AN ALTERNATIVE TO REST APIS

GRAPHQL

WHY GRAPHQL?

- Type system to represent API which allows it to be self documenting and easily explorable by new developers
- Ask for exactly what you want in a JSON-like format
- Get all the data you need without multiple round trips
- Easily wrap other complex APIs with a simplified interface
- Take advantage of node/edge parent -> child resolver relationships to make data retrieval fast and cacheable

GRAPHQL VS REST

- GraphQL does not replace REST but it can serve as an alternative for certain use cases
- Imagine what a REST endpoint would look like to retrieve particular fields
- Many REST APIs center around representing resources around URIs. This leads us to the N+1 problem.
- GraphQL solves this by embracing types with parent -> child relationships
- To sum it up, in REST, the space of accessible data is described as a linear list of endpoints, and in GraphQL it’s a schema with relationships

GRAPHQL TYPES

- GraphQL comes with a set of default scalar types out of the box: Int, Float, String, Boolean, and ID
- Int: A signed 32-bit integer
- Float: A signed double-precision floating point value
- String: A UTF-8 character sequence
- ID: The ID scalar type represents a unique identifier, often used to reflect an object or as the key for a cache

GRAPHQL SCHEMA

- The building blocks of a GraphQL schema are object types and fields (using the scalar types we mentioned above)
- Here is an example (more at http://graphql.org/learn/schema/):

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

DEVELOPER TOOLS

- GraphiQL: A graphical interactive in-browser GraphQL IDE https://github.com/graphql/graphiql (we’re going to use this one)
- Apollo client devtools: An extension for Chrome or Firefox The devtools appear as an "Apollo" tab in your Chrome inspector, along side the “Elements" and "Console" tabs. There are currently 3 main features of the devtools: GraphiQL, store inspector, and query inspector https://github.com/apollographql/apollo-client-devtools

GRAPHQL EXECUTION

- At the top level of every GraphQL schema is a type that represents all of the possible entry points into the GraphQL API, it's often called the Root type or the Query type
- From that entry point we can have types that expose fields which can be of other types. This is why we get this nested "JSON structure"
- For example, say we have a query that returns a list of users. This would be represented as an array of user types in GraphQL
- Each user may also have its own list of fields (which potentially map to another resolver) which can resolve additional details about the user – for instance an array of the cats they own

GRAPHQL BATCHING AND CACHEING

- You might look at the previous slide and start freaking out. What if we have a schema that has a list of users where each user can list a list of their friends and so on and so on. Somebody could create a crazy nested query and bring the server down to its knees. Looks like we just introduced the N+1 problem again from REST
- Enter Dataloader, a mechanism to only load data we need and batch queries where possible
- At a high level Dataloader loads data by using keys. These keys can be derived from a number of things. If Dataloader gets multiple requests for the same key, it will not do extra work to get that same data. We can also configure Dataloader to batch certain requests. For instance, do we need to do a query for each and every friend in the friends array of a user? Probably not. We can batch this will Dataloader and a query to our persistence layer that accommodates that.
- For an in depth explanation, please see: https://github.com/facebook/dataloader

TIME TO GET OUR FEET WET

- Pull down https://github.com/bloodhawk/graphql_react_starter and run:

```
Aarons-MBP:graphql_react aaronrumery$ yarn install && yarn start
```

- It's probably broken if you try to start it. Let's get it working by pulling down: https://github.com/bloodhawk/graphql_starter and running:

```
Aarons-MBP:basic aaronrumery$ yarn install && nodemon index.js
```

- Let's start doing the challenges in the graphql_starter repo

WHERE TO GO FROM HERE?

- Watch: https://www.youtube.com/watch?v=UBGzsb2UkeY
- Explore and learn from https://www.howtographql.com/
- Explore and learn from http://graphql.org/
- Have https://wehavefaces.net/graphql-shorthand-notation-cheatsheet-17cd715861b6 as a reference