MyMovie

Tracking movies so you don't have to.

James Mitchell

Introduction

A series of talks covering the “life cycle” of a project.

Talk 2

“Show me the money!”

Writing the server-side code.

- Database
  - Setup a User model and (social) authentication
  - Write the Movie models
- API
  - Document the API
  - Write views and serialisers

Greenfields Development

Barebones Django

Install a virtualenv with django.

```
$ mkvirtualenv mymovie-server
$ pip install django
$ django-admin startproject mymovie
```

Dependancies

Add a requirements.txt with the project dependancies, and

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

```
PyMySQL==0.6.6                  # Python3 compatible library
psycopg2==2.6                   # add Postgresql for Heroku
Django==1.8.2
djangorestframework==3.1.2      # for API
djangorestframework-jwt==1.2.0  # for API authentication
drf-nested-routers==0.9.0
jsonfield==1.0.3                # for ease of use
django-filter==0.10.0

python-social-auth==0.2.10      # social authentication

Pillow==2.8.1
django-resized==0.3.5
sorl-thumbnail==12.2

django-redis==4.0.0
hiredis==0.2.0

django-compressor==1.5

# for serving the site in most places
```

Djano Settings

- Changed the entry module from “mymovie” to “app”
- Refactored settings.py into settings/base.py, settings/test.py, etc
- Pickup sensitive settings from the environment

```
# settings.development
from .base import *

DEBUG = True
TEMPLATE_DEBUG = True
INSTALLED_APPS += ("debug_toolbar",)
INTERNAL_IPS = ("127.0.0.1",)
MIDDLEWARE_CLASSES += ("debug_toolbar.middleware.DebugToolbarMiddleware",)
ALLOWED_HOSTS = ['*']
```

Snippet of settings.base

We define a helper function to get environment settings, and throw an exception if it isn't defined.

```
# settings.base snippet
def get_env_variable(var_name, default=None):
    """
    Get the environment variable or return the default if provided
    or raise an exception
    """
    try:
        if os.environ[var_name] in ('true', 'false'):
            return bool(strtobool(os.environ[var_name]))
        return os.environ[var_name]
    except KeyError:
        if default is not None:
            return default
        error_msg = "Set the %s environment variable" % var_name
        raise ImproperlyConfigured(error_msg)

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'mymovie',
        'USER': get_env_variable('DB_USER', ''),
```

Project files so far

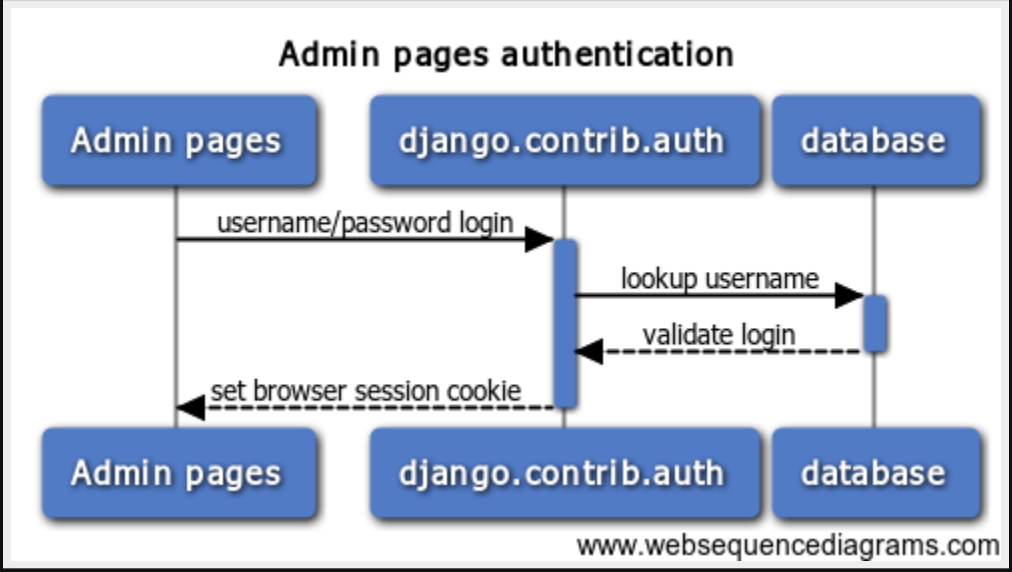
```
.
├── app
│   ├── __init__.py
│   ├── settings
│   │   ├── base.py
│   │   ├── development.py
│   │   ├── __init__.py
│   │   ├── production.py
│   │   └── test.py
│   ├── static
│   │   └── app
│   │       ├── favicon.png
│   │       └── robots.txt
│   ├── tests
│   │   ├── __init__.py
│   │   ├── test_static.py
│   │   └── test_validate.py
│   ├── urls.py
│   └── wsgi.py
├── LICENSE
```

User Authentication

- Support Admin login with username/password
  - django.contrib.auth
- Client UI will perform “social authentication”
  - python-social-auth
- API will authenticate with json web tokens (JWT)
  - djangorestframework-jwt

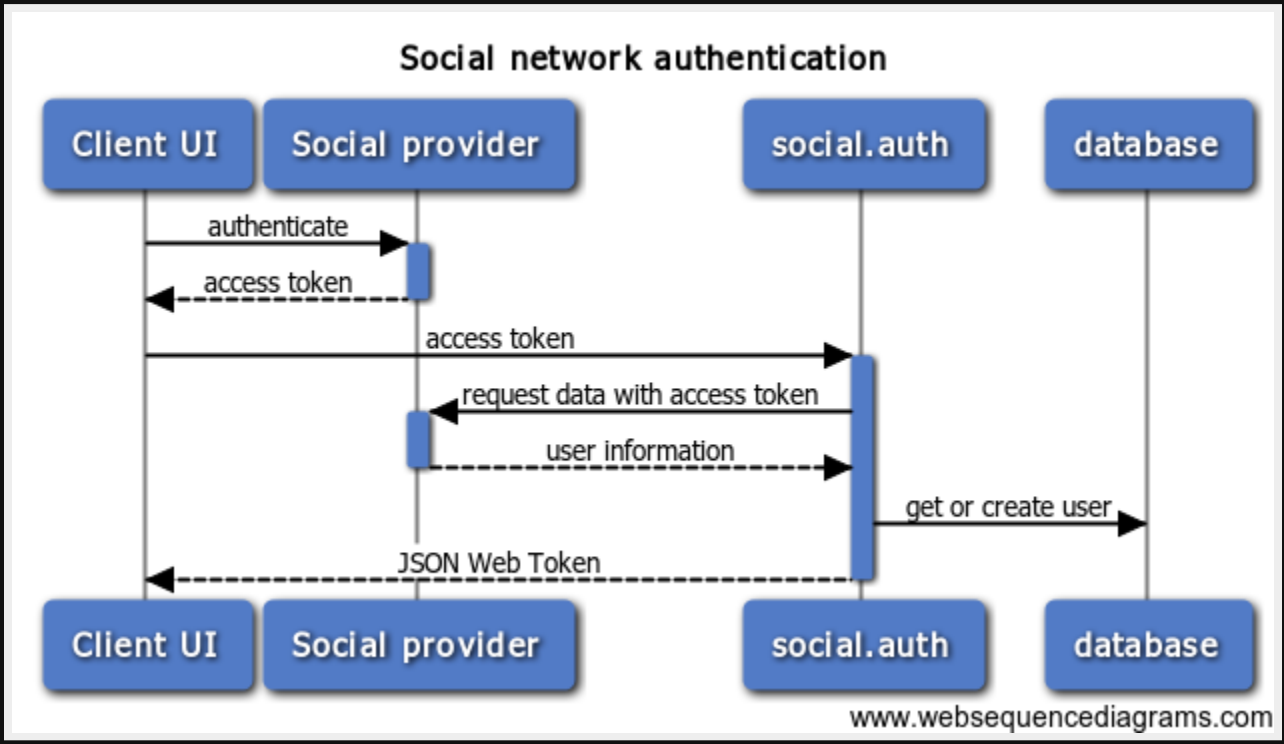
Admin pages

Authentication for django admin using username+password login.



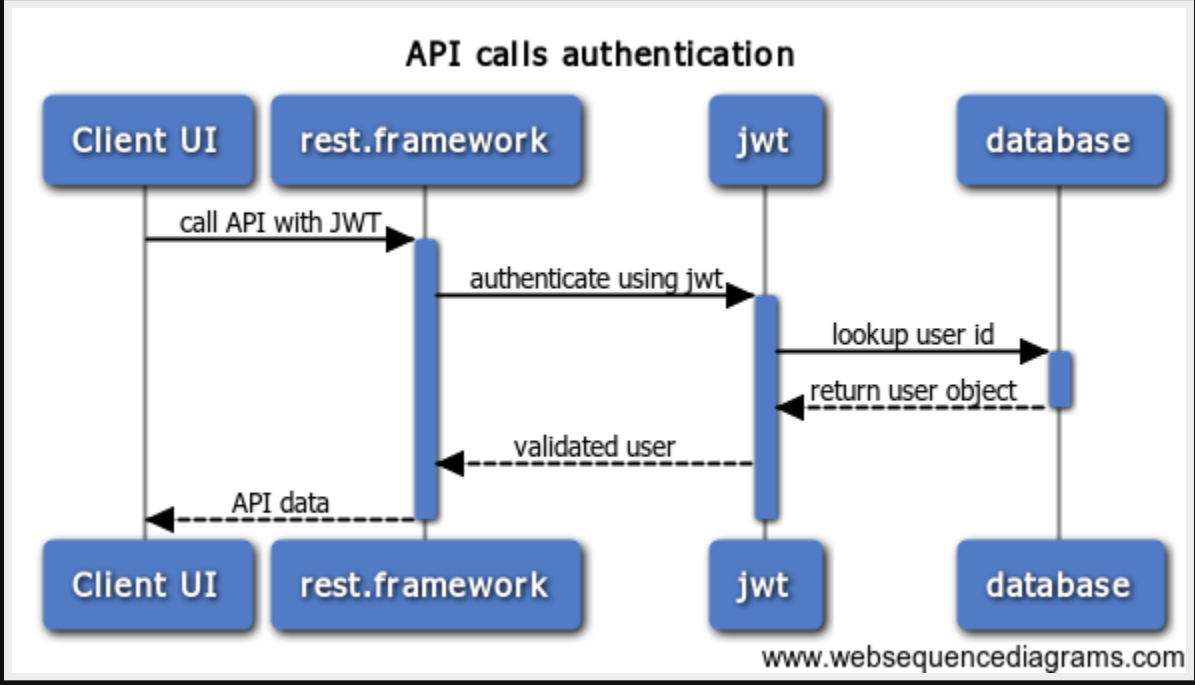
Social authentication

Token, token, who has the token?



API authentication

The JWT is passed as an HTTP Header in the request.



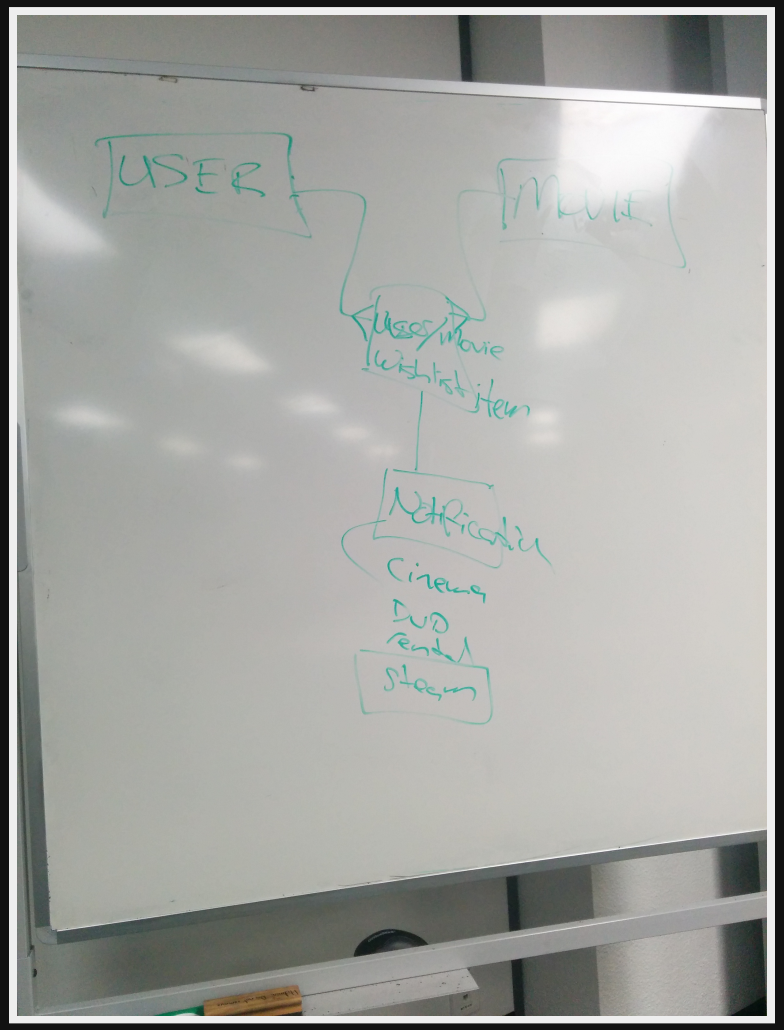
Does it work?

What do the unit tests look like?

```
# -*- coding: utf-8 -*-
from calendar import timegm
from django.contrib.auth import get_user_model
from django.test import TestCase
from mock import patch
from rest_framework import status
from rest_framework.test import APIClient
from rest_framework_jwt import utils
from .factories import UserFactory

class TokenTestCase(TestCase):
    """Methods from the JWT tests to work with tokens."""
    def create_token(self, user, exp=None, orig_iat=None):
        payload = utils.jwt_payload_handler(user)
        if exp:
            payload['exp'] = exp
        if orig_iat:
            payload['orig_iat'] = timegm(orig_iat.utctimetuple())
        token = utils.jwt_encode_handler(payload)
        return token

class TestUserApi(TokenTestCase):
```

Movie Models

All the documentation you really need...



...and what was built

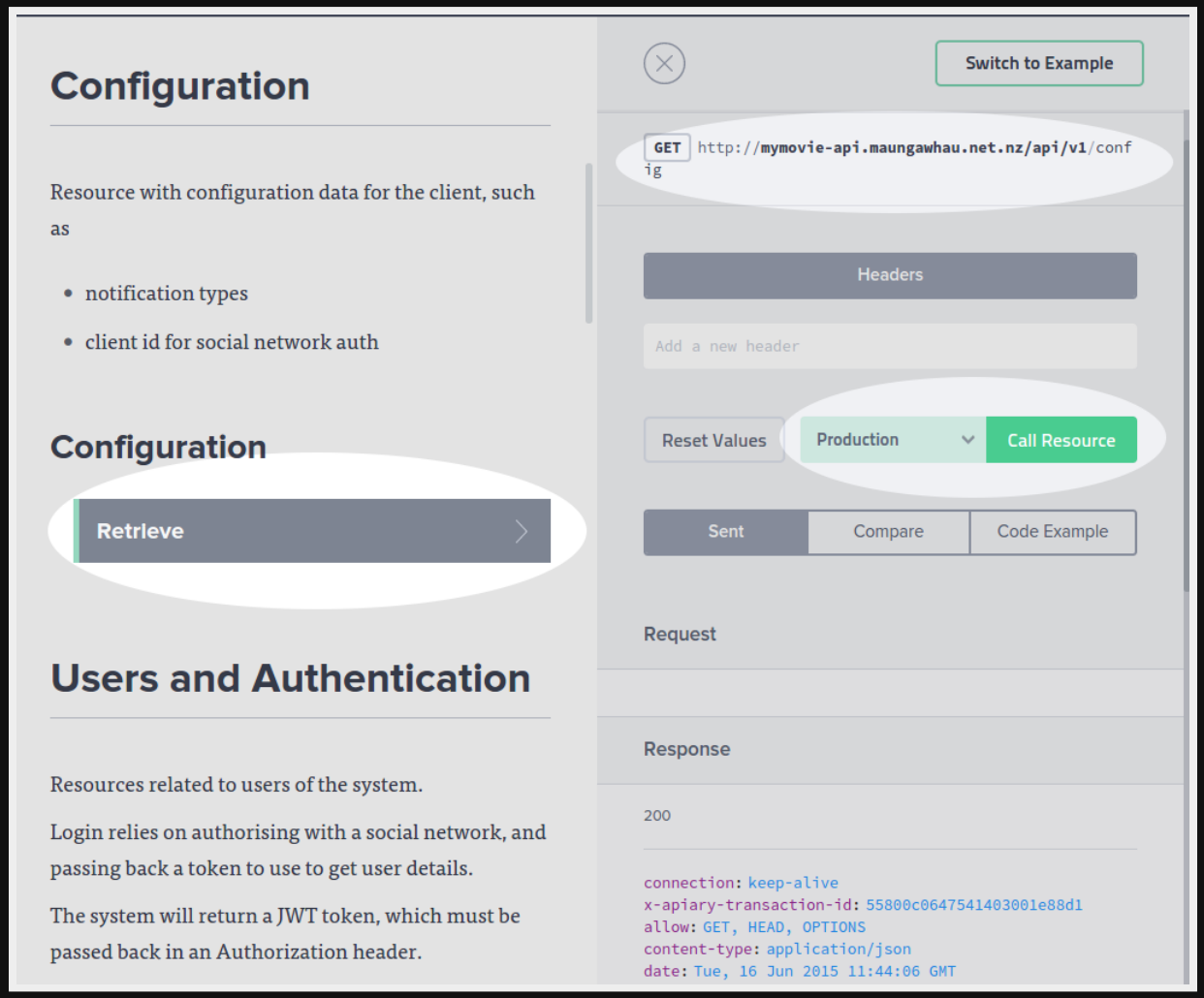
```
# -*- coding: utf-8 -*-
import datetime
from django.conf import settings
from django.db import models
from django.utils.encoding import python_2_unicode_compatible
from jsonfield import JSONField
from movies.service import omdb

SERVICE_OMDB = 'omdb'
SERVICE_CHOICES = (
    (SERVICE_OMDB, 'Open Movie Database'),
    )

class MovieManager(models.Manager):
    def lookup(self, name=None, service=SERVICE_OMDB, service_id=None):
        """
        Lookup a movie.
        If we don't have it already, retrieve data from the service.
        """
        movie = None
        qs = self.filter(
```

OMDB lookup service

```
# -*- coding: utf-8 -*-
import requests
APIHOST = 'http://www.omdbapi.com/'

def get(name=None, service_id=None):
    params = dict(type='movie',
                  r='json')
    if name:
        params.update(dict(s=name))
    if id:
        params.update(dict(i=service_id, tomatoes=True))

    response = requests.get(APIHOST, params=params)

    data = response.json()
    if 'Error' in data and data.get('Response') == 'False':
        raise Exception(data.get('Error'))
    if response.ok:
        return data
    else:
        return None
```

API Documentation

Visit http://docs.mymovie1.apiary.io/ and have a play.



Django Rest Framework

- Adds content negotiation to the Request/Response flow
- Serialization to and from the Django models
- Default endpoints for GET/POST/PUT
  - hooks and decorators to roll-your-own
- Builtin API explorer

Model Serialization

The process of converting the stored data into and out of the transport format, ie JSON or XML

```
# -*- coding: utf-8 -*-
from rest_framework import serializers
from .models import Movie, Watchlist, Notification, ServiceMovie
from users.serializers import UserSerializer

class ServiceSerializer(serializers.ModelSerializer):
    class Meta:
        model = ServiceMovie
        fields = ('id', 'service', 'service_id', 'updated', 'service_data')

class MovieSerializer(serializers.ModelSerializer):
    services = ServiceSerializer(many=True, source='servicemovie_set')

    class Meta:
        model = Movie
        fields = ('id', 'name', 'poster', 'year', 'services')
```

Model Serialization

... gives a JSON response like this...


```
{
    "id": 1,
    "name": "Movie 0",
    "poster": "http://www.kautzer.net/",
    "year": "2015",
    "services": [
        {
            "id": 1,
            "service": "omdb",
            "service_id": "8651",
            "updated": "2015-05-07",
            "service_data": "{'jsonfield': 'Sample data'}"
        }
    ]
}
```

Watchlist Serializer

And now for somethiing completely different.

```
class WatchlistSerializer(serializers.ModelSerializer):
    user = UserSerializer(write_only=True, required=False)
    movie = MovieSerializer(read_only=True)
    notifications = NotificationSerializer(
        read_only=True,
        many=True,
        source='notification_set')

    # fields used when creating a new watchlist
    moviename = serializers.CharField(write_only=True)
    notifywhen = serializers.ListField(write_only=True)
    service = serializers.CharField(write_only=True)
    service_id = serializers.CharField(write_only=True)

    class Meta:
        model = Watchlist

    def create(self, validated_data):
        """
        Save the new watchlist and associated notifications and movies.
        """
```

Create a Watchlist

When you POST

```
{
    "moviename": "Star Trek",
    "service": "omdb",
    "service_id": "tt0796366",
    "notifywhen":  [ "0", "1" ]
}
```

You get

```
{
    "id": 3,
    "movie": {
        "id": 4,
        "name": "Star Trek",
        "poster": "http://ia.media-imdb.com/images/M/MV5BMjE5NDQ5OTE4M15BM15BanBnXkFtZT
        "year": "2009",
        "services": [
            {
                "id": 4,
                "service": "omdb",
                "service_id": "tt0796366",
                "updated": "2015-06-17",
                "service_data": "some stuff goes here"
            }
        ]
    },
    "notifications": [
        {
            "id": 4,
            "notified": false,
```

Views

The view handles the incoming request. It relies on the serializer to validate data, and provide a safe representation to send over the wire.

```
# -*- coding: utf-8 -*-
from rest_framework import viewsets
from custom_rest_framework.viewsets import JWTViewSet
from .models import Movie, Watchlist, Notification
from .serializers import MovieSerializer, \
    WatchlistSerializer, \
    NotificationSerializer

class MovieViewSet(JWTViewSet, viewsets.ModelViewSet):
    serializer_class = MovieSerializer
    queryset = Movie.objects.all()

class WatchlistViewSet(JWTViewSet, viewsets.ModelViewSet):
    serializer_class = WatchlistSerializer

    def get_queryset(self):
        """Filter based on the auth user"""
        user = self.request.user
        return Watchlist.objects.filter(user=user)

    def perform_create(self, serializer):
        """
```

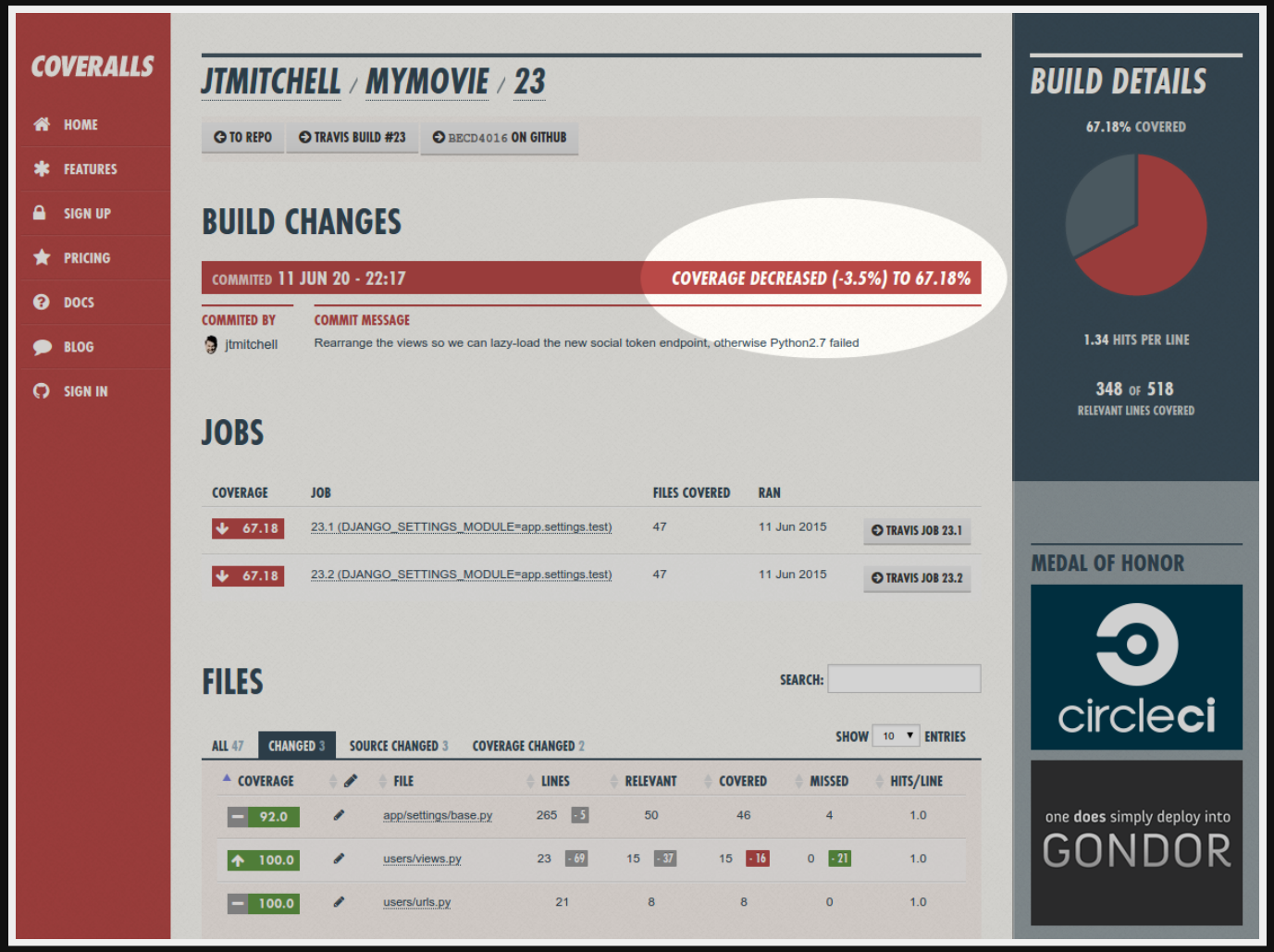
URL Routes / REST Endpoints

Finally the views are connected to the Django URL configuration.

```
# -*- coding: utf-8 -*-
from django.conf.urls import patterns, include, url
from rest_framework_nested.routers import SimpleRouter, NestedSimpleRouter

from .views import MovieViewSet, WatchlistViewSet, NotificationViewSet

router = SimpleRouter(trailing_slash=False)
router.include_format_suffixes = False
router.register(r'v1/movies', MovieViewSet, base_name='movie')
router.register(r'v1/watchlists', WatchlistViewSet, base_name='watchlist')

# register the nested urls for movie routes
wl_router = NestedSimpleRouter(router, r'v1/watchlists',
                               lookup='watchlist', trailing_slash=False)
wl_router.register(r'notifications', NotificationViewSet,
                   base_name='notification')

urlpatterns = patterns('',
                       url(r'', include(router.urls, namespace='movie')),
                       url(r'', include(wl_router.urls, namespace='movie')),
                       )
```

Tests

After all that work I am rewarded with



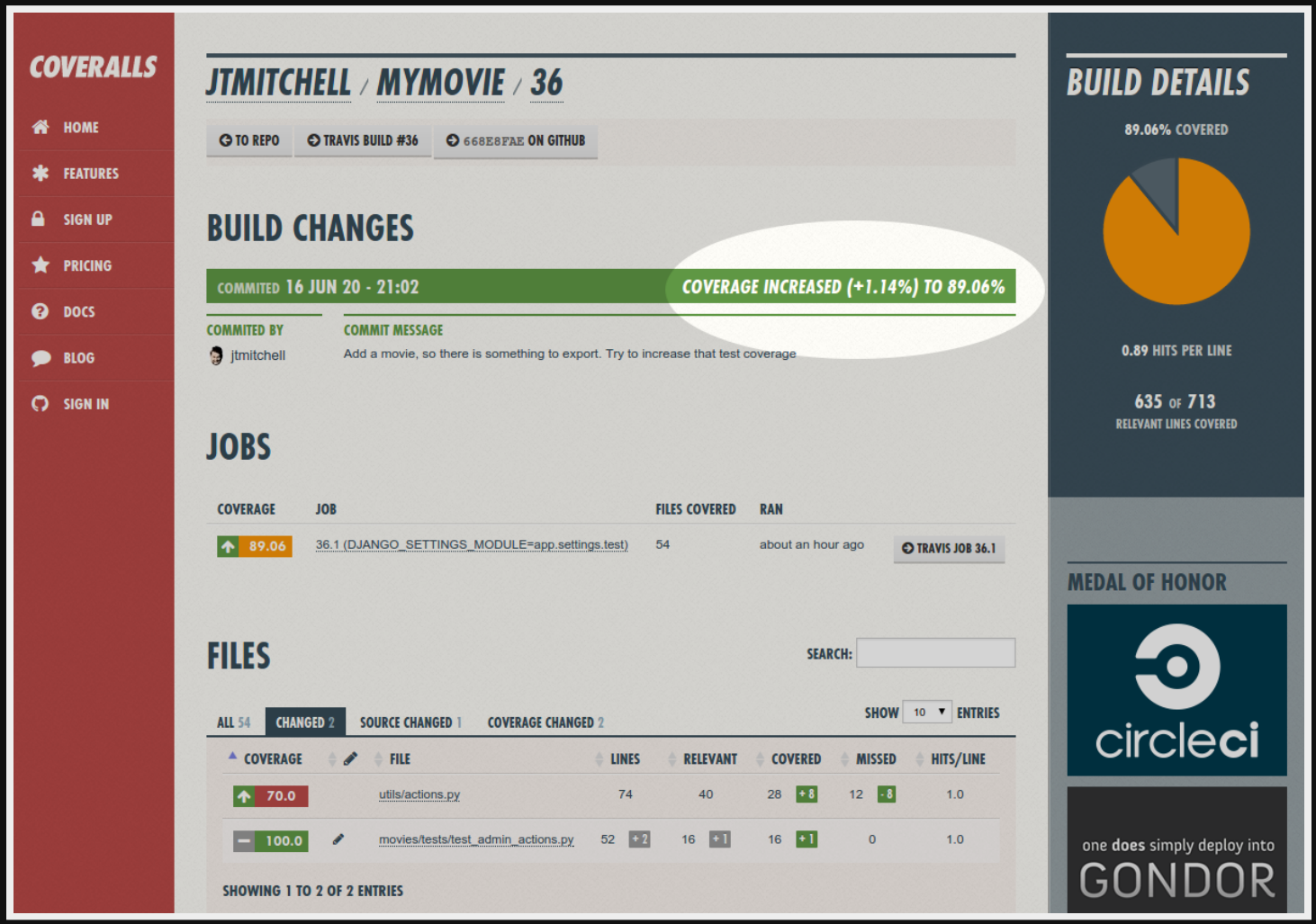
So let's write some Unit Tests

Unit tests check the functionality of a small unit of the code.

```
class TestMovieApi(TokenTestCase):
    """Test the API calls."""
    def setUp(self):
        self.client = APIClient(enforce_csrf_checks=True)
        self.user = UserFactory.create()  # suppress @UndefinedVariable
        self.auth = 'JWT {}'.format(self.create_token(self.user))

    def test_get_movie(self):
        movie = MovieFactory.create()  # suppress @UndefinedVariable
        response = self.client.get(
            '/api/v1/movies/{}'.format(movie.pk),
            HTTP_AUTHORIZATION=self.auth,
            format='json',
            )
        self.assertEqual(response.status_code, status.HTTP_200_OK,
                         'Got error: {}'.format(response.content))
        self.assertEqual(response.data['id'], movie.pk)

    def test_get_watchlist_wrong_user(self):
        watchlist = WatchlistFactory.create()  # suppress @UndefinedVariable
        response = self.client.get(
```

Happy, Happy. Joy, Joy

Look! It went GREEN! I am personally validated!



The Final Slide

ASCII art for the win.

```
.
├── apiary.apib
├── app
│   ├── __init__.py
│   ├── __pycache__
│   ├── settings
│   │   ├── base.py
│   │   ├── development.py
│   │   ├── heroku.py
│   │   ├── __init__.py
│   │   ├── production.py
│   │   ├── __pycache__
│   │   └── test.py
│   ├── static
│   │   └── app
│   │       ├── favicon.png
│   │       ├── logo.png
│   │       ├── logo.svg
│   │       └── robots.txt
│   ├── templates
│   │   └
```

Questions

<insert question here>