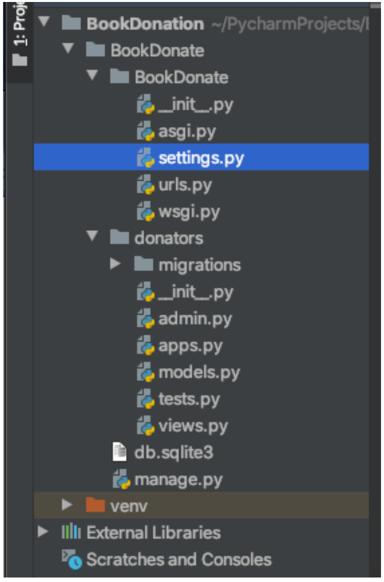
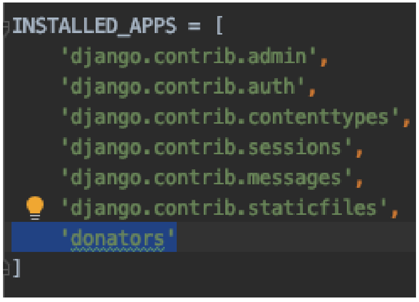
Documentation for a Django App

1. Create a Django project
   $ pip install Django==3.0.3
   $ django-admin startproject BookDonate
   $ cd BookDonate
   $ python manage.py runserver
   $ python manage.py startup app donators
   Now you can see the directory like this:

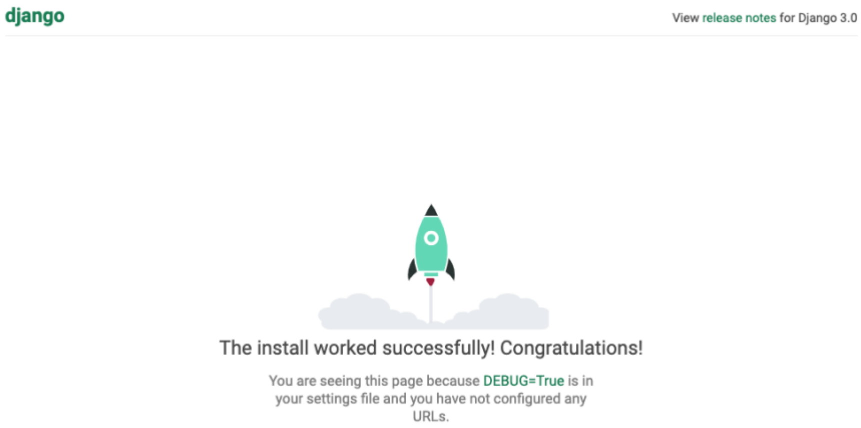


Add this line into the settings.py



```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'donators'
]
```

If you click the http://127.0.0.1:8000/, you can see a website like this:

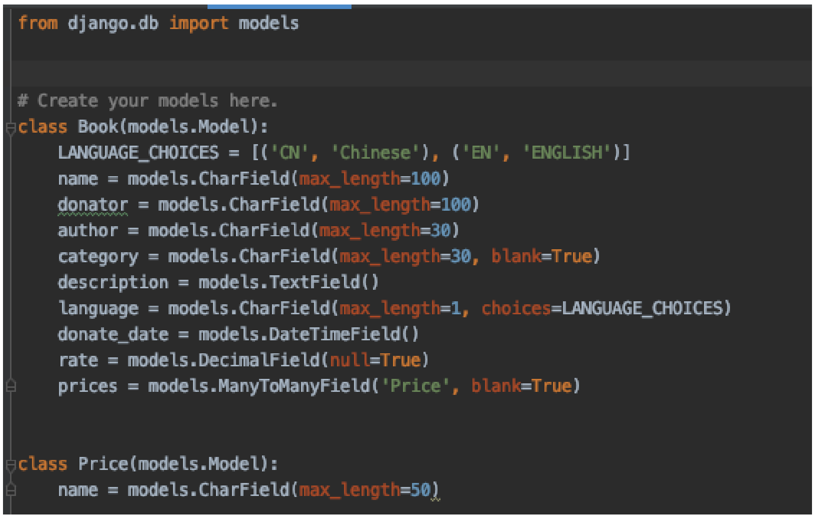


The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

2. Work with Django models and Admin
   a. Django models and fields

A model is a class inherit from Django,db.models.Model and has database fields as class attribute.
Create 2 classes, one is book and the other is price.
They have a many to many relationship.



```
from django.db import models


# Create your models here.
class Book(models.Model):
    LANGUAGE_CHOICES = [('CN', 'Chinese'), ('EN', 'ENGLISH')]
    name = models.CharField(max_length=100)
    donator = models.CharField(max_length=100)
    author = models.CharField(max_length=30)
    category = models.CharField(max_length=30, blank=True)
    description = models.TextField()
    language = models.CharField(max_length=1, choices=LANGUAGE_CHOICES)
    donate_date = models.DateTimeField()
    rate = models.DecimalField(null=True)
    prices = models.ManyToManyField('Price', blank=True)


class Price(models.Model):
    name = models.CharField(max_length=50)
```

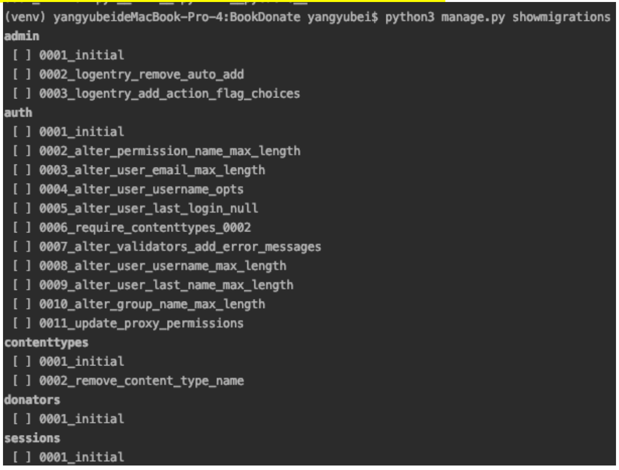
b. Django Migrations

i. $python3.py manage.py makemigrations

Generate migration file; Use current model fields and current database tables

Creates numbered files in appname/migrations

ii. $python3.py manage.py showmigrations



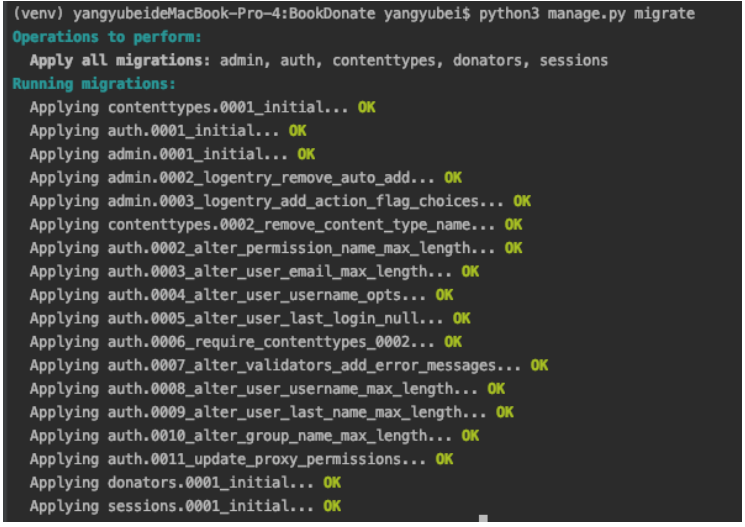
```
(venv) yangyubeideMacBook-Pro-4:BookDonate yangyubei$ python3 manage.py showmigrations
admin
 [ ] 0001_initial
 [ ] 0002_logentry_remove_auto_add
 [ ] 0003_logentry_add_action_flag_choices
auth
 [ ] 0001_initial
 [ ] 0002_alter_permission_name_max_length
 [ ] 0003_alter_user_email_max_length
 [ ] 0004_alter_user_username_opts
 [ ] 0005_alter_user_last_login_null
 [ ] 0006_require_contenttypes_0002
 [ ] 0007_alter_validators_add_error_messages
 [ ] 0008_alter_user_username_max_length
 [ ] 0009_alter_user_last_name_max_length
 [ ] 0010_alter_group_name_max_length
 [ ] 0011_update_proxy_permissions
contenttypes
 [ ] 0001_initial
 [ ] 0002_remove_content_type_name
donators
 [ ] 0001_initial
sessions
 [ ] 0001_initial
```

Empty bracket shows migrations not been applied.

iii. $python3.py manage.py migrate

Can also run only migrations in a specific app to a specific number using

$ python3.py manage.py migrate <appname> <number>



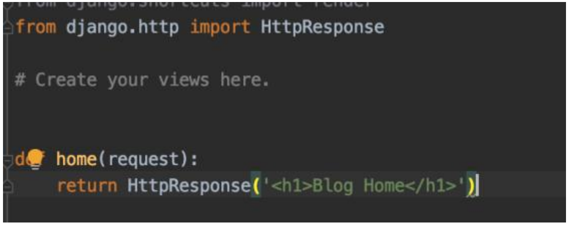
```
(venv) yangyubeideMacBook-Pro-4:BookDonate yangyubei$ python3 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, donators, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying donators.0001_initial... OK
  Applying sessions.0001_initial... OK
```

c.

Documentation for creating Django App BLOG
Chapter 1 Getting Started
Create project
$ Django-admin startproject [projectname]
Run project
$ python3.7 manage.py runserver
Create project
$ python3.7 manage.py startapp ConyBlog

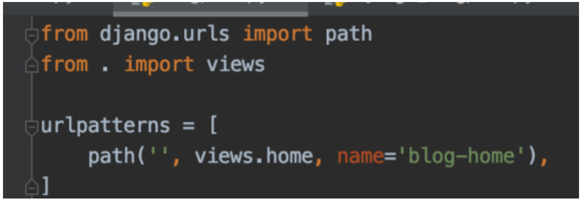
Chapter2 Applications and Routers

Open view.py and add HTTP library. Create function home to handle the traffic from home page of our blog. This function is going to take a request argument. Return some page to user when they sent to this route.



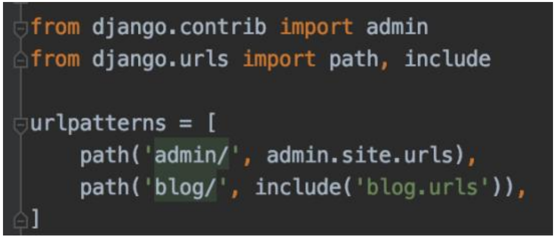
```
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.


def home(request):
    return HttpResponse('<h1>Blog Home</h1>')
```

We now need to map our URL pattern to this view function. Create a new bot module in blog directory called urls.py and in this file can map the urls we want to corresponding view functions.
Import the home function from views.py to urls.py. Create an empty path, add function home that returns http response and name it as 'blog-home'.



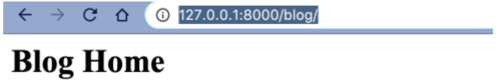
```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='blog-home'),
]
```

Reverse lookup?
Import include in urls.py of project level. Specify which route go to blog URLs.



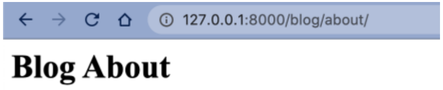
```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('blog/', include('blog.urls')),
]
```

Run server and go to site http://127.0.0.1:8000/blog/ can see a page like this:



Blog Home

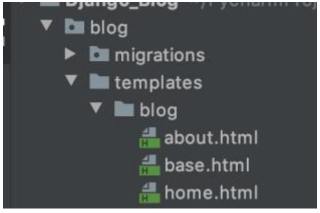
Now we add another page.



Blog About

Chapter 3 Templates

Blog -> templates -> blog -> template.html



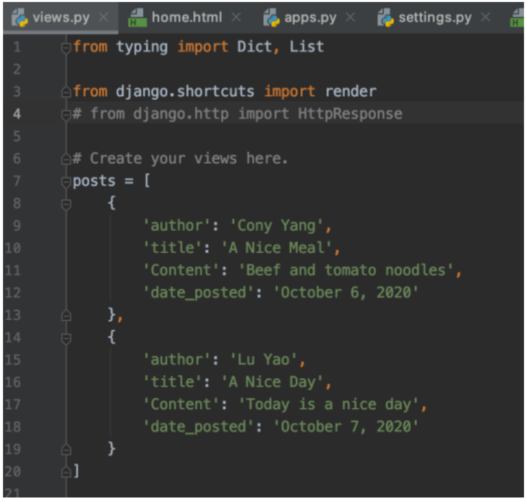
Add this line to settings.py.

```
# Application definition

INSTALLED_APPS = [
    'blog.apps.BlogConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Modify views

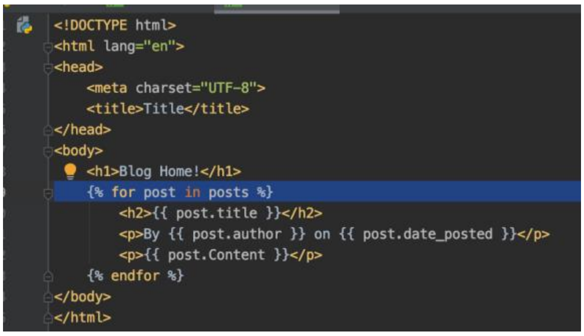
```
def home(request):
    context: Dict[str, List[Dict[str, str]]] = {
        'posts': posts
    }
    return render(request, 'blog/home.html', context)
```

Now we add some dummy data in view.py

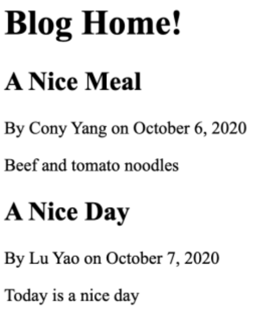


```
from typing import Dict, List

from django.shortcuts import render
# from django.http import HttpResponse

# Create your views here.
posts = [
    {
        'author': 'Cony Yang',
        'title': 'A Nice Meal',
        'Content': 'Beef and tomato noodles',
        'date_posted': 'October 6, 2020'
    },
    {
        'author': 'Lu Yao',
        'title': 'A Nice Day',
        'Content': 'Today is a nice day',
        'date_posted': 'October 7, 2020'
    }
]
```

Modify home.html pass the variable to the template



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>Blog Home!</h1>
    {% for post in posts %}
        <h2>{{ post.title }}</h2>
        <p>By {{ post.author }} on {{ post.date_posted }}</p>
        <p>{{ post.Content }}</p>
    {% endfor %}
</body>
</html>
```



Blog Home!

A Nice Meal

By Cony Yang on October 6, 2020

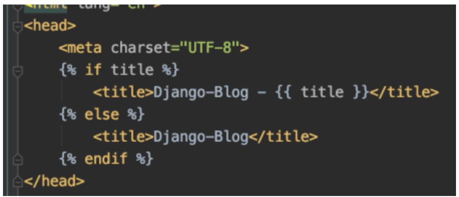
Beef and tomato noodles

A Nice Day

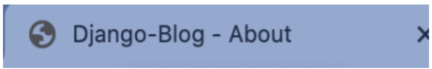
By Lu Yao on October 7, 2020

Today is a nice day

Use if else statement to add title. Pass title to render function



```
<head>
    <meta charset="UTF-8">
    {% if title %}
        <title>Django-Blog - {{ title }}</title>
    {% else %}
        <title>Django-Blog</title>
    {% endif %}
</head>
```



```
title = {'title': 'About'}
return render(request, 'blog/about.html', title)
```

Template Inheritance

Keep the repeated code of about.html and home.html in a new file base.html.

Make the different part as this: Use a content block.

```
<body>
  {% block content %}{% endblock %}
</body>
```

Then in home.html file, delete the repeated part and write an inheritance code.

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

And rewrite the block content.

```
{% block content %}
  <h1>Blog Home!</h1>
  {% for post in posts %}
    <h2>{{ post.title }}</h2>
    <p>By {{ post.author }} on {{ post.date_posted }}</p>
    <p>{{ post.Content }}</p>
  {% endfor %}
{% endblock %}
```

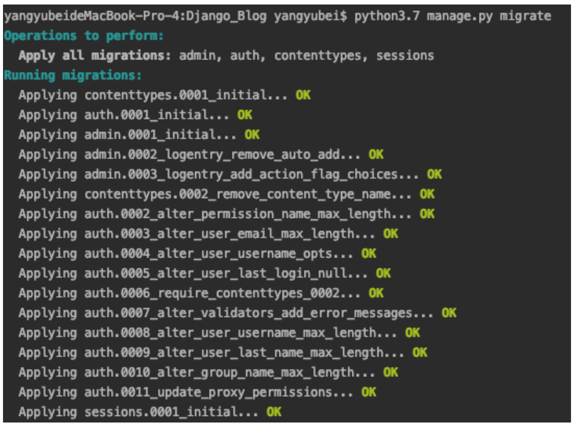
Use URL tags and name of the route instead of hardcoding

```
urlpatterns = [
  path('', views.home, name='blog-home'),
  path('about/', views.about, name='blog-about'),
]
<div class="navbar-nav mr-auto">
 <a class="nav-item nav-link" href="{% url 'blog-home' %}">Home</a>
 <a class="nav-item nav-link" href="{% url 'blog-about' %}">About</a>
</div>
```

Chapter4 Admin-page

Make migrations

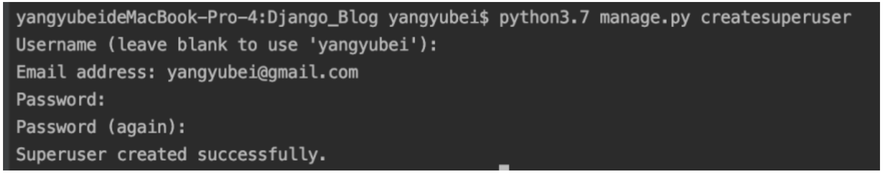
python3.7 manage.py migrate



```
yangyubeideMacBook-Pro-4:Django_Blog yangyubei$ python3.7 manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying sessions.0001_initial... OK
```

Create superuser

python3.7 manage.py createsuperuser



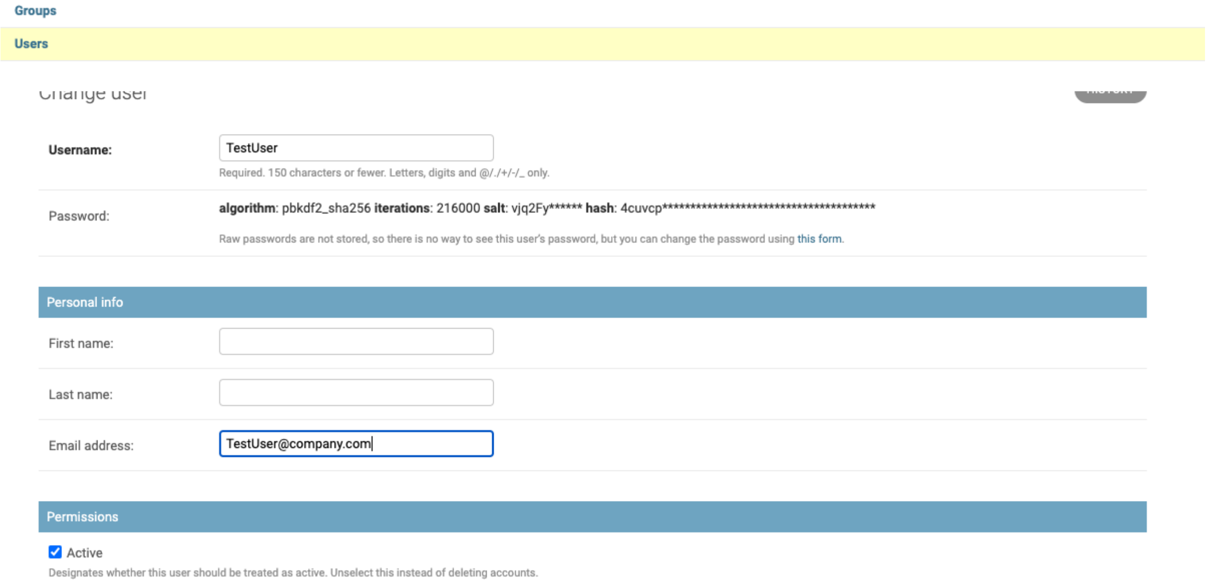
```
yangyubeideMacBook-Pro-4:Django_Blog yangyubei$ python3.7 manage.py createsuperuser
Username (leave blank to use 'yangyubei'):
Email address: yangyubei@gmail.com
Password:
Password (again):
Superuser created successfully.
```

Then we can go to website …/admin to login like this

ConyYang

2001Yyb.

We can also add a new user



Username: TestUser

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **algorithm**: pbkdf2_sha256 **iterations**: 216000 **salt**: vjq2Fy****** **hash**: 4cuvcp**************************************

Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name:

Last name:

Email address: TestUser@company.com

Permissions

☑ Active

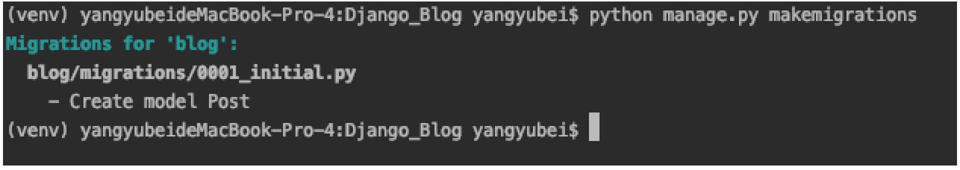
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

Chapter5 Database and Migrations

Make a new user model. Add custom fields. Make a post mode, a model which inherits from Django model class. Go to models.py under blog folder.

Set Foreign Key to author. If the user is deleted, we also want to delete their post.

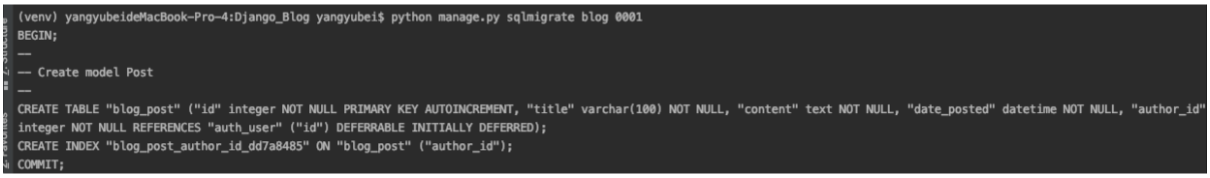
Now we make migrations after creating this model.



```
(venv) yangyubeideMacBook-Pro-4:Django_Blog yangyubei$ python manage.py makemigrations
Migrations for 'blog':
  blog/migrations/0001_initial.py
    - Create model Post
(venv) yangyubeideMacBook-Pro-4:Django_Blog yangyubei$
```

Now we can see a 0001_initial.py file under migrations.

Type in a line to see the actual SQL command that it would run. python manage.py sqlmigrate blog 0001



```
(venv) yangyubeideMacBook-Pro-4:Django_Blog yangyubei$ python manage.py sqlmigrate blog 0001
BEGIN;
--
-- Create model Post
--
CREATE TABLE "blog_post" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(100) NOT NULL, "content" text NOT NULL, "date_posted" datetime NOT NULL, "author_id"
integer NOT NULL REFERENCES "auth_user" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "blog_post_author_id_dd7a8485" ON "blog_post" ("author_id");
COMMIT;
```

Now we make migrations. python manage.py migrate.
Allow us to make changes to our database even after it's created and has data in that database.
So if we didn't have a way to run migrations, we would have to SQL code.

Run the shell to do some SQL queries. Query our user table.
python manage.py shell

```
>>> from blog.models import Post
>>> from django.contrib.auth.models import User
>>> User.objects.all()
<QuerySet [<User: ConyYang>, <User: TestUser>]>
>>> User.objects.first()
<User: ConyYang>
>>> User.objects.filter(username='TestUser')
<QuerySet [<User: TestUser>]>
>>> User.objects.filter(username='TestUser').first()
<User: TestUser>
>>> user = User.objects.filter(username='TestUser').first()
>>> user
<User: TestUser>
>>> user
<User: TestUser>
>>> user.id
2
>>> user.pk
2
>>> user = User.objects.get(id=1)
>>> user
<User: ConyYang>
```

We now have this user. So we can create a new post and make this user the author of this new post. First, we shouldn't have any post right now. So if run a query on the post model

```
>>> Post.objects.all()
<QuerySet []>
```

It returns an empty query.
Add a post. Make user as author

```
>>> post_1 = Post(title='Trip in Singapore', content='A wonderful journey. Nice food and weather.', author=user)
```

We don't specify the date for this post. But our model have a default date of the current date time.

```
>>> Post.objects.all()
 <QuerySet []>
```

We run the query again but it is still empty. It is because we create the post variable but we didn't actually save it to our database.

```
>>> post_1.save()
>>> Post.objects.all()
<QuerySet [<Post: Post object (1)>]>
```

Now we get a query set with one post object.

Now we modify the models.py class.

```
# return how this Post be printed out
def __str__(self):
    return self.title
```

We rerun the shell and now can see the title.

```
>>> Post.objects.all()
<QuerySet [<Post: Trip in Singapore>]>
```

Since we quit the shell. We need to redefine the user.

```
>>> user = User.objects.filter(username='ConyYang').first()
>>> user
<User: ConyYang>
>>> post_2 = Post(title='Trip in LA', content='I determined my future in this place', author=user)
>>> post_2.save()
>>> Post.objects.all()
<QuerySet [<Post: Trip in Singapore>, <Post: Trip in LA>]>
```

We can now view the post use queries.

```
>>> post = Post.objects.first()
>>> post.content
'A wonderful journey. Nice food and weather.'
>>> post.date_posted
datetime.datetime(2020, 10, 9, 7, 47, 11, 13516, tzinfo=<UTC>)
>>> post.author
<User: ConyYang>
>>> post.author.email
'yangyubei0218@gmail.com'
```

We now want to view all the posts written by a user. Django has a special query set to the user model. **Naming convention is the name of the related model then _set.**

```
>>> user
<User: ConyYang>
>>> user.post_set
<django.db.models.fields.related_descriptors.create_reverse_many_to_one_manager.<locals>.RelatedManager object at 0x7f92cc7694c0>
>>> user.post_set.all()
<QuerySet [<Post: Trip in Singapore>, <Post: Trip in LA>]>
```

We don’t need to put author as a variable like this. Django automatically knows the user. Also don’t need to run the save command.

```
>>> user.post_set.create(title='Trip in Brazil', content='I finally visted the 3rd largest waterfall in the world!!')
<Post: Trip in Brazil>
```

We can also make changes to these data in our admin page. First we grab these data and pass it to our views. We want to substitute the dummy data created in views.py running a query on our post model.
So, we first import the post model.

```
from .models import Post
```

```
context: Dict[str, List[Dict[str, str]]] = {
    'posts': Post.objects.all()
}
```

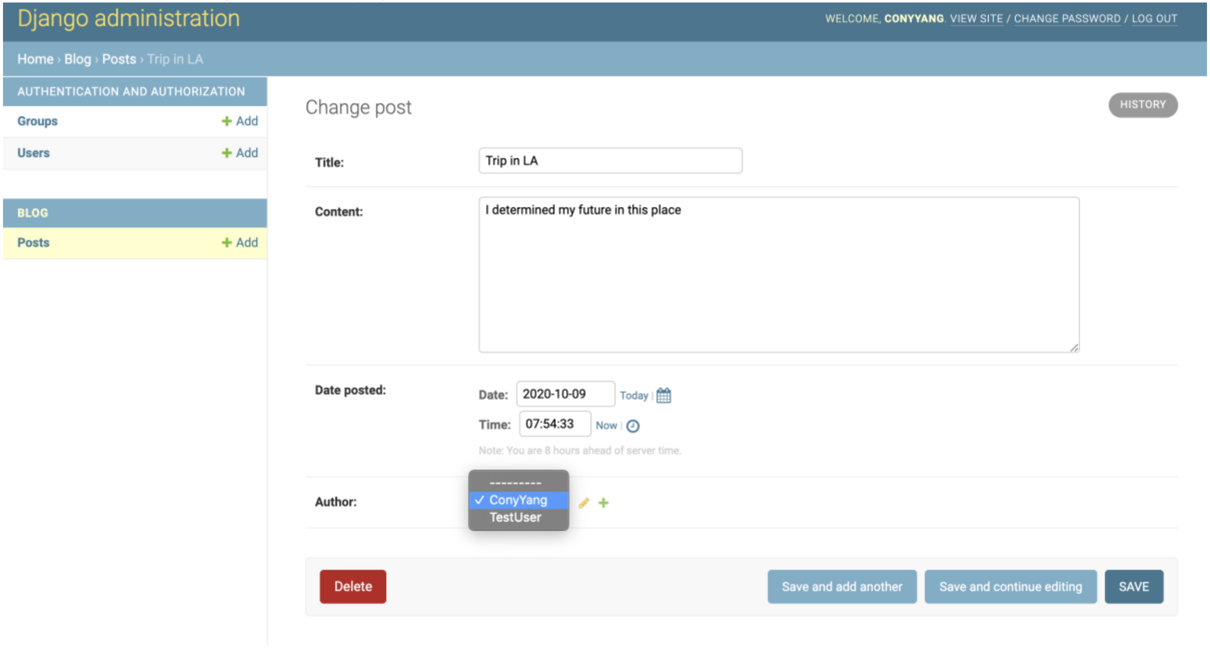
Now run server to see the change.
Change the format of the date in home.html

```
<small class="text-muted">{{ post.date_posted|date:"F d, Y" }}</small>
```

In admin.py, we can register our models to show on the admin page.

```
# Register your models here.
from .models import Post
admin.site.register(Post)
```

We now can see blogs in admin page.


Django administration
WELCOME, CONYYANG. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Blog › Posts › Trip in LA
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
BLOG
Posts + Add
HISTORY
Delete
Save and add another
Save and continue editing
SAVE

Change post

Title: Trip in LA

Content: I determined my future in this place

Date posted: Date: 2020-10-09 Today | Time: 07:54:33 Now |

Note: You are 8 hours ahead of server time.

Author: --------- ✓ ConyYang TestUser

Chapter 6 User Registration

The User account portion of our project is going to have its own forms, templates and routes.
Create a new app inside our project.
$ python manage.py startapp users
First need to add into our installed apps list in our project settings. Go to apps.py file under users folder.

```
class UsersConfig(AppConfig):
    name = 'users'
```

So we add this to the settings.py under Django_Blog.

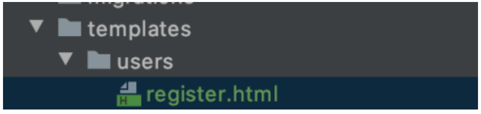
```
INSTALLED_APPS = [
    'users.apps.UsersConfig',
```

Now in views.py, create a form that going to be passed to the template. Create python classes and these classes generate HTML forms for us. Some classes already exist. We want a registration form so users can sign up for our site. Use the user creation form that already exists in Django.

```
from django.contrib.auth.forms import UserCreationForm
```

```
def register(request):
    form = UserCreationForm()
    return render(request, 'users/register.html', {'form': form})
```

Now we need to create a .html template for this form variable to pass in. Create the templates subfolder and the register.html.



In the <div>, we create a new form. Give this method equal to POST. Add a cross-site request forgery token. This will protect our form against certain attacks.

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

{% block content %}
    <div class="content-section">

        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legendn class="border-bottom mb-4"> Join Today</legendn>
                {{ form.as_p }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info" type="submit">Sign Up</button>
            </div>
        </form>

        <div class="border-top pt-3">
            <small class="text-muted">
                Already Have an Account?
                <a class="ml-2" href="#"></a>
            </small>
        </div>

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

This is coding part of the register.html.

Now we modify urls.py. Import users and add the path.

```
from users import views as user_views
```

Modify views.py. Add conditional POST request.
Check if the form is validate, grab the username. The validated username will be in form.cleaned_data() dictionary. Will be auto converted to python types.

```
def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            messages.success(request, f'Account created for {username} !')
            return redirect('blog-home')
    else:
        form = UserCreationForm()
    return render(request, 'users/register.html', {'form': form})
```

Use a flash message to show we receive a valid data. This sends one-time alerts to the template.
We first import this library.

```
from django.contrib import messages
```

There are a lot types of messages.

```
messages.debug()
messages.info()
messages.success()
messages.warning()
messages.error()
```

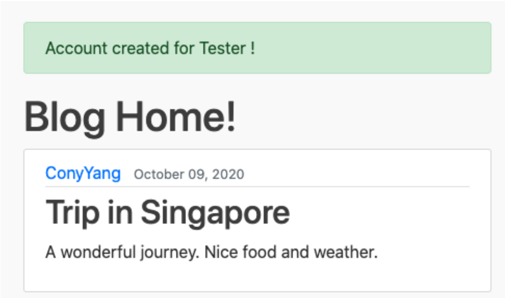
Then we redirect the user to homepage. Import the redirect function.

```
from django.shortcuts import render, redirect
```

The final thing is add the flashed message into base.html.

Add this line, now the user is actually created.

```
if form.is_valid():
    form.save()
```



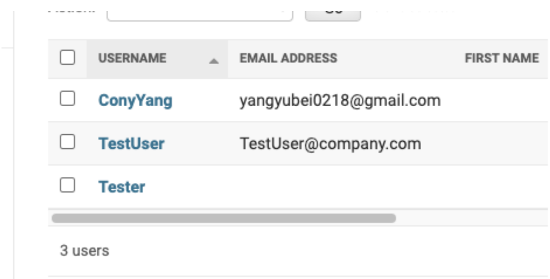
Account created for Tester !

Blog Home!

ConyYang October 09, 2020

Trip in Singapore

A wonderful journey. Nice food and weather.



☐	USERNAME ▲	EMAIL ADDRESS	FIRST NAME
☐	ConyYang	yangyubei0218@gmail.com	
☐	TestUser	TestUser@company.com	
☐	Tester		

3 users

Tester is created. (Password is test123123)
How do we add a form to our user creation form??? Create a new form that inherits our user creation form.
Add another file forms.py in users application directory.

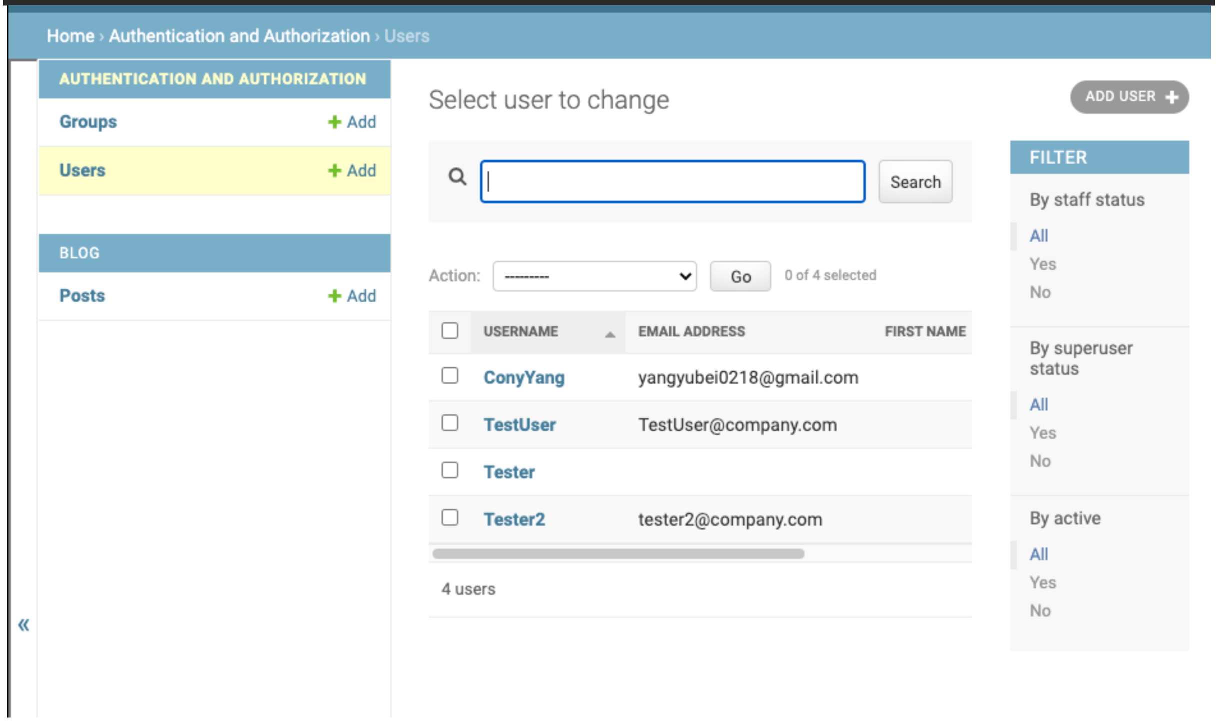
Create a new UserRegister class inherit from UserCreationFrom.

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

Give us nested namespace and keep these configurations in one place.

Go back to view.py and import this class. Replace the UserCreation form to UserRegisterForm.

```
if request.method == 'POST':
    form = UserRegisterForm(request.POST)
```



0 of 4 selected

☐	USERNAME ▲	EMAIL ADDRESS	FIRST NAME
☐	ConyYang	yangyubei0218@gmail.com	
☐	TestUser	TestUser@company.com	
☐	Tester		
☐	Tester2	tester2@company.com	

4 users

Install crispy forms.

```
(venv) yangyubeideMacBook-Pro-4:Django_Blog yangyubei$ pip install django-crispy-forms
Collecting django-crispy-forms
  Downloading
https://files.pythonhosted.org/packages/37/04/06760b426a3dd7eb3e43ed7bcb9d085670c1e60453f4abd56357c4f187ce/django_crispy_forms-1.9.2-py3-none-any.whl (108kB)
   100% |████████████████████████████████| 112kB 5.6MB/s
Installing collected packages: django-crispy-forms
Successfully installed django-crispy-forms-1.9.2
```

Add this installed app into settings.py.

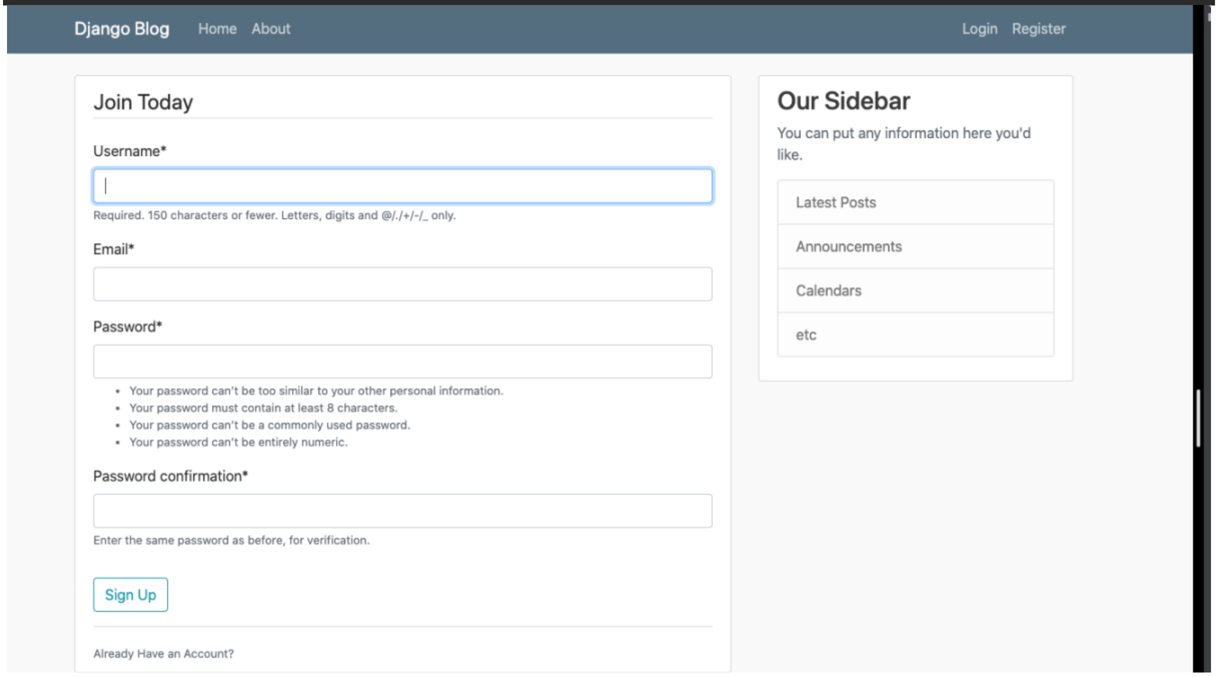
```
INSTALLED_APPS = [


   'users.apps.UsersConfig',
   'blog.apps.BlogConfig',
   'crispy_forms',
```

Tell crispy forms which css we want to use. Use bootstrap 4.

```
CRISPY_TEMPLATE_PACK = 'bootstrap4'
```

Add this crispy form into register.html.

```
{%  load crispy_forms_tags %}
{{ form|crispy }}
```



Join Today

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email*

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

Enter the same password as before, for verification.

Already Have an Account?

Our Sidebar

You can put any information here you'd like.

Now this style looks better.

Chapter 7 Login and Logout System

Import more views in urls.py.

```
from django.contrib.auth import views as auth_views
```

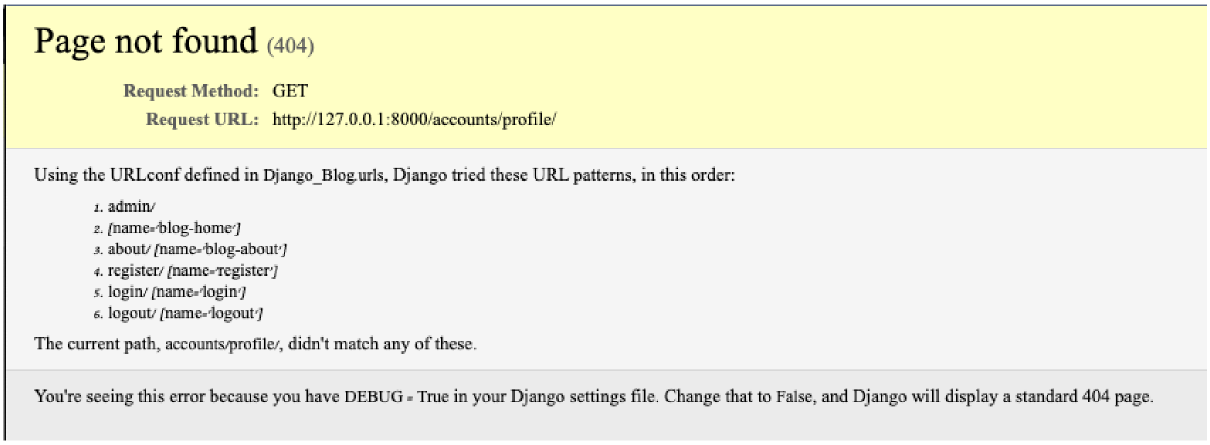
```
path('login/', auth_views.LoginView.as_view(), name='login'),
path('logout/', auth_views.LogoutView.as_view(), name='logout'),
```

pass argument to tell Django where we look for the path in as_View() function.

```
auth_views.LoginView.as_view(template_name='users/login.html'),
```

Add login and logout .html in users folder together with register.html.

If now we login, we don't have this user page. We better redirect this login to main page instead.



Page not found (404)

Request Method: GET
Request URL: http://127.0.0.1:8000/accounts/profile/

Using the URLconf defined in Django_Blog.urls, Django tried these URL patterns, in this order:

1. admin/
2. [name='blog-home']
3. about/ [name='blog-about']
4. register/ [name='register']
5. login/ [name='login']
6. logout/ [name='logout']

The current path, accounts/profile/, didn't match any of these.

You're seeing this error because you have DEBUG = True in your Django settings file. Change that to False, and Django will display a standard 404 page.

We add this line into settings.py.

```
LOGIN_REDIRECT_URL = 'blog-home'
```

Also modify the logout.html.

```
{% extends "blog/base.html" %}
{% block content %}
    <h2> You have been logged out</h2>
    <div class="border-top pt-3">
      <small class="text-muted">
        <a href="{% url 'login' %}">Login again?</a>
      </small>
    </div>
{% endblock %}
```

We want to change the sidebar to indicate if it is login or logged out.

Change the base.html in blog folder.
Django user has an attributed isloggedin? To help use check.

```
<div class="navbar-nav">
  {% if user.is_authenticated %}
    <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
  {% else  %}
   <a class="nav-item nav-link" href="{% url 'login' %}">Login</a>
   <a class="nav-item nav-link" href="{% url 'register' %}">Register</a>
  {% endif %}
</div>
```

Now the navigation bar is changed.

Create a page for a user profile.
Add a profile function in views.py

```
def profile(request):
  return render(request, 'users/profile.html')
```

Add profile.htm;

```
{% extends "blog/base.html" %}
{%  load crispy_forms_tags %}
{% block content %}
   <h1>{{ user.username }}</h1>
{% endblock %}
```

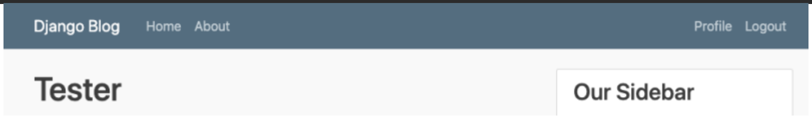
User is built into Django that represents the current logged in user.

Add this url to urlpatterns.

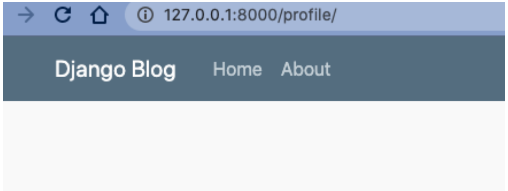
```
path('profile/', user_views.profile, name='profile'),
```

Add this link to the sidebar.

```
{% if user.is_authenticated %}
   <a class="nav-item nav-link" href="{% url 'profile' %}">Profile</a>
   <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
```



It is working.
But there is a bug.



We can easily get something blank like this. We need to add a check so user can access this page before he logs in.
Use a login_requires_decorator that Django provides for us.

We modify the views.py, include the library and decorator.

```
from django.contrib.auth.decorators import login_required
```

Decorator adds functionality to an existing function.

```
  @ login_required
def profile(request):
```

Now the page is not found if not login to see the profile. So wee need to change settings.py.

```
LOGIN_URL = 'login'
```

Login is the name that we gave to our URL pattern for the login route.
http://127.0.0.1:8000/login/?next=/profile/
This is a nice feature. If login we redirect to profile page.

Chapter 8 User Profile and Picture

Extend the current user model to add more fields. Create a new profile model that has 1-1 relationship to user.
Modify models.py under users folder.

```
from django.db import models
from django.contrib.auth.models import User
# Create your models here.



class Profile(models.Model):
  user = models.OneToOneField(User, on_delete=models.CASCADE)
  image = models.ImageField(default='default.jpg', upload_to='profile_pics')
```

CASCADE means if the user is deleted also delete the profile. Profile_pics is where the image is upload to.

We need migrations because we change the dataset.
We need to install pillow. It is a library for working with images in python.

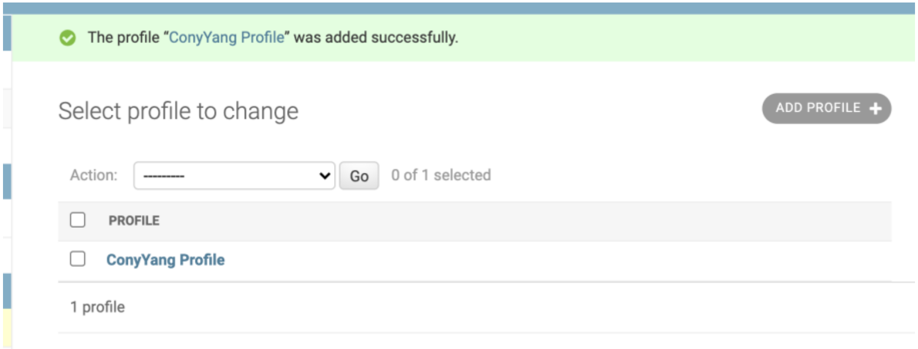
```
(venv) yangyubeideMacBook-Pro-4:Django_Blog yangyubei$ python manage.py makemigrations
Migrations for 'users':
  users/migrations/0001_initial.py
    - Create model Profile
```

```
(venv) yangyubeideMacBook-Pro-4:Django_Blog yangyubei$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, blog, contenttypes, sessions, users
Running migrations:
  Applying users.0001_initial... OK
```

Register this model within the admin page (users folder)of our app

```
# Register your models here.
from .models import Profile
admin.site.register(Profile)
```

Run server and add an image.



The profile "ConyYang Profile" was added successfully.

Select profile to change

□	PROFILE
□	ConyYang Profile

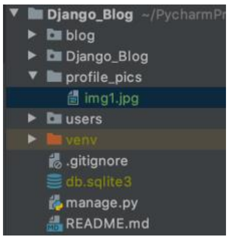
1 profile

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.filter(username='ConyYang')
>>> user = User.objects.filter(username='ConyYang').first()
>>> user
<User: ConyYang>
>>> user.profile
<Profile: ConyYang Profile>
>>> user.profile.image
<ImageFieldFile: profile_pics/img1.jpg>
>>> user.profile.image.width
300
>>> user.profile.image.height
300
>>> user.profile.image.size
17448
>>> user.profile.image.url
'/profile_pics/img1.jpg'
```

The last line gives us the location of the image, which can be used to display in the html.

```
>>> user = User.objects.filter(username='TestUser').first()
>>> user.profile.image.url
'/default.jpg'
```

There is also a default image.

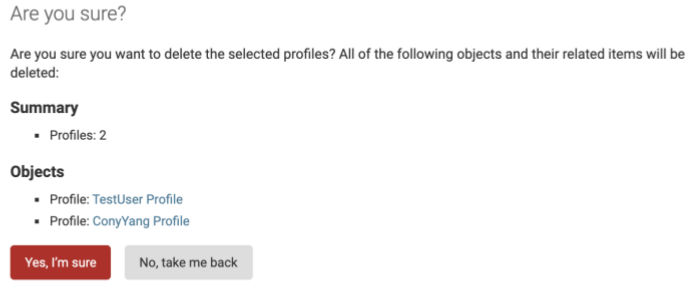


```
▼ Django_Blog ~/PycharmPr
  ▶ blog
  ▶ Django_Blog
  ▼ profile_pics
      img1.jpg
  ▶ users
  ▶ venv
    .gitignore
    db.sqlite3
    manage.py
    README.md
```

Here we have the profile_pics directory. We can change the settings to decide where these images actually saved. We modify it in settings.py.

MEDIA_ROOT is the full path where we like Django stored our uploaded files. MEIDA_URL is publica url of that directory.

We now delete the profiles.



Are you sure?

Are you sure you want to delete the selected profiles? All of the following objects and their related items will be deleted:

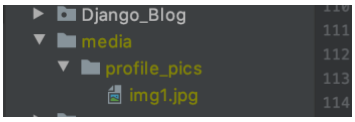
Summary

- Profiles: 2

Objects

- Profile: TestUser Profile
- Profile: ConyYang Profile

We have a new folder.

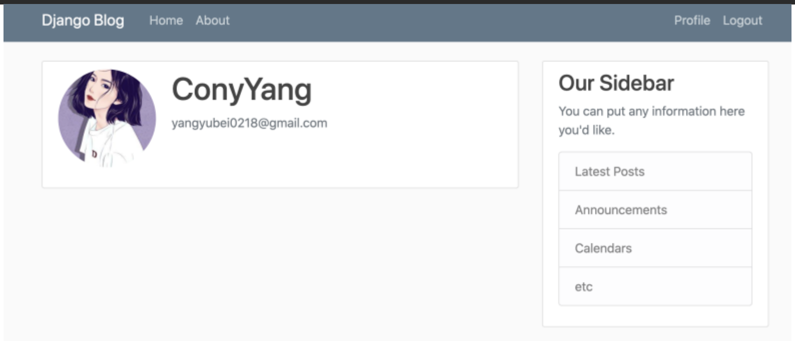


```
  ▶ Django_Blog
  ▼ media
    ▼ profile_pics
        img1.jpg
```

We want to modify the profile.html to make it display username, email address, image and has the function to modify them.

We also need to modify urls.py like this:

```
from django.conf import settings
from django.conf.urls.static import static


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



ConyYang

yangyubei0218@gmail.com

Our Sidebar

You can put any information here you'd like.

- Latest Posts
- Announcements
- Calendars
- etc

We also add default.jpg to media folder. We can check the image path in element analyze to choose the right folder.

Add a signals.py file in users folder.

```
from django.db.models.signals import post_save
from django.contrib.auth.models import User
from django.dispatch import receiver
from .models import Profile


@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)


@receiver(post_save, sender=User)
def save_profile(sender, instance, **kwargs):
    instance.profile.save()
```

Import the signals in app.py in users.

```
from django.apps import AppConfig


class UsersConfig(AppConfig):
    name = 'users'

    def ready(self):
        import users.signals
```

We now can register a new user. And the profile is created accordingly.

Chapter 9 Update User Profile

Create a form to update our user model.

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

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

We want to work with username and email in this updateform.
We also add a profile update form to update the image.

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

Import these 2 forms into views.py.

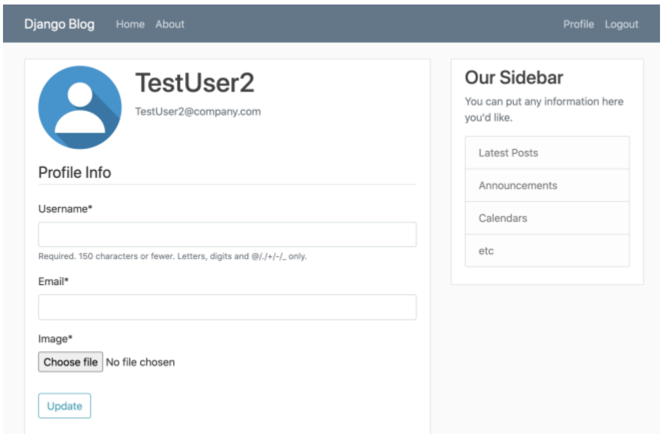
```
@ login_required
def profile(request):
    u_form = UserUpdateForm()
    p_form = ProfileUpdateForm()
    context = {
        'u_form': u_form,
        'p_form': p_form
    }
    return render(request, 'users/profile.html', context)
```

Instantiate these 2 forms and pass them into context.

Open profile.html and add these two forms.

```
<!-- FORM HERE -->
  <form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    <fieldset class="form-group">
      <legend class="border-bottom mb-4">Profile Info</legend>
      {{ u_form|crispy }}
      {{ p_form|crispy }}
    </fieldset>
    <div class="form-group">
      <button class="btn btn-outline-info" type="submit">Update</button>
    </div>
  </form >
```

We need to add the encryption type for the image.



TestUser2

TestUser2@company.com

Profile Info

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email*

Image*

Choose file No file chosen

Update

Our Sidebar

You can put any information here you'd like.

Latest Posts

Announcements

Calendars

etc

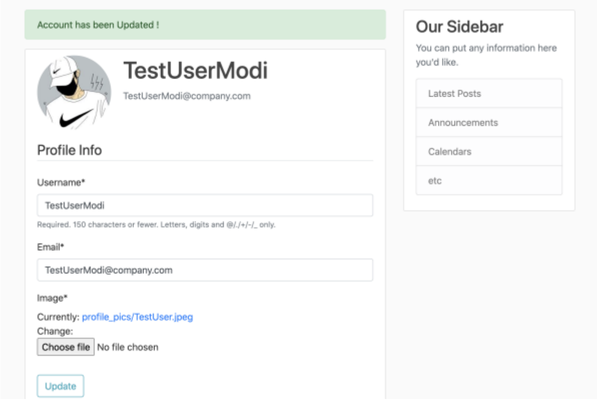
Now we have the form submission put here.

```
def profile(request):
    u_form = UserUpdateForm(instance=request.user)
    p_form = ProfileUpdateForm(instance=request.user.profile)
```

Modify the forms, so now the form also contains current information.

```
if request.method == 'POST':
    u_form = UserUpdateForm(request.POST, instance=request.user)
    p_form = ProfileUpdateForm(request.POST,
                               request.FILES,
                               instance=request.user.profile)

    if u_form.is_valid() and p_form.is_valid():
        u_form.save()
        p_form.save()
        messages.success(request, f'Account has been Updated !')
        return redirect('profile')
```

We also need to check if the form is valid and save them. If success, we give user some feedback and redirect them back to the profile page.



Account has been Updated !

TestUserModi

TestUserModi@company.com

Profile Info

Username*

TestUserModi

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email*

TestUserModi@company.com

Image*

Currently: profile_pics/TestUser.jpeg

Change:

Choose file No file chosen

Update

Our Sidebar

You can put any information here you'd like.

Latest Posts

Announcements

Calendars

etc

See this change in admin site.

How to resize the photo when upload them?

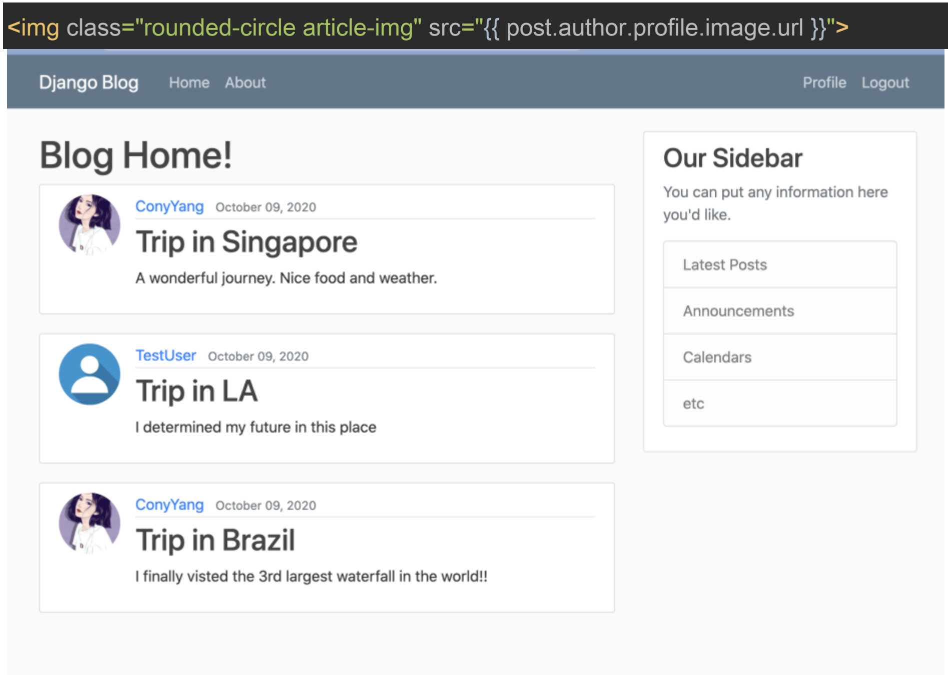
Add this save method to Profile class.

```
def save(self):
    super().save()
```

```
  img = Image.open(self.image.path)
  # resize to 300 pixels
  if img.height > 300 or img.width > 300:
      output_size = (300, 300)
      img.thumbnail(output_size)
      img.save(self.image.path)
```

We modify home.html to present profile photo.

```
<img class="rounded-circle article-img" src="{{ post.author.profile.image.url }}">
```



Django Blog Home About Profile Logout

Blog Home!

ConyYang October 09, 2020

Trip in Singapore

A wonderful journey. Nice food and weather.

TestUser October 09, 2020

Trip in LA

I determined my future in this place

ConyYang October 09, 2020

Trip in Brazil

I finally visted the 3rd largest waterfall in the world!!

Our Sidebar

You can put any information here you'd like.

Latest Posts

Announcements

Calendars

etc

Chapter 10 Create Update, and Delete Posts

Create a class PostlistView inherit from ListView. Add a model equal to Post.

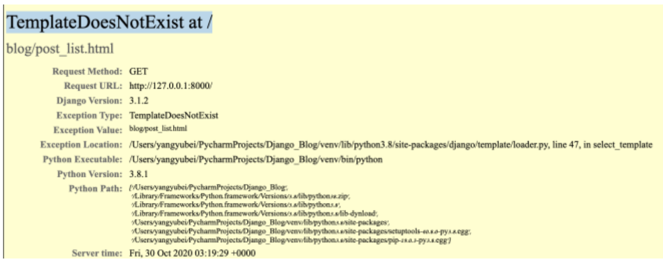
```
class PostListView(ListView):
    model = Post
```

Now modify urls.py.

```
from .views import PostListView


urlpatterns = [
    path('', PostListView.as_view(), name='blog-home'),
    path('about/', views.about, name='blog-about'),
]
```

Call the method ad_view().

If we run server now, we will get an error.



```
TemplateDoesNotExist at /
blog/post_list.html
      Request Method:  GET
         Request URL:  http://127.0.0.1:8000/
      Django Version:  3.1.2
      Exception Type:  TemplateDoesNotExist
     Exception Value:  blog/post_list.html
  Exception Location:  /Users/yangyubei/PycharmProjects/Django_Blog/venv/lib/python3.8/site-packages/django/template/loader.py, line 47, in select_template
   Python Executable:  /Users/yangyubei/PycharmProjects/Django_Blog/venv/bin/python
      Python Version:  3.8.1
         Python Path:  ['/Users/yangyubei/PycharmProjects/Django_Blog',
                        '/Library/Frameworks/Python.framework/Versions/3.8/lib/python38.zip',
                        '/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8',
                        '/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/lib-dynload',
                        '/Users/yangyubei/PycharmProjects/Django_Blog/venv/lib/python3.8/site-packages',
                        '/Users/yangyubei/PycharmProjects/Django_Blog/venv/lib/python3.8/site-packages/setuptools-40.8.0-py3.8.egg',
                        '/Users/yangyubei/PycharmProjects/Django_Blog/venv/lib/python3.8/site-packages/pip-19.0.3-py3.8.egg']
         Server time:  Fri, 30 Oct 2020 03:19:29 +0000
```

We can see it is looking for blog/post_list.html.
We add another two attribute in class.

```
template_name = 'blog/home.html' # <app>/<model>_<viewtype>.html

context_object_name = 'posts'
```

But now the posts are in reversed order. Change the order of our query making to the database.
Add this attribute

```
ordering = ['-date_posted']
```

Create Detailed view for each individual post.
Import Detail view also.

```
from django.views.generic import ListView, DetailView
```

Create new class PostDetailView and import this class into urls,py.

```
class PostDetailView(DetailView):

    model = Post
```

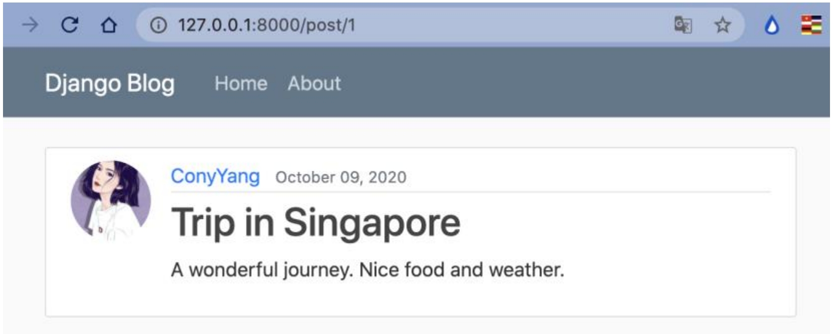
Add this new view to urls.py.

```
path('post/<int:pk>', PostDetailView.as_view(), name='post-detail'),
```

pk refers to primary key, and expect an integer value.
The template looking for by naming convention is:

```
# <app>/<model>_<viewtype>.html
```

Blog/post_detail.html
So we create a template of this name.



ConyYang October 09, 2020

Trip in Singapore

A wonderful journey. Nice food and weather.

Now we can see our first post.

Update the links in home page to make them to specific post. Modify in home.html.
Href should equal to this:

```
{% url 'post-detail' post.id %}
```

Create a create, update and delete view, so we can do posts on the frontend.

In views.py, import createview.

```
from django.views.generic import ListView, DetailView, CreateView
```

The only thing we need to provide is the field we want to be in that form.

```
class PostCreateView(CreateView):
    model = Post
```

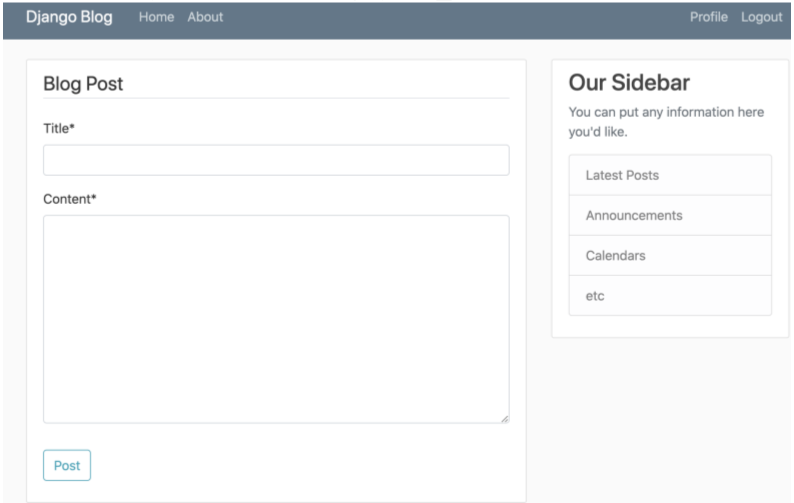
e.g. we want to fill in the content and title for the new post.

```
class PostCreateView(CreateView):
    model = Post
    fields = ['title', 'content']
```

Add these views in to urls.py

```
from .views import (
    PostListView,
    PostDetailView,
    PostCreateView)
path('post/new/', PostCreateView.as_view(), name='post-create'),
```

Create template under blog: post_form.html



Now the page is displayed. To avoid integrity error, Set the author of the form as current login user.

```
class PostCreateView(CreateView):
    model = Post
    fields = ['title', 'content']

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

To solve the redirect error, we need to define where we want to redirect after we submit the post.

Create a get_url_post method in models.py in Post class.

Reverse return the full URL to that route as a string (different from redirect.)

```
from django.urls import reverse
```

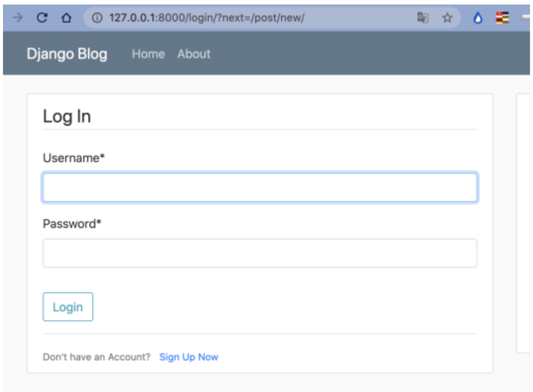
```
def get_absolute_url(self):
    return reverse('post-detail', kwargs={'pk': self.pk})
```

If we try to access post/new page not login, we need to redirect to the login page. Do this with function based views when we create our user profile page.
For function based views, we use login-required decorator.
Add this mixins in the views.py.

```
from django.contrib.auth.mixins import LoginRequiredMixin
```

```
class PostCreateView(LoginRequiredMixin, CreateView):
```

Can inherit from 2 classes. Now we cannot access new post if we don’t login.



Don't have an Account? Sign Up Now

How to update post from frontend?

```
from django.views.generic import (
    ListView,
    DetailView,
    CreateView,
    UpdateView)
```

This UpdateClass is very similar to Createview class.

```
class PostUpdateView(LoginRequiredMixin, UpdateView):
    model = Post
    fields = ['title', 'content']

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

```
from .views import (
    PostListView,
    PostDetailView,
    PostCreateView,
    PostUpdateView)
```

In urls.py

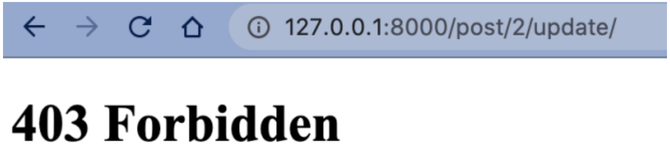
```
path('post/<int:pk>/update/', PostUpdateView.as_view(), name='post-update'),
```

Can go to this url to update
http://127.0.0.1:8000/post/4/update/

Add a check for author. If the login author is the person to update this page.
Import UserPassesTestMixin

```
from django.contrib.auth.mixins import LoginRequiredMixin, UserPassesTestMixin
class PostUpdateView(LoginRequiredMixin, UserPassesTestMixin, UpdateView):
    model = Post
    fields = ['title', 'content']

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

    def test_func(self):
        post = self.get_object()
        # check if is current user
        if self.request.user == post.author:
            # let allow post
            return True
        return False
```

If we now access other people's post to update we can see this 403 message.


127.0.0.1:8000/post/2/update/

403 Forbidden

Now we want to add the delete view function.

```
from django.views.generic import (
    ListView,
    DetailView,
    CreateView,
    UpdateView,
    DeleteView
)
```

The deleteview is very similar to detail view. We also want to require user to login first and the author is the logged in user.

```
class PostDeleteView(DeleteView, UserPassesTestMixin, UpdateView):
    model = Post
    fields = ['title', 'content']
    # the success_redirect url is the homepage.
    success_url = '/'

    def test_func(self):
        post = self.get_object()
        # check if is current user
        if self.request.user == post.author:
            # let allow post
            return True
        return False
```

The last thing is add this into urls.py.

```
from .views import (
    PostListView,
    PostDetailView,
    PostCreateView,
    PostUpdateView,
    PostDeleteView
)
```

```
path('post/<int:pk>/delete/', PostDeleteView.as_view(), name='post-delete'),
```

Create a template to ask user if we sure to delete the post. The if we submit the form the post will be deleted. The form will be called post_confirm_delete.html.

```
<div class="content-section">
    <form method="POST">
        {% csrf_token %}
        <fieldset class="form-group">
            <legend class="border-bottom mb-4"> Delete Blog Post </legend>
            <h2>Are you sure to delete the post? "{{ object.title }}"</h2>
        </fieldset>
        <div class="form-group">
            <button class="btn btn-outline-danger" type="submit">Confirm Delete</button>
            <a class="btn btn-outline-secondary" href="{% url 'post-detail' object.id %}">Cancel</a>
        </div>
```

```
   </form>
</div>
```

Now we want to add links and routes to redirect to these functions.
Add Create a new post in the navigation bar in base.html.

```
<a class="nav-item nav-link" href="{% url 'post-create' %}">New Post</a>
```

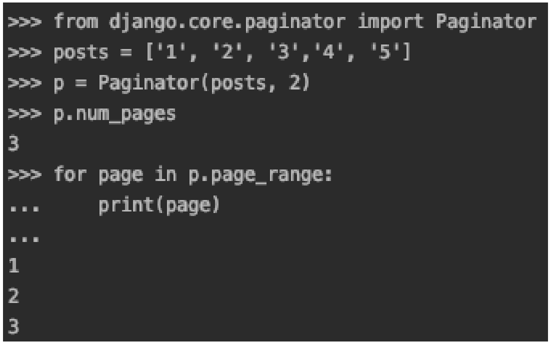
Add update and delete link in post_detail.html.

```
{% if object.author == user %}
   <div>
      <a class="btn btn-secondary btn-sm mt-1 mb-1" href="{% url 'post-update' object.id %}">Update</a>
      <a class="btn btn-danger btn-sm mt-1 mb-1" href="{% url 'post-delete' object.id %}">Delete</a>
   </div>
{% endif %}
```

You cannot see these 2 buttons if try to access post of another user.

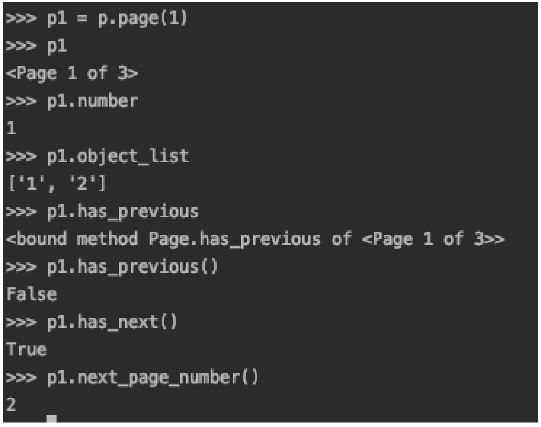
Chapter 11 Pagination

Add more post from other users.
Use some command lines to check the posts.



```
>>> from django.core.paginator import Paginator
>>> posts = ['1', '2', '3','4', '5']
>>> p = Paginator(posts, 2)
>>> p.num_pages
3
>>> for page in p.page_range:
...     print(page)
...
1
2
3
```

Check properties of a specific page.



```
>>> p1 = p.page(1)
>>> p1
<Page 1 of 3>
>>> p1.number
1
>>> p1.object_list
['1', '2']
>>> p1.has_previous
<bound method Page.has_previous of <Page 1 of 3>>
>>> p1.has_previous()
False
>>> p1.has_next()
True
>>> p1.next_page_number()
2
```

Add this paginate_by to PostListView class. But there is no link to other pages.

```
paginate_by = 2
```

Can use hard code url requests to get the page.
http://127.0.0.1:8000/?page=2
Change home.view to add pagination

```
{% if is_paginated %}

  {% if page_obj.has_previous %}
    <a class="btn btn-outline-info mb-4" href="?page=1">First</a>
    <a class="btn btn-outline-info mb-4" href="?page={{ page_obj.previous_page_number }}">Previous</a>
  {% endif %}

  {% for num in page_obj.paginator.page_range %}
    {% if page_obj.number == num %}
      <a class="btn btn-info mb-4" href="?page={{ num }}">{{ num }}</a>
    {% elif num > page_obj.number|add:'-3' and num < page_obj.number|add:'3' %}
      <a class="btn btn-outline-info mb-4" href="?page={{ num }}">{{ num }}</a>
    {% endif %}
  {% endfor %}

  {% if page_obj.has_next %}
    <a class="btn btn-outline-info mb-4" href="?page={{ page_obj.next_page_number }}">Next</a>
    <a class="btn btn-outline-info mb-4" href="?page={{ page_obj.paginator.num_pages }}">Last</a>
  {% endif %}

{% endif %}
```

Create a UserPostListView class.

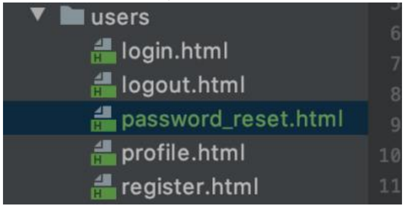
Chapter12 Email and password reset

Create URL pattern called password reset.

```
path('password-reset/',
     auth_views.PasswordResetView.as_view(template_name='users/password_reset.html'),
     name='password_reset'),
```

Provides for user to send reset email.

Add the template in users folder.



```
▼ users
    login.html
    logout.html
    password_reset.html
    profile.html
    register.html
```

Add new paths and templates.

```
path('password-reset/',
     auth_views.PasswordResetView.as_view(
       template_name='users/password_reset.html'),
     name='password_reset'),
path('password-reset/done/',
    auth_views.PasswordResetDoneView.as_view(
        template_name='users/password_reset_done.html'),
    name='password_reset_done'),

path('password-reset-confirm/<uidb64>/<token>/',
    auth_views.PasswordResetDoneView.as_view(
        template_name='users/password_reset_confirm.html'),
    name='password_reset_confirm'),
```

Implement a gmail server.