizer. All sales also include a % for platform fee sent to a treasury wallet address.
   - Each ticket should be a unique NFT (use ERC-721 standard and OpenSea Metadata and Openzeppelin Royalty standard).
   - Each ticket should also include details such as ticket ID and seat number.
2. Frontend & User Interface (30 marks)
   - Develop a simple frontend using React.js or any web framework to interact with the smart contract.
   - Users should be able to:
     - Create an event
     - Primary purchase of tickets using ERC20 called TCOIN from event organizer
     - View purchased tickets and ticket details in their wallet
     - Transfer tickets to other users
     - Sell and purchase tickets on the secondary market
   - Use MetaMask or another Web3 wallet for authentication and transactions.
3. Security & Validation (15 marks)
   - Ensure secure interactions using OpenZeppelin. Read through all of the docs and use what is needed.
   - Implement loading and error handling for failed transactions and smart contract interactions.
4. Documentation & Report (15 marks)
   - Provide a report explaining:
     - The overall system architecture
     - Smart contract plan, logic and implementation
     - Steps to deploy and test the contract
     - Challenges faced and how they were resolved
     - Future improvements and additional features

ROLES AND ACTIONS

1. Buyer
   - Browse available tickets.
   - Purchase tickets using cryptocurrency on primary and secondary market using TCOIN.
   - View purchased tickets in their wallet.
   - View ticket details of any ticket along with the owner's wallet address.
   - Transfer tickets to other users.
   - Verify ticket authenticity via blockchain.
2. Secondary Seller
   - List tickets on sale.
   - Set ticket prices and total supply for their own secondary sale.
   - Receive payments in cryptocurrency.
3. Event Organizer
   - Create and manage events.
   - Set ticket prices and total supply for their own primary sale. These tickets will be new NFTs minted as they are bought.
   - Edit event details: name, date, time, location, description.
   - Receive royalties of secondary sales.
4. Admin
   - Pause all Marketplace functions
   - Change secondary royalty %.
   - Change Platform fee %.
   - Change Admin address.
   - Change Treasury address.

CRITERIA

1. The dApp includes a frontend to show all necessary data for users to participate in ticket purchases and the smart contracts that facilitates the underlying logic and data storage.
2. App can handle Metamask account changes, and detect if the user is on the correct network
3. The smart contracts must be developed using the Tech Stack specified in the later slides.
4. The Solidity code must be fully documented following Natspec.
5. The smart contracts must be tested with written test cases with clear documentation. Make sure there are no security flaws and code is gas optimized.
6. The Smart Contracts should be deployed on a BuildBear network and code is verified. The Explorer links to the contracts should be added to the ReadMe.
7. The frontend code should be Web3 focused and is an organized UI. Take design inspiration from Opensea, Uniswap and TicketMaster. You dont need to copy all the functionalities, just keep the frontend simple and neat. Try to auction, sell and buy an item from Opensea testnet before you begin.
8. The website should be deployed to Netlify. The URL should be in the ReadMe.
9. The code should be uploaded to the course Github in 2 folders: Frontend and SmartContracts.
10. A smart contract plan should be uploaded to the repo. Follow the format shown in the example in the next slide. The example is unrelated to this assignment.

EXAMPLE SMART CONTRACT PLAN

The example is unrelated to this exercise

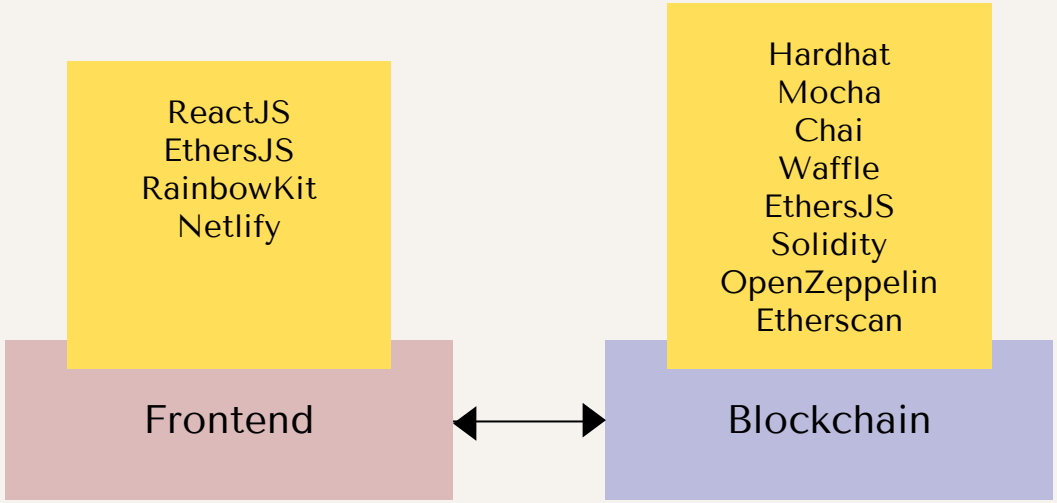
Variables		
admin	address	Address of Fund Manager
minBuy	uint	Min amt ETH to deposit to become a stakeholder
stakeholders	mapping	Mapping of stakeholder address to amount deposited
spending	mapping	Mapping of id to Spending
spendingMin VotePercent	uint	Percent of votes needed from total tokens to pass a spending request. The number should be rounded down
shareToken	address	Address of the Share Token

Spending Struct		
purpose	string	Purpose of spending
amt	uint	ETH amt to spend
receiver	address	Receiver of spending
executed	bool	Is the spending been executed?
approvals	mapping	Mapping of stakeholders to vote bool
approvalCount	uint	Number of approvals

Functions		
constructor	(admin: address, minBuy: uint)	Sets the admin and minBuy
deposit	(depositAmt: uint)	Deposit more money to stakeholder account
createSpending	(receiver: address, spendingAmt: uint)	Admin creates a Spending request
approveSpending	(spendingId: uint, vote: bool)	Stakeholders adds an approval vote to a Spending request
executeSpending	(spendingId: uint)	Send money to address if there are enough approvals

Events		
Deposit	(newStakeholder: address, depositAmt: uint)	Stakeholder has deposited tokens
Vote	(voter: address, vote: bool)	An approval vote has been sent
NewSpending	(receiver: address, spendingAmt: uint)	A new spending request has been made
SpendingExecuted	(executor: address, spendingId: uint)	A spending request has been executed

TECH STACK



Hardhat

The Ethers Project

TECH STACK

Frontend

- React: Frontend library to building Single Page Applications
- EthersJS: JS library used for integrating with EVM
- RainbowKit: Open source library for multi-wallet integration
- Netlify: Platform to host website

Blockchain

- Hardhat: Framework for developing, testing and deploying Smart Contracts. Uses Mocha, Chai and Waffle
- Mocha: helps document and organize tests with "describe", "it", etc
- Chai: assertion library for testing with "expect", "assert", etc
- Waffle: tools for compiling, deploying and testing smart contracts. It also provides extra Chai methods and can be added as an extension of Mocha
- EthersJS: JS library used for integrating with EVM
- Solidity: Language used to build smart contracts
- OpenZeppelin Contracts: a library of well tested smart contracts, including ERC721
- Etherscan: Block explorer
- NFT Storage or Pinata: Decentralized file storage