Blog:

**Setup workspace**

1. Create virtual environment

Go to the path in desktop you want to create Django project and open cmd prompt.

python -m venv env

In order to activate execute the below script:

env\Scripts\activate

1. the Install Django using below script

python -m pip install Django

1. Inorder to identify the version run command

django-admin –version

1. Start a project

django-admin startproject myapp

1. In the package we have the server with which we can test the input and output

First navigate to the myapp

>cd myapp

>python manage.py runserver

Now we came to know , how to run Django project…………

**Project structure**

6)To open the code in visual studio . Go to the path myapp. From that open command and type

>code .

Learn about files:

Asgi, wsgi – used during deployement

Init – For mention python package

Settings – database configuration

**Django app**

1. In visual studio terminal run the command to start the server

python .\manage.py runserver

1. To create new Django app open other terminal and execute the code

python .\manage.py startapp blog

migrations – related to database file

app.s – configuration

models – important model

1. In order to register the app , go to myapp in that settings add your app name under installed\_apps set.

**Creating first view**

1. To create a page index page
2. Go to view.py under blogs and write the commands
3. Create a new file urls in blogs which is used to have all urls related to blog
4. We need to add this url in a management file. Go to myapp in that url add this

path("" , include("blog.urls")) – For root path we are adding blog app url path

can be tested and changed with : path("\blog" , include("blog.urls"))

**More urls and views**

1. Add new method in views nd path in url

**Dynamic url segments**

Change in url module

 path("post/<int:post\_id>",views.detail, name = 'detail')

path("post/<str:post\_id>",views.detail, name = 'detail')

Output:

<http://127.0.0.1:8000/blog/post/100>

**Redirects**

from django.shortcuts import render,redirect

def old\_url\_redirect(request):

    return redirect("new\_url")

def new\_url\_view(request):

    return HttpResponse("This is the new URL")

**Reverse and Named URLS**

We are straight away hardcoding the urls in view functions. But we can use the name of the url to do so. So we are using Reverse api.

Urls:

app\_name = 'blog'

 path("new\_url",views.new\_url\_view, name = "new\_page\_url"),

view:

def old\_url\_redirect(request):

    return redirect(reverse("blog:new\_page\_url"))

**Returning html (templates)**

from django.shortcuts import render,redirect

create a folder templates, under which blog – index.html.

In order to open the template in root need to make the changes in views

def index(request):

    return render(request,'blog/index.html')

**Static files ( css and images linking) # To view page source**

Static template tag

{% load static %}

In index.html

href="{% static 'blog/style.css' %}">

But still it will generate a url but won’t link the style.css. in order to do. Go to settings file in myapp.

In that under static\_url variable create one more. To know the Django project all static files are under this directory

STATICFILES\_DIRS = [

    "/blog/static/blog/"

]

**Creating Layouts: (Like header/ footer in all slides)**

Under templates - > blog create a base.html

Copy all the index.html content. Remove the dynamic content from it.

Add this lines:

<!-- Dynamic content--> - comment

    {% block content %} - indicating block and name of the block is content

    {% endblock %} - indication end of dynamic content

Remove the duplicates from index.html and add

Order to add index.html in base html

{% extends 'base.html' %}

In order to access block in base

{% block content %}

Dynamic contents

{% endblock %}

**Include partial templates**

The amount of code in base.html is too large. So in that case we will be using partial template.

We are going to separate the header and footer put it under templates/blogs/includes folder

Header.html

Footer.html

In base:

 <!-- Header-->

     {% include 'blog/includes/header.html'%}

# It is already added

    <!-- Dynamic content-->

    {% block content %}

    {% endblock %}

    <!--Footer-->

    {% include 'blog/includes/footer.html'%}

**Variable Interpolation**:

A particular variable in the page need to be dynamic then go to views.py.

In that lets say in index page the Latest post need to be changed.

Add 3rd variable in the argument in dict format

def index(request):

    blog\_title = "Latest Post"

    return render(request,'blog/index.html',{'blog\_title: blog\_title'})

     The syntax for variable interpolation is go to index.html latest post over there add it

{{blog\_title}}

That dynamic variable can be used in extent position like base.html as well

**Filters**

In templates if you want to change the variable value according to you.

Syntax:

    <h2 >{{blog\_title | upper}}</h2> ----- Change it to upper case

   <h2 >{{blog\_title | length}}</h2> ----- Show the number of characters in a variable

    <h2 >{{blog\_title | truncatewords:1}}</h2> - Show only the first word. It is used in text elipices

So for details.html we are not passing the dynamic variable in views (It is happening for index , base bcs it was extended)

So we have filter like it is not present put this default value

 <title>{{blog\_title | default:'Document'}}</title>

**Django Vs Code extension**

Download extension , hightlight the tags syntax stuffs

**For Tag**

So if we see the index page we have 6 post which is hard code. So we are making it dynamic like for 2 posts the information about it should be there.

From views we are going to make it dynamic

In views, def index add this list and send it like variable interpolation

 posts = [

        {'title':'Post 1', 'content': 'Content of post 1'},

        {'title':'Post 2', 'content': 'Content of post 2'},

        {'title':'Post 3', 'content': 'Content of post 3'},

        {'title':'Post 4', 'content': 'Content of post 4'},

    ]

return render(request,'blog/index.html',{'blog\_title': blog\_title, 'posts':posts})

From index.html remove the duplicate divs. We are going to have only one div

{% for post in posts %}

 <h5 class="card-title">{{post.title}}</h5>

                                <p class="card-text">{{post.content}}</p>

  {% endfor %}

**If Tag**

If tag to add conditions if a list is empty what needs to be printed alternatively

         {% if posts %}

  {% else %}

                <p>No Posts Available</p>

            {% endif %}

**URL Tag and Dynamic urls:**

Suppose for each and every post link I want to detail page and the url should end with that id.

Add extra id attribute in index method

posts = [

        {'id':1, 'title':'Post 1', 'content': 'Content of post 1'},

        {'id':2,'title':'Post 2', 'content': 'Content of post 2'},

        {'id':3,'title':'Post 3', 'content': 'Content of post 3'},

        {'id':4,'title':'Post 4', 'content': 'Content of post 4'},

    ]

In index.html

Instead of this  <a href="./detail.html">Read More</a>

Add this  <a href="/blog/post/{{post.id}}">Read More</a>

We should not hardcoded the url inorder to do so we are going to use the url name. Since the change is in template we are not using reverse.

Using url:

   <a href="{% url 'blog:detail' post\_id=post.id %}">Read More</a>

**Custom 404:**

If we put a wrong url we will get 404 error . In order to customise we are doing this.

To create a handler ,Create new file views.py under myapp. In order for the entire project

Create a folder templates under like myapp folder in templates add a 404.html file

In that <h1> 404 error found</h1>.

In views:

from django.shortcuts import render

def custom\_page\_not\_found(request, exception):

    return render(request, '404.html',status=404)

In url of myapp configuration

handler404 = 'myapp.views.custom\_page\_not\_found' (handler404 indicates 404 variable)

Go settings and do it as:

DEBUG = False

ALLOWED\_HOSTS = ['localhost']

Also need to include template folder in settings

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': ['templates'],

        'APP\_DIRS': True,

        'OPTIONS': {

**Using logger for debugging:**

def detail(request, post\_id):

    post = next((item for item in posts if item['id'] == post\_id), None)

    return render(request,'blog/detail.html', {'post':post})

In details .template added it but still the title is not changing so we are going to debug

    <h1 class="mb-4">{{post.title}}</h1>

In views:

import logging

logger=logging.getLogger("Testing")

logger.debug(f'Post variable is {post}')

In order to have the logger statement in shell we need to make some changes in settings.

Copy from documentations

Then check in terminal

Connecting Mysql database:

In order to work with server instead of commands we are going to use workbench

You can see the local host server running. Using the Gui we are going create database for the server

CREATE DATABASE blog;

In Django we need to have a package to connect mysql database

(env) C:\Users\vijayashree\OneDrive\Desktop\Blog (Django project)>pip install mysqlclient

(In virtual environment)

Need to change the database configure in myapp settings

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'blog',

        'USER': 'root',

        'PASSWORD': 'Vijixyz20011',

        'HOST'    : 'localhost',

        'PORT'    : '3306'

    }

}

**Creating Models:**

Suppose we want to insert delete or keep the type of columns of table. In Django we will have it which is called as model. Using model we don’t need to right each and everytime sql queries

Go to blog 🡪 models.py

from django.db import models

# Create your models here.

class Post(models.Model):

    title   = models.CharField(max\_length=100)

    content = models.TextField()

    img\_url = models.URLField(null= True)

    def \_\_str\_\_(self):

        return self.title

**Creating Migrations:**

To create table for the model. Instead of directly going to sql and enter data and create table. We will be doing in this way

Create migration file under the migration folder using this command

python manage.py makemigrations

This command to create table

python manage.py migrate

**Inserting Data & Custom command:**

In order to insert demo data we are creating a new file populate\_data.py in blog folder.

In that copy the titles and contents

Go to lorem picsum, and take it <https://picsum.photos/id/1/200/300>.

All data is for sample contents.

To create demo data we need a library

>pip install faker

It will generate a fake data.

from blog.models import Post

for title, content, img\_url in zip(titles, content, img\_url):

    Post.objects.create(title=title, content=content, img\_url=img\_url)

Running the file will initialize. But we need to run it as a command:

from django.core.management.base import BaseCommand

Check that file

These module needs to be present in separate folder create a new folder in blogs management. Under that copy int and create one more folder command. In that again copy init

In init of command folder add this

from .populate\_posts import Command as PopulatePostsCommand , for the Django to knows

If you run the command now all the datas will be inserted in table

python manage.py populate\_posts

**Getting Post data:**

In views of blog we are actually sending the static post data. Where we are going to make the changes

from .models import Post

post=Post.objects.all() – Return all post data in list format

In index.html add this to add images:

     <img style = "height:250; width:300"src="{{post.img\_url}}" class="img-fluid" alt="...">

Now the index page alone is ready

**Getting post data by Id**

In details method in view

#Getting data from model by id

    post=Post.objects.get(id=post\_id)

in detail.html template add this to show img in each detail page

    <img src="{{post.img\_url}}" class="img-fluid mb-4" alt="Blog Image">

Now let see how to add the dated time

   <p class="text-muted">{{post.created\_at}}</p>

<p class="text-muted">{{post.created\_at | date:"F j, Y"}}</p>

F- day j-month y-year

Truncatechars:5 – Is also there

**Handling Error**

from django.http import HttpResponse, Http404

 try:

    #Getting data from model by id

        post=Post.objects.get(pk=post\_id)

        return render(request,'blog/detail.html', {'post':post})

    except Post.DoesNotExist:

        raise Http404("Post not found")

**Generate slug for posts:**

Instead of using id which is primary key in table in order to get details is not safe.

We are leaking data to hackers. So for this we are going to use slugs urls which will be discussed.

First we need to delete the existing data in populate\_data

  Post.objects.all().delete()

Use slug to generate

Go to models, in that add

from django.utils.text import slugify

slug       = models.SlugField(unique=True, null=True, max\_length=191)

    def save(self, \*args, \*\*kwargs):

        self.slug = slugify(self.title)

        super().save(\*args, \*\*kwargs)

python manage.py makemigrations

While making migration set default value as

python manage.py migrate

Then run the command : python manage.py populate\_posts

ALTER TABLE blog.blog\_post AUTO\_INCREMENT = 1; - in sql to have ids from 1

**Get Post by slugs:**

In urls change : path("post/<str:slug>",views.detail, name = 'detail'),

In views, details:

def detail(request, slug):

    try:

    #Getting data from model by id

        post=Post.objects.get(slug=slug)

In index:     <a href="{% url 'blog:detail' slug=post.slug%}">Read More</a>

**Creating category Model for models:**

Go to models module and add

class  Category(models.Model):

    name = models.CharField( max\_length=100)

    def \_\_str\_\_(self):

        return self.name

create new command file populate\_category

from blog.models import Category

from django.core.management.base import BaseCommand

class Command(BaseCommand):

    help= "This commands inserts category data"

    def handle(self, \*args, \*\*options):

        # Deleting existing data

        Category.objects.all().delete()

        categories = ['Sports','Technology','Science','Art','Food']

        for category in categories:

            Category.objects.create(name=category)

        self.stdout.write(self.style.SUCCESS("Completed inserting data"))

**Many to One Relationship**

Many posts may belongs to one category we are going to do with help of foreign keys

Add this line in models module under post class to create new column

category   = models.ForeignKey(Category, on\_delete=models.CASCADE)

on\_delete means if any category was deleted the corresponding post will also will be removed.

Then make migrations. You can see the new column will be added.

Go to populate\_posts module in order to add categories into that make the following changes. To take random category

from blog.models import Category

import random

categories = Category.objects.all()

loop:

  category = random.choice(categories)

 Post.objects.create(title=title, content=content, img\_url=img\_url, category=category)

Run the command

**Delete and recreating tables:**

Del of migration In vs code is not only enough. Go to sql workbench. Over there in Django\_migrations delete the migrations and apply. Also drop the tables blog post or category tables, then we can recreate by running migrate and running commands.

Python manage.py flush – to delete all the data but table will be present

**Show category in template**

Go to index.html and change

href="#">{{post.category}}</a>

**Showing related post based on category**

In details method we are passing post make this change.

 related\_posts = Post.objects.filter(category=post.category).exclude(pk=post.id)

return render(request,'blog/detail.html', {'post':post, 'related\_posts': related\_posts})

Go to details template:

   {% for rel\_post in related\_posts%}

                    <li><a href="{%url 'blog:detail' slug=rel\_pos.slug%}">{{rel\_post.title}}</a></li>

                    {% endfor %}

<h1 class="mb-4">{{post.title}}</h1><small style="font-size:18px"> {{post.category}}</small> - to show current category very small

Giving link to blog

 <a class="text-light text-decoration-none" href"/blog"><h3>Blog</h3></a>

**Pagination:**

To show only we post in index page.

pip install django-pagination

In index method we are getting all the post there we need to use pagination

from django.core.paginator import Paginator

    all\_posts = Post.objects.all()

    #Pagination

    paginator  = Paginator(all\_posts, 5)

    page\_number = request.GET.get('page')

    page\_obj = paginator.get\_page(page\_number)

    return render(request,'blog/index.html',{'blog\_title': blog\_title, 'page\_obj':page\_obj})

-----

Change in index.html

 {% if page\_obj %}

                {% for post in page\_obj %}

Pagination bootstrap code add index.html after end for loop. To have pagination only when post are available. We are going to make it dynamic

 {% if page\_obj.has\_other\_pages%} – to show pagination only when has other pages

   {% if page\_obj.has\_previous%} – It has any previous page

{{page\_obj.previous\_page\_number} – It gives the previous page number

page\_obj.number}} of {{page\_obj.paginator.num\_pages – current page number and total number of pages

{% if page\_obj.has\_next%} --- Whether it has next page

page\_obj.next\_page\_number – Give the next page number

**Working with contact form:**

Copy the code from contact.html from github. Have only the dynamic content

Then in views and url make the changes

 path("contact", views.contact\_view,name="contact")

def contact\_view(request):

    return render(request,'blog/contact.html')

In order to handle forms in django we have concept

Create a new forms.py under blog project

from django import forms

class ContactForm(forms.Form):

    name = forms.CharField(label='Name' , max\_length=100)

    email = forms.Emailfield(label='Email')

    message = forms.CharField(label='Message')

When you try to submit the form it will throw a CSRF verification error which we need to send that cookie when we try to submit the form.

We need to enable in the folder

Add this:

  {% csrf\_token%} in contact.html

When we pass this token the request we are submitting is secured Django will consider

In contact\_view method

def contact\_view(request):

    if request.method == 'POST':

        form = ContactForm(request.POST)

        if form.is\_valid():

            logger=logging.getLogger("Testing")

            logger.debug(f'POST Data is {form.cleaned\_data['name']} {form.cleaned\_data['message']}')

    return render(request,'blog/contact.html')

**Show validation error and success message:**

In the else part of contact method in views gives

 else:

            logger.debug("Form validation failure")

to check the validation. But since we have removed the required thing in boostrap code. Submiting without values will throw error. In forms.py we are giving the field as required.

    name = forms.CharField(label='Name' , max\_length=100, required=True)

    email = forms.EmailField(label='Email', required=True)

    message = forms.CharField(label='Message', required=True)

It will be shown in console inorder to view it in the web page the error.

In contact\_view method last line under if condition add

 return render(request,'blog/contact.html', {'form':form})

In contact.html:

 <h3>Contact Form</h3>

                {%for field in form%}

                {% if field.errors%}

                    <div class="col-12">

                    <span class="fw-bold">{{field.label\_tag}}</span>

                    {% for error in field.errors %}

                        <span class-"text-danger">{{error}}</span>

                    {%endfor%}

                    </div>

                {%endif%}

                {%endfor%}

Okay so when we submit all the values in the field got refreshed we want it to be saved. In order to do go to contact\_view method

ef contact\_view(request):

    if request.method == 'POST':

        form = ContactForm(request.POST)

        name = request.POST.get("name")

        email = request.POST.get("email")

        message = request.POST.get("message")

        logger=logging.getLogger("Testing")

        if form.is\_valid():

            logger.debug(f"POST Data is {form.cleaned\_data['name']} {form.cleaned\_data['message']}")

        else:

            logger.debug("Form validation failure")

        return render(request,'blog/contact.html', {'form':form, 'name':name, 'email':email, 'message':message})

    return render(request,'blog/contact.html')

In contact.html , for each input tag give the value = {{name}} and so on for other fields.

For showing success message In webpage first make changes in contact\_view method accordingly

success\_message = "Your email has been sent"

 return render(request,'blog/contact.html', {'form':form, 'success\_message': success\_message})

Then contact.html add:

 <!--Success message-->

                {% if success\_message %}

                <div class="alert alert-success" role="alert">

                    {{ success\_message }}

                </div>

                {%endif%}

**Admin setup:**

Admin panel in Django. We are going to use that

python manage.py createsuperuser – Create a user for admin panel , he is a admin..

Username (leave blank to use 'vijayashree'): root

Email address: [root@gmail.com](mailto:root@gmail.com)

Name: root

Password: Vijixyz20011

Password (again):

Explore the url: http://127.0.0.1:8000/admin

**Register models to admin:**

Post model

Category model

In order to register the models go to admin.py in blog folder. In that do

from .models import Post, Category

# Register your models here.

admin.site.register(Post)

admin.site.register(Category)

Now you can see all the post and cateogiories in admin path in browser.

In that you can and edit the post and category.

**Customize admin Interface:**

from django.contrib import admin

from .models import Post, Category

# Register your models here.

class PostAdmin(admin.ModelAdmin):

    list\_display = ('title','content')

    search\_fields = ('title', 'content')

    list\_filter = ('created\_at','category')

admin.site.register(Post,PostAdmin)

admin.site.register(Category)

**Dynamic About us:**

Create a new about.html from copying from github removing the header and footer.

In urls.py add the link and create new methods in views.py method.

path("about", views.about\_view,name="about")

def about\_view(request):

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

In includes add

  <a class="text-light text-decoration-none" href="{% url 'blog:contact' %}">Contact</a>

                </div>

For a particular page lets say about us how to give a dynamic content using admin.

Create new model about us. Migrate the model

Connect about us model to Django admin inorder to edit

from .models import Post, Category, AboutUs

admin.site.register(AboutUs)

Run server

Now if you add a content in admin dashboard. It will be automatically added in database. So now we edited the code to take data from first row of the table

def about\_view(request):

    about\_content = AboutUs.objects.first().content

    return render(request,'blog/about.html',{'about\_content':about\_content})

In aboutUs .html, remove the content and added variable interpolation

{{about\_content}} ,,now any changes in content in admin dashboard will be reflected in page

**Came to end have deployed the project in Github**

**Thanksss!!!**

Simple answer:

First, I initialize Git in my Django project folder using git init, then add my project files with git add . and commit them with git commit -m "message". Next, I create a repository on GitHub, copy its remote URL, and connect it with git remote add origin <URL>. Finally, I push the code with git push -u origin main (or master). I also make sure to add venv/ and \_\_pycache\_\_/ to .gitignore so unnecessary files aren’t uploaded.

If they want a one-line summary:

*"I initialize Git, commit my Django code, link the GitHub repo as remote, and push it, making sure to ignore virtual environment and cache files."*

**Reference:**

$ git init

Initialized empty Git repository in C:/Users/vijayashree/OneDrive/Desktop/Blog (Django project)/myapp/.git/

vijayashree@LAPTOP-0A20GU5V MINGW64 ~/OneDrive/Desktop/Blog (Django project)/myapp (master)

$ git add .

vijayashree@LAPTOP-0A20GU5V MINGW64 ~/OneDrive/Desktop/Blog (Django project)/myapp (master)

$ git add .

vijayashree@LAPTOP-0A20GU5V MINGW64 ~/OneDrive/Desktop/Blog (Django project)/myapp (master)

$ git commit -m "First Upload"

Author identity unknown

\*\*\* Please tell me who you are.

Run

git config --global user.email "you@example.com"

git config --global user.name "Your Name"

to set your account's default identity.

Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'vijayashree@LAPTOP-0A20GU5V.(none)')

vijayashree@LAPTOP-0A20GU5V MINGW64 ~/OneDrive/Desktop/Blog (Django project)/myapp (master)

$ git remote add origin https://github.com/vijayashree809/django\_project.git

vijayashree@LAPTOP-0A20GU5V MINGW64 ~/OneDrive/Desktop/Blog (Django project)/myapp (master)

$ git remote -v

origin https://github.com/vijayashree809/django\_project.git (fetch)

origin https://github.com/vijayashree809/django\_project.git (push)

vijayashree@LAPTOP-0A20GU5V MINGW64 ~/OneDrive/Desktop/Blog (Django project)/myapp (master)

$ git push -u origin master

**Runserver live demo:**

PythonAnywhere is a **cloud hosting service** designed specifically for Python applications — including Django.

Think of it as:

* A **remote computer** where your Django project runs 24/7.
* It takes care of the server setup, database hosting, and domain configuration, so you don’t have to manually install or configure Linux servers.
* It gives you a free subdomain like yourusername.pythonanywhere.com, where anyone can view your live app.

Got it — I’ll walk you through **step-by-step** to deploy your Django project so you get a live demo link you can put on your resume.  
We’ll use **PythonAnywhere** because it’s beginner-friendly, free for small apps, and works well with Django.

**Step 1 – Prepare Your Project for Deployment**

Before uploading:

1. **Check requirements.txt is up-to-date**
2. pip freeze > requirements.txt
3. **Check settings.py**
   * In your Django project’s settings.py, find:
   * DEBUG = True

Change to:

DEBUG = False

* + Add allowed hosts:
  + ALLOWED\_HOSTS = ['yourusername.pythonanywhere.com']

1. **Run collectstatic locally**
2. STATIC\_ROOT = BASE\_DIR/'staticfiles'
3. python manage.py collectstatic

(This prepares your CSS/JS/images for production.)

**Step 2 – Create a PythonAnywhere Account**

1. Go to [https://www.pythonanywhere.com](https://www.pythonanywhere.com/)
2. Sign up for the **free plan**.
3. Log in.

**Step 3 – Upload Your Project**

You have two ways:

* **ZIP + Upload** (simplest for small projects)
* **Git Clone from GitHub** (best for keeping code in sync)

I suggest **Git clone** since your project is already on GitHub.

**On PythonAnywhere:**

1. Go to the **Dashboard** → **Bash Console**.
2. In the Bash console:
3. git clone https://github.com/vijayashree809/django\_project.git
4. cd django\_project

**Step 4 – Create a Virtual Environment**

Inside the Bash console:

python3.10 -m venv venv

source venv/bin/activate

pip install -r requirements.txt

**Step 5 – Set Up the Web App**

1. Go to **Dashboard** → **Web** → **Add a new web app**.
2. Choose:
   * **Manual configuration**
   * **Python 3.10** (or version you used locally)
3. Set **Source code** path to your project directory (e.g., /home/username/django\_project)
4. Set **WSGI file** path to point to your Django wsgi.py:
   * Usually: /home/username/django\_project/myapp/wsgi.py

**Step 6 – Configure WSGI**

1. Open the WSGI file in PythonAnywhere editor.
2. Add:
3. import sys
4. import os
5. path = '/home/username/django\_project'
6. if path not in sys.path:
7. sys.path.append(path)
8. os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'myapp.settings')
9. from django.core.wsgi import get\_wsgi\_application
10. application = get\_wsgi\_application()

**Step 7 – Migrate Database & Collect Static**

In Bash console:

source venv/bin/activate

python manage.py migrate

python manage.py collectstatic

**Step 8 – Reload the Web App**

* Go to **Web tab** → click **Reload**.

**Step 9 – Test Your Live Link**

* Your app will now be live at:
* https://yourusername.pythonanywhere.com

✅ **End Result:**

* You’ll have:
  + **GitHub Link** → for your code.
  + **Live Link** → for a working demo.

If you want, I can **customize these steps exactly for your myapp + blog project structure** so you won’t have to guess file paths. That will make the deployment go without errors.

Do you want me to tailor the deployment steps to your exact folder structure?