

Simple tutorial using Django

By: Manuela Herrera-López

Considerations:

Some considerations about the deployment of the web application:

- This web app was deployed under Manjaro linux OS.
- The python environment was made using anaconda.
- Django version 3.1.2.

Installation:

For the installation I suggest you to follow the steps of this link:

<https://anaconda.org/anaconda/django>

Introduction:

In this tutorial you will learn how to deploy a simple Django web application, this one contains two pages, the home page, where you can add some suggestions about one topic and they will appear on the page. These suggestions are also saved in a sqlite database, and you can access them using the same Django application. The second one is the test page, there you can do a simple test and it will return your grade. The quiz and the score is also saved in the database we mentioned yet.

Some information about Django: this is a python framework that implements MVC Or MVT. This is an implementation of the control inversion named model-view-controller. In the Django world it is known as model-view-template, the template does the role of the controller in this case.

Deployment:

For creating the project, we use the following command in the shell:

```
$ django-admin startproject mysite
```

this will create the following structure (you can see that using the *tree* command)

```
mysite/  
  manage.py  
  mysite/  
    __init__.py  
    settings.py  
    urls.py  
    asgi.py  
    wsgi.py
```

The manage.py file will be the most important for the deployment. Through that we will create the other apps, and also create the migrations. In my situation, the project is named "mulatasweb".

Then we proceed to create the different apps, an app is a web application that does something, and the project can contain lots of them.

TestApp

1. The first app I'll create will be the "testApp" app, this one contains the test I've already mentioned. For doing so, we just type:

```
[mhl@mulata mulatasweb]$ python manage.py startapp testApp
```

This will create a similar structure to this one:

```
(venv) (base) [mhl@mulata mulatasweb]$ tree testApp/
testApp/
├── admin.py
├── apps.py
├── __init__.py
├── migrations
│   ├── 0001_initial.py
│   ├── 0002_auto_20201022_2216.py
│   ├── 0003_grade.py
│   ├── __init__.py
│   └── __pycache__
│       ├── 0001_initial.cpython-38.pyc
│       ├── 0002_auto_20201022_2216.cpython-38.pyc
│       ├── 0003_grade.cpython-38.pyc
│       └── __init__.cpython-38.pyc
├── models.py
├── __pycache__
│   ├── admin.cpython-38.pyc
│   ├── apps.cpython-38.pyc
│   ├── __init__.cpython-38.pyc
│   ├── models.cpython-38.pyc
│   ├── urls.cpython-38.pyc
│   └── views.cpython-38.pyc
├── templates
│   └── testApp
│       └── test.html
├── tests.py
├── urls.py
└── views.py
```

You can ignore the migrations and the pycache folder.

2. Then, we need to tell Django that we've created another app. For this purpose, we go to the folder of our project and edit the settings.py file as follows

```
# Application definition

INSTALLED_APPS = [
    'homepage.apps.HomepageConfig',
    'testApp.apps.TestappConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

3. Then, we're going to create the models (tables) of our test application. So we go to the testApp dir and edit the models.py file, like the following one:

```
from django.db import models

class Quiz(models.Model):
    question = models.CharField(max_length=100)
    option1 = models.CharField(max_length=100)
    option2 = models.CharField(max_length=100)
    option3 = models.CharField(max_length=100)

class Grade(models.Model):
    score = models.IntegerField(default=0)
```

There, we create two classes that will be our different tables. The first one is named Quiz, it contains the different attributes as the question, and the three different options of the Quiz. The other one, saves the score you obtained by doing the test. All of the two models contain an id by default, so we don't have to worry about that.

4. After that, we have to create the migrations for the test application, so we type the following without the hashtag:

```
[mhl@mulata mulatasweb]$ python manage.py makemigrations testApp
```

and then

```
[mhl@mulata mulatasweb]$ python manage.py migrate testApp
```

This will create the models of the application.

5. Then, we want to show our models in the admin/ site. So we go to edit on the testApp dir, the admin.py file as follows:

```
from django.contrib import admin
from .models import *

admin.site.register(Quiz)
admin.site.register(Grade)
```

There we are just registering both of our models

6. Then we have to create a super user for accessing the admin page. For that purpose we type:

```
(venv) (base) [mhl@mulata mulatasweb]$ python manage.py createsuperuser
```

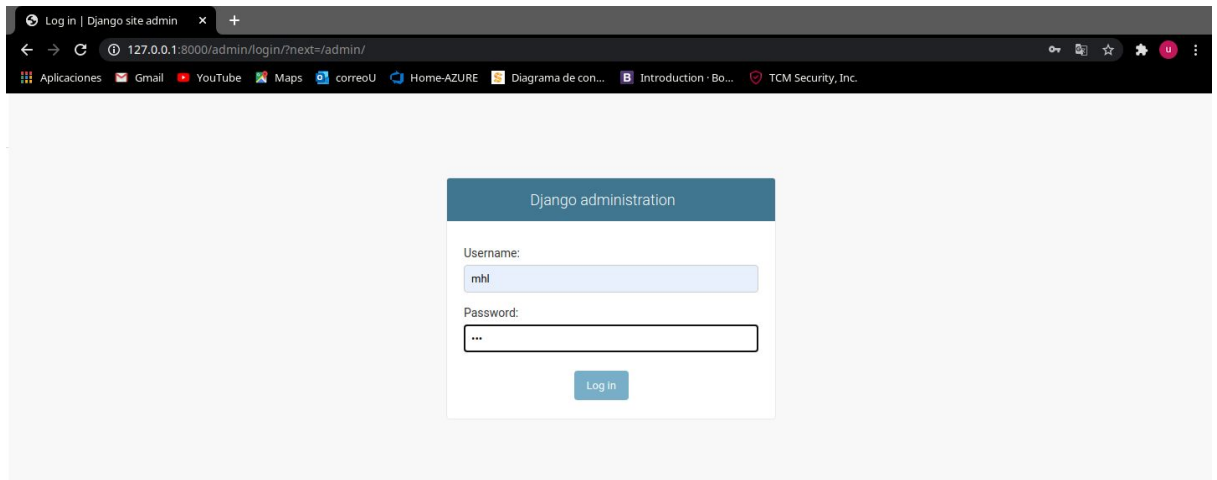
That command will help you to create one, it will ask for username, email and password.

7. Then, we're going to add the different questions of the Quiz in our database. For this purpose we type:

```
(venv) (base) [mhl@mulata mulatasweb]$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
October 24, 2020 - 00:00:38
Django version 3.1.2, using settings 'mulatasweb.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

and we go to the browser and type the blue url, followed by 'admin/'



We fill the camps with the information we previously gave to the super user and we access the database.

After logging in we will see something like this:



we go to the Quiz section and we create our question. Mine looks like this:

Change quiz HISTORY

Question:

Option1:

Option2:

Option3:

Delete Save and add another Save and continue editing SAVE

I added 5 questions, so you have to create as many objects as questions you have.

8. Now we're going to edit the views.py file, there we redirect the requests from one or another web app. Mainly, our views will contain something like the following:

```

from django.http import HttpResponseRedirect
from django.shortcuts import render, redirect
from .models import *

# Create your views here.

def send(request):
    quiz = Quiz.objects.all()
    return render(request, 'testApp/test.html', {'quiz': quiz})

```

The send method receives a request, and shows all the objects we have in the quiz model, the one we saw before. Then it returns a render function which is called with the request, the template we are going to create later and in this case, a dictionary of the questions.

9. Now we're ready to create the template of our test application. For that purpose we have to create a dir named "templates" under our testApp, under that one we've already created, we create another one named as our app, in my case is "testApp". for that do the following:

```

(venv) (base) [mhl@mulata mulatasweb]$ mkdir -p testApp/templates/testApp

```

There we create a file named "test.html", mine has the following structure:

```

{% extends 'homepage/base.html' %}
{% load static %}

{% block content %}
    <h1>How much do you know about feminism?</h1>
    <div>
        <form action="/grade/" method="post"> {% csrf_token %}
            {% for i in quiz %}
                <table>
                    <tr>
                        <td class="text" name> <strong>{{ i.id }} ) {{ i.question }}?</strong> </td>
                    </tr>
                    <tr>
                        <td><input type="radio" name="{{ i.id }}" value="Option1"
                            value="{{ i.option1 }}">{{ i.option1 }}</td>
                    </tr>
                    <tr>
                        <td><input type="radio" name="{{ i.id }}" id="Option2"
                            value="{{ i.option2 }}">{{ i.option2 }}</td>
                    </tr>
                    <tr>
                        <td><input type="radio" name="{{ i.id }}" id="Option3"
                            value="{{ i.option3 }}">{{ i.option3 }}</td>
                    </tr>
                </table> <br>
            {% endfor %}
            <input type="submit" name="Submit"/>
        </form>
    </div>
{% endblock content %}

```

Looking at it line by line:

```

{% extends 'homepage/base.html' %}
{% load static %}

```

In that line we are indicating that we're implementing a base html which is located in the another app we'll create later, that base contains the css information, and the basic style of all the html


```
{% block content %}
    <h1>How much do you know about feminism?</h1>
    <div>
        <form action="/grade/" method="post"> {% csrf_token %}
            {% for i in quiz %}
                <table>
                    <tr>
                        <td class="text" name> <strong>{{ i.id }} ) {{ i.question }}?</strong> </td>
                    </tr>
                    <tr>
                        <td><input type="radio" name="{{ i.id }}" value="Option1"
                            value="{{ i.option1 }}">{{ i.option1 }}</td>
                    </tr>
                    <tr>
                        <td><input type="radio" name="{{ i.id }}" id="Option2"
                            value="{{ i.option2 }}">{{ i.option2 }}</td>
                    </tr>
                    <tr>
                        <td><input type="radio" name="{{ i.id }}" id="Option3"
                            value="{{ i.option3 }}">{{ i.option3 }}</td>
                    </tr>
                </table> <br>
            {% endfor %}
            <input type="submit" name="Submit"/>
        </form>
    </div>
{% endblock content %}
```

There, we represent that we're adding content with the "{% block content %}" Structure.

Then we find the following line:

```
<form action="/grade/" method="post"> {% csrf_token %}
```

There, we call an action that will be another method in view, and we add the token to give privacy to the form.

After that we're iterating our dictionary named "quiz", and we establish our question by numerating by the id we've already talked about, and the question attribute of that one. Then we create a radio type of input, which contains the id of the question, the name associated by the id and the value which is each option.

10. Then, we're going to add the grade view, the one that we call in the action attribute before. For that, we go back to the views file and create another method.


```
def grade(request):
    quiz = Quiz.objects.all()
    cont = 0
    answer1 = request.POST['1']
    answer2 = request.POST['2']
    answer3 = request.POST['3']
    answer4 = request.POST['4']
    answer5 = request.POST['5']

    if answer1 == 'Option3': cont += 1
    if answer2 == 'Option1': cont += 1
    if answer3 == 'Option3': cont += 1
    if answer4 == 'Option1': cont += 1
    if answer5 == 'Option2': cont += 1

    new_grade = Grade(score=cont)
    new_grade.save()

    return HttpResponse("your score: " + str(cont))
```

There, we create a variable named “cont”, which will contain the score you got in the quiz. and also we create different variables for each answer. We iterate the POST method by the name attribute in the html, which is a number between 1 and 5.

Then we just ask for the value of it, if it's the correct, you obtain one point. Then we save the grade in our database and return a HttpResponse with the score.

11. Then, we create a file named “urls.py” in our testApp folder. There we call the different view with the path

```
from django.urls import path
from . import views

urlpatterns = [
    path('test/', views.send),
    path('grade/', views.grade)
]
```

And that's all about the testApp!

Homepage

Now, we're going to create the other app, it will be the homepage. For doing so, we reproduce the same steps of the testApp. the first and second one.

1. We create our models as we did in the past app, mine has a class named SuggestedLink, which has an id, the link of the suggestion and the name organization, it looks like this:

```
class SuggestedLink(models.Model):
    id_link = models.AutoField(primary_key=True)
    link = models.TextField()
    org_name = models.CharField(max_length=50);
```

Then we reproduce the fourth and fifth step of the testApp.

2. Now we're going to create the views of the homepage app, it will contain two methods, the first one will show all the suggestions we've made in our table of suggestions and the another one will add the current item we add on the page. It looks like this:

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .models import *

def suggest(request):
    all_suggested = SuggestedLink.objects.all()
    return render(request, 'homepage/homepage.html', {'items': all_suggested})

def addItem(request):
    new_item = SuggestedLink(link=request.POST['link'], org_name=request.POST['org_name'])
    new_item.save()
    return HttpResponseRedirect('/')
```

3. Now we create the templates dir. First of all we create the base template, the one we saw earlier, it's named "base.html", it looks like this:

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">

    <link rel="stylesheet" type="text/css" href="{% static 'homepage/fem.css' %}">

    {% if title %}
        <title>Mulata's Web - {{ title }}</title>
    {% else %}
        <title>Mulata's Web</title>
    {% endif %}
</head>
```

There we load the static folder, which contains the fem.css and some images we'll see later.

Then, we find the body part of the html, which contains a table of images, and the content section

```
<body>

<table width="500" cellpadding="5" align="center">

    <tr>
        <td align="center" valign="center">
            
        </td>

        <td align="center" valign="center">
            
        </td>

        <td align="center" valign="center">
            
        </td>
    </tr>

</table>
{% block content %}{% endblock %}
</body>
</html>
```

- Now, we create the other html page named "homepage.html", this one contains the form which calls the addItem method when the request is made, and shows, item by item, the ones that are saved in the database

```
{% extends 'homepage/base.html' %}
{% load static %}

{% block content %}
    <h1>Mulata's web</h1>
    <ul>
        {% for item in items %}
            <li><strong> {{ item.org_name }} </strong> {{ item.link }}</li>
        {% endfor %}
    </ul>

    <form action="/addItem/" method="post"> {% csrf_token %}
        <p>Organization link</p>
        <input type="text" name="link"/> <br>
        <p>Organization name</p>
        <input type="text" name="org_name"/>
        <input type="submit" name="Add"/>
    </form>

    <a href="/test/" class="button">the test!</a>
{% endblock %}
```

5. Then, we create the urls.py file for the homepage app. Mine looks like this:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.suggest),
    path('addItem/', views.addItem),
]
```

And that's all for the homepage app!

Then we have to import all the urls of the apps in the project, so we go to the "mulatasweb/urls.py", and add them like this:

```

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('testApp.urls')),
    path('', include('homepage.urls')),
]

```

And then, we run again the server to see how it works. for doing so, we type:

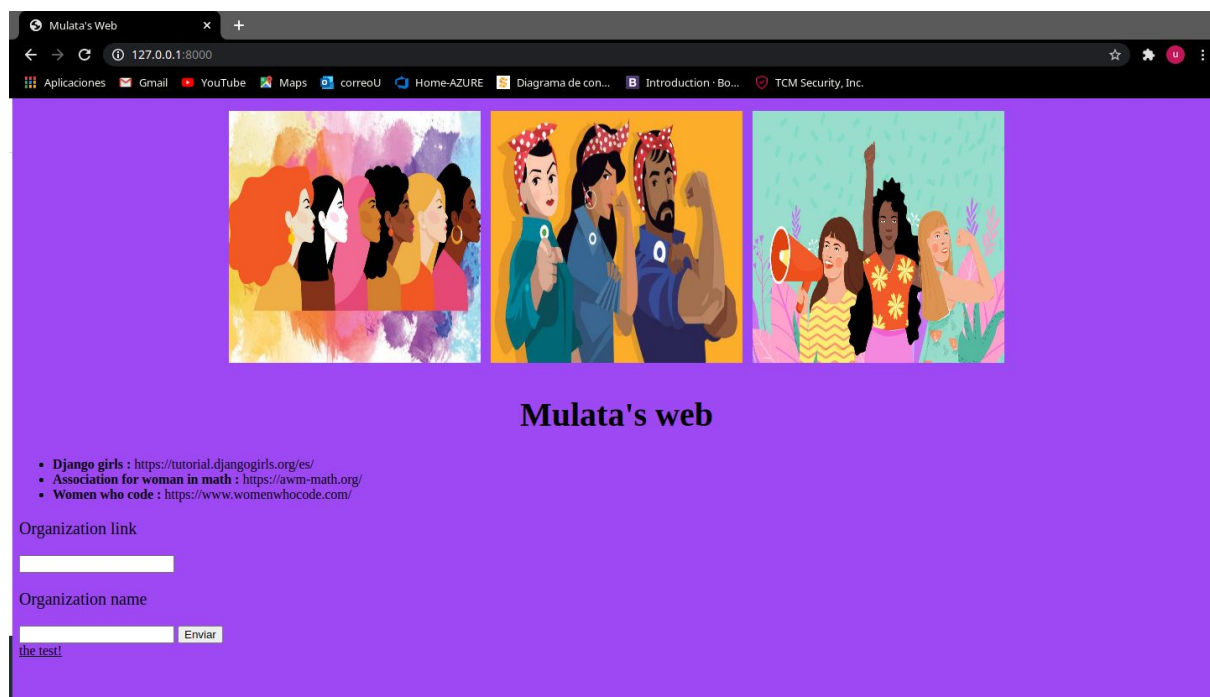
```

(venv) (base) [mhl@mulata mulatasweb]$ python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
October 24, 2020 - 00:00:38
Django version 3.1.2, using settings 'mulatasweb.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

```

And it looks like this:



Now we can add another item, let's see:



Mulata's web

- **Django girls** : <https://tutorial.djangogirls.org/es/>
- **Association for woman in math** : <https://awm-math.org/>
- **Women who code** : <https://www.womenwhocode.com/>
- **National Girls Collaborative Project** : <https://ngcproject.org/>

Organization link

Organization name

Enviar

[the test!](#)

Then we can go to the link that goes to the test




Mulata's Web

Mulata's Web

National Girls Collaborative Project

127.0.0.1:8000/test/

Aplicaciones Gmail YouTube Maps correoU Home-AZURE Diagrama de con... Introduction - Bo... TCM Security, Inc.



How much do you know about feminism?

1) How many were the feminist waves?

- ☒ Three waves
- ☐ One wave
- ☐ Four waves

2) Where did the first-wave started?

- ☒ England and United States
- ☐ China
- ☐ South America mostly

3) Which was one of the first achievements obtained by feminists?

- ☒ Emancipate woman in all topics
- ☐ Remove gender roles
- ☐ Women's suffrage

3) Which was one of the first achievements obtained by feminists?

- ☐ Emancipate woman in all topics
- ☐ Remove gender roles
- ☐ Women's suffrage

4) Which were the main topics of the second-wave?

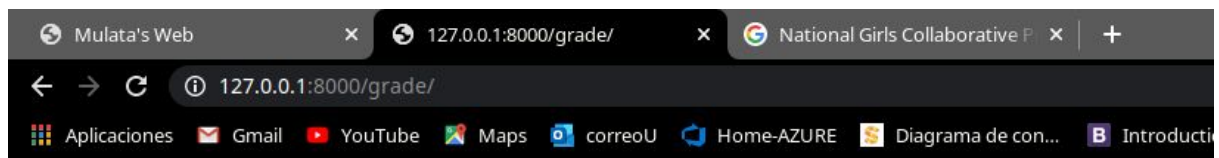
- ☐ Inequality, sexuality, family
- ☐ Investigation
- ☐ Hate men just because (?)

5) How have the feminism impacted the society?

- ☐ Being above men rights
- ☐ Education access for woman, emancipation, voting and denaturin gender roles
- ☐ Exclude LGTBIQ people

Enviar

And that's how the test looks. When we go to "send" button, we obtain:



your score: 3

...

Now, the static folder looks like this:

```
^C(venv) (base) [mhl@mulata mulatasweb]$ tree homepage/static/
homepage/static/
├── homepage
│   ├── allw.jpg
│   ├── fem.css
│   ├── feminis.jpg
│   └── oth.jpg
```

And the css looks like this:


```
(venv) (base) [mhl@mulata mulatasweb]$ cat homepage/static/homepage/fem.css
body {
    background-color: #9E48F4;
}

h1 {
    color: Black;
    text-align: center;
    font-family: "Times New Roman", Times, serif;
    font-size: 40px;
}

p{
    font-size: 20px;
}

a{
    text-align: center;
    color: Black;
}
```

And thats all!