ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ

УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра «Програмної інженерії»

Звіт з Лабораторної роботи №2

з дисципліни «Високорівневі мови програмування та фреймворки»

Виконав:

ст. гр. ПЗПІ-20-4 Зінченко Д. Р. Перевірила:

ас. каф. ПІ

Зибіна К. В.

2023

**Тема:**

Створення простого додатку з використання Node.js, бази даних та клієнтської частини.

**Мета:**

Створити простий додаток з використання Node.js, бази даних та клієнтської частини.

**Хід роботи:**

Для виконання роботи було обрано одну з запропонованих тем, а саме «Додаток зі списком фільмів та серіалів для перегляду.»

Для початку було створено веб апі за допомогою Node.js та пакету Express. Отриманий додаток запускає сервер за адресою localhost:7000, налаштування якого було виконано таким чином:

***import* express *from* 'express';**

***import* router *from* "./router.js"**

***import* cors *from* 'cors';**

***import* { testConnection } *from* './models.js';**

**const app = express();**

**const basicAddress = "/api";**

**app.use(express.json());**

**app.use(cors());**

**app.use(basicAddress, router)**

**app.listen(7000, () =>**

**{**

**console.log('server is running at http://localhost:7000/api');**

**testConnection();**

**})**

Цей додаток побудований за архітектурою MVC де роль моделі виконує бібліотека Sequelize, яка надає можливості ORM для зручного доступу до баз даних. Для розробки та тестування додатку було використано легковісну базу даних SQLite для якої було розроблено наступні моделі для користувача та сутності яка буде собою представляти те що користувачеві потрібно передивитись:

***import* { DataTypes, Sequelize } *from* "sequelize";**

**const sequelize = new Sequelize({**

**dialect: 'sqlite',**

**storage: './database.db'**

**});**

***export* const User = sequelize.define("User", {**

**id: {**

**type: DataTypes.INTEGER,**

**primaryKey: true,**

**autoIncrement: true,**

**allowNull: false,**

**unique: true,**

**},**

**email: {**

**type: DataTypes.STRING,**

**allowNull: false,**

**unique: true,**

**},**

**password: {**

**type: DataTypes.STRING,**

**allowNull: false,**

**}**

**})**

***export* const Item = sequelize.define("Item", {**

**id: {**

**type: DataTypes.INTEGER,**

**primaryKey: true,**

**autoIncrement: true,**

**allowNull: false,**

**unique: true,**

**},**

**userId: {**

**type: DataTypes.INTEGER,**

**allowNull: false,**

**},**

**name: {**

**type: DataTypes.STRING,**

**defaultValue: "none"**

**},**

**cover: {**

**type: DataTypes.STRING,**

**defaultValue: "none"**

**},**

**director: {**

**type: DataTypes.STRING,**

**defaultValue: "none"**

**},**

**description: {**

**type: DataTypes.STRING,**

**defaultValue: "none"**

**},**

**type: {**

**type: DataTypes.STRING,**

**defaultValue: "film"**

**},**

**rate: {**

**type: DataTypes.INTEGER,**

**defaultValue: 0**

**},**

**viewed: {**

**type: DataTypes.INTEGER,**

**defaultValue: 0**

**}**

**})**

***export* const testConnection = async () => {**

***try* {**

***await* sequelize.authenticate();**

**console.log('Connection has been established successfully.');**

**} *catch* (error) {**

**console.error('Unable to connect to the database:', error);**

**}**

**}**

Наступним кроком є налаштування контролерів, які будуть використані для обробки запитів що надходять на сервер:

***import* { Item } *from* "./models.js";**

***export* const getUserItems = async (*req*, *res*) => {**

***try* {**

**const items = *await* Item.findAll({**

**where: {**

**userId: *req*.params.userId**

**},**

**});**

***res*.status(200).json(items);**

**} *catch* (error) {**

***res*.status(400).json({ success: false, error: error });**

**}**

**}**

***export* const postItem = async (*req*, *res*) => {**

***try* {**

**const item = *await* Item.create(*req*.body);**

***res*.status(201).json({ success: true, message: "Added successfully" })**

**} *catch* (error) {**

***res*.status(400).json({ success: false, error: error });**

**}**

**}**

***export* const putItem = async (*req*, *res*) => {**

***try* {**

***await* Item.update(*req*.body, {**

**where: {**

**id: *req*.params.id**

**}**

**})**

***res*.status(200).json({ success: true, message: "Updated successfully" });**

**} *catch* (error) {**

***res*.status(400).json({ success: false, error: error });**

**}**

**}**

***export* const deleteItem = async (*req*, *res*) => {**

***try* {**

***await* Item.destroy({**

**where: {**

**id: req.params.id**

**}**

**});**

**res.status(200).json({ success: true, message: "Deleted successfully" });**

**} *catch* (error) {**

**res.status(400).json({ success: false, error: error });**

**}**

**}**

***export* const setViewed = async (*req*, *res*) => {**

***try* {**

***await* Item.update({viewed: *req*.params.viewed}, {**

**where: {**

**id: *req*.params.id**

**}**

**})**

***res*.status(200).json({ success: true, message: "Updated successfully" });**

**} *catch* (error) {**

***res*.status(400).json({ success: false, error: error });**

**}**

**}**

Останнім штрихом для веб сервера, є налаштування шляхів за якими можна отримати доступ до контролерів. Для цього використано функціонал роутеру з пакету Express, де визначені шляхи та функції що оброблюють запити на кожен з шляхів:

***import* express *from* 'express';**

***import* { deleteItem, getUserItems, postItem, putItem, setViewed } *from* './itemsController.js';**

***import* { getUser, logIn, register } *from* './userController.js';**

**const router = express.Router();**

**router.post("/auth/login", logIn);**

**router.post("/auth/register", register);**

**router.get("/auth/user/:id", getUser);**

**router.get('/items/:userId', getUserItems);**

**router.post('/items', postItem);**

**router.put('/items/:id', putItem);**

**router.delete("/items/:id", deleteItem);**

**router.put("/items/viewed/:viewed/:id", setViewed);**

***export* *default* router;**

На даний момент ми закінчили з функціоналом серверу, і можна перейти на клієнт. Його роль відіграє React.js застосунок, який надсилає запити на сервер, отримує дані у форматі json, та відображає їх у вікні браузера користувача. Повний код додатку наведено нижче:

***import* logo *from* './logo.svg';**

***import* './App.css';**

***import* { useState, useEffect } *from* "react"**

***import* axios *from* "axios";**

**function App() {**

**const API = "http://localhost:7000/api/";**

**const [ email, setEmail ] = useState("");**

**const [ password, setPassword ] = useState("");**

**const [ user, setUser ] = useState("none");**

**const [ items, setItems ] = useState([]);**

**useEffect(() => {**

**const userId = localStorage.getItem("userId");**

***if* (userId) {**

**axios.get(API + "auth/user/" + userId)**

**.then((*response*) => {**

**setUser(*response*.data.email);**

**});**

**axios.get(API + "items/" + userId)**

**.then((*response*) => {**

**setItems(*response*.data);**

**console.log('response', *response*)**

**});**

**}**

***else* {**

**setUser("none");**

**setItems([]);**

**}**

**}, [])**

**async function login() {**

**axios.post(API + "auth/login", {email: email, password: password})**

**.then((*response*) => {**

**localStorage.setItem("userId", *response*.data.id);**

**setUser(email);**

**setEmail("");**

**setPassword("");**

**alert(*response*.data.message);**

**window.location.reload(true);**

**})**

**.catch((*err*) => {**

**console.log('err', *err*)**

**alert(*err*.response.data.error);**

**setPassword("");**

**})**

**}**

**async function register() {**

**axios.post(API + "auth/register", {email: email, password: password})**

**.then((*response*) => {**

**localStorage.setItem("userId", *response*.data.id);**

**setUser(email);**

**setEmail("");**

**setPassword("");**

**alert(*response*.data.message);**

**window.location.reload(true);**

**})**

**.catch((*err*) => {**

**alert(*err*.response.data.error);**

**setPassword("");**

**})**

**}**

**function logout() {**

**setUser("none");**

**localStorage.removeItem("userId");**

**window.location.reload(true);**

**}**

**function changeViewed(*val*, *id*) {**

**axios.put(API + `items/viewed/${*val*}/${*id*}`, {})**

**window.location.reload(true);**

**}**

**function handleChange(*e*, *prop*, *index*)  {**

**let updatedValue = {};**

**updatedValue[*prop*] = *e*.target.value;**

**let newArr = [...items];**

**newArr[*index*] = ({**

**...newArr[*index*],**

**...updatedValue**

**})**

**setItems(newArr);**

**}**

**function onCreate() {**

**axios.post(API + "items/", {userId: localStorage.getItem("userId")})**

**.then(*response* => {**

**window.location.reload(true);**

**})**

**.catch((*error*) =>{**

**alert("Error:" + *error*);**

**});**

**}**

**function onUpdate(*index*) {**

**const item = items[*index*];**

**axios.put(API + "items/" + item.id, item)**

**.then(*response* => {**

**alert("Updated");**

**window.location.reload(true);**

**})**

**.catch((*error*) =>{**

**alert("Error:" + *error*);**

**});**

**}**

**function onDelete(*index*) {**

**const item = items[*index*];**

**axios.delete(API + "items/" + item.id)**

**.then(*response* => {**

**alert("Deleted");**

**window.location.reload(true);**

**})**

**.catch((*error*) =>{**

**alert("Error:" + *error*);**

**});**

**}**

***return* (**

**<>**

**<nav className="navbar bg-body-tertiary">**

**<div className="container-fluid">**

**<a className="navbar-brand" href='#main'>Company name</a>**

**<div className="d-flex mx-5 my-3">**

**<h5>{"Logined as: " + user}</h5>**

**</div>**

**</div>**

**</nav>**

**<div className="container text-center">**

**<div className="row text-center">**

**<div className='col'>**

**<h3>#ItemsToWatch</h3>**

**{items.length === 0 ?**

**(<p>No items found. Add new now!</p>) :**

**(<p>All your saved items listed</p>)**

**}**

**<button**

**className='btn btn-outline-warning px-5 py-3 m-4'**

**onClick={onCreate}>Add new item</button>**

**</div>**

**</div>**

**<div className="row row-cols-3 ">**

**<div className='col col-sm-1'></div>**

**<div className='col col-sm-10 text-start'>**

**{items.map((*item*, *i*) =>**

**<div className="card my-2" key={*i*}>**

**<div className="card-header row m-0 justify-content-between">**

**<div className='col-6 align-middle'>{*item*.name + " | " + (*item*.viewed ? "viewed" : "not viewed")}</div>**

**<div className='col-2'>**

**<button**

**className={*item*.viewed ? "btn btn-outline-success" : "btn btn-outline-danger"}**

**onClick={() => changeViewed(*item*.viewed ? 0 : 1, *item*.id)}>**

**{*item*.viewed ? "Make not viewed" : "Make viewed"}**

**</button>**

**</div>**

**</div>**

**<div className="card-body">**

**<div className='row'>**

**<div className='col-auto'>**

**<img src={*item*.cover} alt='cover' style={{width:300}}></img>**

**</div>**

**<div className='col'>**

**<label className="form-label">Name:</label>**

**<input**

**className="form-control"**

**onChange={(*e*) => handleChange(*e*, "name", *i*)}**

**value={*item*.name}/>**

**<label className="form-label">Director:</label>**

**<input**

**className="form-control"**

**onChange={(*e*) => handleChange(*e*, "director", *i*)}**

**value={*item*.director}/>**

**<label className="form-label">Cover:</label>**

**<input**

**className="form-control"**

**onChange={(*e*) => handleChange(*e*, "cover", *i*)}**

**value={*item*.cover}/>**

**<label className="form-label">Type:</label>**

**<select**

**className="form-control"**

**onChange={(*e*) => handleChange(*e*, "type", *i*)}**

**value={*item*.type}>**

**<option>film</option>**

**<option>series</option>**

**<option>cartoon</option>**

**<option>animation</option>**

**</select>**

**<label className="form-label">Rating:</label>**

**<select**

**className="form-control"**

**onChange={(*e*) => handleChange(*e*, "type", *i*)}**

**value={*item*.rate}>**

**<option value={0}>0</option>**

**<option value={1}>1</option>**

**<option value={2}>2</option>**

**<option value={3}>3</option>**

**<option value={4}>4</option>**

**<option value={5}>5</option>**

**</select>**

**<label className="form-label">Description:</label>**

**<textarea**

**rows="5"**

**className="form-control"**

**onChange={(*e*) => handleChange(*e*, "description", *i*)}**

**value={*item*.description}/>**

**</div>**

**</div>**

**</div>**

**<div className='card-footer'>**

**<button**

**className='btn btn-outline-warning mx-5'**

**onClick={() => onDelete(*i*)}>Delete</button>**

**<button**

**className='btn btn-outline-warning mx-5'**

**onClick={() => onUpdate(*i*)}>Update</button>**

**</div>**

**</div>**

**)}**

**</div>**

**<div className='col col-sm-1'></div>**

**</div>**

**<div className="row text-center">**

**<div className='col'>**

**<h3>Login / Register section</h3>**

**<p>input your credentials and log in or register in the system</p>**

**</div>**

**</div>**

**<div className="row row-cols-3">**

**<div className='col'></div>**

**<div className='col'>**

**<label className="form-label my-3">Email:</label>**

**<input**

**className="form-control"**

**onChange={(*e*) => setEmail(*e*.target.value)}**

**value={email}/>**

**<label  className="form-label my-3">Password:</label>**

**<input**

**type="password"**

**className="form-control"**

**onChange={(*e*) => setPassword(*e*.target.value)}**

**value={password}/>**

**<button**

**className="btn btn-outline-warning m-3"**

**onClick={login}>Login</button>**

**<button**

**className="btn btn-outline-warning m-3"**

**onClick={register}>Register</button>**

**<button**

**className="btn btn-outline-warning m-3"**

**onClick={logout}>Logout</button>**

**</div>**

**<div className='col'></div>**

**</div>**

**</div>**

**</>**

**);**

**}**

***export* *default* App;**

Таким чином ми отримали повноцінний клієнт-серверний застосунок, який несе в собі простий функціонал CRUD операцій, пов’язаний з обраною предметною областю. Далі наведено скріншоти робочого додатку.

**Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис**

Рис 1 – Головна сторінка з користувачем що не зайшов в систему

**Зображення, що містить текст, колесо, знімок екрана, Веб-сайт

Автоматично згенерований опис**

Рис 2 – Головна сторінка з користувачем що зайшов в систему

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

Рис 3 – Додано новий пустий фільм для перегляду

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

Рис 4 – Відредаговано дані нового фільму

**Висновки:** В ходілабораторної роботи ми створили простий додаток з використання Node.js, бази даних та клієнтської частини.