

GraphQL

The basics and beyond

Wyatt Preul

- github.com/geek/graphql-kc
- github.com/geek
- jsgeek.com

Outline

- Introduction
- Schema/Query
- Server
- Mutation
- Client
- Testing
- Summary

Introduction

- Type system for describing data and API
- Query language for interacting with APIs
- Provides extra benefits:
 - Batched requests
 - Req/res validation
 - Usage tracking
 - Responses only contain data client requests

Specification / Libraries

- Facebook project started in 2012 -
- Open spec, free to use, no copyright worries
- GraphQL JS library for execution engine
- <https://github.com/facebook/graphql/>

Schema

```
type User {  
  id: ID!           // ! = required  
  email: String!  
  firstname: String  
  lastname: String  
}  
  
type Query {  
  getUser(email: String!): User  
}
```

Query

```
query {  
  getUser(email: "test@test.com") {  
    firstname  
    lastname  
  }  
}
```

Demo of schema/query

Server Responsibilities

- Load and processing of schema
- Must be able to execute queries

Server libraries

- hapi - Node.js framework for web apps
- graphi - GraphQL plugin for hapi
- [show usage in server1.js]



Colin Ihrig

Co-creator of graphi

Schema Enum

```
type Address {  
  lineone: String!  
  linetwo: String  
  city: String!  
  state: StateCode  
  zipcode: String  
}
```

```
enum StateCode {  
  MO  
  KS  
}
```

Sub query example

```
query {  
  getUser(email: "test@test.com") {  
    email  
    firstname  
    lastname  
    address { lineone }  
  }  
}
```

Batch query example

```
query {  
  user1: getUser(email: "test1@test.com") {  
    address {  
      lineone  
    }  
  }  
  user2: getUser(email: "test1@test.com") {  
    address {  
      lineone  
    }  
  }  
}
```

Variables example

```
query getUser($email1: String!, $email2: String!) {  
  user1: getUser(email: $email1) {  
    email  
    firstname  
    lastname  
  }  
  user2: getUser(email: $email2) {  
    email  
    firstname  
    lastname  
  }  
}
```

Handlers as resolvers

- Help migrate from REST
- Access the full request object
- Utilize hapi auth at a per handler/resolver
- [server3.js]

Mutations

- Same as a query from clients perspective
- Specified as a different type:
- Begin to use `Input` type

Mutation Schema

```
input UserInput {  
  email: String!  
  firstname: String!  
  lastname: String!  
}
```

```
type Mutation {  
  createUser(user: UserInput!): User  
}
```

Mutation Request

```
mutation {  
  createUser(user: {  
    email: "test@test.com"  
    firstname: "Foo"  
    lastname: "Bar"  
  }) {  
    id  
  }  
}
```

Client Requests

- No extra libraries required
- Can ask for only data that is needed
- See example under `client/store`

Testing

- test handlers like in REST
- validate schemas with graphql parse
- test queries with easygraphql-tester [see test/]

Summary

- GraphQL is a powerful addition to your toolset
- Supplement or replacement for REST
- Useful when multiple client teams with varying constraints
- Mature ecosystem and helpful community