GraphQL

version 1.0.0,

October 2022

What is GraphQL

- a query language for APIs
- a runtime for fulfilling queries

How does GraphQL Work?

- Query using JSON-like syntax
- Data returned matches shape of query

Example Query and Response

```
{
    customer {
        firstname
        lastname
        address
        city
        state
        zip
    }
}
```

```
{
    ”customer”: {
        ”firstname”: “June”,
        “lastname”:”Sommerville”,
        “address”:”861 Woodland Terrace”,
        “city”:”Sacramento”,
        “state”:”California”,
        “zip”:”95814”
    }
}
```

GraphQL is a pattern

- Can be implemented using any language
- Tools are available for working with JavaScript, Go, PHP, Java, C#, Python, Swift, Rust, Ruby, and more.

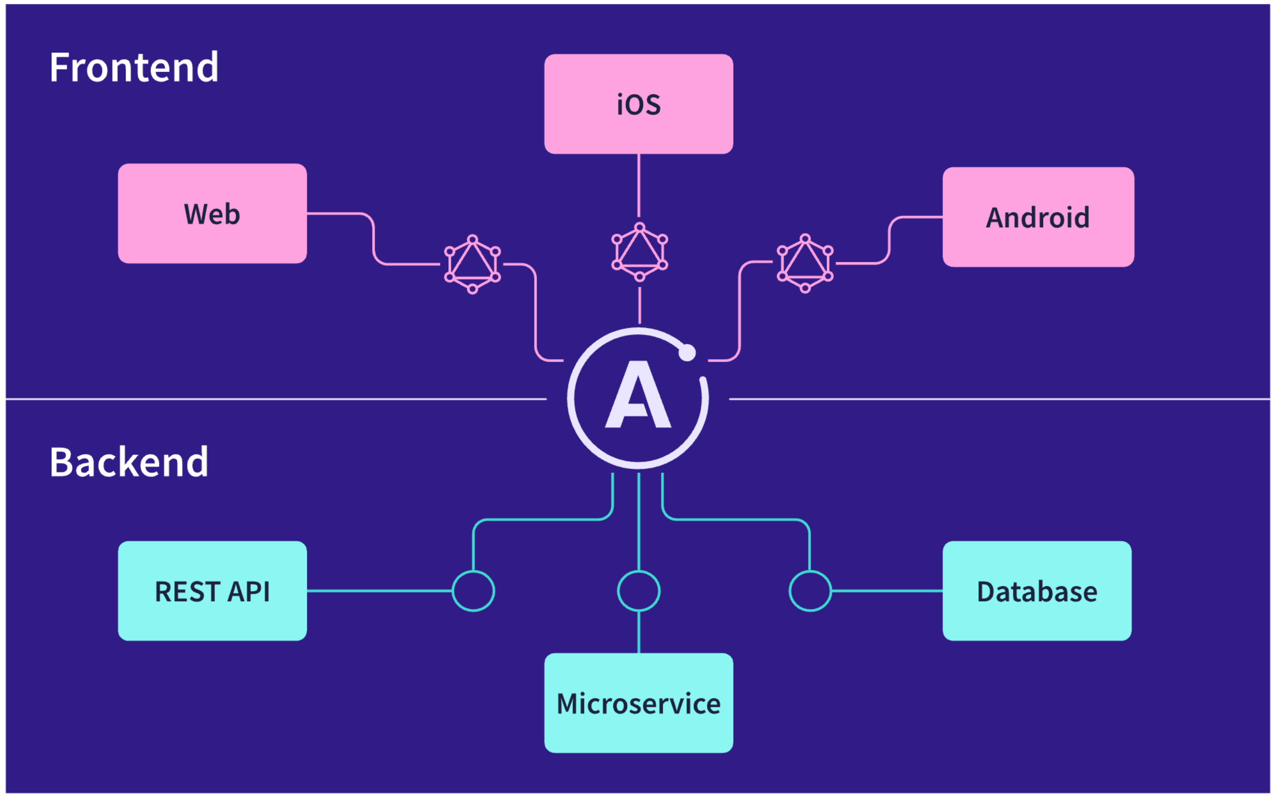
GraphQL and JavaScript

- Server
  - Apollo Server
  - Express GraphQL
- Client
  - Apollo Client
  - AWS Amplify
  - Relay

Lab 1: Getting Started with Apollo Server

What is Apollo Server?

- open source GraphQL server



Apollo Studio Explorer

- a free web-based IDE for GraphQL
- a tool for building GraphQL servers using a schema

GraphQL Schema

- defines object types that can be fetched
- created using GraphQL Schema Language

Components of a GraphQL schema

- Object Types
- Query Types
- Mutation Types
- Subscription Types

Anatomy of a Type

- Types are written similarly to how TypeScript Types are written
- Object types start with type followed by the Name of the object

```
type Customer {
```

- The body is key: type pairs

```
firstName: String
lastName: String
email: String
orders: [Order]
```

Basic Data types

- Available data types are:
  - Scalar types
    - String
    - Int
    - Float
    - Boolean
    - ID (a unique ID not meant for human reading)
  - List types
    - Surround a type with []

Not null

- Add ! after a type to indicate that it's required

```
type Customer {
  id: ID!
  firstName: String!
  lastName: String!
  orders: [Order}
}
```

Query and Mutation Types

- Define entry points to the schema

```
type Query {
customers: [Customer]
orders: [Orders]
}
```

Passing Arguments

- Pass arguments using parentheses after the name.

```
type Query {
  customer(id:ID!):Customer
  order(id:ID!):Order
}
type Mutation {
  addCustomer (firstName:String!,lastName:String!,email:String): Customer
}
```

Lab 2: Using Apollo Studio

Apollo Client

- A state management library for JavaScript
- Manages both local and remote data with GraphQL
- To create a client instance:

```
const client = new ApolloClient({
  uri: 'http://localhost:4000’,
  cache: new InMemoryCache(),
});
```

Apollo Dev Tools

- A Chrome and Firefox extension for Apollo Client
- To enable (after installation):

```
const client = new ApolloClient({
uri: 'http://localhost:4000',
cache: new InMemoryCache(),
connectToDevTools: true,
});
```

Lab 3: Creating a client

Connecting to Data

1. Create a datasource
2. Define resolvers

Lab 4: Connecting to a data source

Lab 5: Creating Resolvers

Lab 6: Setting up the server for Mutations

Lab 7: Making mutations from a client