Projects Descriptions

By Luis Rivero.

- What kind of technologies do I know?

- Solidity: Development of smart contracts, Testing, Using Ethers.js, Hardhat framework.

- JavaScript: React, Svelte, Node.js (Basic).

- C++ and Java: Basic POO.

- Git and GitHub.

- Actual studies

- Systems engineering, **Universidad de Oriente** 10/2016 – Today | Maturin, Venezuela.

- GitHub Link

- https://github.com/Ljrr3045

1. Project Name: VOTING DAPP

- Technology used: Solidity, ethers.js, Hardhat Framework.

- Project Description: Create a voting smart contract, where candidates will be posted, and the winner will be selected after a certain amount of time.

- Challenges during developing:

- Owner can post 5 candidates max
- Every user needs to register in the contract
- Users can only vote for 1 candidate
- Candidates cannot vote for themselves
- After 1 week voting ends, no one can vote
- Make contract upgradeable
- Create voting rounds simultaneously.

- Development explanation:

This project was brought to reality by implementing 4 contracts, each one of which fulfilled a specific role in the project, all this in order to segment the project into simpler units, in order to have better control of each process of the project, later through of the inheritance, all the contracts were merged into a main one called "VotingSistem.sol" which contained all the fundamental logic of the election process.

There is an N2 version of the contract, this was created by checking in the test, if the main contract "VotingSistem.sol" had been successfully created as an upgradeable contract using Openzepplin upgrades contracts.

Each contract has its corresponding testing, all this in the testing folder

Contracts:

- **OwnerContract**: It serves to verify the identity of the owner of the contract or that of the person who displays it
- **CanditContract**: this serves so that only the owner of the contract can register candidates for the election.
- **Regist**: It serves to allow users to register in the system to participate in the election
- **VotingSistem**: In this contract the registration of the votes of each voter is carried out
- **VotingSistemV2**: serves as a test to check if the proxy contract can update the contract

- Notes:

- The operation of the different functions and the explanations of the different things that each part of the contracts does is commented throughout the entire code of the project, together with their respective tests.
- This was the first project I developed in solidity

- Link to GitHub project (Demo):

https://github.com/Ljrr3045/VotingApp

2. Project Name: NFT DROP

- Technology used: Solidity, ethers.js, Hardhat Framework, IPFS (Pinata).

- Project Description: Create an NFT drop paying with ETH, storing images on IPFS

- Challenges during developing:

- Create an ERC721 token with OZ
- Create an ERC1155 token with OZ
- Create a Drop contract choosing either of the standards
- Find some images on the internet and upload them to IPFS
- Drop should have a hidden image before tokens are revealed, also in IPFS
- Contract should have a function to mint by paying 0.01 ETH for each token (user can buy multiple tokens too)
- Drop contract should be Pausable, Burnable, Mintable
- Should have a start state to allow minting
- Drop should have a whitelist to allow certain people to mint before sale starts
- Create Access Control Roles for ADMIN and MINTER
- Test with opensea rinkeby

- Development explanation:

This project is essentially structured in three contracts, it consists of an erc721 token contract and another erc1155, each of these contains within the different required functionalities such as mint, brun, etc... these functions can only be called by the owner of the contract, in order to prevent any user from accessing these functionalities and affecting the contracts, later this contract property will be transferred to the main contract that will fulfill the drop contract role "Drop.sol", in

which the user through of payments with eth can be made with some nft, and thus only the drop contract can create nfts, burn and etc.

This contract has the functionality of creating a white list to carry out a private sale of the nft or a public sale can also be carried out normally, the addresses assigned as admin will be able to modify these characteristics, like others.

Different openzepplin contracts are used throughout the contracts in this project.

This project can be found deployed on the rinkeby network (addresses on github), as well as having nft images uploaded using ipfs (with pinata to be specific), if you own any nft from this collection you can use opensea and trade sunft However you want.

There is a version 2 of the contract which integrates the functionality of buying native tokens of the project in order to be able to buy nfts, all this by changing eth for our token.

This project has various cases in its testing, both for errors and the project's functionality.

- Notes:

- The operation of the different functions and the explanations of the different things that each part of the contracts does is commented throughout the entire code of the project, together with their respective tests.
- The description of the specific role of each contract is found at the top of the code of each contract.

- Link to GitHub project (Demo):

https://github.com/Ljrr3045/NFT-Drop

3. Project Name: NFT Marketplace

- Technology used: Solidity, ethers.js, Hardhat Framework, React.js.

- Project Description: Building a NFT Marketplace! Users will be able to buy and sell ERC1155 Tokens in the marketplace, paying with ETH, DAI or LINK tokens.

- Challenges during developing:

- The marketplace contract should be upgradeable
- Admin will charge 1% fees for every sale (only admin should be able to update fee % and recipient address)

- Seller will create the offer, passing the ERC1155 token address, token ID, amount of tokens, deadline and the price in USD for all the tokens sold. (not price per token)
- When a user buys a token, the ETH, DAI or LINK tokens are sent to the seller
- When paying with ETH, you need to refund the buyer if he sends more ETH than the amount used for the sale
- You need to use chainlink oracles to get the latest price in USD for the payment token.
- Sellers can cancel an offer anytime.
- Buyers need to accept the whole offer, not partially (i.e. can't buy 4 out of 10 tokens offered)
- Users won't transfer the token to the marketplace when selling. They only approve the marketplace to spend their tokens.

- Development explanation:

This project integrates the use of oracles, to be specific the chainlink oracles, whose function is to obtain the current value in usd of different cryptocurrencies and tokens.

Mainly three contracts are developed, each one independently interacting with chainlink to obtain the prices of both ETH, DAI and LINK... which later through inheritance will be integrated into the main contract which is "NFT_Market.sol" .

This main contract has the purpose of working as an nft market, where users can both sell and buy nfts from different people, at no time does the contract keep the nfts, the contract makes use of the approve function of the erc1155 and through transferFron proceed to send an nft from one address to another

The user who is interested in making a sale must specify certain data such as the price of nft, nft stories will sell and the time limit of the sale... after this any user can make the purchase of dihco nft paying with eth, dai or link and here it is based on the price given by the seller (in usd) the price in token will be calculated using the oracles of chainlink. If after the sale time, no user has made the purchase, automatically this sale cannot be made.

This project has a short UI built with react.js. Throughout the project both openzeppelin and chainlink contracts are used.

- Notes:

- The operation of the different functions and the explanations of the different things that each part of the contracts does is commented throughout the entire code of the project, together with their respective tests.

- The description of the specific role of each contract is found at the top of the code of each contract.

- Link to GitHub project (Demo):

https://github.com/Ljrr3045/NFT_Market

4. Project Name: SWAPPER DAPP

- Technology used: Solidity, ethers.js, Hardhat Framework, axios.

- Project Description: In this project we created a tool**,** this tool allows users to make multiple swaps on the decentralized exchanges (DEX) like uniswap, balancer, curve or sushiswap, in one single transaction.

- Challenges during developing:

- The contract should be upgradeable
- Contract will never store any funds
- User will pass the % wanted for each token as an array
- A fee of 0.1% will be charged to the deposited amount, sent to a recipient address
- First version should only use Uniswap V2 or V3
- Second version (upgraded) should have one function that use the best DEX available using paraswap routing.

- Development explanation:

This project is related to the creation of a tool that allows the user to change an amount of ETH or some other token, in different tokens, where the user will indicate the proportion that he wants to be allocated from the initial amount to the new token that he wants. You can in addition to this change the initial amount in multiple tokens in the same operation

This project has two versions, in the first I make use of the unisiswap V3 protocol contracts to carry out these various exchanges. For the second version of this project I use the paraswap protocol to carry out the exchanges, this protocol has the peculiarity that it seeks the best exchange price available in a series of exchanges available in the protocol.

The testing of this project is created for the first version in the ethereun network while for the second version it is developed in the polygon network

For the test of the second version, the axios library is used to make request and post calls in the paraswap api, which enters the encoded data to be able to use its functions in solidity.

- Notes:

- The operation of the different functions and the explanations of the different things that each part of the contracts does is commented throughout the entire code of the project, together with their respective tests.
- The description of the specific role of each contract is found at the top of the code of each contract.

- Link to GitHub project (Demo):

https://github.com/Ljrr3045/SwapperDapp

5. Project Name: NO LOSS LOTTERY DAPP

- Technology used: Solidity, ethers.js, Hardhat Framework.

- Project Description: In this project we building a no loss lottery using DeFi, Users buy lottery tickets with tokens, then all funds are invested in DeFi and earn interest, and the lottery winner gets all the interest earned. Losers can claim back the ticket cost, so they never lose.

- Challenges during developing:

- The contract should be upgradeable
- Users should be able to deposit any of the top stablecoins (DAI, USDC, USDT) or ETH.
- For stablecoin swaps, it is better to use Curve Finance, as the slippage is lower than
- Uniswap.
- You should think about how to define the lottery ticket system and cost, depending on which asset the user deposits or stakes.
- All funds should be invested in Aave or Compound pools. You should choose the pool to invest in. So each lottery will invest in one specific pool (e.g. aDAI in compound)
- Admin will charge 5% fees for every lottery, only from the interest earned.
- Lottery winner is chosen every week, using a random number from chainlink oracles.
- There will be 2 days for users to deposit funds and then the lottery starts and funds are invested in Pools

- If a lottery is already running, the funds deposited by users will be invested in the next week's lottery, to avoid people from participating in a lottery one day before the winner is chosen.
- Lottery winner is chosen 5 days after the lottery starts. So 2 days to prepare and collect funds, and 5 days for interest to be accrued and select a winner

- Development explanation:

This project essentially has two contracts, one of these interacts with the chainlink protocol to obtain a random number, since in solidity there is no native way to generate a random number and make it reliable, therefore we proceed to use oracles like in this case the chainlink.

then there is the main contract, which allows users to buy tickets to participate in a lottery where users will never lose money, in this contract defi protocols are implemented, such as uniswap which is used to exchange ETH For DAI, the curve protocol is also used, which is an exchange which allows us to exchange stable currencies in such a way that the least possible change is lost, we use this to make exchanges between USDT and USDC for Dai.

Subsequently, the money collected in DAI proceeds to be invested in the defi compound protocol, in order to generate interest and obtain profits with the money invested.

After a specific number of days, the winner of the lottery will be able to withdraw their winnings and the losers will be able to withdraw their invested money, people who invested with a stable currency will be able to choose between withdrawing their money in usdt, usdc or dai; as long as the people who bought tickets with ETH received their money in WETH.

- Notes:

- The operation of the different functions and the explanations of the different things that each part of the contracts does is commented throughout the entire code of the project, together with their respective tests.
- The description of the specific role of each contract is found at the top of the code of each contract.

- Link to GitHub project (Demo):

https://github.com/Ljrr3045/NoLossLotteryDapp

6. Project Name: LP STAKING DAPP

- Technology used: Solidity, ethers.js, Hardhat Framework, React.js, CSS, Netlify.

- Project Description: In this project we building a simple staking contract. We should provide a way forusers to add liquidity to Uniswap pools, stake the LP tokens in your staking contract to earn rewards, or do all this in one single transaction, starting with ETH or any of the underlying assets from the liquidity pools you chose.

- Challenges during developing:

- The contract should be upgradeable
- Create your own rewards token.
- Users should be able to:
    - Add liquidity to Uniswap ETH-DAI pool, starting with ETH only
    - Stake the LP tokens in your contract to earn rewards
    - Be able to stake LP using the uniswap “permit” functionality to avoid approval
    - Add liquidity + stake in one single transaction
- You can build your own staking contract, or search for existing ones, like the synthetix or master chef staking contracts.
- Create a basic frontend where users can interact with your contract using Metamask wallet.

- Development explanation:

This project consists of the use of three base contracts, plus a contract for an erc20 token.

One of the contracts fulfills the function of receiving ETH for later, through the uniswap V2 protocol. exchange these ETH for DAI in a suitable proportion which will allow us to add liquidity to the ETH-DAI pool of uniswap, in such a way that we have the least amount of dai or eth as possible.

then we have another contract which will fulfill the role of a contract to make a staking, where users, once after having added liquidity to the uniswap pool, can place their LP tokens in this stake and thus generate profits, which were paid in the erc20 token of the project.

Finally we have the main contract which inherits the previous contracts, in this there are only two functions, one where the user invests money in the project (all the established requirements are carried out and the permit function is used so that the user does everything in an operation) and then we have another function which returns the lp toekns to the user plus their earnings in the erc20.

this project has a front end integration, where react is used to create it in combination with css, the ethers.js library is used to be able to interact with the contracts. This

project is hosted on Neylify, in addition to the fact that it has the use of a subgraph to read data from the block chain in a more efficient way.

This project is deployed on the ropsten network.

- Notes:

- The operation of the different functions and the explanations of the different things that each part of the contracts does is commented throughout the entire code of the project, together with their respective tests.
- The description of the specific role of each contract is found at the top of the code of each contract.

- Link to GitHub project (Demo):

https://github.com/Ljrr3045/LP_StakingDapp

7. Project Name: House Token Subgraph

- Technology used: TypeScript, graphQL, Subgraph Studio.

- Project Description: Subgraph which allows developers to read data from the blockchain in a more efficient way and thus be able to integrate it into user interfaces, in this case it is a subgraph for an erc20 token deployed on the ropsten network.

- Development explanation:

At the time of creating this project, the graph's subgraph studio tool was used, which greatly facilitates the development of this project.

- Link to GitHub project (Demo):

https://github.com/Ljrr3045/HouseTokenSubgraph