جنگو براساس داکیومنتیشن جنگو

مفاهیم توسعه دهنده بک اند و فرانت اند

پیش نیاز های دوره توسعه دهنده بک اند

Python , Sql database , Django , Git , Github , Docker , Linux

Html , Css, Javascripts , React , Node , Angular

Network+

مفاهیم تدریس :

برای حداقل های یک صفحه وب به لحاظ فرم ها ، یادگیری مبانی فریمورک Html مروری بر پایتون و ساخت ماشین مجازی ، نگاهی کوتاه به بررسی زبان

جنگو ، کار با المان ها و پایگاه داده و فاز استقرار و بالا بردن پرفورمنس سایت

تدریس میشودSection دوره در سه

بخش اول : آشنایی با ابزار و نصب و کانفیگ ابزار مورد نیاز

ها و تست و خطا یابیOrm بخش دوم : مبانی جنگو و کار با درخواست ها و صفحات ، پایگاه داده و

بخش پایانی : مفاهیم هاست و پیاده سازی و فاز استقرار و بهبود کیفیت عملکرد سایت و ابزار های آنالیز سایت

سیلابس دوره :

Template tag & form ها ، Viwe ، پایگاه داده و پنل ادمین و Template , static , media مروری بر پایتون ، مبانی جنگو ، کار با

ماژول هلی جنگو ، احراز هویت کاربران . فاز استقرار

شروع به کار :

مروری بر مفاهیم پایتون و شی گرایی . کلاس ها و ارث بری و دکوراتور ها در شی گرایی

تعریف فریمورک

مانند ماژول ها از پیش تعریف شده اند اما میتوان بصورت مستقیم بر روی المان های آن تغییراتی را داد

از این لحظه هر مفهوم و مطلب جدیدی را سرچ میکنیم

بررسی دلایل استفاده از جنگو در سایت جنگو

<https://www.djangoproject.com/start/overview/>

<https://www.djangoproject.com/>

(django vs flask )بررسی و مقایسه جنگو و فلسک

کرومWapplayzer بک اند سایت هایی که با جنگو نوشته شده اند با استفاده از افزونه

در این افزونه گاها نیاز است چند صفحه از سایت را دید تا افزونه بتواند اطلاعات بیشتری بدست بیاورد

Lts (long time support ) بررسی نسخه های مختلف جنگو و در نهایت استفاده از نسخه های

یادگیری قطعی گیت هاب جهت نگهداری و توسعه برنامه و دانلود نسخه دسکتاپ

ساخت اولین ریپازیتوری و نحوه تعامل با گیت هاب

شروع به کار با وی اس کد از طریق گیت هاب

در زمان نصب نسخه جنگو ماژول های قرمز رنگ نیز نصب میشود

tzdata, sqlparse, asgiref, django این ماژول ها دیپندنسی ها هستند

در نهایت برای ماژول ها یک فایل متنی ساخته و بعنوان اولین کامیت نیز در گیت پوش میکنیم

شروع اولین پروژه با جنگو

jango-admin startproject config . ساخت پروژه جدید و کانفیگ جدید در مسیر جاری

search : django cheat sheet >> images

python manage.py runserver استارت یک وب سرور کوچک در سیستم شخصی جهت دیدن ساختار سایت که محتویات سرور از جمله آدرس لوپ بک و ساعت ساخت پروژه و ... در آن قید شده

مد نظر باشد دستورات باید از مسیر جاری فولدر پروژه باشد

بررسی معماری جنگو

Mvc ( model viwe controler )ساختار

ارتباط بین کاربر و سرور و صفحات را برقرار میکند

ها میرود و سپس بررسی میشود که آیا ویو خودش به تنهایی داده ها راViwe در این معماری ابتدا درخواست به سرور یا کنترلر میرود . سپس از سمت سرور به

میتواند نمایش دهد یا نیاز به داده های از دیتا بیس یا مدل دارد

user

(mvt : model viwe template)در جنگو این معماری کمی عوض میشود و جایگشتی بعنوان تمپلیت به جای کنترلر قرار میگیرد

browser

urls

viwes

model

database

متود هاhttp

Get : وقتی که یک درخواستی به سروری ارسال میشود و ما صرفا چیزی میخواهیم و آن اطلاعات در مرو رگر قابل مشاهده هستند را متود گت میگویند . همچنین این متود حکم خواندن اطلاعات را از پایگاه داده دارد

Post : اطلاعاتی که با این متود ارسال میشوند در مرورگر قابل مشاهده نیستند و از طریق روش های خاصی فقط از سمت سرور دریافت میشوند. عموما متود پست برای برای قرار دادن یا آپدیت کردن مقادیری درون پایگاه داده استفاده میشود .

Put & Update & Delete , … : عموما این متود ها برای ایجاد تغییراتی مانند آپدیت کردن یا حذف کردن داده در پایگاه داده استفاده میشوند

من میخواهم با یک اکانت لاگین کنم لذا این اکانت درون پایگاه داده وجو دارد و لذا یک متود گت صرفا جهت خواندن داده ی آن یوزر مورد استفاده قرار میگیرد

من میخواهم یک اکانت جدید بسازم لذا متود پست جهت ساخت ان اکانت مورد استفاده قرار میگیرد و سایر متود ها

Ststus code

مقداری عددی که به معنی پاسخ سرور به کاربر جهت بارگیری صفحه خواسته شده میباشد

شروع کار با جنگو

بیاید و بنا به تعریف تابعی در ویو را اجرا کند url قرار است از طریق مرورگر درخواست بدهیم که به Mvt طبق ساختار

در مسیر خود کانفیگ ایجاد و یک تابع درون آن مینویسیم که رکوئست کاربر به آن پاس میشود Viwes.py لذا یک فایل

تابع را از ویو ایمپورت کرده و بعنوان یک مسیر جدید بعد از ادمین تعریف میشود Urls.py در فایل

همچنین بعد از هر تغییر بدلیل اینکه دیباگ ترو است مدوما در سرور ترمینال خطاها و تغییرات ثبت و گذارش میشوند



دارد http اگر در مرورگر در آدرس لوپ بک لینک را صدا بزنیم اروری صادر میشود که متود گت نمیتواند استرینگ برگرداند و نیاز به یک پاسخ مناسب

را ایمپورت و سپس در تابع اصلاح زیر را انجام میدهیمDjango.http import httprespone لذا در فایل ویو کلاس

درحالت دوم میخواهیم یک جیسون فایل در مرورگر نمایش داده شود



app ساخت

روش ساخت پروژه های جنگو بر اساس استاندارد های جنگو برای هر پروژه جنگو اپلیکیشن آن را ایجاد میکنیم

تعریف میکنیم Urls.py در این روش برای پروژه یک اپلیکیشن ایجاد میشود و سپس در

App 2

App 3

App1

Main django project

که برای مراجعه به فلان درخواست به کدام اپلیکیشن مراجعه کند

python manage.py startapp store

آن اپ آمد به این پوشه هدایت شودurl ابتدا در فایل ستینگ اصلی کانفیگ اپلیکیشن را اضافه میکنیم . دلیل این کار این است که اگر هر درخواستی به

دراپلیکیشن و نام کلاس ایجاد شده میباشدApp.py این کار به هر دو صورت زیر انجام میشود که در حالت دوم مراجعه به فایل

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'store',

]

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'store.apps.StoreConfig',

]

در فایل اصلی میگوییم هرکس به فلان مسیر مراجعه کرد به فایل اپلیکیشن و فایل لینک ها مراجعه کن Urls.py در فایل Include اینبار با استفاده از تابع

from django.contrib import admin

from django.urls import path ,include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('',include('store.urls'))

]

ایجاد و نمایش Urlpatterns یک Path وجود ندارد لذا اقدام به ساخت فایل کرده و با ایمپورت کردن Urls.py درون اپلیکیشن فایل

ویو هارا درون این لیست تعریف میکنیم . و باید ویو مربوط به آن اپلیکیشن درون این فایل ایمپورت شود . و درون ویو توابع را میسازیم

from django.urls import path

from .views import \*

urlpatterns=[

    path('' ,page1)]

from django.shortcuts import render

from django.http import HttpResponse

def page1(req):

    return HttpResponse('<h1>hello django</h>')

تکمیل تر شدن پروژه با سه ویو

from django.contrib import admin

from django.urls import path ,include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('',include('store.urls')),

]

Urls.pyمد نظر باشد وقتی آدرس اصلی را ارجاع میدهیم به اپلیکیشن دیگر برای زیر ادرس ها نیازی نیست داخل فایل اصلی

تعریف کنیم

مد نظر باشد دی این قسمت ما فقط آدرس اپلیکیشن ها را میدهیم

باید تعریف کنیمUrls.py مثلا اگر قرار باشد یک بلاگ باشد که خودش زیر ادرس های خودش را دارد آنموقع در

from django.urls import path

from .views import \*

urlpatterns=[

    path('' ,home),

    path('contact' ,contact\_us),

    path('about' ,about),

]

from django.shortcuts import render

from django.http import HttpResponse

def home(req):

    return HttpResponse('<h1>home page</h>')

def contact\_us(req):

    return HttpResponse('<h1>contact us</h>')

def about(req):

    return HttpResponse('<h1>about us</h>')

کار با تمپلیت ها

به معنی این است که اگر من برای App\_dirs=True مربوط به پروژه تنظیمات در قسمت تمپلیت ها گزینه Setting.py درون فایل

نمایش صفحات تمپلیتی را فراخوانی کردم به فولدر تمپلیتی که درون اپلیکیشن است مراجعه کند لذا ما یک فلودر تمپلیت درون اپلیکیشن میسازیم



و درون آن فایل های اچ تی ام ال را میسازیم

باشدTemplates اسم پوشه باید

درون این پوشه نیز یک پوشه به نام اپلیکیشن ساخته و آدرس دهی رندر را از این پوشه میگذاریم

استفاده میکنیم Lorem ipsom برای نوشتن متن های پیوسته از

درون توابع بجای استفاده از ریسپانس یک رندر از رکوئست و صفحه اچ تی ام ال میسازیم



حتما بعد از این تغییرات سیو میکنیم تا سرور جنگو ری استارت شده تا تغییرات را بتواند بشناسد

گسترش عملکرد تمپلیت ها

هدف اینکار این است که به ازای هر اپلیکیشن یک فولدر تمپلیت نداشته باشم و یک فولدر تمپلیت داشته باشم که برای کل اپلیکیشن های پروژه قابل استفاده باشد

تغغیر زیر را اعمال میکنیم Template در قسمت Setting.py ایجاد و در فایل Templates لذا در پوشه پروژه یک فولدر



TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': [BASE\_DIR, 'templates'],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

سپس محتویات پوشه تمپلیت داخل اپلیکیشن را به داخل این پوشه منتقل میکنیم و پوشه تمپلیت در اپلیکیشن را پاک میکنیم

Setting/system/clipboard/>>turn on history clipboard >> windows+vفعال سازی هیستوری کلیپبورد

انتخاب تمپلیت ها با توجه به سایت ها

Html template download

Colorlib (landing page or one page ) cv or resume

ما بر روی یک تمپلیت بلاگ کار میکنیم

Use wget to download webpages

Download wget for windows

به آن پوشه و باز کردن پاور شل در آن مسیرWget ساخت یک پوشه و انتقال

./Wget.exe –r address previwe page

قالب انتخابی و بررسی قالب انتخابی

را برای هر سایت نیاز داریم همچنین یک بلاگ که پست های ما در آن قرار میگیرد و Home , about , contact سه المان اصلی

سینگل بلاگ که جزییات هر پست را نمایش میدهد

بر ای اینکه بتوان وی اس کد را جهت انتخاب گزینه بازکردن پیش فرض فولدر ها قرار دارد مسیر زیر

Regedit\HKEY\_CLASSES\_ROOT\Directory\shell

تغییر میدهیمOpen with vscode آن را به Key میگذاریم . مقدار دیفالت Vscode زده و نام آن را New درون شل یک

گذاشته و مقدارش را به مسیر اجرایی وی اس کد تغییر میدهیم Icon ساخته و اسمش را String value درون آن یک

"C:\Users\unit\_18\AppData\Local\Programs\Microsoft VS Code\Code.exe"

سپس درون فولدر رجیستری یک فولدر کامند ساخته و مقدار آن را برابر مسیر فایل اجرایی قرار میدهیم و یک درصد وی

"C:\Users\unit\_18\AppData\Local\Programs\Microsoft VS Code\Code.exe" "%V"

فولدر را باز کرده و فایل Live server جهت ویرایش قالب انتخابی دایرکتوری تمپلیت را با وی اس کد باز کرده و پس از نصب افزونه

تمپلیت را راست کلیک و با لایو سرور باز میکنیم

بررسی چندین سایت جهت نمونه تمپلیت های آماده

یک استایل دهی سی اس اس میکنیمIndex.html ابتدا در فایل

مد نظر باشد این فایل هوم فعلا یک صفحه ساده است

این استایل دهی در هد اعمال میشود

<style>

        h1{

            color: red;

        }

    </style>

گاها نیاز است بجای اینکه خودمان بشینیم و برای تک تک تگ ها سی اس اس بنویسیم از مفهومی به عنوان سی دی ان استفاده میشود

این لینک ها مجموعه ای از سی اس اس ها است که بصورت یکجا و در کنار هم جمع شده اند و ما میتوانیم از آن لینک استفاده کنیم

Bootstrap cdn مثلا با سرچ

به این لینک ها میرسیم . لذا لینک و اسکریپت جاوا اسکریپتش را کپی کرده و لینک را در هد و اسکریپت جاوا اسکریپتش را در بادی به مینویسیم

لذا آن سی دی ان به تمام تکست ها اعمال میشود

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB+EYdgRZgiwxhTBTkF7CXvN" crossorigin="anonymous"></script>

مد نظر باشد این روش استفاده از استایل ها اکسترنال یا خارجی میباشند . ماگاها در سمت فرانت خودمان کد بوت استرپی مینویسیم و میخواهیم بصورت اینترنال فراخوانی شوند . لذا باید یک محل استاتیکی داشته باشیم که بجای آدرس دهی اینترنتی در آن استایل یک آدرس داخلی تعریف کنیم که مانند تمپلیت یک فولدر مشترک در پروژه باشد تا بتوانیم از آن ادرس های داخلی یا اینترنال استفاده کنیم

همچنین این استایل دهی ها در سایت ثابت و استاتیک هستند

قسمت های پکیج و هتل و ... را به جز آن سه قسمت اصلی حذف میکنم که میتوان با المنت مکان آن را پیدا کرد و سپس در فایل ایندکی حذف کرد . از طرفی قایل های آن را هم در فولدر تمپلیت پاک میکنم

ابتدا تمپلیت ها را میسازیم

سپس استاتیک و مدیا را ست میکنیم

سپس پروژه را شروع میکنیم

Setting.pyتعریف چهار آیتم زیر در

STATIC\_URL = 'static/'

STATIC\_ROOT = BASE\_DIR.joinpath('/static')

MEDIA\_URL = 'media/'

MEDIA\_ROOT = BASE\_DIR.joinpath('/media')

STATICFILES\_DIRS =[

    BASE\_DIR/'static',

    BASE\_DIR/'media',

]

Urls.py تعاریف مسیر های استاتیک در

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [ path('admin/', admin.site.urls),

]

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

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

برای استفاده از جینجا تمپلیت تگ ها

{% load static %}

“Css/…/… .css”>>> “{% static ‘Css/…/… .css’ %}” : استفاده از روت استاتیک فایل های مربوط به تمپلیت

ارث بری در تمپلیت ها

Create base.html

{% load static %}

Same html code

{% block content %}

{% endblock %}

Same html code

>>>>>

Index.html

{% extends ‘base.html’ %}

{% load static %}

{% block content %}

Html page code

{% endblock %}

داینامیک یو ار ال

{% url ‘app\_name:path name(name=contact)’ %}

Django ORM مفهوم

(object-relational-mapping-layer)

به این معنی که جنگو یک پل ارتباطی بین اپلیکیشن و پایگاه داده های مرسوم دارد

ویژگی ها :

نیاز نیست وارد پایگاه داده شویم و مراحل ساخت جداول و دیتابیس را کوئری کنیم و از طریق خود جنگو و کد های پایتون میتوانیم کوئری مورد نیاز پایگاه داده داشته باشیم

یک رابط گرافیکی تحت وب در اختیار ما میگذارد که میتوانیم المان ها را بررسی کنیم

Migrate , makemigrate

Migrate , make migrate در جنگو شامل دو دستور است . Migration عملیات

از تفسیر Migrate یک ساختار در جنگو را تفسیر میکند به زبان پایگاه داده اما هنوز اعمال نشده اما Makemigrate دستور

استفاده و آن را بر روی پایگاه داده اعمال میکند Makemigrate شده های

Python manage.py migrate را اجرا و سپس دستور Python manage.py makemigrations ابتدا دستور

را اجرا میکنیم تا جداول مربوط به اپلیکیشن های پیشفرض مثل ادمین و اتنتیکیشن در جداول پایگاه داده اس کیو لایت اجرا شود

میتوانیم محتویات پایگاه داده را ببینیم Db browser for sqlite سپس با دانلود

در پایگاه داده هیستوری اتفاقات در حال ثبت است Django-migrations مد نظر باشد در تیبل

در این مرحله نوبت به ساخت یوزر و پسورد پنل ادمین میباشد

نبود باید به پایگاه داده مراجعه و در بخش یوزر ها یک یوزر میساختیم Django orm اگر

اما در جنگو میتوان این کوئری را به شکل دیگری ارسال کرد

جهت دسترسی به بخش گرافیکی باید یک یوزر تعریف کنیم لذا در دیتا بیس یک یوزر میسازیم که دسترسی ادمین دارد و میتواند درون پنل ادمین اقدام به تغییرات کند

Python manage.py createsuperuser

انتخاب نام و ایمیل و پسورد برای یوزر ادمین

اگر بر روی نام یوزر کلیلک کنیم وارد صفحه تنظیمات آن یوزر شده و میتوانیم اطلاعات آن یوزر را تغییر دهیم



وقتی در مدل اپلیکیشن یک کلاس تعریف میشود نام آن کلاس میشود نام آن جدول و اتربیوت های آن کلاس میشوند فیلد ها و جنسشان در جدول

Django model data type میتوان دیتا تایپ های جنگو مدل را دید . با سرچ Geekforgeek در سایت

https://www.geeksforgeeks.org/django-model-data-types-and-fields-list/

ایجاد اولین مدل برای بلاگ پست ها

در اپلیکیشن و در مدل کلاسی ایجاد میکنیم که نام کلاس اسم جدول و اتربیوت های آن فیلد های جدول میباشند

آن پست باشد Content آن پست و Title فرض میکنیم میخواهیم دارای دو اتربیوت

class post(models.Model):

    title = models.CharField(*max\_length*=255)

    content = models.TextField()

سپس برای آماده شدن مدل دستور زیر در ترمینال

python manage.py makemigrations my\_weblog

را آماده ساختن جدول در دیتا بیس کنmy\_weblog در این دستور گفته ام که محتوای داخل مدل ها در اپلیکیشن

در اپلیکیشن مدل آماده اجرا جهت ساخت میشود Migration وقتی دستور اجرا میشود در فولدر

سپس با دستور زیر جدول با فیلد های خواسته شده ایجاد میشود و اگر به دیتابیس مراجعه کنیم قابل مشاهده است

python manage.py migrate

drawsqlدر نهایت استفاده از سایت

هدف این است که شمای کاری دیتابیس را طراحی کنیم و سپس آن را تبدیل به کد کنیم

دیتاهای داخل مدل را آماده میکند برای انتقال به دیتا بیسMakemigrations مد نظر باشد هر دستوری از

اولین بار وقتی ران سرور میکنیم میگوید چندین دیتا برای مایگریشن و انتقال به دیتا بیس از مدهای دیفالت جنگو وجود دارد که هنوز ساخته نشده اند

برای طراحی دیتا بیس سایت نیاز است بدانیم کجاهای سایت نیاز به دریافت اطلاعات داینامیک دارند و چه فیلد هایی نیاز دارند

class post(models.Model):

*# image*

*#author*

    title = models.CharField(*max\_length*=255)

    content = models.TextField()

*# tag*

*# category*

    coounted\_viwes = models.IntegerField(*default*=0)

    status = models.BooleanField(*default*=False)

    published\_date = models.DateTimeField(*null*=True)

    created\_date = models.DateTimeField(*auto\_now\_add*=True)

    updated\_date = models.DateTimeField(*auto\_now*=True)

save مد نظر باشد با دستورات زیر میتوان در شل از این مدل آبجکت های متفاوت ساخت و فیلد ها را مقدار دهی کرد و درنهایت با دستور

آن را در دیتا بیس قرار داد

صفحه را رفرش و المان هارو ریکاوری میکنه . Ctrl + f5 دکمه های ترکیبی

وق آن رسیده که جدول موجود در پایگاه داده را رجیستر کنیم Admins.py در فایل

ابتدا از فایل مدل کلاس ساخته شده را ایموپورت و سپس با دستورات زیر جدول را رجیستر میکنیم

میتوان از داکیومنت جنگو اطلاعات خوبی بدست آورد custom admin form django با سرچ عبارت

*from* django.contrib *import* admin

*from* .models *import* post

admin.site.register(post)

از آنجایی که دیتابیس در پنل ادمین بصورت آبجکت ها نمایش داده میشود لذا با تعریف مجیک متود زیر در کلاس مدل میتوان فرضا نام تایتل را بعنوان نام فیلد ها نمایش داد

class post(models.Model):

*# image*

*#author*

    title = models.CharField(*max\_length*=255)

    content = models.TextField()

*# tag*

*# category*

    coounted\_viwes = models.IntegerField(*default*=0)

    status = models.BooleanField(*default*=False)

    published\_date = models.DateTimeField(*null*=True)

    created\_date = models.DateTimeField(*auto\_now\_add*=True)

    updated\_date = models.DateTimeField(*auto\_now*=True)

    def \_\_str\_\_(*self*):

*return* self.title

میتوانیم از دکوراتور آن استفاده کنیم admin.site.register همچنین بجای استفاده از تابع

این دکوراتور باید بالای کلاسی نوشته شود که جهت شخصی سازی پنل ادمین استفاده میشود

@admin.register(post)

class PostAdmin(admin.ModelAdmin):

    list\_display = ('title', 'content')*# چه فیلد هایی از آن سطر در صفحه نمایش داده شوند*

@admin.register(post)

class PostAdmin(admin.ModelAdmin):

    list\_display = ('id','title', 'status', 'created\_date')*# چه فیلد هایی از آن سطر در صفحه نمایش داده شوند*

    list\_filter = ('status', )*# افزودن یک گزینه فیلتر بر اساس استتوس ها و یا هر فیلد دیگری*

    search\_fields = ('title', )*# یک سرچ فیلد میگذارد که یک عبارت را مثلا در ستون تایتل ها سرچ میکند*

    empty\_value\_display = '-empty-' *# اگر فیلدی خالی بود خالی نشان نده و بجای آن عبارتش را نشان بده*

    fields = ('title', 'content', 'published\_date') *# وقتی بر روی یک سطر کلیک میکنیم کدام بخش ها قابل تغییر باشند*

    ordering = ('created\_date',  )*# بر اساس تاریخ ایجاد صفحه ادمین را سورت میکند*

*#admin.site.register(post, PostAdmin)*

استفاده از کلاس های متا

اگر به مثال بالا دقت کنیم اوردرینگ در کلاسی نوشته شده که پنل ادمین نمایش میدهد

این اوردرینگ نحوه سورت شدن نمایش صفحه است

عموما متا مدل ها ارتباطی به فیلد های پایگاه داده ندارند

برای ایجاد متا مدل باید یک کلاس داخل کلاس مدل تعریف شود

است که بر اساس تاریخ ساخت یا ایجاد سطر قرار میگیردOrdering مد نظر باشد نرمال ترین و همیشگی ترین متا مدل فقط

که برای نوشته کادر بالای دیتا بیس Verbose\_name , verbose\_name\_plural دو اتریبیوت دیگر نیز وجود دارد به نام های

و کناری استفاده میشود

مفهوم این متامدل این است که کوئری هایی که به پایگاه داده پست ارسال میشوند براساس تاریخ ساخت یکی یکی اعمال شوند

*from* django.db *import* models

class post(models.Model):

*# image*

*#author*

    title = models.CharField(*max\_length*=255)

    content = models.TextField()

*# tag*

*# category*

    coounted\_viwes = models.IntegerField(*default*=0)

    status = models.BooleanField(*default*=False)

    published\_date = models.DateTimeField(*null*=True)

    created\_date = models.DateTimeField(*auto\_now\_add*=True)

    updated\_date = models.DateTimeField(*auto\_now*=True)

    class Meta:

        ordering = ['-created\_date']

    def \_\_str\_\_(*self*):

*return* self.title

در ادامه قصد داریم پست هایی که در دیتا بیس وجود دارند را در پروژه نمایش دهیم

Ststus=1نکته ای که وجود دارد این است که پست هایی را قرار میدهیم که در وضعیت انتشار هستند یعنی

در ویو ها

*from* django.shortcuts *import* render

*from* .models *import* cheap\_package, luxary\_package, camping\_package, post

def b\_home(*req*):

    posts=post.objects.filter(*status*=1)

    context = {

        'posts':posts

    }

*return* render(req,'my\_weblog/blog-home.html',*context*=context)

حال چون ساختار تمام پست ها یکسان است فقط یکی را نگه میداریم و از حلقه در کد اچ تی ام ال استفاده میکنیم

{% for post in posts %}

                <div class="single-post row">

                    <div class="col-lg-3  col-md-3 meta-details">

                        <ul class="tags">

                            <li><a href="#">Food,</a></li>

                            <li><a href="#">Technology,</a></li>

                            <li><a href="#">Politics,</a></li>

                            <li><a href="#">Lifestyle</a></li>

                        </ul>

                        <div class="user-details row">

                            <p class="user-name col-lg-12 col-md-12 col-6"><a href="#">Mark wiens</a> <span

                                    class="lnr lnr-user"></span></p>

                            <p class="date col-lg-12 col-md-12 col-6"><a href="#">{{post.published\_date}}</a> <span

                                    class="lnr lnr-calendar-full"></span></p>

                            <p class="view col-lg-12 col-md-12 col-6"><a href="#">{{post.coounted\_viwes}}</a> <span

                                    class="lnr lnr-eye"></span></p>

                            <p class="comments col-lg-12 col-md-12 col-6"><a href="#">06 Comments</a> <span

                                    class="lnr lnr-bubble"></span></p>

                        </div>

                    </div>

                    <div class="col-lg-9 col-md-9 ">

                        <div class="feature-img">

                            <img class="img-fluid" src="{% static 'img/blog/feature-img1.jpg' %}" alt="">

                        </div>

                        <a class="posts-title" href="{% url 'my\_weblog:blog-single' %}">

                            <h3>{{post.title}}</h3>

                        </a>

                        <p class="excert">

                            {{post.content}}

                        </p>

                        <a href="{% url 'my\_weblog:blog-single' %}" class="primary-btn">View More</a>

                    </div>

                </div>

                {% endfor %}

مقادیر را در یک پست جایگزین میکنیم

urlارسال پارامتر های دیگر در

path('<str:name>/<str:family>/<str:age>', test, *name*='test')

def test(*req*,*name*,*family*,*age*):

    context={

        'name':name,

        'family':family,

        'age':age,

    }

*return* render(req,'my\_weblog/test.html',*context*=context)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>Document</title>

</head>

<body>

    {{name}}

    {{family}}

    {{age}}

</body>

</html>

نمونه دیگر این روش استفاده از گت است

path('<int:pid>', test, *name*='test')

def test(*req*,*pid*):

    posts = post.objects.get(*id*=pid)

    context = {

        'post':posts.content

    }

*return* render(req,'my\_weblog/test.html',*context*=context)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>Document</title>

</head>

<body>

    {{post}}

</body>

</html>

در این روش اگر آن آی دی پیدا نشود ارور میدهد اما میخواهم صفحه 404 نمایش داده شود لذا بصورت زیر تغییرات میدهیم مد نظر باشد در این روش فقط یک پست برگردانده میشود و نیاز به حلقه بر روی پست نیست

 path('<int:pid>', test, *name*='test')

*from* django.shortcuts *import* render, get\_object\_or\_404

def test(*req*,*pid*):

*#posts = post.objects.get(id=pid)*

    posts = get\_object\_or\_404(post, *pk*=pid)*#>>>>>> posts = post.objects.get(id=pid)*

    context = {

        'post':posts.content

    }

*return* render(req,'my\_weblog/test.html',*context*=context)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>Document</title>

</head>

<body>

    {{post}}

</body>

</html>

میتوان در قالب سایت به این ترتیب موارد خواسته شده را پیاده سازی کرد

در جینجا تگ ها زمانی که میخواهیم مقداری را نمایش دهد داخا 2 تا آکولاد و وقتی میخواهیم عملیاتی مثل حلقه یا شرط داشته باشیم داخل استفاده کردendfor & endif , … و درصد و در انتها نیز حتمن باید برای حلقه و شرط و ... از

{% for post in posts %}

{% endfor %}

استفاده میکنیمFilter از All برای دریافت پست هایی با یک سری فیلتر بجای

برای این کار دوحالت موجود است . در تمپلیت و در توابع ویو

def b\_home(*req*):

    posts=post.objects.filter(*status*=1)

    context = {

        'posts':posts

    }

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>Document</title>

</head>

<body>

    {% *for* post in posts %}

        {% *if* post.status == 1 %}

            {{post.title}}

        {% *else* %}

            {{post.created\_date}}

        {% *endif* %}

    {% *endfor* %}

</body>

</html>

میخواهیم اطلاعات دیتا بیس را به داخل تمپلیت اصلی بیاوریم

اگر دیتایی که قرار است در فایل اچ تی ام ال مربوطه بگذاریم در تمپلیت به دفعات تکرار شده باشد یک ساختار را نگه داتشته و مابقی را دیلیت و بر روی آن حلقه میزنیم



همانطور که در صفحه بلاگ میبینیم ساختار پستها به دفعات تکرار شده

با مراجعه به فایل اچ تی ام ال خواهیم دید لیستی وجود دارد که این ساختار مداوما در آن تکرار شده



لذا بغیر از اولی همه را پاک کرده و یک حلقه بالای آن ایجاد میکنیم



سپس در داخل تگ دیو قسمت هایی که نیاز است اصلاح میکنیم



مد نظر باشد وقتی ما داریم محتوارا جایگزین میکنیم اگر محتوا زیاد باشد حجم زیادی از صفحه میگیرد که بعد ها یاد میگیریم در قسمتی محتوا در صفحه بلاگ هوم خلاصه آن را بگذاریم

در قسمت بعد برای اینکه با کلیک روی هر پس برود به ویو آن پست و کامل باشد نیاز است آی دی آن پست هنگام کلیلک به صورت پارامتر برای یک ویو ارسال و آن ویو بر اساس آن آی دی صفحه بلاگ سینگل را بسازد

لذا در یو ار ال های آن اپلیکیشن این شکل فراخوانی که قرار است یک پارامتر ارسال شود را مینویسیم

 path('<int:pid>', b\_single, *name*='blog-single' )

از جنس عدد صحیح نیز به تابعی که در یو ار ال ها وجود دارد ارسال میشود pid در خط بالا یک

در ویو نیز ما یک پست میخواهیم صدا زده شود و اگر وجود نداشت ارور 404 برگردانده شود

لذا در تابع ویو به صورت زیر عمل میکنیم

def b\_single(*req*, *pid*):

    posts=get\_object\_or\_404(post, *pk*=pid, *status*= True)

    posts.counted\_viwes += 1

    posts.save()

    context = {

        'post':posts

    }

*return* render(req,'blog/blog-single.html',*context*=context)

در خطوط بالا پستی را با آی دی مربوطه و مثلا استتوس قابل انتشار انتخاب کردیم

نیز قرار داده شده به ازای هر بار ورود آن پست تعداد ویو های آن پست یکی اضافه میشود Conted\_viwe +=1 همچنین علت اینکه

استفاده شده است Save()و چون قرار است در دیتابیس ذخیره شود حتمن از تابع

همچنین به ازای هر پس در صفحه تاریخ و تعداد بازدید راهم میتوانیم در همین صفحه تغییر دهیم

علت اینکه در تابع بلاگ سینگل نیز استتوس برابر یک گذاشتیم این است که اگر کسی دستی در یو ار ال یک آی دی زد نیز قابل مشاهده نباشد

<p class="date col-lg-12 col-md-12 col-6"><a href="#">{{post.published\_date}}</a> <span class="lnr lnr-calendar-full"></span></p>

                                        <p class="view col-lg-12 col-md-12 col-6"><a href="#">{{post.counted\_viwes}} Views</a> <span class="lnr lnr-eye"></span></p>

هایی که به صفحه بلاگ سینگل میروند url حال برای اینکه روی هر پست کلیک کنیم تا به صفحه خودش در بلاگ سینگل برود باید به

یک پارامتر ارسال کنیم تا بلاگ سینگ با آن پارامتر صدا زده شود

فعلا فقط داخل حلقه در بلاگ هوم این تغییرات را اعمال میکنیم فعلا کار به قسمت های دیگر نداریم

<a class="posts-title" href="{% *url* 'blog:blog-single' pid=post.id %}"><h3>{{post.title}}</h3></a>

                                    <p class="excert">

                                        {{post.content}}

                                    </p>

<a href="{% *url* 'blog:blog-single' pid=post.id %}" class="primary-btn">View More</a>



قرار دادن نام و تاریخ و تصویر هر پست

همانطور که در مدل خواهیم دید نویسنده عموما شخصی است که در جدول یوزر

ها وجود دارد و چون آی دی در جدول یوزر پرایمری است نویسنده رفرنس

داده میشود به آی دی در یوزر

*from* django.contrib.auth.models *import* User

author = models.ForeignKey(User, *on\_delete*=models.CASCADE)

در حالت کسکید باعث میشود اگر آن یوزر حذف شد پست هایش نیز حذف شوند

همچنین در مدل پست میتوانیم تغییرات زیر را در ادمین جهت بهبود جدول پست ها انجام بدم

*from* django.contrib *import* admin

*from* .models *import* post

@admin.register(post)

class PostAdmin(admin.ModelAdmin):

    date\_hierarchy = 'created\_date' *# اعمال یک گزینه تاریخ در بالای دیتا بیس*

    list\_display = ('id','title', 'author', 'status', 'created\_date', 'published\_date', 'counted\_viwes')*# چه فیلد هایی از آن سطر در صفحه نمایش داده شوند*

    list\_filter = ('status', )*# افزودن یک گزینه فیلتر بر اساس استتوس ها و یا هر فیلد دیگری*

    search\_fields = ('title', 'author' )*# یک سرچ فیلد میگذارد که یک عبارت را مثلا در ستون تایتل ها سرچ میکند*

    empty\_value\_display = '-empty-' *# اگر فیلدی خالی بود خالی نشان نده و بجای آن عبارتش را نشان بده*

    fields = ('title', 'content', 'published\_date', 'author', 'status')

حالا در بلاگ هوم وقتی میخواهیم مثلا نام نویسنده را نشان دهیم میتوانیم از فیلد ها یوزر نیز استفاده کنیم

<p class="user-name col-lg-12 col-md-12 col-6"><a href="#">{{post.author.first\_name}}

یا برای نمایش ایمیلش از حالت زیر

<p class="user-name col-lg-12 col-md-12 col-6"><a href="#">{{post.author.email}}

و در نهایت با متود زیر در بلاگ هوم اسم و فامیلش را کامل میگیریم

<p class="user-name col-lg-12 col-md-12 col-6"><a href="#">{{post.author.get\_full\_name}}

دسترسی به بقیه فیلد های یوزر با متود های زیر در دسترس است

**Username**: {{ user.username }}

**User Full name**: {{ user.get\_full\_name }}

**Email**: {{ user.email }}

**Session Started at**: {{ user.last\_login }}

اضافه کردن فیلد تصاویر هر پست

استPillow برای افزودن این فیلد نیاز به نصب پکیج

استفاده میکنیمRequirement.txt لذا بعد از نصب این پکیج رو به

داخل مدل جنس فیل ایمیج را بصورت زیر تعریف میکنیم

image = models.ImageField(*upload\_to* = 'blog', *default*= 'default.jpg')

در آرگومان ارسالی گفته ایم که ایمیج ها در داخل پوشه مدیا و پوشه بلاگ درون آن هستند

همچنین یک فایل عکس دیفالت را در پوشه مدیا گذاشته و مقدار دیفالت فیلد آن عکس را میگذاریم

مد نظر باشد وقتی فقط یک اسم فایل میگذاریم خودش درون پوشه مدیا را میگردد

همچنین درون فایل ستینگ تغییرات زیر را اعمال میکنیم

Makemigrations & migrateسپس

STATIC\_URL = 'static/'

STATIC\_ROOT = BASE\_DIR.joinpath('/static')

MEDIA\_URL = 'media/'

MEDIA\_ROOT = BASE\_DIR.joinpath('media')

STATICFILES\_DIRS = [

    BASE\_DIR/'static',

]

درون فایل بلاگ هوم نیز از فیل ایمیج استفاده میکنیم و آنها را از حالت استاتیک خارج میکنیم

<img class="img-fluid" src="{{post.image.url}}" alt="">

اگر عبارت یو ار ال را نگذاریم محتوا فقط یک آدرس متنی داغریکتوری است که با متود یو ار ال نمایشش میدهیم

اگر عکسی که در دیتابیس و از طریق پنل ادمین آپلود کنیم که اسمش تکراری باشد خود جنگو اتوماتیک یونیک سازی میکند و جای نگرانی نیست

اضافه کردن فیلد کتگوری

گاها نیاز است که یک فیلد بتواند مستقیم خودش مقدار دهی شود ولی همزمان در یک جدول اصلی بالا دستی ثبت شود



یا در جدول بالا دستی ثبت شود و در جدول زیر دستی دیده شود

از طرفی داده هایش یک کقدار نیست و قرار است ایتریبلی باشد از داده ها

که بتوان جدا جدا از آنها استفاده کرد. در این مواقع از جداول

استفاده کردManytomany

ما در جدول کتگوری دسته بندی ها را میسازیم و سپس در فیلد کتگوری پست

از انها استفاده میکنیم

جدول کتگوری را ساخته و سپس در مدل فیلدش را تعریف میکنیم

قابل نمایش در صفحه نمایش اطلاعات دیتابیس نیستند MANYTOMANY مد نظر باشد فیلد های

from django.db import models

from django.contrib.auth.models import User

class category(models.Model):

    name = models.CharField(max\_length=255)

    def  \_\_str\_\_(self):

        return self.name

class post(models.Model):

    image = models.ImageField(upload\_to = 'blog', default= 'default.jpg')

    author = models.ForeignKey(User, on\_delete=models.CASCADE)

    title = models.CharField(max\_length=255)

    content = models.TextField()

    # tag

    category = models.ManyToManyField(category)

    counted\_viwes = models.IntegerField(default = 0)

    status = models.BooleanField(default=False)

    published\_date = models.DateTimeField(null=True)

    created\_date = models.DateTimeField(auto\_now\_add=True)

    updated\_date = models.DateTimeField(auto\_now=True)

        class Meta:

        ordering = ['-created\_date']

    def \_\_str\_\_(self):

        return self.title

همچنین جهت نمایش در پنل ادمین لازم است در ادمین رجیستر شود و چون فقط یک نام دارد نیازی به شخصی سازی پنل ادمین برای کتگوری نمیباشیم

from django.contrib import admin

from .models import post, category

admin.site.register(category)

@admin.register(post)

class PostAdmin(admin.ModelAdmin):

    date\_hierarchy = 'created\_date'

    list\_display = ('id','title', 'author', 'status')

    list\_filter = ('status', )

    search\_fields = ('title', 'author' )

    empty\_value\_display = '-empty-'

    fields = ('title', 'content', 'published\_date', 'author', 'status', 'image', 'category')

پیمایش کنیم مثل کتگ.ری فرم تمپلیت تگ آن بصورت زیر میباشد manytomany وقتی در تمپلیت میخواهیم بر روی یک فیلد

{% for cat in post.category.all %}

<li><a href="#">{{cat.name}}</a></li>

{% endfor %}

اگر نخواهیم داخل تگ باشد و به لینک خاصی جوین نشود میتوانیم بجای حلقه از دستور زیذ استفاده کنیم

<ul class="tags">

{{post.category.all|join:", "}}

</ul>

نمایش تاریخ انتشار پست

استفاده کنیم ولی با تمپلیت تگ زیر میتوانیم نحوه نمایشش را تغییر دهیمPublished\_date برای ویرایش تاریخ نمایش میتوانیم مستقیم از

{{post.published\_date|date:"F j, Y"}}

میتوان با سرچ کلید واژه های زیر انواع فرمت تاریخ را بررسی کرد

Django template date format

Available format strings:

| **Format character** | **Description** | **Example output** |
| --- | --- | --- |
| **Day** |  |  |
| **d** | Day of the month, 2 digits with leading zeros. | **'01'** to **'31'** |
| **j** | Day of the month without leading zeros. | **'1'** to **'31'** |
| **D** | Day of the week, textual, 3 letters. | **'Fri'** |
| **l** | Day of the week, textual, long. | **'Friday'** |
| **S** | English ordinal suffix for day of the month, 2 characters. | **'st'**, **'nd'**, **'rd'** or **'th'** |
| **w** | Day of the week, digits without leading zeros. | **'0'** (Sunday) to **'6'** (Saturday) |
| **z** | Day of the year. | **1** to **366** |
| **Week** |  |  |
| **W** | ISO-8601 week number of year, with weeks starting on Monday. | **1**, **53** |
| **Month** |  |  |
| **m** | Month, 2 digits with leading zeros. | **'01'** to **'12'** |
| **n** | Month without leading zeros. | **'1'** to **'12'** |
| **M** | Month, textual, 3 letters. | **'Jan'** |
| **b** | Month, textual, 3 letters, lowercase. | **'jan'** |
| **E** | Month, locale specific alternative representation usually used for long date representation. | **'listopada'** (for Polish locale, as opposed to **'Listopad'**) |
| **F** | Month, textual, long. | **'January'** |
| **N** | Month abbreviation in Associated Press style. Proprietary extension. | **'Jan.'**, **'Feb.'**, **'March'**, **'May'** |
| **t** | Number of days in the given month. | **28** to **31** |
| **Year** |  |  |
| **y** | Year, 2 digits with leading zeros. | **'00'** to **'99'** |
| **Y** | Year, 4 digits with leading zeros. | **'0001'**, …, **'1999'**, …, **'9999'** |
| **L** | Boolean for whether it’s a leap year. | **True** or **False** |
| **o** | ISO-8601 week-numbering year, corresponding to the ISO-8601 week number (W) which uses leap weeks. See Y for the more common year format. | **'1999'** |
| **Time** |  |  |
| **g** | Hour, 12-hour format without leading zeros. | **'1'** to **'12'** |
| **G** | Hour, 24-hour format without leading zeros. | **'0'** to **'23'** |
| **h** | Hour, 12-hour format. | **'01'** to **'12'** |
| **H** | Hour, 24-hour format. | **'00'** to **'23'** |
| **i** | Minutes. | **'00'** to **'59'** |
| **s** | Seconds, 2 digits with leading zeros. | **'00'** to **'59'** |
| **u** | Microseconds. | **000000** to **999999** |
| **a** | **'a.m.'** or **'p.m.'** (Note that this is slightly different than PHP’s output, because this includes periods to match Associated Press style.) | **'a.m.'** |
| **A** | **'AM'** or **'PM'**. | **'AM'** |
| **f** | Time, in 12-hour hours and minutes, with minutes left off if they’re zero. Proprietary extension. | **'1'**, **'1:30'** |
| **P** | Time, in 12-hour hours, minutes and ‘a.m.’/’p.m.’, with minutes left off if they’re zero and the special-case strings ‘midnight’ and ‘noon’ if appropriate. Proprietary extension. | **'1 a.m.'**, **'1:30 p.m.'**, **'midnight'**, **'noon'**, **'12:30 p.m.'** |
| **Timezone** |  |  |
| **e** | Timezone name. Could be in any format, or might return an empty string, depending on the datetime. | **''**, **'GMT'**, **'-500'**, **'US/Eastern'**, etc. |
| **I** | Daylight saving time, whether it’s in effect or not. | **'1'** or **'0'** |
| **O** | Difference to Greenwich time in hours. | **'+0200'** |
| **T** | Time zone of this machine. | **'EST'**, **'MDT'** |
| **Z** | Time zone offset in seconds. The offset for timezones west of UTC is always negative, and for those east of UTC is always positive. | **-43200** to **43200** |
| **Date/Time** |  |  |
| **c** | ISO 8601 format. (Note: unlike other formatters, such as “Z”, “O” or “r”, the “c” formatter will not add timezone offset if value is a naive datetime (see [**datetime.tzinfo**](https://docs.python.org/3/library/datetime.html#datetime.tzinfo)). | **2008-01-02T10:30:00.000123+02:00**, or **2008-01-02T10:30:00.000123** if the datetime is naive |
| **r** | [**RFC 5322**](https://datatracker.ietf.org/doc/html/rfc5322.html#section-3.3) formatted date. | **'Thu, 21 Dec 2000 16:01:07 +0200'** |
| **U** | Seconds since the Unix Epoch (January 1 1970 00:00:00 UTC). |  |

For example:

{{ value|date:"D d M Y" }}

بین عبارت ها میتوان مثلا . یا / یا هر چیزی گذاشت برای جداسازی

{{ value|date:"D/d/M/Y" }}

گاها نیاز هست در یک لوپ بگوییم اگر اولین یا آخرین عنصر یک ایتریبل نبود فلان کار را بکن

<ul class="tags">

       {% for cat in post.category.all %}

             <li><a href="#">{{cat.name}}{% if not forloop.last %}\*\*{% endif %}</a></li>

       {% endfor %}

</ul>

<ul class="tags">

     {% for cat in post.category.all %}

           {% if not forloop.last %}

                 <li><a href="#">{{cat.name}}\*\*</a></li>

           {% else %}

                 <li><a href="#">{{cat.name}}</a></li>

           {% endif %}

     {% endfor %}

</ul>

میخواهیم در صفحه بلاگ هوم اگر متن اصلی زیاد باشد درست نیست تمام متن در صفحه باشد و نیاز است خلاصه آن باشد

میتوان توابعی را در مدل ها تعریف کرد و بجای کال کردن اتربیوت آن کلاس آن تابع را کال کرد

    category = models.ManyToManyField(category)

    counted\_viwes = models.IntegerField(*default* = 0)

    status = models.BooleanField(*default*=False)

    published\_date = models.DateTimeField(*null*=True)

    created\_date = models.DateTimeField(*auto\_now\_add*=True)

    updated\_date = models.DateTimeField(*auto\_now*=True)

    class Meta:

        ordering = ['-created\_date']

    def \_\_str\_\_(*self*):

*return* self.title

    def snipets(*self*):

*return* self.content[:20] + '...'

و در کد اچ تی ام ال بصورت زیر صدا میکنیم

    <a class="posts-title" href="{% url 'blog:blog-single' pid=post.id %}"><h3>{{post.title}}</h3></a>

       <p class="excert">

*<!-- {{post.content}} -->*

           {{post.snipets}}

در جنگو تمپلیت تگ ها یک تگ وجود دارد که نیاز به تعریف متود در مدل نیست

<p class="excert">

{{post.content|truncatechars:9}}

*<!-- {{post.snipets}} -->*

</p>

در این حالت خودش 8 کارکتر از صفر را میدهد و سه نقطه را میگذارد

{{post.content|truncatewords:2}}

در این حالت بجای اسلایس بر اساس کارکتر بر اساس کلمات انجام میدهد

کردن در تمپلیت تگ هاInclude

فرضا قسمتی از کد های تمپلیت وجود دارند که در تعدادی از صفحات تکراری هستند

مانند صفحات بلاگ پست و سینگل که در سمت راست آن بخشی در هر دو صفحه وجود دارد

در این مواقع ما از ساختاری مانند تابع در پایتون استفاده میکنیم که آن قسمت تکراری را در آن مینویسیم و سپس ایمپورتش میکنیم



در عکس اول قسمت تکرارری را به یک فایل در پوشه بلاگ در تمپلیت بردیم

سپس در هر قسمت از صفحات که نیاز به آن قسمت تکراری بود با اینکلود فراخوانی کردیم

کار با تمپلیت تگ ها

شش پست آخر را نمایش دهد Papular post فرض میکنیم قسمت

برای اینکه ما تمام چیزهایی را که میخواهیم در صفحه تمپلیت استفاده کنیم در ویو ننویسیم و باعث بهم ریختن و شلوغ شدن ویو ها نشود میتوانیم از تمپلیت تگ های کاستومایز شده ای استفاده کنیم که در مکان دیگری ساخته میشوند

درون این تمپلیت تگ ها ما توابع خودمان را مینویسیم و سپس خروجی آن تابع را میتوانیم در صفحه تمپلیت نشان ده9یم

مثلا فرض کنیم ما میخواهیم تعداد ویو های یک پست را هم نمایش دهیم مثل بلاگ هوم و سینگل

فقط در ویو ما تعدادش را یکی اضافه میکنیم و در تمپلیت تگ میتوانیم مقدارش را بسازیم

میخواهیم شش پست آخر را نمایش دهیم نیاز نیست در ویو این قابلیت را بسازیم و در صفحه بعنوانPopularepost یا در قسمت

کانتکس ارجاع دهیم . درون تمپلیت تگ میسازیمش و در صفحه استفاده میکنیم

در هر اپلیکیشن باید پوشه تمپلیت تگ جداگانه ساخته شود

Simpleدر این مرحله تمپلیت تگ های ساده مورد تعریف قرار میگیرند

Create templatetags folder in app

Create \_\_init\_\_.py in folder

Create python file

From django import template

Register = template.Library()

@register.simple\_tag() >>> can be a name for call function in template

@register.simple\_tag(*takes\_context*=True, *name*='post\_tag')

در حالت رجیستر بالا میگوییم از کانتکسی که در ویو به تمپلیت ارسال میشود نیز در زمان فراخوانی تابع بتوان استفاده کرد

{% load template tags file name %}

{% extends 'base.html' %}

{% load static %}

{% load blog-simple %} >> file name

{% block content %}

<p class="excert">

   {{post.content|truncatewords:2}}

*<!-- {{post.snipets}} -->*

   {% post\_tag 5 %} >> function name with argument

*from* django *import* template

*from* blog.models *import* post

register = template.Library()

posts = post.objects.filter(*status*=1)

@register.simple\_tag(*name*='post\_tag')

def blog\_tag(*number*:int= 0):

*return* posts[:3]

همچنین میتوان این تگ را داخی یک متغیر ریخت و در صفحه از آن استفاده کرد

<a class="posts-title" href="{% url 'blog:blog-single' pid=post.id %}"><h3>{{post.title}}</h3></a>

     <p class="excert">

         {{post.content|truncatewords:2}}

*<!-- {{post.snipets}} -->*

         {% post\_tag 3 as list %}

             {% for i in list %}

                   {{i.id}}

            {% endfor %}

filterدر روش دیگری از ساخت تمپلیت تگ

در این روش بجای اینکه از دکوراتور سیمپل استفاده شود از دکوراتور فیلتر استفاده میشود و بجای مقدار دهی مثل سیمپل از روش زیر استفاده میشود

@register.filter

def slicing(*value*:str):

*return* value[:3]

<p class="excert">

   {{post.content|truncatewords:2}}

*<!-- {{post.snipets}} -->*

   {{post.content|slicing }}

@register.filter

def slicing(*value*:str, *number*:int):

*return* value.split()[:number]

<p class="excert">

     {{post.content|truncatewords:2}}

*<!-- {{post.snipets}} -->*

     {{post.content|slicing:2 }}

Inclision tag

در این روش ما یک ساختار را در یک صفحه جداگانه میسازیم و آن صفحه را وارد تمپلیت اصلی میکنیم

@register.inclusion\_tag("popular.html")

def population():

    posts = post.objects.filter(*status*=1)

*return* {

        "posts": posts,

    }

در تمپلیت Popular.html ساخت صفحه

<ul>

    {% for post in posts %}

        <li>

            {{post.title}}

        </li>

    {% endfor %}

</ul>

محتویات صفحه جدید

<p class="excert">

   {{post.content|truncatewords:2}}

*<!-- {{post.snipets}} -->*

   {% population %}

Popular postنمایش شش پست آخر در قسمت

استفاده کنیم و آن بخش را در قسمتی بسازیو و هر جا نیاز بود فراخوانی کنیم Inclusion میتوانیم از تمپلیت تگ

میسازیم Inclusion ابتدا تابع آن در تمپلیت تگ از نوع

@register.inclusion\_tag("popular.html")

def population():

    posts = post.objects.filter(*status*=1)

    six\_latest\_post = posts[:6]

*return* {

        "posts": six\_latest\_post

    }

سپس در فایل اچ تی ام الی که تارگت شده در پوشه تمپلیت ها آن بخش را میسازیم

{% load static %}

{% for post in posts %}

<div class="single-post-list d-flex flex-row align-items-center">

    <div class="thumb">

        <img class="img-fluid" src="{{post.image.url}}" alt="">

    </div>

    <div class="details">

        <a href="{% url 'blog:blog-single' pid=post.id %}">

            <h6>{{post.title}}</h6>

        </a>

        <p>{{post.published\_date|date:'d M Y'}}</p>

    </div>

</div>

{% endfor %}

و سپس در تمپلیت اصلی در جایی که میخواهیم این فرم وجود داشته باشد آن را کال میکنیم

<div class="single-sidebar-widget popular-post-widget">

     <h4 class="popular-title">Popular Posts</h4>

     <div class="popular-post-list">

         {% population %}

     </div>

دسته بندی پست ها در ساید بار

در صفحه یک قسمتی هست به نام پست کتگوری که تعداد کتگوری هارا شمارش کرده و هر پستی که آن کتگوری را دارد

تمپلیت تگ استفاده میکنیم Inclusion برای این کار هم از

بدین ترتیب که ابتدا یک دیکشنری خالی ساخته و با ایمپورت کردن مدل کتگوری در تمپلیت تگ یک لیست کلی از آبجکت ها میسازیم

برابر است و سپس تعداد I سپس با پیمایش بر روی این آبجکت ها از کلاس پست فیلتر میکنیم آبجکت هایی را که کتگوری آنها با

آن را بعنوان ولیو به دیکشنری اضافه و سپس آن دیکشنری را راهی صفحه اچ تی ام ال میکنیم

@register.inclusion\_tag("category.html")

def ccategory():

    cat\_dict = {}

    cat = category.objects.all()

*for* i *in* cat:

        cat\_dict[i] = post.objects.filter(*category*=i).count()

*return* {

        'category':cat\_dict

    }

{% load static %}

<div class="single-sidebar-widget post-category-widget">

    <ul class="cat-list">

        {% for name, count in category.items %}

        <li>

            <a href="#" class="d-flex justify-content-between">

                <p>{{name}}</p>

                <p>{{count}}</p>

            </a>

        </li>

        {% endfor %}

    </ul>

<div class="single-sidebar-widget ads-widget">

     <a href="#"><img class="img-fluid" src="{% static 'img/blog/ads-banner.jpg' %}" alt=""></a>

</div>

{% ccategory %}

نمایش پست ها بر اساس دسته بندی کتگوری

ما برای هر پست کتگوری های متفاوتی داریم . هدف این است که با کلیک بر روی هر کتگوری پست هایی که آن کتگوری را دارند نمایش داده شوند

ما میخواهیم در بلاگ هوم با کلیک بر روی هر کتگوری فقط آن تعداد پست ها نمایش داده شوند . لذا ما یک ویو و مسیر در بلاگ ایجاد میکنیم و طبق اصول زیر رفتار میکنیم

در حالت اول

app\_name = 'blog'

urlpatterns = [

    path('', b\_home, *name*= 'blog-home'),

    path('<int:pid>', b\_single, *name*='blog-single' ),

    path('category/<str:cat>', b\_cat, *name*='b-cat' ),

]

def b\_cat(*request*, *cat*):

    posts = post.objects.filter(*status*=1)

    posts = posts.filter(*category\_\_name*=cat)

    context = {

        'posts':posts

    }

*return* render(request,'blog/blog-home.html', *context*=context)

<li><a href="{% url 'blog:b-cat' cat=cat.name %}">{{cat.name}},</a></li>

{% else %}

<li><a href="{% url 'blog:b-cat' cat=cat.name %}">{{cat.name}}</a></li>

در حالت دوم بجای اینکه فانکشنی جدا ساخته شود درون خود فانکشن بلاگ هوم این اتفاقات بصورت زیر تعریف میشوند

دسترسی به یک ویو با یو ار ال های مختلف

app\_name = 'blog'

urlpatterns = [

    path('', b\_home, *name*= 'blog-home'),

    path('<int:pid>', b\_single, *name*='blog-single' ),

    path('category/<str:cat>', b\_home, *name*='blog-home' ),

]

def b\_home(*req*, *cat*=None):

    posts=post.objects.filter(*status*=1)

*if* cat:

        posts = posts.filter(*category\_\_name*=cat)

    context = {

        'posts':posts

    }

*return* render(req,'blog/blog-home.html', *context*=context)

<li><a href="{% url 'blog:blog-home' cat=cat.name %}">{{cat.name}},</a></li>

{% else %}

<li><a href="{% url 'blog:blog-home' cat=cat.name %}">{{cat.name}}</a></li>

فیلتر کردن پست ها بر اساس نام نویسنده

در این حالت هم مانند دسته بندی بر اساس کتگوری میباشد و با یک شرط و یک مسیر جدا پست های آن نویسنده را فیلتر میکنیم

path('blog/<str:username>', b\_home, *name*='author' ),

]

def b\_home(*req*, *cat*=None, *username*=None):

    posts=post.objects.filter(*status*=1)

*if* cat:

        posts = posts.filter(*category\_\_name*=cat)

*if* username:

        posts = posts.filter(*author\_\_username*=username)

    context = {

        'posts':posts

    }

*return* render(req,'blog/blog-home.html', *context*=context)

<a href="{% url 'blog:author' username=post.author.username %}">{{post.author.first\_name}}</a>

دسته بندی بر اساس فرم جستجو

urlpatterns = [

    path('', b\_home, *name*= 'blog-home'),

    path('<int:pid>', b\_single, *name*='blog-single' ),

    path('category/<str:cat>', b\_home, *name*='blog-home' ),

    path('blog/<str:username>', b\_home, *name*='author' ),

    path('search/', search, *name* = 'search')

]

def search(*request*):

    posts = post.objects.filter(*status*=1)

*if* key := request.GET.get('search'):

        posts = posts.filter(*content\_\_contains*=key)

    context = {

        'posts':posts

    }

*return* render(request, 'blog/blog-home.html', *context*=context)

<form class="search-form" action="{% url 'blog:search' %}">

صفحه بندی در صفحه بلاگ هوم

همانطور که میدانیم پست های ما ممکن است زیاد شود و اگر قرار باشد همگی در یک صفحه باشند صفحه بسیار دراز و طولانی میشود لذا میتوانیم این موضوع را کنترل کنیمpagintion ما از طریق

جهت استفاده از این ویژگی نیاز است که این ما؟ژول در ویو ها ایمپورت شوند

From django.core.paginator import Paginator

*from* django.shortcuts *import* render, get\_object\_or\_404

*from* .models *import* post

*from* django.core.paginator *import* Paginator, EmptyPage, PageNotAnInteger

def b\_home(*req*, *cat*=None, *username*=None):

    posts = post.objects.filter(*status*=1)

*if* cat:

        posts = posts.filter(*category\_\_name*=cat)

*if* username:

        posts = posts.filter(*author\_\_username*=username)

    posts = Paginator(posts,3)

*try* :

        page\_number = req.GET.get('page')

        posts = posts.get\_page(page\_number)

*except* PageNotAnInteger:

        posts = posts.get\_page(1)

*except* EmptyPage:

        posts = posts.get\_page(1)

    context = {

        'posts':posts

    }

*return* render(req,'blog/blog-home.html', *context*=context)

<ul class="pagination">

                        <li class="page-item">

                            {% if posts.has\_previous %}

                            <a href="?page={{post.previous\_page\_number}}" class="page-link" aria-label="Previous">

                                <span aria-hidden="true">

                                    <span class="lnr lnr-chevron-left"></span>

                                </span>

                            </a>

                            {% endif %}

                        </li>

                        {% for i in posts.paginator.page\_range %}

                            {% if posts.number == i %}

                                <li class="page-item active"><a href="?page={{i}}" class="page-link">{{i}}</a></li>

                            {% else %}

                                <li class="page-item"><a href="?page={{i}}" class="page-link">{{i}}</a></li>

                            {% endif %}

                        {% endfor %}

                        <li class="page-item">

                            {% if posts.has\_next %}

                            <a href="?page={{post.next\_page\_number}}" class="page-link" aria-label="Next">

                                <span aria-hidden="true">

                                    <span class="lnr lnr-chevron-right"></span>

                                </span>

                            </a>

                            {% endif %}

                        </li>

Django form

newsletterگاها ما اطلاعاتی را از طریق صفحات میخواهیم از کاربر بگیریم و در دیتا بیس ذخیره کنیم . مانن تماس با ما یا بخش

در این گونه موارد ما از فرم ها در صفحات استفاده میکنیم و بجاب استفاده از متو گت که واکشی از پایگاه داده است از متود پست و استفاده میکنیمcsrf-token رمزنگاری

یک مثال ساده از فرم ها

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>Document</title>

</head>

<body>

    <form action="{% url 'my\_weblog:test' %}" method="post">

        {% csrf\_token %}

        <lable>email</lable><br>

        <input name="email" type="email">

        <button type="submit">submit</button>

    </form>

</body>

</html>

urlpatterns = [

    path('', home, *name*= 'home'),

    path('about', about, *name*= 'about'),

    path('contact', Contact, *name*= 'contact'),

    path('contact\_us', contact\_us, *name*= 'contact\_us'),

    path('test', test, *name*= 'test'),

]

def test(*req*):

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

        news = newsletter()

        news.email = req.POST.get('email')

        news.save()

*return* render(req, 'my\_weblog/test.html')

این روش در فرم ها در حالت عادی میباشد که نمیتوان گفت کار درست یا غلطی است اما جنگو برای فرم ها سولوشن های خودش را ارائه میدهد

Forms & Model forms

در پوشه اتپلیکیشنی میباشد که آن صفحه در آن اپلیکیشن قرار دارد Forms.py قدم اول ساخت فایل

میباشد و لذا فایل را د آن اپلیکیشن میسازیم My\_weblog در اپلیکیشن Contact us برای پروژه ما صفحه

*from* django *import* forms

class news\_letter(forms.Form):

    name = forms.CharField(*max\_length*=255)

    email = forms.EmailField()

def test(*req*):

    froms = news\_letter()

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

        news = newsletter()

        news.email = req.POST.get('email')

        news.save()

    context = {

        'forms':froms

    }

*return* render(req, 'my\_weblog/test.html', *context*=context)

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta http-equiv="X-UA-Compatible" content="IE=edge">

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

    <title>Document</title>

</head>

<body>

    <form action="{% url 'my\_weblog:test' %}" method="post">

        {% csrf\_token %}

        {{forms}}

        <button type="submit">submit</button>

    </form>

</body>

</html>

در داکیومنت جنگو این رفتار به صورت زیر تعریف میشود

را به عنوان آرگومان بهش میدهیم Req.POST وقتی آبجکتی را از روی این کلای قرم ساختیم مقدار

ایجاد فرم با استفاده از پایگاه داده

Django model form

ایجاد یک فرم بر اساس یک مدل ساخته شده

)contactform((ایجاد یک کلاس با اسمی مناسب برای فرم

ایجاد یک متا کلاس در کلاس فرم برای اینکه جنگو بداند این فرم بر اساس یک مدل در دیتا بیس است . لذا ان مدل نیز باید ایمپورت شوند

*from* django.forms *import* ModelForm

*from* .models *import* contact

class ContactForm(ModelForm):

    name = forms.CharField(*max\_length*=255)

    email = forms.EmailField()

    subject = forms.CharField(*max\_length*=255)

    text = forms.CharField(*widget*=forms.Textarea)

    class META:

        model = contact

        field = ['name', 'email', 'subject', 'text']

در قسمت فیلد گفته ایم کدام فیلد های آن مدل را میخواهیم تبدیل به فرم کنیم

    path('test', test, *name*= 'test'),

def test(*req*):

    forms = ContactForm()

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

        forms\_edit = ContactForm(req.POST)

*if* forms\_edit.is\_valid():

            forms\_edit.save()

    context = {

        'forms':forms

        }

*return* render(req, 'my\_weblog/test.html', *context*=context)

در این حالت نیاز به کلین دیتا نیست و میگوییم اگر دیتا های فرم مناسب بود آن را سیو کن که چون این فرم برای مکدل خاصی ساخته شده لذا داده هایش در دیتا بیس سیو میشود