جنگو سکشن 2

آنچه باید به آن اشراف داشت

و حداقل یک پروژه پیاده سازی شده Mvt , template , model , view , معماری

آنچه یاد میگیریم

Class base view ( cbv)

Django rest framework (drf)

Custum user model

Authentication api token / jwt

Creating test

Api documentation

Dockeryzing project

Deployment with docker

Cash and celery

Query optimization

Api load testing

And more …

در خود سایت گیت هاب یک پروژه جدید برایش ایجاد میکنیم . این قسمت برای ریپازیتوری یک پلن Project یک ریپازیتاری جدید ایجاد و سپس از قسمت

آماده میکند

را در طول کار پیش میبریم Todo , inprogress , done هستفااده میکنیم و سه بخش Board ما از قسمت

اگر یک ریپازیتوری به نام کاربری گیت هاب درست کنیم در صفحه اول نیز آن ریپازیتوری بصورت دیفالت نمایش داده میشومد

درست کنیم تبدیل به یک سایت رزومه میشودUsername.github.io اگر یک ریپازیتوری به اسم

را نیز دانلود میکنیمWsl2 ابتدا داکر دسکتاپ را دانلود میکنیم و از لینک زیر

بعد از نصب داکر دسکتاپ حتما باید در آن اکانت بسازیم

حتما باید از سرویس های فیل شکن یا تحریم گذر شکن استفاده کنیم

دانلود شده را نصب کنیم و ری استارت کنیم Wsl2 بعد از نصب وقتی باز میکنیم خطای کرنل میدهد که باید

و سیستم را ریست میکنیم

را با محتویات زیر در پروژه میسازیمDockerfile و سپس فایل

FROM python:3.9

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1

WORKDIR /app

COPY requirement.txt /app/

RUN pip3 install --upgrade pip

RUN pip3 install -r requirement.txt

COPY ./ /app/

را با محتویات زیر ساخته و سپس داکرکامپوز را با دستور زیر بیلد و آپ میکنیمDocker-compose.yml سپس فایل

version: "3.9"

services:

  backend:

    build: .

    container\_name: backend

    command: python manage.py runserver 0.0.0.0:8000

    volumes:

      - ./:/app

    ports:

      - "8000:8000"

    environment:

      - SECRET\_KEY=test

      - DEBUG=True

مد نظر باشد داکر دسکتاپ حتمن باید ران باشد

Docker-compose up –build

با دستورات زیر بر روی خود داکر کامپوز اپلیکیشن های مورد نظر یا هر اسکریپتی ران میکنیم

docker-compose exec your "container name" sh -c " your command"

برای متغیر های محیطی در ستینگdecouple همچنین با ماژول

from pathlib import Path

from decouple import config

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

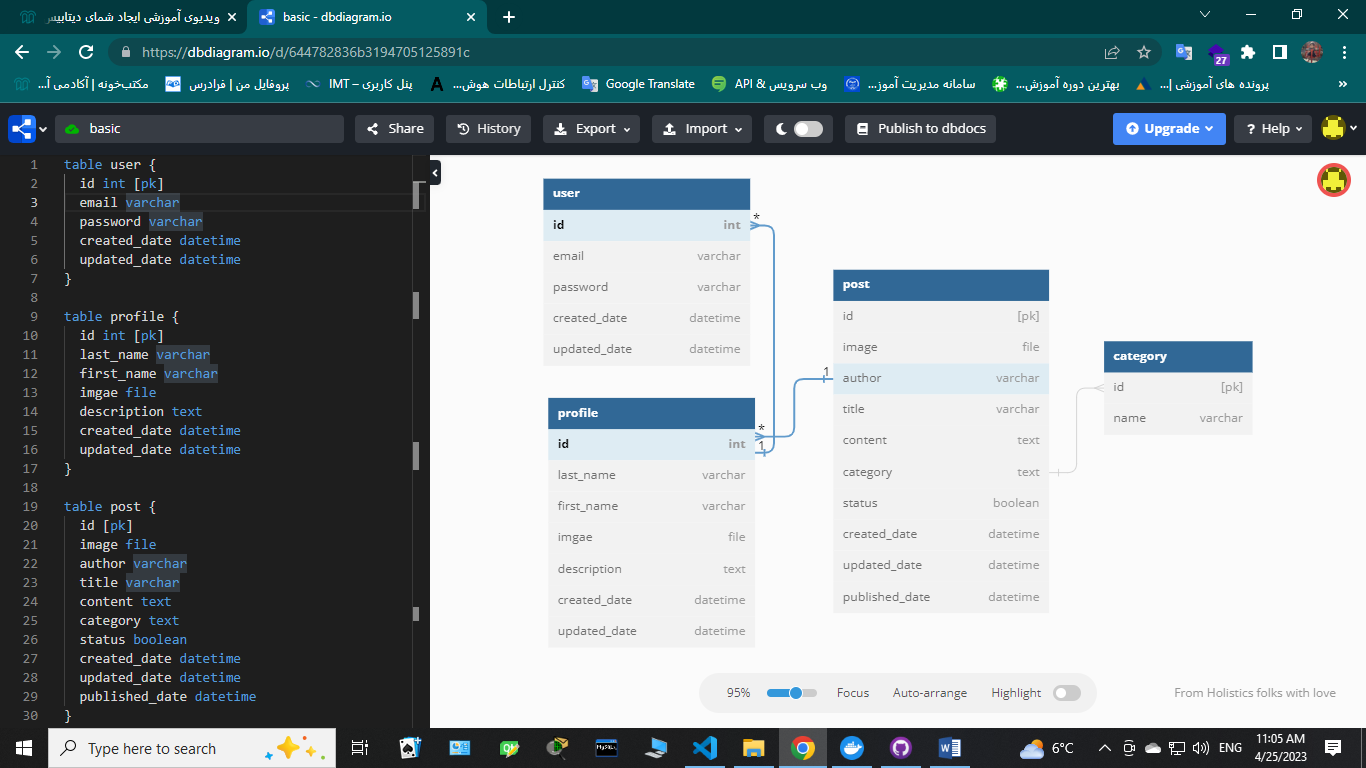
SECRET\_KEY = config('SECRET\_KEY')

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = config('DEBUG', cast=bool)

ALLOWED\_HOSTS = []

dbdiagram.ioشمای اولیه مددلاسیون با



برای ایجاد مدلاسیون اولیه پروژه در مدل کلاس های زیر برای اپ بلاگ ساخته شد

class Category(models.Model):

    name = models.CharField(max\_length=200)

    def \_\_str\_\_(self):

        return self.name

    class Meta:

        ordering = ['name']

class Post(models.Model):

    '''

    this is the model for post in blog application

    '''

    image = models.ImageField(null=True, blank=True)

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

    title = models.CharField(max\_length=200)

    content = models.TextField()

    status = models.BooleanField(default=False)

    category = models.ForeignKey(Category, on\_delete=models.SET\_NULL, null=True)

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

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

    published\_date = models.DateTimeField()

    def \_\_str\_\_(self):

        return self.title

    class Meta:

        ordering = ['-created\_date']

در اپ اکانت در مدل میخواهیم مدلاسیون کاستومایز شده یوزر هارو خودمان تامین کنیم

را داریم Abstractuser , abstractbaseuser دو حالت

https://testdriven.io/blog/django-custom-user-model/

در بیس یوزر این قابلیت وجود دارد که از پایه همه چیز را خومان بنویسیم

from django.db import models

from django.contrib.auth.models import (AbstractBaseUser,BaseUserManager,PermissionsMixin)

# Create your models here.

class UserManager(BaseUserManager):

    '''

    Custom user model manager

    '''

    def create\_user(self,email,password,\*\*extra\_fields):

        if not email:

            raise ValueError('Users must have an email address')

        user = self.model(email=self.normalize\_email(email),\*\*extra\_fields)

        user.set\_password(password)

        user.save()

        return user

    def create\_superuser(self,email,password,\*\*extra\_fields):

        extra\_fields.setdefault('is\_staff',True)

        extra\_fields.setdefault('is\_superuser',True)

        extra\_fields.setdefault('is\_active',True)

        user = self.create\_user(email,password)

        user.is\_staff = True

        user.is\_superuser = True

        user.save()

        if extra\_fields.get('is\_staff') is not True:

            raise ValueError('Superuser must have is\_staff=True.')

        if extra\_fields.get('is\_superuser') is not True:

            raise ValueError('Superuser must have is\_superuser=True.')

        return self.create\_superuser(email,password,\*\*extra\_fields)

class User(AbstractBaseUser, PermissionsMixin):

    '''

    Custom user model

    '''

    email = models.EmailField(max\_length=250,unique=True)

    is\_superuser = models.BooleanField(default=False)

    is\_staff = models.BooleanField(default=False)

    is\_active = models.BooleanField(default=True)

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

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

    USERNAME\_FIELD = 'email'

    REQUIRED\_FIELDS = []

    manager = UserManager()

    def \_\_str\_\_(self):

        return self.email

    class Meta:

        verbose\_name\_plural = "Users"

        verbose\_name = "User"

        ordering = ["-created\_date"]

در قسمت یوزر نیم فیلد میگوییم ایمیل همان یوزر نیم است زیرا اساسا جنگو بر اساس یوزر نیم و پسورد وریفای میکند

زمانی که یک درخواستی برای یک یوزر میفرستیم منیجر یک سری از المان ها رو کنترل میکندMANAGER

در انتها در تنظیمات پروژه مقدار زیر اعمال میشود

AUTH\_USER\_MODEL = "accounts.User"

Makemigrations , migrate,createsuperuserحال اقدام میکنیم به

مد نظر باشد زمانی که از کاستوم یوزر مدل ها استفاده میکنیم تا قبل از ایجاد مدل یوزر نباید مایگریت و میک مایگرت کنیم

ADMINS.PYتنظیمات

from django.contrib import admin

from .models import User

from django.contrib.auth.admin import UserAdmin

class CustomUserAdmin(UserAdmin):

    model = User

    list\_display = ( 'email' , 'is\_staff', 'is\_active')

    list\_filter = ('is\_active',)

    ordering = ['is\_staff']

    fieldsets = (

        ('authentication',

            {'fields': ('email', 'password')}

        ),

        ('Permissions',

            {'fields': ('is\_staff', 'is\_active', 'is\_superuser')}

        ),

        ('group permissions',

            {'fields': ('groups', 'user\_permissions',)}

        ),

        ('important date',

            {'fields': ('last\_login',)}

        ),

    )

    add\_fieldsets = (

        ("authentication",

            {"fields": ("email", "password1","password2")}

        ),

        ("Permissions",

            {"fields": ("is\_staff", "is\_active", "is\_superuser")}

        ),

    )

admin.site.register(User, CustomUserAdmin)

در اینجا چون تنظیمات پنل ادمین دیگر مدل یوزر خود جنگو نیست ما باید در تنظیمات نمایشش بگوییم برای کدام مدل است

ایجاد مدلاسیون پروفایل

class Profile(models.Model):

    '''

    profile model

    '''

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

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

    first\_name = models.CharField(max\_length=255)

    last\_name = models.CharField(max\_length=255)

    image = models.ImageField(null=True,blank=True)

    description = models.TextField(null=True,blank=True)

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

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

    def \_\_str\_\_(self):

        return self.user.email

وقتی این مدل ایجاد شد و در ادمین رجیستر شد و فرایند مایگریت و میک مایگریشن انجام شد ما باید دستی داده هایش را پر کنیم اما میخواهیم این ئپنل اتوماتیک باشد و وقتی یوزر جدیدی ساخته میشود لینک باشد با پروفایل و داده های پروفایلش تکمیل شود

این کار توسط سیگنال انجام میشود

است که میگوییم فلان آبجکت را زمانی بساز که در کلاس بالادستی ذخیره شود بعد ارسال شود Post\_save یک نمونه از این سیگنالها سیگنال

لذا در مدلی که پروفایل در آن قرار دارد بصورت زیر عمل میکنیم

from django.db.models.signals import post\_save

from django.dispatch import receiver

@receiver(post\_save,sender=User)

def create\_user\_profile(sender,instance,created, \*\*kwargs):

    if created:

        Profile.objects.create(user=instance)

مانند دوره مقدمماتی Static , mediaایجاد ساختار

فراخوانی مدل کاستومایز شده یوزر در اپلیکیشن بلاگ

در زمانی که می‌خواهید از مدلاسیون شخصی استفاده کنین می‌بایست بتوانید شئ نگهدارنده آن را فراخوانی کنین که برای این کار 3 روش وجود دارد

کردن مدل مربوطه از اپ مشخص، که به شکل زیر انجام می‌شود import

from accounts import User

class ModelName(models.Model):

author = models.ForeignKeyField(User, … )​

contrib استفاده از تابع فراخوانی مدولاسیون یوزر در

from django.contrib.auth import get\_user\_model

User = get\_user\_model()

class ModelName(models.Model):

author = models.ForeignKeyField(User, … )

setting.py   استفاده از شئ نگهدارنده مدولاسیون در

from django.conf import settings

User = settings.AUTH\_USER\_MODEL

class ModelName(models.Model):

author = models.ForeignKeyField(User, … )

که بیشتر از روش دوم برای این کار استفاده می‌شود چرا که داینامیک تر است.

classbaseviewsاستفاده از

با توجه به اینکه فانکشن ویو میتواند کمی شلوغ و پیچیده باشد کلاس بیس میتواند کمی ساختار ساده تر و از طرفی بخاطر توانایی کلاس ها در ارث بری میتواند امکانات بیشتری به وویو ما اضافه کند و اتلاف وقت کمتری دارد

در حالت های قبل ما یک ویو را بصورت یک تابع مینوشتیم و با ارسال رکوئستی آن را اجرا میکردیم

بصورت مستقیم ویویی را صدا کرد و به آن کانتکسی را ارسال کرد Urlpatterns های Path در ابتدایی ترین حالت کلاس بیس میتوان در یکی از

در ابتدا در پروژه روت تمپلیت و فولدر تمپلیت ها را مشخص میکنیم مانند دوره مقدماتی

from django.urls import path

from .views import \*

from django.views.generic import TemplateView

urlpatterns = [

    path('', test, name='test'),

    path('cbv', TemplateView.as\_view(

    template\_name='blog/cbv.html',extra\_context={'name':'hamid'}), name='cbv'

    ),

]

در حالت تابع تست درون ویو یک فانکشن ساخته شده و فراخوانی شده که میتوانست کانتکس خودش را هم داشته باشد

تعریف شدهUrls.py در حالت کلاس بیس از تمپلیت ویو استفاده شده و کانتکس و تمپلیت مستقیم در همان

میخواهیم تمپلیت ویو را خودمان بنویسیم

from django.shortcuts import render

from django.views.generic import TemplateView

from .models import Post

# Create your views here.

def test(req):

    return render(req, 'blog/blog.html')

class BlogView(TemplateView):

    template\_name = 'blog/cbv.html'

    def get\_context\_data(self, \*\*kwargs):

        context = super().get\_context\_data(\*\*kwargs)

        context['posts'] = Post.objects.all()

        return context

تابع گت کانتکس مفهومش این است که برو از روی والد تابع کانتکس را بصورت خام بساز و حالا با مقادیری که بهت میدهم این دیکشنری را تکمیل کن

from django.urls import path

from .views import \*

#from django.views.generic import TemplateView

urlpatterns = [

    path('', test, name='test'),

    #path('cbv', TemplateView.as\_view(

    #template\_name='blog/cbv.html',extra\_context={'name':'hamid'}), name='cbv'

    #),

    path('cbv',BlogView.as\_view(), name='cbv'),

]

RedirectViewریدایرکت ویو

مفهوم ریدایرکت ویو ارجاع کاربر به یک صفحه یا یو ار ال خاص بعد از انجام یک اتفاقی میباشد

در حالت بیسیک

from django.urls import path

from .views import \*

from django.views.generic import RedirectView

#from django.views.generic import TemplateView

app\_name = 'blog'

urlpatterns = [

    path('', test, name='test'),

    #path('cbv', TemplateView.as\_view(

    #template\_name='blog/cbv.html',extra\_context={'name':'hamid'}), name='cbv'

    #),

    path('cbv',BlogView.as\_view(), name='cbv'),

    path('cbv2',RedirectView.as\_view(url='https://google.com'), name='cbv'),

]

from django.urls import path

from .views import \*

from django.views.generic import RedirectView

#from django.views.generic import TemplateView

app\_name = 'blog'

urlpatterns = [

    path('', test, name='test'),

    #path('cbv', TemplateView.as\_view(

    #template\_name='blog/cbv.html',extra\_context={'name':'hamid'}), name='cbv'

    #),

    path('cbv',BlogView.as\_view(), name='cbv'),

    #path('cbv2',RedirectView.as\_view(url='https://google.com'), name='cbv'),

    path('cbv2',RedirectView.as\_view(url='/blog'), name='cbv'),

]

حال این کلاس را کاستومایز میکنیم

Code spell checkerمعرفی افزونه

در این حالت ما کلاسش را در ویو تعریف و از کلاس مادر تابع را اوررایت کرده و فقط با مقدار یو ار ال انتخابی خودمان ریدایرکتش میکنیم

from django.urls import path

from .views import \*

#from django.views.generic import RedirectView

#from django.views.generic import TemplateView

app\_name = 'blog'

urlpatterns = [

    path('', test, name='test'),

    #path('cbv', TemplateView.as\_view(

    #template\_name='blog/cbv.html',extra\_context={'name':'hamid'}), name='cbv'

    #),

    path('cbv',BlogView.as\_view(), name='cbv'),

    #path('cbv2',RedirectView.as\_view(url='https://google.com'), name='cbv'),

    #path('cbv2',RedirectView.as\_view(url='/blog'), name='cbv'),

    path('cbv2',RedirectToBlog.as\_view(), name='cbv2'),

]

from django.shortcuts import render, redirect

from django.views.generic import TemplateView, RedirectView

from .models import Post

def test(req):

    return render(req, 'blog/blog.html')

class BlogView(TemplateView):

    template\_name = 'blog/cbv.html'

    def get\_context\_data(self, \*\*kwargs):

        context = super().get\_context\_data(\*\*kwargs)

        context['name'] = 'hamid'

        context['posts'] = Post.objects.all()

        return context

class RedirectToBlog(RedirectView):

    url = '/blog/'

    def get\_redirect\_url(self, \*args, \*\*kwargs):

        return super().get\_redirect\_url(\*args, \*\*kwargs)

listviewمعرفی لیست ویو ها

این کلاس و ساختار لیستی از آبجکت های یک مدل را کامل در صفحه وارد میکند

Post\_list.html باشد مثلا Mode name\_list.html تمپلیت مربوط به این ویو به صورت دیفالت اسمش باید

به صفحه ارسال میشوند Object\_list از طرفی بجای اینکه مقادیر با نام پست به صفحه ارسال شوند بصورت دیفالت با نام

در حالت بیسیک این ساختار به صورت زیر است

class PostListView(ListView):

    model = Post

app\_name = 'blog'

urlpatterns = [

    path('', test, name='test'),

    path('cbv',BlogView.as\_view(), name='cbv'),

    path('cbv2',RedirectToBlog.as\_view(), name='cbv2'),

    path('post/',PostListView.as\_view(), name='post'),

]

مد نظر باشد برای رکوئست به پست از اسلش استفاده کرده ایم . زمانی که از لیست ویو ها استفاده میکنیم باید این موضوع را رعایت کنیم

میتوانیم به جای اتربیوت مدل از حالت زیر استفاده کنیم که آنگاه میتوان از فیلتراسیون نیز استفاده کرد

class PostListView(ListView):

    #model = Post

    queryset  = Post.objects.all()

    context\_object\_name = 'posts'

استفاده کن posts به صفحه از نام object\_list همچنین گفته ایم بجای ارسال نام

نیز استفاده کنیم Get\_query طبق حالت زیر میتوانیم از تابع

class PostListView(ListView):

    #model = Post

    #queryset  = Post.objects.all()

    context\_object\_name = 'posts'

    def get\_queryset(self):

        posts =  Post.objects.filter(status=True)

        return posts

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

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

        <title></title>

        <meta name="description" content="">

        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="">

    </head>

    <body>

        {% for post in posts %}

            <h3>{{post.title}}</h3>

            <p>{{post.content}}</p>

        {% endfor %}

    </body>

</html>

ها listview در pagination استفاده از

درکلاس لیست ویو زیر گفته ام که در هر صفحه 2 پست به من نشان بده

برایم ارسال کن ?page=2 داکیومنت جنگو میگوید برای رفتن به صفحه بعد پارامتری را از طریق

class PostListView(ListView):

    #model = Post

    #queryset  = Post.objects.all()

    context\_object\_name = 'posts'

    paginate\_by = 2

    def get\_queryset(self):

        posts =  Post.objects.filter(status=True)

        return posts

ارسال شود و پستی نباشد خودش 404 میدهد ?page=5 حال اگر مثلا از طریق

برای ساختن لینک هایش نیز به صورت زیر عمل میکنیم

به صفحه ارسال میشوند Page\_obj مد نظر باشد این پیج ها از طریق آبجکتی به نام

from django.shortcuts import render, redirect

from django.views.generic import TemplateView, RedirectView , ListView

from .models import Post

def test(req):

    return render(req, 'blog/blog.html')

class BlogView(TemplateView):

    template\_name = 'blog/cbv.html'

    def get\_context\_data(self, \*\*kwargs):

        context = super().get\_context\_data(\*\*kwargs)

        context['name'] = 'hamid'

        context['posts'] = Post.objects.all()

        return context

class RedirectToBlog(RedirectView):

    url = '/blog/'

    def get\_redirect\_url(self, \*args, \*\*kwargs):

        return super().get\_redirect\_url(\*args, \*\*kwargs)

class PostListView(ListView):

    context\_object\_name = 'posts'

    paginate\_by = 1

    def get\_queryset(self):

        posts =  Post.objects.filter(status=True)

        return posts

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

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

        <title></title>

        <meta name="description" content="">

        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="">

    </head>

    <body>

        {% for post in posts %}

            <h3>{{ post.title }}</h3>

        {% endfor %}

        {% if page\_obj.has\_previous %}

            <a href="?page={{ page\_obj.previous\_page\_number }}">Previous Page</a>

        {% endif %}

        {% if page\_obj.has\_next %}

                <a href="?page={{ page\_obj.next\_page\_number }}">Next Page</a>

        {% endif %}

    </body>

</html>

Ordering معرفی

class PostListView(ListView):

    queryset = Post.objects.all()

    context\_object\_name = 'posts'

    paginate\_by = 4

    ordering = 'id'

    #def get\_queryset(self):

    #    posts =  Post.objects.filter(status=True)

    #    return posts

میتوانم بگویم برای نمایش پست ها در صفحه اردرینگ بر اساس چه چیزی باشد

اما مد نظر باشد وقتی اردرینگ استفاده میکنیم دیگر از تابع کوئری ست استفاده نکنیم و خود اتربیوتش را باید استفاده کنیم

detailviewمعرفی

باشدpk مد نظر باشد آرگومان ارسالی از بلاگ هوم بعنوان پارامتر ارسالی حتما نامش باید پ

باشد Model name\_detail.html پیج بلاگ سینگل اسمش حتما باید

آن پست را به صفحه ارسال میکند که مانند بلاگ هوم میتوان نام ارسالی را عوض کرئ Object همچنین به صورت پیش فرض خودش با نام

در کلاس نیز تمپلیت دلخواه را گذاشت Template\_name همچنین میتوان با گزینه

from django.urls import path

from .views import \*

#from django.views.generic import RedirectView

#from django.views.generic import TemplateView

app\_name = 'blog'

urlpatterns = [

    path('',PostListView.as\_view(), name='blog-home'),

    path('post/<int:pk>/',PostDetailView.as\_view(), name='blog-single'),

    ]

from django.shortcuts import render, redirect

from django.views.generic import TemplateView, RedirectView , ListView, DetailView

from .models import Post

class RedirectToBlog(RedirectView):

    url = '/blog/'

    def get\_redirect\_url(self, \*args, \*\*kwargs):

        return super().get\_redirect\_url(\*args, \*\*kwargs)

class PostListView(ListView):

    queryset = Post.objects.all()

    context\_object\_name = 'posts'

    paginate\_by = 2

    ordering = 'id'

class PostDetailView(DetailView):

    model = Post

    context\_object\_name = 'post'

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

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

        <title></title>

        <meta name="description" content="">

        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="">

    </head>

    <body>

            {{post.title}}

    </body>

</html>

FormViewمعرفی

این کلاس زمانی ایجاد میشود که بخواهیم دیتایی را از کاربر بگیریم

برایش اوررایت نشود عملا کار خاصی نمیکندForm\_valid همچنین اگر متود

from django import forms

from .models import Post

class PostCreateForm(forms.ModelForm):

    class Meta:

        model= Post

        fields = ['author','title','content','status','category','published\_date']

from django.urls import path

from .views import \*

app\_name = 'blog'

urlpatterns = [

    path('',PostListView.as\_view(), name='blog-home'),

    path('post/<int:pk>/',PostDetailView.as\_view(), name='blog-single'),

    path('post-create/',PostCreateView.as\_view(), name='post-create'),

    ]

class PostCreateView(FormView):

    template\_name = "home/create\_post.html"

    form\_class = PostCreateForm

    success\_url = "/blog/"

    def form\_valid(self, form):

        form.save()

        return super().form\_valid(form)

<!DOCTYPE html>

<html>

    <head>

        <meta charset="utf-8">

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

        <title></title>

        <meta name="description" content="">

        <meta name="viewport" content="width=device-width, initial-scale=1">

        <link rel="stylesheet" href="">

    </head>

    <body>

            <form method='post'>

                {% csrf\_token %}

                {{form.as\_p}}

                <button type='submit'>send</button>

            </form>

    </body>

</html>