Blockchain Explorer Application

Full Stack Development III (Winter 2022)

Due Tues, Dec 6th, 11:59PM – 25% of Final Grade

System Requirements

Contact: **Mike Denton**

Version #: **2.1**

1 Objective

This document contains a specification of the course assignment. It is a task where students practice skills to build a full stack web application using all the technologies offered by the MERN Stack. This task will also include, working together in a group developing the project, plan, manage and coordinate development activities and present the work done effectively to a deadline.

Reference Full Stack II - Complete Assignment Source code below
https://tinyurl.com/3yemwbsb

Reference Full Stack III - Lab Solution
https://tinyurl.com/bdsmccds

2 Teams

Teams will consist of one to two students (with two being the target number) and will be submitted to Black Board on the Due Date.

3 Backend Server

The backed server will include Node.js and Express with was completed in Full Stack II project and will be reused for this project.

Hardhat or Truffle development environment for Ethereum will also be used to provide a virtual blockchain. Packages for web3.js or ethers.js API can be used to interact with this environment.

4 Database

MongoDB or the cloud hosted MongoDB Atlas will be used. The database layer, including use of mongoose middleware was completed in the Full Stack II project and will be reused for this project.

5 User Interface

The React UI completed in Full Stack I project and integrated in Full Stack II project will be used for this project.

6 Specification

The application for the project will be based on one Part 1 and Part 2 of the Blockchain Explorer.

7 Blockchain Integration

There is required blockchain integration with truffle or hardhat required for this project.

8 Node Backend Server

Build a web server that will have the following functionality:

1. Provide and expose Restful API endpoints via Express
2. Built logical routes using Express router
3. Manage connection to Mongo Db via Mongoose to save and persist data
4. **Integration with Ethereum development environment using web3 or ethers.js**

8.1. Node Custom Modules

1 There are two modules named **accounts** and **transactions**

Accounts module

- will have two public methods **getAddresses** and **getBalance**

getAddresses method:

1. There are no required parameters for this method
2. Returns list of account addresses

- **Replace the static/mock data** from Full Stack II project, with a list of real account addesses from ethereum development blockchain
- **Reference API:** https://web3js.readthedocs.io/en/v1.8.1/web3-eth.html?highlight=getaccounts#getaccounts

Sample output from console when call getAddress node module

```
router account address called
getAddress:
0xa0Ee7A142d267C1f36714E4a8F75612F20a79720
0xBcd4042DE499D14e55001CcbB24a551F3b954096
0x71bE63f3384f5fb98995898A86B02Fb2426c5788
0xFABB0ac9d68B0B445fB7357272Ff202C5651694a
0x1CBd3b2770909D4e10f157cABC84C7264073C9Ec
0xdF3e18d64BC6A983f673Ab319CCaE4f1a57C7097
0xcd3B766CCDd6AE721141F452C550Ca635964ce71
0x2546BcD3c84621e976D8185a91A922aE77ECEc30
0xbDA5747bFD65F08deb54cb465eB87D40e51B197E
0xdD2FD4581271e230360230F9337D5c0430Bf44C0
0x8626f6940E2eb28930eFb4CeF49B2d1F2C9C1199
```

getBalance method

1. one parameter address - the account address to retrieve the balance
2. returns the balance and address of the account

- **Replace the static/mock data** from Full Stack II project, with a balance from a ethereum development node..

- **Reference API:**
  https://web3js.readthedocs.io/en/v1.8.1/web3-eth.html?highlight=getbalance

  Developer Note: The **default node address** for all balance and transaction requests can be one hard-coded account address value or from a target block ie. Last Block.

  Sample output from console when call getBalance node module

```
router GET:account/balance called:
getBalance module called..
account: 0xf39Fd6e51aad88F6F4ce6aB88827279cfffFb92266: balance: 98765432100
```

Transaction module

- - will have two public methods **getTransactionHistory** and **sendTransaction**
  - will have one method **getTransactionHistory (Completed in FS II)**
  - will have one method **sendTransaction**
    - will have three parameters
      - source - the source account address
      - destination - the destination account address
      - value - the amount value to send
    - using web3 send transaction method to send the amount
    - **using Transaction mongoose model, save the transaction (Reference Section 8.2)**
    - returns a transaction receipt object from the web3.js method
  - **Reference API:**
    https://web3js.readthedocs.io/en/v1.8.1/web3-eth.html?highlight=sendtransaction#sendtransaction

8.2 Mongoose and MongoDb

Build models and schemas with mongoose middleware to store data in mongoDb

1. Query data in local mongo database **(Completed FS II)**
2. Build **Transaction** model and schema to retrieve **transaction** history **(Completed FS II)**
    a. schema should include the following fields:
        i. **source** - a source account address
        ii. **destination** - a destination account address
        iii. **amount** - amount value to send
        iv. **status** - result of the transaction
        v. **gasUsed** - the amount of gas used, this is an optional field and will only be stored if transaction was successful
        vi. **receiptHash** - the transaction hash from the receipt object. this is an optional field and will only be stored if the transaction was successful
3. **Save Transaction** - use the model created in FS II and save the transaction. Refer to mongoose **save** method.

    Reference: https://mongoosejs.com/docs/models.html#compiling

8.2 Express and Router

1. Create routes for **accounts**
    a. **GET** request on route **/account/addresses - (Completed FS II)**
    b. **GET** request on route **/account/balance - (Completed FS II)**

2. Create routes for **transaction**
    a. **GET** request on route **/transaction/history - (Completed FS II)**
    b. POST request on route **/transaction/send**
        i. calls the **sendTransaction** method in the transaction custom module (Reference Section 8.1)

** optional: body parser middleware to help post request
https://www.npmjs.com/package/body-parser

9.0 React UI Integration

- integrate the existing React UI from Fullstack II to fetch data from the backend APIs for the read only components. The form input screens including POST requests are in scope for this project.

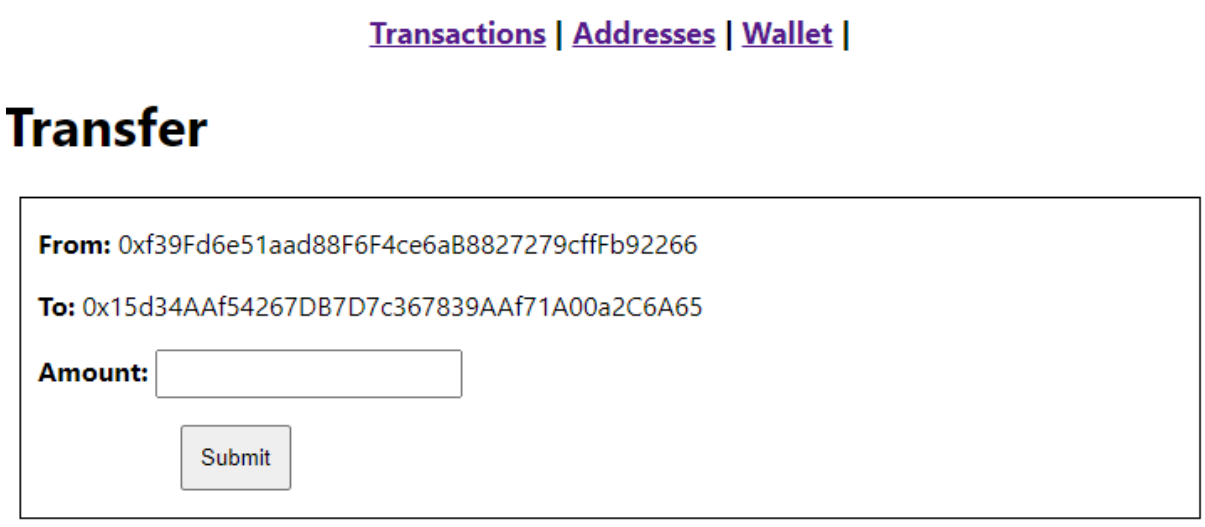
9.1 Blockchain Node Address Component - (Completed in FS II)

- use **fetch** or **axios** to send a **GET** request to the route **/account/addresses** **(Completed FS II -** No new requirements**)**

9.2 Transaction History Component - (Completed in FS II)

- use fetch or axios to send a GET request to the route **/transaction/history**
- **(Completed FS II -** No new requirements**)**

9.3 Transfer/Send Ether



Component Requirements

- **Developer Note:** The **default node address** for all balance and transaction requests can be one hard-coded account address value or from a target block ie. Last Block.

- Interactive screen that displays the **default node/accout address** and the Node address passed via route parameter from the previous Address page.

- This component will contain an interactive form and will handle the submission of the form via the user click action on the submit button.

- This component will send a **POST request** to the Express API route **transaction/send.** The data payload will include the source, destination addresses as well as the amount

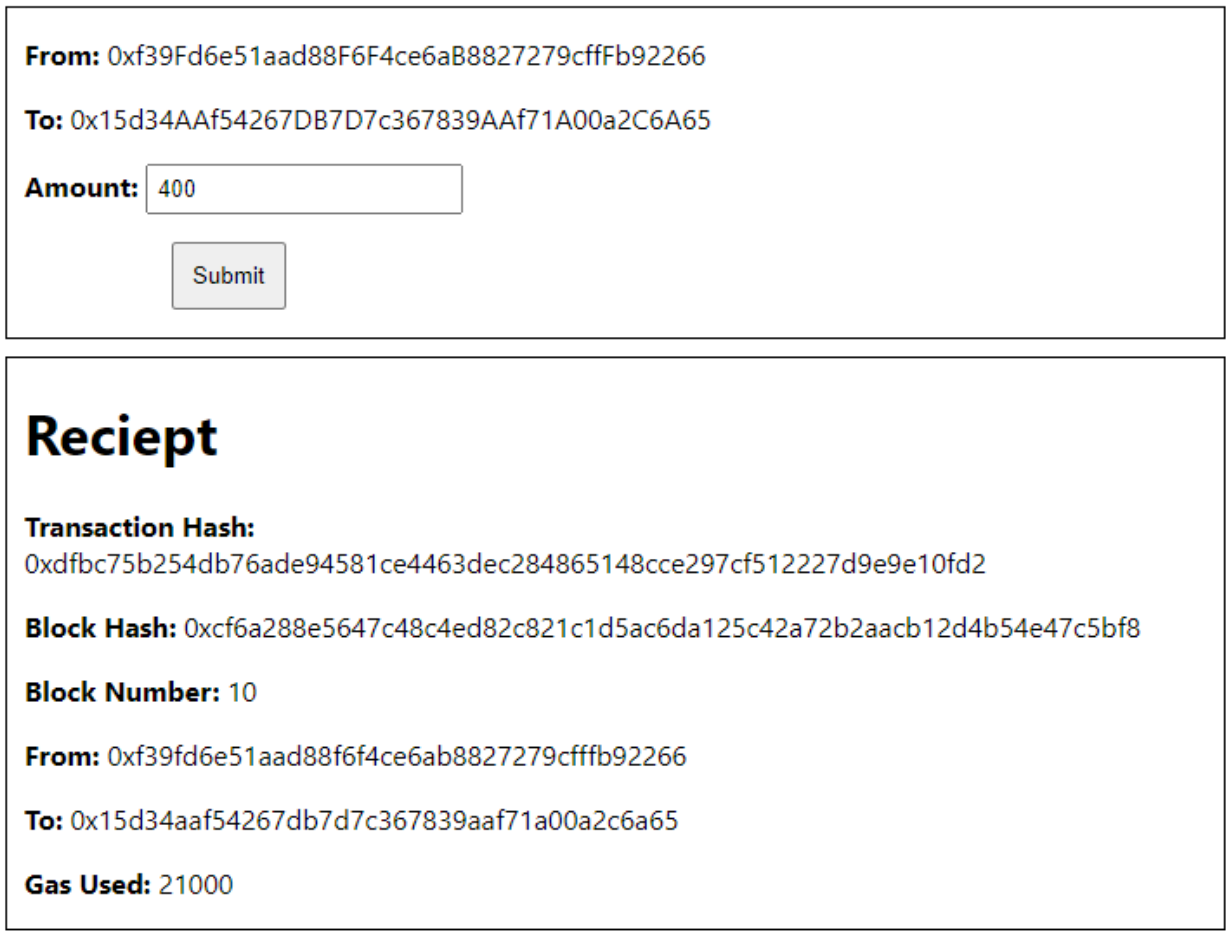
- The component will show the **Transfer Receipt (Section 9.4)**, if the **POST request** was successful. Otherwise, the receipt should not be shown

9.4 Transfer Receipt

Component Requirements

- This component will contain read only data populated from the receipt payload from the POST request in **(Section 9.3)**

- This component will hide/show based on if there is data or not. This will be visible and data populated in the **POST request** was successful.



9.4 Wallet Component - (Completed FS II)

- use fetch or axios to send a GET request to the route **/account/balance**
- **(Completed FS II -** No new requirements**)**

My Wallet

Address: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266

Balance: 98765432100 ETH

-

11 Submission -

1. MERN full stack application submission

This project is expected to compile and run. **This project will be submitted via BlackBoard.** The application should be deployed to Heroku for demo purposes (optional). Each project must include a READ.ME file with the members of the team, with student numbers, along with any instructions or credentials needed to run the application.

Specification	Percentage
React Frontend Application	**30%**
Node/Express/Mongoose	**30%**
Integration with backend Ethereum dev environment	**30%**
Clean Code and Clarity	**10%**