Rethinking REST

A hands-on guide to GraphQL and queryable APIs

Presented by:
Arianne Dee
May 30, 2018

Survey - About you

- What area of the world are you watching from
    - Africa, Asia, Australia, Europe, North America, South America, Middle East, other
- What languages/tech are you comfortable with (multiple)
    - JavaScript, Python, HTML/CSS, None
- What frameworks do you want to use GraphQL with (multiple)
    - Node.js, Django/Flask, .NET, Rails/Sinatra, SpringMVC, Groovy, other [with text input]
- What is your GraphQL knowledge
    - Total newb, played around with API, tried to implement it (client or server), used in production (client or server)
- What is your role (multiple)
    - Front-end developer, back-end developer, project manager, other [with text input]

About Me

Location: Vancouver, Canada

University of British Columbia
Civil Engineering, Computer Science

Software developer: 5 years
Django developer: 3 years
GraphQL: 2 years

My GraphQL timeline

- Back-end developer at 7Geese - February 2016
- GraphQL in production - September 2016
- Meetup presentation - October 2016
- DjangoCon presentation - August 2017 - Video
- Side project: django-graph-api - September 2017

Today's schedule

	Duration
• **Why GraphQL?**	15 mins
• **Explore a GraphQL API**	25 mins
• Q&A + break	15 mins
• **Build a GraphQL client**	45 mins
• Q&A + break	15 mins
• **Build a GraphQL server**	1 hr 45 mins
• Node.js, JavaScript	
• Django, Python	
• Final Q&A	15 mins

Q&A format

- 15 mins at end of each section (3 total)
- Use the Q&A feature

- A few questions read out loud & answered

- Can use group chat to ask each other questions during session

Let's talk about **REST**

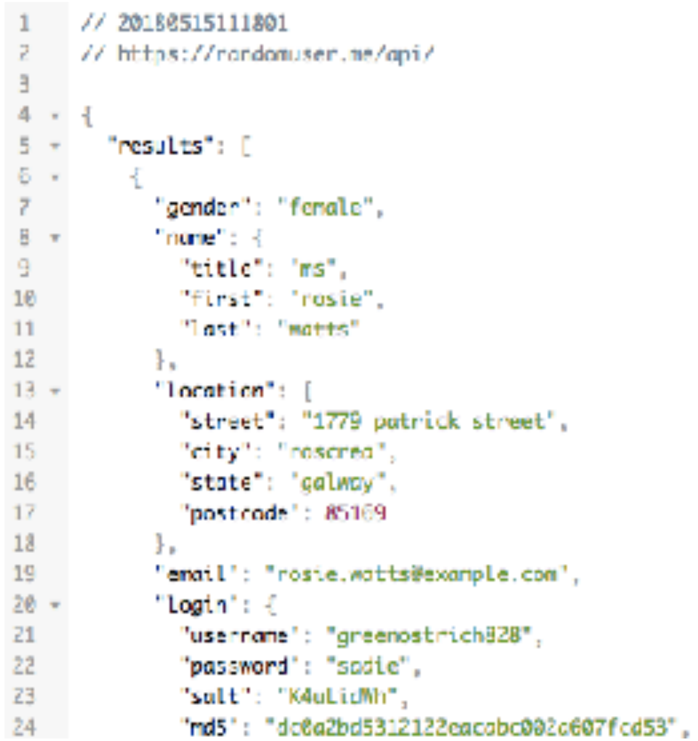
RESTful APIs

- Uses HTTP methods: GET, POST, PUT, DELETE
- One url endpoint per resource
- Can use HTTP error codes: e.g. 200, 400, 403, 404
- Independence of client and server
- Cacheable

Challenge #1:
Over-fetching

RESTful API - resource fields

- e.g. **User resource**
  - Name
  - Username
  - Is admin?
  - Email
  - Profile photo
  - Phone number
  - Id
  - Twitter handle
  - Sign-up date
  - + more



```
// 20180515111801
// https://randomuser.me/api/

{
  "results": [
    {
      "gender": "female",
      "name": {
        "title": "ms",
        "first": "rosie",
        "last": "watts"
      },
      "location": [
        "street": "1779 patrick street",
        "city": "roscrea",
        "state": "galway",
        "postcode": 85169
      },
      "email": "rosie.watts@example.com",
      "login": {
        "username": "greenostrich928",
        "password": "sadie",
        "salt": "K4uLidMh",
        "md5": "dc0a2bd5312122eacabc002c607fcd53",
```

Unneeded data

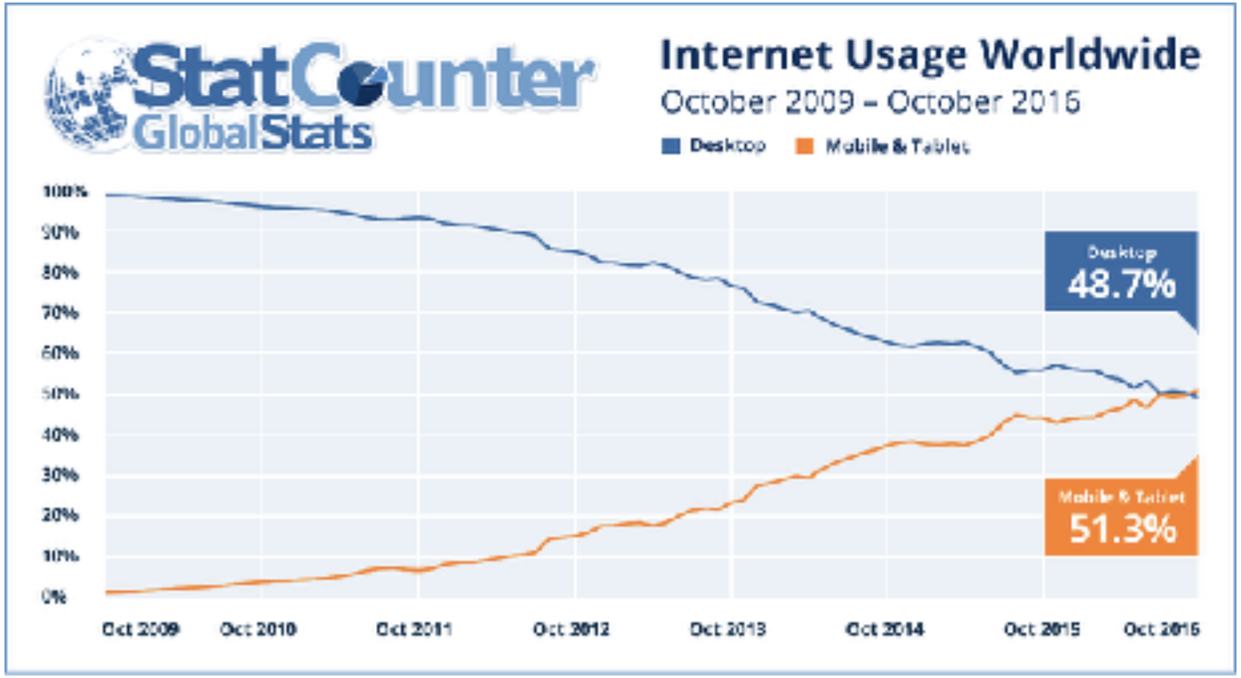
- e.g. User resource
- # fields: 34
- # important fields: 4
  - Id
  - Name
  - Is admin?
  - Profile photo

- Excess fields: 30/34 or 88%

Desktop vs Mobile API design

By Muhammad Rafizeldi [CC BY-SA 3.0], from Wikimedia Commons

Mobile internet use passes desktop for the first time, study finds



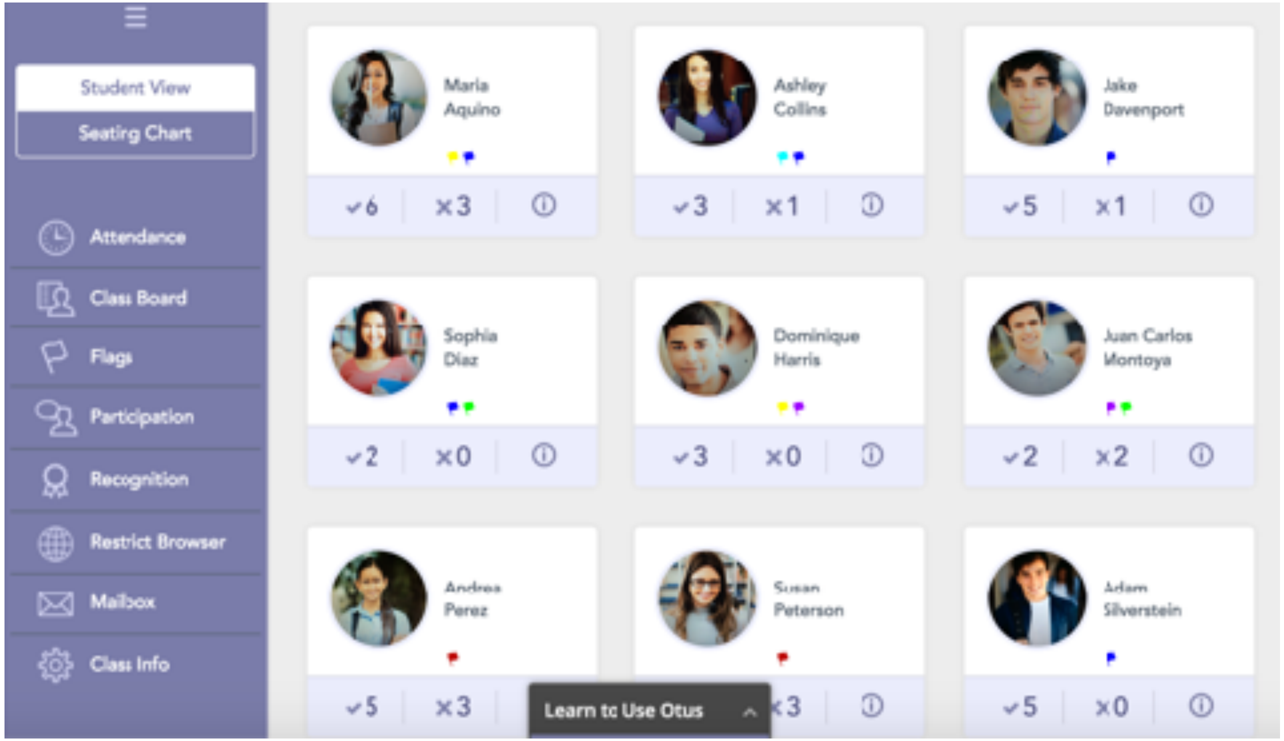
Source: Tech Crunch - Nov 2016

Challenge #2:
Under-fetching

RESTful API - related resources

- /api/user/{pk}/
- /api/user/{pk}/resource/
- /api/user/{pk}/resource/{pk}/related_resource/

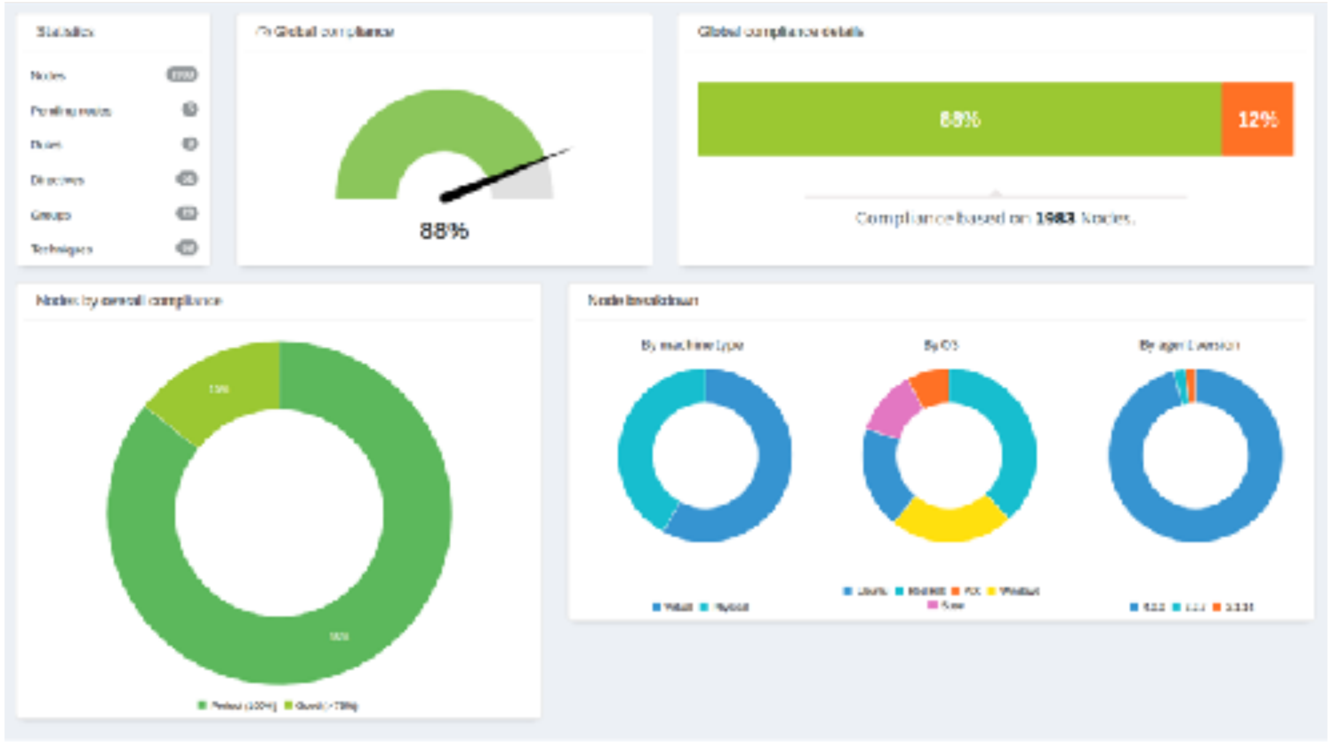
Summaries



Maria Aquino ✓6 ×3
Ashley Collins ✓3 ×1
Jake Davenport ✓5 ×1
Sophia Diaz ✓2 ×0
Dominique Harris ✓3 ×0
Juan Carlos Montoya ✓2 ×2
Andrea Perez ✓5 ×3
Susan Peterson ×3
Adam Silverstein ✓5 ×0

By Spencine [CC BY-SA 4.0], from Wikimedia Commons

Dashboards



By Moostik [CC BY-SA 4.0], from Wikimedia Commons

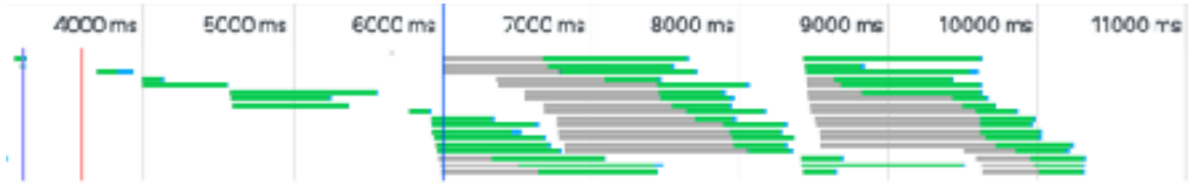
objectives for me
Q2 2017
Overall progress: 52%
On track
Off track
No status
OBJECTIVES OWNED
Active Q2 2017
Become more involved in the Python community
Learn new things
Spread the knowledge
PEOPLE
My team
Name
Last check-in
Objectives
Progress
51%
Overall progress
28
Objectives
None

RESTful API - related resources

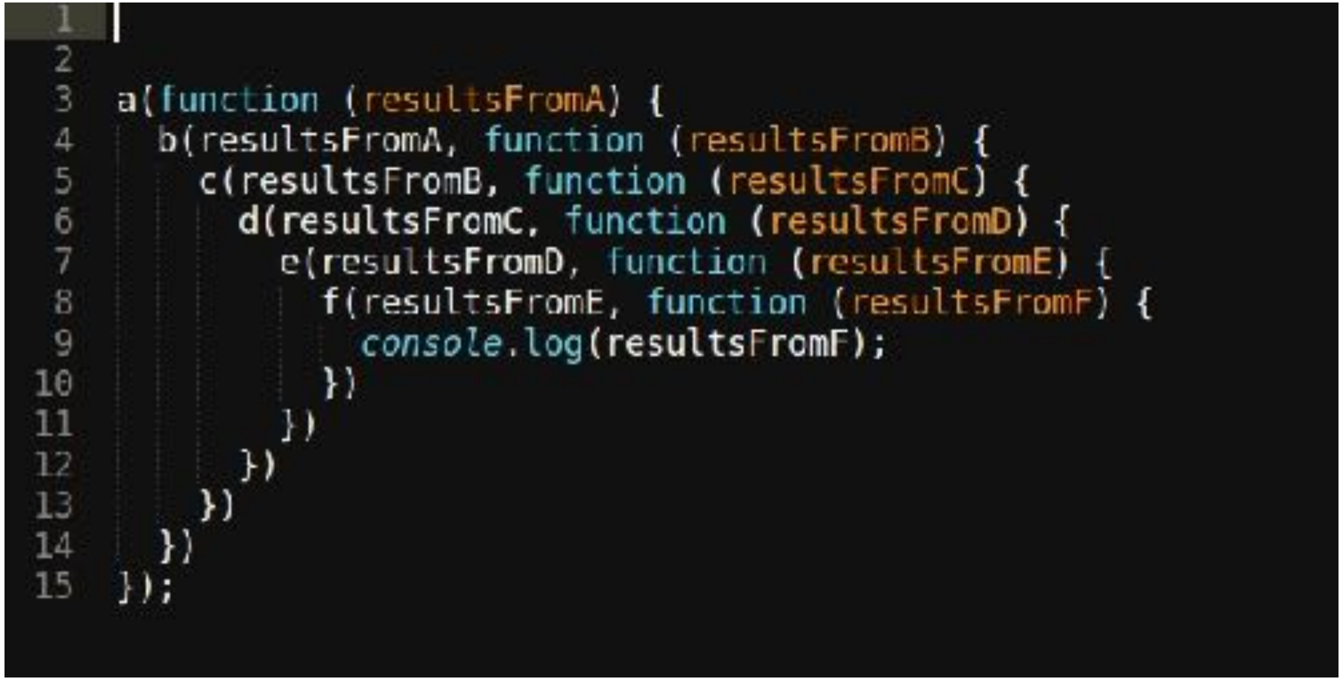
- /api/user/1/
- /api/user/1/teams/
- /api/team/100/members/

- /api/user/1/goal/
- /api/user/1/goal/500/task/

Performance

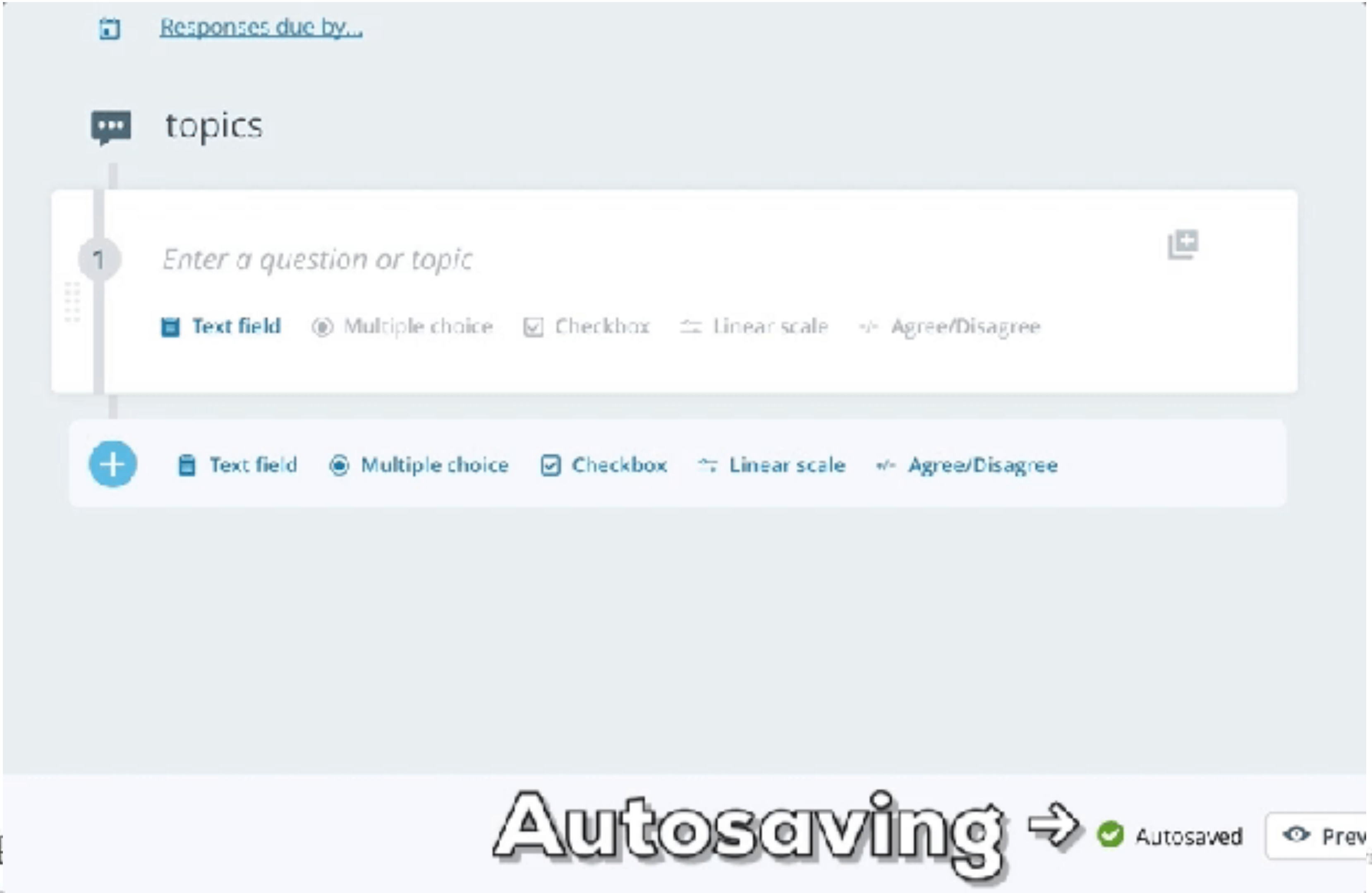


JavaScript callback/promise hell



```
a(function (resultsFromA) {
  b(resultsFromA, function (resultsFromB) {
    c(resultsFromB, function (resultsFromC) {
      d(resultsFromC, function (resultsFromD) {
        e(resultsFromD, function (resultsFromE) {
          f(resultsFromE, function (resultsFromF) {
            console.log(resultsFromF);
          })
        })
      })
    })
  })
});
```

Challenge #3:
Complicated updates


Responses due by...

topics

1 Enter a question or topic

Text field · Multiple choice · Checkbox · Linear scale · Agree/Disagree

Text field · Multiple choice · Checkbox · Linear scale · Agree/Disagree

Autosaving → Autosaved

Prev

Autosaving

1. POST (list) - Create new list
2. POST (item) - Add item
3. PUT (item) - Update item
4. POST (item) - Add item
5. PUT (list) - Reorder item
6. DELETE (item) - Delete item
7. GET (list) - Get current list of items

Great uses of REST

- List and detail UI
- Not too much nesting of data
- Predictable usage
    - Users want X fields from Y resource

Not-so-great uses of REST

- Using the same endpoints for web app & mobile app
- Dashboard/summary views
- Unpredictable API usage
- Keeping track of state after updates

What about queryable API's?

What about queryable API's?

Ask for what you want, and only get what you ask for

Queryable APIs

- Not a new concept

- Query parameters
- OData - Microsoft, SAP
- JSON API

Enter GraphQL

What is GraphQL?

- An API query language
- Created by Facebook in 2012
- Specifications open sourced in 2015

- Spec: http://facebook.github.io/graphql/

Go to: https://developer.github.com/v4/explorer/

Learn more: https://developer.github.com/v4/

#1 - Over-fetching

- Query the API for only the fields that you need

GraphQL Request

```
{
  viewer {
    name
    login
    location
  }
}
```

GraphQL Response

```
{
  "data": {
    "viewer": {
      "name": "Arianne",
      "login": "ariannedee",
      "location": "Vancouver, BC, Canada"
    }
  }
}
```

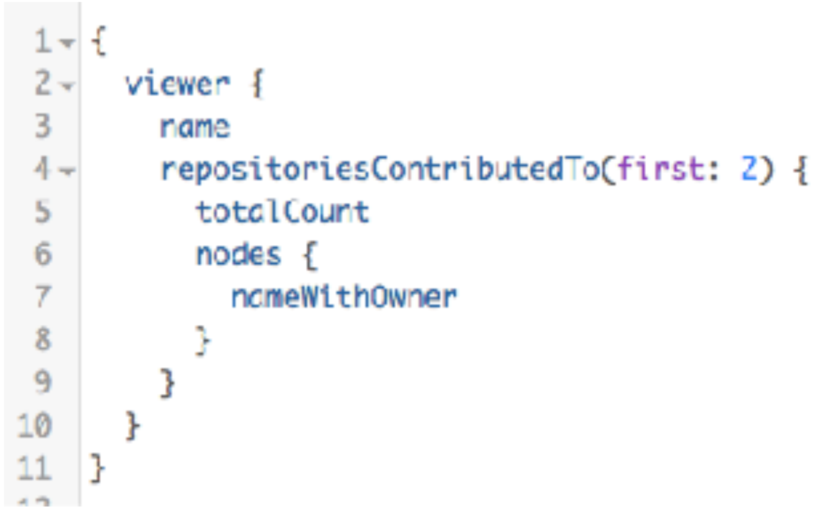
= 100% used

💯 👍

#2 - Under-fetching

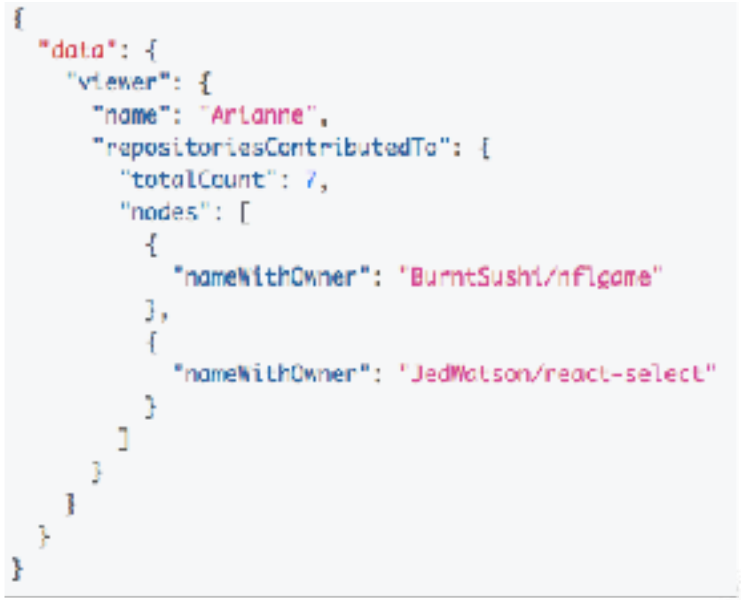
- Query for related resources
- Query for custom fields

GraphQL Request



```
{
  viewer {
    name
    repositoriesContributedTo(first: 2) {
      totalCount
      nodes {
        nameWithOwner
      }
    }
  }
}
```

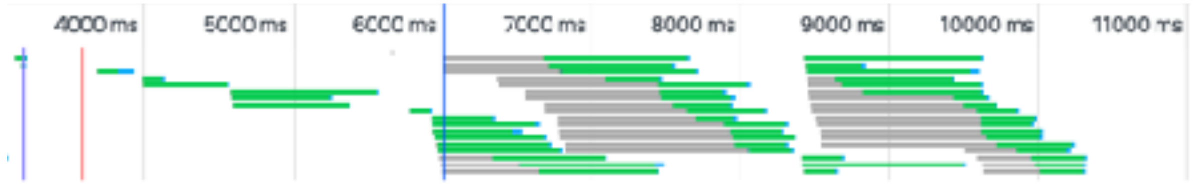
GraphQL Response



```
{
  "data": {
    "viewer": {
      "name": "Arianne",
      "repositoriesContributedTo": {
        "totalCount": 7,
        "nodes": [
          {
            "nameWithOwner": "BurntSushi/nflgame"
          },
          {
            "nameWithOwner": "JedWatson/react-select"
          }
        ]
      }
    }
  }
}
```

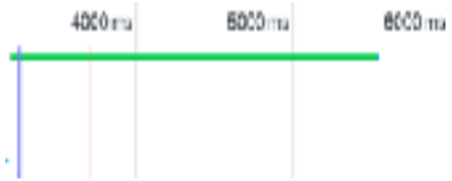
objectives for me
Q2 2017
Overall progress: 52%
On track
Off track
No status
My team
Name
Last check-in
Objectives
Progress
51%
Overall progress
28
Objectives

REST



10.3s

GRAPHQL



5.5s

☝👍

GraphQL APIs

- Uses HTTP methods: GET, **POST** (preferred)
- One url for whole API
- Only uses error code 200
    - Error(s) listed in response body
- Independence of client and server

Some differences from REST

- Single endpoint
    - Harder to cache but not impossible
- No HTTP errors
- Strongly typed
- Self-documenting
- Versioning is not required
    - Versioning in GraphQL vs REST

Questions to consider

- What tasks have been difficult to solve using REST?
- How can my team benefit from using GraphQL?
- What challenges might we face in adopting GraphQL?

GraphQL spec supports

- **Queries**
  - Get some data
- **Mutation**
  - Update some data
- **Directives**
  - Modify query (e.g. skip/include fields)
- **Subscriptions**
  - Server pushes data to client
  - Not discussed today

Language features - Queries

Query

REST equivalent: GET

- Arguments
- Variables
- Fragments
- Aliases
- Unions
- Introspection

Arguments

Request

```
{
  user(login: "foo") {
    name
    Location
    isViewer
  }
}
```

Response

```
{
  "data": {
    "user": {
      "name": "Maciej Pacut",
      "location": null,
      "isViewer": false
    }
  }
}
```

Variables

Request

```
query ($username: String!){
  user(login: $username) {
    name
    location
    isViewer
  }
}
```

Variables

```
{
  "username": "foo"
}
```

Response

```
{
  "data": {
    "user": {
      "name": "Maciej Pacut",
      "location": null,
      "isViewer": false
    }
  }
}
```

Pearson

Fragments

Request

```
{
  viewer {
    ... userFragment
  }
}

fragment userFragment on User {
  name
  location
  isViewer
}
```

Response

```
{
  "data": {
    "viewer": {
      "name": "Arianne",
      "location": "Vancouver, BC,
Canada",
      "isViewer": true
    }
  }
}
```

Aliases

Request

```
{
  viewer {
    name
    place: location
    isViewer
  }
}
```

Response

```
{
  "data": {
    "viewer": {
      "name": "Arianne",
      "place": "Vancouver, BC, Canada",
      "isViewer": true
    }
  }
}
```

Unions

Request

```
{
  search (type: USER, query: "foo", first: 10) {
    nodes {
      ... on User {
        name
        login
        bio
      }
    }
  }
}
```

Introspection

```
{
  __schema {
    queryType {
      name
      kind
      fields {
        name
        type {
          name
          kind
          ofType {
            name
            kind
          }
        }
      }
    }
  }
}
```

Language features

Mutations

REST equivalent: POST, PUT, DELETE

- Type validation
- Return query

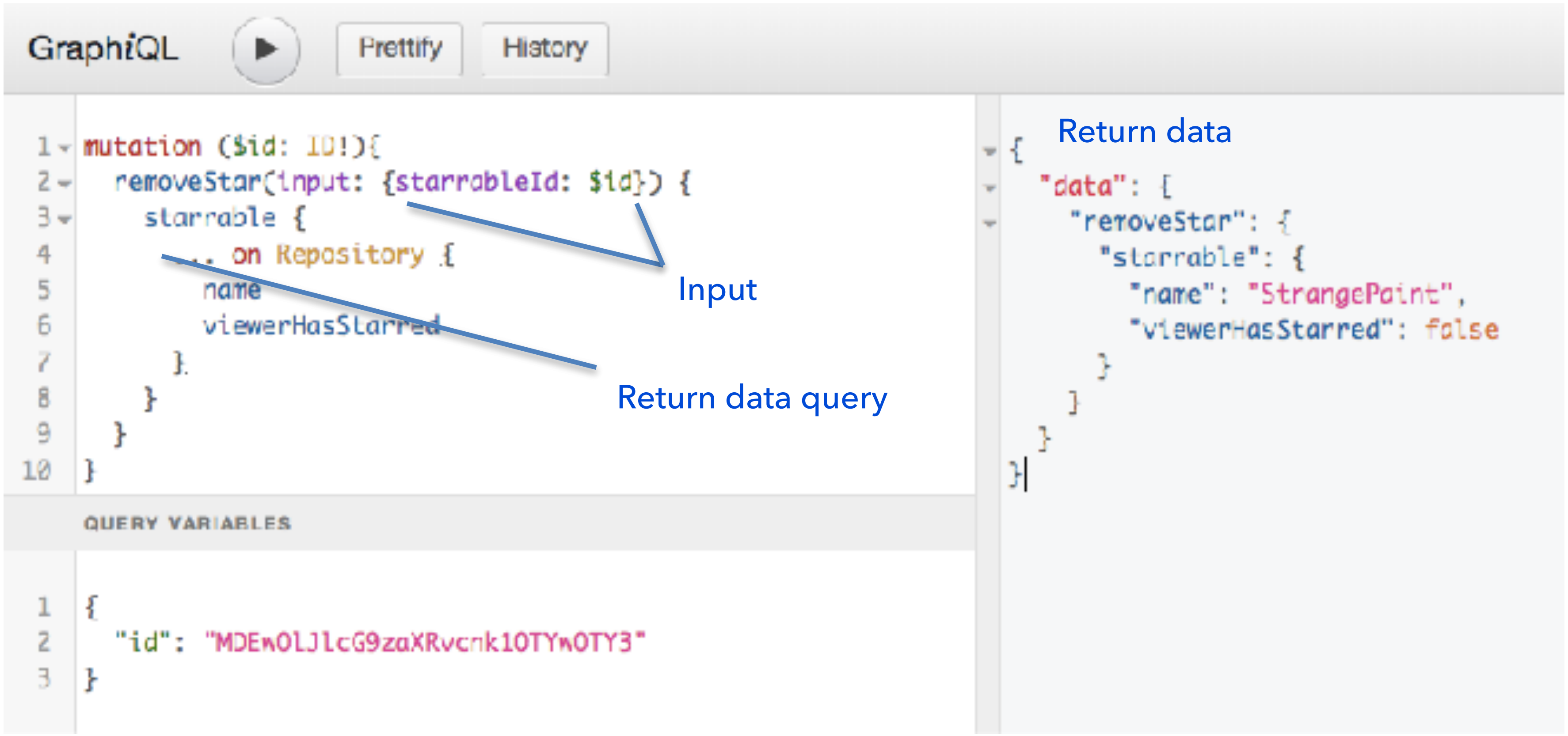
Mutations

Request

```
mutation {
  addStar(input: {starrableId: "1"}) {
    starrable {
      ... on Repository {
        name
        viewerHasStarred
      }
    }
  }
}
```

Response

```
{
  "data": {
    "addStar": {
      "starrable": {
        "name": "StrangePaint",
        "viewerHasStarred": true
      }
    }
  }
}
```



```
mutation ($id: ID!){
  removeStar(input: {starrableId: $id}) {
    starrable {
      ... on Repository {
        name
        viewerHasStarred
      }
    }
  }
}
```

```
{
  "id": "MDEwOlJlcG9zaXRvcnk1OTYwOTY3"
}
```

```
{
  "data": {
    "removeStar": {
      "starrable": {
        "name": "StrangePoint",
        "viewerHasStarred": false
      }
    }
  }
}
```

#3 - Complicated writes

Our solution:

- Send request with entire list contents as input
- Invalid inputs return an error (strongly typed!)
- Backend updates data to match request input

Example mutation

```
mutation update($questionList: QuestionListInput!) {
  updateSurvey (id: 1, questions: $questionList) {
    questions {
      question
      id
    }
  }
}

{
  "questionList": [
    {"question": "1"},
    {"question": "3"},
    {"question": "2"}
  ]
}
```

Example mutation

```
{
  "data": {
    "questions": [
      {"question": "1", "id": 201},
      {"question": "3", "id": 203},
      {"question": "2", "id": 202}
    ]
  }
}
```

Language features

Directives

- @skip
- @include

```
{
  viewer {
    name
    location @include(if: false)
  }
}
```

http://graphql.org/learn/queries/#directives

Language features

Learn GraphQL features

http://graphql.org/learn/

Full spec

http://facebook.github.io/graphql/

Github GraphQL API

https://developer.github.com/v4/explorer/

It's just a query language

Non-spec API features

- Filters
- Ordering
- Pagination

- Mostly up to API designer
- ORM specific
- Python is fairly standardized
- JavaScript has lots of options -> more decisions

Filters

Request

```
{
  viewer {
    repositories (
      isFork: true,
      orderBy: {field: NAME, direction: DESC},
      first: 10
    ) {
      nodes {
        name
      }
    }
  }
}
```

Ordering

Request

```
{
  viewer {
    repositories (
      isFork: true,
      orderBy: {field: NAME, direction: DESC},
      first: 10
    ) {
      nodes {
        name
      }
    }
  }
}
```

Limits

Request

```
{
  viewer {
    repositories (
      isFork: true,
      orderBy: {field: NAME, direction: DESC},
      first: 10
    ) {
      nodes {
        name
      }
    }
  }
}
```

Pagination - Cursor based

Request

```
{
  viewer {
    repositories(first: 10, after: "cursor") {
      pageInfo {
        endCursor
        hasNextPage
     }
      edges {
        cursor
        node {
          name
        }
      }
    }
  }
}
```

GraphQL libraries

- Big list of resources and libraries
  - https://github.com/chentsulin/awesome-graphql
- Cursor-based pagination (using Relay)
  - Understanding Relay pagination
  - Spec
- Offset-based pagination
  - Example API

Security

- Authentication
- Authorization
- Limit large requests
- Throttling

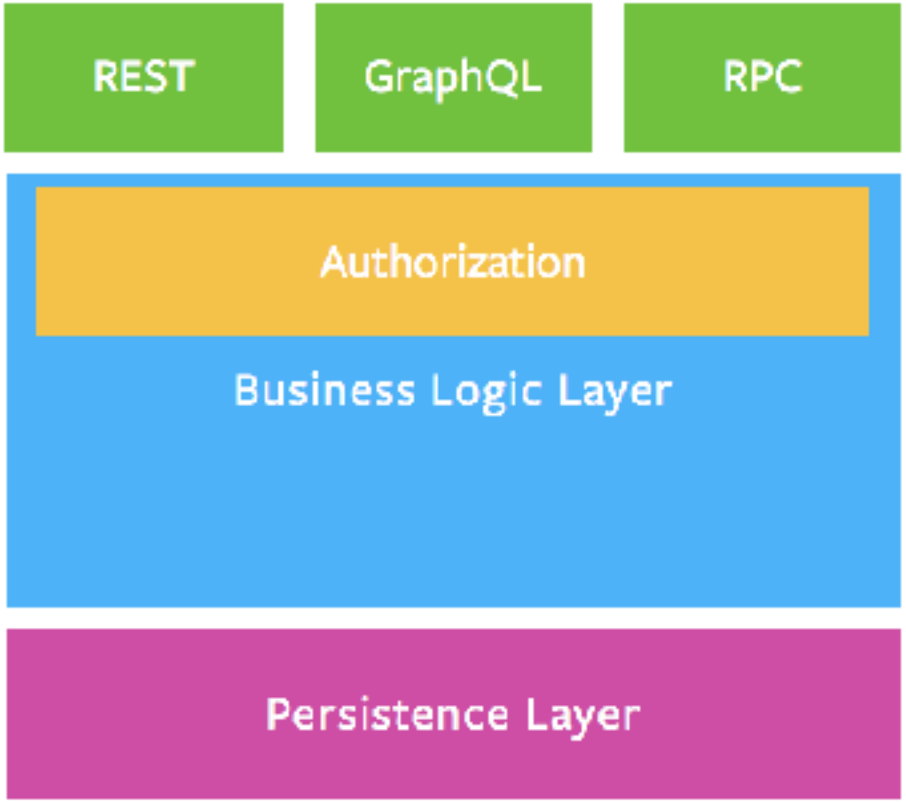
https://www.howtographql.com/advanced/4-security/

Authentication

- Verify logged in user
- Use middleware to authenticate user

Authorization

- Only show data that user can see
- Once you have authentication, user gets sent with request data to schema
- Filter resources based on the authenticated user



http://graphql.org/learn/thinking-in-graphs/#business-logic-layer

Limit large requests

- Whitelist
- Timeout
- Maximum node limit
- Maximum query depth
- Query complexity

Throttling

https://www.howtographql.com/advanced/4-security/

- Based on server time
- Based on query complexity

Security resources

How to GraphQL

https://www.howtographql.com/advanced/4-security/

GitHub

https://developer.github.com/v4/guides/resource-limitations/

Other cool features to look up

- Data Loader
  - Helps cache and minimize query calls
- Create GraphQL schema from REST API
- Schema stitching
  - Combine schemas from different services
- Mocking
- API usage stats
  - Apollo Engine

Questions to consider

- What tasks have been difficult to solve using REST?
- How can my team benefit from using GraphQL?
- What challenges might we face in adopting GraphQL?

Question & Answer

Let's make a client

Using HTML & JavaScript

Let's make stuff

- Go to https://github.com/ariannedee/rethinking-rest
- In your terminal/shell, navigate to where you want to save your project code
- Clone the repository

```
git clone https://github.com/ariannedee/rethinking-
rest.git
```

Pearson

Let's make stuff

- Open the **rethinking-rest/client** folder in your favourite code editor for JavaScript

- Open the file **rethinking-rest/client/index.html** in a browser
  - Supports ES6 syntax
  - Recent version of Chrome, Firefox, Safari, or Edge
  - Not IE

Client - Tech stack

- HTML / CSS
- JavaScript, some ES6 syntax
- JQuery requests to GitHub v4 API

Client - GraphQL Features

- Query
  - Authentication
  - Error handling
- Total count
- Filtering
- Pagination
- Variables
- Mutations

Client - Project tasks

1. Update header to say "Hello {your name}"
2. List your repositories
3. Update the list to be ordered by most recently created
4. When you click on each repo, display its stats on the right side of the page
5. Add functionality to star/un-star a repository

Client - Project tasks

1. **Update header to say "Hello {your name}"**
2. List your repositories
3. Update the list to be ordered by most recently created
4. When you click on each repo, display its stats on the right side of the page
5. Add functionality to star/un-star a repository

Client - Query format

- **Endpoint**
  - https://api.github.com/graphql
- **Method**
  - POST
- **Content type**
  - "application/json"
- **Request header**
  - "Authorization: bearer **token**"
- **Data (JSON.stringified)**
  - query: your query
  - variables: your variables object

Client - Authorization

- Create a personal access token
  - https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/

  - repo: public_repo
  - repo (all)
    - if you want to see private repos
    - don't share this key

Client - Documentation

Official GraphQL client documentation

http://graphql.org/graphql-js/graphql-clients/

Client - Project tasks

1. **Update header to say "Hello {your name}"**
2. List your repositories
3. Update the list to be ordered by most recently created
4. When you click on each repo, display its stats on the right side of the page
5. Add functionality to star/un-star a repository

Client - Project tasks

1. Update header to say "Hello {your name}"
2. **List your repositories**
3. Update the list to be ordered by most recently created
4. When you click on each repo, display its stats on the right side of the page
5. Add functionality to star/un-star a repository

Client - Project tasks

1. Update header to say "Hello {your name}"
2. List your repositories
3. **Update the list to be ordered by most recently created**
4. When you click on each repo, display its stats on the right side of the page
5. Add functionality to star/un-star a repository

Client - Project tasks

1. Update header to say "Hello {your name}"
2. List your repositories
3. Update the list to be ordered by most recently created
4. **When you click on each repo, display its stats on the right side of the page**
5. Add functionality to star/un-star a repository

Client - Project tasks

1. Update header to say "Hello {your name}"
2. List your repositories
3. Update the list to be ordered by most recently created
4. When you click on each repo, display its stats on the right side of the page
5. **Add functionality to star/un-star a repository**

Used features

- Queries
- Mutations
- Arguments
- Variables
- Fragments
- Aliases
- Unions

Features not covered

- Pagination
- Introspection
- Directives
- Subscriptions

Question & Answer

Let's build a server

Using Node.js or Django

What we'll cover

- Create **queryable** schema
  - nodes and edges
- Accept **arguments**
  - filtering, ordering, and formatting
- Support **pagination**
  - Relay-style
- Support **mutations**
  - create, update, and delete data

Server

- Setup project
- Setup GraphQL
- Define queries (GET)
- Add filters
- Define mutations (PUT/POST/DELETE)
- Add pagination

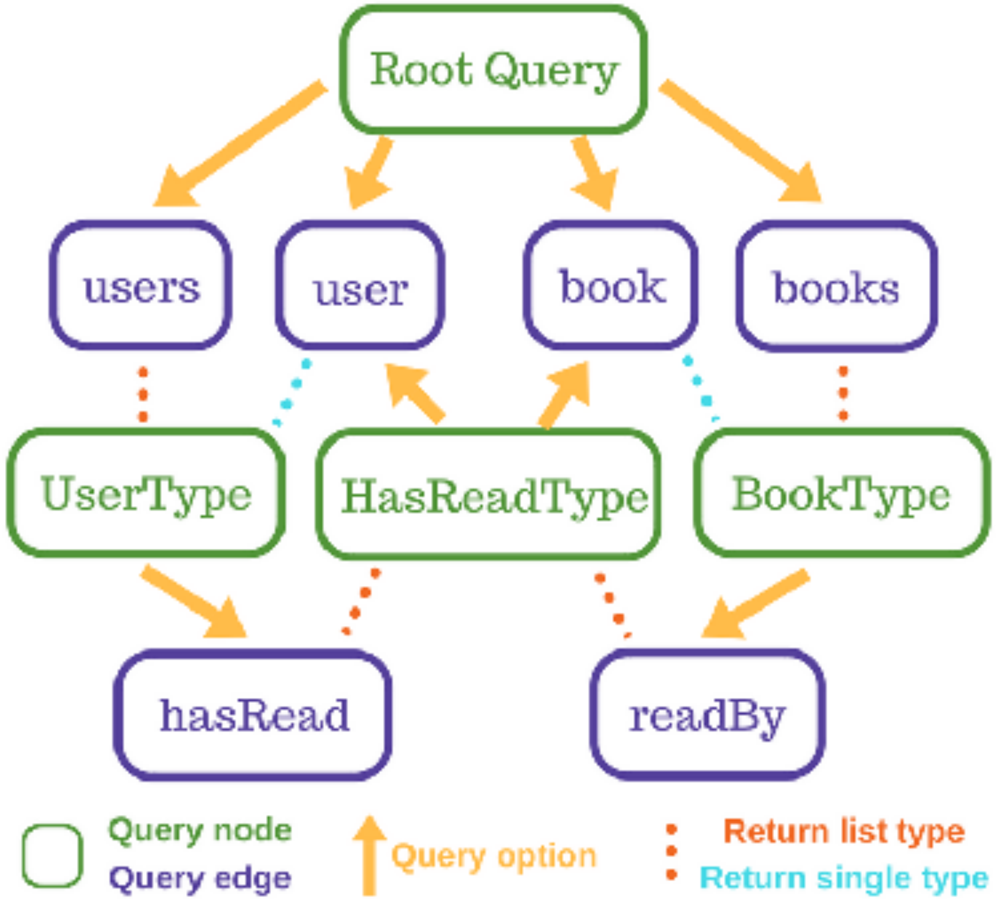
API model

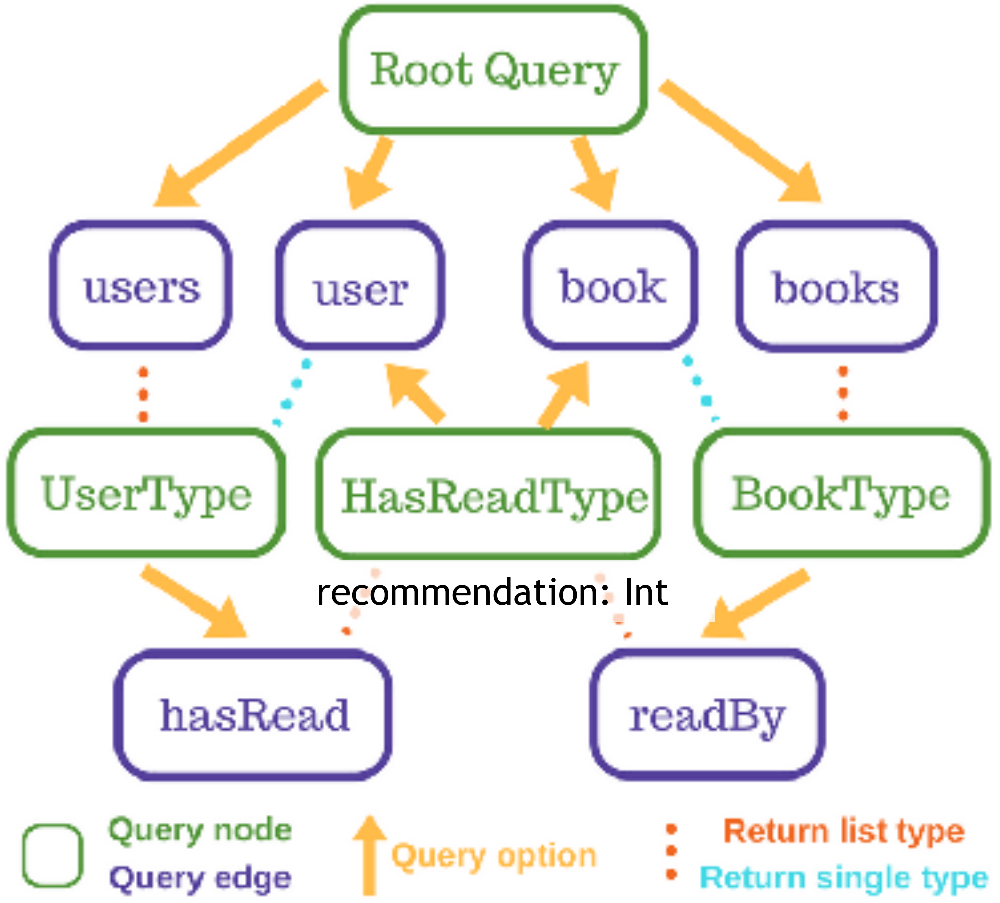
Objects

- Users
- Books

Relationships

- User - hasRead - Book
- Book - has - Category


Root Query
users
user
book
books
UserType
HasReadType
BookType
hasRead
readBy
Query node
Query edge
Query option
Return list type
Return single type



id: ID
username: String
isAdmin: Boolean

recommendation: Int

id: ID
title: String
author: String
fiction: Boolean
publishedYear: Int

Database ORMs

- Node
  - Knex.js - http://knexjs.org/#Builder

- Django
  - Django ORM - https://docs.djangoproject.com/en/2.0/ref/models/querysets/

Survey

- What framework will you be following along with?
    - Node, Django, both, none, another framework

Tutorials

- Node w/ Express
- https://graphql.org/graphql-js/

- Graphene-django
- http://docs.graphene-python.org/projects/django/en/latest/tutorial-plain/

Server

- **Setup project**
- Setup GraphQL
- Define queries (GET)
- Add filters
- Define mutations (PUT/POST/DELETE)
- Add pagination

Server - Node setup

- You should have installed Node.js > 8.9

- In your terminal/shell, go to the **node_server/graphqlAPI/** folder

```
cd rethinking-rest/node_server/project
```

- Install the required packages

```
npm install
```

- Start the server

```
npm start
```

- Go to **localhost:3000/**

Server - Django setup

- In your terminal/shell, go to the **django_server/graphql-api/** folder

```
cd rethinking-rest/django_server/project
```

- Install pipenv (if you don't already have it)

```
pip install pipenv
```

- Install the required packages

```
pipenv install
```

- Start the server

```
pipenv shell
python manage.py runserver
```

- Go to **localhost:8000/**

Pearson

Server

- Setup project
- **Setup GraphQL**
- Define queries (GET)
- Add filters
- Define mutations (PUT/POST/DELETE)
- Add pagination

GraphQL Server library options

- Node
  - GraphQL.js

```
npm install graphql
npm install express-graphql
```

- Django
  - Graphene

```
pipenv install graphene
pipenv install graphene-django
```

Setup GraphQL - Node

In app.js

```
var graphqlHTTP = require('express-graphql');
var schema = require('./src/schema');

// after app=express();
app.use('/graphql', graphqlHTTP({
  schema: schema,
  graphiql: true
}));
```

Setup GraphQL - Node

In src/schema.js

```
var graphql = require('graphql');

// Define the Query type
var queryType = new graphql.GraphQLObjectType({
  name: 'Query',
  fields: {
    hello: {
      type: graphql.GraphQLString,
      resolve () {
        return 'world';
      }
    }
  }
});

// Define the Schema type with the given query type
var schema = new graphql.GraphQLSchema({query: queryType});
module.exports = schema;
```

Go to localhost:3000/graphql

dj

Pearson

Setup GraphQL - Django

In settings.py

```
# add to INSTALLED_APPS
'graphene_django',

GRAPHENE = {
    'SCHEMA': 'app.schema.schema'
}
```

In urls.py

```
from graphene_django.views import GraphQLView

# add to urlpatterns
path('graphql', GraphQLView.as_view(graphiql=True))
```

Setup GraphQL - Django

In schema.py

```
import graphene

class Query(graphene.ObjectType):
    hello = graphene.String()

    def resolve_hello(self, info):
        return "world"

schema = graphene.Schema(query=Query)
```

Go to localhost:8000/graphql

Pearson

Server

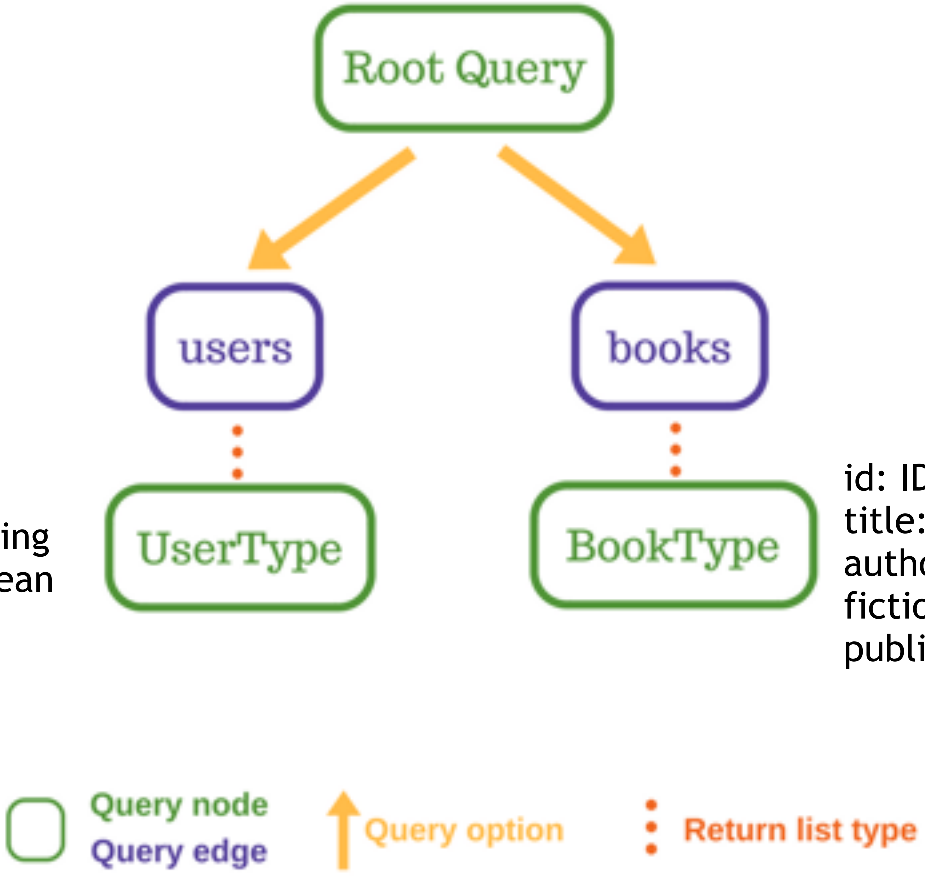
- Setup project
- Setup GraphQL
- **Define queries (GET)**
- Add filters
- Define mutations (PUT/POST/DELETE)
- Add pagination

Define queries (GET)

- All books and users
- Books that a user has read
- Users that have read a book
- Individual user or book
- Each user's average book rating

GraphQL Concepts

- Schema
- Types
    - Object
    - Scalar
        - Int, String, ID, etc...
    - List
- Fields
- Resolvers

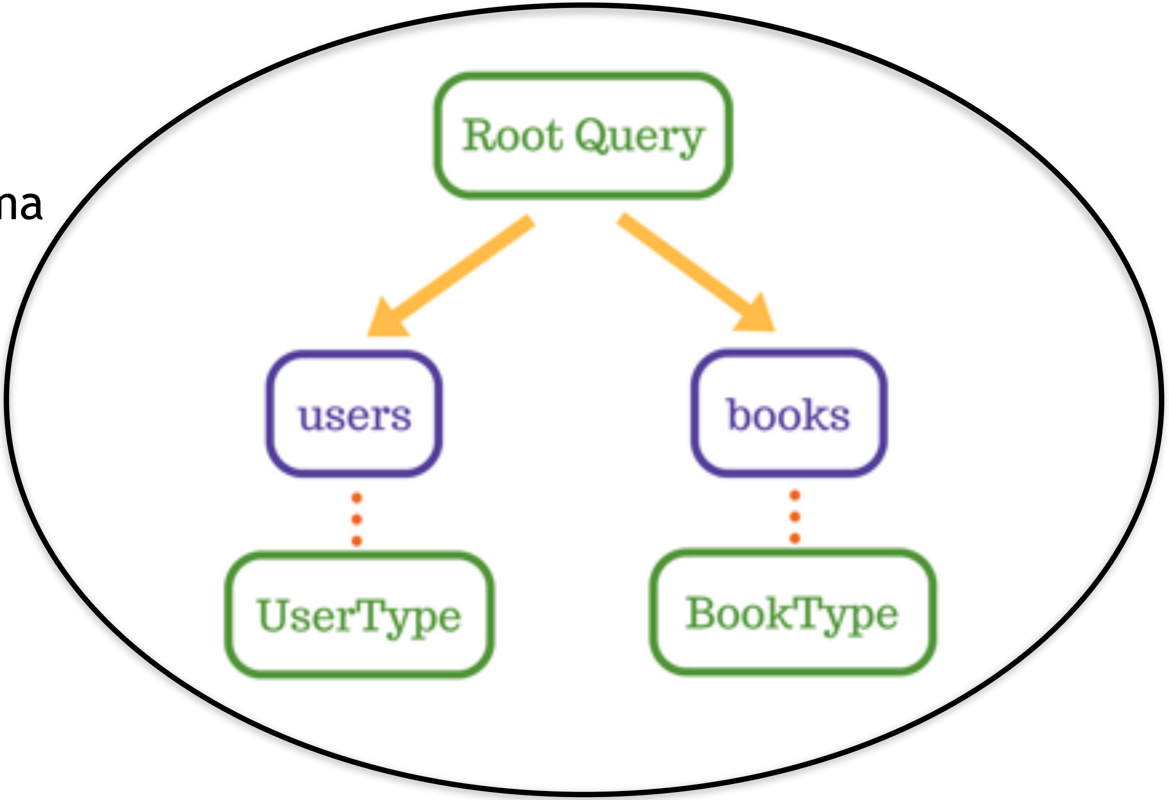


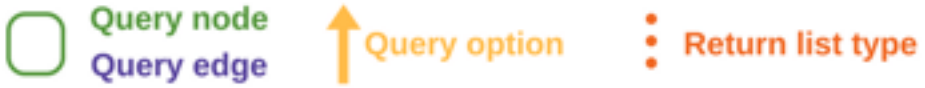
id: ID
username: String
isAdmin: Boolean

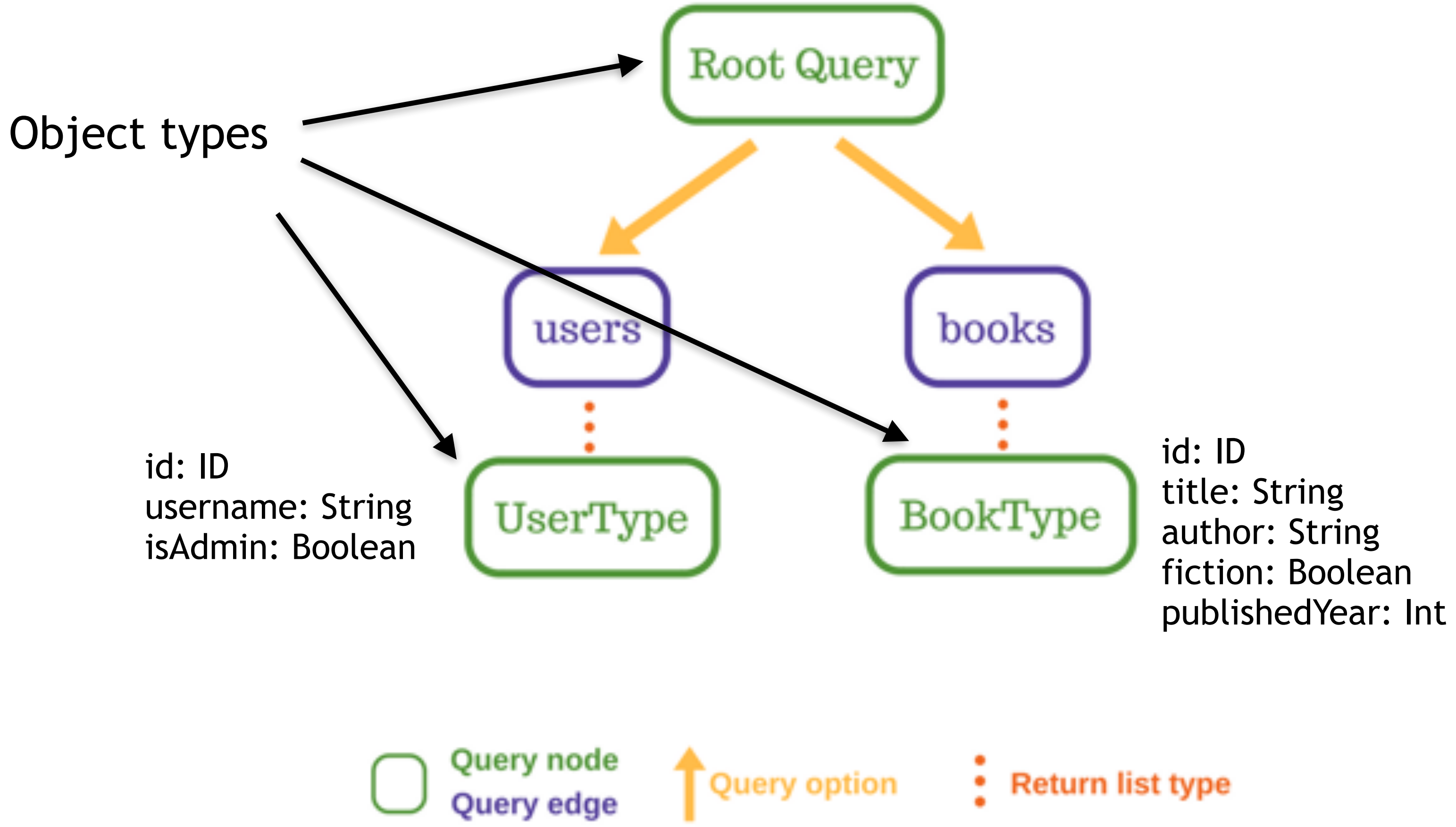
id: ID
title: String
author: String
fiction: Boolean
publishedYear: Int

Pearson

Schema

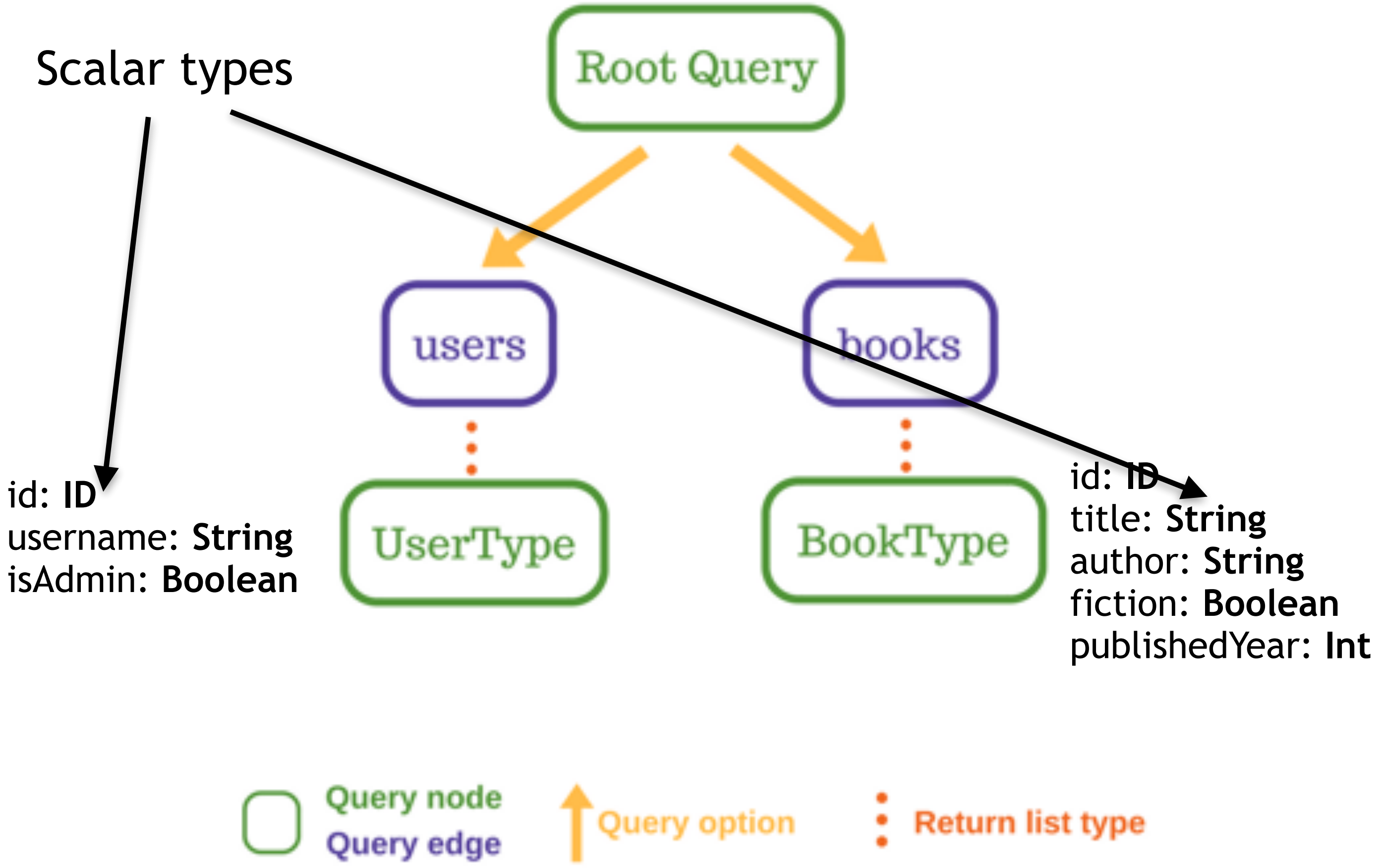

Root Query
users
books
UserType
BookType

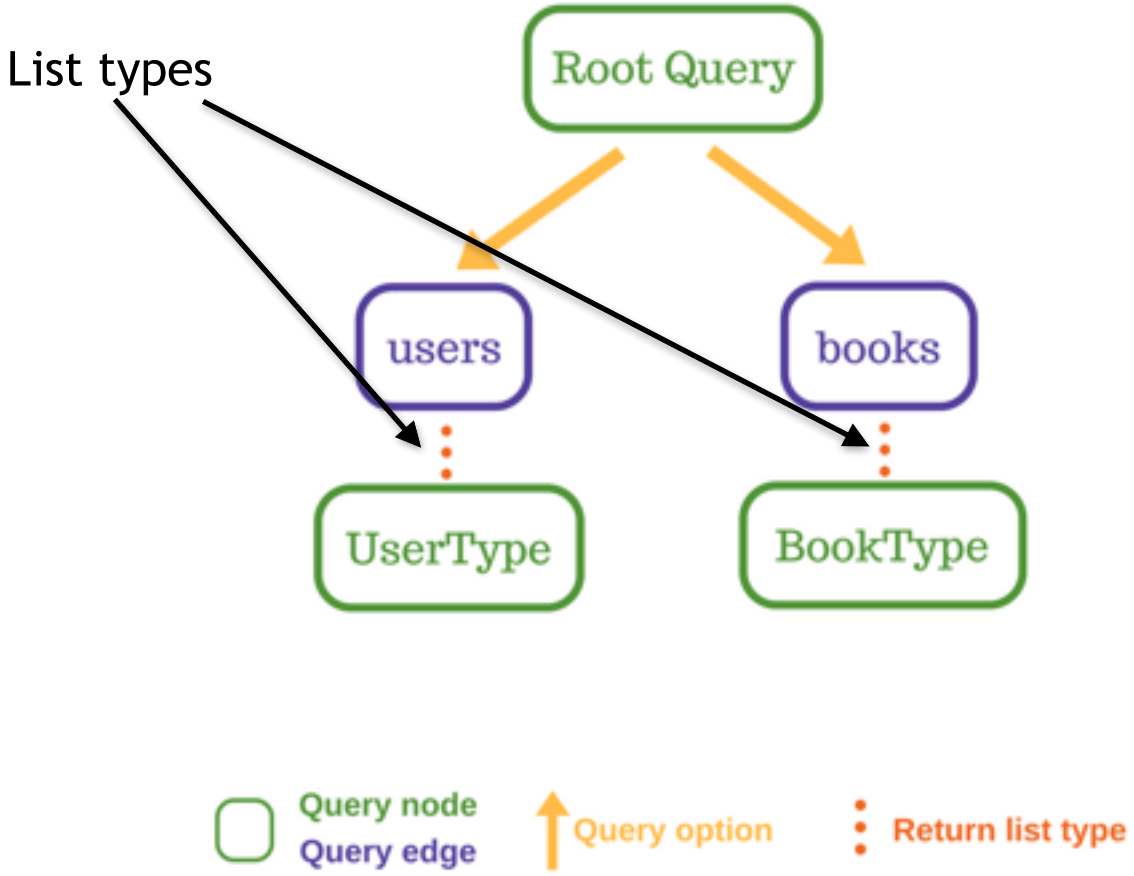

Query node
Query edge
Query option
Return list type

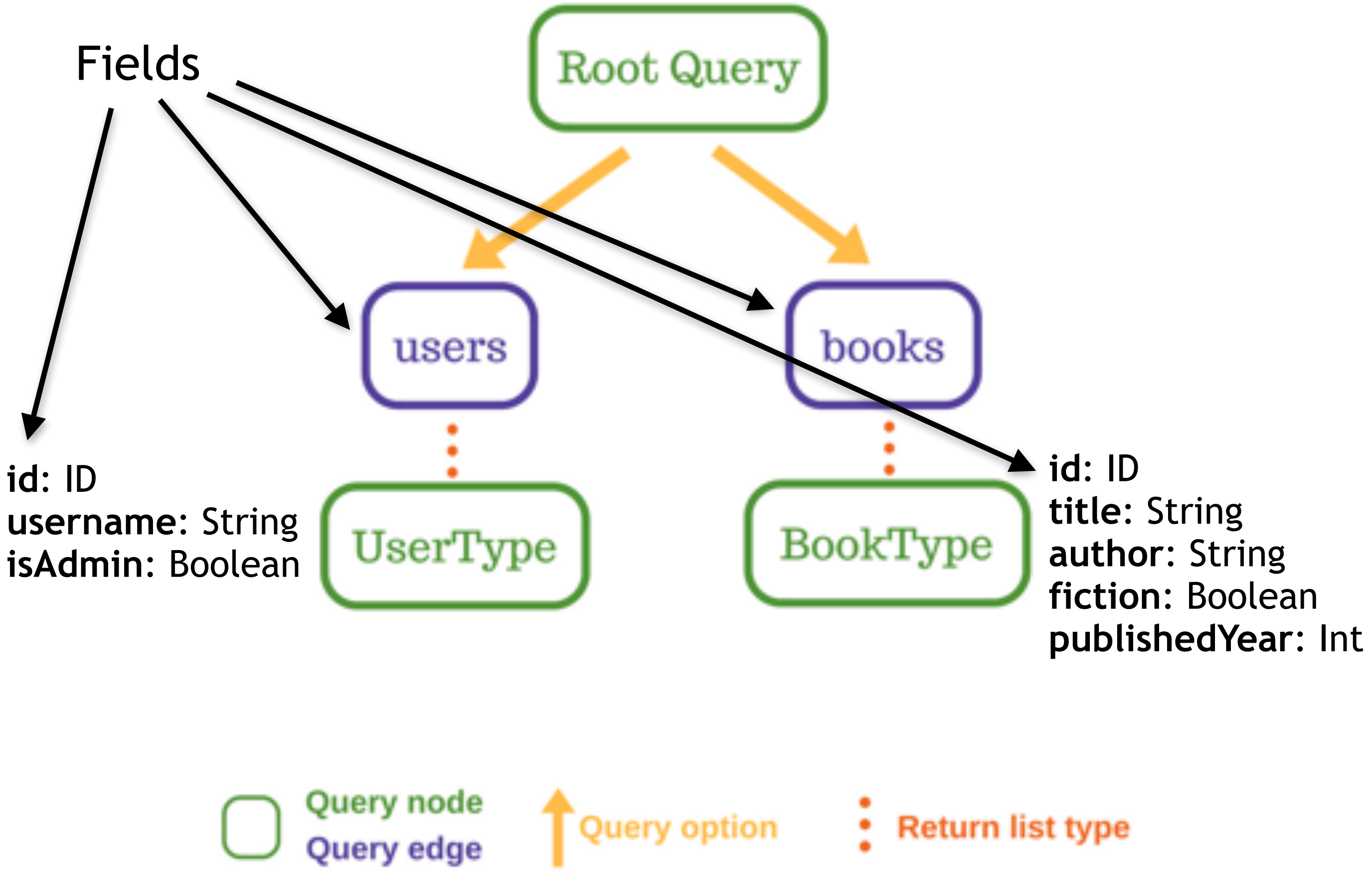


id: ID
username: String
isAdmin: Boolean

id: ID
title: String
author: String
fiction: Boolean
publishedYear: Int

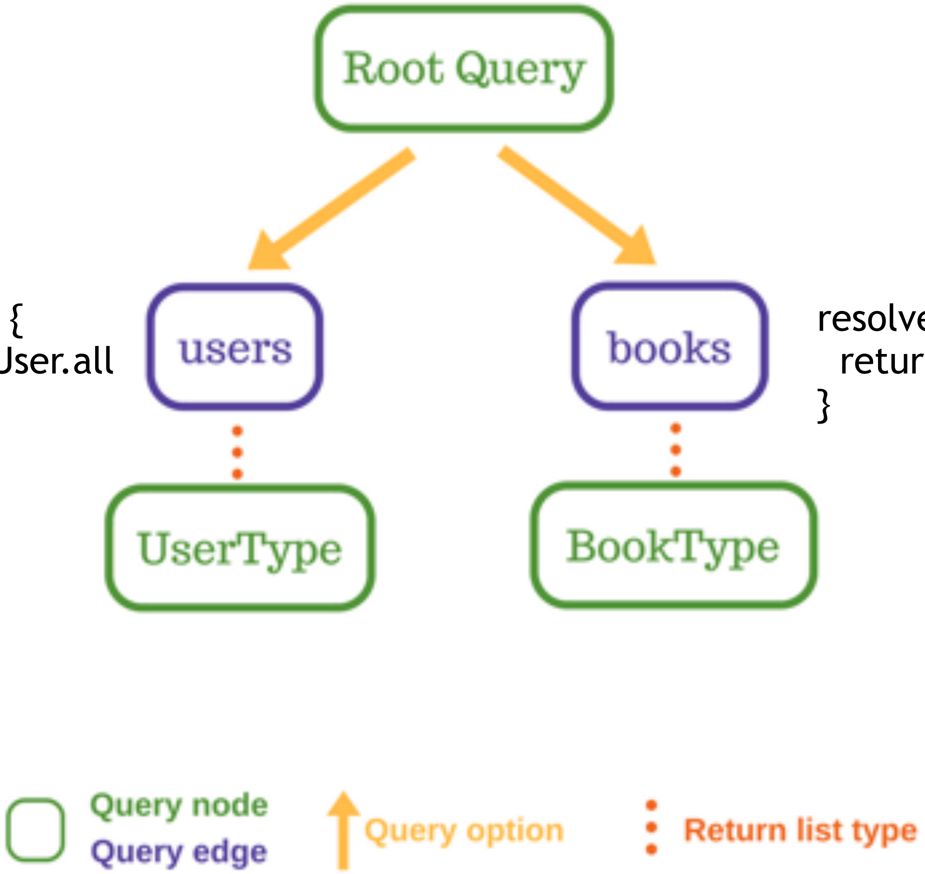

Scalar types
Root Query
users
books
UserType
BookType
id: **ID**
username: **String**
isAdmin: **Boolean**
id: **ID**
title: **String**
author: **String**
fiction: **Boolean**
publishedYear: **Int**
Query node
Query edge
Query option
Return list type


List types
Root Query
users
books
UserType
BookType
Query node
Query edge
Query option
Return list type


Fields
Root Query
users
books
UserType
BookType
id: ID
username: String
isAdmin: Boolean
id: ID
title: String
author: String
fiction: Boolean
publishedYear: Int
Query node
Query edge
Query option
Return list type

Resolvers

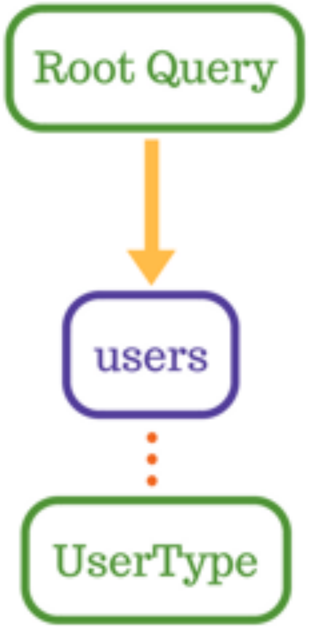
```
resolve() {
  return User.all
}
```



```
resolve() {
  return Book.all
}
```

Define queries (GET)

- **All books and users**
- Books that a user has read
- Users that have read a book
- Individual user or book
- Each user's average book rating


Root Query
users
UserType

Query node
Query edge
Query option
Return list type

Define queries - Node

```
// Define the User type
const UserType = new graphql.GraphQLObjectType({
  name: 'User',
  description: 'This represents a User',
  fields: {
    id: {
      type: graphql.GraphQLID,
      resolve(user) {
        return user.id;
      }
    },
    // ... more fields here
  }
});
```

Define queries - Node

```
// Define the Query type
var queryType = new graphql.GraphQLObjectType({
  name: 'Query',
  fields: {
    users: {
      type: new graphql.GraphQLList(UserType),
      description: 'A list of users',
      resolve(root, args, context) {
        return [{id: 1, username: 'admin'}];  // return fake user
      }
    }
  }
});
```

Define queries - Node

```
// Define the Schema type with the given query type
var schema = new graphql.GraphQLSchema({query:
queryType});

module.exports = { schema };
```

Test to see if it works

Define queries - Node

Return real users from database

```
var knex = require('../db');

...
users: {
  type: new graphql.GraphQLList(UserType),
  description: 'A list of users',
  async resolve(root, args, context) {
    let query = knex('user');
    return await query;
  }
}
```

Knex cheatsheet: https://devhints.io/knex

Define queries - Django

```python
import graphene
import graphene_django
from django.contrib.auth.backends import UserModel

class UserType(graphene_django.DjangoObjectType):
    class Meta:
        model = UserModel

class Query(graphene.ObjectType):
    users = graphene.List(UserType)

    def resolve_users(self, info):
        return UserModel.objects.all()
```

Test to see if it works

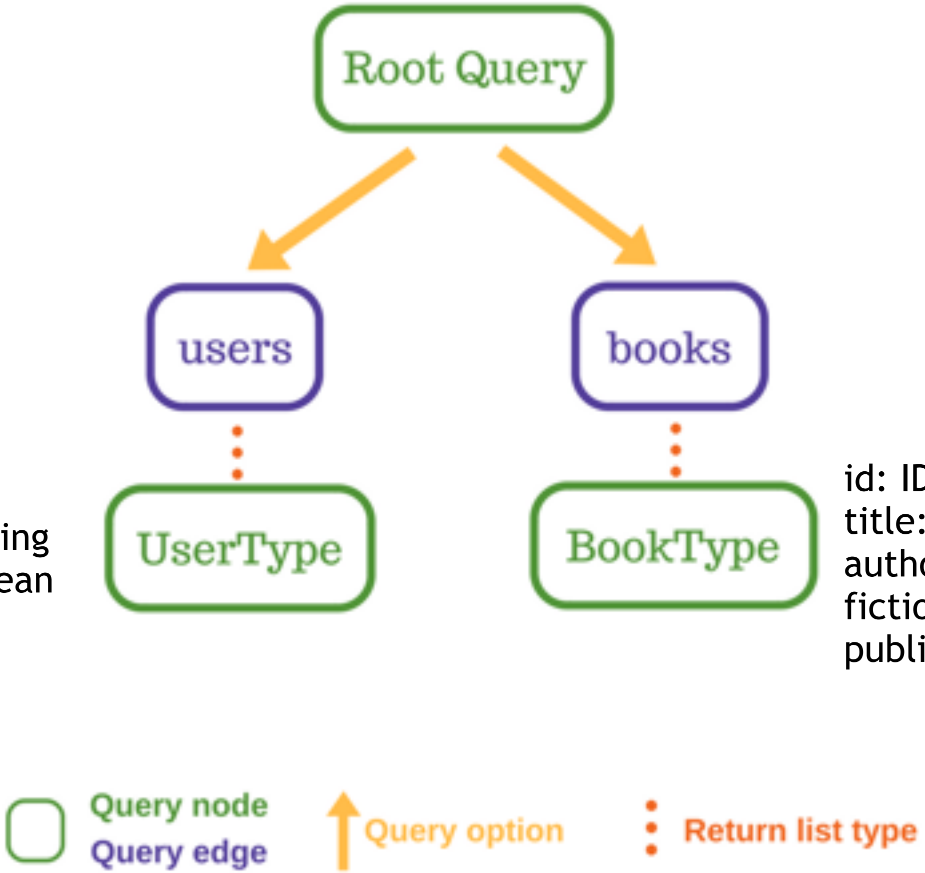
Define queries - custom resolver

```
class UserNode(graphene_django.DjangoObjectType):
  is_admin = graphene.Boolean()

  def resolve_is_admin(self, info):
    return self.is_staff

  class Meta:
    model = UserModel
    only_fields = ('id', 'username')
```

Pearson

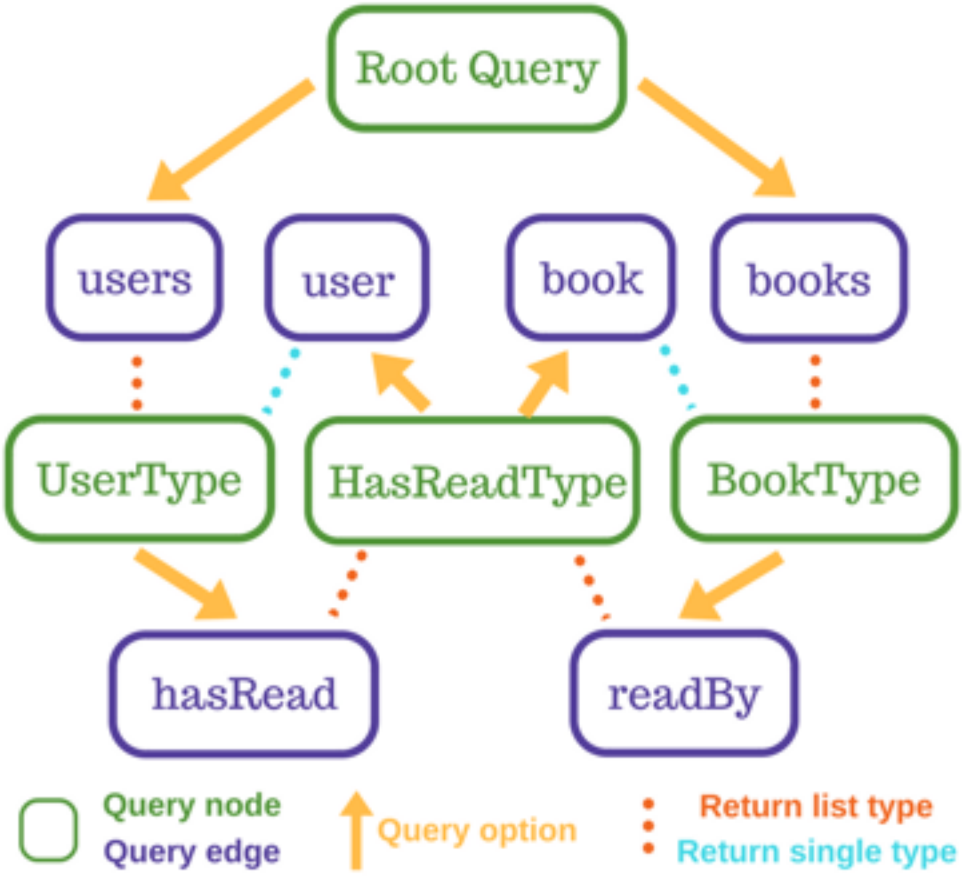


id: ID
username: String
isAdmin: Boolean

id: ID
title: String
author: String
fiction: Boolean
publishedYear: Int

Define queries (GET)

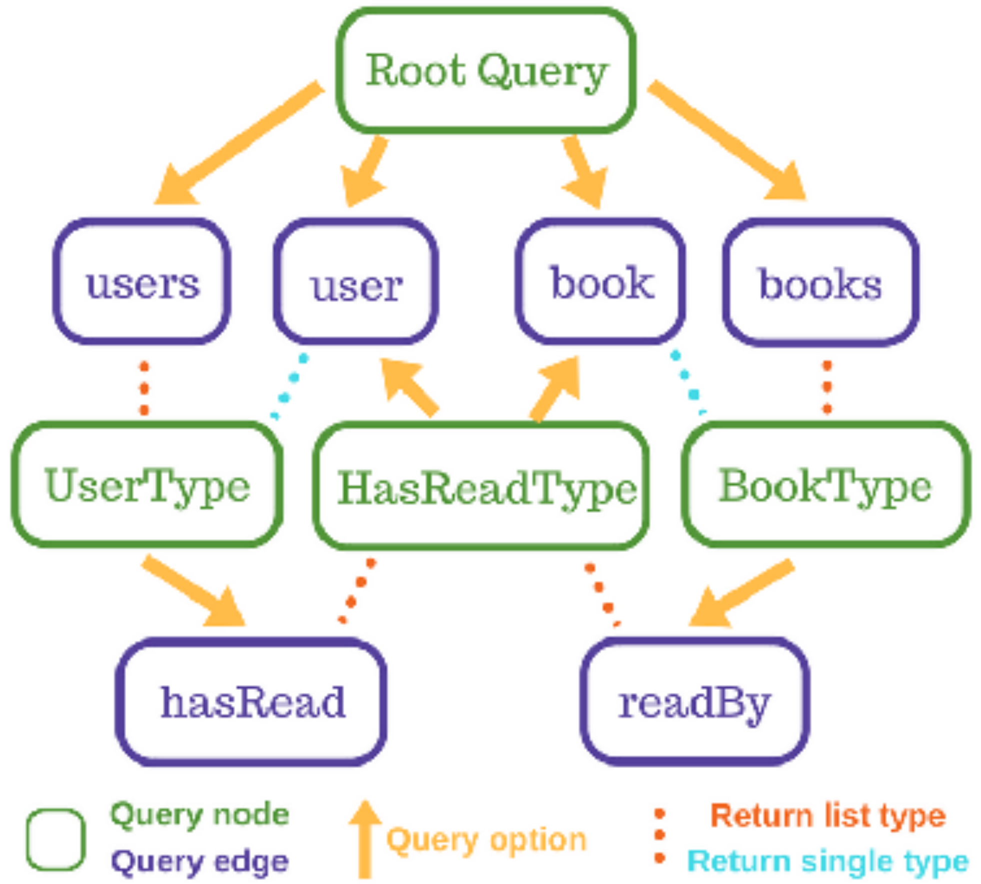
- All books and users
- **Books that a user has read**
- **Users that have read a book**
- Individual user or book
- Each user's average book rating


Root Query
users
user
book
books
UserType
HasReadType
BookType
hasRead
readBy
Query node
Query edge
Query option
Return list type
Return single type

Pearson

Define queries (GET)

- All books and users
- Books that a user has read
- Users that have read a book
- **Individual user or book**
- Each user's average book rating


Root Query
users
user
book
books
UserType
HasReadType
BookType
hasRead
readBy
Query node
Query edge
Query option
Return list type
Return single type

Define queries (GET)

- All books and users
- Books that a user has read
- Users that have read a book
- Individual user or book
- **Each user's average book rating**

Other features

- Enums
- Unions
- Interfaces

Server

- Setup project
- Setup GraphQL
- Define queries (GET)
- **Add filters**
- Define mutations (PUT/POST/DELETE)
- Add pagination

Pearson

Server

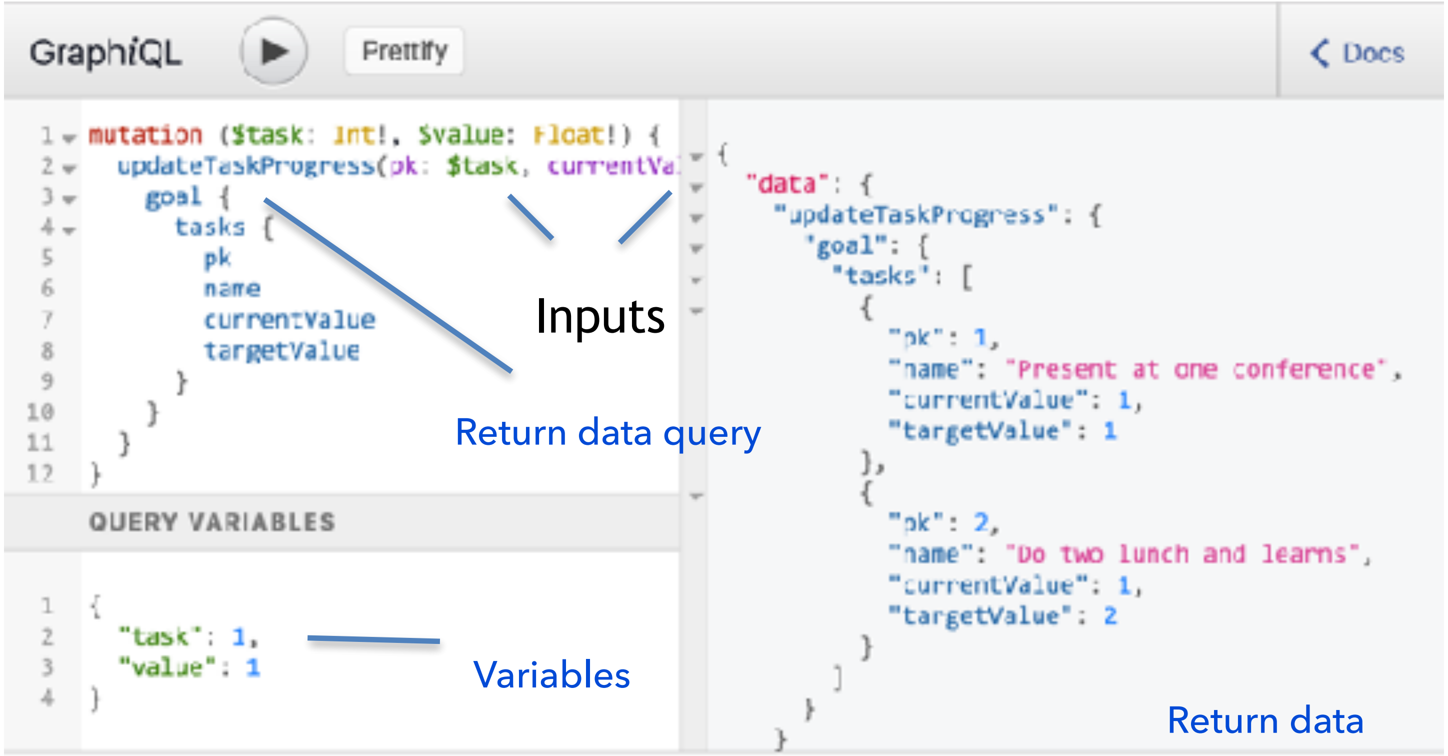
- Setup project
- Setup GraphQL
- Define queries (GET)
- Add filters
- **Define mutations (PUT/POST/DELETE)**
- Add pagination

Authenticate

- Node
  - Passport: http://www.passportjs.org/
  - Express JWT: https://github.com/auth0/express-jwt
  - Express Session: https://github.com/expressjs/session
- Django
  - Built-in with auth 👍

Define mutations

- Read/rate a book
- Update a book's rating
- Remove a book from your list of read books



```
mutation ($task: Int!, $value: Float!) {
  updateTaskProgress(pk: $task, currentVa
    goal {
      tasks {
        pk
        name
        currentValue
        targetValue
      }
    }
  }
}
```

```
{
  "task": 1,
  "value": 1
}
```

```
{
  "data": {
    "updateTaskProgress": {
      "goal": {
        "tasks": [
          {
            "pk": 1,
            "name": "Present at one conference",
            "currentValue": 1,
            "targetValue": 1
          },
          {
            "pk": 2,
            "name": "Do two lunch and learns",
            "currentValue": 1,
            "targetValue": 2
          }
        ]
      }
    }
  }
}
```

Server

- Setup project
- Setup GraphQL
- Define queries (GET)
- Add filters
- Define mutations (PUT/POST/DELETE)
- **Add pagination**

Pagination

- Basic principles: http://graphql.org/learn/pagination/
- **Node:**
  - relay-js
    - https://github.com/graphql/graphql-relay-js
  - Build-it-yourself tutorial
    - https://medium.com/@mattmazzola/graphql-pagination-implementation-8604f77fb254
- **Django:**
  - graphene-relay
    - http://docs.graphene-python.org/en/latest/relay/

That's it!

Q&A

Fill out feedback survey

Resources

- GraphQL resource list (GitHub)
- GraphQL.js documentation
- GraphQL specs

Overview

- Zero to GraphQL (video)
- Intro to GraphQL (blog post)

Pearson

Advanced features

- Security - GitHub
- Pagination
- GraphQL in the Wild - video
  - My DjangoCon talk on supporting GraphQL in production

Tutorials

- How to GraphQL
  - Lots of different server options
- Apollo full-stack tutorial
  - React + Node
  - Includes subscriptions
- Graphene-Django
- Node + Express

Opinions

- GraphQL vs REST overview

Pearson