Django

Introduction

Examples and source code available on GitHub

https://github.com/giachell/FIS_21-22

Introduction

*“ Django is a high-level Python **web framework** that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It’s **free** and **open source**. ”*

https://www.djangoproject.com

Supporting materials

Django documentation: https://docs.djangoproject.com/en/3.2/

Django cheat sheets: https://bit.ly/3wSqKHC

Tutorials:

- https://learndjango.com/
- https://simpleisbetterthancomplex.com/
- https://dev.to/t/django?ref=hackernoon.com

Podcasts:

https://djangochat.com/

Why django ?

Fast

Secure

Scalable

Free and Open Source

How to install

https://docs.djangoproject.com/en/3.2/intro/install/

1. Install Python (the latest version will be fine) from https://www.python.org/downloads/
2. Install Pip (the package installer for Python) from https://pip.pypa.io/en/latest/installation/
3. Create a Python/Conda Virtual Environment for your Django installation (**Optional**)

   Create the virtual environment:

   ```
   python3 -m venv /path/to/new/virtual/environment
   ```

   Activate the virtual environment:

   ```
   source /path/to/new/virtual/environment/bin/activate
   ```

4. Install Django using Pip:

   ```
   $ python -m pip install Django
   ```

NOTE: The dollar sign ($) indicates the prompt terminal of your system

How to install

https://docs.djangoproject.com/en/3.2/intro/install/

RELAX! Django is already installed in the computers available in the laboratory P140, so you don't need to install it!

However, if you prefer to use your laptop follow the procedure in the previous slide or check out the Django documentation:

https://docs.djangoproject.com/en/3.2/topics/install/#install-the-django-code

Check installation

https://docs.djangoproject.com/en/3.2/intro/install/

Check if Django has been installed correctly:

```
$ python -m django --version
```

Or alternatively:

```
$ django-admin version
```

Check available commands

$ django-admin help --commands

```
(fis) fabiogiachelle@Fabios-MacBook-Air LAB5 % django-admin help --commands
check
compilemessages
createcachetable
dbshell
diffsettings
dumpdata
flush
inspectdb
loaddata
makemessages
makemigrations
migrate
runserver
sendtestemail
shell
showmigrations
sqlflush
sqlmigrate
sqlsequencereset
squashmigrations
startapp
startproject
test
testserver
```

Display a list of the available commands, such as ***startapp*** and ***startproject***.

Run a command

https://docs.djangoproject.com/en/3.2/ref/django-admin/#usage

Alternative Syntaxes to execute some commands

```
$ django-admin <command> [options]
```

```
$ python manage.py <command> [options]
```

```
$ python -m django <command> [options]
```

Example: creates a Django project directory structure for the given project name in the current directory or the given destination.

```
$ django-admin startproject name [directory]
```

First web app

https://docs.djangoproject.com/en/3.2/intro/tutorial01/

1. Create a web app project called ***myproj***

```
$ django-admin startproject myproj
```

2. Run the web application

```
$ python manage.py runserver
```

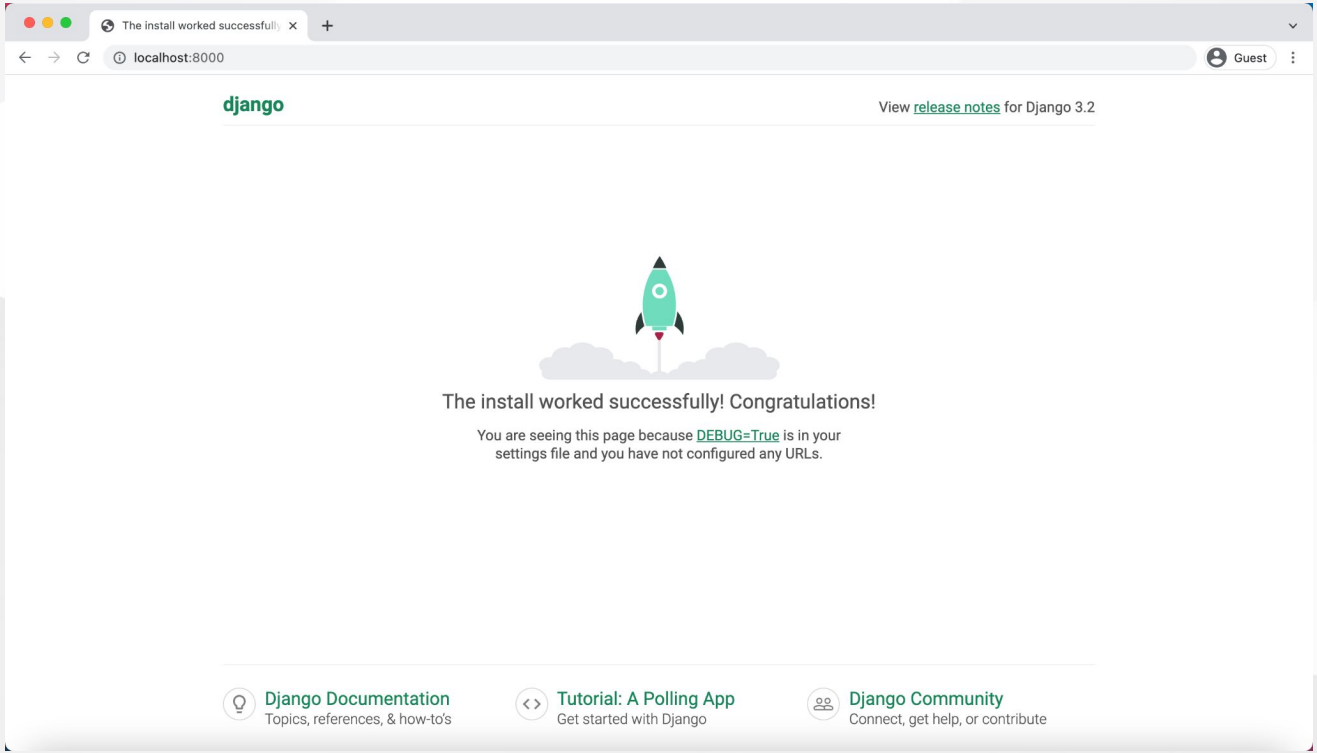
You can also eventually specify the URL used to access the web app, as follows:

```
$ python manage.py runserver localhost:8080
```

Where ***localhost*** is the hostname and ***8080*** is the port used by our web app.

First web app

https://docs.djangoproject.com/en/3.2/intro/tutorial01/

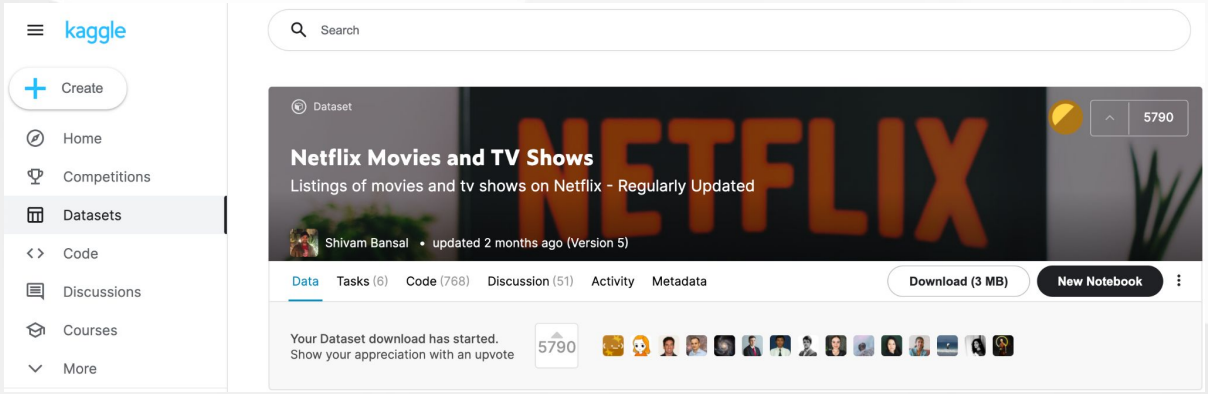


Netflix Movies and TV Shows web app

GOAL: We want to design a web app for managing data concerning Netflix Movies and TV Shows.

For this purpose we use a publicly available dataset from Kaggle, providing a list of movies and tv shows on Netflix.

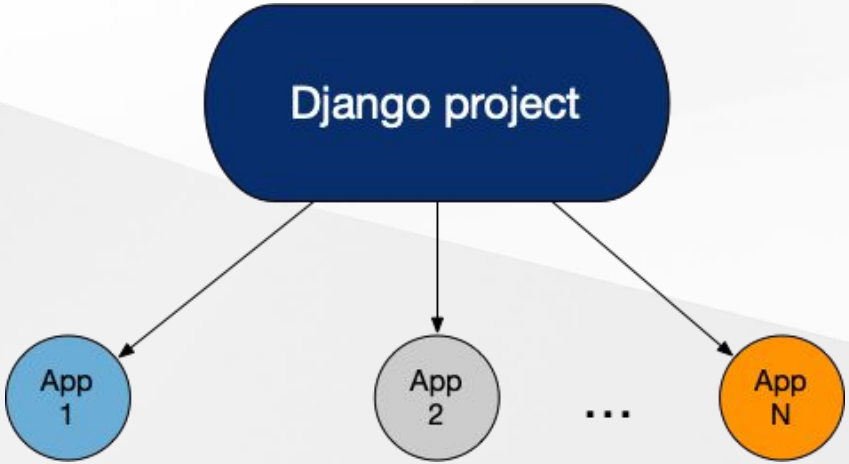
Dataset link: https://www.kaggle.com/shivamb/netflix-shows



Project organization

Project: a Django project can contain one or more ***apps***, that contribute to the functionality of the overall project.

App: a project ***app*** provides specific functionalities, for instance, as web pages for the overall ***project***.



Create a new app in our project

Create a new app called ***netflixapp***

```
$ python manage.py startapp netflixapp
```

This is the project directory organization after running the ***startapp*** command.

```
(fis) fabiogiachelle@Fabios-MacBook-Air myproj % tree
.
├── db.sqlite3
├── manage.py
├── myproj
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-38.pyc
│   │   ├── settings.cpython-38.pyc
│   │   ├── urls.cpython-38.pyc
│   │   └── wsgi.cpython-38.pyc
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── netflixapp
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Create a new app in our project

https://docs.djangoproject.com/en/3.2/ref/applications/#configuring-applications

Now we have to register our ***netflixapp*** inside ***myproj/settings.py*** for using it in our project.

```
INSTALLED_APPS = [
    # include here netflixapp
    'netflixapp',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Databases

https://docs.djangoproject.com/en/3.2/intro/tutorial02/#database-setup

Django can connect to several different databases such as MySQL and PostgreSQL.

By default Django uses a SQLite database, which stores data in a ***db.sqlite3*** file, automatically created in the project directory.

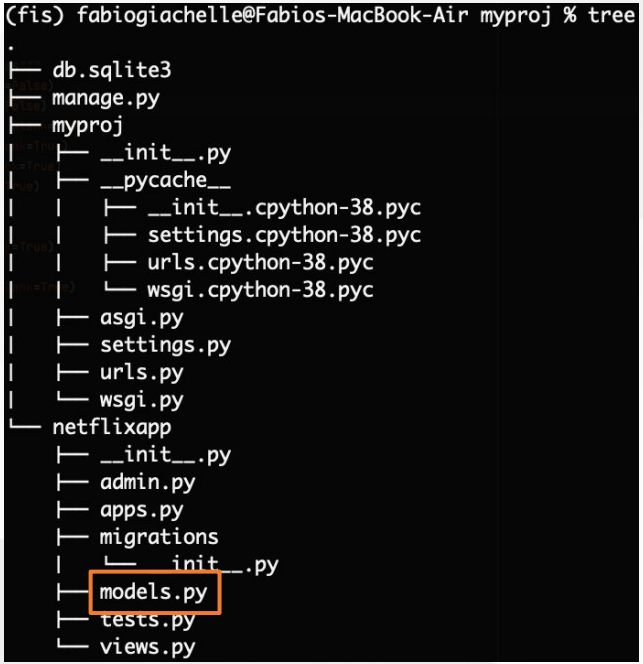
```
(fis) fabiogiachelle@Fabios-MacBook-Air myproj % tree
.
├── db.sqlite3
├── manage.py
├── myproj
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-38.pyc
│   │   ├── settings.cpython-38.pyc
│   │   ├── urls.cpython-38.pyc
│   │   └── wsgi.cpython-38.pyc
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── netflixapp
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations
    │   └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Database models

https://docs.djangoproject.com/en/3.2/topics/db/models/

Django let us design the structure of the data we want to save in the database, by defining custom ***models***. The models specify the prototypes of the entities we want to store in our database. For instance, a ***model*** for a movie represents all the attributes that describe a movie (e.g. title, director, duration, etc …).

We can define our models in the ***models.py*** file, under our project app (i.e., ***netflixapp***).



Database models

https://docs.djangoproject.com/en/3.2/topics/db/models/

```
from django.db import models
# Create your models here.

class Movie(models.Model):
    """A data structure for Netflix movies and series"""
    title = models.CharField(max_length=1000, blank=False)
    type = models.CharField(max_length=1000, blank=False)
    description = models.CharField(max_length=1000,
blank=True)
    director = models.CharField(max_length=1000, blank=True)
    country = models.CharField(max_length=1000, blank=True)
    cast = models.CharField(max_length=1000, blank=True)
    date_added = models.DateField(blank=True)
    release_year = models.IntegerField(blank=True)
    rating = models.CharField(max_length=1000, blank=True)
    duration = models.CharField(max_length=1000, blank=True)
    listed_in = models.CharField(max_length=1000, blank=True)
```

Database models

https://docs.djangoproject.com/en/3.2/topics/db/models/

Now we have to register our model inside ***netflixapp/admin.py*** for the administration interface.

```
from django.contrib import admin

# Register your models here.

from .models import Movie

admin.site.register(Movie)
```

Database models

https://docs.djangoproject.com/en/3.2/topics/db/models/

Now we have to make a ***migration*** called ***netflixapp*** (the name of our app), which tells Django to create the data structures for our models in the database.

```
$ python manage.py makemigrations netflixapp
```

```
(fis) fabiogiachelle@Fabios-MacBook-Air myproj % python3.8 manage.py makemigrations netflixapp
Migrations for 'netflixapp':
  netflixapp/migrations/0001_initial.py
    - Create model Movie
```

Database models

https://docs.djangoproject.com/en/3.2/topics/db/models/

Once the migration has been created, we need to apply it:

```
$ python manage.py migrate
```

```
(fis) fabiogiachelle@Fabios-MacBook-Air myproj % python3.8 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, netflixapp, sessions
Running migrations:
  Applying netflixapp.0001_initial... OK
```

Create a superuser

https://docs.djangoproject.com/en/1.8/intro/tutorial02/#creating-an-admin-user

To access the Django administration interface, we have to create a new admin user account:

```
$ python manage.py createsuperuser
```

Then we are asked to provide the admin credentials. To keep it simple, you can use ***admin*** as username and password. Notice that you need to answer ***y*** (yes) to bypass password validation.

```
(fis) fabiogiachelle@Fabios-MacBook-Air myproj % python3.8 manage.py createsuperuser
Username (leave blank to use 'fabiogiachelle'): admin
Email address: admin@admin.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]:
```

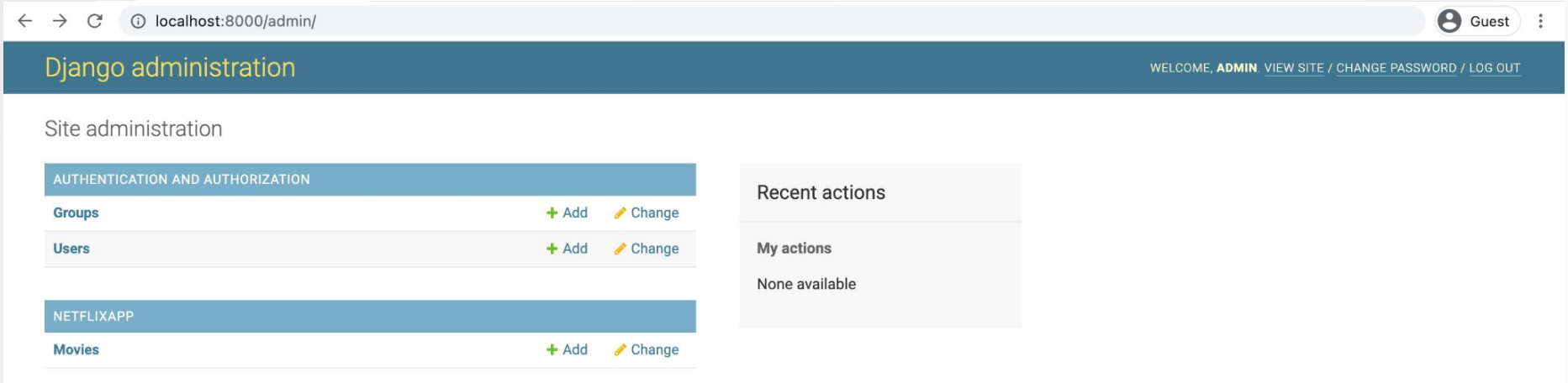
Django administration

https://docs.djangoproject.com/en/3.2/ref/contrib/admin/#module-django.contrib.admin

We can connect to the administration interface at: http://localhost:8000/admin/

From the administration interface the admin can manage all the entities information stored in the database.

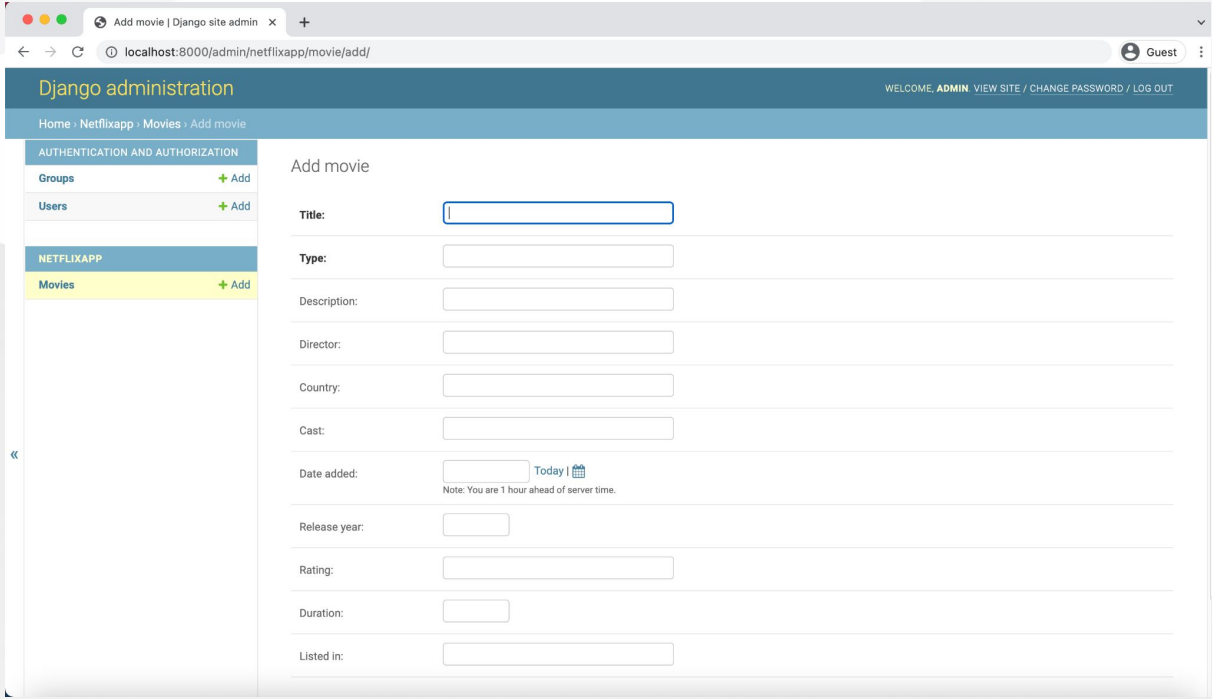
A panel for our ***netflixapp*** with the ***Movie*** model is provided.



Django administration

https://docs.djangoproject.com/en/3.2/ref/contrib/admin/#module-django.contrib.admin

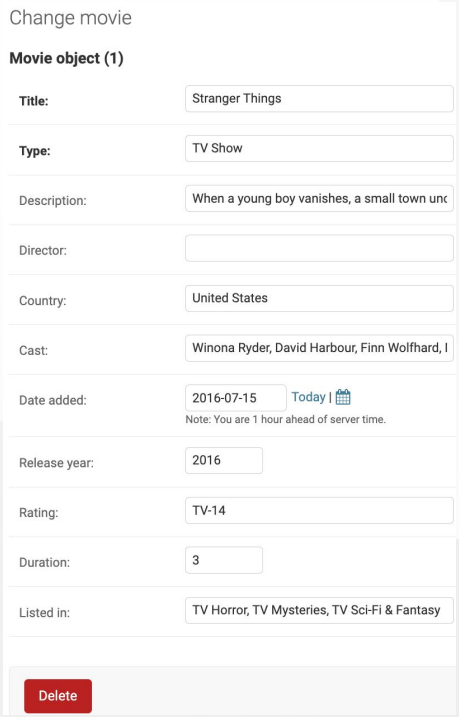
We can add a new movie and manage the existing ones using the administration interface.



Django administration

https://docs.djangoproject.com/en/3.2/ref/contrib/admin/#module-django.contrib.admin

We can add a new movie and manage the existing ones using the administration interface.



Django shell

https://docs.djangoproject.com/en/3.2/ref/django-admin/#shell

Django provides also a ***shell*** to interact, for instance, with the database models.

```
$ python manage.py shell
```

```
(fis) fabiogiachelle@Fabios-MacBook-Air myproj % python3.8 manage.py shell
Python 3.8.12 | packaged by conda-forge | (default, Oct 12 2021, 21:50:56)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.28.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from netflixapp.models import Movie

In [2]: movies = Movie.objects.all()

In [3]: print(movies[0].title)
Stranger Things
```

Django shell

https://docs.djangoproject.com/en/3.2/ref/django-admin/#shell

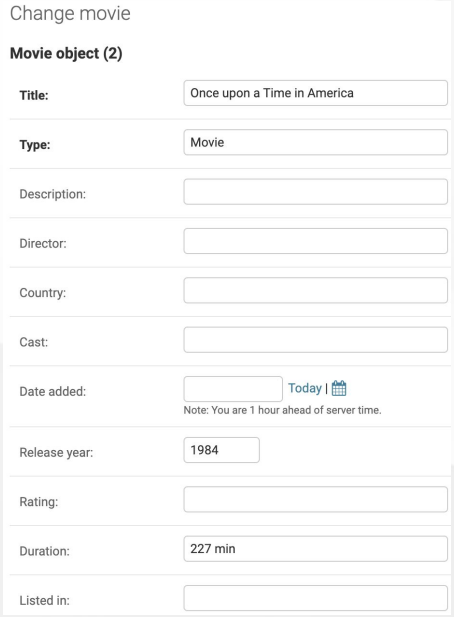
Using the Django ***shell*** we can create new entries (i.e., movies in this case) in our database.

```
(fis) fabiogiachelle@Fabios-MacBook-Air myproj % python3.8 manage.py shell
Python 3.8.12 | packaged by conda-forge | (default, Oct 12 2021, 21:50:56)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.28.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from netflixapp.models import Movie

In [2]: m = Movie(title='Once upon a Time in America', type='Movie', duration='227 min', release_year='1984')

In [3]: m.save()
```

Django shell

https://docs.djangoproject.com/en/3.2/ref/django-admin/#shell

We can inspect the new movie created in the database using the administration interface.



Data ingestion

Now we need to populate our database models with the data from the Netflix dataset.

```
def ingestionDB(filepath):
    """ Save the Netflix dataset into the database """
    # delete existing movies (if any) in the database
    Movie.objects.all().delete()
    # read the dataset
    df = pd.read_csv(filepath)
    # loop over the DataFrame's rows
    for row in df.itertuples():
        # create and save a new movie
        m = Movie(title=row.title, type=row.type, description=row.description,
                  director=row.director, country=row.country, cast=row.cast,
                  date_added=convertDate(row.date_added),
                  release_year=row.release_year, rating=row.rating,
                  duration=row.duration, listed_in=row.listed_in)
        m.save()
```

Data ingestion

After the ingestion procedure, we can use the administration interface to check if our database has been correctly populated with the data from the Netflix dataset.

```
In [5]: movies = Movie.objects.all()

In [6]: len(movies)
Out[6]: 8807
```

This is the number of movies inserted in our database. Since the original number of rows in the dataset is still ***8807*** we can deduce that the ingestion procedure has completed as expected (i.e., all the movies in the dataset have been imported correctly).

Django ORM

https://docs.djangoproject.com/en/3.2/topics/db/queries/#making-queries

Django provides an Object-Relation Mapping (***ORM***) system to simplify the way users can query a database.

```
# import model 'Movie' from our Django models
from netflixapp.models import Movie
```

Creating objects

```
m = Movie(title='Once upon a Time in America', type='Movie', duration='227 min', release_year='1984')
m.save()
```

Saving changes to objects

```
m.director = 'Sergio Leone'
m.save()
```

Retrieving all objects

```
movies = Movie.objects.all()
```

Django ORM

https://docs.djangoproject.com/en/3.2/topics/db/queries/#making-queries

Retrieving specific objects with filters

```
def filteringMoviesDB(title=False, director=False, release_year=False,
                      country=False, cast=False, duration=False,
                      listed_in=False, date_added=False, type=False):
    """ Filter movies according to multiple filters (e.g. director, country, duration, etc...)"""
    movies = Movie.objects.all()

    if title:
        movies = movies.filter(title__contains=title)
    if director:
        movies = movies.filter(director=director)
    if release_year:
        movies = movies.filter(release_year=release_year)
    if country:
        movies = movies.filter(country=country)
    if cast:
        movies = movies.filter(cast__contains=cast)
    if duration:
        movies = movies.filter(duration=duration)
    if date_added:
        movies = movies.filter(date_added=date_added)
    if type:
        movies = movies.filter(type=type)
    if listed_in:
        movies = movies.filter(listed_in__contains=listed_in)
    return movies
```

Django ORM

https://docs.djangoproject.com/en/3.2/topics/db/queries/#making-queries

Retrieving specific objects with filters

```
movies = filteringMoviesDB(listed_in='Comedies', cast='Bill Murray')
printTitles(movies)
```

Output:

- Rock the Kasbah
- Charlie's Angels
- A Very Murray Christmas
- A Glimpse Inside the Mind of Charles Swan III
- Get Smart
- Kingpin
- Stripes
- Zombieland

Thank you for your kind attention