GraphQL

Mimanshu Maheshwari

December 4, 2024

Contents

List of Figures

Listings

Chapter 1

Why and What is GraphQL?

1.1 Why GraphQL

We all know that REST contributes more towards API development. In recent years, REST has become the dominant style for creating web services that are reusable and manageable. Everything in REST is represented as a resource which in turn has a unique URI to get identified. Also, REST works with a set of HTTP methods/verbs to define the operations being performed on the REST resources. Each one of us, the developers, will accept that REST is a great approach to proceed with API development as

- It leverages the usage of the standard HTTP protocol

- It is stateless

- It supports caching

- It is light-weight

- It supports multiple representations for resources and so on

But, we should agree that REST has its own limitations as well. And, the limitations are

- **Over-fetching**: The client might be interested to get the id and name of a resource, but the REST call will fetch the whole set of fields corresponding to that specific resource.

- **Multiple requests**: in case the client needs multiple data - REST API routes get longer when the data goes increasingly complex.

Because of these complexities, the client devices will slow down when there is limited network bandwidth. One more limitation of REST that needs to be put forth here is, it's important for the client to know the exact location of every single resource - If the API server holds multiple resources named customer, plan and so on, the client is expected to know the absolute URI of all the resources that are available with the server. Do we have anything in hand to overcome these issues?

Yes, we have something called GraphQL to help us with these issues.

1.2 What is GraphQL?

GraphQL is a server-side runtime and a query language for the APIs that we write It was developed by Facebook to optimize the REST calls to meet the growing requirements Later, Facebook open-sourced GraphQL as a specification for building APIs

GraphQL makes it possible to organize our data in the form of a graph and uses a powerful query syntax to traverse over the graph to retrieve and modify the data being stored in the graph.

From the definition above, it is evident that GraphQL is not only a specification for how to build APIs but also a specification for how to access the APIs from the client end. GraphQL queries are declarative by nature.

Now let's take a look at the advantages being offered by GraphQL that this course covers with detailed dem

- No over and under fetching – Ask for what is required and get that alone.

- A single request can help to retrieve multiple resources

- Describes well what's possible with a well-structured type system to help the clients understand the API better

- Move faster - in the sense of rapid API development with the help of powerful developer tools.

Resource utilization is much optimized because of the power of asking for only the things that are required and the ability to get multiple resources in a single hit/request. Hence, the client devices can perform better even on very less network bandwidth.

1.3   Ask for what do you want

We can send GraphQL queries to the APIs and get exactly what we need - nothing more, nothing less. A GraphQL query will always return a predictable result. Applications that use GraphQL are both fast and stable. Unlike RESTful calls, a GraphQL client call can actually restrict data that needs to be retrieved from the server. The following example can help us understand better. Let us consider an object Employee with the attributes - id, firstName, lastName, and technologyName. And, assume that we have a mobile application that needs to fetch the employees' firstName and id alone. If we design a RESTful endpoint for this requirement- `/api/v1/employees`, it just will end up fetching/retrieving data for every single field of the employee objects. This literally means, data is being over-fetched by the RESTful service. This problem can efficiently be addressed with the power of GraphQL. Using GraphQL, we can create a query similar to the one being shown below.

```
1  {
2    employees {
3      id
4      firstName
5    }
6  }
```
Listing 1.1: Employee fetch id and name only using GraphQL query

Now, this query shall return the values of the `id` and `firstName` fields alone and will ignore other attributes like `lastName` and `technologyName` of the employee object. Observe the response for the above query:

```
1    {
2     "data": {
3       "employees": [
4         {
5           "id": "E1001",
6           "firstName": "John"
7         },
8         {
9           "id": "E1002",
10          "firstName": "Jane"
11        }
12      ]
13    }
```

```
14  }
```

Listing 1.2: Employee fetch id and name only response

1.4   Get many resources in a single request

GraphQL queries can help us smoothly retrieve multiple resources in a single request. In contrast to this, a typical REST API might require loading data from multiple URLs. Since the API calls are optimized to a great extent, applications that use GraphQL can perform faster even on a slow mobile network connection! The following example can make us understand this concept even better. Let us consider an object employee that has fields like id, firstName, lastName, jobLevel and companyId. Let us consider another object, Company which has attributes – name, and description. The employee resource can be provided by the endpoint, `/api/v1/employees` and the company resource can be provided by the endpoint, `/api/v1/companies` ↪ . If the client wants both employee and company details, the client should two calls, on to fetch the employee details (`/api/v1/employees`) and the other to fetch the company details (`/api/v1/companies`). This is nothing but under-fetching of data where multiple calls are required to fetch the needed information. Hence, applications are forced to give multiple calls to the server to get the data that is required. However, GraphQL can help an application to fetch details for both Employee and Company in a single request. Observe the below GraphQL query to fetch the employee and the company together.

```
1  query
2  {
3      employees{firstName,lastName,jobLevel},
4
5      companies{id,name,description}
6  }
```

Listing 1.3: fetch multiple resource in single query

```
1   {
2     "data": {
3       "employees": [
4       {
5         "firstName": "Steffy",
6         "lastName": "Mayor",
7         "jobLevel": 5
8       }
9       {
10        "firstName": "Steffy2",
11        "lastName": "Mayor2",
12        "jobLevel": 6
13      }
14      ],
15      "companies": [
16      {
17        "id": "C232",
18        "name": "Product Company",
19        "description": "Tech Ex"
20      }
21      {
22        "id": "C233",
23        "name": "Tech Company",
24        "description": "Infy"
25      }
```

```
26          ]
27      }
28    }
```
Listing 1.4: fetch multiple resource in single query response

1.5 Describe what's possible with a type system

GraphQL is strongly typed. GraphQL queries are based on the fields and their associated data types. If there is a type mismatch in a GraphQL query, the server applications will return error messages that are very clear and much helpful to the clients to introspect further in way smoother. Along with this, GraphQL provides various client-side libraries that actually can help to reduce explicit way of data conversion and parsing.

```
1    type Query {
2      employees: [Employee]
3    }
4    type Employee {
5
6      id: ID!
7      firstName: String
8      lastName: String
9      fullName: String
10     company: Company
11   }
12   type Company {
13     id: ID!
14     name: String
15     description: String
16     rating: Float
17     employees: [Employee]
18   }
```
Listing 1.5: Example of GraphQL typing

1.6 Hit the ground running with powerful developer tools

GraphQL offers a decent number of developer tools that can help to document the APIs and testing the queries.

- **GraphiQL** is a wonderful tool that can help us generate documentation for the query and the schema to which the query belongs. It also provides a query editor to test our GraphQL APIs. Apart from these, GraphiQL has an intelligent code completion capability that helps us quickly build our queries.

- **GraphQL Voyager** is yet another tool that can help us represent/depict a GraphQL API as an interactive graph! GraphQL Voyager's motto is – "It's time to finally see the graph behind GraphQL"! It also lets us explore several public GraphQL APIs from its list. It also includes a wide range of examples of GraphQL schemas and allows us to connect them to our own GraphQL endpoints.

- **Apollo SandBox**: A unique Apollo Studio mode which is a web IDE for creating, running and managing graphQL operations called Apollo Sandbox aids with local development.

GraphQL is a server-side runtime and a query language for the APIs.

- No over and under fetching – Ask for what is required and get that alone.

7

- A single request can help to retrieve multiple resources.

- Describes well what's possible with a well-structured type system to help the clients understand the API better.

- Move faster - in the sense of rapid API development with the help of powerful developer tools.

Chapter 2

GraphQL Application Architecture: An Overview

2.1 Various GraphQL architecture approaches

2.1.1 GraphQL application architecture

GraphQL is just a specification that set out how a GraphQL server should behave. This essentially means, a set of guidelines will be defined by any GraphQL based application - on how requests and responses to be handled by the server.

The guidelines generally include:

- What are the supported protocols?

- Which format of data can be acceptable for the server?

- Which format of response is possible for the server to return? etc.

The GraphQL mindset

The request from a GraphQL client to the GraphQL server is known as Query. GraphQL is transport-layer agnostic. That is, GraphQL actually is agnostic with respect to how data flows over the network. GraphQL server can work with protocols other than HTTP as well. To quote a few, WebSockets and TCP. But, the most widely used one is HTTP indeed. GraphQL is neutral to databases. That is, we are free to use GraphQL with SQL or NoSQL databases or even the combination of both!

Various GraphQL app architecture possibilities

A GraphQL server can be created using either of the methods being listed below –

1. GraphQL server with a connected database

2. GraphQL server that integrates existing (possibly legacy) systems

3. Hybrid approach – which is a mix of the above two approaches

2.1.2 GraphQL Server with Connected Database

In this architecture, we can find GraphQL Servers with a tightly coupled/integrated database. Upon receiving a query from the client, the GraphQL server will first make sure reading the request payload and then fetching the required data from the database. This is known as resolving the query which in turn results in a response from the GraphQL server. The response is returned to the client and it adheres to the format being specified
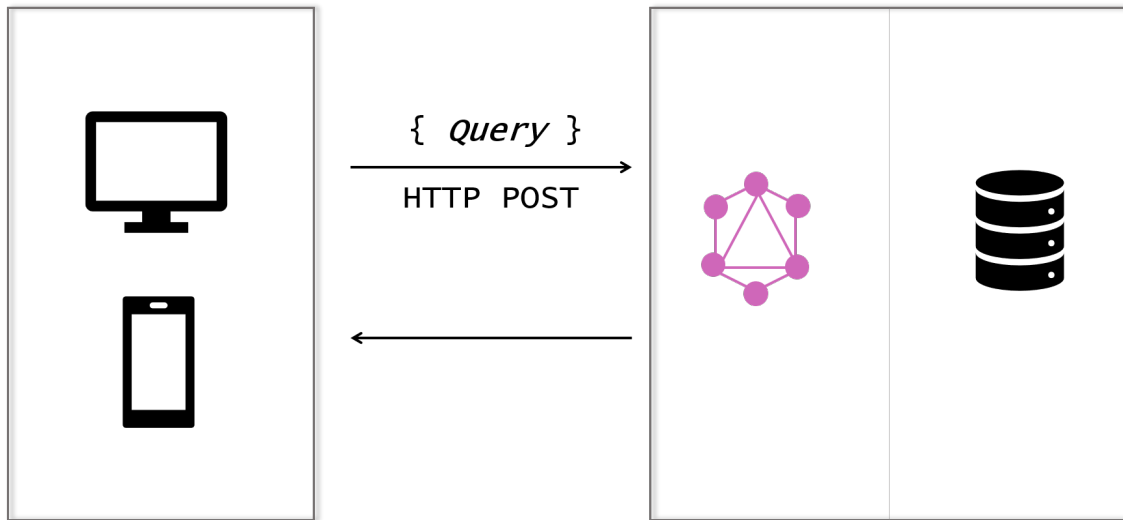
9

Figure 2.1: GraphQL Server with a Connected DataBase

in the GraphQL specification that is provided by the Schema and the Type system. This kind of architecture can often be used in projects that start from scratch.

In the diagram being shown above, the GraphQL server and the database (SQL or NoSQL) are made to be available on a single node. The Client (desktop or mobile) communicates with the GraphQL server over HTTP. The server processes the request fetches/retrieve data from the DB and, returns it finally to the client.

2.1.3 GraphQL server integrating the existing Systems

This approach would be much helpful for the companies that have legacy infrastructure and possess a collection of different APIs. GraphQL can be employed in this kind of architecture to unify microservices, legacy infrastructure, and even third-party APIs. Observe the below block diagram:
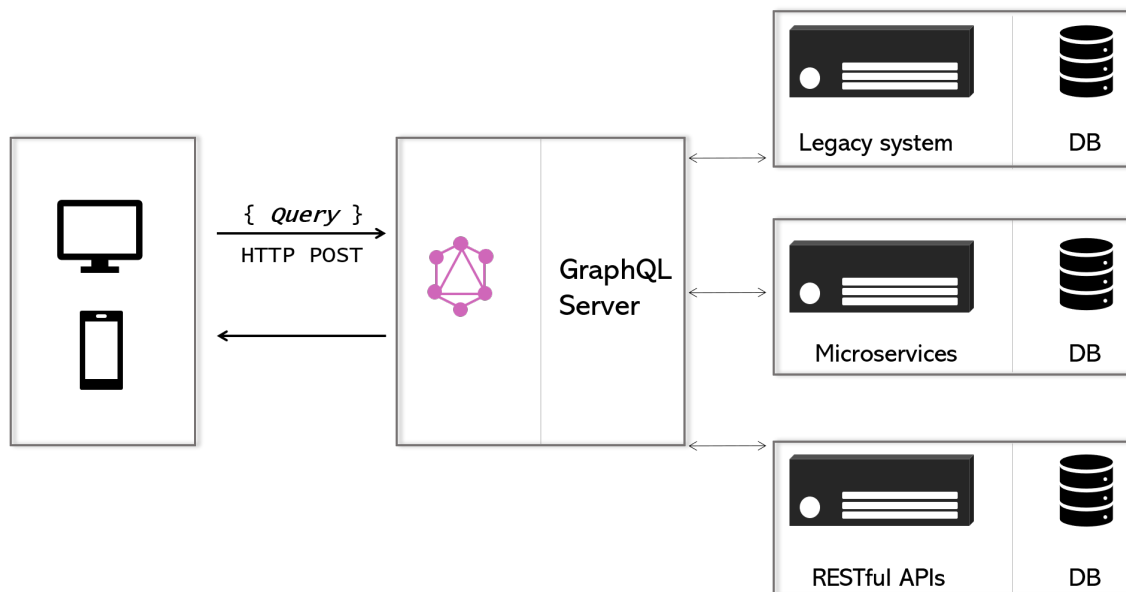

Figure 2.2: GraphQL server integrating the existing Systems

10

In the given diagram, GraphQL Server acts as an interface between the client and the existing systems. Client applications reach out to the GraphQL server which in turn, resolves the query and gets data from the existing APIs, and renders the response back to the client applications.