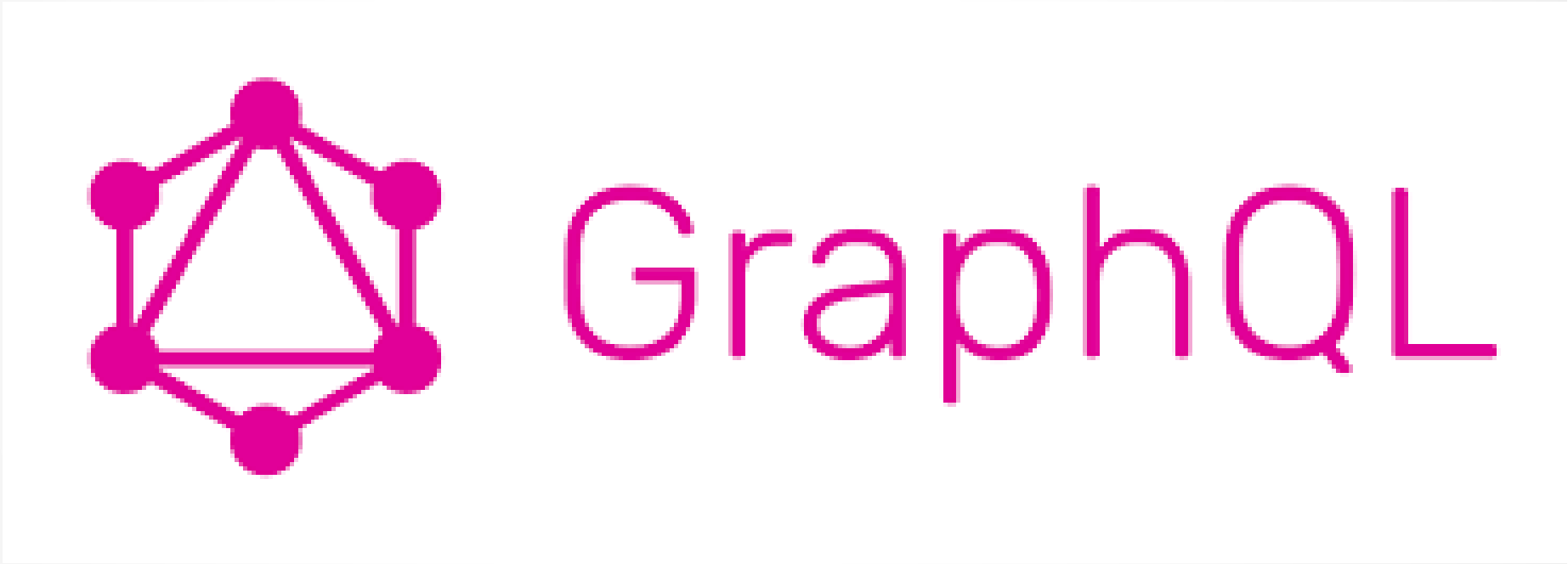
GRAPHQL



What is GraphQL ?

GraphQL is a query language

GraphQL is an open source server-side technology

which was developed by Facebook to optimize RESTful API calls

It is an execution engine and a data query language.

Concepts of GraphQL –

1. Implement GraphQL API using GraphQL Server
2. Test GraphQL API using GraphiQL
3. Client applications to consume the API

Advantages of GraphQL

1. Ask for what you want – and get it

- Send a GraphQL query to your API and get exactly what you need.
- GraphQL queries always return predictable results.
- Applications using GraphQL are fast and stable.

Example :

1. Consider a business object ViewProfile with the attributes MatriId, firstName, lastName and LastCommunication.
2. Suppose a mobile application needs to fetch only the firstName and MatriId.
3. If we design a REST endpoint like /api/v1/viewprofile, it will end up fetching data for all the fields for a ViewProfile object.

This problem can be solved by using GraphQL.

Consider the GraphQL query given below −

```
{
  ViewProfile {
      MatriId
      firstName
  }
}
```

This will return values only for the MatriId and firstname fields.

```
{
  "data": {
      "viewprofile": [
          {
            "MatriId": "M1001",
            "firstName": "durai"
          },
          {
            "MatriId": "M1002",
            "firstName": "Sathrak"
          }
      ]
  }
}
```

2. Get many resources in a single request

- **GraphQL APIs fetch all the data your application need in a single request.**
- **Applications using GraphQL can be quick even on slow mobile network connections.**

Example :

1. Consider one more business object, Communication which has the attributes: name and location.

2. The ViewProfile business object has an association relationship with the Communication object.

3. If we were to use a REST API in order to fetch the details of ViewProfile and their Communication, we will end up making two requests to the server like /api/v1/ViewProfile and /api/v1/Communication.

4. So mobile applications are forced to make multiple calls to the server to get the desired data.

Mobile application can fetch details for both Student and College objects in a single request by using GraphQL.

The following is a GraphQL query to fetch data −

```
{
   viewprofile{
         MatriId
         firstName
         lastName
         communication{
            name
            location
         }
   }
}
```

The output of the above query contains exactly those fields we have requested for as shown below −


```
{
   "data": {
         "viewprofile": [
            {
               "MatriId": "M1001",
               "firstName": "Mohtashim",
               "lastName": "Mohammad",
               "communication": {
                  "name": "CUSAT",
                  "location": "Kerala"
               }
            },
            {
               "MatriId": "M1003",
               "firstName": "Kiran",
               "lastName": "Panigrahi",
               "communication": {
                  "name": "AMU",
                  "location": "Uttar Pradesh"
               }
            }
         ]
   }
}
```

3. Describe what's possible with a type system

- GraphQL is strongly typed and the queries are based on fields and their associated data types.

- If there is type mismatch in a GraphQL query, server applications return clear and helpful error messages.

Example :

viewprofile and communication data types is given below −

```
type Query {
   viewprofile:[View]
}

type View {
   MatriId:ID!
   firstName:String
   lastName:String
   fullName:String
   communication:CMC
}

type CMC {
   name:String
   location:String
   rating:Float
}
```

4. Move faster with powerful developer tools

- GraphQL provides rich developer tools for documentation and testing queries.

- GraphiQL is an excellent tool which generates documentation of the query and its schema.

How to Build a GraphQL server with Nodejs

Create a Project Folder "graphqlServer"

Create package.json and Install the Dependencies

```
npm install express
```

Install GraphQL using the following command. We will be installing graphql and graphql for express.

```
npm install express-graphql graphql
```

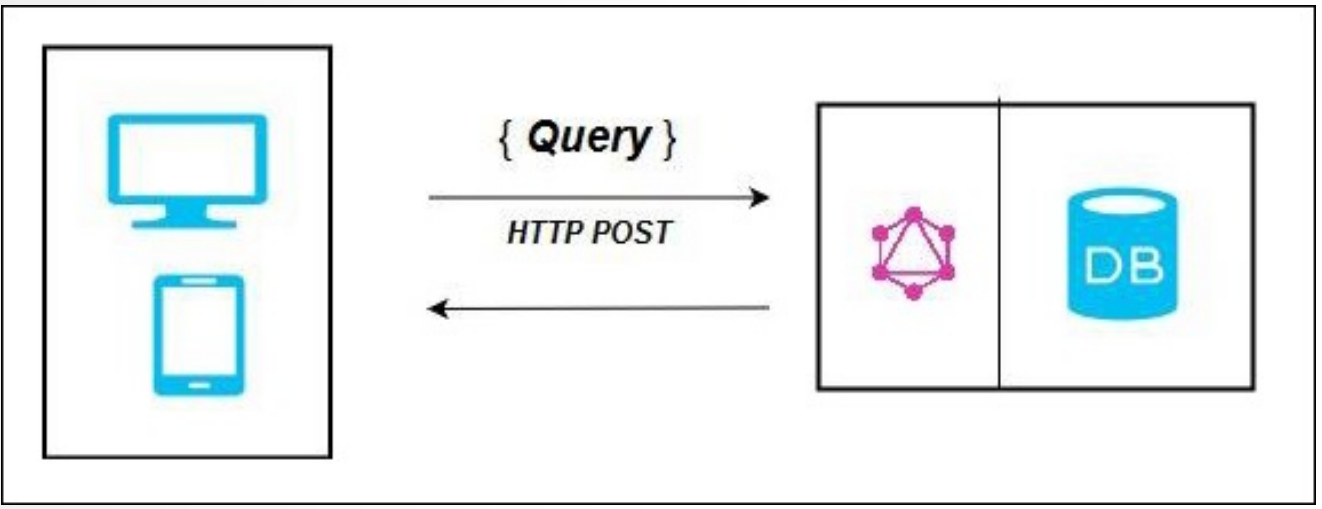
GraphQL - Architecture

> GraphQL server with connected database

> GraphQL server that integrates existing systems

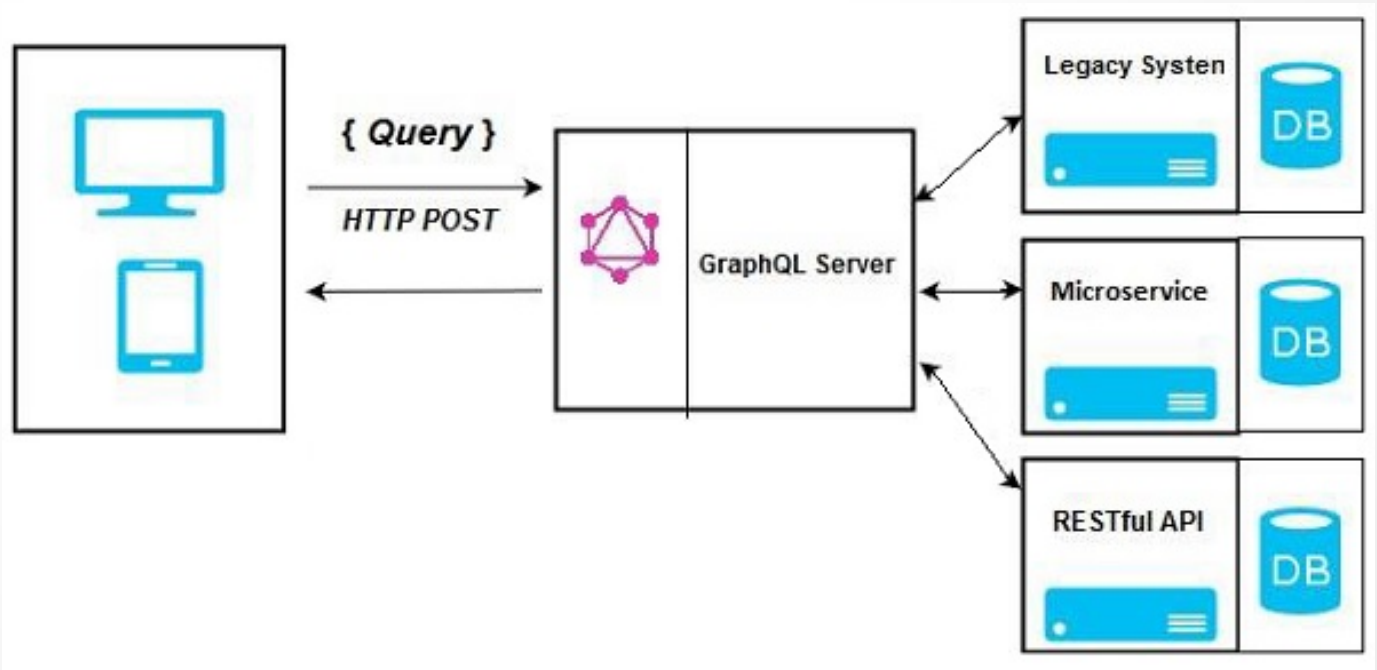
> Hybrid approach

GraphQL - Architecture



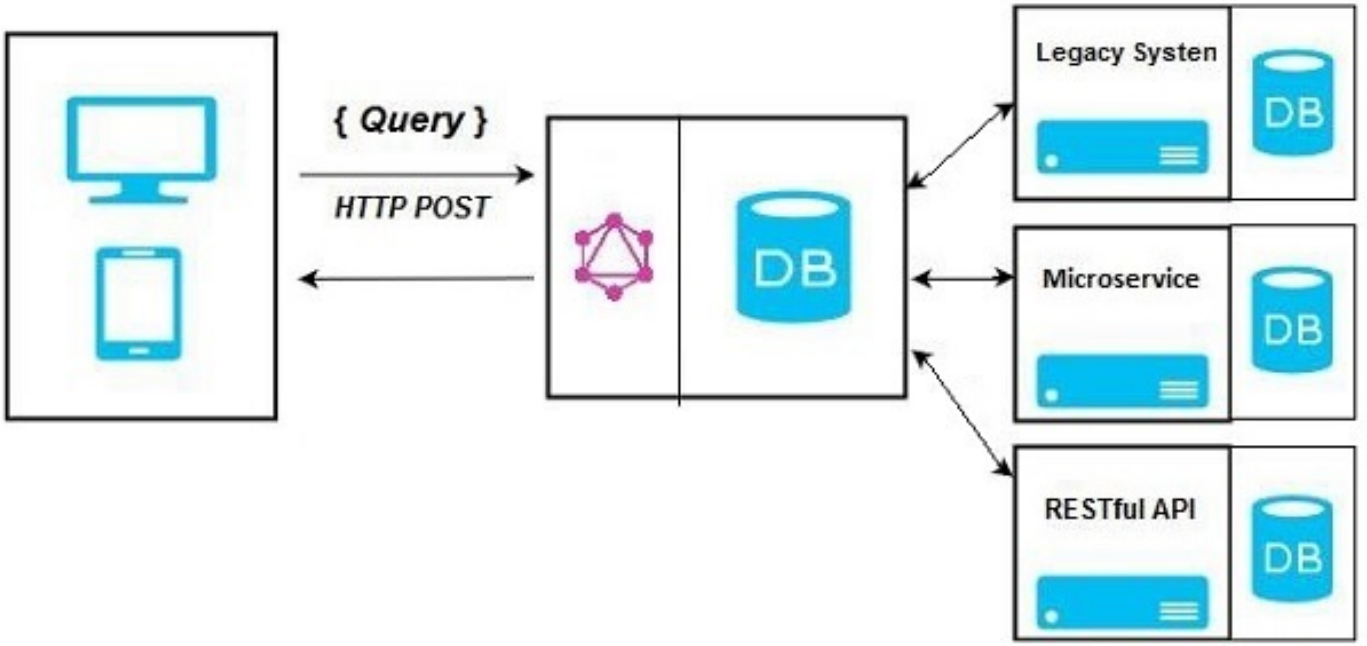
GraphQL Server with Connected Database

GraphQL Server Integrating Existing Systems



Hybrid Approach

It will either retrieve data from connected database or from the integrated API's.



GraphQL - Type System

> GraphQL is a strongly typed language.

> Type System defines various data types that can be used in a GraphQL application.

Types & Description

1. Scalar Type

Stores a single value

Int − Signed 32-bit Integer

Float − Signed double precision floating point value

String − UTF - 8-character sequence

Boolean − True or false

ID − A unique identifier

The syntax for defining a scalar type is as follows −

field: data_type

Firstname: String

2. Object Type :

Fetch a group of fields. Each field inside an object type maps to another type, thereby allowing nested types.

Define an object type

```
type Viewprofile {
  stud_id:ID
  firstname: String
  age: Int
  score:Float
}
```

Defining a GraphQL schema

```
type Query
{
  viewprofile:[Viewprofile]
}
```

3. Query Type :

A GraphQL query is used to fetch data. GraphQL uses the Schema Definition Language (SDL) to define a Query.

The syntax for defining a Query is as given below −

```
type Query {
  field1: data_type
  field2:data_type
  field2(param1:data_type,param2:data_type,...paramN:data_type):data_type
}
```

4. Mutation Type :

Mutations are operations sent to the server to create, update or delete data. These are analogous to the PUT, POST, PATCH and DELETE verbs to call REST-based APIs.

For example,

```
type Mutation {
    ViewprofileAdd(firstName: String, lastName: String): ViewProfile
}
```

5. Non-Nullable Type :

Exclamation mark (!) can be appended to a type

```
type ViewProfile {
  MatriId:ID!
  firstName:String
  lastName:String
  fullName:String
  Communication:Communication
}
```

6. Enum Type :

An Enum is similar to a scalar type.

7. List Type:

Lists can be used to represent an array of values of specific type.

```
type Query {
  todos: [String]
}
```

GraphQL - Schema

1. Schema is at the core of any GraphQL server implementation.
2. The root of the schema will be Query type.

Example :- query has two fields − Communication and viewprofile that returns String and a list of viewprofile respectively.

```
var SchemaType = `
    type Query {
      communication(ID: String!,VIEWEDID: String!,APPTYPE: String!):Communications
      viewprofile(ID: String!,VIEWEDID: String!,APPTYPE: String!): viewprofiles
    }

    type viewprofiles {
      RESPONSECODE: Int
      ERRCODE: Int
      PROFILEDET:ProfileDet
    }

    type ProfileDet{
      VPFLAG : Int
      WEBNOTIFICATION : Int
    }

    type Communications {
      RESPONSECODE : Int
      ERRCODE : Int
      CMCFLAG : Int
    }
}`
```

GraphQL - Resolver

GraphQL – Resolver

Resolver is a collection of functions that generate response for a GraphQL query.

```
var resolver = {
  viewprofile(args){
    return this.api.get('/view',{params:args}).then(res => res.data)
  },
  communication(args){
    return this.comapi.get('/cmc',{params:args}).then(res => res.data);
  }
};
```

GraphQL - Query

GraphQL - Query

```
var query = {
    "viewprofile":`{
        ERRCODE,
        RESPONSECODE,
        PROFILEDET
    }`,
    "communication":`{
        ERRCODE,
        RESPONSECODE,
        CMCFLAG,
        COMMUNICATIONACTION
    }`
};
```

GraphQL -

```
 graphql(schemaType, query, resolvers).then((response) => {
 res.send(response);
next();
});
```

GRAPHQL - DEMO

http://192.168.20.69:4000/graphql


Thanks