GraphQL Lab - Day 1 Homework

Student Management System

🎯 Objective

Build a complete GraphQL API for managing students and courses with full CRUD operations.

📋 Requirements

Part 1: Schema Design (20 points)

Create types for:

Student Type

- `id`: ID (required)
- `name`: String (required)
- `email`: String (required)
- `age`: Int (required)
- `major`: String (optional)
- `courses`: Array of Course (required, can be empty)

Course Type

- `id`: ID (required)
- `title`: String (required)
- `code`: String (required) - e.g., "CS101"
- `credits`: Int (required)
- `instructor`: String (required)
- `students`: Array of Student (required, can be empty)

Part 2: Queries (25 points)

Implement the following queries:

1. **getAllStudents** - Returns all students
2. **getStudent(id: ID!)** - Returns a specific student by ID
3. **getAllCourses** - Returns all courses
4. **getCourse(id: ID!)** - Returns a specific course by ID
5. **searchStudentsByMajor(major: String!)** - Returns students filtered by major

---

Part 3: Mutations (35 points)

Implement the following mutations:

Student Operations

1. **addStudent(name, email, age, major)** - Create a new student
2. **updateStudent(id, name, email, age, major)** - Update student info (all fields optional except id)

3. **deleteStudent(id)** - Delete a student and return success boolean

Course Operations

4. **addCourse(title, code, credits, instructor)** - Create a new course
5. **updateCourse(id, title, code, credits, instructor)** - Update course info
6. **deleteCourse(id)** - Delete a course

Install dependencies: npm i apollo-server-express express graphql --legacy-peer-deps

🚀 Starter Code

File: `server.js`

```
const express = require("express");
const { ApolloServer, gql } = require("apollo-server-express");

// In-memory data storage
let students = [
  {
    id: "1",
    name: "Ahmed Hassan",
    email: "ahmed@iti.edu",
    age: 22,
    major: "Computer Science"
  },
  {
    id: "2",
    name: "Fatma Ali",
    email: "fatma@iti.edu",
    age: 21,
    major: "Information Systems"
  }
];

let courses = [
  {
    id: "1",
    title: "Data Structures",
    code: "CS201",
    credits: 3,
    instructor: "Dr. Mohamed"
  },
  {
    id: "2",
    title: "Database Systems",
    code: "CS301",
```

```
    credits: 4,
    instructor: "Dr. Sarah"
  }
];

// Enrollment tracking (studentId -> [courseIds])
let enrollments = {
  "1": ["1", "2"],  // Ahmed enrolled in both courses
  "2": ["2"]        // Fatma enrolled in Database Systems
};

// ============= TODO: DEFINE YOUR SCHEMA HERE =============
const typeDefs = gql`
  type Query {
    hello: String!
  }
`;

// ============= TODO: IMPLEMENT YOUR RESOLVERS HERE
=============
const resolvers = {
  Query: {
    hello: () => 'hello world'
  }
};

// ============= SERVER SETUP (DO NOT MODIFY) =============
async function start() {
  const app = express();
  const server = new ApolloServer({
    typeDefs: typeDefs,
    resolvers: resolvers,
  });

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

  app.listen(5000, () => {
    console.log("App Running on http://localhost:5000/graphql");
  });
}

start();
```

📝 Step-by-Step Guide

Step 1: Define Types

```
type Student {
  # TODO: Add Student fields
}

type Course {
  # TODO: Add Course fields
}
```

Step 2: Define Query Type

```
type Query {
  # TODO: Add all query operations
  getAllStudents: [Student!]!
  # ... add more
}
```

Step 3: Define Mutation Type

```
type Mutation {
  # TODO: Add all mutation operations
  addStudent(name: String!, email: String!, age: Int!, major: String): Student!
  # ... add more
}
```

Step 4: Implement Query Resolvers

```
const resolvers = {
  Query: {
    getAllStudents: () => {
      // TODO: Return all students
    },
    getStudent: (_, { id }) => {
      // TODO: Find and return student by id
    },
    // ... implement others
  }
};
```

Step 5: Implement Mutation Resolvers

```
Mutation: {
  addStudent: (_, { name, email, age, major }) => {
    // TODO: Create new student
```

```
    // Generate new ID
    // Add to students array
    // Return new student
  },

  deleteStudent: (_, { id }) => {
    // TODO: Remove student from array
    // Return true if successful, false otherwise
  },

  enrollStudent: (_, { studentId, courseId }) => {
    // TODO: Add courseId to enrollments[studentId]
    // Return the updated student
  },

  // ... implement others
}
```

Step 6: Implement Nested Resolvers

```
Student: {
  courses: (parent) => {
    // parent = the student object
    // TODO: Get courseIds from enrollments[parent.id]
    // Return matching courses
  }
},

Course: {
  students: (parent) => {
    // parent = the course object
    // TODO: Find all students enrolled in this course
    // Check enrollments to see which students have this course
  }
}
```