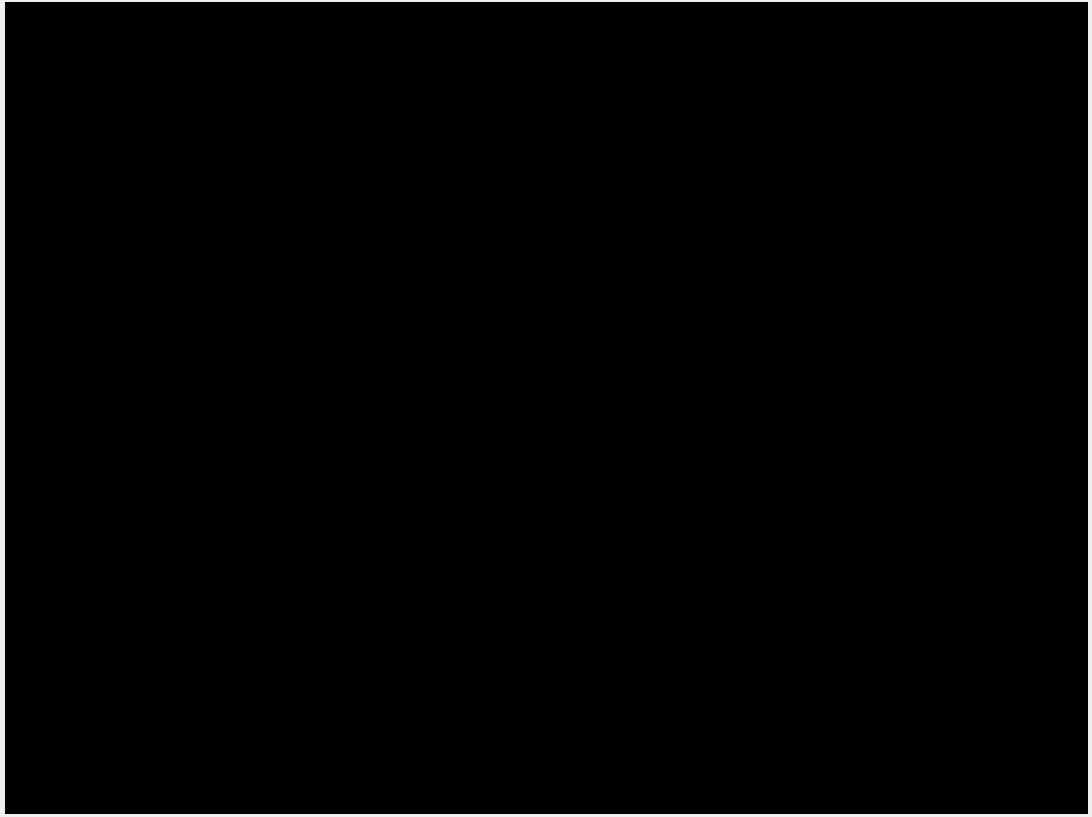
Building a Ruby GraphQL API: Awesome, Easy, Fast

Anna Headley and Trey Pendragon
Princeton University Library

Context of our project



Awesome: Why we chose GraphQL

Front-end developers were interested in using it

We knew we wanted an API based on a specification (assumed REST), which was going to be significant work either way so we might as well try something new

Single endpoint

GraphiQL in-browser REPL (read evaluate print loop)

Good Documentation

Some GraphQL Introduction

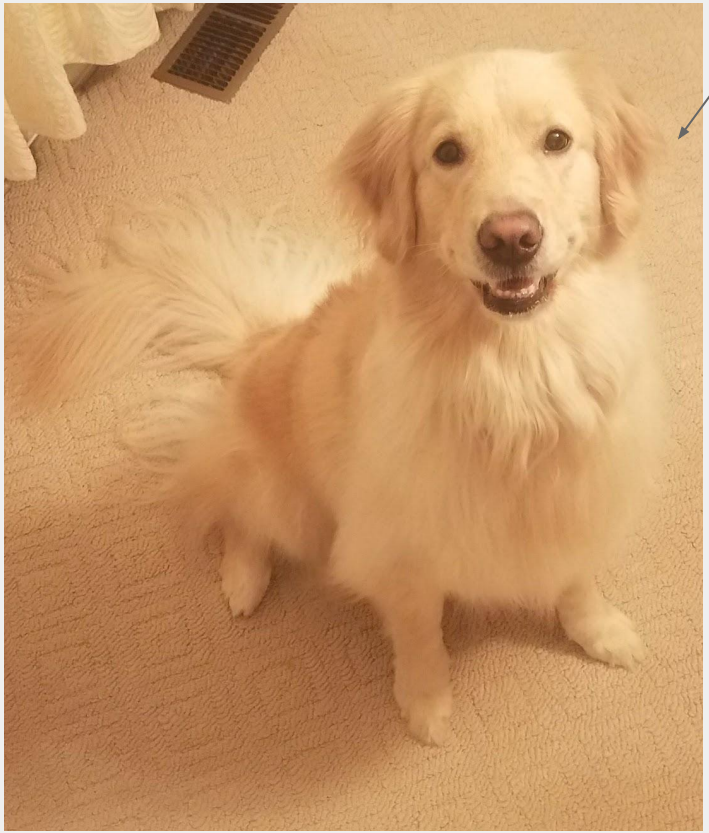
Queries

Types

Interfaces

(Inline) fragments

Mutations



GraphQL intro: Queries

"At its simplest, GraphQL is about asking for specific fields on objects."

```
{
resource(id: "c0304962-5b54-4581-ba44-64ce6cd7f9af") {
  id,
  label
  members {
    id,
    thumbnail {
      iiifServiceUrl
    }
  }
}
}
```

```
{
  "data": {
    "resource": {
      "id": "c0304962-5b54-4581-ba44-64ce6cd7f9af",
      "label": "Multi volume work",
      "members": [
        {
          "id": "9728541a-c370-45a9-99a3-1fa5d60d1e40",
          "thumbnail": {
            "iiifServiceUrl": "http://localhost:3000/image-service/310e38b3-9c1a-4647-8bc0-c1b3502d694b"
          }
        },
        {
          "id": "ecfc2f10-33a2-4477-ad29-a4640ad25841",
          "thumbnail": {
            "iiifServiceUrl": "http://localhost:3000/image-service/f8542dde-6d23-4e6a-a623-d6541502825a"
          }
        }
      ]
    }
  }
}
```

GraphQL intro: Schemas & Types

Resource **ScannedResource** ×

Search ScannedResource...

No Description

IMPLEMENTS

Resource

FIELDS

id: String
label: String
manifestUrl: String
members: [Resource!]
sourceMetadataIdentifier: String
startPage: String
thumbnail: Thumbnail
url: String
viewingDirection: ViewingDirectionEnum
viewingHint: String

Every GraphQL service defines a set of types which completely describe the set of possible data you can query on that service. Then, when queries come in, they are validated and executed against that schema.

GraphQL intro: Query is a type

Schema **Query** ×

Search Query...

The query root of this schema

FIELDS

resource(id: ID!): Resource

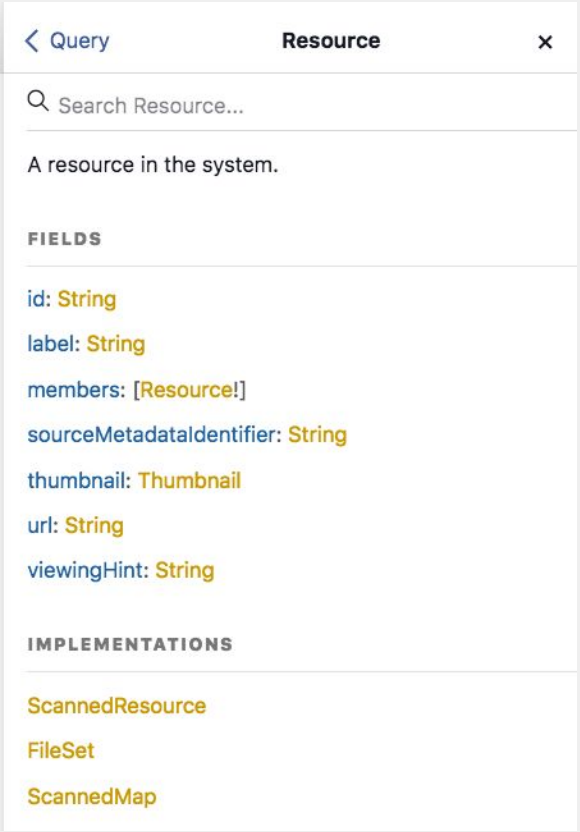
Find a resource by ID

resourcesByBibid(bibId: String!): [Resource!]

Find a resource by BibID

Query is a type, too. Its fields are the queries you defined for your endpoint.

GraphQL intro: Interfaces



Interfaces are just like they are in any object oriented context: a definition of fields another type has to provide in order to be considered implementations of the interface.

It's one way to allow more than one type to return from a query (there's also union types, which don't share common fields)

GraphQL intro: (inline) Fragments

```
{
resource(id: "c0304962-5b54-4581-ba44-64ce6cd7f9af") {
  id,
  label
  members {
    id
  }
  ... on ScannedResource {
    viewingDirection
  }
}
}
```


```
{
  "data": {
    "resource": {
      "id": "c0304962-5b54-4581-ba44-64ce6cd7f9af",
      "label": "Multi volume work",
      "members": [
        {
          "id": "9728541a-c370-45a9-99a3-1fa5d60d1e40"
        },
        {
          "id": "ecfc2f10-33a2-4477-ad29-a4640ad25841"
        }
      ],
      "viewingDirection": "LEFTTORIGHT"
    }
  }
}
```

GraphQL intro: Mutations

Mutation is a type just like query; its fields are the mutations you've created for your endpoint. Writing a mutation looks like this.

```
mutation {
  updateResource(input: {
    id: "c0304962-5b54-4581-ba44-64ce6cd7f9af",
    viewingDirection: BOTTOMTOTOP})
  {
    resource {
      id
      members {
        id
      }
      ... on ScannedResource {
        viewingDirection
      }
    }
  }
}
```

```
{
  "data": {
    "updateResource": {
      "resource": {
        "id": "c0304962-5b54-4581-ba44-64ce6cd7f9af",
        "members": [
          {
            "id": "9728541a-c370-45a9-99a3-1fa5d60d1e40"
          },
          {
            "id": "ecfc2f10-33a2-4477-ad29-a4640ad25841"
          }
        ],
        "viewingDirection": "BOTTOMTOTOP"
      }
    }
  }
}
```

Easy: Using the API

Fast: Ruby graphql library

https://github.com/rmosolgo/graphql-ruby

Installation

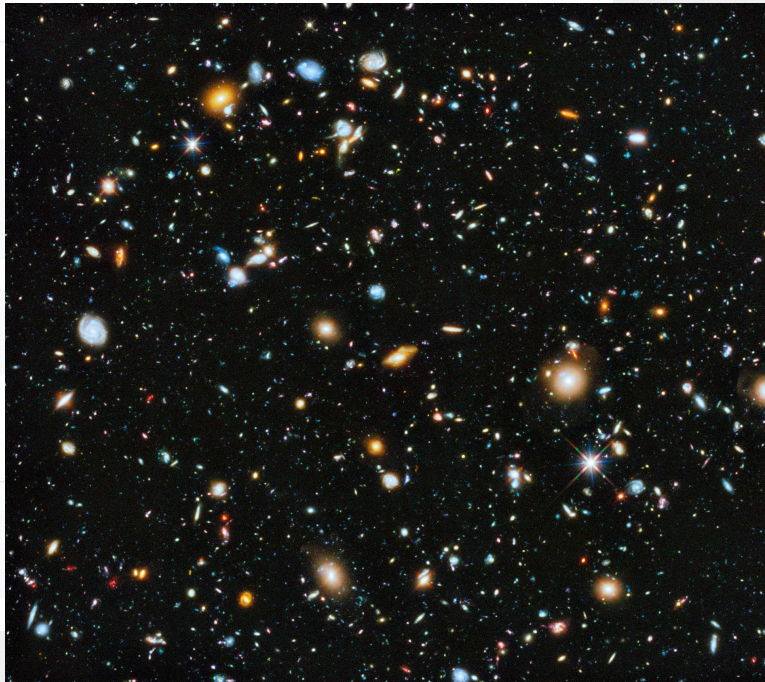
Install from RubyGems by adding it to your `Gemfile`, then bundling.

```
# Gemfile
gem 'graphql'
```

```
$ bundle install
```

Getting Started

```
$ rails generate graphql:install
```



Generated code: routes, controller

```
2   Rails.application.routes.draw do
3     if Rails.env.development?
4       mount GraphiQL::Rails::Engine, at: "/graphiql", graphql_path: "/graphql"
5     end
6
7     post "/graphql", to: "graphql#execute"
```

```
1  class GraphqlController < ApplicationController
2    def execute
3      variables = ensure_hash(params[:variables])
4      query = params[:query]
5      operation_name = params[:operationName]
6      context = {
7        # Query context goes here, for example:
8        # current_user: current_user,
9      }
10     result = FiggySchema.execute(query, variables: variables, context: context, operation_name: operation_name)
11     render json: result
12   end
```

Generated code: Queries, Mutations

```
1  class FiggySchema < GraphQL::Schema
2    mutation(Types::MutationType)
3    query(Types::QueryType)
4  end
```

```
1  class Types::QueryType < Types::BaseObject
2    # Add root-level fields here.
3    # They will be entry points for queries on your schema.
4
5    # TODO: remove me
6    field :test_field, String, null: false,
7      description: "An example field added by the generator"
8    def test_field
9      "Hello World!"
10   end
11 end
```

```
1  class Types::MutationType < Types::BaseObject
2    # TODO: remove me
3    field :test_field, String, null: false,
4      description: "An example field added by the generator"
5    def test_field
6      "Hello World"
7    end
8  end
```

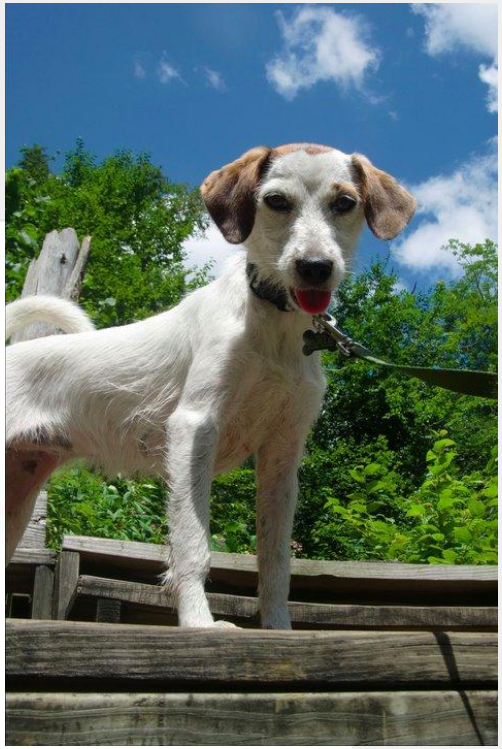
Our first query

```ruby
# frozen_string_literal: true
class Types::QueryType < Types::BaseObject
  description "The query root of this schema"

  # First describe the field signature:
  field :scanned_resource, ScannedResourceType, null: true do
    description "Find a Scanned Resource by ID"
    argument :id, ID, required: true
  end

  # Then provide an implementation:
  def scanned_resource(id:)
    query_service.find_by(id: id)
  end

```



Our first type

```
1  class Types::ScannedResourceType < Types::BaseObject
2    field :title, [String], null: true
3    field :viewing_hint, String, null: true
4
5    def viewing_hint
6      Array.wrap(super).first
7    end
8  end
```

Using an interface

```ruby
# frozen_string_literal: true
module Types::Resource
  include Types::BaseInterface
  description "A resource in the system."
  orphan_types Types::ScannedResourceType

  field :label, String, null: true
  field :viewing_hint, String, null: true

  definition_methods do
    def resolve_type(object, _context)
      "Types::#{object.class}Type".constantize
    end
  end
end
```

Using an interface, ct'd

```
1  1    # frozen_string_literal: true
2  2    class Types::ScannedResourceType < Types::BaseObject
3     -   field :label, String, null: true
4     -   field :viewing_hint, String, null: true
5     -
      3 +   implements Types::Resource
6  4       def viewing_hint
7  5         Array.wrap(super).first
8  6       end
```

Adding authentication

```ruby
class GraphqlController < ApplicationController
  protect_from_forgery with: :null_session
  def execute
    authorize! :read, :graphql
    variables = ensure_hash(params[:variables])
    query = params[:query]
    operation_name = params[:operationName]
    context = {
      ability: current_ability
    }
    result = FiggySchema.execute(query, variables: va
    render json: result
  end
```

```ruby
class Types::QueryType < Types::BaseObject
  description "The query root of this schema"

  field :resource, Types::Resource, null: true do
    description "Find a resource by ID"
    argument :id, ID, required: true
  end

  def resource(id:)
    resource = query_service.find_by(id: id)
    return unless ability.can? :read, resource
    resource
  end

  def ability
    context[:ability]
  end
```

Adding a mutation

```ruby
# frozen_string_literal: true
class Mutations::UpdateScannedResource < Mutations::BaseMutation
  delegate :query_service, :persister, to: :metadata_adapter
  null true

  argument :id, ID, required: true
  argument :viewing_hint, String, required: false

  field :scanned_resource, ::Types::ScannedResourceType, null: false
  field :errors, [String], null: true

  def resolve(id:, viewing_hint:)
    scanned_resource = query_service.find_by(id: id)
    change_set = DynamicChangeSet.new(scanned_resource).prepopulate!
    if change_set.validate(viewing_hint: viewing_hint)
      saved_resource = change_set_persister.save(change_set: change_set)
      {
        scanned_resource: saved_resource
      }
    else
      {
        scanned_resource: scanned_resource,
        errors: change_set.errors.full_messages
      }
    end
  end
```

Writing tests: unit tests

See http://graphql-ruby.org/schema/testing.html for good advice

We used rspec-graphql_matchers for unit testing type definitions

```
describe "class methods" do
  subject { described_class }

  # Note! These field names use a javascript-y camel-case variable style
  it { is_expected.to have_field(:viewingHint).of_type(String) }
  it { is_expected.to have_field(:label).of_type(String) }
end
```

(Regular unit tests on supporting methods for type behavior)

Writing tests: integration tests

```ruby
RSpec.describe FiggySchema do
  # You can override `context` or `variables` in
  # more specific scopes
  let(:context) { {} }
  let(:variables) { {} }
  # Call `result` to execute the query
  let(:result) do
    res = described_class.execute(
      query_string,
      context: context,
      variables: variables
    )
    # Print any errors
    pp res if res["errors"]
    res
  end

  describe "a specific query" do
    # provide a query string for `result`
    let(:scanned_resource) { FactoryBot.create_for_repository(:scanned_resource, viewing_hint: "individuals") }
    let(:id) { scanned_resource.id }
    let(:query_string) { %|{ scannedResource(id: "#{id}") { viewingHint } }| }

    it "returns a viewing hint" do
      # calling `result` executes the query
      expect(result["data"]["scannedResource"]["viewingHint"]).to eq("individuals")
    end
  end
end
```

Tests gotcha!

Field names are snake_case in the type definition, camelCase in the test

```
field :viewing_hint, String, null: true
```

```
# Note! These field names use a javascript-y camel-case variable style
it { is_expected.to have_field(:viewingHint).of_type(String) }
```

Finishing touches

- Creating types for all our different resource models
- Actually implementing the new front end
    - Note we've completely skipped actual front-end implementation details, that work was done by another developer in vue.js using a client-side graphql library called apollo.

New use case

Our catalog needs to find thumbnails based on the bib id.

- During a meeting about this use case, had a moment where we said “huh, our GraphQL endpoint is 95% there.”
- Meant opening the endpoint itself since our previous use case was internal to the site -- security implications? So far we don't have any hugely resource-intensive queries.

Final thoughts

We were able to get up and running enough for the front-end dev to start implementing in 3 days.

Front-end dev could rely on GraphQL documentation for that implementation

It's easy to expand the API as needed.

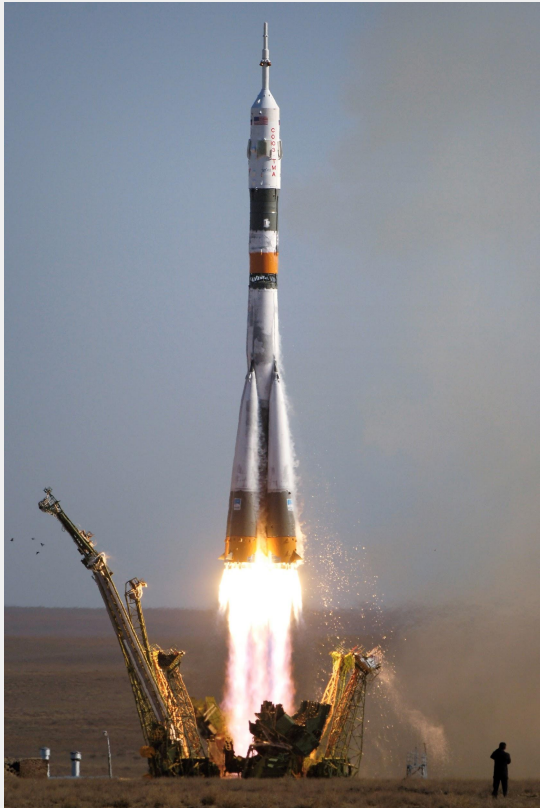
We still love GraphiQL

Bound-withs work now

Nice to be using IIIF for what it's good at instead of trying to use it as a CRUD API.

Thank you!

samvera



Things that are awesome (picture credits)

outer space: https://en.wikipedia.org/wiki/Universe

volcanoes: https://en.wikipedia.org/wiki/Hawaii_hotspot#/media/File:Puu_oo.jpg

fractals: https://commons.wikimedia.org/wiki/File:Romanesco_Broccoli_detail_-_(1).jpg#/media/File:Romanesco_Broccoli_detail_-_(1).jpg

treehouses: https://www.flickr.com/photos/127478577@N02/16968477139

rainbows: https://en.wikipedia.org/wiki/File:Full_featured_double_rainbow_at_Savonlinna_1000px.jpg

more things that are awesome

ghost ships:
https://www.flickr.com/photos/ishtaure-dawn/14133707200/in/photostream/

kittens: https://www.flickr.com/photos/stignygaard/18858761288

my dog: https://drive.google.com/open?id=0ByzpjkVnVKMMYXFTSVctTnRzNVk

MY dog

mario speed runs:
https://thumbs.gfycat.com/ConcreteUnrulyAmericanwirehair-size_restricted.gif

more things that are awesome

samvera community:
https://wiki.duraspace.org/display/samvera/Logos+for+presentations+and+articles

being in the woods:
https://drive.google.com/open?id=12UrUGDTgFXXj8nq2f_RikyvMqMn6IvdJ

helping people:
https://drive.google.com/open?id=1WIFjShVL9uvubS5Hpu2GpmeZjmFATTgu

animals that glow:
https://en.wikipedia.org/wiki/Siphonophorae#/media/File:Marrus_orthocanna_crop.jpg

more things that are awesome

Ice sculptures: https://www.geograph.org.uk/photo/5641274

Rockets:
https://upload.wikimedia.org/wikipedia/commons/9/9a/Soyuz_TMA-9_launch.jpg

Castles:
https://upload.wikimedia.org/wikipedia/commons/a/ae/Castle_Neuschwanstein.jpg

Sphinx:
https://www.maxpixel.net/static/photo/2x/Sphinx-Pyramid-Egypt-2987112.jpg