GRAPHQL

Stanko Krtalic Rusendic

github.com/Stankec
@monorkin


Floating
Point

01

REST PROBLEMS

Representational state transfer

1 MINUTE AGO

STANKO

Hi

OTTO

STANKO

Cool!

17:56 · Delivered

OTTO

TYPE A MESSAGE

GIF

IDEA

●○○○○ bonbon 17:57 84%

OTTO THE BOT

1 MINUTE AGO

STANKO

Hi

OTTO

STANKO

Cool!

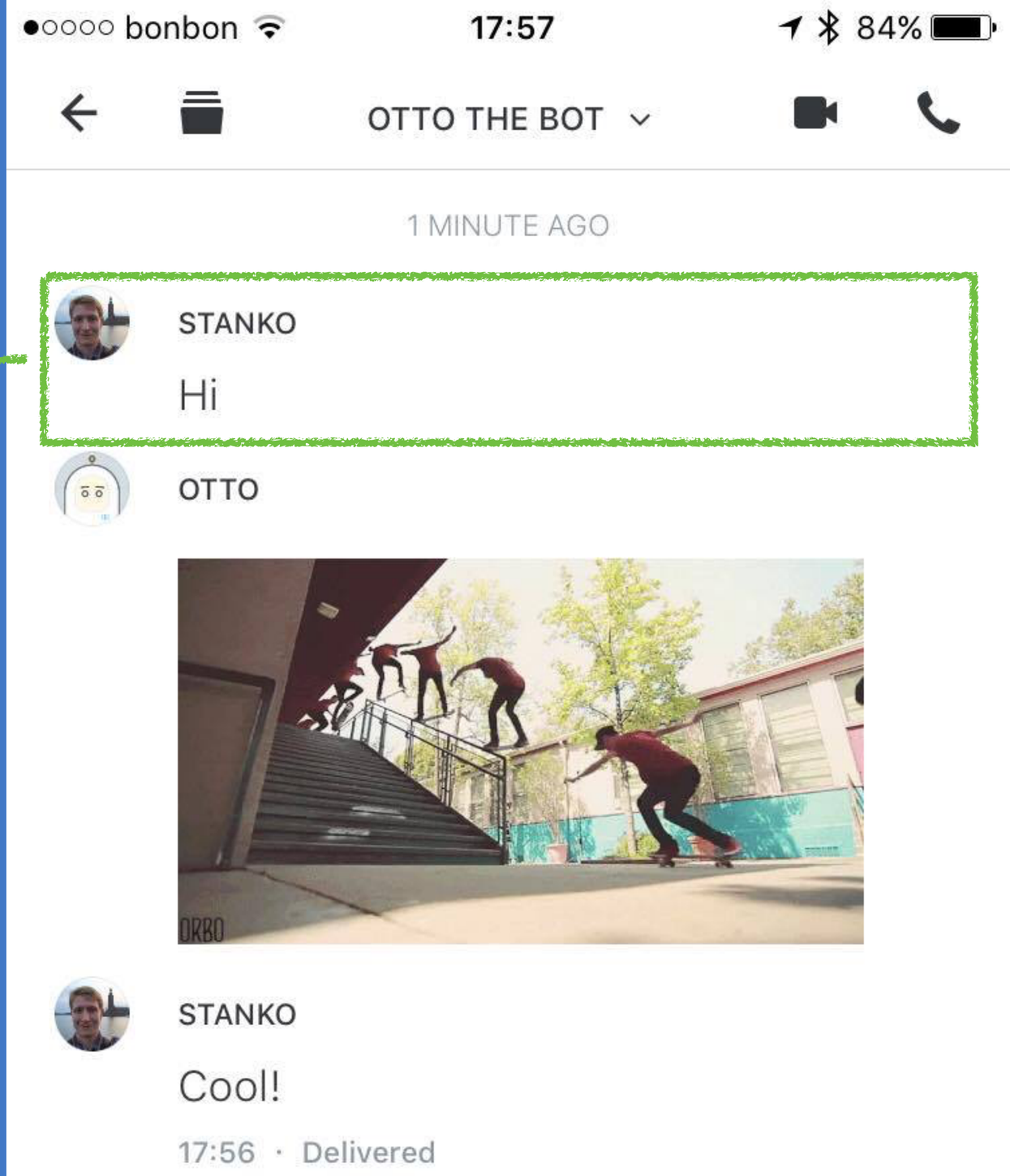
17:56 · Delivered

/api/conversations/53

```
{
  name: “Otto the bot”,
  participant_user_ids: [1, 1337],
  last_message_sent_at: 1337002,
  archived: false
}
```

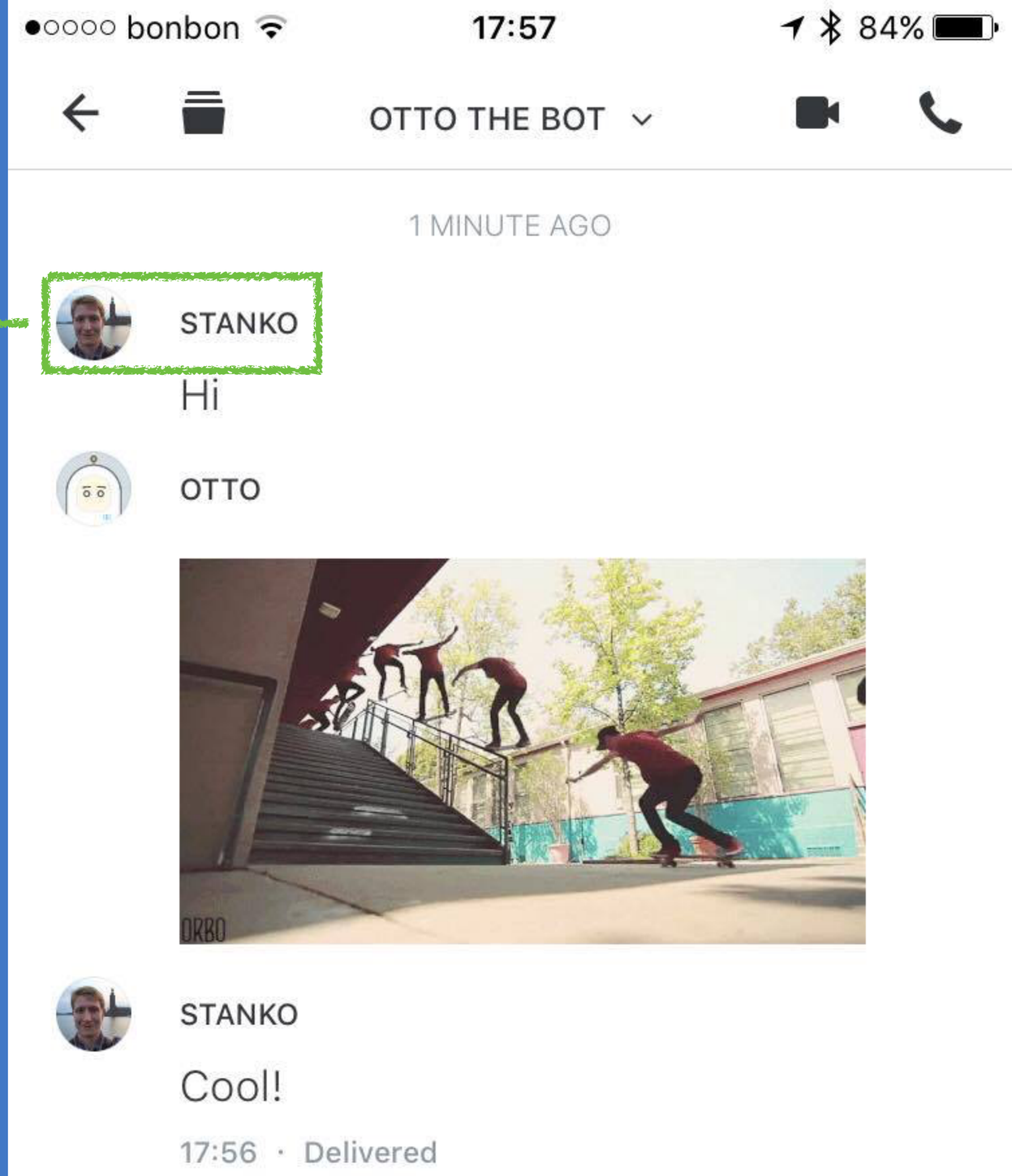
/api/conversations/53/message/1

```
{
  message_type: “plain”,
  body: “Hi”,
  sender_id: “1337”,
  created_at: 1337001,
  updated_at: 1337001,
  previous_version_ids: [],
  status: “delivered”,
  seen_by_participant_ids: [1]
}
```

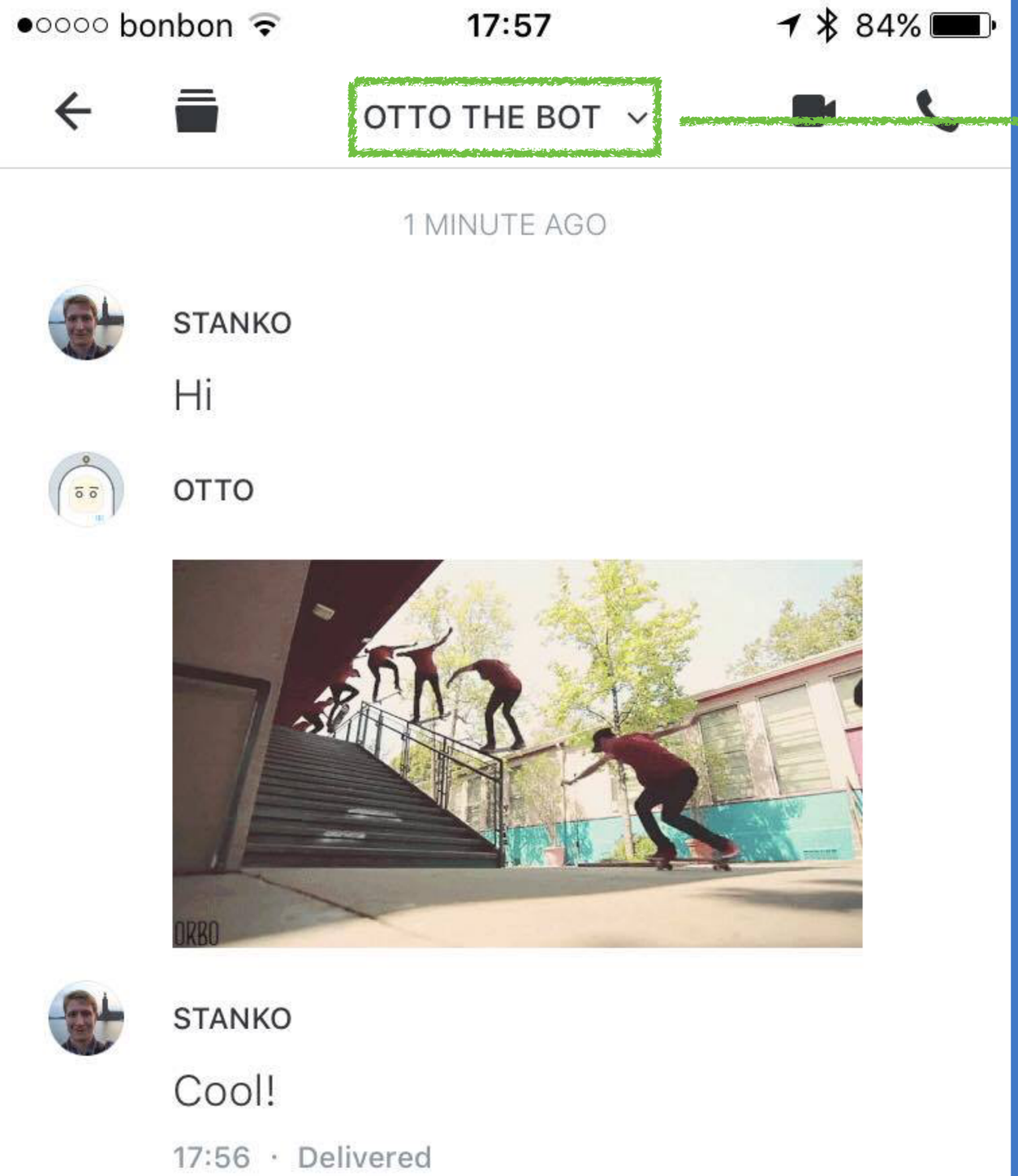


/api/users/1337

```
{
  first_name: “Stanko”,
  last_name: “Krtalic Rusendic”,
  avatar_url: “https://…”,
  last_online_at: 1337001,
  last_signin_ip: 133.10.45.99,
  sex: “male”,
  timezone: “GMT+1”,
  last_known_location: “16.0E45.0N”
}
```



REALITY

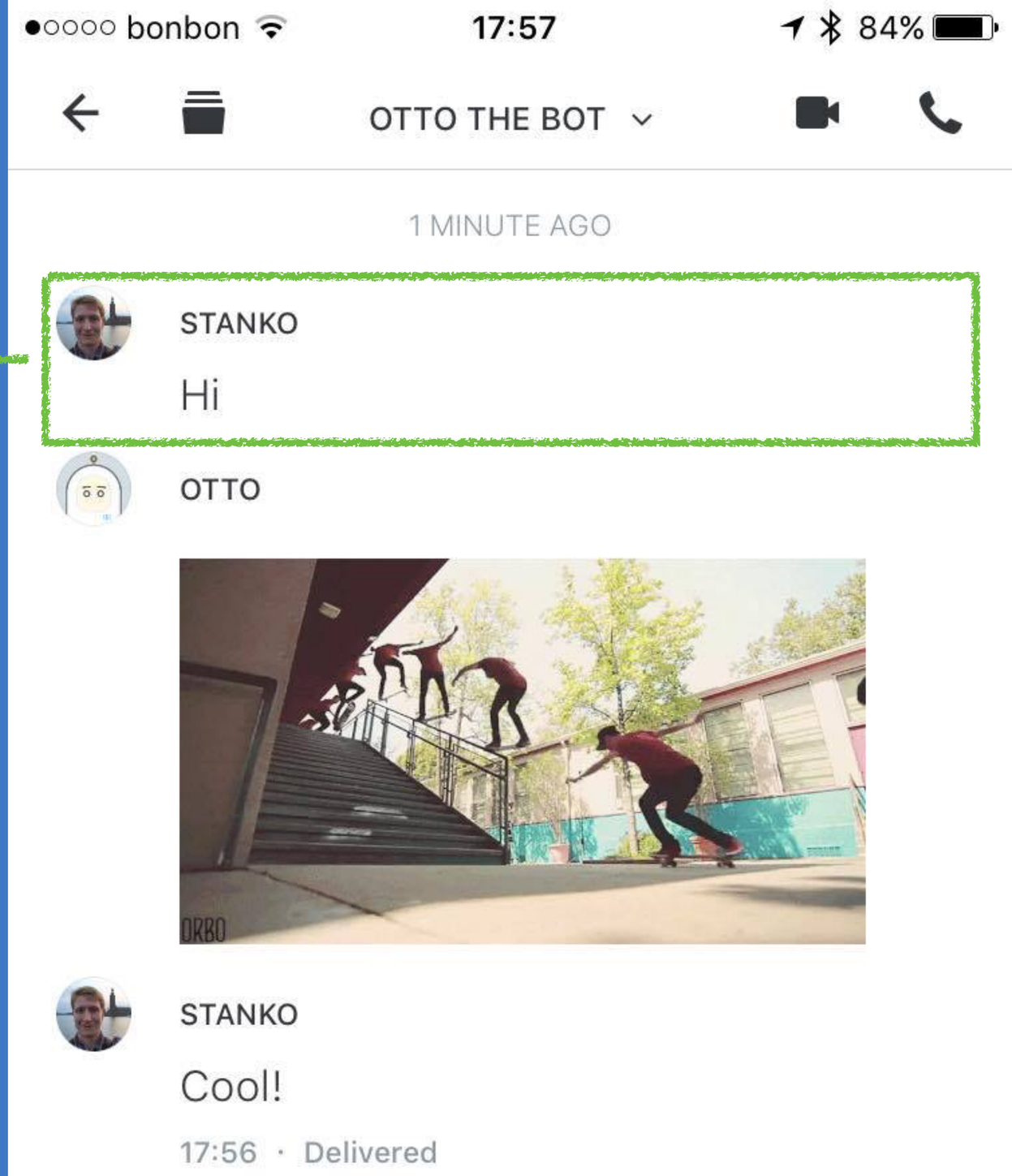


/api/conversations/53

```
{
  name: “Otto the bot”,
  participant_user_ids: [1, 1337],
  last_message_sent_at: 1337002,
  last_message_sent_by: {
    first_name: “Stanko”,
    last_name: “Krtalic Rusendic”
  }
  archived: false,
  last_message: {
    type: “plain”,
    body: “Cool!”,
    status: “delivered”
  }
}
```

/api/conversations/53/message/1

```
{
  message_type: “plain”,
  body: “Hi”,
  sender_id: “1337”,
  created_at: 1337001,
  updated_at: 1337001,
  previous_version_ids: [],
  status: “delivered”,
  seen_by_participants: [
    { first_name: “Otto” }
  ],
  sender: {
    first_name: “Stanko”,
    avatar_url: “https://…”
  }
}
```

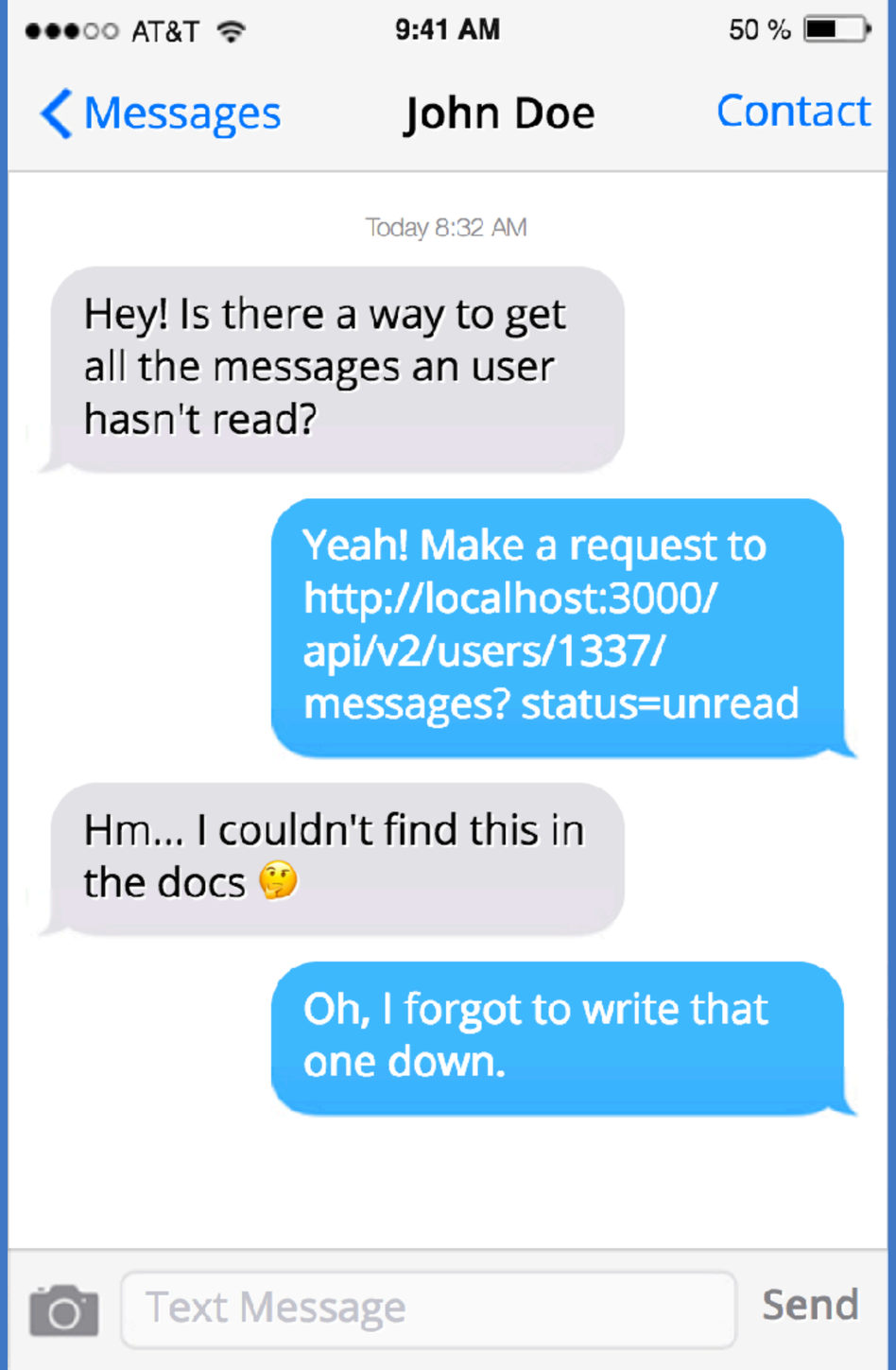


REST is pointless if your endpoints respond to a screen in your app

JSON::API
HAL

02

DOCUMENTATION


AT&T
9:41 AM
50 %
Messages
John Doe
Contact
Today 8:32 AM
Hey! Is there a way to get all the messages an user hasn't read?
Yeah! Make a request to http://localhost:3000/api/v2/users/1337/messages? status=unread
Hm... I couldn't find this in the docs 🤔
Oh, I forgot to write that one down.
Text Message
Send

Documentation is boring.

Documentation will always be lacking,
or at least lag behind the
implementation

Swagger
Apiary
API Blueprint
(documentation derived from tests)

Require manual labor

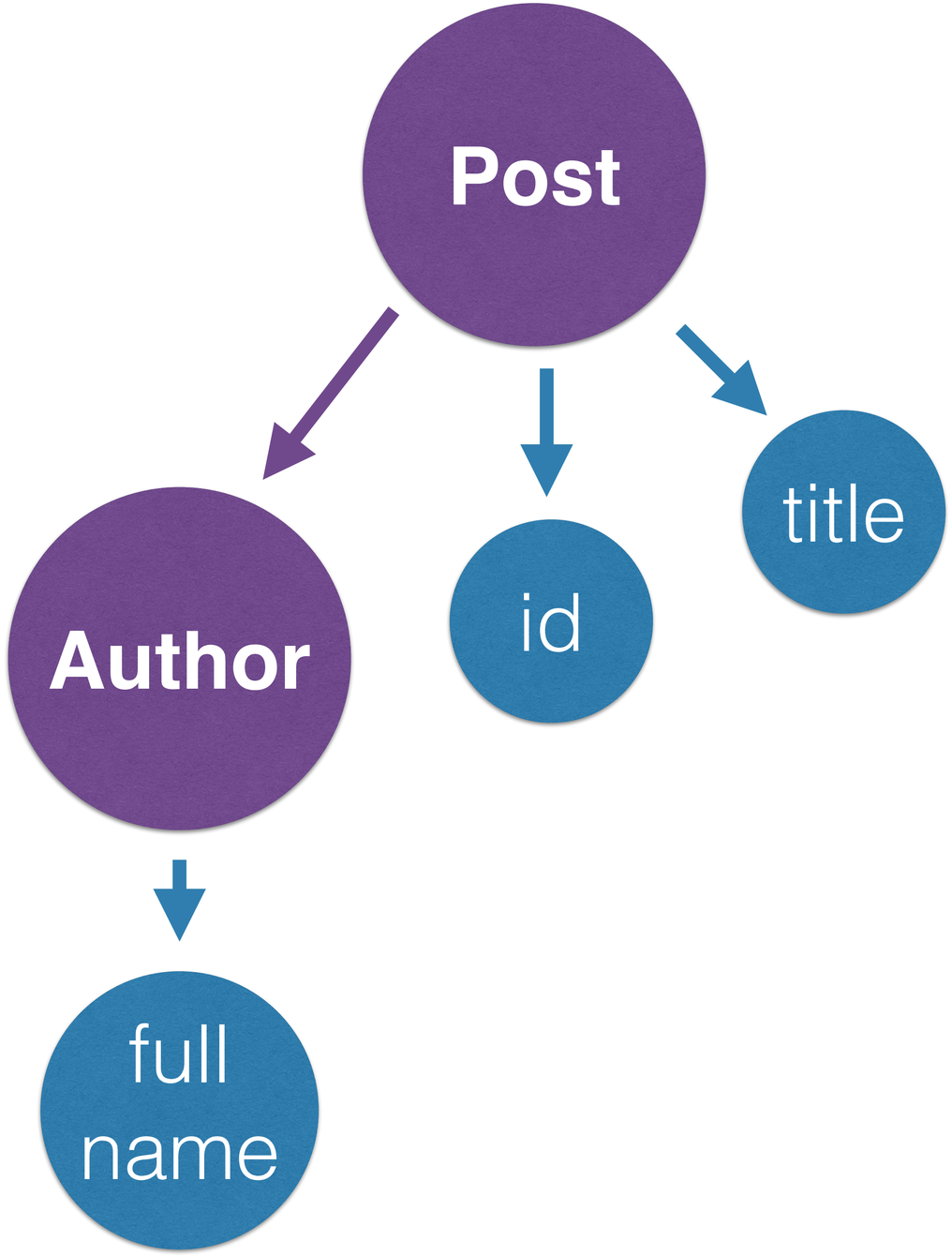
Laborious

03

GRAPHQL

Think of your resources as endpoints of a graph

```
query {
  posts {
    id
    title
    created_at
    author {
      full_name
    }
  }
}
```



```
SELECT "posts".id, "posts".title, "posts".body, "posts".created_at
FROM "posts"
ORDER BY "posts".created_at DESC
```

```
query {
  posts {
    id
    title
    created_at
    author {
      full_name
    }
  }
}
```

```
{
  "data": {
    "posts": [
      {
        "id": "1",
        "title": "Lilies of the Field",
        "created_at": "2017-02-25T18:45:29Z",
        "author": {
          "full_name": "Stanko Krtalic Rusendic"
        }
      },
```

String
Boolean
Int
Float
ID (String or Int)

Post

A blog post

FIELDS

author: Author!

body: String!

comment_count: Int!

comments: [Comment!]

created_at: DateTime!

id: ID!

title: String!

Queries
Mutations

```
query {
  post(id: 1){
    id
    title
    created_at
    author {
      full_name
    }
    comments {
      author {
        first_name
      }
      body
    }
  }
}
```

```
{
  "data": {
    "post": {
      "id": "1",
      "title": "Lilies of the Field",
      "created_at": "2017-02-25T18:45:29Z",
      "author": {
        "full_name": "Stanko Krtalic Rusendic"
      },
      "comments": [
        {
          "author": {
            "first_name": "Dario"
          },
          "body": "It was summer… and it was hot. Rachel was there… A
lonely grey couch…”OH LOOK!” cried Ned, and then the kingdom was his
forever. The End."
        },
        {
          "author": {
            "first_name": "Tomislav"
          },
          "body": "Raspberries? Good. Ladyfingers? Good. Beef? GOOD!"
        },
```

Queries

~~Mutations~~

Query-Mutations

```
mutation {
  createComment(input: {
    postId: 1
    authorId: 1
    body: "This was created using GraphQl 🙌"
  })
  {
    post {
      comments {
        body
      }
    }
  }
}
```

```
{
  "data": {
    "createComment": {
      "post": {
        "comments": [
          {
            "body": "This was created with GraphQL 🙌"
          }
        ]
      }
    }
  }
}
```

```
query {
  post_1: post(id: "1") {
    ...postFields
  },
  post_2: post(id: "2") {
    ...postFields
  }
  post_3: post(id: "3") {
    ...postFields
  }
}

fragment postFields on Post {
  title
  author {
    full_name
  }
  body
  comment_count
}
```

```
{
  "data": {
    "post_1": {↔},
    "post_2": {↔},
    "post_3": {
      "title": "I Sing the Body Electric",
      "author": {
        "full_name": "Stanko Krtalic Rusendic"
      },
      "body": "If we override the driver, we can get to the GB
application through the multi-byte XML alarm!\nWe need to index the 1080p
RAM interface!\nTry to transmit the AI bus, maybe it will connect the
digital interface!\nYou can't connect the interface without connecting
the solid state SQL driver!\nUse the optical EXE bus, then you can
compress the haptic alarm!\nThe PCI interface is down, input the open-
source capacitor so we can back up the SMTP feed!\nOverriding the pixel
won't do anything, we need to synthesize the virtual usb card!\nI'll
connect the 1080p RSS driver, that should bandwidth the GB array!",
      "comment_count": 5
    }
  }
}
```

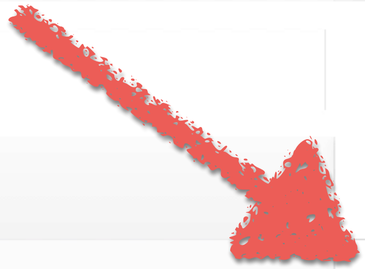
```
{
  search(text: "an") {
    __typename
    ... on Human {
      name
    }
    ... on Droid {
      name
    }
    ... on Starship {
      name
    }
  }
}
```

```
{
  "data": {
    "search": [
      {
        "__typename": "Human",
        "name": "Han Solo"
      },
      {
        "__typename": "Human",
        "name": "Leia Organa"
      },
      {
        "__typename": "Starship",
        "name": "TIE Advanced x1"
      }
    ]
```

GraphiQL

/localhost:3000/graphql

Method POST Edit HTTP Headers



```
{
  "data": {
    "post_1": {↔},
    "post_2": {↔},
    "post_3": {
      "title": "I Sing the Body Electric",
      "author": {
        "full_name": "Stanko Krtalic Rusendic"
      },
      "body": "If we override the driver, we can get to the GB
application through the multi-byte XML alarm!\nWe need to index the 1080p
RAM interface!\nTry to transmit the AI bus, maybe it will connect the
digital interface!\nYou can't connect the interface without connecting
the solid state SQL driver!\nUse the optical EXE bus, then you can
compress the haptic alarm!\nThe PCI interface is down, input the open-
source capacitor so we can back up the SMTP feed!\nOverriding the pixel
won't do anything, we need to synthesize the virtual usb card!\nI'll
connect the 1080p RSS driver, that should bandwidth the GB array!",
      "comment_count": 5
    }
  }
}
```

Documentation Explorer

Search the schema ...

A GraphQL schema provides a root type for each kind of operation.

ROOT TYPES

query: Query

mutation: Mutation

04

GRAPHQL IN RUBY

```
21 gem 'kaminari'
22 gem 'trix'
23 gem 'graphql'
24 gem 'graphql-batch'
25 gem 'shrine', '~> 2.2'
```

```ruby
Rails.application.routes.draw do
  root to: 'posts#index'
  resources :comments
  resources :posts
  resources :authors

  # You only need this for GraphQl
  resource :graphql, only: [:create], controller: 'graphql'
end
```

```ruby
class GraphqlController < ApplicationController
  skip_before_action :verify_authenticity_token

  def create
    query_string = params[:graphql][:query]
    variables = params[:graphql][:variables].try(:to_unsafe_hash)
    context = {
      # current_user: current_user
    }

    graphql_response = GraphQlSchema.execute(
      query_string,
      variables: variables,
      context: context
    )

    render json: graphql_response
  end
end
```

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

```ruby
module Graph
  module Types
    QueryType = GraphQL::ObjectType.define do
      name 'Query'
      description 'The query root of this schema'

      # Individual getters
      field :post, PostType,
            field: Queries::BasicGetter.by_id(type: PostType, model: Post)
      field :author, AuthorType,
            field: Queries::BasicGetter.by_id(type: AuthorType, model: Author)
      field :comment, CommentType,
            field: Queries::BasicGetter.by_id(type: CommentType, model: Comment)

      # Batch getters
      field :authors, types[!AuthorType],
            field: Queries::BasicGetter.batch(
              type: types[!AuthorType],
              model: Author
            )
      field :comments, types[!CommentType], field: Queries::CommentsQuery
      field :posts, types[!PostType],
            field: Queries::BasicGetter.batch(
              type: types[!PostType],
              model: Post
            )
    end
  end
end
```

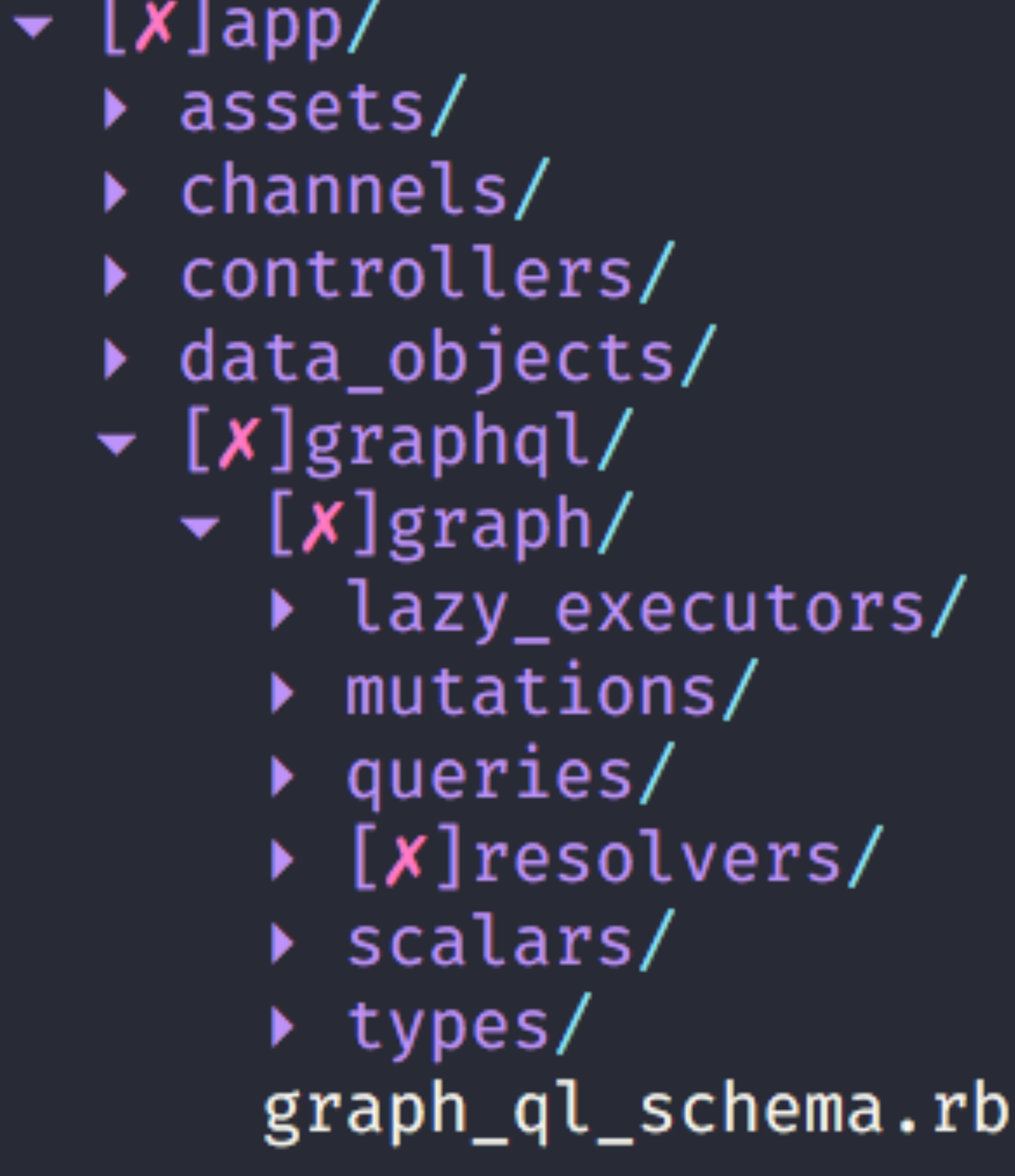
```ruby
module Graph
  module Types
    MutationType = GraphQL::ObjectType.define do
      name 'Mutation'
      description 'The mutation root of this schema'

      field :createComment, field: Mutations::CreateCommentMutation.field
    end
  end
end
```

```ruby
module Graph
  module Types
    PostType = GraphQL::ObjectType.define do
      name 'Post'
      description 'A blog post'
      field :id, !types.ID
      field :title, !types.String
      field :body, !types.String
      field :author, !AuthorType
      field :comments, types[!CommentType]
      field :comment_count, !types.Int
      field :created_at, !Scalars::DateTimeScalar
    end
  end
end
```

Organization

```ruby
module Graph
  module Mutations
    CreateUserMutation = GraphQL::Relay::Mutation.define do
      name 'CreateUser'

      input_field :email, types.String
      input_field :password, types.String
      input_field :fullName, types.String
      input_field :firstName, types.String
      input_field :lastName, types.String
      input_field :phone, types.String
      input_field :receivePromotionalEmails, types.Boolean

      return_field :user, Types::UserType
      return_field :token, types.String
      return_field :errors, Scalars::JsonScalar
      return_field :error_messages, types[types.String]

      resolve Resolvers::User::Creator
    end
  end
end
```

```ruby
module Graph
  module Resolvers
    module User
      class Creator
        attr_reader :object
        attr_reader :inputs
        attr_reader :ctx

        def self.call(object, inputs, ctx)
          new(object, inputs, ctx).call
        end

        def initialize(object, inputs, ctx)
          @object = object
          @inputs = inputs
          @ctx = ctx
        end
```


[✗]app/
assets/
channels/
controllers/
data_objects/
[✗]graphql/
[✗]graph/
lazy_executors/
mutations/
queries/
[✗]resolvers/
scalars/
types/
graph_ql_schema.rb

Optimizing

```ruby
GraphQlSchema = GraphQL::Schema.define do
  query Graph::Types::QueryType
  mutation Graph::Types::MutationType

  lazy_resolve(Graph::LazyExecutors::Single, :resolve)
  lazy_resolve(Graph::LazyExecutors::Batch, :resolve)
end
```

```ruby
module Graph
  module Types
    ProductType = GraphQL::ObjectType.define do
      name 'Product'
      description 'A product'
      field :id, types.ID
      field :name, types.String
      field :sku, types.String
      field :available_on, Scalars::DateTimeScalar
      field :description, types.String
      field :meta_description, types.String
      field :meta_keywords, types.String
      field :meta_title, types.String
      field :discontinue_on, Scalars::DateTimeScalar
      field :prices, types[PriceType]
      field :variants, types[VariantType]
      field :variant_images, types[ImageType]
      field :taxonomies, types[Types::BrandType]
      field :created_at, Scalars::DateTimeScalar
      field :updated_at, Scalars::DateTimeScalar
      field :price, types.Float
      field :currency, types.String
      field :recommendations, types[Types::ProductType] do
        resolve ->(obj, args, ctx) do
          LazyExecutors::Batch
            .new(Spree::Product, ctx, obj.recommended_product_ids)
        end
      end
    end
  end
end
```

```ruby
def initialize(model_class, query_ctx, id)
  @model_class = model_class
  @query_ctx = query_ctx
  @id = id

  @lazy_state = query_ctx[:lazy_find_product] ||= {
    pending_ids: Set.new,
    loaded_ids: {},
  }

  lazy_state[:pending_ids] << id
end

def resolve
  loaded_record = lazy_state[:loaded_ids][id]

  return loaded_record if loaded_record

  pending_ids = lazy_state[:pending_ids].to_a
  records = scope.where(id: pending_ids)
  records.each { |record| lazy_state[:loaded_ids][record.id] = record }
  lazy_state[:pending_ids].clear
  lazy_state[:loaded_ids][id]
end
```

```ruby
def self.batch(model:, type:, scope: nil)
  return_type = type
  GraphQL::Field.define do
    type(return_type)
    argument(
      :page, types.Int,
      "Index of the page, by default only 10 elements are shown per page"
    )
    description("Return all #{model.name.pluralize}")
    resolve ->(obj, args, ctx) {
      new_scope = scope && scope.respond_to?(:call) &&
        scope.call(obj, args, ctx)
      scope = new_scope || scope
      relation = scope || model.all
      GraphQl::Loader.call(ctx, relation)
    }
  end
end
```

Questions?