DI-FCUL **MEI/MI/MSI/SDS**

Intrusion (Detection and) Tolerance 2022/2023

Project 3 – Decentralized Finance

This project aims to construct a decentralized finance application that facilitates token swapping between ETH and a custom DEX (fungible) token on the Ethereum blockchain. The application allows users to buy and sell DEX. Also, it enables users to leverage their DEX holdings and NFTs as collateral to borrow ETH. Specifically, they can borrow up to half the value of their DEX holding or NFT tokens in ETH. Once the loan is repaid, users can retain ownership of the DEX or NFT tokens presented as collateral.

Project Description

This project is composed of three main components, following the same ideas as presented in the decentralized applications lab:

1. *A smart contract*: the server-side logic of the application.
2. *A java-script file*: the client-side logic of the application.
3. *A HTML file*: the application's layout.

In the following, we describe how each component must be implemented.

Smart Contract

Numerous token standards exist on the Ethereum blockchain, each presented to serve a specific purpose. Previously, we explored ERC721, the standard for *non-fungible tokens* (NFTs), where each token is distinct and unique. Another significant standard is ERC20 [1,2], designed for *fungible tokens*, like traditional currencies such as dollars or euros. Since in this project, we want to define our custom token, DEX, it is more appropriate to use the ERC20 standard to develop the smart contract component.

Implementing this smart contract requires defining the following variables:

- **owner**: that determines the owner of the contract.
- **struct Loan:**
  - **deadline:** determines the deadline of a loan. rule like paga +5% a cada minuto que passa
  - **amountEth:** determines the amount of a loan in ETH.
  - **lender:** determines the address of the lender of a loan (the lender could be this smart contract or another user).
  - **borrower:** that determines the address of the borrower.
  - **isBasedNft:** this variable is true if an NFT is used as collateral; otherwise, it is false.
  - **nftContract:** in case *isBasedNft* is true, this variable stores the contract managing the NFT presented as collateral.

    - **nftId:** in case *isBasedNft* is true, this variable determines the id of the NFT presented as collateral.
- **maxLoanDuration**: each loan has a specific deadline for repayment; this variable determines the maximum loan duration.
- **rateEthToDex:** the swap rate of ETH and DEX tokens.
- **ethTotalBalance**: the total amount of ETH held by the contract.
- **loans**: a map to track the loans (each loan must be mapped to a unique id, that can be implemented using a counter).

The smart contract must provide the following functions:

- **constructor():** in the constructor, in addition to initializing variables, 10^{30} DEX tokens must be minted (created).
- **buyDex()**: using this function, a user can buy DEX tokens by paying ETH. Converting ETH to DEX or vice versa requires considering the swap rate.
- **sellDex(uint256 dexAmount):** using this function, users can exchange their DEX tokens for ETH if there is a sufficient balance of ETH available in the contract.
- **loan(uint256 dexAmount, uint256 deadline):** this function enables users to take out loans in ETH by using their DEX tokens as collateral. Users need to specify the number of DEX tokens they wish to stake for the loan, as well as the desired payback deadline, which should not exceed the *maxLoanDuration*. To calculate the loan value in ETH, we first calculate an *initial value* based on two considerations:
    - the longer the payback deadline, the lower the value of ETH per DEX, and
    - *rateEthToDex* must be considered to calculate the initial value.

  Then, the function transfers half of the calculated initial value from the contract to the user's account. This action must be logged in the *loans* mapping and return the loanId.
- **returnLoan(unit256 loanId, uint256 ethAmount):** this function allows a user to pay back his borrowed ETH, recovering a proportional amount of DEX. Notice that users can make partial repayments on their loans that are not NFT-based. This could benefit the users who can repay only part of the loan amount by the deadline.
- **getEthTotalBalance():** this function returns the total amount of ETH available in the contract.
- **setRateEthToDex(uint256 rate):** using this function, the owner of the contract can change the swap rate.
- **getDex():** this function returns the total amount of DEX tokens that a user owns.
- **makeLoanRequestByNft(IERC721 nftContract, uint256 nftId, uint256 loanAmount, uint256 deadline):** this function allows a user to make a loan request using an NFT as collateral. The user specifies the NFT contract and the specific NFT id, the desired loan amount in ETH, and the deadline for loan repayment. This function creates an instance of *Loan,* without determining a lender, i.e., setting the lender field as the default address 0.
- **cancelLoanRequestByNft(IERC721 nftContract, uint256 nftId):** this function allows a user to cancel an existing NFT-based loan request. The user needs to specify the NFT contract and the specific NFT id associated with the loan request.
- **loanByNft(IERC721 nftContract, uint256 nftId):** assuming user A asked for a loan using an NFT as collateral, this function allows another user B to lend the requested amount to A. In further detail, given a loan instance from struct *Loan,* the contract sets the lender address to B and locks the required amount of DEX tokens from B’s balance; if A repays the loan within

the deadline, the contract releases the NFT and the staked DEX tokens back to their owners. However, if A fails to repay the loan within the deadline, the contract transfers the ownership of the NFT to B and keeps the staked DEX tokens.

- **checkLoan(unit256 loanId):** this function allows the owner of the contract to check the status of a loan. If the loan's repayment deadline has passed without repayment, the function must take the necessary steps to punish the borrower (described in *loanByNft* function).
- **event loanCreated(address borrower, uint256 amount, uint256 deadline):** this event gets emitted when a new loan is created.

Java-Script and HTML

The client-side logic and the application layout should allow the owner of the contract to:

- set the exchange rate,
- be informed when a loan is created, and
- check the status of created loans every 10 minutes by calling *checkLoan* (It is worth mentioning that since Solidity does not have a timeout mechanism, without an off-chain application, the smart contract component cannot check the status of the loans and punish the borrowers who did not repay their loans on time. Therefore, we need this or similar mechanisms to deal with loans whose deadlines have expired).

Further, the client-side logic and the layout of the application should allow the users to:

- buy DEX tokens,
- see the total amount of his/her DEX tokens,
- sell his/her DEX tokens,
- ask for a loan (using DEX or NFT),
- return the borrowed ETH,
- see the total amount of ETH that the contract has,
- see the exchange rate,
- see the available NFTs to lend ETH to other users, and
- see the total amount of borrowed and not paid back ETH tokens.

Project Delivering

The delivery of the project should be a zip file containing:

1. The code of the project
2. A README file explaining how to run the project.

The project must be delivered on the course Moodle on Abril 28th (23:55hs).

References

[1] https://ethereum.org/en/developers/docs/standards/tokens/erc-20/
[2] https://docs.openzeppelin.com/contracts/4.x/erc20