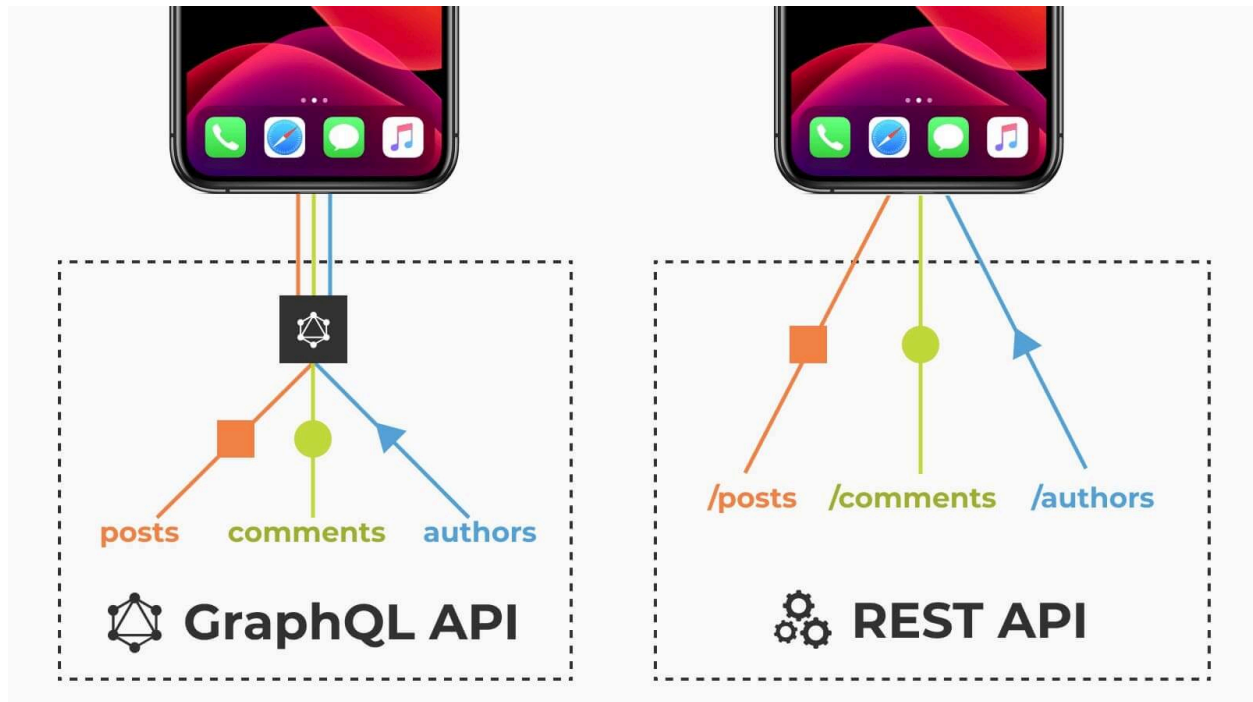


## GraphQL <https://graphql.org/>

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](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](https://stash.texastech.edu/projects/SDP/repos/graphql_student_chart_service/browse)

## Packages

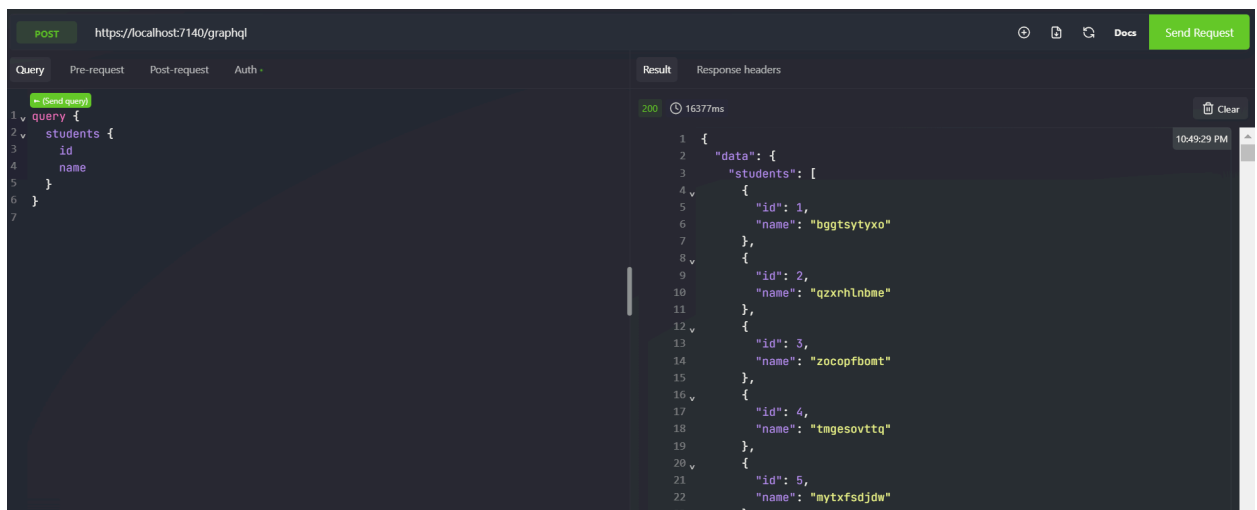
1. **GraphQL**  
Provides core GraphQL functionality for building a GraphQL API in .NET.
2. **GraphQL.Server.Transports.AspNetCore**  
Enables integration of GraphQL with ASP.NET Core, handling HTTP requests and routing.
3. **GraphQL.Server.Ui.Altair** <https://localhost:7140/ui/altair>  
Adds Altair 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.

	<b>GraphQL</b> by Joe McBride	8.1.0
	GraphQL for .NET	8.2.1
	<b>GraphQL.Server.Transports.AspNetCore</b> by Shane Krueger, Pekka Heikura	8.0.3
	HTTP middleware for GraphQL	8.1.0
	<b>GraphQL.Server.Ui.Altair</b> by Shane Krueger, Pekka Heikura, Ivan Maximov	8.0.3
	GraphQL Altair integration for ASP.NET Core	8.1.0
	<b>GraphQL.SystemTextJson</b> by GraphQL.SystemTextJson	8.1.0
	System.Text.Json serializer for GraphQL.NET	8.2.1

## 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**



The screenshot shows the GraphQL Playground interface. The URL is `https://localhost:7140/graphql`. The query is:

```

1 query {
2   students {
3     id
4     name
5   }
6 }

```

The response is a JSON object:

```

1 {
2   "data": {
3     "students": [
4       {
5         "id": 1,
6         "name": "bggtsytxo"
7       },
8       {
9         "id": 2,
10        "name": "qzxrhlbnme"
11      },
12      {
13        "id": 3,
14        "name": "zocopfboat"
15      },
16      {
17        "id": 4,
18        "name": "tagesovttq"
19      },
20      {
21        "id": 5,
22        "name": "mytxfsdjdjw"
23      }
24    ]
25  }
26 }

```

**Description:** Fetches all student records.

- **Retrieve a Student by ID**

The screenshot shows the GraphQL Playground interface. The URL is `https://localhost:7140/graphql`. The query is a POST request. The query text is:

```
1 query {  
2   student(id: 5) {  
3     id  
4     name  
5     email  
6     phone  
7     country  
8     dateOfBirth  
9     terms  
10    courses  
11    enrolledYear  
12    isWaiverApplicable  
13    vPDI  
14  }  
15 }  
16
```

The result is a 200 status with a response time of 10528ms. The JSON response is:

```
1 {  
2   "data": {  
3     "student": {  
4       "id": 5,  
5       "name": "mytxfsdjd",  
6       "email": "Osborne14@yahoo.com",  
7       "phone": "(686) 458-4812",  
8       "country": "Lebanon",  
9       "dateOfBirth": "1998-01-16",  
10      "terms": [  
11        "\2814282\"  
12      ],  
13      "courses": [  
14        "\CS5382\"  
15        "\CS5383\"  
16        "\CS534\"  
17      ],  
18      "enrolledYear": 2017,  
19      "isWaiverApplicable": false,  
20      "vPDI": "TTU"  
21    }  
22  }  
23 }
```

**Description:** Fetches a student by their unique ID.

- **Retrieve Students by Enrolled Year**

The screenshot shows the GraphQL Playground interface. The URL is `https://localhost:7140/graphql`. The query is a POST request. The query text is:

```
1 query {  
2   studentsByEnrolledYear(year: 2005) {  
3     id  
4     name  
5     email  
6     phone  
7     country  
8     enrolledYear  
9     isWaiverApplicable  
10    vPDI  
11  }  
12 }
```

The result is a 200 status with a response time of 857ms. The JSON response is:

```
1 {  
2   "data": {  
3     "studentsByEnrolledYear": [  
4       {  
5         "id": 27,  
6         "name": "kfqrzonrno",  
7         "email": "Dorothea15@gmail.com",  
8         "phone": "(633) 351-0353",  
9         "country": "Kazakhstan",  
10        "enrolledYear": 2005,  
11        "isWaiverApplicable": false,  
12        "vPDI": "TTU"  
13      },  
14      {  
15        "id": 32,  
16        "name": "npfgppgvhl",  
17        "email": "Mayra_Bergnaum@hotmail.com",  
18        "phone": "(786) 889-1584",  
19        "country": "Guinea",  
20        "enrolledYear": 2005,  
21        "isWaiverApplicable": false,  
22        "vPDI": "ELP"  
23      }  
24    ]  
25  }  
26 }
```

**Description:** Filters students based on their enrollment year.

- **Retrieve Students Eligible for Waivers**

The screenshot shows a GraphQL query in the Playground interface. The query is: `query { studentsWithWaiver { id name enrolledYear } }`. The response is a JSON object: `{ "data": { "studentsWithWaiver": [ { "id": 1, "name": "bggtsytyxo", "enrolledYear": 2019, "isWaiverApplicable": true }, { "id": 2, "name": "qzxrhlnbme", "enrolledYear": 2000, "isWaiverApplicable": true }, { "id": 4, "name": "tagesovtta", "enrolledYear": 2006, "isWaiverApplicable": true }, { "id": 6, "name": "vpxrnkpqcy", "enrolledYear": 2014, "isWaiverApplicable": false }, { "id": 181, "name": "owilqbqbbw", "enrolledYear": 2020, "isWaiverApplicable": false }, { "id": 523, "name": "Ali27@hotmail.com", "enrolledYear": 1984, "isWaiverApplicable": false } ] } }`. The status is 200 and the response time is 3345ms.

**Description:** Fetches students eligible for a waiver.

- **Get Total Student Count**

The screenshot shows a GraphQL query in the Playground interface. The query is: `query { totalStudents }`. The response is a JSON object: `{ "data": { "totalStudents": 1000000 } }`. The status is 200 and the response time is 12119ms.

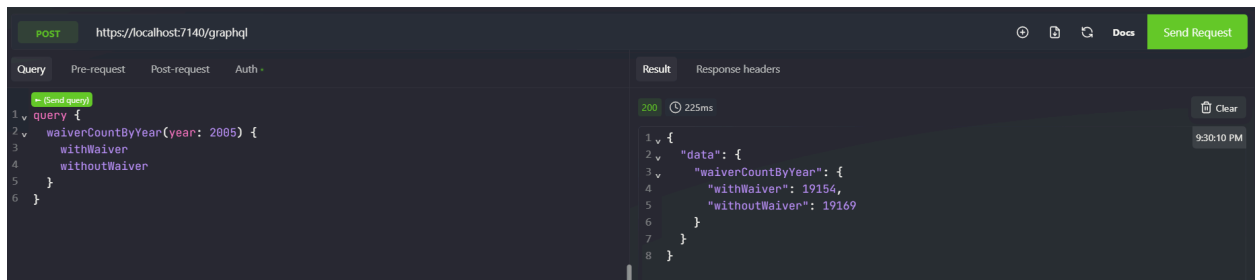
**Description:** Returns the total number of students.

- **Retrieve Students by Country**

The screenshot shows a GraphQL query in the Playground interface. The query is: `query { studentsByCountry(country: "Germany") { id name email phone country enrolledYear isWaiverApplicable vPDI } }`. The response is a JSON object: `{ "data": { "studentsByCountry": [ { "id": 62, "name": "yvvrnkpgcy", "email": "Dolly.0Kon@hotmail.com", "phone": "(940) 566-3688", "country": "Germany", "enrolledYear": 2014, "isWaiverApplicable": false, "vPDI": "ELP" }, { "id": 181, "name": "owilqbqbbw", "email": "Ali27@hotmail.com", "phone": "(523) 492-9864", "country": "Germany", "enrolledYear": 2020, "isWaiverApplicable": false, "vPDI": "ELP" } ] } }`. The status is 200 and the response time is 738ms.

**Description:** Filters students based on their country.

- **Waiver Count by Year**



The screenshot shows a GraphQL query in the Playground interface. The query is a POST request to `https://localhost:7140/graphql`. The query text is:

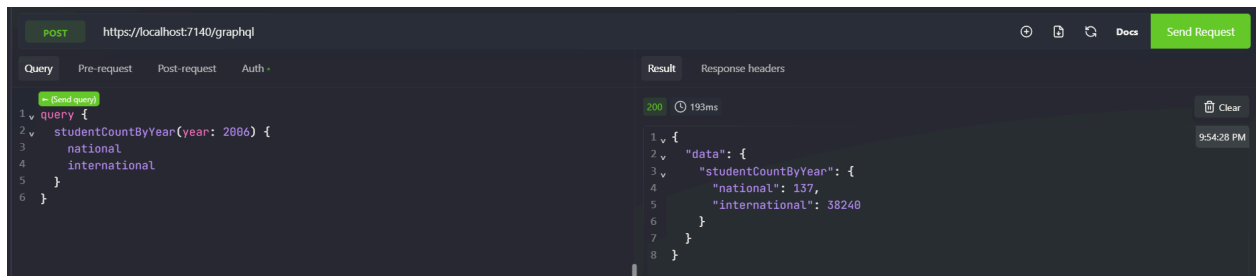
```
1 query {  
2   waiverCountByYear(year: 2005) {  
3     withWaiver  
4     withoutWaiver  
5   }  
6 }
```

The response is a 200 status code with a 225ms response time. The JSON response is:

```
1 {  
2   "data": {  
3     "waiverCountByYear": {  
4       "withWaiver": 19154,  
5       "withoutWaiver": 19169  
6     }  
7   }  
8 }
```

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

- **Student Count by Year**



The screenshot shows a GraphQL query in the Playground interface. The query is a POST request to `https://localhost:7140/graphql`. The query text is:

```
1 query {  
2   studentCountByYear(year: 2006) {  
3     national  
4     international  
5   }  
6 }
```

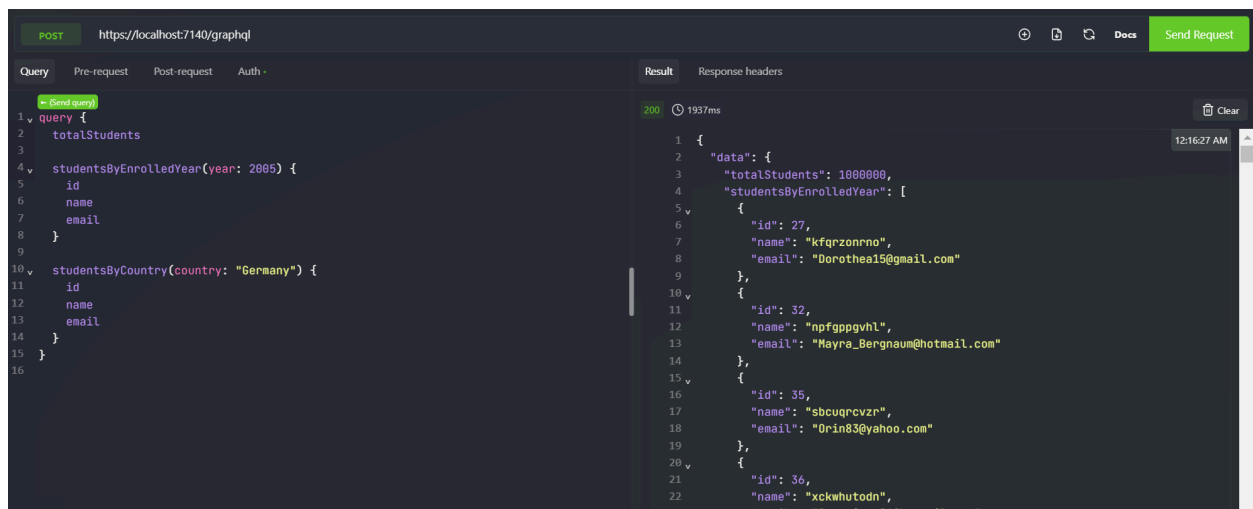
The response is a 200 status code with a 193ms response time. The JSON response is:

```
1 {  
2   "data": {  
3     "studentCountByYear": {  
4       "national": 137,  
5       "international": 38248  
6     }  
7   }  
8 }
```

**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:



The screenshot shows a GraphQL query in the Playground interface. The query is a POST request to `https://localhost:7140/graphql`. The query text is:

```
1 query {  
2   totalStudents  
3  
4   studentsByEnrolledYear(year: 2005) {  
5     id  
6     name  
7     email  
8   }  
9  
10  studentsByCountry(country: "Germany") {  
11    id  
12    name  
13    email  
14  }  
15 }  
16
```

The response is a 200 status code with a 1937ms response time. The JSON response is:

```
1 {  
2   "data": {  
3     "totalStudents": 1000000,  
4     "studentsByEnrolledYear": [  
5       {  
6         "id": 27,  
7         "name": "kfqrzonnno",  
8         "email": "Dorothea15@gmail.com"  
9       },  
10      {  
11        "id": 32,  
12        "name": "npfgppgvhl",  
13        "email": "Mayra_Bergnaum@hotmail.com"  
14      },  
15      {  
16        "id": 35,  
17        "name": "sbcugprcvzn",  
18        "email": "Orin83@yahoo.com"  
19      },  
20      {  
21        "id": 36,  
22        "name": "xckwhutodn",  
23        "email": "Gertriana94@hotmail.com"  
24      }  
25    ]  
26   }  
27 }
```

## 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**

The screenshot shows a GraphQL Playground interface with a POST request to `https://localhost:7140/graphql`. The query is a mutation to create a student with the following fields: `name`, `email`, `phone`, `country`, `dateOfBirth`, `terms`, `courses`, `enrolledYear`, `isWaiverApplicable`, and `vPDI`. The response is a 200 status with a 222ms response time. The JSON result shows the created student's data, including a unique ID and the same input fields.

```
POST https://localhost:7140/graphql

mutation {
  createStudent(
    name: "John Doe"
    email: "john.doe@example.com"
    phone: "1234567890"
    country: "USA"
    dateOfBirth: "2000-01-01"
    terms: ["Fall 2023", "Spring 2024"]
    courses: ["Math", "Science"]
    enrolledYear: 2023
    isWaiverApplicable: true
    vPDI: "TTU"
  ) {
    id
    name
    email
    phone
    terms
    courses
    enrolledYear
    isWaiverApplicable
    vPDI
  }
}
```

```
{
  "data": {
    "createStudent": {
      "id": "1000001",
      "name": "John Doe",
      "email": "john.doe@example.com",
      "phone": "1234567890",
      "terms": [
        "Fall 2023",
        "Spring 2024"
      ],
      "courses": [
        "Math",
        "Science"
      ],
      "enrolledYear": 2023,
      "isWaiverApplicable": true,
      "vPDI": "TTU"
    }
  }
}
```

**Description:** Adds a new student record.

- **Login**

The screenshot shows a GraphQL Playground interface with a POST request to `https://localhost:7140/graphql`. The query is a mutation to login a user with `username` and `password`. The response is a 200 status with a 965ms response time. The JSON result shows the login success, including the username and a long JWT token.

```
POST https://localhost:7140/graphql

mutation {
  login(username: "admin", password: "admin123") {
    username
    token
  }
}
```

```
{
  "data": {
    "login": {
      "username": "admin",
      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1IjoieWVWRTA4IiwiaWF0IjE5MzI1NjQ0TgsImV4cCI6MTczMjE2ODAwIiwiaWF0IjoxNjY0NDk4LCJpc3MiOiJhcFwFMX1N0dWRlbnRfQ2hcnRfU2VydmljZSIsImF1dGUiOiJkdyYXBoUWxvU3R1ZGVudF90aGFydF90bGllbnQ1fQ._DwXncu36y5eLU5F02hik8W_DBAnWB6nKMzhDkyYEs"
    }
  }
}
```

**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.