CTW SUMMIT

Igniting the Power of Technology

GraphQL Ins & Outs

A Talk/Workshop by Jorge Duque & Toby Selway

A bit about REST

```
GET /api/v1/characters/5cd99d4bde30eff6ebccfc15

{
  "data": {
    "_id": "5cd99d4bde30eff6ebccfc15",
    "height": 1.06,
    "race": "Hobbit",
    "gender": "Male",
    "birth": "22 September, TA 2968",
    "spouse": "",
    "death": "Unknown (Last sighting: September 29, 3021) (SR 1421)",
    "realm": "",
    "hair": "Brown",
    "name": "Frodo Baggins",
    "avatar": "https://cards.scryfall.io/large/front/d/e/de1c0399-9db8-4901-b72e-0010eb9b92b0.jpg"
  },
}
```

A bit about REST

```
// POST /api/v1/characters?race=hobbits&gender=male

// Request
{
  "birth": "22 September, TA 2968",
  "hair": "Brown",
  "name": "Frodo Baggins",
}

// Response
// 201 Created

{
  "data": {
    "_id": "5cd99d4bde30eff6ebccfc15",
    "race": "Hobbit",
    "gender": "Male",
    "birth": "22 September, TA 2968",
    "hair": "Brown",
    "name": "Frodo Baggins",
  },
}
```

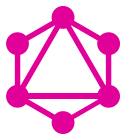
A bit about REST

```
// GET /api/v1/characters/5cd99d4bde30eff6ebccfc15/friends

// Response
{
  "data": [
    "520d3da9-f423-4dc7-8942-5463d14a1210",
    "b86e3bd5-b27a-496e-8444-96c52c717035",
    "99df0821-41f8-4e62-bec9-5cad93d97884"
  ],
}

// GET /api/v1/characters/520d3da9-f423-4dc7-8942-5463d14a1210  (times the number of friends)

// Response
{
  "data": {
    "_id": "520d3da9-f423-4dc7-8942-5463d14a1210",
    "race": "Maiar",
    "gender": "Male",
    "birth": "Before the the Shaping of Arda",
    "hair": "Grey, later white",
    "name": "Gandalf",
    "avatar": "https://cards.scryfall.io/large/front/c/c/cc9cfcc7-be64-4871-8d52-851e43fe3305.jpg"
  },
}
```

History

A bit about GraphQL

Created by Facebook in 2012



Made Open-Source in 2015

Hosted by Linux Foundation

Became a SDL (Schema Definition Language) in 2018

Our Quest

Let us begin our adventure

- Schema
  - Scalar types
  - Non-scalar types
  - Lists
  - Nullable
  - Enums
  - Input types
  - Unions
    - Error handling
    - ...
- Queries
- Mutations
- Resolvers
- Schema Stitching (?)
- Federation (?)
- Subscriptions (?)

Pros

- No overfetching (less network overhead)
- Frontend & Backend share contract
- Reusability (more versatile queries that share fields)
- Tooling
  - Playground is pretty nice
  - Automatic documentation
  - Community

Cons

- It’s not REST (not "standard")
- Chatty microsservices (when wrapping existing API)
- Caching difficulties
- Collisions (federation)

Up to you

- Error handling left up to implementation

Schema

```
type Post {
  id: ID!
  body: String!
  image: String
  likes: Int!
}

type User {
  id: ID!
  name: String!
  email: String!
  disabled: Boolean!
  roles: [Role!]!
}
```

```
type Role {
  title: String!
  permissions: [Permission!]!
}

enum Permission {
  CREATE_POSTS
  VIEW_POSTS
  EDIT_POSTS
  DELETE_POSTS
}
```

```
type Query {
  post(id: ID!): Post
  posts: [Post!]!
}

type Mutation {
  deletePost(id: ID!)
  newPost(input: PostInput!): Post
}

input PostInput {
  body: String!
  image: String
}
```

Query

```
# Schema
type Query {
  post(id: ID!): Post
  posts: [Post!]!
}
```

```
# Client
query {
  post(id: "42") {
    id
    body
    image
    likes
  }
}
```

```
// Response
{
  "data": {
    "post": {
      "id": "42",
      "body": "Lorem ipsum",
      "image": null,
      "likes": 96
    }
  }
}
```

```
# Client
query {
  posts {
    id
    likes
  }
}
```

```
// Response
{
  "data": {
    "posts": [
      { "id": "41", "likes": 73 },
      { "id": "42", "likes": 96 },
      { "id": "43", "likes": 19 }
    ]
  }
}
```

Mutation

```
# Schema
type Mutation {
  deletePost(id: ID!): ID!
  newPost(input: PostInput!): Post
}

input PostInput {
  body: String!
  image: String
}
```

```
# Client
mutation {
  deletePost(id: "42")
}
```

```
// Response
{
  "data": {
    "deletePost": "42"
  }
}
```

```
# Client
mutation {
  newPost(input: {
    body: "They're taking the Hobbits to Isengard"
    image: "https://i.scdn.co/image/ab67706c0000da84f0ce261134f6e62f71fe03ec"
  }) {
    id
  }
}
```

```
// Response
{
  "data": {
    "post": {
      "id": "44"
    }
  }
}
```

Variables

```
mutation {
  newPost(input: {
    body: "They're taking the Hobbits to Isengard"
    image: "https://i.scdn.co/image/ab67706c0000da84f0ce261134f6e62f71fe03ec"
  }) {
    id
  }
}

```

```
input PostInput {
  body: String!
  image: String
}
```

```
mutation CreateNewPost($postInput: PostInput) {
  newPost(input: $postInput) {
    id
  }
}
```

```
// Variables
{
  "postInput": {
    "body": "They're taking the Hobbits to Isengard",
    "image": "https://i.scdn.co/image/ab67706c00..."
  }
}
```

Error Handling

Left to implementation

```graphql
# Schema
type User {
  name: String!
  email: String!
}

type ErrorNotFound {
  id: ID!
}

union UserResult = User | ErrorNotFound

type Query {
  user(id: ID!): UserResult
}
```

```graphql
# Client
query {
  user(id: "42") {
    ... on User {
      name
      email
    }
    ... on ErrorNotFound {
      id
    }
  }
}
```

There are other ways of handling errors, depending on client implementation 🔗

Fragments

Lil’ bits of reusability

```
# Schema
type User {
  name: String!
  email: String!
  password: String! # Terrible idea
  phone: String
  address: String
}
```

```
# Client
fragment LoginFields on User {
  email
  password
}

fragment ContactInfo on User {
  email
  phone
  address
}

query {
  user(id: "42") {
    name
    ...ContactInfo
  }
}
```

```
// Response
{
  "data": {
    "user": {
      "name": "Frodo Baggins",
      "email": "frodo@shire.me",
      "phone": "123456789",
      "address": "8 Bag-End, Shire"
    }
  }
}
```

Query Resolvers

```
# Schema
type Query {
  users: [User!]!
  user(id: ID!): User
}
```

```
// Backend
const resolvers = {
  Query: {

    users() {
      return users;
    },

    user(parent, args) {
      return users.find((user) => user.id === args.id);
    },

  },
};
```

Mutation Resolvers

```
# Schema
type Mutation {
  deletePost(id: ID!): ID!
  newPost(input: PostInput!): Post
}

input PostInput {
  body: String!
  image: String
}
```

```
// Backend
const resolvers = {
  Mutation: {

    deletePost(parent, args) {
      posts.splice(posts.findIndex((post) => post.id == args.id), 1);
      return args.id;
    },

    newPost(parent, args) {
      const post = {
        id: posts.length,
        body: args.input.body,
        image: args.input.image,
      };
      posts.push(post);
      return post;
    },

  },
};
```

Further resources

Where to go next?

- GraphQL
- Apollo
- Apollo Federation
- Mesh
- Roadmap.sh for GraphQL
- GitHub GraphQL API Explorer
- GraphQL Security Considerations
- Error Handling