Intro to Client GraphQL with React

By developer :: RaJu Paan WAla

GraphQL, the big picture

What is GraphQL?

A spec that describes a declarative query language that your clients can use to ask an API for the exact data they want. This is achieved by creating a strongly typed Schema for your API, ultimate flexibility in how your API can resolve data, and client queries validated against your Schema.

It’s just a spec. There are several implementations and variations

Server Side

- Type Definitions
- Resolvers
- Query Definitions
- Mutation Definitions
- Composition
- Schema

Client Side

- Queries
- Mutations
- Fragments

Where does GraphQL fit in?

- A GraphQL server with a connected DB (most greenfields)
- A GraphQL server as a layer in front of many 3rd party services and connects them all with one GraphQL API
- A hybrid approach where a GraphQL server has a connected DB and also communicates with 3rd party services

Queries and Mutations from the client

Operation names

Unique names for your client side Query and Mutation operations. Used for client side caching, indexing inside of tools like GraphQL playground, etc. Like naming your functions in JS vs keeping them anonymous.

Variables with operations

Operations can define arguments, very much like a function in most programming languages. Those variables can then be passed to query / mutation calls inside the operation as arguments. Variables are expected to be given at run time during operation execution from your client.

Variables with operations

Operations can define arguments, very much like a function in most programming languages. Those variables can then be passed to query / mutation calls inside the operation as arguments. Variables are expected to be given at run time during operation execution from your client.

Apollo Client

What is Apollo Client

Encapsulates HTTP logic used to interact with a GraphQL API. Doubles as a client side state management alternative as well. If your GraphQL API is also an Apollo Server, provides some extra features. Offers a plug approach for extending its capabilities. It’s also framework independent.

Storing data from your API

- All nodes are stored flat by an unique ID
- Unique ID is defaulted to .id or ._id from nodes. You can change this
- Every node should send an .id or ._id, or none at all. Or you have to customize that logic

Queries in React

Mutations in React

Keeping Cache in Sync

Why is the cache out of sync?

If you perform a mutation that updates or creates a single node, then apollo will update your cache automatically given the mutation and query has the same fields and id.

If you perform a mutation that updates a node in a list or removes a node, you are responsible for updating any queries referencing that list or node. There are many ways to do this with apollo.

Keeping cache in sync

- Refetch matching queries after a mutation
- Use update method on mutation
- Watch Queries

Optimistic UI

What is a Optimistic UI?

Your UI does not wait until after a mutation operation to update itself. Instead, it anticipates the response from the API and proceeds as if the API call was sync. The the API response replaces the generated one. This gives the illusion of your being really fast.

Optimistic UI with mutations

Apollo provides a simple hook that allows you to write to the local cache after a mutation.

Client Side Schemas

Why?

In addition to managing data from your API, apollo client can also local state originated from your front end app. Stuff you would normally store in something like Redux or Vuex. You can create a schema to define that state which allows you to query for that state the same way you query your API for data.

How?

The exact same way as the server. You just have to extend the Types from your server schema. You then use a directive to access local state from your queries and mutations.

You made it 💯