Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4

з дисципліни «Основи WEB - технологій»

Тема: «NodeJS. Робота з БД. Додаток що реалізує CRUD операції в БД.»

Перевірив:
Доц. Голубєв Л. П.

Виконав:
студент групи ІМ-22
Балахон Михайло
Варіант 11

Київ - 2025

Завдання.

Створити додаток, що реалізує CRUD операції з БД – додавання, читання, редагування та видалення записів БД (Додаток 1. Таблиця 1).

Забезпечити роутінг запитів та виведення результатів запитів на WEB-сторінку.

Додати нові роути для виведення інформації у вигляді json-файлу (API).

Хід роботи:

Хід виконання роботи

1. Підготовка проекту

1.1. Створення структури проекту

```
mkdir lab4

cd lab4

mkdir public

npm init -y
```

1.2. Встановлення необхідних пакетів

```
npm install express mongodb dotenv cors

npm install --save-dev nodemon
```

Встановлені пакети:

- `express` (v5.1.0) - веб-фреймворк для Node.js
- `mongodb` (v6.3.0) - офіційний драйвер MongoDB для Node.js
- `dotenv` (v16.3.1) - для роботи з змінними оточення

- `cors` (v2.8.5) - для налаштування CORS політики
- `nodemon` (v3.1.7) - для автоматичного перезапуску під час розробки

2. Налаштування конфігурації

2.1. Файл .env

```
MONGODB_URI=mongodb://localhost:27017/lab4_db

PORT=3004
```

2.2. Налаштування MongoDB

1. **Встановлення MongoDB:**
    - Завантаження з офіційного сайту
    - Встановлення локального сервера
2. **Запуск MongoDB:**

```
mongod
```

3. Реалізація серверної частини

3.1. Ініціалізація сервера та підключення до БД

```
const express = require('express');

const { MongoClient, ObjectId } = require('mongodb');

const cors = require('cors');

require('dotenv').config();


const app = express();

const PORT = process.env.PORT || 3004;

const MONGODB_URI = process.env.MONGODB_URI || 'mongodb://localhost:27017/lab4_db';
```

```
let db;


MongoClient.connect(MONGODB_URI, { useUnifiedTopology: true })
    .then(client => {
        console.log('Connected to MongoDB');
        db = client.db();
        // Створення індексу для оптимізації пошуку
        db.collection('products').createIndex({ name: 1 });
    })
    .catch(error => console.error('MongoDB connection error:', error));
```

3.2. Структура даних продукту

```
{
  _id: ObjectId,
  name: String,               // Назва продукту
  description: String,        // Опис
  category: String,          // Категорія
  price: Number,            // Ціна
  quantity: Number,         // Кількість на складі
  createdAt: Date,         // Дата створення
  updatedAt: Date          // Дата оновлення
}
```

4. Реалізація API endpoints

4.1. CRUD операції

CREATE - Створення продукту:

```
app.post('/api/products', async (req, res) => {
    const { name, description, price, category, quantity } = req.body;

    if (!name || !price || !category) {
        return res.status(400).json({ error: 'Name, price, and category are required'
});
    }

    const newProduct = {
        name,
        description: description || '',
        price: parseFloat(price),
        category,
        quantity: parseInt(quantity) || 0,
        createdAt: new Date(),
        updatedAt: new Date()
    };

    const result = await db.collection('products').insertOne(newProduct);
    res.status(201).json(newProduct);
});
```

READ - Отримання продуктів з фільтрацією:

```
app.get('/api/products', async (req, res) => {
    const { category, minPrice, maxPrice, sort } = req.query;
    const query = {};

    // Побудова запиту з фільтрами
    if (category) query.category = category;
    if (minPrice || maxPrice) {
        query.price = {};
        if (minPrice) query.price.$gte = parseFloat(minPrice);
        if (maxPrice) query.price.$lte = parseFloat(maxPrice);
    }

    // Налаштування сортування
    let sortOption = {};
    if (sort === 'price-asc') sortOption = { price: 1 };
    else if (sort === 'price-desc') sortOption = { price: -1 };
    else if (sort === 'name') sortOption = { name: 1 };

    const products = await db.collection('products')
        .find(query)
        .sort(sortOption)
        .toArray();

    res.json(products);
```

```
});
```

UPDATE - Оновлення продукту:

```
app.put('/api/products/:id', async (req, res) => {
    const updateDoc = {
        $set: {
            updatedAt: new Date()
        }
    };

    // Динамічне додавання полів для оновлення
    if (name !== undefined) updateDoc.$set.name = name;
    if (description !== undefined) updateDoc.$set.description = description;
    if (price !== undefined) updateDoc.$set.price = parseFloat(price);
    if (category !== undefined) updateDoc.$set.category = category;
    if (quantity !== undefined) updateDoc.$set.quantity = parseInt(quantity);

    const result = await db.collection('products').updateOne(
        { _id: new ObjectId(req.params.id) },
        updateDoc
    );

    res.json(updatedProduct);
});
```

DELETE - Видалення продукту:

```
app.delete('/api/products/:id', async (req, res) => {
    const result = await db.collection('products').deleteOne({
        _id: new ObjectId(req.params.id)
    });

    if (result.deletedCount === 0) {
        return res.status(404).json({ error: 'Product not found' });
    }

    res.json({ message: 'Product deleted successfully' });
});
```

4.2. Додаткові endpoints

Отримання списку категорій:

```
app.get('/api/categories', async (req, res) => {
    const categories = await db.collection('products').distinct('category');
    res.json(categories);
});
```

Статистика:

```
app.get('/api/statistics', async (req, res) => {
```

```
    // Агрегація даних по категоріях
    const stats = await db.collection('products').aggregate([
        {
            $group: {
                _id: '$category',
                count: { $sum: 1 },
                totalValue: { $sum: { $multiply: ['$price', '$quantity'] } },
                avgPrice: { $avg: '$price' }
            }
        },
        {
            $project: {
                category: '$_id',
                count: 1,
                totalValue: { $round: ['$totalValue', 2] },
                avgPrice: { $round: ['$avgPrice', 2] },
                _id: 0
            }
        }
    ]).toArray();

    res.json({ byCategory: stats, total: totalStats[0] });
});
```

5. Реалізація клієнтської частини

5.1. HTML структура

Створено single-page application з секціями:

- Статистика (загальні показники)
- Форма додавання/редагування продукту
- Фільтри та сортування
- Список продуктів
- Статистика за категоріями

5.2. JavaScript функціональність

Завантаження та відображення продуктів:

```
async function loadProducts() {
    const category = document.getElementById('categoryFilter').value;
    const minPrice = document.getElementById('minPrice').value;
    const maxPrice = document.getElementById('maxPrice').value;
    const sort = document.getElementById('sortBy').value;

    let url = `${API_URL}/products?`;
    if (category) url += `category=${encodeURIComponent(category)}&`;
    if (minPrice) url += `minPrice=${minPrice}&`;
    if (maxPrice) url += `maxPrice=${maxPrice}&`;
    if (sort) url += `sort=${sort}`;

    const response = await fetch(url);
    const products = await response.json();

    displayProducts(products);
}
```

Обробка форми:

```
async function handleSubmit(e) {
    e.preventDefault();

    const productData = {
        name: document.getElementById('name').value,
        description: document.getElementById('description').value,
        category: document.getElementById('category').value,
        price: parseFloat(document.getElementById('price').value),
        quantity: parseInt(document.getElementById('quantity').value)
    };

    let response;
    if (editingProductId) {
        response = await fetch(`${API_URL}/products/${editingProductId}`, {
            method: 'PUT',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(productData)
        });
    } else {
        response = await fetch(`${API_URL}/products`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(productData)
        });
```

```
    }
}
```

5.3. CSS стилізація

- Адаптивний дизайн з використанням Grid та Flexbox
- Анімації при наведенні
- Мобільна адаптація через media queries
- Кольорова схема з акцентами

6. Функціональні можливості додатку

6.1. CRUD операції

1. **Створення:** Додавання нових продуктів через форму
2. **Читання:** Відображення списку продуктів з пагінацією
3. **Оновлення:** Редагування існуючих продуктів
4. **Видалення:** Видалення продуктів з підтвердженням

6.2. Фільтрація та сортування

- Фільтр за категорією (випадаючий список)
- Фільтр за діапазоном цін (мін/макс)
- Сортування за назвою (алфавітний порядок)
- Сортування за ціною (зростання/спадання)

6.3. Статистика

- Загальна кількість продуктів
- Загальна кількість одиниць на складі
- Загальна вартість товарів
- Статистика по кожній категорії:
    - Кількість товарів
    - Середня ціна
    - Загальна вартість

6.4. Користувацький інтерфейс

- Інтуїтивна навігація

- Валідація форм на клієнті
- Повідомлення про успішні операції
- Обробка помилок з інформативними повідомленнями
- Автозаповнення категорій

7. Архітектура додатку

7.1. Backend архітектура

```
├── server.js          # Головний файл сервера
├── /api               # API endpoints
│   ├── /products    # CRUD операції з продуктами
│   ├── /categories  # Отримання категорій
│   └── /statistics  # Статистичні дані
```

7.2. Frontend архітектура

```
├── /public
│   ├── index.html    # Головна сторінка
│   ├── styles.css    # Стилі
│   └── script.js     # Клієнтська логіка
```

7.3. База даних

- **MongoDB колекція:** products
- **Індекси:** name (для оптимізації пошуку)
- **Валідація:** на рівні API

8. Безпека та оптимізація

8.1. Безпека

- Валідація вхідних даних
- Використання ObjectId для MongoDB
- CORS налаштування
- Захист від XSS через правильне відображення даних

8.2. Оптимізація

- Індексація полів для швидкого пошуку
- Агрегація даних на рівні БД
- Мінімізація кількості запитів
- Кешування категорій на клієнті

9. Тестування

9.1. Тестування API

1. CRUD операції через Postman/Insomnia
2. Перевірка валідації даних
3. Тестування фільтрів та сортування
4. Перевірка обробки помилок

9.2. Тестування інтерфейсу

1. Створення продуктів різних категорій
2. Редагування існуючих записів
3. Видалення з підтвердженням
4. Перевірка фільтрації та сортування
5. Адаптивність на різних пристроях

9.3. Тестування бази даних

1. Підключення до MongoDB
2. Створення та видалення записів
3. Перевірка індексів
4. Робота з великими обсягами даних

Структура проекту

```
lab4/
├── server.js           # Основний файл сервера
├── package.json        # Конфігурація проекту
├── .env                # Змінні середовища
├── .gitignore          # Ігноровані файли
├── README.md           # Документація
└── public/             # Статичні файли
    ├── index.html      # Головна сторінка
    ├── styles.css      # Стилі
    └── script.js       # Клієнтський JavaScript
```

Результати роботи

Створено повнофункціональний веб-додаток з наступними можливостями:

1. **REST API** з повним набором CRUD операцій
2. **MongoDB інтеграція** для збереження даних
3. **Веб-інтерфейс** з адаптивним дизайном
4. **Система фільтрації** за категорією та ціною
5. **Сортування** за різними критеріями
6. **Статистичний модуль** з агрегацією даних
7. **Валідація даних** на клієнті та сервері
8. **Обробка помилок** з інформативними повідомленнями

Скріншоти роботи додатку:

- Головна сторінка з статистикою
- Форма додавання продукту

- Список продуктів з фільтрами
- Редагування продукту
- Статистика за категоріями
- Мобільна версія інтерфейсу

Розгортання

Додаток готовий для розгортання на різних платформах:

- **Локально:** з використанням локальної MongoDB
- **MongoDB Atlas:** хмарна база даних
- **Heroku:** з MongoDB Atlas
- **DigitalOcean:** VPS з Docker
- **Vercel/Netlify:** frontend + serverless backend

Висновки

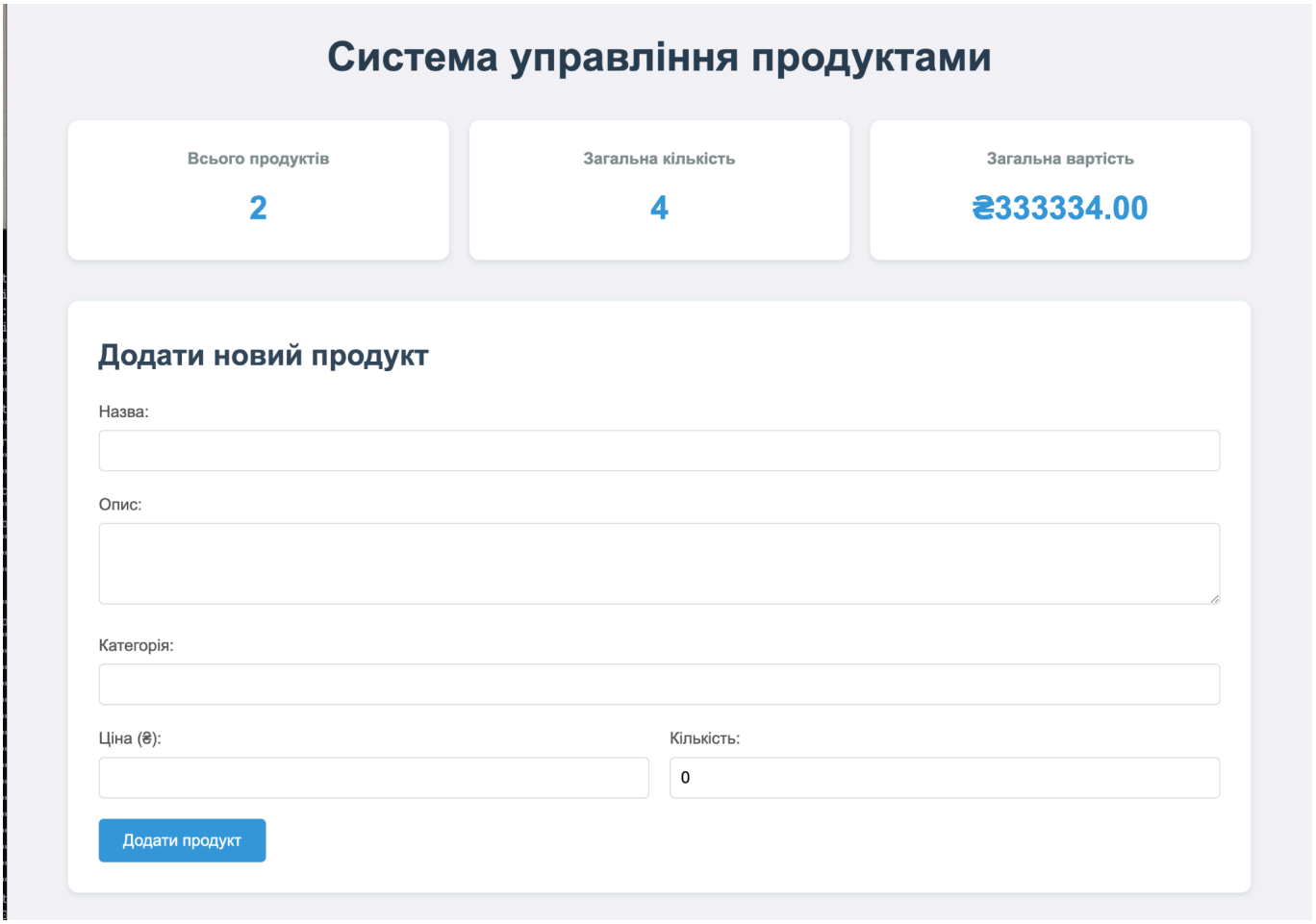
В ході виконання лабораторної роботи:

- Освоєно роботу з MongoDB та її драйвером для Node.js
- Реалізовано повний CRUD функціонал через REST API
- Створено інтерактивний веб-інтерфейс
- Впроваджено систему фільтрації та сортування даних
- Реалізовано збір та візуалізацію статистики
- Отримано навички роботи з NoSQL базами даних
- Вивчено принципи побудови single-page applications

MongoDB показала себе як потужна та гнучка система управління базами даних, що ідеально підходить для веб-додатків з динамічною структурою даних. Використання агрегаційного фреймворку дозволило ефективно обробляти статистичні запити.

Проект демонструє практичне застосування сучасного стеку технологій (Node.js, Express, MongoDB) для створення повнофункціональних веб-додатків з управління даними.

Скріншоти:


Система управління продуктами
Всього продуктів
2
Загальна кількість
4
Загальна вартість
₴333334.00
Додати новий продукт
Назва:
Опис:
Категорія:
Ціна (₴):
Кількість:
0
Додати продукт

Фільтри та сортування

Всі категорії | Мін. ціна | Макс. ціна | Без сортування | Застосувати

Список продуктів

Хліб

Редагувати

Видалити

Бородинський найкращий свіжий

Категорія: Їжа

Кількість: 1

₴1.00

Дорогий макбук

Редагувати

Видалити

опис

Категорія: Техніка

Кількість: 3

₴111111.00

Статистика за категоріями

Категорія	Кількість	Середня ціна	Загальна вартість
Техніка	1	₴111111.00	₴333333.00
Їжа	1	₴1.00	₴1.00