Django Framework

By

Amar Panchal

9821601163

FRAMEWORK SETUP

1. RUN POWERSHELL IN ADMIN MODE
2. SAY: Set-ExecutionPolicy Unristructed-------say yes
3. Create dir for Django
4. In directory create virtual environment
    1. pip install virtualenv
5. Activate virtual environment
    1. virtualenv .
6. Activate scripts
    1. ./Scripts/activate
7. pip install django

Starting Project

1. Create folder for app
2. To create project
    1. django-admin.exe startproject <name>
3. Locate manage.py
4. Run server of Django
    1. python manage.py runserver
5. Check browser
    1. 127.0.0.0:8000 or localhost:8000
6. Can also handle migration error if any by
    1. Python manage.py migrate
7. Can see admin panel by
    1. Localhost:8000/admin/login

Creating super user

- Python manage.py createsuperuser
- Follow instruction on screen and remember username and password

Add module to your App

- Go to directory with manage.py
  - Python manage.py startapp <name>

Common Files Seen

- __init__.py
- admin.py
- apps.py
- models.py
- test.py
- views.py
- urls.py
- settings.py

To register app with base app

- Open settings.py from main app
  - To the list of Installed_app=[] add ‘<name of app>’,

Use urls.py

- Open and edit to make changes
  - See how route works in urlpatterns=[]
- For new sub app
  - Create urls.py in it local folder
  - Copy content of urls.py of base app
  - Remove admin statements

Add routes of sub app to main app urls.py

- Add
  1. From Django.urls import path,incude
  2. From <newapp> import views
  3. Add path(‘’,include(‘<subapp name>.urls’)),

Working with views.py

- In urls of subapp
  - from . imports views
- In views.py

  def home(request):

  return render(request,'home.html',{})

- Create templates folder in subapp
  - Create new file →home.html
  - Code home .html
  - Save in templates folder only
  - In urls.py add
    - path('',views.home,name='home')

Working with templates

- Create a base file that is needed on every page
- Django creates base file and then extends it on every page

- Steps
  - Create base .html---code it
  - Add code blocks

        {% block <name> %}
        {% endblock %}
  - At the end and save

On other pages

- Add extends block

  {% extends 'base.html' %}
- Add block

  {% block <name> %}

  Page code

  {% endblock %}

For page title handling

- Create block title in title of base and then use it on every page
- <title>
- {% block title %}
-   name the title
- {% endblock %}

Django links(dynamic)

- One can call pages by Django’s url name given
- use

  <% url ‘name of page’ %>

Passing Paramenters

- See views .py which has a dictionary
- {key:value}
- One can define them and then call it directly or via data base
- Make changes in render of views.py

```
def home(request):
    name="amar"
    return render(request,"home.html",{'name':name})
```

Database handling

- Edit models.py in <new app>

- Create class that inherits (models.Model)
- Create all variables needed in the data base
  - Also code def __str__(self):

    Return self.<data>
- Use datatypes of Django
- From powershell
  - Python manage.py makemigrations
  - Python manage.py migrate

Data base in admin page

- Edit admin.py
- Add lines
- From .models import <classname>
- Admin.site.register(<classname>)

Adding database to page

- Edit views.py
- Add
    - From .models import <nameofdatabase>
    - Var=<database>.objects.all(if specific)

  At home page

  {‘key’:var}

  Use

  {%...%}

  For operations

Database creation and handling

- Steps
- 1 create class in models.py
- 2 create migration
- 3 push migration in database

In models.py

- class <classname>(models.Model):
- item=models.CharField(max_length=200)
- complete=models.BooleanField(default=False)

- def __str__(self):
- return self.item #what to return

migration

- Class→ DDL →Database (automatically)
- Python manage.py makemigrations
- Python manage.py migrate

Register database in Admin section

- Use admin.py
- from .models import <class of models.py>
- admin.site.register(<class of models.py>)

To add database to page

- In views.py
- From .models import <name of class>
- To read all data
  - variable=<class>.objects.all
- At home():
  - Add {'key':var}
- On home.html add
  - {% for data in variable %}
    - {{data.items}}

Adding forms for input

- In forms.py(to be created)

```
from django import forms
from .models import Appdatabase

class AppdatabaseForm(forms.ModelForm):
  class Meta:
        model=Appdatabase
        fields=["item","complete"]
```

- In views.py add
  - from .forms import AppdatabaseForm

On base.html

- <form class="form-inline" method="POST">
-         {%csrf_token%}
-     <input class="form-control mr-sm-2" type="search" placeholder="Data to add" aria-label="" name="item">
-     <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Add to list</button>
-   </form>

Views.py

- We need to add
  - from .forms import AppdatabaseForm

```
if request.method=="POST":
        form=AppdatabaseForm(request.POST or None)
        if form.is_valid():
            form.save()
            data=Appdatabase.objects.all
            return render (request, "home2.html" ,{'data':data})

    else:
        data=Appdatabase.objects.all
        return render (request, "home2.html" ,{'data':data})
```

Adding a prompt to a page

- Add
  - from django.contrib import messages
  - messages.success(request,("------>data added"))
- On home.html

```
{% if messages %}
    {% for message in messages %}
    <div class="alert alert-warning" role="alert">
        {{message}}
    </div>
    {% endfor %}
{% endif %}
```

Deleting from a form

- Add in urls.py
  - path("delete/<Appdatabase_id>",views.delete,name="delete"),
- In views.py

```
def delete(request,Appdatabase_id):
    item = Appdatabase.objects.get(pk=Appdatabase_id)
    item.delete()
    messages.success(request, ('Item Has Been Deleted!'))
    return redirect('home2')
```

On top of views.py

- from django.http import HttpResponseRedirect
- from django.shortcuts import render,redirect

- In home.html
- <td><a href="{% url 'delete' d.id%}"> Delete</a></td>

Adding CSS

- In main app create folder “static”
  - In that create folder css.,image,js
- In settings.py
- STATICFILES_DIRS=[os.path.join(BASE-DIR,’static’),]
- On top of pages
  - {%load static %}