

# GraphQL with React

•••

Daniel Zen  
@danielzen @zendigital  
training@zen.digital

© 2019 zen.digital

# My Background

- Systems Architect, Developer & Trainer
- Extensive Engineering & Agile/XP Background
  - MIT, Google, Pivotal Labs
- Started [AngularNYC](#) Meetup 2012
- Founder of [zen.digital](#)
  - Enterprise Training
  - Consulting & Best Practices
  - Team Augmentation
  - Full Stack JavaScript & Java
  - DevOps, CI/CD & Testing Practices
  - Mobile Solutions



# Tell me a little bit about yourself ...

- Are you all JavaScript developers?
  - <http://zen.digital/javascript/quiz>
- Any designers?
- Managers?
- What types of industry?
- Anything else you would like me to know?

# What is GraphQL

- A query language for APIs and a runtime for fulfilling those queries with your existing data
- Response to modern web application needs
- Alternative to REST
- Developed internally by Facebook in 2012
- Released publicly in 2015
- Specification
  - <https://graphql.github.io/graphql-spec/June2018/>
  - <https://github.com/graphql/graphql-spec>



Describe your data

```
type Project {  
  name: String  
  tagline: String  
  contributors: [User]  
}
```

Ask for what you want

```
{  
  project(name: "GraphQL") {  
    tagline  
  }  
}
```

Get predictable results

```
{  
  "project": {  
    "tagline": "A query language for APIs"  
  }  
}
```

Get Started

Learn More

# A query language for your API

# Who is using GraphQL

- AirBnb
- Coursera
- Facebook
- GitHub
- Intuit
- Meteor
- New York Times
- Pinterest
- Shopify
- Spotify
- Ticketmaster
- Twitter
- Yelp
- You?



SERVERLESS



The New York Times



intuit®



credit karma



amazon

Walmart



Product Hunt



glassdoor



METEOR

EXPRESS



NBC UNIVERSAL



# Why did Facebook create GraphQL

- Invented during the move from **HTML5** to **native**
- Problems encountered using REST
- Hierarchical - **Multiple round trips** may overload the network (mobile)
- REST Client depends on server - breaking changes
- GraphQL is strongly-typed
- Introspection



Explore

GraphQL

Tutorials

Guides

Case Studies

Enterprise

Conference →

GraphQL.org →

## All of the data you need, in one request

GraphQL is an open spec for a flexible API layer. Put GraphQL over your existing backends to build products faster than ever before.

[Start Exploring](#)

[Official Documentation →](#)



### Faster frontend development

Iterate quickly on apps without waiting for new backend endpoints. Simplify data fetching and management code by getting the data in the shape you need.

### Use your existing data

You can use GraphQL on top of your existing infrastructure: REST, SOAP, existing databases, or anything else. Organize your data into a clean, unified API and query it all at once.

### Fewer bytes and roundtrips

Make your apps more responsive than ever before by only loading the data you're actually using, and reduce the number of roundtrips to fetch all of the resources for a particular view.

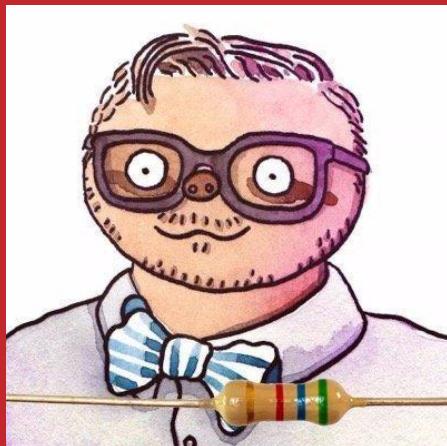
Used in production at



zen.digital

© 2019

# GraphQL creators



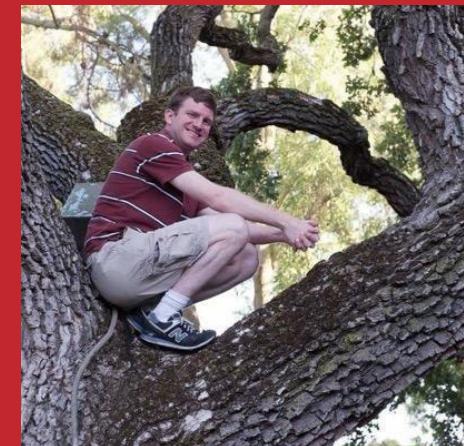
**Lee Byron**

[@leeb](https://twitter.com/leeb)



**Nick Schrock**

[@schrockn](https://twitter.com/schrockn)



**Dan Schafer**

[@dlschafer](https://twitter.com/dlschafer)

# GraphQL Timeline

- 2012 - GraphQL created at Facebook
- 2013 - React is released
- 2014 - React Native is released
- 2015 - GraphQL is open sourced
  - - Relay Classic is open sourced
- 2016 - GitHub announces GraphQL API
  - - New GraphQL website [graphql.org](https://graphql.org)
  - - First GraphQL Summit

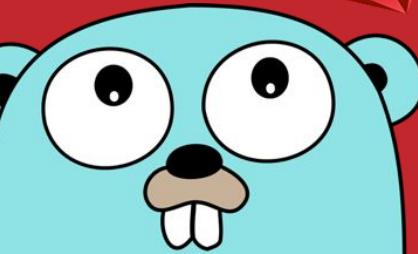
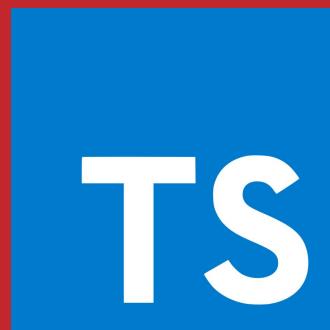
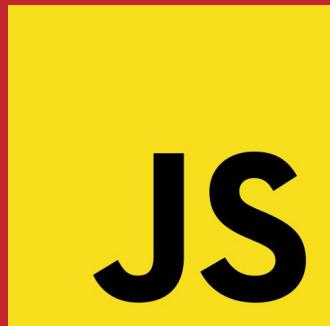
# GraphQL Timeline (cont)

- 2017 - Apollo Client 1.0
- - Relay Modern 1.0
- - Apollo Client 2.0
- 2018 - Prisma 1.0
- - Apollo Engine
- 2019 - Apollo Client 2.5

# GraphQL Backend Servers

- Node middleware libraries
  - Express
  - Hapi
  - Koa
- GraphQL Baas
  - Graphcool / Prisma
  - Scaphold
  - Hasura
- PostGraphile (for PostgreSQL)
- [json-graphql-server](#)

# Implementations



elixir



# Query Language ⚡

```
// GET '/graphql'          // result
allTodos(first: 1) {
  complete
  text
}
• Query has same shape as result
• Check out the following page
• https://graphql.github.io/learn/
}
{
  "data": {
    "allTodos": [
      {
        "complete": true,
        "text": "Learn GraphQL"
      }
    ]
  }
}
```

# Common Persistence Practices

- CRUD - Create, read, update and delete

Each letter in the acronym can map to a standard SQL statement and HTTP method:

Operation`	SQL	HTTP (REST)
Create	INSERT	PUT / POST
Read (Retrieve)	SELECT	GET
Update (Modify)	UPDATE	PUT / PATCH
Delete (Destroy)	DELETE	DELETE

# Typically we employed CRUD via RESTful APIs

- RESTful APIs - Representational state transfer

The name “Representational State Transfer” is intended to evoke an image of how a well-designed Web application behaves: a network of Web pages forms a virtual state machine, allowing a user to progress through the application by selecting a link or submitting a short data-entry form, with each action resulting in a transition to the next state of the application by transferring a representation of that state to the user. - Roy Fielding

- HTTP 1.1 based on REST

# Common REST Practices

Resource	POST create`	GET read	PUT update	DELETE delete
/accounts	Create a new account	List accounts	Bulk update accounts	Delete all accounts
/accounts/123	Error	Show account 123	If exists update account 123  If not error	Delete account 123
/customers	Create a new customer	List customers	Bulk update customers	Delete all customers
/customers/456	Error	Show customer 456	If exists update customer 456  If not error	Delete customer 456

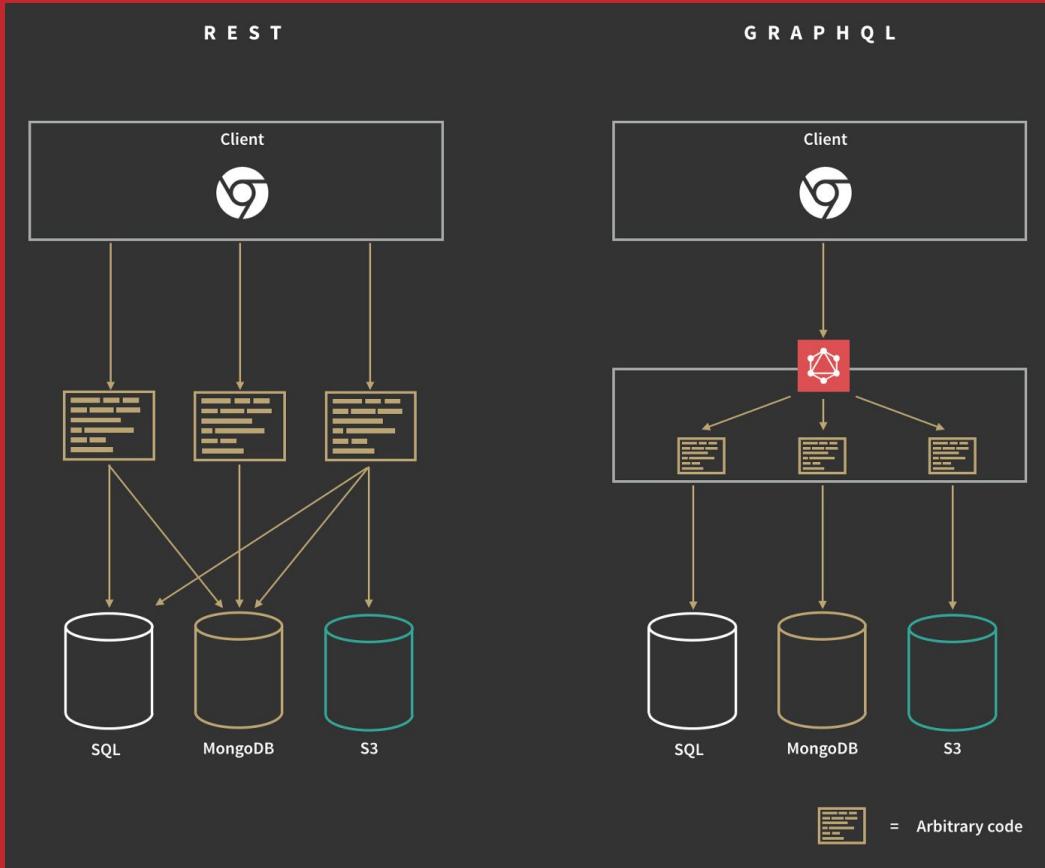
# REST isn't a fit for modern applications

The point-to-point nature of REST, a procedural API technology, forces the authors of services and clients to coordinate each use case ahead of time. When frontend teams must constantly ask backend teams for new endpoints, often with each new screen in an app, development is dramatically slowed down. Both teams need to move fast independently.

# A data graph at the center of your architecture

GraphQL decouples apps from services by introducing a flexible query language. Instead of a custom API for each screen, app developers describe the data they need, service developers describe what they can supply, and GraphQL automatically matches the two together. Teams ship faster across more platforms, with new levels of visibility and control over how their data is used.

# Overview:



GraphQL replaces REST “best practices” with a clear & simple structure

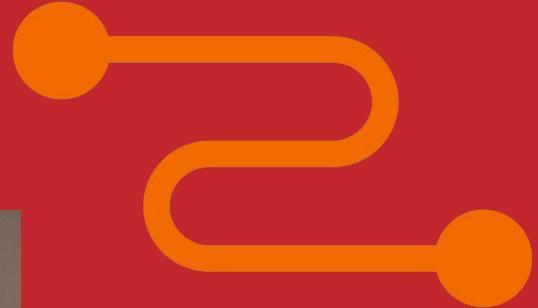
# Why use GraphQL

# Some reasons

- Declarative
- Decoupled from storage
- Validated and structured
- Facilitates Collaboration
- Super fast

# GraphQL Clients

- [Apollo Client](#)
- [Relay Modern](#)
- [urql](#)



# Tools

# GraphQL Playground (server GUI from Prisma)

- All GraphiQL features (original GUI from GraphQL/FB)
  - Standalone App available
- Multi-server (endpoint per tab)
- Prettify, History, HTTP Headers
- Supports real-time subscriptions and tracing (Apollo)
- Standalone App

# Apollo Client Developer Tools

 **Apollo Client Developer Tools**  
offered by [Apollo Team](#)

★★★★★ (33) | [Developer Tools](#) | 15,614 users

[OVERVIEW](#) [REVIEWS](#) [RELATED](#) [G+](#)

All queries Requests: 1.9. Frequency: 379/sec Average Duration: 94 msec

**Schema**: App -> metrics: [Metric], artifactsCount: Int

**Queries**: Duration: Total Time: 1.24m

**GraphQL**: [Pretty](#) [Load from cache](#)

```
1 > query queryMyServices($envServiceId: Int!, $clientName: String!) { 2   "data": { 3     "id": "service", { 4       "name": "Service", 5       "report": { 6         "from": "ServiceReport", 7         "aggregated": { 8           "name": "MyReport", 9           "queries": { 10             "count": 99, 11             "query": "query myReport { 12               service{id, name} 13             } 14           } 15         } 16       } 17     } 18   } 19 }
```

**Query Variables**:

```
1 > { 2   "envServiceId": "acme-production", 3   "clientName": null, 4   "timeframe": "30d", 5 }
```

**Documentation Explorer**

**ROOT TYPES**: Query, Mutation, Subscription

**Documentation**: [Website](#) [Report Abuse](#)

**Additional Information**: Version: 2.0.5, Updated: October 23, 2017, Size: 838KB, Language: English (United States)

**RELATED**

 Xdebug helper ★★★★★ (405)	 ColorPick Eyedropper ★★★★★ (1023)	 Page Ruler ★★★★★ (1734)	 ColorZilla ★★★★★ (2233)
---	---	---	--

# Newish Dev Tools Stuff (over a year old now)

- Newer Architecture
- Stability and performance
- Apollo Client 2.x (2.5)
  - Apollo Link
  - Apollo Link State

# Don't Versioning - Evolve your API

While there's nothing that prevents a GraphQL service from being versioned just like any other REST API, GraphQL takes a strong opinion on avoiding versioning by providing the tools for the continuous evolution of a GraphQL schema.

Why do most APIs version? When there's limited control over the data that's returned from an API endpoint, any change can be considered a breaking change, and breaking changes require a new version. If adding new features to an API requires a new version, then a tradeoff emerges between releasing often and having many incremental versions versus the understandability and maintainability of the API.

In contrast, GraphQL only returns the data that's explicitly requested, so new capabilities can be added via new types and new fields on those types without creating a breaking change. This has led to a common practice of always avoiding breaking changes and serving a versionless API.

# Subscription Support

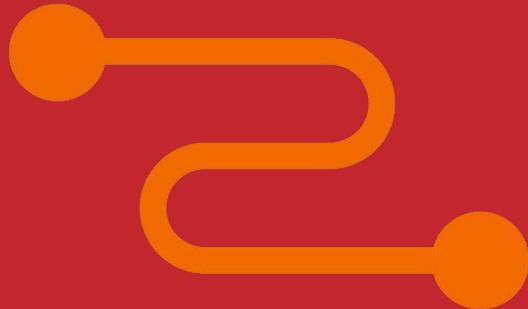
The screenshot shows a GitHub repository page for the 'facebook/react' repository. At the top, there are navigation links for 'GitHunt', 'Top', 'Hot', and 'New'. On the right, there are buttons for '+ Submit', 'jbaxleyiii', and 'Log out'. Below the header, the repository name 'facebook/react' is displayed, along with a description: 'A declarative, efficient, and flexible JavaScript library for building user interfaces.' Below the description, it shows 'Stars: 87356' and 'Issues: 369', with a note that it was 'Submitted 23 hours ago by Slava'. A text input field for comments is present, followed by a 'Submit' button. A recent comment from 'jbaxleyiii' is shown, followed by a link to 'what is wrong?'. At the bottom, there's a note about it being an Apollo example app. The bottom half of the screenshot shows the Apollo GraphQL interface. It has tabs for 'Elements', 'Console', 'Sources', 'Network', 'Performance', 'Memory', 'Application', 'Security', 'Audits', 'React', and 'Apollo'. The 'Apollo' tab is selected. On the left, there's a sidebar with icons for 'GraphiQL', 'Queries', 'Mutations', and 'Cache'. The main area shows a GraphQL query:

```
1 v fragment CommentsPageComment on Comment {  
2   id  
3   postedBy {  
4     login  
5     html_url  
6   }  
7   createdAt  
8   content  
9 }  
10 v subscription onCommentAdded($repoFullName: String!) {  
11   commentAdded(repoFullName: $repoFullName) {  
12     ...CommentsPageComment  
13   }  
14 }
```

Below the query, the response is shown:

```
v {  
v   "data": {  
v     "commentAdded": {  
v       "id": 16,  
v       "postedBy": {  
v         "login": "jbaxleyiii",  
v         "html_url": "https://github.com/jbaxleyiii"  
v       },  
v       "createdAt": 1517492286572,  
v       "content": "data"  
v     }  
v   }  
v }
```

# Relay Modern



# Relay Modern

Relay is Facebook's homegrown GraphQL client that they open-sourced alongside GraphQL in 2015. It incorporates all the learnings that Facebook gathered since they started using GraphQL in 2012. Relay is heavily optimized for performance and tries to reduce network traffic where possible. An interesting side-note is that Relay itself actually started out as a routing framework that eventually got combined with data loading responsibilities. It thus evolved into a powerful data management solution that can be used in Javascript apps to interface with GraphQL APIs.

# Relay Modern Overview

- Maintained by **Facebook**
- Latest release: **v5**
- View-agnostic runtime
- Integrations: **React**
- GitHub: Over **13K** stars
- MIT license

# Relay Main Features

- Focused on Performance
  - Relay compiler/babel plugin
  - Optimisations
- Facebook abstractions
  - Colocation
  - Custom Store
    - Connection, Records, Fields
  - Viewer, Pagination

# Apollo Client



# Apollo Client

[Apollo Client](#) is a community-driven effort to build an easy-to-understand, flexible and powerful GraphQL client. Apollo has the ambition to build one library for every major development platform that people use to build web and mobile applications. Right now there is a Javascript client with bindings for popular frameworks like [React](#), [Angular](#), [Ember](#) or [Vue](#) as well as early versions of [iOS](#) and [Android](#) clients. Apollo is production-ready and has handy features like caching, optimistic UI, subscription support and many more.

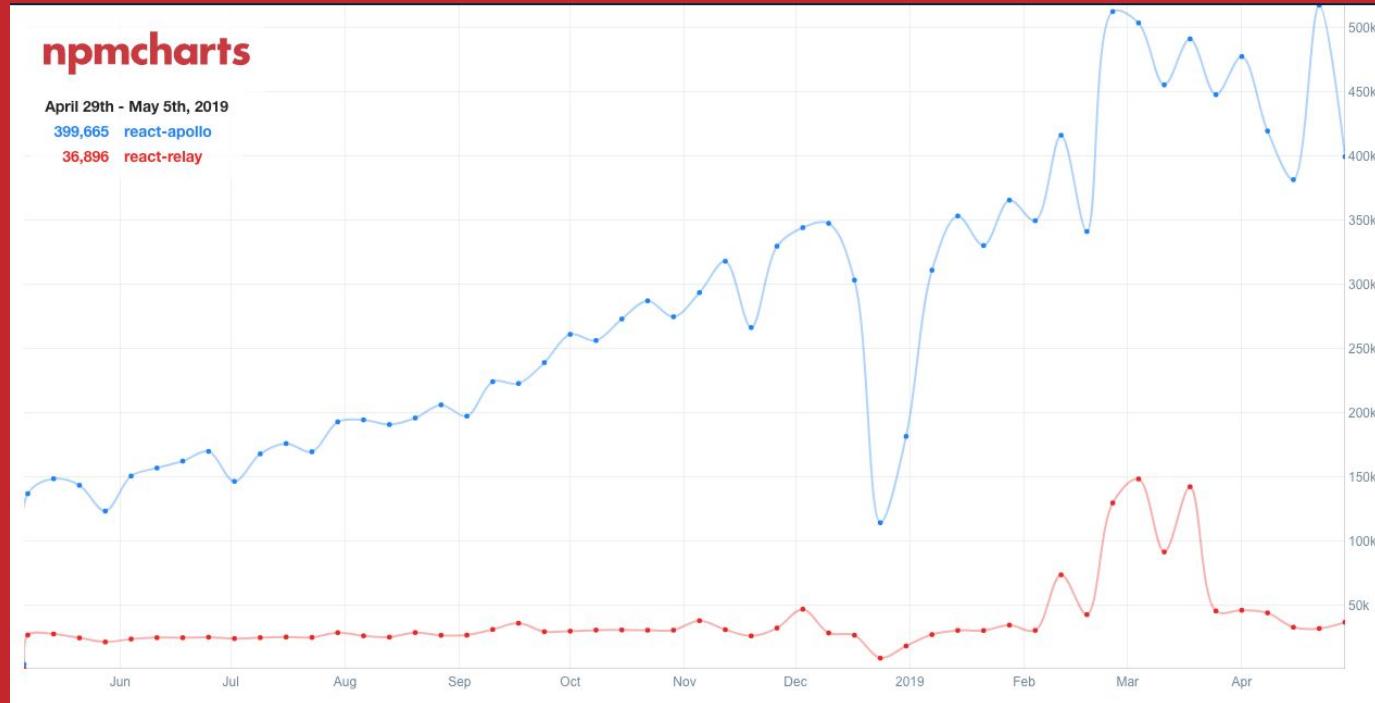
# Apollo Client Overview

- Maintained by **Meteor**
- Latest release: **2.6.3**
- Integrations: **React, Angular, Vue**
- GitHub: Over **11.5K stars**
- MIT license

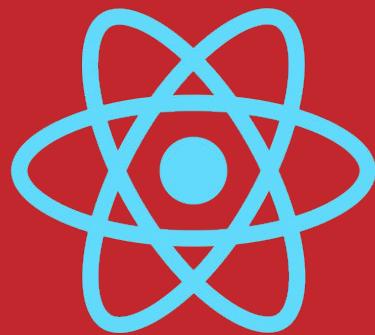
# Apollo Client Features

- Community driven
- Simplicity
- Integration with multiple frameworks

# Relay vs Apollo Downloads



CLIENT



React



Relay  
Modern



Apollo  
Client

SERVER



GraphCool

CLIENT



Angular

SERVER



Apollo  
Client



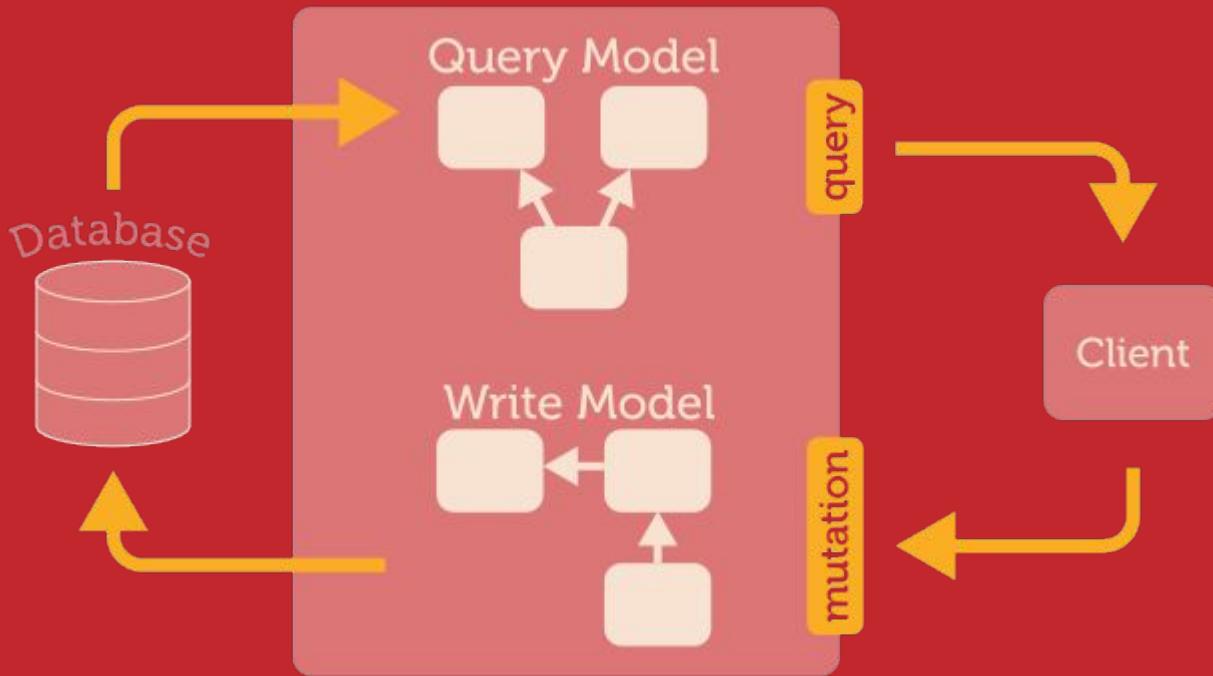
GraphCool

# Setup



- Relay compiler/  
Babel plugin
- Environment
  - Network
  - Store
- Via attribute
- Loader graphql-tag
- ApolloClient
  - Apollo Link
  - Apollo Cache
- ApolloProvider HOC

# Queries



source: [gist](#)

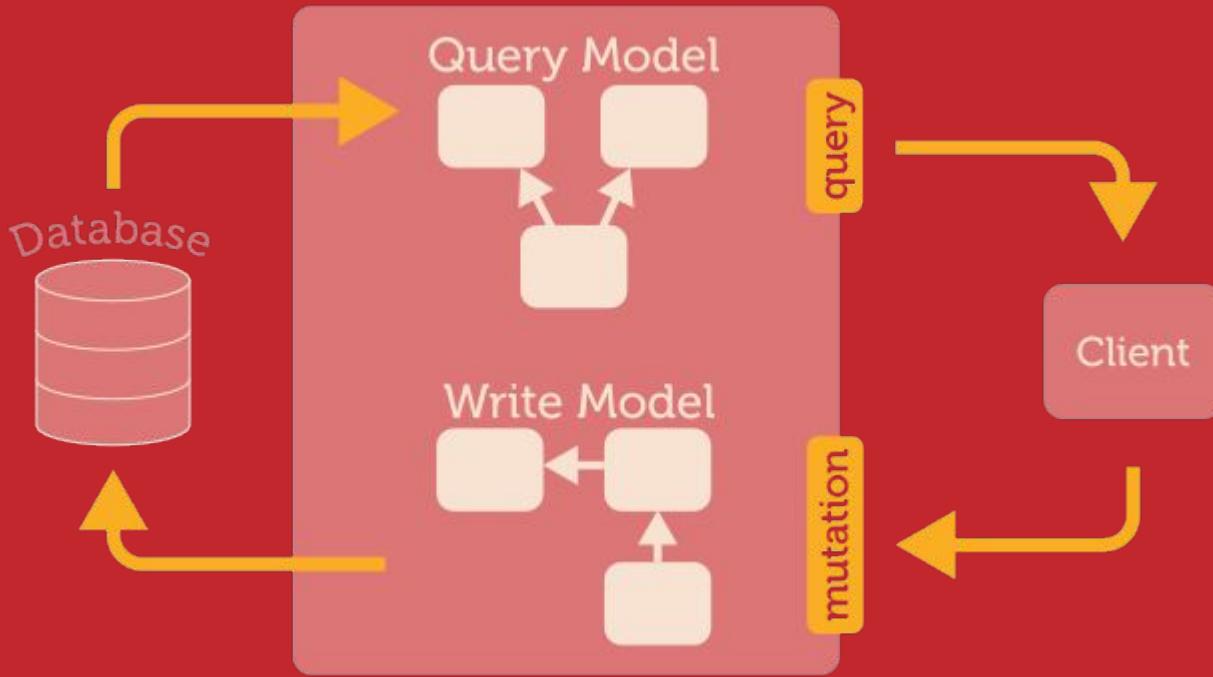
# Queries



- QueryRenderer
    - Environment
    - Query
  - Queries generation
    - Automated
  - Opinionated
- 
- graphql HOC
  - Queries generation
    - Manual
  - Flexible

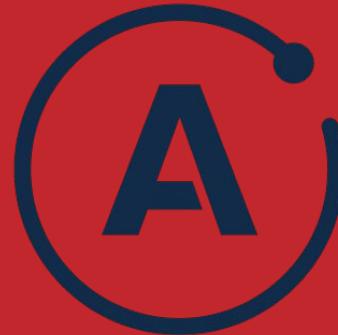


# Mutations



source: [gist](#)

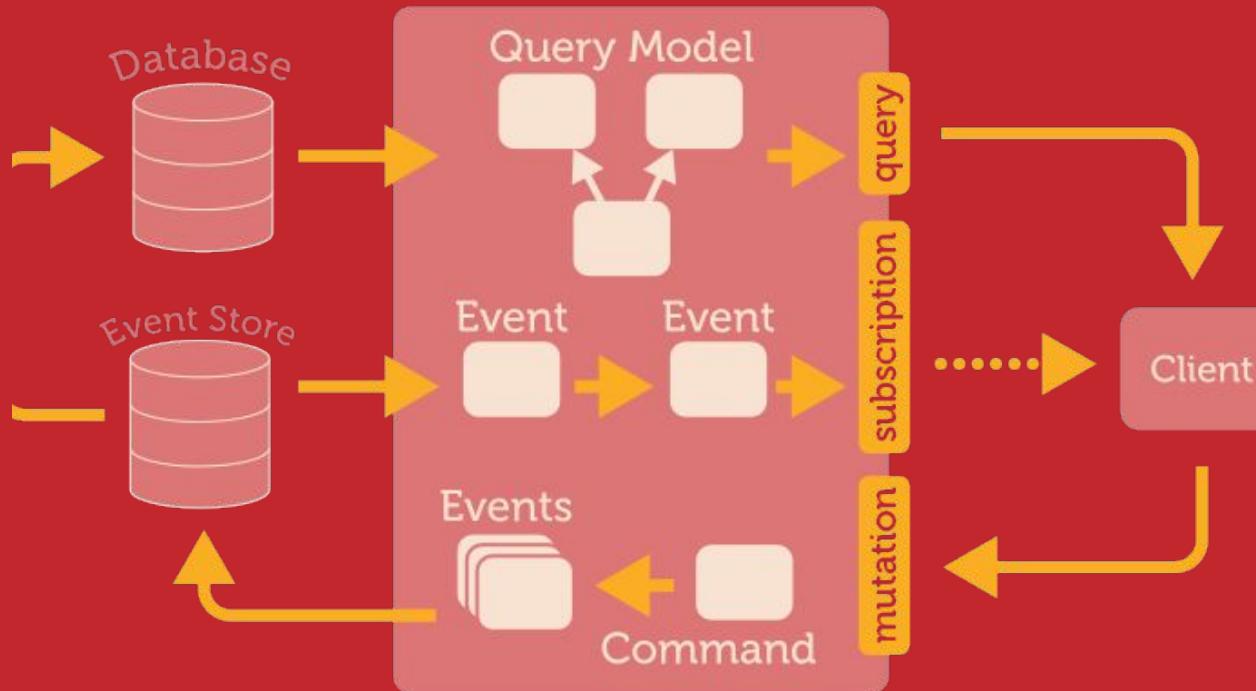
# Mutations



- commitMutation
  - Environment
- Update Store via
  - Connection
  - Fields/LinkedRecord
- Optimistic Updates
- Added via props
- graphql HOC/mutate
- Update Store via Cache/arguments
- Optimistic Updates

Apollo considered better at mutations

# Subscriptions



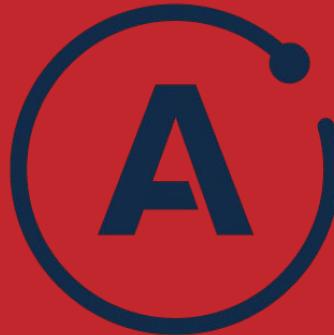
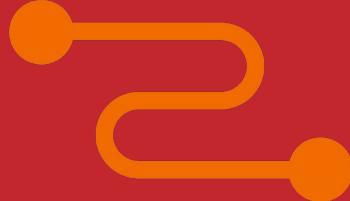
source: [gist](#)

# Subscriptions



- `requestSubscription`
- Environment
- Update Store via
  - Connection
  - Fields/LinkedRecord
- `subscribeToMore`
- Update Store via reducer/arguments

# Summary



- Awesome option
- Familiar with Facebook-way
  - Viewer/Connector
  - Pagination
- Schema with lots of fragments

- Awesome option
- May need other Frameworks
- Modular design
- Less concepts to learn

# GraphQL service

- Define types (User) and fields on types (id, name, dateOfBirth)

```
type User @model {  
    id: ID! @isUnique  
    name: String  
    dateOfBirth: DateTime  
}
```

@model are top-level entities in the generated API

@isUnique are unique elements

ID! The ! means this is a required/non-nullable field

# GraphQL Schema

# Schema Overview

- Entry points
  - Query
  - Mutation
  - Subscription
- Features
  - Validation, Documentation and Introspection

# Type System

- Scalar Types:

- Int
- Float
- String
- Boolean
- ID

- Object Types:

- Post

# Schema Syntax

- Mandatory: `String!`, `Post!`
- Optional: `String`, `Post`
- Lists: `[String]`, `[Post]`

# GraphQL Schema for ToDo app

```
schema {  
  query: Query  
  mutation: Mutation  
  subscription: Subscription  
}
```

```
type Query {  
  allTodos(first: Int): [Todo]  
}
```

```
type Todo {  
  id: ID!  
  text: String!  
  complete: Boolean!  
}
```

Playground

```
allTodos(last: 2) {  
  id, text, complete  
}  
}
```

with alias:

```
todos: allTodos(last: 2) {  
  id, text, complete  
}  
}
```



# [apis-guru/graphql-voyager](https://apis-guru/graphql-voyager)

 **GRAPHQL  
VOYAGER**

**CHANGE INTROSPECTION**

**Type List**

**Root** `root` ⓘ  
No Description

**Film** ⓘ  
A single film.

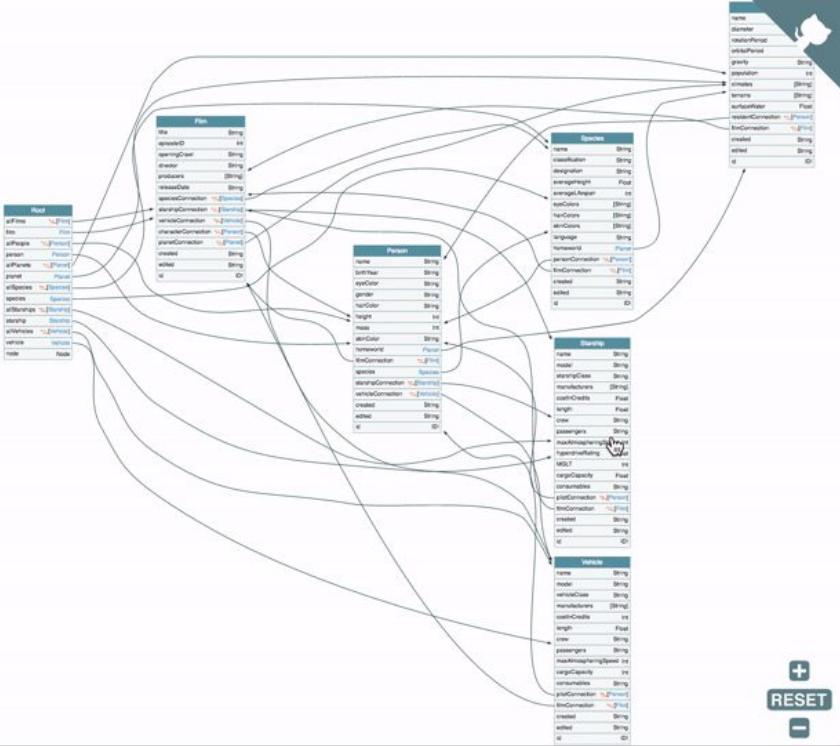
**Person** ⓘ  
An individual person or character within the Star Wars universe.

**Planet** ⓘ  
A large mass, planet or planetoid in the Star Wars Universe, at the time of 0 ABY.

**Species** ⓘ  
A type of person or character within the Star Wars Universe.

**Starship** ⓘ  
A single transport craft that has hyperdrive capability.

**Vehicle** ⓘ  
A single transport craft that does not have hyperdrive capability.



+  
RESET  
-



zen.digital

© 2019

# Download and install a local copy of swapi-graphql

- The Star Wars API was often used for REST examples
- The GraphQL version can be run locally
- It has the GraphiQL UI built in (not the Playground):

```
{ allFilms {  
  films {  
    title  
  }  
}}
```

```
git clone https://github.com/graphql/swapi-graphql.git  
cd swapi-graphql  
npm start
```

# Query Syntax

# Quick ApolloClient Lab

Let's go over the Get started for Apollo at:

- <https://www.apollographql.com/docs/react/essentials/get-started/>

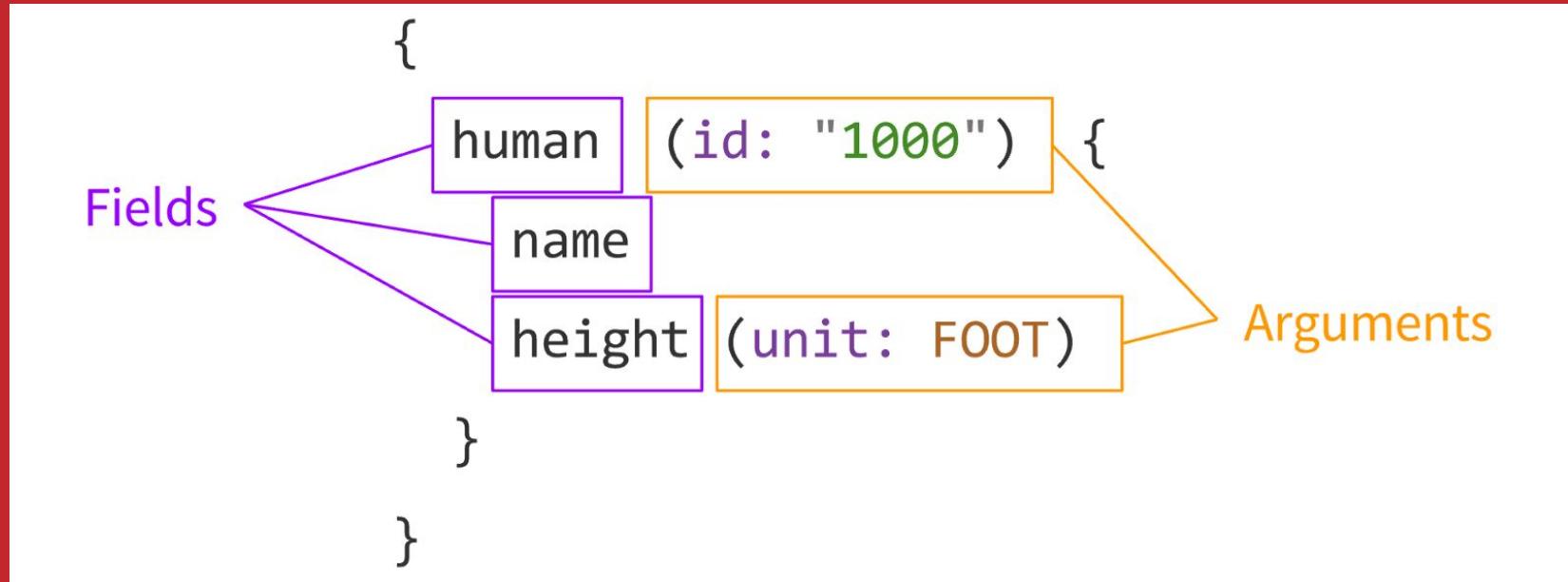
This is a very quick lab that will show you how little it takes to get up and running with ApolloClient.

We will be connecting to an Exchange Rate GraphQL service at:

- <https://48p1r2roz4.sse.codesandbox.io>

```
{  
  rates(currency: "USD") {  
    currency  
  }  
}
```

# Anonymous Query



Example from: <https://graphql.org/learn/queries/>

# Named Query

Operation type

Operation name

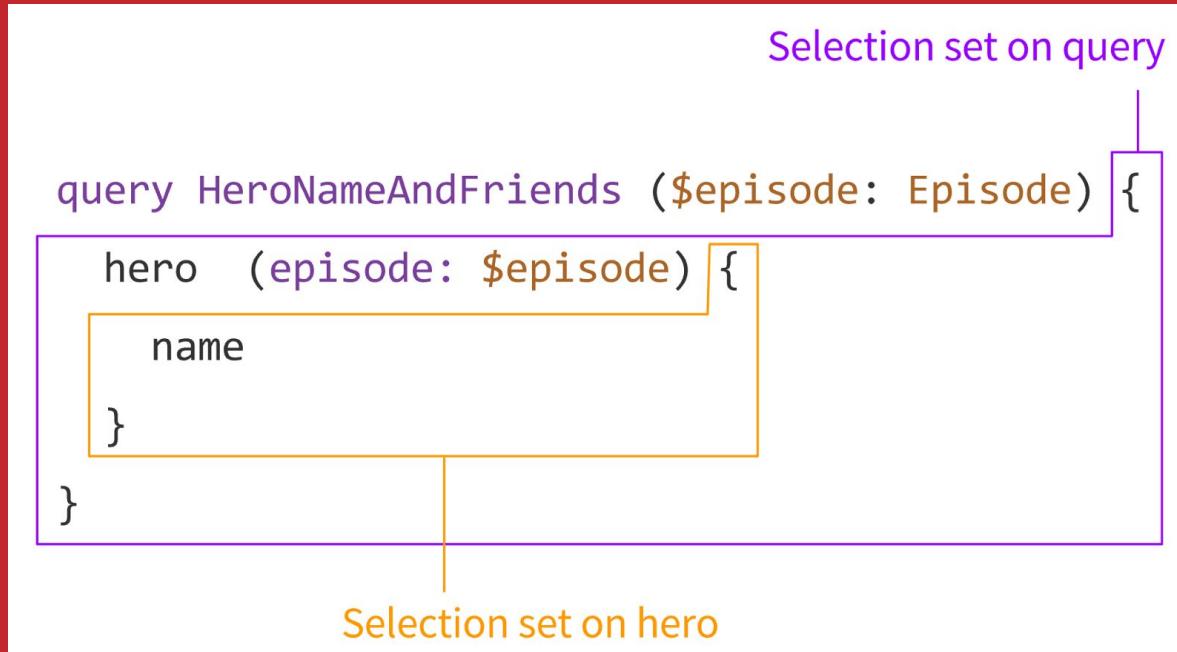
Variable definitions

```
query HeroNameAndFriends ($episode: Episode) {  
  hero(episode: $episode) {  
    name  
  }  
}
```

Query Variables:

```
{  
  "episode" : "JEDI"  
}
```

# Selection Sets



# Fragments

```
Fragment name          Type condition  
|  
fragment comparisonFields on Character {  
    name  
    appearsIn  
    friends {  
        name  
    }  
}
```

Selection set



zen.digital

© 2019

# Using Fragments

```
query HeroForEpisode {  
  hero {  
    name  
    ...comparisonFields  
    ... on Droid {  
      primaryFunction  
    }  
  }  
}
```

The diagram illustrates the structure of a GraphQL query. A main block contains a query for 'HeroForEpisode'. Inside, there's a 'hero' field with a 'name' field and two fragments. The first fragment, '...comparisonFields', is labeled 'Fragment spread'. The second fragment, '... on Droid { primaryFunction }', is labeled 'Inline fragment'. A vertical line from the closing brace of the inline fragment points down to the closing brace of the main 'hero' block, labeled 'Type condition'.

Fragment spread

Inline fragment

Type condition

# Directives

```
query HeroForEpisode($var: Boolean!) {  
  hero @skip(if: $var) {           # on a field  
    name  
    ...comparisonFields @include(if: $var) # on a fragment spread  
    ... on Droid @skip(if: $var) {        # on an inline fragment  
      primaryFunction  
    }  
    ... @include(if: $var) {           # on an inline fragment  
      name                         # with no type condition  
    }  
  }  
}
```

Directive arguments

The diagram illustrates the structure of GraphQL directives. A purple bracket labeled "Directive" points to the first `@skip` directive in the query. Another purple bracket labeled "Directive arguments" points to the `if: $var` argument of the second `@include` directive, which is located within an inline fragment. The code uses `$var` as a variable to conditionally skip or include fields based on its value.

# Exercises



# Graphcool Lab

# Todo App

# Todo Schema

project.graphcool

```
schema {  
  query: Query  
  mutation: Mutation  
  subscription: Subscription  
}
```

```
type Query {  
  allTodos(first: Int): [Todo]  
}
```

```
type Todo @model {  
  id: ID! @isUnique  
  text: String!  
  complete: Boolean!  
}
```

demo

# Apollo Client



# Apollo Cache

# Apollo Cache Packages

- apollo-cache-inmemory
- apollo-cache-hermes
- apollo-cache-persist

# InMemoryCache

- Normalised and flattened
- `this.props.client.cache`
- `readQuery`, `writeQuery`
- `readFragment`, `writeFragment`

# Apollo Cache Example

*Query*

```
query todos {  
  allTodos {  
    id  
    text  
    complete  
    __typename  
  }  
}
```

*Result*

```
{  
  "data": {  
    "allTodos": [  
      {  
        "id": "1",  
        "text": "Learn GraphQL 📚",  
        "complete": false,  
        "__typename": 'Todo',  
      },  
      {  
        "id": "2",  
        "text": "Learn Apollo Client ✨🚀",  
        "complete": true,  
        "__typename": 'Todo',  
      }]  
    }  
}
```



zen.digital

© 2019

# Normalised and flattened

typename:id

```
{  
  ROOT_QUERY: {  
    allTodos: [ 'Todo:1', 'Todo:2' ],  
  },  
  'Todo:1': {  
    id: '1',  
    text: 'Learn GraphQL 📚',  
    completed: false  
  },  
  'Todo:2': {  
    id: '2',  
    text: 'Learn Apollo Client ✨🚀',  
    completed: false  
  }  
}
```

*read/writeQuery*

*read/writeFragment*



zen.digital

© 2019

# Retrieving todos from cache

```
const query = gql`query AllTodos {
  allTodos { id text complete }
}`;
const data = store.readQuery({ query });
const newTodo = {
  id: createTodo.id,
  text: input.value,
  complete: false,
  __typename: "Todo"
};
store.writeQuery({
  query,
  data: {
    allTodos: [...data.allTodos, newTodo],
  },
});
```

# Set security headers

```
import ApolloClient from 'apollo-boost';

const client = new ApolloClient({
  uri: 'https://your-server/graphql',
  request: operation => {
    operation.setContext(context => ({
      headers: {
        ...context.headers,
        authorization: localStorage.getItem('authToken')
      }
    }))
  }
});
```

# GraphQL Server



# [apollographql/graphql-tools](#)

## GraphQL-tools: generate and mock GraphQL.js schemas

[npm package](#) [build](#) [coverage](#) [slack](#)

This package provides a few useful ways to create a GraphQL schema:

1. Use the GraphQL schema language to [generate a schema](#) with full support for resolvers, interfaces, unions, and custom scalars. The schema produced is completely compatible with [GraphQL.js](#).
2. [Mock your GraphQL API](#) with fine-grained per-type mocking
3. Automatically [stitch multiple schemas together](#) into one larger API

## Documentation

[Read the docs.](#)

## Binding to HTTP

If you want to bind your JavaScript GraphQL schema to an HTTP server, we recommend using [Apollo Server](#), which



**zen.digital**

© 2019

# graphql-tools

- Runtime Schema:
  - resolvers
  - interfaces
  - unions
  - custom scalars
- Allows mocking types

# Schema

```
const typeDefs = `

  type Post {
    title: String
  }
  type Query {
    posts: [Post]
  }
  type Mutation {
    upvotePost (postId: ID!): Post
  }

schema {
  query: Query
  mutation: Mutation
}`;
```

# Resolvers

```
const resolvers = {  
  
  Query: {  
    posts() {  
      return posts;  
    },  
  },  
  
  Mutation: {  
    upvotePost(_, { postId }) {  
      const post = find(posts, { id: postId });  
      post.votes += 1;  
      return post;  
    },  
  },  
};
```

# makeExecutableSchema

```
import { makeExecutableSchema } from 'graphql-tools';

const executableSchema = makeExecutableSchema({
  typeDefs,
  resolvers,
});
```

# launchpad.graphql.com - Still works for now

The screenshot shows a web browser window for `launchpad.graphql.com`. On the left, there's a code editor with a green header bar containing 'Launchpad' and navigation buttons. The code editor displays a GraphQL schema and resolver code:

```
1 // Welcome to Launchpad!
2 // Log in to edit and save pads, run queries in GraphiQL on the right.
3 // Click "Download" above to get a zip with a standalone Node.js server.
4
5 // graphql-tools combines a schema string with resolvers.
6 import { makeExecutableSchema } from 'graphql-tools';
7
8 // Construct a schema, using GraphQL schema language
9 const typeDefs = `
10   type Query {
11     hello: String
12   }
13 `;
14
15 // Provide resolver functions for your schema fields
16 const resolvers = {
17   Query: {
18     hello: (root, args, context) => {
19       return 'Hello world!';
20     },
21   },
22 };
23
24 // Required: Export the GraphQL.js schema object as "schema"
25 export const schema = makeExecutableSchema({
26   typeDefs,
```

To the right of the code editor is a GraphiQL playground with tabs for 'GraphiQL' and 'Prettify'. It includes buttons for 'Save', 'Reset', and 'Headers'. Below the playground is a 'QUERY VARIABLES' section with a single variable '1'. At the bottom of the page, there are links for 'Secrets', 'npm Deps', 'Logs', 'GraphQL Endpoint' (set to `https://7v7lqrjxj.lp.gql.zone/graphql`), 'Docs', and 'Powered by Auth0 Extend'.

# Check out Launchpad Running at:

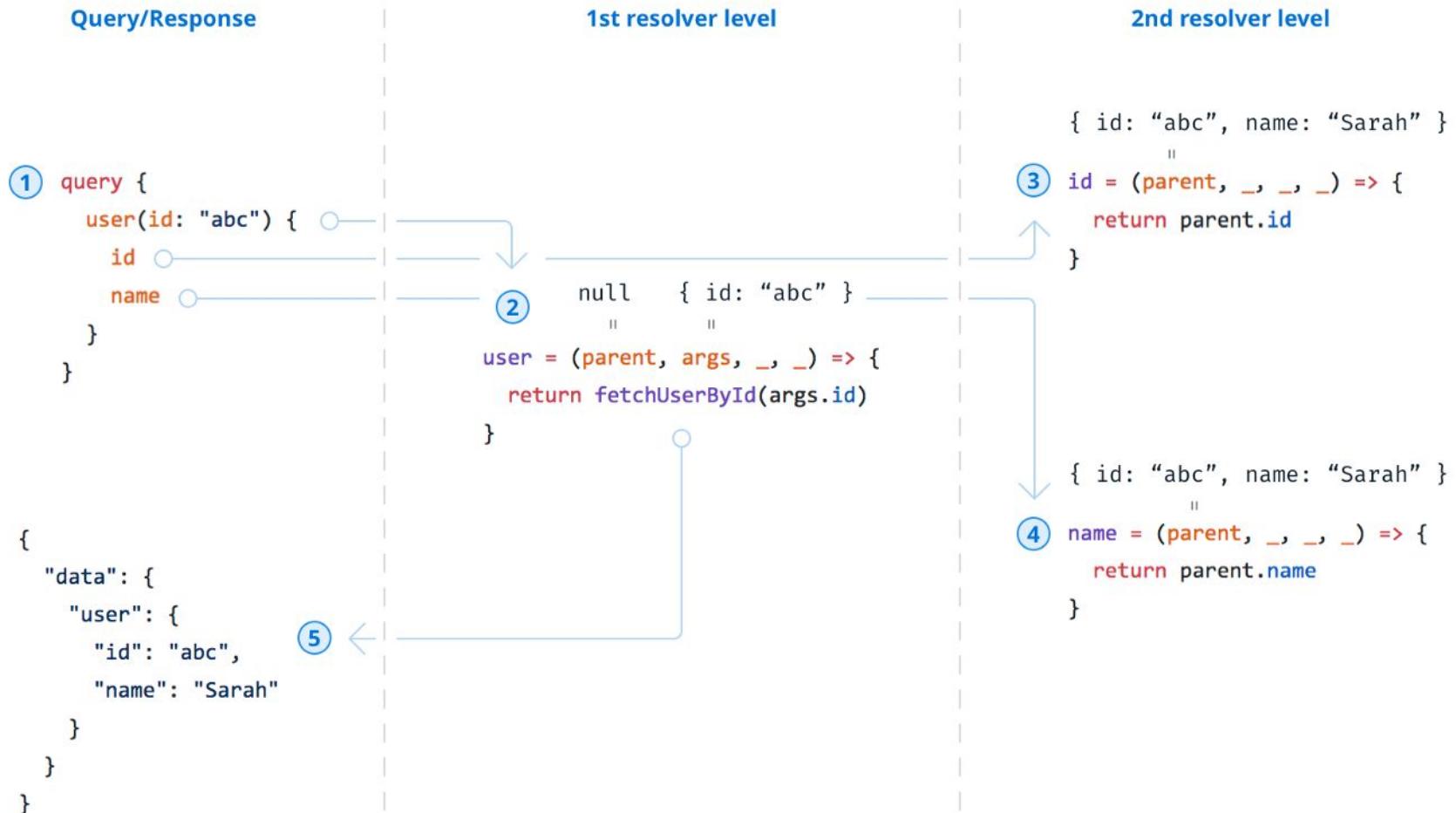
- <https://codesandbox.io/s/apollo-server>
- <https://glitch.com/~apollo-launchpad>

# Resolvers

- Each field in a GraphQL schema is backed by a resolver
- Resolver parameters
  - root/parent, result previous call
  - args, query parameters
  - context, share information
  - info, query AST

# Resolvers Example

```
const resolvers = {
  Query: {
    feed(parent, args, ctx, info) {
      return ctx.db.query.posts({ where: { isPublished: true } }, info)
    },
    drafts(parent, args, ctx, info) {
      return ctx.db.query.posts({ where: { isPublished: false } }, info)
    },
    post(parent, { id }, ctx, info) {
      return ctx.db.query.post({ where: { id } }, info)
    },
  },
}
```



# Query Execution

# GraphQL Tooling



# prisma/graphql-config

## graphql-config

The README reflects new [graphql-config protocol](#). Old graphql-config-parser documentation [can be found here](#)

The easiest way to configure your development environment with your GraphQL schema (supported by most tools, editors & IDEs)

### Supported by...

#### Language Services

- [graphql-language-service](#) - An interface for building GraphQL language services for IDEs (*pending*)

#### Editors

- [js-graphql-intellij-plugin](#) - GraphQL language support for IntelliJ IDEA and WebStorm, including Relay.QL tagged templates in JavaScript and TypeScript (*pending*)
- [atom-language-graphql](#) - GraphQL support for Atom text editor (*pending*)

#### Tools

# GraphQL Config

- Built by Facebook and graphcool
- Broad IDE support
- Plugin Architecture

# .graphqlconfig

```
{  
  "schemaPath": "schema.graphql",  
  "extensions": {  
    "endpoints": {  
      "dev": {  
        "url": "https://example.com/graphql",  
        "headers": {  
          "Authorization": "Bearer ${env:AUTH_TOKEN_ENV}"  
        },  
        "subscription": {  
          "url": "ws://example.com/graphql",  
          "connectionParams": {  
            "Token": "${env:YOUR_APP_TOKEN}"  
          }  
        }  
      }  
    }  
  }  
}
```



# Programmatic API

```
import { getGraphQLConfig } from 'graphql-config'

const config = getGraphQLConfig('./optionalProjectDir')
const schema =
config.getConfigForFile(filename).getSchema()

// use schema for your tool/plugin
```



# graphql-cli/graphql-cli

## graphql-cli

npm package 2.13.1 circled passing

💻 Command line tool for common GraphQL development workflows

### Features

- Helpful commands to improve your workflows like `get-schema`, `diff` & `playground`
- Compatible with editors and IDEs based on [graphql-config](#)
- Powerful plugin system to extend `graphql-cli` with custom commands

You can see it in action here:



A terminal window showing the execution of the `graphql init` command. The window has a dark background with red, yellow, and green window control buttons at the top left. The command is entered at the prompt, followed by two questions about schema file path and endpoint URL.

```
~> graphql init
? Local schema file path: schema.graphql
? Endpoint URL (Enter to skip): https://s
```

# GraphQL CLI

- Fetch Schema from Server
- Diff schemas
- Run queries
- Create `.graphqlconfig`
- Plugins: `voyager`, `codegen`, `bindings`



# graphql-binding/graphql-binding

## graphql-binding

circleci passing npm package 1.2.0

### Overview

🔗 GraphQL bindings are **modular building blocks** that allow to embed existing GraphQL APIs into your own GraphQL server. Think about it as turning (parts of) GraphQL APIs into reusable LEGO building blocks.

The idea of GraphQL bindings is introduced in detail in this blog post: [Reusing & Composing GraphQL APIs with GraphQL Bindings](#)

### Install

```
yarn add graphql-binding
```

# graphql-binding

- Exposes GraphQL APIs
- Allows composition
- Type safety (TypeScript)
- Static and dynamic bindings

# graphql-binding Example

```
// schema
type Query {
  user(id: ID!): User
}
type Mutation {
  createUser(): User!
}

// resolvers
findUser(parent, args, context, info) {
  return binding.user({ id: args.id }, context, info);
}
newUser(parent, args, context, info) {
  return binding.createUser({}, context, info);
}
```



# [graphcool/graphql-import](#)

## graphql-import

[circleci](#) passing [npm package](#) 0.4.3

Import & export definitions in GraphQL SDL (also referred to as GraphQL modules)

### Install

```
yarn add graphql-import
```

### Usage

```
import { importSchema } from 'graphql-import'
import { makeExecutableSchema } from 'graphql-tools'

const typeDefs = importSchema('schema.graphql')
const resolvers = {}
```

# graphql-import

- Follows Schema Definition Language
- Allows Modular Type Definitions
- Allows recursive imports
- Allows cyclic dependencies

# Modular type definitions

```
// root.graphql
#import Todo from 'todo.graphql'
type MyTodo {
  user: String
  todos: [Todo]
}

// todo.graphql
type Todo {
  id: ID!
  text: String!
  complete: Boolean!
}
```

# importSchema(file)

```
console.log(importSchema('root.graphql'))  
  
// output  
type MyTodo {  
  user: String  
  todos: [Todo]  
}  
  
type Todo {  
  id: ID!  
  text: String!  
  complete: Boolean!  
}
```



# [apollographql/apollo-tracing](#)

## Apollo Tracing

Apollo Tracing is a GraphQL extension for performance tracing.

Thanks to the community, Apollo Tracing already works with most popular GraphQL server libraries, including Node, Ruby, Scala, Java, and Elixir, and it enables you to easily get resolver-level performance information as part of a GraphQL response.

Apollo Tracing works by including data in the extensions field of the GraphQL response, which [is reserved by the GraphQL spec for extra information that a server wants to return](#). That way, you have access to performance traces alongside the data returned by your query.

It's already supported by [Apollo Engine](#), and we're excited to see what other kinds of integrations people can build on top of this format.

We think this format is broadly useful, and we'd love to work with you to add support for it to your tools of choice. If you're looking for a first idea, we especially think it would be great to see support for Apollo Tracing in [GraphiQL](#) and the [Apollo Client developer tools](#)!

# apollo-tracing

- Allows performance tracing
- Returns traces in Response
- Requires GraphQL Server setup
- Supported by GraphQL Playground

# apollo-tracing Format

```
{  
  "data": <>, "errors": <>,  
  "extensions": {  
    "tracing": {  
      "version": 1, "startTime": <>, "endTime": <>, "duration": <>,  
      "parsing": { "startOffset": <>, "duration": <> },  
      "validation": { "startOffset": <>, "duration": <> },  
      "execution": {  
        "resolvers": [ {  
          "path": [ <>, ... ],  
          "parentType": <>, "fieldName": <>, "returnType": <>,  
          "startOffset": <>, "duration": <>,  
        },  
        ...  
      ]  
    }  
  }  
}
```

# Apollo Engine

Apollo Engine is a cloud service for schema management and performance metrics monitoring





[graphcool/graphql-yoga](#)



@[graphql-yoga](#)

[circleci](#) [passing](#) [npm package](#) [1.2.5](#)

Fully-featured GraphQL Server with focus on easy setup, performance & great developer experience

## Overview

- **Easiest way to run a GraphQL server:** Sensible defaults & includes everything you need with minimal setup.
- **Includes Subscriptions:** Built-in support for GraphQL subscriptions using WebSockets.
- **Compatible:** Works with all GraphQL clients (Apollo, Relay...) and fits seamless in your GraphQL workflow.



zen.digital

© 2019

# graphql-yoga

- Seed project to create GraphQL Servers
- Uses common building blocks
- Minimal setup & best practices

# graphql-yoga dependencies

- apollo-server
- graphql-subscriptions
- graphql-tools
- graphql-playground

# graphql-yoga features

- File Upload, Subscriptions
- Types (TypeScript), Playground
- Apollo tracing
- Extensible via Express Middlewares
- Accepts application/json/graphql



# GraphQL Community

Find out what's happening in GraphQL Meetup groups around the world and start meeting up with the ones near you.

**60,408**  
members

**125**  
groups



**zen.digital**

© 2019