django

Django Introduction:

• Django is a Python-based web framework that allows you to quickly create efficient web applications.
• Django provides built-in features for everything including Django Admin Interface, default database – SQLlite3, etc.
• When you're building a website, you always need a similar set of components: a way to handle user authentication (signing up, signing in, signing out), a management panel for your website, forms, a way to upload files, etc. Django gives you ready-made components to use and that too for rapid development.

Django architecture

• Django is based on MVT (Model-View-Template) architecture. MVT is a software design pattern for developing a web application.
• MVT Structure has the following three parts –
  ◇ **Model:** Model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySql, Postgres).
  ◇ **View:** The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.
  ◇ **Template**: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

Installation of django

Install Python
Install Pycharm
Start django project in Pycharm

After creating a django project, your directory structure will like,

• djanog_project
  ◇ django_project
    ▪ __init__.py
    ▪ settings.py
    ▪ urls.py
    ▪ wsgi.py
  ◇ manage.py

To run the project:
  python manage.py runserver

creating new app

To create a new app:
  python manage.py startapp blog

  where blog is the name of the app that need to be created.

After creating blog app, your directory structure will be,
• djanog_project
  ◇ blog

- __init__.py
- admin.py
- apps.py
- migrations
  - __init.py
- models.py
- tests.py
- views.py

◇ db.sqlite3

◇ django_project
- __init__.py
- settings.py
- urls.py
- wsgi.py

◇ manage.py

Strating with a project:

Creating blog home

File Name:

• django_project
  ◇ blog
    - views.py

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

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

Now, this need to be mapped with urls. For this, create new file urls.py in blog folder

• django_project
  ◇ blog
    - views.py
    - urls.py

modify this urls.py as,

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

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

Also, map this url in maiin urls.py file

- django_project
  - django_project
    - urls.py

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

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

Creating About Page

File Name:
- django_project
  - blog
    - views.py

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

#Create new function
def home(request):
    return HttpResponse('<h1>Blog Home </h1>')

def about(request):
    return HttpResponse('<h1>Blog About </h1>')
```

- django_project
  - blog
    - views.py
    - urls.py

modify this urls.py as,

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

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

Templates

Create new directory call templates in blog app.

In templates, again creates a directory with the same name as app name

So, your project structure directory like,

- dianog_project
  - blog
    - templates
      - blog
    - __init__.py
    - admin.py
    - apps.py

- migrations
  - __init.py
- models.py
- tests.py
- views.py

◇ db.sqlite3
◇ django_project
- __init__.py
- settings.py
- urls.py
- wsgi.py

◇ manage.py

Create home.html in this folder

• djanog_project
◇ blog
- templates
  - blog
    → home.html

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        <h1> Blog Home! </h1>
    </body>

<\html>
```

Add blog app to list of installed app

Add app configuration to settings.py modules
App configuration is located inside of app.py file

• djanog_project
◇ blog
- apps.py

```
from django.apps import AppConfig

class BlogConfig(AppConfig):
    name = 'blog'
```

Add app configuration to settings.py file

- djanog_project
  - ◇ django_project
    - ▪ settings.py

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

Map home.html in views.py file

- django_project
  - ◇ blog
    - ▪ views.py

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

#Create new function
def home(request):
    return render(request, 'blog/home.html')

def about(request):
    return HttpResponse('<h1>Blog About </h1>')
```

Create about.html in this folder

- djanog_project
  - ◇ blog
    - ▪ templates
      - - blog

→ home.html
→ about.html

```
<html>
    <head>
        <title></title>
    </head>
    <body>
        <h1> About Page! </h1>
    </body>

<\html>
```

Update views.py file

- django_project
  - ◇ blog
    - ▪ views.py

```
from django.shortcuts import render

#Create new function
def home(request):
    return render(request, 'blog/home.html')

def about(request):
    return render(request, 'blog/about.html')
```

Passing data to template

Create dummy data called post in views.html

- django_project
  - ◇ blog
    - ▪ views.py

```
from django.shortcuts import render

posts = [
    {
        'author' :'Dimple',
        'title' : 'Blog Post 1',
        'content': 'First post content',
        'date_posted' : 'August 27, 2018',
    },
    {
        'author' :'Piyush',
        'title' : 'Blog Post 2',
        'content': 'Second post content',
        'date_posted' : 'August 28, 2018',
    }

]
#Create new function
def home(request):
    context = {
    'posts' : posts
    }
    return render(request, 'blog/home.html', context)

def about(request):
    return render(request, 'blog/about.html')
```

Print in home.html

- djanog_project
  - blog
    - templates
      - blog
        - → home.html

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
        {% for post in posts %}
            <h1> {{ post.title }}</h1>
            <p> By {{post.author}} on
{{ post.date_posted }} </p>
            <p> {{post.content}} </p>
        {% endfor %}
    </body>

<\html>
```

Use of if else statement

Changing title when specified

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html

```
<!DOCTYPE html>
<html>
    <head>
        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        {% for post in posts %}
            <h1> {{ post.title }}</h1>
            <p> By {{post.author}} on
{{ post.date_posted }} </p>
            <p> {{post.content}} </p>
        {% endfor %}
    </body>

<\html>
```

Do the same for about template

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html
        - → about.html

```
<html>
    <head>
        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        <h1> About Page! </h1>
    </body>

<\html>
```

Passing title to one of the view.

- django_project
  - blog
    - views.py

```
from django.shortcuts import render

posts = [
    {
        'author' :'Dimple',
        'title' : 'Blog Post 1',
        'content': 'First post content',
        'date_posted' : 'August 27, 2018',
    },
    {
        'author' :'Piyush',
        'title' : 'Blog Post 2',
        'content': 'Second post content',
        'date_posted' : 'August 28, 2018',
    }

]
#Create new function
def home(request):
    context = {
    'posts' : posts
    }
    return render(request, 'blog/home.html', context)

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

As, out home page and about page has lot of code similar, it would be better to write common code in one single file.

Create new template base.html in blog app.

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html
        - → about.html
        - → base.html

```
<html>
    <head>
        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        {% block content %}
        {% endblock %}
    </body>

<\html>
```

Overwrite content in home.html

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <h1> {{ post.title }}</h1>
        <p> By {{post.author}} on {{ post.date_posted }} </
p>
        <p> {{post.content}} </p>
    {% endfor %}
{% endblock content %}
```

Also modify about template

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → about.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <h1> About Page </h1>
    {% endfor %}
{% endblock content %}
```

Inserting bootstrap in base.html file

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html
        - → about.html
        - → base.html

```
<html>
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">

        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="https://
maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        <header class="site-header">
          <nav class="navbar navbar-expand-md navbar-dark
bg-steel fixed-top">
            <div class="container">
              <a class="navbar-brand mr-4" href="/">Django
Blog</a>
              <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarToggle" aria-
controls="navbarToggle" aria-expanded="false" aria-
label="Toggle navigation">
              <span class="navbar-toggler-icon"></span>
              </button>
              <div class="collapse navbar-collapse"
id="navbarToggle">
                <div class="navbar-nav mr-auto">
                  <a class="nav-item nav-link"
href="/">Home</a>
```

```
                    <a class="nav-item nav-link" href="/
about">About</a>
                </div>
                <!-- Navbar Right Side -->
                <div class="navbar-nav">
                  <a class="nav-item nav-link"
href="#">Login</a>
                  <a class="nav-item nav-link"
href="#">Register</a>
                </div>
              </div>
            </div>
          </nav>
        </header>

        <main role="main" class="container">
          <div class="row">
            <div class="col-md-8">
              {% block content %}{% endblock %}
            </div>
            <div class="col-md-4">
              <div class="content-section">
                <h3>Our Sidebar</h3>
                <p class='text-muted'>You can put any
information here you'd like.
                  <ul class="list-group">
                    <li class="list-group-item list-group-
item-light">Latest Posts</li>
                    <li class="list-group-item list-group-
item-light">Announcements</li>
                    <li class="list-group-item list-group-
item-light">Calendars</li>
                    <li class="list-group-item list-group-
item-light">etc</li>
                  </ul>
                </p>
              </div>
            </div>
          </div>
```

```
        </main>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS --
>
    <script src="https://code.jquery.com/
jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/
GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.12.9/umd/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/
bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5
+76PVCmYl" crossorigin="anonymous"></script>
    </body>
<\html>
```

There are few custom styles that are located in main.css file. The static files like css and java scripts need to be located in static directory within our app.

Create a directory called static in blog app. Also create subdirectory with same name as that of app.
Create a main.css file within this new directory.

- django_project
  - ◇ blog
    - ▪ static
      - - blog
        - → main.css

```css
body {
  background: #fafafa;
  color: #333333;
  margin-top: 5rem;
}

h1, h2, h3, h4, h5, h6 {
  color: #444444;
}

ul {
  margin: 0;
}

.bg-steel {
  background-color: #5f788a;
}

.site-header .navbar-nav .nav-link {
  color: #cbd5db;
}

.site-header .navbar-nav .nav-link:hover {
  color: #ffffff;
}

.site-header .navbar-nav .nav-link.active {
  font-weight: 500;
}

.content-section {
  background: #ffffff;
  padding: 10px 20px;
  border: 1px solid #dddddd;
  border-radius: 3px;
  margin-bottom: 20px;
}

.article-title {
```

```
  color: #444444;
}

a.article-title:hover {
  color: #428bca;
  text-decoration: none;
}

.article-content {
  white-space: pre-line;
}

.article-img {
  height: 65px;
  width: 65px;
  margin-right: 16px;
}

.article-metadata {
  padding-bottom: 1px;
  margin-bottom: 4px;
  border-bottom: 1px solid #e3e3e3
}

.article-metadata a:hover {
  color: #333;
  text-decoration: none;
}

.article-svg {
  width: 25px;
  height: 25px;
  vertical-align: middle;
}

.account-img {
  height: 125px;
  width: 125px;
```

```
  margin-right: 20px;
  margin-bottom: 16px;
}

.account-heading {
  font-size: 2.5rem;
}
```

Include this main.css file in base.html file.

First, we need to load static file using,
{% load static %}

Then, we can add css file
We can link the file using,
{% static 'blog/main.css' %}

- djanog_project
  - blog
    - templates
      - blog
        - → home.html
        - → about.html
        - → base.html

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">

        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="https://
maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
        <link rel = "stylesheet" type = "text/css" href =
"{% static 'blog/main.css' %}">


        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        <header class="site-header">
          <nav class="navbar navbar-expand-md navbar-dark
bg-steel fixed-top">
            <div class="container">
              <a class="navbar-brand mr-4" href="/">Django
Blog</a>
              <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarToggle" aria-
controls="navbarToggle" aria-expanded="false" aria-
label="Toggle navigation">
              <span class="navbar-toggler-icon"></span>
              </button>
```

```
              <div class="collapse navbar-collapse"
id="navbarToggle">
                <div class="navbar-nav mr-auto">
                  <a class="nav-item nav-link"
href="/">Home</a>
                  <a class="nav-item nav-link" href="/
about">About</a>
                </div>
                <!-- Navbar Right Side -->
                <div class="navbar-nav">
                  <a class="nav-item nav-link"
href="#">Login</a>
                  <a class="nav-item nav-link"
href="#">Register</a>
                </div>
              </div>
            </div>
          </nav>
        </header>

        <main role="main" class="container">
          <div class="row">
            <div class="col-md-8">
              {% block content %}{% endblock %}
            </div>
            <div class="col-md-4">
              <div class="content-section">
                <h3>Our Sidebar</h3>
                <p class='text-muted'>You can put any
information here you'd like.
                  <ul class="list-group">
                    <li class="list-group-item list-group-
item-light">Latest Posts</li>
                    <li class="list-group-item list-group-
item-light">Announcements</li>
                    <li class="list-group-item list-group-
item-light">Calendars</li>
                    <li class="list-group-item list-group-
item-light">etc</li>
```

```
                    </ul>
                  </p>
                </div>
              </div>
            </div>
          </main>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>
    </body>
<\html>
```

Adding some bootstrap in home.html

- djanog_project
  - ◇ blog
    - templates
      - blog
        - → home.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <article class="media content-section">
            <div class="media-body">
                <div class="article-metadata">
                    <a class="mr-2" href="#">
{{ post.author }}</a>
                    <small class="text-muted">
{{ post.date_posted }}</small>
                </div>
                <h2><a class="article-title" href="#">
{{ post.title }}</a></h2>
                <p class="article-content">
{{ post.content }}</p>
            </div>
        </article>
    {% endfor %}
{% endblock content %}
```

Now, In the base.html file, we have hard coded link. So if we need to change the location of file, then every time we are require to change this path. So, instead of using link to the file, we can use name given to that url as,

{% url 'blog-home' %}"

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html
        - → about.html
        - → base.html

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">

        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="https://
maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
        <link rel = "stylesheet" type = "text/css" href =
"{% static 'blog/main.css' %}">


        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        <header class="site-header">
          <nav class="navbar navbar-expand-md navbar-dark
bg-steel fixed-top">
            <div class="container">
              <a class="navbar-brand mr-4" href="{% url
'blog-home' %}">Django Blog</a>
              <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarToggle" aria-
controls="navbarToggle" aria-expanded="false" aria-
label="Toggle navigation">
              <span class="navbar-toggler-icon"></span>
              </button>
```

```
              <div class="collapse navbar-collapse"
id="navbarToggle">
                <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>
                <!-- Navbar Right Side -->
                <div class="navbar-nav">
                  <a class="nav-item nav-link"
href="#">Login</a>
                  <a class="nav-item nav-link"
href="#">Register</a>
                </div>
              </div>
            </div>
          </nav>
        </header>

        <main role="main" class="container">
          <div class="row">
            <div class="col-md-8">
              {% block content %}{% endblock %}
            </div>
            <div class="col-md-4">
              <div class="content-section">
                <h3>Our Sidebar</h3>
                <p class='text-muted'>You can put any
information here you'd like.
                  <ul class="list-group">
                    <li class="list-group-item list-group-
item-light">Latest Posts</li>
                    <li class="list-group-item list-group-
item-light">Announcements</li>
                    <li class="list-group-item list-group-
item-light">Calendars</li>
                    <li class="list-group-item list-group-
item-light">etc</li>
```

```
                    </ul>
                  </p>
                </div>
              </div>
            </div>
          </main>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></script>
    </body>
<\html>
```

Admin Page

For starting admin page, the url is

localhost:8000/admin

For creating admin user,

python manage.py createsuperuser

Directly typing this will give an error: no such table

As we have not created any database. To create default database, we need to make migration.

python manage.py makemigrations
python manage.py migrate

After running some migration, we can create a new user

Database and Migrations

• To work with database, django has its own built in ORM.
• A Django model is the built-in feature that Django uses to create tables, their fields, and various constraints.
• SQL (Structured Query Language) is complex and involves a lot of different queries for creating, deleting, updating or any other stuff related to database.
• Django models simplify the tasks and organize tables into models. Generally, each model maps to a single database table.
• Django lets us interact with its database models, i.e. add, delete, modify and query objects, using a database-abstraction API called ORM(Object Relational Mapper).
• Django already has a built in authentication system and already has a user model

Creating a database:

- django_project
  - ◇ blog
    - ▪ models.py

```
from django.db import models
from django.utils import timezone  #for default time
from django.contrib.auth.models import User

class Post(models.Model):
    title = models.CharField(max_length = 100)
    content = models.TextField()
    date_posted = models.DateTimeField(default =
timezone.now)          #auto_now = True will update the
date every time we modify the post
# auto_now_add = True will update only once when post is
created.
#using default we can change the date as we want
    #Since post model and user model will be having a
relationship as users are going to be author of the post.
So basically, this will be one to many relationship.
    author = models.ForeignKey(User, on_delete =
models.CASCADE)
```

In order to run the changes made in database, run the migrations

python manage.py makemigrations
python manage.py migrate

To view the SQL code, run the following command before migrate.

python manage.py sqlmigrate blog ooo1

blog - app name
0001 - migration number

migrate is useful because it allows us to chage the database even after it created and has data in that database.

To interact with model, we will make use of shell. We can run the shell by using

python manage.py shell

```
from django.contrib.auth.models import User
from blog.models import Post

User.objects.all()
#This will print all the users in database User
output: < QuerySet [<User:Dimple>, <User:Piyush>] >

User.objects.first()
# This will return the first user
output: <User:Dimple>

User.objects.filter(username = 'Dimple')
# This will return all the rows with username = Dimple
output: < QuerySet [<User:Dimple>]>

User.objects.filter(username = 'Dimple').first()
#This will return first user with username = Dimple
output: <User:Dimple>

user = User.objects.filter(username = 'Dimple').first()
# The user is captured in user variable

user.id
output: 1

user.pk #pk stands for primary key
output: 1

user = User.objects.get(id = 1)
user
output : <User:Dimple>

#Creating new post
Post.objects.all()
output : <QuerySet[]>

post_1 = Post(title = 'Blog 1', content = 'First Post
Content!', author = user)
```

```
Post.objects.all()
output : <QuerySet[]>
# since post_1 is not saved hence there is no post

post_1.save()
Post.objects.all()
output : <QuerySet [<Post:Post object(1)>]>
# This post model is not descriptive In order to have it
descriptive, we need to user dunder str method
```

Add str method to str:

- django_project
  - ◇ blog
    - ▪ models.py

```
from django.db import models
from django.utils import timezone  #for default time
from django.contrib.auth.models import User

class Post(models.Model):
    title = models.CharField(max_length = 100)
    content = models.TextField()
    date_posted = models.DateTimeField(default =
timezone.now)          #auto_now = True will update the
date every time we modify the post
# auto_now_add = True will update only once when post is
created.
#using default we can change the date as we want
    #Since post model and user model will be having a
relationship as users are going to be author of the post.
So basically, this will be one to many relationship.
    author = models.ForeignKey(User, on_delete =
models.CASCADE)

    def __str__(self):
        return self.title
```

python manage.py shell

```
from django.contrib.auth.models import User
from blog.models import Post


Post.objects.all()
output : <QuerySet [<Post:Blog 1>]>

user = User.objects.filter(username = 'Dimple').first()
post_2 = Post(title = 'Blog 2', content = 'Second Post
Content!', author_id = user.id)
post_2.save()

Post.objects.all()
output : <QuerySet [<Post:Blog 1>,<Post:Blog 2>]>

post = Post.objects.first()
post.content
output : 'First Post Content!'

post.date_posted
output : .......

post.author
output : <User : Dimple>

post.auther.email
# Getting an information from foreign key
'dimplepatel982@gmail.com'

# the related model can be accessed using .modelname_set
user.post_set
# Gives output of all the post written by the user

user.post_set.all()
output : <QuerySet [<Post:Blog 1>,<Post:Blog 2>]>

user.post_set.create(title = 'Blog 3', content = 'Third
Post Content')
```

```
# we didn't specify the author name because django knows
that the post is created for the user variable. Also we
don't need to save, it automatically saves in database.

Post.objects.all()
output : <QuerySet [<Post:Blog 1>,<Post:Blog 2>, <Post:Blog
3>]>
```

Making use of these query inorder to fetch data from database

- django_project
  - ◇ blog
    - ▪ views.py

first import post model

```
from django.shortcuts import render
from .models import Post

#Create new function
def home(request):
    context = {
    'posts' : Post.objects.all()
    }
    return render(request, 'blog/home.html', context)

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

To change the format of date, we can use date filter.

It can be used by using verticle line

post.date_posted|date:"F d, Y"

- djanog_project
  - ◇ blog
    - ▪ templates
      - - blog
        - → home.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <article class="media content-section">
            <div class="media-body">
                <div class="article-metadata">
                    <a class="mr-2" href="#">
{{ post.author }}</a>
                    <small class="text-muted">
{{ post.date_posted|data: "F d, Y" }}</small>
                </div>
                <h2><a class="article-title" href="#">
{{ post.title }}</a></h2>
                <p class="article-content">
{{ post.content }}</p>
            </div>
        </article>
    {% endfor %}
{% endblock content %}
```

To add Post model in admin site, we need to register that model into admin page.

- django_project
  - ◇ blog
    - ▪ admin.py

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

User Registration

Create Registration Page

The user account of the project is going to have its own form, templates and routes and hence it going to be separate from blog app.
Create new app for the project users

python manage.py startapp users

File structure will be

- django_project
  - ◇ blog
  - ◇ users

Add this app to installed app in setting.py file

- djanog_project
  - ◇ django_project
    - ▪ settings.py

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

Open views.py in users app

- django_project
  - ◇ blog
  - ◇ users
    - ▪ views.py

When we write some model, django automatically create some html templates for us. and some classes are already exists. User registration form is already exists in django. we can import this form using

from django.contrib.auth.forms import UserCreationForm

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

def register(request):
    form = UserCreationForm() #creating instance of this class
    return render(request, 'users/register.html', {'form':form})
```

Create template register.html

create new folder called template in users app. create subfolder users in this templates folder. Then create register.html file.

- django_project
  - ◇ blog
  - ◇ users
    - ▪ templates
      - - users
        - → register.html

Even within user app, we can user blog base.html file.
We need to add {% csrf_token %} to work form.

use fieldset tag which is used to group related element in a form.

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <div class = "content-section">
            <form method = "POST">
                {% csrf_token %}
                    <fieldset class = "form-group">
                        <legend class = "border-bottom
mb-4">
                            Join Today
                        </legend>
                        {{ form }}
                    </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 = "#">Sign In</a>
                </small.
            </div>
        </div>
    {% endfor %}
{% endblock content %}
```

Create URL patten for this register file.

- django_project
  - django_project
    - urls.py

```
from django.contrib import admin
from django.urls import path, include
from users import views as user_views

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

Use as_p tag that will render the form and paragraph tags and it will split this into few more lines

Open register.html and make changes of
form.as_p

- django_project
  - ◇ blog
  - ◇ users
    - ▪ templates
      - - users
        - → register.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <div class = "content-section">
            <form method = "POST">
                {% csrf_token %}
                    <fieldset class = "form-group">
                        <legend class = "border-bottom
mb-4">
                            Join Today
                        </legend>
                        {{ 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 = "#">Sign In</a>
                </small.
            </div>
        </div>
    {% endfor %}
{% endblock content %}
```

Handle the post request for this register page

If we get a post request, the we will validate this form data and if its get then we simply display the form.

- django_project
  - ◇ blog
  - ◇ users
    - ▪ views.py

Flash message is an easy way to show one time alert to a template that will be displayed on once and will be disappeared in the

next request. we can import this using

from django.contrib import messages

There are different types of messages like

messages.debug
messages.info
messages.success
messages.error
messages.warning

```
from django.shorcuts import renders, redirect
from django.contrib.auth.forms import UserCreationForm
from django.contrib import messages

def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST) # create a
form that has post data
        # validating form
        if form.is_valid():
            #grabing username
            username = form.cleaned_data.get('username')
#the validated form data will be in form.cleaned_data
dictionary
            #using flash message to show that valid data
is recieved.
            messages.success(request, f'Account created
for {username}!')
            # redirecting user to different form. For this
we need to import redirect
            retur redirect('blog-home')

    else:
        form = UserCreationForm() #creating instance of
this class
    return render(request, 'users/register.html',
{'form':form})
```

We haven't updated out template to show this flash message. Putting this to base.html page so that any flash message is generated, it will be displayed.

- djanog_project
  - blog
    - templates
      - blog
        - → home.html
        - → about.html
        - → base.html

message.tags is used to grab the type of message like warning, success, error etc.

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">

        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="https://
maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
        <link rel = "stylesheet" type = "text/css" href =
"{% static 'blog/main.css' %}">


        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        <header class="site-header">
          <nav class="navbar navbar-expand-md navbar-dark
bg-steel fixed-top">
            <div class="container">
              <a class="navbar-brand mr-4" href="{% url
'blog-home' %}">Django Blog</a>
              <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarToggle" aria-
controls="navbarToggle" aria-expanded="false" aria-
label="Toggle navigation">
              <span class="navbar-toggler-icon"></span>
              </button>
```

```
              <div class="collapse navbar-collapse"
id="navbarToggle">
                <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>
                <!-- Navbar Right Side -->
                <div class="navbar-nav">
                  <a class="nav-item nav-link"
href="#">Login</a>
                  <a class="nav-item nav-link"
href="#">Register</a>
                </div>
              </div>
            </div>
          </nav>
        </header>

        <main role="main" class="container">
          <div class="row">
            <div class="col-md-8">
                {% if messages %}
                    {% for message in messages %}
                        <div class = "alert alert-
{{ message.tags}}">
                            {{ message }}
                        </div>
                    {% endfor %}
                {% endif %}
              {% block content %}{% endblock %}
            </div>
            <div class="col-md-4">
              <div class="content-section">
                <h3>Our Sidebar</h3>
                <p class='text-muted'>You can put any
information here you'd like.
                  <ul class="list-group">
```

```
                    <li class="list-group-item list-group-
item-light">Latest Posts</li>
                    <li class="list-group-item list-group-
item-light">Announcements</li>
                    <li class="list-group-item list-group-
item-light">Calendars</li>
                    <li class="list-group-item list-group-
item-light">etc</li>
                  </ul>
                </p>
              </div>
            </div>
          </div>
        </main>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS --
>
    <script src="https://code.jquery.com/
jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/
GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.12.9/umd/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/
bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5
+76PVCmYl" crossorigin="anonymous"></script>
    </body>
<\html>
```

If the form data is valid then save it to user database by using form.save()

- django_project
  - ◇ blog
  - ◇ users
    - ▪ views.py

```
from django.shorcuts import renders, redirect
from django.contrib.auth.forms import UserCreationForm
from django.contrib import messages

def register(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST) # create a
form that has post data
        # validating form
        if form.is_valid():
            form.save() # This will automatically hased
the password and save the data to database.
            #grabing username
            username = form.cleaned_data.get('username')
#the validated form data will be in form.cleaned_data
dictionary
            #using flash message to show that valid data
is recieved.
            messages.success(request, f'Account created
for {username}!')
            # redirecting user to different form. For this
we need to import redirect
            retur redirect('blog-home')

    else:
        form = UserCreationForm() #creating instance of
this class
    return render(request, 'users/register.html',
{'form':form})
```

This will save the user data but as email is not there in form, the user will be saved without email address.
So we need to add email field to registration form.
In order to add new field, we need to add in form itself. we can not have in other form.
So, for this we create a new form that inherits the usercreation form
For this we first create a new file where we can put this.

Create form.py file in user application

- django_project
  - users
    - forms.py

Class meta gives us a nested namespace for configurations and keep the configuration in one place. and within the configuration, we are saying that the model that would be affected is the user model
For example, when we do a form.save() its going to save this user model and the fields that we have here in this list are the fields that we want in the form and in what order

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

class UserRegisterForm(UserCreationForm): # inheriting
UserCreationForm
    email = forms.EmailField()

    class Meta:
        #specifing the model that we need to interact with
        model = User # model is user because whenever the
form is validated, it going to create a new user
        fields = ['username', 'email', 'password1',
'password2']
        # This is the field that are going to be shwon on
our form
```

Now use this form in a view instead of the usercreation form

- django_project
  - ◇ blog
  - ◇ users
    - ▪ views.py

Importing the form that we just created

from .forms import UserRegisterForm

and replace UserCreationForm with UserRegisterForm

```
from django.shorcuts import renders, redirect
from django.contrib import messages
from .forms import UserRegisterForm

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST) # create a
form that has post data
        # validating form
        if form.is_valid():
            form.save() # This will automatically hased
the password and save the data to database.
            #grabing username
            username = form.cleaned_data.get('username')
#the validated form data will be in form.cleaned_data
dictionary
            #using flash message to show that valid data
is recieved.
            messages.success(request, f'Account created
for {username}!')
            # redirecting user to different form. For this
we need to import redirect
            return redirect('blog-home')

    else:
        form = UserRegisterForm() #creating instance of
this class
    return render(request, 'users/register.html',
{'form':form})
```

Give style to form

This can be done by using crispy form in django. This will style our form in a bootstrap fashion and also there are other CSS framworks that we can use with crispy form as well.
First install the crispy form

pip install django-crispy-forms

we need to tell django that this is an installed app

add crispy_forms

also add which css framework we need to use

add CRISPY_TEMPLATE_PACK = ‘bootstrap4’ at the bottom

- djanog_project
  - ◇ django_project
    - ▪ settings.py

```
import os

# Build paths inside the project like this: os.path.join
(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath
(__file__)))


# Quick-start development settings - unsuitable for
production
# See https://docs.djangoproject.com/en/3.0/howto/
deployment/checklist/

# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = 'ezmx*&72_-a42@i##kc(up7vzp6qqha1=d)+=^@@w3-
_=5w!r)'

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

ALLOWED_HOSTS = []


# Application definition

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

MIDDLEWARE = [
```

```
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'web_learning.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',

'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'web_learning.wsgi.application'


# Database
```

```
# https://docs.djangoproject.com/en/3.0/ref/settings/
#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-
password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimil
arityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValid
ator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordVali
dator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordVal
idator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'

CRISPY_TEMPLATE_PACK = 'bootstrap4'
```

Load crispy form tag

- django_project
  - ◇ blog
  - ◇ users
    - ▪ templates
      - - users
        - → register.html

{% load crispy_forms _tags %}

now we can use the crispy filter on any of our forms. Now we no longer need our as_p method since crispy form will take care of formating

form|crispy

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags    %}
{% block content %}
        <div class = "content-section">
            <form method = "POST">
                {% csrf_token %}
                    <fieldset class = "form-group">
                        <legend class = "border-bottom
mb-4">
                            Join Today
                        </legend>
                        {{ form|crispy }}
                    </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 = "#">Sign In</a>
                </small.
            </div>
        </div>
{% endblock content %}
```

Login and Logout System

Creating login and logout templates

Using default login views. Importing this login and logout views within our projects URLs modules.

- django_project
  - ◇ django_project
    - ▪ urls

Importing views that provide login and logout functionality

from django.contrib.auth import views

The inbuilt view of login and logout in django doesn't handles templates. we need to create them.

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

urlpatterns = [
    path('admin/',admin.site.urls),
    path('register/', user_views.register, name =
'register'),
    path('login/', auth_views.LoginView.as_view(), name =
'login'),
    path('logout/', auth_views.LogoutView.as_view(), name
= 'logout'),
    path('blog/', include('blog.urls')),
]
```

When we run our server, it will gives an error templateDoesNotExists.
ExceptionType : registration/login.html
This means it is looking for login.html page.

But we couldn't create registration directory inside of our templates and create a login.html template there.
but we can create login template inside our users templates and tell django to look there instead
For doing this, we can pass parameter in as_view() as template_name = 'users/login.html' so that it will search there for login template.

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

urlpatterns = [
    path('admin/',admin.site.urls),
    path('register/', user_views.register, name =
'register'),
    path('login/', auth_views.LoginView.as_view
(template_name = 'users/login.html'), name = 'login'),
    path('logout/', auth_views.LogoutView.as_view
(template_name = 'users/login.html'), name = 'logout'),
    path('blog/', include('blog.urls')),
]
```

Create login.html template

- djnago_project
  - ◇ users
    - ▪ templates
      - users
        - → login.html

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags    %}
{% block content %}
        <div class = "content-section">
            <form method = "POST">
                {% csrf_token %}
                    <fieldset class = "form-group">
                        <legend class = "border-bottom
mb-4">
                            Log In
                        </legend>
                        {{ form|crispy }}
                    </fieldset>
                    <div class = "form-group">
                        <button class = "btn btn-outline-
info" type = "submit">
                            Login
                        </button>
                    </div>
            </form>
            <div class = "border-top pt-3">
                <small class = "text-muted">
                    Need an account? <a class = "ml-2"
href = "{% url 'register' %}">Sign Up Now</a>
                </small.
            </div>
        </div>
{% endblock content %}
```

Also, add link for login in register page

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags    %}
{% block content %}
        <div class = "content-section">
            <form method = "POST">
                {% csrf_token %}
                    <fieldset class = "form-group">
                        <legend class = "border-bottom
mb-4">
                            Join Today
                        </legend>
                        {{ form|crispy }}
                    </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 = "{% url 'login' %}">Sign In</a>
                </small.
            </div>
        </div>
{% endblock content %}
```

Now, when we login with correct username and password, we will get an 404 error which means that it looks for a route that doesn't exists.
It trying to access a URL that doesn't have a view attached to it.

URL it trying to access is accounts/profile/

we can modify this location using settings.py file

- djanog_project
  - django_project
    - settings.py

At the bottom, add

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

```
import os

# Build paths inside the project like this: os.path.join
(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath
(__file__)))


# Quick-start development settings - unsuitable for
production
# See https://docs.djangoproject.com/en/3.0/howto/
deployment/checklist/

# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = 'ezmx*&72_-a42@i##kc(up7vzp6qqha1=d)+=^@@w3-
_=5w!r)'

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

ALLOWED_HOSTS = []


# Application definition

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

MIDDLEWARE = [
```

```
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'web_learning.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',

'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'web_learning.wsgi.application'


# Database
```

```
# https://docs.djangoproject.com/en/3.0/ref/settings/
#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-
password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimil
arityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValid
ator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordVali
dator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordVal
idator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'

CRISPY_TEMPLATE_PACK = 'bootstrap4'

LOGIN_REDIRECT_URL = 'blog-home'
```

Now, after login, it will redirect the blog home page.

Now after registration, we were redirecting to the home page. Instead of it, we will now redirect to the login page.

- django_project
  - blog
  - users
    - views.py

```
from django.shorcuts import renders, redirect
from django.contrib import messages
from .forms import UserRegisterForm

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST) # create a
form that has post data
        # validating form
        if form.is_valid():
            form.save() # This will automatically hased
the password and save the data to database.
            #grabing username
            username = form.cleaned_data.get('username')
#the validated form data will be in form.cleaned_data
dictionary
            #using flash message to show that valid data
is recieved.
            messages.success(request, f'Your account has
been created! You are now able to log in.')
            # redirecting user to different form. For this
we need to import redirect
            return redirect('login')

    else:
        form = UserRegisterForm() #creating instance of
this class
    return render(request, 'users/register.html',
{'form':form})
```

Now, if we don't pass
auth_views.LogoutView.as_view(**template_name = 'users/login.html'**),
template name here, then it will give logged out page of django admin. Then it gives link to login again which will redirect us on admin login page. so we need to create logout template in templates directory.

so we have to mention the template name.

Create logout.html template

- djnago_project
  - users
    - templates

- users
  → logout.html

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

Now, we have a login, logout and register page, we will change navigation bar.

• djanog_project
  ◇ blog
    ▪ templates
      - blog
        → home.html
        → about.html
        → base.html

if the user is already logged in, then we don't want login and register option in navbar.
djanog provides a user variable that contains the current user and it has an attribute called is_authenticated that allows us to check if the user is currently logged in or not.

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">

        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="https://
maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
        <link rel = "stylesheet" type = "text/css" href =
"{% static 'blog/main.css' %}">


        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        <header class="site-header">
          <nav class="navbar navbar-expand-md navbar-dark
bg-steel fixed-top">
            <div class="container">
              <a class="navbar-brand mr-4" href="{% url
'blog-home' %}">Django Blog</a>
              <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarToggle" aria-
controls="navbarToggle" aria-expanded="false" aria-
label="Toggle navigation">
              <span class="navbar-toggler-icon"></span>
              </button>
```

```
              <div class="collapse navbar-collapse"
id="navbarToggle">
                <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>
                <!-- Navbar Right Side -->
                <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>
              </div>
            </div>
          </nav>
        </header>

        <main role="main" class="container">
          <div class="row">
            <div class="col-md-8">
                {% if messages %}
                    {% for message in messages %}
                        <div class = "alert alert-
{{ message.tags}}">
                            {{ message }}
                        </div>
                    {% endfor %}
                {% endif %}
              {% block content %}{% endblock %}
            </div>
            <div class="col-md-4">
```

```
              <div class="content-section">
                <h3>Our Sidebar</h3>
                <p class='text-muted'>You can put any
information here you'd like.
                  <ul class="list-group">
                    <li class="list-group-item list-group-
item-light">Latest Posts</li>
                    <li class="list-group-item list-group-
item-light">Announcements</li>
                    <li class="list-group-item list-group-
item-light">Calendars</li>
                    <li class="list-group-item list-group-
item-light">etc</li>
                  </ul>
                </p>
              </div>
            </div>
          </div>
        </main>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS --
>
    <script src="https://code.jquery.com/
jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/
GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.12.9/umd/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/
bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5
+76PVCmYl" crossorigin="anonymous"></script>
    </body>
<\html>
```

Putting restriction on certain routes so that you can only access if you are logged in.

create a route for user profile that can be accessed only after they login.

- django_project
  - ◇ blog
  - ◇ users
    - ▪ views.py

create profile view

```
from django.shorcuts import renders, redirect
from django.contrib import messages
from .forms import UserRegisterForm

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST) # create a
form that has post data
        # validating form
        if form.is_valid():
            form.save() # This will automatically hased
the password and save the data to database.
            #grabing username
            username = form.cleaned_data.get('username')
#the validated form data will be in form.cleaned_data
dictionary
            #using flash message to show that valid data
is recieved.
            messages.success(request, f'Your account has
been created! You are now able to log in.')
            # redirecting user to different form. For this
we need to import redirect
            return redirect('login')

    else:
        form = UserRegisterForm() #creating instance of
this class
    return render(request, 'users/register.html',
{'form':form})

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

Create profile.html

- djnago_project
  - ◇ users
    - ▪ templates
      - users
        - → profile.html

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

Now create route to this profile view.

- django_project
  - ◇ django_project
    - ▪ urls.py

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

urlpatterns = [
    path('admin/',admin.site.urls),
    path('register/', user_views.register, name =
'register'),
    path('profile/', user_views.profile, name = 'profile'),
    path('login/', auth_views.LoginView.as_view
(template_name = 'users/login.html'), name = 'login'),
    path('logout/', auth_views.LogoutView.as_view
(template_name = 'users/login.html'), name = 'logout'),
    path('blog/', include('blog.urls')),
]
```

•

Add this profile link to the navigation bar if user is logged in

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html
        - → about.html
        - → base.html

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">

        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="https://
maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
        <link rel = "stylesheet" type = "text/css" href =
"{% static 'blog/main.css' %}">


        {% if title %}
            <title> Django Blog - {{ title }}</title>
        {% else %}
            <title> Django Blog </title>
        {% endif %}
    </head>
    <body>
        <header class="site-header">
          <nav class="navbar navbar-expand-md navbar-dark
bg-steel fixed-top">
            <div class="container">
              <a class="navbar-brand mr-4" href="{% url
'blog-home' %}">Django Blog</a>
              <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarToggle" aria-
controls="navbarToggle" aria-expanded="false" aria-
label="Toggle navigation">
              <span class="navbar-toggler-icon"></span>
              </button>
```

```
              <div class="collapse navbar-collapse"
id="navbarToggle">
                <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>
                <!-- Navbar Right Side -->
                <div class="navbar-nav">
                {% 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>
                {% 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>
              </div>
            </div>
          </nav>
        </header>

        <main role="main" class="container">
          <div class="row">
            <div class="col-md-8">
                {% if messages %}
                    {% for message in messages %}
                        <div class = "alert alert-
{{ message.tags}}">
                            {{ message }}
                        </div>
                    {% endfor %}
                {% endif %}
              {% block content %}{% endblock %}
```

```
            </div>
            <div class="col-md-4">
              <div class="content-section">
                <h3>Our Sidebar</h3>
                <p class='text-muted'>You can put any
information here you'd like.
                  <ul class="list-group">
                    <li class="list-group-item list-group-
item-light">Latest Posts</li>
                    <li class="list-group-item list-group-
item-light">Announcements</li>
                    <li class="list-group-item list-group-
item-light">Calendars</li>
                    <li class="list-group-item list-group-
item-light">etc</li>
                  </ul>
                </p>
              </div>
            </div>
          </div>
        </main>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS --
>
    <script src="https://code.jquery.com/
jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/
GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.12.9/umd/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/
bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5
+76PVCmYl" crossorigin="anonymous"></script>
    </body>
<\html>
```

But here, if we go on profile page using url, then it will not show any user.
Here we need to check if the user is logged in or not to access the profile page
To do this we use login required decorator that django provides

- django_project
  - ◇ blog
  - ◇ users
    - ▪ views.py

here, import log in required decorator

```
from django.shorcuts import renders, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .forms import UserRegisterForm

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST) # create a
form that has post data
        # validating form
        if form.is_valid():
            form.save() # This will automatically hased
the password and save the data to database.
            #grabing username
            username = form.cleaned_data.get('username')
#the validated form data will be in form.cleaned_data
dictionary
            #using flash message to show that valid data
is recieved.
            messages.success(request, f'Your account has
been created! You are now able to log in.')
            # redirecting user to different form. For this
we need to import redirect
            return redirect('login')

    else:
        form = UserRegisterForm() #creating instance of
this class
    return render(request, 'users/register.html',
{'form':form})

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

Now, if we load the profile page using url , it will give an error and is looking for page accounts/login
We need to tell django where to redirect
We need to redirect to login page
This can be done by adding login url variable in settings.py file

- djanog_project
  - django_project
    - settings.py

At the bottom, add

LOGIN_URL = ‘login’

```
import os

# Build paths inside the project like this: os.path.join
(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath
(__file__)))


# Quick-start development settings - unsuitable for
production
# See https://docs.djangoproject.com/en/3.0/howto/
deployment/checklist/

# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = 'ezmx*&72_-a42@i##kc(up7vzp6qqha1=d)+=^@@w3-
_=5w!r)'

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

ALLOWED_HOSTS = []


# Application definition

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

MIDDLEWARE = [
```

```
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'web_learning.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',

'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'web_learning.wsgi.application'


# Database
```

```
# https://docs.djangoproject.com/en/3.0/ref/settings/
#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-
password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimil
arityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValid
ator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordVali
dator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordVal
idator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'

CRISPY_TEMPLATE_PACK = 'bootstrap4'

LOGIN_REDIRECT_URL = 'blog-home'
LOGIN_URL = 'login'
```

User Profile and Picture

Upload user profile

Create user profile model

This profile model has one to one relation with user model as each user can have only one profile.

- django_project
  - users
    - models.py

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

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

    def __str__(self):
        return f'{self.user.username} Profile'
```

Migrate this model

python manage.py makemigrations

When we run this, we get an error Can not user Imagefield becuase Pillow is not installed

Pillow is the library for working with images in python
So install this Pillow library

pip install Pillow

Run makemigrations command again

python manage.py makemigrations

python manage.py migrate

To view this user profile in admin page, we need to register this profile model in admin.py file on app.

- django_project
  - users
    - admin.py

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

admin.site.register(Profile)
```

Now you can add profile from admin page.

For accessing this Profile model run python shell

python manage.py shell

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

user = User.objects.filter(username = 'Dimple').first()
user
output: <User:Dimple>

# As we have one to one relationship with profile, we can now access profile using this user variable

user.profile
output: <Profile: Dimple Profile>

user.profile.image

user.profile.image.width

user.profile.image.url
```

We don't access images direcly, we access using it's location.

This images are actually saved in

- djanog_project
  - profile_pics

This is not a proper way to store images. If multiple models saves different kinds of images, then the root directory will be meshed up.

To have all the image in proper location, make some changes in settings.py file

- djanog_project
  - ◇ django_project
    - ▪ settings.py

At the bottom, add

MEDIA_ROOT = os.path.join(BASE_DIR, 'media')

This is saying root to the base directory which is django_project and then directory call media

MEDIA_URL = '/media/'

media root is the full path to a directory where we like django to store uploaded file by default, this files are stored in the file system and not in the database .
So this media root is the directory is the directory where uploaded file will be saved.

```
import os

# Build paths inside the project like this: os.path.join
(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath
(__file__)))


# Quick-start development settings - unsuitable for
production
# See https://docs.djangoproject.com/en/3.0/howto/
deployment/checklist/

# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = 'ezmx*&72_-a42@i##kc(up7vzp6qqha1=d)+=^@@w3-
_=5w!r)'

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

ALLOWED_HOSTS = []


# Application definition

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

MIDDLEWARE = [
```

```
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'web_learning.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',

'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'web_learning.wsgi.application'


# Database
```

```
# https://docs.djangoproject.com/en/3.0/ref/settings/
#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-
password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimil
arityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValid
ator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordVali
dator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordVal
idator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'


MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media.'

CRISPY_TEMPLATE_PACK = 'bootstrap4'

LOGIN_REDIRECT_URL = 'blog-home'
LOGIN_URL = 'login'
```

Now when we upload an image, it will create profile_pic directory inside of this media directory and put the image in there.
and media directory is located in the base directory.

So, now if we upload the image, it will be located in

- django_project
  - media
    - profile_pics

Updating user profile page

In profile page, we have only username.
But there shold be profile pic, email address, update form

- djnago_project
  - users

- templates
  - users
    - → profile.html

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags    %}
{% block content %}
    <div class="content-section">
        <div class="media">
            <img class="rounded-circle account-img"
src="{{ user.profile.image.url }}">
            <div class="media-body">
              <h2 class="account-heading">
{{ user.username }}</h2>
              <p class="text-secondary"> {{ user.email }}</
p>
        </div>
    </div>
        <!-- FORM HERE -->
    </div>
{% endblock content %}
```

In order to get {{ user.profile.image.url }} to work, we need to add those media route to our our pattern.

add,
from django.conf import settings
from django.conf.urls.static import static

- django_project
  - ◇ django_project
    - urls.py

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

urlpatterns = [
    path('admin/',admin.site.urls),
    path('register/', user_views.register, name =
'register'),
    path('login/', auth_views.LoginView.as_view
(template_name = 'users/login.html'), name = 'login'),
    path('logout/', auth_views.LogoutView.as_view
(template_name = 'users/login.html'), name = 'logout'),
    path('blog/', include('blog.urls')),
]

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

Add default image in media folder.

Now, we need to make sure that when an user is created, they automatically get profile aswell which would include the default profile picture.

This can be done by adding django signal.

Create new file in user app called signals.

- django_project
  - users
    - signals.py

from django.db.models.signals import post_save

This is the signal that is fired when an object is saved.
In this case, we need a post_save signal when a user is created. So we also need to import the built in user model

from django.contrib.auth.models import User

we also need to create a receiver
Receiver is a function that gets this signal and performs some task.
so we need to import receiver also.

from django.dispatch import receiver

finally we need to import profile from the model since we will be creating profile in a function

receiver is a decorator

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


#when the user is created, it will send this post_save
signal
# This signal is received by this receiver and the
receiver is the create_profile function
#instance is instance of the user and if user is created
then create a profile where user = instance
@receiver(post_save, sender = User)
def create_profile( sender, instance, created, **kargs):
    if created:
        Profile.object.create(user = instance)

#create a save profile when user is saved
@receiver(post_save, sender = User)
def create_profile( sender, instance, **kargs):
    instance.profile.save()
```

Now, we have to import the signal inside of the ready function of users app.py file

- django_project
  - users
    - app.py

create ready method and imort those signals.

```
from django.apps import AppConfig

class UsersConfig(AppConfig):
    name = 'users'

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

Update user Profile

Updating user profile form

• django_project

- ◇ users
  - ▪ forms.py

Here, we will create a model form and it allows us to create form that will work with a specific database model
here, we want a form that update our user model

we are working with a profile model, we need to import this profile models

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

class UserRegisterForm(UserCreationForm): # inheriting
UserCreationForm
    email = forms.EmailField()

    class Meta:
        #specifing the model that we need to interact with
        model = User # model is user because whenever the
form is validated, it going to create a new user
        fields = ['username', 'email', 'password1',
'password2']
        # This is the field that are going to be shwon on
our form

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

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

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

- django_project
  - ◇ blog
  - ◇ users
    - ▪ views.py

import UserUpdateForm and ProfileUpdateForm

```
from django.shorcuts import renders, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .forms import UserRegisterForm, UserUpdateForm,
ProfileUpdateForm

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST) # create a
form that has post data
        # validating form
        if form.is_valid():
            form.save() # This will automatically hased
the password and save the data to database.
            #grabing username
            username = form.cleaned_data.get('username')
#the validated form data will be in form.cleaned_data
dictionary
            #using flash message to show that valid data
is recieved.
            messages.success(request, f'Your account has
been created! You are now able to log in.')
            # redirecting user to different form. For this
we need to import redirect
            return redirect('login')

    else:
        form = UserRegisterForm() #creating instance of
this class
    return render(request, 'users/register.html',
{'form':form})

@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)
```

- djnago_project
  - ◇ users
    - ▪ templates
      - users
        - → profile.html

In profile template, we have two forms.
but we will put this into single html form.

also add special encoding type to form
this is require in oreder to our form to pass image data for profile picture

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags    %}
{% block content %}
    <div class="content-section">
        <div class="media">
            <img class="rounded-circle account-img"
src="{{ user.profile.image.url }}">
            <div class="media-body">
              <h2 class="account-heading">
{{ user.username }}</h2>
              <p class="text-secondary"> {{ user.email }}</
p>
        </div>
    </div>
        <form method = "POST" enctype = "mutlipart/form-
data">
                {% csrf_token %}
                    <fieldset class = "form-group">
                        <legend class = "border-bottom
mb-4">
                            Profile Details
                        </legend>
                        {{ u_form|crispy }}
                        {{ p_form|crispy }}
                    </fieldset>
                    <div class = "form-group">
                        <button class = "btn btn-outline-
info" type = "submit">
                            Update
                        </button>
                    </div>
            </form>
    </div>
{% endblock content %}
```

Now, while running this, the update form will not display current information
This are models form that are expecting working on a specific model object.

- django_project
  - blog
  - users
    - views.py

```
from django.shorcuts import renders, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .forms import UserRegisterForm, UserUpdateForm,
ProfileUpdateForm

def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST) # create a
form that has post data
        # validating form
        if form.is_valid():
            form.save() # This will automatically hased
the password and save the data to database.
            #grabing username
            username = form.cleaned_data.get('username')
#the validated form data will be in form.cleaned_data
dictionary
            #using flash message to show that valid data
is recieved.
            messages.success(request, f'Your account has
been created! You are now able to log in.')
            # redirecting user to different form. For this
we need to import redirect
            return redirect('login')

    else:
        form = UserRegisterForm() #creating instance of
this class
    return render(request, 'users/register.html',
{'form':form})

@login_required
def profile(request):
    if request.method == 'POST':
        u_form = UserUpdateForm(requrest.POST, instance =
request.user)
        p_form = ProfileUpdateForm(requrest.POST,
request.FIlES, instance = request.user.profile)
```

```
        #request.FILES is for the image that user will
upload

        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!')
            # redirecting user to different form. For this
we need to import redirect
            return redirect('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/profile.html', context)
```

Resizing large Image

For resizing the image, we need to override the save method of profile model.

- django_project
  - users
    - models.py

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

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

    def __str__(self):
        return f'{self.user.username} Profile'

    def save(self):
        super().save()
        img = Image.open(self.image.path)
        if img.width > 300 or img.height > 300:
            output_size = (300,300)
            img.thumbnail(output_size)
            img.save(sel.image.path)
```

Display image beside each block

- djanog_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <article class="media content-section">
                <img class = "rounder-circle article-img"
src = "{{ post.author.profile.image.url }}">
            <div class="media-body">
                <div class="article-metadata">
                    <a class="mr-2" href="#">
{{ post.author }}</a>
                    <small class="text-muted">
{{ post.date_posted|data: "F d, Y" }}</small>
                </div>
                <h2><a class="article-title" href="#">
{{ post.title }}</a></h2>
                <p class="article-content">
{{ post.content }}</p>
            </div>
        </article>
    {% endfor %}
{% endblock content %}
```

Create Update and Delete Post

Create Class based view for home page
There are different kinds of class based views
- list views
- detail views
- create views
- update views
- delete views

These are the generic views which django provides

open
- django_project
  - ◇ blog
    - ▪ views.py

import
from django.views.generic import ListView

This views are class based views, hence we will create a class PostListView

within a ListView, we need to create a variable called model and this will tell our list view what model to query in order to create the list.
In this case, we want it to be our Post

```
from django.shortcuts import render
from django.views.generic import ListView
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post


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

In order to use this list view open blog urls.py modules

- django_project

- ◇ blog
  - ▪ urls.py

Import PostListView

from .views import PostListView

instead of using home view, replace it with PostListView

since we are using class based views, it can not be directly passed like this
It has to be converted into an actual view.
Method is availble to do this called as_view()

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

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

]
```

When we run this, it may not work as by default, class based view looks for templates of certain naming pattern
it is looking for blog/post_list.html
in general,
<app>/<model>_<viewtype>.html

So, it is looking for this template.
But we can also change which template we want to use.
We can do that in views.py file

- django_project
  - ◇ blog
    - ▪ views.py

```
from django.shortcuts import render
from django.views.generic import ListView
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'   #<app>/
<model>_<viewtype>.html


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

This is not going to be work because it doesn't know what we want the variable to be named in our templated that we are going to be looping over.
By default, the list view is going to call that variable object list instead of post
Create a variable in list view to know the looping is to be done in post

- django_project
  - ◇ blog
    - ▪ views.py

add
context_object_name = 'posts' posts is a variable we already passed in home view

```
from django.shortcuts import render
from django.views.generic import ListView
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'  #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'


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

Run it
Now, ordering of the post is not proper
The latest post should be at the top
For this, we will change the order is making to the database.
Add ordering attribute with the field we want to order on

- django_project
  - blog
    - views.py

Create new attribute

Ordering = ['-date_posted']

put minus sign for reverse

```
from django.shortcuts import render
from django.views.generic import ListView
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'  #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']


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

Create a view for individual post.
This is going to be detail view.
To create detail view, import Detailview

open
- django_project
  - blog
    - views.py

import
from django.views.generic import ListView,DetailView

```
from django.shortcuts import render
from django.views.generic import ListView, DetailView
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'   #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']

class PostDetailView(DetailView):
    model = Post


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

Create URL pattern for this

- django_project
  - ◇ blog
    - ▪ urls.py

from .views import PostDetailView

Create route
For this, we need to create URL pattern that contains a variable
Create a route where id of the post is part of the route, this can be done using

‘post/<int:pk>/’

```
from django.urls import path
from .views import PostListView, PostDetailView
from . import views

urlpatterns = [
    # path('', views.Home, name = 'blog-home'),
    path('', PostListView.as_view(), name = 'blog-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name
= 'post-detail'),
    path('about/', views.about, name='blog-about'),

]
```

Now we need to create a template that will display details of post.
By default, it will look for
blog/post_detail.html

so, lets create a template with that name so that we don't need to specify the template name

- django_project
  - ◇ blog
    - ▪ templates
      - blog
        - → post_detail.html

```
{% extends "blog/base.html" %}
{% block content %}
        <article class="media content-section">
        <img class="rounded-circle article-img" src =
"{{ object.author.profile.image.url }}">
          <div class="media-body">
            <div class="article-metadata">
              <a class="mr-2" href="#">{{ object.author }}
</a>
              <small class="text-muted">
{{ object.date_posted|date:"F d, Y" }}</small>
            </div>
            <h2 class="article-title">{{ object.title }}</
h2>
            <p class="article-content">{{ object.content }}
</p>
          </div>
        </article>
{% endblock content %}
```

Lets add link to these routes for the individual posts on our home page

- django_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <article class="media content-section">
        <img class="rounded-circle article-img" src =
"{{ post.author.profile.image.url }}">
          <div class="media-body">
            <div class="article-metadata">
              <a class="mr-2" href="#">{{ post.author }}</
a>
              <small class="text-muted">
{{ post.date_posted|date:"F d, Y" }}</small>
            </div>
            <h2><a class="article-title" href="{% url
'post-detail' post.id %}">{{ post.title }}</a></h2>
            <p class="article-content">{{ post.content }}</
p>
          </div>
        </article>
    {% endfor %}
{% endblock content %}
```

Create, Update and Delete View

CreateView

- django_project
  - ◇ blog
    - ▪ views.py

import CreateView

```
from django.shortcuts import render
from django.views.generic import (
            ListView,
            DetailView,
            CreateView
            )                       #Parenthesis to break the
line. We can have it in signle line also
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'  #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']

class PostDetailView(DetailView):
    model = Post


class PostCreateView(CreateView):
    model = Post
    fields = ['title', 'content']   #fields that we want


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

Update the URL with this new created view

• django_project

◇ blog
▪ urls.py

```
from django.urls import path
from .views import PostListView, PostDetailView,
PostCreateView
from . import views

urlpatterns = [
    # path('', views.Home, name = 'blog-home'),
    path('', PostListView.as_view(), name = 'blog-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name
= 'post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-
create'),
    path('about/', views.about, name='blog-about'),

]
```

We need a template for this view. but it will not be post_create view.
This will share a template with the update view that we will create.
So the name for the template will be post_form.html
Create this view

- django_project
  ◇ blog
    ▪ templates
      - blog
        → post_form.html

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags %}
{% block content %}
    <div class = "content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">
                    Blog Post
                </legend>
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info"
type="submit">
                    Post
                </button>
            </div>
        </form>
    </div>
{% endblock content %}
```

If we create a post, then it will give an error - Null author
we need author for the post which is current logged in user
We can do this by overriding form valid method

- django_project
  - blog
    - views.py

```
from django.shortcuts import render
from django.views.generic import (
            ListView,
            DetailView,
            CreateView
            )                       #Parenthesis to break the
line. We can have it in signle line also
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'  #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']

class PostDetailView(DetailView):
    model = Post


class PostCreateView(CreateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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


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

Now we will get error for redirect page as we don't have redirect page

We will redirect to the detail page

To find the URL of a model object is to create a get absolute URL method in our model that returns the path to any specific instance

- djnago_project
  - blog
    - models.py

Open this and within our post model, we need to create that get absolute URL method so that Django knows how to find the location to a specific post
First we are going to get the URL of particular route and
For this, we need to use reverse function
reverse will return the full URL to that route as a string
and here, we simply need to return an URL as a string and let the view handles the redirect for us

first, import reverse function

from django.urls import reverse

and then create the absolute URL method to any specific instance of a post

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

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    date_posted = models.DateTimeField(default=
timezone.now)
    author = models.ForeignKey(User,
on_delete=models.CASCADE)

    def __str__(self):
        return self.title

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

The post should not be created without login

if we try to access this route and if the user is not logged in then we will redirect it to login page
Here, we can't use decorator on class based view
Here, we will use login mixin and its a class that we inherit from that will add login functionality to the view

- django_project
  - blog
    - views.py

```
from django.shortcuts import render
from django.contrib.auth.mixins import  LoginRequiredMixin
from django.views.generic import (
            ListView,
            DetailView,
            CreateView
            )                    #Parenthesis to break the
line. We can have it in signle line also
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'  #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']

class PostDetailView(DetailView):
    model = Post


class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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


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

Create Update View

- django_project
  - blog
    - views.py

```
from django.shortcuts import render
from django.contrib.auth.mixins import  LoginRequiredMixin
from django.views.generic import (
            ListView,
            DetailView,
            CreateView,
            UpdateView
            )                       #Parenthesis to break the
line. We can have it in signle line also
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'  #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']

class PostDetailView(DetailView):
    model = Post


class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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

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

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

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

- djanog_project
  - ◇ blog
    - ▪ urls.py

```
from django.urls import path
from .views import PostListView, PostDetailView,
PostCreateView, PostUpdateView
from . import views

urlpatterns = [
    # path('', views.Home, name = 'blog-home'),
    path('', PostListView.as_view(), name = 'blog-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name
= 'post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-
create'),
    path('post/<int:pk>/update', PostUpdateView.as_view(),
name = 'post-update'),
    path('about/', views.about, name='blog-about'),

]
```

For template, we will be using same post_form template

Now this will work and update the form

Here, we are not checking that the author of the post is the person trying to access this update page

For this, we will be using mixin and first we need to import it

import UserPassesTestMixin

- django_project
  - ◇ blog
    - ▪ views.py

We can create a method called testfunc and that is the function that our user passes test mixin will run in order to see if our user passes a certain test condition

```
from django.shortcuts import render
from django.contrib.auth.mixins import
LoginRequiredMixin, UserPassesTestMixin
from django.views.generic import (
            ListView,
            DetailView,
            CreateView,
            UpdateView
            )                       #Parenthesis to break the
line. We can have it in signle line also
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'   #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']

class PostDetailView(DetailView):
    model = Post


class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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

class PostUpdateView
(UserPassesTestMixin ,LoginRequiredMixin, UpdateView):
```

```
    model = Post
    fields = ['title', 'content']   #fields that we want

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

    def test_func(self):
        #get exact post that we're currently updating this
can be acheived using method of updated view call get
object
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False

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

Create Delete View

- django_project
  - blog
    - views.py

```
from django.shortcuts import render
from django.contrib.auth.mixins import
LoginRequiredMixin, UserPassesTestMixin
from django.views.generic import (
            ListView,
            DetailView,
            CreateView,
            UpdateView,
            DeleteView
            )                       #Parenthesis to break the
line. We can have it in signle line also
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'   #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']

class PostDetailView(DetailView):
    model = Post


class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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

```
class PostUpdateView
(UserPassesTestMixin ,LoginRequiredMixin, UpdateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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

    def test_func(self):
        #get exact post that we're currently updating this
can be acheived using method of updated view call get
object
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False

#Delete view also required to be author of the post and
should be logged in
class PostDeleteView
(UserPassesTestMixin ,LoginRequiredMixin, DeleteView):
    model = Post

    def test_func(self):
        # get exact post that we're currently updating
this can be acheived using method of updated view call get
object
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False

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

- djanog_project

◇ blog
▪ urls.py

```
from django.urls import path
from .views import PostListView, PostDetailView,
PostCreateView, PostUpdateView, PostDeleteView
from . import views

urlpatterns = [
    # path('', views.Home, name = 'blog-home'),
    path('', PostListView.as_view(), name = 'blog-home'),
    path('post/<int:pk>/', PostDetailView.as_view(), name
= 'post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-
create'),
    path('post/<int:pk>/update', PostUpdateView.as_view(),
name = 'post-update'),
    path('post/<int:pk>/delete', PostDeleteView.as_view(),
name = 'post-delete'),
    path('about/', views.about, name='blog-about'),

]
```

Now, we need a template
And the template this expects is just a form that asks if we sure that we want to delete the post and if we submit the form then the post will be deleted
the template will be

post_confirm_delete.html

- django_project
  ◇ blog
    ▪ templates
      - blog
        → post_confirm_delete.html

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

If we delete the blog, then it doesn't know where to redirect

- django_project
  - ◇ blog
    - ▪ views.py

In success view, we need to add a success URL attribute
and we will set it to home page

Add all the link

- django_project
  - ◇ blog
    - ▪ templates
      - blog
        - → base.html

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

    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://
maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/
bootstrap.min.css" integrity="sha384-Gn5384xqQ1aoWXA
+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

    <link rel="stylesheet" type="text/css" href="{% static
'blog/main.css' %}">

    {% if title %}
        <title>Django Blog - {{ title }}</title>
    {% else %}
        <title>Django Blog</title>
    {% endif %}
</head>
<body>
    <header class="site-header">
      <nav class="navbar navbar-expand-md navbar-dark bg-
steel fixed-top">
        <div class="container">
          <a class="navbar-brand mr-4" href="{% url 'blog-
home' %}">Django Blog</a>
          <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarToggle" aria-
controls="navbarToggle" aria-expanded="false" aria-
label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
          </button>
```

```
          <div class="collapse navbar-collapse"
id="navbarToggle">
            <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>
            <!-- Navbar Right Side -->
            <div class="navbar-nav">
                {% if user.is_authenticated %}
                  <a class="nav-item nav-link" href="{%
url 'post-create' %}"> New Post</a>
                  <a class="nav-item nav-link" href="{%
url 'profile' %}">Profile</a>
                  <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>
          </div>
        </div>
      </nav>
    </header>
    <main role="main" class="container">
      <div class="row">
        <div class="col-md-8">
            {% if messages %}
                {% for message in messages %}
                    <div class = "alert alert-
{{ message.tags }}">
                        {{ message }}
                    </div>
                {% endfor %}
            {% endif %}
```

```
          {% block content %}{% endblock %}
        </div>
        <div class="col-md-4">
          <div class="content-section">
            <h3>Our Sidebar</h3>
            <p class='text-muted'>You can put any
information here you'd like.
              <ul class="list-group">
                <li class="list-group-item list-group-item-
light">Latest Posts</li>
                <li class="list-group-item list-group-item-
light">Announcements</li>
                <li class="list-group-item list-group-item-
light">Calendars</li>
                <li class="list-group-item list-group-item-
light">etc</li>
              </ul>
            </p>
          </div>
        </div>
      </div>
    </main>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS --
>
    <script src="https://code.jquery.com/
jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/
GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.12.9/umd/popper.min.js" integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/
ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script src="https://maxcdn.bootstrapcdn.com/
bootstrap/4.0.0/js/bootstrap.min.js" integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5
+76PVCmYl" crossorigin="anonymous"></script>
</body>
```

```
</html>
```

Update and Delete link will be in post detail template

- django_project
  - ◇ blog
    - ▪ templates
      - blog
        - → post_detail.html

```
{% extends "blog/base.html" %}
{% block content %}
        <article class="media content-section">
        <img class="rounded-circle article-img" src =
"{{ object.author.profile.image.url }}">
          <div class="media-body">
            <div class="article-metadata">
              <a class="mr-2" href="#">{{ object.author }}
</a>
              <small class="text-muted">
{{ object.date_posted|date:"F d, Y" }}</small>
                {% 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 %}
            </div>
            <h2 class="article-title">{{ object.title }}</
h2>
            <p class="article-content">{{ object.content }}
</p>
          </div>
        </article>
{% endblock content %}
```

Pagination

Create More Posts

Create posts.json file in django_projcet

- django_project
  - ◇ posts.json

add some post

In terminal,
run

python manage.py

```
import json

from blog.models import Post

with open('posts.json') as f:
...     posts_json = json.load(f)


 for post in posts_json:
...     post = Post(title = post['title'], content = post['content'], author_id = post['user_id'])
...     post.save()


exit()
```

Loading this many blogs will take time so we will load only few post and add link at the bottom
For doing this we will work with pagination object

python manage.py shell

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

Using this concept

- django_project
  - ◇ blog
    - ▪ views.py

add paginate_by = 2

```
from django.shortcuts import render
from django.contrib.auth.mixins import
LoginRequiredMixin, UserPassesTestMixin
from django.views.generic import (
            ListView,
            DetailView,
            CreateView,
            UpdateView,
            DeleteView
            )                       #Parenthesis to break the
line. We can have it in signle line also
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'   #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']
    paginate_by = 2

class PostDetailView(DetailView):
    model = Post


class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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

```
class PostUpdateView
(UserPassesTestMixin ,LoginRequiredMixin, UpdateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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

    def test_func(self):
        #get exact post that we're currently updating this
can be acheived using method of updated view call get
object
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False

#Delete view also required to be author of the post and
should be logged in
class PostDeleteView
(UserPassesTestMixin ,LoginRequiredMixin, DeleteView):
    model = Post
    success_url = '/'

    def test_func(self):
        # get exact post that we're currently updating
this can be acheived using method of updated view call get
object
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False

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

Now when we run this we see only 2 posts but we can't see link for the other post
We can manually type it in url like

/?page=12

- djnago_project
  - ◇ blog
    - ▪ templates
      - blog
        - → home.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <article class="media content-section">
        <img class="rounded-circle article-img" src =
"{{ post.author.profile.image.url }}">
          <div class="media-body">
            <div class="article-metadata">
              <a class="mr-2" href="#">{{ post.author }}</
a>
              <small class="text-muted">
{{ post.date_posted|date:"F d, Y" }}</small>
            </div>
            <h2><a class="article-title" href="{% url
'post-detail' post.id %}">{{ post.title }}</a></h2>
            <p class="article-content">{{ post.content }}</
p>
          </div>
        </article>
    {% endfor %}
    {% 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 %}
{% endblock content %}
```

Change number of posts per pages.

Now when we click on user name, only the post from that user should be displayed
And if the user has bunch of code, then it should also be paginated

Create new view called UserPostListView in views.py file

- django_project
  - blog
    - views.py

```
from django.shortcuts import render, get_object_or_404
from django.contrib.auth.mixins import
LoginRequiredMixin, UserPassesTestMixin
from django.contrib.auth.models import User
from django.views.generic import (
            ListView,
            DetailView,
            CreateView,
            UpdateView,
            DeleteView
            )                       #Parenthesis to break the
line. We can have it in signle line also
from .models import Post

def Home(request):
    context = {
        'posts': Post.objects.all()
    }
    return render(request, 'blog/home.html',context)

class PostListView(ListView):
    model = Post
    template_name =  'blog/home.html'  #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    ordering = ['-date_posted']
    paginate_by = 5

class UserPostListView(ListView):
    model = Post
    template_name =  'blog/user_posts.html'  #<app>/
<model>_<viewtype>.html
    context_object_name = 'posts'
    paginate_by = 5

    def get_queryset(self):
        user = get_object_or_404(User, username =
self.kwargs.get('username'))
```

```
        return Post.objects.filter(author=user).order_by('-
date_posted')



class PostDetailView(DetailView):
    model = Post


class PostCreateView(LoginRequiredMixin, CreateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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

class PostUpdateView
(UserPassesTestMixin ,LoginRequiredMixin, UpdateView):
    model = Post
    fields = ['title', 'content']   #fields that we want

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

    def test_func(self):
        #get exact post that we're currently updating this
can be acheived using method of updated view call get
object
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False

#Delete view also required to be author of the post and
should be logged in
class PostDeleteView
(UserPassesTestMixin ,LoginRequiredMixin, DeleteView):
```

```
    model = Post
    success_url = '/'

    def test_func(self):
        # get exact post that we're currently updating
this can be acheived using method of updated view call get
object
        post = self.get_object()
        if self.request.user == post.author:
            return True
        return False

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

Create URL Pattern

- django_project
  - blog
    - urls.py

```
from django.urls import path
from .views import PostListView, PostDetailView,
PostCreateView, PostUpdateView, PostDeleteView,
UserPostListView
from . import views

urlpatterns = [
    # path('', views.Home, name = 'blog-home'),
    path('', PostListView.as_view(), name = 'blog-home'),
    path('user/<str:username>', UserPostListView.as_view(),
name = 'user-posts'),
    path('post/<int:pk>/', PostDetailView.as_view(), name
= 'post-detail'),
    path('post/new/', PostCreateView.as_view(), name='post-
create'),
    path('post/<int:pk>/update', PostUpdateView.as_view(),
name = 'post-update'),
    path('post/<int:pk>/delete', PostDeleteView.as_view(),
name = 'post-delete'),
    path('about/', views.about, name='blog-about'),

]
```

Create template user_posts.html

- django_project
  - ◇ blog
    - ▪ templates
      - - blog
        - → user_posts.html

```
{% extends "blog/base.html" %}
{% block content %}
    <h1 class="mb-3">Posts by {{ view.kwargs.username }}
({{ page_obj.paginator.count }})</h1>
    {% for post in posts %}
        <article class="media content-section">
        <img class="rounded-circle article-img" src =
"{{ post.author.profile.image.url }}">
          <div class="media-body">
            <div class="article-metadata">
              <a class="mr-2" href="{% url 'user-posts'
post.author.username %}">{{ post.author }}</a>
              <small class="text-muted">
{{ post.date_posted|date:"F d, Y" }}</small>
            </div>
            <h2><a class="article-title" href="{% url
'post-detail' post.id %}">{{ post.title }}</a></h2>
            <p class="article-content">{{ post.content }}</
p>
          </div>
        </article>
    {% endfor %}
    {% 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 %}
{% endblock content %}
```

Also add the link to home and post detail template

- django_project
  - ◇ blog
    - ▪ templates
      - - blog
        - → home.html

```
{% extends "blog/base.html" %}
{% block content %}
    {% for post in posts %}
        <article class="media content-section">
        <img class="rounded-circle article-img" src =
"{{ post.author.profile.image.url }}">
          <div class="media-body">
            <div class="article-metadata">
              <a class="mr-2" href="{% url 'user-posts'
post.author.username %}">{{ post.author }}</a>
              <small class="text-muted">
{{ post.date_posted|date:"F d, Y" }}</small>
            </div>
            <h2><a class="article-title" href="{% url
'post-detail' post.id %}">{{ post.title }}</a></h2>
            <p class="article-content">{{ post.content }}</
p>
          </div>
        </article>
    {% endfor %}
    {% 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 %}
{% endblock content %}
```

- django_project
  - ◇ blog
    - ▪ templates
      - blog
        - → post_detail.html

```
{% extends "blog/base.html" %}
{% block content %}
        <article class="media content-section">
        <img class="rounded-circle article-img" src =
"{{ object.author.profile.image.url }}">
          <div class="media-body">
            <div class="article-metadata">
              <a class="mr-2" href="{% url 'user-posts'
object.author.username %}">{{ object.author }}</a>
              <small class="text-muted">
{{ object.date_posted|date:"F d, Y" }}</small>
                {% 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 %}
            </div>
            <h2 class="article-title">{{ object.title }}</
h2>
            <p class="article-content">{{ object.content }}
</p>
          </div>
        </article>
{% endblock content %}
```

Email and Password Reset

- django_project
  - ◇ django_project
    - ▪ urls.py

Password reset views are built in to the auth view that we already imported
Create URL pattern for password reset

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

This will set password reset instruction to email

```
"""web_learning URL Configuration

The `urlpatterns` list routes URLs to views. For more
information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home,
name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(),
name='home')
Including another URLconf
    1. Import the include() function: from django.urls
import include, path
    2. Add a URL to urlpatterns:  path('blog/', include
('blog.urls'))
"""
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from users import views as  user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name =
'register'),
    path('profile/', user_views.profile, name = 'profile'),
    path('login/', auth_views.LoginView.as_view
(template_name = 'users/login.html'), name = 'login'),
    path('logout/', auth_views.LogoutView.as_view
(template_name = 'users/logout.html'), name = 'logout'),
    path('password-reset/',
auth_views.PasswordResetView.as_view(template_name =
'users/password_reset.html'), name = 'password_reset'),
```

```
    path('',include('blog.urls')),
]

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

Create password_reset template

- django_project
  - ◇ users
    - ▪ templates
      - - users
        - → password_reset.html

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags %}
{% block content %}
    <div class = "content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">
                    Reset Password
                </legend>
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info"
type="submit">
                    Request Password Reset
                </button>
            </div>
        </form>
    </div>
{% endblock content %}
```

Now, we need to create a form after this is created successfully

This will be the route that confirms that the email has been sent and tells you to check your inbox

- django_project
  - django_project
    - urls.py

```
"""web_learning URL Configuration

The `urlpatterns` list routes URLs to views. For more
information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home,
name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(),
name='home')
Including another URLconf
    1. Import the include() function: from django.urls
import include, path
    2. Add a URL to urlpatterns:  path('blog/', include
('blog.urls'))
"""
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from users import views as  user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name =
'register'),
    path('profile/', user_views.profile, name = 'profile'),
    path('login/', auth_views.LoginView.as_view
(template_name = 'users/login.html'), name = 'login'),
    path('logout/', auth_views.LogoutView.as_view
(template_name = 'users/logout.html'), name = 'logout'),
    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('',include('blog.urls')),
]

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

Now, we need to create password_reset_done.html template

- django_project
  - ◇ users
    - ▪ templates
      - users
        - → password_reset_done.html

```
{% extends "blog/base.html" %}
{% block content %}
    <div class="alert alert-info">
        An email has been sent with instruction to reset
your password
    </div>

{% endblock content %}
```

Now, when we run and enter the email, it will give an error

It is looking for password_reset_confirm

It is trying to create password_reset_confirm route with some values. and the name of the template that through this error is password_reset_email.html
This are the template that django using in a background to create the email to send to the user so that they can reset their password
It not only try to access password_reset_confirm file but also try to pass the parameter like, UIDB64 and token
These two parameter we need to access in our URL so that we know that the user who requested the password reset is the person trying to access that page
UIDB64 is the users id encoded in base64 and the tocken is the tocken to check that the password is valid and these are required since the view is expecting them.

Add this routes

- django_project
  - django_project
    - urls.py

```
"""web_learning URL Configuration

The `urlpatterns` list routes URLs to views. For more
information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home,
name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(),
name='home')
Including another URLconf
    1. Import the include() function: from django.urls
import include, path
    2. Add a URL to urlpatterns:  path('blog/', include
('blog.urls'))
"""
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from users import views as  user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name =
'register'),
    path('profile/', user_views.profile, name = 'profile'),
    path('login/', auth_views.LoginView.as_view
(template_name = 'users/login.html'), name = 'login'),
    path('logout/', auth_views.LogoutView.as_view
(template_name = 'users/logout.html'), name = 'logout'),
    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.PasswordResetConfirmView.as_view(template_name
= 'users/password_reset_confirm.html'), name =
'password_reset_confirm'),
    path('',include('blog.urls')),
]

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

Create password_reset_confirm template

- django_project
  - ◇ users
    - ▪ templates
      - users
        - → password_reset_confirm.html

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags %}
{% block content %}
    <div class = "content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">
                    Reset Password
                </legend>
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info"
type="submit">
                    Reset Password
                </button>
            </div>
        </form>
    </div>
{% endblock content %}
```

We are still getting an error
The error says connection refused
We don't have an email server to send an email
We will do this in Gmail
Search
Google App Password

this will give an information about how to sign in to google account through different applications
If you don't have two factor authentication, then you can just tell Google to allow sign in from less secure app.
If you have two factor authentication, then you can create password specifically for the application that you want to sign in from

- django_project
  - django_project
    - settings.py

```
"""
Django settings for web_learning project.

Generated by 'django-admin startproject' using Django
3.0.5.

For more information on this file, see
https://docs.djangoproject.com/en/3.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.0/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join
(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath
(__file__)))


# Quick-start development settings - unsuitable for
production
# See https://docs.djangoproject.com/en/3.0/howto/
deployment/checklist/

# SECURITY WARNING: keep the secret key used in production
secret!
SECRET_KEY = 'ezmx*&72_-a42@i##kc(up7vzp6qqha1=d)+=^@@w3-
_=5w!r)'

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

ALLOWED_HOSTS = []


# Application definition
```

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

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'web_learning.urls'

TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',

'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',
```

```
'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'web_learning.wsgi.application'


# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/
#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}


# Password validation
# https://docs.djangoproject.com/en/3.0/ref/settings/#auth-
password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimil
arityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValid
ator',
    },
    {
```

```
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordVali
dator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordVal
idator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
MEDIA_URL = '/media/'

CRISPY_TEMPLATE_PACK = 'bootstrap4'

LOGIN_REDIRECT_URL = 'blog-home'
LOGIN_URL = 'login'

#all less secure app in gmail search for google app
passwords
```

```
EMAIL_BACKEND =
'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST = 'localhost'
EMAIL_HOST_USER = os.environ.get('EMAIL_USER') or
"yourgmail"
EMAIL_HOST_PASSWORD = os.environ.get('EMAIL_PASSWORD') or
"yourpassword"
```

This will run
when we reset the password and submit it
it will give an error of noreversematch

This can be solved using route password_reset_complete and then we can login with the new password

- django_project
  - django_project
    - urls.py

```
"""web_learning URL Configuration

The `urlpatterns` list routes URLs to views. For more
information please see:
    https://docs.djangoproject.com/en/3.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home,
name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(),
name='home')
Including another URLconf
    1. Import the include() function: from django.urls
import include, path
    2. Add a URL to urlpatterns:  path('blog/', include
('blog.urls'))
"""
from django.contrib import admin
from django.contrib.auth import views as auth_views
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from users import views as  user_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('register/', user_views.register, name =
'register'),
    path('profile/', user_views.profile, name = 'profile'),
    path('login/', auth_views.LoginView.as_view
(template_name = 'users/login.html'), name = 'login'),
    path('logout/', auth_views.LogoutView.as_view
(template_name = 'users/logout.html'), name = 'logout'),
    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.PasswordResetConfirmView.as_view(template_name
= 'users/password_reset_confirm.html'), name =
'password_reset_confirm'),
    path('password-reset-complete/',
auth_views.PasswordResetCompleteView.as_view(template_name
= 'users/password_reset_complete.html'), name =
'password_reset_complete'),
    path('',include('blog.urls')),
]

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

Create template

- django_project
  - ◇ users
    - ▪ templates
      - - users
        - → password_reset_complete.html

```
{% extends "blog/base.html" %}
{% block content %}
    <div class="alert alert-info">
        Your password has been set.
    </div>
    <a href="{% url 'login' %}">Sign In Here</a>
{% endblock content %}
```

Add forgot password in login.html file

- django_project
  - ◇ users

- templates
  - users
    - → login.html

```
{% extends "blog/base.html" %}
{% load crispy_forms_tags %}
{% block content %}
    <div class = "content-section">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">
                    Log In
                </legend>
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info"
type="submit">
                    Log In
                </button>
                <small class="text-muted ml-2">
                    <a href="{% url 'password_reset'
%}">Forgot Password?</a>
                </small>
            </div>
        </form>
        <div class="border-top pt-3">
            <small class="text-muted">
                Need an Account? <a class="ml-2" href="{%
url 'register' %}">Sign Up Now</a>
            </small>

        </div>

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

Deploying application

Deploying to linux server

• Deploying to linux server means deploying to a virtual machine that is hosted by a company like digitalocean, AWS, linode
• Open users models.py file and make some correction before deploying your application

• django_project
  ◇ users
    ▪ models.py

add keyword and postional argument in save method of Profile class

```
from django.db import models
from django.contrib.auth.models import User
from PIL import Image

# 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')

    def __str__(self):
        return f'{self.user.username} Profile'

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)

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

Now, deploying this application

Depoying in linode (Linux Server)

Paid