CODEBOOSTERS TECH - NODE JS & GRAPHQL PROJECT LIST

1. E-Commerce Product Catalog API with GraphQL

Project Objective:

The goal of this project is to design and implement a robust product catalog API for an e-commerce platform using GraphQL and Node.js. This project aims to provide a flexible and scalable API to manage and interact with product data, enabling customers to browse products, apply filters, and sort the catalog dynamically.

Detailed Objectives:

1. **Set up a GraphQL Server**: Use Node.js with Express.js to set up a GraphQL server. Students will learn how to configure and implement the server to handle various queries and mutations.
2. **Design GraphQL Schema**: Students will design a schema that defines product types (e.g., name, description, price, category), and allow filtering and sorting. The schema will also define mutations for adding, updating, and deleting products.
3. **Pagination and Filtering**: Implement pagination to allow users to request a subset of products and avoid overwhelming the system with large queries. Students will create filtering options for price range, category, and other attributes like rating or brand.
4. **Integration with MongoDB/MySQL**: Students will learn how to integrate a database (MongoDB for flexibility or MySQL for relational data) to persist product data. This will include designing tables/collections for products, categories, and other related entities.
5. **GraphQL Queries and Mutations**:
    - **Queries**: Implement queries for searching, listing, and viewing product details. Students will learn how to handle complex queries with sorting and filtering.
    - **Mutations**: Implement mutations for adding, updating, and deleting products from the catalog. This will involve interacting with the database and returning appropriate success or failure responses.
6. **Error Handling**: Implement robust error handling in GraphQL to ensure that users receive meaningful error messages when something goes wrong (e.g., when a product

cannot be found or deleted).

7. **Authentication and Authorization (Optional)**: Integrate user authentication (e.g., admin users who can add or modify products) to control access to certain mutations, ensuring only authorized users can make changes to the catalog.

---

2. Social Media Dashboard API with GraphQL

Project Objective:

This project aims to develop a GraphQL API for a social media platform's dashboard. It will enable users to post updates, like posts, follow other users, and interact with content. The objective is to create a scalable API that handles user interactions and social activities using GraphQL queries and mutations.

Detailed Objectives:

1. **Set Up GraphQL Server**: Create a GraphQL server using Node.js and Express, enabling flexible interactions with the system. Students will learn to configure an efficient GraphQL server for handling user data and content.
2. **Design a GraphQL Schema for Social Interactions**:
    - Define user types (e.g., name, email, bio), post types (e.g., content, timestamp, likes), and follow relationships (e.g., follower and following).
    - Allow users to post updates, follow/unfollow other users, and like/unlike posts using GraphQL queries and mutations.
3. **User Authentication**: Implement a basic authentication mechanism (e.g., JWT) for users to sign up, log in, and manage their session. This will ensure secure interactions when posting, liking, or following.
4. **GraphQL Queries and Mutations**:
    - **Queries**: Develop queries to fetch posts, user profiles, follower information, and liked posts. These will support filtering based on time, likes, or followers.
    - **Mutations**: Allow users to create new posts, like posts, and follow/unfollow other users. The mutation logic will interact with the database to persist data.
5. **Handling Relationships**: Implement functionality for managing relationships, such as following/unfollowing other users. Students will work with GraphQL's nested queries to

fetch related data (e.g., a user's followers and their posts).

6. **Database Integration**: Choose between MongoDB or MySQL for storing user data, posts, likes, and follow relationships. Students will learn how to structure the database schema for social media interactions.
7. **Error Handling**: Proper error handling in GraphQL will ensure smooth user experience, including validation for incorrect inputs (e.g., posting an empty content or following an already followed user).
8. **Pagination and Sorting**: Implement pagination for fetching posts and followers. Sorting will allow users to view posts in a specific order (e.g., most recent or most liked).

---

3. Online Learning Platform API with GraphQL

Project Objective:

The objective of this project is to design a GraphQL API for an online learning platform where users can browse courses, enroll in them, and track their learning progress. The goal is to create a flexible, scalable backend that supports course management, student progress tracking, and interactive features like quizzes or assessments.

Detailed Objectives:

1. **Set Up GraphQL Server**: Build a GraphQL server using Node.js and Express to handle requests related to courses, students, and enrollment. Students will learn how to configure a server and optimize it for complex educational data.
2. **Design GraphQL Schema for Courses and Students**:
    - Define types for courses (e.g., title, description, duration, lessons), users (students, instructors), and progress tracking (e.g., courses completed, lessons viewed).
    - Allow users to browse available courses, enroll in courses, and view their progress.
3. **GraphQL Queries and Mutations**:
    - **Queries**: Implement queries for browsing the list of courses, fetching specific course details (e.g., lessons, instructors), and retrieving a user's enrolled courses and progress.
    - **Mutations**: Implement mutations for enrolling in courses, updating progress (e.g., completing lessons), and possibly leaving or un-enrolling from a course.

4. **Database Integration**: Integrate MongoDB/MySQL to store course data, user data, and progress tracking information. The database schema will involve tables/collections for courses, students, and progress.
5. **Progress Tracking**: Implement logic for tracking the user's progress through courses. This could involve storing which lessons or modules have been completed and generating recommendations based on progress.
6. **User Authentication and Role-Based Access**:
    - Implement user authentication (e.g., JWT) for students to log in and track their progress securely.
    - Implement role-based access control for different user roles (e.g., instructors, students), with instructors being able to create or update courses, while students can only enroll in and progress through them.
7. **Error Handling**: Ensure proper error handling for cases like trying to enroll in a course that is full or accessing a course that the user isn't enrolled in. Provide meaningful error messages.
8. **Pagination and Filtering**: Use pagination for listing courses to handle large datasets and filtering based on categories (e.g., programming, design). This will allow students to narrow down their search and find courses of interest more easily.
9. **Optional Features**:
    - **Quizzes/Assessments**: Implement a system for adding quizzes or assessments at the end of courses to test students' understanding.
    - **Course Recommendations**: Create a recommendation system that suggests courses based on the student's progress and interest areas.

These objectives allow students to implement complex backend systems involving user management, course management, and dynamic querying while using the full potential of GraphQL and integrating databases efficiently.
