دانشکده فنی و حرفه ای دختران تهران
دکتر شریعتی

کاردانی رشته‌ی نرم‌افزار

موضوع پروژه:
وبلاگ

استاد راهنما:
بهناز آقابالایی

نام دانشجو :
عاطفه خضرایی دوست زارع

نیمسال دوم 99-1398

تشکر و قدردانی:

کمال تشکر و قدردانی را از استاد عزیزم ، دارم که مرا راهنمایی کردند.

چکیده

هر چند که وبلاگ موضوعی ساده است ، اما قابل انکار نیست که حجم عظیمی از اینترنت را بلاگ ها تشکیل می دهند . بلاگ محیطی است برای افراد ، تا بتوانند در آن نظرات ، خاطرات و دانسته ها ، علایق خود و ... را با مخاطبانشان در میان بگذارند . بلاگر ها می توانند با محتوای تولید شده ی خود بسیار تاثیر گذار باشند ... این روز ها با آمدن رسانه های اجتماعی ، شاید بلاگ ها کمرنگ تر دیده شوند . شاید اینگونه به نظر برسد که در این حجم از هیاهو های فضای مجازی ، نویسنده ها و بلاگر ها دیگر جایی ندارند برای ماندن .... ولی شاید ما برنامه نویس ها باید برای آن ها خانه ای امن بسازیم ... ما باید کیفیت ، سرعت ، امنیت ، ... را برای وبلاگ ها بیاوریم . چرا که در فضای اینترنت ، محتوای با ارزش ، محتوای با کیفیت ، بی نهایت ارزش دارد و تولید کننده های این محتوا ها ، لیاقت بهترین ها را دارند . من در این پروژه سعی کردم با استفاده از زبان پایتون و فریم ورک قدرتمند جنگو ، سیستم یک وبلاگ را طوری بسازم که از لحاظ سرعت ، امنیت و کیفیت Back end ، چیزی از یک سایت بزرگ کم نداشته باشد .

فهرست مطالب

عنوان **صفحه**

فهرست اشکال

عنوان **صفحه**

فصل1

تعریف مسئله

1-1. تعریف وبلاگ :

وبلاگ سیستمی ساده است که استفاده از اینترنت را برای همگان ساده تر می کند . در واقع ابزاری ساده اما کاربردی برای تولید محتوا است . مخاطبان خود ، و کاربران خاص خود را دارد .

از نظر مفهومی تفاوت وبلاگ و وب سایت را می توان اینگونه در نظر گرفت:

- وبلاگ برای تولید محتوا و اطاع رسانی ساخته می شود و در بازه های مشخص و نا مشخص آپدیت می شود . در وبلاگ بر عکس وب سایت ها ، نیازی به ارائه خدمات نیست . هر چه هست ، تولید محتوا است . اما وب سایت ها برای ارائه خدمات مختلف ساخته می شوند و از تولید محتوا برای SEO خود استفاده می کنند.

اما از نظر نرم افزاری و پیاده سازی تعاریف زیر را می توان ارائه داد:

-وبلاگ مدلی ساده است ، برای تولید و نگهداری محتوا در اینترنت . از خیلی جهات شبیه به یک وب سایت است . تنها تفاوت در ارائه خدمات است . وبلاگ می تواند قدمی باشد در راه کالبد شکافی کردن وب سایت های پیچیده تر از لحاظ کد نویسی و پیاده سازی .

2-1. اهداف تحقیق

همانطور که در بالا گفته شد ، هدف من از انجام این پروژه ، طراحی مدلی از وبلاگ بود کـه از لحاظ BackEnd و دیتابیس ، سرعت و امنیت ، چیزی از وب سایت های بزرگ کم نداشته باشد . همچنین دوست داشتم برای درک کردن هرچه بهتر کد نویسی پروژه های بزرگ تر و پیچیده تر ، ابتدا یک پروژه کوچک با تکنولوژی های روز و فقط و فقط با کد نویسی ، پیاده سازی کنم تا با امکانات فریم ورک جنگو آشنا بشوم و بتوانم یک قدم به آرزوی برنامه نویس خـوب شـدن نزدیک تر بشوم.

فصل2

ابزار های بکار رفته

2- sqlite3

این نرم‌افزار مشهورترین سیستم ذخیره فایلی اطلاعات به شمار می‌رود. شهرت SQLite به دلیل پشتیبانی از انواع مختلف سیستم‌عامل‌ها از جمله ویندوز، لینوکس، آندروئید و مک او اس و همچنین رایگان و قدرتمند بودن آن است.

SQLite با زبان C برنامه‌نویسی شده است و به طور پیوسته در حال بهبود و توسعه است. به همین دلیل سرعت و کارایی بسیار بالایی دارد. در نگارش‌های جدید که در آینده منتشر خواهند شد، بهینه‌سازی‌های گسترده‌ای روی این سیستم به انجام رسیده است که سرعت عملکرد آن را بیش از پیش افزایش داده است.

SQLite نرم‌افزاری با مجوز استفاده Public Domain است. به این معنی که حق مالکیتی ندارد و هر فرد یا سازمانی می‌تواند بدون هیچ محدودیتی از آن به هر شکلی استفاده کند.

2-1-1. تاریخچه

دکتر ریچارد هیپ اس‌کیوال لایت را در سال ۲۰۰۰ در زمانی که از طریق جنرال

داینامیکس با نیروی دریایی ایالات متحده آمریکا کار می‌کرد طراحی کرد. او در آن زمان مشغول کار روی برنامه‌های مربوط به ناوشکن‌هایی بود که به موشک‌های هدایت‌شونده مجهز بودند و تا آن زمان از پایگاه داده آی‌بی‌ام اینفورمیکس (به انگلیسی: IBM Informix) استفاده می‌کردند. هدف از ساخت اس‌کیوال لایت این بود که این برنامه‌ها بتوانند بدون نصب یا مدیریت پایگاه داده مستقل اجرا شوند. نسخهٔ اول نرم‌افزار در اوت ۲۰۰۰ انتشار یافت. در نسخهٔ ۲/۰ ساختار داخلی‌ی اس‌کیوال لایت تغییر یافت و از یک درخت بی در آن استفاده گردید. در نسخه ۳/۰ که قسمتی از هزینه‌هایش توسط ای‌اوال تأمین شد، پشتیبانی چندزبانی و چند تغییر بزرگ دیگر در اس‌کیوال لایت رخ داد.

2-2. زبان برنامه نویسی python

python یک زبان برنامه نویسی مفسری، چندمنظوره و سطح بالاست که از شی گرایی و برنامه نویسی ساختار یافته به طور کامل پشتیبانی می‌کند.

از این زبان برنامه نویسی به طور گسترده در دنیا استفاده می‌شود و برای آن فرقی نمی‌کند که هدف شما از استفاده آن ایجاد وب اپلیکیشن و برنامه نویسی دسکتاپ است و یا حتی برنامه نویسی هوش مصنوعی و یادگیری ماشینی، این زبان به بهترین نحو از عهده تمام آن‌ها بر

خواهد آمد و به جرات می‌توان ادعا کرد که در دیگر زمینه‌های برنامه نویسی شما را تنها نخواهد گذاشت. برای اینکه بدانید که مهمترین ویژگی‌های پایتون چیست که آن را به چنین زبان قدرتمندی تبدیل کرده است، باید با ساختار آن آشنا شوید. برای پیاده سازی این پروژه از فریم ورک جنگو که یکی از قدرت مند ترین فریم ورک های برنامه نویسی وب بری پایتون است ، استفاده کرده ام .

فصل3

پایگاه داده

3-1. مقدمه

پایگاه داده یا همان Database ، مجموعه ای سازمان داده شده از اطلاعات است که به شکل جداول ذخیره می شوند. برای سازمان دهی این اطلاعات روشهای متفاوتی وجود دارد که هدف تمامی آنها فراهم کردن روشهایی مناسب برای سهولت در برقراری ارتباط با پایگاه های داده و استفاده از اطلاعات موجوددر آنها است. سیستم مدیریت MVT در فریم ورک جنگو ، مکانیزمی را جهت ذخیره سازی و بازیابی اطلاعات در پایگاه داده فراهم می نماید. در حقیقت باعث می شود تا برنامه نویس بدون نگرانی در باره چگونگی ذخیره سازی داده ها در پایگاه داده و ساختار آنها ، به اطلاعات دسترسی پیدا کند و بتواند داده های جدید را در آن ذخیره نماید.

فصل4

توضیح کد

4-1. ایجاد یک APP

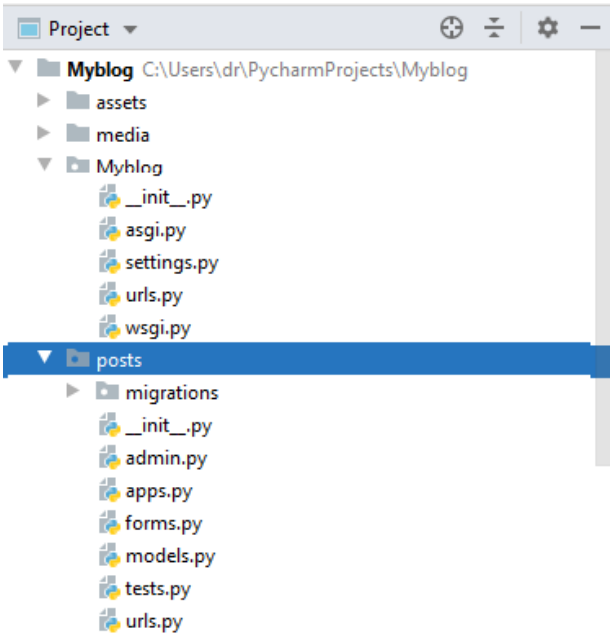
بر در پروژه های جنگو ، قسمت های مختلف سایت به برنامه های کوچک تر دسته بندی می شوند که به آن ها App می گویند . دلیل انجام این کار این است که تمام تخم مرغ های مان را در یک سبد نگذاریم و نگرانی های مان را به شکل منطقی تقسیم کنیم . هدف این است که اگر بخشی از سایت به مشکل بخورد فقط همان بخش کار نکند.

برای ایجاد یک App در terminal دستور زیر را اجرا می کنیم

```
Python manage.py startapp نامapp
```

در این پروژه ، من فقط APP برای پست گذاری یک وبلاگ را ساخته ام و نام آن را posts گذاشته ام

4-2. فایل های مهم پروژه ی جنگو



جنگو فایل هایی را به صورت پیش فرض به صورت خود کار می سازد که ما آن ها را تغییر نمی دهیم و فقط از آن ها استفاده می کنیم . فایل هایی که برای پروژه ی من لازم بود باهاشون کار بشود عبارت اند از:

Setting.py : تنظیمات مهم برنامه مانند دیتابیس ، برنامه های نصب شده ، تمپلیت و... در اینجا تعریف می شوند.

urls.py : این فایل مشخص می کنه چی کجاست . برای آدرس دهی برنامه مون استفاده می شود . به صورت پیش فرض برای برنامه اصلی جنگو ساخته می شود اما بهتر است برای App هامون خودمون یک فایل urls.py بسازیم . اینطوری آدرس دهی هر برنامه داخل خودش تعریف میشه و در فایل اصلی فقط آدرس app هامون رو تعریف می کنیم .

admin.py : جنگو به صورت اتومات برای وب سایت ما صفحه ی admin می سازد . در این فایل ما می توانیم صفحه ی admin را شخصی سازی کنیم و امکانات مد نظر خودمون را در آن اضافه یا کم کنیم .

models.py : یکی از مهم ترین فایل هایی که مدام باهاش سر و کار داریم هست . این فایل برای تعریف کلاس ها و ارتباط با دیتابیس استفاده می شود .

Setting.py : تنظیمات مهم برنامه مانند دیتابیس ، برنامه های نصب شده ، تمپلیت و... در اینجا تعریف می شوند.

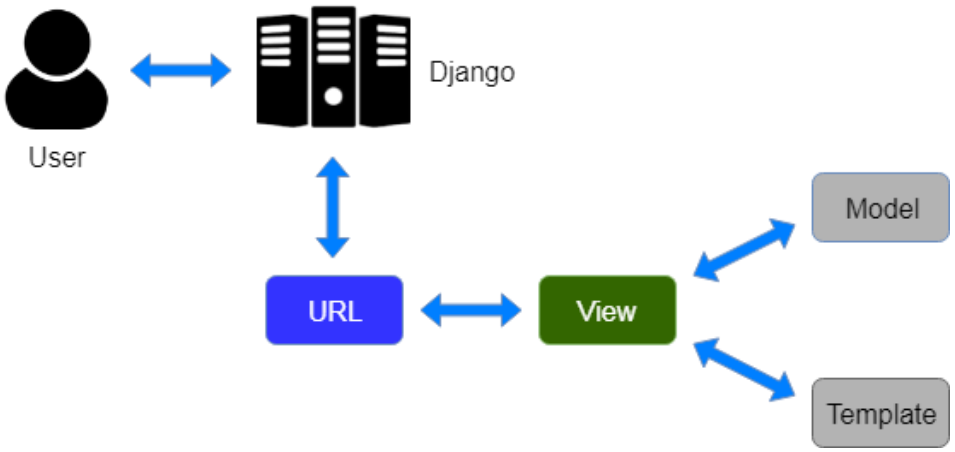
views.py : این فایل مهم ترین فایل در تمام برنامه است چرا که تمام منطق برنامه در آن نوشته می شود .

templates : در این فولدر تمام صفحات HTML برنامه مون را قرار می دهیم .

form.py : این فایل به صورت دستی توسط من ایجاد شده و به صورت پیش فرض وجود ندارد.

3-4. مفاهیم مورد نیاز برای پروژه

برای اینکه بخواهیم یک پروژه ی جنگو بسازیم ابتدا لازم است با یک سری مفاهیم آشنا بشویم .



MVT(model view template)

یک روش معماری نرم افزار است مشابه به MVC ولی با این تفاوت که در اینجا Views نقش کنترلر را دارد و تمام منطق برنامه را در خود نگه می دارد و template ظاهر برنامه را در خود نگه میدارد . model هم نقش ارتباط با دیتابیس را دارد .

CRUD(create read update delete)

کراد چهار تابع اصلی هستند که ما باید در برنامه نویسی مبتنی بر وب باهاش کار کنیم هستند. تابعی برای ساختن اطلاعات در دیتابیس ، تابعی برای خواندن اطلاعات در دیتابیس ، تابعی برای آپدیت اطلاعات نوشته شده در دیتابیس ، و در نهایت تابعی برای پاک کردن اطلاعات موجود در دیتابیس.

4-4. ارتباط با دیتابیس

Models.py : جنگو قابلیت برنامه نویسی شی گرا را دارد . همچنین با استفاده از امکانی که جنگو فراهم کرده ما می توانیم محتویات یک کلاس را با چند دستور مایگریشن در دیتابیس قرار دهیم . برای همینم توابعی که در این فایل قرار می دهیم اهمیت زیادی دارند . دقت کنید که کلاسی که می توانیم اطلاعاتش را در دیتا بیس قرار دهیم اسمش model است و برای اینکه یک مودل تعریف کنیم به هنگام تعریف باید models.Models در داخل پرانتز جلوش قرار دهیم تا جنگو بفهمد ما می خوایم مودل بسازیم نه یک کلاس معمولی.

```
from django.db import models
from django.urls import reverse
from django.conf import settings
import posts

# from posts import views
```

```
# Create your models here.
class post(models.Model):

user=models.ForeignKey(settings.AUTH_USER_MODEL,default=1,on_delet
e=models.CASCADE)
    title=models.CharField(max_length=500)
    height_field = models.IntegerField(default=0)
    width_field = models.IntegerField(default=0)
    image =
models.ImageField(null=True,blank=True,width_field="width_field",h
eight_field="height_field")
    content=models.TextField()
    publish=models.DateField(auto_now=False,auto_now_add=False)
    draft=models.BooleanField(default=False)

timestamp=models.DateTimeField(auto_now=False,auto_now_add=True)
    updated=
models.DateTimeField(auto_now=False,auto_now_add=True)
```

در کد بالا ، مدل posts برای post های وبلاگ تعریف شده اند . فیلد های مدل ، اطلاعاتی هستند که هر پست وبلاگ لازم است داشته باشد و ما میخواهیم آن ها را ذخیره کنیم . با اجرای دستورات مایگریشن ، جنگو جدولی در دیتابیس درست می کند که نام جدول برابر با نام مدل است . و ستون های آن برابر فیلد های مدل . برای همین هم مهم هست که این فیلد ها به روش جنگو تعریف شوند تا دستورات مایگریشن آن ها را بشناسند .

دستورات مایگریشن : پس از تعریف مدل ، برای ایجاد جدول برای آن در دیتابیس ، باید از دستورات زیر استفاده کنیم :

```
>>Python manage.py makemigrations
>>python manage.py sqlmigrate نامapp  شماره فایل مایگریت
>>python manage.py migrate
```

کل بعد از اجرای دستور اول ، فایلی در فولدر Migrations ساخته می شود که وظیفه ی تبدیل Model ما را به دستورات sql را دارد . در دستور دوم ما این فایل را صدا می زنیم و اجرا می کنیم . و در نهایت تمام تغییرات را در دیتابیس ثبت می کنیم .

بعضی موقع ها ممکن است که ما بخوایم model مون رو تغییر بدیم. یه فیلد اضافه یا کم کنیم ، در این مواقع باید remigrate کنیم تا تغییرات در دیتابیس اعمال شود.

برای remigrate کردن :

```
>>python manage.py makemigrations
>>python manage.py migrate
```

4-5. ایجاد پست جدید(create)

همان طور که قبلا اشاره کردیم ، منطق برنامه را داخل views.py می نویسیم . توابع کراد نیز همین جا پیاده سازی می شوند . در کد زیر شما می توانید کد مربوط به ایجاد پست جدید را مشاهده کنید :

```
def posts_create(request):
    # if not request.user.is_staff or not
request.user.is_superuser :
    #     raise Http404
    if not request.user.is_authenticated:
        raise Http404
    form =post_form(request.POST,request.FILES  or None)
    if form.is_valid():
        instance = form.save(commit=False)
        instance.user = request.user
        instance.save()
        messages.success(request,"sucsessfuly created")
        #return HttpResponseRedirect(instance.get_absulute_url())

    # if request.method == "POST":
        # print(request.POST.get("title"))
        # print(request.POST.get("content"))
    context={
        "form":form
    }
    return render(request,"post_form.html",context)
```

به هر آدرسی که کاربر در مرورگر خود می زند ، یک request گفته می شود . این تابع در آدرس http://127.0.0.1:8000/posts/create/ اجرا می شود . البته آدرس دهی کاملا تحت اختیار ماست و می توانیم آن را طبق نیاز خودمون تغییر بدهیم .

```
if not request.user.is_authenticated:
    raise Http404
```

در کد بالا چک می کند که آیا request فرستاده شده به تابع ، توسط یک کاربر لاگین شده است یا خیر . کسانی که کاربر وبلاگ نیستند نباید اجازه ی ایجاد پست جدید داشته باشند . اگر کسی که لاگ این نشده این request را بفرستد ، با صفحه ی Error 404 مواجه می شود .

```
form =post_form(request.POST,request.FILES  or None)
```

در کد بالا تمام محتویات فرم ، با متد پست گرفته می شود و در داخل متغیری به نام form قرار می گیرد . فایل های فرستاده شده درواقع همان عکس پست ها هستند . بعضی از پست های وبلاگ ممکن است عکس نداشته باشند . برای همین هم ما از or None استفاده می کنیم تا این آپشن موجود باشه که پست بدون عکس بسازند.

```
if form.is_valid():
    instance = form.save(commit=False)
    instance.user = request.user
    instance.save()
    messages.success(request,"sucsessfuly created")
```

این تیکه کد برای اضافه کردن نام نویسنده به محتویات فرم پر شده است . ابتدا چک می کند که آیا محتویات پر شده معتبر است یا خیر و اگر بود ، نام نویسنده ای که آن فرم را پر کرده در instance به همراه تمام محتویات دیگر فرم نگهداری می کند.Instance شی است که از مدل post در تابع post_detail ساخته ایم . با صدا زدن

تابع save(). تمامی اطلاعات دریافت شده در دیتابیس ریخته می شود .

```
context={
    "form":form
}
return render(request,"post_form.html",context)
```

اطلاعات فرم را در یک متغیر نگه می داریم . در نهایت برای خروجی تابع ، آدرس post_form.html که در تمپلیت هامون قرار دارد را می دهیم . دقت کنید که این قسمت حداقل دو بار اجرا می شود . دفعه ی اول چون فرم پست بک نکرده است خروجی تابع صرفا نمایش فرم خالی است . اما دفعه دوم که پست بک اتفاق افتاده و دیتا در دیتابیس نشسته است ، متغیر context که برابر محتویات فرم نیز هست پاس داده می شود به فرم .

4-6. نمایش پست ها (Read)

برای نمایش پست های ثبت شده در دیتابیس ، ما نیاز به دو تابع داریم . تابع اول ، تابعی است که تمام پست ها را لود می کند و صرفا بخشی از متن آن ها را به عنوان خلاصه نشان می دهد و تابع دوم ، تابعی است که باید محتویات پستی که روش کلیک شده را کامل نشان دهد . زمانی که روی بیشتر بخوانیم یا عنوان پست کلیک شد این تابع ، اون پست را کامل نشان می دهد .

تابع اول :

```
def posts_list(request):#list items

queryset=post.objects.filter(draft=False).filter(publish__lte=time
zone.now()).order_by("-timestamp")
    paginator = Paginator(queryset, 6) # Show 6 post per page.
```

```
    page_number = request.GET.get('page')
    page_obj = paginator.get_page(page_number)
    try:
        queryset=paginator.page(1)
    except PageNotAnInteger:
        Paginator.page(1)
    except EmptyPage :
        queryset = paginator.page(paginator.num_pages)
    context={
        "objectsList":queryset,
        "title":"list"

    }
    return render(request,"posts_list.html",context)
```

تابع فوق در آدرس http://127.0.0.1:8000/posts/ اجرا میشود . حال بیایید به تحلیل خط به خط این تابع مهم بپردازیم .

```
queryset=post.objects.filter(draft=False).filter(publish__lte=time
zone.now()).order_by("-timestamp")
```

در این کد تمام پست های ثبت شده در جدول post را که ثبت موقت نیستند و تاریخ انتشار آن ها از تاریخ حاضر عقب تر است را نمایش می دهیم . به این ترتیب که آن هایی که اخیرا بودند بالا تر دیده شوند .

```
paginator = Paginator(queryset, 6) # Show 6 post per page.
page_number = request.GET.get('page')
page_obj = paginator.get_page(page_number)
try:
    queryset=paginator.page(1)
except PageNotAnInteger:
    Paginator.page(1)
except EmptyPage :
    queryset = paginator.page(paginator.num_pages)
```

کد بالا مربوط به پیج بندی صفحات بلاگ است . نشان دادن تمام پست ها در صفحه ی اصلی اصلا ایده ی خوبی نیست . چرا که صفحه اصلی سنگین می شود ، سرعت لود آن پایین

می آید . بهترین کار این است که محدودیتی بذاریم که تعدادی از پست ها را در صفحه بیشتر لود نکند . و برای دیدن بیشتر در پایین صفحه امکان ورق زدن داشته باشد .

```
context={
    "objectsList":queryset,
    "title":"list"

}
return render(request,"posts_list.html",context)
```

در این قسمت نیز queryset حاوی اطلاعات پست های قابل نمایش و عنوان صفحه که به انتخاب list است را به post_list.html برای نمایش پاس می دهیم .

تابع دوم :

```
def posts_detail(request,id):#retrive
    instance=get_object_or_404(post,id= id)
    context={
        "title":instance.title,
        "instance":instance,

    }
```

این تابع محتویات صفحه ای که id آن request شده است را نشان می دهد .

4-7. بروز رسانی یک پست (update)

گاهی وقت لازم است پستی که در دیتابیس ثبت شده است را editکنیم . برای این کار ما نیاز به یک تابع update داریم .تابع بروزرسانی :

```
def posts_update(request,id=None):
    # non users can't see this form
    if not request.user.is_staff or not request.user.is_superuser
:
        raise Http404
    instance=get_object_or_404(post,id= id)
    form =post_form(request.POST or None,request.FILES  or
None,instance=instance)
    if form.is_valid():
        instance=form.save(commit=False)
```

```
        instance.save()
        messages.success(request, "sucsessfuly created")
        #return HttpResponseRedirect(instance)

    queryset=post.objects.all()
    context={
        "objectsList":queryset,
        "title":"list",
        "form":form,
    }
    return render(request,"post_form.html",context)
```

توضیح فوق در آدرسی مشابه http://127.0.0.1:8000/posts/17/update اجرا می شود . عدد 17 در این آدرس ، id پستی هس که میخوایم آن را بروز رسانی کنیم . حال بیاید خط به خط آن را تحلیل کنیم :

```
if not request.user.is_staff or not request.user.is_superuser :
    raise Http404
```

در کد بالا محدودیتی ایجاد می کنیم تا افرادی که لاگین نیستند نتوانند پست ها را ادیت کنند .

```
instance=get_object_or_404(post,id= id)
form =post_form(request.POST or None,request.FILES  or
None,instance=instance)
if form.is_valid():
    instance=form.save(commit=False)
    instance.save()
    messages.success(request, "sucsessfuly created")
```

در کد بالا ابتدا اطلاعات پستی که id آن را از request دریافت کرده است را داخل instance می ریزد. سپس اگر پست بک اتفاق افتاده باشد اطلاعات فرستاده شده را در دیتابیس save می کند .

```
queryset=post.objects.all()
context={
    "objectsList":queryset,
    "title":"list",
```

```
    "form":form,
}
return render(request,"post_form.html",context)
```

در انتها تمام محتویات queryset را به فرم پاس می دهیم .

4-8. حذف یک پست (delete)

برای حذف یک پست ما تابع زیر را داریم :

```
def posts_delete(request,id=None):
    if not request.user.is_staff or not request.user.is_superuser
:
        raise Http404
    instance = get_object_or_404(post, id=id)
    instance.delete()
    messages.success(request, "sucsessfuly deleted")
    return redirect("list")
def posts_delete(request,id=None):
    if not request.user.is_staff or not request.user.is_superuser
:
        raise Http404
    instance = get_object_or_404(post, id=id)
    instance.delete()
    messages.success(request, "sucsessfuly deleted")
    return redirect("list")
```

حال بیاین خط به خط این تابع را تحلیل کنیم :

```
if not request.user.is_staff or not request.user.is_superuser :
    raise Http404
```

در کد بالا ما محدودیتی ایجاد می کنیم تا کسانی که عضو نیستند ، به این پیج دسترسی نداشته باشند .

```
instance = get_object_or_404(post, id=id)
instance.delete()
```

در اینجا پست مد نظرمون رو در instance می ریزیم و با صدا زدن تابع delete آن را delete می کنیم.

4-9. صفحه ی admin

جنگو به صورت اتومات برای ما صفحه ی adminرا میسازد اما با کد نویسی آن را شخصی سازی کرد :

```
class PostAdmin(admin.ModelAdmin):
    list_display = ["title","__str__","timestamp","updated",]
    list_display_links = ["updated"]
    list_filter = ["updated","timestamp"]
    list_editable = ["title"]
    search_fields = ["title","content"]
    class meta:
        model=post
# Register your models here.
admin.site.register(post,PostAdmin)
```

در کلاسPostAdmin فیلد های اضافی که میخوایم نشون بده رو مشخص می کنیم . اینکه چجوری مشخص کنیم چی رو نشون بده رو باید طبق داکیومنت های جنگو انجام بدیم . در انتها هم کلاس post رو به شکل PostAdmin میگیم نمایش بده .

توجه : صفحه ی Admin در آدرس http://127.0.0.1:8000/admin/ قرار دارد . نام کاربری admin و پسورد آن 1234 است.

4-10. themplate

در جنگو امکان داینامیک کردن تمپلیت نیز وجود دارد . معمولا دیزاین تمپلیت ها ثابت و دیتای آن ها تغییر می کند . آن قسمت هایی که ثابت هست را داخل یک صفحه ی html به نام base.html ذخیره می کنیم . نام اختیاری است اما نامی که همه معمولا می گذارند base.html است .

محتویات فایل base.html :

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

<html lang="en">
<head>
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstra
p.min.css" integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">

<!-- Optional theme -->
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstra
p-theme.min.css" integrity="sha384-
rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp"
crossorigin="anonymous">
    <meta charset="UTF-8">
    <title>Atefeh Khazraei Blog</title>
    <link rel="stylesheet" href="{% static
"/static/css/base_css.css" %}">
</head>

<body>
<div id="fb-root"></div>
<script async defer crossorigin="anonymous"
src="https://connect.facebook.net/en_GB/sdk.js#xfbml=1&version=v7.
0" nonce="8QGOwJz4"></script>
<div class="container">
{% block content %}


{% endblock content %}

</div>

<!-- Latest compiled and minified JavaScript -->
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.
min.js" integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
crossorigin="anonymous"></script>
</body>
</html>
```

توضیح :

```
{% load static %}
```

کد بالا را در بالای صفحه ی base.html می گذاریم تا جنگو تشخیص دهد این فایل همون فایل اصلی هست که ثابته و دیتا رو باید بین تگ های زیر قرار بدهد :

```
{% block content %}


{% endblock content %}
```

کد زیر کدی هست که توسط facebook برای امکان کامنت گذاری ارائه شده است .

```
<div id="fb-root"></div>
<script async defer crossorigin="anonymous"
src="https://connect.facebook.net/en_GB/sdk.js#xfbml=1&version=v7.
0" nonce="8QGOwJz4"></script>
<div class="container">
```

حال برای محتوایی که تغییر می کند ، با توجه به این که کدوم تابع توی views.py کار دارد ، صفحه طراحی می کنیم .

صفحه ی posts.html برای نمایش لیست کامل پست ها هست که کد های زیر را شامل می شود :

```
{% extends "base.html" %}

{% block content %}

    <div class="col-sm-8 col-sm-offset-2">
    <h1>Lists</h1>
 <div class="row">
{% for obj in objectsList %}

  <div class="col-sm-6 col-sm-offset-6"">
    <div class="thumbnail">
    {% if obj.image %}
     <img src="{{ obj.image.url }}" class="img-responsive"></img>
        {% endif %}
      <div class="caption">
        <h3><a href="{% url "detail" id=obj.id %}">{{ obj.title
}}</a> <br><small> {{ obj.publish }}<br></small></h3>
        <p>{{ obj.content|linebreaks|truncatechars:120 }} <br></p>
                {% if obj.user.get_full_name %}
    <p> Author : {{ obj.user.get_full_name }}</p>
    {% endif %}
        <p><a href="{% url "detail" id=obj.id %}" class="btn btn-
primary" role="button">Read More</a> </p>
```

```
      </div>
    </div>
  </div>
{% endfor %}

    <div class="pagination">
    <span class="step-links">
        {% if objectsList.has_previous %}
            <a href="?page=1">&laquo; first</a>
            <a href="?page={{ objectsList.previous_page_number
}}">previous</a>
        {% endif %}

        <span class="current">
            Page {{ objectsList.number }} of {{
objectsList.paginator.num_pages }}.
        </span>

        {% if objectsList.has_next %}
            <a href="?page={{ objectsList.next_page_number
}}">next</a>
            <a href="?page={{ objectsList.paginator.num_pages
}}">last &raquo;</a>
        {% endif %}
    </span>
</div>


</div>
    </div>
{% endblock content %}


<div class="row">
  <div class="col-sm-6 col-md-4">
    <div class="thumbnail">
      <img src="..." alt="...">
      <div class="caption">
        <h3>Thumbnail label</h3>
        <p>...</p>
        <p><a href="#" class="btn btn-primary"
role="button">Button</a> <a href="#" class="btn btn-default"
role="button">Button</a></p>
      </div>
    </div>
  </div>
</div>
```

توضیح کد :

```
{% extends "base.html" %}
```

این کد به جنگو می فهماند که محتوای این پیج باید داخل base.html قرار بگیرد.

```
{% for obj in objectsList %}
```

به تعداد پست هایی که وجود دارد ، حلقه ای ایجاد کن

```
{% if obj.image %}
 <img src="{{ obj.image.url }}" class="img-responsive"></img>
    {% endif %}
```

اگر پست ، عکس داشت نشان بده

```
<h3><a href="{% url "detail" id=obj.id %}">{{ obj.title }}</a>
<br><small> {{ obj.publish }}<br></small></h3>
```

عنوان پست و تاریخ نشر شو نشان بده

```
<p>{{ obj.content|linebreaks|truncatechars:120 }} <br></p>
```

محتوای پست رو تا 120 کاراکتر نشون بده . (میخوایم خلاصه شو نشون بدیم). امکان اینکه یک خط خالی Enter بکنه کاربر هم وجود داشته باشه .

```
{% if obj.user.get_full_name %}
<p> Author : {{ obj.user.get_full_name }}</p>
```

اگه پستی که از دیتابیس خوندی ، نویسنده اش ، اسم کامل داشت ، آنوقت بیا اسم اون رو به عنوان نویسنده چاپ کن .

```
<p><a href="{% url "detail" id=obj.id %}" class="btn btn-primary"
role="button">Read More</a> </p>
```

اگه روی خواندن بیشتر کلیک کرد برو به آدرسی که همون آدرس بار خودمونه ولی جلوش id پست رو بذار.

```
    <div class="pagination">
    <span class="step-links">
        {% if objectsList.has_previous %}
            <a href="?page=1">&laquo; first</a>
            <a href="?page={{ objectsList.previous_page_number
}}">previous</a>
        {% endif %}

        <span class="current">
            Page {{ objectsList.number }} of {{
objectsList.paginator.num_pages }}.
        </span>
```

```
        {% if objectsList.has_next %}
            <a href="?page={{ objectsList.next_page_number
}}">next</a>
            <a href="?page={{ objectsList.paginator.num_pages
}}">last &raquo;</a>
        {% endif %}
    </span>
</div>
```

این تیکه کد مربوط به صفحه بندی است .

صفحه ی هر پست

این صفحه را هر وقت که میخواهیم محتویات یک پست را کامل نشان دهیم لازم داریم:

```
{% extends "base.html" %}

{% block content %}



{% if messages %}
<ul class="messages">
    {% for message in messages %}
    <li{% if message.tags %} class="{{ message.tags }}"{% endif
%}>{{ message }}</li>
    {% endfor %}
</ul>
{% endif %}
اگر پیغامی باشد که بر اثر ادیت کردن پست یا پاک کردنش به وجود آمده
. باشد آن را نشان می دهد

<div class="col-sm-6 col-sm-offset-6"/>
    <img src="{{ instance.image.url }}" class="img-
responsive"></img>
    <b>{{ instance.title }}</b> <br> {{ instance.publish }}
<small>{% if instance.draft %} <span style="color:
red">Draft</span>
    {% endif %} </small> <br><br>
    {{ instance.content|linebreaks }} <br>
    {% if instance.user.get_full_name %}
    <p> Author : {{ instance.user.get_full_name }}</p>
    {% endif %}
<div class="fb-comments" data-href="{{ request.built_absolute_url
}}" data-numposts="5" data-width=""></div>

</div>
{% endblock content %}
```

```
</body>
</html>
```

در نهایت form

```
{% extends "base.html" %}

{% block content %}
    <div class="col-sm-6 col-sm-offset-3">
<form method="post" action="" enctype="multipart/form-data">
    {% csrf_token %}
{{ form.as_p }}
<input class="btn btn-default" type="submit"  value="Create Post">
</form>
    </div>
{% endblock content %}
</body>
</html>
```

توضیح کد:

```
{% csrf_token %}
```

این کد برای امنیت است و اگر نگذاریم فرم درست کار نمی کند.

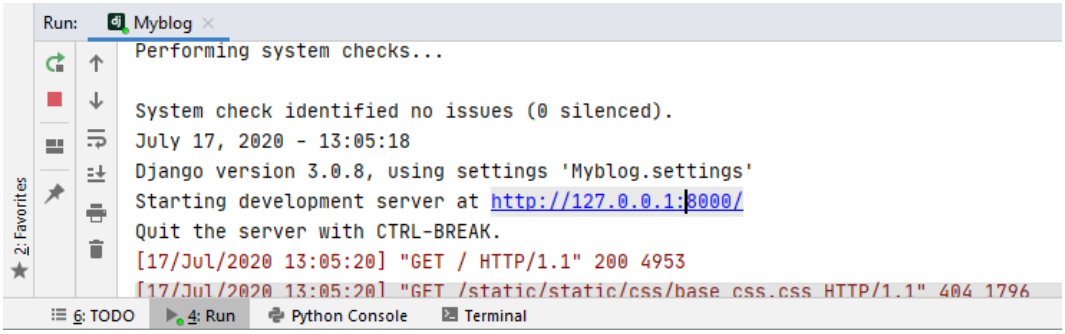
فصل5

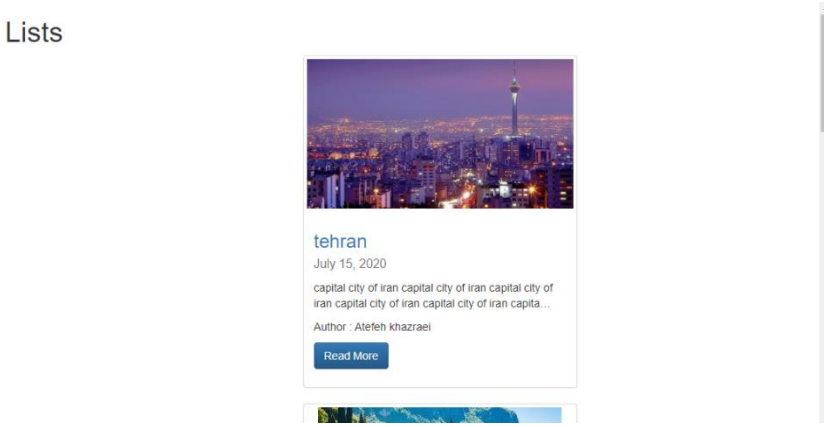
راهنمای برنامه

5-1. شروع کار با برنامه

ابتدا برنامه را در pycharm pro Run کنید .

سپس از پنجره ی Run روی لینک سرور مورد نظر Click کنید .





در صفحه اصلی وبلاگ را مشاهده می کنید . برای ایجاد پست جدید آدرس زیر را وارد کنید :

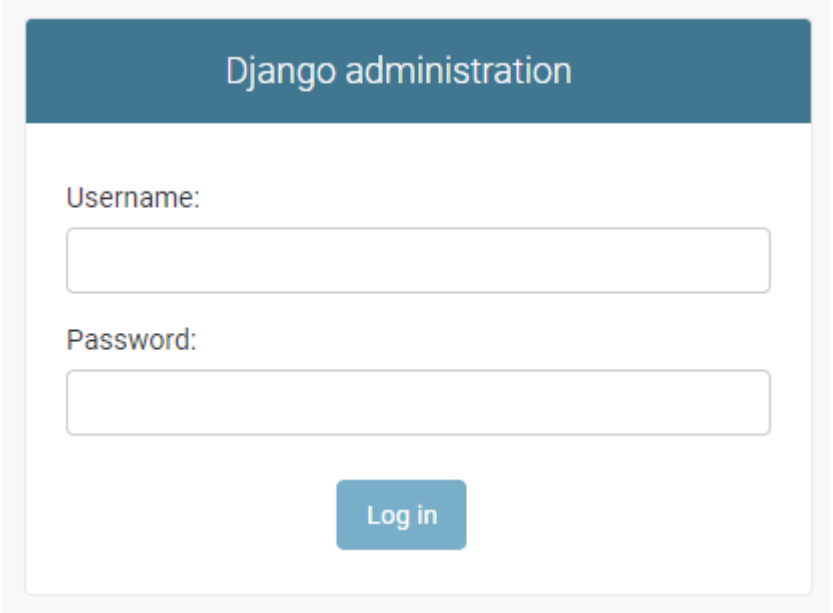
http://127.0.0.1:8000/create/

برای ادیت کردن یه پست باید به Readmore اون پست بروید و به آدرس بالا /update اضافه کنید .

برای delete کردن ی پست نیز باید به Readmore اون پست بروید و به آدرس بالا /delete اضافه کنید .

برای مشاهده صفحه لاگین admin آدرس زیر را وارد کنید :

http://127.0.0.1:8000/admin/



نام کاربری :admin

پسورد:1234

نتیجه گیری :

با استفاده از برنامه ی وبلاگ می توان تولید متوا را در اینترنت بسیار ساده ، راحت ، سریع و امن کرد . همچنین می شود از این برنامه در پیاده سازی وب سایت های دیگر نیـز اسـتفاده کرد .

پیشنهاد :

می توان با نوشتن APP های لایک و کامنت و شیر ، این وبلاگ را گسترش داد . من بـرای کامنت گذاری از کد های Facebook استفاده کردم و اگر کاربر در Facebook لاگین باشد می تواند کامنت برای پست های وبلاگ بگذارد.

فهرست منابع

https://docs.djangoproject.com/en/3.0/

https://developers.facebook.com/docs/plugins/comments#moderation