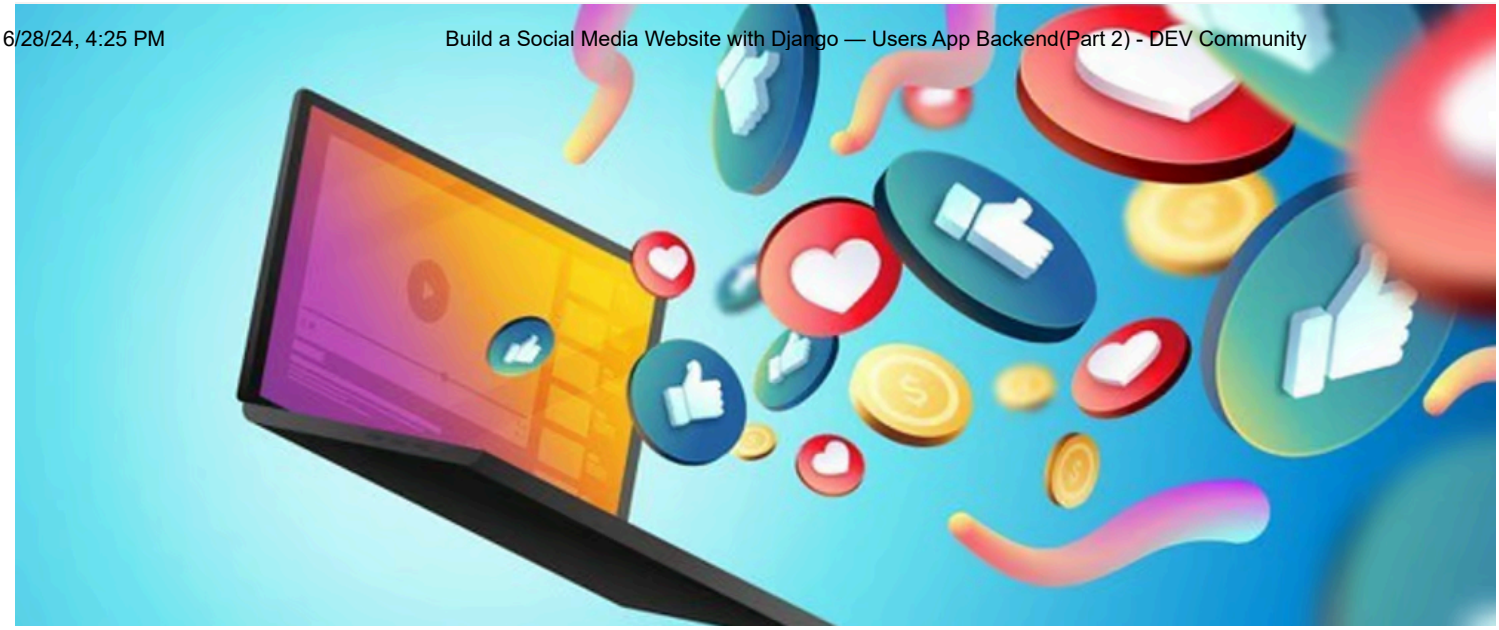

Kumar Shubham
Posted on Jan 4, 2021 • Originally published at towardsdatascience.com

Build a Social Media Website with Django — Users App Backend(Part 2)

#django #python #webdev #programming

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

So, in the first part of the tutorial, we learnt how to set up our project and setup various authentication backends and other details regarding various installs and app setups needed in the settings file.

There would be lots of terms which you may be encountering for the first time, like you may have heard about views or models for the first time, so I would be briefing about them a little. I cannot talk about those in details because our focus is not to teach you basic concepts, but to build a social media website. So, I would be linking to good resources for you to refer to those specific topics and then continue with this tutorial.

So, let's go ahead and build the Users app we created in the last 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, we will have two models — one for the user's profile and the other for Friend Requests.

We will need to import many things before we begin writing the models. We will be using the Default User model of Django to have One to One Relationship with our Profile Model i.e. each user will have one profile.

We are also using autoslug to automatically produce slugs for the URLs based on the username.

For e.g.: A user with the name Tom would have slug as tom. This will help us make meaningful URLs as users/tom will lead to Tom's profile rather than numbers.

```
pip install django-autoslug
```

After installing it, we can import it in models.py by using the line:

```
from autoslug import AutoSlugField
```

After finishing off all the required imports, we can begin writing the models.

So, our first model is the Profile model. It has five parameters:-

1. user — This is a One to One Relationship with Django User model. The on_delete=models.CASCADE means on the deletion of User, we destroy the Profile too.
2. image — This will store the profile picture of the user. We have provided a default image also. We need to define where to save the pictures.
3. slug — This will be the slug field. We use the AutoSlugField and will set it to make slug from the user field.
4. bio — This will store the small introduction about the user. Here, blank=True means it can be left blank.
5. friends — This is a Many to Many Field with Profile model and can be left blank. It means every user can have multiple friends and can be friends to multiple people.

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

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

Next, we define a function to make a profile as soon as we create the user so that the user doesn't have to manually create a profile.

Next, we define our Friends Model. It will have three parameters:-

1. to_user — This denotes the user to whom the friend request will be sent. It will have the same on_delete parameter which decides when the user is deleted, we delete the friend request too.

3. timestamp — It is not really necessary to add. It stores the time when the request was sent.

As you can notice both to_user and from_user uses the same ForeignKey so to differentiate we need to use the related_name field.

So, that finishes our models.py file. Have a look at the code below which shows the models.py file.

```
from django.db import models
from django.contrib.auth.models import User
from django.urls import reverse
from django.utils import timezone
from django.db.models.signals import post_save
from django.conf import settings
from autoslug import AutoSlugField

class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    image = models.ImageField(default='default.png', upload_to='profile_pics')
    slug = AutoSlugField(populate_from='user')
    bio = models.CharField(max_length=255, blank=True)
    friends = models.ManyToManyField("Profile", blank=True)

    def __str__(self):
        return str(self.user.username)

    def get_absolute_url(self):
        return "/users/{}".format(self.slug)

def post_save_user_model_receiver(sender, instance, created, *args, **kwargs):
    if created:
        try:
            Profile.objects.create(user=instance)
        except:
            pass

post_save.connect(post_save_user_model_receiver, sender=settings.AUTH_USER_MODEL)

class FriendRequest(models.Model):
    to_user = models.ForeignKey(settings.AUTH_USER_MODEL, related_name='to_user', on_de
```

```
    def __str__(self):
        return "From {}, to {}".format(self.from_user.username, self.to_user.username)
```

After models.py file, we move forward to admin.py file.

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 both our models here.

```
from django.contrib import admin
from .models import Profile, FriendRequest

admin.site.register(Profile)
admin.site.register(FriendRequest)
```

Next, we move to forms.py.

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 three forms in our forms.py file.

1. UserRegisterForm — This is for registration of a new user. We user the Django's default UserCreationForm and we define what should be in the forms. We set the email to be Django's EmailField. Then we tell Django that model is User and the fields that we would ask the user to fill while registering.
2. UserUpdateForm — This form will let users update their profile. It will have all the same fields as Registration form but we would use the Django Model form instead of UserCreationForm.
3. ProfileUpdateForm — This form will let users update their profile.

So, adding these three forms would complete our forms.py file. Look at the code below:

```
from django import forms
from django.contrib.auth.models import User
```

```
class UserRegisterForm(UserCreationForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']


class UserUpdateForm(forms.ModelForm):
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email']

class ProfileUpdateForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ['bio', 'image']
```

So, after this, we have our forms.py file created. Next, we would be seeing views.py file.

views.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.

Read this official tutorial by Django to understand views in a better way. After reading the tutorial, we move forward.
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. users_list — This view will form the user list to be recommended to any user to help them discover new users to make friends with. We will filter out our friends from that list and will exclude us too. We will make this list by first adding our friend's friends who are not our friends. Then if our user list has still low members, we will add random people to recommend (mostly for a user with no friends).
2. friend_list — This view will display all the friends of the user.

we can send him the request.

4. cancel_friend_request — It will cancel the friend request we sent to the user.
5. accept_friend_request — It will be used to accept the friend request of the user and we add user1 to user2's friend list and vice versa. Also, we will delete the friend request.
6. delete_friend_request — It will allow the user to delete any friend request he/she has received.
7. delete_friend — This will delete the friend of that user i.e. we would remove user1 from user2 friend list and vice versa.
8. profile_view — This will be the profile view of any user. It will showcase the friend's count and posts count of the user and their friend list. Also, it would showcase the friend request received and sent by the user and can accept, decline or cancel the request. Obviously, we would add conditions and checks, so that only the concerned user is shown the requests and sent list and they only have the power to accept or reject requests and not anyone viewing his/her profile.
9. register — This will let users register on our website. It will render the registration form we created in forms.py file.
10. edit_profile — This will let the users edit their profile with help of the forms we created.
11. search_users — This will handle the search function of the users. It takes in the query and then filters out relevant users.
12. my_profile — This is same as profile_view but it will render your profile only.

Go through the code below for better understanding.

```
from django.shortcuts import render, redirect, get_object_or_404
from .models import Profile
from feed.models import Post
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from django.contrib.auth import get_user_model
from django.conf import settings
from django.http import HttpResponseRedirect
from .models import Profile, FriendRequest
from .forms import UserRegisterForm, UserUpdateForm, ProfileUpdateForm
import random
```

```
@login_required
def users_list(request):
    users = Profile.objects.exclude(user=request.user)
    sent_friend_requests = FriendRequest.objects.filter(from_user=request.user)
    my_friends = request.user.profile.friends.all()
    sent_to = []
    friends = []
    for user in my_friends:
        friend = user.friends.all()
        for f in friend:
            if f in friends:
                friend = friend.exclude(user=f.user)
        friends += friend
    for i in my_friends:
        if i in friends:
            friends.remove(i)
    if request.user.profile in friends:
        friends.remove(request.user.profile)
    random_list = random.sample(list(users), min(len(list(users)), 10))
    for r in random_list:
        if r in friends:
            random_list.remove(r)
    friends += random_list
    for i in my_friends:
        if i in friends:
            friends.remove(i)
    for se in sent_friend_requests:
        sent_to.append(se.to_user)
    context = {
        'users': friends,
        'sent': sent_to
    }
    return render(request, "users/users_list.html", context)

def friend_list(request):
    p = request.user.profile
    friends = p.friends.all()
    context={
    'friends': friends
    }
    return render(request, "users/friend_list.html", context)

@login_required
```

```
        frequest, created = FriendRequest.objects.get_or_create(
                from_user=request.user,
                to_user=user)
        return HttpResponseRedirect('/users/{}'.format(user.profile.slug))

    @login_required
    def cancel_friend_request(request, id):
        user = get_object_or_404(User, id=id)
        frequest = FriendRequest.objects.filter(
                from_user=request.user,
                to_user=user).first()
        frequest.delete()
        return HttpResponseRedirect('/users/{}'.format(user.profile.slug))

    @login_required
    def accept_friend_request(request, id):
        from_user = get_object_or_404(User, id=id)
        frequest = FriendRequest.objects.filter(from_user=from_user, to_user=request.user).
        user1 = frequest.to_user
        user2 = from_user
        user1.profile.friends.add(user2.profile)
        user2.profile.friends.add(user1.profile)
        if(FriendRequest.objects.filter(from_user=request.user, to_user=from_user).first()):
            request_rev = FriendRequest.objects.filter(from_user=request.user, to_user=from
            request_rev.delete()
        frequest.delete()
        return HttpResponseRedirect('/users/{}'.format(request.user.profile.slug))

    @login_required
    def delete_friend_request(request, id):
        from_user = get_object_or_404(User, id=id)
        frequest = FriendRequest.objects.filter(from_user=from_user, to_user=request.user).
        frequest.delete()
        return HttpResponseRedirect('/users/{}'.format(request.user.profile.slug))

    def delete_friend(request, id):
        user_profile = request.user.profile
        friend_profile = get_object_or_404(Profile, id=id)
        user_profile.friends.remove(friend_profile)
        friend_profile.friends.remove(user_profile)
        return HttpResponseRedirect('/users/{}'.format(friend_profile.slug))

    @login_required
```

```
        u = p.user
        sent_friend_requests = FriendRequest.objects.filter(from_user=p.user)
        rec_friend_requests = FriendRequest.objects.filter(to_user=p.user)
        user_posts = Post.objects.filter(user_name=u)

        friends = p.friends.all()

        # is this user our friend
        button_status = 'none'
        if p not in request.user.profile.friends.all():
            button_status = 'not_friend'

            # if we have sent him a friend request
            if len(FriendRequest.objects.filter(
                from_user=request.user).filter(to_user=p.user)) == 1:
                    button_status = 'friend_request_sent'

            # if we have recieved a friend request
            if len(FriendRequest.objects.filter(
                from_user=p.user).filter(to_user=request.user)) == 1:
                    button_status = 'friend_request_received'

        context = {
            'u': u,
            'button_status': button_status,
            'friends_list': friends,
            'sent_friend_requests': sent_friend_requests,
            'rec_friend_requests': rec_friend_requests,
            'post_count': user_posts.count
        }

        return render(request, "users/profile.html", context)

    def register(request):
        if request.method == 'POST':
            form = UserRegisterForm(request.POST)
            if form.is_valid():
                form.save()
                username = form.cleaned_data.get('username')
                messages.success(request, f'Your account has been created! You can now logi
                return redirect('login')
        else:
            form = UserRegisterForm()
```

```
@login_required
def edit_profile(request):
    if request.method == 'POST':
        u_form = UserUpdateForm(request.POST, instance=request.user)
        p_form = ProfileUpdateForm(request.POST, request.FILES, instance=request.user.p
        if u_form.is_valid() and p_form.is_valid():
            u_form.save()
            p_form.save()
            messages.success(request, f'Your account has been updated!')
            return redirect('my_profile')
    else:
        u_form = UserUpdateForm(instance=request.user)
        p_form = ProfileUpdateForm(instance=request.user.profile)
    context ={
        'u_form': u_form,
        'p_form': p_form,
    }
    return render(request, 'users/edit_profile.html', context)

@login_required
def my_profile(request):
    p = request.user.profile
    you = p.user
    sent_friend_requests = FriendRequest.objects.filter(from_user=you)
    rec_friend_requests = FriendRequest.objects.filter(to_user=you)
    user_posts = Post.objects.filter(user_name=you)
    friends = p.friends.all()

    # is this user our friend
    button_status = 'none'
    if p not in request.user.profile.friends.all():
        button_status = 'not_friend'

        # if we have sent him a friend request
        if len(FriendRequest.objects.filter(
            from_user=request.user).filter(to_user=you)) == 1:
                button_status = 'friend_request_sent'

        if len(FriendRequest.objects.filter(
            from_user=p.user).filter(to_user=request.user)) == 1:
                button_status = 'friend_request_received'

    context = {
```

```
            'friends_list': friends,
            'sent_friend_requests': sent_friend_requests,
            'rec_friend_requests': rec_friend_requests,
            'post_count': user_posts.count
        }

        return render(request, "users/profile.html", context)

    @login_required
    def search_users(request):
        query = request.GET.get('q')
        object_list = User.objects.filter(username__icontains=query)
        context ={
            'users': object_list
        }
        return render(request, "users/search_users.html", context)
```

It sums up our Users app. We are left with the urls.py which we won't include in users app. We will add it directly to our photoshare app.

urls.py (photoshare)

This file contains all the URLs for the website. It has an include('feed.urls') which includes all URLs from the Feed app which we will build in the next tutorial.

And we include all the URLs for the photoshare app directly in the main urls.py file. Have a look at the file below:

```
from django.contrib import admin
from django.urls import path, include
from users import views as user_views
from django.contrib.auth import views as auth_views
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('feed.urls')),
    path('users/', user_views.users_list, name='users_list'),
    path('users/<slug>/', user_views.profile_view, name='profile_view'),
    path('friends/', user_views.friend_list, name='friend_list'),
```

```
    path('users/friend-request/accept/<int:id>/', user_views.accept_friend_request, nam
    path('users/friend-request/delete/<int:id>/', user_views.delete_friend_request, nam
    path('users/friend/delete/<int:id>/', user_views.delete_friend, name='delete_friend
    path('edit-profile/', user_views.edit_profile, name='edit_profile'),
    path('my-profile/', user_views.my_profile, name='my_profile'),
    path('search_users/', user_views.search_users, name='search_users'),
    path('register/', user_views.register, name='register'),
    path('login/', auth_views.LoginView.as_view(template_name='users/login.html'), name
    path('logout/', auth_views.LogoutView.as_view(template_name='users/logout.html'), n
    path('password-reset/', auth_views.PasswordResetView.as_view(template_name='users/p
    path('password-reset/done/', auth_views.PasswordResetView.as_view(template_name='us
    path('password-reset-confirm/<uidb64>/<token>/', auth_views.PasswordResetConfirmVie
    path('password-reset-complete/', auth_views.PasswordResetCompleteView.as_view(templ
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

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

1. Build a Social Media Website with Django - Set up the Project (Pa...
2. **Build a Social Media Website with Django — Users App Back...**
3. Build a Social Media Website with Django — Part 3 (Users App Te...
4. Build a Social Media Website with Django — Feed App Backend (...
5. Build a Social Media Website with Django — Part 5 (Feed App Te...

Top comments (0)

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