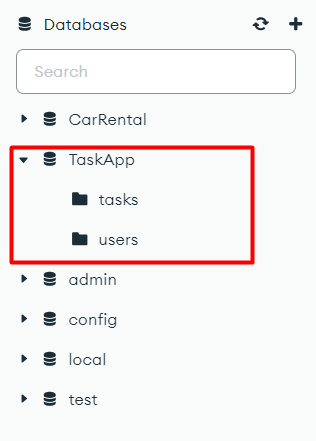
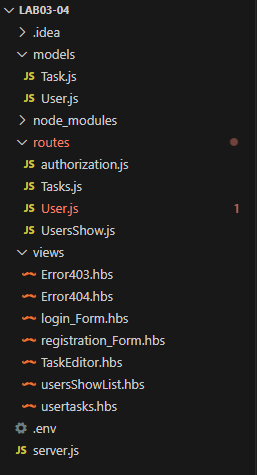
**Лабораторна робота №3-5**

**Тема: Mongoose Rest API/ API Authentication and Security/ TaskApp**

***Мета роботи:*** навчитися працювати з Mongoose.

**Хід роботи:**





**Лістинг Models:**

**User.js:**

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

    nickname: {

        type: String,

        required: true

    },

    email: {

        type: String,

        required: true,

        unique: true

    },

    password: {

        type: String,

        required: true

    },

    age: {

        type: Number,

        required: true

    },

    ID: {

        type: String,

        required: true

    },

    role: {

        type: String,

        required: true

    }

});

const User = mongoose.model('User', userSchema);

module.exports = User;

**Task.js:**

const mongoose = require('mongoose');

const taskSchema = new mongoose.Schema({

    userId: {

        type: mongoose.Schema.Types.ObjectId,

        ref: 'User',

        required: true

    },

    title: {

        type: String,

        required: true

    },

    description: {

        type: String

    },

    completed: {

        type: Boolean,

        default: false

    }

});

const Task = mongoose.model('Task', taskSchema);

module.exports = Task;

**Routes:**

**authorization.js:**

const express = require('express');

const router = express.Router();

const bcrypt = require('bcrypt');

const User = require('../models/User');

// Маршрут для обробки даних входу

router.post('/login', async (req, res) => {

    const { email, password } = req.body;

    try {

        const user = await User.findOne({ email });

        if (!user) {

            res.status(401).send('Invalid email or password');

            return;

        }

        const passwordMatch = await bcrypt.compare(password, user.password);

        if (passwordMatch) {

            req.session.userId = user.\_id;

// Перевіряємо роль користувача

            if (user.role === 'superuser') {

                res.redirect('/user/userlist'); // Перенаправляємо суперкористувача насторінку списку користувачів

            } else {

                res.redirect('/user/tasks'); // Перенаправляємо звичайного користувача насторінку із завданнями

            }

        } else {

            res.status(401).send('Invalid email or password');}

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

});

// Маршрут для відображення сторінки входу

router.get('/login', (req, res) => {

    res.render('loginForm');

});

// Маршрут для виходу користувача із системи

router.get('/logout', (req, res) => {

// Видаляємо сесію користувача

    req.session.destroy((err) => {

        if (err) {

            console.error(err);

            res.status(500).send('Internal Server Error');

        } else {

            res.redirect('/auth/login');

        }

    });

});

module.exports = router;

**Tasks.js:**

const express = require('express');

const router = express.Router();

const bcrypt = require('bcrypt');

const Task = require('../models/Task');

const User = require('../models/User');

function requireAuth(req, res, next) {

    if (req.session.userId) {

        next();

    } else {

        res.status(403);

        res.render('Error403'); // Рендер шаблону для помилки 403

    }

}

// Middleware для перевірки прав доступу до завдання

async function checkTaskOwnership(req, res, next) {

    const userId = req.session.userId;

    const taskId = req.params.taskId;

    try {

        const task = await Task.findOne({ \_id: taskId, userId: userId });

        if (!task) {return res.status(404).render('Error404');

        }

        next(); // Продовжуємо виконання наступного маршруту, якщо користувач має доступдо завдання

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

}

// Маршрут для обробки POST-запиту на додавання нового завдання

router.post('/', requireAuth, async (req, res) => {

    const userId = req.session.userId;

    const { title, description } = req.body;

// Проверка наличия title в теле запроса

    if (!title) {

        return res.status(400).send('Title is required');

    }

    try {

        const newTask = new Task({ title, description, userId });

        await newTask.save();

        res.redirect('/user/tasks');

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

});

// Маршрут GET для редагування завдання

router.get('/:taskId/edit', requireAuth, checkTaskOwnership, async (req, res) => {

    const taskId = req.params.taskId;

    try {

        const task = await Task.findById(taskId);

        res.render('editTask', { task });

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

});

// Маршрут POST для редагування завдання

router.post('/:taskId/edit', requireAuth, checkTaskOwnership, async (req, res) => {

    const taskId = req.params.taskId;

    const { title, description } = req.body;

    try {

        const updatedTask = await Task.findByIdAndUpdate(taskId, { title, description },

            { new: true });if (!updatedTask) {

            return res.status(404).render('Error404'); // Рендер шаблону для помилки 404

        }

        res.redirect('/user/tasks');

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

});

// Маршрут для видалення завдання

router.post('/:taskId/delete', requireAuth, checkTaskOwnership, async (req, res) => {

    const taskId = req.params.taskId;

    try {

        const deletedTask = await Task.findByIdAndDelete(taskId);

        if (!deletedTask) {

            return res.status(404).render('Error404'); // Рендер шаблону для помилки 404

        }

        res.redirect('/user/tasks');

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

});

module.exports = router;

**User.js:**

const express = require('express');

const router = express.Router();

const bcrypt = require('bcrypt');

const Task = require('../models/Task');

const User = require('../models/User');

function requireAuth(req, res, next) {

    if (req.session.userId) {

        next();

    } else {

        res.status(403);

        res.render('Error403'); // Рендер шаблону для помилки 403

    }

}

// Middleware для перевірки ролі суперкористувача

async function requireSuperuser(req, res, next) {

    const userId = req.session.userId;

    try {

        const user = await User.findById(userId);

        if (!user || user.role !== 'superuser') {

            res.status(403).send('Error403'); // Рендер шаблону для помилки 403

        } else {

            next(); // Продовжуємо виконання наступного маршруту

        }

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

}

// Маршрут для відображення сторінки реєстрації користувача

router.get('/registration', (req, res) => {

    res.render('registrationForm');

});// Маршрут для обработки данных регистрации пользователя

router.post('/registration', async (req, res) => {

    const { nickname, email, password, age } = req.body;

// Валідація пошти

    const emailRegex = /\S+@\S+\.\S+/;

    if (!emailRegex.test(email)) {

        return res.status(400).send('Invalid email format');

    }

// Валідація пароля

    const passwordRegex = /^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])[0-9a-zA-Z]{8,}$/;

    if (!passwordRegex.test(password)) {

        return res.status(400).send('Password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, and one digit');

    }

// Валідація віку

    if (age < 18) {

        return res.status(400).send('Age must be at least 18');

    }

    try {

// Хешуємо пароль перед збереженням

        const hashedPassword = await bcrypt.hash(password, 10);

        const newUser = new User({

            nickname,

            email,

            password: hashedPassword, // Зберігаємо захешований пароль

            age,

            ID: 'NO',

            role: 'user'

        });

        await newUser.save();

        res.send('User registered successfully!');

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

});

// Маршрут для відображення списку завдань для всіх користувачів

router.get('/tasks', requireAuth, async (req, res) => {

    try {

        const tasks = await Task.find(); // Відображення списку тасків

        res.render('userTasks', { tasks }); // Рендер шаблону із завданнями для всіх користувачів

    } catch (error) {// Маршрут для обработки данных регистрации пользователя

        router.post('/registration', async (req, res) => {

            const { nickname, email, password, age } = req.body;

// Валідація пошти

            const emailRegex = /\S+@\S+\.\S+/;

            if (!emailRegex.test(email)) {

                return res.status(400).send('Invalid email format');

            }

// Валідація пароля

            const passwordRegex = /^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])[0-9a-zA-Z]{8,}$/;

            if (!passwordRegex.test(password)) {

                return res.status(400).send('Password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, and one digit');

            }

// Валідація віку

            if (age < 18) {

                return res.status(400).send('Age must be at least 18');

            }

            try {

// Хешуємо пароль перед збереженням

                const hashedPassword = await bcrypt.hash(password, 10);

                const newUser = new User({

                    nickname,

                    email,

                    password: hashedPassword, // Зберігаємо захешований пароль

                    age,

                    ID: 'NO',

                    role: 'user'

                });

                await newUser.save();

                res.send('User registered successfully!');

            } catch (error) {

                console.error(error);

                res.status(500).send('Internal Server Error');

            }

        });

// Маршрут для відображення списку завдань для всіх користувачів

        router.get('/tasks', requireAuth, async (req, res) => {

            try {

                const tasks = await Task.find(); // Відображення списку тасків

                res.render('userTasks', { tasks }); // Рендер шаблону із завданнями для всіх користувачів

            } catch (error) {console.error(error);

                res.status(500).send('Internal Server Error');

            }

        });

// Маршрут для відображення сторінки списку користувачів

        router.get('/userlist', requireSuperuser, async (req, res) => {

            try {

                const users = await User.find(); // Отримуємо список користувачів

                res.render('userList', { users }); // Рендер шаблону зі списком користувачів

            } catch (error) {

                console.error(error);

                res.status(500).send('Internal Server Error');

            }

        });

        router.get('/logout', (req, res) => {

// Видаляємо дані про користувача із сесії

            req.session.destroy(err => {

                if (err) {

                    console.error(err);

                    res.status(500).send('Internal Server Error');

                } else {

                    res.redirect('/auth/login'); // Перенаправляємо користувача на сторінку входу

                }

            });

        });

        module.exports = router;}

**UsersShow.js**

const express = require('express');

const router = express.Router();

const User = require('../models/User');

// Маршрут для відображення сторінки списку користувачів

router.get('/userlist', async (req, res) => {

    try {

// Фільтруємо користувачів, щоб відобразити тільки тих, хто не є суперкористувачами

        const users = await User.find({ role: { $ne: "superuser" } });

        res.render('userList', { users });

    } catch (error) {

        console.error(error);

        res.status(500).send('Internal Server Error');

    }

});

// Маршрут для видалення користувача

router.post('/user/delete/:userId', async (req, res) => {

    const userId = req.params.userId;

    try {// Видаляємо користувача за його ідентифікатором

        await User.findByIdAndDelete(userId);

        res.redirect('/user/userlist');

    } catch (error) {

        console.error(error);

        res.status(500).send('Failed to delete user');

    }

});

module.exports = router;

**Views:**

**TaskEditor.hbs:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Edit Task</title>

    <style>

        <!-- Стилі -->

    </style>

</head>

<body>

<div class="container">

    <h1>Edit Task</h1>

    <a href="/user/tasks">Back to Tasks</a>

    <form action="/user/tasks/{{ task.\_id }}/edit" method="POST">

        <!-- Використовуємо метод PUT для оновлення завдання --><input type="hidden" name="\_method" value="PUT">

        <!-- Вставляємо CSRF токен для безпеки -->

        <input type="hidden" name="\_csrf" value="{{csrfToken}}">

        <label for="title">Title:</label>

        <input type="text" id="title" name="title" value="{{ task.title }}" required><br>

        <label for="description">Description:</label>

        <textarea id="description" name="description" required>{{

        task.description }}</textarea><br>

        <button type="submit">Update Task</button>

    </form>

</div>

</body>

</html>

**Error403.hbs:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Error 403 - Forbidden</title>

    <style>

        <!-- Стилі --> </style>

</head>

<body>

<div class="container">

    <h1>Error 403 - Forbidden</h1>

    <p>Access to this resource is forbidden. Please <a

            href="/auth/login">login</a> to continue.</p>

</div>

</body>

</html>

**Error404.hbs:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0"><title>Error 404 - Not Found</title>

    <style>

        <!-- Стилі --> </style>

</head>

<body>

<div class="container">

    <h1>Error 404 - Not Found</h1>

    <p>The requested resource could not be found on this server. Please check the

        URL or go back to <a href="/user/tasks">homepage</a>.</p>

</div>

</body>

</html>

**Login\_Form.hbs:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>User Login</title>

    <style>

        <!-- Стилі --> </style>

</head>

<body>

<div class="container">

    <h1>User Login</h1>

    <form action="/auth/login" method="POST">

        <label for="email">Email:</label>

        <input type="email" id="email" name="email" required>

        <label for="password">Password:</label>

        <input type="password" id="password" name="password" required>

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

    </form>

    <form action="/user/registration" method="GET">

        <button type="submit" class="back-btn">Go to Registration</button>

    </form>

</div>

</body>

</html>

**Registration\_Form.hbs:**

<!DOCTYPE html>

<html lang="en">

<head><meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>User Registration</title>

    <style>

        <!-- Стилі --> </style>

</head>

<body>

<div class="container">

    <h1>User Registration</h1>

    <form action="/user/registration" method="POST">

        <label for="nickname">Nickname:</label>

        <input type="text" id="nickname" name="nickname" required>

        <label for="email">Email:</label>

        <input type="email" id="email" name="email" required>

        <label for="password">Password:</label>

        <input type="password" id="password" name="password" required>

        <label for="age">Age:</label>

        <input type="number" id="age" name="age" required>

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

    </form>

    <form action="/auth/login" method="GET">

        <button type="submit" class="back-btn">Back to Login</button>

    </form>

</div>

</body>

</html>

**usersShowList.hbs:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>User List</title>

    <style>

        <!-- Стилі --> </style>

</head>

<body>

<h1>User List</h1>

<table>

    <thead>

    <tr>

        <th>Name</th>

        <th>Email</th>

        <th>Age</th><th>Actions</th>

    </tr>

    </thead>

    <tbody>

    {{#each users}}

        <tr>

            <td>{{this.nickname}}</td>

            <td>{{this.email}}</td>

            <td>{{this.age}}</td>

            <td>

                <form action="/user/delete/{{this.\_id}}" method="POST">

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

                </form>

            </td>

        </tr>

    {{/each}}

    </tbody>

</table>

<a href="/auth/logout" class="logout-button">Logout</a>

</body>

</html>

**UsersTasks.hbs:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>User Tasks</title>

    <style>

        <!-- Стилі --> </style>

</head>

<body>

<div class="container">

    <h1>User Tasks</h1>

    <!-- Форма для додавання нового таску -->

    <form action="/user/tasks" method="POST">

        <label for="title">Title:</label>

        <input type="text" id="title" name="title" placeholder="Enter title" required>

        <label for="description">Description:</label>

        <textarea id="description" name="description" placeholder="Enter description" required></textarea>

        <button type="submit">Add Task</button>

    </form>

    <!-- Таблиця із завданнями -->

    <table>

        <thead>

        <tr>

            <th>Title</th><th>Description</th>

            <th>Actions</th>

        </tr>

        </thead>

        <tbody>

        {{#each tasks}}

            <tr>

                <td>{{this.title}}</td>

                <td>{{this.description}}</td>

                <td class="actions">

                    <a href="/user/tasks/{{this.\_id}}/edit">Edit</a>

                    <a href="/user/tasks/{{this.\_id}}/delete" onclick="event.preventDefault(); if(confirm('Are you sure you want to delete this task?')) { document.getElementById('delete-form-{{this.\_id}}').submit(); }">Delete</a>

                    <form id="delete-form-{{this.\_id}}" action="/user/tasks/{{this.\_id}}/delete" method="POST" style="display: none;">

                        <input type="hidden" name="\_method" value="DELETE">

                        <input type="hidden" name="\_csrf" value="{{csrfToken}}">

                    </form>

                </td>

            </tr>

        {{/each}}

        </tbody>

    </table>

    <a href="/auth/logout" class="logout-button">Logout</a>

</div>

</body>

</html>

**Server.js:**

require('dotenv').config();

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const session = require('express-session');

// Підключення маршутів

const userRoutes = require('./routes/User');

const authRoutes = require('./routes/authorization');

const taskRoutes = require('./routes/Tasks');

const userListRoutes = require('./routes/UsersShow');

// Екземпляр Express

const app = express();

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json());

app.use(session({

    secret: 'secret',

    resave: true,

    saveUninitialized: false

}));

function requireAuth(req, res, next) {

    if (req.session.userId) {

        next();

    } else {

        res.redirect('/auth/login');

    }

}

app.use((req, res, next) => {

    res.locals.userId = req.session.userId;

    next();

});

// Налаштування для файлів .hbs

app.set('views', \_\_dirname + '/views');

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

// Підключення до бд MongoDB

mongoose.connect(process.env.MONGO\_URL, { useNewUrlParser: true, useUnifiedTopology:

        true })

    .then(() => console.log('MongoDB Connected')) // Перевірка звязку з бд

    .catch(err => console.error(err));

// Підключаємо маршрути до додатка

app.use('/user', userRoutes);

app.use('/auth', authRoutes);

app.use('/user/tasks', requireAuth, taskRoutes);

app.use(userListRoutes);

app.use((err, req, res, next) => {

    console.error(err.stack);

    res.status(500).send('Something broke!');

});

app.use((req, res, next) => {

    res.redirect('/auth/login');

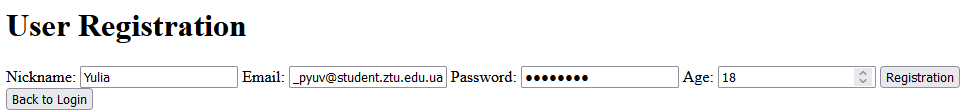
});

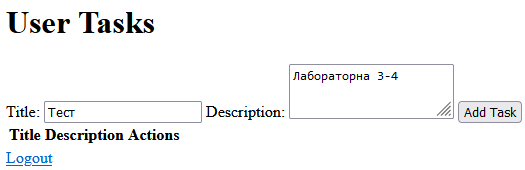
const PORT = process.env.PORT || 3000;

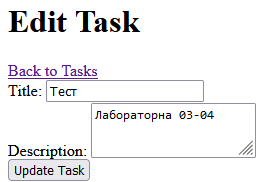
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

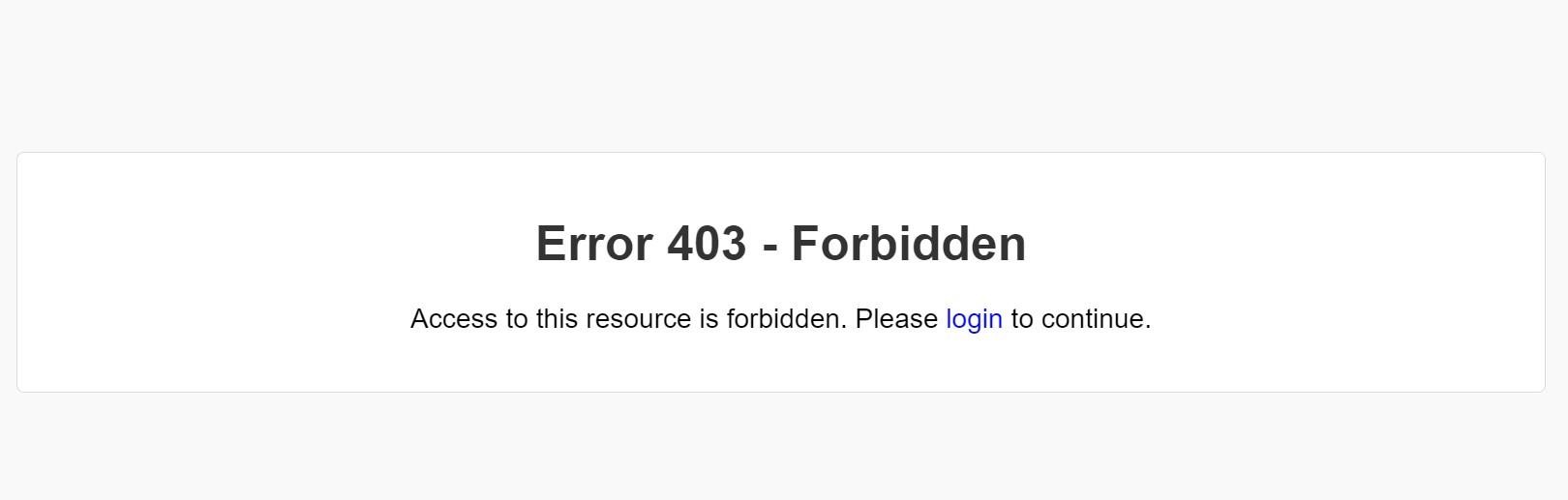
**File .env:**

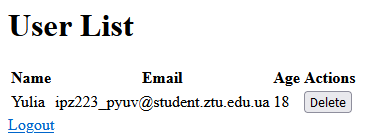
MONGO\_URL=mongodb://localhost:27017/TaskApp

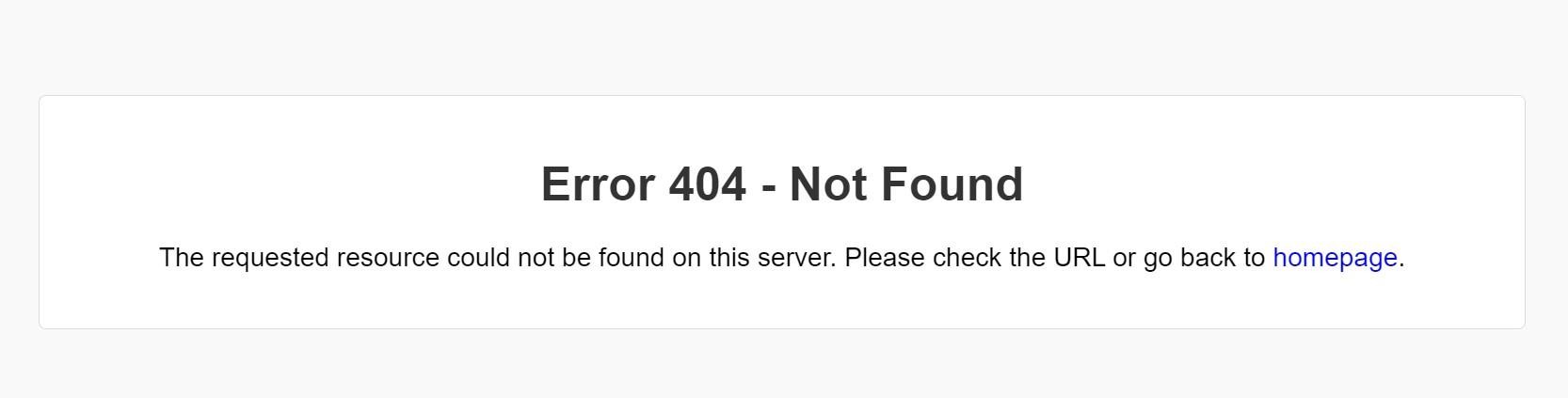












**Висновок:** навчилася використовувати mongoDB в NodeJS.