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

Лабораторна робота №5

з дисципліни «Основи WEB - технологій»

Тема: «GraphQL. Створення Schema GraphQL та Resolvers. Створення Query та Mutation. »

Перевірив:
Доц. Голубєв Л. П.

Виконав:
студент групи ІМ-22
Балахон Михайло
Варіант 11

Київ - 2025

Завдання.

На своїй БД (розробленої в лаб. роб. #4) за допомогою Schema Definition Language (SDL) створити схему GraphQL.

Додати Resolvers для виконання операцій GraphQL.

Створити та виконати Query та Mutation для виконання операцій додавання, редагування та видалення інформації (CRUD) в БД.

Виконати дослідження роботи створених query та mutation за допомогою Postman.

Хід роботи:

1. Підготовка проекту

1.1. Створення структури проекту

```
mkdir lab5

cd lab5

mkdir schema models

npm init -y
```

1.2. Встановлення необхідних пакетів

```
npm install apollo-server-express express graphql mongoose dotenv cors

npm install --save-dev nodemon
```

Встановлені пакети:

- `apollo-server-express` (v3.12.0) - GraphQL сервер для Express
- `graphql` (v16.8.1) - основна бібліотека GraphQL
- `mongoose` (v8.0.3) - ODM для MongoDB
- `express` (v5.1.0) - веб-фреймворк
- `dotenv` (v16.3.1) - для змінних середовища
- `cors` (v2.8.5) - для налаштування CORS
- `nodemon` (v3.1.7) - для розробки

2. Створення MongoDB моделей

2.1. Модель книги та автора

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


const authorSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true
  },
  bio: String,
  birthDate: Date,
  country: String
}, {
  timestamps: true
});


const bookSchema = new mongoose.Schema({
  title: {
```

```
  type: String,
  required: true,
  trim: true
},
author: {
  type: authorSchema,
  required: true
},
description: String,
publishedYear: {
  type: Number,
  min: 1000,
  max: new Date().getFullYear() + 1
},
category: {
  type: String,
  required: true,
  enum: ['Fiction', 'Non-Fiction', 'Science', 'Technology', ...]
},
price: {
  type: Number,
  min: 0
},
inStock: {
  type: Boolean,
  default: true
```

```
  }
}, {
  timestamps: true
});
```

2.2. Індексування для оптимізації

- Текстовий пошук: `{ title: 'text', 'author.name': 'text', description: 'text' }`
- Категорії: `{ category: 1 }`
- Ціна: `{ price: 1 }`
- Рік видання: `{ publishedYear: 1 }`

3. Створення GraphQL Schema

3.1. Типи даних (Type Definitions)

```
scalar Date

type Author {
  id: ID!
  name: String!
  bio: String
  birthDate: Date
  country: String
  createdAt: Date!
  updatedAt: Date!
}
```

```
type Book {
  id: ID!
  title: String!
  author: Author!
  description: String
  publishedYear: Int
  category: Category!
  price: Float
  inStock: Boolean!
  isbn: String
  pages: Int
  language: String
  publisher: String
  rating: Float
  createdAt: Date!
  updatedAt: Date!
}

enum Category {
  FICTION
  NON_FICTION
  SCIENCE
  TECHNOLOGY
  HISTORY
  BIOGRAPHY
  FANTASY
```

```
  MYSTERY
  ROMANCE
  THRILLER
}
```

3.2. Input типи для мутацій

```
input BookInput {
  title: String!
  author: AuthorInput!
  description: String
  publishedYear: Int
  category: Category!
  price: Float
  inStock: Boolean = true
  isbn: String
  pages: Int
  language: String = "English"
  publisher: String
  rating: Float
}

input AuthorInput {
  name: String!
  bio: String
  birthDate: Date
```

```
  country: String
}
```

3.3. Фільтри та сортування

```
input BookFilters {
  category: Category
  minPrice: Float
  maxPrice: Float
  minYear: Int
  maxYear: Int
  inStock: Boolean
  search: String
  language: String
}

enum BookSortField {
  TITLE
  PUBLISHED_YEAR
  PRICE
  RATING
  CREATED_AT
}

input BookSort {
  field: BookSortField!
```

```
  order: SortOrder = ASC
}
```

4. Реалізація Query операцій

4.1. Отримання книг з фільтрацією та пагінацією

```
books: async (parent, { filters = {}, sort, page = 1, limit = 10 }) => {
  const query = {};

  // Застосування фільтрів
  if (filters.category) query.category = filters.category;
  if (filters.minPrice || filters.maxPrice) {
    query.price = {};
    if (filters.minPrice) query.price.$gte = filters.minPrice;
    if (filters.maxPrice) query.price.$lte = filters.maxPrice;
  }
  if (filters.search) {
    query.$text = { $search: filters.search };
  }

  // Підрахунок загальної кількості
  const totalCount = await Book.countDocuments(query);

  // Сортування
  let sortObj = {};
```

```
  if (sort) {
    const sortOrder = sort.order === 'DESC' ? -1 : 1;
    sortObj[sort.field.toLowerCase()] = sortOrder;
  }

  // Пагінація
  const skip = (page - 1) * limit;
  const books = await Book.find(query)
    .sort(sortObj)
    .skip(skip)
    .limit(limit);

  return {
    books,
    totalCount,
    hasNextPage: skip + limit < totalCount,
    hasPreviousPage: page > 1
  };
}
```

4.2. Пошук книг

```
searchBooks: async (parent, { query, limit = 10 }) => {
  const books = await Book.find({
    $text: { $search: query }
  }, {
```

```
    score: { $meta: 'textScore' }
  })
  .sort({ score: { $meta: 'textScore' } })
  .limit(limit);

  return books;
}
```

4.3. Статистика за категоріями

```
categoryStats: async () => {
  const stats = await Book.aggregate([
    {
      $group: {
        _id: '$category',
        count: { $sum: 1 },
        averagePrice: { $avg: '$price' },
        averageRating: { $avg: '$rating' }
      }
    },
    {
      $project: {
        category: '$_id',
        count: 1,
        averagePrice: { $round: ['$averagePrice', 2] },
        averageRating: { $round: ['$averageRating', 1] },
```

```
        _id: 0
      }
    },
    { $sort: { count: -1 } }
  ]);

  return stats;
}
```

5. Реалізація Mutation операцій

5.1. Створення книги

```
addBook: async (parent, { input }) => {
  try {
    const book = new Book({
      ...input,
      category: input.category.replace('_', '-')
    });

    const savedBook = await book.save();
    return savedBook;
  } catch (error) {
    if (error.code === 11000) {
      throw new Error('Book with this ISBN already exists');
    }
```

```
    throw new Error(`Error creating book: ${error.message}`);
  }
}
```

5.2. Оновлення книги

```
updateBook: async (parent, { id, input }) => {
  const book = await Book.findByIdAndUpdate(
    id,
    input,
    { new: true, runValidators: true }
  );

  if (!book) {
    throw new Error('Book not found');
  }

  return book;
}
```

5.3. Видалення книги

```
deleteBook: async (parent, { id }) => {
  const book = await Book.findByIdAndDelete(id);
  return !!book;
}
```

5.4. Масове додавання книг

```
addBooks: async (parent, { books }) => {
  const booksToCreate = books.map(book => ({
    ...book,
    category: book.category.replace('_', '-')
  }));

  const savedBooks = await Book.insertMany(booksToCreate, { ordered: false });
  return savedBooks;
}
```

6. Налаштування Apollo Server

6.1. Створення серверу

```
const server = new ApolloServer({
  typeDefs,
  resolvers,
  context: ({ req }) => ({
    user: req.user || null,
    req
  }),
  introspection: process.env.NODE_ENV !== 'production',
  playground: process.env.NODE_ENV !== 'production',
```

```
  formatError: (error) => ({
    message: error.message,
    code: error.extensions?.code,
    path: error.path
  })
});
```

6.2. Інтеграція з Express

```
await server.start();
server.applyMiddleware({ app, path: '/graphql' });

app.get('/', (req, res) => {
  res.json({
    message: '🚀 GraphQL Lab 5 Server',
    graphqlEndpoint: '/graphql',
    playgroundUrl: '/graphql',
    status: 'running'
  });
});
```

7. Кастомні скалярні типи

7.1. Date скаляр

```
const dateScalar = new GraphQLScalarType({
```

```
  name: 'Date',
  description: 'Date custom scalar type',
  serialize(value) {
    return value instanceof Date ? value.toISOString() : null;
  },
  parseValue(value) {
    return new Date(value);
  },
  parseLiteral(ast) {
    if (ast.kind === Kind.STRING) {
      return new Date(ast.value);
    }
    return null;
  }
});
```

8. Приклади використання

8.1. Query запит з фільтрами

```
query GetBooks {
  books(
    filters: {
      category: FICTION,
      minPrice: 10,
      maxPrice: 50,
```

```
      inStock: true
    }
    sort: { field: TITLE, order: ASC }
    page: 1
    limit: 5
  ) {
    books {
      id
      title
      author {
        name
        country
      }
      price
      category
      rating
    }
    totalCount
    hasNextPage
  }
}
```

8.2. Mutation додавання книги

```
mutation AddBook {
  addBook(input: {
```

```
    title: "1984"
    author: {
      name: "George Orwell"
      bio: "English novelist and essayist"
      country: "United Kingdom"
    }
    category: FICTION
    publishedYear: 1949
    price: 12.99
    pages: 328
    inStock: true
    description: "Dystopian social science fiction novel"
  }) {
    id
    title
    author {
      name
    }
    category
    publishedYear
  }
}
```

8.3. Складний запит зі статистикою

```
query ComplexQuery {
```

```
  # Отримання книг
  books(limit: 3) {
    books {
      title
      author { name }
      price
    }
    totalCount
  }

  # Статистика за категоріями
  categoryStats {
    category
    count
    averagePrice
    averageRating
  }

  # Пошук
  searchBooks(query: "science fiction", limit: 2) {
    title
    author { name }
    description
  }
}
```

9. Оптимізація та безпека

9.1. Database індекси

- Текстовий пошук для швидкого знаходження книг
- Індекси на часто використовувані поля (category, price)
- Унікальний індекс на ISBN

9.2. Валідація

- Схема валідації на рівні Mongoose
- Перевірка типів через GraphQL Schema
- Кастомна валідація в резолверах

9.3. Обробка помилок

- Graceful error handling в резолверах
- Форматування помилок для клієнта
- Логування помилок на сервері

10. Тестування в GraphQL Playground

10.1. Основні можливості

- Автодоповнення запитів
- Документація API
- Валідація запитів в реальному часі
- Історія запитів

10.2. Приклади тестування

1. **Створення тестових даних:**
    - Додавання книг різних категорій
    - Створення авторів з різних країн
2. **Тестування Query операцій:**
    - Фільтрація за різними критеріями
    - Пагінація великих наборів даних
    - Пошук за текстом
3. **Тестування Mutation операцій:**
    - CRUD операції

- Валідація даних
- Обробка помилок

Структура проекту

```
lab5/
├── server.js              # Головний файл сервера
├── schema/                 # GraphQL схеми
│   ├── typeDefs.js        # Type definitions
│   └── resolvers.js       # Resolvers
├── models/                 # MongoDB моделі
│   └── Book.js            # Модель книги та автора
├── package.json           # Конфігурація проекту
├── .env                   # Змінні середовища
├── .gitignore            # Файли для ігнорування
└── README.md             # Документація
```

Результати роботи

Створено повнофункціональний GraphQL сервер з наступними можливостями:

1. **Повна GraphQL Schema** з типами, input types та enum
2. **Комплексні Query операції:**
   - Фільтрація та сортування
   - Пагінація
   - Текстовий пошук

    - Агрегація даних
3. **CRUD Mutation операції:**
    - Створення, оновлення, видалення книг
    - Масове додавання
    - Часткові оновлення
4. **Розширена функціональність:**
    - Кастомні скалярні типи
    - Складні фільтри
    - Статистичні запити
    - Валідація та обробка помилок
5. **Оптимізація продуктивності:**
    - Database індекси
    - Ефективні MongoDB запити
    - Пагінація великих наборів даних
6. **Інтерактивне тестування:**
    - GraphQL Playground
    - Автодокументація API
    - Валідація запитів

Скріншоти GraphQL Playground:

- Схема API з типами та запитами
- Query операції з фільтрацією
- Mutation операції створення книг
- Складні запити з вкладеними полями
- Статистичні запити та агрегація

Переваги GraphQL над REST

1. **Гнучкість запитів:** клієнт отримує тільки потрібні дані
2. **Сильна типізація:** автоматична валідація та документація
3. **Єдина точка входу:** один endpoint для всіх операцій
4. **Інтроспекція:** автоматична генерація документації
5. **Real-time підписки:** можливість отримання оновлень

Висновки

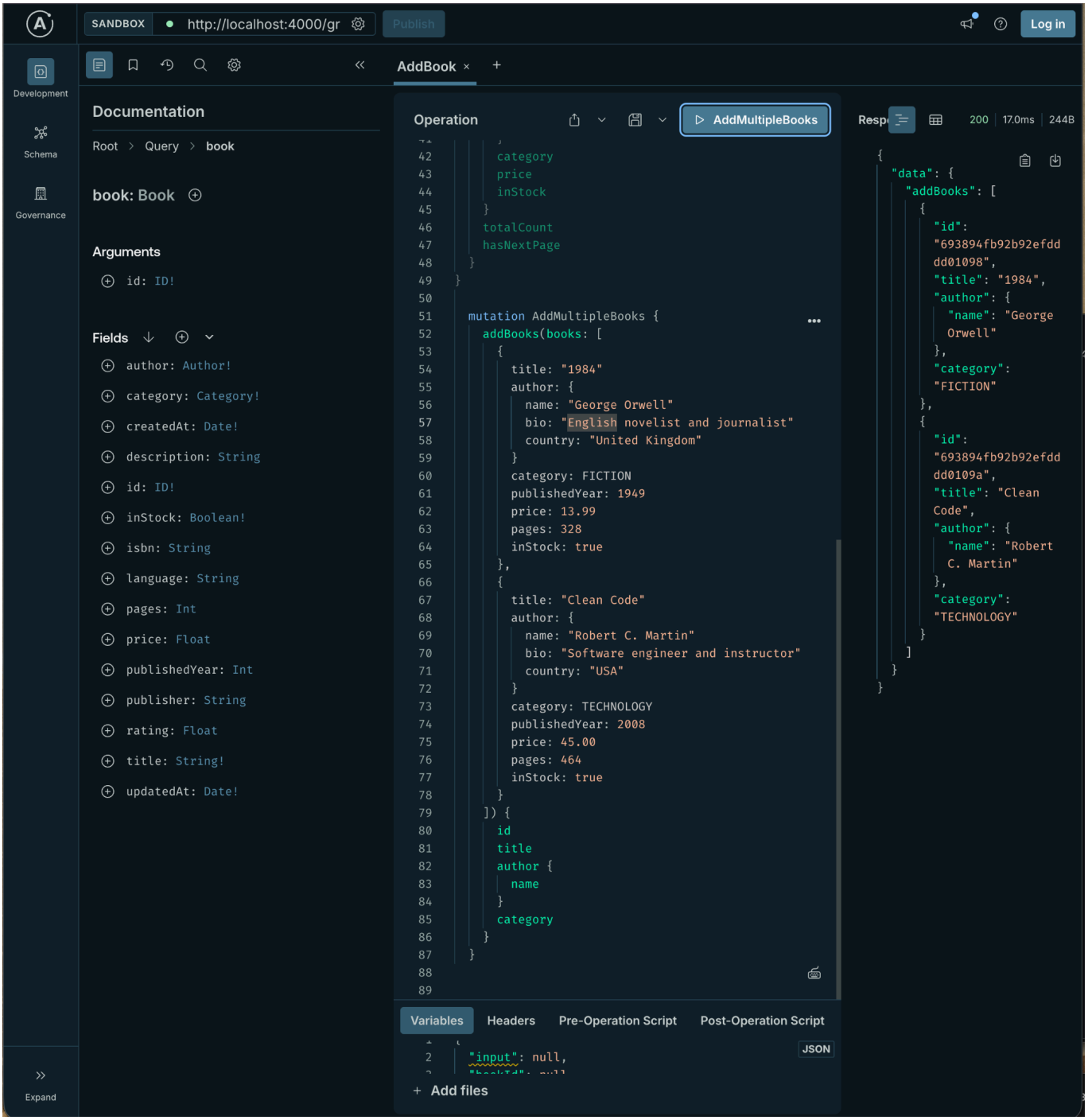
В ході виконання лабораторної роботи:

- Освоєно принципи роботи з GraphQL
- Створено повну GraphQL Schema з типами та зв'язками
- Реалізовано Resolvers для обробки запитів
- Впроваджено CRUD операції через Mutation
- Інтегровано MongoDB через Mongoose ODM
- Вивчено Apollo Server та його можливості
- Освоєно GraphQL Playground для тестування API

GraphQL показав себе як потужна альтернатива REST API, що надає більшу гнучкість у роботі з даними та автоматичну генерацію документації. Використання Apollo Server значно спростило створення GraphQL сервера з усіма необхідними можливостями.

Проект демонструє практичне застосування сучасних технологій для створення API нового покоління з сильною типізацією та гнучкими можливостями запитів.

Скріншоти:


SANDBOX
http://localhost:4000/gr
Publish
Log in
Development
Schema
Governance
Documentation
Root > Query > book
book: Book
Arguments
id: ID!
Fields
author: Author!
category: Category!
createdAt: Date!
description: String
id: ID!
inStock: Boolean!
isbn: String
language: String
pages: Int
price: Float
publishedYear: Int
publisher: String
rating: Float
title: String!
updatedAt: Date!
Expand
AddBook
Operation
AddMultipleBooks
mutation AddMultipleBooks {
addBooks(books: [
{
title: "1984"
author: {
name: "George Orwell"
bio: "English novelist and journalist"
country: "United Kingdom"
}
category: FICTION
publishedYear: 1949
price: 13.99
pages: 328
inStock: true
},
{
title: "Clean Code"
author: {
name: "Robert C. Martin"
bio: "Software engineer and instructor"
country: "USA"
}
category: TECHNOLOGY
publishedYear: 2008
price: 45.00
pages: 464
inStock: true
}
]) {
id
title
author {
name
}
category
}
}
Variables
Headers
Pre-Operation Script
Post-Operation Script
JSON
"input": null,
Add files
200
17.0ms
244B
"data": {
"addBooks": [
{
"id": "693894fb92b92efddd01098",
"title": "1984",
"author": {
"name": "George Orwell"
},
"category": "FICTION"
},
{
"id": "693894fb92b92efddd0109a",
"title": "Clean Code",
"author": {
"name": "Robert C. Martin"
},
"category": "TECHNOLOGY"
}
]
}
}