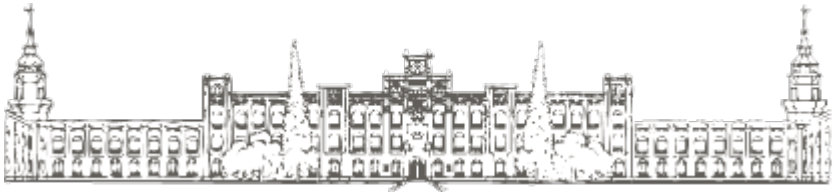

Міністерство освіти і науки України

Національний технічний університет України „КПІ імені Ігоря Сікорського ”

Факультет інформатики та обчислювальної техніки Кафедра інформатики і програмної інженерії

ЗВІТ

лабораторної роботи № 5

з курсу «Основи WEB - технологій»

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

Перевірив:

Викл. Альбрехт Й.О.

Виконав:

Студент ІП-15 Мєшков А.І

Київ 2024

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.

Варіант 7

Варіант 7. Спроектувати базу даних про оцінки, отриманих студентами на іспитах: прізвище, група, предмет, номер квитка, оцінка, викладач.

Хід роботи

Було програму nodejs з використанням Apollo-server, GraphQL, MongoDB.

server.js

```
require('dotenv').config();
const { ApolloServer } = require('apollo-server');
const connectDB = require('./utils/db');

const typeDefs = require('./typeDefs');
const resolvers = require('./resolvers');

connectDB();

const server = new ApolloServer({
  typeDefs,
  resolvers,
});

server.listen().then(({ url }) => {
  console.log(`Server ready at ${url}`);
});
```

Db.js

```
const mongoose = require('mongoose');
require('dotenv').config();

const dbUrl = process.env.DB_URL || '';

const connectDB = async () => {
    try {
        const data = await mongoose.connect(dbUrl);
        console.log(`Database connected with ${data.connection.host}`);
    } catch (error) {
        console.log(error.message);
        setTimeout(connectDB, 5000);
    }
};


module.exports = connectDB;
```

Grade.js

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

const gradeSchema = new mongoose.Schema({
  lastName: { type: String, required: true},
  group: { type: String, required: true},
  subject: { type: String, required: true},
  ticketNumber: { type: Number, required: true},
```

```
  grade: { type: Number, required: true},
  professor: { type: String, required: true},
}, {timestamps: true});

module.exports = mongoose.model('Grade', gradeSchema);
```

typeDefs.js

```
const { gql } = require('apollo-server');

const typeDefs = gql`
  type Grade {
    _id: ID!
    lastName: String!
    group: String!
    subject: String!
    ticketNumber: Int!
    grade: Int!
    professor: String!
    createdAt: String
    updatedAt: String
  }

  type Query {
    getAllGrades: [Grade!]!
    getGrade(_id: ID!): Grade
  }

  type Mutation {
    addGrade(
      lastName: String!
      group: String!
      subject: String!
      ticketNumber: Int!
      grade: Int!
      professor: String!
    ): Grade!

    updateGrade(
      _id: ID!
      lastName: String
      group: String
      subject: String
      ticketNumber: Int
      grade: Int
      professor: String
    ): Grade!

    deleteGrade(_id: ID!): String!
  }
`;
```

```
module.exports = typeDefs;
```

resolvers.js

```
const Grade = require('./models/Grade');

const resolvers = {
    Query: {
        async getAllGrades() {
            try{
                const result = await Grade.find();
                if (!result.length) {
                    throw new Error("No Posts Added!");
                }
                return result;
            } catch (e) {
                throw(e);
            }
        },
        async getGrade(_, { _id }) {
            try{
                const result = await Grade.findById(_id);
                if (!result) {
                    throw new Error("Post does not exists!");
                }
                return result;
            } catch (e) {
                throw(e);
            }
        },
    },
    Mutation: {
        async addGrade(_, { lastName, group, subject, ticketNumber, grade, professor }) {
            try{
                let now = new Date();
                const newGrade = new Grade({
                    lastName,
                    group,
                    subject,
                    ticketNumber,
                    grade,
                    professor,
                    createdAt:now,
                    updatedAt:now,
                });
                return await newGrade.save();
            } catch (e) {
                throw(e);
            }
        },
        async updateGrade(_, { _id, ...updates }) {
```

```
            try{
                const result = await Grade.findByIdAndUpdate(_id, updates, { new: true });
                if (!result) {
                    throw new Error("Post does not exists!");
                }
                return result;
            } catch (e) {
                throw(e);
            }
        },
        async deleteGrade(_, { _id }) {
            try{
                await Grade.findByIdAndDelete(_id);
                return `Grade with ID ${_id} deleted successfully.`;
            } catch (e) {
                throw(e);
            }
        },
    },
};

module.exports = resolvers;
```

1. Отримані результати

На рис 5.1-5.5 можна побачити запити postman.

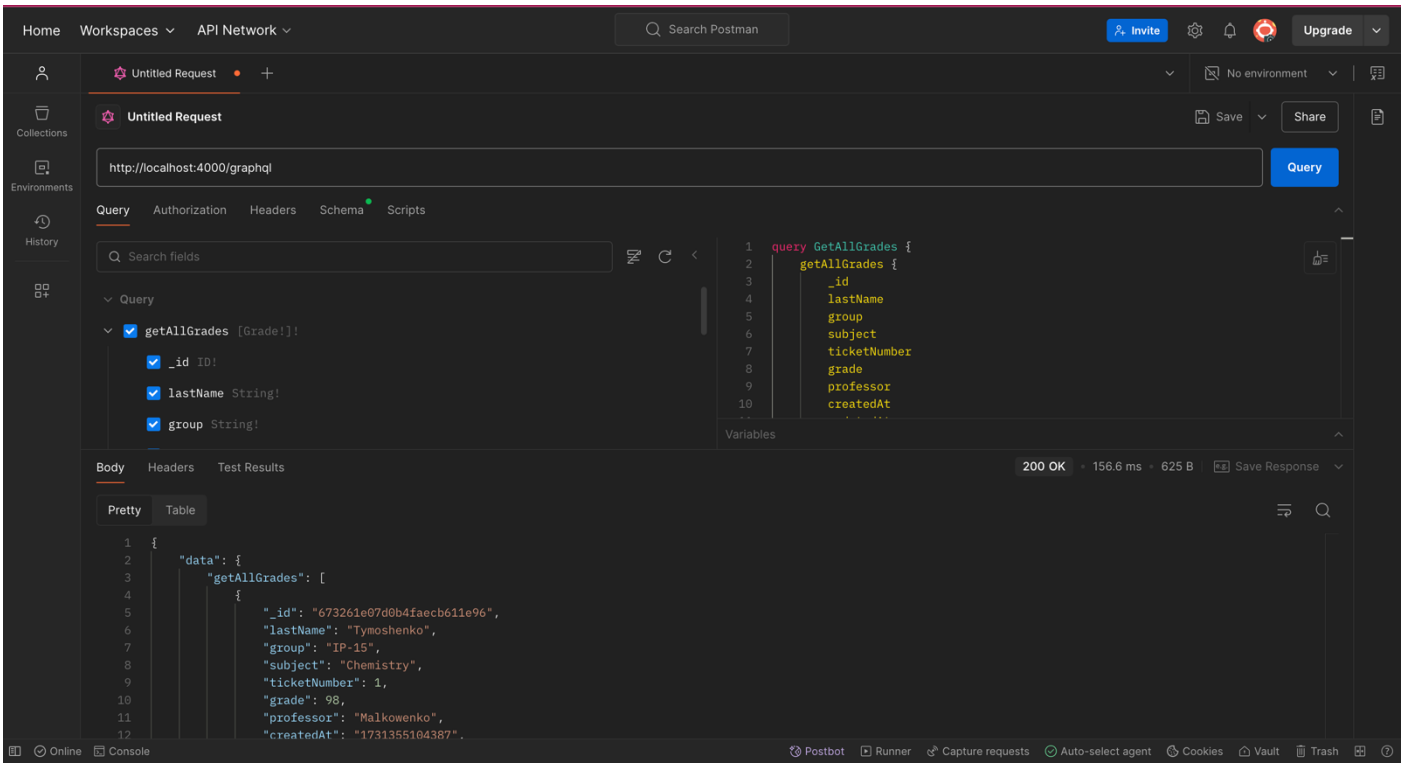

Рис. 5.1. Query для виведення всіх документів в БД

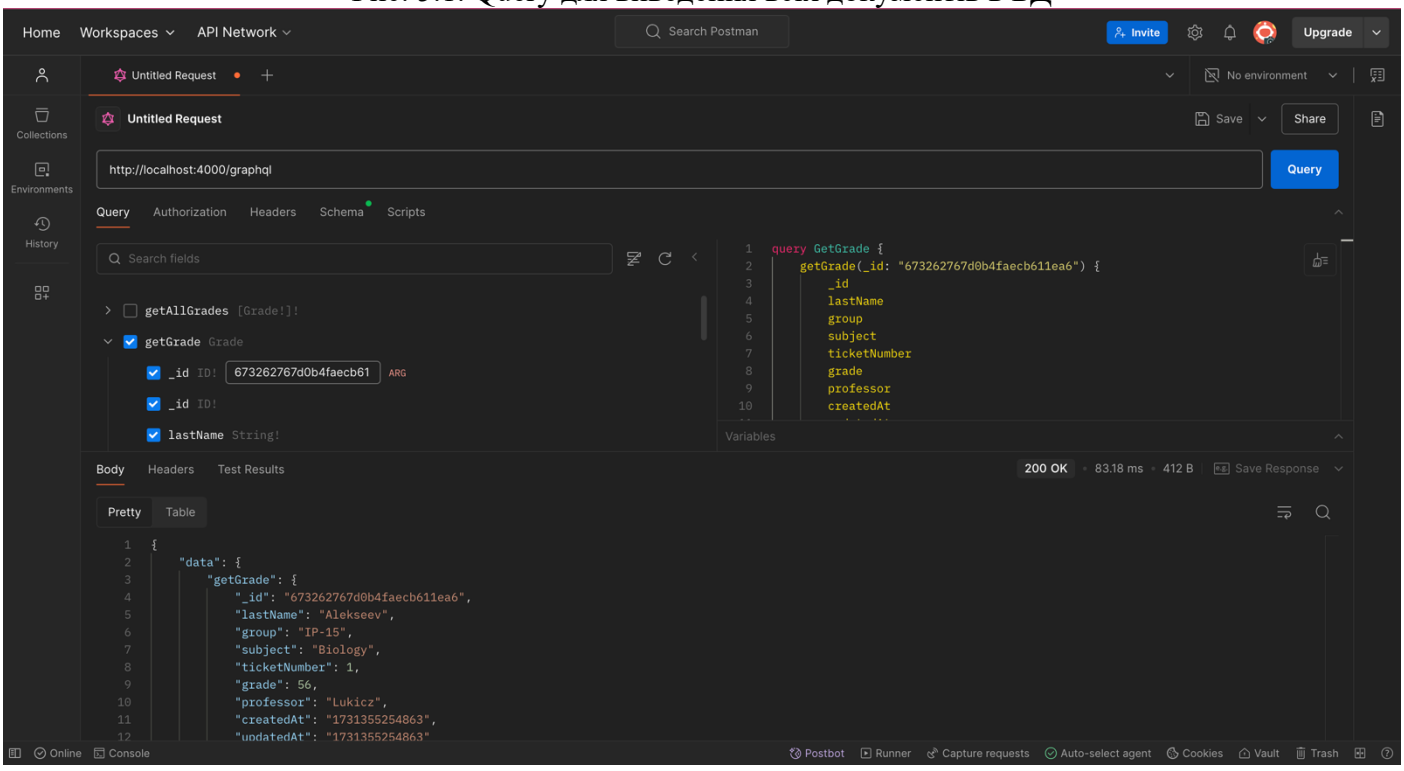

Рис. 5.2. Query для виведення заданого документу в БД

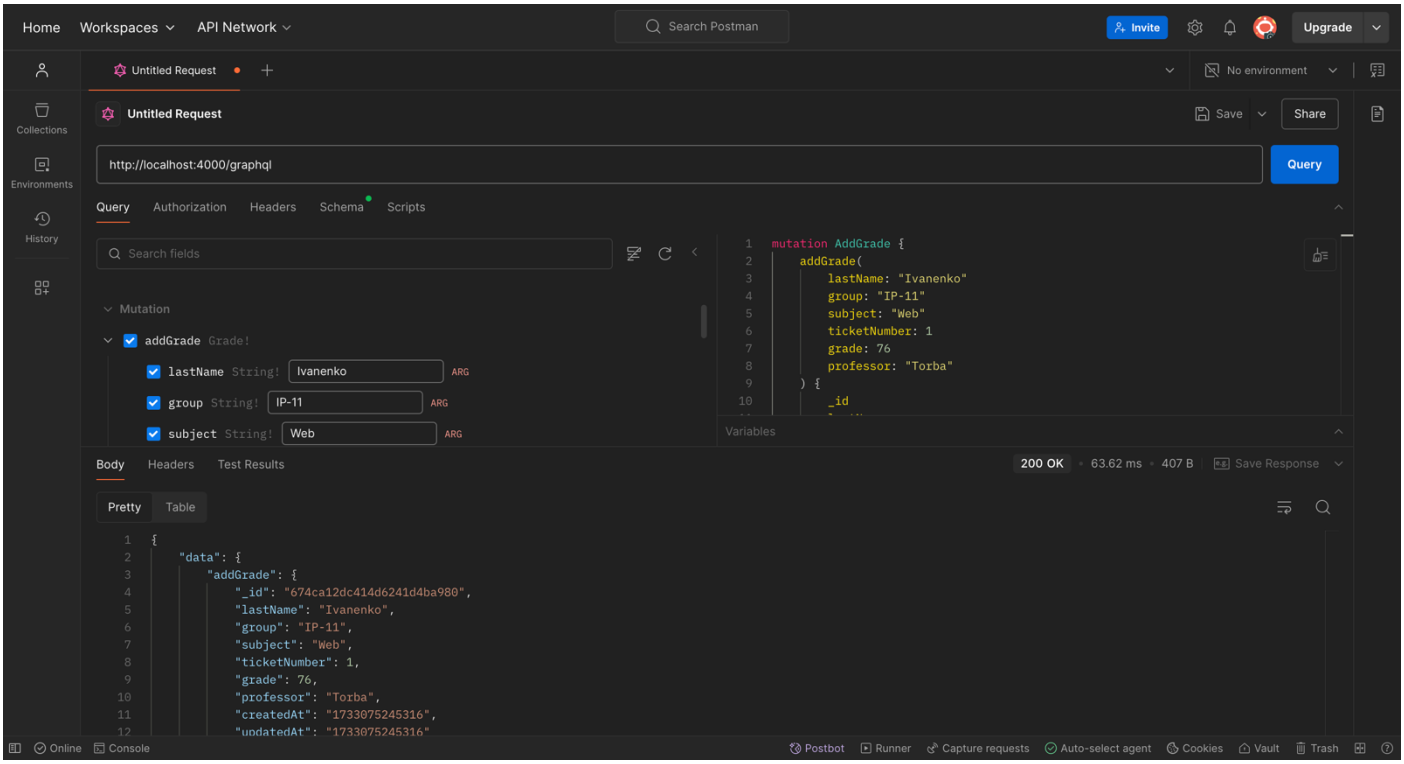

Рис. 5.3. Mutation для додавання документу в БД

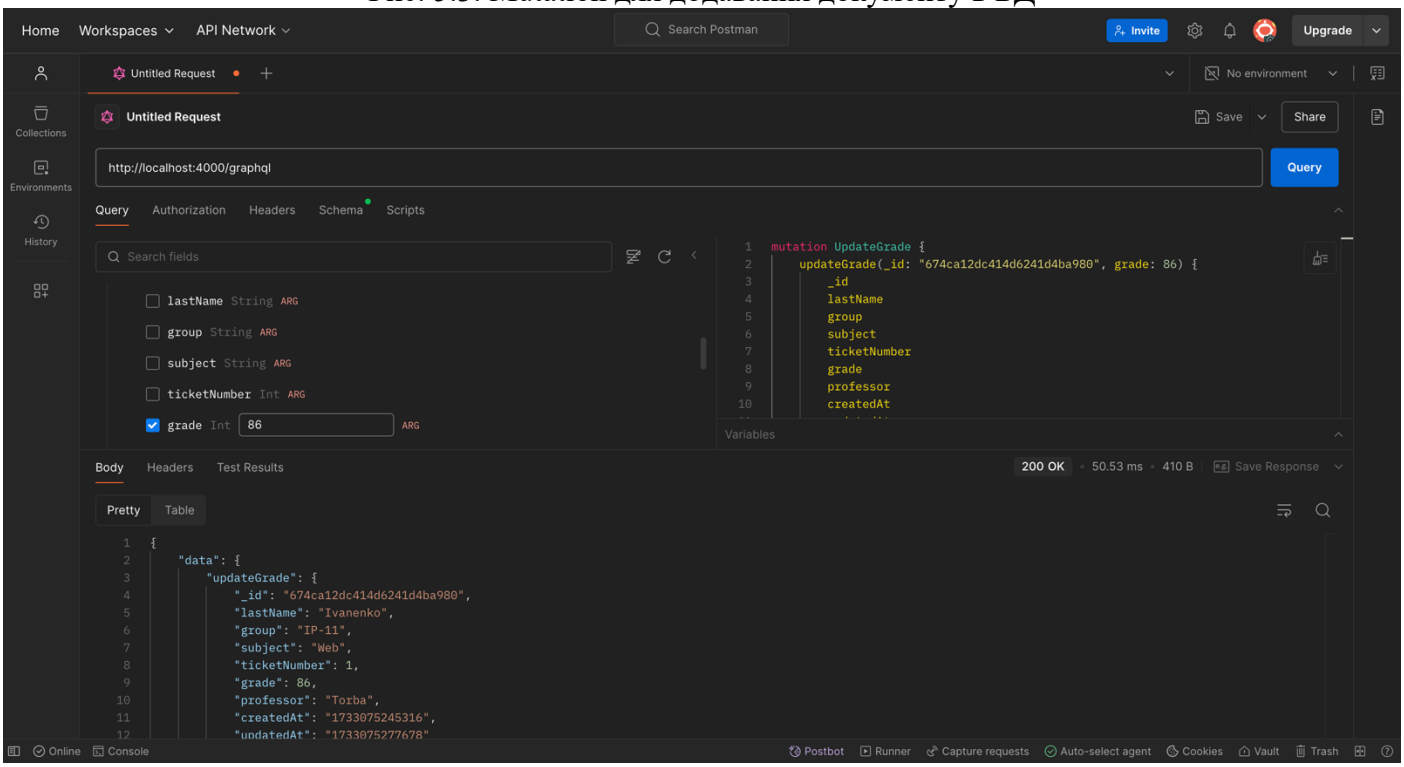

Рис. 5.4. Mutation для оновлення документу в БД

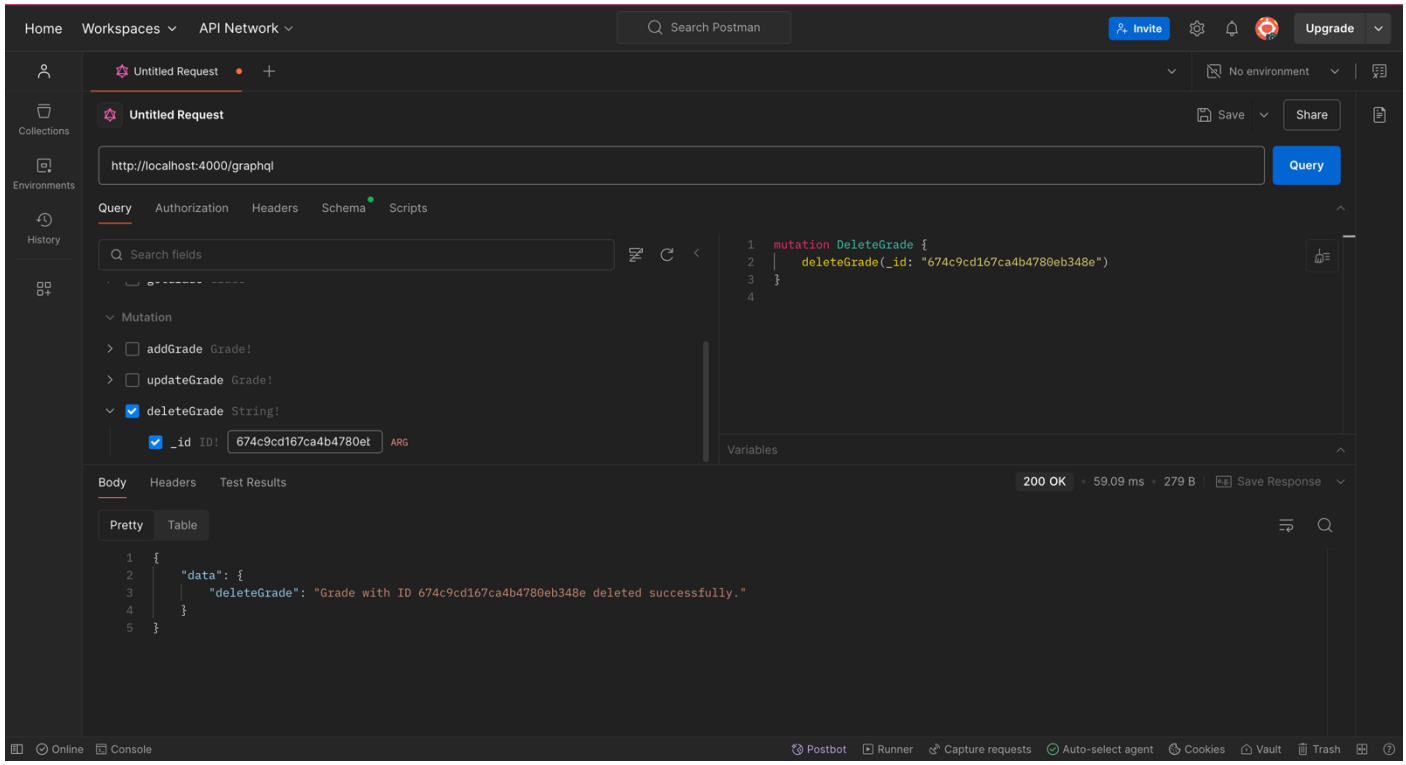

Рис. 5.5. Mutation для видалення документу в БД

ВИСНОВОК

Під час виконання даної лабораторної роботи було розроблено серверну частину веб-застосунку з використанням технології GraphQL. Основними результатами роботи стали:

1. Створення GraphQL-схеми (SDL) для роботи з даними в базі даних MongoDB, що відповідає завданням CRUD (Create, Read, Update, Delete).

2. Реалізація Resolver'ів для обробки запитів та мутацій, що дозволяють виконувати операції над документами у базі даних.

3. Налаштування та запуск GraphQL-сервера за допомогою бібліотеки Apollo Server.

4. Проведення тестування створених Query та Mutation за допомогою Postman, яке підтвердило коректність роботи серверної частини та відповідність отриманих результатів вимогам завдання.

Використання GraphQL значно спростило обробку запитів, надавши можливість отримувати лише необхідні дані та забезпечило гнучкість у роботі з базою даних. Цей підхід дозволяє оптимізувати взаємодію клієнтської частини з серверною, зменшуючи обсяг переданої інформації.

Лабораторна робота сприяла закріпленню навичок роботи з технологіями Node.js, GraphQL, MongoDB, а також налаштування API-серверів для виконання CRUD-операцій.