# Django Basics

## Virtual Environment:

Virtual environment provide us the isolated space in same machine/computer. In each environment we can install different libraries, packages etc. and that will be applicable for that particular environment only.

eg:

Your Computer

python2

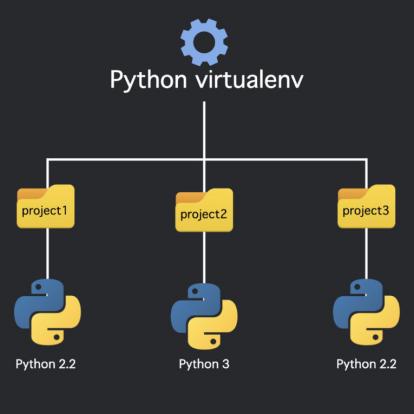
pandas4

python3

pandas1

env1

env2

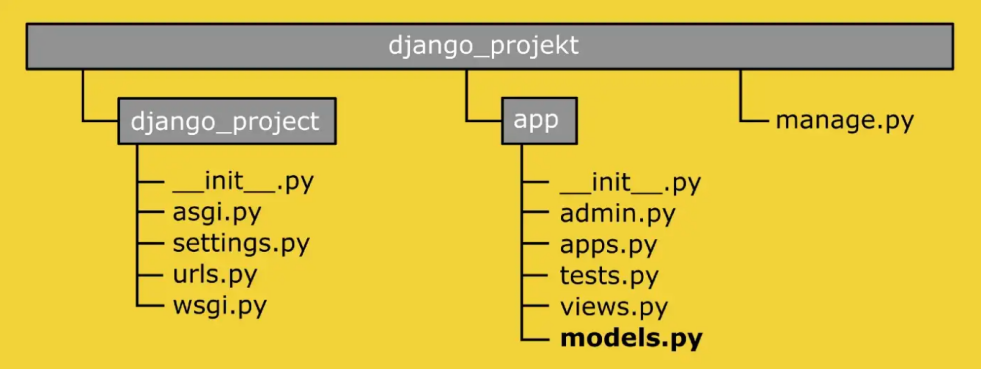


### How to create virtual environment

1. Create a new folder for your working directory.
2. Open and cmd tool.
3. Syntax: **python -m venv name\_of\_environment**
   1. **eg: python -m venv myEnv**
4. New folder will get created for your virtual environment in the current working directory with the same name which you have given in the syntax. All your packages, libraries will get install in that folder.
5. Activating the environment: Before you install any libraries, you need to activate your environment.
   1. For Windows: myEnv\Scripts\activate
   2. For Linux/Mac: source myEnv/bin/activate
6. Check installed packages in your environment: **pip freez**

## Installing Django in virtual environment

Open cmd-> activate your desired environment-> pip install Django



## Creating a project in Django:

1. Create a directory for your project with project name and cd inside that folder.
2. Open cmd.
3. Syntax: **django-admin startproject project\_name**
   1. eg; django-admin startproject watchmate
4. It will create a new folder in current directory with a watchmate name.

### Creating a app inside Django project:

1. Go to the project directory which you have created (directory where manage.py file resides).
2. Syntax: **python manage.py name\_of\_app**
   1. eg; cd waychmate -> python manage.py watchlist\_app

### Run server:

**Syntax: python manage.py runserver**

### Access your application:

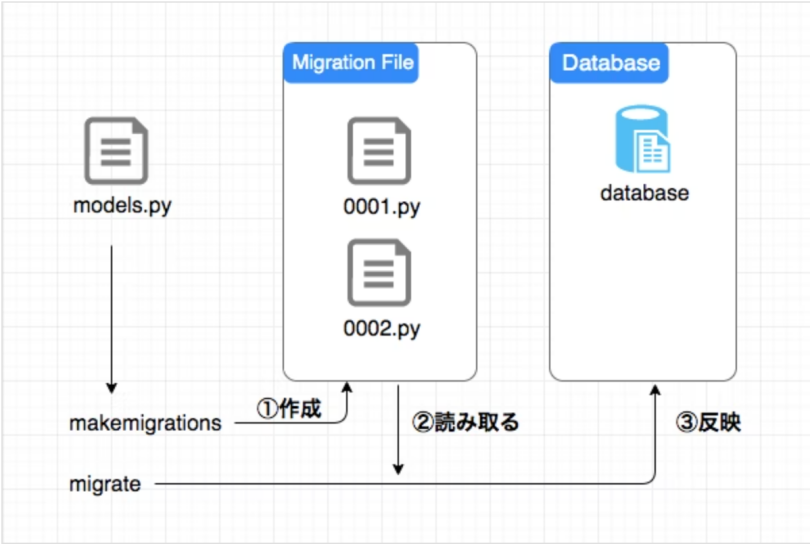
Go to browser and open: <http://127.0.0.1:8000/>

## Registering newly created app in Django project:

1. Open settings.py file which is inside your main project directory, here settings.py which is inside watchmate folder.
2. Inside settings.py file you will find INSTALLED\_APPS list, add your application name in that list. Here, add ‘watchlist\_app’ in the INSTALLED\_APPS list.

## Migrations:

Whenever we do any changes to the database models (Schema is refered as model in Django) we need to migrate them. In the Django we don’t write SQL commands to create schema, rather we define our schema in a python classes inside models.py file which is under your application folder directory. We need to convert these classes into language which SQL understands for that we use makemigrations command. Once that is done we need to run those SQL understandable commands which we generated using migrate command, so to do that we use another command which is migrate.



Step 1: python manage.py makemigrations <app\_name>

Step 2: python manage.py migrate <app\_name>

## Superuser

Django gives us a very interactive admin panel using which we can access data of our database. To access that admin panel we need a super user. We can create a super user as many we want.

Synatx: **python manage.py createsuperuser**

## Others:

### urls.py

urls.py file is used to map our url endpoints with the views. When we visit any particular url, the respctive view get called. views is the place where actual business logic persists.

urls.py file get autogenerated under our main project directory when we create a project.

We can create a new urls.py file under our app folder and write our app related routes in that file.

urls.py

urls.py

app2

Main Project

urls.py

app1

eg;

urls.py file from watchlist project

urls.py file watchlist\_app application

we can send arguments through url to the views.

eg; path(‘<int:pk>’,myView,name=’movie-list’)

myView will recieve a argument in view defination

eg;

def myView(request,pk):

......

### models.py

models.py file get generated with each app. We define our schema in the models.py file. We use classes to define the schema.

eg; models.py file from watchlist\_app application

Syntax:

class SchemaName(models.Model):

name = models.CharField()

........

eg;

class Movie(models.Model):

name = models.CharField(max\_length=50)

description= models.CharField(max\_length=50)

active = models.BooleanField(default=True)

def \_\_str\_\_(self):

return self.name

#### Registering models to admin portal

1. Go to admin.py file which get auto generated when you create any application inside the project.
2. Syntax: **admin.site.register(ModelName)**

### views.py

This file get auto generated when you create any application inside the project. We write our business logic in this file. This views gets mapped with urls.

eg;

from watchlist\_app.models import Movie // Import model from models.py file

def movie\_list(request):

movies = Movie.objects.all() //Fetch all movies from Movie schema, this method will returns a QuerySet and it will get store to movies variable.

data = {

‘movies’:list(movies.values()) //Create a dictionary to send a json response

}

return JsonResponse(data)

def movie\_details(request,pk):

movie = Movie.object.get(pk=pk) //Fetch movie with pk

data = {

‘name’ = movie.name,

‘description’ = movie.description,

‘active’ = movie.active

}

return JsonResponse(data)

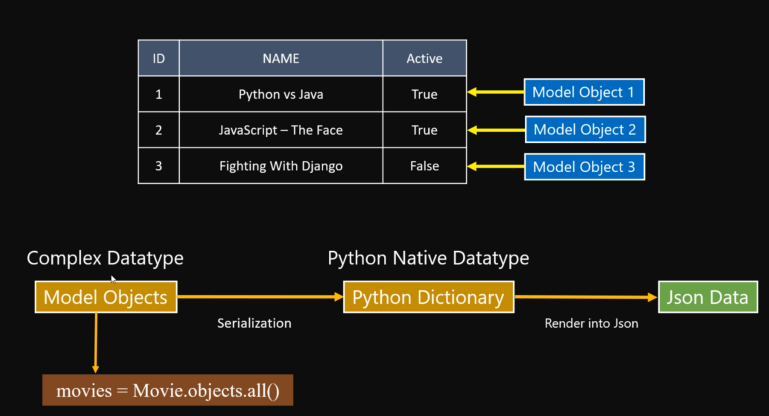
# Django REST Framework

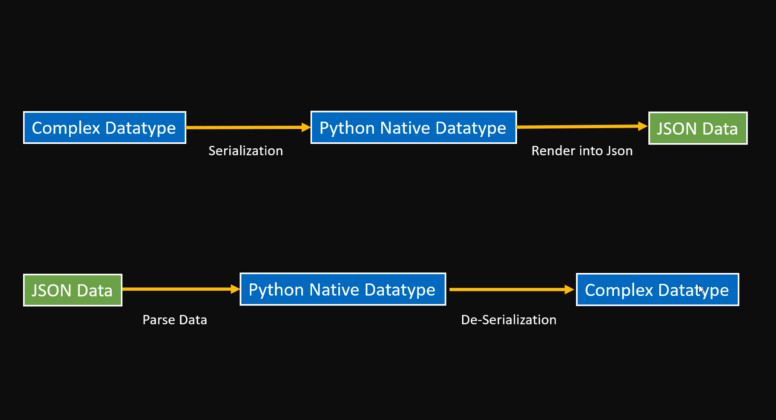
## Installation:

Step 1: **pip install djangorestframework**

Step 2: Add 'rest\_framework' to your INSTALLED\_APPS setting.

## Serialization and De-serialization





Mainly there are two types of serializers

1. serializers.Serializer
2. serializers.ModelSerializer

Mainly there are two types of views

1. Class base views
   1. Generic views
   2. Mixins
   3. Concrete View Classes
   4. ViewSets
2. Function based views
   1. @api\_view()

## serializers.py

### serializers.Serializer

from rest\_framework import serializers

class MovieSerializer(serializers.Serializer): //We have to map all the names with

id = serializers.IntegerField(read\_only=True) //names in Movie model

name = models.CharField()

description= models.CharField()

active = models.BooleanField()

def create(self,validated\_data): //Will get invoke when .save() method gets called in view of this serializer.

return Movie.objects.create(\*\*validated\_data)

def update(self, instance, validated\_data):

instance.name = validated\_data.get('name', instance.name)

instance.description = validated\_data.get('description', instance.description)

instance.active = validated\_data.get('active', instance.active)

instance.save()

return instance

|  |  |  |
| --- | --- | --- |
| **Request Type** | **Method called in view** | **Method invoked in serializer** |
| GET | - | - |
| POST | serializer\_instance.save() | create(self,validated\_data) |
| PUT, UPDATE | serializer\_instance.save() | update(self, instance, validated\_data) |
| DELETE | db\_object.delete() | - |

### serializers.ModelSerializer

Instead of writing all field in serializer manually and mapping them with models, we can use model itself as a serializer.

Syntax:

class <name\_of\_serializer>(serializers.ModelSerializer):

class Meta:

model = <name\_of\_model\_for \_which\_this\_seriallizer\_is>

fields=”\_\_all\_\_” //name of fields from model

eg:

class MovieSerializer(serializers.ModelSerializer):

class Meta:

model = Movie

fields = '\_\_all\_\_'

# exclude = ['active']

# fields = ['id', 'name']

## Function based views:

MOVIE LIST VIEW USING SERIALIZER

@api\_view([‘GET’,’POST’])

def movie\_list(request):

if request.method == ‘GET’:

movies = Movie.objects.all()

serializer = MovieSerializer(movies,many=True)

return Response(serialiser.data)

if request.method == ‘POST’:

serializer = MovieSerializer(data=request.data)

if seializer.is\_valid():

serializer.save() //This will call update or create method inside the MovieSerializer

return Response(serializer.data)

else:

return Response(serializer.error)

@api\_view(['GET', 'PUT', 'DELETE'])

def movie\_details(request, pk):

if request.method == 'GET':

try:

movie = Movie.objects.get(pk=pk)

except Movie.DoesNotExist:

return Response({'Error': 'Movie not found for given id'}, status=status.HTTP\_404\_NOT\_FOUND)

serializer = MovieSerializer(movie)

return Response(serializer.data, status=status.HTTP\_200\_OK)

if request.method == 'PUT':

try:

movie = Movie.objects.get(pk=pk)

except Movie.DoesNotExist:

return Response({'Error': 'Movie not found for given id'}, status=status.HTTP\_404\_NOT\_FOUND)

serializer = MovieSerializer(movie, request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

else:

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

if request.method == 'DELETE':

movie = Movie.objects.get(pk=pk)

movie.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

### Class based views:

from rest\_framework.views import APIView

class MovieListAV(APIView):

def get(self,request):

movies = Movie.objects.all()

serializer = MovieSerializer(movies,many=True)

return Response(serialiser.data)

def post(self,request):

serializer = MovieSerializer(data=request.data)

if seializer.is\_valid():

serializer.save() //This will call update or create method inside the MovieSerializer

return Response(serializer.data)

else:

return Response(serializer.error)

class WatchDetailsAV(APIView):

def get(self, request, pk):

try:

movie = WatchList.objects.get(pk=pk)

except WatchList.DoesNotExist:

return Response({'Error': 'Not found for given id'}, status=status.HTTP\_404\_NOT\_FOUND)

serializer = WatchListSerializer(movie)

return Response(serializer.data, status=status.HTTP\_200\_OK)

def put(self, request, pk):

try:

movie = WatchList.objects.get(pk=pk)

except WatchList.DoesNotExist:

return Response({'Error': 'Movie not found for given id'}, status=status.HTTP\_404\_NOT\_FOUND)

serializer = WatchListSerializer(movie, request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

else:

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

def delete(self, request, pk):

movie = WatchList.objects.get(pk=pk)

movie.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

## Validation in serializers:

We will write validators in serializers.py file

### Field-level validators

write below functions in a serializer class

def validate\_<field\_name>(self,value): //value will get the value of <field\_name>

if len(value)>5:

return value

eg;

def validate\_title(self, value):

if len(value) < 3:

raise serializers.ValidationError("Title should be > 3 characters")

else:

return value

### Object Level validators

we get access to whole object

eg;

def validate(self, data):

if data['title'] == data['description']:

raise serializers.ValidationError(

"Name and Description should not be same")

else:

return data

### Custom validators

We can write our own functions and pass as a argument to field in serializer.

e.g

name = serializers.CharField(validators=[array\_of\_validating\_functions])