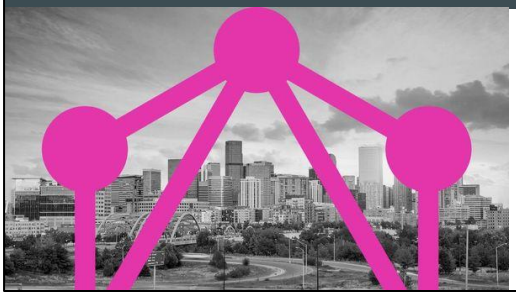
# Creating GraphQL Schemas with the OpenAPI Spec




## Craig Freeman

@craigfreeman


GraphQL Denver 01.28.20

# Agenda

---

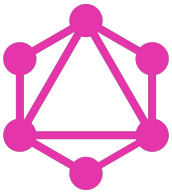
- GraphQL Review
- Sales Pitch
- OpenAPI Spec
- Conversion
- Demo

This talk is geared towards those who have an understanding of GraphQL and are excited to use it, but may be having some difficulty convincing your team member or managers to use it.
We'll briefly go over the OpenAPI Spec or Swagger spec and then walk through how to get a GraphQL service up and running with your existing API using some tooling.

# Intro

- Senior Software Engineer at Twilio
- Developing for the web since 2010
- Developing with GraphQL since 2018

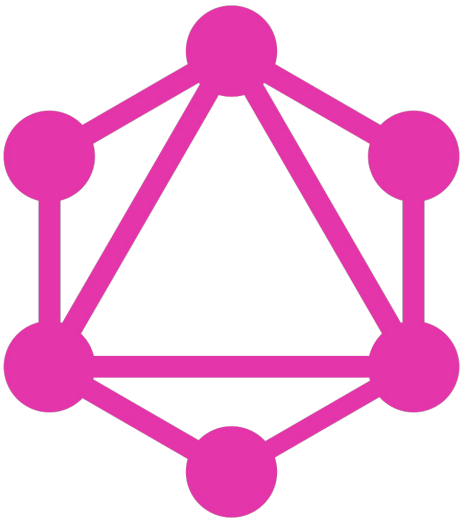




# Why Should I Use GraphQL?

---

- Consolidation of API requests
- Backend-for-Frontend
- Self-documenting
- Great for multiple interfaces



- Consolidates a multitude of APIs down to a single endpoint. Not only that, but multiple requests for data can be combined into a single query that is executed in parallel on the server.
- It puts the consumer of the API in full control over what data they receive, ideally in a single smaller HTTP request, which prevents overfetching (querying unneeded data).
- Uses a single, evolving version of the schema by deprecating fields when no longer needed.
- Web app, iOS, Android all using the same data

# Why Is It A Tough Sell?

- Schemas
- Scalars
- Resolvers
- Queries
- Mutations
- Subscriptions
- Caching
- Directives
- Fragments
- Introspection
- Authorization?
- Mocks?
- Scaling?
- How is schema modeling different from database

There’s a huge learning curve. There’s a lot to learn, lots of new concepts, and it’s a departure from standard REST API development. It can be overwhelming.
The argument is a reasonable one: Time is money. Why should the business devote resources to all that?

# Why Is It A Tough Sell?

---

Build a Proof of Concept

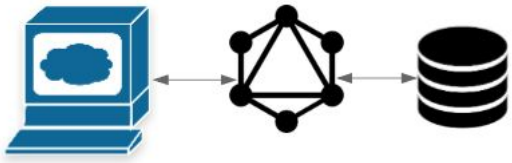
Options:

- Greenfield
- Gateway
- Wrapper
- Hybrid

How do we prove its value without throwing a lot of resources at it? How do we show how awesome it is from a business perspective?
Create a quick POC that can demonstrate the value

# POC: Greenfield Approach

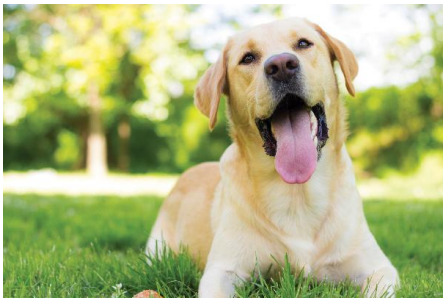
Start from scratch

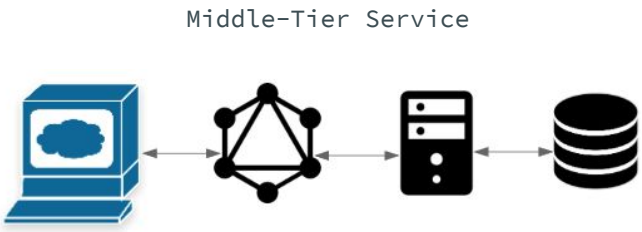


Starting over, creating a new GraphQL server that connects to a datasource.
It's time consuming, it can be hard to justify working on it

Level of Effort Dog Sizing Estimate: Great Dane

# POC: Gateway Approach



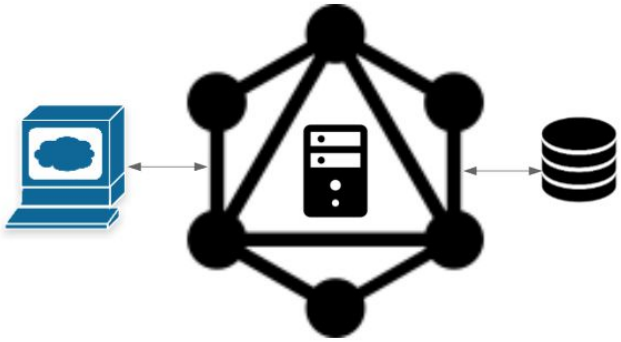



Keep existing APIs, but point the UI to a GraphQL middle-tier service in front of them. This requires modeling of data types to connect your GraphQL schema to the data returned by your APIs.
Tedious and time-consuming. Better if using DTO or adapter pattern

Level of Effort Dog Sizing Estimate: Lab, medium sized

# POC: Wrapper Approach

`OpenAPI Schema -> GraphQL Schema`

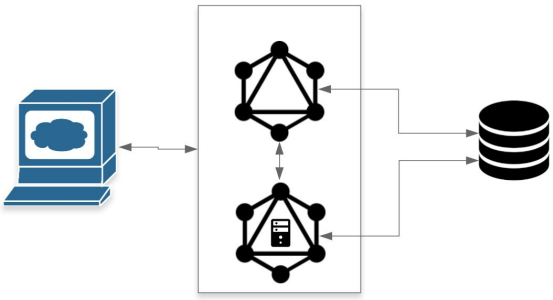


Taking the OAS schema from existing API and wrapping a GraphQL service around it. Data types are already defined by the OpenAPI Schema, just needs to be converted.

Level of Effort: Teacup Yorkie, tiny

# POC: Hybrid Approach

Gateway Approach + Wrapper Approach



Generate a GraphQL schema using OpenAPI schema and use it to build a Gateway service.
Take advantage of existing API while building a new GraphQL service along side it.

Dog Sizing Estimate: Beethoven. St Bernard, larger effort

# OpenAPI

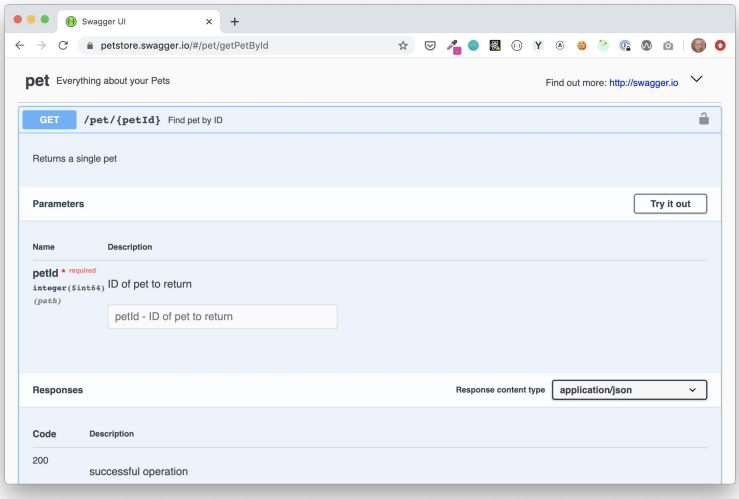
What is it?

Why?

- Documentation
- Rudimentary Postman

How?

- Docs in, code out
- Code in, docs out




- OpenAPI spec
- OAS
- Swagger

Terms are used interchangeably,
It's a way to describe the functionality/behavior of APIs in a standardized way.

**Why**

- Generates a UI for documentation, which is great for collaboration
- Able to try out requests against the working API

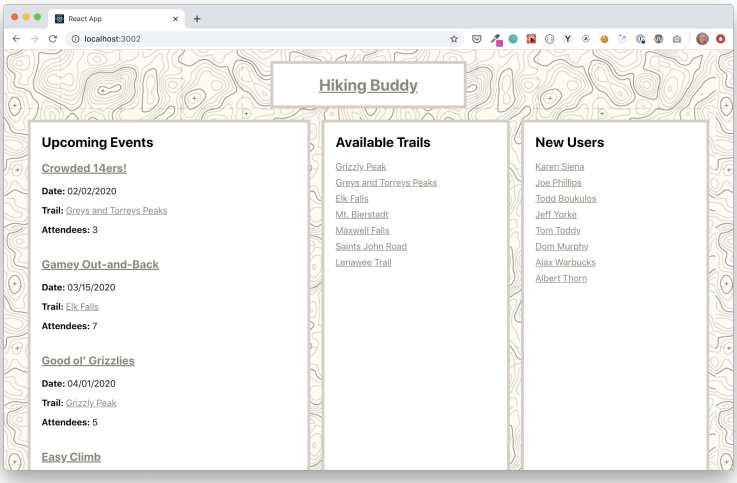
**How**

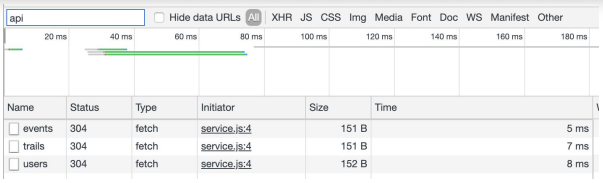
High-level:

- Docs in, code out: Top-down (design-first) approach
  - API is completely tied to the OAS schema, limited to what the OAS package can do.
- Code in, docs out: Bottom-up approach
  - Able to write pure express apps, but the docs must be maintained independently of the app.
  - Room for divergence, tread carefully

# Demo Time

Hiking Buddy app



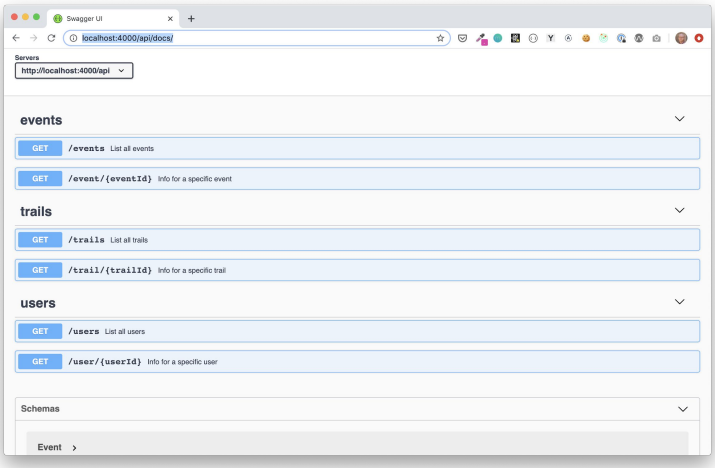




Created a demo app to show how we can use the Wrapper Approach to quickly generate a GraphQL schema using an OAS schema.
It has 3 API calls, which all return more data than I need.

# Demo Time

Swagger UI

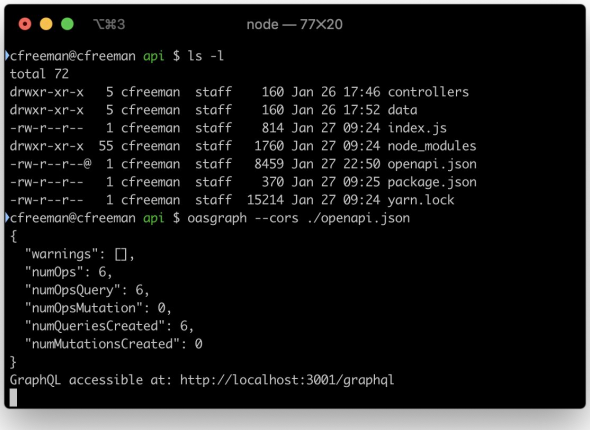
```
const [events, setEvents] = useState([]);
const [trails, setTrails] = useState([]);
const [users, setUsers] = useState([]);

useEffect(() => {
  const getDashboardData = async () => {
    const [
      eventsData,
      trailsData,
      usersData,
    ] = await Promise.all([
      getAllEvents(),
      getAllTrails(),
      getAllUsers(),
    ]);

    setEvents(eventsData);
    setTrails(trailsData);
    setUsers(usersData);
  };
  getDashboardData();
}, []);
```




The app connects up to an API using traditional REST methods and I created a Swagger UI for it using the Swagger Editor to manually create this, but ideally there'd be some tooling in place to generate them for me, maybe via JSDoc or a TypeScript decorator.

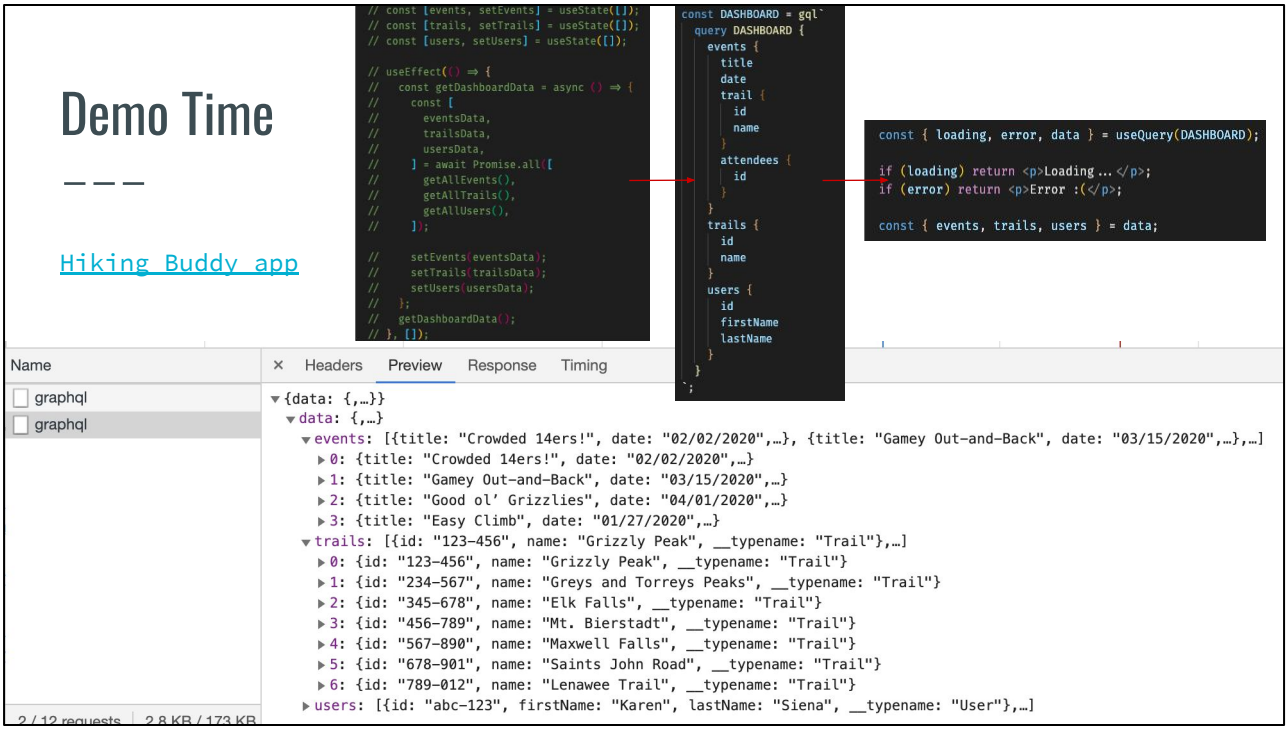
# Demo Time

Run openapi-to-graphql CLI




Create the GraphQL service

- Using IBM's openapi-to-graphql CLI to create a wrapper service. API already running.
  - *oasgraph --cors ./openapi.json*

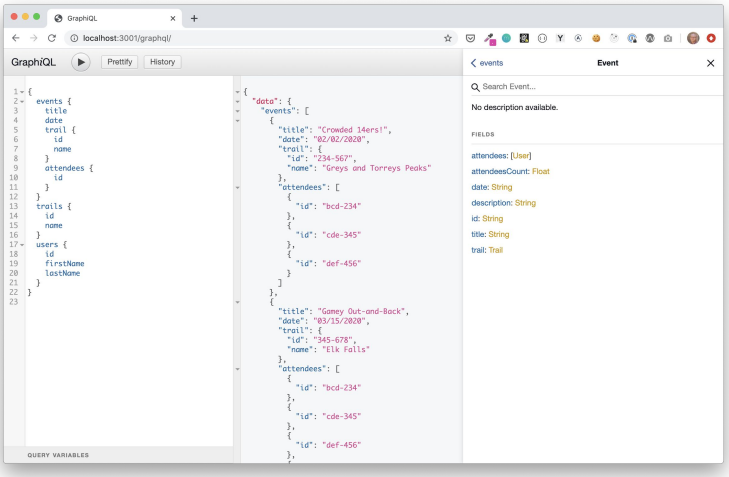



Swap out API calls for GraphQL a single query

- *Change port in package.json from 4000 to 3001*
- *Update code to use Apollo Client*

# Demo Time

GraphiQL from
openapi-to-graphql library




Out of the box we get GraphiQL tools that work against our existing API

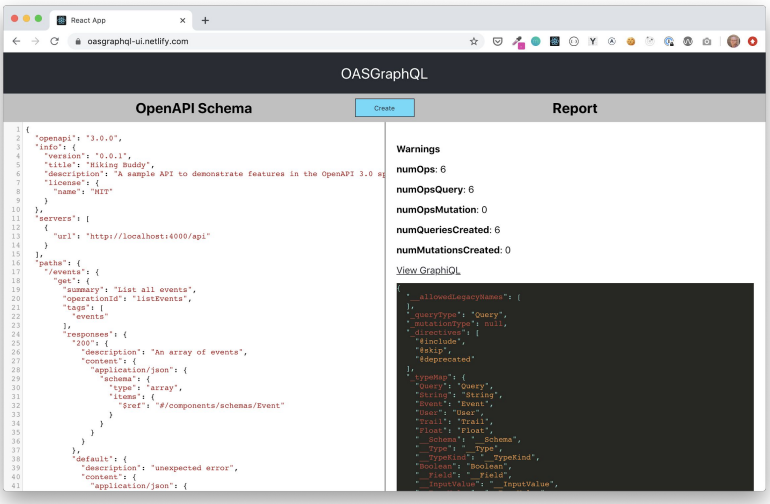
How long did that take?

# Extra Credit

OASGraphQL Converter app

Type definitions

```
type Author {
  id: Int!
  firstName: String
  lastName: String
  """
  the list of Posts by this author
  """
  posts: [Post]
}

type Post {
  id: Int!
  title: String
  author: Author
  votes: Int
}

# the schema allows the following query:
type Query {
  posts: [Post]
  author(id: Int!): Author
}

# this schema allows the following mutation:
type Mutation {
  upvotePost (
    postId: Int!
  ): Post
}
```




- Generate schema via app
  - Besides creating a working GraphQL app for you, the openapi-to-graphql library can convert an OAS doc into a GraphQL schema for you to use in your own GraphQL app.
  - Take generated OAS doc and convert it using a service I created that uses this library. It gives you some insight into the conversion and will report warnings if there are any. You may have to tweak the swagger doc a bit to get the schema working.
  - We also get a look into the GraphQL schema. You may be used to seeing the GraphQL type definitions (left), but tools like Apollo Server will parse that into an AST and then transform the AST into this (object in report on right), an object that can resolve GraphQL queries.
  - If we pipe this schema into our GraphQL tooling, we get instant documentation.
  - It's interesting, it helps you to better understand what's actually happening, but not really valuable in terms of selling the idea of GraphQL.

# Summary

- Used existing tools to generate a GraphQL service for our existing API
- Consolidated API requests
- Backend-for-Frontend
- Upgraded documentation


Let’s review

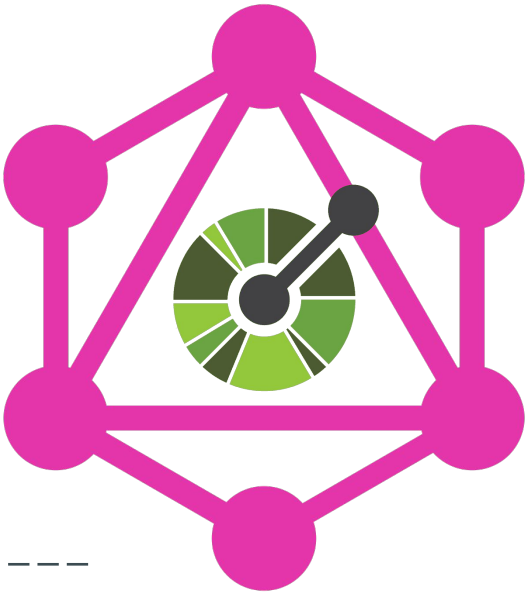
- Leveraged our existing tooling to create a brand new GraphQL service for our existing API.
- Consolidated our requests for data down to one endpoint. Minimized browser network requests and data processing. If we take advantage of the fragment caching in the Apollo Client, these are measurable performance wins - especially for a larger app
- BFF. Put the UI devs in the driver’s seat and given them the keys. No more Agilefall, no more API contract negotiating. People can just go build.
- Upgraded our documentation, BE/FE - all on the same page without having to do extra work.

I think this is a great way to very quickly prove the value of GraphQL and to get your coworkers excited about it

# Thank You

Craig Freeman

@craigfreeman
craigfreeman.net



# Resources

- IBM's openapi-to-graphql library
- OpenAPI
- Swagger Editor
- OASGraphQL Converter app
- OASGraphQL UI repo
- OASGraphQL API repo
- Adding Swagger to Existing Node.js Project
- Understanding the GraphQL AST
- OASGraph: Enabling Nested Queries
- Advanced GraphQL Patterns: Embrace the AST!
- Loopback 4 Node.js API Framework - Auto-generates OAS schema
- Nest Node.js API Framework - Decorators

Honorable mentions:

- LoopBack 4 Node API framework - TypeScript - generates swagger schema for free. You can extend it.
- Nest.js Node framework also has TypeScript decorators to help generate a swagger schema