Лаб 3 Середа Іван Вт-22-1[1]

**Завдання :**

Розробити RESTful API-додаток з обробкою таких запитів для даних users і tasks:

GET (/users, /tasks)

GET (/users/:id, /tasks/:id)

POST (/users, /tasks)

PATCH (/users/:id, /tasks/:id)

DELETE (/users/:id, /tasks/:id)

DELETE (/users, /tasks)

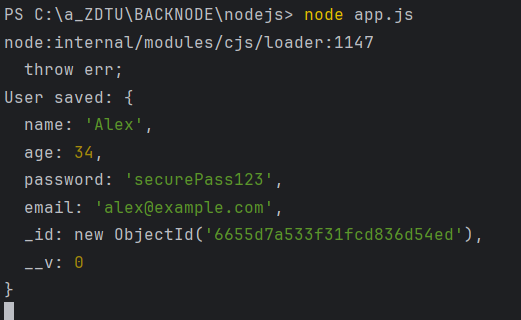
**Завдання 1. Створіть проект TaskApp**

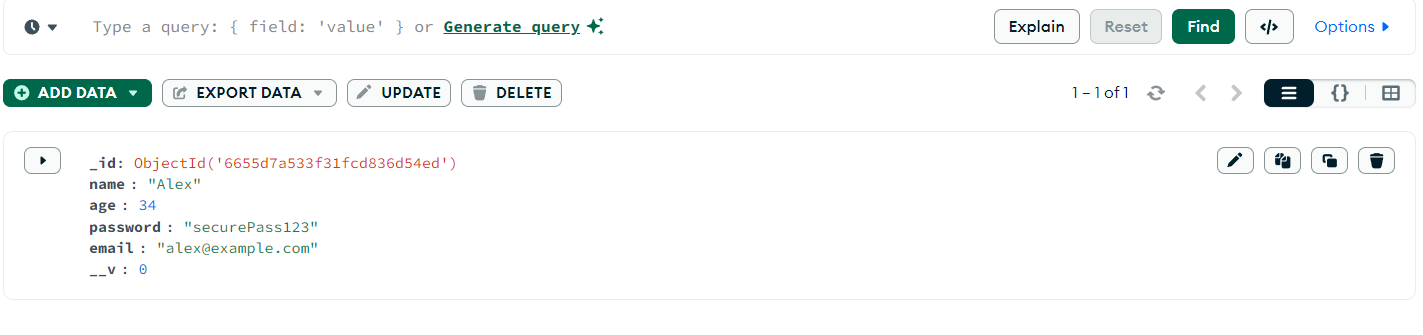
**Завдання 2. Підключимось до бази даних та створимо модель User**

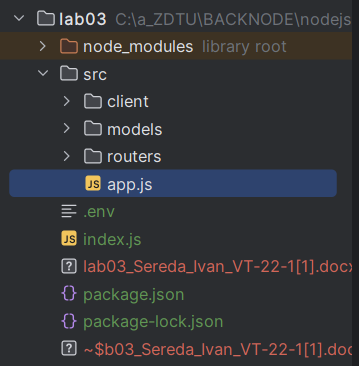
Лістинг :

require('dotenv').config();  
const express = require('express');  
const mongoose = require('mongoose');  
const *User* = require('./models/user');  
const *Task* = require('./models/task');  
  
const app = express();  
const url = *process*.env.MONGODB\_URI || 'mongodb://localhost:27017/node\_db';  
*console*.log(*process*.env.MONGODB\_URI);  
*console*.log(url)  
mongoose.connect(url)  
 .then(() => {  
 *console*.log('Connected to MongoDB');  
  
 const user = new *User*({  
 name: 'Alex',  
 age: 34,  
 email: 'alex@example.com',  
 password: 'securePass123'  
 });  
  
 user.save()  
 .then(() => {  
 *console*.log('User saved:', user);  
 })  
 .catch((error) => {  
 *console*.error('Error saving user:', error);  
 });  
 })  
 .catch((error) => {  
 *console*.error('Error connecting to MongoDB:', error);  
 });  
  
app.listen(3000, () => {  
 *console*.log('Server is running on port 3000');  
});

Результат:







**Завдання для моделі User**

Лістинг модуля:

const mongoose = require('mongoose');  
  
const userSchema = new mongoose.Schema({  
 name: {  
 type: String,  
 required: [true, 'Name is required'],  
 minlength: [3, 'Name must be at least 3 characters long'],  
 maxlength: [20, 'Name must be at most 20 characters long'],  
 match: [/^[a-zA-Z\s]\*$/, 'Name must contain only letters'],  
 unique: true // Забезпечує унікальність значень у полі name  
 },  
 age: {  
 type: Number,  
 required: [true, 'Age is required'],  
 min: [13, 'Age must be at least 13 years old'],  
 max: [90, 'Age must be at most 90 years old']  
 },  
 password: {  
 type: String,  
 required: [true, 'Password is required'],  
 minlength: [8, 'Password must be at least 8 characters long'],  
 validate: {  
 validator: function(v) {  
 return !/password/i.test(v);  
 },  
 message: 'Password cannot contain the word "password".'  
 }  
 },  
 email: {  
 type: String,  
 required: [true, 'Email is required'],  
 match: [/\S+@\S+\.\S+/, 'Email is not valid'],  
 unique: true  
 }  
});  
  
  
userSchema.index({ name: 1, email: 1 }, { unique: true });  
  
const *User* = mongoose.model('User', userSchema);  
  
*module*.exports = *User*;

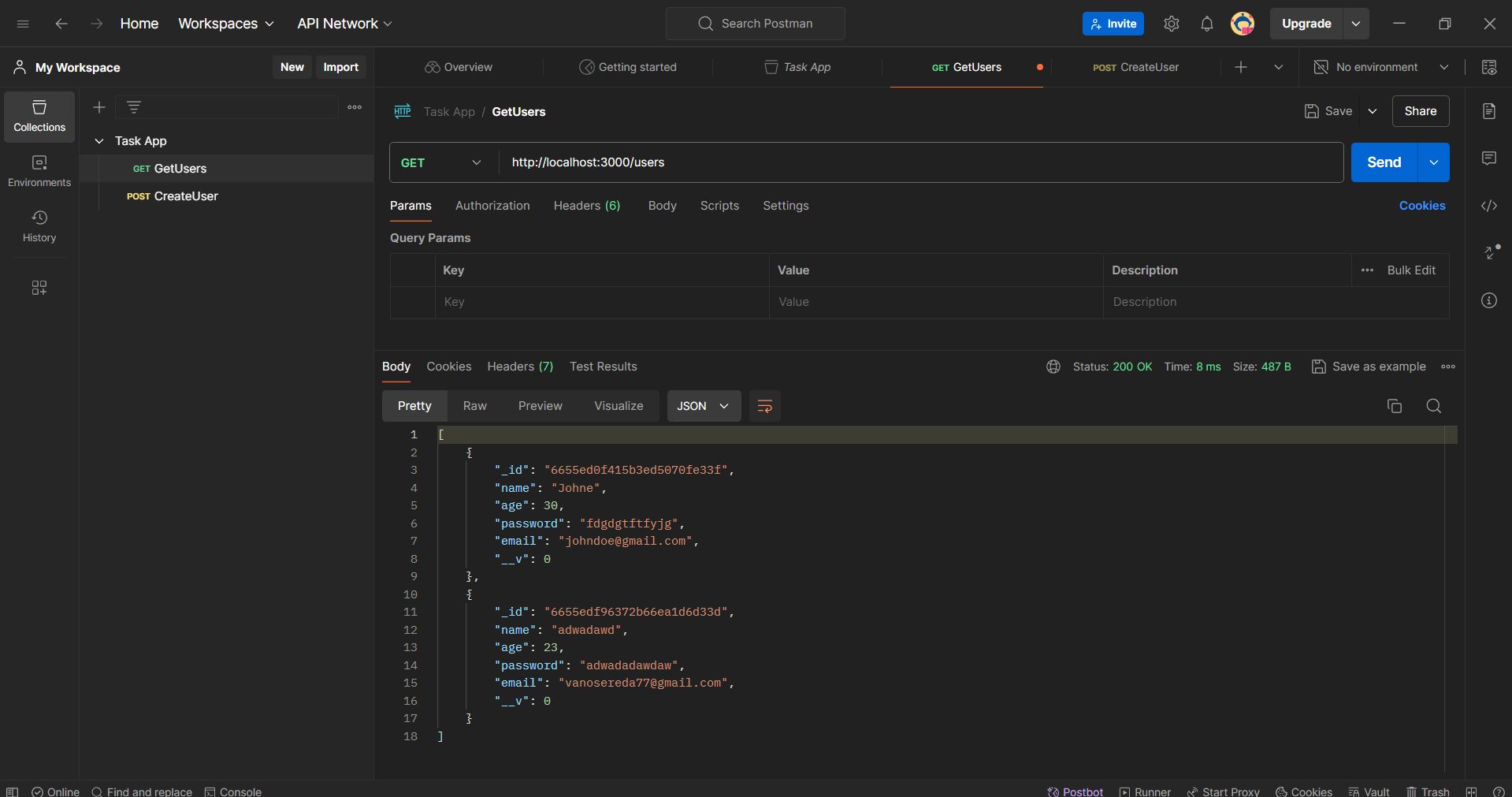
**Завдання для моделі Task**

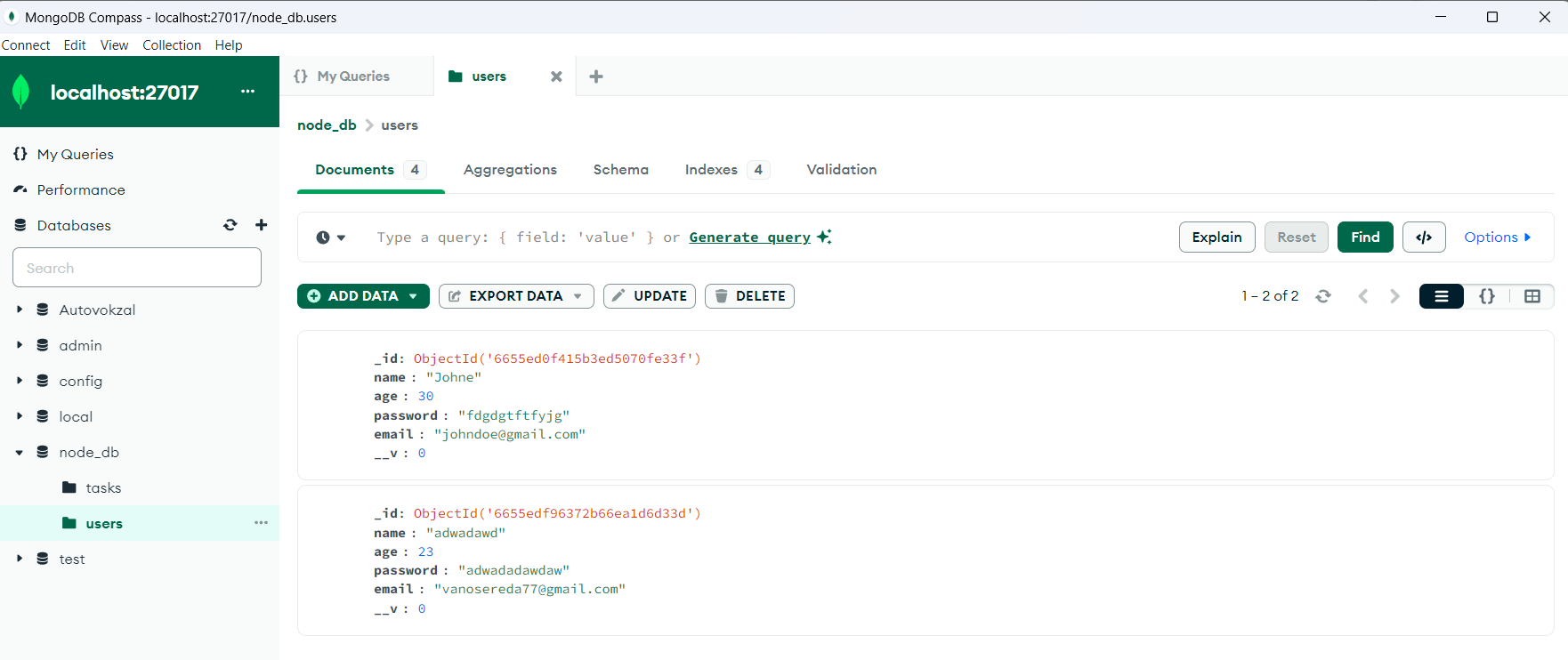
Лістинг модуля:

const mongoose = require('mongoose');  
  
const taskSchema = new mongoose.Schema({  
 title: {  
 type: String,  
 required: true,  
 trim: true  
 },  
 description: {  
 type: String,  
 required: true,  
 trim: true  
 },  
 completed: {  
 type: Boolean,  
 default: false  
 }  
});  
  
const *Task* = mongoose.model('Task', taskSchema);  
  
*module*.exports = *Task*;

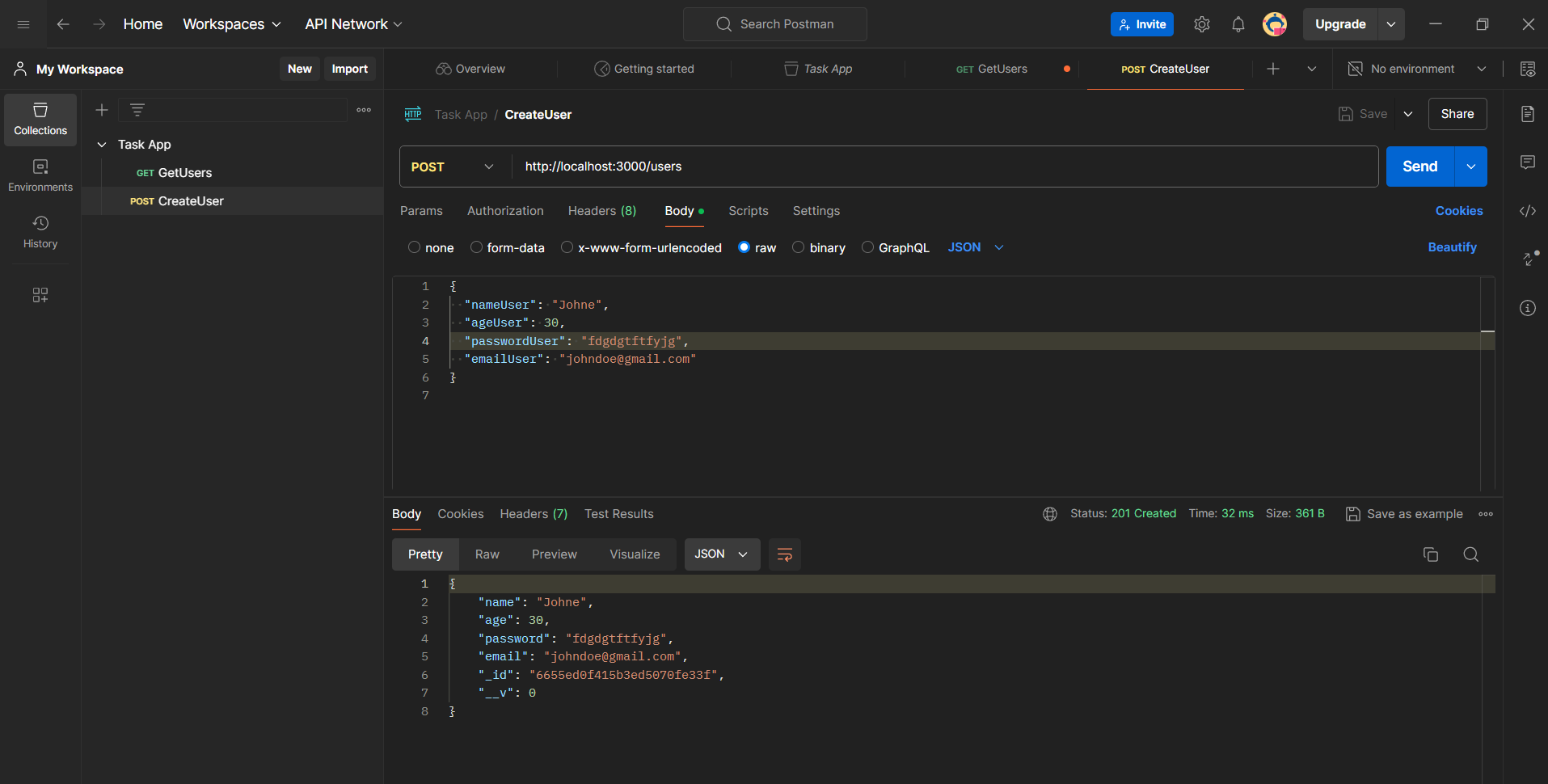
POSTMAN:

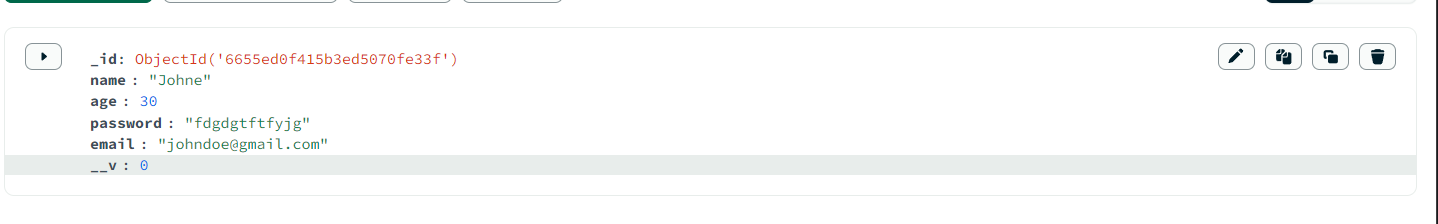
Get:





Post:





**Завдання. Створіть обробку запитів**

**Здійсніть рефакторинг коду в додатку Task Application з використанням async/await**

**Створіть обробку запиту видалення по id для моделей User і Task**

**Separate Route Files (маршрутизатори)**

Лістинг index.js :

require('dotenv').config();  
const express = require('express');  
const mongoose = require('mongoose');  
const path = require('path');  
const *User* = require('./src/models/user');  
const *Task* = require('./src/models/task');  
const *userRouter* = require('./src/routers/userRouter');  
const *taskRouter* = require('./src/routers/taskRouter');  
  
const app = express();  
const port = *process*.env.PORT  
const url = *process*.env.MONGODB\_URI ;  
  
app.use(express.*json*());  
app.use(express.*static*(path.join(*\_\_dirname*, 'src', 'client')));  
app.use(*userRouter*);  
app.use(*taskRouter*);  
  
app.get('/', (req, res) => {  
 res.sendFile(path.join(*\_\_dirname*, 'src', 'client', 'html','index.html'));  
});  
  
mongoose.connect(url, { useNewUrlParser: true, useUnifiedTopology: true })  
 .then(() => {  
 *console*.log('Connected to MongoDB');  
  
 })  
 .catch((error) => {  
 *console*.error('Error connecting to MongoDB:', error);  
 *process*.exit(1);  
 });  
  
  
app.listen(port, () => {  
 *console*.log(`Server is running on port ${port}`);  
});

Routers

Лістинг taskRouter.js :

const express = require('express');  
const *router* = new express.Router();  
const *Task* = require('../models/task');  
  
// Create new task  
*router*.post('/tasks', async (req, res) => {  
 const { title, description ,completed } = req.body;  
 try {  
 const task = new *Task*({  
 title: title,  
 description: description,  
 completed:completed  
 });  
 *console*.log(req.body);  
 await task.save();  
 res.status(201).send(task);  
 *console*.log("Task saved successfully");  
 } catch (error) {  
 *console*.error("Error saving user:", error.message);  
 res.status(400).send(error.message);  
 }  
});  
  
  
// Get all task  
*router*.get('/tasks', async (req, res) => {  
 try {  
 const tasks = await *Task*.find({});  
 res.send(tasks);  
 } catch (error) {  
 res.status(500).send(error.message);  
 }  
});  
  
// Get task by ID  
*router*.get('/tasks/:id', async (req, res) => {  
 const { id } = req.params;  
 try {  
 const task = await *Task*.findById(id);  
 if (!task) return res.status(404).send('Task not found');  
 res.send(task);  
 } catch (error) {  
 res.status(500).send(error.message);  
 }  
});  
  
// Update task by ID  
*router*.patch('/tasks/:id', async (req, res) => {  
 const { id } = req.params;  
 const updates = req.body;  
 try {  
 const task = await *Task*.findByIdAndUpdate(id, updates, { new: true, runValidators: true });  
 if (!task) return res.status(404).send('Task not found');  
 res.send(task);  
 } catch (error) {  
 res.status(400).send(error.message);  
 }  
});  
  
// Delete task by ID  
*router*.delete('/tasks/:id', async (req, res) => {  
 const { id } = req.params;  
 try {  
 const task = await *Task*.findByIdAndDelete(id);  
 if (!task) return res.status(404).send('Task not found');  
 res.send(task);  
 } catch (error) {  
 res.status(500).send(error.message);  
 }  
});  
  
// Delete all task  
*router*.delete('/tasks', async (req, res) => {  
 try {  
 await *Task*.deleteMany({});  
 res.send('All task deleted');  
 } catch (error) {  
 res.status(500).send(error.message);  
 }  
});  
  
*module*.exports = *router*;

Лістинг userRouter.js:

const express = require('express');  
const *router* = new express.Router();  
const *User* = require('../models/user');  
  
// Create new user  
*router*.post('/users', async (req, res) => {  
 const { nameUser, ageUser, passwordUser, emailUser } = req.body;  
 try {  
 const user = new *User*({  
 name: nameUser,  
 age: ageUser,  
 password: passwordUser,  
 email: emailUser  
 });  
 *console*.log(req.body);  
 await user.save();  
 res.status(201).send(user);  
 *console*.log("User saved successfully");  
 } catch (error) {  
 *console*.error("Error saving user:", error.message);  
 res.status(400).send(error.message);  
 }  
});  
  
  
// Get all users  
*router*.get('/users', async (req, res) => {  
 try {  
 const users = await *User*.find({});  
 res.send(users);  
 } catch (error) {  
 res.status(500).send(error.message);  
 }  
});  
  
// Get user by ID  
*router*.get('/users/:id', async (req, res) => {  
 const { id } = req.params;  
 try {  
 const user = await User.findById(id);  
 if (!user) return res.status(404).send('User not found');  
 res.send(user);  
 } catch (error) {  
 res.status(500).send(error.message);  
 }  
});  
  
// Update user by ID  
router.patch('/users/:id', async (req, res) => {  
 const { id } = req.params;  
 const updates = req.body;  
 try {  
 const user = await User.findByIdAndUpdate(id, updates, { new: true, runValidators: true });  
 if (!user) return res.status(404).send('User not found');  
 res.send(user);  
 } catch (error) {  
 res.status(400).send(error.message);  
 }  
});  
  
// Delete user by ID  
router.delete('/users/:id', async (req, res) => {  
 const { id } = req.params;  
 try {  
 const user = await User.findByIdAndDelete(id);  
 if (!user) return res.status(404).send('User not found');  
 res.send(user);  
 } catch (error) {  
 res.status(500).send(error.message);  
 }  
});  
  
// Delete all users  
router.delete('/users', async (req, res) => {  
 try {  
 await User.deleteMany({});  
 res.send('All users deleted');  
 } catch (error) {  
 res.status(500).send(error.message);  
 }  
});  
  
module.exports = router;

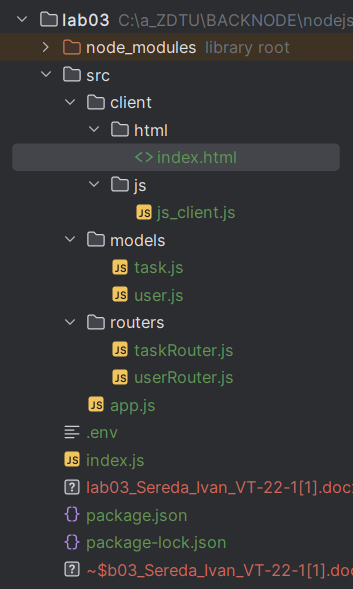
Лістинг js\_client.js:

*document*.getElementById('get\_users\_button').addEventListener('click', async () => {  
 try {  
 const response = await fetch('/users', { method: 'GET' });  
 if (!response.ok) throw new Error(await response.text());  
 const users = await response.json();  
 *console*.log(users);  
 let table = *document*.getElementById('users\_table');  
 for (let i = 0; i < users.length; i++) {  
 let tr = *document*.createElement('tr');  
 let tdName = *document*.createElement('td');  
 let tdEmail = *document*.createElement('td');  
 let tdPass = *document*.createElement('td');  
 let tdAge = *document*.createElement('td');  
 tdName.innerText = users[i].name;  
 tdEmail.innerText = users[i].*email*;  
 tdPass.innerText = users[i].password;  
 tdAge.innerText = users[i].age;  
 tr.appendChild(tdName);  
 tr.appendChild(tdEmail);  
 tr.appendChild(tdPass);  
 tr.appendChild(tdAge);  
 table.appendChild(tr);  
 }  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
  
// Get user by ID  
*document*.getElementById('get\_id\_user\_button').addEventListener('click', async () => {  
 const userId = *document*.getElementById('get\_id\_user').value;  
 try {  
 const response = await fetch(`/users/${userId}`, { method: 'GET' });  
 if (!response.ok) throw new Error(await response.text());  
 const user = await response.json();  
 *console*.log(user);  
 let table = *document*.getElementById('user\_table');  
  
 let tr = *document*.createElement('tr');  
 let tdName = *document*.createElement('td');  
 let tdEmail = *document*.createElement('td');  
 let tdPass = *document*.createElement('td');  
 let tdAge = *document*.createElement('td');  
 tdName.innerText = user.name;  
 tdEmail.innerText = user.*email*;  
 tdPass.innerText = user.password;  
 tdAge.innerText = user.age;  
 tr.appendChild(tdName);  
 tr.appendChild(tdEmail);  
 tr.appendChild(tdPass);  
 tr.appendChild(tdAge);  
 table.appendChild(tr);  
  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
  
// Post new user  
*document*.getElementById('post\_user\_form').addEventListener('submit', postFormUser);  
  
async function postFormUser(e) {  
 e.preventDefault();  
  
 const formData = new FormData(e.target);  
  
 const jsonData = {};  
 formData.forEach((value, key) => {  
 jsonData[key] = value;  
 });  
  
 *console*.log('Form Data:', jsonData);  
  
 try {  
 const response = await fetch('/users', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: *JSON*.stringify(jsonData)  
 });  
  
  
  
 if (!response.ok) throw new Error(await response.text());  
  
 const responseData = await response.json();  
 *console*.log(responseData);  
 alert("User added successfully");  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
}  
  
  
// Patch user by ID  
*document*.getElementById('patch\_user\_button').addEventListener('click', async () => {  
 const userId = *document*.getElementById('patch\_user').value;  
  
 try {  
 const response = await fetch(`/users/${userId}`, { method: 'GET' });  
 if (!response.ok) throw new Error(await response.text());  
  
 const user = await response.json();  
  
 *document*.getElementById('patch\_name').value = user.name;  
 *document*.getElementById('patch\_age\_user').value = user.age;  
 *document*.getElementById('patch\_password\_user').value = user.password;  
 *document*.getElementById('patch\_email\_user').value = user.*email*;  
  
 *document*.getElementById('patch\_user\_form').style.display = 'block';  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
*document*.getElementById('patch\_user\_form').addEventListener('submit', async (e) => {  
 e.preventDefault();  
 const userId = *document*.getElementById('patch\_user').value;  
  
 const patchData = {  
 name: *document*.getElementById('patch\_name').value,  
 age: *document*.getElementById('patch\_age\_user').value,  
 password: *document*.getElementById('patch\_password\_user').value,  
 email: *document*.getElementById('patch\_email\_user').value  
 };  
  
 try {  
 const response = await fetch(`/users/${userId}`, {  
 method: 'PATCH',  
 headers: { 'Content-Type': 'application/json' },  
 body: *JSON*.stringify(patchData)  
 });  
  
 if (!response.ok) throw new Error(await response.text());  
  
 const responseData = await response.json();  
 *console*.log(responseData);  
 alert("User updated successfully");  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
// Delete user by ID  
*document*.getElementById('delete\_user\_by\_id\_button').addEventListener('click', async () => {  
 const userId = *document*.getElementById('delete\_user\_by\_id').value;  
  
 try {  
 const response = await fetch(`/users/${userId}`, { method: 'DELETE' });  
 if (!response.ok) throw new Error(await response.text());  
 alert("User deleted successfully");  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
// Delete all users  
*document*.getElementById('delete\_users\_button').addEventListener('click', async () => {  
 try {  
 const response = await fetch('/users', { method: 'DELETE' });  
 if (!response.ok) throw new Error(await response.text());  
 alert("All users deleted successfully");  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
  
  
//#################################################################################  
  
*document*.getElementById('get\_tasks\_button').addEventListener('click', async () => {  
 try {  
 const response = await fetch('/tasks', { method: 'GET' });  
 if (!response.ok) throw new Error(await response.text());  
 const tasks = await response.json();  
 *console*.log(tasks);  
 let table = *document*.getElementById('tasks\_table');  
 for (let i = 0; i < tasks.length; i++) {  
 let tr = *document*.createElement('tr');  
 let tdTitle = *document*.createElement('td');  
 let tdDescription = *document*.createElement('td');  
 let tdCompleted = *document*.createElement('td');  
  
 tdTitle.innerText = tasks[i].title;  
 tdDescription.innerText = tasks[i].description;  
 tdCompleted.innerText = tasks[i].completed?'Yes':"No";  
  
 tr.appendChild(tdTitle);  
 tr.appendChild(tdDescription);  
 tr.appendChild(tdCompleted);  
  
 table.appendChild(tr);  
 }  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
  
// Get task by ID  
// Get task by ID  
*document*.getElementById('get\_id\_task\_button').addEventListener('click', async () => {  
 const taskId = *document*.getElementById('get\_id\_task').value;  
 try {  
 const response = await fetch(`/tasks/${taskId}`, { method: 'GET' });  
 if (!response.ok) throw new Error(await response.text());  
 const task = await response.json();  
 *console*.log(task);  
 let table = *document*.getElementById('task\_table');  
  
 let tr = *document*.createElement('tr');  
 let tdTitle = *document*.createElement('td');  
 let tdDescription = *document*.createElement('td');  
 let tdCompleted = *document*.createElement('td');  
  
 tdTitle.innerText = task.title;  
 tdDescription.innerText = task.description;  
 tdCompleted.innerText = task.completed ? 'Yes' : "No";  
  
 tr.appendChild(tdTitle);  
 tr.appendChild(tdDescription);  
 tr.appendChild(tdCompleted);  
  
 table.appendChild(tr);  
  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
  
  
// Post new task  
*document*.getElementById('post\_task\_form').addEventListener('submit', postFormTask);  
  
async function postFormTask(e) {  
 e.preventDefault();  
  
 const formData = new FormData(e.target);  
  
 const jsonData = {};  
 formData.forEach((value, key) => {  
 jsonData[key] = value;  
 });  
  
 jsonData.completed = e.target.querySelector('#post\_comleted\_task').checked;  
  
 *console*.log('Form Data:', jsonData);  
  
 try {  
 const response = await fetch('/tasks', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: *JSON*.stringify(jsonData)  
 });  
  
 if (!response.ok) throw new Error(await response.text());  
  
 const responseData = await response.json();  
 *console*.log(responseData);  
 alert("Task added successfully");  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
}  
  
// Patch user by ID  
*document*.getElementById('patch\_task\_button').addEventListener('click', async () => {  
 const taskId = *document*.getElementById('patch\_task').value;  
  
 try {  
 const response = await fetch(`/tasks/${taskId}`, { method: 'GET' });  
 if (!response.ok) throw new Error(await response.text());  
  
 const task = await response.json();  
  
 *document*.getElementById('patch\_desciption').value = task.title;  
 *document*.getElementById('patch\_title').value = task.description;  
  
 *document*.getElementById('patch\_task\_form').style.display = 'block';  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
*document*.getElementById('patch\_task\_form').addEventListener('submit', async (e) => {  
 e.preventDefault();  
 const taskId = *document*.getElementById('patch\_task').value;  
  
 const patchData = {  
 title: *document*.getElementById('patch\_title').value,  
 description: *document*.getElementById('patch\_desciption').value,  
 completed: true  
 };  
  
 try {  
 const response = await fetch(`/tasks/${taskId}`, {  
 method: 'PATCH',  
 headers: { 'Content-Type': 'application/json' },  
 body: *JSON*.stringify(patchData)  
 });  
  
 if (!response.ok) throw new Error(await response.text());  
  
 const responseData = await response.json();  
 *console*.log(responseData);  
 alert("Task updated successfully");  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
  
// Delete task by ID  
*document*.getElementById('delete\_task\_by\_id\_button').addEventListener('click', async () => {  
 const taskId = *document*.getElementById('delete\_task\_by\_id').value;  
  
 try {  
 const response = await fetch(`/tasks/${taskId}`, { method: 'DELETE' });  
 if (!response.ok) throw new Error(await response.text());  
 alert("Task deleted successfully");  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});  
  
  
// Delete all users  
*document*.getElementById('delete\_tasks\_button').addEventListener('click', async () => {  
 try {  
 const response = await fetch('/tasks', { method: 'DELETE' });  
 if (!response.ok) throw new Error(await response.text());  
 alert("All task deleted successfully");  
 } catch (error) {  
 *console*.error('Error:', error.message);  
 }  
});

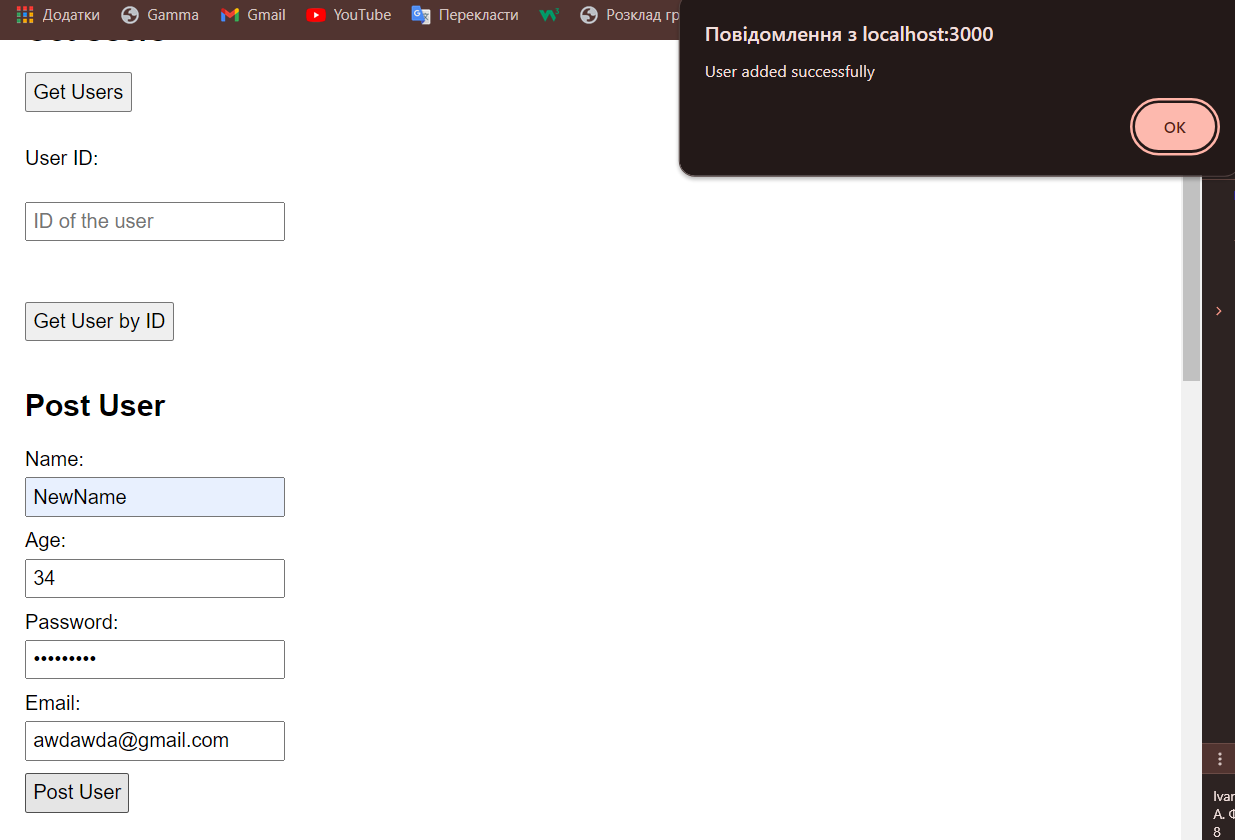
Лістинг index.html:

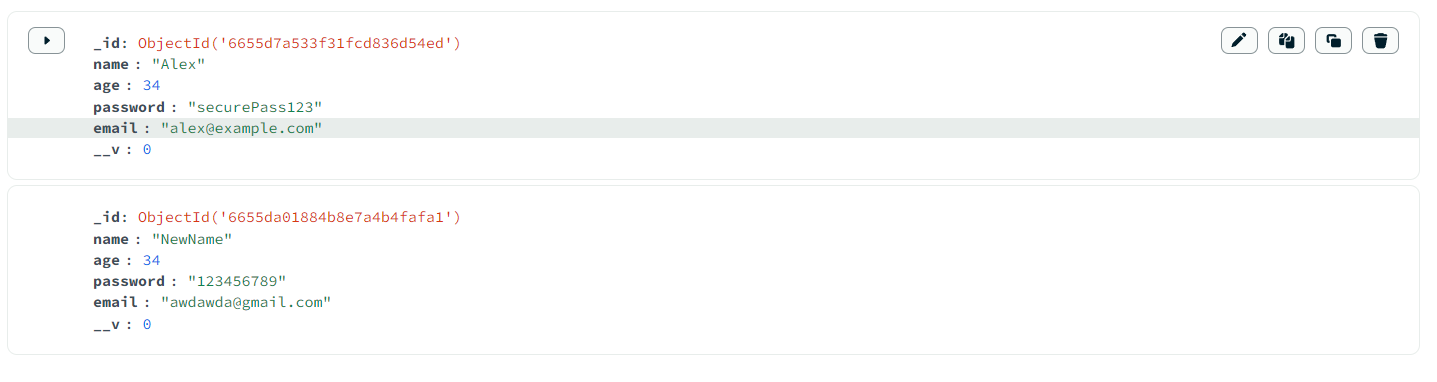
<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Title</title>  
 <style>  
 body {  
 font-family: Arial, sans-serif;  
 margin: 20px;  
 }  
 form {  
 margin-bottom: 20px;  
 }  
 label {  
 display: block;  
 margin-top: 10px;  
 }  
 input, button {  
 display: block;  
 margin-top: 5px;  
 padding: 5px;  
 font-size: 16px;  
 }  
 button {  
 margin-top: 10px;  
 cursor: pointer;  
 }  
  
 table {  
 width: 100%;  
 border-collapse: collapse;  
 margin-top: 20px;  
 }  
  
 th, td {  
 padding: 8px;  
 border: 1px solid #ddd;  
 text-align: left;  
 }  
  
 th {  
 background-color: #f2f2f2;  
 color: #333;  
 }  
  
 tr:nth-child(even) {  
 background-color: #f9f9f9;  
 }  
  
 tr:hover {  
 background-color: #e2e2e2;  
 }  
 </style>  
</head>  
<body>  
<h1>User Management</h1>  
<script defer src="../js/js\_client.js"></script>  
  
<section>  
 <h2>Get Users</h2>  
 <button id="get\_users\_button">Get Users</button><br>  
 <label for="get\_id\_user">User ID:</label>  
 <table id="users\_table"></table>  
 <input id="get\_id\_user" placeholder="ID of the user"><br>  
 <table id="user\_table"></table>  
 <button id="get\_id\_user\_button">Get User by ID</button><br>  
</section>  
  
<section>  
 <h2>Post User</h2>  
 <form id="post\_user\_form">  
 <label for="post\_name\_user">Name:</label>  
 <input name="nameUser" id="post\_name\_user" placeholder="Name">  
  
 <label for="post\_age\_user">Age:</label>  
 <input type="number" name="ageUser" id="post\_age\_user" placeholder="Age">  
  
 <label for="post\_password\_user">Password:</label>  
 <input type="password" name="passwordUser" id="post\_password\_user" placeholder="Password">  
  
 <label for="post\_email\_user">Email:</label>  
 <input type="email" name="emailUser" id="post\_email\_user" placeholder="Email">  
  
 <button type="submit" id="post\_user\_button">Post User</button>  
 </form>  
</section>  
  
<section>  
 <h2>Patch User</h2>  
 <label for="patch\_user">User ID:</label>  
 <input id="patch\_user" placeholder="ID of the user">  
 <button id="patch\_user\_button">Patch User</button>  
  
 <form id="patch\_user\_form" style="display: none">  
 <label for="patch\_name">Name:</label>  
 <input id="patch\_name" placeholder="New Name">  
  
 <label for="patch\_age\_user">Age:</label>  
 <input type="number" id="patch\_age\_user" placeholder="New Age">  
  
 <label for="patch\_password\_user">Password:</label>  
 <input type="password" id="patch\_password\_user" placeholder="New Password">  
  
 <label for="patch\_email\_user">Email:</label>  
 <input type="email" id="patch\_email\_user" placeholder="New Email">  
  
 <button type="submit" id="patch\_user\_button\_form">Patch User</button>  
 </form>  
</section>  
  
<section>  
 <h2>Delete User</h2>  
 <label for="delete\_user\_by\_id">User ID:</label>  
 <input id="delete\_user\_by\_id" placeholder="ID of the user">  
 <button id="delete\_user\_by\_id\_button">Delete User by ID</button>  
 <button id="delete\_users\_button">Delete All Users</button>  
</section>  
  
  
  
<section>  
 <h2>Get Tasks</h2>  
 <button id="get\_tasks\_button">Get Tasks</button><br>  
 <label for="get\_id\_task">Task ID:</label>  
 <table id="tasks\_table"></table>  
 <input id="get\_id\_task" placeholder="ID of the user"><br>  
 <table id="task\_table"></table>  
 <button id="get\_id\_task\_button">Get Task by ID</button><br>  
</section>  
  
<section>  
 <h2>Post Task</h2>  
 <form id="post\_task\_form">  
 <label for="post\_title\_task">Title:</label>  
 <input name="title" id="post\_title\_task" placeholder="Title">  
  
 <label for="post\_description\_task">Desciption:</label>  
 <textarea name="description" id="post\_description\_task" placeholder="Desciption"></textarea>  
  
 <label for="post\_comleted\_task">Is Comlete:</label>  
 <input type="radio" name="completed" id="post\_comleted\_task" placeholder="">  
  
 <button type="submit" id="post\_task\_button">Post Task</button>  
 </form>  
</section>  
  
<section>  
 <h2>Patch Task</h2>  
 <label for="patch\_task">Task ID:</label>  
 <input id="patch\_task" placeholder="ID of the task">  
 <button id="patch\_task\_button">Patch Task</button>  
  
 <form id="patch\_task\_form" style="display: none">  
 <label for="patch\_title">Title:</label>  
 <input id="patch\_title" placeholder="Title">  
  
 <label for="patch\_desciption">Desciption:</label>  
 <textarea id="patch\_desciption" placeholder="Desciption"></textarea>  
  
 <label for="patch\_comleted\_task">Is Comlete:</label>  
 <input type="radio" name="completed" id="patch\_comleted\_task" placeholder="">  
  
 <button type="submit" id="patch\_task\_button\_form">Patch Task</button>  
 </form>  
</section>  
  
<section>  
 <h2>Delete Task</h2>  
 <label for="delete\_task\_by\_id">Task ID:</label>  
 <input id="delete\_task\_by\_id" placeholder="ID of the task">  
 <button id="delete\_task\_by\_id\_button">Delete Task by ID</button>  
 <button id="delete\_tasks\_button">Delete All Task</button>  
</section>  
  
</body>  
</html>

Структура файлів:

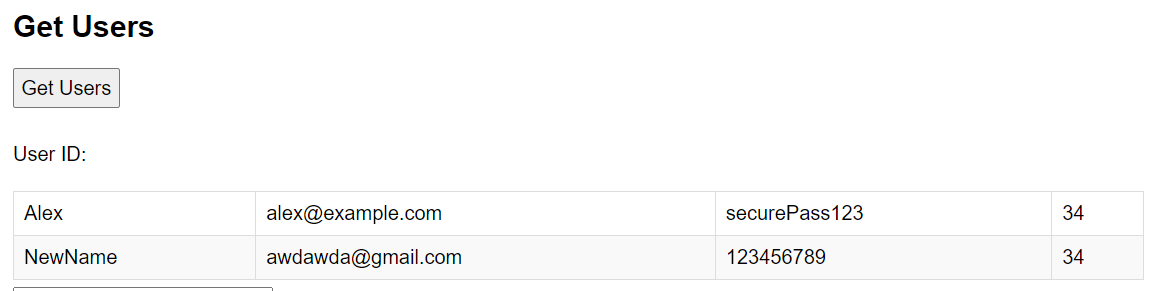


Post user:

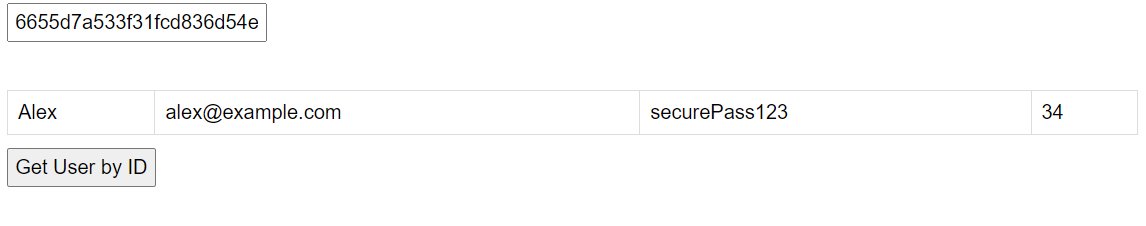




Get users:



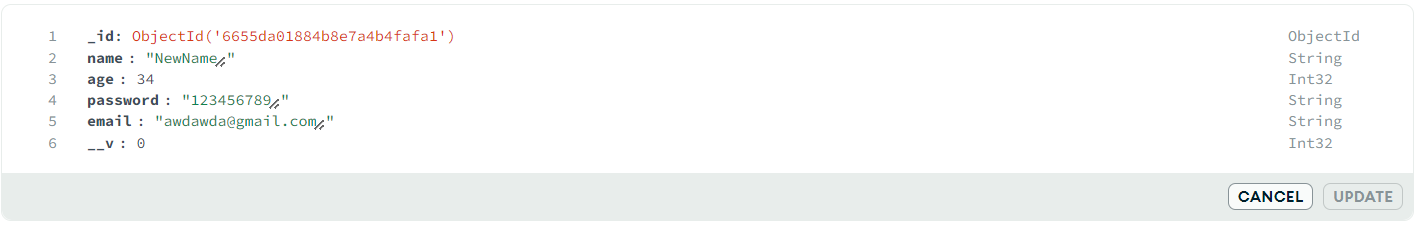
Get user by id:



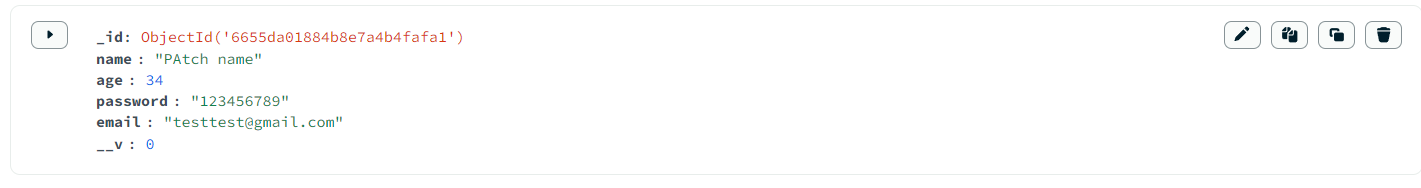
Patch user:



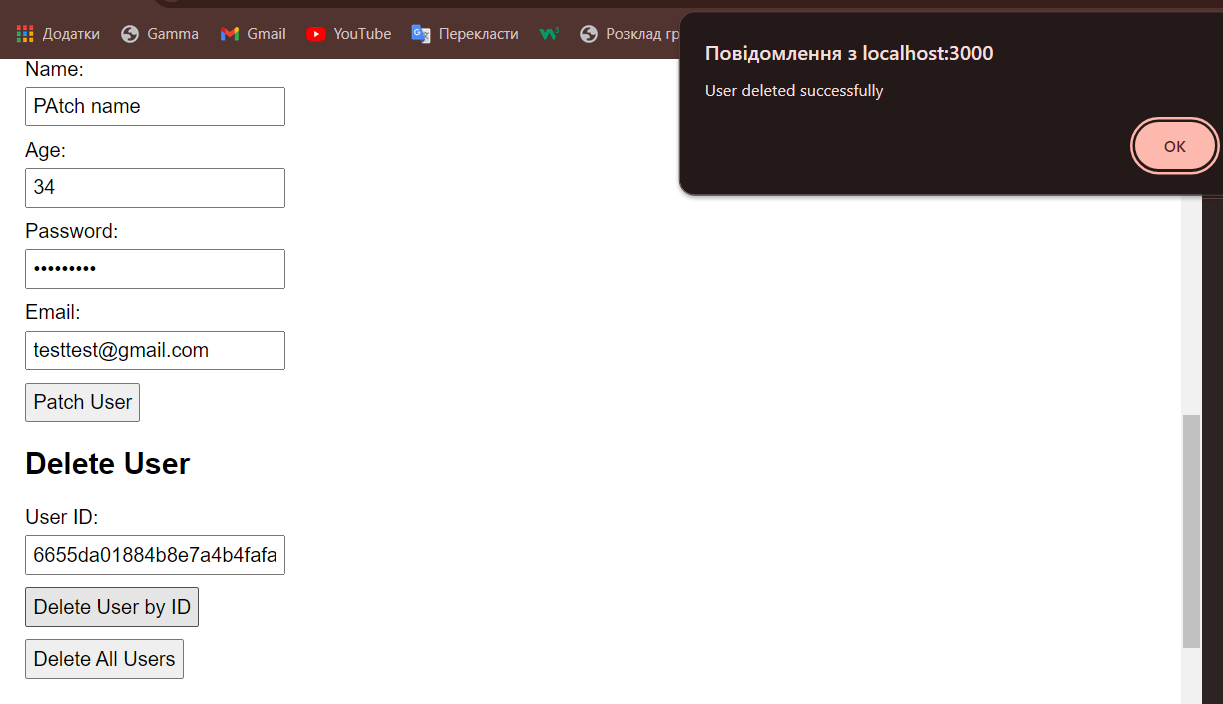
До :

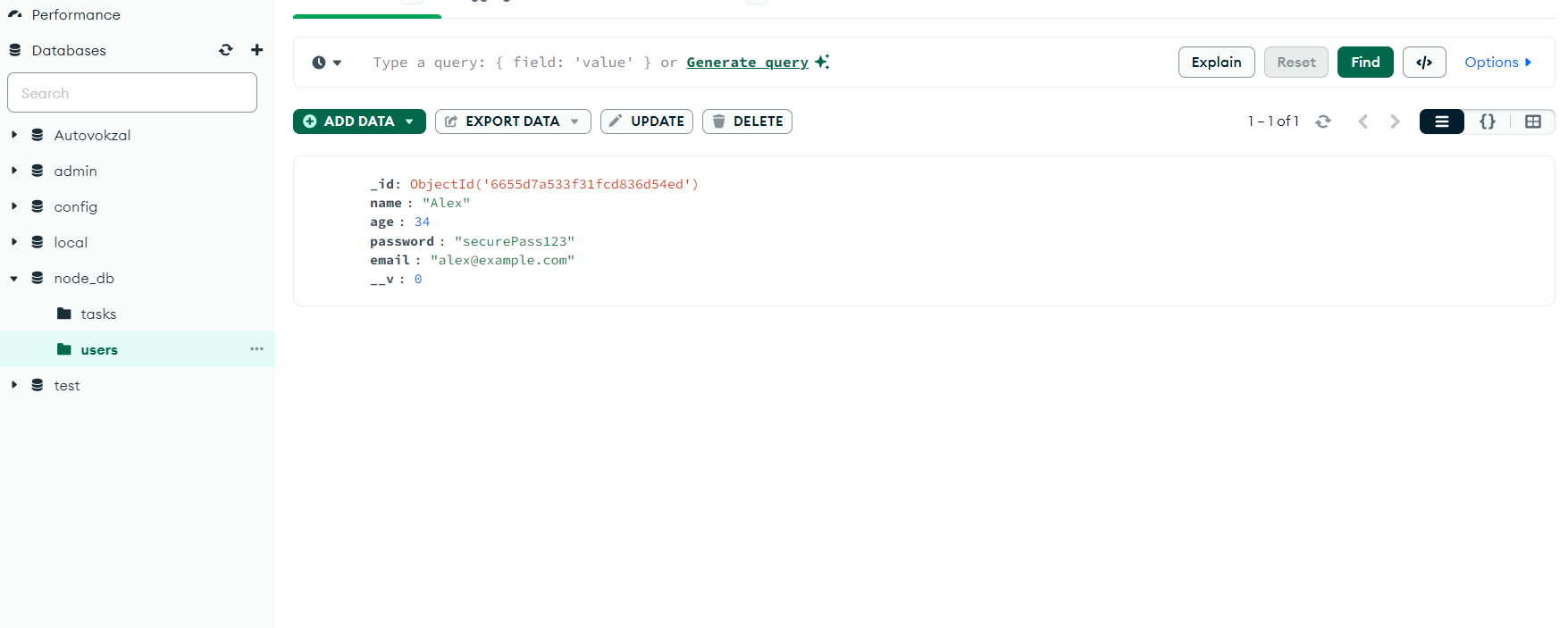


Після :

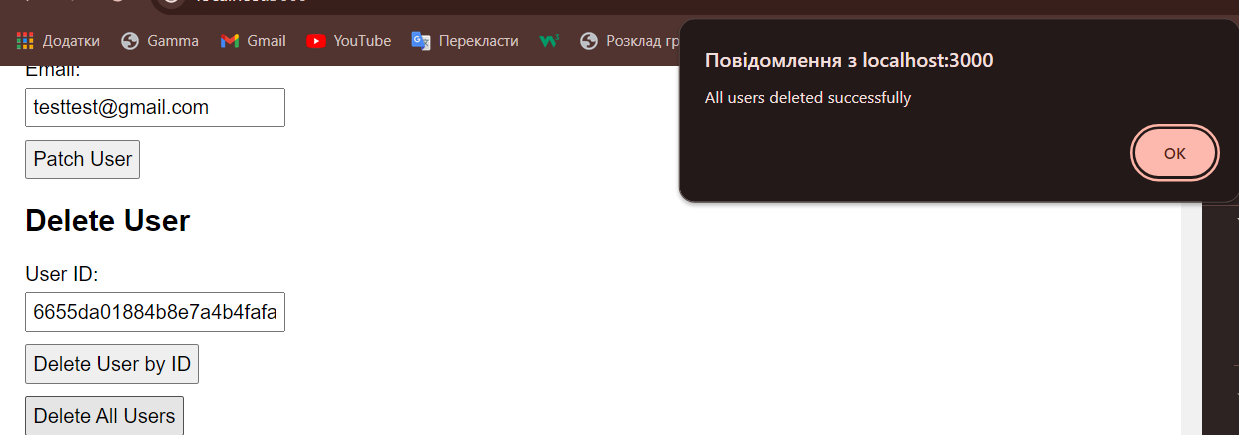


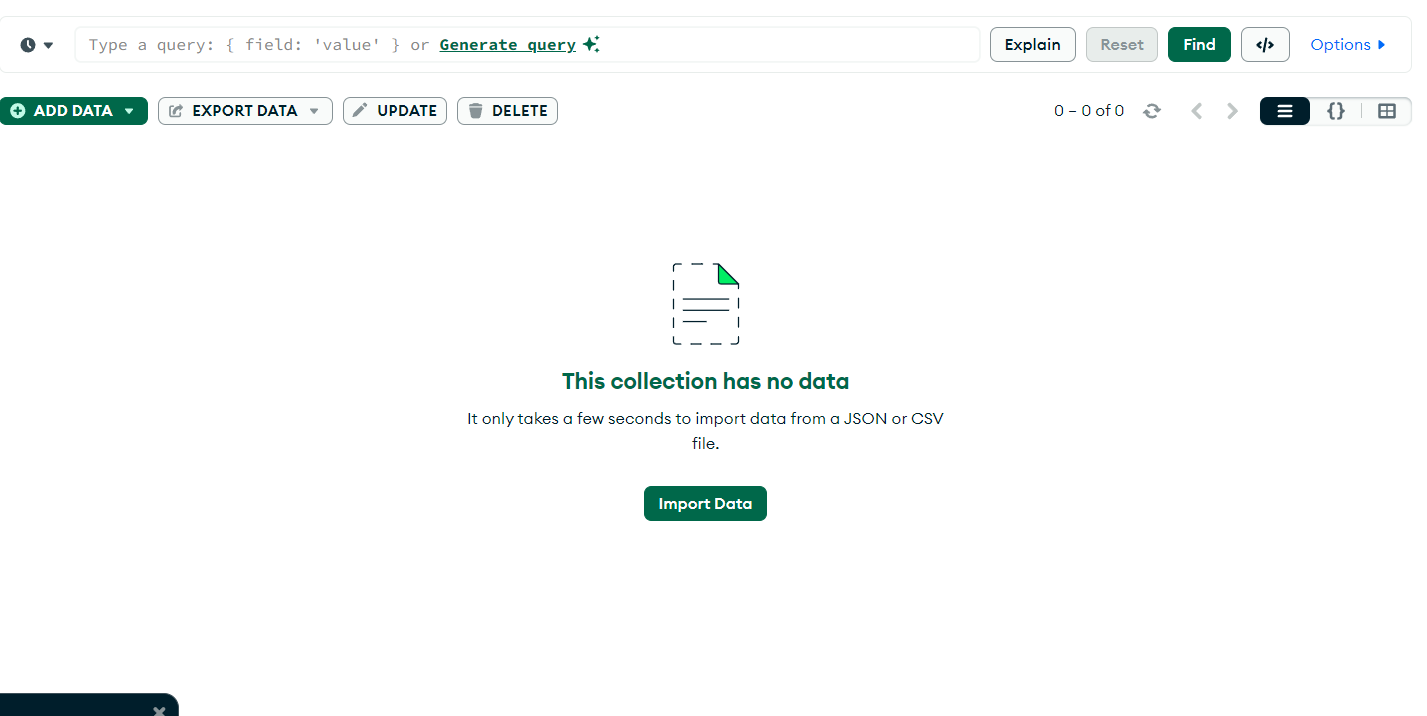
Delete user by id :

Того самого user видаляємо 



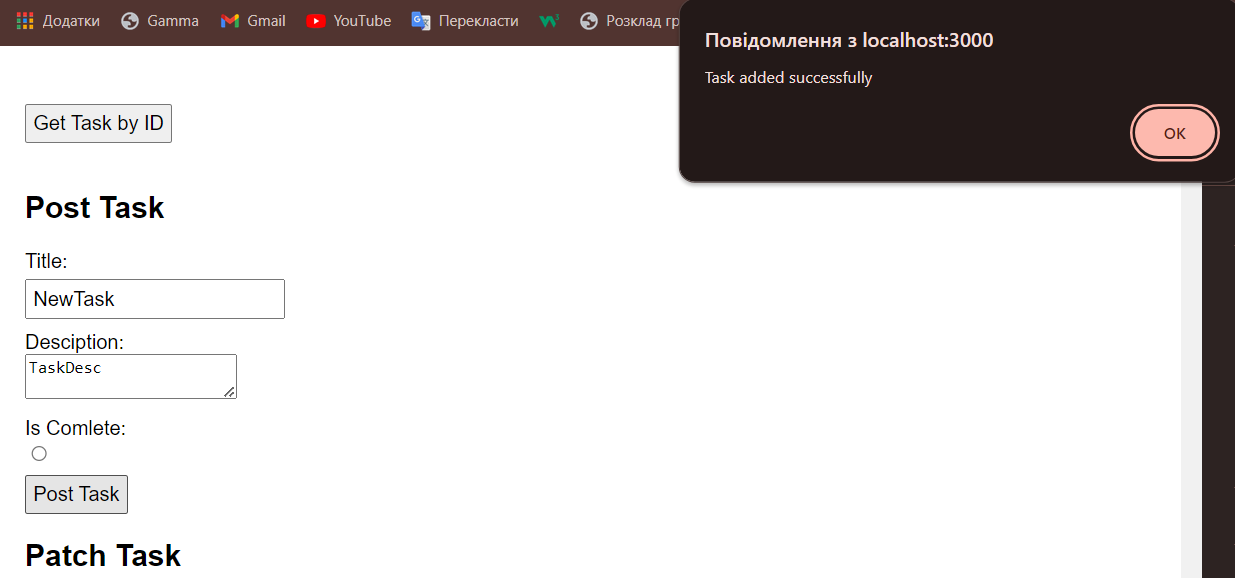
Delete all users:

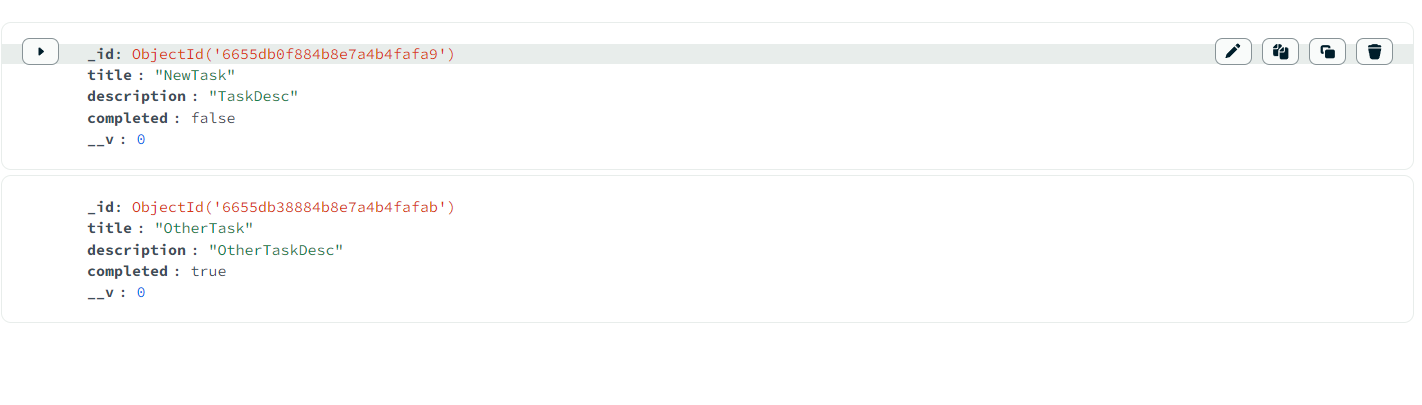




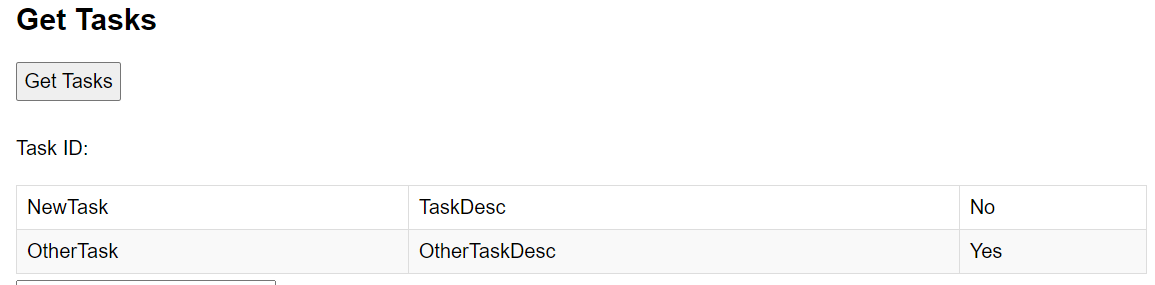
Task CRUD:

Post task:

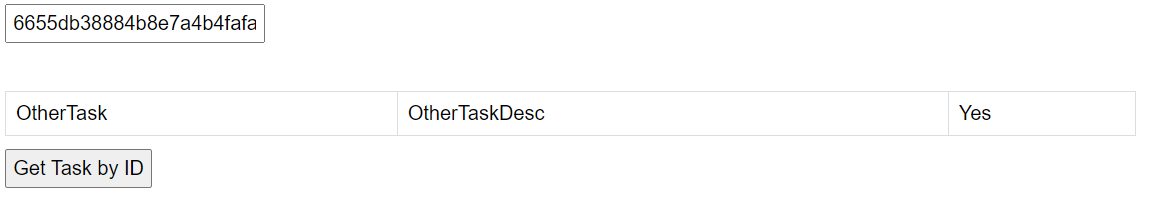




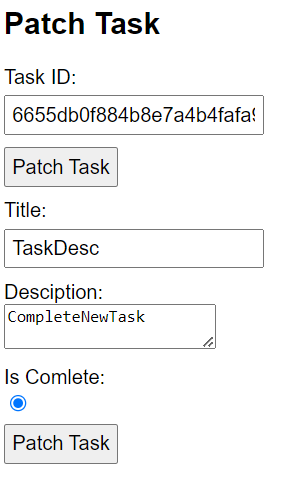
Get tasks:

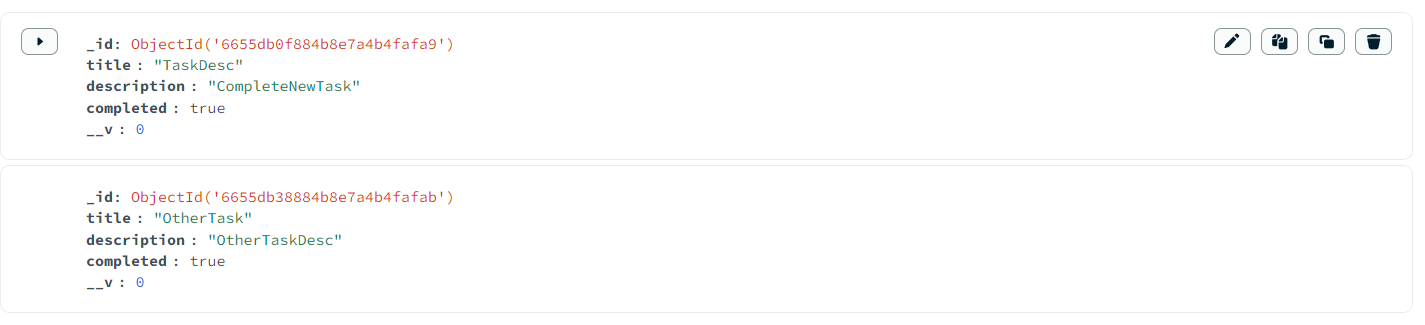


Get task by id:



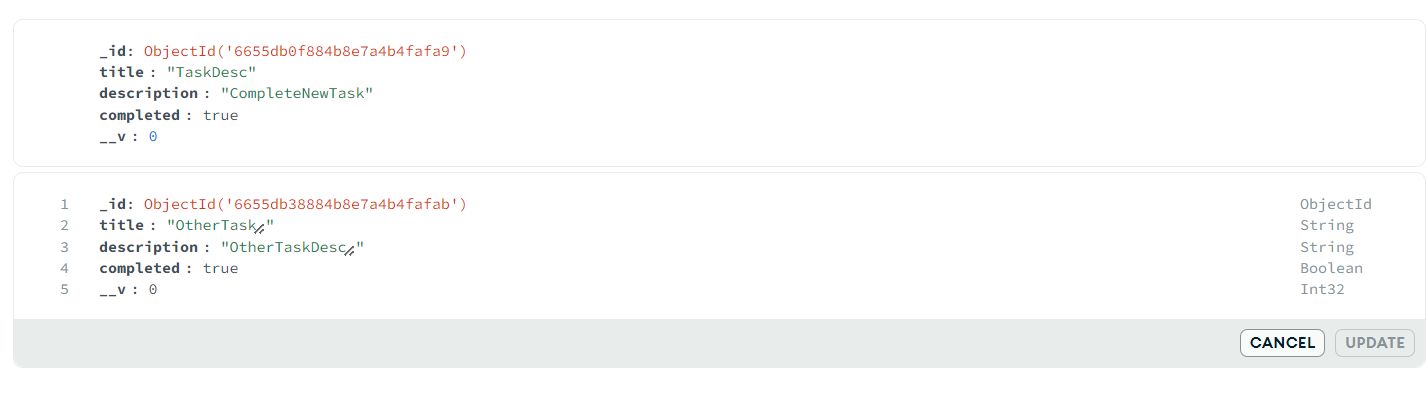
Patch Task:



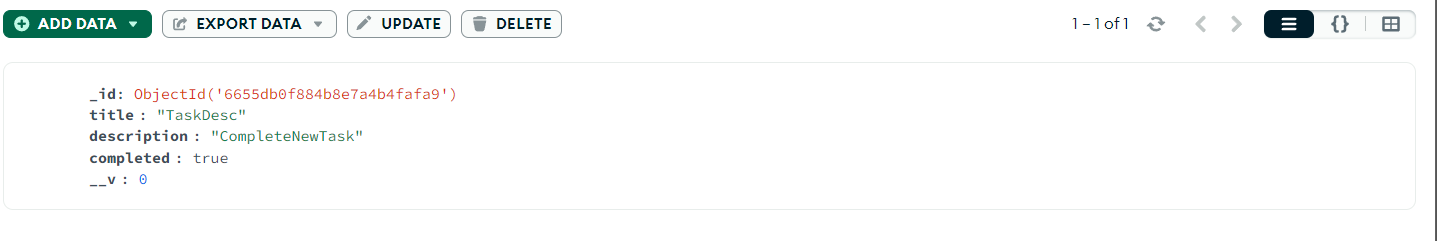


Delete task by id :

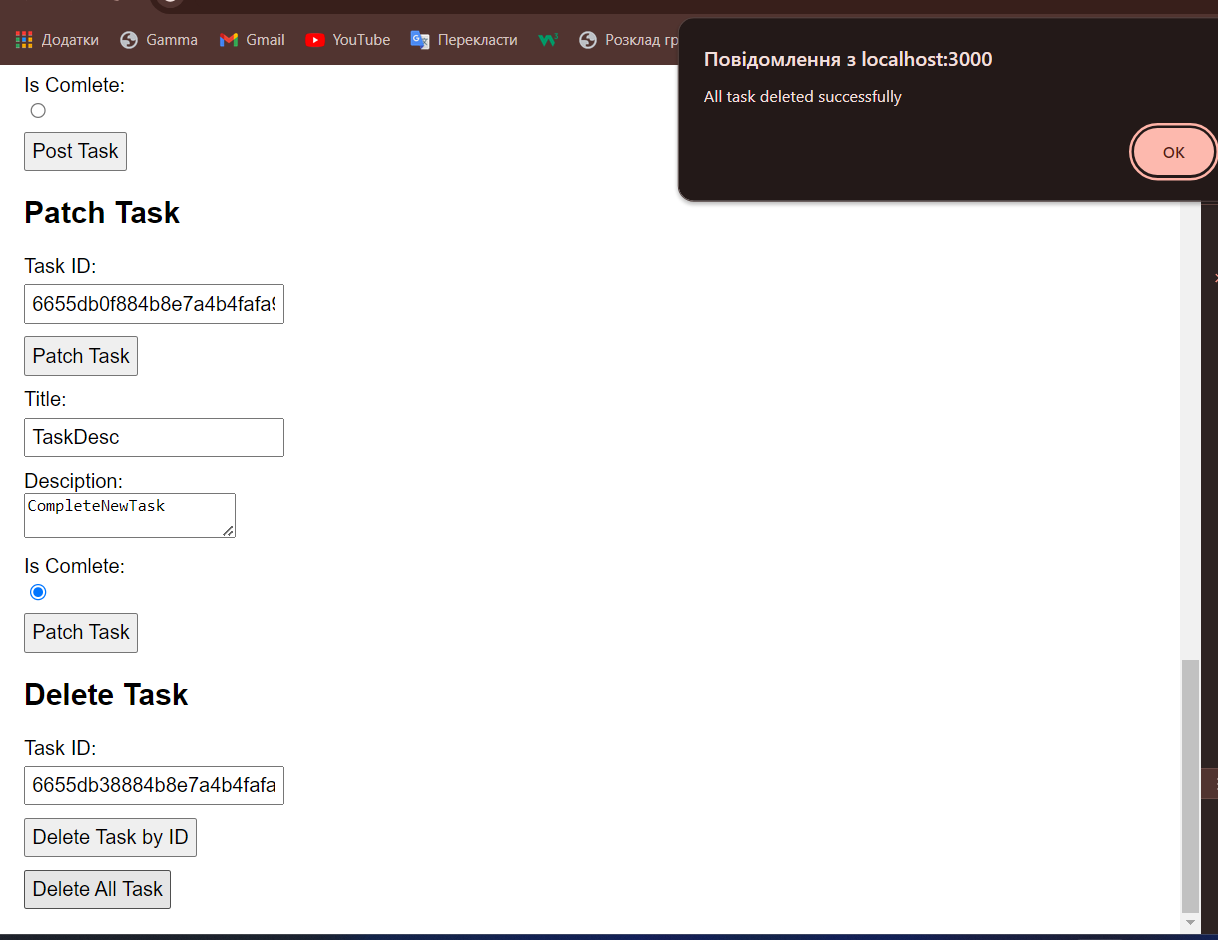
  
До :

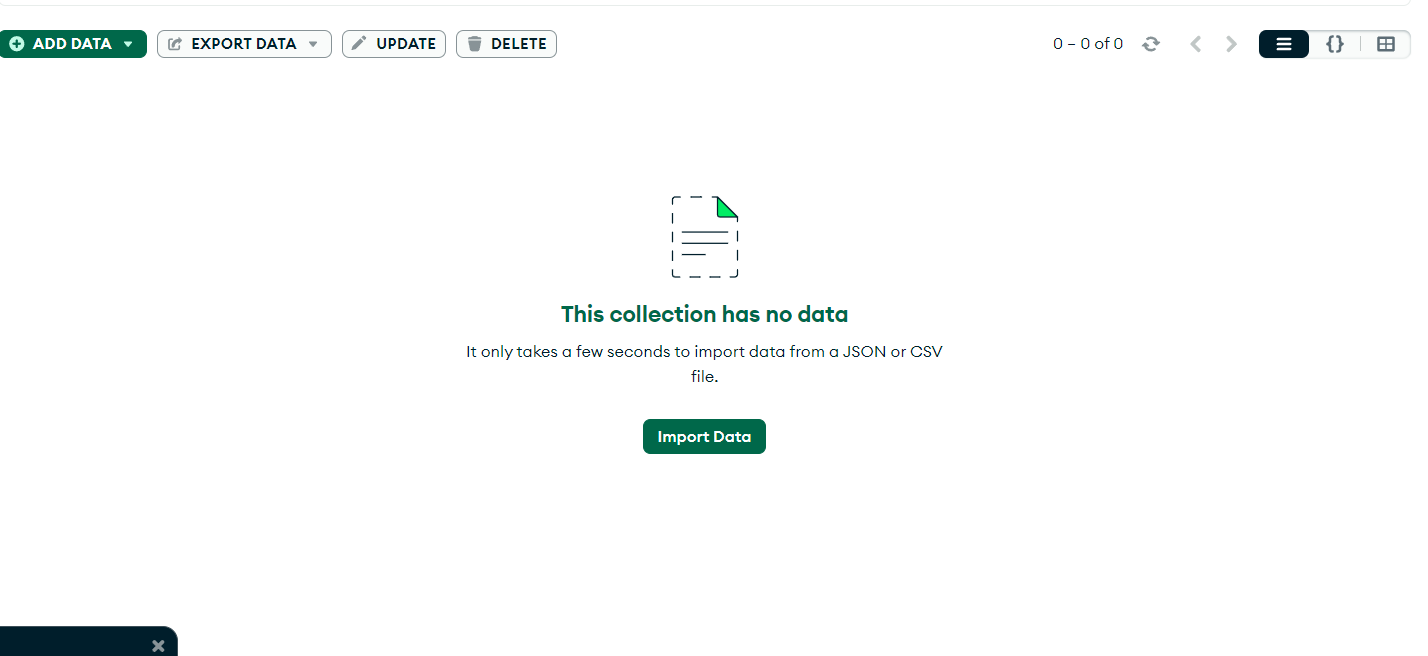


Після :



Delete all task:





Висновок : на цій лабараторній роботі я навчився працювати з бібліотеками mongoose, express, dotenv, а також писати CRUD операціїдля Rest API.