Державний вищий навчальний заклад

Ужгородський національний університет

Факультет інформаційних технологій

Кафедра програмного забезпечення систем

**ЛАБОРАТОРНА РОБОТА № 6**

**Тема:** Реалізація фронт-енд частини. Ч3 – бізнес логіка

Виконав студент:

ІІІ курсу спеціальності «Інженерія програмного забезпечення»

Нищий Богдан Вячеславович

**Ужгород-2024**

**Some news project**

1. Реалізувати бізнес логіки свого проєкту

2. Створити користувацькі хуки для отримання (fetch/зщие) даних із/на сервера та для авторизації використовуючи fetch  або axios (по бажанню)

3. Можливе використання бібліотеки tanstack-query (react-query стара назва), по бажанню

4. Створити пул-реквест

5. Оформити відповідного взірця звіт

**Хід роботи**

Проєкт some news включає в себе логіку авторизації та реєстрації, перегляд та створення статей. Для отримання статей використовується news.api – сервіс, що дає змогу безкоштовно отримувати набори новин за різними параметрами у режимі розробки. Для усіх функцій проєкту було створено сервер, який обробляє запити з фронт-енд частини програми. Проте щоб передати чи обробити отримані дані з сервера необхідно реалізувати бізнес логіку фронт-енд частини.

Для створення з’єднання фронт-енда та сервера використовується бібліотека axios.



Рис. 1. Підключена залежність для axios.

**Авторизація та реєстрація**

Розглянути встановлення зв’язку між сервером та фронт-ендом можна на прикладі авторизації та реєстрації.

Коли користувач знаходиться на сторінці «/login» він має можливість ввести дані свого акаунту, або перейти до сторінки реєстрації.

Вважаймо, що користувач ще не має акаунта в some news project, тож він переходить на сторінку реєстрації та вводить необхідні дані для створення акаунта.

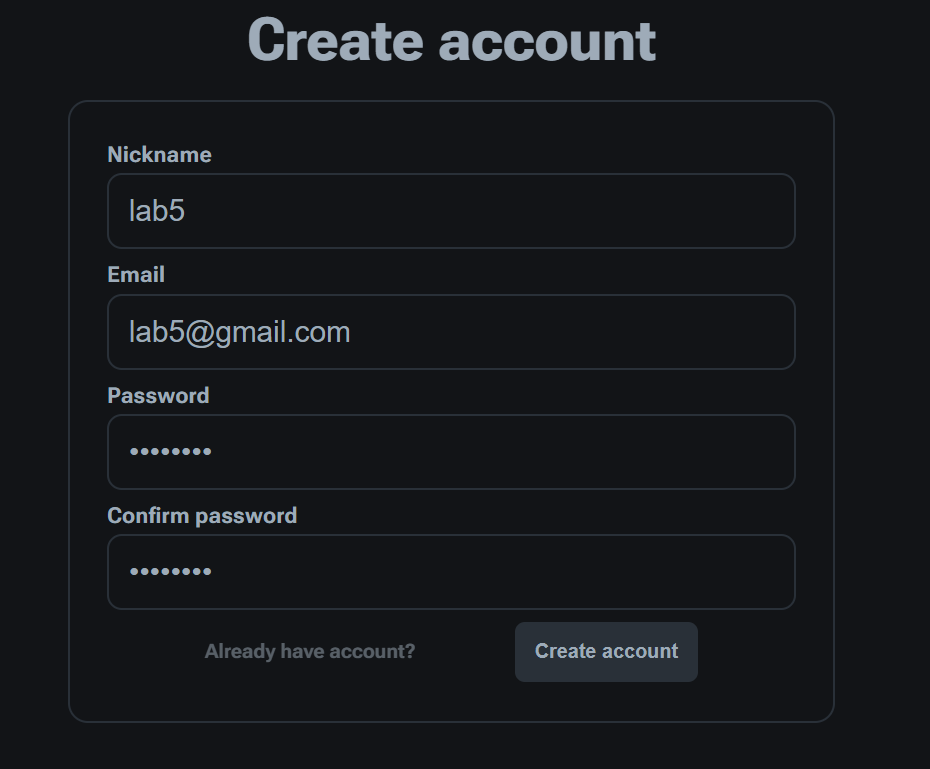


Рис. 2. Введені дані для реєстрації.

Форма, в яку користувач ввів дані, має атрибут onSubmit. Кнопка «Create account» має атрибут type зі значенням «submit», отже підтвердження форми виконається після натискання відповідної кнопки. Після натискання кнопки «Create account», за умови, що всі дані введені вірно, виконується наступний код:

**const** [nickname, setNickname] = useState('');

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

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

**const** [confirmPassword, setConfirmPassword] = useState('');

**const** [error, setError] = useState([]);

**const** navigate = useNavigate();

**const** registerSubmitted = async (event) => {

event.preventDefault();

**try** {

**const** res = await axios.post('http://localhost:5000/api/auth/register', {

nickname,

email,

password,

confirmPassword,

});

localStorage.setItem('token', res.data.token);

navigate('/home');

} **catch** (error) {

console.error(error);

**if** (error.response && error.response.status === **400**) {

**if** (error.response.data.errors) {

**const** errorMessages = error.response.data.errors.map((err) => err.msg);

setError(errorMessages);

} **else** {

setError([error.response.data.message]);

}

} **else** **if** (error.response && error.response.status === **500**) {

setError(['Something went wrong on the server!']);

} **else** {

setError(['An unexpected error occurred']);

}

}

};

Тут react hook useState() створює відповідні змінні для полів форми, і оновлює їх поки користувач вводить дані для реєстрації. У методі registerSubmitted() в конструкції try catch створюється запит на сервер, для створення акаунту користувача.

**const** res = await axios.post('http://localhost:5000/api/auth/register', {

nickname,

email,

password,

confirmPassword,

});

На сервер передаються дані про нікнейм користувача, його email та пароль, разом з його підтвердженням. Ці дані передаються на сервер, за маршрутом /api/auth/register. Якщо на сервері не виникло помилкових ситуацій, або проблем з валідацією даних, відповідний маршрут повертає jwt токен, де фактично зашифрована інформація про користувача. Якщо реєстрація успішна, з допомогою react-router та useNavigate() користувача перенаправляє на головну сторінку.

Якщо помилкові ситуації все ж виникли, catch обробляє їх та виводить користувачу. Основні проблеми, що можуть виникнути – це проблеми з правильністю введеного нікнейму користувача чи нерівність паролів, або ж користувач з заданим email вже існує в системі.

Коли ж користувач вже має акаунт і хоче увійти в нього, він вводить дані для авторизації у відповідні поля та натискає login.

За тим самим принципом виконується підтвердження форми авторизації на сторінці «/login».

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

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

**const** [error, setError] = useState('');

**const** navigate = useNavigate();

**const** loginSubmitted = async (e) => {

e.preventDefault();

**try** {

**const** res = await axios.post('http://localhost:5000/api/auth/login', { email, password });

localStorage.setItem('token', res.data.token);

navigate('/home');

} **catch** (error) {

console.error(error);

**if** (error.status === **500**) {

setError('Something went wrong!');

}

setError('Invalid credentials, please try again.');

}

};

Тут створюються змінні для email та паролю користувача. З допомогою axios та маршруту /api/auth/login надсилається запит на перевірку правильності даних та наявності користувача. Якщо все вірно, створюється jwt токен та користувача перенаправляє на головну сторінку. Якщо щось іде не так, catch обробляє помилки та виводить їх користувачу.

Важливо зазначити: якщо користувач уже авторизований і увійшов в акаунт, то при переході по маршруту /login в браузері, програма перенаправить його на головну сторінку, так як повторна реєстрація чи авторизація неможлива під час активної сесії користувача. Це реалізовано з допомогою react hook useEffect, який запускається при завантажені відповідної сторінки.

useEffect(() => {

**if** (isAuthenticated()) {

navigate('/home');

}

}, [navigate]);

Метод isAuthenticated береться з окремого файлу auth.js в директорії utils.

**export** **const** isAuthenticated = () => {

**const** token = localStorage.getItem('token');

**if** (!token) {

**return** **false**;

}

**try** {

**const** decodedToken = jwtDecode(token);

**const** currentTime = Date.now() / **1000**;

**if** (decodedToken.exp < currentTime) {

localStorage.removeItem('token');

**return** **false**;

}

**return** **true**;

} **catch** (error) {

console.error('Invalid token:', error);

**return** **false**;

}

};

Відповідно, для неавторизованого користувача, який намагається увійти на сторінку, до прикладу, profile було створено користувацький react hook useAuthentication(), який перевіряє чи авторизований користува з допомогою того ж метода, і додатково зберігає дані користувача в користувацькому контексті AuthContext.

**import** { useEffect } from 'react';

**import** { useNavigate } from 'react-router-dom';

**import** { useAuthContext } from '../contexts/AuthContext';

**import** { getUser } from '../utils/userService';

**import** useIsAuthenticated from './useIsAuthenticated';

**export** **const** useAuthentication = () => {

useIsAuthenticated();

**const** navigate = useNavigate();

**const** { user, setUser, setLoading } = useAuthContext();

**const** fetchUserData = async () => {

**if** (!user || user.nickname.length === **0**) {

setLoading(**true**);

**try** {

**const** userData = await getUser();

setUser(userData);

console.log('Found user');

} **catch** (error) {

console.error('Error fetching user data:', error.message);

} **finally** {

setLoading(**false**);

}

}

};

useEffect(() => {

fetchUserData();

}, [fetchUserData]);

};

useIsAuthenticated() – користувацький react hook який якраз і використовує метод розташований в utils.

Дані про користувача беруться з користувацького контекста і зберігаються там же, щоб програма могла використовувати ці дані глобально впродовж сесії користувача.

**const** { user, setUser, setLoading } = useAuthContext();

**Сесія користувача. News.api fetch**

Для отримання даних новин з news.api також було створено відповідний маршрут, до якого звертається сервер, коли отримує запит на отримання даних про новини.

Для цієї задачі також створено користувацький react hook та контекст, який зберігає дані про новини певної категорії для мінімізації кількості запитів до news.api.

При переході користувача на сторінку певної категорії новин (Home, Sport, Science, Technology), apiRoute звідти передається в компонент NewsPages.jsx, який займається створенням запитів до сервера та виведенням даних новин. Далі apiRoute рухається в useFetchNews() для передавання його значення на сервер.

useAuthentication();

useFetchNews({ apiRoute });

**const** { news, loading, error } = useNewsApiContext();

**if** (loading) **return** <BounceLoader className="loader" color="#ffffff" speedMultiplier={**1**} />;

**if** (error) **return** <p>Error loading news: {error.message}</p>;

Тут, в NewsPages.jsx перевіряється чи авторизований користувач з допомогою хука useAuthentication(). Лише після цього викликається useFetchNews(), в який передається apiRoute. Після виконання цих дій, компонент використовує змінні, дані яких вже збережені в контексті.

Виведення лоадера та помилок залежить від їхніх значень в NewsApiContext.

**import** axios from 'axios';

**import** { useEffect } from 'react';

**import** { useNewsApiContext } from '../contexts/NewsApiContext';

**export** **default** **function** useFetchNews({ apiRoute }) {

**const** { news, setNews, setLoading, setError, route, setRoute } = useNewsApiContext();

**const** fetchNews = async () => {

**if** (news.length === **0** || !apiRoute.includes(route)) {

setLoading(**true**);

**try** {

**const** response = await axios.get(apiRoute);

setNews(response.data);

setRoute(apiRoute);

setLoading(**false**);

} **catch** (error) {

console.error('Error fetching news:', error);

setError(error);

setLoading(**false**);

}

}

};

useEffect(() => {

fetchNews();

}, [apiRoute, fetchNews]);

}

Перед виконанням fetchNews використовуються дані з NewsApiContext, які будуть редагуватись і зберігатись глобально у проєкті. Якщо список новин порожній, або сторінка з категоріями змінена, тоді setLoading() встановлюється як true для показу лоадера. Після цього, з допомогою axios робиться запит, куди передається маршрут, за яким сервер виконує цей запит та повертає відповідну категорію новин. Поточна категорія встановлюється в контексті з допомогою setRoute(apiRoute). Дані оновлюються та виводяться на відповідній сторінці для користувача, лоадер зупиняється. При появі помилок, вони обробляються та виводяться.

Усі користувацькі react hooks проєкту, а також користувацькі контексти можна подивитись на [репозиторії проєкту в github](https://github.com/nyshchyibohdan/SomeNewsProject).

Висновок: Було представлено бізнес логіку фронт-енд частини проєкту some news. Було створено користувацькі react hooks та контексти для збереження певних даних глобально в програмі. Було додано зв’язок з сервером та виконано відповідні запити для авторизації, реєстрації та отримання новин з news.api. Для цих завдань було обрано axios.