Ethereum Decentralized Apps: Implementing a Frontend Using web3

1 Overview

In this exercise sheet, you are asked to implement a frontend to the BBSEBank decentralized application that you have built in the previous exercise sheets. The goal of this exercise sheet is to help you get familiar with web3.

- You can use the GitHub repository for BBSEBank `https://github.com/sebischair/bbse-bank-2.0`.
- You can fork, clone, or just download the ZIP version of the repo. You don't need to upload your solution anywhere.

2 Requirements & Tasks

Choose your favorite framework (e.g., React, Angular, Vue) to build a frontend for BBSEBank. The frontend has the following requirements:

- User should be notified if MetaMask is not installed. Remember that MetaMask is required to inject a blockchain connection to the client-side code.
- A button should exist for users to connect their MetaMask wallet to the application.
- User address, ETH balance, and BBSE token balance should be displayed upon connection.
- **Deposit**
  - User should be able to deposit ETH.
  - Minimum allowed amount should be 1 ETH.
  - Info about current yearly return rate and interest per second (based on entered deposit amount and yearly return rate) should be displayed.
  - For an active deposit, deposit amount, start time, and accumulated interest (depending on the current known block number) should be displayed.
  - User should be able to withdraw their deposit.
- **Loan**
  - User should be able to borrow ETH.
  - Info about collateralization rate and loan fee of the bank should be displayed.
  - An approximately required minimum collateral amount should be displayed, based on the entered loan amount. It is approximate since collateral value depends on ETH/BBSE rate in the oracle.
  - When a loan is taken, the loan amount (ETH) and collateral value (BBSE) should be displayed.
  - User should be able to pay their loan.
- **Approve**
  - User should be able to set an allowance for BBSEBank to transfer BBSE tokens from their account when taking a collateral.
  - Info about current bank allowance should be displayed.

3 Tips

- Don't forget to run `npm i` inside the root directory when you first clone the project.
- Deploy the contracts to Ganache using `truffle migrate`.
- Run the provided test suites to make sure the contracts are working as expected.
- Remember to run the oracle server after you have deployed the contracts. Check out the previous exercise sheet to see how you can run it.
- Message value is represented in Wei, not Ether. **Don't forget to do the necessary conversions!** If you are confused, use a tool like `https://eth-converter.com/`.
- Useful functions
  - **Web3(window.ethereum)**: Obtain a web3 object by injecting blockchain connection through MetaMask's global API window.etherum.
  - **web3.eth.Contract(abi, address)**: Get an instance of a deployed contract
  - **web3.eth.getBalance(address)**: Get ETH balance of an address
  - **web3.eth.getBlockNumber()**: Get current block number
  - **web3.utils.fromWei(amount, 'ether')**: Convert Wei to Ether
  - **web3.utils.toWei(amount, 'ether')**: Convert Ether to Wei
  - **web3.eth.sendTransaction({from:account, to: bank.address, value: web3.utils.toWei(amount, 'ether')})**: Send Ether from an account to the bank contract. Remember that bank must hold enough ETH to give out loans.
  - **contract.methods.function_name().call()**: Use this notation to call any view function or public variable available in a contract. No transactions will be issued.
  - **tokenContract.methods.balanceOf(address).call()**: Get token balance of an address
  - **bankContract.methods.investors(address).call()**: Access deposit object of an address
  - **bankContract.methods.MIN_DEPOSIT_AMOUNT().call()**: Access min. deposit amount
  - **contract.methods.function_name().send({from: account, value: amount})**: Use this notation to call any function on a contract. A transaction will be issued.
  - **bankContract.methods.deposit().send({from: account, value: web3.utils.toWei(depositAmount, 'ether'), })**: Calls deposit function on the bank contract. A transaction will be issued.
  - **token.methods.allowance(account, bankContract.address).call()**: Check token transfer allowance of bank contract on given account.
  - **token.methods.approve(bankContract.address, allowanceAmount)).send({ from: account, })**: Approve allowanceAmount tokens to be transferable by the bank contract.
- Don't forget that calls to a blockchain are asynchronous. So, you should **await** them.
- If MetaMask says it wasn't able to estimate the required gas amount, then probably the transaction will fail. This could be due to things such as the bank contract not having enough ETH, the user not setting required amount of allowance, or the user not having enough BBSE tokens to collateralize.
- Make sure that your bank has enough ETH and the user sets the required amount of allowance before trying to take a loan.

- Remember that allowance amount gets updated when the allowed account transfers tokens from the allowing account.
- **Be careful when copying code or commands from the exercise sheet!**