COV882 : Project Report

Crypto Exchange for Buying & Selling ERC20 tokens using ETH

https://cov882.jaskaransinghbhalla.in/
https://github.com/jaskaransinghbhalla/cov882

Submitted to :
Prof. Subodh Sharma

Submitted by :
Jaskaran Singh Bhalla (2021TT11139)
Pranav Sharma (2021EE30614)

Problem Statement

Create a crypto exchange where users can buy and sell ERC20 tokens using ETH.

In this project our goal is to build and deploy smart contracts which allow users to buy and sell ERC20 tokens created by us using their ether. We also need to create the frontend through which the user can interact with the smart contract using his web browser and wallet. The idea behind making the project is to learn solidity, blockchain and smart contract development and trying out various frameworks and libraries like ganache, truffle, hardhat, thirdweb.js. We also learn about making our own ERC20 tokens from scratch, although ensuring their security is quite hard, so we have used existing smart contracts by third-web. All the smart contracts are deployed on Polygon Amoy Testnet.

Background

Cryptocurrency exchanges are like digital marketplaces where you can buy, sell, or trade cryptocurrencies on various blockchains. They help users convert one type of cryptocurrency into another or into traditional fiat currencies. These platforms usually charge fees for their services. So basically the idea is to buy the tokens from one exchange and sell on the other or hold them until their value increases or decreases.

ERC20 tokens are a type of cryptocurrency built on the Ethereum blockchain. They adhere to a specific standard called ERC20, which ensures compatibility with Ethereum wallets and exchanges. These tokens are widely used for crowdfunding through Initial Coin Offerings (ICOs) and for creating decentralised applications (DApps). ERC20 tokens can represent assets, ownership stakes, or utility within a decentralised ecosystem.

Web3 technologies are crucial for advancing the internet into a decentralised, more secure, and transparent ecosystem. By enabling peer-to-peer interactions, they reduce reliance on centralised authorities, enhancing user privacy and data ownership. Moreover, Web3 fosters innovation by empowering developers to create decentralised applications (DApps) with smart contracts, paving the way for new economic models and disrupting traditional industries. Ultimately, Web3 holds the potential to democratise access to information, finance, and services, promoting a fairer and more inclusive digital society. So we need some exchange where we can go, buy and sell crypto tokens.

Design Considerations & Control Flow

We have written three smart contracts for the project. The first smart contract allows the user to buy the tokens from the vendor provided that he has enough ether with him. The second smart contract on the other hand allows the user to sell the tokens to the vendor provided that vendor has enough ether to return to the user. The third smart contract can be used by the vendor of the crypto exchange to withdraw all tokens from the exchange. The owner of the exchange and the exchange itself are two different smart contracts. So, the exchange is an “ownable” smart contract that means that it can be owned by another address. Amount of tokens per matic or ether becomes an important parameter. It decides the exchange rate of the token or value of 1 token.

ERC20 tokens contract consists of six major functions: totalSupply, balanceOf, transfer, transferFrom, approve and allowance.

1. **totalSupply:** This function returns the total number of tokens issued by the contract.
2. **balanceOf:** It allows anyone to check the token balance of a particular address.
3. **transfer:** This function allows the owner of the tokens to send a certain amount of tokens to another address.
4. **transferFrom:** This function allows a third party (approved by the token owner) to transfer tokens on behalf of the owner. It's commonly used in scenarios like decentralised exchanges.
5. **approve:** This function allows the owner of the tokens to authorise another address to spend tokens on their behalf, typically up to a certain limit.
6. **allowance**: It allows anyone to check the amount of tokens that an owner has allowed a spender to withdraw from their account, using the transferFrom function.

We can write all of the above functions from scratch, but there are vulnerabilities like reentrancy attacks. A reentrancy attack is a type of vulnerability that can occur in smart contracts, particularly on the Ethereum blockchain. It involves a situation where an external contract calls back into the calling contract before the initial call completes. This can lead to unexpected behaviour and potentially allow an attacker to manipulate the contract's state or steal funds. There can be lack of access control to contract, economics attacks etc.

We have used the ERC20 base smart contract made available by third-web through oppenzeppline, which is one of the most used packages for this purpose. We have deployed an ERC20 token named “Pranav Jaskaran Coin”, symbolised “PJC” on Polygon Amoy Testnet using thirdweb’s client.

The address for the smart contract is
“0xff348fac3e57469943F8C85eAF76b14EDa05e253”.
We have minted 10000000 tokens using theMATIC from Polygon faucet. The owner of this smart contract is our wallet address and we can communicate with it for any kind of changes.

The vendor smart contract is also deployed on the Polygon Amoy Testnet and takes the token smart contract’s address as input and links to the instance. This smart contract is entirely written by our team. This smart contract is created and owned by our wallet address. In this smart contract we have implemented three main functions.

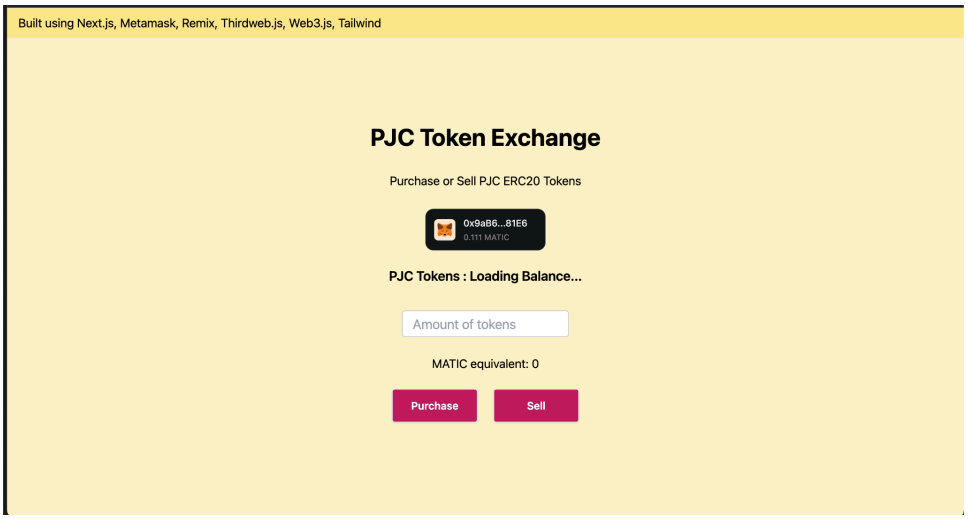
1. **buyTokens :** allows the wallet to buy tokens
2. **sellTokens** : allows the wallet to sell tokens
3. **Withdraw** : allows the owner address to withdraw all the tokens

In the frontend logic, we allow the user to input the tokens and then we calculate the equivalent amount in MATIC. The user can then buy or sell the tokens depending upon the MATIC in the account. MATIC is the native cryptocurrency of the Polygon network. We have also indicated the current wallet address and the amount of PJC tokens owned by the current wallet address.

In the backend javascript logic, we basically communicate with the deployed smart contracts using web3.js. We have primary two functions to buy and sell tokens respectively. The buy function gets the current wallet address and communicates with the vendor smart contract and executes the buyTokens function by providing the appropriate params. The sell function also does a similar thing except that sets the allowance to sell the tokens.

During the project we also tried to run a local blockchain using ganache and compile smart contracts using truffle and deploying on it. This seems to be an older method, so we shifted to deploying on testnet as addresses won’t change on Testnet and we need addresses for our contracts to remain fixed.

The image below shows the UI used for the web page.



Technologies & Packages

1. **Javascript:** A widely used programming language that is commonly used for building web applications and interacting with blockchain networks through libraries like web3.js.
2. **thirdweb.js:** A JavaScript library for building decentralised applications (dApps) on the ThirdWeb platform, providing tools and utilities for interacting with blockchain networks and smart contracts.
    a. **@thirdweb-dev/react** : A React framework package provided by ThirdWeb for developing user interfaces for decentralised applications.
    b. **@thirdweb-dev/sdk** : ThirdWeb's software development kit (SDK) offering a suite of tools and utilities for building decentralised applications.
3. **web3.js:** A widely used JavaScript library that provides an interface for interacting with the Ethereum blockchain, allowing developers to read data from and write data to the blockchain.
4. **Metamask:** A popular browser extension that serves as a cryptocurrency wallet and allows users to interact with Ethereum-based decentralised applications directly from their web browsers.
5. **Ethereum:** A decentralised blockchain platform that enables the execution of smart contracts and the development of decentralised applications (dApps).
6. **Polygon:** A scaling solution for Ethereum that aims to improve scalability and reduce transaction costs by using a sidechain architecture.

7. **Tailwind CSS:** A utility-first CSS framework that provides a set of pre-designed components and utility classes to simplify the process of styling web applications.
8. **Next.js:** A React framework for building server-side rendered and statically generated web applications, providing features such as routing, code splitting, and server-side rendering out of the box.
9. **Remix IDE:** An integrated development environment (IDE) for writing, testing, and deploying smart contracts on the Ethereum blockchain.
10. **Solidity:** A programming language used for writing smart contracts on the Ethereum blockchain. All the smart contracts have been compiled using solidity 0.8.20.

Deployment

The project has been deployed using the standard next.js deployment script on Vercel and the repo for the code is hosted on github. The domain for the website is a subdomain of our personal webpage via godaddy.
https://cov882.jaskaransinghbhalla.in/

Conclusion & Future Prospects

The overall project goal was achieved as it was an introductory project to blockchains where we tried to build a basic version of one of the most advanced decentralised applications. We learnt about the basics of blockchain, writing smart contracts, making frontend communicate with the smart contracts, learning various technologies and concepts.
However the scope of the project can be expanded by deploying multiple crypto currencies contracts on the blockchain and creating something called a liquidity pool for each of them. The pricing of each token can be determined dynamically. The tokens can be used to communicate and perform transactions with other decentralised applications.

References

1. Third web documentation (https://portal.thirdweb.com/)
2. Minimum-Viable-Exchange(https://medium.com/@austin_48503/%EF%B8%8F-minimum-viable-exchange-d84f30bd0c90)
3. https://www.youtube.com/watch?v=kWhC-BfbpmU
4. https://www.youtube.com/watch?v=UnNPv6zEbwc