DJANGO REST FRAMEWORK

What you will learn:

1. What is REST?
2. Getting started with the DRF
3. Serializers
4. Views
5. ViewSets
6. Routers
7. Permissions
8. Actions

PREREQUISITES

Note:

- This course does not cover Django
- Django models, views, urls, admin files
- `curl` `https://curl.haxx.se/`

VERSIONS

Note:

- Code samples were tested using:
  - Python 3.9.0
  - Django 3.1.2
  - Django REST framework 3.12.1
- Code formatted with `Black`

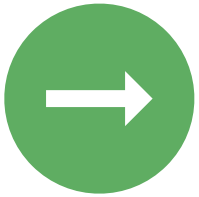
OVERVIEW

- REST is a protocol used to send data back and forth between servers
- Not a standard
- Uses HTTP methods and URLs to express:
  - Listing
  - Details
  - Creating
  - Updating
  - Patching
  - Deleting
- Mostly used to send serialized JSON or XML

OVERVIEW

- Django REST framework (DRF)
- Works in conjunction with Django to send and receive data
- Django models work with DRF `Serializers`
- Django views work with DRF views, `ViewSets`, and `Routers`
- Supports multiple flavors of data rendering, including JSON and a web interface
- Built-in permission structures

NEXT UP...



What is REST?

TABLE OF CONTENTS

WHAT IS REST?

- **REST**: **RE**presentational **S**tate **T**ransfer
- Defined by Roy Fielding in a PhD dissertation “Architectural Styles and the Design of Network-based Software Architectures”, 2000
- Based on the existing HTTP standard at the time

SIX ARCHITECTURAL CONSTRAINTS

1. Client-server Architecture
    - Separation of UI from data storage / processing
2. Statelessness
    - No client context is stored on the server between calls
3. Cacheability
    - Clients can cache, responses must indicate if they are cacheable
4. Layered system
    - Client does not care if there are layers between it and the server, e.g. proxies, load balancers, etc.
5. Code on demand (optional)
    - Servers can optionally transmit executable code

SIX ARCHITECTURAL CONSTRAINTS

6. Uniform Interface; Interface has four properties:
    i. **Resources are identified as requests:** URL maps to data
    ii. **Resource manipulation through representation:** data itself has enough info to modify or delete the data
    iii. **Self descriptive messages:** message contains info on how to process message, e.g. type tags indicating what parser to use
    iv. **Hypermedia As The Engine Of Application State (HATEOAS):** data contains links needed to do other things with the data, i.e. no need to hard code anything on the client side

YOUR MILEAGE MAY VARY

> "The code is more what you'd call 'guidelines' than actual rules."
>
> *-- Geoffrey Rush as Hector Barbossa*
> *Pirates of the Carribean (2003)*

- Not quite a standard
- Often only a subset is implemented
- APIs built using REST principles are documented
- Clients tend to do the minimum necessary
- Lighter weight than similar technologies such as SOAP

A SUGGESTION BUILT ON STANDARDS

HTTP Method	Purpose
`GET`	Retrieve a resource or listing of resources
`POST`	Create a resource. Request body contains fields describing properties of the new resource.
`PUT`	Replace contents of a resource with new information. Request body contains fields describing new property values.
`PATCH`	Update a resource changing only those properties indicated in the request body.
`DELETE`	Delete a resource

EXAMPLES

HTTP Method	URL	Description
GET	`http://example.com/books/`	Retrieve a list of books
GET	`http://example.com/books/12/`	Retrieve book with id=12
POST	`http://example.com/books/` *`title=Metamorphosis`* *`author=Kafka`*	Create a new book with the title *Metamorphosis* and the author *Kafka*. Typically returns the newly created item.

EXAMPLES

HTTP Method	URL	Description

PUT	`http://example.com/books/23/`
	`title=Metamorphosis`
	`author=Franz Kafka`
		Replace the fields in the book with id=23, new values are a title of *Metamorphosis* and author of *Franz Kafka*

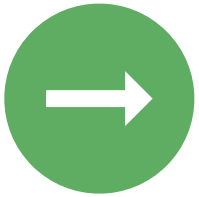
PATCH	`http://example.com/books/23/` `title=The Metamorphosis`	Replace the title field with *The Metamorphosis* in the book with id=23
DELETE	`http://example.com/books/12/`	Delete book with id=12

PAYLOADS

- REST does not specify the format of the payload
- JSON and XML are the most common

```
$ curl -s http://127.0.0.1:8000/books/ | python -m json.tool
{
    "books": [
        {
            “id”: 23,
            "title": “The Metamorphosis",
            "author": “Franz Kafka”,
        },
        {
          "id": 28,
          "title": “The Stranger”,
          "author": “Albert Camus"
        },
    ]
}
```

NEXT UP...



DRF

TABLE OF CONTENTS

DJANGO REST FRAMEWORK (DRF)

- Toolkit for developing Web APIs built on top of Django platform
- Integrates with Django models, views and URL patterns
- Provides mechanisms for both function and class based views
- Serialization for both ORM and non-ORM based data
- Built-in web interface
- More information:

  `https://www.django-rest-framework.org/`

GETTING STARTED

- You will need both Django and Django REST framework installed
- Preferably in a virtual environment

```
$ python -m pip install django djangorestframework
Collecting django
  Using cached Django-3.1.2-py3-none-any.whl (7.8 MB)
Collecting djangorestframework
  Using cached djangorestframework-3.12.1-py3-none-any.whl (913 kB)
Collecting sqlparse>=0.2.2
  Using cached sqlparse-0.4.1-py3-none-any.whl (42 kB)
Collecting asgiref~=3.2.10
  Using cached asgiref-3.2.10-py3-none-any.whl (19 kB)
Collecting pytz
  Using cached pytz-2020.1-py2.py3-none-any.whl (510 kB)
Installing collected packages: sqlparse, asgiref, pytz, django, djangorestframework
Successfully installed asgiref-3.2.10 django-3.1.2 djangorestframework-3.12.1
pytz-2020.1 sqlparse-0.4.1
```

GETTING STARTED

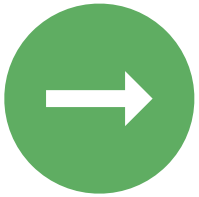
- `curl` for browsing from the command line
- Comes built-in with most operating systems
- Can be downloaded here:

  `https://curl.haxx.se/`
- Sample code available in the Supporting Material drop-down

SUMMARY

- Create the project and app
- Create the `Person` model
- Create the `PersonSerializer`
- Add a view that uses a `Response` object to send a serialization of the `Person.objects.all()` query set
- Register the URLs

NEXT UP...



ViewSets

TABLE OF CONTENTS

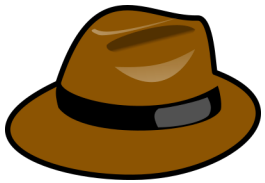
DRF COMPONENTS

- `Serializers`:
  - Change objects into text and text back into objects
  - With or without the Django ORM
- Views:
  - Utilities to write Django views that serialize and deserialize objects
- `ViewSets`:
  - Class based view utilities encapsulating common REST/HTTP methods
- `Routers`:
  - Map between `ViewSets` and Django url routes

VIEWSET EXAMPLE FROM DOCS

```
class UserViewSet(viewsets.ViewSet):
    """Example empty viewset demoing standard actions handled by a router class."""
    def list(self, request):
        pass

    def create(self, request):
        pass

    def retrieve(self, request, pk=None):
        pass

    def update(self, request, pk=None):
        pass

    def partial_update(self, request, pk=None):
        pass

    def destroy(self, request, pk=None):
        pass
```

BUILDING ON FEDORA



- Add another app

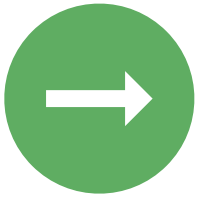
```
$ python manage.py startapp artifacts
```

- Edit `Fedora/settings.py`, adding `artifacts` to `INSTALLED_APPS`
- Edit `Fedora/urls.py`, including `artifacts.urls` in the `url_patterns` list
- Follow along with `models.py`, `serializers.py`, `views.py`, and `urls.py`, as shown
- Create an `admin.py` file for the models if desired
- Run `makemigrations` and `migrate` as needed
- Use the `admin` or the `loaddata` command to add some data

DRF VIEWSETS

- By specifying a `ViewSet` you automatically get the REST methods:
  - List, Retrieve, Create, Update, Update Partial, and Delete
- Routers define all the URL mappings needed for your `ViewSet`

NEXT UP...



Web Interface

TABLE OF CONTENTS

WEB INTERFACE

- Out of the box, DRF comes with a web interface
- Type of response is based on:
  - Renderers installed
  - **HTTP Accept** header
- For example, if the installed renderers were YAML and HTML:

```
Accept: application/json; indent=4, application/json, application/yaml, text/html, */*
```

  - Results in YAML being used

SETTING THE RENDERER

- Renderer can be set globally in `settings.py`:

```
REST_FRAMEWORK = {
    'DEFAULT_RENDERER_CLASSES': [
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',
    ]
}
```

- Above is the default
- For performance reasons you may not want to include `BrowsableAPIRenderer`

SETTING THE RENDERER

- Or with a view decorator:

```
@api_view(['GET'])
@renderer_classes([JSONRenderer])
def user_count_view(request, format=None):
    user_count = User.objects.filter(active=True).count()
    content = {'user_count': user_count}
    return Response(content)
```

SETTING THE RENDERER

- Or within an APIView, which all ViewSets inherit from:

```
from django.contrib.auth.models import User
from rest_framework.renderers import JSONRenderer
from rest_framework.response import Response
from rest_framework.views import APIView

class UserCountView(APIView):
    """
    A view that returns the count of active users in JSON.
    """
    renderer_classes = [JSONRenderer]

    def get(self, request, format=None):
        user_count = User.objects.filter(active=True).count()
        content = {'user_count': user_count}
        return Response(content)
```

TYPES OF RENDERERS

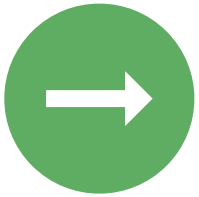
- Built-in renderers:
  - `JSONRenderer`
  - `TemplateHTMLRenderer`
  - `StaticHTMLRenderer`
  - `BrowsableAPIRenderer`
  - `AdminRenderer`
  - `HTMLFormRenderer`
  - `MultiPartRenderer`

TYPES OF RENDERERS

- Third-party renderers:
  - `YAML`
  - `XML`
  - `JSONP, UltraJSON, CamelCase JSON`
  - `MessagePack`
  - `XLSX`
  - `CSV`
  - `Pandas CSV, Excel, PNG`
  - `LaTeX`
- Write your own

  `https://www.django-rest-framework.org/api-guide/renderers/`

NEXT UP...



Permissions

TABLE OF CONTENTS

PERMISSIONS

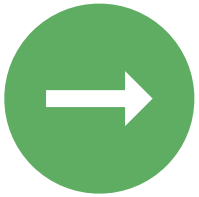
- Django REST framework has a robust permission mechanism
- ViewSets support the use of lists of Permission classes
- Permissions are at both the request and object level

ACCOUNT BASED PERMISSIONS

- `Fedora` demos account based permissions
- You will need some accounts:
  - Username: `indy`, `is_staff=True`
  - Username: `marion`
- `Fedora` needs a login page

  `https://realpython.com/django-user-management/`

NEXT UP...



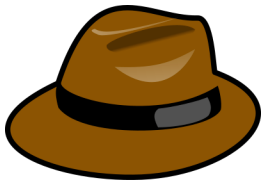
DRF Permissions

TABLE OF CONTENTS

DRF PERMISSIONS PREP

- Installed Django contrib auth
- Added a login page
- Created two accounts:
  - Username: `indy`, `is_staff=True`
  - Username: `marion`

BUILDING ON FEDORA



- Add another app

```
$ python manage.py startapp books
```

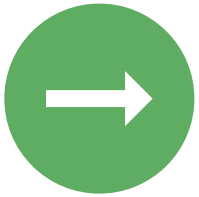
- Edit `Fedora/settings.py`, adding `books` to `INSTALLED_APPS`
- Edit `Fedora/urls.py`, including `books.urls` in the `url_patterns` list
- Follow along with `models.py`, `serializers.py`, `views.py`, and `urls.py` as shown
- Create an `admin.py` file for the models if desired
- Run `makemigrations` and `migrate` as needed
- Used the `admin` or the `loaddata` command to add some data

WARNING!

Default permissions allow entry rather than deny it!

Permissions don’t apply filters!

NEXT UP...



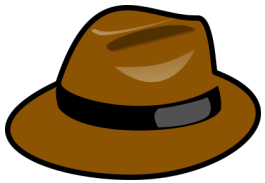
More serialization

TABLE OF CONTENTS

SERIALIZE OBJECTS

- Serialization is not restricted to Django database ORM objects
- Use the base `serializers.Serializer` class combined with serializer fields to construct arbitrary objects
- Similar to how you declare a model ORM object

BUILDING ON FEDORA



- Add another app

```
$ python manage.py startapp vehicles
```

- Edit `Fedora/settings.py`, adding `vehicles` to `INSTALLED_APPS`
- Edit `Fedora/urls.py`, including `vehicles.urls` in the `url_patterns` list
- Follow along with models, serializers, views, and urls files as shown

SERIALIZER FIELDS

- Boolean:
  - `BooleanField`
  - `NullBooleanField`
- String:
  - `CharField`
  - `EmailField`
  - `RegexField`
  - `SlugField`
  - `URLField`
  - `UUIDField`
  - `FilePathField`
  - `IPAddressField`
- Numeric:
  - `IntegerField`
  - `FloatField`
  - `DecimalField`
- Date and Time:
  - `DateTimeField`
  - `DateField`
  - `TimeField`
  - `DurationField`
- Choice/Selection:
  - `ChoiceField`
  - `MultipleChoiceField`

SERIALIZER FIELDS

- File:
  - `FileField`
  - `ImageField`
- Composite:
  - `ListField`
  - `DictField`
  - `HStoreField`
  - `JSONField`
- Others:
  - `ReadOnlyField`
  - `HiddenField`
  - `ModelField`
  - `SerializerMethodField`
- Custom
- Third-party:
  - DRF Compound Fields
  - DRF Extra Fields
  - djangorestframework-recursive
  - django-rest-framework-gis

SERIALIZER FIELD ARGUMENTS

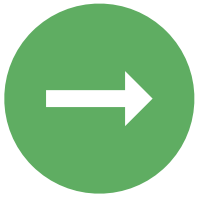
- Common serializer field arguments:
  - `read_only`
  - `write_only`
  - `required`
  - `default`
  - `allow_null`
  - `source`
  - `validators`
  - `error_messages`
  - `label`
  - `help_text`
  - `initial`
  - `style`

SERIALIZER FIELDS DOCUMENTATION

- Full documentation on all the fields:

  `https://www.django-rest-framework.org/api-guide/fields/`

NEXT UP...



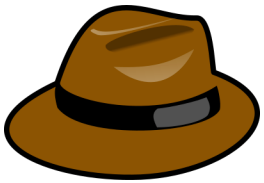
Serializing nested ORM objects

TABLE OF CONTENTS

SERIALIZE RELATED OBJECTS

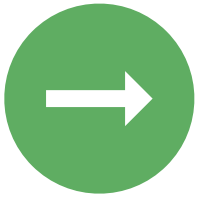
- DRF provides methods for serializing related ORM objects
- Reference foreign keys by id
- Nest serialized relationships

BUILDING ON FEDORA



- Building on top of the vehicles app
- Follow along with models, serializers, views, and urls files as shown
- Create an `admin.py` file for the models if desired
- Run `makemigrations` and `migrate` as needed
- Used the `admin` or the `loaddata` command to add some data

NEXT UP...



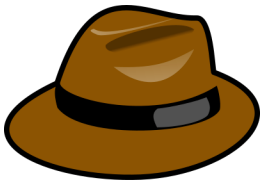
Views, `ViewSets`, and actions

TABLE OF CONTENTS

COMPOUNDED SERIALIZATION

- A common pattern is to declare an API that includes multiple objects
- Everything you might need in a single-page-application
- Declare a view and nest multiple serializers

BUILDING ON FEDORA



- Add another app

```
$ python manage.py startapp api
```

- Edit `Fedora/settings.py`, adding `api` to `INSTALLED_APPS`
- Edit `Fedora/urls.py`, including `api.urls` in the `url_patterns` list
- Best practice: version your API
- Follow along with views and urls files as shown

WITHOUT THE ROUTER

- If you are not using `ViewSets` and `Routers`, you don't get the benefit of having your view listed in the root listing
- Can write a `ViewSet` that does the same thing as any view

VIEWSET CLASSES

- `ViewSet`
- `GenericViewSet`
  - adds the `queryset`, `get_object()`, and `get_queryset()`
- `ModelViewSet`
- `ReadOnlyModelViewSet`

VIEWSET EXAMPLE FROM DOCS

```
class UserViewSet(viewsets.ViewSet):
    """Example empty viewset demoing standard actions handled by a router class."""
    def list(self, request):
        pass

    def create(self, request):
        pass

    def retrieve(self, request, pk=None):
        pass

    def update(self, request, pk=None):
        pass

    def partial_update(self, request, pk=None):
        pass

    def destroy(self, request, pk=None):
        pass
```

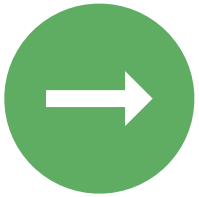
VIEWSETS MIXINS

- `mixins.CreateModelMixin`
- `mixins.ListModelMixin`
- `mixins.RetrieveModelMixin`
- `mixins.UpdateModelMixin`
  - Provides both `update()` and `partial_update()`
- `mixins.DestroyModelMixin`

ADDITIONAL ACTIONS

- Declare your own views/routes by adding an **action**
- Decorate a method of a `ViewSet` subclass
- Common usage: mass operations

NEXT UP...



Putting it altogether

TABLE OF CONTENTS

SAMPLE SPA-ish APPLICATION

- Common use of REST is in Single Page Applications
- Django as the back-end:
  - Managing business logic
  - ORM / database
- Vue.js, React, Angular, etc. as the front-end

SIMPLE SCHEDULING APPLICATION

- Two page: list of schedules and a timeline editor
- Vue.js
  - Model-view Javascript front-end using declarative rendering

    `https://vuejs.org/`
- django-awl
  - Miscellaneous collection of Django utilities
  - RankedModel is an abstract model that provides ordering

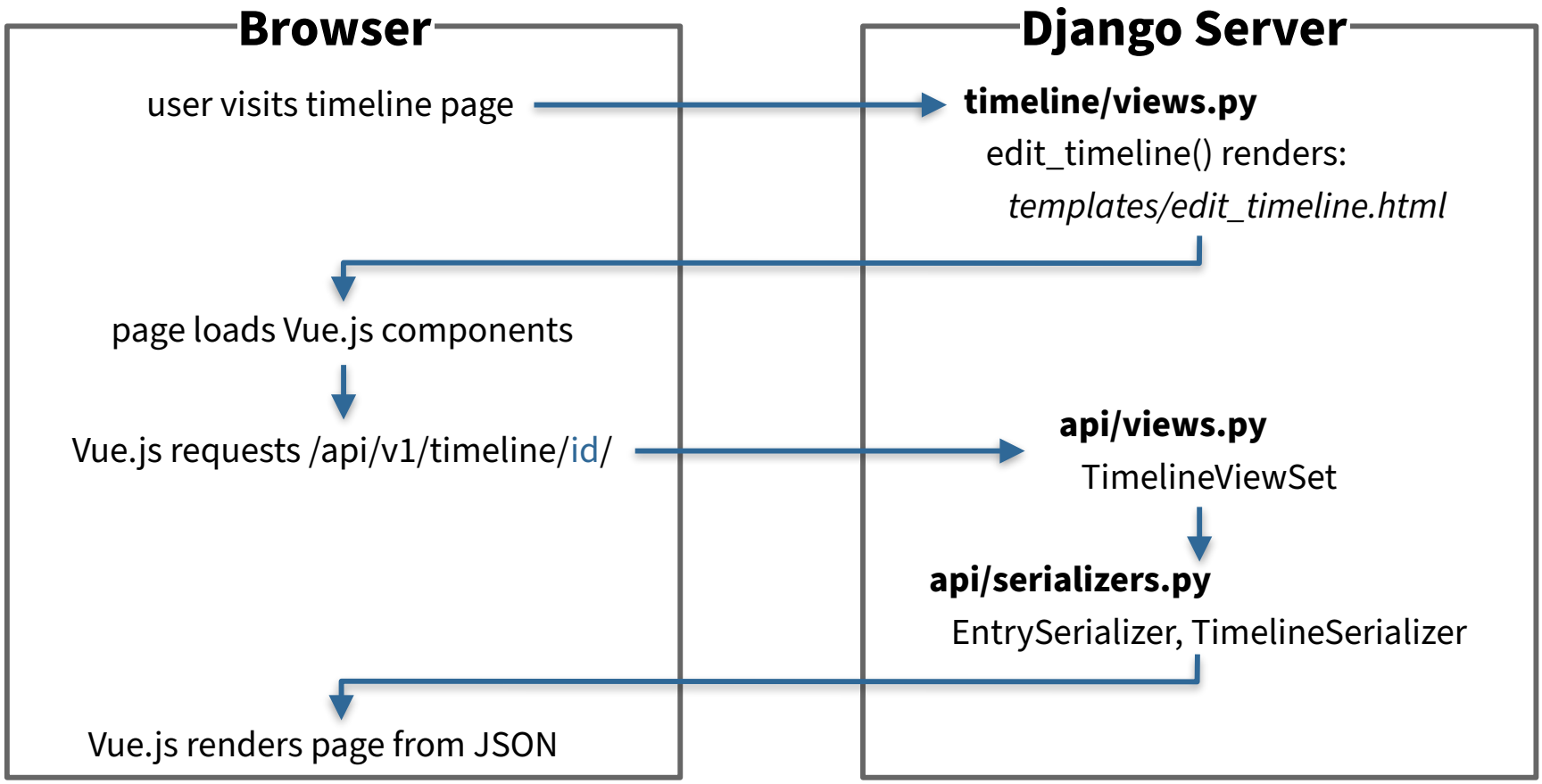
    `https://github.com/cltrudeau/django-awl`
- django-bstrap-modals
  - Django templates for Bootstrap modal dialogs, includes wrappers for REST actions

    `https://github.com/cltrudeau/django-bstrap-modals`

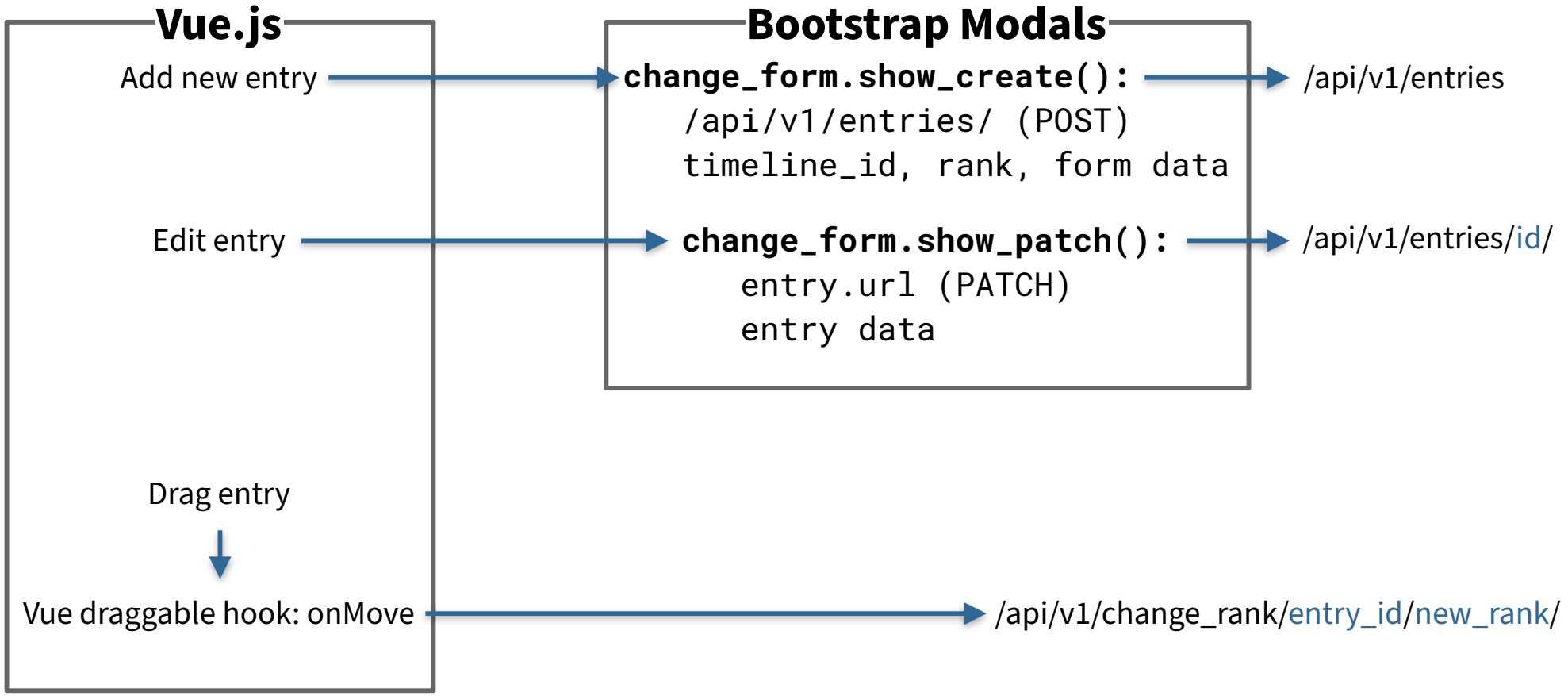
SCHED PIECES

- Data: timeline app models:
  - `Timeline`
  - `Entry`
- API:
  - `api/serializers.py`:
    - `TimelineSerializer`
    - `EntrySerializer`
  - `api/views.py`
    - `TimelineViewSet`
    - `EntryViewSet`
    - `change_rank()` view

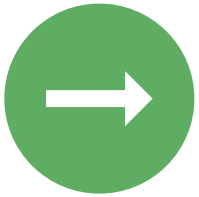
SCHED DATA FLOW



SCHED DATA FLOW: CLIENT SIDE



NEXT UP...



SPA Code Walk Through

TABLE OF CONTENTS

WALK THROUGH OF SCHED

- Code overview only
- DRF parts
- Overview of key parts of Javascript on client side

GETTING STARTED

- Sched's `requirements.txt`:

```
Django==3.1.2
django-awl==1.4.0
django-bstrap-modals==2.1.0
djangorestframework==3.12.1
```

- Install dependencies:

```
$ python -m pip install -r requirements.txt
```

GETTING STARTED

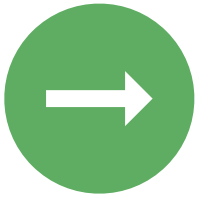
- Setup database and load sample data:

```
$ ./resetdb.sh
```

- Sched's `resetdb.sh`:

```
#!/bin/bash

find . -name "*.pyc" -exec rm {} \;
rm db.sqlite3

python manage.py wipe_migrations
python manage.py makemigrations
python manage.py migrate
python manage.py loaddata timelines
```

NEXT UP...



Summary

TABLE OF CONTENTS

SUMMARY

- REST is a loose protocol for listing, creating, updating, patching, and deleting server side data
- Uses existing HTTP methods and URLs to perform actions
- Django REST framework (DRF) works with Django to perform REST activities with Django model objects and views

SERIALIZERS

- DRF Serializers specify how to serialize and de-serialize data
- `serializer.ModelSerializer` maps Django ORM objects to serialized data with very little code
- Serialize related objects with nested responses
- Serialize Python classes that aren't Django ORM objects

VIEWS AND VIEWSETS

- DRF provides decorators and the `Response` class to add REST compatible views
- `ViewSets` encapsulate multiple HTTP methods, typically all the operations on a Django ORM model
- Register a `ViewSet` with a `Router` and DRF takes care of the URL patterns
- Specify the permissions and query-sets for a `ViewSet`, giving you fine-grained control on what is seen and by whom
- Add custom actions to a `ViewSet` in addition to the usual routes

RENDERERS

- DRF supports several different ways of rendering your data
- By default uses both JSON and a web-based interface
- Control how things are rendered with settings or decorators

FURTHER READING

- Official documentation:

  https://www.django-rest-framework.org/
- Huge list of third-party packages:

  https://www.django-rest-framework.org/community/third-party-packages/#existing-third-party-packages

Thanks!