МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА”

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

ЕЛЕКТРОННИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

**ЗВІТ**

про виконання проектно-технологічної практики

Виконав:

студентка гр. КІ-213 Алефіренко М.Ю.

Перевірив:

к.т.н., доцент Бивойно Т.П.

Чернігів 2024

ЗМІСТ

[1 ЛАБОРАТОРНА РОБОТА № 1. 3](#_Toc170488530)

[2 ЛАБОРАТОРНА РОБОТА № 2. 7](#_Toc170488531)

[3 ЛАБОРАТОРНА РОБОТА № 3. 12](#_Toc170488532)

[4 ЛАБОРАТОРНА РОБОТА № 4. 20](#_Toc170488533)

[5 ЛАБОРАТОРНА РОБОТА № 5. 23](#_Toc170488534)

[ВИСНОВОК 28](#_Toc170488535)

[ПОСИЛАННЯ НА GITHUB 28](#_Toc170488536)

# 1 ЛАБОРАТОРНА РОБОТА № 1.

**ПОЧАТОК РОБОТИ З NODE.JS. ПАКЕТНИЙ МЕНЕДЖЕР NPM. ФРЕЙМВОРК EXPRESS.**

**ПАКЕТИ ДЛЯ ТЕСТУВАННЯ JEST ТА SUPERTEST**

**1.1 Мета роботи**

Метою даної лабораторної роботи є ознайомлення з основами роботи в середовищі Node.js, використання пакетного менеджера npm для керування залежностями проекту, створення веб-додатків за допомогою фреймворку Express та проведення тестування веб-сервісів з використанням бібліотек Jest та Supertest.

**1.2 Завдання на роботу**

* Встановити Node.js
* Ініціювати проект, вказати автора і стартовий файл
* Додати в проект Express, Jest, supertest, dotenv
* Програма повинна слухати порт 3000
* Для адреси ”/” має повертатись Hello World
* Запустити програму, побачити в браузері Hello World
* Написати тест на Jest та supertest, який перевіряє чи дійсно сервіс повертає Hello World
* За допомогою dotenv змінити текст відповіді (як завгодно)

**1.3 Результати виконання роботи**

На рисунку 1.1 зображено результат встановлення Node.js.

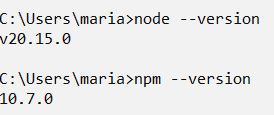


Рисунок 1.1 – Результат встановлення Node.js

Зміст файлу «package.json» наведено на лістингу 1.1.

Лістинг 1.1 – Зміст файлу «package.json»

{

  "name": "practice",

  "version": "1.0.0",

  "main": "app.js",

  "scripts": {

    "test": "jest",

    "start": "node app.js"

  },

  "author": "Mariia Alefirenko",

  "license": "ISC",

  "description": "Practice of the student of the group KI-213 Mariia Alefirenko.",

  "devDependencies": {

    "dotenv": "^16.4.5",

    "jest": "^29.7.0",

    "supertest": "^7.0.0"

  },

  "dependencies": {

    "express": "^4.19.2"

  }

}

На лістингу 1.2 наведено код головного файлу «app.js», а на лістингу 1.3 – файлу тесту «app.test.js». Результати роботи цих програм у браузері та консолі зображено на рисунку 1.2.

Лістинг 1.2 – Код головної програми

const express = require("express");

const app = express();

const port = 3000;

app.get("/", (req, res) => {

  res.send("Hello World");

});

if (require.main === module) {

  app.listen(port, () => {

    console.log(`Server is running on http://localhost:${port}`);

  });

}

module.exports = app;

Лістинг 1.3 – Код програми тесту

const request = require("supertest");

const app = require("./app");

describe("GET /", () => {

  it("responds with Hello World", async () => {

    const response = await request(app).get("/");

    expect(response.text).toBe("Hello World");

  });

});

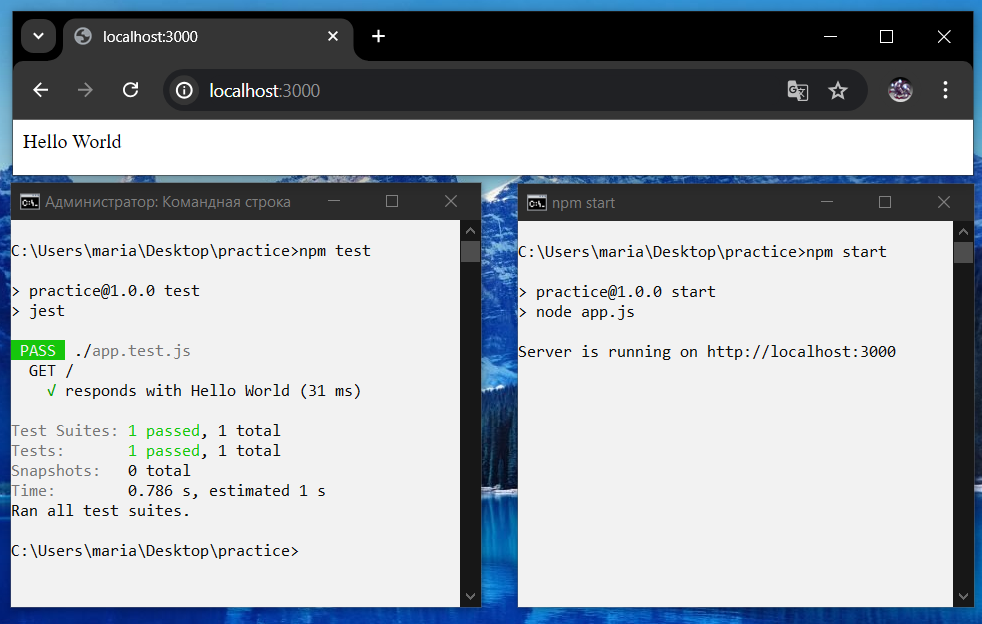


Рисунок 1.2 – Результат роботи основної програми та тесту

Тепер, змінимо код головної програми для використання «dotenv», та створимо файл «.env». Результати цих дій наведено на лістингу 1.4 та 1.5, а також зображено на рисунку 1.3.

Лістинг 1.4 – Код головної програми після змін

const express = require("express");

const app = express();

const dotenv = require("dotenv");

dotenv.config();

const port = 3000;

app.get("/", (req, res) => {

  res.send(process.env.RESPONSE\_TEXT);

});

module.exports = app;

if (require.main === module) {

  app.listen(port, () => {

    console.log(`Server is running on http://localhost:${port}`);

  });

}

Лістинг 1.5 – Текст файлу «.env»

RESPONSE\_TEXT = Hello from dotenv!

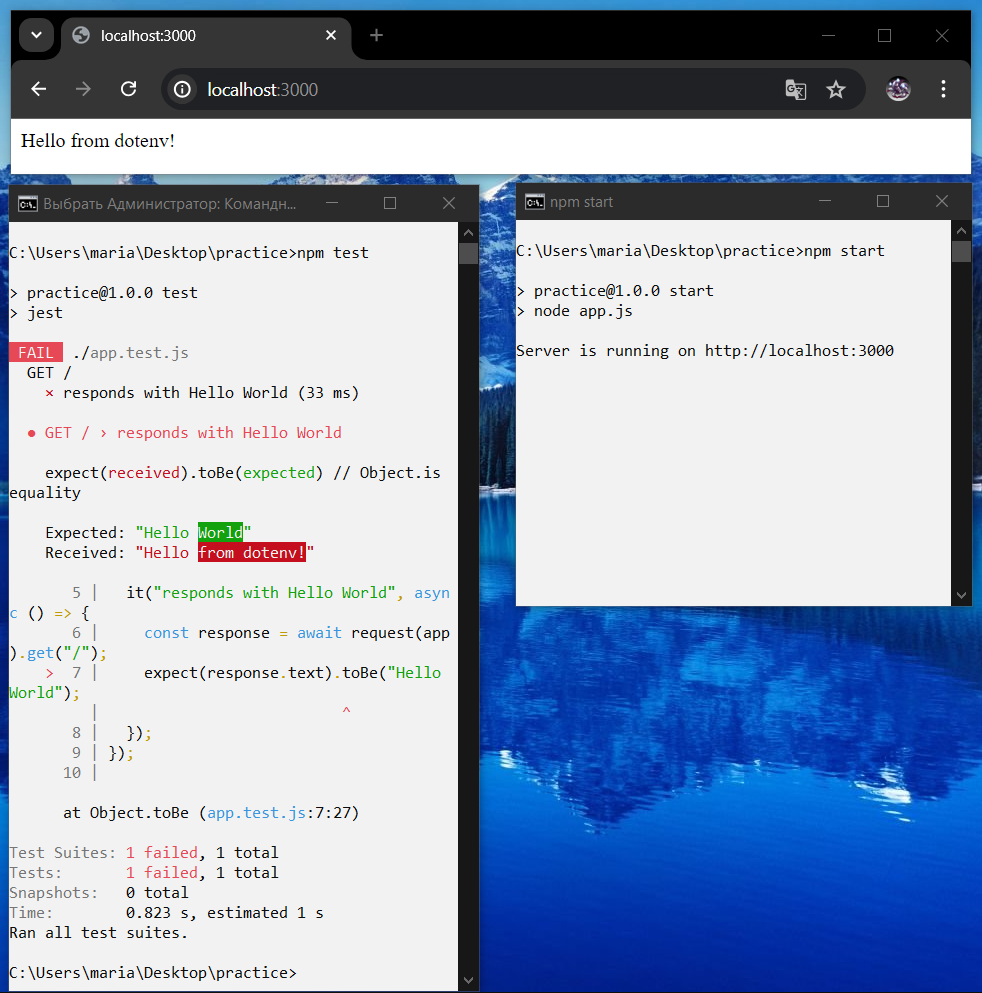


Рисунок 1.3 – Результат роботи основної програми та тесту з використанням «dotenv»

**1.4 Висновки**

Виконавши лабораторну роботу, ми ознайомилися з основами роботи в середовищі Node.js, використанням пакетного менеджера npm для керування залежностями проекту, створенням веб-додатків за допомогою фреймворку Express та проведенням тестування веб-сервісів з використанням бібліотек Jest та Supertest.

# 2 ЛАБОРАТОРНА РОБОТА № 2.

**КОНВЕЄР ОБРОБКИ ЗАПИТУ ТА MIDDLEWARE.**

**ВІДПРАВЛЕННЯ ВІДПОВІДІ.**

**МАРШРУТИЗАЦІЯ. ПЕРЕАДРЕСАЦІЯ**

**2.1 Мета роботи**

Мета лабораторної роботи полягає в освоєнні конвеєра обробки запиту та middleware веб-додатків. А ще в отриманні досвіду з використання відповідей сервера. Також будуть розглянуті основи маршрутизації та переадресації.

**2.2 Завдання на роботу**

* У корені проекту створити каталог "api". Перемістити всі файли з лабораторної 1 у каталог "api".
* У корені проекту створити каталог "proxy".
* У каталозі "proxy" створити додаток, який схожий на "api", але використовує порт 3001 і для шляху '/' відправляє запит до "api" та повертає отриманий результат.
* Для "proxy" через .env файл і пакет convict вказати url додатка "api" (localhost:port) і порт, який використовується додатком "proxy".
* Розбити застосунок "proxy" на шари - сервер, роутинг, сервіс, клієнт (використовує axios). Запустити обидва застосунки.
* У браузері ввести адресу застосунку "proxy" і побачити Hello world від застосунку "api".

**2.3 Результати виконання роботи**

На рисунку 2.1 зображено зміні в каталозі проекту.

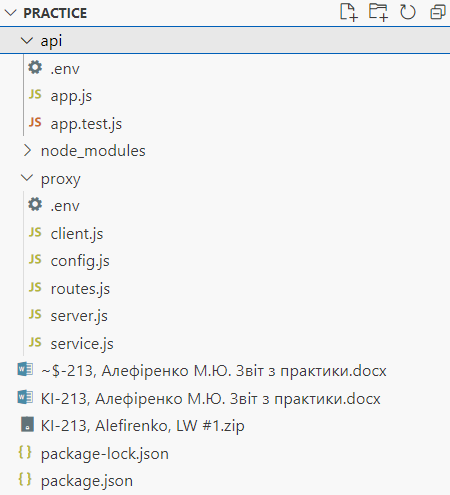


Рисунок 2.1 – Зміні в каталозі проекту

Змінений зміст файлу «package.json» наведено на лістингу 2.1.

Лістинг 2.1 – Зміст файлу «package.json»

{

  "name": "practice",

  "version": "1.0.0",

  "main": "app.js",

  "scripts": {

    "test": "jest",

    "start-api": "node api/app.js",

    "start-proxy": "node proxy/server.js"

  },

  "author": "Mariia Alefirenko",

  "license": "ISC",

  "description": "Practice of the student of the group KI-213 Mariia Alefirenko.",

  "devDependencies": {

    "dotenv": "^16.4.5",

    "jest": "^29.7.0",

    "supertest": "^7.0.0"

  },

  "dependencies": {

    "axios": "^1.7.2",

    "convict": "^6.2.4",

    "express": "^4.19.2"

  }

}

На лістингу 2.2 наведено код файлу «proxy/server.js», на лістингу 2.3 файлу «proxy/config.js», на лістингу 2.4 «proxy/client.js», а на лістингу 2.5 – 2.7 відповідно «proxy/routes.js», «proxy/service.js» та «proxy/.env».

Лістинг 2.2 – Код «proxy/server.js»

const express = require("express");

const dotenv = require("dotenv");

const config = require("./config");

const routes = require("./routes");

dotenv.config();

const app = express();

const port = config.get("proxy.port");

app.use("/", routes);

app.listen(port, () => {

  console.log(`Proxy server is running on http://localhost:${port}`);

});

Лістинг 2.3 – Код «proxy/config.js»

const convict = require("convict");

convict.addFormat({

  name: "url",

  validate: function (val) {

    const urlRegex = /^(ftp|http|https):\/\/[^ "]+$/;

    if (!urlRegex.test(val)) {

      throw new Error("Invalid URL format");

    }

  },

  coerce: function (val) {

    return val.trim();

  },

});

const config = convict({

  api: {

    url: {

      doc: "The URL of the API server",

      format: "url",

      default: "http://localhost:3000",

      env: "API\_URL",

    },

  },

  proxy: {

    port: {

      doc: "The port to bind the proxy server",

      format: "port",

      default: 3001,

      env: "PROXY\_PORT",

    },

  },

});

config.validate({ allowed: "strict" });

module.exports = config;

Лістинг 2.4 – Код «proxy/client.js»

const axios = require("axios");

const config = require("./config");

const apiClient = axios.create({

  baseURL: config.get("api.url"),

});

module.exports = apiClient;

Лістинг 2.5 – Код «proxy/routes.js»

const express = require("express");

const router = express.Router();

const service = require("./service");

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

  try {

    const data = await service.getHelloWorld();

    res.send(data);

  } catch (error) {

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

  }

});

module.exports = router;

Лістинг 2.6 – Текст файлу «proxy/service.js»

const apiClient = require("./client");

const getHelloWorld = async () => {

  const response = await apiClient.get("/");

  return response.data;

};

module.exports = {

  getHelloWorld,

};

Лістинг 2.7 – Текст файлу «proxy/.env»

API\_URL = http://localhost:3000

PROXY\_PORT = 3001

Результати виконання лабораторної роботи зображено на рисунку 2.2.

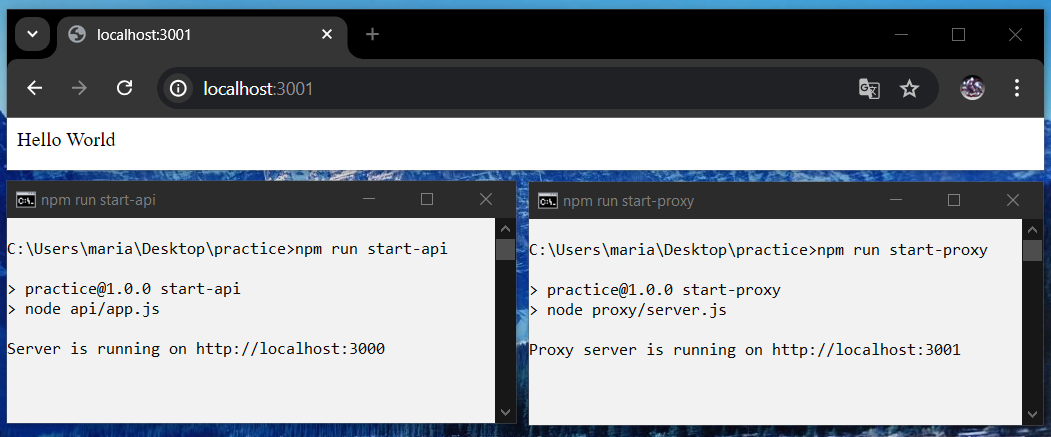


Рисунок 2.2 – Результати виконання лабораторної роботи

**2.4 Висновки**

Виконавши лабораторну роботу, ми освоїли конвеєр обробки запиту та middleware. А ще отримали досвід з використанням відповідей сервера. Також були розглянуті основи маршрутизації та переадресації.

# 3 ЛАБОРАТОРНА РОБОТА № 3.

**NODE.JS ТА MONGODB**

**3.1 Мета роботи**

Метою лабораторної роботи є розробка веб-додатку з використанням Node.js та MongoDB. Робота спрямована на ознайомлення студентів із сучасними інструментами розробки веб-додатків та баз даних, здатних працювати з неструктурованими даними.

**3.2 Завдання на роботу**

Модіфікувати додаток з використанням власної бази MongoDB.

**3.3 Результати виконання роботи**

На рисунку 3.1 зображено структуру проекту.

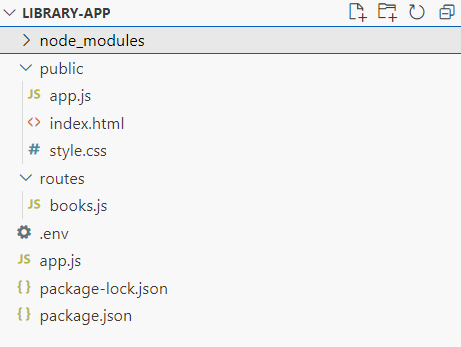


Рисунок 3.1 – Результат встановлення Node.js

Зміст файлу «package.json» наведено на лістингу 3.1.

Лістинг 3.1 – Зміст файлу «package.json»

{

  "name": "library-app",

  "version": "1.0.0",

  "description": "Laboratory Work #3. Web application for managing library records.",

  "main": "app.js",

  "scripts": {

    "start": "node app.js"

  },

  "author": "Mariia Alefirenko",

  "license": "ISC",

  "dependencies": {

    "dotenv": "^16.4.5",

    "express": "^4.19.2",

    "mongodb": "^6.8.0"

  }

}

На лістингу 3.2 наведено код файлу «app.js», на лістингу 3.3 файлу «public/app.js», на лістингу 3.4 «routes/books.js», а на лістингу 3.5 – 3.7 відповідно «public/index.html», «public /style.css» та «.env».

Лістинг 3.2 – Код «app.js»

const express = require("express");

const { MongoClient } = require("mongodb");

const dotenv = require("dotenv");

const path = require("path");

dotenv.config();

const app = express();

app.use(express.json());

app.use(express.static(path.join(\_\_dirname, "public")));

const client = new MongoClient(process.env.MONGO\_URI, {

  useNewUrlParser: true,

  useUnifiedTopology: true,

});

client

  .connect()

  .then(() => {

    console.log("Connected to MongoDB");

    const db = client.db();

    const booksRouter = require("./routes/books")(db);

    app.use("/api/books", booksRouter);

  })

  .catch((err) => console.error("Could not connect to MongoDB...", err));

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

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

Лістинг 3.3 – Код «public/app.js»

async function addBook() {

  const author = document.getElementById("author").value;

  const title = document.getElementById("title").value;

  const quantity = document.getElementById("quantity").value;

  const response = await fetch("/api/books", {

    method: "POST",

    headers: {

      "Content-Type": "application/json",

    },

    body: JSON.stringify({ author, title, quantity }),

  });

  const book = await response.json();

  loadBooks();

}

async function loadBooks() {

  const response = await fetch("/api/books");

  const books = await response.json();

  const booksTable = document.getElementById("books");

  booksTable.innerHTML = "";

  books.forEach((book) => {

    const row = document.createElement("tr");

    row.innerHTML = `

        <td>${book.\_id}</td>

        <td>${book.author}</td>

        <td>${book.title}</td>

        <td>${book.quantity}</td>

        <td>

          <button onclick="deleteBook('${book.\_id}')">Delete</button>

          <button onclick="editBook('${book.\_id}')">Edit</button>

        </td>

      `;

    booksTable.appendChild(row);

  });

}

async function editBook(id) {

  const author = prompt("Enter new author:");

  const title = prompt("Enter new title:");

  const quantity = prompt("Enter new quantity:");

  const response = await fetch(`/api/books/${id}`, {

    method: "PUT",

    headers: {

      "Content-Type": "application/json",

    },

    body: JSON.stringify({ author, title, quantity }),

  });

  const updatedBook = await response.json();

  loadBooks();

}

async function deleteBook(id) {

  await fetch(`/api/books/${id}`, { method: "DELETE" });

  loadBooks();

}

async function deleteAllBooks() {

  const response = await fetch("/api/books");

  const books = await response.json();

  books.forEach(async (book) => {

    await fetch(`/api/books/${book.\_id}`, { method: "DELETE" });

  });

  loadBooks();

}

Лістинг 3.4 – Код «routes/books.js»

const express = require("express");

const { ObjectId } = require("mongodb");

module.exports = function (db) {

  const router = express.Router();

  const booksCollection = db.collection("books");

  // Get all books

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

    try {

      const books = await booksCollection.find().toArray();

      res.send(books);

    } catch (error) {

      res.status(500).send(error.message);

    }

  });

  // Add a new book

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

    const { author, title, quantity } = req.body;

    const book = { author, title, quantity };

    try {

      await booksCollection.insertOne(book);

      res.send(book);

    } catch (error) {

      res.status(500).send(error.message);

    }

  });

  // Update a book

  router.put("/:id", async (req, res) => {

    const { author, title, quantity } = req.body;

    try {

      const result = await booksCollection.updateOne(

        { \_id: new ObjectId(req.params.id) },

        { $set: { author, title, quantity } }

      );

      if (result.matchedCount === 0) {

        return res.status(404).send("Book not found");

      }

      res.send(

        await booksCollection.findOne({ \_id: new ObjectId(req.params.id) })

      );

    } catch (error) {

      res.status(500).send(error.message);

    }

  });

  // Delete a book

  router.delete("/:id", async (req, res) => {

    try {

      const result = await booksCollection.deleteOne({

        \_id: new ObjectId(req.params.id),

      });

      if (result.deletedCount === 0)

        return res.status(404).send("Book not found");

      res.send(result);

    } catch (error) {

      res.status(500).send(error.message);

    }

  });

  return router;

};

Лістинг 3.5 – Код «public/index.html»

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

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

    <title>Library App</title>

    <link rel="stylesheet" href="style.css" />

  </head>

  <body>

    <div class="container">

      <h1>MongoDB Library App</h1>

      <div class="form-container">

        <h2>Add a Book</h2>

        <input type="text" id="author" placeholder="Author" />

        <input type="text" id="title" placeholder="Title" />

        <input type="number" id="quantity" placeholder="Quantity" />

        <button onclick="addBook()">Add</button>

      </div>

      <div class="actions">

        <h2>Books</h2>

        <button onclick="loadBooks()">Load Books</button>

        <button onclick="deleteAllBooks()">Delete All Books</button>

      </div>

      <table>

        <thead>

          <tr>

            <th>ID</th>

            <th>Author</th>

            <th>Title</th>

            <th>Quantity</th>

            <th>Actions</th>

          </tr>

        </thead>

        <tbody id="books"></tbody>

      </table>

    </div>

    <script src="app.js"></script>

  </body>

</html>

Лістинг 3.6 – Код «public /style.css»

body {

  font-family: Arial, sans-serif;

  background-color: #f8f9fa;

  margin: 0;

  padding: 0;

}

.container {

  max-width: 1000px;

  margin: 30px auto;

  padding: 20px;

  background-color: #fff;

  border-radius: 8px;

  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

}

h1 {

  text-align: center;

  margin-bottom: 20px;

}

.form-container,.actions {

  margin-bottom: 20px;

}

.form-container h2,.actions h2 {

  margin-bottom: 10px;

}

input[type="text"],input[type="number"] {

  width: calc(100% - 22px);

  padding: 10px;

  margin-bottom: 10px;

  border: 1px solid #ccc;

  border-radius: 4px;

  box-sizing: border-box;

}

button {

  width: 100%;

  padding: 10px;

  background-color: #fa93ff;

  border: solid purple;

  border-radius: 4px;

  color: #fff;

  font-size: 16px;

  cursor: pointer;

}

button:hover {

  background-color: #9e44c8;

}

.actions button {

  width: auto;

  display: inline-block;

  margin-right: 10px;

}

table {

  width: 100%;

  border-collapse: collapse;

  margin-top: 20px;

}

th,td {

  border: 1px solid #ddd;

  padding: 8px;

  text-align: left;

}

th {

  background-color: #f2f2f2;

}

td {

  vertical-align: middle;

}

td button {

  margin: 5px;

}

Лістинг 3.7 – Текст файлу «.env»

MONGO\_URI = mongodb://localhost:27017/library

Результати виконання лабораторної роботи зображено на рисунку 3.2.

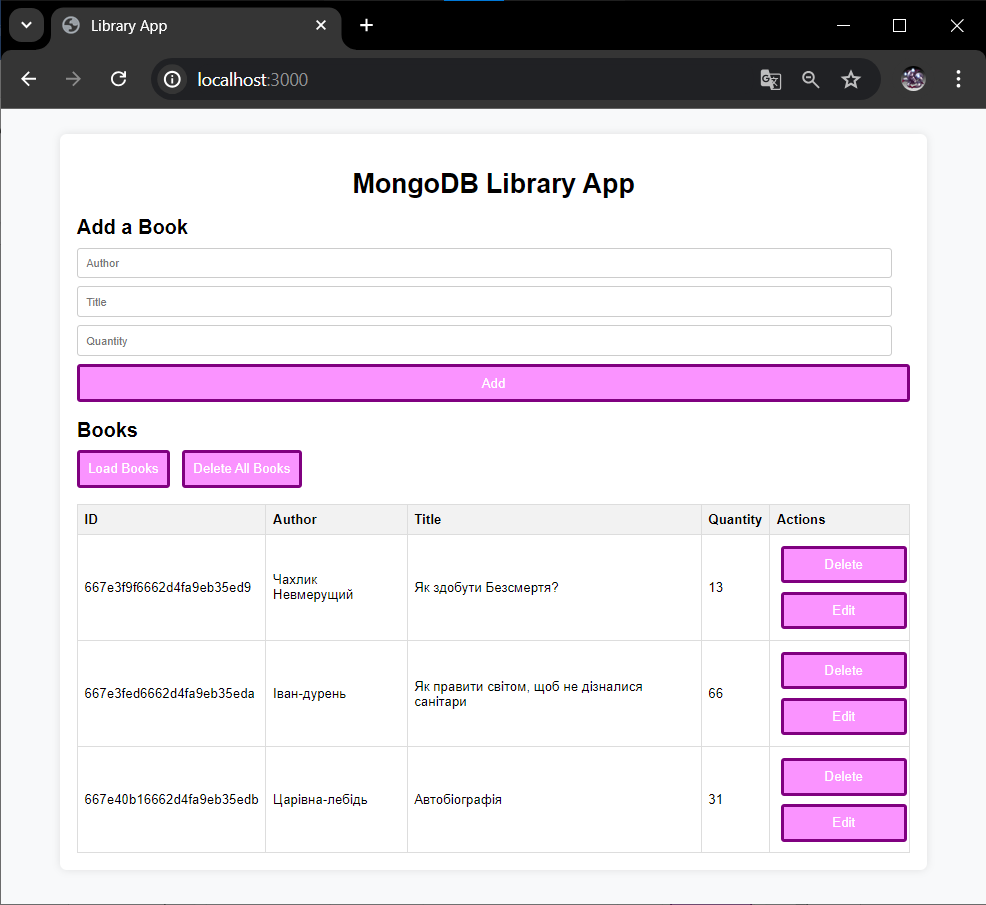


Рисунок 3.2 – Результати виконання лабораторної роботи

**3.4 Висновки**

Виконавши лабораторну роботу, ми розробили веб-додаток з використанням Node.js та MongoDB та ознайомилися студентів із сучасними інструментами розробки веб-додатків та баз даних, здатних працювати з неструктурованими даними.

# 4 ЛАБОРАТОРНА РОБОТА № 4.

**MONGOOSE**

**4.1 Мета роботи**

Метою лабораторної роботи є оволодінні технікою роботи з «mongoose» для зручного та ефективного взаємодії з базою даних.

**4.2 Завдання на роботу**

Модіфікувати створений у додаток з використанням mongoose і власної бази MongoDB.

**4.3 Результати виконання роботи**

Зміни у змісті файлу «package.json» наведено на лістингу 4.1.

Лістинг 4.1 – Зміни у змісті файлу «package.json»

  "dependencies": {

    "dotenv": "^16.4.5",

    "express": "^4.19.2",

    "mongoose": "^8.4.4"

  }

На лістингу 4.2 наведено зміни у змісті файлу «app.js», де використовується «mongoose» для підключення до MongoDB замість прямого використання «mongodb».

Лістинг 4.2 – Зміни у змісті «app.js»

const express = require("express");

const mongoose = require("mongoose");

const dotenv = require("dotenv");

const path = require("path");

dotenv.config();

const app = express();

app.use(express.json());

app.use(express.static(path.join(\_\_dirname, "public")));

mongoose

  .connect(process.env.MONGO\_URI, {

    useNewUrlParser: true,

    useUnifiedTopology: true,

  })

  .then(() => console.log("Connected to MongoDB"))

  .catch((err) => console.error("Could not connect to MongoDB...", err));

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

// Routes

const booksRouter = require("./routes/books");

app.use("/api/books", booksRouter);

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

Введено новий файл «routes/Book.js», де визначено схему книги і модель «Book» за допомогою «mongoose»(лістинг 4.3).

Лістинг 4.3 – Код «routes/Book.js»

const mongoose = require("mongoose");

const bookSchema = new mongoose.Schema({

  author: { type: String, required: true },

  title: { type: String, required: true },

  quantity: { type: Number, required: true },

});

const Book = mongoose.model("Book", bookSchema);

module.exports = Book;

В «routes/books.js» використовується «mongoose» для операцій з базою даних. Результати виконання лабораторної роботи зображено на рисунку 4.1.

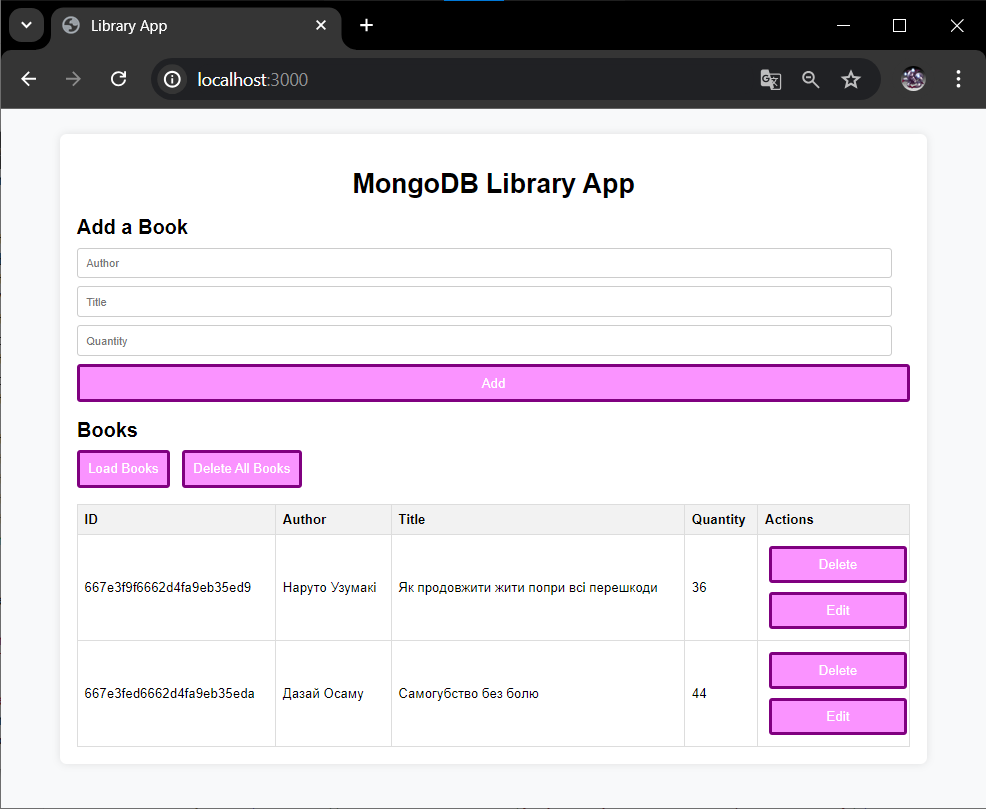


Рисунок 4.1 – Результати виконання лабораторної роботи

**4.4 Висновки**

Виконавши лабораторну роботу, ми модифікували існуючий додаток з використанням технології «mongoose» та власної бази даних MongoDB. Також оволоділи технікою роботи з «mongoose» для зручного та ефективного взаємодії з базою даних.

# 5 ЛАБОРАТОРНА РОБОТА № 5.

**ВИКОРИСТАННЯ DOCKER ДЛЯ РОЗГОРТАННЯ ДОДАТКУ NODE.JS**

**5.1 Мета роботи**

Метою лабораторної роботи є ознайомлення з базовими принципами використання Docker для розгортання додатків на прикладі середовища Node.js.

**5.2 Завдання на роботу**

Використати додаток з минулої лабораторної та модифікували його так, аби він працював за допомогою контейнерів docker.

**5.3 Результати виконання роботи**

Спочатку було встановлено Docker, результат зображено на рисунку 5.1.

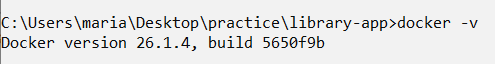


Рисунок 5.1 – Результати встановлення Docker

Створили файл «dockerfile» та файл «.dockerignore», вміст файлів наведено в лістингу 5.1 та 5.2.

Лістинг 5.1 – Вміст файлу «dockerfile»

FROM node:latest

RUN mkdir -p /usr/src/app/

WORKDIR /usr/src/app/

COPY . /usr/src/app/

RUN npm install

EXPOSE 3000

CMD ["node", "app.js"]

Лістинг 5.2 – Вміст файлу «.dockerignore»

node\_modules

Dockerfile

package-lock.json

Далі було завантажено образ «mongodb» за допомогою команди «docker pull mongo», а потім запустили контейнер «mongodb» за допомогою команди «docker run -d -p 27017:27017 --name mongo\_container mongo», результат зображено на рисунку 5.2.

Результат перевірки цих дій у вигляді інтерфейсу «Docker» наведено на рисунку 5.3.

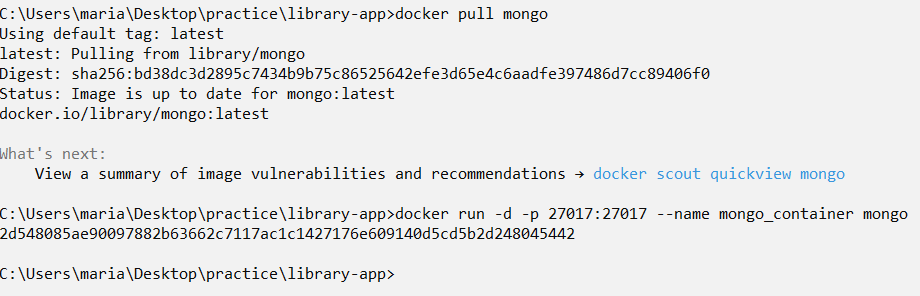


Рисунок 5.2 – Завантаження образу «mongodb» та запуск контейнера «mongodb»

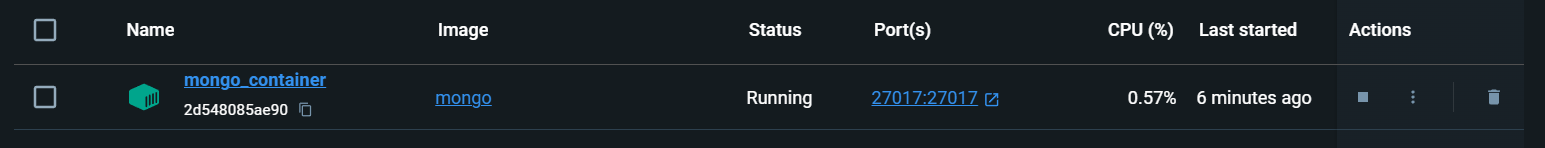


Рисунок 5.3 – Інтерфейс «Docker»

Створили власний образ Docker за допомогою команди «docker build -t node\_app .». Результат виконання команди наведено на рисунку 5.4.



Рисунок 5.4 – Результат виконання команди

Запустили контейнер, та переконалися, що він працює, на рисунку 5.5.

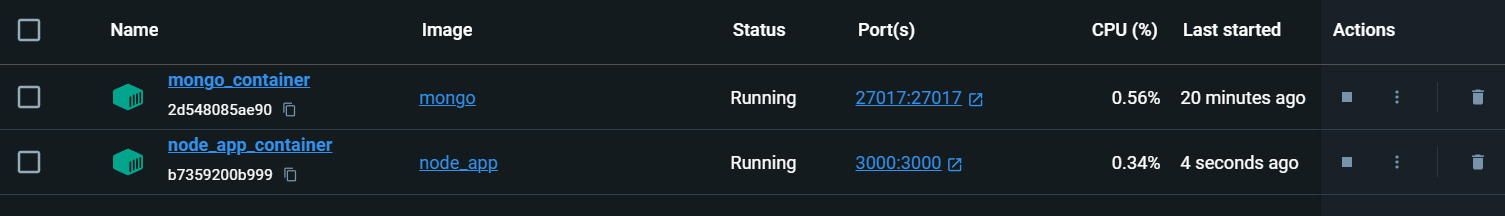


Рисунок 5.5 – Інтерфейс «Docker»

Для використання «Docker compose» для автоматизації, створюємо файл «docker-compose.yaml» у корені проекту. Вміст цього файлу наведено у лістингу 5.3.

Лістинг 5.3 – Вміст файлу «docker-compose.yaml»

version: "3"

volumes:

  mongo\_db\_data:

services:

  my\_mongodb:

    image: mongo:latest

    volumes:

      - mongo\_db\_data:/data/db

  app:

    build: .

    ports:

      - "3000:3000"

    environment:

      - MONGO\_DB\_HOSTNAME=my\_mongodb

      - MONGO\_DB\_PORT=27017

      - MONGO\_DB=usersdb

Також нам потрібно оновити файли «.dockerignore» і «app.js», що наведено в лістингу 5.4 та 5.5.

Лістинг 5.4 – Змінений вміст файлу «.dockerignore»

node\_modules

docker-compose.yaml

Dockerfile

package-lock.json

Лістинг 5.5 – Змінений вміст файлу «app.js»

async function addBook() {

  const author = document.getElementById("author").value;

  const title = document.getElementById("title").value;

  const quantity = document.getElementById("quantity").value;

  const response = await fetch("http://app:3000/api/books", {

    method: "POST",

    headers: {

      "Content-Type": "application/json",

    },

    body: JSON.stringify({ author, title, quantity }),

  });

  const book = await response.json();

  loadBooks();

}

async function loadBooks() {

  const response = await fetch("http://app:3000/api/books");

  const books = await response.json();

  const booksTable = document.getElementById("books");

  booksTable.innerHTML = "";

  books.forEach((book) => {

    const row = document.createElement("tr");

    row.innerHTML = `

        <td>${book.\_id}</td>

        <td>${book.author}</td>

        <td>${book.title}</td>

        <td>${book.quantity}</td>

        <td>

          <button onclick="deleteBook('${book.\_id}')">Delete</button>

          <button onclick="editBook('${book.\_id}')">Edit</button>

        </td>

      `;

    booksTable.appendChild(row);

  });

}

async function editBook(id) {

  const author = prompt("Enter new author:");

  const title = prompt("Enter new title:");

  const quantity = prompt("Enter new quantity:");

  const response = await fetch(`http://app:3000/api/books/${id}`, {

    method: "PUT",

    headers: {

      "Content-Type": "application/json",

    },

    body: JSON.stringify({ author, title, quantity }),

  });

  const updatedBook = await response.json();

  loadBooks();

}

async function deleteBook(id) {

  await fetch(`http://app:3000/api/books/${id}`, { method: "DELETE" });

  loadBooks();

}

async function deleteAllBooks() {

  const response = await fetch("http://app:3000/api/books");

  const books = await response.json();

  books.forEach(async (book) => {

    await fetch(`http://app:3000/api/books/${book.\_id}`, { method: "DELETE" });

  });

  loadBooks();

}

Тепер запустимо «Docker-compose» командою «docker-compose up». На рисунку 5.6 та 5.7 зображено роботу додатку.

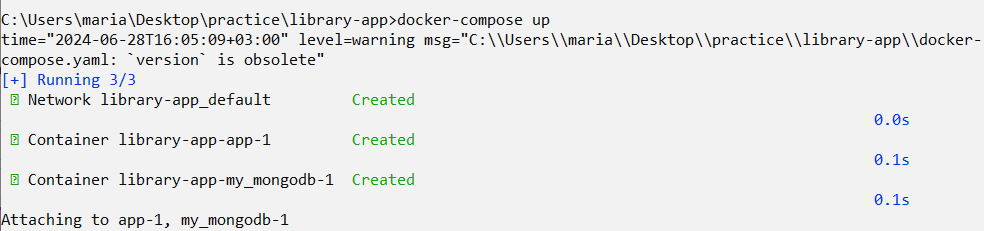


Рисунок 5.6 – Запуск контейнера

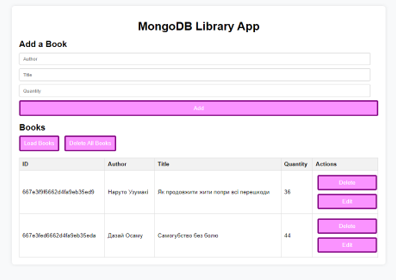


Рисунок 5.7 – Додаток запущений через «docker»

**5.4 Висновки**

Виконавши лабораторну роботу, ми ознайомилися з docker та його можливостями.

# ВИСНОВОК

Виконавши серію лабораторних робіт, ми здобули значний досвід та знання в декількох ключових аспектах сучасної веб-розробки. Зокрема, ми:

* Ознайомилися з основами роботи в середовищі Node.js та використанням пакетного менеджера npm для керування залежностями проекту.
* Освоїли створення веб-додатків за допомогою фреймворку Express та тестування веб-сервісів з використанням бібліотек Jest та Supertest.
* Зрозуміли принципи роботи конвеєра обробки запитів та middleware, а також отримали практичний досвід використання відповідей сервера.
* Розглянули основи маршрутизації та переадресації в веб-додатках.
* Розробили веб-додаток з використанням Node.js та MongoDB, що дозволило ознайомитися з інструментами для роботи з неструктурованими даними.
* Модифікували існуючий додаток з використанням технології «mongoose» та власної бази даних MongoDB, що допомогло оволодіти технікою зручної та ефективної взаємодії з базою даних.
* Ознайомилися з Docker та його можливостями для розгортання та управління контейнерами, що є важливим аспектом сучасної розробки та деплойменту додатків.

Ці навички та знання є фундаментальними для успішної кар'єри в сфері веб-розробки, дозволяючи створювати, тестувати та ефективно розгортати сучасні веб-додатки.

# ПОСИЛАННЯ НА GITHUB

<https://github.com/MariaAlefirenko/Practice>