A QUERY LANGUAGE FOR YOUR API

GRAPHQL

WHY

- Most API calls are wasteful. They send back more resources than are called for
- Allows a single API call to easily get data from multiple sources in a single request (helps to handle high latency situations)
- Self documenting APIs by default (GraphiQL is awesome)

CONCEPTS

- Top level types
  - Query - Entry points into your graph that allow you to retrieve data
  - Mutators - Allow you to make changes to a resource in the graph

CONCEPTS

- Query Date types
  - Field - a property on the object. Fields can be primitives, enums, other data types, etc
  - Argument - used to retrieve a specific instance of an object or to filter a list. Work just like path and query params in traditional REST APIs
  - Alias - Allow you to change the return name of a property without the GraphQL owner having to make changes to the data contract
  - Fragment - Reusable chunk of GraphQL query. Useful for when you want to request the same fields from multiple nodes in the graph
  - Variable - Allows you to pass variables into your arguments

CONCEPTS

- Mutation Data Types
  - An action with arguments
  - Also contains the list of fields that you want returned
  - An example is inserting a new row into a table and requesting that the generated ID property be returned
- Many more concepts around query and mutation, but not enough time to cover them. See graphql.org for more.

CONCEPTS

- Schema
  - Defines the data types that your graph can make use of
  - Can be defined in the GraphQL schema language (nice and clean), or programmatically (I personally find this syntax very hard to read and very complex but to each their own)
  - Examples of each here: http://dev.apollodata.com/tools/graphql-tools/generate-schema.html

CONCEPTS

- Resolvers
  - Resolvers are what actually query your DB, make a call to another API, or just pull your data from wherever it happens to live.
  - Resolvers can either be synchronous or return a promise (they play well with async/await)
  - GraphQL libraries will let you omit resolvers this simple and will just assume that if a resolver isn't provided for a field, that a property of the same name should be read and returned

CONCEPTS

- Resolvers can be applied at multiple levels:
    - The whole object
        - Required for top level nodes on the graph
        - This is where you will fetch the initial data load
        - Not required for children data type only if the initial data load contains the data needed for the child type and it's in the correct format
    - Individual property
        - Allow you to fetch data for a specific property from a secondary date score
    - Allow you to create new properties that don't exist in your stored data (think along the lines of combining a firstName and lastName field into a displayName field)

- THERE ARE FAR TOO MANY CONCEPTS TO DISCUSS IN THE ALLOTTED TIME
- CHECKOUT GRAPNEL.ORG FOR SOME AMAZING DOCUMENTATION
- CHECKOUT DEV.APOLLODATA.COM FOR IMPLEMENTATION DETAILS