**How to use Django**

**1.1) Installation of Django**

Django requires python 3, which can be downloaded from <https://www.python.org/downloads/> .

To ensure that the environment you are working in does not clash with other programs, it is a good practice to use a virtual environment. Note:Text which begins with a ‘>>‘ indicate using an interpreter like command prompt (cmd).

To install a virtual environment,

>>pip install virtualenv #pip is a tool used for installation

>>mkdir myproject #mkdir is making a directory or a folder (code 1)

>>cd myproject #cd is to change directory, or to enter a folder.

>>virtualenv venv #this is to create a virtual environment called venv

>>venv\Scripts\activate #this activates the virtual environment

Then, to install Django in that virtual environment,

>>pip install Django

**1.2) How Django works**

The following section involves the fundamentals of Django, and how Django works. At its core, when Django works surmised by the following. When a user visits a website, they would enter a url. This url(urls.py) would direct to a particular set of content that we would like to show to our users, which in Django is known as a view(views.py). Then, there are other files which help segment and aid in the process above.

To start a project,

>>django-admin startproject myproject #(code 2)

What the directory looks like.

myproject/ <-- higher level folder #from code 1

|-- myproject/ <-- django project folder #from code 2

| |-- myproject/

| | |-- \_\_init\_\_.py

| | |-- settings.py

| | |-- urls.py

| | |-- wsgi.py

| +-- manage.py

+-- venv/ <-- virtual environment folder

Important files to know

settings.py : file contains the project’s configurations i.e. settings

urls.py : file contains the urls and paths

manage.py : called when you want to execute commands that uses Django-admin

To run your code,

>>py manage.py runserver

You should see a “Congratulations!” page with a rocket taking off.

Under a project, you would have many different apps,

>>django-admin startapp boards

myproject/

|-- myproject/

| |-- boards/ <-- our new django app!

| | |-- migrations/

| | | +-- \_\_init\_\_.py

| | |-- \_\_init\_\_.py

| | |-- admin.py

| | |-- apps.py

| | |-- models.py

| | |-- tests.py

| | +-- views.py

| |-- myproject/

| | |-- \_\_init\_\_.py

| | |-- settings.py

| | |-- urls.py

| | |-- wsgi.py

| +-- manage.py

+-- venv/

Important files to know

migrations/: store files to track changes made to models.py file. Used to sync database and models

admin.py : built-in to Django for admin features

models.py: database tables

tests.py : for tests

views.py : handle request/response

In the settings.py file and under installed apps, add in the name of your app. (In this case its ‘Boards’)

As mentioned, to make changes to the webpage you would have to edit views.py

First, define a function in your views.py file

from django.http import HttpResponse

**def** **home**(request):

**return** HttpResponse('Hello, World!')

Then, to call the function, you would need to send a request. This is done via urls.py.

from django.conf.urls import url

from django.contrib import admin

from boards import views

urlpatterns **=** [

url('', views**.**home, name**=**'home'),

url('admin/', admin**.**site**.**urls),

]

The url is structured as such

‘admin/’ is the url appended

admin.site.urls is the location of the file you are calling the function from

name=”admin” is the name you assign to the url.

**1.3) Implementing a login feature**

To implement a login feature, we would need to think about how our website will look like. In this case, we will we introducing a taskbar at the top of our webpage, where users would be able to return to home by clicking a button, as well as manage their log in settings.



To begin, we would need to create a templates folder to store our html templates.

myproject/

|-- myproject/

| |-- boards/

| |-- myproject/

| |-- templates/ <-- here!

| +-- manage.py

+-- venv/

To let Django know where the html files are located, we would have to set a directory inside the settings.py file.

import os

'DIRS': [

os**.**path**.**join(BASE\_DIR, 'templates'),

]

We would also be using other resources like CSS to compose the user interface, hence we would need to install them aswell.

First, create a folder named static, with another folder inside it called css.

myproject/

|-- myproject/

| |-- boards/

| |-- myproject/

| |-- templates/

| |-- static/ <-- here

| | +-- css/ <-- and here

| +-- manage.py

+-- venv/

Go to [www.getbootstrap.com](http://www.getbootstrap.com) and download the latest compiled CSS and JS version. Extract the zip file and copy the file to the css folder

myproject/

|-- myproject/

| |-- boards/

| |-- myproject/

| |-- templates/

| |-- static/

| | +-- css/

| | +-- bootstrap.min.css <-- here

| +-- manage.py

+-- venv/

Again, to instruct Django on where to look, add the following to settings.py

STATIC\_URL **=** '/static/'

STATICFILES\_DIRS **=** [

os**.**path**.**join(BASE\_DIR, 'static'),

]

Now, to use bootstrap, all we have to do would be to use {% load static %} in our html files.

As the taskbar would be often present in the webpage, we would not want to always write the same code for every page. This is why we have a base file, which can be called upon from other html files. To use this file in every page, we would simply need to just use **{%** **extends** 'base.html' **%}**

First, create a file name base.html under the templates folder

**{%** load static **%}<!DOCTYPE html>**

<html>

<head>

<meta charset="utf-8">

<title>**{%** **block** title **%}**NEO**{%** **endblock** **%}**</title>

<link rel="stylesheet" href="**{%** static 'css/bootstrap.min.css' **%}**">

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container">

<a class="navbar-brand" href="**{%** url 'home' **%}**">NEO</a>

</div>

</nav>

<div class="container">

<ol class="breadcrumb my-4">

**{%** **block** breadcrumb **%}**

**{%** **endblock** **%}**

</ol>

**{%** **block** content **%}**

**{%** **endblock** **%}**

</div>

</body>

</html>

To use this navbar, we would need to again follow the structure of Django, that of urls-views-templates.

Since we have already set the url to direct to views.home before, we should change the home function in views.py

def home(request):

return(request,’home.html’)

Having changed the view, we would need to create a home.html file under templates.

{% extends ‘base.html’%}

{% load static %}

{% block breadcrumb %}

{% endblock %}

Next, to implement the login feature, we would need to create a separate app. In the same page as manage.py, run

>>django-admin startapp accounts

myproject/

|-- myproject/

| |-- accounts/ <-- our new django app!

| |-- boards/

| |-- myproject/

| |-- static/

| |-- templates/

| |-- db.sqlite3

| +-- manage.py

+-- venv/

You should then add the new app ‘accounts’ to settings.py, under installed apps.

**Signup**

To begin, lets create the sign up view.

Go the urls.py, and add an url for the signup page

From accounts import views as accounts\_views

Urlpatterns =[

url…

url(‘signup/’, accounts\_views.signup, name=’signup’),

]

Then, we head over to views.py under the accounts app, and create a new function named signup:

from django.shortcuts import render

**def** **signup**(request):

**return** render(request, 'signup.html')

Create a new template, named signup.html

**{% extends** 'base.html' **%}**

**{%** **block** content **%}**

<h2>Sign up</h2>

**{%** **endblock** **%}**

Open the URL http;//127.0.0.1:8000/signup/. You should get a text with Sign up, as well as a taskbar at the top.

For authentication views, we would not be using the task bar at the top, so we would be making some changes to base.html

{% load static **%}<!DOCTYPE html>**

<html>

<head>

<meta charset="utf-8">

<title>**{%** **block** title **%}**Django Boards**{%** **endblock** **%}**</title>

<link href="https://fonts.googleapis.com/css?family=Peralta" rel="stylesheet">

<link rel="stylesheet" href="**{%** static 'css/bootstrap.min.css' **%}**">

<link rel="stylesheet" href="**{%** static 'css/app.css' **%}**">

**{%** **block** stylesheet **%}{%** **endblock** **%}** *<!-- HERE -->*

</head>

<body>

**{%** **block** body **%}** *<!-- HERE -->*

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container">

<a class="navbar-brand" href="**{%** url 'home' **%}**">Django Boards</a>

</div>

</nav>

<div class="container">

<ol class="breadcrumb my-4">

**{%** **block** breadcrumb **%}**

**{%** **endblock** **%}**

</ol>

**{%** **block** content **%}**

**{%** **endblock** **%}**

</div>

**{%** **endblock** body **%}** *<!-- AND HERE -->*

</body>

</html>

Changes were made to the naming of the blocks. By doing this, we are able to use the stylesheet block in base.html, without having to use the navbar.

Make the following changes to your signup.html file

**{% extends** 'base.html' **%}**

**{%** **block** body **%}**

<h2>Sign up</h2>

**{%** **endblock** **%}**

The body block in signup.html will supersede the body block containing the navbar, hence removing the navbar.

To create the sign up form, we can use the built in form in Django called UserCreationForm. In your views.py, import it, and change the signup function.

From django.contrib.auth import login **as** auth\_login

from django.contrib.auth.forms import UserCreationForm

from django.shortcuts import render, redirect

**def** **signup**(request):

**if** request**.**method **==** 'POST':

form **=** UserCreationForm(request**.**POST)

**if** form**.**is\_valid():

user **=** form**.**save()

auth\_login(request, user)

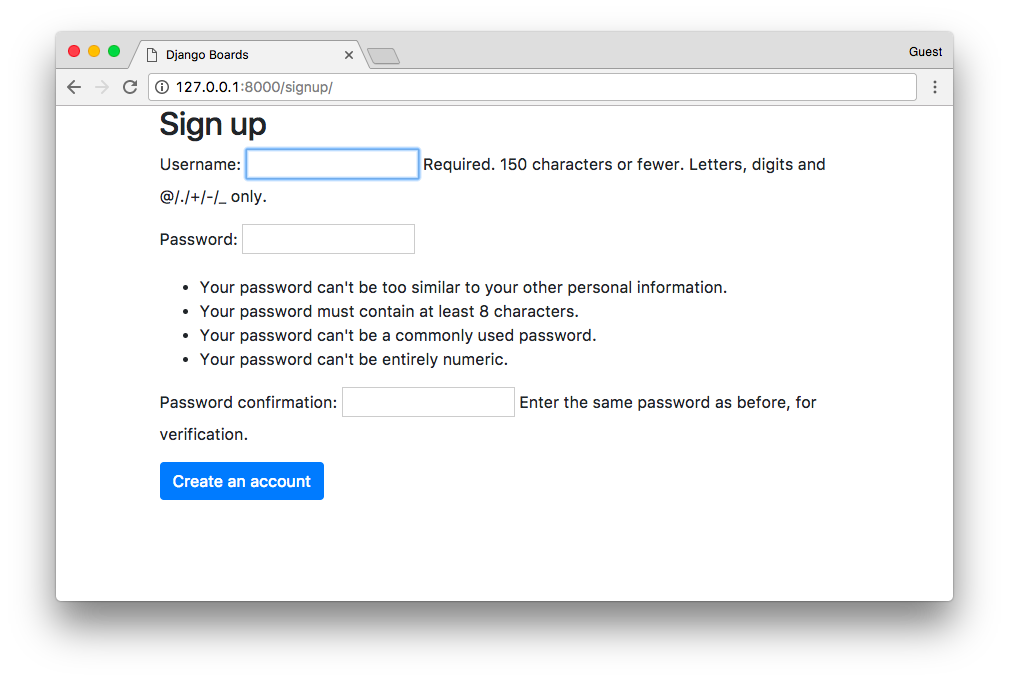
**return** redirect('home')

**else**:

form **=** UserCreationForm()

**return** render(request, 'signup.html', {'form': form})

You should get the following when you go to your signup page.

When you create an account, you would be logged in. To make it reflect that we are logged in, we can make changes to base.html.

{% **block** body **%}**

<nav class="navbar navbar-expand-sm navbar-dark bg-dark">

<div class="container">

<a class="navbar-brand" href="**{%** url 'home' **%}**">Django Boards</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#mainMenu" aria-controls="mainMenu" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="mainMenu">

<ul class="navbar-nav ml-auto">

<li class="nav-item">

<a class="nav-link" href="#">**{{** user.username **}}**</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container">

<ol class="breadcrumb my-4">

**{%** **block** breadcrumb **%}**

**{%** **endblock** **%}**

</ol>

**{%** **block** content **%}**

**{%** **endblock** **%}**

</div>

**{%** **endblock** body **%}**

After the changes are made, your username should appear on the right of the navbar.

**Logout**

Now, with a sign up feature, the natural progression is a logout feature.

The process is similar to that mentioned above. First you set new url for the view.

From django.contrib.auth import views as auth\_views

url(‘logout/’, auth\_views.LogoutView.as\_view(), name=’logout’)

We want to be directed to the homepage when we log out. Hence, we go to settings.py

LOGOUT\_REDIRECT\_URL = ‘home’

To have dropdown menu to logout, we would need the following.

Download the compressed production jQuery at [www.jquery.com/download/](http://www.jquery.com/download/)

Inside the static folder, create a new folder named js. Copy the downloaded file here.

We would also need popper. Download popper at [www.popper.js.org](http://www.popper.js.org)

Copy popper.min.js to our js folder.

Use the same bootstrap 4 file downloaded before and copy it to the js folder as well.

The final result should be as shown:

myproject/

|-- myproject/

| |-- accounts/

| |-- boards/

| |-- myproject/

| |-- static/

| | |-- css/

| | +-- js/

| | |-- bootstrap.min.js

| | |-- jquery-3.2.1.min.js

| | +-- popper.min.js

| |-- templates/

| |-- db.sqlite3

| +-- manage.py

+-- venv/

At the bottom of the base.html file, add the scripts in.

**{%** **endblock** body **%}**

<script src="**{%** static 'js/jquery-3.2.1.min.js' **%}**"></script>

<script src="**{%** static 'js/popper.min.js' **%}**"></script>

<script src="**{%** static 'js/bootstrap.min.js' **%}**"></script>

</body>

Now, to add the dropdown menu, change base.html to the following.

<nav class="navbar navbar-expand-sm navbar-dark bg-dark">

<div class="container">

<a class="navbar-brand" href="**{%** url 'home' **%}**">Django Boards</a>

<button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#mainMenu" aria-controls="mainMenu" aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="mainMenu">

**{%** **if** user.is\_authenticated **%}**

<ul class="navbar-nav ml-auto">

<li class="nav-item dropdown">

<a class="nav-link dropdown-toggle" href="#" id="userMenu" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">

**{{** user.username **}}**

</a>

<div class="dropdown-menu dropdown-menu-right" aria-labelledby="userMenu">

<div class="dropdown-divider"></div>

<a class="dropdown-item" href="**{%** url 'logout' **%}**">Log out</a>

</div>

</li>

</ul>

**{%** **else** **%}**

<form class="form-inline ml-auto">

<a href="#" class="btn btn-outline-secondary">Log in</a>

<a href="**{%** url 'signup' **%}**" class="btn btn-primary ml-2">Sign up</a>

</form>

**{%** **endif** **%}**

</div>

</div>

</nav>

**Login**

Add a new url route

url(‘login/’,auth\_views.LoginView.as\_view(template\_name =’login.html’), name=”login”)

This allows us to skip having to use a view a directly refer to login.html.

Edit the settings.py file

LOGIN\_REDIRECT\_URL = ‘home’

Add the login url to base.html

ref="**{%** url 'login' **%}**" class="btn btn-outline-secondary">Log in</a>

Create a login webpage name login.html

{% extends 'base.html' %}

{% load static %}

{% block stylesheet %}

<link rel="stylesheet" href="{% static 'css/accounts.css' %}">

{% endblock %}

{% block body %}

<div class="container">

<h1 class="text-center logo my-4">

<a href="{% url 'home' %}">NEO</a>

</h1>

<div class="row justify-content-center">

<div class="col-lg-4 col-md-6 col-sm-8">

<div class="card">

<div class="card-body">

<h3 class="card-title">Log in</h3>

<form method="post" novalidate>

{% csrf\_token %}

{{ form.as\_p }}

<button type="submit" class="btn btn-primary btn-block">Log in</button>

</form>

</div>

<div class="card-footer text-muted text-center">

New to Django Boards? <a href="{% url 'signup' %}">Sign up</a>

</div>

</div>

</div>

</div>

</div>

{% endblock %}

We have complete the accounts feature!

Refer to the full guide here: <https://simpleisbetterthancomplex.com/series/>