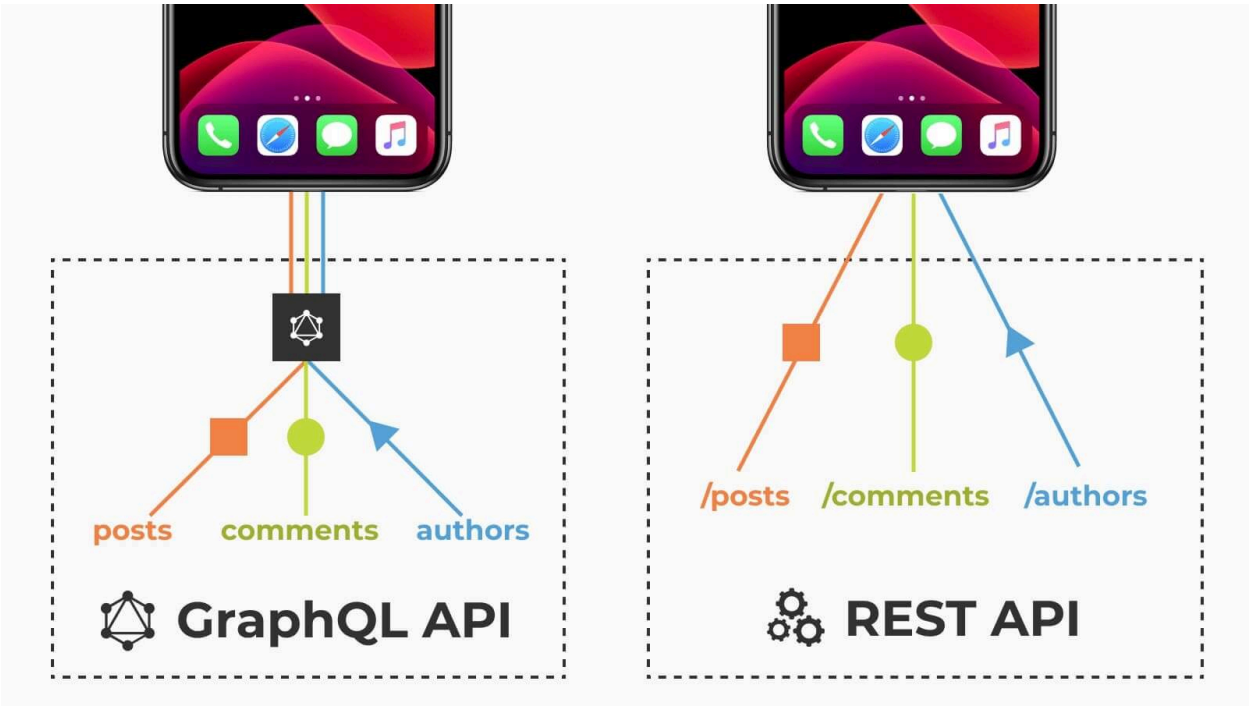
GraphQL https://graphql.org/

GraphQL is an open-source query language and runtime for APIs, designed to provide a more flexible, efficient, and developer-friendly alternative to traditional REST APIs. Developed by Facebook in 2012 and released publicly in 2015, GraphQL allows clients to request only the data they need, in the format they prefer, through a single endpoint.

Comparison: GraphQL vs REST API

Feature	GraphQL	REST API
Endpoint Structure	Single endpoint for all queries/mutations.	Multiple endpoints for different resources.
Data Fetching	Clients request only the data they need.	Fixed endpoints often return more data than required.
Over-fetching/Under-fetching	Avoided by querying specific fields.	Common due to rigid endpoint structures.
Schema	Strongly typed with schema introspection.	Implicit; depends on documentation.
Response Shape	Matches the query.	Predefined and static for each endpoint.
Real-Time Support	Subscriptions enable real-time updates.	Relies on WebSocket or other protocols; less standardized.
Batching Requests	Allows combining multiple queries or resource requests into a single call, reducing the number of network requests.	Multiple round-trips for related data.
Caching	Custom strategies are required.	Native HTTP caching support.



Repositories

GraphQL_Student_Chart_Client:
https://stash.texastech.edu/projects/SDP/repos/graphql_student_chart_client/browse

GraphQL_Student_Chart_Service:
https://stash.texastech.edu/projects/SDP/repos/graphql_student_chart_service/browse

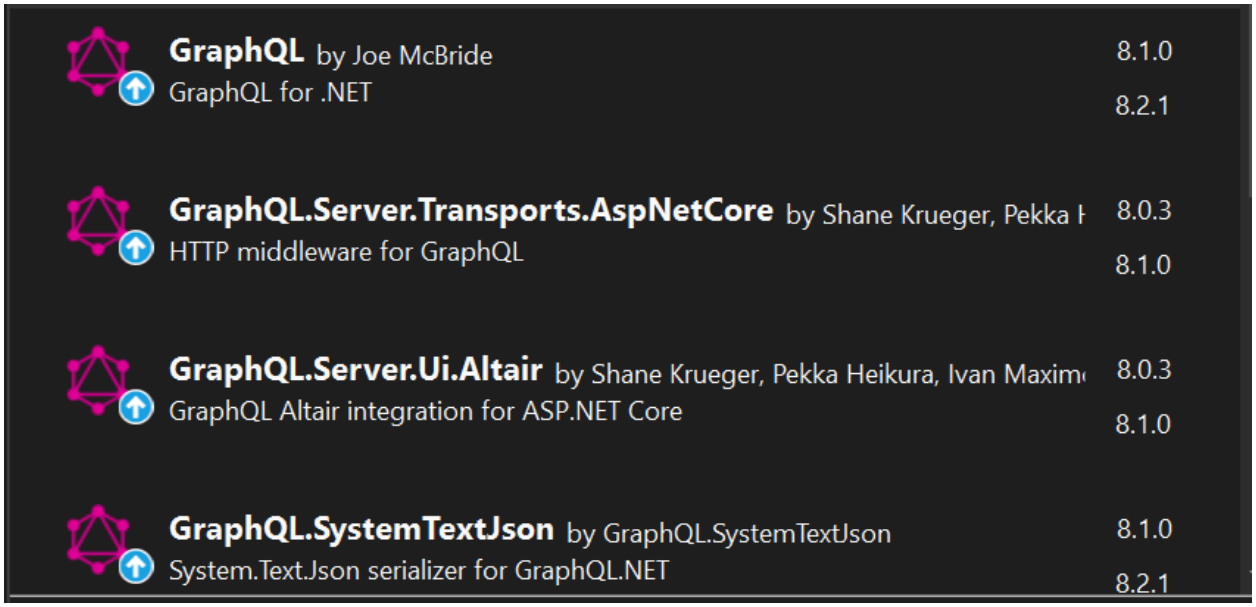
Packages

1. **GraphQL**
   Provides core GraphQL functionality for building a GraphQL API in .NET.

2. **GraphQL.Server.Transports.AspNetCore**
   Enables integration of GraphQL with ASP.NET Core, handling HTTP requests and routing.

3. **GraphQL.Server.Ui.Altair** https://localhost:7140/ui/altair
   Adds Altair GraphQL client integration for testing and debugging GraphQL APIs through a web UI.

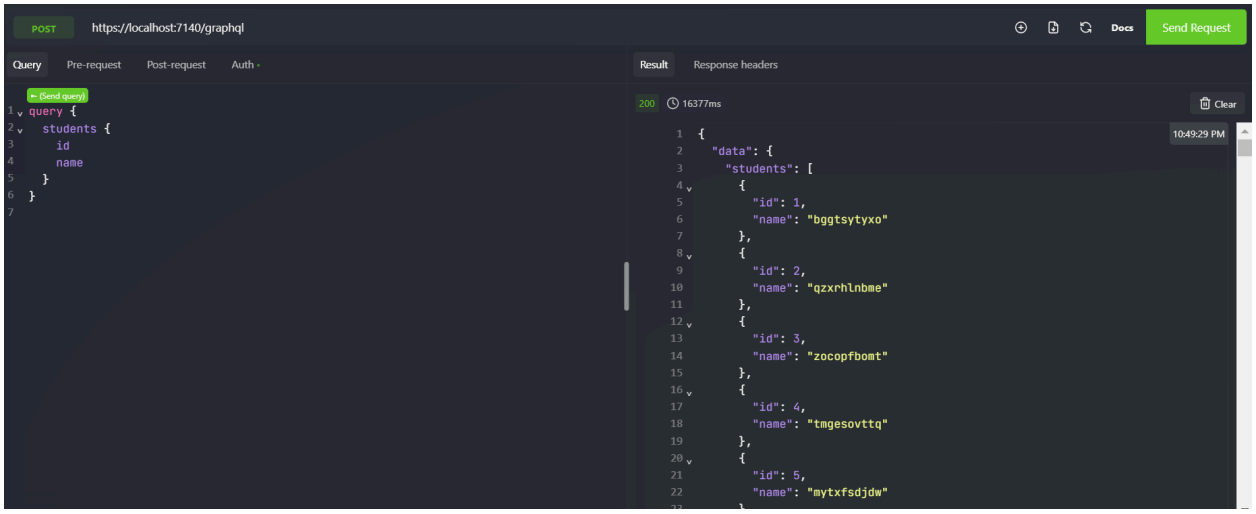
4. **GraphQL.SystemTextJson**
   Provides support for JSON serialization and deserialization using System.Text.Json in GraphQL applications.



GraphQL Queries

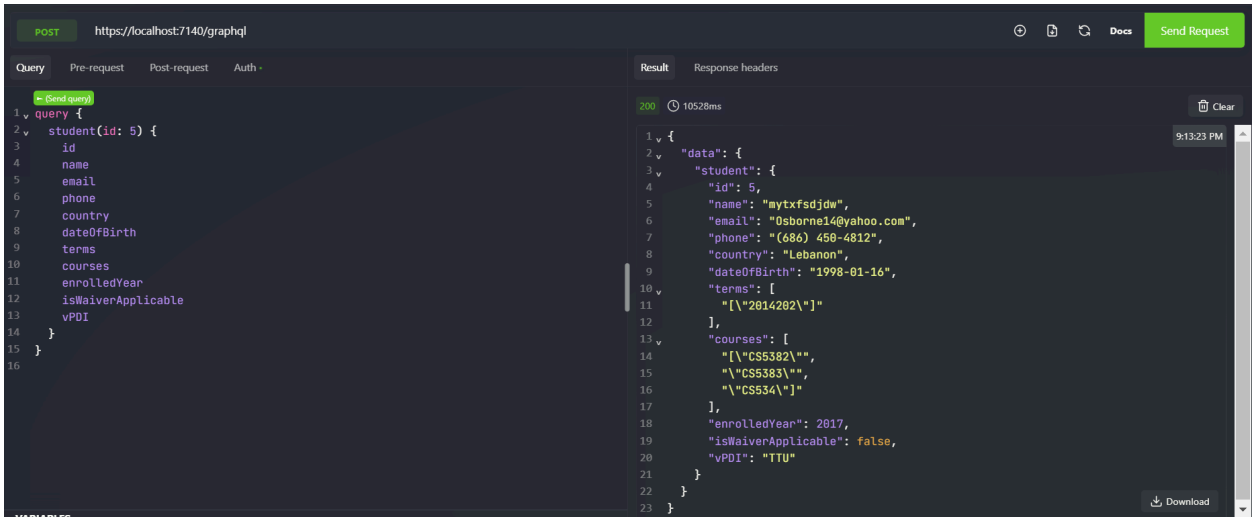
GraphQL queries retrieve data from the server. They allow clients to specify the exact fields and related data they need, reducing over- and under-fetching. Queries are **read-only** operations.

- **Retrieve All Students**



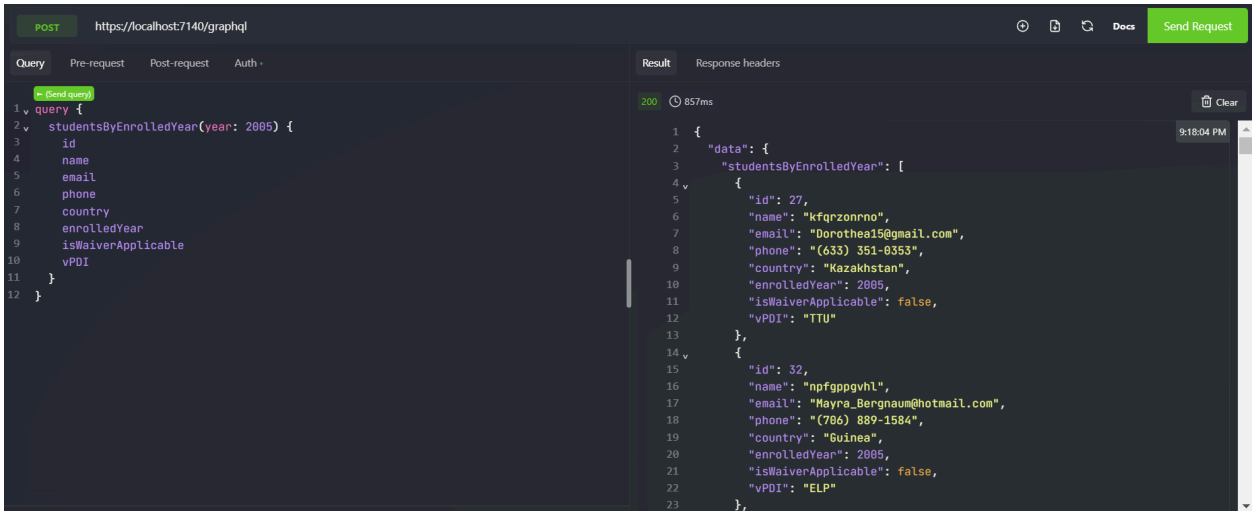
Description: Fetches all student records.

- **Retrieve a Student by ID**



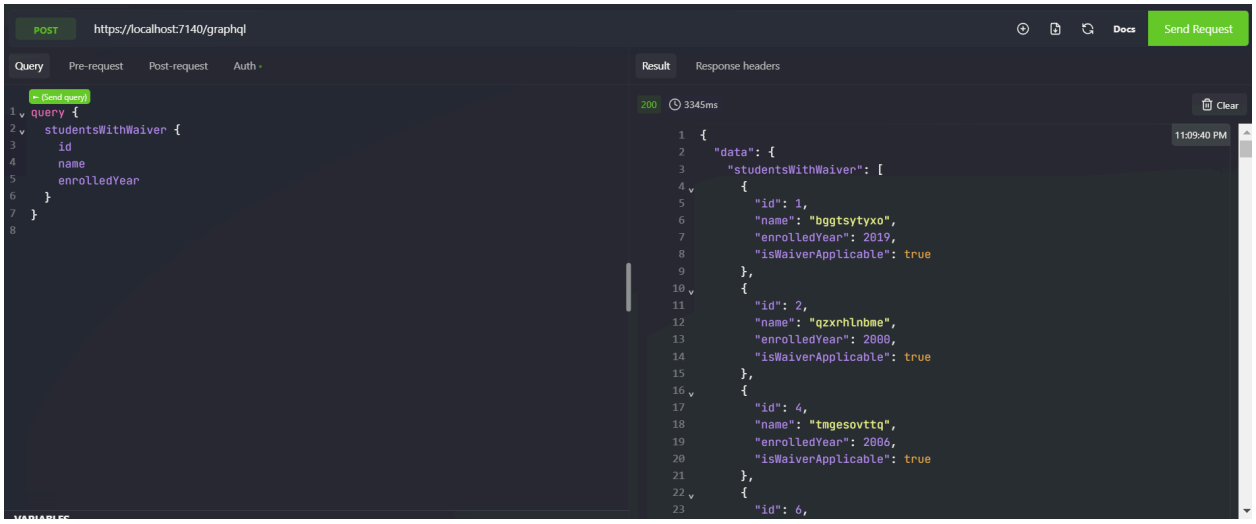
Description: Fetches a student by their unique ID.

- **Retrieve Students by Enrolled Year**



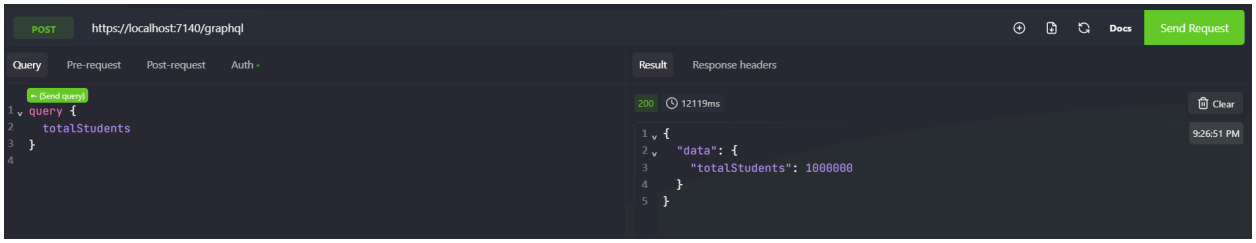
Description: Filters students based on their enrollment year.

- **Retrieve Students Eligible for Waivers**



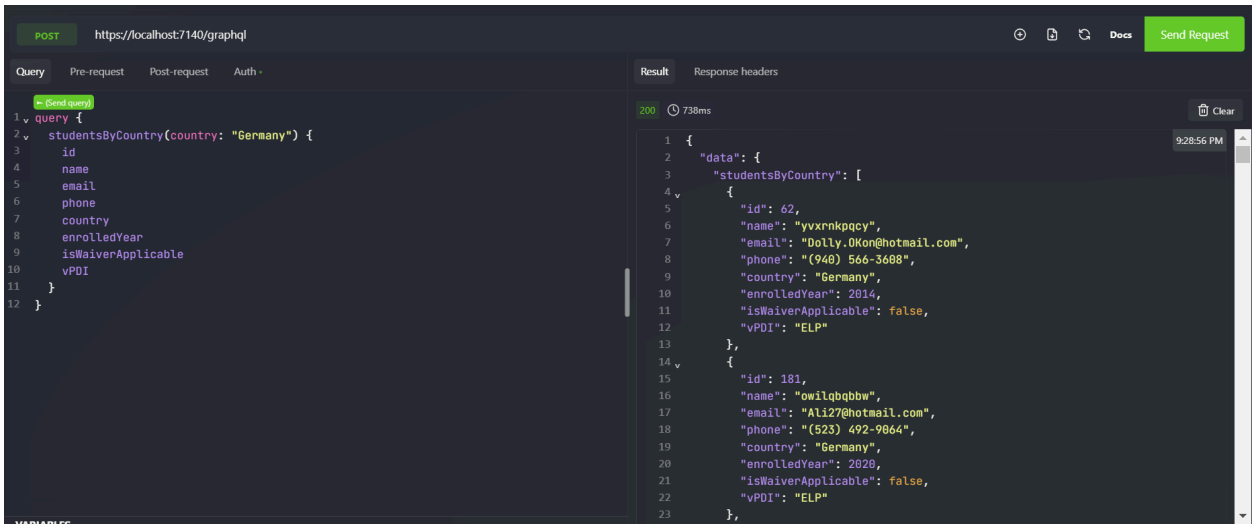
Description: Fetches students eligible for a waiver.

- **Get Total Student Count**



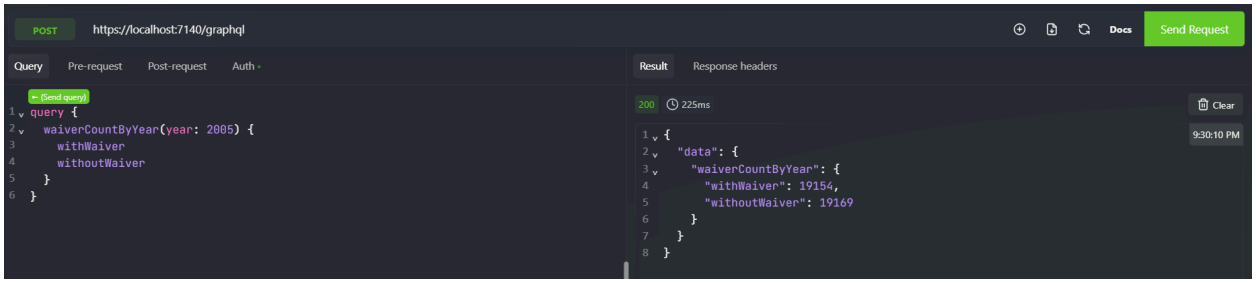
Description: Returns the total number of students.

- **Retrieve Students by Country**



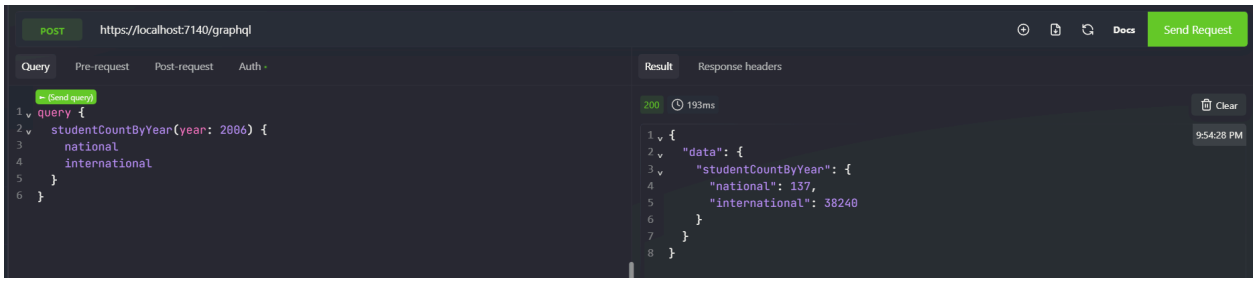
Description: Filters students based on their country.

- **Waiver Count by Year**



Description: Provides counts of students with and without waivers for a given year.

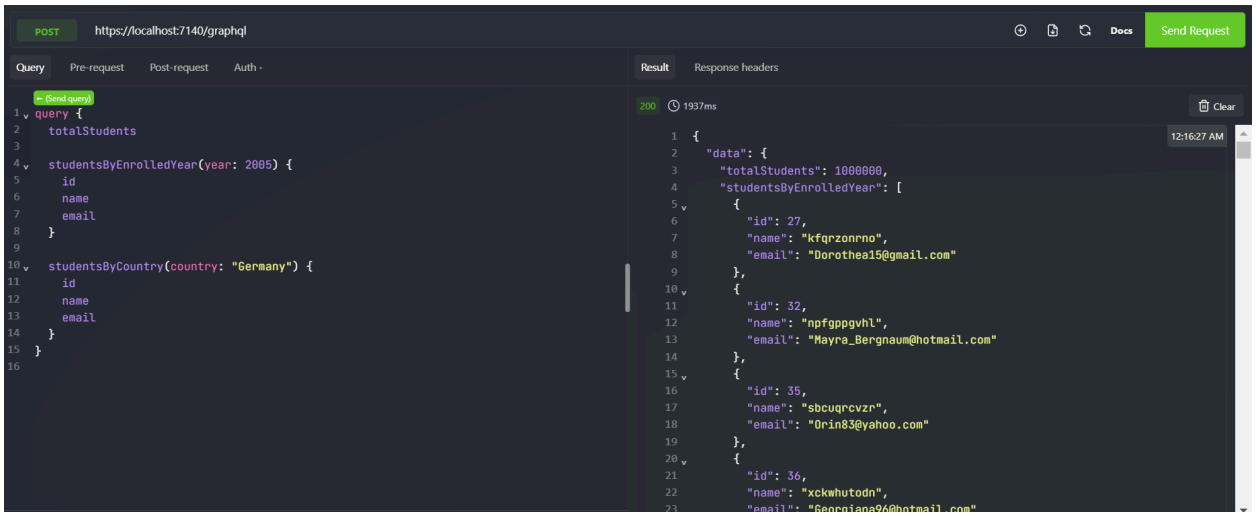
- **Student Count by Year**



Description: Returns the count of national and international students for a specific year.

Combining Multiple GraphQL Queries

In GraphQL, combining multiple queries into a single request allows the client to retrieve different pieces of data in one go. This approach reduces the number of requests and improves the efficiency of data fetching. The example below demonstrates how multiple queries can be combined to fetch various student-related data in one request:



Important

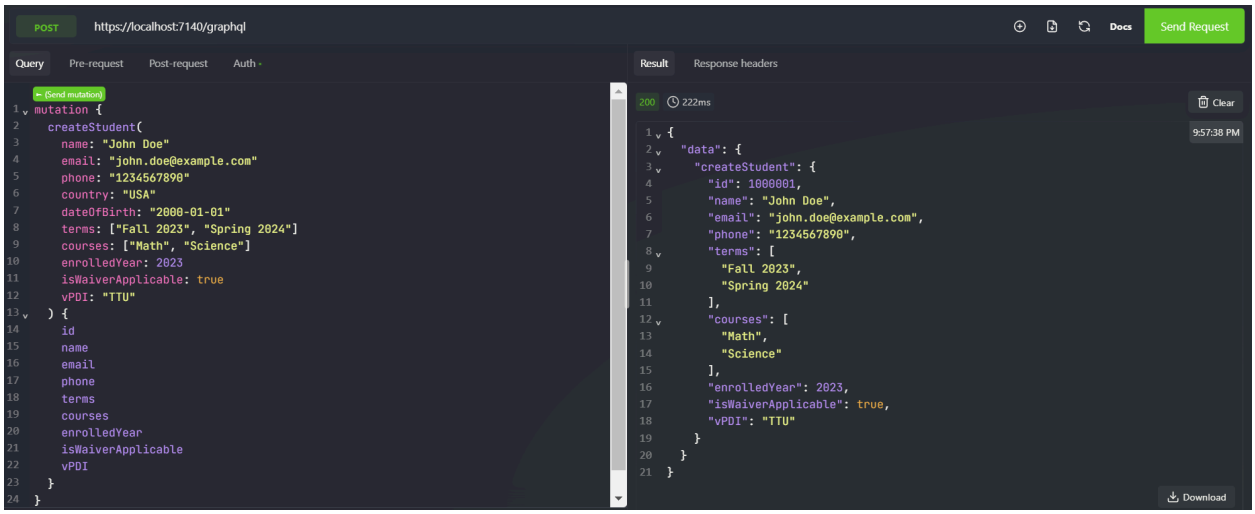
Handling Bulk Data and Pagination: When dealing with large datasets, such as 1 million records, attempting to fetch all data in a single query can result in errors like **"Maximum response size reached"**. This happens because the response payload exceeds the server or client's maximum allowed size.

Solution: To handle large datasets efficiently, use **pagination** for queries that fetch extensive data. Pagination divides the data into smaller chunks, minimizing the payload size and preventing server overload.

GraphQL Mutations

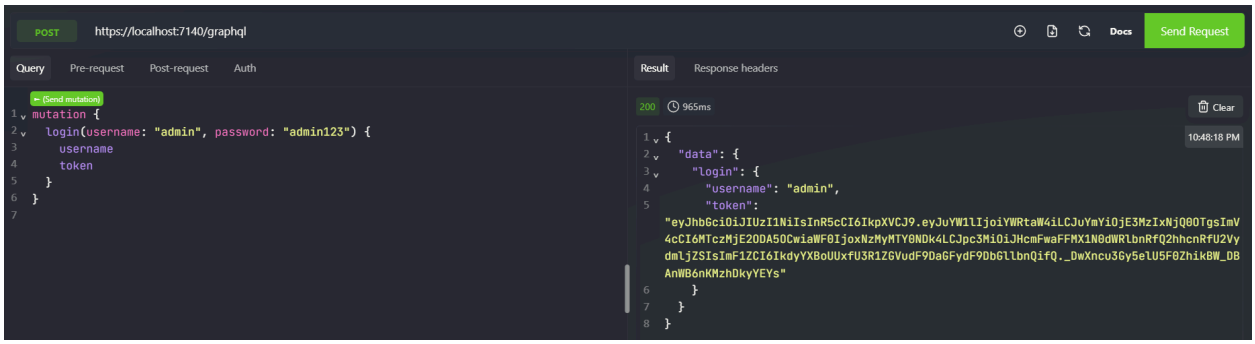
GraphQL mutations are used to modify server-side data, such as creating, updating, or deleting records. Mutations return the updated or affected data after the operation.

- **Create a New Student**



Description: Adds a new student record.

- **Login**



Description: Authenticates a user and generates a JWT token.

Authentication & Authorization Flow

- **Login Process (Authentication)**
  - A user sends their username and password via the login mutation.
  - The server validates credentials and issues a JWT if the credentials are valid.
  - The user then will include this JWT in the Authorization header for subsequent requests. **[Authorization: Bearer <jwt-token>]**

- **Resource Access (Authorization)**
  - The server validates the JWT (signature, issuer, audience, and expiration).
  - Authorized users are granted access to protected GraphQL queries/mutations.
  - Unauthorized requests are rejected with an error.

JSON Web Token https://jwt.io/

JWT (JSON Web Token), is a popular, secure, and stateless method to authenticate users and ensure only authorized users can access specific resources.

- **JWT Structure:** A JWT consists of three parts:
  - **Header:** Specifies the token type (JWT) and signing algorithm (e.g., HS256).
  - **Payload:** Contains user-specific claims such as the username.
  - **Signature:** Ensures the token is valid and not tampered with by signing the header and payload using a secret key.

- **Algorithm Used:** We used HMAC-SHA256 (HS256) to sign the token.

- **Expiration:** The token's lifespan is set to 60 minutes.

- **Issuer and Audience Validation:** Specifies the token's intended issuer and audience.