Set up projects

- 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`
- Follow server instructions in README for either Node or Django

Rethinking REST

A hands-on guide to GraphQL and queryable APIs

Survey - About you

- What is your GraphQL knowledge
  - Total new to it, played around with API, tried to implement it (client or server), used in production (client or server)
- What is your role (single)
  - Front-end developer, back-end developer, full-stack, project manager, other [say in group chat]

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
- Safari Live Trainings - May 2018 - present

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

- 10 mins at end of each section (3 total) followed by a 5 minute break
- Use the Q&A feature
- A few questions read out loud & answered
- Can use group chat to ask each other questions during session

First thing first, set up your projects

Set up projects

- 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`
- Follow server instructions in README for either Node or Django

Let's talk about REST

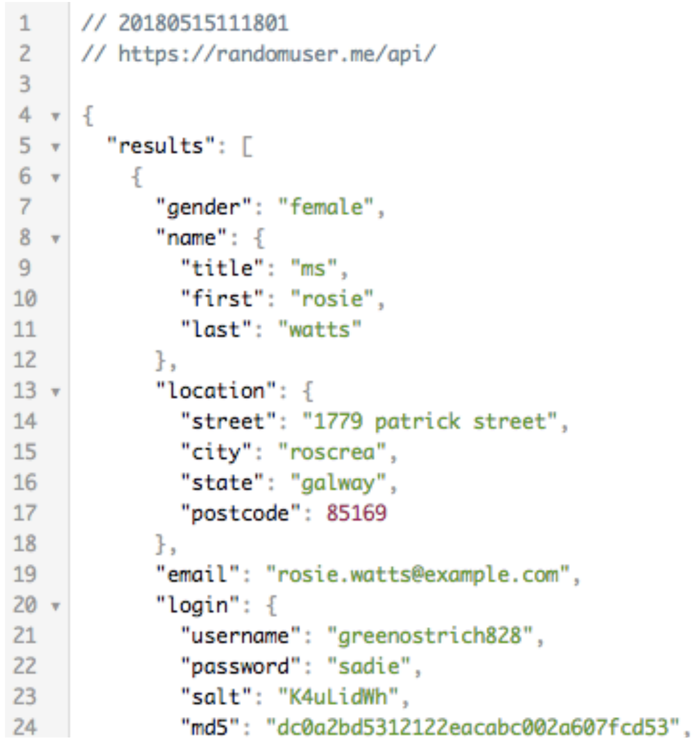
RESTful APIs

- 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": "greenostrich828",
        "password": "sadie",
        "salt": "K4uLidWh",
        "md5": "dc0a2bd5312122eacabc002a607fcd53",
```

Desktop vs Mobile API design

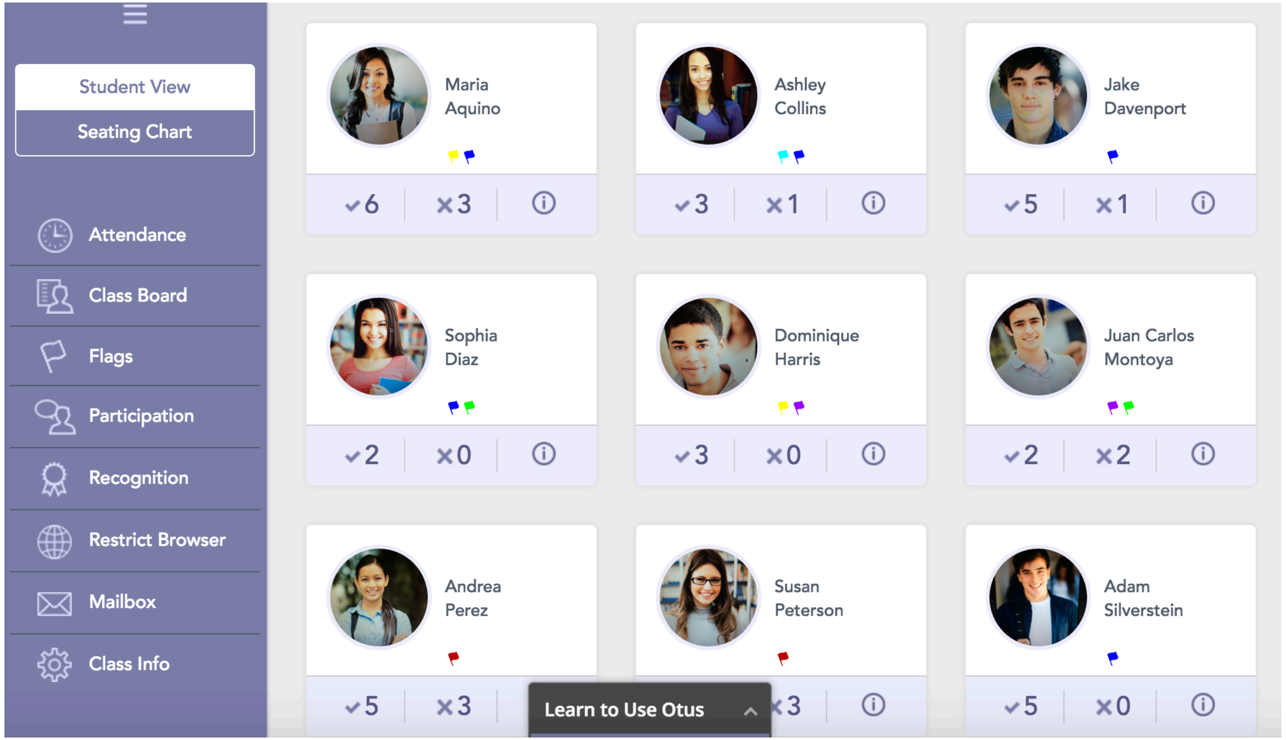
By Muhammad Rafizeldi [CC BY-SA 3.0], from Wikimedia Commons

Challenge #2:
Under-fetching

RESTful API - related resources

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

Summaries


Student View

Seating Chart

Attendance

Class Board

Flags

Participation

Recognition

Restrict Browser

Mailbox

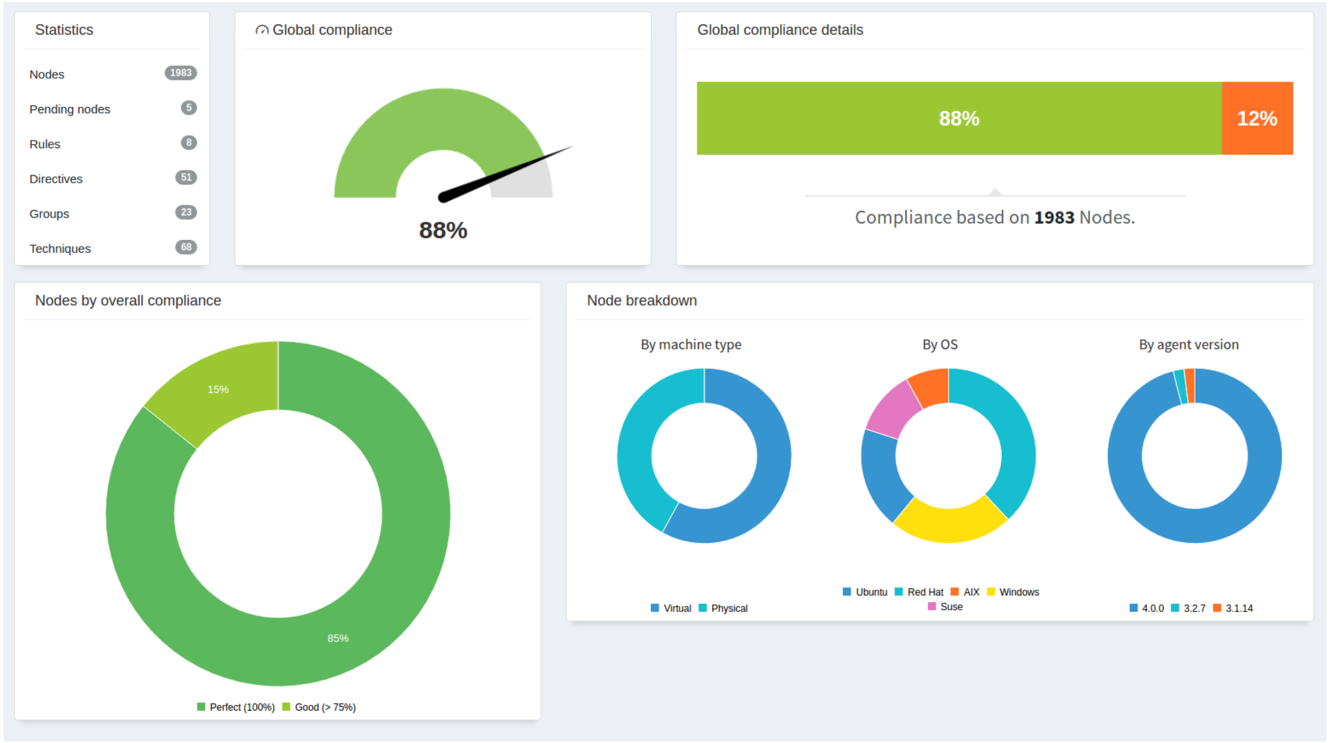
Class Info

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	ⓘ

Learn to Use Otus

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

Dashboards



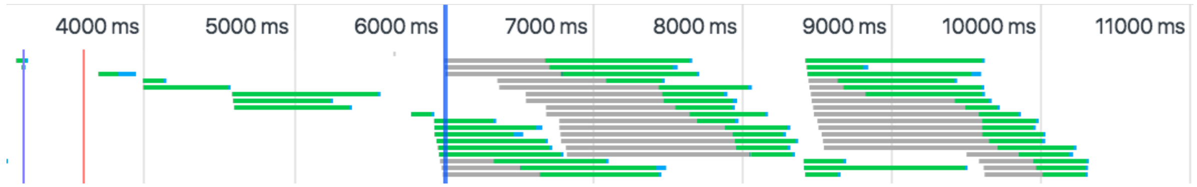
By Moostik [CC BY-SA 4.0], from Wikimedia Commons

RESTful API - related resources

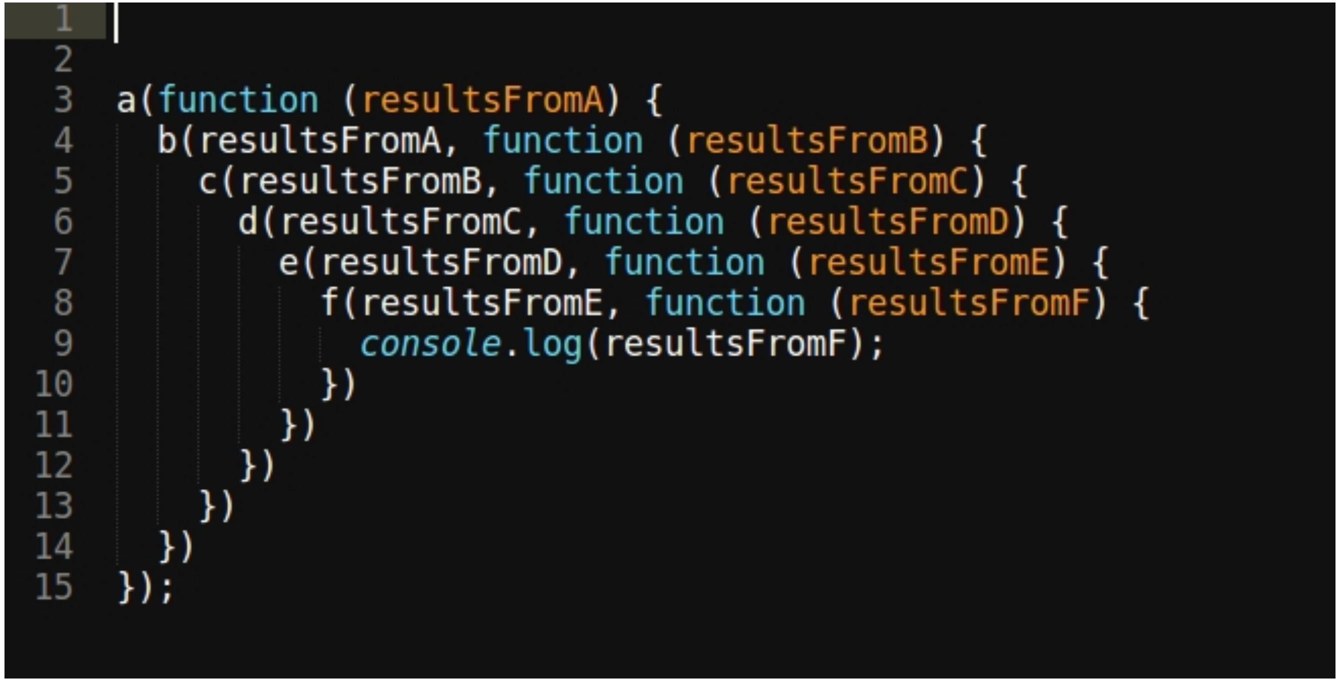
- /api/user/1/
- /api/user/1/teams/
- /api/team/100/members/

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

Performance



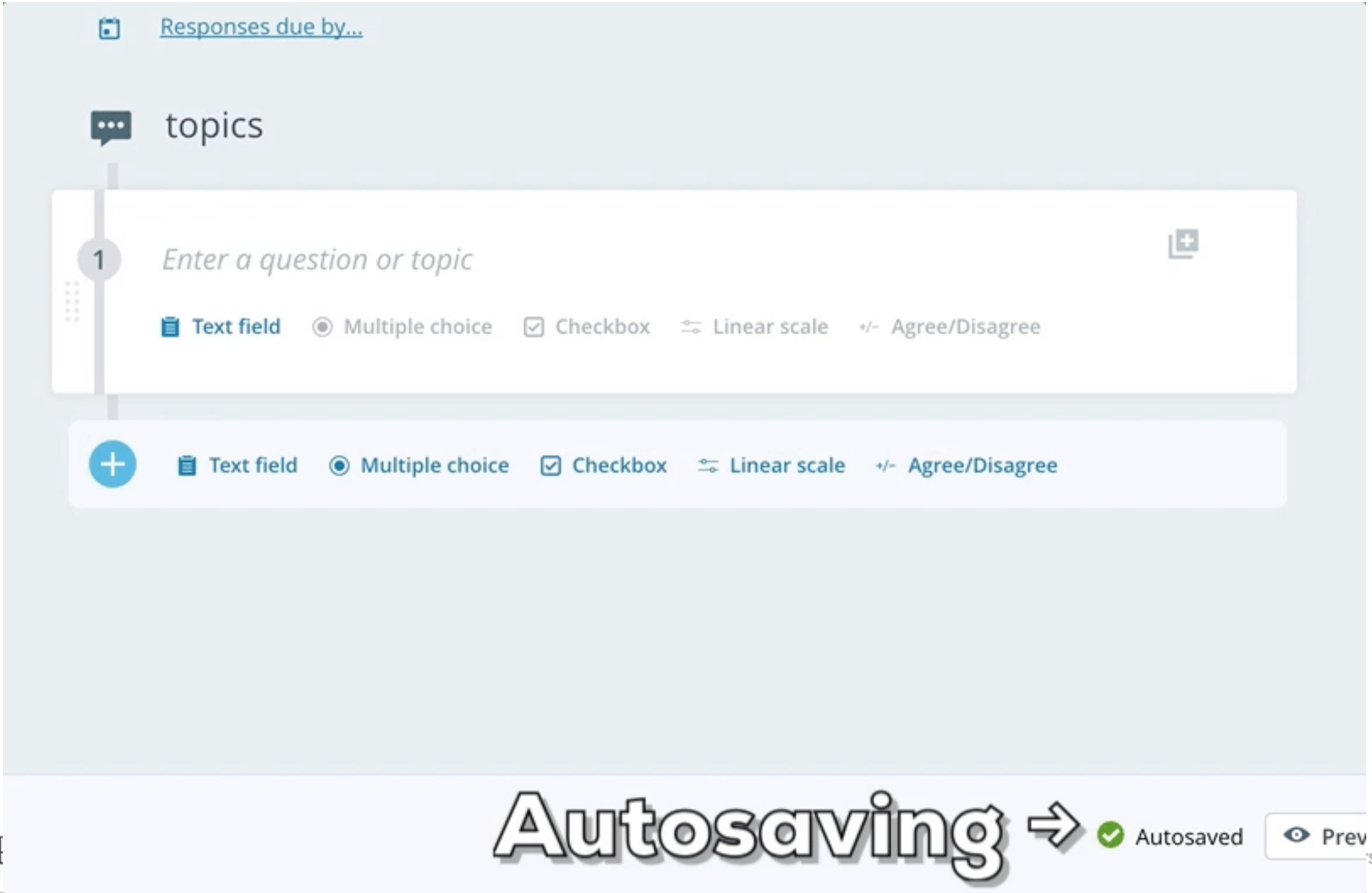
JavaScript callback/promise hell



```

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

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

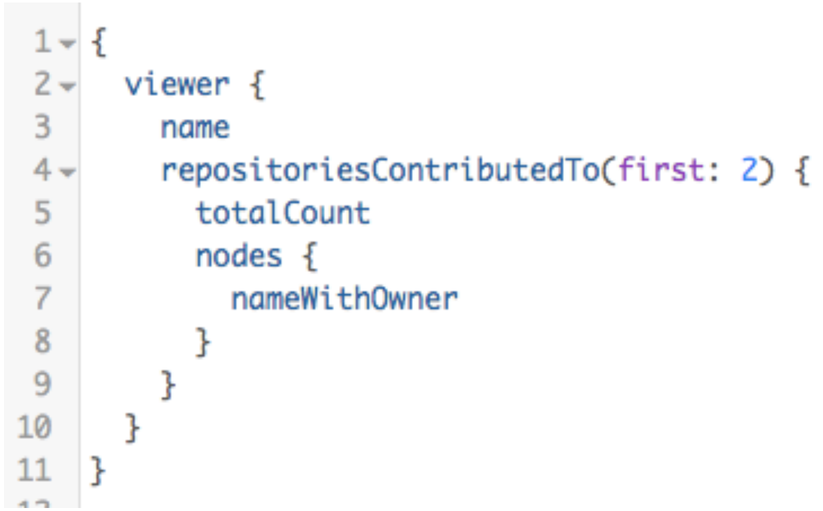
```
{
  "data": {
    "viewer": {
      "name": "Arianne",
      "login": "ariannedee",
      "location": "Vancouver, BC, Canada"
    }
  }
}
```

= 100% used

#2 - Under-fetching

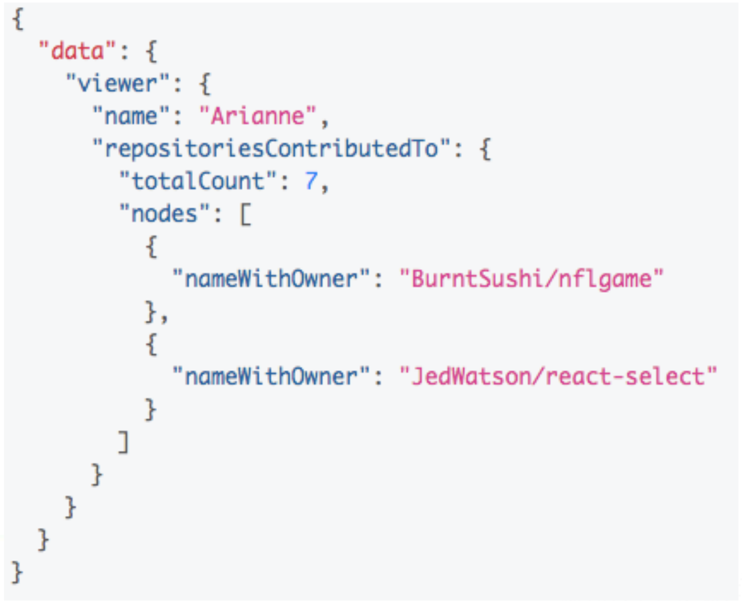
- Query for related resources
- Query for custom fields

GraphQL Request



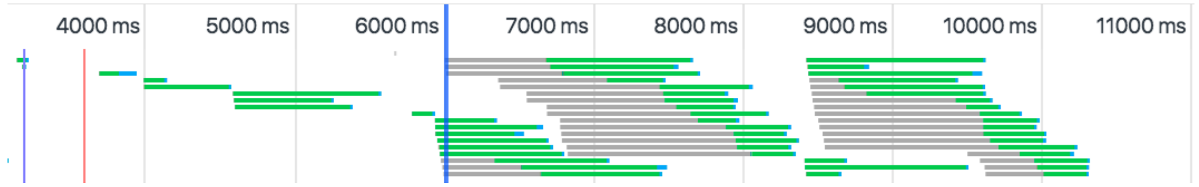
```
{
  viewer {
    name
    repositoriesContributedTo(first: 2) {
      totalCount
      nodes {
        nameWithOwner
      }
    }
  }
}
```

GraphQL Response



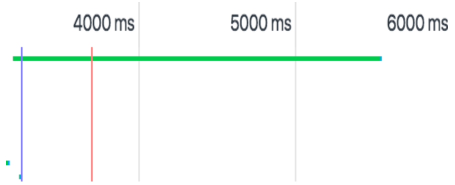
```
{
  "data": {
    "viewer": {
      "name": "Arianne",
      "repositoriesContributedTo": {
        "totalCount": 7,
        "nodes": [
          {
            "nameWithOwner": "BurntSushi/nflgame"
          },
          {
            "nameWithOwner": "JedWatson/react-select"
          }
       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

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

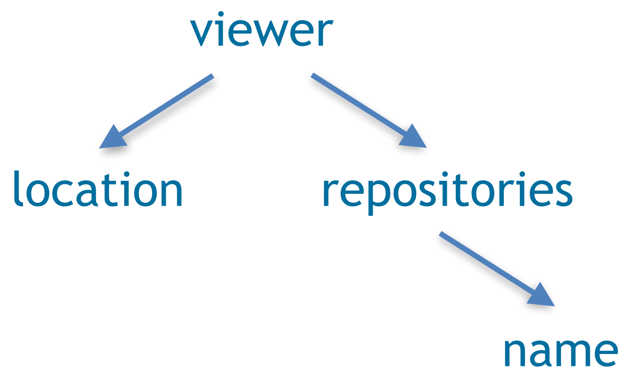
Some differences from REST

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

How does it work? Is it magic?

How does it work

- Tree of functions
- Functions are called "resolvers"
- Tree is called "schema"
- More detail when we create the server


$$\begin{tikzcd}
 & \text{viewer} \arrow[dl] \arrow[dr] & \\
\text{location} & & \text{repositories} \arrow[dr] & \\
 & & & \text{name}
\end{tikzcd}$$

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
    }
  }
}
```

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

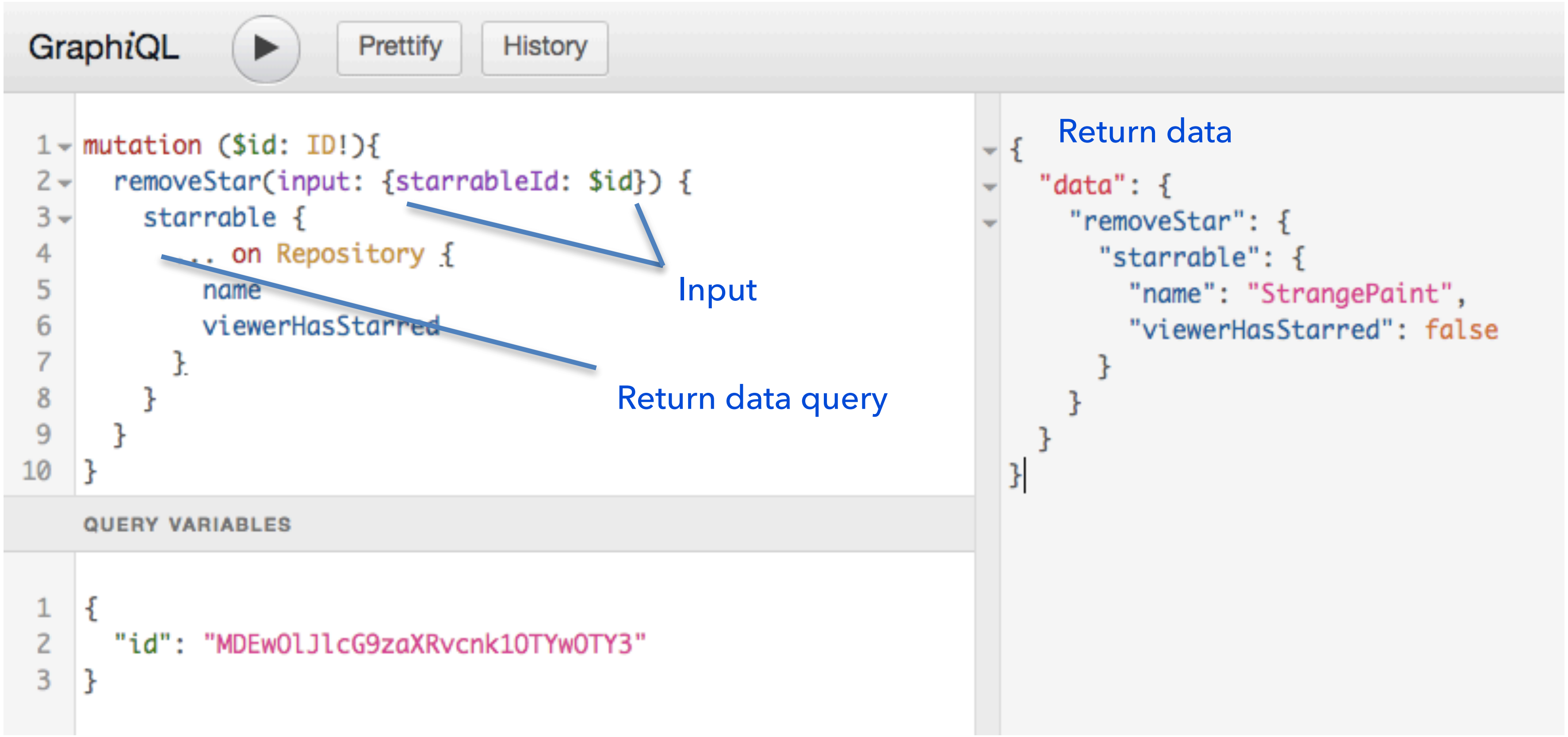
Request

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

Response

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

Pearson



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

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

```
{
  "data": {
    "removeStar": {
      "starrable": {
        "name": "StrangePaint",
        "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
        }
      }
    }
  }
}
```

Pearson

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

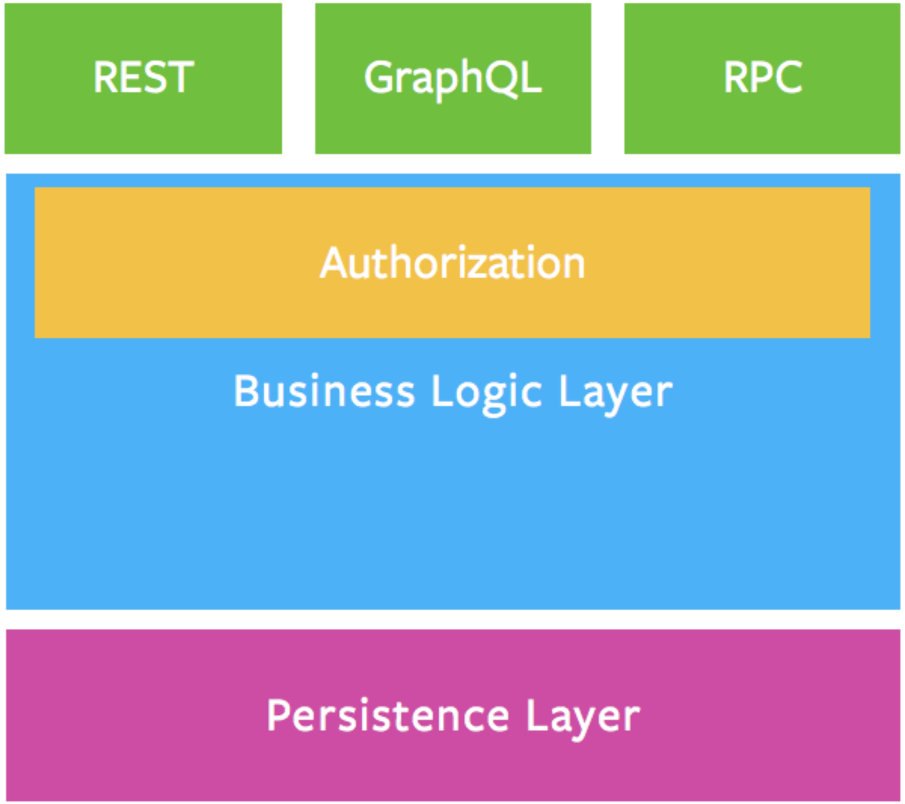
https://www.howtographql.com/advanced/4-security/

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

Question & Answer - 10 mins

Break - 5 mins

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
```

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. Create a card list of your repositories
3. Update the cards to be ordered by most recently created
4. Add some stats to each card
5. **Bonus:** Add functionality to star/un-star a repository

Client - Project tasks

1. **Update header to say "Hello {your name}"**
2. Create a card list of your repositories
3. Update the cards to be ordered by most recently created
4. Add some stats to each card
5. **Bonus:** 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. Create a card list of your repositories
3. Update the cards to be ordered by most recently created
4. Add some stats to each card
5. **Bonus:** Add functionality to star/un-star a repository

Client - Project tasks

1. Update header to say "Hello {your name}"
2. **Create a card list of your repositories**
3. Update the cards to be ordered by most recently created
4. Add some stats to each card
5. **Bonus:** Add functionality to star/un-star a repository

Client - Project tasks

1. Update header to say "Hello {your name}"
2. Create a card list of your repositories
3. **Update the cards to be ordered by most recently created**
4. Add some stats to each card
5. **Bonus:** Add functionality to star/un-star a repository

Client - Project tasks

1. Update header to say "Hello {your name}"
2. Create a card list of your repositories
3. Update the cards to be ordered by most recently created
4. **Add some stats to each card**
   - # open issues
   - # open PRs
   - # commits
5. **Bonus:** Add functionality to star/un-star a repository

Client - Project tasks

1. Update header to say "Hello {your name}"
2. Create a card list of your repositories
3. Update the cards to be ordered by most recently created
4. Add some stats to each card
5. **Bonus: Add functionality to star/un-star a repository**

Used features

- Queries
- Arguments
- Fragments
- Aliases

Bonus:

- Mutations
- Variables

Features not covered

- Unions
- Pagination
- Introspection
- Directives
- Subscriptions

Question & Answer - 10 mins

Break - 5 mins

Pearson

Let's build a server

Using Node.js or Django

Survey

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

What we'll cover

- Create **queryable** schema
  - nodes and edges
  - custom fields
- Accept **arguments**
  - filtering

Bonus solution content

- Support **pagination**
  - offset-based
- Support **mutations**
  - update data

Server

- Setup project
- Setup GraphQL
- Define queries (GET)
- Add filters

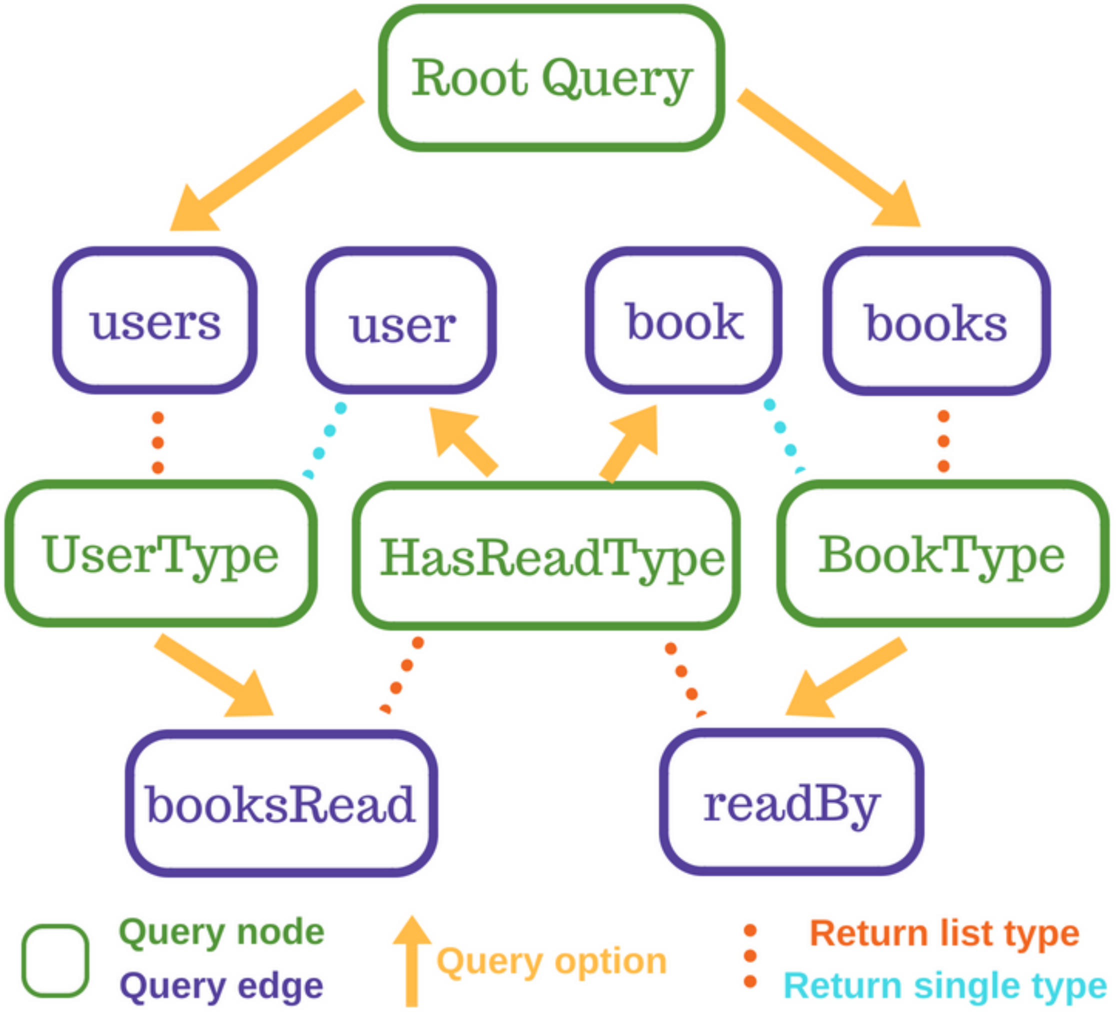
API model

Objects

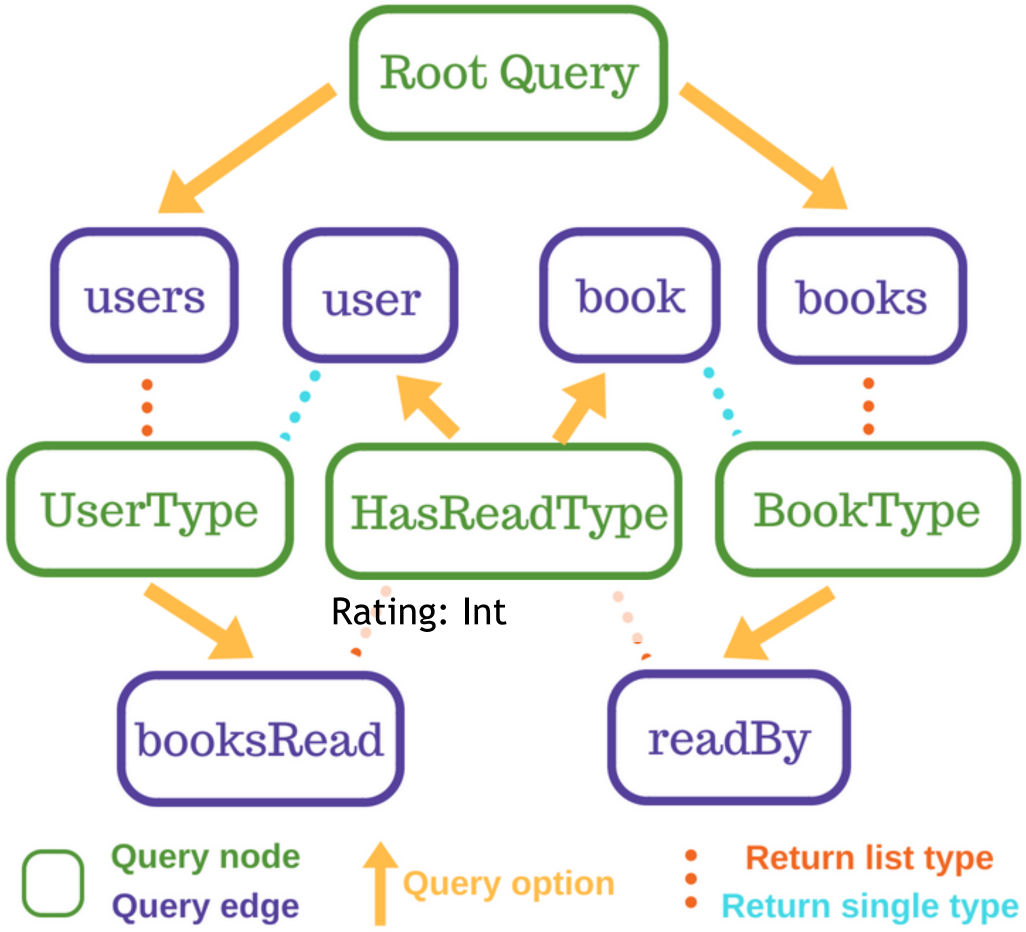
- Users
- Books

Relationships

- User - HasRead - Book


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

Pearson



id: ID
username: String
isAdmin: Boolean

Rating: Int

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

What does a GraphQL server do?

- Accepts requests that are sent to the GraphQL **endpoint**
- **Parses** the request (query and variables)
- Allows you to define a GraphQL **schema**
- **Validates** the query and returns errors
- **Executes** the query against the defined schema
- Returns resulting data and execution errors

What do you have to do?

- Define the endpoint that the server listens to
- Create the schema
  - Determines what queries and mutations are available to the user
- Set up authentication and other security measures

Server

- **Setup project**
- Setup GraphQL
- Define queries (GET)
- Add filters

Database ORMs

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

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

Tutorials

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

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

Server - Node setup

- You should have installed Node.js > 8.9

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

Server

- Setup project
- **Setup GraphQL**
- Define queries (GET)
- Add filters

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

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

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

const schema = new graphql.GraphQLSchema({query: queryType});
module.exports = schema;
```

Go to localhost:3000/graphql

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

Server

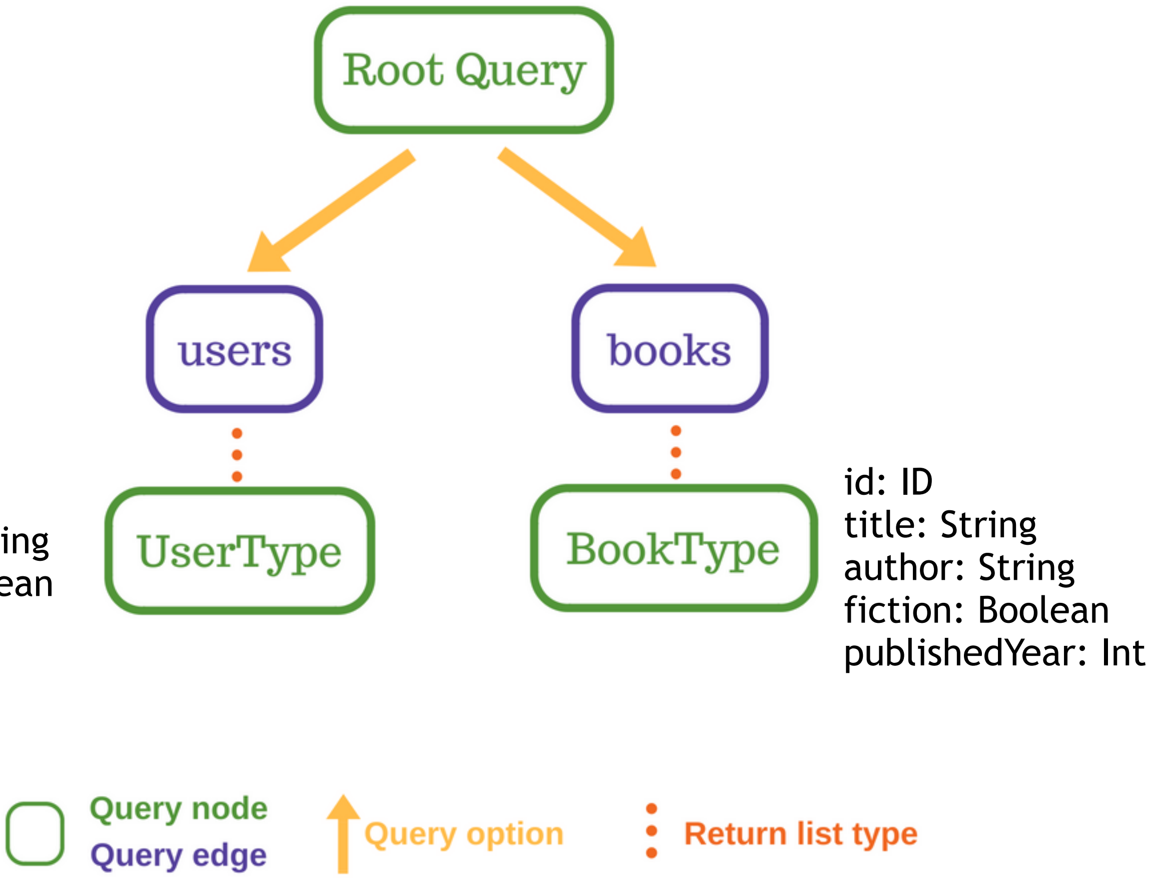
- Setup project
- Setup GraphQL
- **Define queries (GET)**
- Add filters

Define queries (GET)

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

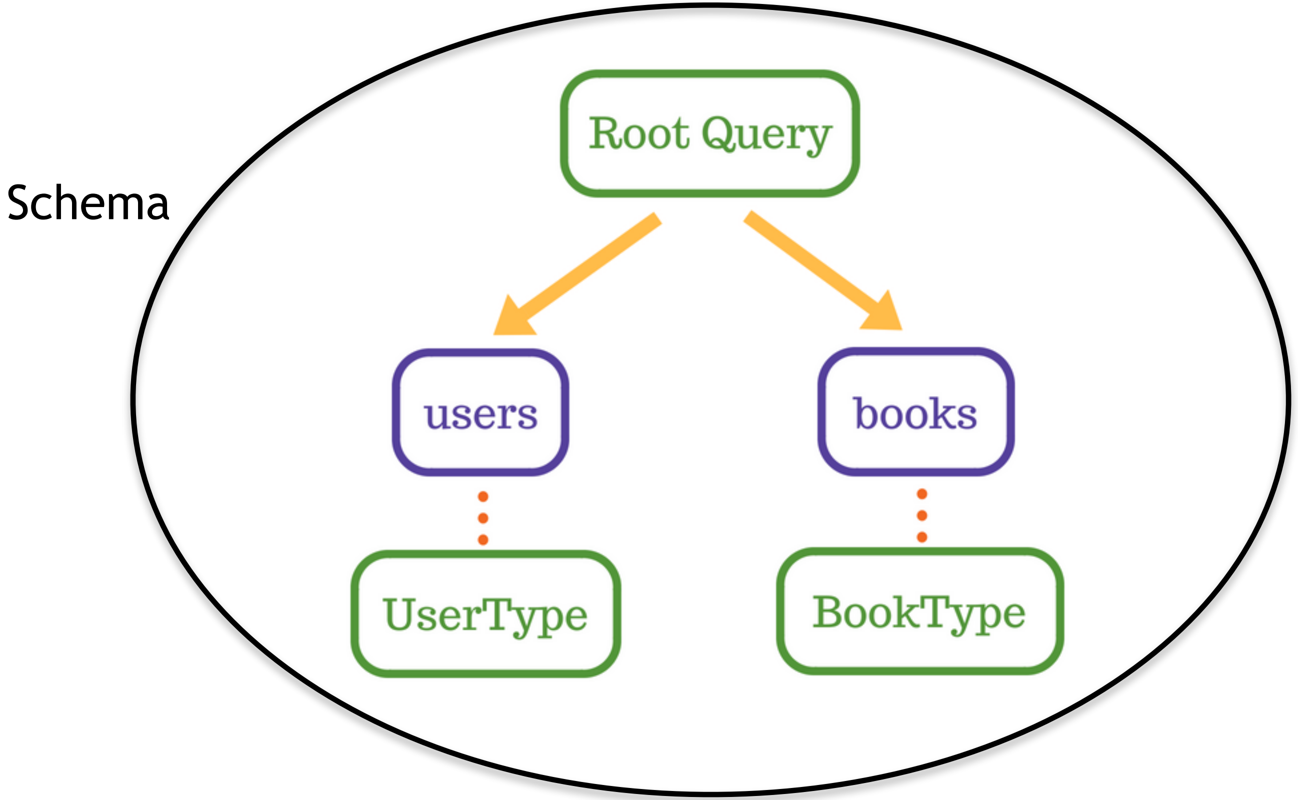
GraphQL Concepts

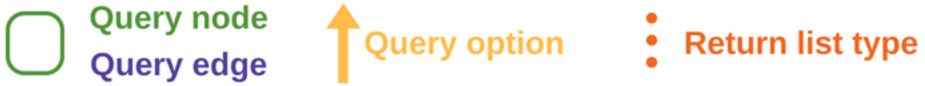
- Schema
- Types
  - Object
  - Scalar
    - Int, String, ID, etc…
  - List
- Fields
- Resolvers

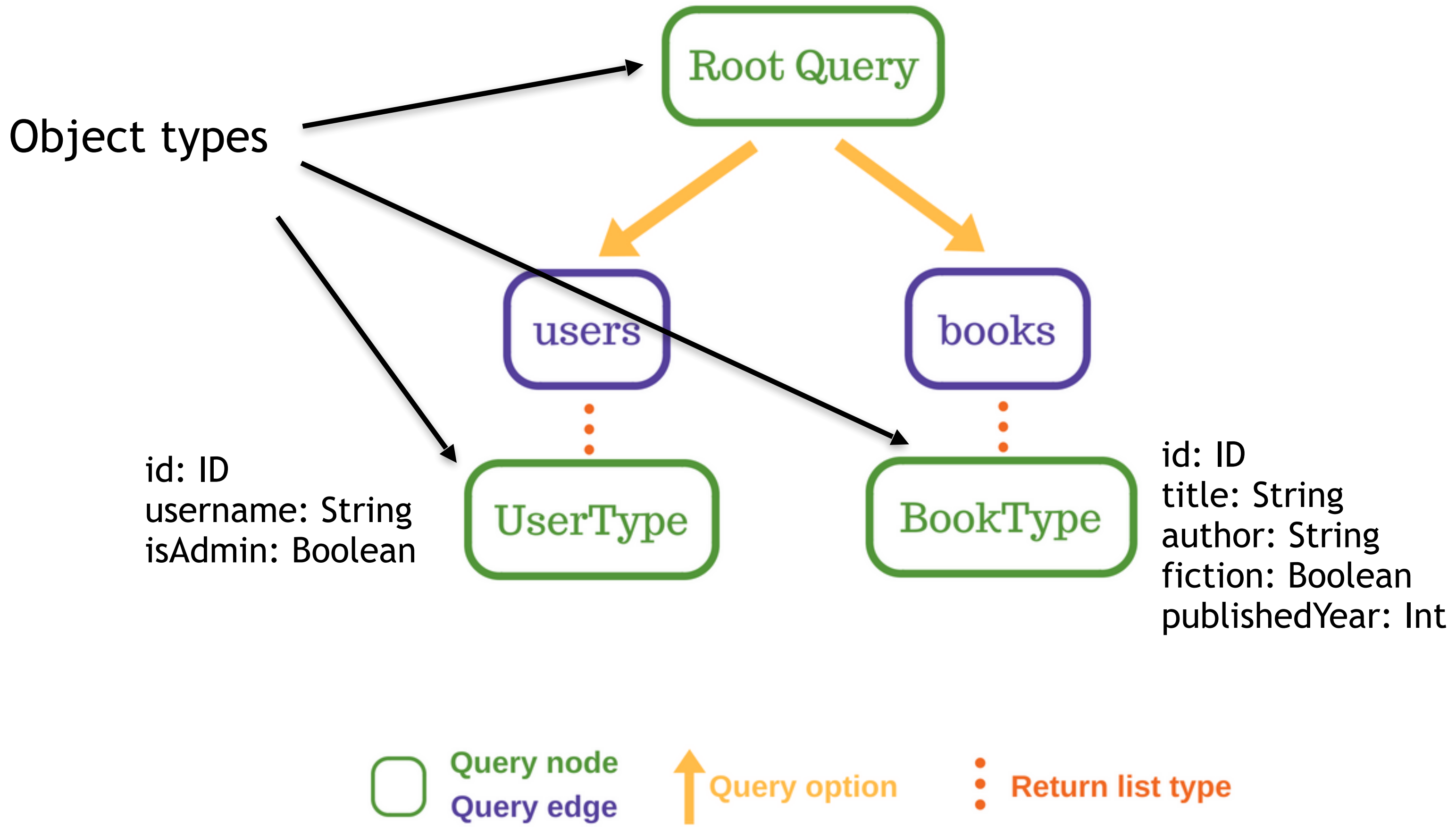


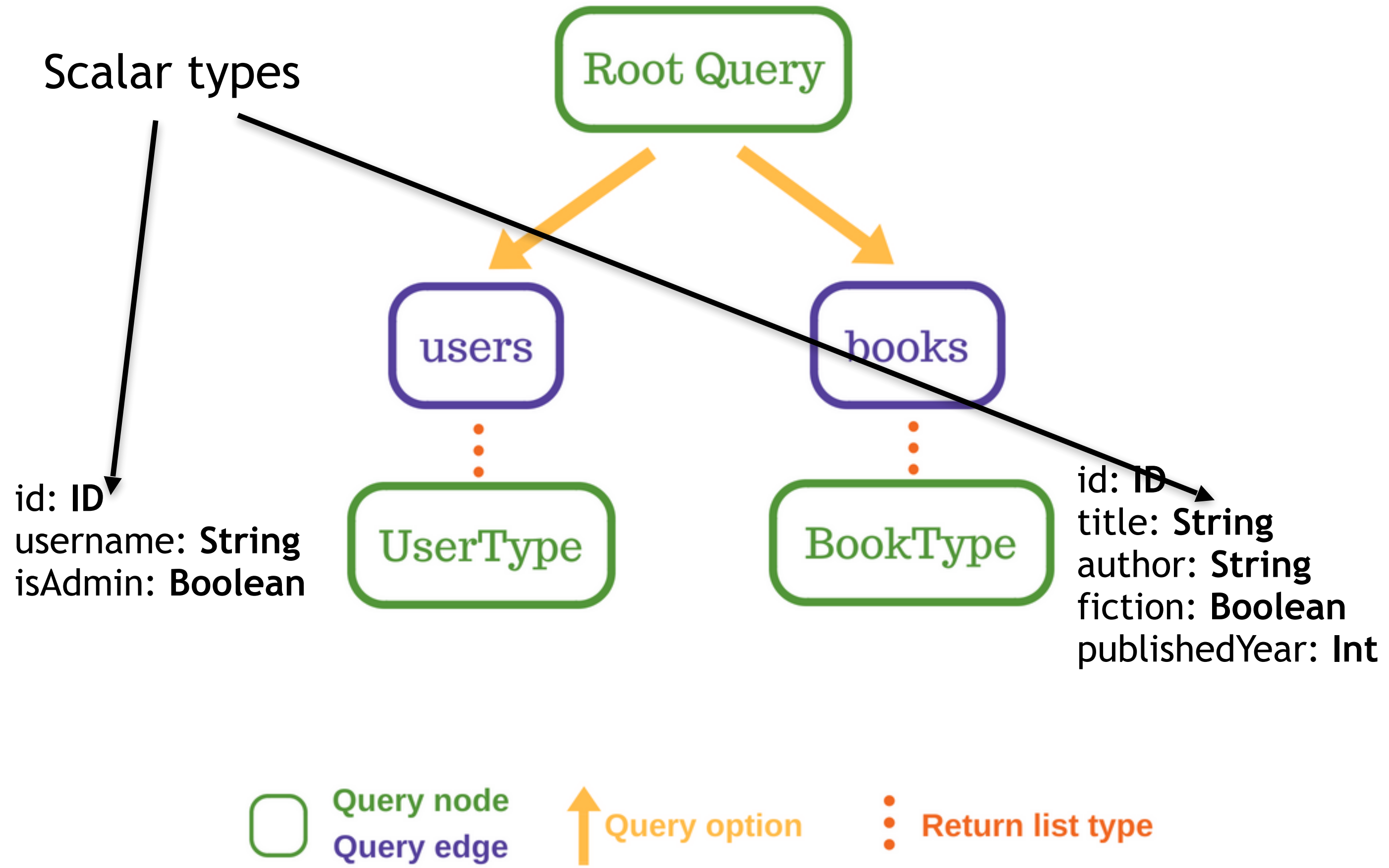
id: ID
username: String
isAdmin: Boolean

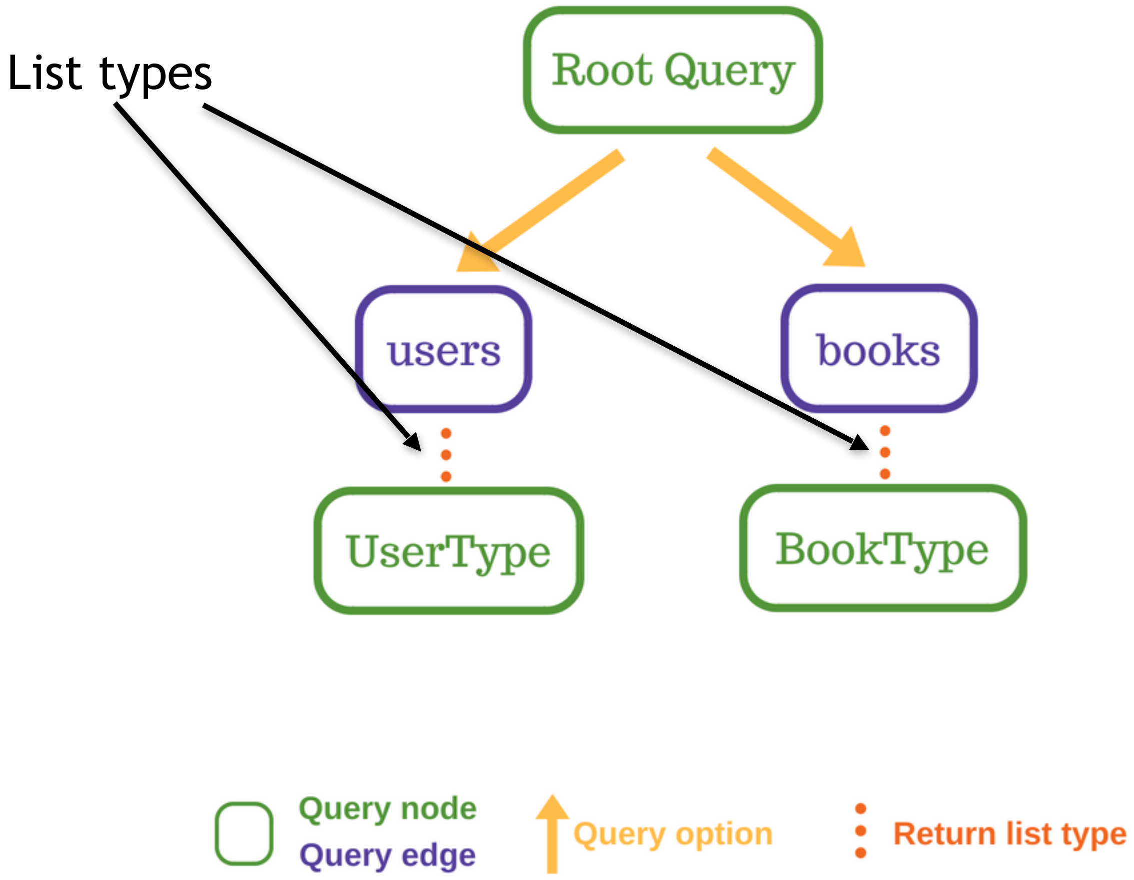
id: ID
title: String
author: String
fiction: Boolean
publishedYear: Int

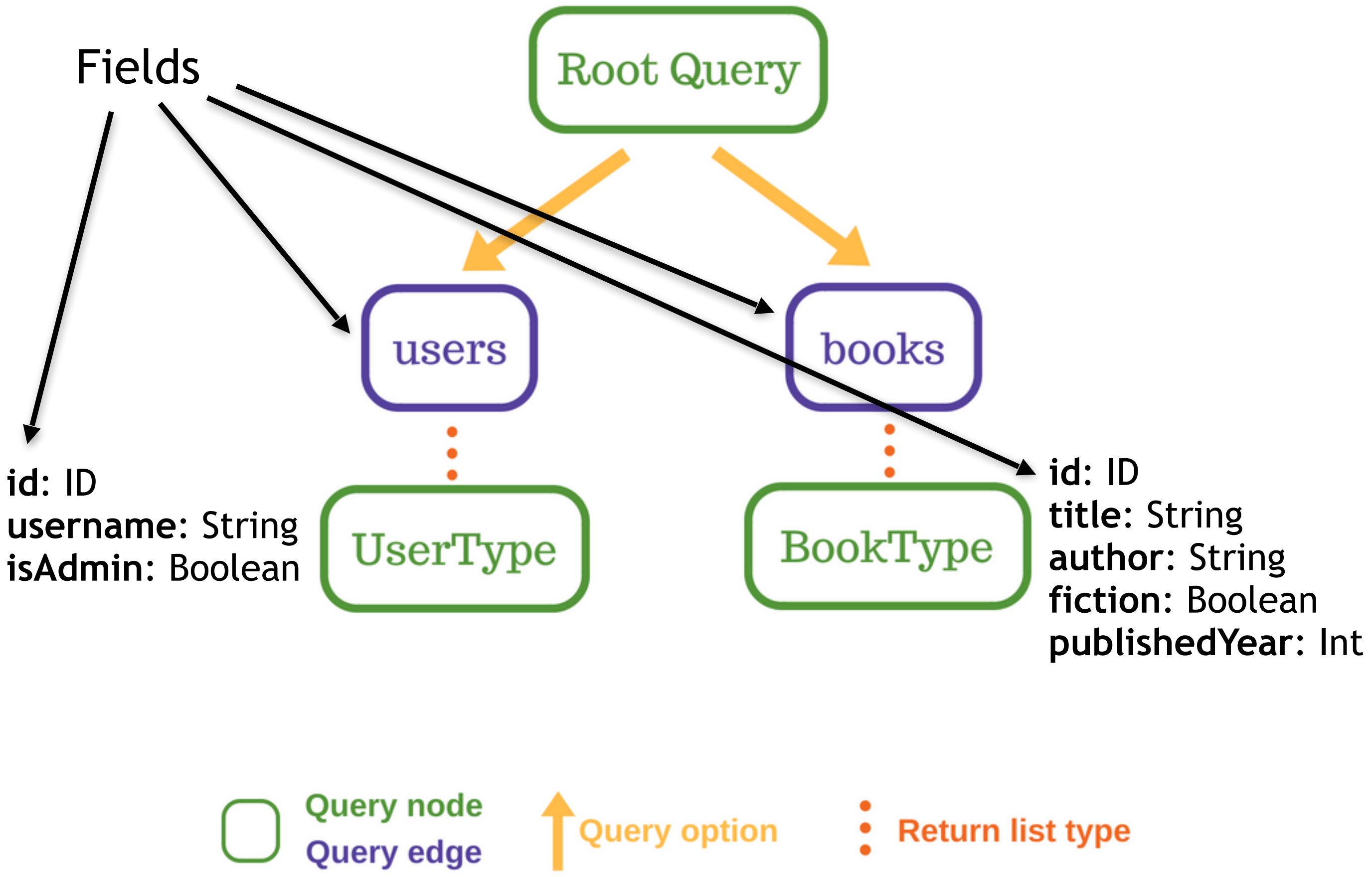

Schema
Root Query
users
books
UserType
BookType


Query node
Query edge
Query option
Return list type


Object 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


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

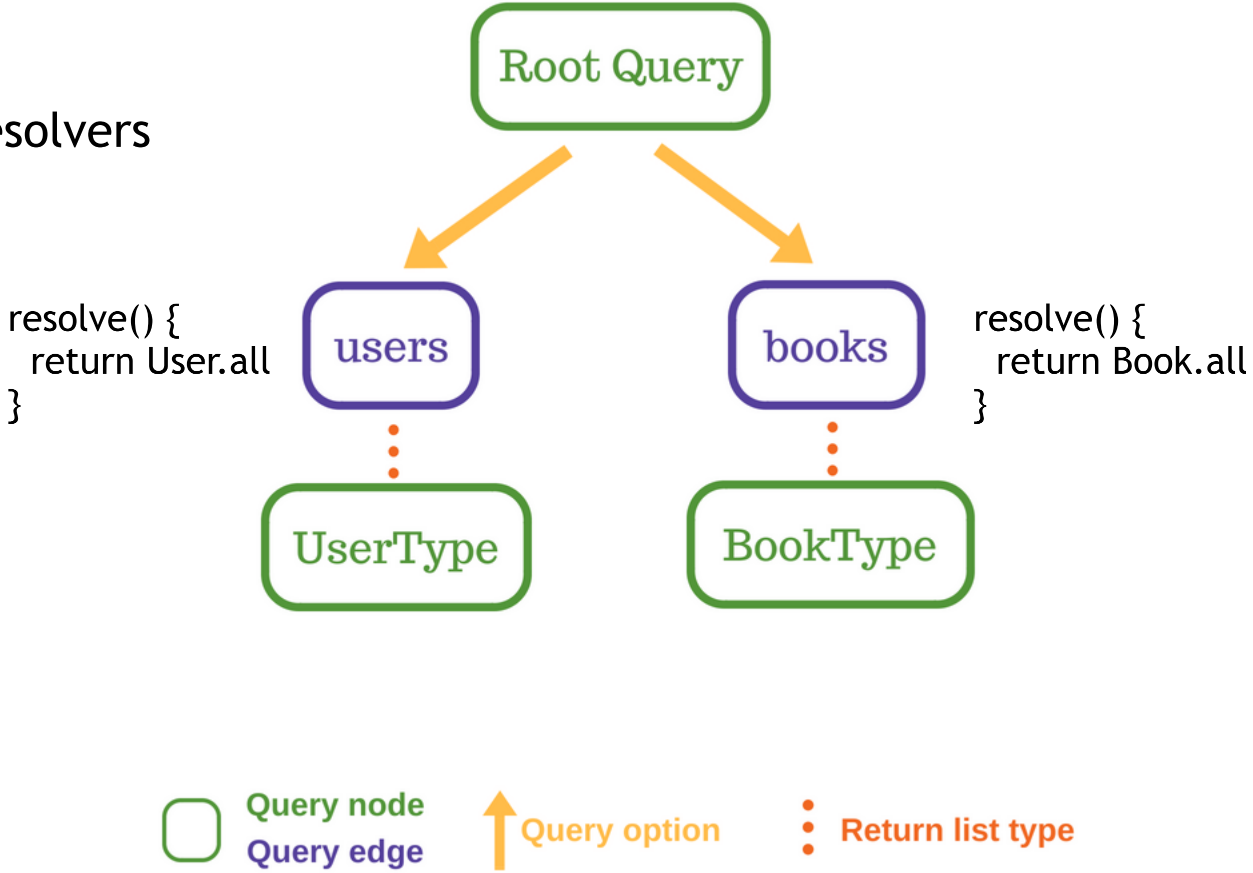

List types
Root Query
users
books
UserType
BookType
Query node
Query edge
Query option
Return list type



id: ID
username: String
isAdmin: Boolean

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

Resolvers

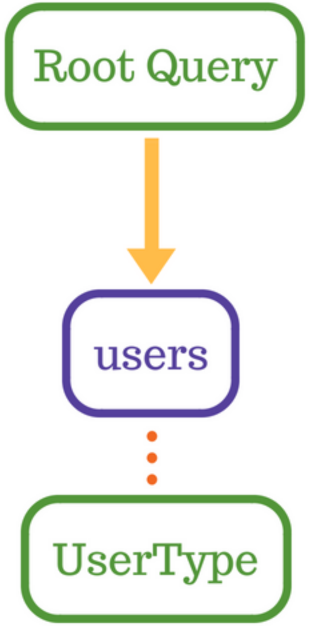


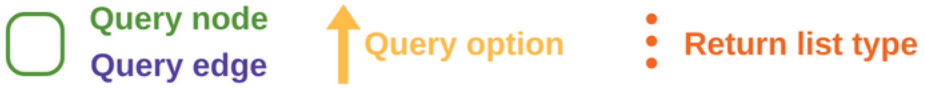
```
resolve() {
  return User.all
}
```

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

Define queries (GET)

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


Root Query
users
UserType


Query node
Query edge
Query option
Return list type

Pearson

Define queries - Node

```javascript
// 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() {
        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

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

...
users: {
  type: new graphql.GraphQLList(UserType),
  description: 'A list of users',
  resolve() {
    return knex('user');
  }
}
```

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

Define queries - Django

```
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()
```

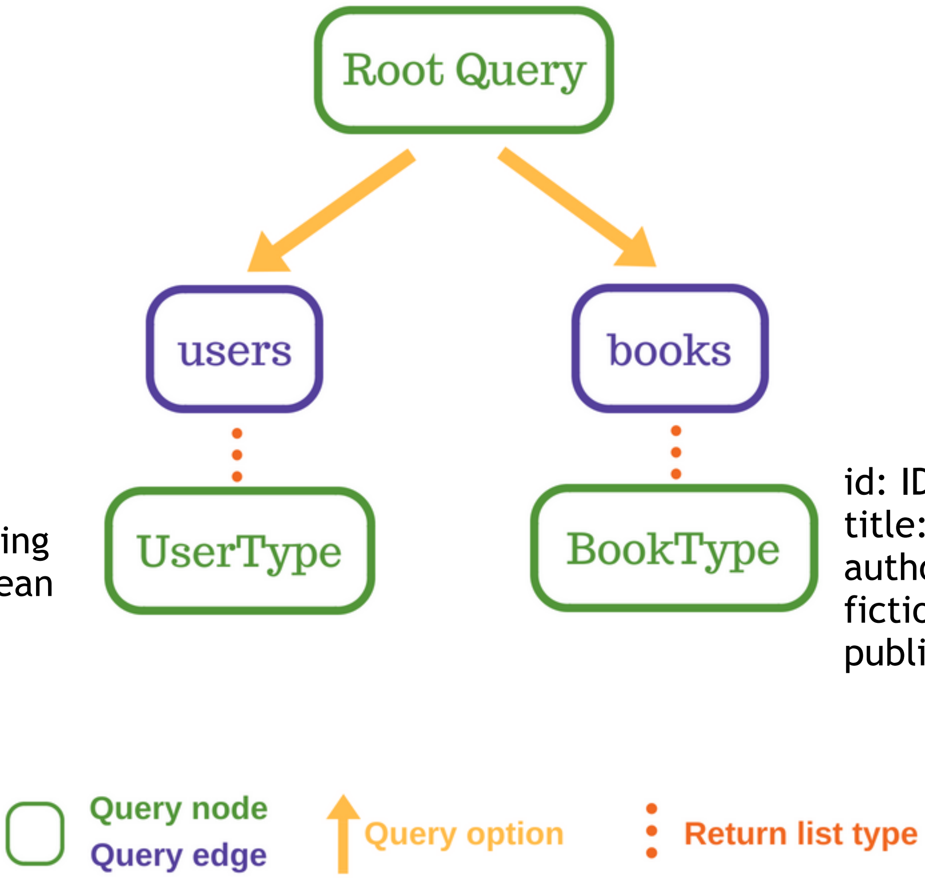
Test to see if it works

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')
```



id: ID
username: String
isAdmin: Boolean

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

Pearson

Add books - define Book type

```
const BookType = new graphql.GraphQLObjectType({
  name: 'Book',
  fields:{
    id: {
      type: graphql.GraphQLID,
      resolve(book) {
        return book.id;
      }
    },
  }
});
```

Add books - add to Query

```
const queryType = new graphql.GraphQLObjectType({
  name: 'Query',
  fields: {
    books: {
      type: graphql.GraphQLList(BookType),
      resolve(root, args, context) {
        return knex('book');
      }
    }
  }
});
```

Add books

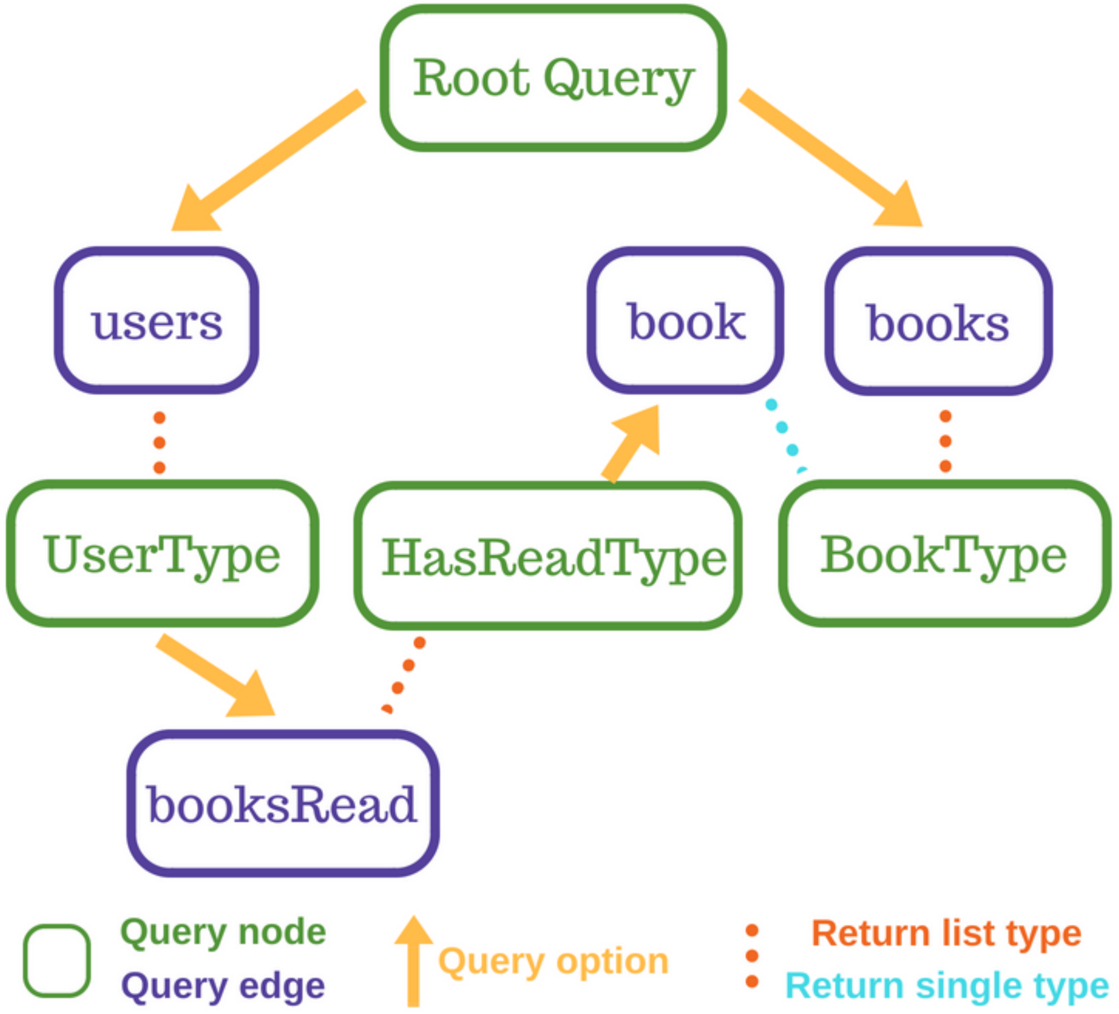
```
class BookType(graphene_django.DjangoObjectType):
    class Meta:
        model = Book


class Query(graphene.ObjectType):
    books = graphene.List(BookType)

    def resolve_books(self, info):
        return Book.objects.all()
```

Define queries (GET)

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


Root Query
users
book
books
UserType
HasReadType
BookType
booksRead
Query node
Query edge
Query option
Return list type
Return single type

Add HasRead type

```
const HasReadType = new graphql.GraphQLObjectType({
  name: 'HasRead',
  fields: {
    rating: {
      type: graphql.GraphQLInt,
      resolve(hasRead) {
        return hasRead.rating;
      }
    },
    book: {
      type: BookType,
      resolve(hasRead) {
        return knex('book').where('id', hasRead.bookId).first();
      }
    },
  }
});
```

Add booksRead field to User type

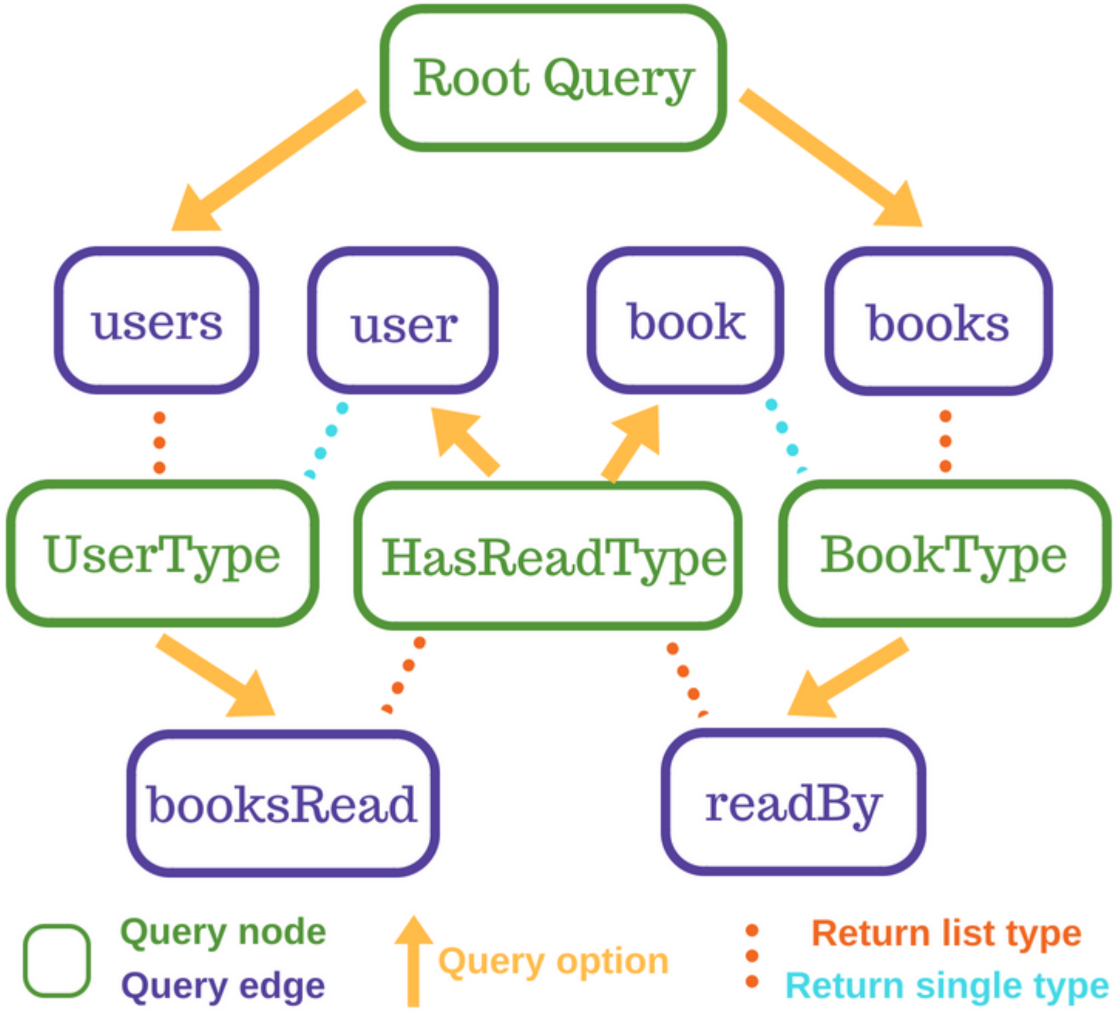
```
const UserType = new graphql.GraphQLObjectType({
  name: 'User',
  fields: () => {
    return {
      ...
      booksRead: {
        type: graphql.GraphQLList(HasReadType),
        resolve(user) {
          return knex('hasRead').where('userId', user.id);
        }
      }
    }
  }
});
```

Add HasReadType

```
class HasReadType(graphene_django.DjangoObjectType):
    class Meta:
        model = HasRead
```

Define queries (GET)

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


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

Add readBy field to Book type

```
const BookType = new graphql.GraphQLObjectType({
  name: 'Book',
  fields: () => {
    return {
      ...,
      readBy: {
        type: graphql.GraphQLList(HasReadType),
        resolve(book) {
          return knex('hasRead').where('bookId', book.id);
        }
      }
    }
  }
});
```

Define queries (GET)

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

Server

- Setup project
- Setup GraphQL
- Define queries (GET)
- **Add filters**
  - fiction/non-fiction books

Other features not covered

- Enums
- Unions
- Interfaces

Thanks!

Questions?

Email me at arianne.dee.studios@gmail.com

Pearson

More courses by me, Arianne

Live Trainings

- **Introduction to Python Programming**
  - Very beginner
- **Programming with Python: Beyond the Basics**
  - Beginner
- **Object-Oriented Programming in Python**
  - Intermediate

Videos

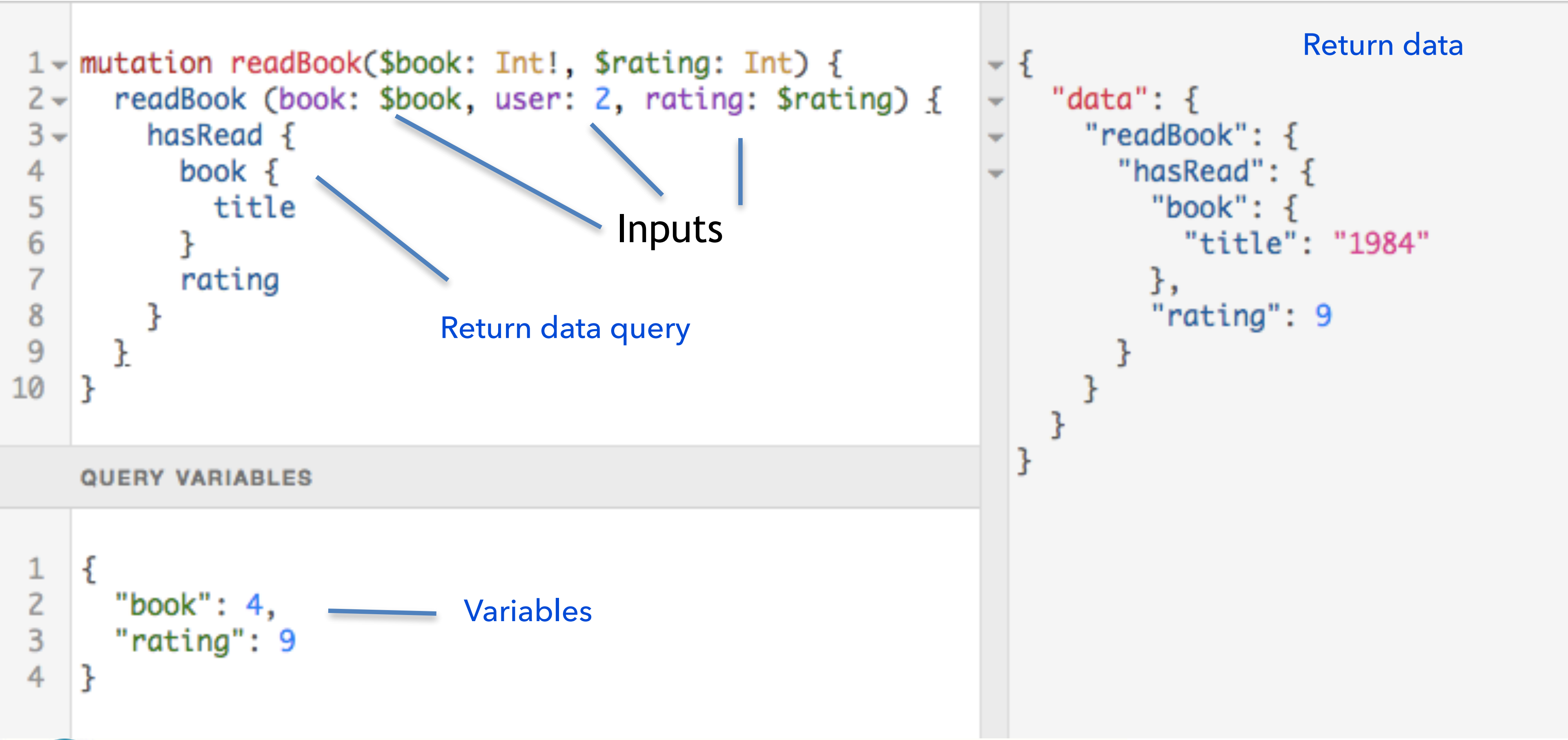
- **Introduction to Python LiveLessons** - link
- **Rethinking REST: A hands-on guide to GraphQL** - link

Bonus material

Mutations and pagination

Define mutations

- Read/rate a book
- Update a book's rating



```
mutation readBook($book: Int!, $rating: Int) {
  readBook (book: $book, user: 2, rating: $rating) {
    hasRead {
      book {
        title
      }
      rating
    }
  }
}
```

QUERY VARIABLES

```
{
  "book": 4,
  "rating": 9
}
```

```
{
  "data": {
    "readBook": {
      "hasRead": {
        "book": {
          "title": "1984"
        },
        "rating": 9
      }
    }
  }
}
```

Define Mutation type

```
const mutationType = new graphql.GraphQLObjectType({
    name: 'Mutation',
    fields: () => {
        return {
            readBook: {
                type: HasReadType,
                description: 'Read and rate (optional) a book',
                args: {
                # What data does the user have to give?
                },
                async resolve(source, args, context) {
                # What does calling the mutation do?
                }
            }
        }
    }
});
```

Add Mutation to schema

```
# Update schema to include mutation
const schema = new graphql.GraphQLSchema({query: queryType,
mutation: mutationType});
```

Fill in mutation details

```
readBook: {
    args: {
        user: {
            type: graphql.GraphQLNonNull(graphql.GraphQLInt)
        },
        book: {
            type: graphql.GraphQLNonNull(graphql.GraphQLInt)
        },
        rating: {
            type: graphql.GraphQLInt,
            defaultValue: null
        }
    },
    async resolve(source, args, context) {
        userId = args.user;
        bookId = args.book;
        rating = args.rating;
        return await readBook(bookId, userId, rating);
   }
}
```

Create mutation

```
class ReadBook(graphene.Mutation):
    class Arguments:
        book = graphene.Int(required=True)
        user = graphene.Int(required=True)
        rating = graphene.Int()

    hasRead = graphene.Field(HasReadType)

    def mutate(self, info, **kwargs):
        book_id = kwargs['book']
        user_id = kwargs['user']
        rating = kwargs.get('rating')

        hasRead = read_book(book_id, user_id, rating)
        return ReadBook(hasRead=hasRead)
```

Add Mutation type to schema

```
class Mutations(graphene.ObjectType):
    read_book = ReadBook.Field()

schema = graphene.Schema(query=Query, mutation=Mutations)
```

Pagination

- Solutions include DIY offset based pagination
  - Allow first, last, and offset arguments
  - In resolver, update your query based on provided arguments

- Most libraries for pagination are for cursor based pagination (Relay framework)

Pagination - resources

- 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/

Authentication

- 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

Resources

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

Overview

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

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