Development of a GUI for Interaction with a Solidity Smart Contract - Project Report

CMPE483 - Project 1
Fall Term 2023
Alp Tuna - 2019400288
Roaa Bahaa - 2023680081
Lynn Janzen - 2023690267

1 Overview

In this seminar project report we explain the creation of a GUI using React and Material-UI, selected for their user-friendly and efficient interface design capabilities. The system is backed by a Node.js server, chosen for its robust handling of backend operations. The UI is integrated with a Solidity smart contract, which was deployed on the Sepolia test network.

2 Teamwork

In our team of three, we adopted a collaborative approach to tackle various aspects of our blockchain smart contract application project. Lynn was the main responsible person for the GUI. She helped us with a user-friendly design of the interface and implemented all the components. Alp was the main responsible person for the deployment of the smart contract and of connecting front-end and back-end. Roa was the main person writing the report and manually testing the application. Overall, we split the tasks relatively equally. However, we did not ignore the other members' work, we reviewed and checked the correctness of their code logic.

3 Implementation:

3.1 Backend

The implementation is based on node.js, specifically web3.js
You can find the initialization function below. It detects the provider, which is most of the time metamask and detects the change of account in case selected account on metamask changes. In addition, myGov token contract is defined to interact with the contract with the correct parameters such as its address and ABI.

```
const contractAddress = "0x2b8D8681d3Ad24eEc713251192bE36b932dA173f";

export const init = async () => {
  let provider = window.ethereum;

  if (typeof provider !== "undefined") {
    // Metamask is installed.
    provider
      .request({ method: "eth_requestAccounts" })
      .then((accounts) => {
        selectedAccount = accounts[0];
        console.log(`SelectedAccount is: ${selectedAccount}`);
      })
      .catch((err) => {
        console.log(err);
        return;
      });
  }

  window.ethereum.on("accountsChanged", function (accounts) {
    selectedAccount = accounts[0];
    console.log(`SelectedAccount changed to: ${selectedAccount}`);
  });

  const web3 = new Web3(provider);

  // const networkId = await web3.eth.net.getId();

  const MyGovContractBuildABI = //import here
    (myGovContract = new web3.eth.Contract(
      myGovABI,
      "0x2b8D8681d3Ad24eEc713251192bE36b932dA173f"
    ));

  isInitialized = true;
};
```

Then, we have a bunch of function calls which simply triggers the corresponding smart contract function. Below is a simple illustration of those functions.

```
export const subscribeToAccountChanges = (accountChangedCallback) => {
  window.ethereum.on("accountsChanged", function (accounts) {
    accountChangedCallback(accounts[0]);
  });
};

export const faucet = async () => {
  if (!isInitialized) {
    await init();
  }
  console.log(`selected account in faucet: ${selectedAccount}`);
  return myGovContract.methods.faucet().send({ from: selectedAccount });
};

export const balanceOf = async () => {
  if (!isInitialized) {
    await init();
  }
  console.log(`selected account in balanceOf: ${selectedAccount}`);
  return myGovContract.methods.balanceOf(selectedAccount).call();
};

export const myGovBalanceOfContract = async () => {
  if (!isInitialized) {
    await init();
  }
  console.log(`get balance Of: ${contractAddress}`);
  return myGovContract.methods.balanceOf(contractAddress).call();
```

3.2 Front-end

The implementation of the front end is based on React and Material-UI for an efficient user interface. The application is structured as a single-page application in React and begins with necessary imports including React hooks, CSS, and Material-UI components for styling and layout.

The core component, `App`, is a functional component that uses the `useState` hook for managing state, specifically for maintaining the token balance of the user interacting with the contract. For the design, we created a custom theme using Material-UI's `createTheme` to provide a consistent look and feel across the app. The `useEffect` hook is used to initialize the application state when the component mounts.

The app contains multiple forms, each corresponding to a different functionality of the smart contract interaction. These functionalities include fetching the balance (`fetchBalanceOf`), calling the contract's faucet (`callFaucet`), donating ether (`sendDonateEther`), and various other actions like submitting project proposals, voting, and handling surveys. Each form utilizes Material-UI components like `TextField` and `Button` for user input and actions.

Each form has an associated handler function. These functions interact with the blockchain through imported Web3 functions, such as `balanceOf`, `faucet`, and `donateEther`, which

are defined in a separate module. These functions abstract the complexity away from the front-end logic.

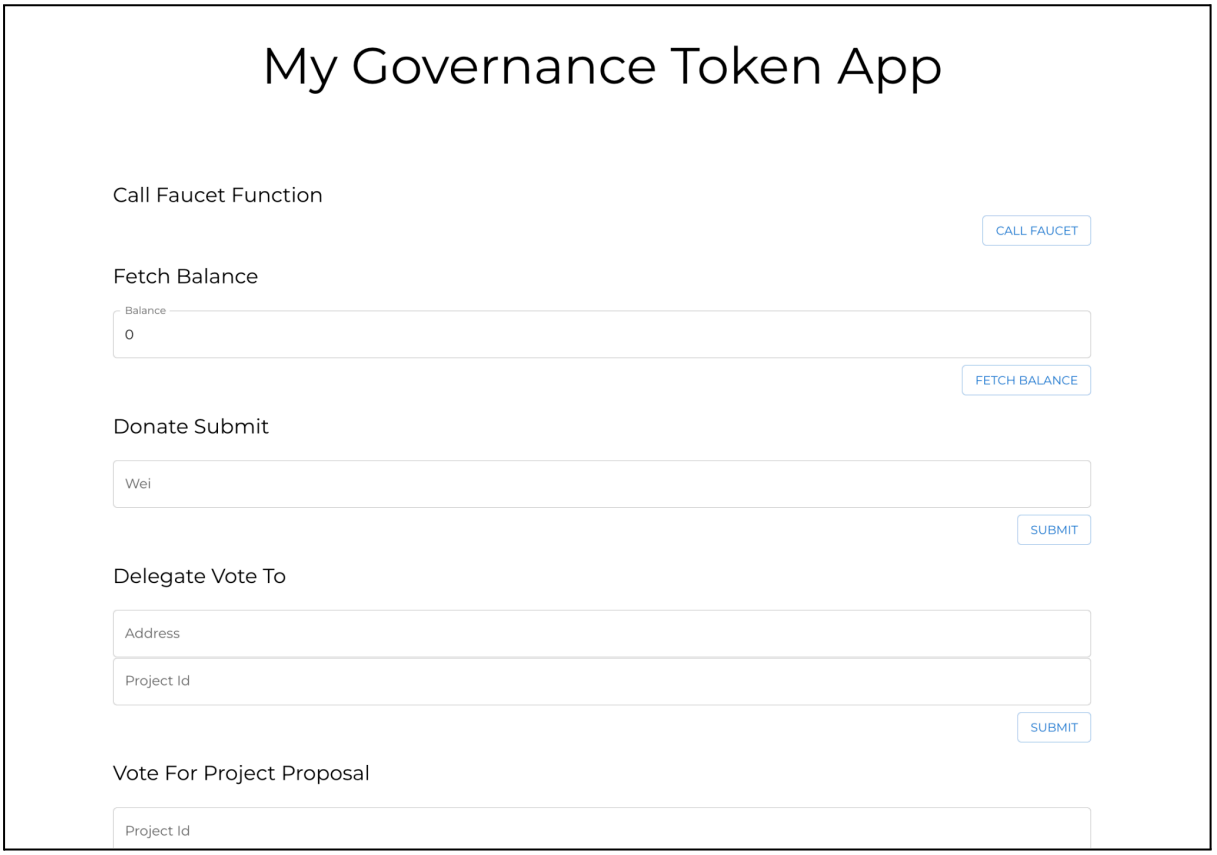

Image: Screenshot of our app's interface. The parameters for the underlying smart contract functions can be entered in text fields and the functions are called through a web3 interface on button click.

4 Possible improvements

4.1 Comprehensive Unit Testing

4.2 Increased Testing with Multiple Accounts

4.3 Exploration of Testnets in Addition to Localnets

4.4 Gas Consumption Optimization