قسمت اول: (ایجاد فولدر، نصب جنگو، ایجاد پروژه، انتقال به گیت هاپ)

1. ایجاد فولدر پروژه و ایجاد محیط اینوایرومنت:

برای ایجاد یک پروژه جدید جنگو بهتر است درون فولدری که میخواهیم پروژه را در آن ایجاد کنیم، یک ویژوال اینوایرومنت یا محیط مجازی بسازیم و پایتون خود را به همراه تمام کتابخانه ها و ... در این محیط ویژوال قرار دهیم . این کار کمک میکند تا وابستگی های پروژه به صورت جدا از پروژه های دیگر ما نصب شود و اگر قصد انتقال پروژه یا تحویل پروژه به دولوپر های دیگر را داشتیم وابستگی های پروژه های دیگر ما به همراه این پروژه انتقال داده نشود و همچنین باعث میشود حجم کتابخانه های لوکال ما بیخود افزایش نیابد. برای انجام این کار در پوشه پروژه با کمک cmd کد زیر را وارد میکنیم.

python -m venv env

بعد از ساخت پوشه env که محتویات محیط مجازی پایتون ما است باید این محیط را برای پروژه خود فعال کنیم تا بتوانیم شروع به ساخت پروژه در این محیط کنیم. برای این منظور با کد زیر وارد پوشه env و از درون آن وارد پوشه scripts میشویم.

cd env\scripts

سپس با کد زیر اقدام به اکتیو کردن محیط مجازی میکنیم

Activate

پس از فعال کردن محیط اینوایرومت (env) قبل از خط کامنت ظاهر میشود که نشان فعال شدن محیط مجازی است. (برای غیر فعال کردن محیط مجازی از دستور deactivate استفاده میکنیم)

1. نصب جنگو روی پایتون در محیط مجازی :

برای نصب فرم ورک جنگو ابتدا از فولدر scripts به فولدر اول پروژه برمیگردیم:

cd..\..\ دو فولدر به عقب برمیگردیم

سپس دستور زیر را برای نصب جنگو اجرا میکنیم:

python -m pip install Django

1. ایجاد پروژه :

حال که فرم ورک جنگو را در محیط مجازی پایتون خود نصب کردیم نوبت به ایجاد پروژه جنگو میرسد. برای ایجاد پروژه ای با نام دلخواه مثلا Tutorial کد زیر را وارد میکنیم:

django-admin startproject Tutorial

با این فرمان فایل های مربوط به پروژه جنگو در پوشه tutorial داخل فولدر پروژه ایجاد میشود.

حال برای اجرای پروژه و دیدن صفحه اول پیشفرض سایت باید از cmd با فرمان زیر وارد پوشه tutorial شویم :

Cd tutorial

بعد با فرمان زیر پروژه را روی سرور لوکال با پورت 8000 لود میکنیم:

manage.py runserver

حال اگر در مرورگر سیستم خود آدرس <http://127.0.01:8000> یا <http://localhost:8000> را وارد کنم میتوانیم صفحه پیشفرض اولیه سایت را ببینیم.

تبریک ☺

**قبل از شروع مراحل بعدی اگر قصد داریم از گیت یا گیت هاب استفاده کنیم و پروژه را پله به پله و با هر بار توسعه در گیت هاب ذخیره کنیم، میتوانیم چند گام کوتاه ولی فوق العاده مفید زیر را برداریم.**

**ابتدا به سایت گیت هاب رفته و یک ریپازیتوری جدید میسازیم تا پروژه را به آن انتقال دهیم.**

**بعد در cmd کدهای زیر را به ترتیب میزنیم:**

**Git init به داخل فایل اصلی پروژه رفته و این دستور را آنجا اجرا میکنیم**

**این دستور یک فایل با نام .git به صورت مخفی در فولدر پروژه ایجاد میکند**

**Git add . با این دستور همه فولدر های پروژه را به گیت هاب اضافه میکنیم**

**Git commit -m "your comment" با این دستور یک توضیح برای پرژه مینویسید**

**Git branch -m master انتخاب مسیر مستر برای برنچ پروژه**

**Git remote add origin YOUR\_REPOSITORY\_ADDRESS  
دادن آدرس ریپازیتوری ساخته شده**

**Git push -u origin master ذخیره فایل ها در مخزن گیت هاب**

قسمت دوم: ( views و urls )

بعد از ایجاد پروژه اولین اَپ توسط خود جنگو ساخته خواهد شد ( در مثال فوق به نام Tutorial) و این اَپ به عنوان اَپ اصلی و مادر در پروژه جنگو تازه ساخته شده، در نظر گرفته خواهد شد.

در اپ ساخته شده توسط جنگو ( به زودی ما خودمان اپ های متعددی برای ایجاد سایت خواهیم ساخت اما اپ اصلی ما اپی است که جنگو ساخته است ) چند فایل ساخته شده که برای مدیریت سایت ما لازم است. یکی از این فایل ها urls.py است که تمامی درخواست ها از url سایت مستقیم به این فایل ارسال شده و در این فایل مشخص میشود که با این درخواست باید چه کرد یا به عبارتی این فایل مشخص میکند که یک url خاص از سایت (مثلا [www.mysite.com/about](http://www.mysite.com/about) ) اگر فراخوانی شد کدام فانکشن، در کدام آدرس باید اجرا شود.

برای درک بهتر این موضوع در فولدر اپ اصلی (tutorial ) یک فایل به نام views.py ایجاد میکنیم و کد های زیر را درون آن می نویسیم:

from django.http import HttpResponse

def about(request):

return HttpResponse(" About me ")

حال به فایل urls.py رفته و ابتدا در بالای صفحه فایل views.py را ایمپورت می کنیم:

from . import views

حال در قسمت urlpatterns در زیر مسیری که برای admin به صورت پیشفرض وجود دارد یک مسیر یا path جدید میسازیم و آدرس فانکشن about در صفحه views را به آن میدهیم.

Urlpatterns = [

Path('admin/' , admin.site.urls) ,

Path('about/' , views.about ) ,

]

حال اگر در قسمت آدرس بار مرورگر خود بعد از آدرس سرور about را تایپ کنیم و اینتر بزنیم عبارت About me که در فانکشن about در فایل views.py ریترن شده بود به نمایش در خواهد آمد. در واقع دستور (request) ما مستقیم به فایل urls.py از اپ اصلی سایت رفته و در آنجا به صفحه views.py و فانکشن about هدایت میشود.

قسمت سوم: (templates)

تا اینجا توانستیم درخواست ارسال شده را به فایل urls.py هدایت کرده و از طریق این فایل آن را به سمت فانکشنی که در فایل views.py برای این درخواست در نظر گرفته ایم هدایت کردیم و در خروجی مقدار return شده از طریق فانکشن را مشاهده کردیم. اما معمولا در پروژه های واقعی ما نیاز به نمایش یک متن و یا پیغام نداریم و میخواهیم یک صفحه html کامل را فراخوانی کنیم. از این رو یک فولدر به اسم templates در فولدر اصلی سایت ( فولدری که فایل manage.py قرار دارد ) می سازیم و تمام فایل های html که قرار است فراخوانی شود را در این فولدر قرار میدهیم. برای نمونه فایل contact.html را ساخته و کد ساده زیر را برای تست در آن قرار میدهیم:

<h1> Contact us </h1>

<p>phone: 9120000</p>

حال برای فراخوانی این صفحه سه گام را باید برداریم :

در گام اول : به صفحه urls.py در اپ اصلی میرویم و یک آدرس جدید به urlpatterns اضافه میکنیم. به شکل زیر:

path('contact/', views.contact),

در گام دوم وارد فولدر اپ اصلی و مادر (در این مثال اپ tutorial ) شده و در پوشه settings.py به سمت پایین اسکرول کرده و مقدار دیکشنری TEMPLATES را پیدا میکنیم و در میان پراپرتی های آن مقدار DIRS یا دایرکتوری را به صورت استرینگ برابر 'templates' قرار میدهیم.

در گام سوم به فولدر views.py رفته، در بالای صفحه کد زیر را مینویسیم:

from django.shortcuts import render

بعد در همین فایل یک فانکشن جدید به شکل زیر مینویسیم:

def contact(request):

return render(request, 'contact.html')

دقت داشته باشید که تابع render دو مقدار میگیرد که مقدار اول همیشه همان request است و مقدار دوم آدرس فایلی که باید فراخوانی شود، ولی در قسمت دوم همانطور که ملاحظه میکنید آدرس فایل کامل داده نشده و فقط نام فایل را نوشته ایم علت این امر این است که ما قبلا در فایل settings.py در فولدر اصلی اپ مقدار DIRS را مساوی با templates قرار داده ایم که باعث میشود جنگو در این فولدر به دنبال صفحه html های ما بگردد. حال به مرورگر رفته و آدرس زیر را تایپ میکنیم و اینتر را میزنیم:

http://localhost:8000/contact/

اگر تمام مراحل را درست انجام داده باشیم و سرور در cmd ران باشد باید محتویات فایل contact.html را مشاهده کنیم. ☺

قسمت چهارم: (Make APPS)

یکی از اصلی ترین و مهم ترین قابلیت هایی که فِرِموُرک هایی مثل جنگو و ری اکت و لاراول و ... برای ما به ارمغان می آورند این است که کل سایت را با تمام پیچیدگی هایش به بخش های کوچک و ساده ای تبدیل میکنند که با هم در ارتباط هستند و هر بخش در هر جایی که به آن نیاز باشد فراخوانی میشود. در اغلب موارد به هر کدام از این بخش های کوچک یک APP گفته میشود.

در جنگو هم با ایجاد پروژه اولین APP ( اَپ اصلی پروژه ) به طور خودکار توسط جنگو ساخته میشود که در مثال ما اَپ tutorial بود. اما برای استفاده از این قابلیت بزرگ جنگو باید بتوانیم به تعداد دلخواه APP بسازیم و هر بخش از سایت را در یک APP قرار دهیم.

برای ایجاد APP جدید در cmd با فشردن کلیدهای Ctrl+c از حالت اجرای سرور خارج می شویم و درحالی که در پوشه اصلی جنگو یعنی پوشه ای که manage.py وجود دارد هستیم، کد زیر را برای ساخت یک APP جدید به نام footer مینویسیم:

python manage.py startapp footer

پس از اجرای کد بالا فولدری با نام footer در پروژه ساخته میشود. اما برای اینکه جنگو این فولدر را به عنوان APP بپذیرد باید در اَپ اصلی وارد فایل settings.py بشویم و در قسمت INSTALLED\_APPS زیر گزینه های موجود نام اَپ ساخته شده را (footer) وارد میکنیم.

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'footer',

]

اگر داخل فولدر APP جدید را مشاهده کنیم می بینیم که اکثر فایل های موجود در APP اصلی پروژه در این APP هم وجود دارد. حتی فایل views.py که ما در اپ اصلی به شکل دستی ساختیم در این اَپ به شکل پیشفرض وجود دارد . تنها فایلی که وجود ندارد فایل urls.py است که ما باید این فایل را در APP های جدید که میسازیم به شکل دستی بسازیم.

پس قبل از هر کاری فایلی با نام urls.py در اَپ جدید میسازیم.

حال برای راحتی کار تمام کد های نوشته شده در فایل urls.py واقع در APP اصلی را در این فایل هم کپی میکنیم و فقط آدرس های داخل urlpatterns را پاک میکنیم تا آدرس های جدید را جایگزین آن کنیم.

حال طبق روال گذشته اول از همه به سراغ اپ اصلی و فایل urls.py میرویم تا آدرس یا path جدید را وارد کنیم. ولی از آنجا که قرار است درخواست را به APP جدید بفرستیم و در آنجا آدرس دهی کرده و تعیین تکلیف کنیم ابتدا در صفحه urls.py اپ اصلی جایی که path را از django.urls ایمپُرت کرده ایم include را هم ایمپرت میکنیم:

from django.urls import path, include

و در ادامه path جدید را به شکل زیر میسازیم:

path('footer/', include('footer.urls')),

با این کد درخواست را برای تعیین تکلیف به اَپ footer و فایل urls.py داخل آن می فرستیم. حال در داخل فایل urls.py در اَپ جدید ( footer ) باید آدرس فانکشنی را که پس از این درخواست باید اجرا شود مینویسیم.

from django.contrib import admin

from django.urls import path

from . import views

urlpatterns = [

path(' ', views.footer),

]

باید توجه داشت که قسمت اول path بالا خالی است چون مقدار footer در اَپ اصلی و در فایل urls.py آدرس دهی شده است و در اَپ جدید تنها در صورتی که بخواهیم برای درخواست ارسالی با مقداری اضافه تر از footer مثلا [www.mysite/footer/new](http://www.mysite/footer/new) آدرس دهی کنیم مقدار اولیه path را به آن اختصاص میدهیم، در غیر صورت مثل مثال بالا خالی میماند.

مقدار دوم در path بالا درخواست را برای تعیین تکلیف به فایل views.py در اَپ footer می فرستد و فانکشن footer را فراخوانی میکند.

حال به پوشه views.py رفته و این فانکشن را میسازیم:

from django.shortcuts import render

def footer(request):

return render(request,'footer.html')

در فانکشن بالا فایل footer.html فراخوانی شده است که جنگو با توجه به تنظیمات ما آن را از فایل templates میخواند پس به این فولدر رفته و فایل مورد نظر را میسازیم و کدهای تست زیر را در آن می نویسیم:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Footer</title>

</head>

<body>

<h1>Footer</h1>

<p>this is Footer of site</p>

</body>

</html>

سپس آدرس زیر را در مرورگر فراخوانی میکنیم:

<http://localhost:8000/footer/>

که باید صفحه با نام footer را لود کند.

همچنین میتوانستیم برای هر APP یک فولدر templates جداگانه بسازیم و داخل آن یک پوشه هم نام APP ساخته و صفحه های html مخصوص این APP را درون آن بریزیم. ولی ما برای سادگی کار فعلا از انجام آن صرفه نظر کرده ایم.

قسمت پنچم: ( اتصال و ارتباط با دیتابیس )

اولین نکته ای که باید در مورد دیتابیس در جنگو بدانیم این است که وقتی جنگو را نصب و یک پروژه ایجاد میکنیم، به شکل پیشفرض دیتابیس sqlite3 (فایل db.sqlite3 در اپ اصلی پروژه) برای پروژه در نظر گرفته میشود که اصلا دیتابیس قوی و کاملی نیست. ولی خبر خوب این است که ما میتوانیم از هر دیتابیس دیگری برای استفاده در پروژه استفاده کنیم، فقط باید تنظیمات لازم را در فایل settings.py انجام دهیم. برای این منظور ما از دیتابیس محبوب همیشگیمان یعنی mysql استفاده خواهیم کرد. برای راحتی کار از سرورهای مجازی مانند xampp و یا wampserver استفاده میکنیم و یک دیتابیس با نام mydatabase در آن میسازیم و سرور مجازی را فعال نگاه میداریم.

در گام اول باید mysql را در محیط اینوایرومت (env) پروژه نصب کنیم پس در cmd سیستم بعد از ورود به محیط اینوایرومنت دستور زیر را مینویسیم:

pip install mysqlclient

حال به تنظیمات فایل settings.py میپردازیم:

در پوشه اپ اصلی پروژه به فایل settings.py رفته و با اسکرول به سمت پایین به قسمت DATABASES میرسیم و محتویات آن را با کد های زیر جایگزین میکنیم.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': ' mydatabase',

'USER': 'root',

'PASSWORD': '',

'HOST': '127.0.0.1',

'PORT': '3306',

}

}

موارد name و user و password را به دلخواه میتوانید تغییر دهید.

حال باید دستور زیر را برای پیکر بندی جدید جنگو استفاده کنیم:

python manage.py makemigrations

و بعد از آن دستور میگریت را :

python manage.py migrate

"اگر هنگام انجام عملیات migrations یا migrate با اروری مبنی بر نصب نبودن psycopg2 برخورد کردید آن را با کد pip install psycopg2 نصب کنید"

پس از انجام موفقیت آمیز migrate پروژه، در محیط سرور مجازی خودمان شاهد اضافه شدن چندین جدول میباشیم که توسط خود جنگو به شکل پیش فرض طراحی شده و بعد از عملیات migrate در دیتابیس ساخته شده است.

برای مطمئن شدن از کارکرد پروژه بهتر است یک کاربر به شکل ادمین (superuser) ایجاد کنیم و در آدرس <http://localhost:8000/admin> لاگین کنیم و جداول از پیش تعریف شده جنگو در دیتابیس را هم چک کنیم. ( به زودی خودمان جداولی را ایجاد و اطلاعاتی را روی آن ذخیره میکنیم و در جاهای مختلف پروژه آن اطلاعات را از دیتابیس فراخوانی میکنیم)

برای ایجاد یک سوپر یوزر دستور زیر را در cmd تایپ میکنیم:

python manage.py createsuperuser

بعد از اجرای این دستور جنگو از ما میخواهد یک نام کاربری برای یوزر خود انتخاب کنیم که ما در این پروژه admin را انتخاب میکنیم و بعد از از ما ایمیل میخواهد که ما فعلا خالی میگذاریم و اینتر را میزنیم و بعد از آن دو بار از ما پسورد را میخواهد که باید هر دو بار یک پسورد یکسان را وارد نماییم. ما در این پروژه پسورد را هم admin میزنیم. حال جنگو به ما اخطار میدهد که پسورد شما خیلی مطمئن نیست و آیا میخواهید ادامه دهید که ما با تایپ حرف Y و بعد اینتر کردن تایید میکنیم و به همین راحتی یک سوپر یوزر با نام کاربری و پسورد admin ساختیم.

حال اگر به آدرس <http://localhost:8000/admin> برویم و با یوزرنیم و پسورد admin لاگین کنیم وارد پنل ادمین سایت خود میشویم و جدول کاربرها (Users) و گروه ها (Groups) را که توسط خود جنگو در دیتابیس ما ایجاد شده بود مشاهده میکنیم. با این پنل بعد از این بسیار کار خواهیم داشت.

قسمت ششم: (Models)

بسیار خوب ، حالا که با موفقیت توانستیم یک پایگاه داده را به پروژه خود اضافه کنیم وقت آن رسیده است که به ساخت جداول دلخواه و مناسب برای پروژه خودمان بپردازیم. مدل ها (Models) در جنگو وظیفه ایجاد جداول را در دیتابیس ما به عهده دارند. در واقع ما یک مدل را به صورت یک کلاس در فایل models.py هر یک از APP های پروژه مینویسیم و بعد از آن با انجام دستورهای makemigrations و migrate جداول جدید در پایگاه داده ساخته شده و یا مدل های اصلاح شده در پایگاه داده اصلاح میشوند. بیایید در اَپ footer که قبلا ایجاد کرده ایم یک مدل بسازیم. به اپ footer رفته و در فایل models.py آن کلاس (مدل) Article را میسازیم:

from django.db import models

class Article(models.Model):

title = models.CharField(max\_length=100)

body = models.TextField()

date = models.DateTimeField(auto\_now\_add = True)

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

return self.title

به وسیله کلاس بالا (یا مدل بالا) ما خواهیم توانست یک جدول به نام Article با چهار ستون که یکی را خود جنگو به شکل اتوماتیک با نام id میسازد و سه تای دیگر همان title و body و date میباشند بسازیم. در مورد دو خط پایانی کد و چیزی که کلاس return میکند جلوتر توضیح میدهیم.

برخی دیگر از فیلد های پر کاربرد برای ساخت ستون ها موارد زیر هستند:

models.IntegerField() models.BooleanField() models.EmailField()

models.UrlField()

اما همانطور که قبلا هم مشاهده کرده اید ساخت یک مدل به معنی ساخته شدن جدول در دیتابیس نیست بلکه بعد از ساخت یا تغییر در هر مدل باید اول سرور مجازی خودمان مثلا wampserver را run کنیم و بعد به ترتیب دو دستور زیر را اجرا کنیم:

python manage.py makemigrations

این دستور کد های مخصوص ساخت جدول در دیتابیس را در فولدر migrations و در فایلی با نامی مشابه 0001\_initial.py ذخیره میکند.

Python manage.py migrate

این دستور از طریق فایل ایجاد شده در فولدر migrations اقدام به ساخت جداول در دیتابیس میکند.

حال اگر به آدرس <http://localhost/phpmyadmin> رفته و با همان یوزرنیم و پسوردی

که هنگام ساخت دیتابیس به آن داده ایم لاگین کنیم و روی نام دیتابیس خودمان (در مثال ما mydatabase ) کلیک کنیم میبینیم که جدولی به نام footer\_article با همان ستون های نوشته شده در مدل ما ایجاد شدهف که فعلا خالی است و مقداری در آن ثبت نشده است.

حال بیایید برای مشاهده بهتر تغییرات در جدول footer\_article ، محتوای این جدول را به صفحه admin اضافه کنیم. برای این کار در همان فولدر اَپ footer به فایل admin رفته و بعد از ایمپرت کردن مدل خود آن را به صفحه ادمین اضافه میکنیم:

from django.contrib import admin

from .models import Article

admin.site.register(Article)

بعد از ذخیره کد بالا به صفحه ادمین به آدرس <http://localhost:8000/admin/> رفته و با نام سوپر یوزیر و پسورد آن (که در مثال ما نام کاربری و پسورد هردو admin بود) لاگین میکنیم مشاهده میکنید که قسمتی با عنوان FOOTER اضافه شده است که در آن نام جدول Article وجود دارد. همچنین دو گزینه Add و Change نیز وجود دارد که میتوان با این گزینه ها به این جدول دیتا اضافه کرد و یا دیتای موجود را تغییراتی داد برای نمونه روی Add کلیک کرده و در فیلد Title متن First Title را مینویسیم و در فیلد Body متن دلخواهی را می نویسیم.

حال اگر دوباره صفحه ادمین را رفرش کنیم و روی Articles بزنیم خواهیم دید که در لیست آن یک عنوان جدید ایجاد شده است که چیزی نیست جز مقداری که در فیلد Title وارد کرده ایم (First Title) و این به فانکشنی که در دو خط آخر کلاس (مدل) مربوطه نوشتیم برمیگردیم.

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

return self.title

همچنین اضافه شدن مقادیر بالا را در آدرس <http://localhost/phpmyadmin> در جدول دیتابیس هم میتوانید مشاهده کنید.

قسمت هفتم: (Tag های پایتون در صفحات HTML جنگو)

ما توانستیم داده های داخل دیتابیس را در پنل ادمین سایت ببینیم ولی این فقط برای سوپر یوزر یا ادمین سایت مفید است و کاربران دیگر به آن دسترسی ندارند، لذا نوبت به آن میرسد که داده ها را در صفحات عمومی سایت هم از دیتابیس فراخوانی کنیم.

این کار منطقا در دو مرحله انجام میشود.

مرحله اول: ارسال داده های مورد نیاز هر صفحه به آن می باشد

مرحله دوم: دریافت داده ها در صفحه مورد نظر و در جایی که برای آن در نظر گرفته شده است

برای انجام مرحله اول (یعنی ارسال داده به صفحه ای خاص) باید به سراغ views.py و فانکشنی که وظیفه لود این صفحه را دارد برویم و بعد از ایمپرت کردن مدل مربوط به اطلاعاتی که می خواهیم به این صفحه اسال کنیم باید در قسمت return فانکشن مربوطه دیتای مورد نظر را داخل یک دیکشنری رندر کنیم. فرض کنید در مثال خودمان میخواهیم داده های جدول footer\_article که ساخته بودیم را در صفحه footer نمایش دهیم. به صفحه views.py در اَپ footer رفته و کدهای زیر را وارد میکنیم:

from django.shortcuts import render

from .models import Article

def footer(request):

articles = Article.objects.all().order\_by('date')

return render(request,'footer.html', { 'articles' : articles } )

\*مقدار Article.objects.all() برابر کل مقادیر جدول footer\_article میباشد.

ارسال داده ها به صفحه footer.html تمام شد، حال باید به این صفحه رفته و اطلاعات را دریافت و در جای خودش به نمایش بگذاریم.

برای این منظور در جنگو دو نوع Tag وجود دارد یکی {% %} که برای نوشتن کد های پایتون در صفحه html استفاده میشود و دیگری {{ }} که برای فراخوانی داده ای که به صفحه ارسال شده است استفاده میشود.

در صفحه footer.html تگهای داخل تگ body را به شکل زیر تغییر میدهیم:

<h1>Footer</h1>

<p>this is Footer of site</p>

{% for article in articles %}

<div style="background: #ddd;margin: 5px;padding: 10px;">

<p style="color: blue;margin: 5px;">{{article.title}}</p><hr>

<p style="margin: 5px;">{{article.body}}</p><hr>

<p style="margin: 5px;">{{article.date}}</p>

</div>

{% endfor %}

در خط سوم از کد بالا ما یک دستور for پایتون (حلقه for ) را شروع و در خط آخر آن را به پایان رسانده ایم و در بدنه حلقه داده های موجود در articles ، که به صفحه ارسال کرده بودیم را فراخوانی کرده ایم. اگر بخواهیم در قسمت body فقط چند چرف اول متن به نمایش در بیاید (مثلا 50 کاراکتر اول ) باید فانکشن زیر را نیز به فایل models.py اضافه کنیم:

def snippet(self):

return self.body[:50]

حال در هنگام فراخوانی در صفحه footer.html به جای article.body باید بنویسیم article.snippet

قسمت هشتم : ( static )

همانطور که در مثال قبل و تنظیمات صفحه footer مشاهده کردید، ما مقداری استایل به صفحه دادیم مثل بک گراند و مارجین و پدینگ و ... ، اما این معمولا راه درست و تمیزی برای استایل دهی به صفحات html نیست همینطور اگر ما کدهای جاوا اسکریپتی برای صفحه html خودمان داشته باشیم میتوانیم آنها در تگ script در همان صفحه بنویسیم ولی این هم راه درستی نیست.

در واقع دروش صحیح اضافه کردن استایل ها و کد های جاوا اسکریپتی و از این دست ایجاد فایل های استاتیک جداگانه و مستقل و ایمپرت کردن آن در صفحه مورد نظر است. با این روش میتوانیم یک استایل خاص را که یک بار نوشته ایم، به تگ های مختلف در صفحات مختلف اعمال کنیم و علاوه بر خوانایی بالاتر کدها، از کد نویسی کمتر و تمیز تری برخوردار خواهیم بود.

ما ایجاد فولدری برای فایل های استاتیک و استفاده از آن را در سه گام انجام میدهیم:

* به پوشه اصلی پروژه ( که فایل manage.py در آن قرار دارد ) رفته و فولدری به نام static میسازیم و در این فولدر فایلی به نام style.css برای استایل دهی به یک صفحه خاص و در فایل style.css برای تست کد های زیر را میزنیم:

p{color: blue;}

با این کار تگ های p در صفحه به رنگ آبی در خواهد آمد.

* حال تنظیمات فایل settings.py در اَپ اصلی پروژه را به ترتیب زیر انجام میدهیم :

STATIC\_URL = "static/" معمولا به شکل پیش فرض همین است

STATICFILES\_DIRS = [BASE\_DIR / 'static'] این خط را باید دستی اضافه کنیم

STATICFILES\_DIRS = [os.path.join(BASE\_DIR, 'static')] اگر کار نکرد این

* حال تنها کافی است به صفحه ای که میخواهیم این استایل استفاده شود ( مثلا footer.html ) رفته و در بالای صفحه کد زیر را وارید کنیم:

{% load static %}

بعد داخل header صفحه لینک زیر را قرار میدهیم:

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

حال اگر به آدرس <http://localhost:8000/footer> برویم میبینیم که متن های پاراگراف آبی شده اند .

قسمت نهم: ( extends )

شاید یکی از مهم ترین دلایلی که ما را برای استفاده از فرم ورک هایی مثل جنگو مشتاق میکند این است که این فرم ورک ها سایت را به قطعات کوچکتری تقسیم میکنند که قابلیت فراخوانی در صفحات دیگر را دارند و به این شکل امکان اصلاح، رفع ایراد و بهبود قسمت های مختلف سایت را به شکل مستقل فراهم میکند و همچنین امکان اجرا و توسعه یک سایت به شکل گروهی را بسیار ساده تر و اصولی تر میکند.

برای کد نویسی در یک صفحه مستقل و انتقال آن کد ها به صفحه دیگری از سایت در جنگو از extends استفاده میکنیم به این صورت که ابتدا یک صفحه جدید میسازیم و در بالای آن فایلی را که میخواهیم کد های این صفحه به آن منتقل بشوند را extends میکنیم و بعد با استفاده از block ها جای نمایش این کد ها در صفحه مورد نظر را مشخص میکنیم.

بیایید یک فایل base.html برای مثال خودمان ایجاد کنیم که به منزله اسکلت یا قالب سایت باشد و فقط حاوی مطالبی که در همه صفحات سایت نمایش داده میشوند ( مثل header در بالای صفحه یا contact us در پایین صفحه ) باشد و بدنه اصلی صفحه بسته به اینکه چه صفحه ای را فراخوانی کرده ایم بارگذاری شود. پس در پوشه templates یک فایل با نام base.html میسازیم و کدهای زیر را در آن مینویسیم:

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

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

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

</head>

<body>

<div class="header" style="width: 100%;top:0px;padding:10px; display: flex;justify-content: center;align-items: center;background: #A154BD;">

<a style="margin:0 40px" href="">home</a>

<a style="margin:0 40px" href="">page 1</a>

<a style="margin:0 40px" href="">page 2</a>

</div>

<div class="body" style="width: 100%;top: 0px;padding:10px; display: flex;justify-content: center;align-items: center;flex-direction: column;">

{% block content %}

{% endblock content %}

</div>

<div class="footer" style="width: 100%;bottom: 0px;padding:10px; display: flex;justify-content: center;align-items: center;position: relative; background:#A154BD;">

<p>contact us: [mr.smvtn@gmail.com</p](mailto:mr.smvtn@gmail.com%3c/p)>

</div>

</body>

</html>

استایل های داده شده فقط برای نمایش همه تغییر ها در یک صفحه هستند و در پروژه های واقعی این استایل ها در همان فایل static به نام style.css که در جلسات قبل ساختیم نوشته میشوند.

در بین بلوک های آبی در کد بالا ( block title و block contant ) کد های صفحه footer.html و صفحات دیگر را میتوانیم نمایش دهیم .

برای نمونه به فایل footer.html رفته و کدها را به شکل زیر تغییر میدهیم:

{% extends 'base.html' %}

{% block title %} footer {% endblock title %}

{% block content %}

<h1>Footer</h1>

<p>this is Footer of site</p>

<div>

{% for article in articles %}

<div style="background: #ddd;margin: 5px;padding: 10px;width:50vw">

<p style="color: blue;margin: 5px;font-weight: bold;">{{article.title}}</p><hr>

<p style="margin: 5px;">{{article.body}}</p><hr>

<p style="margin: 5px;">{{article.date}}</p>

</div>

{% endfor %}

</div>

{% endblock content %}

حال اگر آدرس <http://localhost:8000/footer> را در مرورگر فراخوانی کنیم مشاهده خواهید کرد قسمت بالایی و پایینی سایت از فایل base.html و قسمت وسط از فایل footer.html لود شده است. به همین صورت میشود تمام صفحات سایت را در همین فایل base.html لود کرد.

در قسمت بعدی یک صفحه جدید میسازیم تا روی هر کدام از آرتیکل های موجود در صفحه footer که کلیک شد در همان قالب base.html جزئیات آن آرتیکل را به همراه متن کامل آن نمایش دهد.

قسمت دهم: ( نمایش جزئیات article ها در قالب base.html )

میخواهیم وقتی کاربر روی هر کدام از article ها (مقاله ها یا پست های نمایش داده شده از جدول footer\_article در آدرس <http://localhost:8000/footer> ) که کلیک کرد به آدرس جدیدی هدایت شود و در آن آدرس جدید، همان article یا پست را با جزئیات کامل (مثلا متن کامل پست که قبلا به صورت خلاصه نمایش داده میشد) مشاهده کند.

گام اول:

بیایید اول آدرس جدید را به جنگو معرفی کنیم. چون آدرس هایی که بعد از نام سایت footer را دارند از طریق فایل urls.py در اَپ اصلی به اَپ footer و فایل urls.py آن هدایت میشوند، پس آدرس جدید را در urls.py اَپ footer میسازیم و یک خط به آن اضافه میکنیم:

urlpatterns = [

path('', views.footer, name='footer'),

path('<int:id>/', views.footer\_detile, name='article\_detile'),

]

آدرس دوم آدرس جدید ماست که در آن دو چیز جدید مشاهده میکنیم، اول (<int:id>) که نشان میدهد که در آخر آدرس url ما یک داده به صورت اینتیجر یا عددی وجود دارد که این داده id همان پستی است که قرار است در این صفحه نمایش داده شود و جلوتر به نحوه ارسال این داده میپردازیم. دومین چیز متفاوت در این آدرس (name='article\_detile') است که یک نام است که به این آدرس اختصاص داده شده تا در هر کجای این اَپ که به این آدرس احتیاج داشتیم فقط نام آن را صدا بزنیم (جلوتر نحوه صدا زدن آن را میبینیم)

گام دوم:

در گام اول ادرس را به فانکشن footer\_detile در فایل views ارجاع دادیم پس بیاید در گام دوم این فانکشن را بسازیم. به فایل views.py رفته و فانکشن زیر را به آن اضافه میکنیم:

def footer\_detile(request, id):

articles = Article.objects.get(id = id)

return render(request, 'article\_detile.html', {'articles':articles})

(متد Article برای فانکشن قبلی ایمپرت شده پس نیازی به ایمپرت مجدد نیست)

در خط دوم فانکشن بالا با فانکشن Article.objects.get(id=id) متفاوت از فانکشن قبلی که همه اطلاعات را از طریق متد Article میگرفت، ما فقط اطلاعات مربوط به ردیفی که id آن برابر عدد داخل url (آدرس) است میگیریم. ( مثلا اطلاعات مربوط با پست اول با آی دی یک را که شامل تیتر و متن و تاریخ پست اول است) و بعد در خط آخر فانکشن این اطلاعات را در صفحه article\_detile.html لود کرده و return میکنیم(نمایش میدهیم).

گام سوم:

چون قرار است روی هر کدام از پست های نمایش داده شده در صفحه footr.html که کلیک شود وارد صفحه جدید برای نمایش جزئیات شویم، پس باید تمام این پست ها را به داخل یک تگ a (لینک) ببریم.

بنابراین صفحه footer.html را به شکل زیر تغییر میدهیم:

{% extends 'base.html' %}

{% block title %} footer {% endblock title %}

{% block content %}

<h1>Footer</h1>

<p>this is Footer of site</p>

<div>

{% for article in articles %}

<div style="background: #ddd;margin: 5px;padding: 10px;width:50vw">

<a href="{% url 'article\_detile' article.id %}">

<p style="color: blue;margin: 5px;font-weight: bold;">{{article.title}}</p><hr>

<p style="margin: 5px;">{{article.snippet}}</p><hr>

<p style="margin: 5px;">{{article.date}}</p>

</a>

</div>

{% endfor %}

</div>

{% endblock content %}

با دقت در آدرسی که ما به href تگ a پست ها دادیم (url 'article\_detile' article.id) می فهمیم که پس از url نام آدرسی که در گام اول ساختیم را آورده و پس از آن id هر پست را در جلوی آدرس href تگِ a آن پست قرار داده ایم بنابر این مثلا اگر روی پست اول که آی دی آن 1 است کلیک کنیم به آدرس <http://localhost:8000/footer/1/> هدایت میشویم.

گام چهارم :

درنهایت باید صفحه article\_detile.html را بسازیم. برای این کار از همان فایل base.html استفاده میکنیم تا قسمت بالا و پایین سایت به شکل بقیه صفحات باشد و فقط متن وسط صفحه تغییر کند، به شکل زیر:

{% extends 'base.html' %}

{% block title %} Article\_detile {% endblock title %}

{% block content %}

<h1>Article\_detile</h1>

<hr>

<div>

<div style="background: #ddd;margin: 5px;padding: 10px;width:50vw">

<p style="color: blue;margin: 5px;font-weight: bold;">{{articles.title}}</p><hr>

<p style="margin: 5px;">{{articles.body}}</p><hr>

<p style="margin: 5px;">{{articles.date}}</p>

</div>

</div>

{% endblock content %}

به این نکته توجه داشته باشید که در کد بالا مثل کد صفحه footer.html از for و حلقه اسفاده نکردیم چون فقط اطلاعات یک ردیف از جدول دیتابیس را داریم پس اینجا مستقیم از articles اطلاعات را میگیریم نه اینکه مثل صفحه footer.html هر مقدار از articles را داخل یک article بریزیم و بعد اطلاعات از article بگیریم. و اینکه در این صفحه اطلاعات body یا همان متن پست را از articles.body میگیریم و نه از article.snippet که فقط چند کاراکتر اول متن را نمایش میداد.

قسمت یازدهم ( delete ) :

برای حذف یک آرتیکل یا پست ابتدا لینک delete را با href زیر برای هر لینک اضافه میکنیم:

**article\_detile.html:**

{% extends 'base.html' %}

{% block title %} Article\_detile {% endblock title %}

{% block content %}

<h1>Article\_detile</h1>

<hr>

<div>

<div style="background: #ddd;margin: 5px;padding: 10px;width:50vw">

<p style="color: blue;margin: 5px;font-weight: bold;">{{articles.title}}</p><hr>

<p style="margin: 5px;">{{articles.body}}</p><hr>

<p style="margin: 5px;">{{articles.date}}</p>

<a href="{% url 'delete\_post' articles.id %}">Delete</a>

</div>

</div>

{% endblock content %}

حال مسیر مربوط به delete را در urls.py ایجاد میکنیم:

**urls.py:**

urlpatterns = [

….

path('<int:id>/delete/', views.delete\_post\_view, name='delete\_post'),

]

و در آخر فانکشن مربوطه را در views.py میسازیم:

views.py:

def delete\_post\_view(request,id):

post = Article.objects.get(id=id)

post.delete();

return redirect('footer'

حال با کلیک کردن روی delete آرتیکل مربوطه حذف میشود.

قسمت دوازدهم: ( فرم ها و ارسال داده به دیتابیس )

قبلا در ارتباط با ساخت دیتابیس و فراخوانی داده ها از آن صحبت کردیم، اما اگر بخواهیم اطلاعاتی را به دیتابیس ارسال کنیم چه؟ وظیفه ارسال داده ها به دیتابیس را فرم ها به عهده دارند. بیایید در مثال خودمان به جای ارسال داده ها به جدول footer\_article از پنل ادمین در بالای همان صفحه footer.html که پست ها را از دیتابیس فراخوانی میکردیم و نمایش میدادیم فرم ایجاد کنیم تا از همانجا و نه از صفحه ادمین بتوانیم به جدول footer\_article اطلاعاتی اضافه کنیم و در همان صفحه هم نمایش دهیم.

ساخت فرم در جنگو به روش های مختلفی ممکن است ولی یکی از ساده ترین و در عین حال امن ترین روش ها این است که ساخت آن را به خود جنگو بسپاریم! در واقع ما یک فایل به نام forms.py در اپ مورد نظر خودمان میسازیم و در آن پس از ایمپرت کردن forms از django یک کلاس تعریف میکنیم و مشخصات فرمی را که میخواهیم ( در واقع مشخصات فیلد های فرم را ) در این کلاس تعریف میکنیم:

from django import forms

class Send\_Article(forms.Form):

Title = forms.CharField(label='Title', max\_length=100)

Body = forms.CharField(widget=forms.Textarea)

خوب در همینجا کار ما با صفحه forms.py تمام شد. درواقع ما به جنگو گفته ایم یک مدل فرم را بسازد و حالا در هر صفحه ای که بخواهیم از طریق views.py و فانکشن همان صفحه فرم را به صفحه مورد نظر (در این مثال footer.html ) ارسال میکنیم. پس حالا باید به فانکشن footer در صفحه views.py برویم و آن را به شکل زیر تغییر دهیم تا علاوه بر نمایش پست های جدول footer\_articl که تا به حال انجام میداد فرم ما را هم چک کند تا اگر اطلاعاتی از طریق آن ارسال کرده بودیم، به این جدول در دیتابیس اضافه کند:

from django.shortcuts import render

from .models import Article

from .forms import Send\_Article

def footer(request):

if request.method == "POST":

form = Send\_Article(request.POST)

if form.is\_valid():

title = form.cleaned\_data['Title']

body = form.cleaned\_data['Body']

article = Article.objects.create(title = title, body = body)

article.save()

form = Send\_Article()

else:

form = Send\_Article()

articles = Article.objects.all().order\_by('date')

return render(request,'footer.html', {'articles':articles, 'form':form})

در خط دوم کد، چک میکنیم آیا اطلاعاتی از طریق متد POST برای صفحه ارسال شده است یا نه. اگر ارسال شده بود، خط سوم تا هفتم کد اجرا شده و اطلاعات را به دیتابیس می فرستد ( در خط هفتم اطلاعات فرم دوباره خالی میشود تا دوباره بشود از آن استفاده کرد ) و بعد دو خط پایانی اجرا میشود که مانند گذشته پست های جدول را از دیتابیس میخواند و همه را به صفحه footer.html میفرستد تا به نمایش در بیایند با این تفاوت که حالا مقدار form را هم به صفحه ارسال مکنیم. برای ارسال داده به دیتابیس نیز از همان مدلی که قبلا در فایل models.py برای ایجاد جدول footer\_article در دیتابیس تعریف کردیم، یعنی کلاس Article استفاده میکنیم.

حال فقط یک کار مانده است آن هم اینکه فرم ارسال شده به صفحه footer.html را، در این صفحه نمایش دهیم :

{% extends 'base.html' %}

{% block title %} footer {% endblock title %}

{% block content %}

<h1>Footer</h1>

<p>this is Footer of site</p>

<form action="" method="POST">

{% csrf\_token %}

{{form.as\_p}}

<input type="submit" value="Submit">

</form>

<div>

{% for article in articles %}

<div style="background: #ddd;margin: 5px;padding: 10px;width:50vw">

<a href="{% url 'article\_detile' article.id %}">

<p style="color: blue;margin: 5px;font-weight: bold;">{{article.title}}</p><hr>

<p style="margin: 5px;">{{article.snippet}}</p><hr>

<p style="margin: 5px;">{{article.date}}</p>

</a>

</div>

{% endfor %}

</div>

{% endblock content %}

کد {% csrf\_token %} برای ارسال توکن هش شده همراه فرم است که به امنیت ارسال داده کمک فراوانی میکند و در جنگو استفاده از آن الزامی است.

{{form.as\_p}} هم تمام فیلد های پستی که ساختیم را به شکل پاراگراف های زیر هم نمایش میدهد. ( نگران شکل ظاهری فرم نباشید زیرا ظاهر فرم را در فایل css که در فولدر استاتیک ساختیم میتوانیم تغییر دهیم. )

قسمت سیزدهم: (فرم ثبت نام )

حال که به طور کلی با نحوه ایجاد و استفاده از فرم ها در جنگو آشنا شدیم، زمان آن فرارسیده است که دو فرم اصلی و پرکاربردی، که در بیشتر سایتها استفاده میشود را با هم بسازیم. فرم ثبت نام و فرم لاگین.

اول فرم ثبت نام را با حالت پیش فرض جنگو یعنی فرمی دارای 3 فیلد ( username و password1 و password2 ) می سازیم، بعد آن را به حالت دلخواه گسترش داده و فیلد ایمیل را به آن اضافه میکنیم.

برای انجام امور مربوط به ثبت نام و لاگین کاربر ها یک APP دیگر به نام accounts میسازیم:

manage.py startapp accounts

پس از اضافه کردن نام اپ در فایل settings.py و اضافه کردن آدرس در فایل urls.py اپ اصلی پروژه، در اپ accounts یک فایل به اسم urls.py برای آدرس دهی می سازیم و path را در آن ایمپرت میکنیم همچنین آدرس registery را برای آن میسازیم:

from django.urls import path

from . import views

urlpatterns = [

path('registery/', views.registery, name='registery'),

]

حال فانکشن رجیستری را در فولدر views.py همین اپ میسازیم

from django.shortcuts import render, redirect

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth import authenticate, logout, login

from django.contrib import messages

def registery(request):

if request.method=="POST":

form = UserCreationForm(request.POST)

if form.is\_valid():

form.save()

username = form.cleaned\_data['username']

password = form.cleaned\_data['password1']

user=authenticate(username=username,password=password)

login(request, user)

messages.success(request, ('Registration Successful!'))

return redirect(' footer ')

else:

form = UserCreationForm()

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

همانطور که ملاحظه میکنید کد فانکشن registery نتیجه را به صفحه register.html ارسال میکند که باید آن را در فولدر تمپلت ها بسازیم.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Register</title>

</head>

<body style="display: flex;justify-content:center;align-items: center;flex-direction: column;">

<div style="width: 60%;margin: 20px;display: flex;justify-content:center;align-items: center;background: #708C90;flex-direction: column;border-radius: 10px;">

<h1>Register Form </h1>

{% if form.errors %}

<p style='color:red'>there was an erore with your form ..! <br> {{ form.errors }}</p>

{% endif %}

<form action="" method="POST" enctype="multipart/form-data" style="width: 50%;margin: 20px;display: flex;justify-content:center;align-items: center;flex-direction: column;">

{% csrf\_token %}

{{form.as\_p}}

<Button type="submit">Register</Button>

</form>

</div>

</body>

</html>

حال با رفتن به صفحه <http://localhost:8000/accounts/registery> فرم ثبت نام را ملاحظه کرده و میتوانیم در سایت ثبت نام کنیم.

حال بیایید به فیلد های پیشفرض جنگو سه فیلد نام و نام خانوادگی و ایمیل را هم اضافه کنیم.

برای این منظور ابتدا در اَپ accounts یک فایل با نام forms.py برای ایجاد فیلد های دلخواه ایجاد میکنیم و پس از ایمپرت کردن UserCreationForm و User و forms کلاسی برای شخصی سازی فرم ثبت نام میسازیم تا از این کلاس در صفحه views.py به جای UserCreationForm استفاده کنیم:

فایل forms.py :

from django import forms

from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth.models import User

class RegisterUserForm(UserCreationForm):

email = forms.EmailField()

first\_name = forms.CharField(max\_length=70)

last\_name = forms.CharField(max\_length=70)

class Meta:

model = User

fields = ('first\_name', 'last\_name','username', 'email', 'password1', 'password2' )

در کلاس Meta داخل این کلاس تمام فیلد هایی را که میخواهیم در فرم باشند را می آوریم.

در واقع اینبار کلاس UserCreationForm اول در کلاس ساخته شده در forms.py را خوانی میشود و بعد این کلاس در فایل views.py فراخوانی میشود:

فایل views.py :

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, logout, login

from django.contrib import messages

from .forms import RegisterUserForm

def registery(request):

if request.method=="POST":

form = RegisterUserForm(request.POST)

if form.is\_valid():

form.save()

username = form.cleaned\_data['username']

password = form.cleaned\_data['password1']

user = authenticate(username=username, password=password)

login(request, user)

messages.success(request, ('Registration Successful!'))

return redirect('footer')

else:

form = RegisterUserForm()

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

حال اگر مجدداَ به آدرس <http://localhost:8000/accounts/registery> برویم فیلد های دلخواه اضافه شده است و میتوانیم با آنها ثبت نام کنیم.

قسمت چهاردهم : (بهینه سازی فرم ثبت نام )

الف) چک کردن ایمیل غیر تکراری:

در دو سمت قبل ما ابتدا یک فرم ثبت نام ده شکل پیشفرض جنگو ساختیم و بعد چند فیلد دلخواه را به آن اضافه کردیم، فرم ثبت نام ما به خوبی میتواند کاربر را ثبت نام کرده و اطلاعات آن را در دیتابیس ذخیره نماید، اما یک مشکل وجود دارد و آن هم اینکه در فرم پیشفرض جنگو هنگام ثبت نام چک میشد که کاربر با نام کاربری تکراری ثبت نام نکند و هر کاربر یک نام کاربری واحد داشته باشد. اگر ما بخواهیم کاربرانمان دارای ایمیل واحدی نیز باشند، چه؟

برای این منظور کافیست قبل از ذخیره فرم در دیتابیس ایمیل را به شکل زیر چک کنیم:

**Views.py:**

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, logout, login

from django.contrib import messages

from .forms import RegisterUserForm

from django.contrib.auth.models import User

def registery(request):

if request.method=="POST":

form = RegisterUserForm(request.POST)

if form.is\_valid():

email = form.cleaned\_data['email']

user\_email\_count = User.objects.filter(email=email).count()

if user\_email\_count == 0:

form.save()

username = form.cleaned\_data['username']

password = form.cleaned\_data['password1']

user = authenticate(username=username, password=password)

login(request, user)

messages.success(request, ('Registration Successful!'))

return redirect('footer')

else:

messages.success(request, ('A user with this email already exists'))

return redirect('registery')

else:

form = RegisterUserForm()

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

ب) ایجاد فرم ثبت نام با امکان آپلود عکس:

پیش نیازها:

python -m pip install Pillow نصب پیلو برای کار با عکس ها

Import os در settings.py ایمپرت اُ اِس در فایل ستینگ برای مسیردهی

اضافه کردن کد زیر در صفحه settings.py برای ایجاد مسیری برای آپلود عکس ها :

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

MEDIA\_URL = '/media/'

اضافه کردن کد زیر در صفحه urls.py اپ اصلی پروژه (بعد از ایمپرت settings و static به صفحه urls.py)

if settings.DEBUG:

urlpatterns += static(settings.MEDIA\_URL,document\_root=settings.MEDIA\_ROOT)

برای ایجاد فرم ثبت نام با امکان آپلود عکس پروفایل بعد از انجام پیش نیازهای بالا، سه فایل models.py و forms.py و views.py را باز نویسی میکنیم:

در این فرم ثبت نام درواقع ما اطلاعات کاربر (User ) و پروفایل ( در ایم مثال عکس ) را با هم در یک فرم ارسال میکنیم و به جداول خودشان در دیتابیس اضافه میکنیم. درواقع اطلاعات کاربر از طریق یک فرم ساده مثل گذشته در پوشه forms.py ساخته میشود ولی اطلاعات پروفایل از طریق یک مدل که در models.py میسازیم به فرم اضافه میشود و در پایان به فایل views.py ارسال شده و رندر میشود.

با فایل models.py شروع میکنیم:

from django.db import models

from django.contrib.auth.models import User

class Profile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

image = models.ImageField(upload\_to='profile\_pics', blank=True)

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

return self.user.username +' --> '+ self.user.email

در خط اول داخل کلاس پروفایل ما مقدار user را مساوی با مدل یک به یک User قرار داردیم، به این معنی که به ازای هر کاربر فقط یک پروفایل و به ازای هر پروفایل فقط برای یک کاربر میباشد.

دو خط آخر را هم برای نمایش در پنل ادمین اضافه کردیم که برای این منظور باید کد زیر را در فایل admin.py وارد کنیم:

from .models import Profile

admin.site.register(Profile)

حال به سراغ کدهای فایل forms.py میرویم:

from django import forms

from django.contrib.auth.forms import UserCreationForm

from .models import Profile

from django.contrib.auth.models import User

class RegistrationForm(UserCreationForm):

email = forms.EmailField()

image = forms.ImageField(required=False)

class Meta:

model = User

fields = ['first\_name','last\_name','username', 'email', 'password1', 'password2', 'image']

def save(self, commit=True):

user = super(RegistrationForm, self).save(commit=False)

user.email = self.cleaned\_data['email']

if commit:

user.save()

profile = Profile(user=user, image=self.cleaned\_data.get('image'))

profile.save()

return user

در کد بالا ابتدا فیلدهای email و image را که در فرم پیشفرض جنگو نیست به آن اضافه میکنیم و بعد در کلاس Meta تمام فیلد های فرم را به ترتیب نمایش می نویسیم و در فانکشن save اول نوع ذخیره سازی پیش فرض یوزر را false کرده و بعد نوع ذخیره سازی کاربر و پروفایل در جای خودشان را میسازیم، تا پس از فراخوانی و اجرا در views.py اطلاعات همانگونه که میخواهیم ذخیره شود.

Views.py

from django.shortcuts import render, redirect

from .forms import RegistrationForm

def register(request):

if request.method == 'POST':

form = RegistrationForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

return redirect('home')

else:

form = RegistrationForm()

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

با اضافه کردن 4 خط کد میتوان پس از ثبت نام کاربر را لاگین کرده و صفحه home هدایت کنیم:

Views.py

from django.shortcuts import render, redirect

from .forms import RegistrationForm

from django.contrib.auth import authenticate, login

def register(request):

if request.method == 'POST':

form = RegistrationForm(request.POST, request.FILES)

if form.is\_valid():

form.save()

username = form.cleaned\_data['username']

password = form.cleaned\_data['password1']

user = authenticate(username=username, password=password)

login(request, user)

return redirect('home')

else:

form = RegistrationForm()

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

قسمت پانزدهم: ( فرم login و Logout )

فرم لاگین در سایت های مختلف به دو شکل است. یا از نام کاربری و پسورد برای لاگین استفاده میشود یا از ایمیل و پسورد. چون به شکل پیش فرض فرم لاگین در جنگو از نام کاربری و پسورد استفاده میکند ما نیز ابتدا فرم را به همین شکل ساخته و بعد از آن نحوه حذف نام کاربری از فرم ثبت نام و لاگین کردن با ایمیل و پسورد را آموزش می بینیم.

فرم لاگین ( با نام کاربری و پسورد ) :

ابتدا در فایل forms.py کلاس زیر را برای ایجاد فرم می نویسیم

class LoginForm(forms.Form):

email = forms.EmailField()

password = forms.CharField(widget=forms.PasswordInput)

سپس در فایل views.py فانکشن login\_view را مینویسیم:

from .forms import LoginForm

def login\_view(request):

if request.method == 'POST':

form = LoginForm(request.POST)

if form.is\_valid():

username = form.cleaned\_data['username']

password = form.cleaned\_data['password']

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

return redirect('footer')

else:

form.add\_error(None, 'Invalid email or password')

else:

form = LoginForm()

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

فقط میماند اضافه کردن آدرس صفحه login به urls.py :

path('login/', views.login\_view, name='login'),

حال باید فرم را در یک صفحه مانند صفحه رجیستر ولی با نام login.html در templates وارد کنیم.

فرم لاگین ( با ایمیل و پسورد ):

برای تنظیم فرم لاگین به نحوی که از ایمیل و پسورد برای لاگین کردن استفاده کنیم کافیست ابتدا فیلد username را از فرم ثبت نام حذف کنیم و کاری کنیم به تور اتوماتیک مقدار آن برابر ایمیل کاربر بشود. برای انجام این کار فانکشن RegistrationForm را در فایل forms.py به شکل زیر تغییر میدهیم:

class RegistrationForm(UserCreationForm):

email = forms.EmailField()

image = forms.ImageField(required=False)

class Meta:

model = User

fields = ['first\_name','last\_name','email', 'password1', 'password2', 'image']

def save(self, commit=True):

user = super(RegistrationForm, self).save(commit=False)

user.email = self.cleaned\_data['email']بدون این خط هم کار میکند ولی برای اطمینان از تمیز بودن ایمیل مینویسیم

user.username = self.cleaned\_data['email']

if commit:

user.save()

profile = Profile(user=user, image=self.cleaned\_data.get('image'))

profile.save()

return user

حال تغییرات فانکشن login\_view در فایل views.py را به شکل زیر انجام میدهیم:

def login\_view(request):

if request.method == 'POST':

form = LoginForm(request.POST)

if form.is\_valid():

email = form.cleaned\_data['email']

password = form.cleaned\_data['password']

user = authenticate(request, username=email, password=password)

# user = authenticate(request, username=email, password=password)

if user is not None:

login(request, user)

return redirect('footer')

else:

form.add\_error(None, 'Invalid email or password')

else:

form = LoginForm()

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

Logout

برای logout کردن کار بسیار ساده تر است و تنها کافی است یک آدرس جدید در urls.py بسازیم و فانکشن مربوط به آن در views.py را به شکل زیر بنویسیم:

urls.py:

urlpatterns = [

….

path('logout/', views.logout\_view,name='logout'),

]

views.py:

def logout\_view(request):

logout(request)

return redirect('footer')

قسمت شانزدهم: ( change password )

اینبار برای ایجاد فرم تغییر پسورد در جنگو از کلاس های آماده و پیشفرض موجود در جنگو استفاده میکنیم تا بسیار ساده و کوتاه و مطمئن کار را به انجام برسانیم.

برای این منظور اول در فولدر templates یک صفحه به نام change\_password.html میسازیم و محتویاتی مثل آنچه در فرم لاگین بود را در این فرم مینویسیم تا فرم را به آن ارسال کنیم. بعد نوبت به ساخت فرم در فایل forms.py میرسد. برای این منظور بعد ازاینپورت کردن PasswordChangeForm کلاس زیر را میسازیم:

from django.contrib.auth.forms import PasswordChangeForm

class ChangePassword(PasswordChangeForm):

old\_password = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-control'}))

new\_password1 = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-control'}))

new\_password2 = forms.CharField(widget=forms.PasswordInput(attrs={'class': 'form-control'}))

حال برای استفاده از کلاس های آماده جنگو به فایل views.py رفته و کلاس زیر را مینویسیم:

from .forms import ChangePassword

from django.contrib.auth.views import PasswordChangeView

from django.urls import reverse\_lazy

class passwordChangeView(PasswordChangeView):

form\_class = ChangePassword

template\_name = 'change\_password.html'

success\_url = reverse\_lazy('change\_pass\_success') در اینجا آدرسی برای ریدایرکت کردن کاربر بعد از عوض کردن پسورد وارد میکنیم

تنها کاری که مانده است این است که در urls.py اپ اصلی آدرسی که فانکشن بالا را اجرا کند بنویسیم:

urlpatterns = [

…

path('change\_pass/', views.passwordChangeView.as\_view(), name='change\_pass'),

]

توجه داشته باشید چون اینبار به جای فانکشن، کلاس پیشفرض جنگو را فراخوانی میکنیم باید در انتهای آن در as\_view() را بیاوریم.

فراموشی رمز عبور:

به علت اینکه در شرایط فراموشی رمز عبور کاربر در ایمیل خود لینک تغییر رمز را دریافت میکند، و تنظیمات ارسال ایمیل از روی هاست با لوکال هاست متفاوت است، لذا بعد از آپلود پروژه بر روی هاست به ایجاد سیستم فراموشی رمز عبور خواهیم پرداخت.

قسمت هفدهم (Serializers کردن اطلاعات و API ها در جنگو )

در بسیاری از مواقع نیاز میشود اطلاعاتی را از دیتابیس خودمان به اپلیکیشن ها و سایت های متفاوت در دیوایس های مختلفی ارسال کنیم و تا از این اطلاعات در آنجا استفاده شود. این امر توسط API ها انجام میشود. مثل api های قیمت لحظه ای ارز و ... . برای ساخت api در جنگو باید از REST framework استفاده کرد. برای نصب آن از دستور زیر استفاده میکنیم:

pip install djangorestframework

pip install markdown # Markdown support for the browsable API.

pip install django-filter # Filtering support

و بعد آن را به APP های پروژه جنگو اضافه میکنیم:

INSTALLED\_APPS = [

……

'rest\_framework',

]

Serializers کردن اطلاعات :

به اَپِ footer رفته و یک فولدر به اسم api میسازیم و داخل آن به علت اینکه یک پوشه داخل یک اپ است ابتدا یک فایل با نام \_\_init\_\_.py میسازیم.با این کار تمام فایلهای دیگر پروژه با فایل های داخل پوشه api میتوانند در ارتباط باشند.

حال در همین فولدر api یک فایل دیگر به نام serializers.py میسازیم و کد های زیر را در آن مینویسیم:

from rest\_framework import serializers

from footer.models import Article

class ArticleSerializer(serializers.ModelSerializer):

class Meta:

model = Article

fields=['title', 'body', 'date']

کار سریالایز کردن مدل تا اینجا به پایان رسید.

گرفتن اطلاعات (GET):

برای ساخت api باید در همان پوشه api یک فایل views.py مخصوص به api ها را ایجاد کنیم و همچنین یک فایل urls.py برای ایجاد آدرس های مخصوص api میسازیم. درواقع فولدر api مانند یک app در داخل یک app دیگر می باشد.

فایل views.py را به شکل زیر میسازیم:

from rest\_framework import status

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from footer.models import Article

from footer.api.serializers import ArticleSerializer

@api\_view(['GET'])

def articleList(request):

if request.method == "GET":

query = Article.objects.all()

serializer = ArticleSerializer(query)

return Response(serializer.data)

حال باید آدرس های مربوط به api را در urls.py اپ اصلی یا اپ api داخل footer بسازیم:

urls.py (app اصلی):

from footer.api.views import articleList

urlpatterns = [

……

path('footer/api/', articleList, name='footer\_API'),

]

urls.py (footer.app) :

from django.urls import path, include

from footer.api.views import articleList

urlpatterns = [

path('', articleList, name='footer\_API'),

]

حال اگر با نرم افزار postman با متد GET یک درخواست برای آدرس <http://localhost:8000/footer/api> ارسال کنیم، لیست تمام پست ها را به صورت JSON خواهیم دید.

ویرایش اطلاعات (PUT):

برای جایگزینی اطلاعات به جای اطلاعات قبلی (ویرایش اطلاعات) از متد PUT استفاده میکنیم.

footer.api.views.py :

@api\_view(['PUT'])

def updateAPI(request, pk):

try:

query = Article.objects.get(pk=pk)

except Article.DoesNotExist:

return Response(status = status.HTTP\_404\_NOT\_FOUND)

mass = {}

if request.method == "PUT":

query = Article.objects.get(pk=pk)

serializer = ArticleSerializer(query, data=request.data)

if serializer.is\_valid():

serializer.save()

mass['mass1'] = 'update is down'

return Response(data=mass)

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

ساخت آدرس urls این بار در فایل urls.py در داخل فولدر (app) api :

from django.urls import path

from footer.api.views import updateAPI

urlpatterns = [

path('<int:pk>', updateAPI, name='footer\_update\_API'),

]

تنظیم آدرس بالا در اپ اصلی:

urlpatterns = [

……..

path('footer/api/update/',include('footer.api.urls'), name='update\_article'),

]

حال در نرم افزار پست من آدرس <http://localhost:8000/footer/api/update/1> را در حالی که متد PUT انتخاب شده و در قسمت Body و زیر مجموعه form-data مقدار title و body را تنظیم کردیم ارسال می کنیم تا اطلاعات آپدیت شود.

نکته : اگر در آدرس url بعد از <int:pk> اسلش (/) گذاشته باشیم باید در نرم افزار postman هم در آدرس بعد از 1 ، / را بگذاریم.

حذف اطلاعات (DELETE):

modelName.objects.all().filter(sotonName\_\_startwith="exalmple")

modelName.objects.all().filter(dateName\_\_year=2020)

modelName.objects.all().exclude(dateName\_\_year=2020)

modelName.objects.all().filter(sotonName\_\_startwith="exalmple").exclude(dateName\_\_lte=datetime.date.today()) کوچکتر از امروز

modelName.objects.all().filter(sotonBodyText\_\_icontains="exalmple") شامل این عبارت باشد

modelName.objects.all().order\_by("name")

modelName.objects.all().order\_by("name")[:10]ده مورد اول

modelName.objects.values() به صورت یک دیکشنری یا کی و ولیو

modelName.objects.values("name") فقط نام ها

modelName.objects.values\_list() به صورت لیست تاپل

modelName.objects.count()تعداد

modelName.objects.get(id = 10)

modelName.objects.filter(Q(name="Django") | Q(name="Java") )

modelName.objects.filter(entry\_\_blog\_\_name="Django") ارتباط بین چند جدول با پرایمری کی

new\_blog, created = modelName.objects..get\_or\_create(name="new" , tagname= "new new")