International Conference on Innovative Research in Engineering, Management and Sciences
International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2019 IJSRCSEIT | Volume 4 | Issue 9 | ISSN : 2456-3307

Analyzing GraphQL and implementing the framework on Android devices

S Sivan Chakravarthy*1

*1Department of Computer Science and Engineering Bangalore, India
sivan.sundar@gmail.com1

ABSTRACT

This paper is a highlights how mobile devices can query data and information efficiently by using GraphQL. This paper reviews the the Graph QL framework and discusses its role in making intelligent requests possible. To improve data efficiency and reduce device overhead, we will be using GraphQL in relaying queries to APIs.

Keywords : GraphQL, Querying Language, Android, Mobile devices.

I. INTRODUCTION

Data Query Languages (DQL's) or Query Languages (QL's) are used to create queries to communicate with information systems and databases. QL's make it easier to handle data from server sources which hosts huge chunks of data. They were primarily made to handle creating, deleting, accessing, and modifying data with databases. There are multiple QL's like Contextual Query Language (CQL), Java Query Language (JQL) etc but the one query language in our day to day services that we are accustomed to is the REST QL's or simply REST APIs. Recently it was convoluted that REST API's still faced a couple of problems with respect to making multiple routes to endpoints and retrieving a bunch of data which is not going to be of full purpose.

Although the REST architecture was phenomenal, it had its own shortcomings which weren't addressed. When it comes to REST, everything component is handled as a resource. Using HTTP allows you to operations like GET, POST, PUT and DELETE but the problem was the multiple rounds it was making at multiple endpoints to retrieve data. Another common problem that REST possessed was over fetching and under fetching of data. For every request initiated, we would retrieve a huge dataset from which we need to extract the data that we need. This indeed posed a huge load on the network receptors and the devices too. For example, if a blog post consisted of properties like : id, user, title and body, using a REST request we would end up downloading the entire set and there would be no way to limit the response to contain only certain specific fields like title and user.

II. GRAPHQL

In 2015, Facebook decided to come up with a new query language to solve the existing problem that the REST API's faced. It ended up creating a dent in the online space when it came to consuming data. Graph Query Language or GraphQL was born. GraphQL was an excitingly new prospect to help imagine data

in a new way. The major shortcomings that REST posed were eliminated with this new venture. Although it was established in 2015, it gained prominence in no time.

GraphQL is not wired in to any specific storage structure or database and instead is backed by your data and code. A GraphQL service is created by defining fields and types and then providing functions for each.

Once a GraphQL service is set up, (usually on a web service) it can be sent GraphQL queries to verify and execute. The query is first checked to ensure it refers to the defined fields and types and then the functions are executed to produce the required result.

2.1 - SCHEMAS AND TYPES

We will now look into the schemas and types associated with GraphQL and how we can create leverage out of them. As GraphQL can be used by any programming language or framework, we will look into the concepts rather than the implementation-specific details.

GraphQL services can be written in any language. As we can't stick to one specific language to talk about or handle GraphQL like lets say Python, Rails or Javascript, we define our own language called the GraphQL Schema language. It helps us to communicate in a language-agnostic way. The GraphQL schema consists of a basic component called object-types which represents the kind of object that can be retrieved from your service. Let us consider this piece of code :

```
type Character {
  name: String!
  appearsIn: [Episode!]!
}
```

Fig 2.1 - Graph QL Type

Character is a GraphQL object type with some fields. The appearsIn and name are fields on the character type. String! is a non-nullable built-in scalar types which holds in strings and Episodes! is a non-nullable type which holds in an array of Episode objects.

2.2 - INTERFACES

GraphQL supports interfaces. It is nothing but an abstract type which includes certain fields that a type requires to implement an interface. Let us consider this interface :

```
interface Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
}
```

Fig 2.2 (a) - An Interface.

Here, the type that implements this interface should needs to have the same fields with the same return types and arguments.

```
type Human implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  starships: [Starship]
  totalCredits: Int
}

type Droid implements Character {
  id: ID!
  name: String!
  friends: [Character]
  appearsIn: [Episode]!
  primaryFunction: String
}
```

Fig 2.2 (b) - Implementing an Interface

Both of these types have fields from Character interface, but also contain extra fields that are pertinent only to that specific Character type. When we want to return a specific set of objects, interfaces are useful.

III. ADVANTAGES OF GRAPHQL

One of the biggest advantages of GraphQL is that it is client-driven which means that you get what you want. We get to define the type of response and

therefore the client on server has more control power. We can end up doing multiple calls which places a huge burden on both the device and server. Instead of bouncing off multiple endpoints, we can hit one endpoint and get what we want.

IV. SETTING GRAPHQL ON ANDROID

To get started on Android, we need to have a bunch of libraries and dependencies. At first, we are going to configure apollo-graphql which is a caching library for graphql written in Java with the following lines of code being added to the project.gradle file.

```
//In your project build.gradle
dependencies {
    classpath 'com.android.tools.build:gradle:3.0.1'
    classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:1.2.21"
    classpath 'com.apollographql.apollo:gradle-plugin:0.3.2' //Add
this

    // NOTE: Do not place your application dependencies here; they
belong
    // in the individual module build.gradle files
}
```

Fig 4.0.1 - Dependencies

There are a bunch of other dependencies to be added like

- implementation “com.apollographql.apollo:apollo-runtime:0.3.2'
- implementation "com.apollographql.apollo:apollo-android-support:0.3.2"

Additional plugins need to be added to build.gradle file like

- apply plugin: 'com.android.application'
- apply plugin: 'kotlin-android'
- apply plugin: 'kotlin-android-extensions'
- apply plugin: 'com.apollographql.android'

Now we need to be able to generate code-gen files which will allow us to take the schemas of the GraphQL queries and convert them into Java classes. Apollo-codegen is an amazing library to get this job done. For this, we need to install apollo-codegen via npm and then create a directory called “graphql” under the /src/main directory. This will host the schema file with a .json extension which will contain the responses of the introspection queries.

Now since we have all the dependencies set up, it is time to wire in. To demonstrate an example in this paper, we will be considering GraphQL Api of Github. We are going to use OkHTTP as our networking client and add headers and receptors to it if needed. This client also supports a level 3 caching so we have this ar our disposal too. Now we need to create an apollo-client object and attach our OkHTTP networking object which is going to initiate requests. Make sure you specify your base url of the api too. A FeedQuery object will be able to set parameters like limit and type to our GraphQL queries. The getters and setters are automatically generated by apollo. The .graphQL file will contain our queries so let us make sure we have them well defined.

```
query FindQuery($owner:String!,$name:String!){
  repository(owner:$owner, name:$name) {
    name
    description
    forkCount
    url
  }
}
```

Fig 4.0.2 - GraphQL queries in the .graphQL file

Now we must create an Apollo call, which takes in the data of the FeedQuery object as its type. Also we need to set the query of this to the feed-query object that we initially created. GraphQL supports both RxJava and callback methods. As far as the callback method is concerned we will call the enqueue method on the callback ApolloCall object and the enqueue method executes asynchronously which does not affect the main thread. We then obtain a nonnull response object from which we can obtain the raw data.

V. VALIDATION

The type system tells us if a query is valid or not. This helps the developers by keeping them informed about the validity of the query and if runtime checks can be performed on it or not. A test file can also be run on the queries to check the correctness of them.

```
{
  hero {
    ...NameAndAppearances
    friends {
      ...NameAndAppearances
      friends {
        ...NameAndAppearances
      }
    }
  }
}

fragment NameAndAppearances on Character {
  name
  appearsIn
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Luke Skywalker",
          "appearsIn": [
            "NEWHOPE",
            "EMPIRE",
            "JEDI"
          ],
          "friends": [
```

Fig 5.0.1 - A valid query

VI. EXECUTION

After validating, the query is executed by a GraphQL server which returns a result which resembles the requested query in a JSON structure. The GraphQL query cannot execute without a type system. Each query of a GraphQL function can be described as a function or a method. Each field is backed by a resolver. If a field returns a number or a string, then it is complete.If the field returns an object, then the query contains a selection of fields that pertain to that object.

The graphQL server represents all possible entry points to GraphQL API queries as root type.

```
Query: {
  human(obj, args, context, info) {
    return context.db.loadHumanByID(args.id).then(
      userData => new Human(userData)
    )
  }
}
```

Fig 6.0 - Rootfields

The above query provides human is a query type that accepts id as the argument. To access a database, context is used to grant access. The query returns a promise since it is asynchronous. GraphQL waits for tasks, futures and promises to complete which ensures optimal concurrency.

VII. CONCLUSION

In this paper, we have clearly analyzed how to implement GraphQL in mobile systems and migrate to a better way of retrieving data from APIs. We have also discussed the use of GraphQL and how advantageous it is when it comes to bouncing off one endpoint to retrieve necessary data. Being able to quickly configure endpoints for your server with the language of your choice makes it much more flexible to use this querying language. There is no language or best practices as such for different platforms while using graphQL but this is more of a methodology which focuses on the implementation aspects of the technology.

FUTURE WORKS

There are a few setbacks to GraphQL but it has really come a long way in providing an alternative protocol to query data. GraphQL as such has quite a few problems and this can be addressed in improving the efficiency of this querying language. Complex querying is an issue in graphQL because once the user requests too many nested objects at once, the components struggle to handle this situation, so for smaller applications, REST still works best. Rate limiting is also something that needs to be considered since everything between expensive to cheap operations can be performed and that at

someway undermines capacity of the data being requested.

VIII. REFERENCES

[1] "GraphQL: Core Features, Architecture, Pros and Cons." AltexSoft, www.altexsoft.com/blog/engineering/graphql-core-features-architecture-pros-and-cons/.

[2] Why GraphQL: Advantages, Disadvantages & Alternatives. (2018, July 03). Retrieved from https://www.robinwieruch.de/why-graphql-advantages-disadvantages-alternatives/

[3] GraphQL: A query language for APIs. Retrieved from https://graphql.org/

[4] /@pranayairan. (2017, July 10). Hello Apollo Writing Your First Android App With GraphQL. Retrieved from https://android.jlelse.eu/hello-apollo-writing-your-first-android-app-with-graphql-d8edabb35a2

[5] /@stubailo. (2018, April 6). GraphQL vs. REST. Retrieved from https://blog.apollographql.com/graphql-vs-rest-5d425123e34b