Advanced GraphQL V2

Scott Moss
FrontEnd Masters

Refresher

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.

The main parts

- Type Definitions
- Resolvers
- Schema
- Data Sources
- Queries
- Mutations

Tools and libs

- graphql-js
- Apollo Server
- Express
- GraphQL Playground

Let’s build a GraphQL API!

Follow along

Authentication & Authorization

Authentication

Used to identify a user. To determine if they are who they say they are.

Authorization

Used to determine if a user is allowed to perform certain operations on certain resources.

A good auth system in GraphQL

Authorization

- Should not be coupled to a resolver
- Can provide field level custom rules
- Can authorize some of your schema and not all

Authentication

- Provides the user to resolvers
- Should not be coupled to a resolver
- Can protect some of your Schema and not all of it
- Can provide field level protection

So many ways to auth

- Outside of GraphQL
- When creating context
- Inside the resolvers

Outside of GraphQL

- Using something like Express middleware before the GraphQL Server executes
- Completely locks down all GraphQL queries and mutations
- Extra complexity of passing auth info to GraphQL

When creating context

- Use context creation function when creating your apollo server
- Can access the incoming request to determine authentication
- No extra work to pass to GraphQL resolvers

Inside resolvers

- Business logic tied up with authentication logic
- The simplest to implement
- The hardest to reuse

Your turn

- Create authenticated helper
- Create authorized helper
- Wrap resolvers that use users for db queries with authenticated helper
- Wrap resolvers that require an Admin role with authorized helper

Let’s get **real-**time

Real-time with GraphQL

- **Subscriptions** - A well supported GraphQL operation that's useful for notifying clients of events
- **Live Queries** - Client side implementation to be notified when data changes

Subscriptions vs Live Queries

Subscriptions

- Part of the spec
- Event observation
- Great support

Live Queries

- Experimental
- Data observation
- Support is getting there

Both

- Flexible transports and protocols
- Predictable responses

Just Use Subscriptions

But why?

- If manual refetching and polling have too high of a latency cost (chat apps)
- Initial state is huge, but changes are small
- Live queries are just not there, yet

Adding Subscription Support

- Subscriptions must be added to your Schema like Queries and Mutations
- Setup PubSub protocol server side
- Create Subscription event resolvers
- Add any needed authentication and context
- Client side setup

Let’s create some Subscriptions.

Follow along

Your turn

- Add a subscription for new posts
- Publish to the Subscription when a post is created
- Add authentication to the subscription
- Add filter for Subscription (extra credit)
- Try out your Subscriptions with GraphQL playground

Error handling

Throw your errors, they will be caught

Let’s break some things.

Follow along

Testing

Testing Resolvers

- Unit test resolver functions
- Mock out data sources
- Mock out db calls

Testing Schema

- Convert TypeDefs into Schema
- Unit test Object types

Testing the server

- Integration testing the entire server
- Create a test client to use to issue queries and mutations with against a testing instance of your server
- Mock out variables, context, data sources

Your turn

- Use Apollo error classes to handle errors
- Create integration tests for your queries and mutations
- Create unit tests for your resolvers

Creating Flexibility with Directives

What are they?

Allows you to add logic and metadata to your Schemas, Queries, or Mutations. Directives can act like middleware for your Schemas, or post processing hooks for your Queries and Mutations.

Why Use Directives?

- Fine-grain control down to the field level on your TypeDefs
- Eliminate post processing on your clients after you query
- Extendable
- Extremely powerful

Using Directives

Follow along

Creating Custom Directives

- Can be challenging if you’re unfamiliar with how GraphQL works
- Requires a definition in your schema
- Create logic for when Directive is used

Let’s create some Directives

Follow along

Your turn

- Create a custom date format directive for fields that formats timestamps. (except args extra credit)
- Create field level auth directive (extra credit)
- Add your directives to your Schema

Production check

Checklist

- Set environment to Production
- Don’t hard code your port
- Pick a platform

Let’s go live

Follow along

Caching

Many types of caching

- Application Caching
    - DB
    - External data source
    - Resolvers
- Network Caching
    - HTTP caching
- Client-side caching

Application Caching

- Prefered way to cache GraphQL right now
- Have many options and levels to cache depending on your server
- A bunch of misunderstandings around HTTP caching and GraphQL

HTTP Caching

- Can be more complicated unless you use services and features like Apollo Cache Control, Engine, and Automatic Persisted Queries
- Can use edge applications to program your own cache logic
- Restrict or handle Mutations over /GET

Client side caching

- Apollo client handles this pretty well
- Use any client-side state management you already use (redux, vuex, rx, etc)
- Persisted Queries in coordination with the server

How should you cache?

- If you’re able to use HTTP caching, enable it
- Cache external HTTP data sources
- Cache client-side

QUESTIONS?

You are now a GraphQL API pro 💯