

Міністерство освіти і науки України

Національний технічний університет України „КПІ імені Ігоря Сікорського ”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

**Звіт до комп’ютерного практикуму №5**

**З дисципліни «Основи Back-end технологій»**

Прийняв: Виконав:

Викладач Зубко Роман Анатолійович Студент 3 курсу,гр. ІП-13

Ал Хадам Мурат

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2024 р.

**2024 р.**

**Комп’ютерний практикум №5.**

**NodeJS. 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.

**Хід роботи**

1. Створимо сервер, де реалізуються CRUD операції та забезпечимо роутінг для API. В якості БД будемо використовувати MongoDB.

Для початку реалізуємо підключення до БД MongoDB.

app.set('view engine', 'ejs');

app.use(express.json());

app.use(cors());

app.use('/api', postRoutes);

mongoose

    .connect(config.mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => {

        console.log('MongoDB Connected')

    })

    .catch(err => {

        console.log(err)

    })

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}`)

});

Створимо схему таблиці.

const Schema = mongoose.Schema;

const postSchema = new Schema({

    title: {

        type: String,

        required: true

    },

    author: {

        type: String,

        required: true

    },

    text: {

        type: String,

        required: true

    }

}, { timestamps: true });

const Post = mongoose.model('Post', postSchema);

module.exports = Post;

Імплементуємо контролери для наших операцій.

// Контролер для створення посту

exports.createPost = async (req, res) => {

    try {

        const post = await Post.create(req.body);

        res.status(201).json({ success: true, data: post });

    } catch (error) {

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

    }

};

// Контролер для отримання всіх постів

exports.getPosts = async (req, res) => {

    try {

        const posts = await Post.find();

        res.status(200).json({ success: true, data: posts });

    } catch (error) {

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

    }

};

// Контролер для оновлення маршруту за ідентифікатором

exports.updatePost = async (req, res) => {

    try {

        const post = await Post.findByIdAndUpdate(req.params.id, req.body, { new: true });

        res.status(200).json({ success: true, data: post });

    } catch (error) {

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

    }

};

// Контролер для видалення маршруту за ідентифікатором

exports.deletePost = async (req, res) => {

    try {

        await Post.findByIdAndDelete(req.params.id);

        res.status(204).json({ success: true, data: null });

    } catch (error) {

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

    }

};

// Контролер для отримання маршрутів у форматі JSON

exports.getPostsJSON = async (req, res) => {

    try {

        const posts = await Post.find();

        res.status(200).json(posts);

    } catch (error) {

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

    }

};

Пропишемо маршрути API для виконання CRUD операцій.

// Маршрути для CRUD операцій над маршрутами

// Створення маршруту

router.post('/posts', postController.createPost);

// Отримання всіх маршрутів

router.get('/posts', postController.getPosts);

// Оновлення маршруту

router.put('/posts/:id', postController.updatePost);

// Видалення маршруту

router.delete('/posts/:id', postController.deletePost);

// Отримання всіх маршрутів у форматі JSON

router.get('/posts-json', postController.getPostsJSON);

module.exports = router;

Протестуємо роботу API.

Тестуємо GET запит.

Зображення, що містить знімок екрана, текст, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Протестуємо додавання нового посту в БД.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Протестуємо запит на оновлення даних.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Запис в БД так само оновився.

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Протестуємо запит на видалення запису з БД.

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Бачимо, що запис відсутній

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

Автоматично згенерований опис

Протестуємо запит на виведення інформації у вигляді JSON.

Зображення, що містить знімок екрана, текст, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

1. За допомогою React реалізуємо клієнтську частину, яка виводить результати запитів на WEB-сторінку.

Це головний компонент, який координує взаємодію між іншими компонентами, обробляє запити до сервера і забезпечує динамічне оновлення інтерфейсу користувача. Функції handleSave та handleDelete забезпечують відповідне додавання, оновлення та видалення постів через API. Компоненти PostForm і PostList взаємодіють з цими функціями для надання можливості редагування, збереження та видалення постів.

const App = () => {

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

  const [selectedPost, setSelectedPost] = useState(null);

  const fetchPosts = async () => {

    const response = await api.get('/posts');

    setPosts(response.data.data);

  };

  useEffect(() => {

    fetchPosts();

  }, []);

  const handleEdit = (post) => {

    setSelectedPost(post);

  };

  const handleSave = async (post) => {

    if (selectedPost) {

      // оновлюємо пост

      const response = await api.put(`/posts/${post.\_id}`, post);

      setPosts(posts.map((p) => (p.\_id === post.\_id ? response.data.data : p)));

    } else {

      // додаємо новий пост

      const response = await api.post('/posts', post);

      setPosts([...posts, response.data.data]);

    }

    setSelectedPost(null);

  };

  const handleDelete = async (id) => {

    await api.delete(`/posts/${id}`);

    setPosts(posts.filter((post) => post.\_id !== id));

  };

  return (

    <div className="container">

      <h1>CRUD Posts</h1>

      <PostForm post={selectedPost} onSave={handleSave} />

      <PostList posts={posts} onEdit={handleEdit} onDelete={handleDelete} />

    </div>

  );

};

Компонент, який представляє елемент зі списку постів

const PostItem = ({ post, onDelete, onEdit }) => {

  return (

    <li className="post-item">

      <h3>{post.title}</h3>

      <p><strong>Author:</strong> {post.author}</p>

      <p>{post.text}</p>

      <button onClick={() => onEdit(post)} className="edit-button">Edit</button>

      <button onClick={() => onDelete(post.\_id)} className="delete-button">Delete</button>

    </li>

  );

};

Компонент, який представляє сам список постів.

const PostList = ({ posts, onEdit, onDelete }) => {

  return (

    <div>

      <h2>Posts</h2>

      <ul className="post-list">

        {posts.map((post) => (

          <PostItem

            key={post.\_id}

            post={post}

            onDelete={onDelete}

            onEdit={onEdit}

          />

        ))}

      </ul>

    </div>

  );

};

Компонент, що представляє форму для створення та редагування записів.

const PostForm = ({ post, onSave }) => {

  const [formData, setFormData] = useState({

    title: '',

    author: '',

    text: ''

  });

  useEffect(() => {

    if (post) {

      setFormData(post);

    } else {

      setFormData({ title: '', author: '', text: '' });

    }

  }, [post]);

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({ ...formData, [name]: value });

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    onSave(formData);

  };

  return (

    <form onSubmit={handleSubmit} className="post-form">

      <input

        type="text"

        name="title"

        value={formData.title}

        onChange={handleChange}

        placeholder="Title"

        required

      />

      <input

        type="text"

        name="author"

        value={formData.author}

        onChange={handleChange}

        placeholder="Author"

        required

      />

      <textarea

        name="text"

        value={formData.text}

        onChange={handleChange}

        placeholder="Text"

        required

      />

      <button type="submit">Save</button>

    </form>

  );

};

Скріншот веб-застосунку.

Зображення, що містить знімок екрана, текст, програмне забезпечення, Комп’ютерна піктограма

Автоматично згенерований опис

**Скрипт КП:**

postController.js

const Post = require('../models/postModel');

// Контролер для створення посту

exports.createPost = async (req, res) => {

    try {

        const post = await Post.create(req.body);

        res.status(201).json({ success: true, data: post });

    } catch (error) {

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

    }

};

// Контролер для отримання всіх постів

exports.getPosts = async (req, res) => {

    try {

        const posts = await Post.find();

        res.status(200).json({ success: true, data: posts });

    } catch (error) {

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

    }

};

// Контролер для оновлення маршруту за ідентифікатором

exports.updatePost = async (req, res) => {

    try {

        const post = await Post.findByIdAndUpdate(req.params.id, req.body, { new: true });

        res.status(200).json({ success: true, data: post });

    } catch (error) {

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

    }

};

// Контролер для видалення маршруту за ідентифікатором

exports.deletePost = async (req, res) => {

    try {

        await Post.findByIdAndDelete(req.params.id);

        res.status(204).json({ success: true, data: null });

    } catch (error) {

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

    }

};

// Контролер для отримання маршрутів у форматі JSON

exports.getPostsJSON = async (req, res) => {

    try {

        const posts = await Post.find();

        res.status(200).json(posts);

    } catch (error) {

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

    }

};

postModel.js

const mongoose = require('mongoose');

const Schema = mongoose.Schema;

const postSchema = new Schema({

    title: {

        type: String,

        required: true

    },

    author: {

        type: String,

        required: true

    },

    text: {

        type: String,

        required: true

    }

}, { timestamps: true });

const Post = mongoose.model('Post', postSchema);

module.exports = Post;

postRoutes.js

const express = require('express');

const router = express.Router();

const postController = require('../controllers/postController');

// Маршрути для CRUD операцій над маршрутами

// Створення маршруту

router.post('/posts', postController.createPost);

// Отримання всіх маршрутів

router.get('/posts', postController.getPosts);

// Оновлення маршруту

router.put('/posts/:id', postController.updatePost);

// Видалення маршруту

router.delete('/posts/:id', postController.deletePost);

// Отримання всіх маршрутів у форматі JSON

router.get('/posts-json', postController.getPostsJSON);

module.exports = router;

index.js

const express = require('express');

const app = express();

const mongoose = require('mongoose');

const config = require('./config.js');

const cors = require('cors');

const postRoutes = require('./routes/postRoutes.js');

const PORT = 3000;

app.set('view engine', 'ejs');

app.use(express.json());

app.use(cors());

app.use('/api', postRoutes);

mongoose

    .connect(config.mongoURI, { useNewUrlParser: true, useUnifiedTopology: true })

    .then(() => {

        console.log('MongoDB Connected')

    })

    .catch(err => {

        console.log(err)

    })

app.listen(PORT, () => {

    console.log(`Server is running on port ${PORT}`)

});

App.jsx

import { useState, useEffect } from 'react';

import PostList from './components/PostList';

import PostForm from './components/PostForm';

import api from './api';

import './App.css';

const App = () => {

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

  const [selectedPost, setSelectedPost] = useState(null);

  const fetchPosts = async () => {

    const response = await api.get('/posts');

    setPosts(response.data.data);

  };

  useEffect(() => {

    fetchPosts();

  }, []);

  const handleEdit = (post) => {

    setSelectedPost(post);

  };

  const handleSave = async (post) => {

    if (selectedPost) {

      // оновлюємо пост

      const response = await api.put(`/posts/${post.\_id}`, post);

      setPosts(posts.map((p) => (p.\_id === post.\_id ? response.data.data : p)));

    } else {

      // додаємо новий пост

      const response = await api.post('/posts', post);

      setPosts([...posts, response.data.data]);

    }

    setSelectedPost(null);

  };

  const handleDelete = async (id) => {

    await api.delete(`/posts/${id}`);

    setPosts(posts.filter((post) => post.\_id !== id));

  };

  return (

    <div className="container">

      <h1>CRUD Posts</h1>

      <PostForm post={selectedPost} onSave={handleSave} />

      <PostList posts={posts} onEdit={handleEdit} onDelete={handleDelete} />

    </div>

  );

};

export default App;

api.jsx

import axios from 'axios';

const api = axios.create({

  baseURL: 'http://localhost:3000/api'

});

export default api;

PostForm.jsx

import { useState, useEffect } from 'react';

import '../App.css';

const PostForm = ({ post, onSave }) => {

  const [formData, setFormData] = useState({

    title: '',

    author: '',

    text: ''

  });

  useEffect(() => {

    if (post) {

      setFormData(post);

    } else {

      setFormData({ title: '', author: '', text: '' });

    }

  }, [post]);

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({ ...formData, [name]: value });

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    onSave(formData);

  };

  return (

    <form onSubmit={handleSubmit} className="post-form">

      <input

        type="text"

        name="title"

        value={formData.title}

        onChange={handleChange}

        placeholder="Title"

        required

      />

      <input

        type="text"

        name="author"

        value={formData.author}

        onChange={handleChange}

        placeholder="Author"

        required

      />

      <textarea

        name="text"

        value={formData.text}

        onChange={handleChange}

        placeholder="Text"

        required

      />

      <button type="submit">Save</button>

    </form>

  );

};

export default PostForm;

PostItem.jsx

import '../App.css';

const PostItem = ({ post, onDelete, onEdit }) => {

  return (

    <li className="post-item">

      <h3>{post.title}</h3>

      <p><strong>Author:</strong> {post.author}</p>

      <p>{post.text}</p>

      <button onClick={() => onEdit(post)} className="edit-button">Edit</button>

      <button onClick={() => onDelete(post.\_id)} className="delete-button">Delete</button>

    </li>

  );

};

export default PostItem;

PostList.jsx

import PostItem from './PostItem';

import '../App.css';

const PostList = ({ posts, onEdit, onDelete }) => {

  return (

    <div>

      <h2>Posts</h2>

      <ul className="post-list">

        {posts.map((post) => (

          <PostItem

            key={post.\_id}

            post={post}

            onDelete={onDelete}

            onEdit={onEdit}

          />

        ))}

      </ul>

    </div>

  );

};

export default PostList;

App.css

body {

    font-family: Arial, sans-serif;

    background-color: #f8f9fa;

    margin: 0;

    padding: 0;

  }

  .container {

    padding: 20px;

    max-width: 800px;

    margin: 0 auto;

  }

  /\* Стилі для PostForm \*/

  .post-form {

    display: flex;

    flex-direction: column;

    gap: 10px;

    margin-bottom: 20px;

  }

  .post-form input,

  .post-form textarea {

    padding: 10px;

    border: 1px solid #ccc;

    border-radius: 5px;

    resize: vertical;

  }

  .post-form button {

    padding: 10px;

    border: none;

    border-radius: 5px;

    background-color: #007bff;

    color: white;

    cursor: pointer;

  }

  .post-form button:hover {

    background-color: #0056b3;

  }

  /\* Стилі для PostList \*/

  .post-list {

    list-style: none;

    padding: 0;

  }

  .post-list h2 {

    text-align: center;

  }

  /\* Стилі для PostItem \*/

  .post-item {

    background-color: white;

    padding: 15px;

    margin-bottom: 10px;

    border-radius: 5px;

    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

  }

  .post-item h3 {

    margin-top: 0;

  }

  .post-item p {

    margin: 5px 0;

  }

  .post-item button {

    padding: 5px 10px;

    border: none;

    border-radius: 3px;

    cursor: pointer;

    margin-right: 5px;

  }

  .post-item .edit-button {

    background-color: #ffc107;

    color: white;

  }

  .post-item .edit-button:hover {

    background-color: #e0a800;

  }

  .post-item .delete-button {

    background-color: #dc3545;

    color: white;

  }

  .post-item .delete-button:hover {

    background-color: #c82333;

  }