# GraphQL API with Apollo Server – Documentation

## Overview

This project is a GraphQL API built with Apollo Server, TypeScript, and Zod for validation. It simulates a basic backend for managing Users and Products using in-memory data, supporting common CRUD operations and filtered queries.  
  
The project also includes input validation, custom error handling, and dynamic ID reordering after deletions to maintain clean and predictable data.

## Purpose

The primary goals of this project include:  
- Practicing modular GraphQL API design using Apollo Server.  
- Implementing robust validation with Zod for all inputs.  
- Allowing clean querying, filtering, creating, updating, and deleting of data.  
- Providing informative error messages and ensuring user-friendly responses.  
- Simulating real-world scenarios like:  
 - Avoiding duplicate entries (e.g. email uniqueness).  
 - Filtering by product name or price.  
 - Querying users by name or email.  
 - Maintaining consistent IDs after deletion.

## Project Structure

├── src/

│ ├── data/ → Static in-memory data

│ │ ├── users.ts → Hardcoded users array

│ │ └── products.ts → Hardcoded products array

│

│ ├── loaders/ → DataLoader setup for batching

│ │ ├── userLoader.ts → Batch-load users by ID

│ │ └── productLoader.ts → Batch-load products by ID

│

│ ├── schema/ → GraphQL resolvers and validation

│ │ ├── resolvers/

│ │ │ ├── user.resolver.ts → User resolver functions

│ │ │ └── product.resolver.ts → Product resolver functions

│ │ ├── validation/ → Zod schemas for input validation

│ │ │ ├── user.schema.ts → User validation schema

│ │ │ └── product.schema.ts → Product validation schema

|

│ ├── utils/ → Utility functions

│ │ ├── validateInput.ts → Zod validation helper

│ │ ├── reorderIds.ts → ID reordering utility after deletion

│

│ ├── validation/ → Zod schemas

│ │ ├── user.schema.ts → Create & update schema for user

│ │ └── product.schema.ts → Create & update schema for product

│

│ └── index.ts → Apollo Server initialization

## Features

User Functionalities:  
- Create User  
- Get All Users  
- Get User by Email  
- Get Users by Name  
- Update User  
- Delete User (by id or email)  
  
Product Functionalities:  
- Create Product  
- Get All Products  
-   
- Filter Products by Name or Price  
- Update Product  
- Delete Product

## Setup Instructions

1. Clone the Repository:  
 git clone https://github.com/your-username/GraphQL-with-Apollo.git  
 cd GraphQL-with-Apollo  
  
2. Install Dependencies:  
 npm install  
  
3. Run the Server:  
 npm run dev  
  
4. Access the Playground:  
 Visit: http://localhost:4000/

## Sample Queries

Create a User:  
mutation {  
 createUser(input: { name: "Alice", email: "alice@example.com" }) {  
 id  
 name  
 email  
 }  
}  
  
Update a Product:  
mutation {  
 updateProduct(input: { id: "1", name: "Updated Product" }) {  
 id  
 name  
 price  
 }  
}  
  
Get User by Email:  
query {  
 getUserByEmail(email: "alice@example.com") {  
 id  
 name  
 email  
 }  
}  
  
Filter Products:  
query {  
 filterProducts(name: "bread") {  
 id  
 name  
 price  
 }  
}

## Postman Testing

Endpoint: http://localhost:4000/  
Method: POST  
Headers: { "Content-Type": "application/json" }  
Body (raw JSON):  
{  
 "query": "query { users { id name email } }"  
}

## Challenges & Solutions

1. Email Duplication on Create:  
- Problem: Duplicate users were being added.  
- Solution: Email check before creation, throws error if email exists.  
  
2. Updating with Blank Inputs:  
- Problem: Empty strings passed validation.  
- Solution: Inputs are cleaned and empty strings are rejected.  
  
3. ID Management After Deletion:  
- Problem: Gaps in IDs broke queries.  
- Solution: reorderIds utility maintains sequential IDs.  
  
4. Empty List Handling:  
- Problem: Queries returned empty arrays with no context.  
- Solution: Error messages like "No users found" were added.

## Conclusion

This project simulates a GraphQL API backend with:  
- Clear input validation  
- Modular and scalable architecture  
- Strong error feedback  
- Full query/mutation coverage  
  
It serves as a solid base for learning or expanding to real database-backed APIs.