Cost Tracking Application

Solidity Blockchain Dev

GitHub projekta: <https://github.com/MarkoBojanic18/solidity_bootcamp_domaci.git>

|  |  |  |
| --- | --- | --- |
| Name | Email | Github |
| Marko Bojanić | markobojanic@gmail.com | MarkoBojanic18 |
| Jovana Tajković | jovana.trajkovic.best@gmail.com | jovanatrajkovic |
| Aleksandar Matović | Aleksandarmatovic49@gmail.com | Robiko55 |

Content

[1 Application Description 3](#_Toc161393413)

[2 Implementation 3](#_Toc161393414)

[2.1 Backend 3](#_Toc161393415)

[2.2 Frontend 3](#_Toc161393416)

[3 User Manual 3](#_Toc161393417)

# Application Description

Osnovni zadatak ove aplikacije jeste pracenje troskova korisnika. Da bi korisnik mogao da cuva pomenute podatke mora da se nalazi u listi korisnika. Prijavljivanje korisnika u listu korisnika vrsi se pomocu adrese MetaMask novcanika. Na taj nacin obezbedjuje se da za jednu adresu moze biti kreiran samo jedan korisnik. Korisnik moze da pikazuje sve svoje troskove, moze da prikaze odredjene troskove, moze da prikaze ukupnu sumu troskova, moze da iskljuci trosak koji zeli da izbaci iz racunice, troskovi mogu da se sortiraju u opadajucem ili rastucem smeru po svakoj koloni.

# Implementation

## Backend

Backend kod je pisan u Remix okruzenju tj. Koriscena je Ethereum platforma za razvoj pamentih ugovora. Celokupna logika je podeljena u dva kontrakta: User.sol i UserFactory.sol. Kontrakt User kao promenljive sadzi samo adresu korisnika aplikacije i listu njegovih troskova. Troskovi su napravljeni kao struktura i svaki trosak ima svoj jedinstveni identifikacioni broj koji omogucava laksu manipulaciju troskovima i pretragu. Takodje kontrakt user ima konstruktor koji sluzi za kreiranje korisnika aplikacije.

Od funkcija contract User sadrzi:

1. getUserAddress() -> get funkcija koja vraca adresu meta mask novcanika korisnika
2. getExpenses() -> funckija koja vraca niz troskova korisnika
3. addExpense() -> funkcija koja kao parametre prima parametre koji su neophodni za pravljenje Troska i dodaje Trosak u listu troskova
4. changeExpense() -> funkcija koja prolazi kroz listu troskova korisnika i pronalazi odredjeni trosak na osnovu njegovog jedinstvenog broja za tog korisnika i menja podatke koje je korisnik izmenio
5. getExpensesByCategory() -> funkcija koja prolazi kroz listu troskova korisnika i vraca samo troskove koji pripadaju kategoriji troskova koju je korisnik odabrao

**Contract User.sol :**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.4;

contract User{

    address private userAddress;

    uint256 \_id = 0;

    Expenses[] private expenses;

    struct Expenses{

        uint256 id;

        string category;

        uint amount;

        uint date;

        string description;

        bool canceled;

    }

    constructor(address \_userAddress){

        userAddress = \_userAddress;

    }

    function getUserAddress() public view returns(address){

        return userAddress;

    }

    function getExpenses() public view returns(Expenses[] memory){

        return expenses;

    }

    function addExpense(string memory \_category, uint \_amount, string memory \_description) public {

        uint \_date = block.timestamp;

        \_id++;

         Expenses memory expense = Expenses(\_id, \_category,\_amount,\_date,\_description,false);

         expenses.push(expense);

    }

    function changeExpense(uint256 \_idn,string memory \_category,uint256 \_amount, string memory \_description, bool \_canceled) public {

        uint256 arrayLength = expenses.length;

        for (uint256 i = 0; i < arrayLength; i++) {

        if (expenses[i].id == \_idn) {

            expenses[i].category = \_category;

            expenses[i].canceled = \_canceled;

            expenses[i].amount = \_amount;

            expenses[i].description = \_description;

        }

    }

    }

    function getExpensesByCategory(string memory \_category) public view returns(Expenses[] memory){

       uint256 arrayLength = expenses.length;

    Expenses[] memory categoryExpenses = new Expenses[](arrayLength);

    uint256 count = 0;

    for (uint256 i = 0; i < arrayLength; i++) {

        if (keccak256(bytes(expenses[i].category)) == keccak256(bytes(\_category))) {

            // If the category matches, add the expense to the result array

            categoryExpenses[count] = expenses[i];

            count++;

        }

    }

    // Resize the array to remove any uninitialized elements

    assembly {

        mstore(categoryExpenses, count)

    }

    return categoryExpenses;

    }

}

**Contract UserFactory.sol**

U okviru ovog kontrakta implementiramo funkcionalnosti iz kontrakta User.sol. Takodje, ovde pravimo i cuvamo listu svih korisnika nase aplikacije (users).   
  
Od funkcija contract UserFactory sadrzi:

1. createUser() -> funkcija koja je odgovorna za dodavanje korisnika u niz korisnika. Ona najpre poziva private funkciju checkNewUser() koja proverava da li korisnik sa datom adresom meta mask novcanika vec postoji u nizu i u zavisnosti od toga vraca true ili false. Ukoliko korisnik postoji u listi (true) korisnik nece biti dodat u niz, a ukoliko ne postoji u listi (false) bice kreiran novi korisnik i daodat u listu.
2. checkNewUser() -> privatna funkcija koja sluzi samo za upotrebu od strane drugih funkcija u UserFactory kontraktu. Ova funkcija kao ulazni parametar prima objekat User i prolazi kroz niz svih korisnika i proverava da li korisnik postoji u listi na osnovu adrese meta mask novcanika korisnika. Ukoliko postoji ona vraca true, a ukoliko korinsik ne postoji u nizu onda vraca false.
3. checkIfUserExistInTheList() -> funkcija koja za okidaca funkcije proverava da li se on nalazi u nizu korisnika i vraca true ili false u zavisnoti od toga.
4. createExpense() -> funkcija koja na osnovu prosledjenih parametara kreira novi trosak za korisnika koji poziva funkciju za unos troska.
5. changeExpenseForUser() -> funkcija koja za datog user-a poziva funckiju za promenu informacija odredjenog troska
6. getUserData() -> funkcija koja pronalazi kroz niz korisnika i ukoliko se korisnik koji poziva funkciju nalazi u listi korisnika, ona vraca instancu ka User objektu.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.4;

import "./User.sol";

contract UserFactory{

    User[] private users;

    function createUser() public {

        User newUser = new User(msg.sender);

        if(!checkNewUser(newUser)){

            users.push(newUser);

        }

    }

     function checkNewUser(User newUser)private view returns(bool){

        uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == newUser.getUserAddress()){

                return true;

            }

        }

        return false;

    }

    function checkIfUserExistInTheList()public view returns(bool){

        uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == msg.sender){

                return true;

            }

        }

        return false;

    }

    function createExpense(string memory \_category, uint \_amount, string memory \_description) public {

         uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == msg.sender){

                users[i].addExpense(\_category,\_amount,\_description);

            }

        }

    }

    function changeExpenseForUser(uint256 \_idn,string memory \_category,uint256 \_amount, string memory \_description, bool \_canceled) public {

        uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == msg.sender){

                users[i].changeExpense(\_idn,\_category,\_amount,\_description,\_canceled);

            }

        }

    }

    function getUserData() public view returns(User){

         uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == msg.sender){

                return users[i];

            }

        }

        return User(address(0));

    }

}

## Frontend

Povezivanje korisnika sa Meta Mask novcanikom vrsi se u Main.js komponenti u asinhronoj funkciji connectWallet(). Ukoliko korisnik nije povezan sa Meta Mask novcanikom onda ce mu se prikazati dugme connect with metamask. Ukoliko je korisnik povezan sa MetaMask novcanikom prikazace mu se njegova adresa tj. Adresa ulogovanog korisnika.  
  
**Prikaz Main.js komponente:**

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.4;

import "./User.sol";

contract UserFactory{

    User[] private users;

    function createUser() public {

        User newUser = new User(msg.sender);

        if(!checkNewUser(newUser)){

            users.push(newUser);

        }

    }

     function checkNewUser(User newUser)private view returns(bool){

        uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == newUser.getUserAddress()){

                return true;

            }

        }

        return false;

    }

    function checkIfUserExistInTheList()public view returns(bool){

        uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == msg.sender){

                return true;

            }

        }

        return false;

    }

    function createExpense(string memory \_category, uint \_amount, string memory \_description) public {

         uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == msg.sender){

                users[i].addExpense(\_category,\_amount,\_description);

            }

        }

    }

    function changeExpenseForUser(uint256 \_idn,string memory \_category,uint256 \_amount, string memory \_description, bool \_canceled) public {

        uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == msg.sender){

                users[i].changeExpense(\_idn,\_category,\_amount,\_description,\_canceled);

            }

        }

    }

    function getUserData() public view returns(User){

         uint arrayLength = users.length;

        for(uint i = 0; i < arrayLength; i++){

            if(users[i].getUserAddress() == msg.sender){

                return users[i];

            }

        }

        return User(address(0));

    }

}

Komponenta ExpensesList sluzi da poziva odgovarajuce komponte u zavisnosti da li je korisnik kliknuo da zeli da vidi sve troskove ili da vidi samo troskove izlistane po kategorijama. Takodje ova komonenta sprecava korisnika koji nije ulogovan da vidi dugme za dodavanje troskova, za prikaz svih troskova i dugme za pretragu troskova po kategorijama. Umesto toga ispisuje korisniku poruku „ You are not in the list of users, please make account!“  
  
**Prikaz ExpenseList.js komponente:**

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

import UserFactoryABI from "../../contracts/UserFactory.json";

import "./ExpensesList.css";

import CreateNewExpenseModal from "../CreateNewExpenseModal/CreateNewExpenseModal.js";

import AllExpenses from "../AllExpenses/AllExpenses.js";

import ExpensesByCategory from "../ExpensesByCategory/ExpensesByCategory.js";

const ExpensesList = ({ web3, account, userFactoryAddress }) => {

  const [user, setUser] = useState("");

  const [userFound, setUserFound] = useState(false);

  const [showCreateExpenseModal, setShowCreateExpenseModal] = useState(false);

  const [showAllExpenses, setShowAllExpenses] = useState(false);

  const [showExpensesByCategory, setShowExpensesByCategory] = useState(false);

  // const loadUserExpenses = async () => {

  //   try {

  //     const clientFactory = new web3.eth.Contract(

  //       ClientFactoryABI.abi,

  //       clientFactoryAddress

  //     );

  //     const clientsFromContract = await clientFactory.methods

  //       .getAllClients()

  //       .call();

  //     setClients(clientsFromContract);

  //   } catch (error) {

  //     console.error("Error while loading client list:", error);

  //   }

  // };

  function openComponent(id) {

    if (id == 1) {

      setShowAllExpenses(true);

      setShowExpensesByCategory(false);

    } else if (id == 2) {

      setShowAllExpenses(false);

      setShowExpensesByCategory(true);

    }

  }

  const loadUser = async () => {

    try {

      const userFactory = new web3.eth.Contract(

        UserFactoryABI.abi,

        userFactoryAddress

      );

      console.log("SVE JE UREDU");

      console.log({ account });

      const userFromContract = await userFactory.methods

        .getUserData()

        .call({ from: account });

      console.log("User from contract is: ", userFromContract);

      if (userFromContract != "0x0000000000000000000000000000000000000000") {

        setUserFound(true);

        setUser(userFromContract);

      }

    } catch (error) {

      console.error(

        "Error while checking client in order to show expenses view:",

        error

      );

    }

  };

  useEffect(() => {

    if (web3) {

      loadUser();

    }

  }, [web3]);

  // const openDetailsModal = (client) => {

  //   setSelectedClient(client);

  // };

  return (

    <div className="client-list">

      <h1 className="client-list-title">EXPENSES</h1>

      {!userFound ? (

        <p style={{ color: "red" }}>

          You are not in the list of users, please make account!

        </p>

      ) : (

        <div>

          <button

            className="create-new-client-button"

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

          >

            Create New Expense

          </button>

          <button

            className="create-new-client-button"

            onClick={() => openComponent(1)}

          >

            Show All Expenses

          </button>

          <button

            className="create-new-client-button"

            onClick={() => openComponent(2)}

          >

            Search Expenses By Category

          </button>

          {showCreateExpenseModal && account && (

            <CreateNewExpenseModal

              className="create-new-client-modal"

              web3={web3}

              account={account}

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

              userFactoryAddress={userFactoryAddress}

            />

          )}

          {showAllExpenses && (

            <AllExpenses

              web3={web3}

              account={account}

              userFactoryAddress={userFactoryAddress}

              user={user}

            />

          )}

          {showExpensesByCategory && (

            <ExpensesByCategory

              web3={web3}

              account={account}

              userFactoryAddress={userFactoryAddress}

              user={user}

            />

          )}

        </div>

      )}

      {/\* {clients.map((client, index) => (

        <div

          key={index}

          className="client-item"

          onClick={() => openDetailsModal(client)}

        >

          {index + 1}

        </div>

      ))}

      {selectedClient && (

        <ClientDetailsModal

          web3={web3}

          account={account}

          client={selectedClient}

          onClose={() => setSelectedClient(null)}

        />

      )} \*/}

    </div>

  );

};

export default ExpensesList;

Komponenta ExpensesByCategory sluzi za prikaz troskova po kategoriji koju je korisnik izabrao. Postoje cetiri osnovne kategorije koje korisnik moze da bira. Takdoje tu se prikazuje i ukupna suma troskova. Ovde se ne prikazuju troskovi koji su oznaceni kao „Canceled“ i ne uzimaju se u racunicu za ukupnu sumu troskova za tu odredjenu kategoriju.

**Prikaz ExpensesByCategory.js komponente:**

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

import UserABI from "../../contracts/User.json";

import "./ExpensesByCategory.css";

const ExpensesByCategory = ({ web3, user }) => {

  const [expenses, setExpenses] = useState([]);

  const [totalAmount, setTotalAmount] = useState(0);

  const [category, setCategory] = useState("");

  const loadExpensesByCategory = async () => {

    try {

      const userFactory = new web3.eth.Contract(UserABI.abi, user);

      const expensesFromContract = await userFactory.methods

        .getExpensesByCategory(category)

        .call();

      console.log(expensesFromContract);

      setExpenses(expensesFromContract);

      // Calculate total amount

      const sum = expensesFromContract.reduce((acc, expense) => {

        if (Number(expense.canceled) == 0) {

          return acc + Number(expense.amount);

        }

        return acc;

      }, 0);

      setTotalAmount(sum);

    } catch (error) {

      console.error("Error while loading expenses for user:", error);

    }

  };

  function formatDate(\_date) {

    // Convert Unix timestamp to milliseconds

    const milliseconds = \_date \* 1000;

    // Create a new Date object with the converted milliseconds

    const dateObject = new Date(milliseconds);

    // Extract individual date and time components

    const date = dateObject.toLocaleDateString();

    const time = dateObject.toLocaleTimeString();

    const date\_time = date + " " + time;

    return date\_time;

  }

  const handleChange = (e) => {

    setCategory(e.target.value);

  };

  useEffect(() => {

    if (web3) {

      loadExpensesByCategory();

    }

  }, [web3]);

  //   const openDetailsModal = (client) => {

  //     setSelectedClient(client);

  //   };

  return (

    <div className="client-list">

      <select name="category" onChange={handleChange} className="modal-input">

        <option value="">Select a category</option>

        <option value="Kupovina u prodavnici">Kupovina u prodavnici</option>

        <option value="Racun u kaficu">Racun u kaficu</option>

        <option value="Racun na benzinskoj pumpi">

          Racun na benzinskoj pumpi

        </option>

        <option value="Racun za parking">Racun za parking</option>

      </select>

      <button onClick={loadExpensesByCategory}>Search</button>

      <h1 className="client-list-title">Expenses by category: {category}</h1>

      <h3>Total amount: {totalAmount}</h3>

      <table>

        <tbody>

          <tr>

            <td>Category</td>

            <td>Amount</td>

            <td>Date</td>

          </tr>

          {expenses

            .filter((expense) => Number(expense.canceled) === 0)

            .map((expense, index) => (

              <tr key={index}>

                <td>{expense.category}</td>

                <td>{Number(expense.amount)}</td>

                <td>{formatDate(Number(expense.date))}</td>

              </tr>

            ))}

        </tbody>

      </table>

    </div>

  );

};

export default ExpensesByCategory;

ExpenseDetailsModal je komponenta koja se poziva klikom na dugme „change“ u okviru komponente AllExpenses.js. Ova komponenta omogucaba izmenu troska korisnika. Svi podaci mogu da se menjaju osnim datuma troska koji ce se automatski izracunati u samoj blockchain funkciji koja se poziva u pozadini. Takodje, ovde korisnik moze da oznaci trosak da bude „Canceled“.

**Prikaz ExpenseDetailsModal.js komponente:**

import React, { useState } from "react";

import UserFactoryABI from "../../contracts/UserFactory.json";

import "./ExpenseDetailsModal.css";

const ExpenseDetailsModal = ({

  onClose,

  web3,

  account,

  userFactoryAddress,

  expense,

}) => {

  const [expenseData, setExpenseData] = useState({

    id: Number(expense.id),

    category: expense.category,

    amount: Number(expense.amount),

    description: expense.description,

    canceled: Number(expense.canceled),

  });

  function formatDate(\_date) {

    // Convert Unix timestamp to milliseconds

    const milliseconds = \_date \* 1000;

    // Create a new Date object with the converted milliseconds

    const dateObject = new Date(milliseconds);

    // Extract individual date and time components

    const date = dateObject.toLocaleDateString();

    const time = dateObject.toLocaleTimeString();

    const date\_time = date + " " + time;

    return date\_time;

  }

  const handleChange = (e) => {

    const { name, value, type, checked } = e.target;

    // Handle checkbox differently

    if (type === "checkbox") {

      setExpenseData({ ...expenseData, [name]: checked ? 1 : 0 });

    } else {

      setExpenseData({ ...expenseData, [name]: 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 userFactory = new web3.eth.Contract(

        UserFactoryABI.abi,

        userFactoryAddress

      );

      const transactionParameters = {

        to: userFactoryAddress,

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

        data: userFactory.methods

          .changeExpenseForUser(

            expenseData.id,

            expenseData.category,

            expenseData.amount,

            expenseData.description,

            expenseData.canceled

          )

          .encodeABI({ from: account }),

      }; // 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);

      console.log(expenseData);

      onClose();

    } catch (error) {

      console.error("Error during add of saving a new expense:", error);

    }

  };

  return (

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

      <div className="modal-content">

        <select

          name="category"

          defaultValue={expenseData.category}

          onChange={handleChange}

          className="modal-input"

        >

          <option value="">Select a category</option>

          <option value="Kupovina u prodavnici">Kupovina u prodavnici</option>

          <option value="Racun u kaficu">Racun u kaficu</option>

          <option value="Racun na benzinskoj pumpi">

            Racun na benzinskoj pumpi

          </option>

          <option value="Racun za parking">Racun za parking</option>

        </select>

        <input

          className="modal-input"

          name="amount"

          placeholder="amount"

          value={expenseData.amount}

          onChange={handleChange}

        />

        <input

          className="modal-input"

          value={formatDate(Number(expense.date))}

          disabled

        />

        <textarea

          className="modal-input"

          name="description"

          placeholder="description"

          value={expenseData.description}

          onChange={handleChange}

        />

        {/\* Check if the expense is canceled, if so, apply the round-box class \*/}

        <label>

          Canceled:

          <input

            type="checkbox"

            name="canceled"

            checked={expenseData.canceled === 1}

            onChange={handleChange}

          />

        </label>

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

          Create Expense

        </button>

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

          Cancel

        </button>

      </div>

    </div>

  );

};

export default ExpenseDetailsModal;

CreateNewUserModal komponenta sluzi za kreiranje novog korisnika. Klikom na dugme „Create New User“ kreira se novi korinsnik na back-u u zavisnosti od toga da li je klijent vec u listi ili je zapravo novi korisnik.

**Prikaz CreateNewUserModal.js komponente:**

import React, { useState } from "react";

import UserFactoryABI from "../../contracts/UserFactory.json";

import "./CreateNewUserModal.css";

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

  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 userFactory = new web3.eth.Contract(

        UserFactoryABI.abi,

        userFactoryAddress

      );

      const transactionParameters = {

        to: userFactoryAddress,

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

        data: userFactory.methods.createUser().encodeABI(),

      }; // call to contract method

      const txHash = await window.ethereum.request({

        method: "eth\_sendTransaction",

        params: [transactionParameters],

      });

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

      onClose();

    } catch (error) {

      console.error("Error during creation of a new user:", error);

    }

  };

  return (

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

      <div className="modal-content">

        <input

          className="modal-input"

          readOnly

          name="account"

          value={account}

        />

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

          Create User

        </button>

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

          Cancel

        </button>

      </div>

    </div>

  );

};

export default NewClientModal;

**Prikaz CreateNewExpenseModal.js komponente:**import React, { useState } from "react";

import UserFactoryABI from "../../contracts/UserFactory.json";

import "./CreateNewExpenseModal.css";

const CreateNewExpenseModal = ({

  onClose,

  web3,

  account,

  userFactoryAddress,

}) => {

  const [expenseData, setExpenseData] = useState({

    category: "",

    amount: 0,

    description: "",

  });

  const handleChange = (e) => {

    setExpenseData({ ...expenseData, [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 userFactory = new web3.eth.Contract(

        UserFactoryABI.abi,

        userFactoryAddress

      );

      const transactionParameters = {

        to: userFactoryAddress,

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

        data: userFactory.methods

          .createExpense(

            expenseData.category,

            expenseData.amount,

            expenseData.description

          )

          .encodeABI({ from: account }),

      }; // 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 during add of saving a new expense:", error);

    }

  };

  return (

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

      <div className="modal-content">

        <select name="category" onChange={handleChange} className="modal-input">

          <option value="">Select a category</option>

          <option value="Kupovina u prodavnici">Kupovina u prodavnici</option>

          <option value="Racun u kaficu">Racun u kaficu</option>

          <option value="Racun na benzinskoj pumpi">

            Racun na benzinskoj pumpi

          </option>

          <option value="Racun za parking">Racun za parking</option>

        </select>

        <input

          className="modal-input"

          name="amount"

          placeholder="amount"

          onChange={handleChange}

        />

        <textarea

          className="modal-input"

          name="description"

          placeholder="description"

          onChange={handleChange}

        />

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

          Create Expense

        </button>

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

          Cancel

        </button>

      </div>

    </div>

  );

};

export default CreateNewExpenseModal;

**Prikaz AllExpenses.js komponente:**

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

import UserABI from "../../contracts/User.json";

import "./AllExpenses.css";

import UserFactoryABI from "../../contracts/UserFactory.json";

import ExpenseDetailsModal from "../ExpenseDetailsModal/ExpenseDetailsModal";

const AllExpenses = ({ web3, account, userFactoryAddress, user }) => {

  const [expenses, setExpenses] = useState([]);

  const [totalAmount, setTotalAmount] = useState(0);

  const [selectedExpense, setSelectedExpense] = useState(null);

  const [sortCriteria, setSortCriteria] = useState(null);

  const [sortDirection, setSortDirection] = useState("asc");

  const loadExpenses = async () => {

    try {

      const userFactory = new web3.eth.Contract(UserABI.abi, user);

      const expensesFromContract = await userFactory.methods

        .getExpenses()

        .call();

      console.log(expensesFromContract);

      setExpenses(expensesFromContract);

      // Calculate total amount

      const sum = expensesFromContract.reduce((acc, expense) => {

        return acc + Number(expense.amount);

      }, 0);

      setTotalAmount(sum);

    } catch (error) {

      console.error("Error while loading expenses for user:", error);

    }

  };

  function formatDate(\_date) {

    // Convert Unix timestamp to milliseconds

    const milliseconds = \_date \* 1000;

    // Create a new Date object with the converted milliseconds

    const dateObject = new Date(milliseconds);

    // Extract individual date and time components

    const date = dateObject.toLocaleDateString();

    const time = dateObject.toLocaleTimeString();

    const date\_time = date + " " + time;

    return date\_time;

  }

  const sortBy = (key) => {

    if (sortCriteria === key) {

      // If already sorting by the same criteria, reverse direction

      setSortDirection(sortDirection === "asc" ? "desc" : "asc");

    } else {

      // If sorting by a new criteria, set the criteria and direction

      setSortCriteria(key);

      setSortDirection("asc");

    }

  };

  const sortedExpenses = [...expenses].sort((a, b) => {

    if (sortCriteria === "category") {

      return sortDirection === "asc"

        ? a.category.localeCompare(b.category)

        : b.category.localeCompare(a.category);

    } else if (sortCriteria === "amount") {

      return sortDirection === "asc"

        ? Number(a.amount) - Number(b.amount)

        : Number(b.amount) - Number(a.amount);

    } else if (sortCriteria === "date") {

      return sortDirection === "asc"

        ? new Date(Number(a.date)) - new Date(Number(b.date))

        : new Date(Number(b.date)) - new Date(Number(a.date));

    }

    // If no sorting criteria is selected, return original order

    return 0;

  });

  const openDetailsModal = (expense) => {

    setSelectedExpense(expense);

  };

  useEffect(() => {

    if (web3) {

      loadExpenses();

    }

  }, [web3]);

  return (

    <div className="client-list">

      <h1 className="client-list-title">ALL expenses</h1>

      <h3>Total amount: {totalAmount}</h3>

      <table>

        <thead>

          <tr>

            <th onClick={() => sortBy("category")}>

              Category{" "}

              {sortCriteria === "category" && sortDirection === "asc" && "↑"}

              {sortCriteria === "category" && sortDirection === "desc" && "↓"}

            </th>

            <th onClick={() => sortBy("amount")}>

              Amount{" "}

              {sortCriteria === "amount" && sortDirection === "asc" && "↑"}

              {sortCriteria === "amount" && sortDirection === "desc" && "↓"}

            </th>

            <th onClick={() => sortBy("date")}>

              Date {sortCriteria === "date" && sortDirection === "asc" && "↑"}

              {sortCriteria === "date" && sortDirection === "desc" && "↓"}

            </th>

          </tr>

        </thead>

        <tbody>

          {sortedExpenses.map((expense, index) => (

            <tr

              key={index}

              className={Number(expense.canceled) === 1 ? "canceled-row" : ""}

            >

              <td>{expense.category}</td>

              <td>{Number(expense.amount)}</td>

              <td>{formatDate(Number(expense.date))}</td>

              <td>

                <button onClick={() => openDetailsModal(expense)}>

                  change

                </button>

              </td>

            </tr>

          ))}

        </tbody>

      </table>

      {selectedExpense && (

        <ExpenseDetailsModal

          web3={web3}

          account={account}

          expense={selectedExpense}

          onClose={() => setSelectedExpense(null)}

          userFactoryAddress={userFactoryAddress}

        />

      )}

    </div>

  );

};

export default AllExpenses;

# User Manual

Prikaz kada je korisnik ulogovan na Meta Mask novcanik:

A screenshot of a computer

Description automatically generated

Ukoliko korisnik nije u listi korisnika i pokusa da klikne da dugme „Expenses“ dobice prikaz:  
  
A screenshot of a computer

Description automatically generated

Prikaz modala za kreiranje novog korisnika:  
  
A screenshot of a computer

Description automatically generated

Klikom na dugme „Create User“ izvrsava se transakcija ukoliko korisnik nije u listi korisnika:

A screenshot of a computer

Description automatically generated

Kada je korisnik uspesno ulogovan i nalazi se u listi onda klikom na dugme „Expenses“ moze da vidi sledeci prikaz:  
  
A screenshot of a computer

Description automatically generated

Klikom na dugme „Create New Expense“ korisnik moze da kreira novi trosak i da ga doda u svoju listu troskova:  
  
A screenshot of a computer

Description automatically generated

Klikom na dugme „Show All Expenses“ korisnik moze da prikaze sve svoje troskove tabelarno i moze da vidi ukupnu sumu svih troskova. Sve kolone mogu da se sortiraju opadajuce i rastuce. Kolona je obojena u crveno ukoliko je taj trosak oznacen kao „Canceled“ (namerno se ovde prikazuje jer su ovo all expenses, ali „canceled“ troskovi ce biti iskljuceni iz prikaza i racunice totalAmount u prikazu troskova po kategoriji):  
  
A screenshot of a computer

Description automatically generated

Klikom na dugme „change“ korisnik moze da izmeni izabrani trosak:  
  
A screenshot of a computer

Description automatically generated

Klikom na dugme „Search Expenses By Category“ korisnik moze da izabere kategoriju po kojoj zeli da prikaze troskove i klikom na dugme „Search“ dobija prikaz troskova za tu kategoriju. Takodje, dobija i prikaz ukupnih troskova za tu kategoriju. Ovom priliom prikazuju se samo troskovi za tu odredjenu kategoriju koji nisu oznaceni kao „canceled“ i takodje oni ne ulaze u ukupnu sumu troskova po kategoriji.

A screenshot of a computer

Description automatically generated