

Міністерство освіти і науки України
Національний технічний університет України „КПІ імені Ігоря
Сікорського ”

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт до комп'ютерного практикуму №5

З дисципліни «Основи Back-end технологій»

Прийняв:

Викладач Зубко Роман Анатолійович

«__» _____ 2024 р.

Виконав:

Студент 3 курсу, гр. ІІІ-13

Бабашев Олексій Дмитрович

2024 р.

Лабораторна робота №5

Node JS. Робота з БД MongoDB. Додаток, що реалізує CRUD операції в БД.

Завдання:

1. Створити додаток, що реалізує CRUD операції з БД – додавання, читання, редагування та видалення записів БД.
2. Забезпечити роутінг запитів та виведення результатів запитів на WEB-сторінку.
3. Додати новий роут для виведення інформації у вигляді json-файлу.

Лістинг коду:

Серверна частина

index.js є основним файлом серверної частини веб-застосунку, який відповідає за налаштування, запуск та обробку запитів до сервера. Цей файл забезпечує основну конфігурацію серверного додатку, підключення до бази даних та налаштування маршрутизації для обробки HTTP-запитів.

index.js

```
import express from 'express';
import mongoose from 'mongoose';
import cors from 'cors';
import router from './router.js';
import dotenv from 'dotenv';

const PORT = 3000;

dotenv.config();

const app = express();
app.set('view engine', 'ejs');
app.use(express.json());
app.use(cors());
app.use('/api', router);

mongoose
  .connect(process.env.MONGO_URI)
  .then(() => {
    console.log('MongoDB Connected')
  })
  .catch(error => {
    console.log(error)
  })

app.listen(PORT, () => {
  console.log(`Listening on port ${PORT}`)
});
```

router.js відповідає за налаштування маршрутизації для роботи з ресурсами "пости" в додатку на базі Express. У ньому імпортуються необхідні модулі, створюється екземпляр маршрутизатора і визначаються маршрути для різних HTTP-запитів до ресурсів "пости". Цей файл забезпечує зв'язок між HTTP-запитами, отриманими від клієнтів, і відповідними методами контролера, які обробляють ці запити.

router.js

```
import { Router } from "express";
import PostController from "../post/PostController.js";

const router = new Router();

router.post("/posts", PostController.createPost);
router.get("/posts", PostController.getPosts);
router.get("/posts/:id", PostController.getPost);
router.put("/posts/:id", PostController.updatePost);
router.delete("/posts/:id", PostController.deletePost);
router.get("/posts-json", PostController.getPostsJSON);

export default router;
```

Post.js визначає модель даних для постів у базі даних MongoDB за допомогою бібліотеки mongoose. Цей файл забезпечує визначення та експорт моделі Post, яка використовується для взаємодії з колекцією постів у базі даних MongoDB.

Post.js

```
import mongoose from 'mongoose';

const Post = new mongoose.Schema({
  title: {
    type: String,
    required: true
  },
  author: {
    type: String,
    required: true
  },
  text: {
    type: String,
    required: true
  }
}, { timestamps: true });

export default mongoose.model('Post', Post);
```

PostController.js визначає контролер для обробки HTTP-запитів, пов'язаних з постами. Контролер використовує сервіси, що надаються PostService, і містить методи для створення, отримання, оновлення та видалення постів.

Методи контролера:

- **createPost:** Створення нового поста. Приймає дані поста з тіла запиту (req.body) і використовує метод createPost сервісу. Повертає статус 201 (створено) з даними поста або статус 500 (внутрішня помилка сервера) при помилці.
- **getPosts:** Отримання всіх постів. Викликає метод getPosts сервісу і повертає статус 200 (успішно) з даними постів або статус 500 при помилці.
- **getPost:** Отримання конкретного поста за ідентифікатором (req.params.id). Викликає метод getPost сервісу і повертає статус 200 з даними поста або статус 500 при помилці.
- **updatePost:** Оновлення поста за ідентифікатором. Приймає дані поста з тіла запиту і викликає метод updatePost сервісу. Повертає статус 200 з оновленими даними поста або статус 500 при помилці.
- **deletePost:** Видалення поста за ідентифікатором. Викликає метод deletePost сервісу. Повертає статус 204 (немає змісту) або статус 500 при помилці.
- **getPostsJSON:** Отримання всіх постів у форматі JSON. Викликає метод getPosts сервісу і повертає статус 200 з даними постів у форматі JSON або статус 500 при помилці.

Цей файл забезпечує логіку обробки HTTP-запитів для постів, використовуючи сервіси, що надаються PostService, і повертає відповідні відповіді клієнтам.

PostController.js

```
import PostService from "../PostService.js";

class PostController {

  async createPost(req, res) {
    try {
      const post = await PostService.createPost(req.body);
      res.status(201).json({ success: true, data: post });
    } catch (error) {
      res.status(500).json(error);
    }
  };

  async getPosts(_, res) {
    try {
      const posts = await PostService.getPosts();
      res.status(200).json({ success: true, data: posts });
    }
  };
}
```

```

    } catch (error) {
      res.status(500).json({ success: false, error: error.message });
    }
  };

  async getPost(req, res) {
    try {
      const posts = await PostService.getPost(req.params.id);
      res.status(200).json({ success: true, data: posts });
    } catch (error) {
      res.status(500).json({ success: false, error: error.message });
    }
  };

  async updatePost(req, res) {
    try {
      const post = await PostService.updatePost(req.params.id, req.body,
true);
      res.status(200).json({ success: true, data: post });
    } catch (error) {
      res.status(500).json({ success: false, error: error.message });
    }
  };

  async deletePost(req, res) {
    try {
      await PostService.deletePost(req.params.id);
      res.status(204).json({ success: true, data: null });
    } catch (error) {
      res.status(500).json({ success: false, error: error.message });
    }
  };

  async getPostsJSON(_, res) {
    try {
      const posts = await PostService.getPosts();
      res.status(200).json(posts);
    } catch (error) {
      res.status(500).json({ success: false, error: error.message });
    }
  };
}

export default new PostController();

```

PostService.js визначає сервіс для виконання операцій з постами у базі даних за допомогою моделі Post.

Методи сервісу:

- **createPost(postData):** Створення нового поста. Приймає дані поста і створює новий документ у колекції Post. Повертає створений пост.
- **getPosts():** Отримання всіх постів. Виконує запит на отримання всіх документів у колекції Post. Повертає масив постів.
- **getPost(id):** Отримання конкретного поста за ідентифікатором. Приймає ідентифікатор поста і виконує запит на його пошук у колекції Post. Якщо ідентифікатор не надано, викидає помилку. Повертає знайдений пост.
- **updatePost(id, postData, isNew):** Оновлення поста за ідентифікатором. Приймає ідентифікатор поста, нові дані поста і прапорець isNew для визначення, чи повертати оновлений документ. Виконує запит на оновлення поста. Якщо ідентифікатор не надано, викидає помилку. Повертає оновлений пост.
- **deletePost(id):** Видалення поста за ідентифікатором. Приймає ідентифікатор поста і виконує запит на його видалення з колекції Post. Якщо ідентифікатор не надано, викидає помилку. Повертає видалений пост.

Цей файл забезпечує логіку для операцій з постами, включаючи створення, читання, оновлення та видалення постів у базі даних MongoDB.

PostService.js

```
import Post from './Post.js';
class PostService {
  async createPost(postData) {
    const post = await Post.create(postData);
    return post;
  };
  async getPosts() {
    const posts = await Post.find();
    return posts;
  };
  async getPost(id) {
    if (!id) throw new Error('ID is required');
    const post = await Post.findById(id);
    return post;
  }
  async updatePost(id, postData, isNew) {
    if (!id) throw new Error('ID is required');
    const post = await Post.findByIdAndUpdate(id, postData, { new: isNew });
    return post;
  };
  async deletePost(id) {
    if (!id) throw new Error('ID is required');
    const post = await Post.findByIdAndDelete(id);
    return post;
  };
};
export default new PostService();
```

Клієнтська частина

Post.jsx визначає React-компонент для відображення окремого поста з можливістю редагування та видалення.

Post.jsx

```
import styles from "../Post.module.css";
import { Link } from "react-router-dom";

export default function Post({ post, deleteFunc }) {
  return (
    <div className={styles.wrapper}>
      <div className={styles.headingWrapper}>
        <h3 className={styles.postID}>Post #{post._id}</h3>
        <div className={styles.buttons}>
          <Link className={styles.editBtn} to={`/edit-post/${post._id}`}>
            
          </Link>
          <button className={styles.deleteBtn} onClick={() =>
deleteFunc(post._id)}>
            
          </button>
        </div>
      </div>
      <h4 className={styles.title}>{post.title}</h4>
      <p className={styles.text}>{post.text}</p>
      <p className={styles.author}>by {post.author}</p>
    </div>
  );
}
```

AllPosts.jsx визначає React-компонент для відображення всіх постів, що зберігаються на сервері. Цей компонент забезпечує завантаження та відображення всіх постів з сервера, а також надає можливість видалення постів через відповідні функції.

AllPosts.jsx

```
import { useState, useEffect } from "react";
import Post from "../../components/post/Post";
import styles from "../AllPosts.module.css";

export default function AllPosts() {

  const [posts, setPosts] = useState([]);

  useEffect(() => {
    const fetchPosts = async () => {
      const response = await fetch("http://localhost:3000/api/posts");
      const data = await response.json();
      setPosts(data.data);
    };
  });
}
```

```

    };
    fetchPosts();
  }, [])

  const deletePost = async (postId) => {
    await fetch(`http://localhost:3000/api/posts/${postId}`, {
      method: "DELETE",
    });

    setPosts(posts.filter(post => post._id !== postId));
  };

  return (
    <section className={styles.section}>
      <h1 className={styles.heading}>All Posts</h1>
      <div className={styles.postGrid}>
        {posts.map((post) => (
          <Post key={post._id} post={post} deleteFunc={deletePost}/>
        ))}
      </div>
    </section>
  );
}

```

CreatePost.jsx містить React-компонент для створення нового поста або редагування існуючого. Компонент отримує дані поста для редагування (якщо ідентифікатор передано через маршрут) та надає можливість вводу заголовку, автора та тексту поста. Після введення даних користувач може надіслати форму, яка відправляє дані на сервер для збереження або оновлення поста. Після успішної операції користувач перенаправляється на сторінку всіх постів.

CreatePost.jsx

```

import { useParams } from "react-router-dom";
import styles from "./CreatePost.module.css";
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";

export default function CreatePost() {
  const [isSubmitting, setIsSubmitting] = useState(false);
  const [title, setTitle] = useState("");
  const [author, setAuthor] = useState("");
  const [text, setText] = useState("");
  const navigate = useNavigate();

  const params = useParams();
  const id = params.id;

  useEffect(() => {
    if (id) {
      fetch(`http://localhost:3000/api/posts/${id}`)
    }
  });
}

```



```

        .then((response) => response.json())
        .then((data) => {
            setTitle(data.data.title);
            setAuthor(data.data.author);
            setText(data.data.text);
        });
    }
}, [id]);

function handleTitleChange(e) {
    setTitle(e.target.value);
}

function handleAuthorChange(e) {
    setAuthor(e.target.value);
}

function handleTextChange(e) {
    setText(e.target.value);
}

function handleSubmit(e) {
    e.preventDefault();
    const post = {
        title: title,
        author: author,
        text: text,
    };

    setIsSubmitting(true);

    if (id) {
        fetch(`http://localhost:3000/api/posts/${id}`, {
            method: "PUT",
            headers: {
                "Content-Type": "application/json",
            },
            body: JSON.stringify(post),
        })
        .then((response) => response.json())
        .then((data) => {
            setIsSubmitting(false);
            navigate("/posts");
            console.log(data);
        })
        .catch((error) => {
            console.error("Error:", error);
        });
    } else {
        fetch(`http://localhost:3000/api/posts`, {
            method: "POST",

```

```

        headers: {
            "Content-Type": "application/json",
        },
        body: JSON.stringify(post),
    })
    .then((response) => response.json())
    .then((data) => {
        setIsSubmitting(false);
        navigate("/posts");
        console.log(data);
    })
    .catch((error) => {
        console.error("Error:", error);
    });
}
}
return (
    <section className={styles.section}>
        <div className={styles.wrapper}>
            <h1 className={styles.heading}>{id ? `Editing post #${id}` : "Creating
new Post"}</h1>
            <form className={styles.form} action="" onSubmit={(e) =>
handleSubmit(e)}>
                <input
                    className={styles.input}
                    type="text"
                    name="title"
                    value={title}
                    onChange={(e) => handleTitleChange(e)}>
                <input
                    className={styles.input}
                    type="text"
                    name="author"
                    value={author}
                    onChange={(e) => handleAuthorChange(e)}>
                <textarea
                    className={styles.textarea}
                    name="text"
                    value={text}
                    onChange={(e) => handleTextChange(e)}
                ></textarea>
                <button disabled={isSubmitting} className={styles.button}
type="submit">
                    {isSubmitting ? "Submitting..." : "Submit"}
                </button>
            </form>
        </div>
    </section>
);
}

```

Header.jsx містить React-компонент для відображення заголовка сторінки з навігаційним меню.

Header.jsx

```
import styles from './Header.module.css';
import { Link } from 'react-router-dom';

export default function Header() {
  return (
    <header className={styles.header}>
      <ul className={styles.ul}>
        <li className={styles.li}><Link to={"/"}>Home</Link></li>
        <li className={styles.li}><Link to={"/posts"}>Posts</Link></li>
        <li className={styles.li}><Link to={"/create-post"}>New
Post</Link></li>
      </ul>
    </header>
  );
}
```

Layout.jsx містить React-компонент для створення макету сторінки. Компонент складається з двох частин: Header (заголовок сторінки) та Outlet (вихід), який відображає контент відповідно до маршрутів, визначених в додатку. В основній частині компонента розміщений контейнер <main>, в якому відображається зміст сторінки, який залежить від поточного маршруту.

Layout.jsx

```
import { Outlet } from 'react-router-dom';
import Header from '../header/Header';

const Layout = () => {
  return (
    <>
      <Header />
      <main>
        <Outlet />
      </main>
    </>
  );
}

export default Layout;
```

HomePage.jsx відповідає за відображення головної сторінки додатка. Він імпортує головний компонент додатка App, встановлює строгий режим React (StrictMode) та використовує BrowserRouter для забезпечення маршрутизації в додатку. Далі, використовуючи ReactDOM.createRoot, компонент App рендериться в кореневий елемент з ідентифікатором "root" на HTML-сторінці.

HomePage.jsx

```
import styles from './HomePage.module.css';

export default function HomePage() {
  return (
    <section className={styles.section}>
      <h1 className={styles.home}>Personal Blog</h1>
      <p className={styles.p}>Made with React and Express</p>
    </section>
  );
}
```

App.jsx визначає основний компонент додатка, який встановлює маршрутизацію за допомогою компонентів Routes та Route з бібліотеки react-router-dom.

App.jsx

```
import { Routes, Route } from "react-router-dom";
import Layout from "../components/layout/Layout";
import HomePage from "../pages/HomePage/HomePage";
import AllPosts from "../pages/AllPosts/AllPosts";
import CreatePost from "../pages/CreatePost/CreatePost";

function App() {
  return (
    <Routes>
      <Route path="/" element={<Layout />}>
        <Route index element={<HomePage />} />
        <Route path="/posts" element={<AllPosts />} />
        <Route path="/create-post" element={<CreatePost />} />
        <Route path="/edit-post/:id" element={<CreatePost />} />
      </Route>
    </Routes>
  );
}

export default App;
```

main.jsx відповідає за використання ReactDOM для рендерингу головного компонента додатка (App) у кореневому елементі з ідентифікатором "root" на HTML-сторінці. Також він використовує строгий режим React (StrictMode) для виявлення потенційних проблем в додатку.

main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "../App.jsx";
import "../index.css";
import { BrowserRouter as Router } from "react-router-dom";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <Router>
      <App />
    </Router>
  </React.StrictMode>
);
```

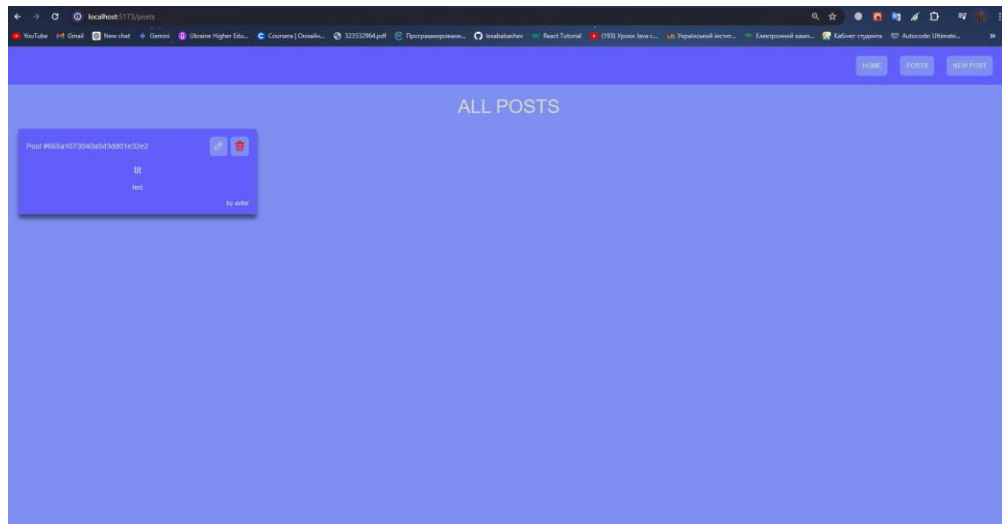
Цей HTML-код представляє собою базовий шаблон для сторінки, яка використовується для рендерингу React-додатка.

index.html

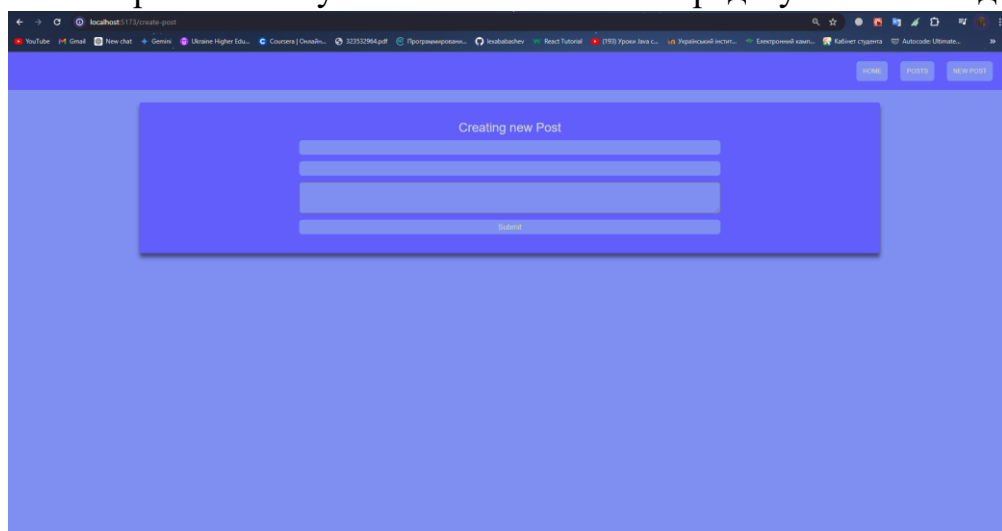
```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Результати виконання коду:

Застосунок було розроблено та успішно протестовано.

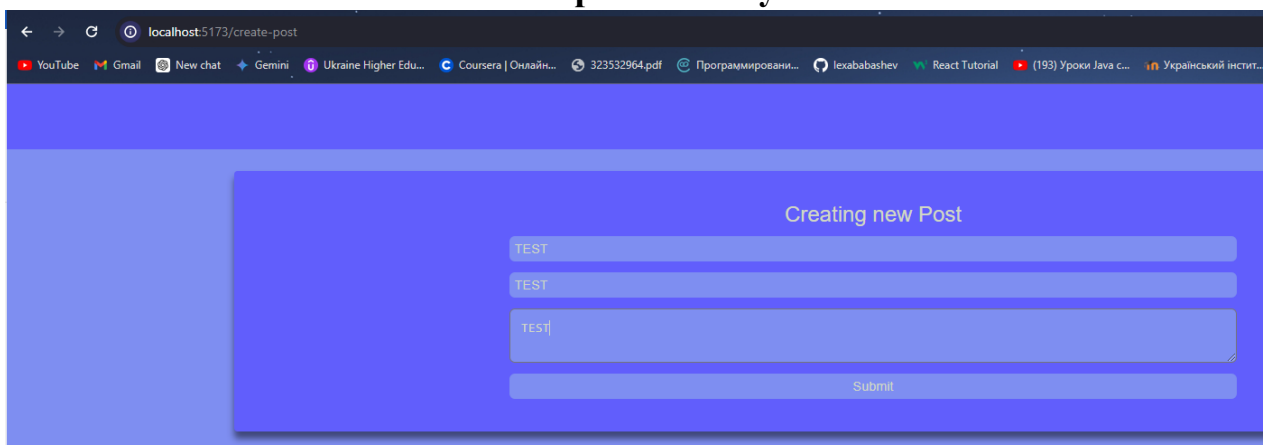


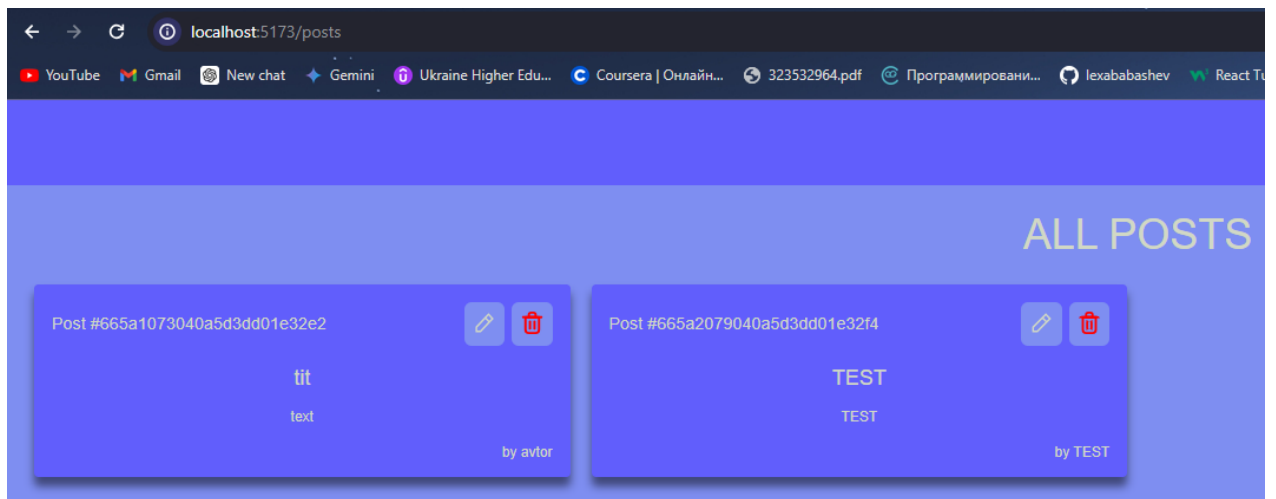
Мал 1.1 – Сторінка списку постів із можливістю редагування та видалення



Мал 1.2 – Сторінка з формою для створення постів.

Створення посту





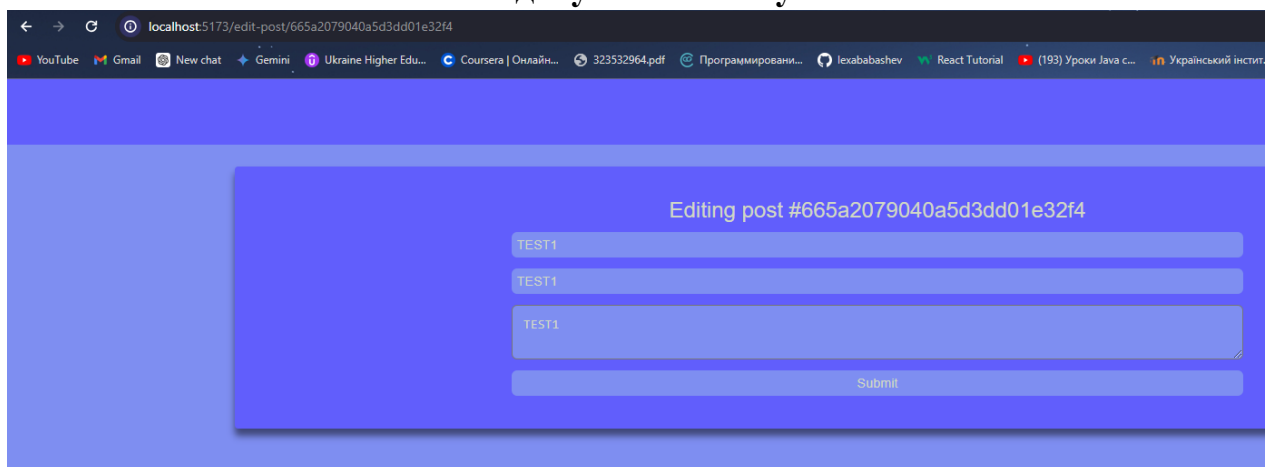
QUERY RESULTS: 1-2 OF 2

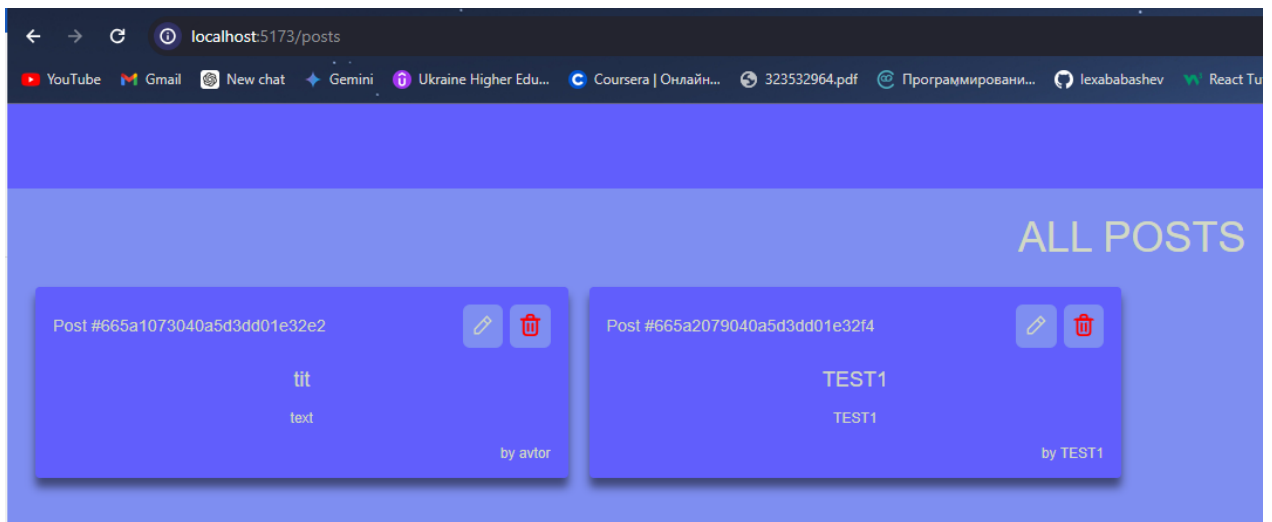
```
_id: ObjectId('665a1073040a5d3dd01e32e2')
title: "tit"
author: "avtor"
text: "text"
createdAt: 2024-05-31T18:01:23.725+00:00
updatedAt: 2024-05-31T18:09:55.356+00:00
__v: 0
```



```
_id: ObjectId('665a210c040a5d3dd01e3305')
title: "TEST"
author: "TEST"
text: "TEST"
createdAt: 2024-05-31T19:12:12.015+00:00
updatedAt: 2024-05-31T19:12:12.015+00:00
__v: 0
```

Редагування посту





QUERY RESULTS: 1-2 OF 2

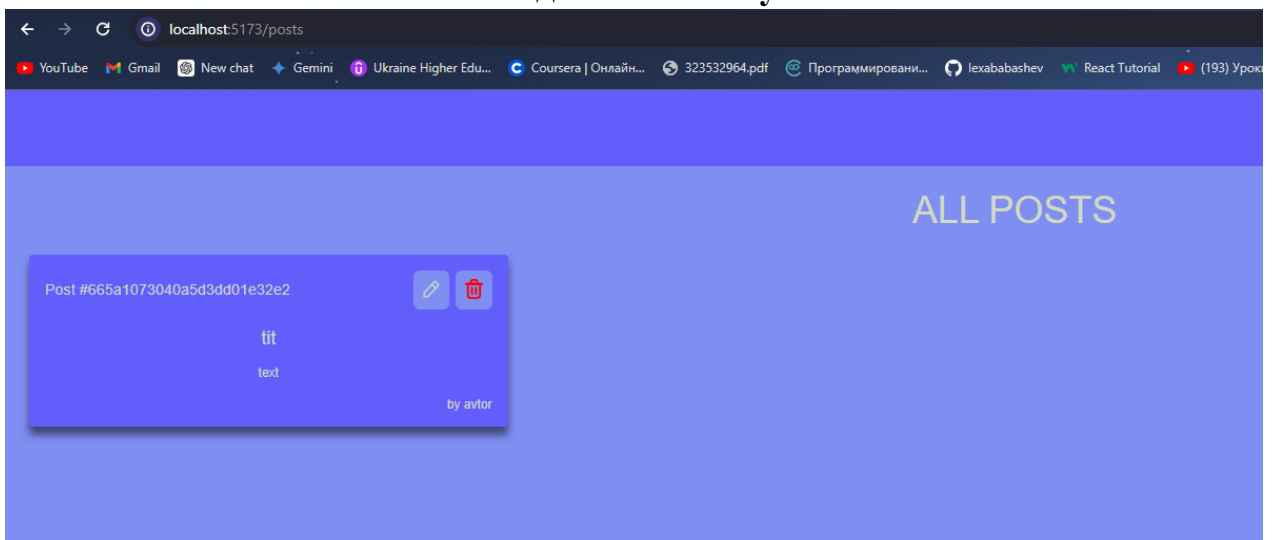
```
_id: ObjectId('665a1073040a5d3dd01e32e2')
title: "tit"
author: "avtor"
text: "text"
createdAt: 2024-05-31T18:01:23.725+00:00
updatedAt: 2024-05-31T18:09:55.356+00:00
__v: 0
```



```
_id: ObjectId('665a210c040a5d3dd01e3305')
title: "TEST1"
author: "TEST1"
text: "TEST1"
createdAt: 2024-05-31T19:12:12.015+00:00
updatedAt: 2024-05-31T19:13:25.374+00:00
__v: 0
```

TEST1

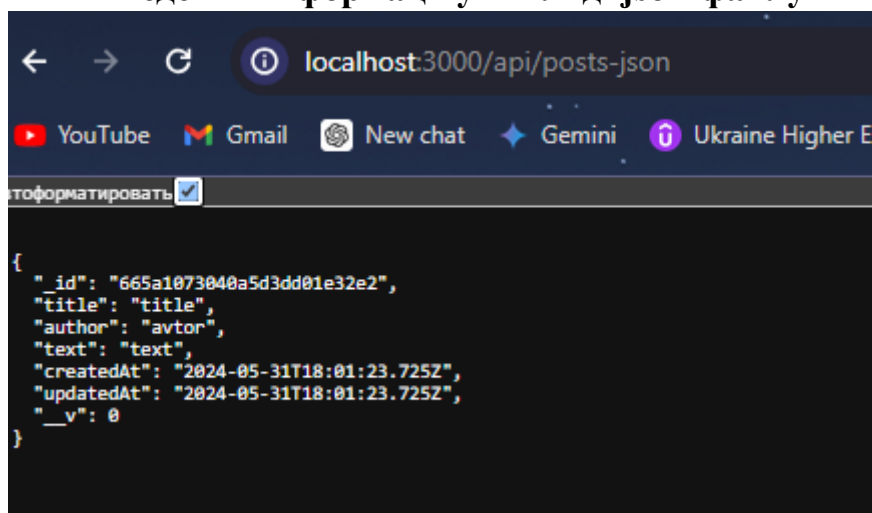
Видалення посту



QUERY RESULTS: 1-1 OF 1

```
_id: ObjectId('665a1073040a5d3dd01e32e2')
title : "tit"
author : "avtor"
text : "text"
createdAt : 2024-05-31T18:01:23.725+00:00
updatedAt : 2024-05-31T18:09:55.356+00:00
__v : 0
```

Виведення інформації у вигляді json-файлу



```
{
  "id": "665a1073040a5d3dd01e32e2",
  "title": "title",
  "author": "avtor",
  "text": "text",
  "createdAt": "2024-05-31T18:01:23.725Z",
  "updatedAt": "2024-05-31T18:01:23.725Z",
  "__v": 0
}
```

Висновок

У цій лабораторній роботі ми створили веб-додаток з використанням Node.js та MongoDB, що реалізує CRUD операції з базою даних. Веб-додаток дозволяє нам взаємодіяти з базою даних MongoDB через веб-інтерфейс та отримувати дані у форматі JSON.