Kumar Shubham
Posted on Jan 4, 2021 • Originally published at towardsdatascience.com

3 1

Build a Social Media Website with Django — Feed App Backend (Part 4)

#django #python #webdev #programming

Build a Social Media Website with Django (5 Part Series)

So, in the 2nd part and the 3rd part of the tutorial series, we have discussed Users app backend and its templates respectively.

So, in this part (the 4th one), we will start discussing Feed app. In this part, we will focus on the backend of Feed app (models, views, forms etc) and we will discuss templates of Feed app in the next part.

Before moving forward, I want to make sure that you have read all the previous parts of the series otherwise it would not make much sense reading this one, though you can if you want to focus on specific details only rather than the complete website.

As I mentioned in previous parts, I won't be going in details about various terms as it would make it too long. I would focus on the main aspects of the code rather than indulging in simple terms.

To better understand everything, you should be familiar with Django terms. Hopefully, you would be familiar with them from 2nd part of the series but still, if you have doubts, Django official website is a good place to learn about them.

So, let's go ahead and build the Feed app we created in the first part of the tutorial. We will start by creating the models first.

models.py

In this python file, we will define our models. As you might already know, models are defined to tell Django what to save in the database and to define the relationship between different models and what characteristics they have.

To learn more about Django models, do visit this amazing tutorial on models by Mozilla Developers. It talks about models in depth.
After you are comfortable with how models work, you can proceed to make models for our social media website.

So, in this Feed app, we will have three models — one for posts, one for comments and the last one for likes.

So, as usual, we will have to import various items first up. These would include the default Django User model and timezone.

So, let’s have a look at our first model — Post model. It will have five parameters:-

1. description — This is the part of the post where the user would put a small description relevant to the picture he is posting. It is optional since we do not want to force the user to put a description. It has a maximum length of 255 characters and is a CharField.
2. pic — This is the most important part of the post — the picture. Users will upload a picture of their choice for uploading. It would be saved in the file path mentioned. It uses an ImageField.
3. date_posted — It will use the DateTimeField of Django and will set the timestamp to each post. We will use the default time as the current time.
4. user_name — This is a ForeignKey relationship. It is a Many to One relationship since a user can have many posts but a post can only belong to one user. When the user is deleted, the post will be deleted too as evidenced by the usage of on_delete=models.CASCADE. It links up the post with the User model.
5. tags — This is used to take in relevant tags for the post. It can be left blank and is of the maximum of 100 characters. Tags can help to search for relevant posts.

Next, we describe the `__str__` which decides how Django will show our model in the admin panel. We have set it to show the description as the Query object.

We also define the `get_absolute_url` to get the absolute URL for that post.

Next, we have the Comments model. It has four parameters:-

1. post — This is a foreign key which connects the post and comment. A comment can be for a single post but a single post can have multiple comments. Deletion of the post will delete the comments too.
2. username — This is a foreign key which relates a comment to the user. When the user is deleted, the comment will also be deleted.
3. comment — This is the CharField which will hold the relevant comment. It has a maximum character limit of 255 characters.
4. comment_date — It will use the DateTimeField of Django and will set the timestamp to each comment. We will use the default time as the current time.

Next up, we have our final model — Likes. It has two parameters:-

1. user — It represents the user who has liked the post. Deleting the user deletes the like.
2. post — It is the post on which the like is given. Deleting the post deletes all its likes too.

So, this sums up our models.py file. Let’s have a look at the code:-

```
from django.db import models
from django.contrib.auth.models import User
from django.urls import reverse
from django.utils import timezone

class Post(models.Model):
    description = models.CharField(max_length=255, blank=True)
    pic = models.ImageField(upload_to='path/to/img')
    date_posted = models.DateTimeField(default=timezone.now)
    user_name = models.ForeignKey(User, on_delete=models.CASCADE)
    tags = models.CharField(max_length=100, blank=True)

    def __str__(self):
        return self.description

    def get_absolute_url(self):
        return reverse('post-detail', kwargs={'pk': self.pk})


class Comments(models.Model):
    post = models.ForeignKey(Post, related_name='details', on_delete=models.CASCADE)
    username = models.ForeignKey(User, related_name='details', on_delete=models.CASCADE
    comment = models.CharField(max_length=255)
    comment_date = models.DateTimeField(default=timezone.now)

class Like(models.Model):
    user = models.ForeignKey(User, related_name='likes', on_delete=models.CASCADE)
    post = models.ForeignKey(Post, related_name='likes', on_delete=models.CASCADE)
```

admin.py

It will be short and it will consist of a few lines only. It denotes the models which we will be registering to our admin panel. We will be registering all three of our models here.

```
from django.contrib import admin
from .models import Post, Comments, Like

admin.site.register(Post)
admin.site.register(Comments)
admin.site.register(Like)
```

forms.py

To know more about working of forms in Django, do visit this official tutorial by Django itself. Then proceed into the tutorial.

We define two forms in our forms.py file.

1. NewPostForm — This is for posting a new post by any user. It takes in three fields namely description, fields and tags. user_name is supplied while saving since it is not to be asked to the user.
2. NewCommentForm — Similar to NewPostForm, we have this form to accept new comments. We only take in the comment to be posted and supply the post and the user later.

```
from django import forms
from .models import Comments, Post

class NewPostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['description', 'pic', 'tags']

class NewCommentForm(forms.ModelForm):

    class Meta:
        model = Comments
        fields = ['comment']
```

view.py

Now, we will define the views.py file. It would contain all our views (how to render the files in the web browser). It directly passes data to the templates.

We will first import all the required things we need in our views.py file.

Since the views file is too large, we can make it as we like, so I would give a simple overview of what each view does, and you can read the code below for better understanding. So, let's go through them one by one:

1. PostListView — This view handles the display of all the posts in an order which puts newer posts first up. Each page displays 10 posts and then we need to move to the next page to view more. Also, if the user is not authenticated, we do not give him the option to like on the post. If the user is authenticated, we show whether the user has liked or not.
2. UserPostListView — This view is almost similar to the PostListView. Ordering and pagination are the same. The only difference is that this page shows the posts by a certain user only.
3. post_detail — This view handles the display of a single post. It also displays the comment form and lets a user comment on the post and also displays all the comments. It also shows like count and allows you to like or unlike.
4. create_post — This view handles the creation of a new post. It saves the data and adds the current user as the user_name and commits it.
5. PostUpdateView — This view handles the updating of the post. We can edit our posts with the help of this view.
6. post_delete — This view handles the deletion of posts. It deletes the post when the user wants to.
7. search_posts — This works in a similar fashion to search_users which we did in the Users app backend (Part 2). It takes in the input and searches for posts considering the tags.
8. like — This view handles the like event for the posts. It is done with the help of AJAX requests so that the page does not refresh each time a user likes or unlikes. It works in a way that if the post is already liked by the user, clicking on the like button would remove the like but if it is not already liked, it will like the post. Then it dumps the response as JSON and passes it as an HTTP response.

This sums up the views.py file. Below is the code for the file.

```
from django.shortcuts import get_object_or_404, render, redirect
from django.http import HttpResponseRedirect, HttpResponse, JsonResponse
from django.urls import reverse
from django.contrib import messages
from django.core.paginator import Paginator
from django.contrib.auth.models import User
from .forms import NewCommentForm, NewPostForm
from django.views.generic import ListView, UpdateView, DeleteView
from django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
from .models import Post, Comments, Like
from django.contrib.auth.decorators import login_required
from django.views.decorators.http import require_POST
import json

class PostListView(ListView):
    model = Post
    template_name = 'feed/home.html'
    context_object_name = 'posts'
    ordering = ['-date_posted']
    paginate_by = 10
    def get_context_data(self, **kwargs):
        context = super(PostListView, self).get_context_data(**kwargs)
        if self.request.user.is_authenticated:
            liked = [i for i in Post.objects.all() if Like.objects.filter(user = self.r
            context['liked_post'] = liked
        return context

class UserPostListView(LoginRequiredMixin, ListView):
    model = Post
    template_name = 'feed/user_posts.html'
    context_object_name = 'posts'
    paginate_by = 10

    def get_context_data(self, **kwargs):
        context = super(UserPostListView, self).get_context_data(**kwargs)
        user = get_object_or_404(User, username=self.kwargs.get('username'))
        liked = [i for i in Post.objects.filter(user_name=user) if Like.objects.filter(
        context['liked_post'] = liked
        return context

    def get_queryset(self):
        user = get_object_or_404(User, username=self.kwargs.get('username'))
        return Post.objects.filter(user_name=user).order_by('-date_posted')


@login_required
def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    user = request.user
    is_liked =  Like.objects.filter(user=user, post=post)
    if request.method == 'POST':
        form = NewCommentForm(request.POST)
        if form.is_valid():
            data = form.save(commit=False)
            data.post = post
            data.username = user
            data.save()
            return redirect('post-detail', pk=pk)
    else:
        form = NewCommentForm()
    return render(request, 'feed/post_detail.html', {'post':post, 'is_liked':is_liked,

@login_required
def create_post(request):
    user = request.user
    if request.method == "POST":
        form = NewPostForm(request.POST, request.FILES)
        if form.is_valid():
            data = form.save(commit=False)
            data.user_name = user
            data.save()
            messages.success(request, f'Posted Successfully')
            return redirect('home')
    else:
        form = NewPostForm()
    return render(request, 'feed/create_post.html', {'form':form})

class PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = Post
    fields = ['description', 'pic', 'tags']
    template_name = 'feed/create_post.html'

    def form_valid(self, form):
        form.instance.user_name = self.request.user
        return super().form_valid(form)

    def test_func(self):
        post = self.get_object()
        if self.request.user == post.user_name:
            return True
        return False

@login_required
def post_delete(request, pk):
    post = Post.objects.get(pk=pk)
    if request.user== post.user_name:
        Post.objects.get(pk=pk).delete()
    return redirect('home')



@login_required
def search_posts(request):
    query = request.GET.get('p')
    object_list = Post.objects.filter(tags__icontains=query)
    liked = [i for i in object_list if Like.objects.filter(user = request.user, post=i)
    context ={
        'posts': object_list,
        'liked_post': liked
    }
    return render(request, "feed/search_posts.html", context)

@login_required
def like(request):
    post_id = request.GET.get("likeId", "")
    user = request.user
    post = Post.objects.get(pk=post_id)
    liked= False
    like = Like.objects.filter(user=user, post=post)
    if like:
        like.delete()
    else:
        liked = True
        Like.objects.create(user=user, post=post)
    resp = {
        'liked':liked
    }
    response = json.dumps(resp)
    return HttpResponse(response, content_type = "application/json")
```

urls.py

This file contains all the URLs for the Feed app. These all URLs would be included in the main urls.py file.

Here is the code for urls.py file for the Feed App.

```
from django.contrib import admin
from django.urls import path, include
from . import views
from .views import PostUpdateView, PostListView, UserPostListView

urlpatterns=[
    path('', PostListView.as_view(), name='home'),
    path('post/new/', views.create_post, name='post-create'),
    path('post/<int:pk>/', views.post_detail, name='post-detail'),
    path('like/', views.like, name='post-like'),
    path('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
    path('post/<int:pk>/delete/', views.post_delete, name='post-delete'),
    path('search_posts/', views.search_posts, name='search_posts'),
    path('user_posts/<str:username>', UserPostListView.as_view(), name='user-posts'),
]
```

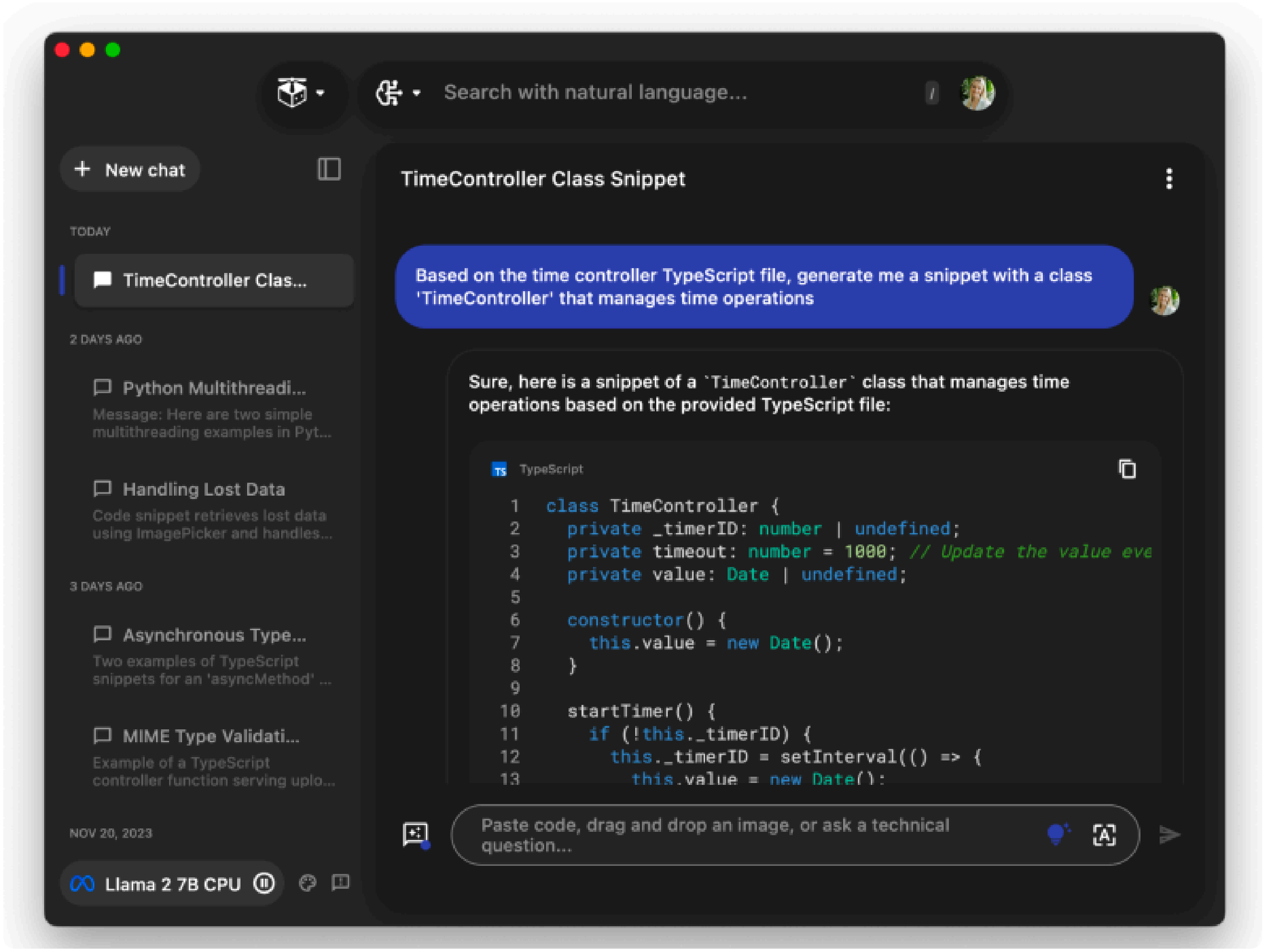
Build a Social Media Website with Django (5 Part Series)

Top comments (0)

Code of Conduct • Report abuse

Pieces.app PROMOTED

A Workflow Copilot. Tailored to You.


Search with natural language...
New chat
TODAY
TimeController Clas...
2 DAYS AGO
Python Multithreadi...
Message: Here are two simple multithreading examples in Pyt...
Handling Lost Data
Code snippet retrieves lost data using ImagePicker and handles...
3 DAYS AGO
Asynchronous Type...
Two examples of TypeScript snippets for an 'asyncMethod' ...
MIME Type Validati...
Example of a TypeScript controller function serving uplo...
NOV 20, 2023
Llama 2 7B CPU
TimeController Class Snippet
TypeScript
Paste code, drag and drop an image, or ask a technical question...

Based on the time controller TypeScript file, generate me a snippet with a class 'TimeController' that manages time operations

Sure, here is a snippet of a `TimeController` class that manages time operations based on the provided TypeScript file:

```
class TimeController {
  private _timerID: number | undefined;
  private timeout: number = 1000; // Update the value eve
  private value: Date | undefined;

  constructor() {
    this.value = new Date();
  }

  startTimer() {
    if (!this._timerID) {
      this._timerID = setInterval(() => {
        this.value = new Date();
```

Our desktop app, with its intelligent copilot, streamlines coding by generating snippets, extracting code from screenshots, and accelerating problem-solving.

Read the docs

Kumar Shubham

Full Stack Web Development | Studies at IIT BHU

LOCATION
Bhagalpur, India

EDUCATION
IIT (BHU), Varanasi

WORK
Student

JOINED
Dec 4, 2020

More from Kumar Shubham

Build a Blog App with React - Finishing the Project (Part 4)
#react #javascript #programming

Build a Blog App with React - Components and Hooks (Part 3)
#react #javascript #programming

Build a Blog App with React —Building Components (Part 2)
#react #javascript #programming