

➤ GraphQL :-

- GraphQL is a query language for API.
- GraphQL is an open-source server-side technology, now maintained by a large community of companies and individual all over the world.
- GraphQL is a runtime for fulfilling those queries with your existing data.
- GraphQL provides a complete and understandable description of the data in API, gives clients the power to ask for exactly what they need and nothing more.

➤ GraphQL Advantage:-

- Fetching data with a single API call.
- Good fit for complex systems and microservices.
- No over-fetching and under-fetching problems.
- Hierarchical Structure: - GraphQL follows a hierarchical structure where relationships between objects are defined in a graphical structure.

➤ GraphQL Support Concept: -

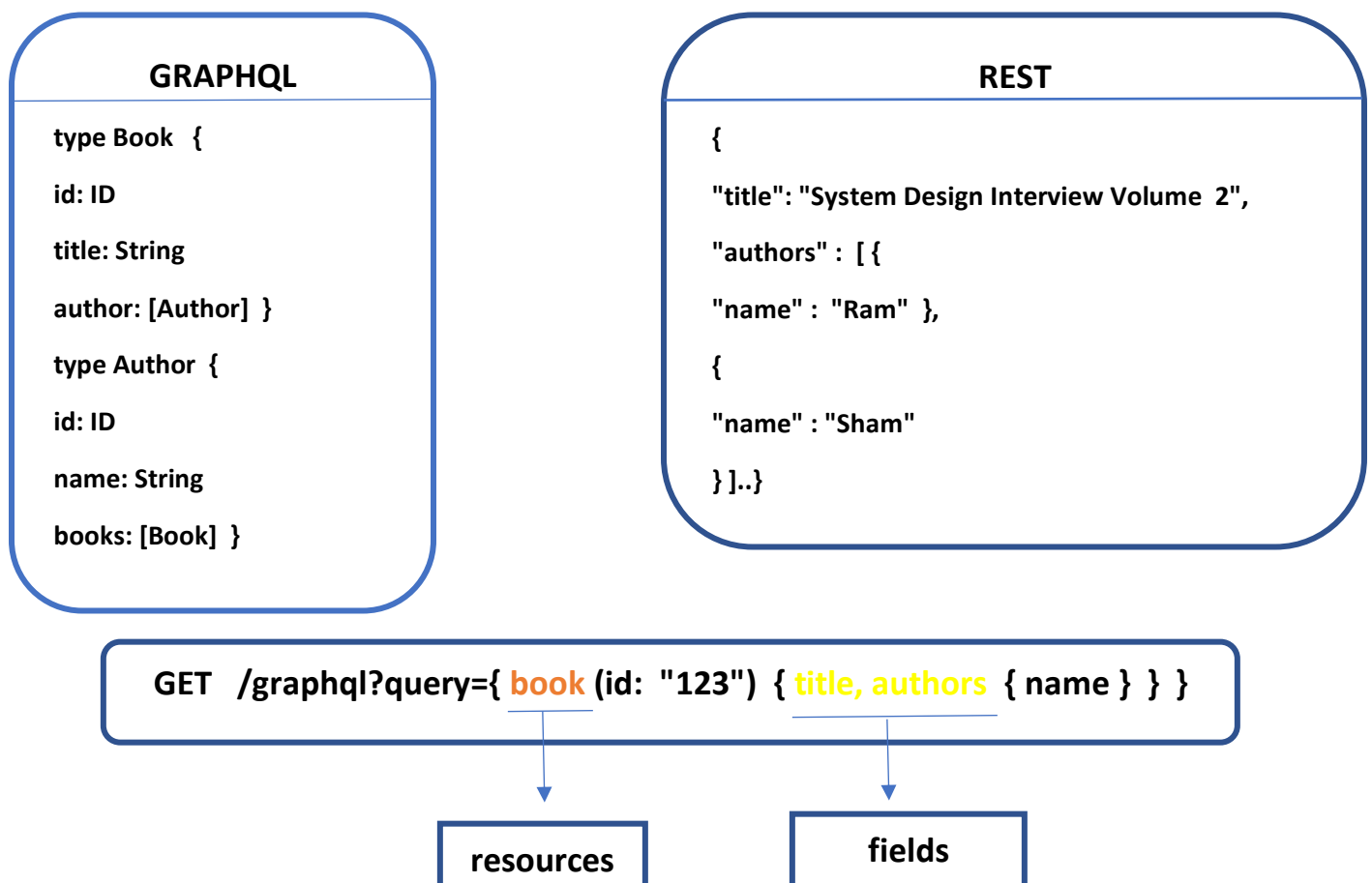
1. **Mutation:-** Mutations are GraphQL way of applying data modification to resources.

```
mutation Login($email: String!) {  
  login(email: $email) {  
    token }  }
```

2. **Subscription:-** Subscriptions are GraphQL way for clients to receive notification on data modifications.
3. **Query:-** When you want to get data from your GraphQL server.

➤ GraphQL VS Rest :-

- GraphQL and REST send HTTP request and receive HTTP response.
- Example:-



- GraphQL we specify the exact resources we want, and which fields we want. The client decides what to include.
- Rest example, the API implementer decided for us that authors are included as related resources.

➤ **GraphQL Apollo Server:-**

- Apollo Server is an open-source, spec-compliant GraphQL server that's compatible with any GraphQL client, including Apollo Client.

❑ **Started with Apollo Server:-**

1. **Create new project:-**

```
mkdir graphql-server-example  
cd graphql-server-example
```

2. **Install dependencies:-**

```
npm install @apollo/server graphql
```

3. Define GraphQL schema:-

```
import { ApolloServer } from '@apollo/server';
import { startStandaloneServer } from '@apollo/server/standalone';

const typeDefs = `#graphql
  type Employee {
    name: String!
    company: String!
  }
  type Query {
    employees: [Employee]!
  }
`;
```

4. Define data set:-

```
const employees = [
  {
    name: 'RAM',
    company: 'Apple',
  },
  {
    name: 'Sham',
    company: 'Zensar',
  },
];
```

5. Create an instance of ApolloServer:-

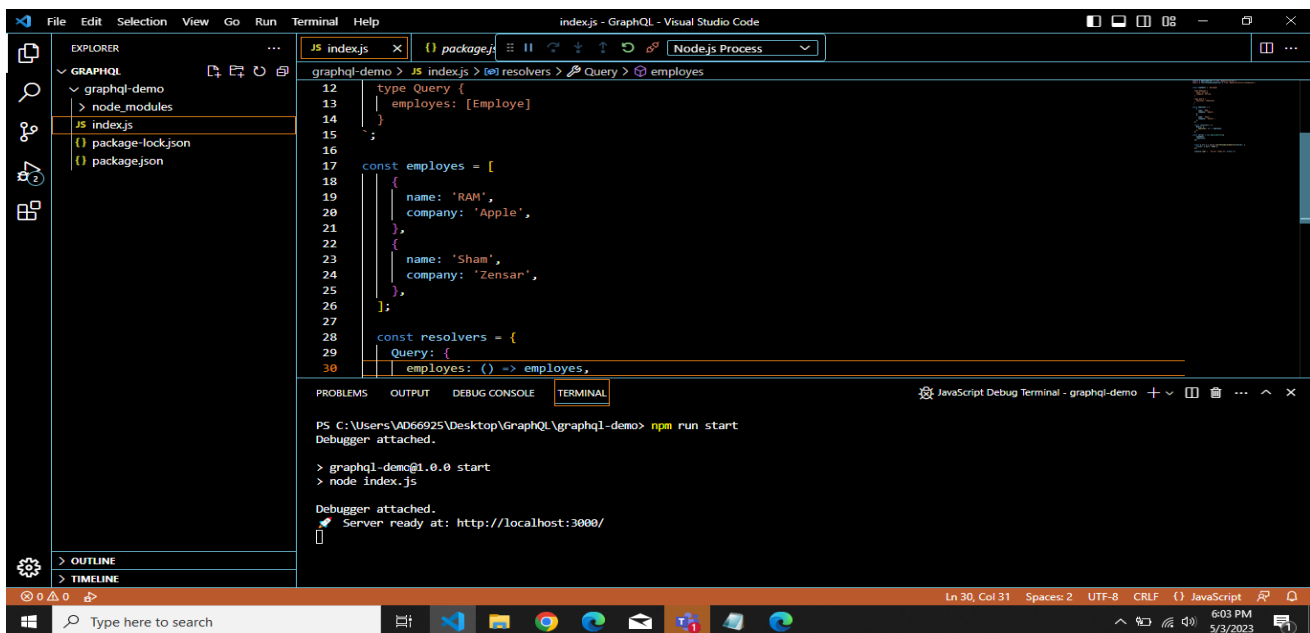
```
const server = new
ApolloServer({

  typeDefs,

  resolvers,

});
```

6. Start the server:-



The screenshot displays the Visual Studio Code interface for a project named 'graphql-demo'. The Explorer sidebar on the left shows the file structure with 'index.js' selected. The main editor window shows the content of 'index.js', which includes a GraphQL schema with a 'Query' type containing an 'employees' field, and a resolver for 'employees' that returns a list of employee objects. The terminal at the bottom shows the command 'npm run start' being executed, followed by the output 'Server ready at: http://localhost:3000/'.

```
index.js - GraphQL - Visual Studio Code

EXPLORER
  graphql-demo
    > node_modules
    JS index.js
    package-lock.json
    package.json

index.js
12 type Query {
13   employees: [Employee]
14 }
15
16
17 const employees = [
18   {
19     name: 'RAM',
20     company: 'Apple',
21   },
22   {
23     name: 'Sham',
24     company: 'Zensar',
25   },
26 ];
27
28 const resolvers = {
29   Query: {
30     employees: () => employees,
```

TERMINAL

```
PS C:\Users\VD66925\Desktop\GraphQL\graphql-demo> npm run start
Debugger attached.

> graphql-demos@1.0.0 start
> node index.js

Debugger attached.
Server ready at: http://localhost:3000/
```

7. Execute first query:-

The screenshot displays the Apollo Client Sandbox web application running in a browser at `localhost:3000`. The interface is divided into several sections:

- Documentation:** On the left, it shows the schema for the `employees` field, including the type `Employee` and its fields `company: String` and `name: String`.
- Operation:** The central panel shows a GraphQL query named `ExampleQuery` with the following structure:

```
1 query ExampleQuery {  
2   employees {  
3     company  
4   }  
5 }  
6
```
- Response:** The right panel displays the JSON response from the query execution:

```
{  
  "data": {  
    "employees": [  
      {  
        "company": "Apple"  
      },  
      {  
        "company": "Zensar"  
      }  
    ]  
  }  
}
```
- Variables:** At the bottom, there are tabs for `Variables`, `Headers`, and `Script`. The `Variables` tab is active, showing a `JSON` input field with a `+ Add files` button.

The status bar at the bottom right indicates a `STATUS 200`, a response time of `17.0ms`, and a size of `65B`. The system clock shows `6:03 PM 5/3/2023`.