GraphQL with React

...

Daniel Zen
@danielzen @zendigital
training@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

zen.digital

Tell me a little bit about yourself ...

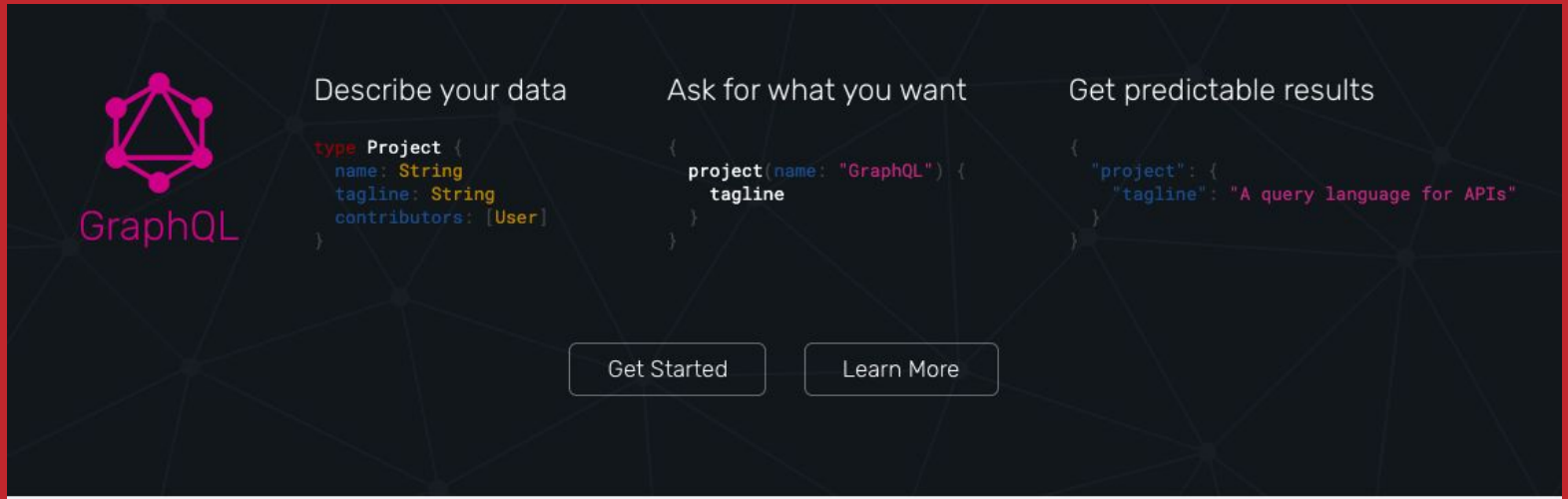
- Are you all JavaScript developers?
  - http://zen.digital/javascript/quiz
- Any designers?
- Managers?
- What types of industry?
- Anything else you would like me to know?

zen.digital

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

zen.digital



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?

zen.digital


HEROKU
SERVERLESS
UDACITY
CNN
yelp
OpenTable
The New York Times
EXPRESS
CONCUR
intuit
zendesk
IBM
airbnb
twitch
NATIONAL GEOGRAPHIC
Expo
VISA
credit karma
amazon
Walmart
glassdoor
Product Hunt
coursera
MLS
AMERICAN EXPRESS
METEOR
NBC UNIVERSAL
EXPRESS
ticketmaster

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

zen.digital

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

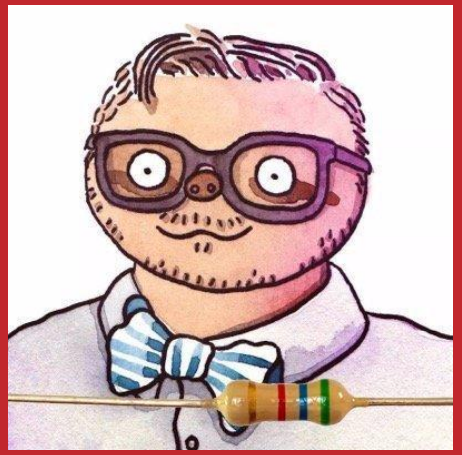
You can use GraphQL on top of your existing infrastructure: REST, SOAP, existing databases, or anything else. Organize your data into a clean, unified API and query it all at once.

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.

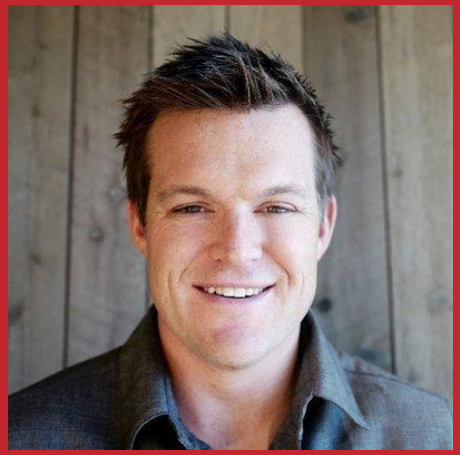
Used in production at

GraphQL creators



Lee Byron

@leeb



Nick Schrock

@schrockn

Dan Schafer

@dlschafer

zen.digital

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
- - First GraphQL Summit

zen.digital

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

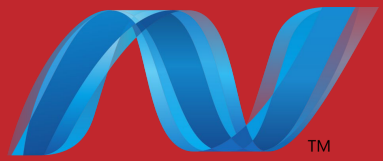
zen.digital

GraphQL Backend Servers

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

zen.digital

Implementations



Query Language 

```
// GET '/graphql'
  allTodoes(first: 1) {
    complete
    text
  }
```

- Query has same shape as result
- Check out the following page
- https://graphql.github.io/learn/

```
// result
{
  "data": {
    "allTodoes": [
      {
        "complete": true,
        "text": "Learn GraphQL"
      }
    ]
  }
}
```

zen.digital

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

zen.digital

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

zen.digital

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.

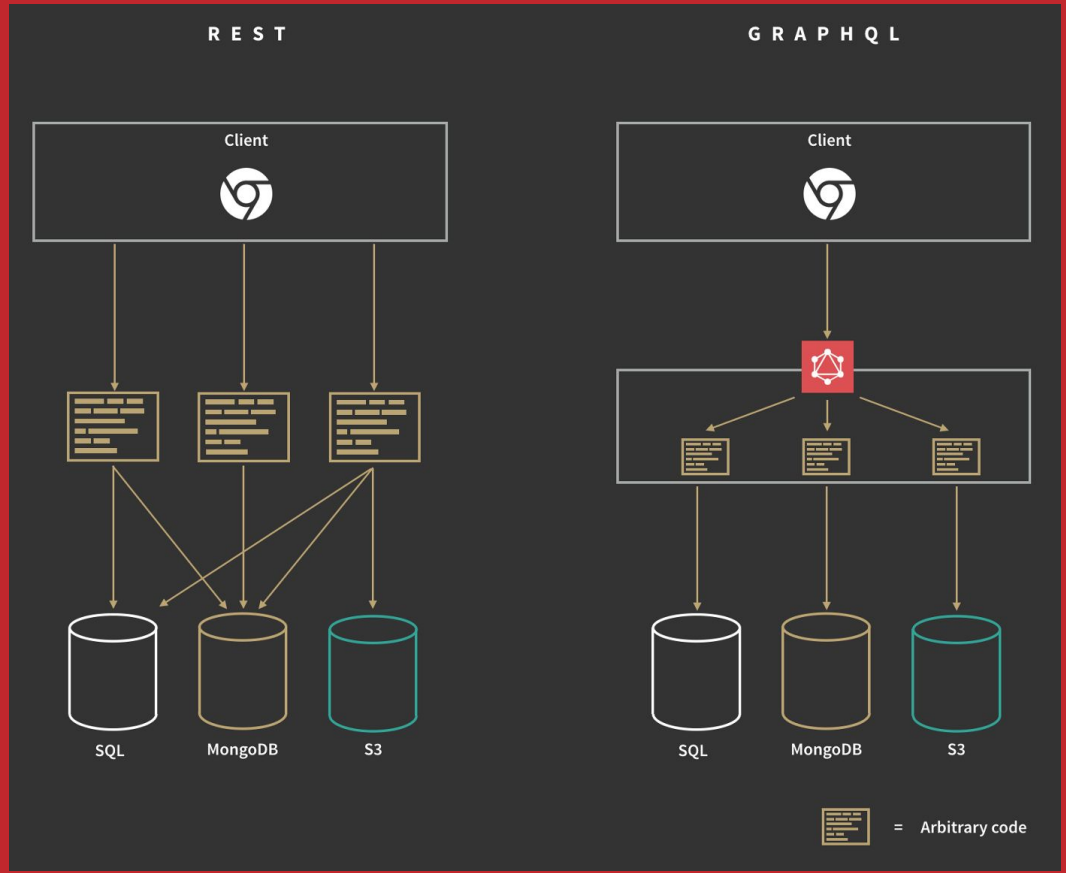
zen.digital

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.

zen.digital

Overview:



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

zen.digital

Why use GraphQL

Some reasons

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

zen.digital

GraphQL Clients

- Apollo Client
- Relay Modern
- urql

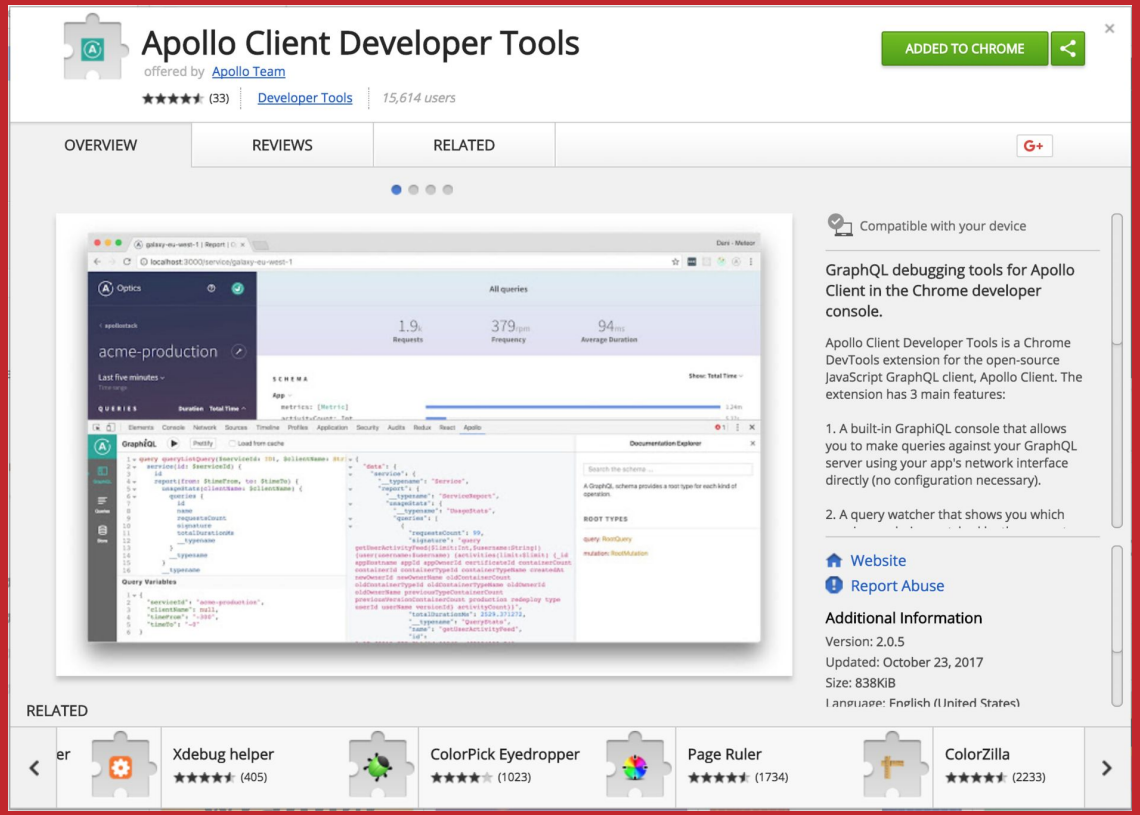
Tools

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

zen.digital

Apollo Client Developer Tools



zen.digital

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

zen.digital

Don't Versioning - Evolve your API

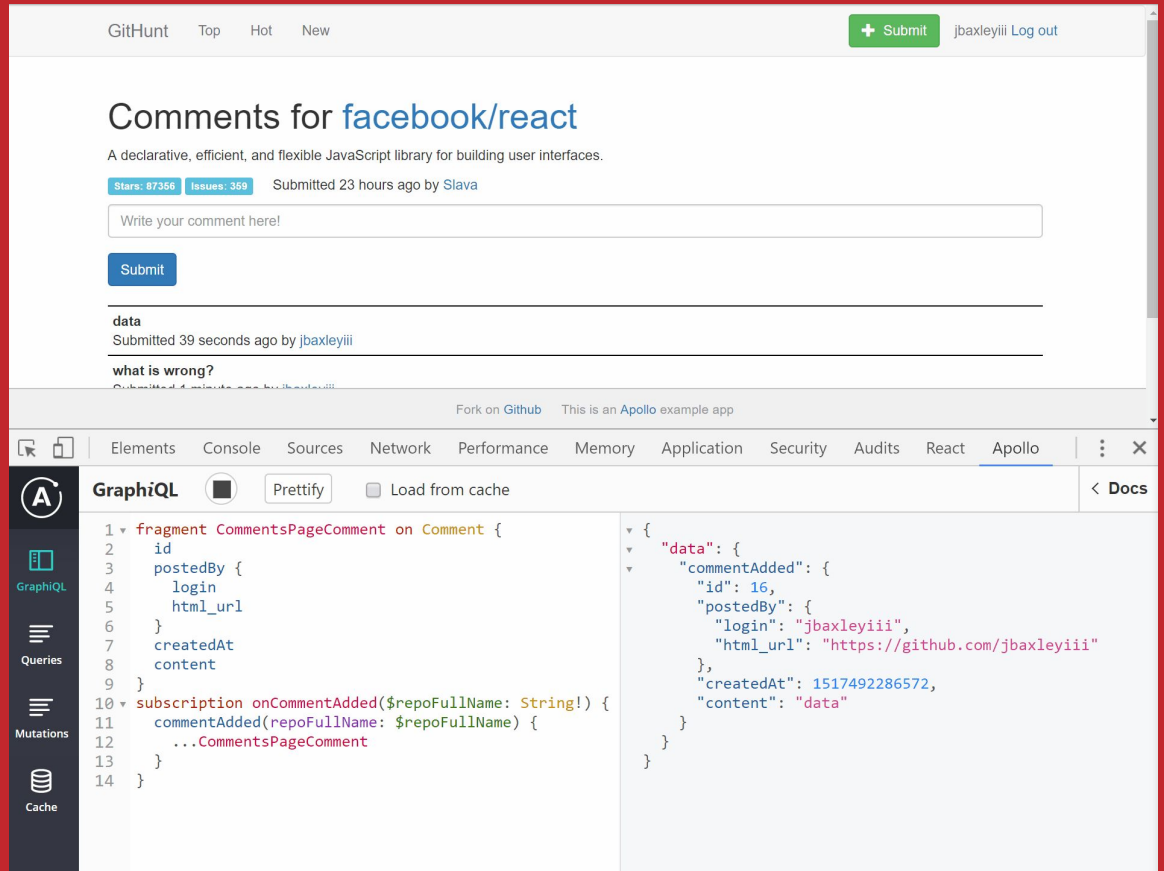
While there's nothing that prevents a GraphQL service from being versioned just like any other REST API, GraphQL takes a strong opinion on avoiding versioning by providing the tools for the continuous evolution of a GraphQL schema.

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.

zen.digital

Subscription Support



zen.digital

Relay Modern

zen.digital

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

zen.digital

Apollo Client

zen.digital

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 mor

zen.digital

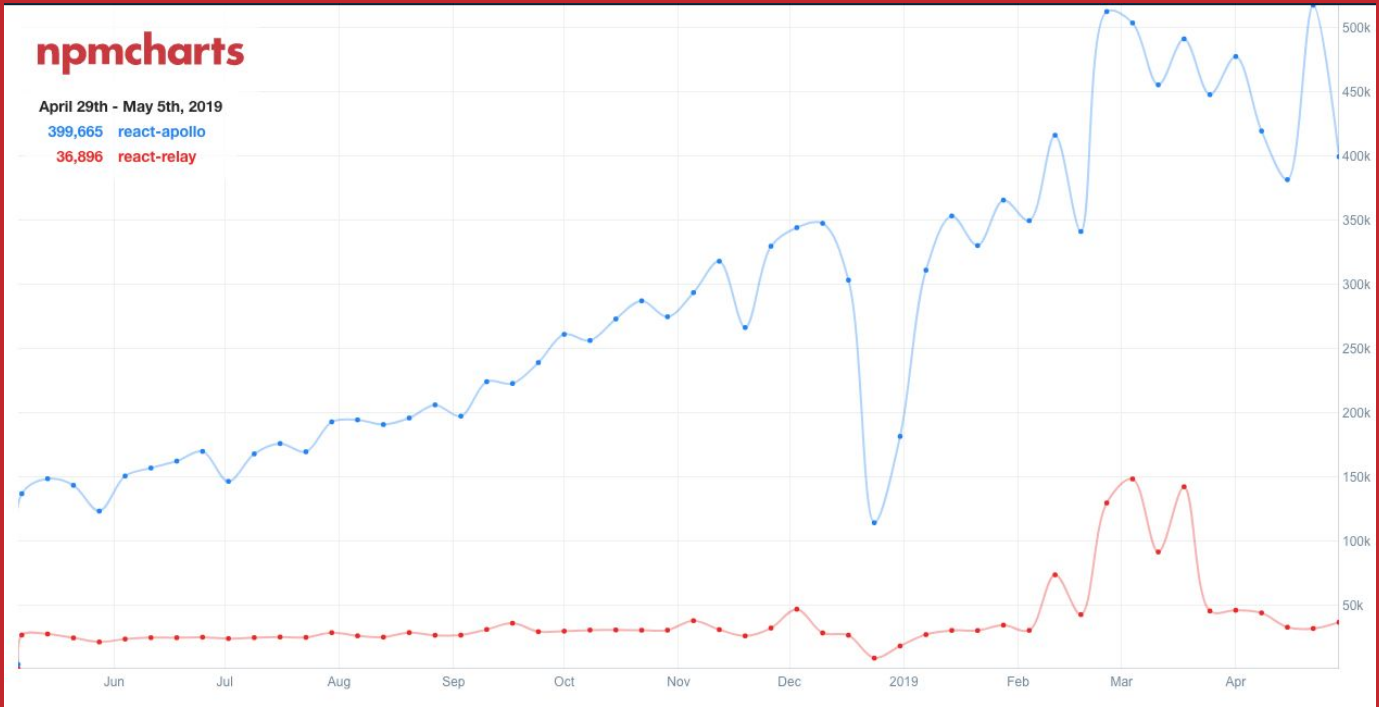
Apollo Client Overview

- 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

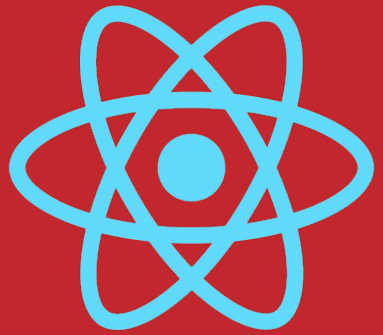
Relay vs Apollo Downloads



zen.digital

CLIENT

SERVER



React

Relay
Modern

Apollo
Client

GraphCool

zen.digital

CLIENT

SERVER

Angular

Apollo
Client

GraphCool

zen.digital

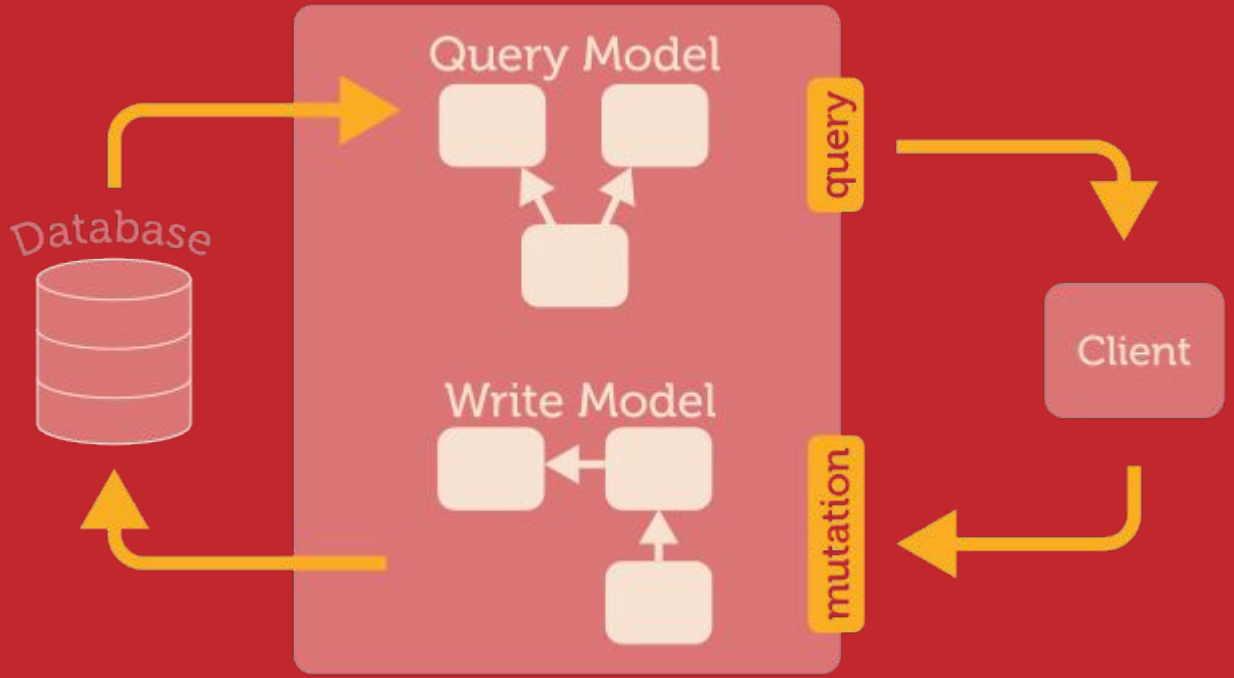
Setup

- Relay compiler/ Babel plugin
- Environment
  - Network
  - Store
- Via attribute

- Loader graphql-tag
- ApolloClient
  - Apollo Link
  - Apollo Cache
- ApolloProvider HOC

zen.digital

Queries



source: gist

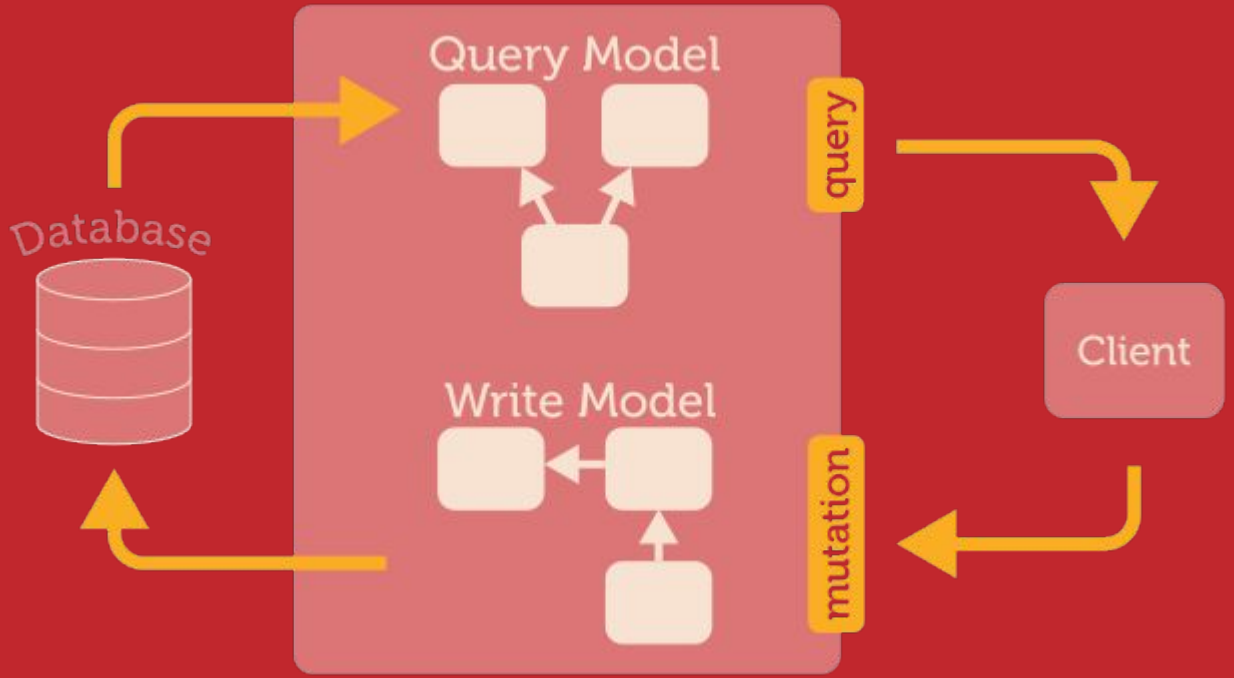
Queries

- QueryRenderer
  - Environment
  - Query
- Queries generation
  - Automated
- Opinionated

- graphql HOC
- Queries generation
  - Manual
- Flexible

zen.digital

Mutations



source: gist

Mutations

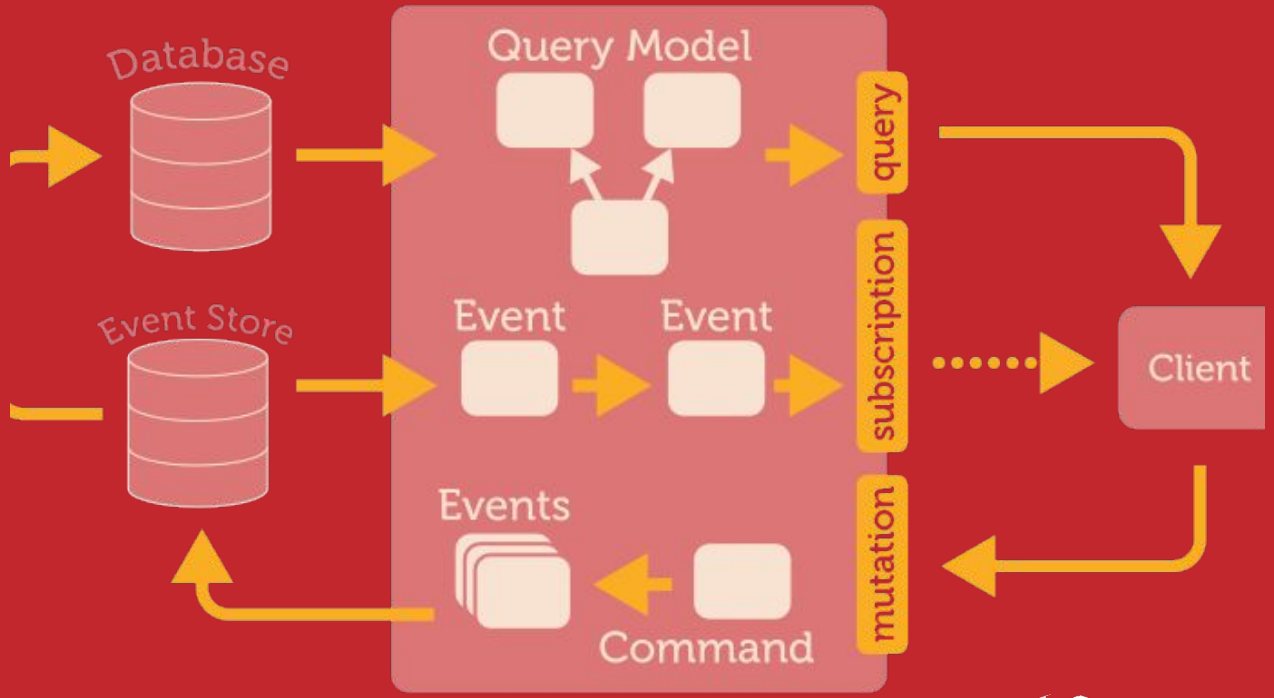
- commitMutation
  - Environment
- Update Store via
  - Connection
  - Fields/LinkedRecord
- Optimistic Updates

- Added via props
- graphql HOC/mutate
- Update Store via Cache/arguments
- Optimistic Updates

Apollo considered better at mutations

zen.digital

Subscriptions



source: gist

zen.digital

Subscriptions

- requestSubscription
- Environment
- Update Store via
  - Connection
  - Fields/LinkedRecord

- subscribeToMore
- Update Store via reducer/arguments

zen.digital

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

zen.digital

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

zen.digital

GraphQL Schema

zen.digital

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 {
  allTodoes(first: Int): [Todo]
}

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

Playground

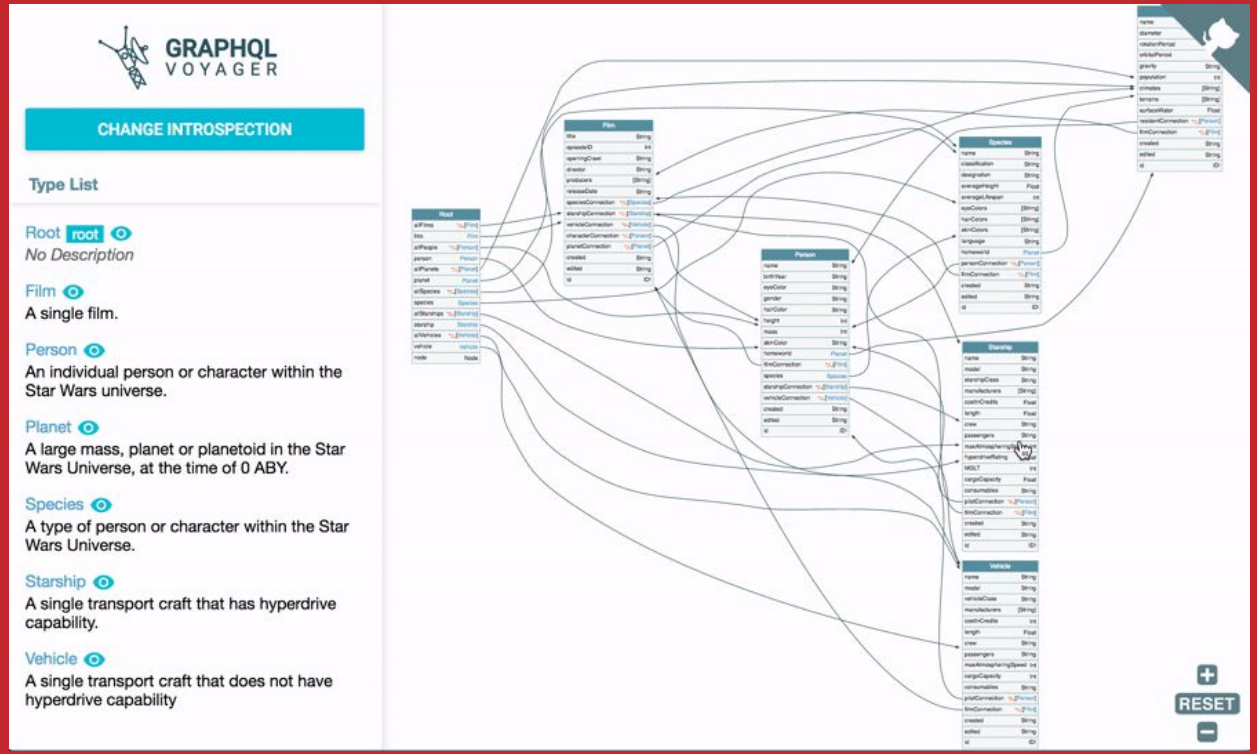
```
allTodoes(last: 2) {
    id, text, complete
  }
}
```

with alias:

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

zen.digital

apis-guru/graphql-voyager



zen.digital

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
```

zen.digital

Query Syntax

Quick ApolloClient Lab

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

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

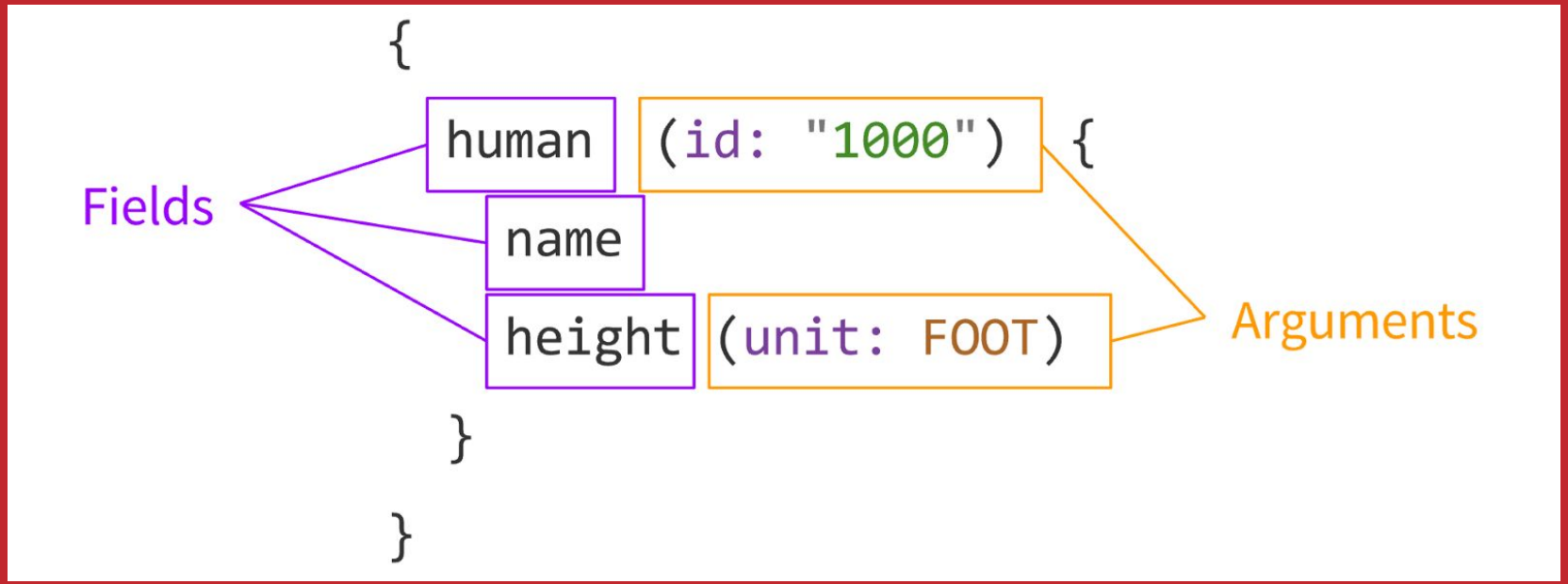
This is a very quick lab that will show you how little it takes to get up and running with ApolloClient.

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/

zen.digital

Named Query

Operation type

Operation name

Variable definitions

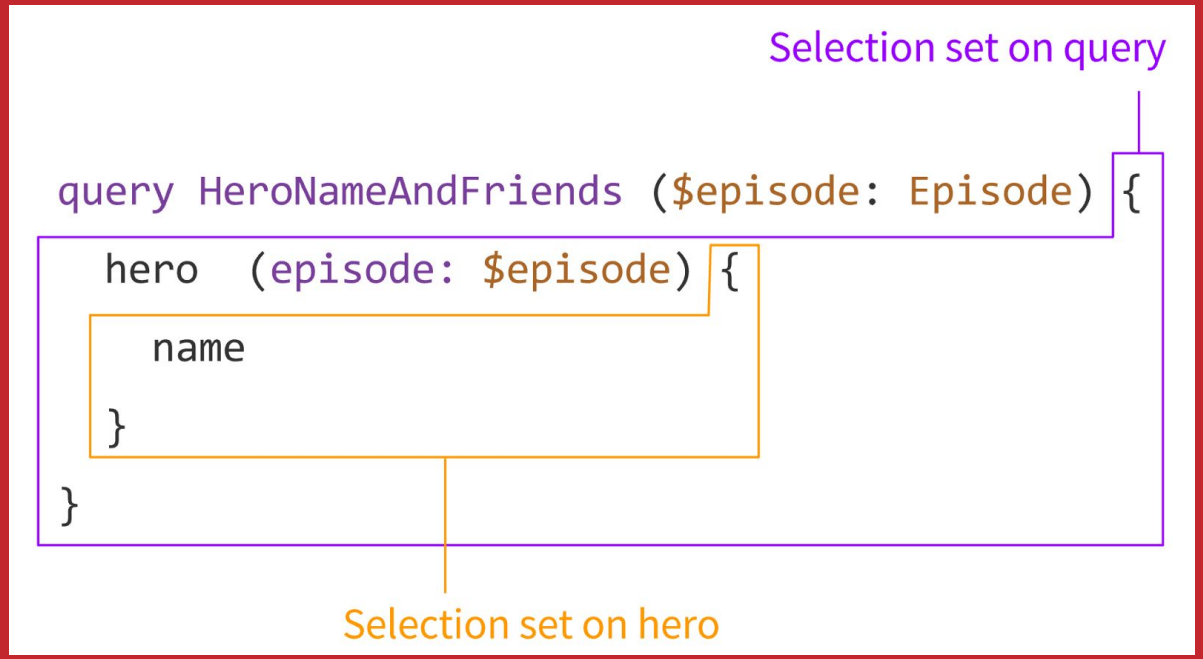
```
query HeroNameAndFriends ($episode: Episode) {
  hero(episode: $episode) {
    name
  }
}
```

Query Variables:

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

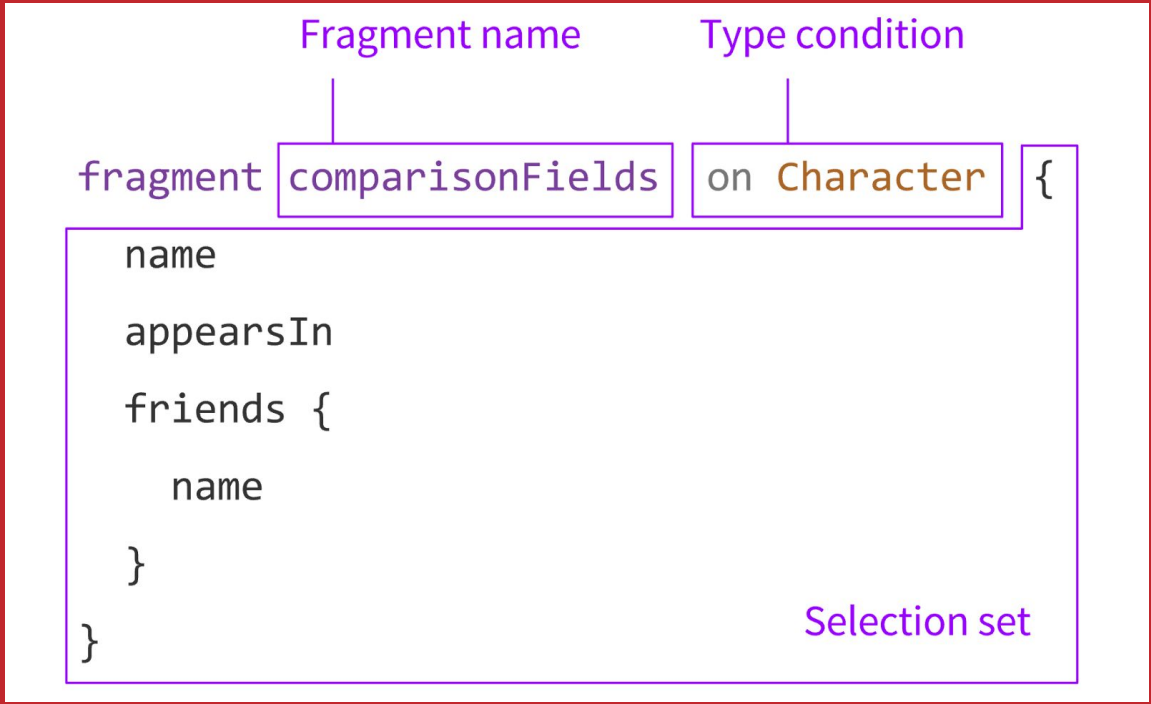
zen.digital

Selection Sets



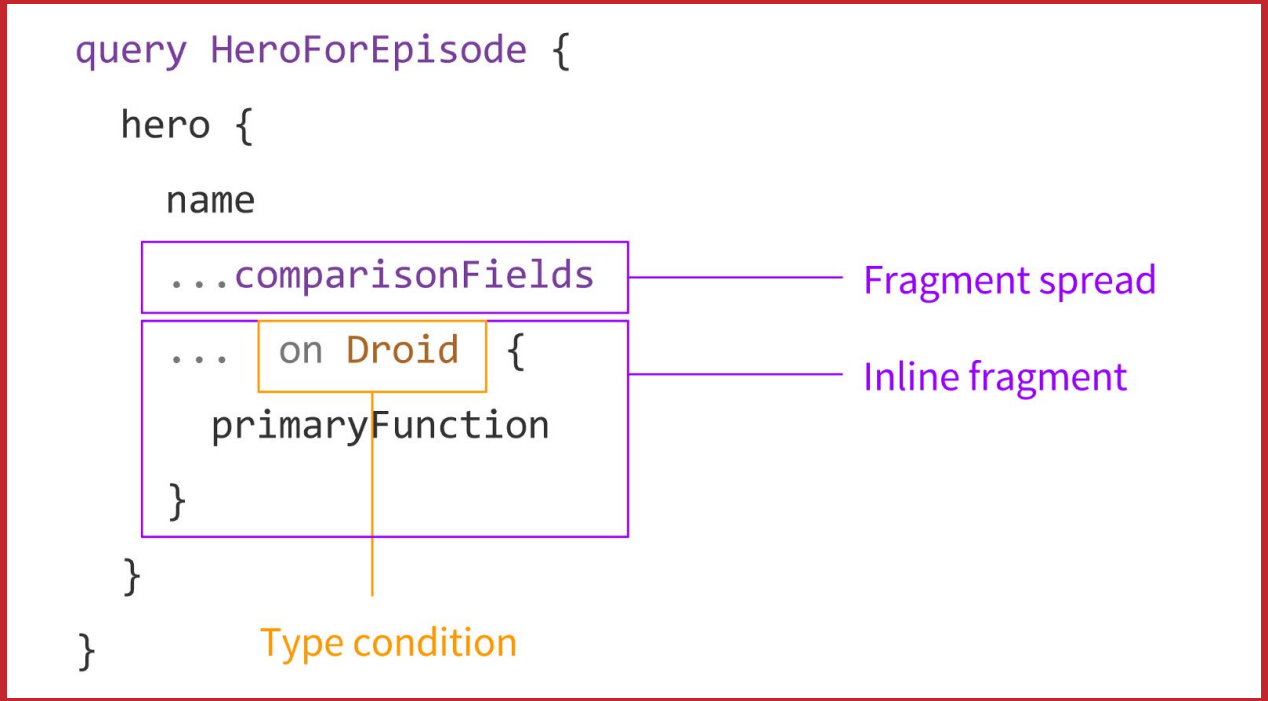
zen.digital

Fragments



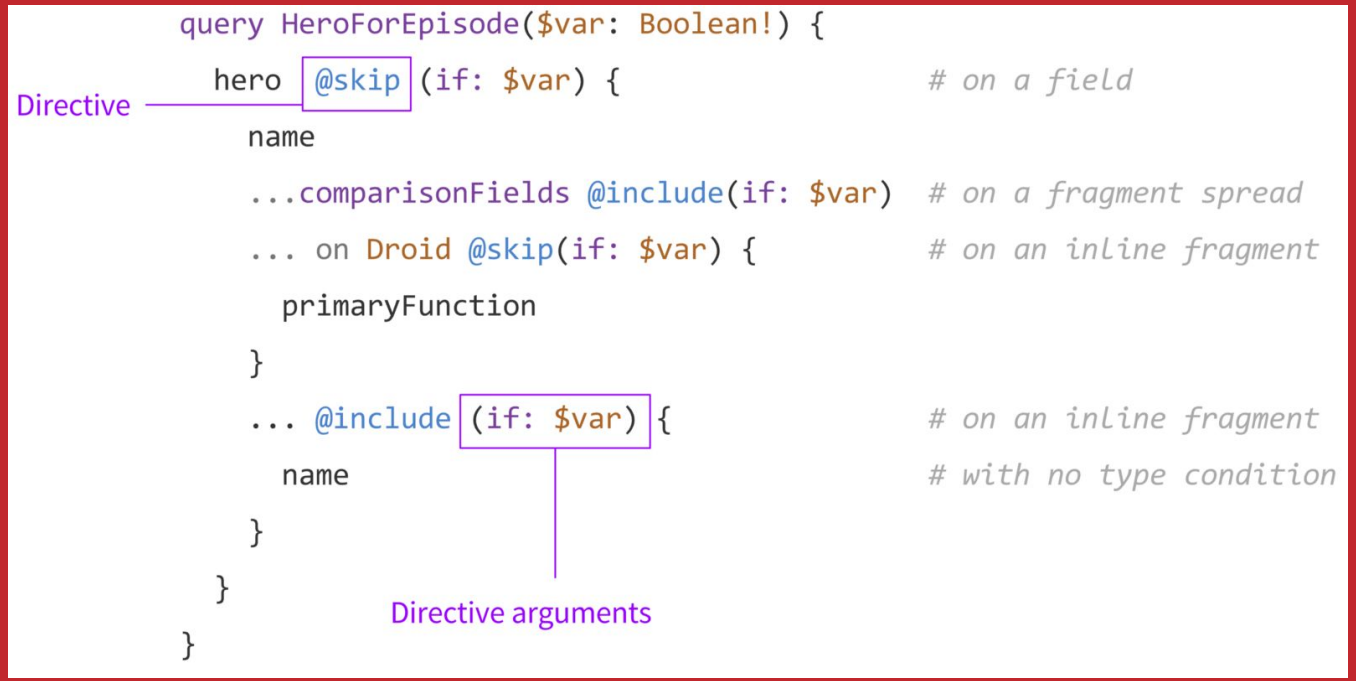
zen.digital

Using Fragments



zen.digital

Directives



zen.digital

Exercises

zen.digital

Graphcool Lab

Todo App

zen.digital

Todo Schema

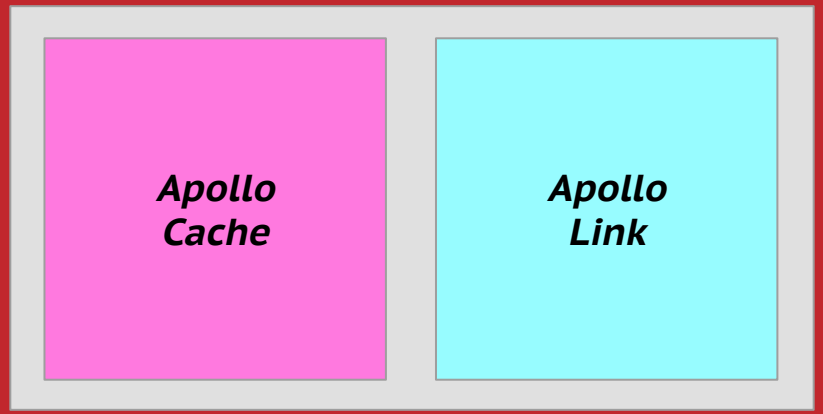
project.graphcool

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

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

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

zen.digital

demo

Apollo Client



zen.digital

Apollo Cache

Apollo Cache Packages

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

zen.digital

InMemoryCache

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

zen.digital

Apollo Cache Example

Query

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

Result

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

zen.digital

Normalised and flattened

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

__typename:id

read/writeQuery

read/writeFragment

zen.digital

Retrieving todos from cache

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

zen.digital

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')
      }
    }))
  }
});
```

zen.digital

GraphQL Server

apollographql/graphql-tools

GraphQL-tools: generate and mock GraphQL.js schemas

npm package 2.20.2 | build failing | coverage 93% | slack join

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

graphql-tools

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

zen.digital

Schema

```
const typeDefs = `

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

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

zen.digital

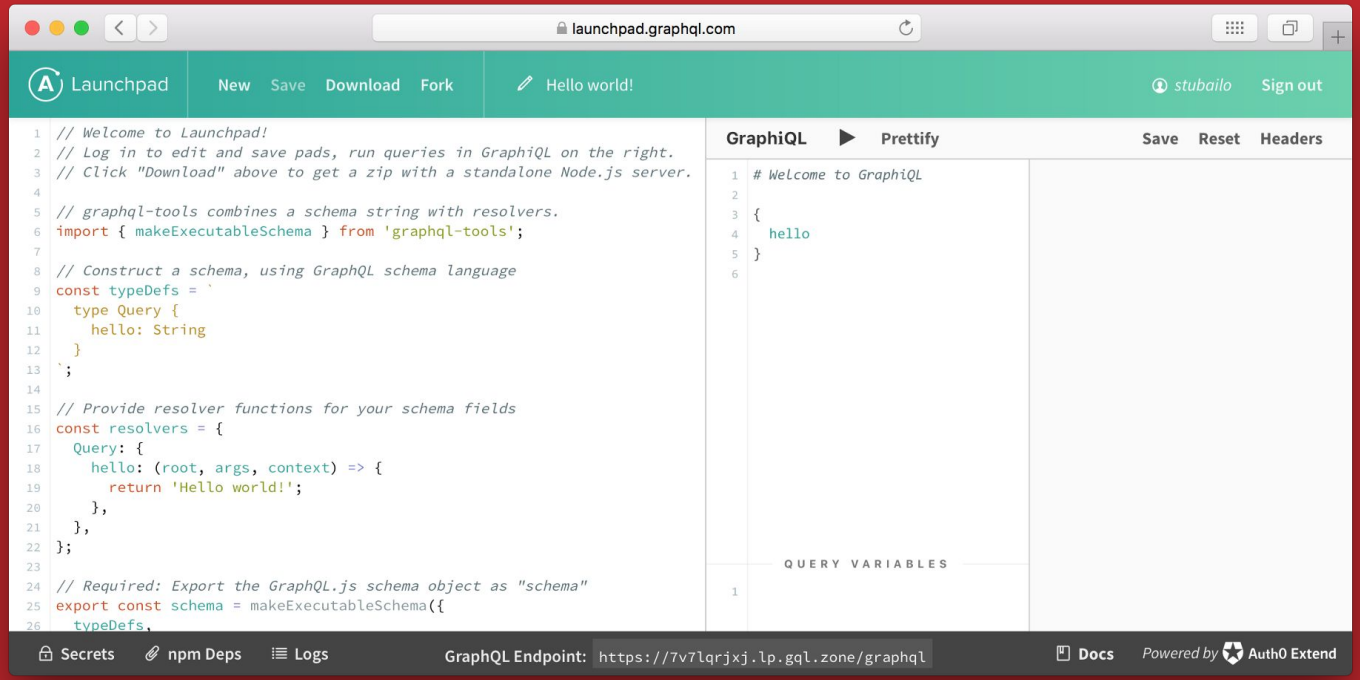
Resolvers

```
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



zen.digital

Check out Launchpad Running at:

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

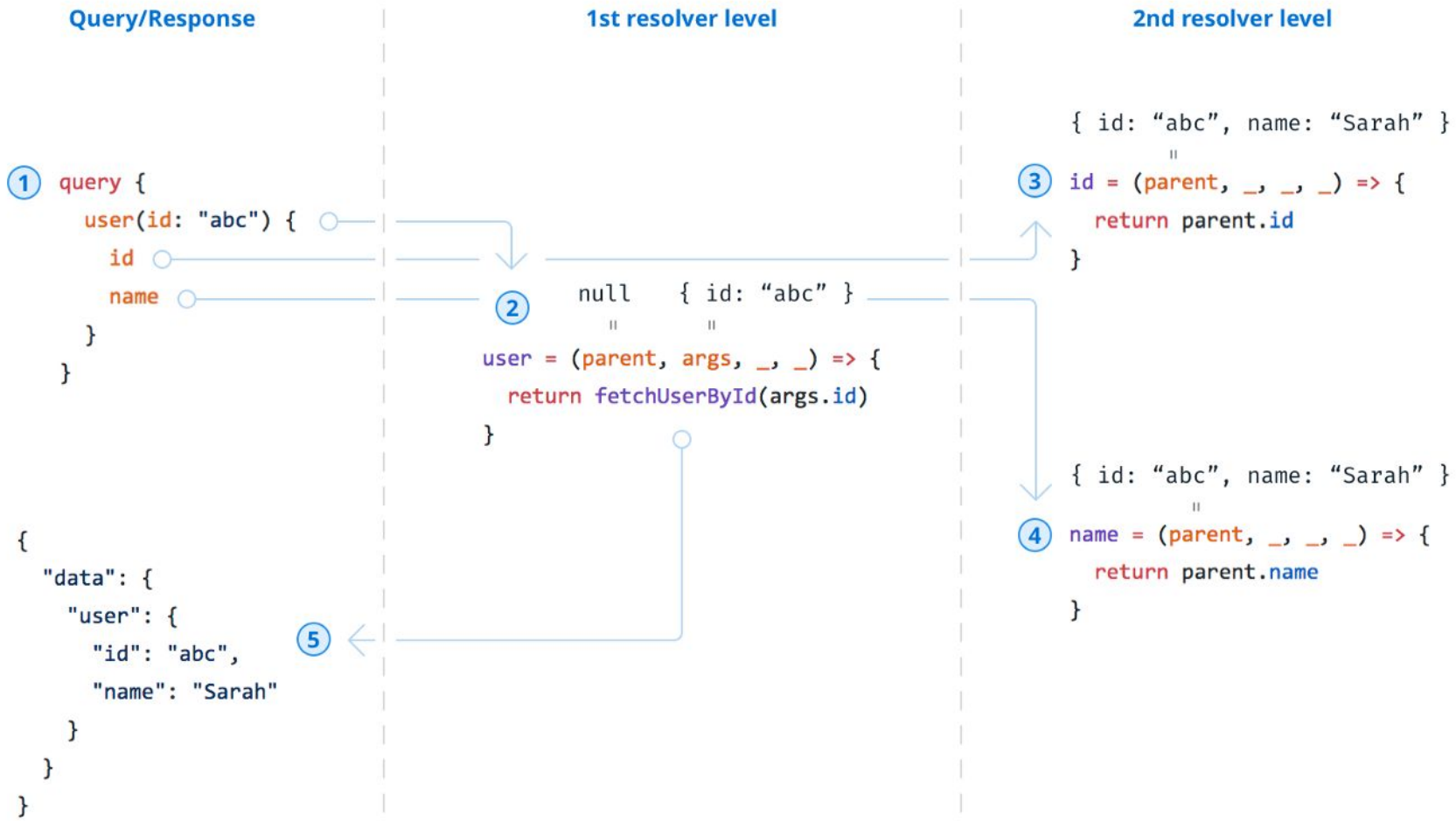
Resolvers

- 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)
    },
  },
}
```

zen.digital



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

zen.digital

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

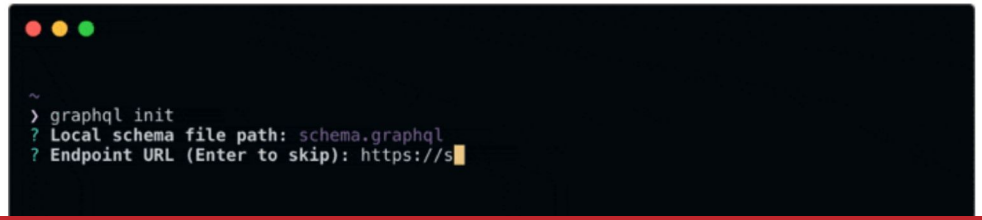
npm package 2.13.1 | circleci passing

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:



zen.digital

GraphQL CLI

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

zen.digital

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
```

zen.digital

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);
}
```

zen.digital

graphcool/graphql-import

graphql-import

Import & export definitions in GraphQL SDL (also refered 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 = {}
```

zen.digital

graphql-import

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

zen.digital

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!
}
```

zen.digital

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!

zen.digital

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

zen.digital

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

graphql-yoga

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

zen.digital

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

zen.digital

zen.digital

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