

# GraphQL

Concepts & Challenges

- I'm Robert Mosolgo
- Work from home Ruby developer
- From Charlottesville VA
- For GitHub

# Rails

## API

## WHY

- You have your Rails app, why bother with an API?
- You have clients. Native app team, fancy front end, integrators
- Do stuff, render views (even tasks have some kind of output)

# APIs

GET /decks/100.json

```
{  
  "deck": {  
    # ...  
  }  
}
```


- Traditionally, Resource-based REST API
- Endpoints with “representations” of objects
- This is great, people start fetching your data and doing weird stuff with it

# APIs

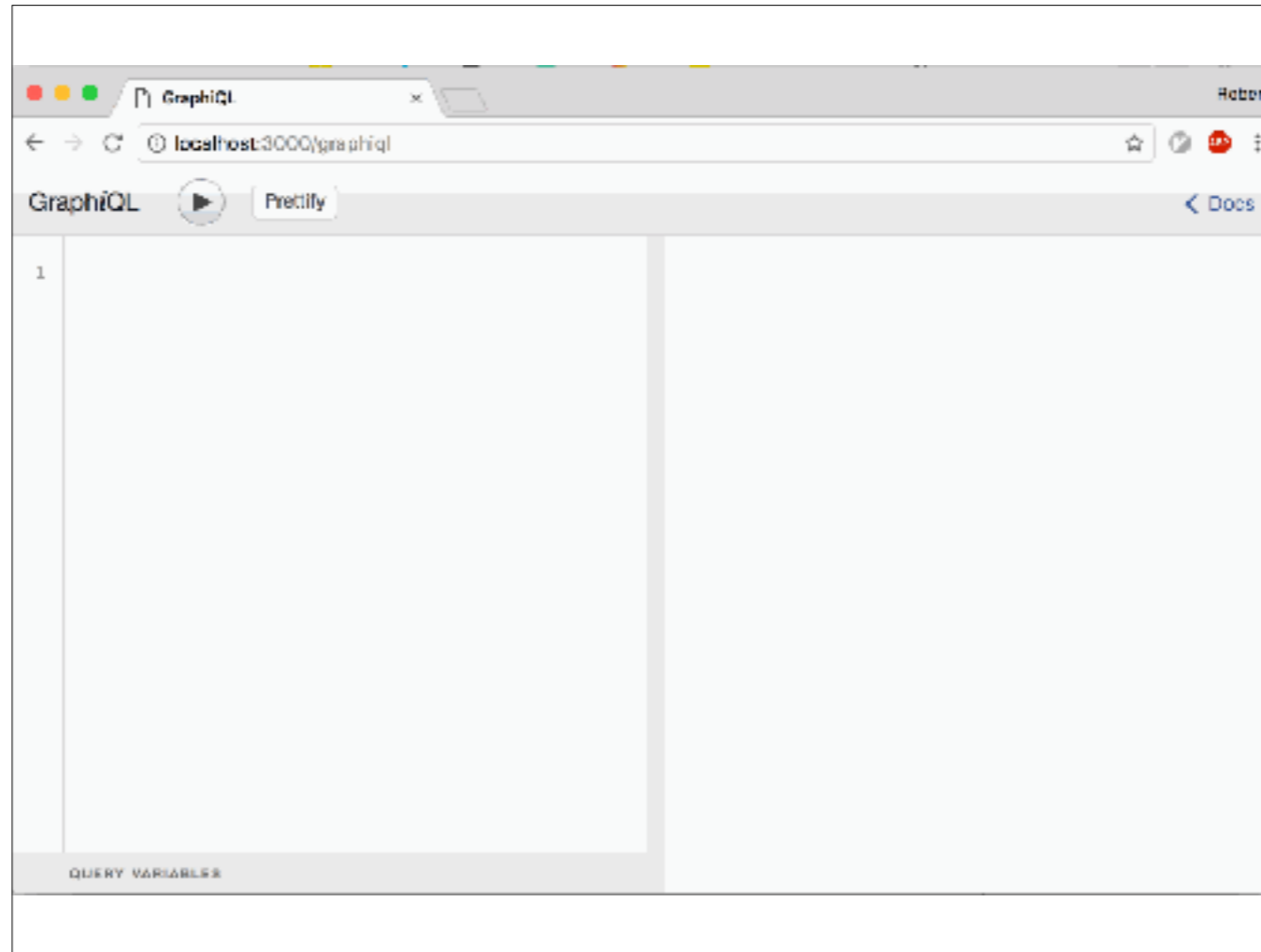


- As a server developer, this is bad news.
- Someone will find your API and hammer it
- They'll send you bad inputs

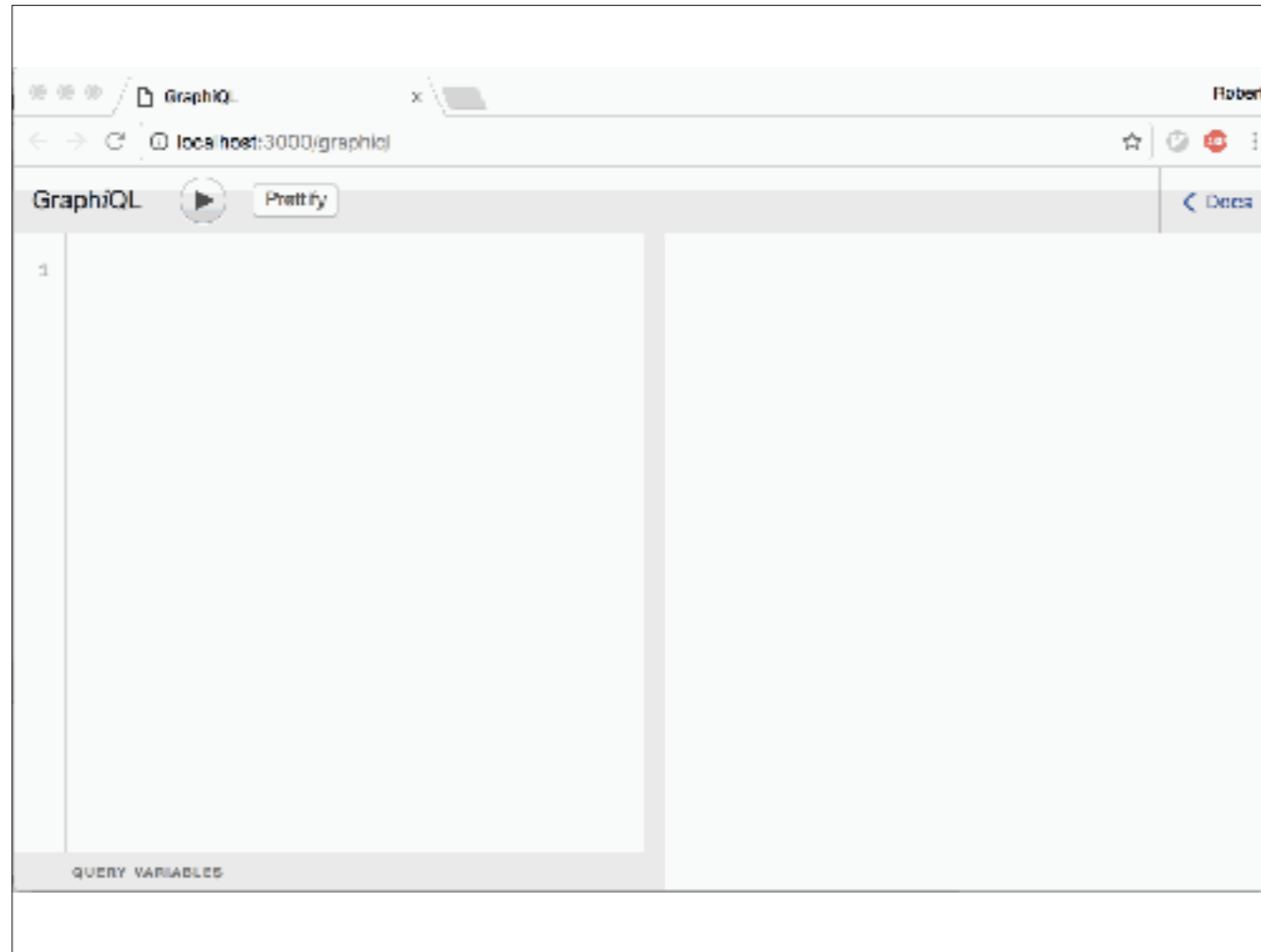
# APIs

Magic Cards 	GET /decks/100.json GET /favorites.json
<b>GWB Junk</b> 60 cards	
♥ 4x <a href="#">Doran, the Siege Tower (V11)</a>	GET /cards/5.json GET /expansions/43.json
♥ 4x <a href="#">Baneslayer Angel (V15)</a>	
♥ 4x <a href="#">Bitterblossom (MM2)</a>	GET /cards/12.json GET /expansions/39.json
♥ 3x <a href="#">Maelstrom Pulse (MPS)</a>	
♥ 3x <a href="#">Qasali Pridemage (DDH)</a>	GET /cards/26.json GET /cards/35.json
♥ 4x <a href="#">Inquisition of Kozilek (MM3)</a>	

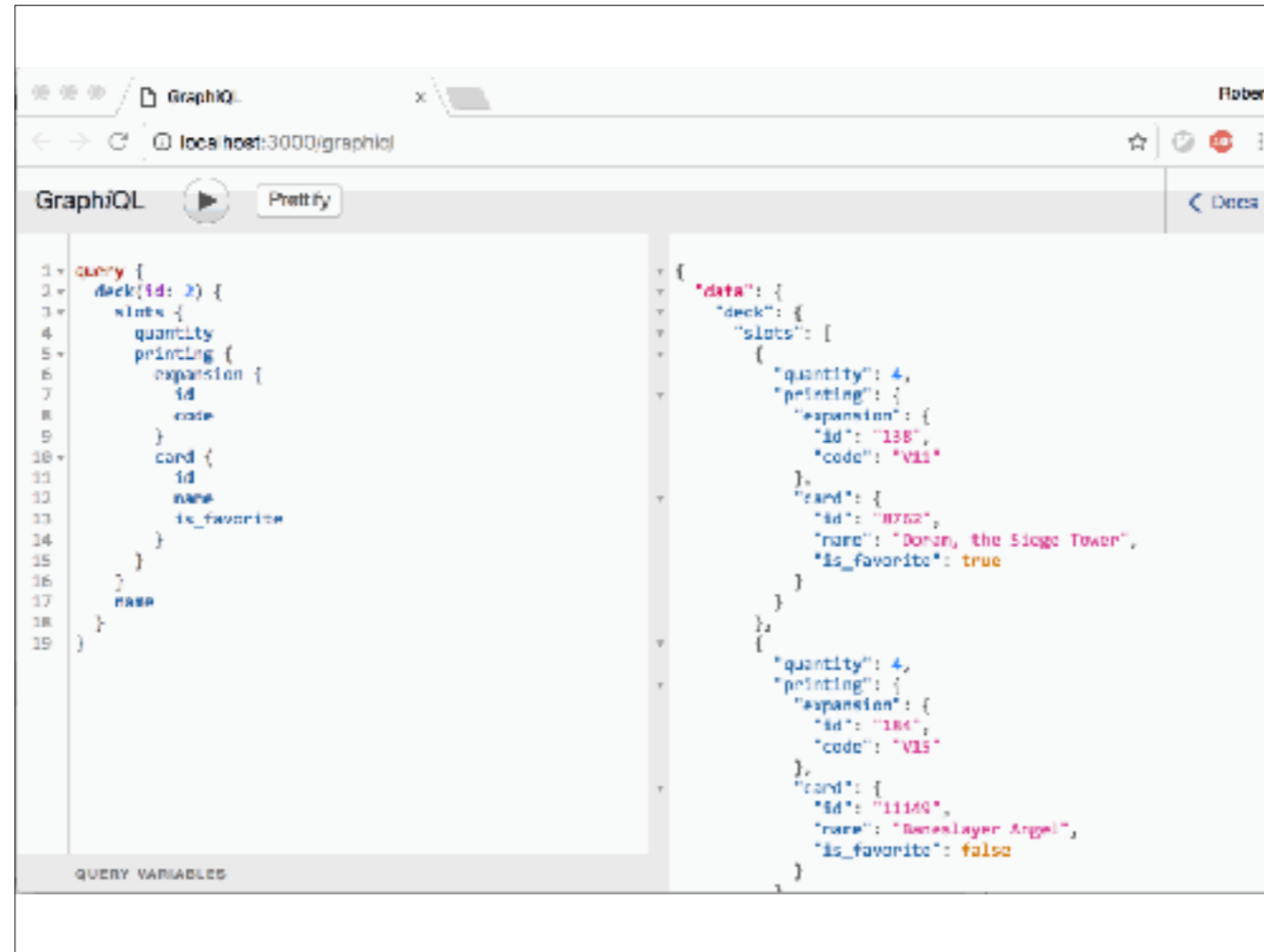
- The good news is, it's also a pain for your clients
- Rendering a useful view requires a lot of different kinds of data



- This is GraphiQL, an in-browser IDE for GraphQL queries
- Simple query
- query keyword
- matching response shape

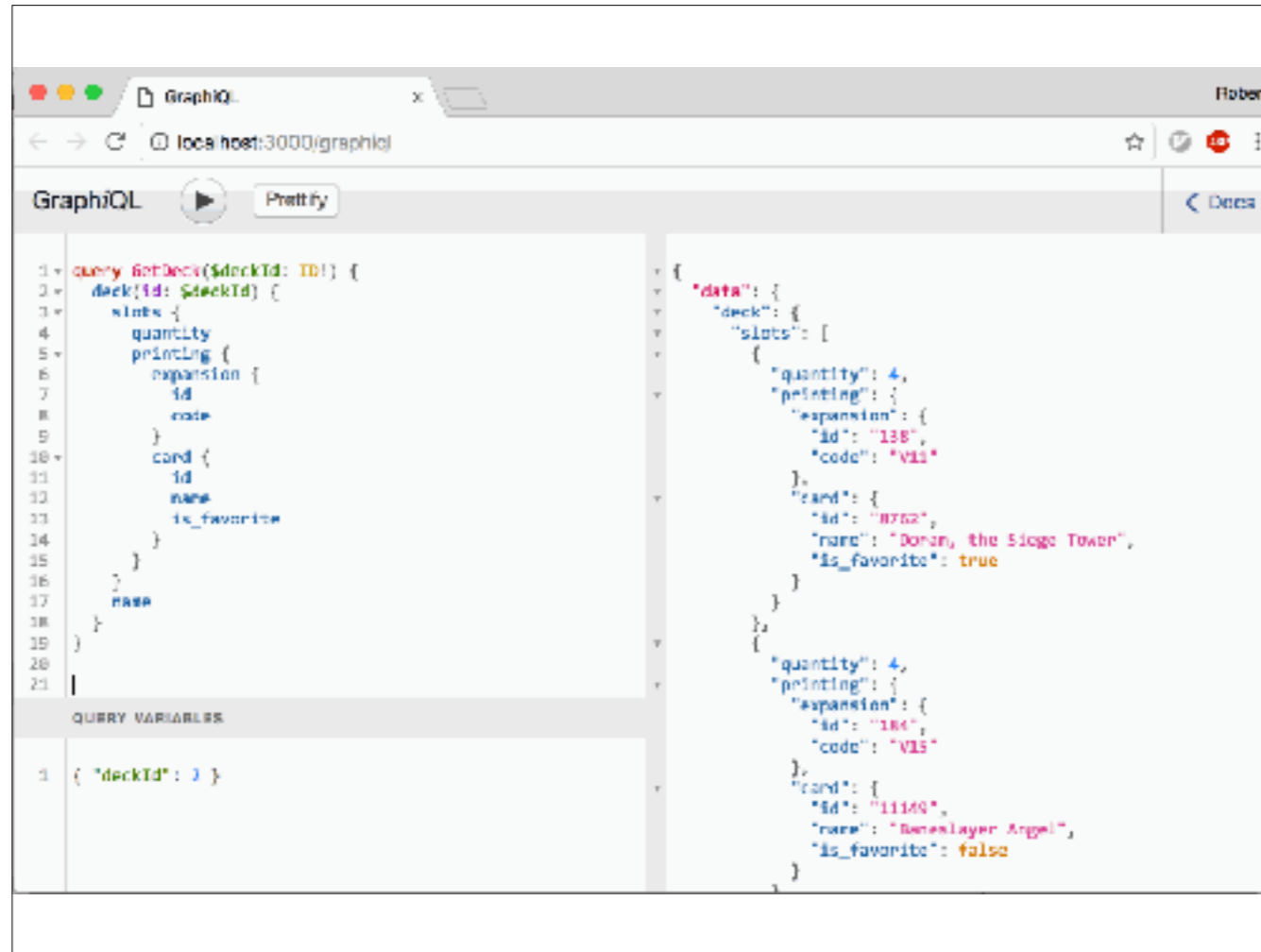


- Let's make a query to render that UI
- nested resources



- We have the query for rendering a view, but the 2 is hardcoded
- Extract it to a variable, give the query a name
- Now the query is like a function





- Extracting logic is good
- How about fragment, you can share bits of queries between views

# REST vs GraphQL?

GET /decks/100.json

# ??

POST /graphql

```
type Deck {  
  name: String!  
  id: ID!  
  rating: Int  
  cards: [Card!]!  
}
```

How is GraphQL Different from REST?

- Structure is stable and client defined
- Strongly-typed attributes prevent accidents

# REST vs GraphQL?

```
GET /decks/100.json
GET /cards/5.json
GET /cards/12.json
GET /cards/26.json
GET /cards/35.json
```

```
{
  deck(id: 100) {
    name
    cards {
      name
      colors
    }
  }
}
```

How is GraphQL Different from REST?

- Fewer roundtrips, simpler client code
- Resources are structured according to client's needs

# REST vs GraphQL?

```
query {  
  user(login: "rmosolgo") {  
    decks(first: 10000) {  
      cards(first: 10000) {  
        artist {  
          name  
        }  
        # ...  
      }  
    }  
  }  
}
```

How is GraphQL Different from REST?

- Worse for servers
- Client can make insane requests: high volume, high complexity

# SQL vs GraphQL?

How is GraphQL different from opening up your SQL server?

- Runs Ruby code
- Storage-agnostic
- Application logic, access control

# GraphQL Ruby

```
Types::CardType = GraphQL::ObjectType.define do
  name "Card"
  description "A printed card which may be played"
  field :id, types.ID
  field :name, types.String
  field :expansion, Types::ExpansionType
  field :image_path, types.String do
    resolve ->(card, args, ctx) {
      card.latest_printing.image_path
    }
  end
end
```

## Example Ruby code

- Type, docs built in
- Strongly-typed fields
- Relationship between objects
- Fields call methods, or custom logic



We're using GraphQL for a few things:

- Rails views fetch data with GraphQL
- GraphQL API
- Why?
  - Client experience
  - Centralization: maintenance, monitoring, improvements

# GraphQL

- Let's talk about a few API-related topics
- You might be familiar with these categories from resource-based API development
- I hope we can bridge the gap



# Authentication

```
class GraphQLController < ApplicationController
  before_action :authorize

  def execute
    context = { current_user: current_user }
    # run GraphQL query
    result = CardsSchema.execute(
      params[:query],
      context: context
    )

    render json: result
  end
end
```

## Authentication

- GraphQL over HTTP
- User identity is the same
- Use GraphQL context, Sam introduced this

# Authorization

```
field :search, types[Types::CardType] do
  argument :term, types.String
  resolve ->(obj, args, ctx) {
    current_user = ctx[:current_user]
    current_user.cards.search(term: args[:term])
  }
end
```

## Authorization

- You've got the user, now, give them permitted data
- Scope your DB loads
- You probably do this in controller actions already
- Check at the last minute
- HackerOne's gem

# Authorization

```
def call_with_auth(obj, args, ctx)

  result = @field.call(obj, args, ctx)

  if !authorized?(ctx[:current_user], result)
    raise "Authorization failure!"
  end

  result
end
```

[https://github.com/Hacker0x01/protected\\_attribute](https://github.com/Hacker0x01/protected_attribute)

## Authorization

- Check at the last minute
- HackerOne's gem

# Rate Limiting

```
query {  
  deck(id: 100) { ← 1 deck  
    name  
    cards(first: 10) { ← 10 cards  
      name  
      colors  
      artist { ← 1 artist * 10 cards  
        name  
      }  
    }  
  }  
}
```

21 nodes

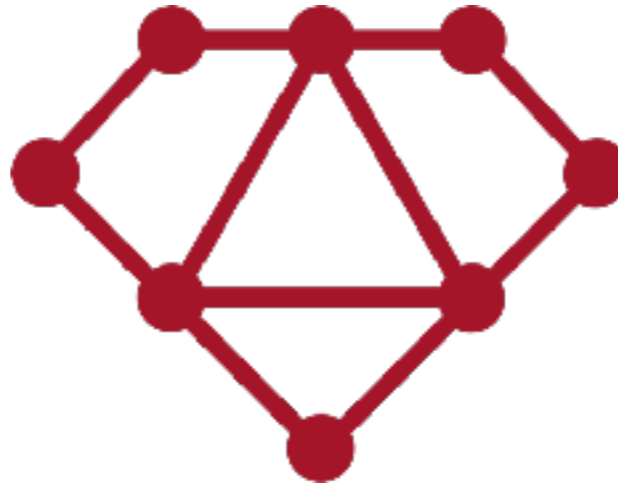
[http://graphql-ruby.org/queries/complexity\\_and\\_depth](http://graphql-ruby.org/queries/complexity_and_depth)

## Rate Limiting

- Clients can abuse your system
- With REST you count requests in a window
- Count max nodes in a query, limit it within window

- N+1 Queries
- Static Analysis
- Timeout
- Instrumentation

- some things we didn't talk about but I wish we had time for



<http://graphql.org>

<http://graphql-ruby.org>

@rmosolgo

- I work from home so please come talk to me about GraphQL