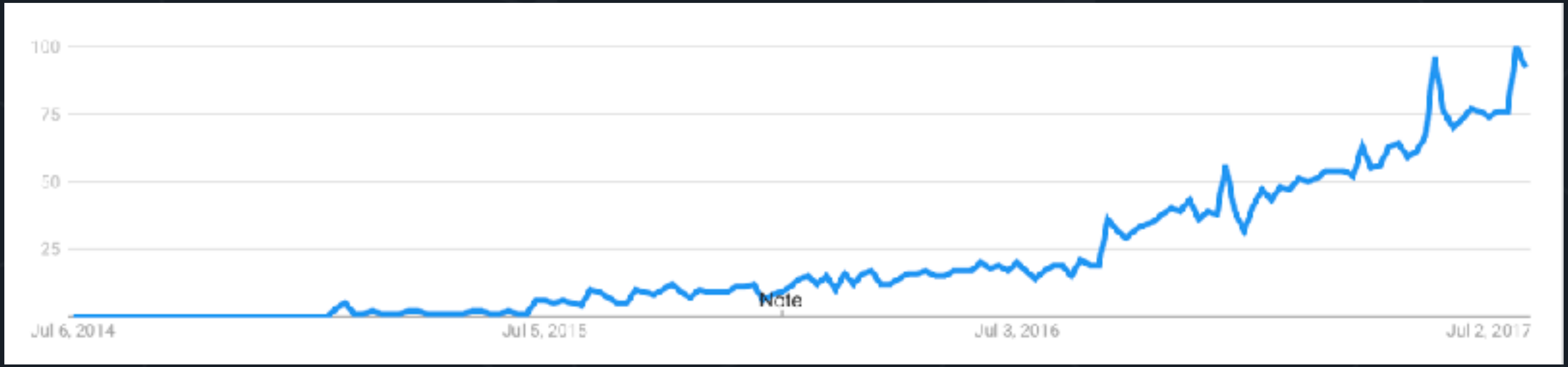
GraphQL and You

{'Nic Ollis' => @nic_ollis}

get ‘graphql/about’ => ‘graphql#about’

- Easy to pickup, Easy to Read
- Query your data
- Get back what you need
- Easy to extend



…but before we start

Whats so wrong with REST?

Nothing

get ‘REST/:info’ => ‘rest#info’

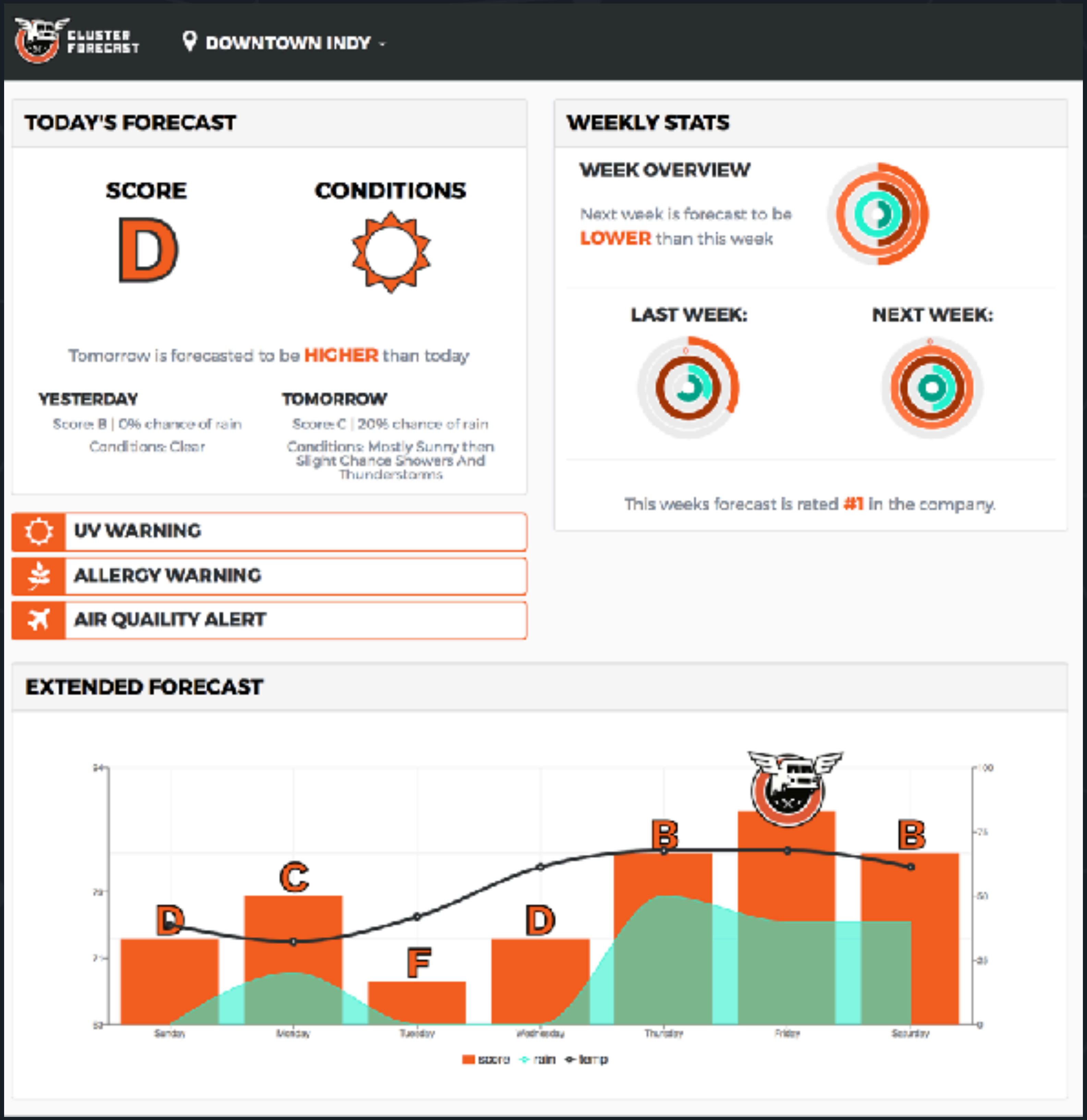
- REST is an architectural concept for network-based software
- Utilizes the uniform interface of the protocols it exist in.
- One main focus of REST is hypermedia controls
  - (see. HATEOAS)
    - GraphQL is a query language, specification, and collection of tools
    - GraphQL invents its own conventions
    - Not using hypermedia controls? GraphQL could be a more relevant

Who’s Using GraphQL?

http://graphql.org/users/

Lets take a look…

A simple UI



TODAY'S FORECAST

SCORE D

CONDITIONS

Tomorrow is forecasted to be **HIGHER** than today

YESTERDAY
Score: B | 0% chance of rain
Conditions: Clear

TOMORROW
Score: C | 20% chance of rain
Conditions: Mostly Sunny then Slight Chance Showers And Thunderstorms

UV WARNING

ALLERGY WARNING

AIR QUAILITY ALERT

WEEKLY STATS

WEEK OVERVIEW

Next week is forecast to be **LOWER** than this week

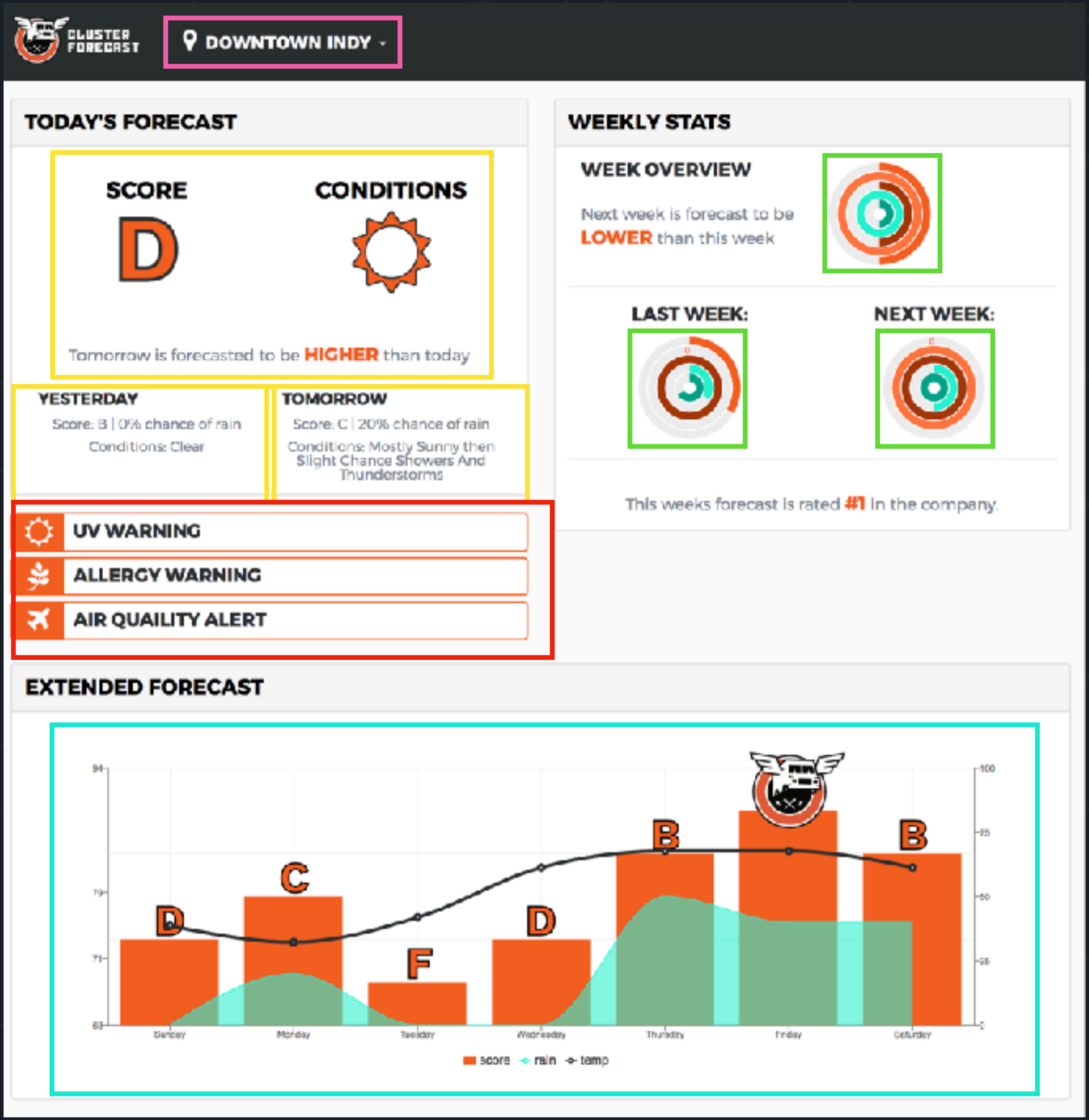
LAST WEEK:

NEXT WEEK:

This weeks forecast is rated **#1** in the company.

EXTENDED FORECAST

What kind of data is needed?



TODAY'S FORECAST

SCORE

D

CONDITIONS

Tomorrow is forecasted to be **HIGHER** than today

YESTERDAY

Score: B | 0% chance of rain

Conditions: Clear

TOMORROW

Score: C | 20% chance of rain

Conditions: Mostly Sunny then Slight Chance Showers And Thunderstorms

UV WARNING

ALLERGY WARNING

AIR QUAILITY ALERT

WEEKLY STATS

WEEK OVERVIEW

Next week is forecast to be **LOWER** than this week

LAST WEEK:

NEXT WEEK:

This weeks forecast is rated **#1** in the company.

EXTENDED FORECAST

Location

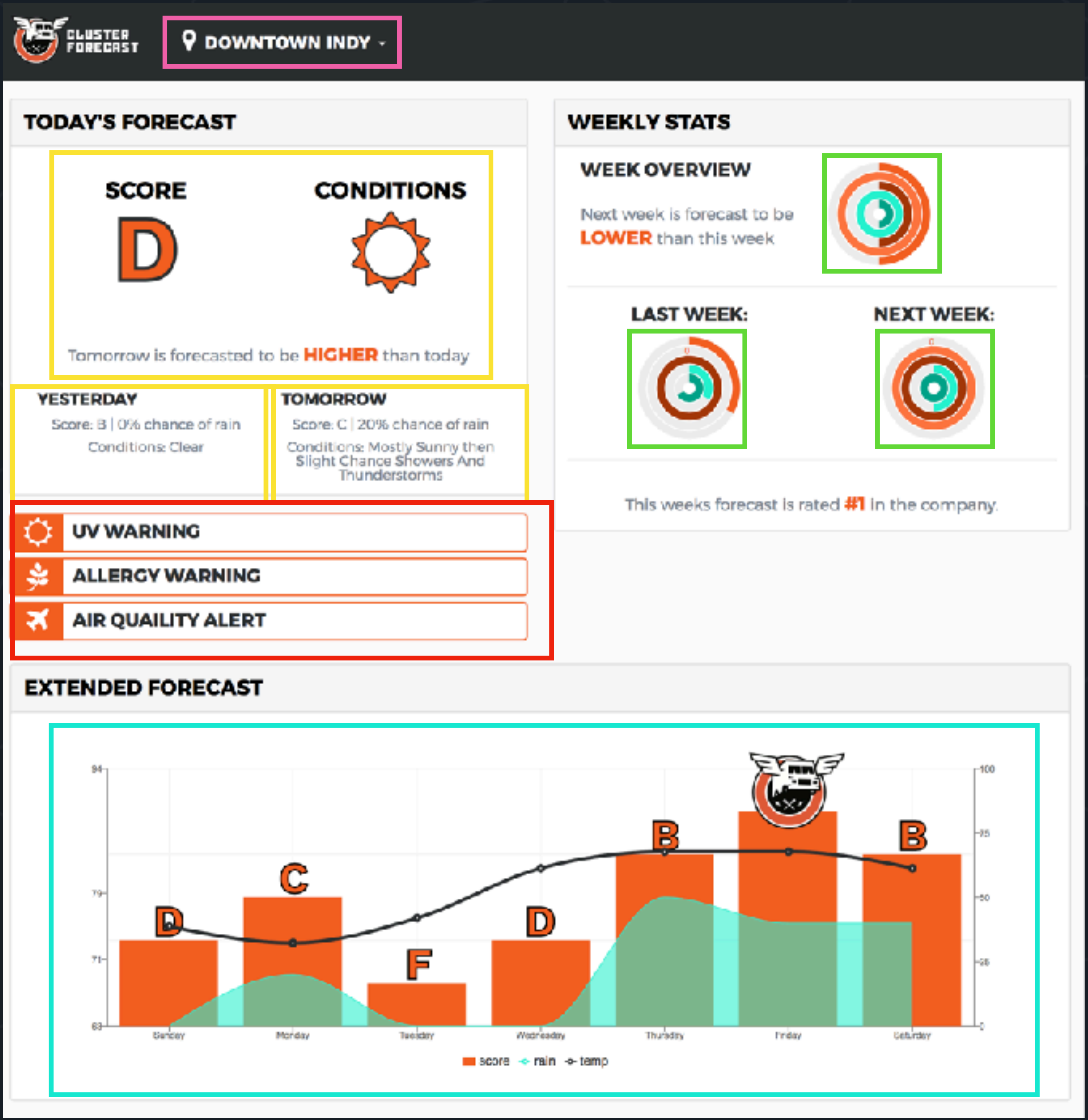
Daily Summaries

Weekly Summaries

Alerts

Extended Forecast

RESTify



CLUSTER FORECAST

DOWNTOWN INDY

TODAY'S FORECAST

SCORE

D

CONDITIONS

Tomorrow is forecasted to be **HIGHER** than today

YESTERDAY

Score: B | 0% chance of rain

Conditions: Clear

TOMORROW

Score: C | 20% chance of rain

Conditions: Mostly Sunny then Slight Chance Showers And Thunderstorms

UV WARNING

ALLERGY WARNING

AIR QUAILITY ALERT

WEEKLY STATS

WEEK OVERVIEW

Next week is forecast to be **LOWER** than this week

LAST WEEK:

NEXT WEEK:

This weeks forecast is rated **#1** in the company.

EXTENDED FORECAST

/locations

/daily/07242017

/daily/07232017

/daily/07252017

/weekly/26

/weekly/25

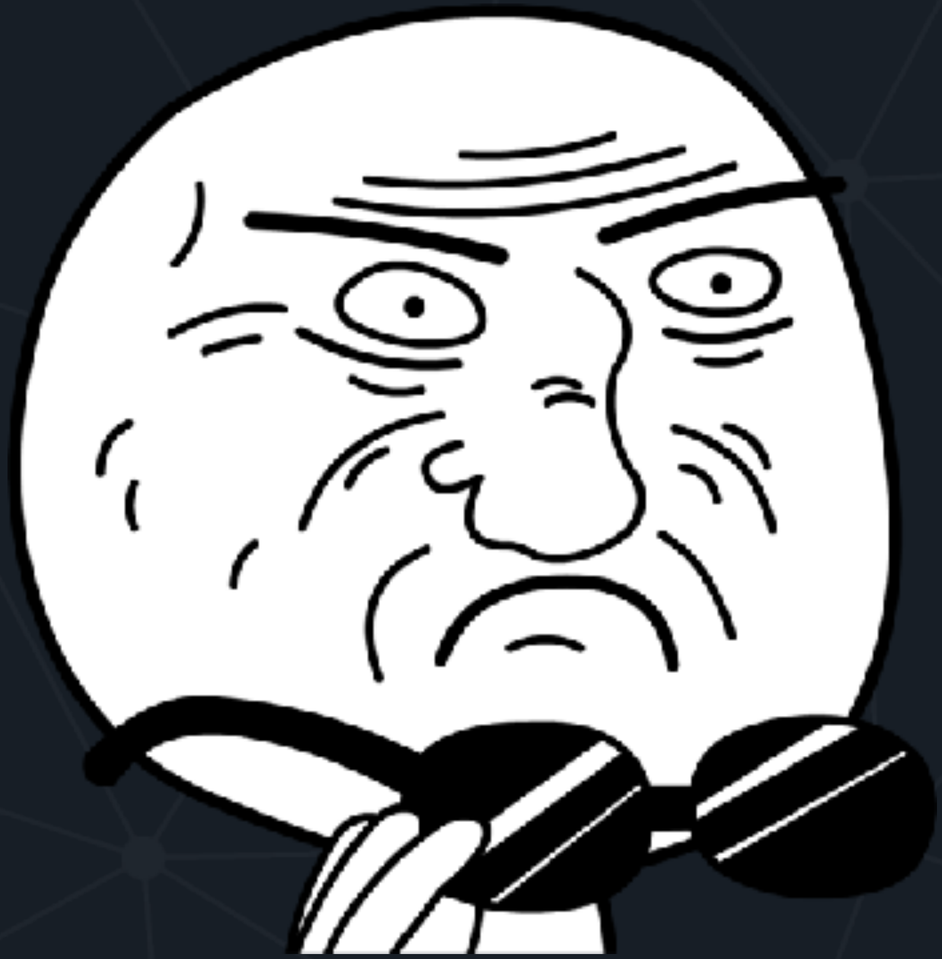
/weekly/27

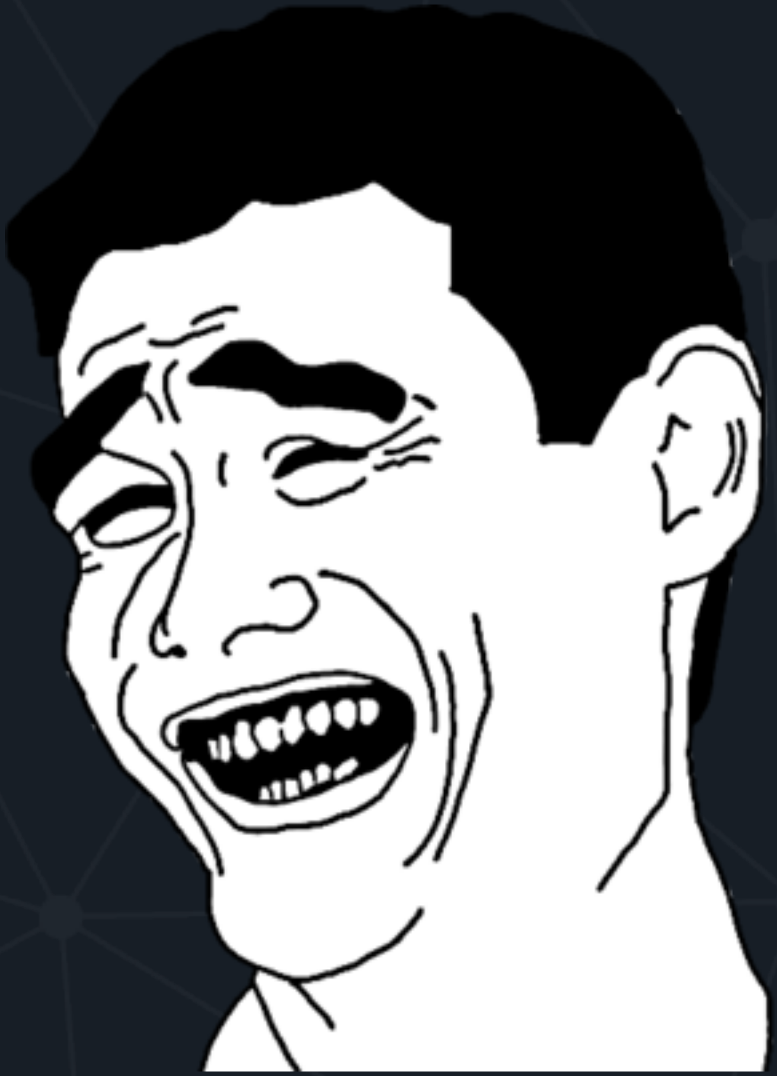
/alerts

/forecast/10days

$$1 + (8 * n)$$

*n = number of locations





Custom Endpoint’ify

/all_the_weather_and_location_data

/all_the_weather_and_location_data

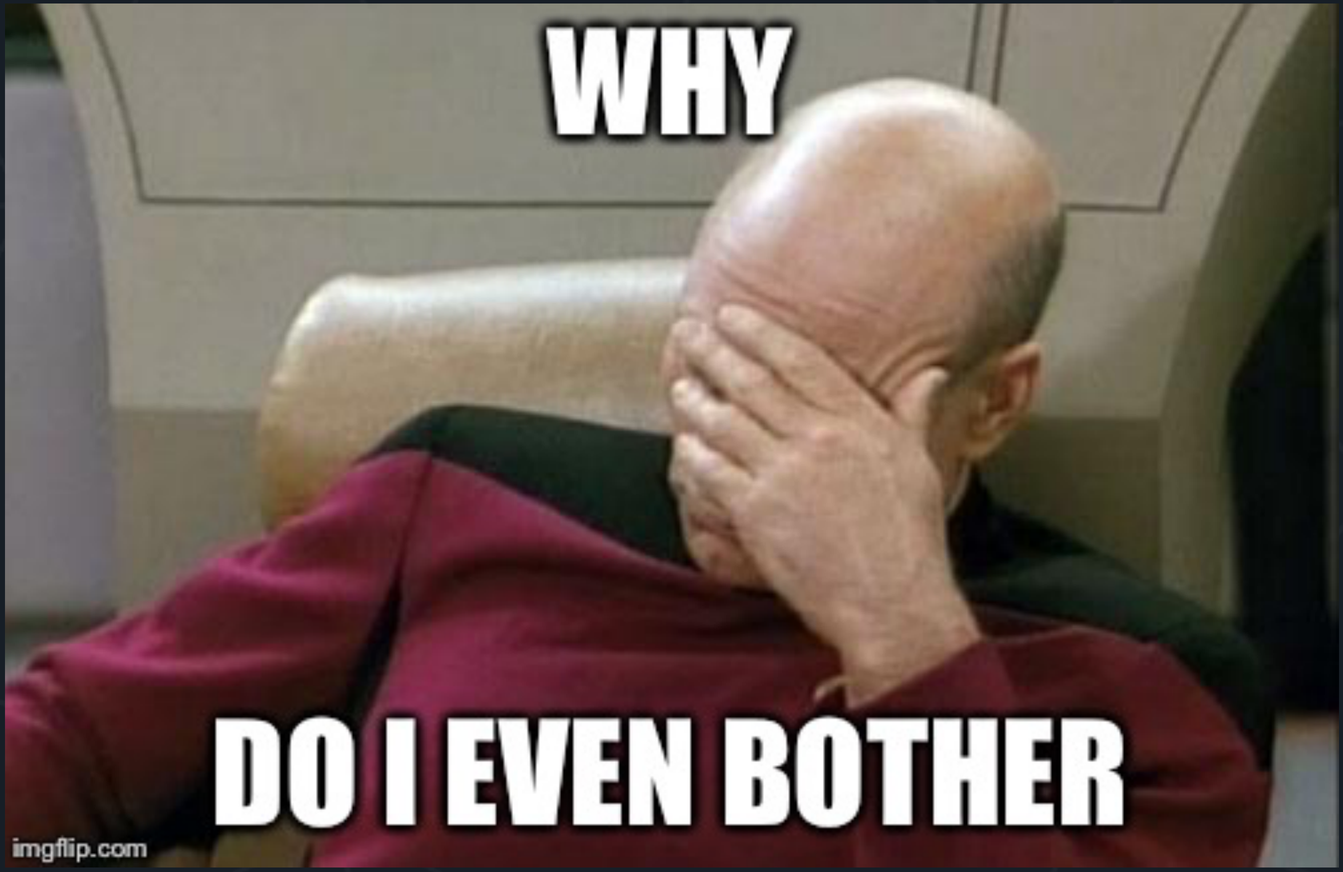
/just_the_location_data

/all_the_weather_data_but_7_day_forecast

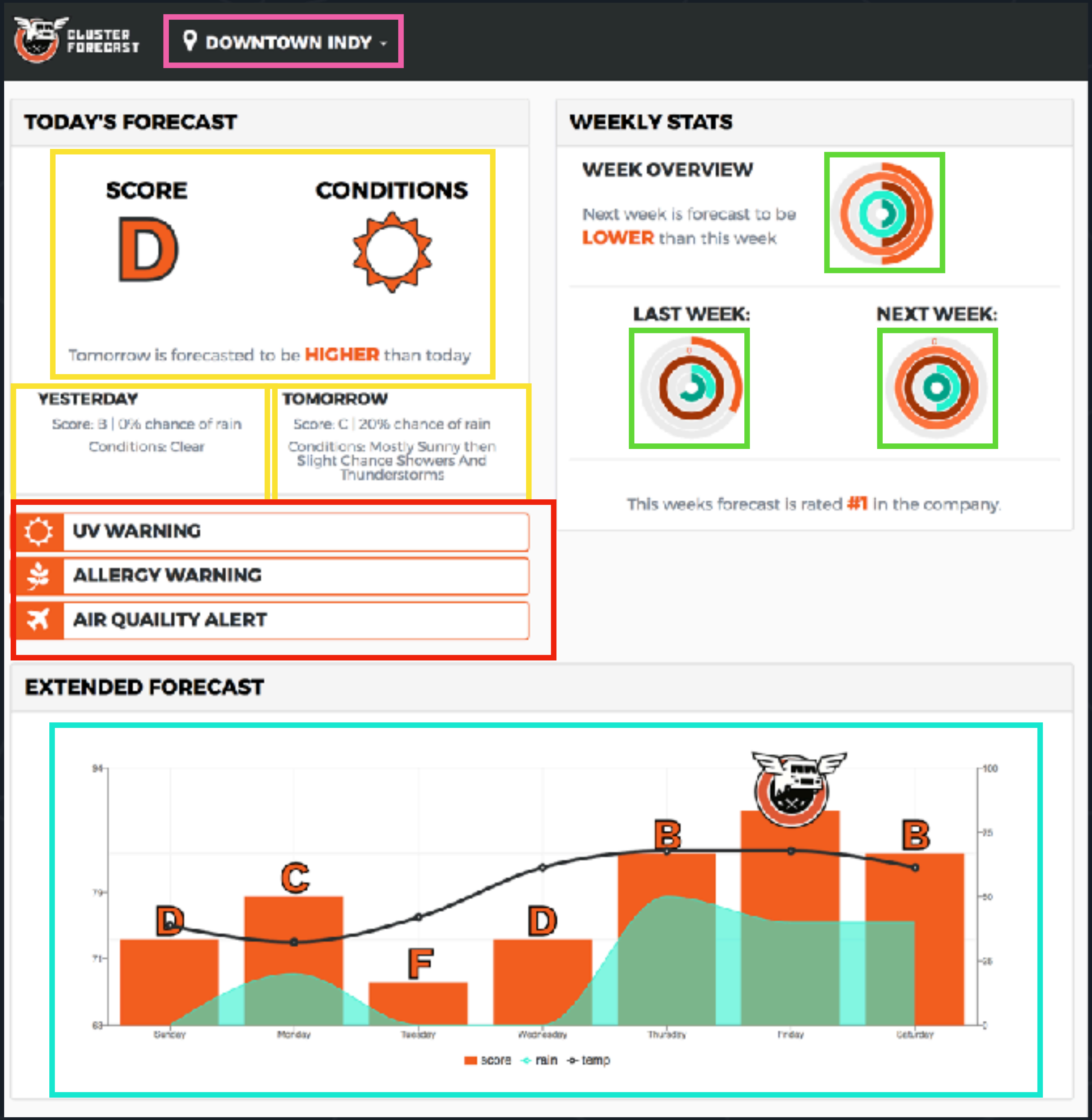
/all_the_weather_data_but_no_history

/all_the_weather_data_for_mobile

/all_the_things_v2


WHY
DO I EVEN BOTHER
imgflip.com

ify



TODAY'S FORECAST

SCORE

D

CONDITIONS

Tomorrow is forecasted to be **HIGHER** than today

YESTERDAY

Score: B | 0% chance of rain

Conditions: Clear

TOMORROW

Score: C | 20% chance of rain

Conditions: Mostly Sunny then Slight Chance Showers And Thunderstorms

UV WARNING

ALLERGY WARNING

AIR QUAILITY ALERT

WEEKLY STATS

WEEK OVERVIEW

Next week is forecast to be **LOWER** than this week

LAST WEEK:

NEXT WEEK:

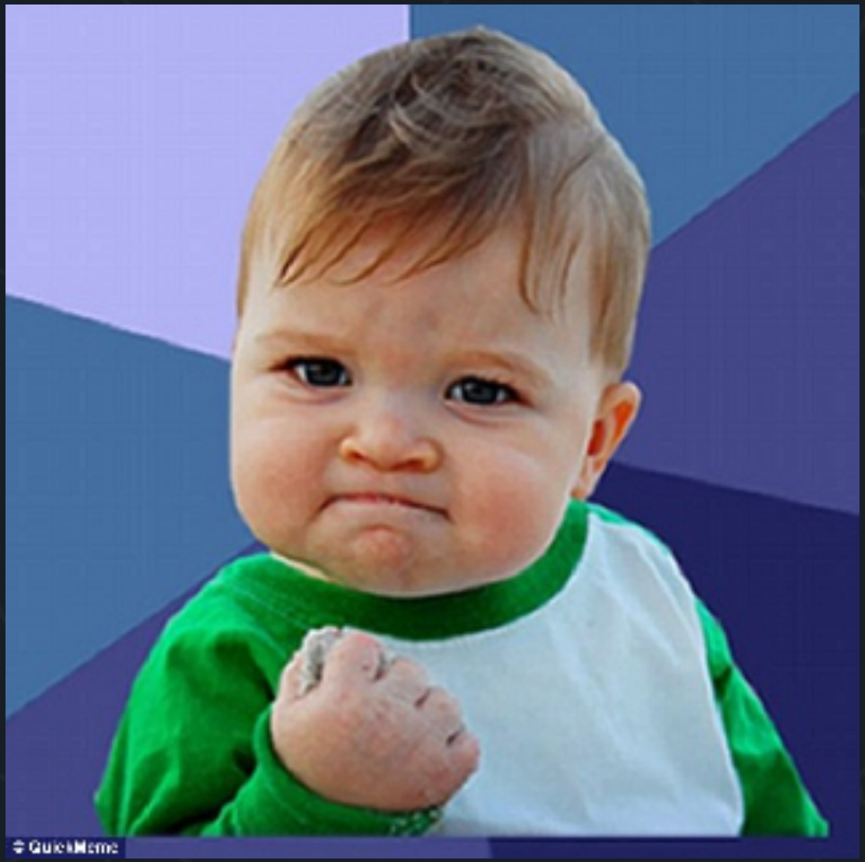
This weeks forecast is rated **#1** in the company.

EXTENDED FORECAST

```
{
 weather(location: "Indy") {
  daily(date: today.date) {
   score,
   condition,
   chance_of_rain
  }
  weekly(week: this.week) {
    weekly_scores
  }
  alerts
  forecast(length: 10)
 }
}
```

$1 + n$

*n = number of locations


QuickMeme

Describe your data

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for what you want

```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

Get predictable results

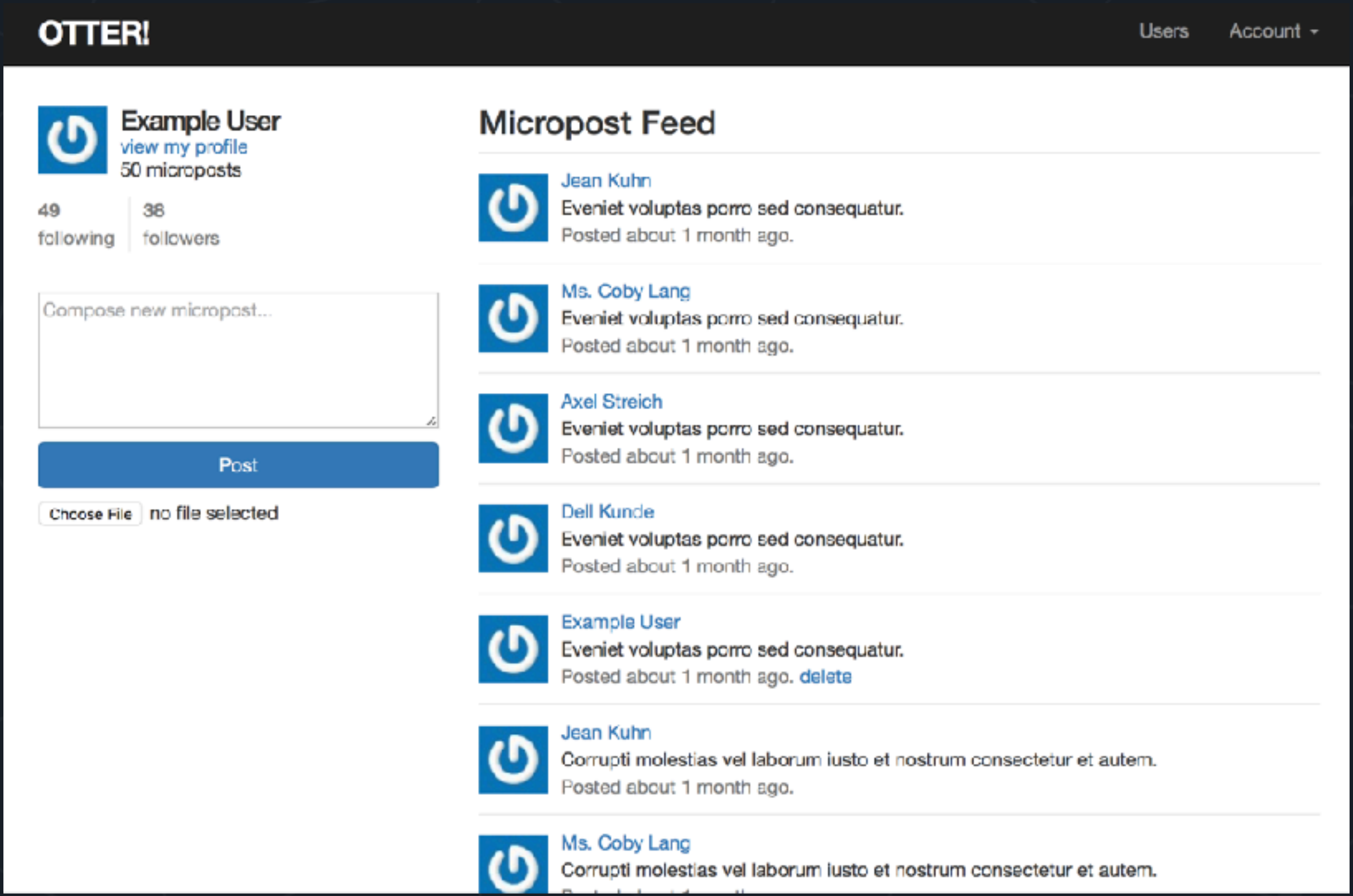
```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

get ‘graphql/avoid’ => ‘graphql/avoid’

- GraphQL does not play nice with the rest of the web, because it treats HTTP as a dump pipe.
- GraphQL—without a custom implementation—will make any caching layer too specific, and thus mostly useless.

GraphQL on Rails

Adding to an Existing Project



Example User
view my profile
50 microposts

49 following | 38 followers

no file selected

Micropost Feed

Jean Kuhn
Eveniet voluptas porro sed consequatur.
Posted about 1 month ago.

Ms. Coby Lang
Eveniet voluptas porro sed consequatur.
Posted about 1 month ago.

Axel Streich
Eveniet voluptas porro sed consequatur.
Posted about 1 month ago.

Dell Kunde
Eveniet voluptas porro sed consequatur.
Posted about 1 month ago.

Example User
Eveniet voluptas porro sed consequatur.
Posted about 1 month ago. delete

Jean Kuhn
Corrupti molestias vel laborum iusto et nostrum consectetur et autem.
Posted about 1 month ago.

Ms. Coby Lang
Corrupti molestias vel laborum iusto et nostrum consectetur et autem.

http://bit.ly/indy_rb_otter

Add a Gem

```ruby
# API
gem 'graphql'
```

Run

```
Rails g graphql:install
```

creating our user description

```ruby
UserType = GraphQL::ObjectType.define do
  name "User"
    description "A User"
    field :id, types.ID
    field :name, types.String
    field :email, types.String
    field :followers do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.followers.limit(args[:size])
      }
    end
    field :following do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.following.limit(args[:size])
      }
    end
end
```

app/types/user_type.rb

```ruby
UserType = GraphQL::ObjectType.define do
  name "User"
    description "A User"
    field :id, types.ID
    field :name, types.String
    field :email, types.String
    field :followers do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.followers.limit(args[:size])
      }
    end
    field :following do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.following.limit(args[:size])
      }
    end
end
```

app/types/user_type.rb

```ruby
UserType = GraphQL::ObjectType.define do
  name "User"
    description "A User"
    field :id, types.ID
    field :name, types.String
    field :email, types.String
    field :followers do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.followers.limit(args[:size])
      }
    end
    field :following do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.following.limit(args[:size])
      }
    end
end
```

app/types/user_type.rb

```ruby
UserType = GraphQL::ObjectType.define do
  name "User"
    description "A User"
    field :id, types.ID
    field :name, types.String
    field :email, types.String
    field :followers do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.followers.limit(args[:size])
      }
    end
    field :following do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.following.limit(args[:size])
      }
    end
end
```

app/types/user_type.rb

```
UserType = GraphQL::ObjectType.define do
  name "User"
    description "A User"
    field :id, types.ID
    field :name, types.String
    field :email, types.String
    field :followers do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.followers.limit(args[:size])
      }
    end
    field :following do
      type types[UserType]
      argument :size, types.Int
      resolve -> (user, args, ctx) {
        user.following.limit(args[:size])
      }
    end
end
```

app/types/user_type.rb

building out a schema

```ruby
QueryType = GraphQL::ObjectType.define do
  name "Query"
  description "The query root for this schema"

  field :user do
    type UserType
    argument :id, !types.ID
    resolve -> (obj, args, ctx) {
      User.find(args[:id])
    }
  end
end
```

app/types/query_type.rb

```ruby
QueryType = GraphQL::ObjectType.define do
  name "Query"
  description "The query root for this schema"

  field :user do
    type UserType
    argument :id, !types.ID
    resolve -> (obj, args, ctx) {
      User.find(args[:id])
    }
  end
end
```

app/types/query_type.rb

```ruby
QueryType = GraphQL::ObjectType.define do
  name "Query"
  description "The query root for this schema"

  field :user do
    type UserType
    argument :id, !types.ID
    resolve -> (obj, args, ctx) {
      User.find(args[:id])
    }
  end
end
```

app/types/query_type.rb

hooking in our QueryType

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

app/types/schema.rb

building a controller

```ruby
class GraphqlController < ApplicationController
  def execute
    variables = ensure_hash(params[:variables])
    query = params[:query]
    operation_name = params[:operationName]
    context = {
      # Query context goes here, for example:
      # current_user: current_user,
    }
    result = OtterSchema.execute(query, variables:
    variables, context: context, operation_name:
    operation_name)
    render json: result
  end

  private

  # Handle form data, JSON body, or a blank value
  def ensure_hash(ambiguous_param)
    case ambiguous_param
    when String
      if ambiguous_param.present?
        ensure_hash(JSON.parse(ambiguous_param))
      else
        {}
      end
    when Hash, ActionController::Parameters
      ambiguous_param
    when nil
      {}
    else
      raise ArgumentError, "Unexpected parameter: #{
      ambiguous_param}"
    end
  end
end
```

add to our routes

```
# API
07    post "/graphql", to: "graphql#execute"
```

running a query

localhost:3000/graphql

```
query {
  user(id:1){
    name
    email
    followers {
      email
    }
  }
}
```

```json
{
    "data": {
        "user": {
            "name": "Example User",
            "email": "example@ollis.me",
            "followers": [
                {
                    "email": "example-3@ollis.me"
                },
                {
                    "email": "example-4@ollis.me"
                },
                {…}
```

TIRED.
JUST TIRED.
quickmeme.com

http://graphql-ruby.org

http://graphql.org

Why GraphQL?

- One endpoint to access your data
- Retrieve only the data your client needs in a single request (flexibility)
- No need to tailor endpoints for your views
- No versioning

Thank you!

```
{
  name: ‘Nic Ollis’,
  web: ‘ollis.me’,
  twitter: @nic_ollis,
}
```