GraphQL

Eric Clemmons

What is GraphQL?

WHAT WE WANT

```
{
  user: {
    id: 123,
    name: "Eric Clemmons",
    email: "eric@smarterspam.com",
    avatar: {
      uri: "http://.../pic.jpg",
      width: 50,
      height: 50
    }
  }
}
```

HOW WE GET IT

```
{
  user(id: 123) {
    id,
    name,
    emal,
    avatar(size: 50) {
      uri,
      width,
      height
    }
  }
}
```

Declarative

Query responses are decided by the client rather than the server. A GraphQL query returns exactly what a client asks for and no more.

Compositional

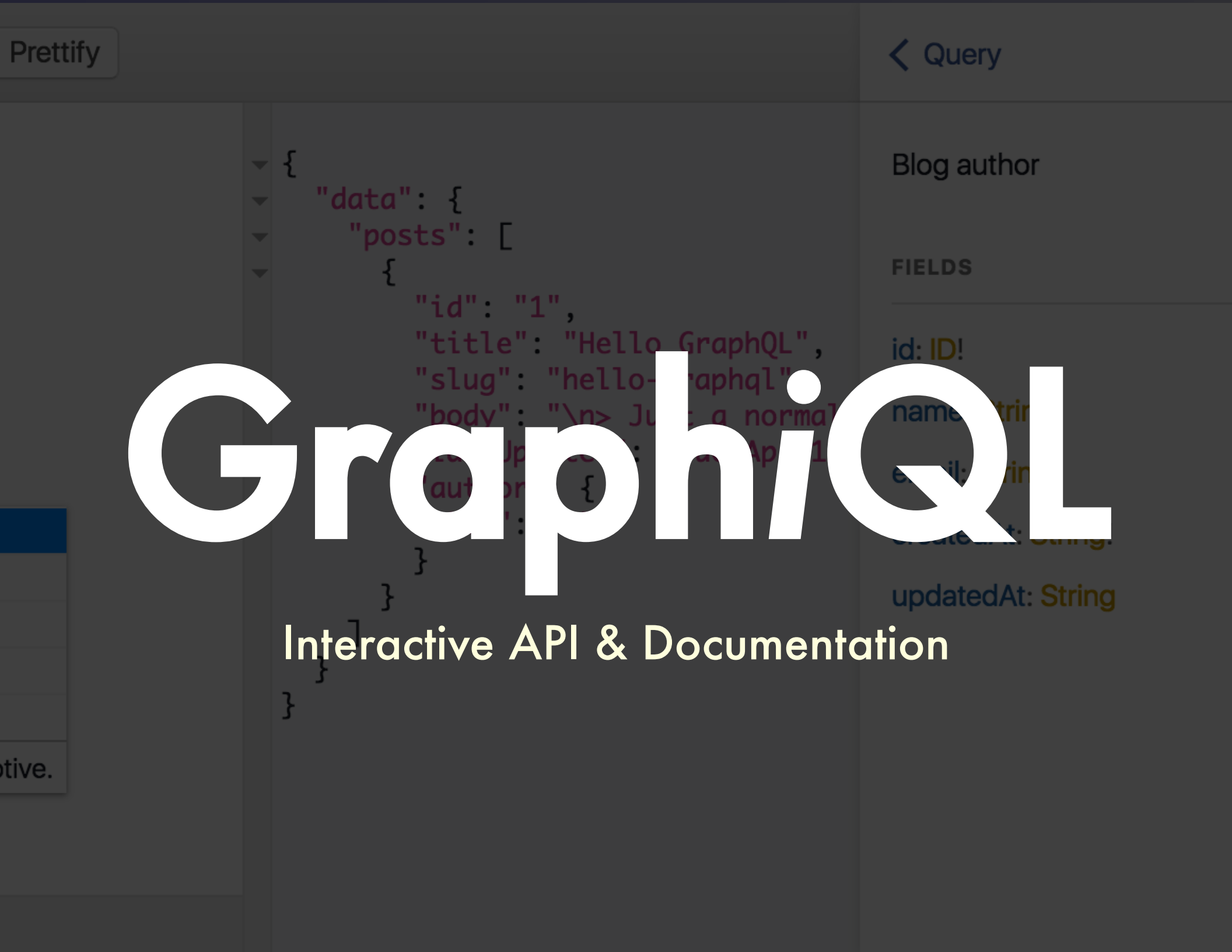
A GraphQL query itself is a hierarchical set of fields. The query is shaped just like the data it returns. It is a natural way for product engineers to describe data requirements.

Strong-typed

A GraphQL query can be ensured to be valid within a GraphQL type system at development time allowing the server to make guarantees about the response. This makes it easier to build high-quality client tools.

Why GraphQL?

- REST is great for CRUD, not graphs.
- Client defines *only* the data it needs.
- Client makes a *single* request.
- Server is responsible for:
  - Schema
  - Querying
  - Caching
- How you fetch the data is up to *you*.


Prettify
Query
{
"data": {
"posts": [
{
"id": "1",
"title": "Hello GraphQL",
"slug": "hello-graphql"
}
]
}
}
Blog author
FIELDS
id: ID!
updatedAt: String
GraphiQL
Interactive API & Documentation

Where can you use GraphQL?

- JavaScript
- Ruby
- PHP
- Python
- Java
- Go
- ...

When can you use GraphQL?

Today.

How to use GraphQL

EXPRESS

```
1. import { middleware as webpack } from "@terse/webpack";
2. import express from "express";
3.
4. import config from "../webpack.config.browser.babel.js";
5.
6. import api from "./middleware/api";
7. import files from "./middleware/files";
8. import view from "./middleware/view";
9.
10. express()
11.   .use(webpack(module, config))
12.   .use(files)
13.   .use(api)
14.   .use(view)
15.   .listen(3000, "localhost", (err) => {
16.     if (err) {
17.       throw err;
18.     }
19.
20.     console.info("   Listening at http://localhost:3000/"); //
```

```
 9.   client: "mysql",
10.   connection: {
11.     database: "graphql_demo",
12.     user: "root",
13.   },
14. });
15.
16. const loaders = {
17.   post: new DataLoader(function fetchBySlugs(slugs) {
18.     console.info("[post] Fetching by ", slugs);
19.
20.     return db("post").whereIn("slug", slugs);
21.   }),
22. };
23.
24. export default express.Router()
25.   .all("/api", graphql({
26.     context: { db, loaders },
27.     graphiql: true,
28.     pretty: true,
29.     schema,
30.   }))
31. ;
```

Pass /api requests to GraphQL

SCHEMA

```
1. import { GraphQLSchema } from "graphql";
2.
3. import MutationType from "./MutationType";
4. import QueryType from "./QueryType";
5.
6. export default new GraphQLSchema({
7.   mutation: MutationType,
8.   query: QueryType,
9. });
10.
```

query for reads, mutation for writes

```
16.
17.   fields() {
18.     return {
19.       author: {
20.         type: AuthorType,
21.
22.         args: {
23.           id: { type: new GraphQLNonNull(GraphQLID) },
24.         },
25.
26.         resolve(parent, args, context) {
27.           const { db } = context;
28.           const { id } = args;
29.
30.           return db("author")
31.             .first()
32.             .where("id", id)
33.           ;
34.         },
35.       },
36.
37.       authors: {
38.         type: new GraphQLList(AuthorType),
```

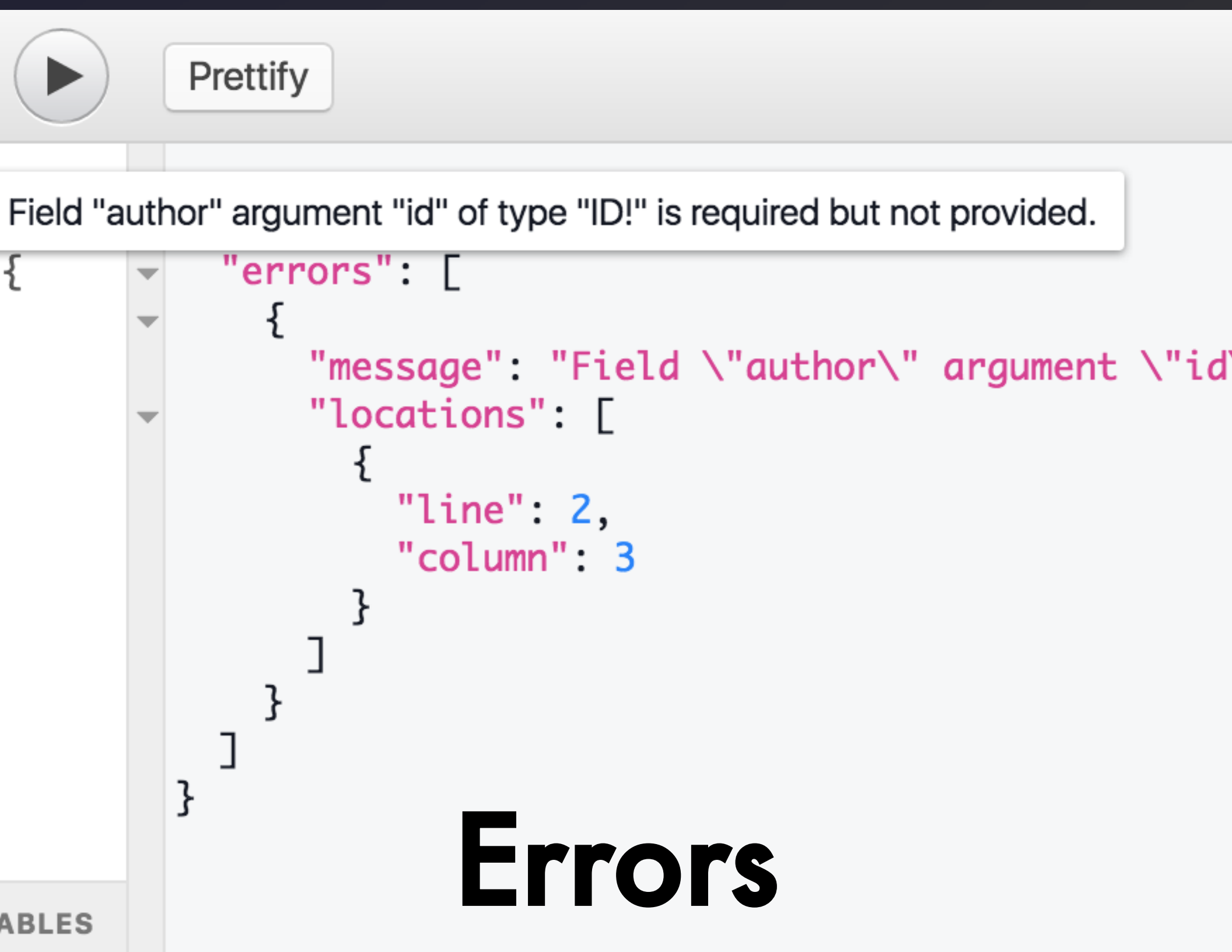
Fetching the author #1

```
42.
43.           return db("author");
44.         },
```

```
 2.   GraphQLID,
 3.   GraphQLNonNull,
 4.   GraphQLObjectType,
 5.   GraphQLString,
 6. } from "graphql";
 7.
 8. export default new GraphQLObjectType({
 9.   name: "Author",
10.   description: "Blog author",
11.
12.   fields() {
13.     return {
14.       id: { type: new GraphQLNonNull(GraphQLID) },
15.       name: { type: new GraphQLNonNull(GraphQLString) },
16.       email: { type: new GraphQLNonNull(GraphQLString) },
17.       createdAt: { type: new GraphQLNonNull(GraphQLString) },
18.       updatedAt: { type: GraphQLString },
19.     };
20.   },
21. });
22.
```

Database columns

Querying an Author


Prettify
Field "author" argument "id" of type "ID!" is required but not provided.
{
"errors": [
{
"message": "Field \"author\" argument \"id
"locations": [
{
"line": 2,
"column": 3
}
]
}
]
}
ABLES

Errors


Prettify
id: 1) {
{
"data": {
"author": {
"id": "1",
"name": "Eric Clemmo
"email": "eric@smart
}
}
}
ABLES
Inline

Prettify

```
Author($id: ID!) {
id: $id) {
```

ABLES

```
{
  "data": {
    "author": {
      "id": "1",
      "name": "Eric Clemmo
      "email": "eric@smart
    }
  }
}
```

Variables

Prettify

```
orInfo on Author {

or($id:ID!) {
$id) { ...AuthorInfo }
```

```
{
  "data": {
    "author": {
      "id": "1",
      "name": "Eric Clemmons
      "email": "eric@smarter
    }
  }
}
```

Fragments

DataLoader

Simple wrapper for fetching & caching queries.

```
 6. import schema from "../schema";
 7.
 8. const db = knex({
 9.   client: "mysql",
10.   connection: {
11.     database: "graphql_demo",
12.     user: "root",
13.   },
14. });
15.
16. const loaders = {
17.   post: new DataLoader(function fetchBySlugs(slugs) {
18.     console.info("[post] Fetching by ", slugs);
19.
20.     return db("post").whereIn("slug", slugs);
21.   }),
22. };
23.
24. export default express.Router()
25.   .all("/api", graphql({
26.     context: { db, loaders },
27.     graphiql: true,
28.     pretty: true,
32.
```

API

post loader

QUERYTYPE.JS

```
post { ... }
```

```
71.           map(({ slug }) => slug)
72.           body: { type: new GraphQLNonNull(GraphQLString) },
73.         },
74.
75.         resolve(parent, args, context) {
76.           const { db } = context;
77.           const { post } = context.loaders;
78.           const { title, slug, body } = args;
79.
80.           const newSlug = kebabCase(title);
81.
82.           return db("post")
83.             .where("slug", slug)
84.             .update({
85.               title,
86.               slug: newSlug,
87.               body,
88.               updated_at: new Date(),
89.             })
90.             .then(() => post.clear(slug))
91.             .then(() => post.load(newSlug))
92.           ;
93.         },
94.       }
95.     }
96.   });
97. });
98.
```

updatePost(...) { ... }

Demo

Recommended Resources

- learngraphql.com
- github.com/mugli/learning-graphql
- graphql.org
- github.com/chentsulin/awesome-graphql
- github.com/facebook/dataloader
- github.com/matthewmueller/graph.ql
- github.com/graphql/graphql-js

Thanks!

https://github.com/ericclemmons/graphql-demo