Univerzitet u Beogradu

Fakultet organizacionih nauka

Katedra za elektonsko poslovanje

SOLIDITY-REACT projekat

Razvoj naprednih aplikacija elektronskog poslovanja

Mentor: Student:

Milica Simić Marko Bojanić 2023/3270

Beograd, 2024.

Contents

[1. Opis aplikacije 2](#_Toc156386143)

[2. Prikaz Main.js fajla: 7](#_Toc156386144)

[3. Prikaz Main.css fajla 9](#_Toc156386145)

[4. Prikaz NewClientModal.js 10](#_Toc156386146)

[5. Prikaz NewClientModal.css fajla 12](#_Toc156386147)

[6. Prikaz Transaction.js fajla 14](#_Toc156386148)

[7. Prikaz Transaction.css fajla 18](#_Toc156386149)

[8. Prikaz ClientList.js fajla 19](#_Toc156386150)

[9. Prikaz ClientList.css fajla 20](#_Toc156386151)

[10. Prikaz ClientDetailsModal.js fajla 21](#_Toc156386152)

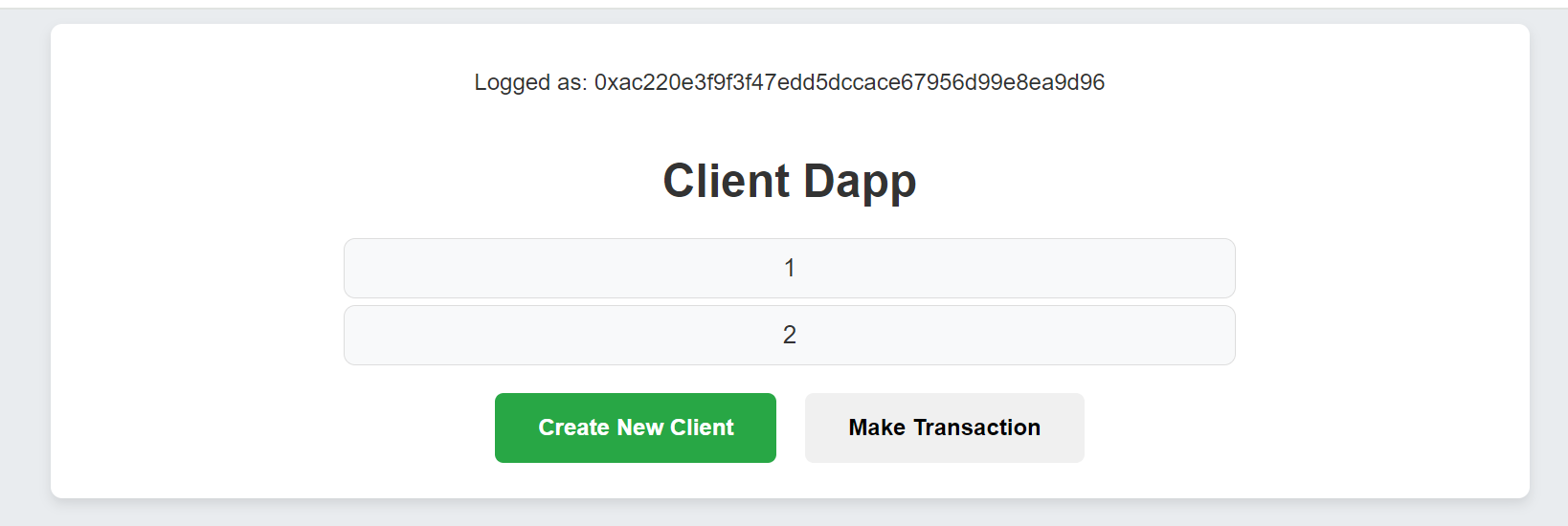
[11. Prikaz ClientDetailsModal.css fajla 23](#_Toc156386153)

[12. GitHub link 25](#_Toc156386154)

# Opis aplikacije

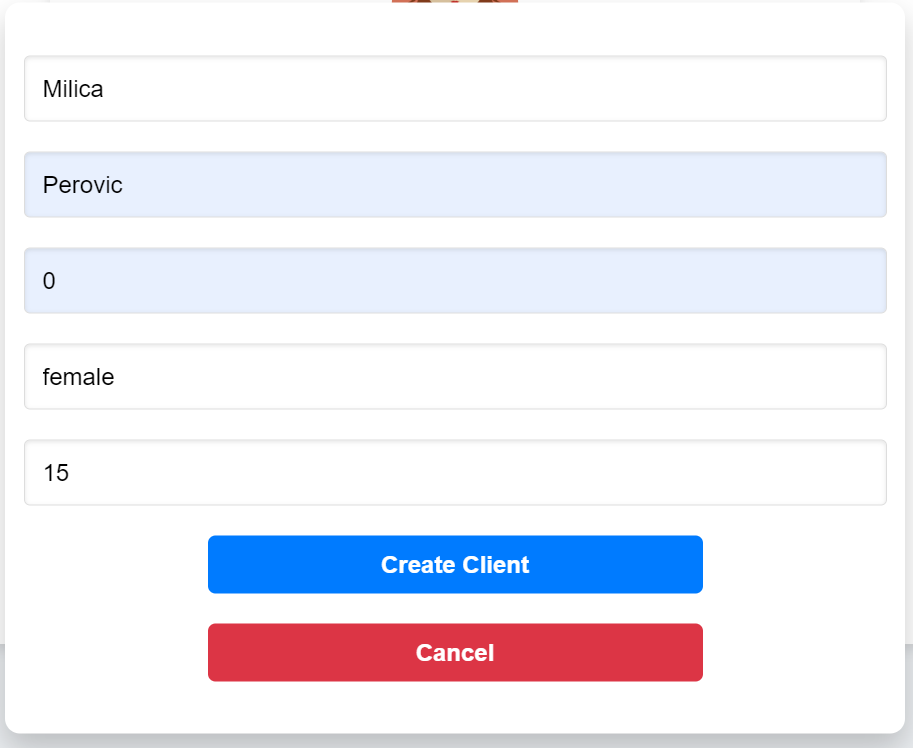
U ovom projektu pravio sam aplikaciju koja radi na principu bankarske aplikacije. S tim, da u mom slučaju klijenti trguju nečim imaginarnim što nije konkretna vrednost na MetaMask wallet-u. Aplikacija se zove Client Dapp.

Izgled aplikacije prikazan je na slici 1. Kada se metamask account poveze sa mojom aplikacijom imamo prikazano ko je ulogovani korisnik. Korinsnik mora da se nalazi u listi korisnika, a ukoliko to nije slučaj onda neće moći da vidi svoj portfolio ni da izvrši transakciju.



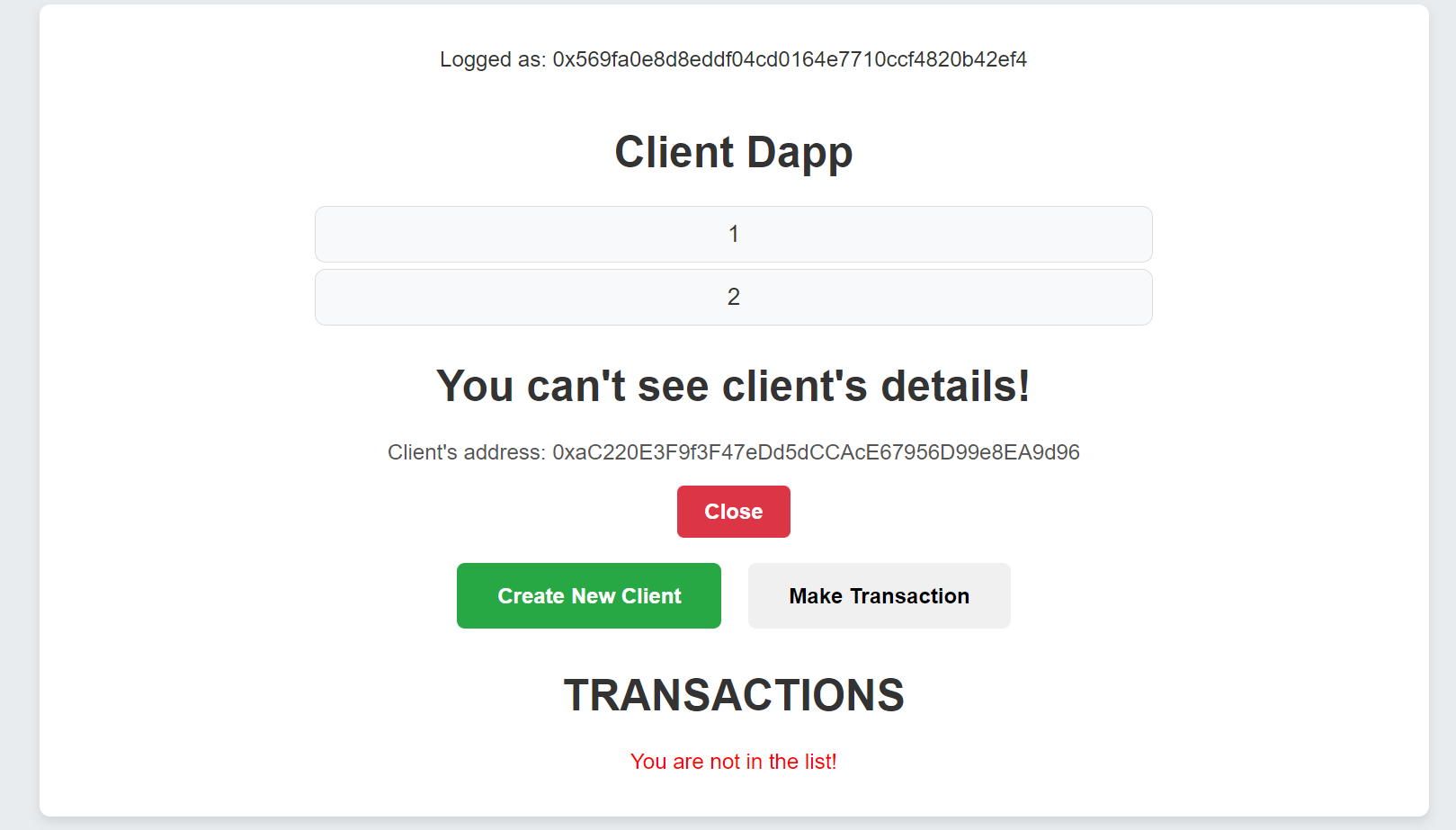
Slika 1 – Početni prikaz aplikacije

Klijent se dodaje u listu klijenata klikom na dugme “Create New Client”. Tom prilikom klijnet mora da popuni formu sa svojim osnovnim podacima, kaon a slici 2.



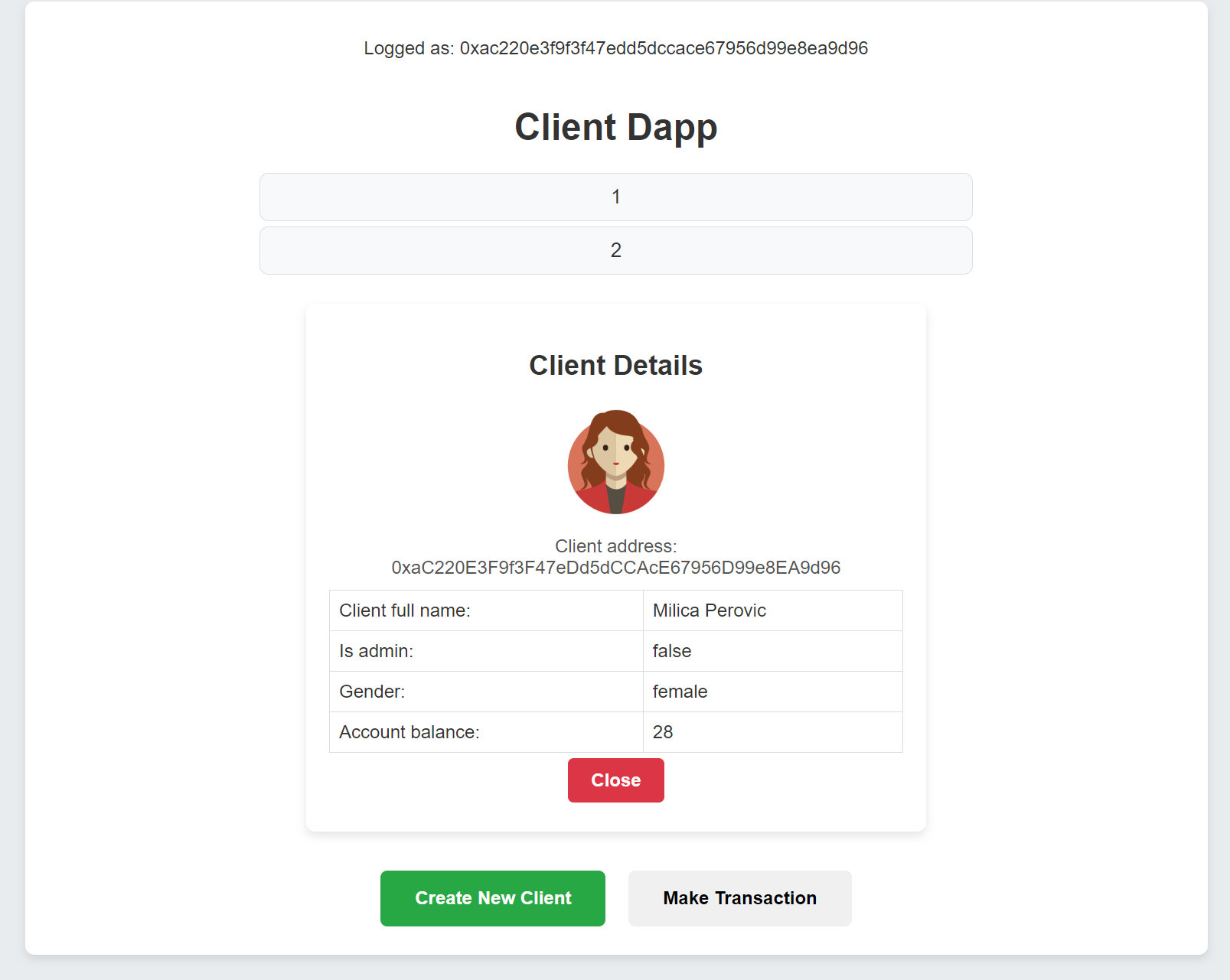
Slika 2 – Prikaz forme za dodavanje klijenta u listu klijenata

Ukoliko klijent nije u listi on nece moci da vidi nista na samoj stranici. To je prikazano na slici 3.



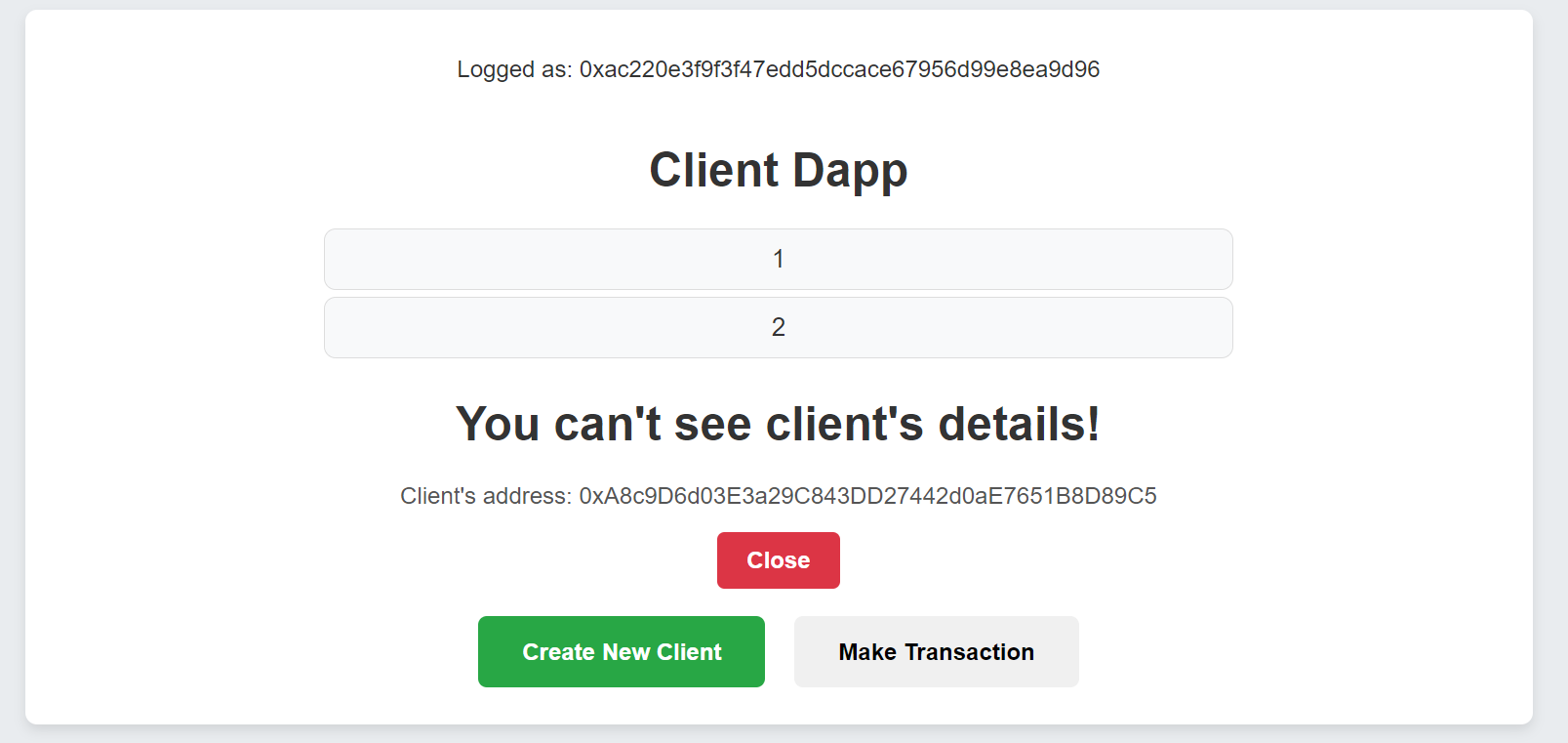
Slika 3 – Prikaz aplikacije ukoliko ulogovani klijent preko MetaMask wallet-a nije u listi klijenata

Ukoliko se ulogovani klijent nalazi u listi on ce moci da vidi svoje podatke klikom na svoj redni broj u listi. To je prikazano na slici 4.



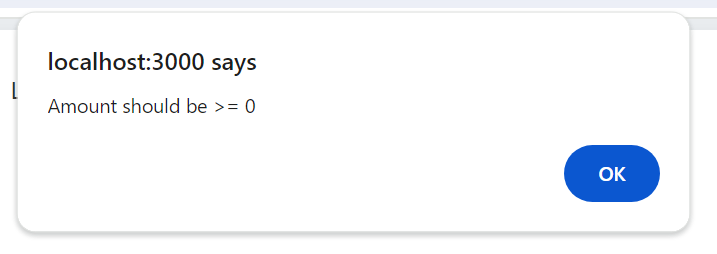
Slika 4 – Prikaz korisnikovog portfolio

Ulogovani korisnik, koji se nalazi u listi, ne moze da vidi portfolio drugih korisnika osim svog. To je prikazano na slici 5.



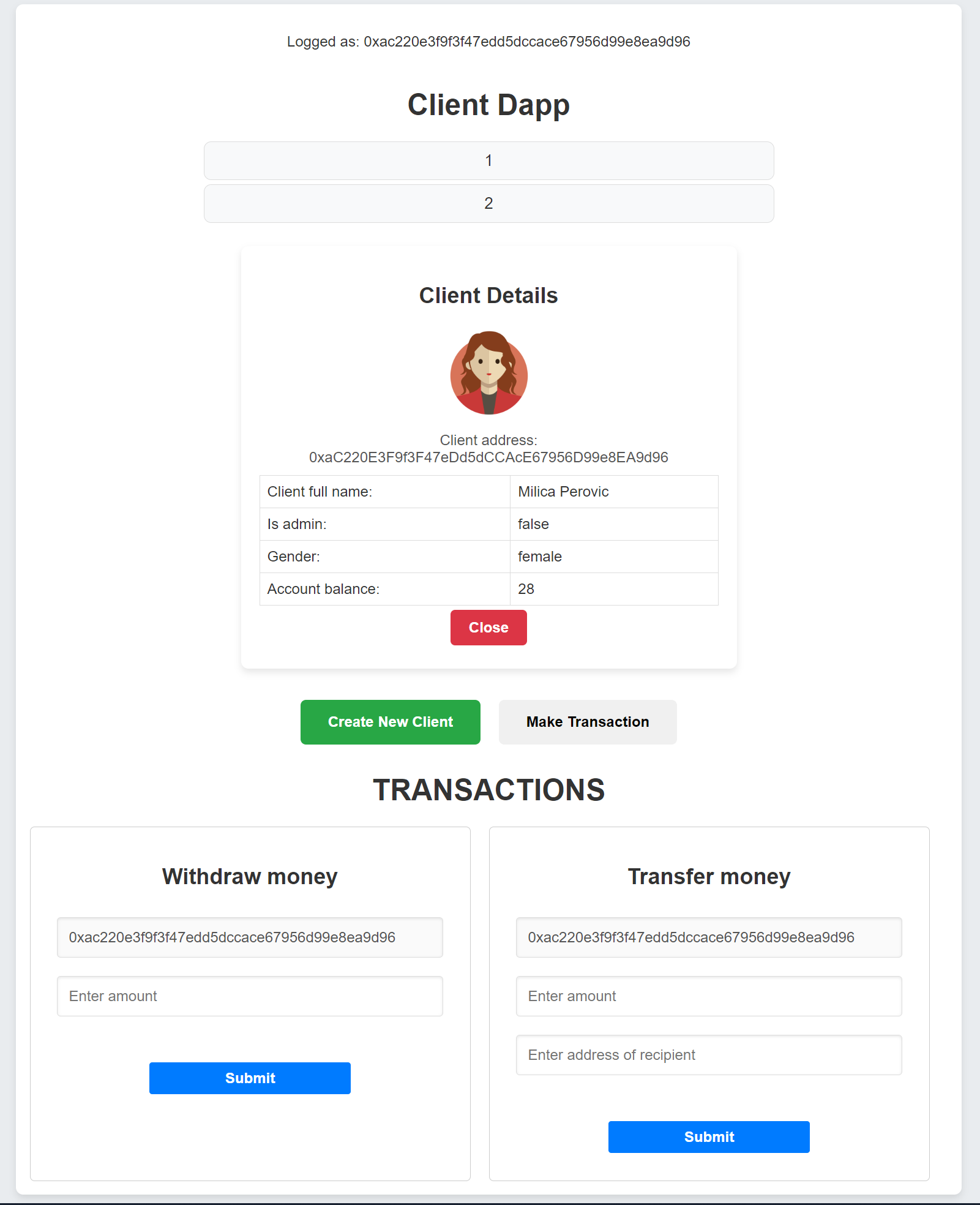
Slika 5 – Prikaz userinterface-a kada korisnik ne moze da vidi tudj portfolio

Ukoliko je korinsik u listi, klikom na dugme “Make Transaction” prikazace mu se komponenta za vrsenje transakcija. To je prikazano na slici 7. Korisnik moze da povlaci sredstva sa svog naloga ili da vrsi transfer svojih sredstava na tudj nalog. Informacije o nalog korisnika koji vrsi transakciju su automatski popunjena. Ostala polja korisnik mora da unese. Ukoliko prilikom unosa korisnik unese 0 za amount ili ne unese nalog primaoca prilikom vrsenja transakcije iskocice pop up sa odgovarajucom porukom kako bi klijent znao sta bi trebalo da izmeni. Pop up sa greskom je prikazan na slici 6.



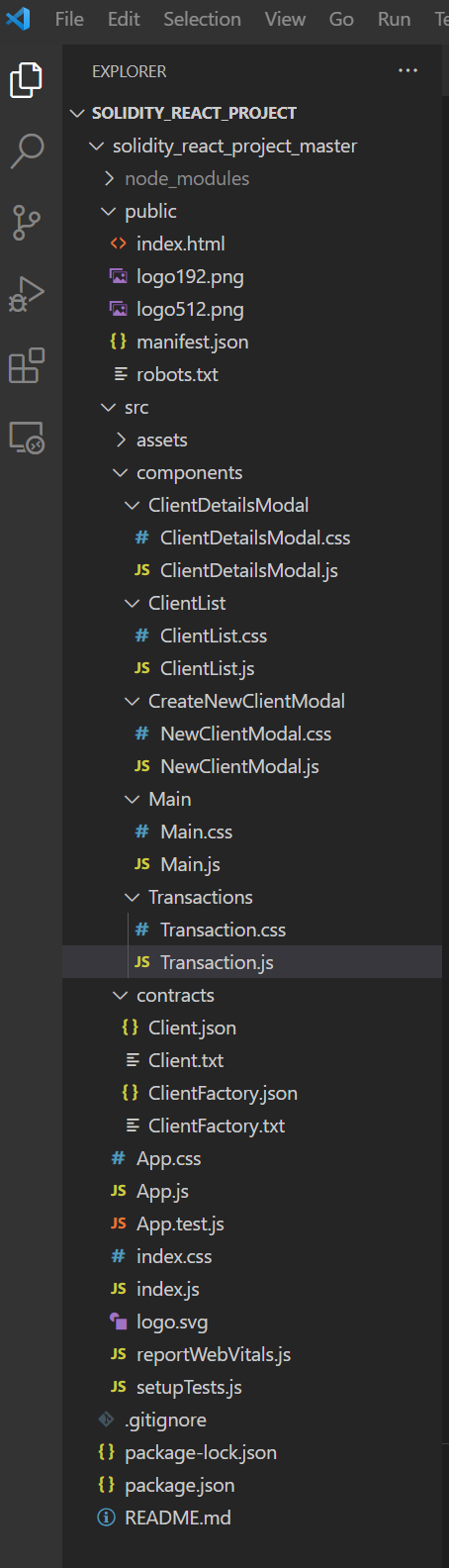
Slika 6 – Pop up sa greskom koja je nastala prilikom neadekvatno popunjene forme za transakciju

Prilikom vršenja svake transakcije, kao i prilikom dodavanja novog klijenta u listu klijentu se skidaju tokeni sa njegovog MetaMask wallet-a. I takodje, svaku transakciju klijent mora da odobri klikom na dugme confirm i prilikom toga mora da ima dovljno tokena na svom nalogu kako bi mogao da izvrsi transakciju.



Slika 7 – Prikaz komponente koja se bavi transakcijama

Ugovori su prvo pisani u Remix-u, pa su potom prebaceni u Visual Studio Code u folder “Contracts” u JSON formatu. Takodje, za svaki ugovor cuva se i njegov kod za Remix u .txt fajlu u istom folderu. Celokupan poziv aplikacije odradjen je u Main.js komponenti, koja potom poziva sve ostale komponente. Prikaz svih komponenti i structure fajlova u Visual Studio Code-u prikazan je na slici 8. Kljucne komponete nalaze se u folderu “Components”, a potom svaka komponenta ima svoj zaseban folder. Tako imamo: ClientDetailsModal, ClientList, CreateNewClientModal, Main, Transactions.



Slika 8 – Prikaz strukture rada/fajlova

# Prikaz Main.js fajla:

import React, { useState, useEffect } from "react";

import ClientList from "../ClientList/ClientList.js";

import NewTransaction from "../Transactions/Transaction.js";

import CreateNewClientModal from "../CreateNewClientModal/NewClientModal.js";

import Web3 from "web3";

import "./Main.css";

const clientFactoryAddress = "0x024794804cAE60F10392a18c29f92be7515f7174";

const sepoliaRPCUrl =

  "https://sepolia.infura.io/v3/67bc1009f5a547cc978659e13579ddf0";

const Main = () => {

  const [showCreateModal, setShowCreateModal] = useState(false);

  const [showTransactions, setShowTransactions] = useState(false);

  const [web3, setWeb3] = useState(null);

  const [account, setAccount] = useState(null);

  const connectWallet = async () => {

    try {

      if (window.ethereum) {

        const accounts = await window.ethereum.request({

          method: "eth\_requestAccounts",

        });

        setAccount(accounts[0]);

        console.log("Connected to Ethereum account: ", accounts[0]);

        console.log("Account ", account);

        console.log(typeof accounts[0]);

        window.ethereum.on("accountsChanged", (newAccounts) => {

          setAccount(newAccounts[0]);

          console.log("Switched to account: ", newAccounts[0]);

        });

      } else {

        console.log("MetaMask is not installed.");

      }

    } catch (error) {

      console.error("Error connecting to MetaMask: ", error);

    }

  };

  useEffect(() => {

    const web3Instance = new Web3(sepoliaRPCUrl);

    console.log(web3Instance);

    setWeb3(web3Instance);

    connectWallet();

    console.log("Web3 instance set up: ", web3);

  }, []);

  return (

    <div className="main-container">

      {!account ? (

        <button className="connect-wallet-button" onClick={connectWallet}>

          Connect with metamask

        </button>

      ) : (

        <p>Logged as: {account}</p>

      )}

      <ClientList

        className="client-list"

        web3={web3}

        account={account}

        clientFactoryAddress={clientFactoryAddress}

      />

      <button

        className="create-new-client-button"

        onClick={() => setShowCreateModal(true)}

      >

        Create New Client

      </button>

      {showCreateModal && (

        <CreateNewClientModal

          className="create-new-client-modal"

          web3={web3}

          account={account}

          onClose={() => setShowCreateModal(false)}

          clientFactoryAddress={clientFactoryAddress}

        />

      )}

      <button

        className="show-transactions-button"

        onClick={() => setShowTransactions(true)}

      >

        {" "}

        Make Transaction

      </button>

      {showTransactions && (

        <NewTransaction

          web3={web3}

          account={account}

          clientFactoryAddress={clientFactoryAddress}

        />

      )}

    </div>

  );

};

export default Main;

# Prikaz Main.css fajla

1. body {
2. font-family: "Arial", sans-serif;
3. background-color: #e9ecef;
4. margin: 0;
5. padding: 0;
6. color: #333;
7. }
8. .main-container {
9. max-width: 1000px;
10. margin: 10px auto;
11. padding: 15px;
12. text-align: center;
13. background-color: white;
14. border-radius: 8px;
15. box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
16. }
17. button {
18. padding: 15px 30px;
19. border: none;
20. border-radius: 6px;
21. cursor: pointer;
22. font-weight: bold;
23. font-size: 1rem;
24. transition: background-color 0.3s ease, transform 0.2s ease;
25. margin: 10px;
26. }
27. .connect-wallet-button {
28. background-color: #007bff;
29. color: white;
30. }
31. .connect-wallet-button:hover {
32. background-color: #0056b3;
33. transform: scale(1.05);
34. }
35. .create-new-client-button {
36. background-color: #28a745;
37. color: white;
38. }
39. .create-new-client-button:hover {
40. background-color: #218838;
41. transform: scale(1.05);
42. }
43. .create-new-client-modal {
44. position: fixed;
45. top: 50%;
46. left: 50%;
47. transform: translate(-50%, -50%);
48. background-color: white;
49. padding: 25px;
50. border-radius: 10px;
51. box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);
52. z-index: 1000;
53. width: 90%;
54. max-width: 550px;
55. min-height: 350px;
56. }

# Prikaz NewClientModal.js

import React, { useState } from "react";

import ClientFactoryABI from "../../contracts/ClientFactory.json";

import "./NewClientModal.css";

const NewClientModal = ({ onClose, web3, account, clientFactoryAddress }) => {

  const [clientData, setClientData] = useState({

    firstName: "",

    lastName: "",

    admin: "",

    gender: "",

    accountBalance: "",

  });

  const handleChange = (e) => {

    setClientData({ ...clientData, [e.target.name]: e.target.value });

  };

  const handleSubmit = async () => {

    if (typeof window.ethereum === "undefined" || !window.ethereum.isMetaMask) {

      console.log("MetaMask is not installed or not connected!");

      return;

    }

    if (!web3 || !account) {

      alert("Web3 instance or account is not available.");

      return;

    }

    try {

      const clientFactory = new web3.eth.Contract(

        ClientFactoryABI.abi,

        clientFactoryAddress

      );

      const transactionParameters = {

        to: clientFactoryAddress,

        from: account, // must match user's active address

        data: clientFactory.methods

          .createNewClient(

            clientData.firstName,

            clientData.lastName,

            clientData.admin,

            clientData.gender,

            clientData.accountBalance

          )

          .encodeABI(),

      }; // call to contract method

      // txHash is a hex string

      const txHash = await window.ethereum.request({

        method: "eth\_sendTransaction",

        params: [transactionParameters],

      });

      console.log("Transaction Hash:", txHash);

      onClose();

    } catch (error) {

      console.error("Error sending transaction:", error);

    }

  };

  return (

    <div className="create-new-client-modal">

      <div className="modal-content">

        <input

          className="modal-input"

          name="firstName"

          placeholder="first name"

          onChange={handleChange}

        />

        <input

          className="modal-input"

          name="lastName"

          placeholder="last name"

          onChange={handleChange}

        />

        <input

          className="modal-input"

          name="admin"

          placeholder="admin"

          onChange={handleChange}

        />

        <input

          className="modal-input"

          name="gender"

          placeholder="gender"

          onChange={handleChange}

        />

        <input

          className="modal-input"

          name="accountBalance"

          placeholder="account balance"

          onChange={handleChange}

        />

        <button className="modal-button" onClick={handleSubmit}>

          Create Client

        </button>

        <button className="modal-button cancel-button" onClick={onClose}>

          Cancel

        </button>

      </div>

    </div>

  );

};

export default NewClientModal;

# Prikaz NewClientModal.css fajla

1. .create-new-client-modal {
2. position: fixed;
3. top: 50%;
4. left: 50%;
5. transform: translate(-50%, -50%);
6. background-color: #fff;
7. padding: 30px;
8. border-radius: 8px;
9. box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
10. width: 100%;
11. max-width: 500px;
12. text-align: center;
13. z-index: 1000;
14. }
15. .modal-content {
16. display: flex;
17. flex-direction: column;
18. align-items: center;
19. }
20. .modal-input {
21. width: 100%;
22. padding: 12px;
23. margin: 10px 0;
24. border-radius: 4px;
25. border: 1px solid #ddd;
26. box-shadow: inset 0 1px 3px rgba(0, 0, 0, 0.1);
27. font-size: 1rem;
28. }
29. .modal-button {
30. width: 60%;
31. padding: 10px;
32. border: none;
33. border-radius: 5px;
34. cursor: pointer;
35. font-weight: bold;
36. margin-top: 10px;
37. transition: background-color 0.3s, transform 0.2s;
38. background-color: #007bff;
39. color: white;
40. }
41. .modal-button:hover {
42. background-color: #0056b3;
43. transform: scale(1.05);
44. }
45. .cancel-button {
46. background-color: #dc3545;
47. }
48. .cancel-button:hover {
49. background-color: #c82333;
50. }

# Prikaz Transaction.js fajla

1. import React, { useState, useEffect } from "react";
2. import ClientFactoryABI from "../../contracts/ClientFactory.json";
3. import "./Transaction.css";
4. const Transaction = ({ web3, account, clientFactoryAddress }) => {
5. const [accountFound, setAccountFound] = useState(false);
6. const [withdrawAmount, setWithdrawAmount] = useState(0);
7. const [transferData, setTransferData] = useState({
8. amount: 0,
9. recipientAddress: "",
10. });
11. const handleChangeWithdraw = (e) => {
12. // Update the inputValue state when the input value changes
13. setWithdrawAmount(e.target.value);
14. };
15. const handleSubmitWithdrawalAmount = async () => {
16. if (transferData.amount == 0) {
17. alert("Amount should be >= 0");
18. } else {
19. if (
20. typeof window.ethereum === "undefined" ||
21. !window.ethereum.isMetaMask
22. ) {
23. console.log("MetaMask is not installed or not connected!");
24. return;
25. }
26. if (!web3 || !account) {
27. alert("Web3 instance or account is not available.");
28. return;
29. }
30. try {
31. const clientFactory = new web3.eth.Contract(
32. ClientFactoryABI.abi,
33. clientFactoryAddress
34. );
35. const transactionParameters = {
36. to: clientFactoryAddress,
37. from: account, // must match user's active address
38. data: clientFactory.methods.withdrawMoney(withdrawAmount).encodeABI(),
39. }; // call to contract method
40. // txHash is a hex string
41. const txHash = await window.ethereum.request({
42. method: "eth\_sendTransaction",
43. params: [transactionParameters],
44. });
45. console.log("Transaction Hash:", txHash);
46. const currentBalance = await clientFactory.methods
47. .showClientBalance()
48. .call({ from: account });
49. console.log(currentBalance);
50. } catch (error) {
51. console.error(
52. "Error while trying to execute withdrawal function:",
53. error
54. );
55. }
56. }
57. };
58. const handleChangeTransfer = (e) => {
59. // Update the inputValue state when the input value changes
60. setTransferData({ ...transferData, [e.target.name]: e.target.value });
61. };
62. const handleSubmitTransferAmount = async () => {
63. if (transferData.amount == 0) {
64. alert("Amount should be >= 0");
65. } else if (transferData.recipientAddress == "") {
66. alert("Recipient address should be populated!");
67. } else {
68. if (
69. typeof window.ethereum === "undefined" ||
70. !window.ethereum.isMetaMask
71. ) {
72. console.log("MetaMask is not installed or not connected!");
73. return;
74. }
75. if (!web3 || !account) {
76. alert("Web3 instance or account is not available.");
77. return;
78. }
79. try {
80. const clientFactory = new web3.eth.Contract(
81. ClientFactoryABI.abi,
82. clientFactoryAddress
83. );
84. const transactionParameters = {
85. to: clientFactoryAddress,
86. from: account, // must match user's active address
87. data: clientFactory.methods
88. .transferMoney(transferData.amount, transferData.recipientAddress)
89. .encodeABI(),
90. }; // call to contract method
91. // txHash is a hex string
92. const txHash = await window.ethereum.request({
93. method: "eth\_sendTransaction",
94. params: [transactionParameters],
95. });
96. console.log("Transaction Hash:", txHash);
97. } catch (error) {
98. console.error(
99. "Error while trying to execute withdrawal function:",
100. error
101. );
102. }
103. }
104. };
105. const loadClients = async () => {
106. try {
107. const clientFactory = new web3.eth.Contract(
108. ClientFactoryABI.abi,
109. clientFactoryAddress
110. );
111. const accounts = await window.ethereum.request({
112. method: "eth\_requestAccounts",
113. });
114. console.log(accounts[0]);
115. const existAccount = await clientFactory.methods
116. .doesClientExistInTheList()
117. .call({ from: accounts[0] });
118. setAccountFound(existAccount);
119. } catch (error) {
120. console.error("Error while loading client list:", error);
121. }
122. };
123. useEffect(() => {
124. if (web3) {
125. loadClients();
126. }
127. }, [web3]);
128. // loadClients();
129. return (
130. <div>
131. <h1>TRANSACTIONS</h1>
132. {!accountFound ? (
133. <p style={{ color: "red" }}>You are not in the list!</p>
134. ) : (
135. <div className="form-container">
136. <form
137. onSubmit={handleSubmitWithdrawalAmount}
138. className="form-withdraw"
139. >
140. <h2>Withdraw money</h2>
141. <input id="address" name="address" value={account} disabled />
142. <input
143. id="amount"
144. name="amount"
145. onChange={handleChangeWithdraw}
146. placeholder="Enter amount"
147. />
148. <button className="withdraw\_money" type="submit">
149. Submit
150. </button>
151. </form>
152. <form onSubmit={handleSubmitTransferAmount} className="form-transfer">
153. <h2>Transfer money</h2>
154. <input
155. type="addressSender"
156. id="addressSender"
157. name="addressSender"
158. value={account}
159. disabled
160. onChange={handleChangeTransfer}
161. />
162. <input
163. id="amountTranfer"
164. name="amount"
165. placeholder="Enter amount"
166. onChange={handleChangeTransfer}
167. />
168. <input
169. id="recipientAddress"
170. name="recipientAddress"
171. placeholder="Enter address of recipient"
172. onChange={handleChangeTransfer}
173. />
174. <button className="transfer\_money" type="submit">
175. Submit
176. </button>
177. </form>
178. </div>
179. )}
180. </div>
181. );
182. };
183. export default Transaction;

# Prikaz Transaction.css fajla

1. /\* Style for forms container \*/
2. .form-container {
3. display: flex;
4. }
5. /\* Style for individual forms \*/
6. .form-transfer,
7. .form-withdraw {
8. flex: 1;
9. margin-right: 20px; /\* Adjust spacing between forms \*/
10. padding: 20px;
11. border: 1px solid #ccc;
12. border-radius: 5px;
13. }
14. /\* Style for form inputs \*/
15. input {
16. width: 90%;
17. padding: 12px;
18. margin: 10px 0;
19. border-radius: 4px;
20. border: 1px solid #ddd;
21. box-shadow: inset 0 1px 3px rgba(0, 0, 0, 0.1);
22. font-size: 1rem;
23. }
24. .transfer\_money,
25. .withdraw\_money {
26. background-color: #007bff;
27. color: white;
28. border: none;
29. border-radius: 3px;
30. cursor: pointer;
31. width: 50%;
32. padding: 8px;
33. margin-bottom: 10px;
34. margin-top: 40px;
35. }
36. .transfer\_money:hover {
37. background-color: #0056b3;
38. }
39. .withdraw\_money:hover {
40. background-color: #0056b3;
41. }

# Prikaz ClientList.js fajla

1. import React, { useState, useEffect } from "react";
2. import ClientFactoryABI from "../../contracts/ClientFactory.json";
3. import ClientDetailsModal from "../ClientDetailsModal/ClientDetailsModal.js";
4. import "./ClientList.css";
5. const ClientList = ({ web3, account, clientFactoryAddress }) => {
6. const [clients, setClients] = useState([]);
7. const [selectedClient, setSelectedClient] = useState(null);
8. const loadClients = async () => {
9. try {
10. const clientFactory = new web3.eth.Contract(
11. ClientFactoryABI.abi,
12. clientFactoryAddress
13. );
14. const clientsFromContract = await clientFactory.methods
15. .getAllClients()
16. .call();
17. setClients(clientsFromContract);
18. } catch (error) {
19. console.error("Error while loading client list:", error);
20. }
21. };
22. useEffect(() => {
23. if (web3) {
24. loadClients();
25. }
26. }, [web3]);
27. const openDetailsModal = (client) => {
28. setSelectedClient(client);
29. };
30. return (
31. <div className="client-list">
32. <h1 className="client-list-title">Client Dapp</h1>
33. {clients.map((client, index) => (
34. <div
35. key={index}
36. className="client-item"
37. onClick={() => openDetailsModal(client)}
38. >
39. {index + 1}
40. </div>
41. ))}
42. {selectedClient && (
43. <ClientDetailsModal
44. web3={web3}
45. account={account}
46. client={selectedClient}
47. onClose={() => setSelectedClient(null)}
48. />
49. )}
50. </div>
51. );
52. };
53. export default ClientList;

# Prikaz ClientList.css fajla

1. .client-list {
2. padding: 4px;
3. display: flex;
4. flex-direction: column;
5. align-items: center;
6. }
7. .client-item {
8. background-color: #f8f9fa;
9. border: 1px solid #ddd;
10. border-radius: 8px;
11. padding: 10px 10px;
12. margin-bottom: 5px;
13. cursor: pointer;
14. transition: background-color 0.3s ease;
15. width: 80%;
16. max-width: 600px;
17. text-align: center;
18. font-size: 1.1rem;
19. color: #333;
20. }
21. .client-item:hover {
22. background-color: #e2e6ea;
23. }

# Prikaz ClientDetailsModal.js fajla

1. import React, { useState, useEffect } from "react";
2. import ClientABI from "../../contracts/Client.json";
3. import "./ClientDetailsModal.css";
4. import manPicture from "../../assets/man.png";
5. import femalePicture from "../../assets/female.png";
6. const ClientDetailsModal = ({ client, onClose, web3, account }) => {
7. const [firstName, setFirstName] = useState("");
8. const [lastName, setLastName] = useState("");
9. const [clientAccount, setClientAccount] = useState(account);
10. const [admin, setAdmin] = useState("");
11. const [gender, setGender] = useState("");
12. const [accountBalance, setAccountBalance] = useState("");
13. const loadClientDetails = async () => {
14. if (web3 && client) {
15. const clientContract = new web3.eth.Contract(ClientABI.abi, client);
16. const clientAccount = await clientContract.methods.getAccount().call();
17. const firstName = await clientContract.methods.getFirstName().call();
18. const lastName = await clientContract.methods.getLastName().call();
19. const admin = await clientContract.methods.isAdmin().call();
20. const gender = await clientContract.methods.getGender().call();
21. const accountBalance = await clientContract.methods
22. .getAccountBalance()
23. .call();
24. setFirstName(firstName);
25. setLastName(lastName);
26. setClientAccount(clientAccount);
27. if (admin) {
28. setAdmin("true");
29. } else {
30. setAdmin("false");
31. }
32. setGender(gender);
33. setAccountBalance(accountBalance + "");
34. }
35. };
36. useEffect(() => {
37. loadClientDetails();
38. }, [web3, client]);
39. return (
40. <div>
41. {clientAccount.toLowerCase() == account ? (
42. <div className="client-details-modal">
43. <h2 className="modal-title">Client Details</h2>
44. <img src={gender === "male" ? manPicture : femalePicture} alt="" />
45. <p className="client-info">Client address: {clientAccount}</p>
46. <table>
47. <tbody>
48. <tr>
49. <td>Client full name:</td>
50. <td>{firstName + " " + lastName}</td>
51. </tr>
52. <tr>
53. <td>Is admin:</td>
54. <td>{admin}</td>
55. </tr>
56. <tr>
57. <td>Gender:</td>
58. <td>{gender}</td>
59. </tr>
60. <tr>
61. <td>Account balance:</td>
62. <td>{accountBalance}</td>
63. </tr>
64. </tbody>
65. </table>
66. <button className="close-button" onClick={onClose}>
67. Close
68. </button>
69. </div>
70. ) : (
71. <div>
72. <h1>You can't see client's details!</h1>
73. <p className="client-info">Client's address: {clientAccount}</p>
74. <button className="close-button" onClick={onClose}>
75. Close
76. </button>
77. </div>
78. )}
79. </div>
80. );
81. };
82. export default ClientDetailsModal;

# Prikaz ClientDetailsModal.css fajla

1. .client-details-modal {
2. background-color: #fff;
3. padding: 20px;
4. border-radius: 8px;
5. box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
6. width: 100%;
7. max-width: 500px;
8. margin: 20px auto;
9. text-align: center;
10. }
11. table {
12. font-family: arial, sans-serif;
13. border-collapse: collapse;
14. width: 100%;
15. }
16. td,
17. th {
18. border: 1px solid #dddddd;
19. text-align: left;
20. padding: 8px;
21. }
22. img {
23. height: 100px;
24. width: 100px;
25. }
26. .modal-title {
27. font-size: 1.5rem;
28. color: #333;
29. margin-bottom: 20px;
30. }
31. .client-info {
32. color: #555;
33. margin: 10px 0;
34. }
35. .close-button {
36. padding: 10px 20px;
37. border: none;
38. border-radius: 5px;
39. cursor: pointer;
40. font-weight: bold;
41. margin: 5px;
42. transition: background-color 0.3s ease;
43. }
44. .close-button {
45. background-color: #dc3545;
46. color: white;
47. }
48. .close-button:hover {
49. background-color: #c82333;
50. }

# GitHub link

<https://github.com/MarkoBojanic18/solidity_react_project_master.git>