REST is dead

GraphQL in Ruby

me: Piotr Szeremeta

piotr.szeremeta@swmansion.com

ruby, rails, elixir

REST

Method

URL - Uniform Resource Locator

GET /users/1/posts/1

```
{
  “title”: “First post”,
  “content”: “Hello world”
}
```

Resource

REST

- Resource driven
- Tied to HTTP
- One request - one resource
- One resource - whole resource

GraphQL

- Data driven
- Platform independent
- One request - anything you need
- Every resource in any shape

GraphQL

Single point of entry

POST /query

```
{
  post(id: 1, user_id: 1) {
    title
    content
    user { name }
  }
```

GraphQL

POST /query

```
{
  post(id: 1, user_id: 1) {
    title
    content
    user { name }
  }
```

GraphQL

POST /query

```
{
  post(id: 1, user_id: 1) {
    title
    content
    user { name }
  }
```

GraphQL

POST /query

```
{
  post(id: 1, user_id: 1) {
    title
    content
    user { name }
  }
```

GraphQL

POST /query

```
{
  post(id: 1, user_id: 1) {
    title
    content
    user { name }
  }
```

GraphQL

POST /query

```
{
  post(id: 1, user_id: 1) {
    title
    content
    user { name }
  }
```

```
{
  "data": {
    "post": [
      {
        "title": “Post title”,
        “content”: “Post content”
        “user”: {“name”: “Tom”}
      }
}
```

Boilerplate

```ruby
Rails.application.routes.draw do

  post "/query" ⇒ "page#query"
end
```

```ruby
class PageController < ApplicationController

  def query
    render json: Query.call(request_body, variables: params[:variables])
  end

  def request_body
    request.body.rewind
    request.body.read
  end
end
```

gem “graphql"

```
module Query

  QueryType = GraphQL::ObjectType.define do
    name "query"

    field :users do
      type types[types.Int]

      resolve →(obj, args, ctx) { □ }
    end
  end

  Schema = GraphQL::Schema.define do
    query QueryType
  end

  def self.call(graph_request, options = {})
    Schema.execute(graph_request, options)
  end
end
```

Object Type

```ruby
UserType = GraphQL::ObjectType.define do
  name "user"
  field :name, types.String
  field :description, types.String
  field :age, types.Int
end
```

Fields

```
QueryType = GraphQL::ObjectType.define do
  name "query"

  field :users do
    type types[UserType]

    resolve →(obj, args, ctx) { User.all }
  end

  field :user do
    type UserType

    argument :id, !types.Int

    resolve →(obj, args, ctx) { User.find(args["id"]) }
  end
end
```

Fields

POST /query

Request

```
{
  users { name description }
}
```

Response


```
{
  "data": {
    "users": [
      {
        "name": "Frank",
        "description": "StarWars fan"
      }
    ]
  }
}
```

Fields

POST /query

Request

```
{
  user(id: 1) { name description }
}
```

Response

```
{
  "data": {
    "user": {
      "name": "Frank",
      "description": "StarWars fan"
    }
  }
}
```

Fields

POST /query

Request

```
{
  users { age }
  current_user: user(id: 2) {
     name
     description
   }
}
```

Response

```
{
  "data": {
    "users": [
      {
        "age": 20
      }
    ],
    "current_user": {
      "name": "Frank",
      "description": "StarWars fan"
    }
  }
}
```

Methods

```
UserType = GraphQL::ObjectType.define do
  name "user"
  field :name, types.String
  field :description, types.String
  field :age, types.Int

  field :url do
    type types.String

    argument :user, types.String
    resolve ->(obj, args, ctx) {
      "http://localhost:3000/users/#{obj.id}/#{args[:user]}"
    }
  end
end
```

Methods

POST /query

```
{
  users {
    name
    description
    age
    url(user: "me")
    }
}
```

```
{
  "data": {
    "users": [
      {
        "name": "Frank",
        "description": "StarWars fan",
        "age": 20,
        "url": "http://localhost:3000/users/2/me"
      }
    ]
  }
}
```

Methods

```
{
  users {
    name
    description
    age
    url(user: 20)
    }
}
```

```
{
  "errors": [
    {
      "message": "Argument 'user' on Field 'url' has an
invalid value. Expected type 'String'.",
      "locations": [
        {
          "line": 6,
          "column": 5
        }
      ],
      "fields": [
        "query",
        "users",
        "url",
        "user"
      ]
    }
  ]
}
```

Subfields

```ruby
PostType = GraphQL::ObjectType.define do
  name "post"
  field :title, types.String
  field :content, types.String

  field :user, UserType
end

UserType = GraphQL::ObjectType.define do
  # ...

  field :posts, types[PostType]
end
```

```ruby
QueryType = GraphQL::ObjectType.define do
  name "query"

  field :posts do
    type types[PostType]

    resolve -> (obj, args, ctx) {
      Post.all
    }
  end
```

Subfields

```
{
  user(id: 2) {
    name
    description
    age
    posts {
      title
      user {
        name
      }
    }
  }
  posts {
    title
    user {
      name
    }
  }
}
```

```
{
  "data": {
    "user": {
      "name": "Frank",
      "description": "StarWars fan",
      "age": 20,
      "posts": [
        {
          "title": "Monday",
          "user": {
            "name": "Frank"
          }
        }
      ]
    },
    "posts": [
      {
        "title": "Monday",
        "user": {
          "name": "Frank"
        }
      },
      {
        "title": "Tuesday",
        "user": {
          "name": "Tom"
        }
      }
    ]
  }
}
```

Scopes

```ruby
class Post < ActiveRecord::Base
  belongs_to :user

  scope :liked, ->{ where(like: true) }
end
```

```ruby
UserType = GraphQL::ObjectType.define do
  #  ...
  #
  field :likedPosts do
    type types[PostType]

    resolve -> (obj, args, ctx) {
      obj.posts.merge(Post.liked)
    }
  end

  # ...
end
```

Scopes

```
{
  user(id: 2) {
    name
    description
    age
    likedPosts {
      title
    }
  }
}
```

```
{
  "data": {
    "user": {
      "name": "Frank",
      "description": "StarWars fan",
      "age": 20,
      "likedPosts": [
        {
          "title": "Monday"
        }
      ]
    }
  }
}
```

Mutations

```ruby
Schema = GraphQL::Schema.define do
  query QueryType
  mutation MutationType
end
```

```ruby
MutationType = GraphQL::ObjectType.define do
  name "mutation"

  field :createPost, field: CreatePostMutation.field
end
```

Mutations

```ruby
CreatePostMutation = GraphQL::Relay::Mutation.define do
  name "createPost"

  input_field :user_id, !types.ID
  input_field :title, !types.String
  input_field :content, !types.String

  return_field :post, ::Query::PostType

  resolve ->(obj, inputs, ctx) do
    user = User.find(inputs[:user_id])
    {post: user.posts.create(title: inputs[:title], content: inputs[:content])}
  end
end
```

Mutations

```
mutation CreatePost {
  createPost(input: {user_id: 2, title: "New Post", content: "New content"}){
     post {
       title
  }
}
```

```
{
  "data": {
    "createPost": {
      "post": {
        "title": "New Post"
      }
    }
  }
}
```

Mutations

```
mutation CreatePost {
  createPost(input: {user_id: 2, title: 1000, content: "New content"}){
     post {
       title
  }
}
```

```
{
  "errors": [
        ...
       {
     "message": "Argument 'title' on InputObject 'createPostInput' has
an invalid value. Expected type 'String!'.",
     "locations": [
       {
         "line": 3,
         "column": 23
       }
     ]
    }
  ]
}
```

Variables

```
def query
  render json: Query.call(request_body, variables: params)
end
```

Now we can send variables in parameters

Variables

```
mutation CreatePost($user_id: ID!, $title: String!, $content: String!){
  createPost(input: {user_id: $user_id, title: $title, content: $content}){
    post {
      title
    }
  }
}
```

```
{
  "data": {
    "createPost": {
      "post": {
        "title": "hello"
      }
    }
  }
}
```

Niceties

- Discoverability!
- Comments in queries!
- Code reuse!
- Conditional fields!
- Authorization!

Relay

- Declarative!
- Optimized queries!
- Automatic mutation handling!

Graphql 

We're hiring!

rekrutacja@swmansion.com

SOFTWARE MANSION

Thanks. Q&A time!