Project Report on

GraphQL

API for User Management

By:

Pragyan Borthakur

Abstract

The project focuses on building an efficient and scalable API for a Threads-based application using GraphQL. The goal is to allow flexible querying of data, enabling users to create, view, and manage threads, replies, and user interactions in a seamless manner. The project leverages PostgreSQL as the database, Prisma for ORM, and Docker for containerized deployment. By utilizing GraphQL, the API allows clients to fetch only the required data, minimizing overhead and improving performance.

Introduction

The Threads App API is designed to manage user-generated threads efficiently. Unlike traditional REST APIs, GraphQL provides precise data fetching capabilities,

reducing under-fetching or over-fetching of data. This API enables users to create and manage threads and replies while maintaining flexibility and scalability.

Problem Domain

Challenges addressed by this project:

1. Inefficient data retrieval in REST APIs.
2. Scalability issues with growing user interactions.
3. The need for a flexible, query-optimized API.

Solution Domain

The API integrates GraphQL for efficient querying, PostgreSQL for relational data storage, Prisma ORM for database interaction, and Docker for deployment. Key features include:

- Thread creation and management
- User authentication and profile handling
- Efficient querying of related data

System Domain

The system follows a structured backend approach:

1. PostgreSQL for data persistence.
2. Prisma ORM for type-safe queries.
3. GraphQL API for efficient data fetching.
4. Dockerized setup for consistent deployment.

Application Domain

Potential use cases include:

- Online forums
- Q&A platforms
- Community discussion boards

Methodology

Development steps:

1. Data modeling using Prisma and PostgreSQL.
2. GraphQL schema design for queries and mutations.
3. API implementation with Docker for scalability.

Expected Outcome

The expected outcomes include:

- A scalable, efficient GraphQL API.
- Optimized data retrieval with flexible queries.
- Secure and structured database integration.

Conclusion

This project demonstrates the advantages of GraphQL in managing threads-based applications. With PostgreSQL, Prisma, and Docker, the API is designed for scalability, efficiency, and future extensibility.