DJANGO

SR NO	TOPICS
1	INTRODUCTION
2	HTTP Response, function-based view, url register(Project-1)
3	Include (Each app makes individual urls.py) (project-2)
4	Rendering templates and use of static files (project-3)
5	Dynamic Template Language (project-4)
6	Template inheritance (project-5)
7	url tag(project-6)
8	Model and Dynamic URL (project-7)
9	Csrf (project-8)
10	Signup,login,logout(project-9)
11	Demo-1(project-10) findmusic
12	Demo-2(project-11) all in one

Model View Template (MVT)

The MVT is a design pattern that separates an application into three main logical components Model, View, and Template.

Each of these components has its own role in a Project.

Model: The model is the central component of MVT. It represents the data in the application and is responsible for storing and retrieving the data. The model is typically implemented as a class in Python.

View: The view is responsible for displaying the data to the user. It typically does this by rendering a template. The view is typically implemented as a function in Python.

Template: The template is a file containing the application's static HTML code. It also contains variables that are used to dynamically insert data from the model. The template is typically written in Jinja2, a templating language that is designed for Python

Create Django Project

- python -m django startproject projectname
  Example
- python -m django startproject project1

How to runserver

Django provides built-in server which we can use to run our project.

runserver – This command is used to run built-in server of Django.

Steps:-

Go to Project Folder

Then run the command

python manage.py runserver

Server Started

Visit *http://127.0.0.1:8000* or *http://localhost:8000*

You can specify the Port number ***python manage.py run server 5555***

Visit *http://127.0.0.1:5555* or *http://localhost:5555*

How to Stop Server

ctrl + c is used to stop Server

Note – Sometimes when you make changes in your project you may need to restart the server.

How to Start/Create Application

A Django project contains one or more applications which means we create application inside Project Folder.

Syntax:- python manage.py startapp appname

Creating One Application Inside Project:-

Go to Project Folder

Run Command python manage.py startapp course

Creating Multiple Applications Inside Project:-

Go to Project Folder

python manage.py startapp course

python manage.py startapp fees

python manage.py startapp result

How to Install Application in Our Project

As we know a Django project can contain multiple applications so just creating an application inside a project is not enough we also have to install an application in our project.

We install the application in our project using the settings.py file.

The settings.py file is available at the Project Level which means we can install all the applications of the project.

This is compulsory otherwise Application won't be recognized by Django.

Open settings.py file

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'application_name1',
```

'application_name2',

]

Save settings.py File

Project-1

A function-based view, is a Python function that takes a Web request and returns a Web response.

This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image or anything.

Each view function takes an HttpRequest object as its first parameter.

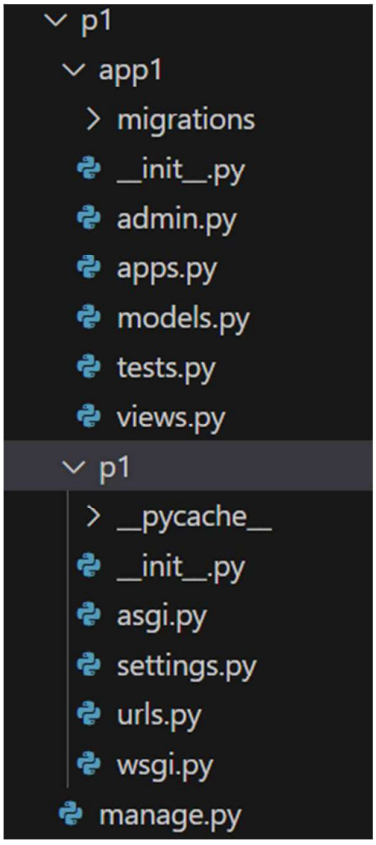
The view returns an HttpResponse object that contains the generated response. Each view function is responsible for returning an HttpResponse object.

We will call these functions as *view function* or *view function of application or view*.

Step 1: python -m django startproject p1 (Terminal)

Step2: cd p1 (Terminal)

Step3: python manage.py startapp app1



Step4: install an app in setting.py (each app install first)

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app1',
]
```

Step 5: First **write urls.py** in p1 for each function defined in views.py

```
from django.contrib import admin
from django.urls import path
from app1 import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('learndj/', views.learn_django),
    path('learnpy/', views.learn_python),
```

```
    path('',views.home),
]
```

Step 6: write functions in views.py

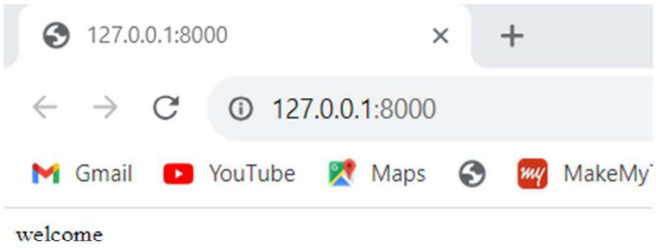
```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.

def learn_django(request):
    return HttpResponse('Hello Django')
def learn_python(request):
    return HttpResponse('<h1>Hello Python</h1>')
def home(request):
    return HttpResponse('welcome')
```

Step 7: runserver

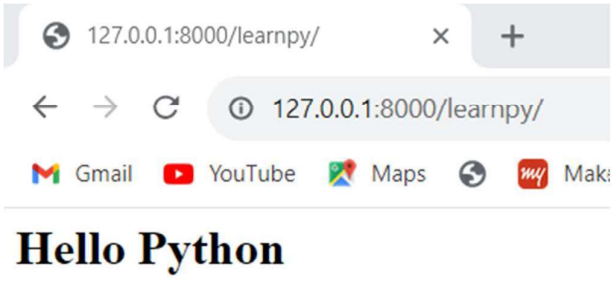
python manage.py runserver (Terminal)

output (depend on url and view function)

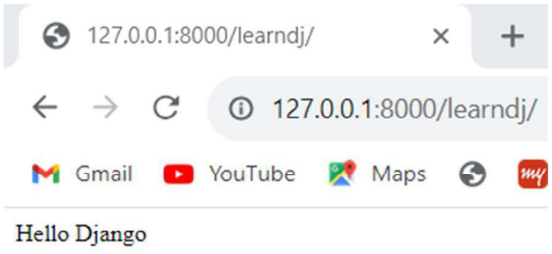
case 1: home function and '' url



Case-2 learn_python function and learnpy url



Case-3 learn_django function and learndj url



Project 2

include ( )

A function that takes a full Python import path to another URLconf module that should be “included” in this place.

Optionally, the application namespace and instance namespace where the entries will be included into can also be specified.

include() also accepts as an argument either an iterable that returns URL patterns or a 2-tuple containing such iterable plus the names of the application namespaces.

urlpatterns can “include” other URLconf modules.

Syntax:-

include(module, namespace=None)

include(pattern_list)

include((pattern_list, app_namespace), namespace=None)

Where,

module – URLconf module (or module name)

namespace (str) – Instance namespace for the URL entries being included

pattern_list – Iterable of path() and/or re_path() instances.

app_namespace (str) – Application namespace for the URL entries being included

Syntax:-

include(module, namespace=None)

include(pattern_list)

include((pattern_list, app_namespace), namespace=None)

Example:-

from django.urls import include, path

```
urlpatterns = [
        path('cor/', include('course.urls')),
]
urlpatterns = [
        path('cor/', include([
                path('learndj/', views.learn_django),
                path('learnpy/', views.learn_python)
        ])),
]
```

Step 1: python -m django startproject vishal2 (Terminal)

Step2: cd p1 (Terminal)

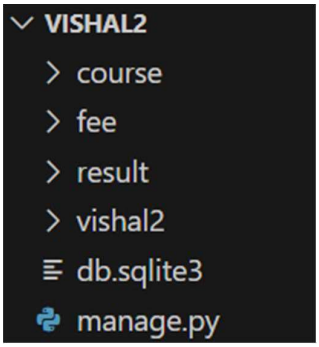
Step3: python manage.py startapp course

python manage.py startapp fee

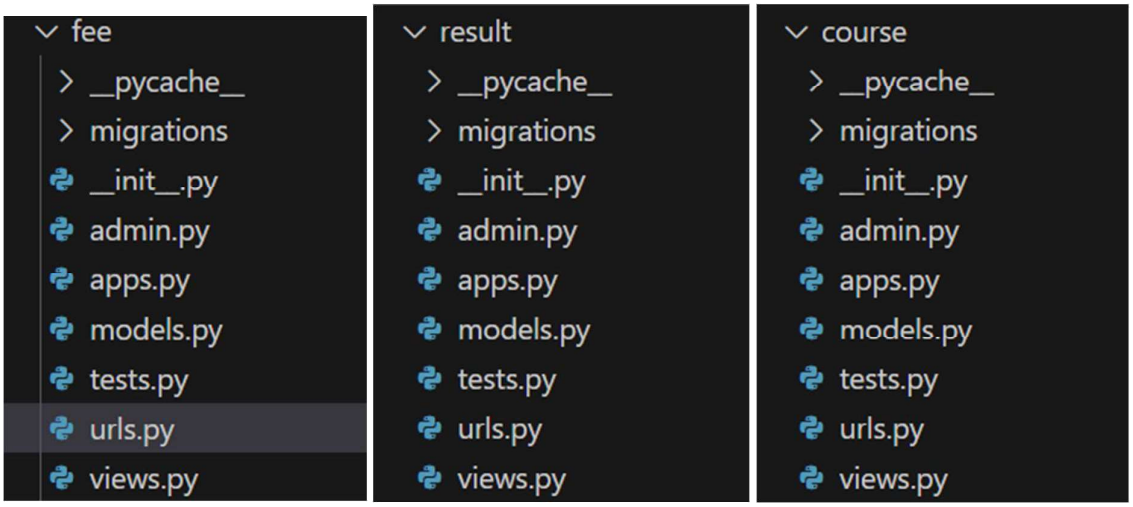
python manage.py startapp result

and install app

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



Step4: make individual urls.py



Step 5: define the function in views.py and urls.py for each application

Course

Views.py

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.

def learn_django(request):
    return HttpResponse('Hello Django')

def learn_python(request):
    return HttpResponse('<h1>Hello Python</h1>')
```

urls.py

```
from course import views
from django.urls import path
urlpatterns = [
    path('learndj/', views.learn_django),
    path('learnpy/', views.learn_python),
]
```

Fee

Views.py

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.

def fees_django(request):
    return HttpResponse('300')
```

urls.py

```
from fee import views
from django.urls import path
urlpatterns = [
    path('fee/', views.fees_django),

]
```

Result

Views.py

```
from django.shortcuts import render
from django.http import HttpResponse
# Create your views here.

def result_django(request):
    return HttpResponse('300')
```

urls.py

```
from result import views
from django.urls import path
urlpatterns = [
    path('result/', views.result_django),

]
```

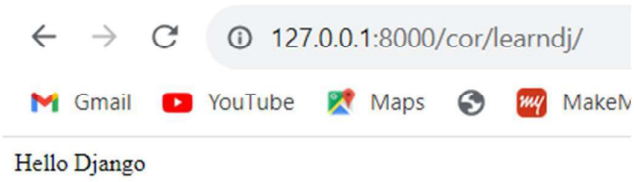
Step 6: register all url in main (inner project folder) urls.py

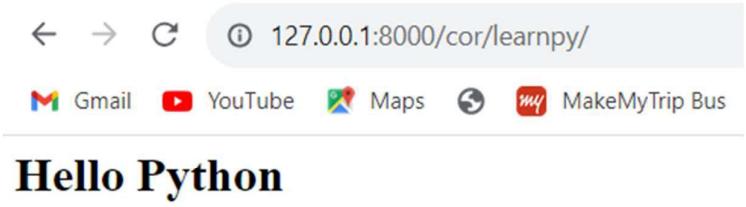
Vishal2>urls.py

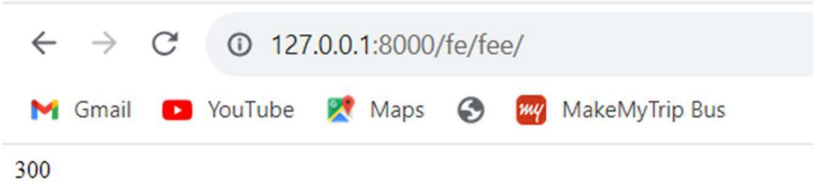
```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('cor/', include('course.urls')),
    path('fe/', include('fee.urls')),
    path('re/', include('result.urls')),

]
```

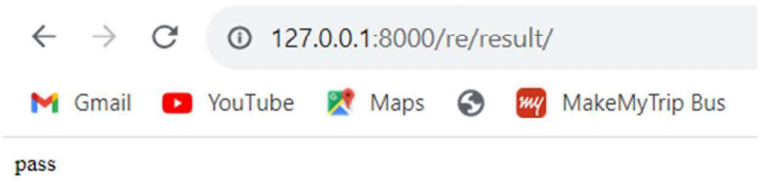
Step 7: python manage.py runserver

Output for each URL (show in the address bar in screenshots)









Project-3(templates, static)

Template

- A template is a text file. It can generate any text-based format (HTML, XML, CSV, etc.).
- A template contains variables, which get replaced with values when the template is evaluated, and tags, which control the logic of the template.
- Template is used by view function to represent the data to user.
- User sends request to view then view contact template afterthat view get information from the template and then view gives response to the users.

We create templates folder inside Project Folder. templates folder will contain all html files.

- Create Django Project: *django-admin startproject project3*
- Create Django Application1: *python manage.py startapp course*
- Create Django Application2: *python manage.py startapp fees*
- *Create Django Applicatio3:python manage.py startapp result*
- Add/Install Applications to Django Project (course and fees to project3) using settings.py INSTALLED_APPS

- Create **templates** folder inside Root Project Directory
- Add **templates** directory in settings.py

```
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')
TEMPLATES_DIR = BASE_DIR / 'templates'
TEMPLATES = [ {
        'DIRS': [TEMPLATES_DIR],      } ]
```

- Create Separate Directory for each application, inside templates directory
- Create template files inside templates/temp_application folder/directory
- Write View Function inside views.py file
- Define url for view function of application using urls.py file

Rendering Templates Files

For rendering file

If you create course app .after create templates folder in course app. After create vha.html file in templates folder.

```
def learn_django(request):
    return render(request, 'vha.html')
```

If you create course app .after creating a templates folder in the course app. After creating the course folder in the templates folder.After creating vha.html file in the course folder.

```
def learn_django(request):
    return render(request, 'course/vha.html')
```

By Creating Template file for application we separate business logic and presentation from the application *views.py* file. Now we will write business logic in views.py file and presentation code in html file.

Still *views.py* will be responsible to process the template files for this we will use *render()* function in *views.py* file.

views.py

```
from django.shortcuts import render
def function_name(request):
    Dynamic Data, if else, any python code logic
    return render(request, template_name, context=dict_name,
content_type=MIME_type, status=None, using=None)
def learn_django(request):
    return render(request, 'course/courseone.html')
```

Note: your exam does not require creating a virtualenv

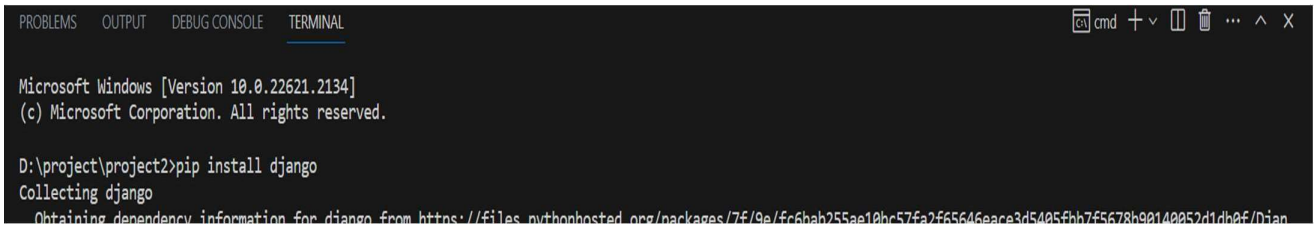
Use Visual Studio code (vs code)

Use cmd Terminal in vs code

Open command palette(ctrl+shift+p)

Python: select interpreter

- **HOW TO INSTALL DJANGO**
- **pip install Django**



- **For checking purpose**

PROJECT3(upload on drive project3)

Steps: complete project

Step1: python-m djsngo startproject

project1Step2:cd project1

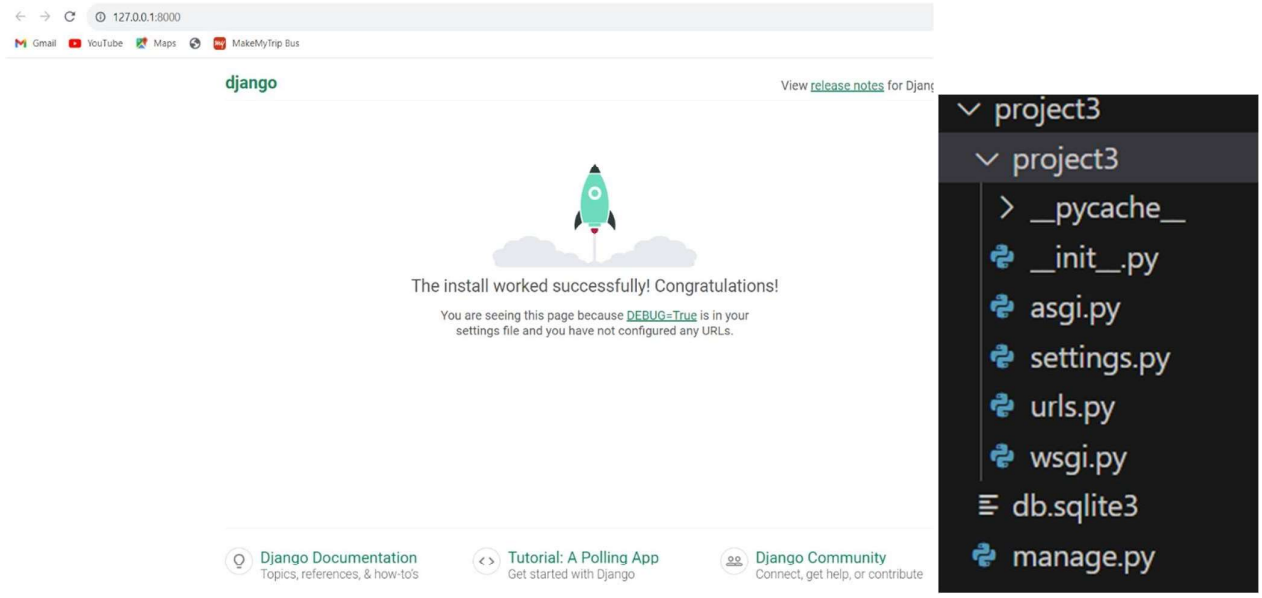
Step3: python manage.py makemigrations

Step4: python manage.py migrate

Step 5: python manage.py runserver

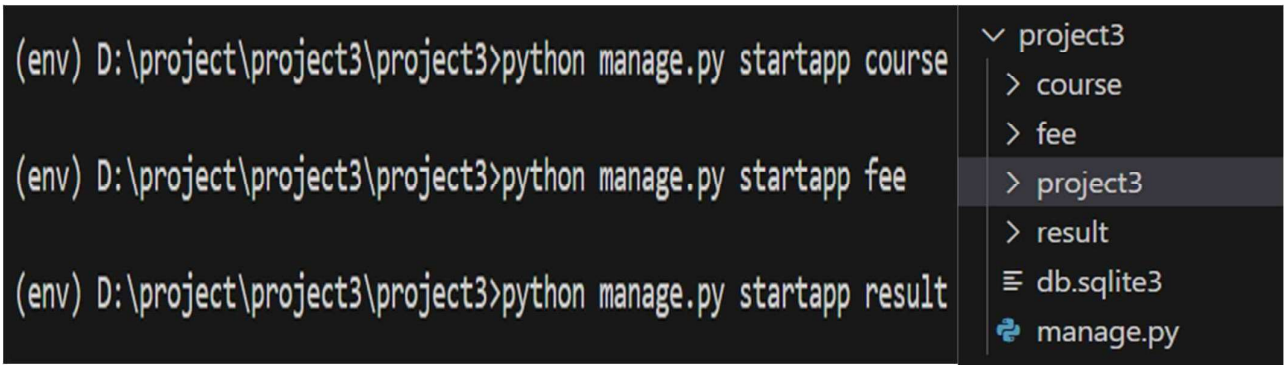
```
(env) D:\project\project3>django-admin startproject project3

(env) D:\project\project3>cd project3

(env) D:\project\project3\project3>python manage.py makemigrations
No changes detected

(env) D:\project\project3\project3>python manage.py migrate
Operations to perform:
```

Paste the http://121.0.0.1:8000/ in the browser



Step 6 Create an application

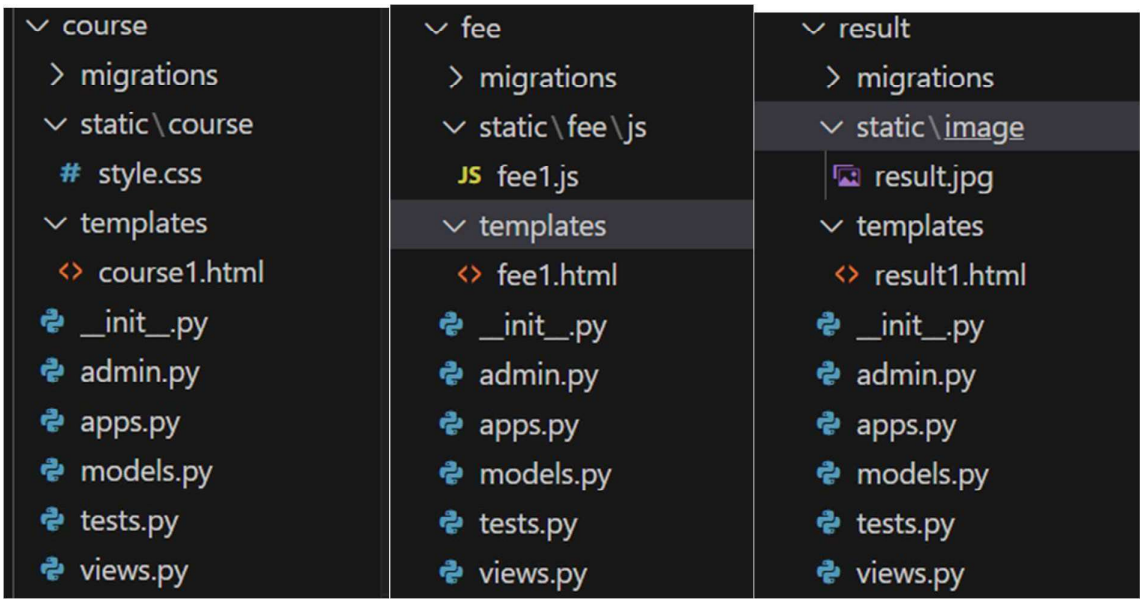
Python manage.py startapp app1



Step 7 Install apps in the inner project directory>setting.py

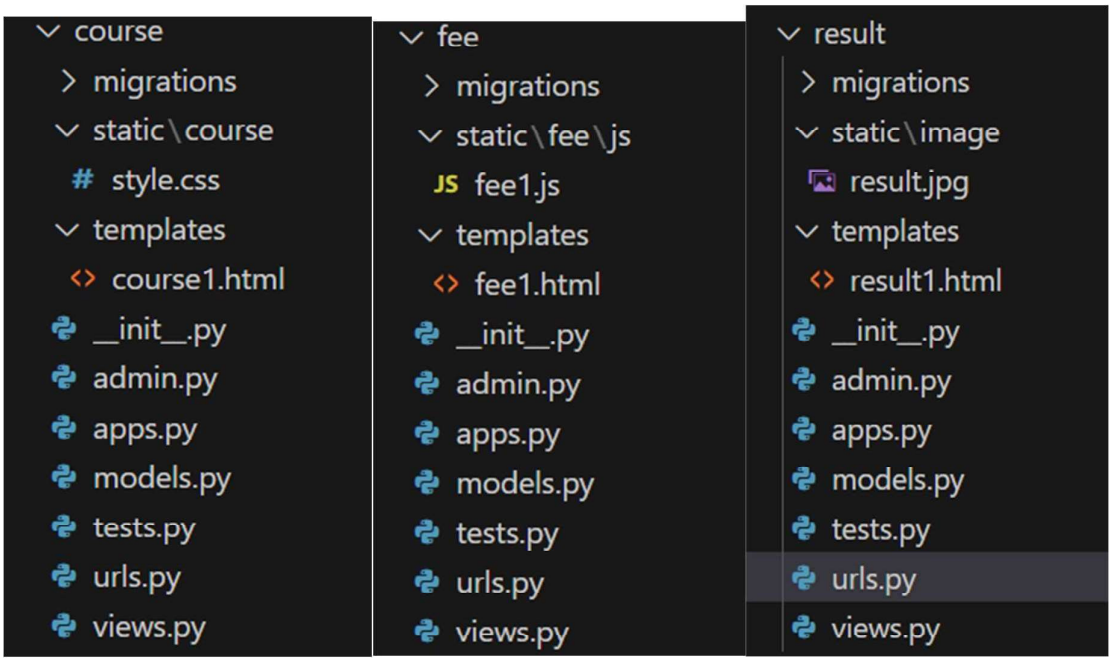
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'course',
    'fee',
    'result',

]
```

Step 8 Create static files (CSS, image, javascript) and templates in each app.(note: not required any change in setting.py for app static and templates folder)



Here first make 3 app structures (generally one by one)

Step 9 Make individual URLs for each app (urls.py)



Step 10 Register this URL in the inner project2 directory>urls.py

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('course/', include('course.urls')),
    path('fee/', include('fees.urls')),
    path('result/',include('result.urls')),
]
```

Step 11 Write URL in individual apps

urls.py

project3 > course > urls.py > ...

```python
from django.urls import path
from course import views
urlpatterns = [
    path('cse/',views.cseb1 ),

]
```

urls.py

project3 > fee > urls.py > ...

```python
from django.urls import path
from fee import views
urlpatterns = [
    path('feecse/',views.feecseb1 ),

]
```

urls.py

project3 > result > urls.py > ...

```python
from django.urls import path
from result import views
urlpatterns = [
    path('resultcse/',views.resultcseb1 ),
]
```

Step 12 Write views.py for each app

views.py

project3 > course > views.py > ...

```python
from django.shortcuts import render

# Create your views here.
def cseb1(request):
    return render(request,"course1.html")
```

views.py

project3 > fee > views.py > ...

```python
from django.shortcuts import render

# Create your views here.
def feecseb1(request):
    return render(request,"fee1.html")
```

views.py

project3 > result > views.py > ...

```python
from django.shortcuts import render

# Create your views here.
def resultcseb1(request):
    return render(request,"result1.html")
```

Step 13 Write HTML,CSS, and js code

Course1.html

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
```

```
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="{% static 'course/style.css' %}">
    <title>Home</title>
  </head>
  <body>
    <h3>Hello</h3>
  </body>
</html>
```

Style.css

```
body {
    background-color: darkslategray;
  }
```

Fee1.html

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
       <title>fee</title>
  </head>
  <body>
    <h2>cse Fees: 4000</h2>

    <hr />
    <form action="">
      <input type="button" value="Click Me" onclick="disp()" />
    </form>
    <script src="{% static 'fee/fee1.js' %}"></script>
  </body>
```

Fee1.js

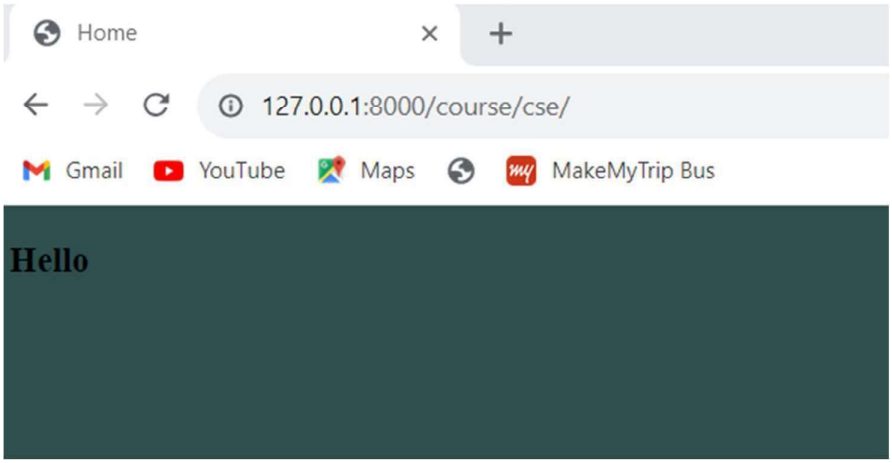
```
function disp() {
    alert("Main hoon JavaScript Fees Ka");
  }
```

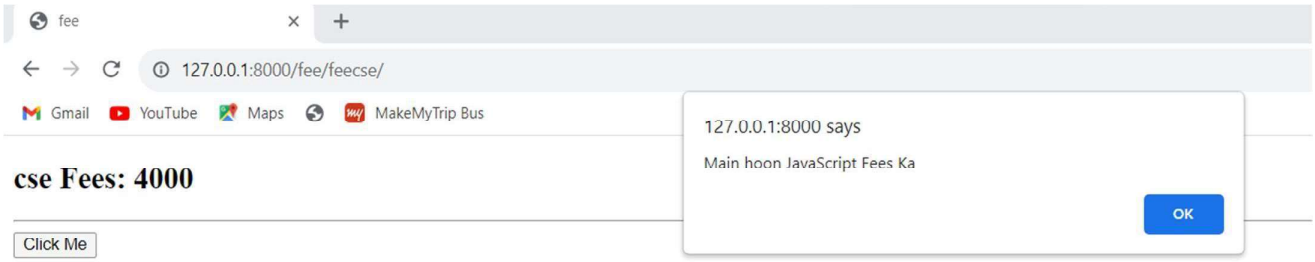
Result1.html

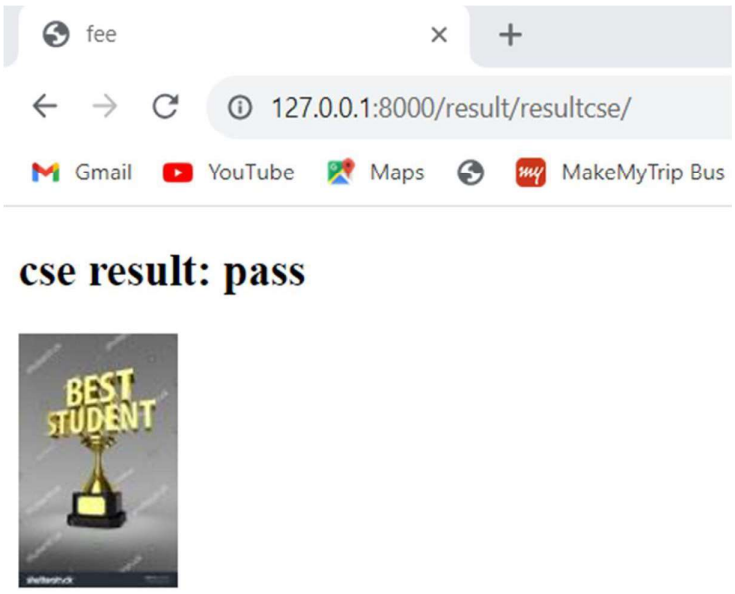
```
<!DOCTYPE html>
{% load static %}
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
```

```
        <title>fee</title>
  </head>
  <body>
    <h2>cse result: pass</h2>
    <img src="{% static '/images/result.jpg'%}" alt="vishal" />
      </body>
</html>
```

Output:

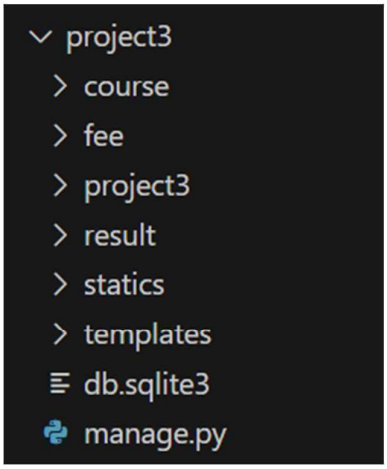






Step14: Create static and templates in an outer directory

settings.py



TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')

STATIC_DIR = os.path.join(BASE_DIR, 'static')

TEMPLATES = [

{

'DIRS': [TEMPLATES_DIR]

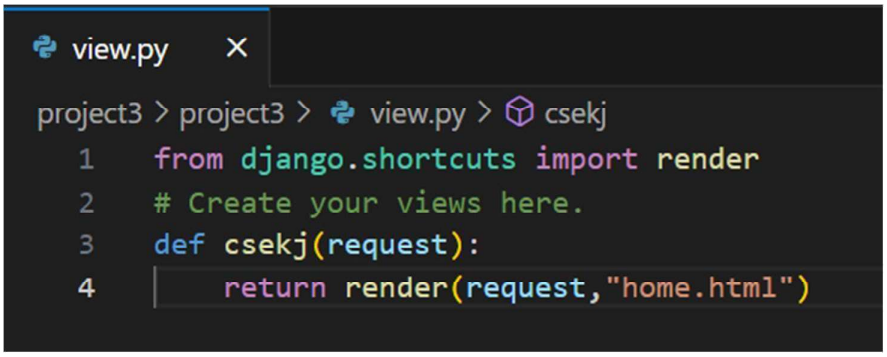
}

]

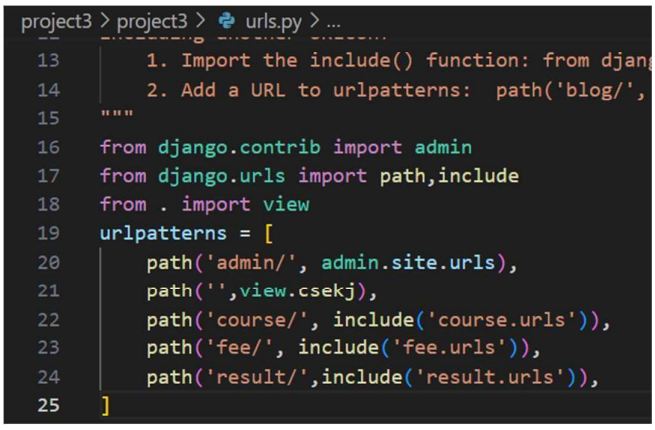
STATIC_URL = '/static/'

STATICFILES_DIRS = [STATIC_DIR]

Step 15: Create a view function in the inner project directory



Step 16: Register URLs



Step 17: write html and CSS file

styles.css

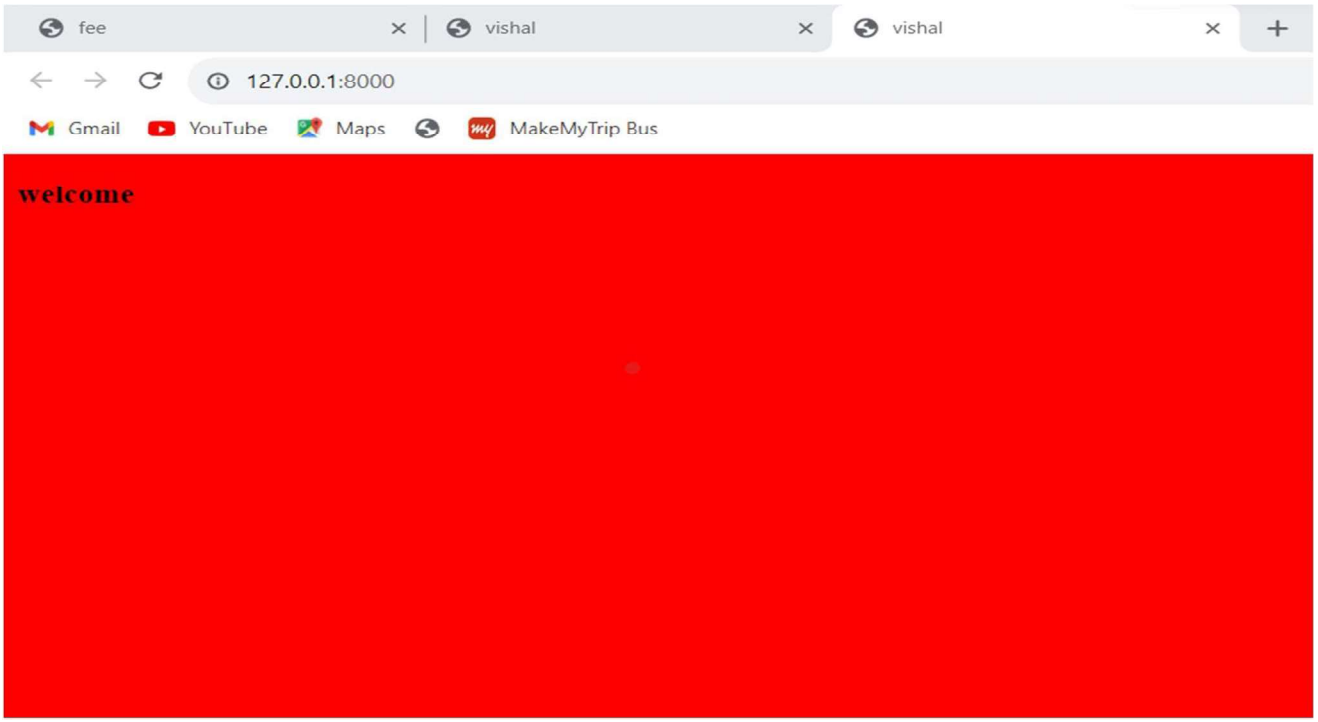
project3 > static > # styles.css > body

```
1  body {
2      background-color: red;
3  }
```

home.html

project3 > templates > home.html

```
1  <!DOCTYPE html>
2  {% load static %}
3  <html lang="en">
4    <head>
5      <meta charset="UTF-8" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <link rel="stylesheet" href="{% static 'styles.css' %}">
8      <title>vishal</title>
9    </head>
10   <body>
11     <h3>welcome</h3>
12   </body>
13 </html>
```



static Template Tag

{% static filename %} – This tag is used to link to static files that are saved in STATIC_ROOT. If the django.contrib.staticfiles app is installed, the tag will serve files using url() method of the storage specified by STATICFILES_STORAGE.

Syntax:-

{% load static %}

{% static filename %}

{% static path/filename %}

{% static path/filename as variable %}

Example:-

<link rel="stylesheet" href="{% static 'style.css' %}" >

<link rel="stylesheet" href="{% static 'css/style.css' %}" >

<img src="{% static 'images/love.jpg' %}">

{% static "images/love.jpg" as mylove %}

<img src="{{ mylove }}">

Project 4

Dynamic Template Files

views.py

```
from django.shortcuts import render
def learn_django(request):
        cname = 'Django'
        duration = '4 Months'
        seats = 10
        django_details = {'nm':cname, 'du':duration, 'st':seats}
        return render(request, 'course/courseone.html', django_details)
```

templates/course

***courseone*.html**

```
<html>
  <body>
        <h2> Course Name:{{nm}} Duration:{{du}} and Total Seats: {{st}}</h2>
    </body>
</html>
```

if Tag

Syntax:-

```
{% if variable %}
        ……..
{% endif %}
```

Example:-

```
{% if nm %}
        </h1>Hello {{nm}}</h1>
{% endif %}
```

Example:-

```
{% if nm == 'Django'  %}
        </h1>Hello {{nm}}</h1>
{% endif %}
```

if elif else Tag

Syntax:-

```
{% if variable %}
        ……..
{% elif variable %}
        ……..
{% else %}
        ……..
{% endif %}
```

Example:-

```
{% if nm %}
        </h1>Hello {{nm}}</h1>
{% elif st %}
        </h1>Seats {{st}}</h1>
{% else}
        <h1>No Course Available</h1>
{% endif %}
```

Example:-

```
{% if nm==‘Django’ %}
        </h1>Hello {{nm}}</h1>
{% elif st==5 %}
        </h1>Seats {{st}}</h1>
{% else}
        <h1>No Course Available</h1>
{% endif %}
```

for loop Tag

Syntax:-

```
{% for variable in variables %}
   {{ variable }}
{% endfor %}
```

Example:-

```
<ul>
{% for stu in student %}
   <li>{{ stu }}</li>
{% endfor %}
</ul>
```

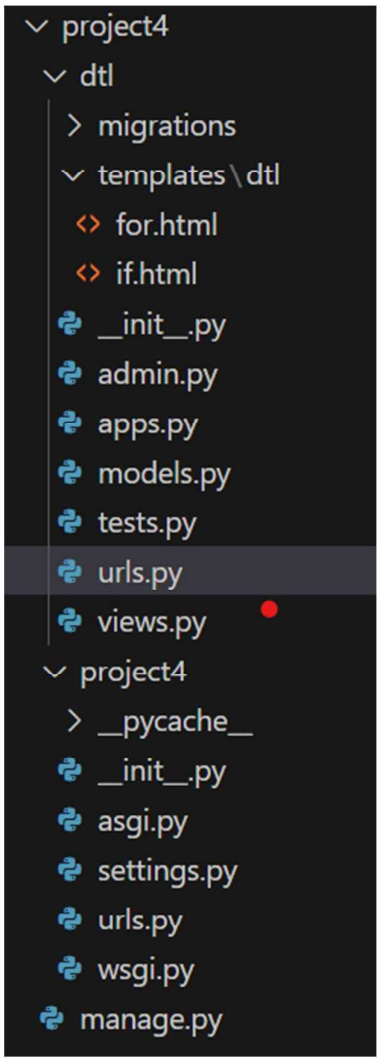
Syntax:-

```
{% for variable in variables %}
   {{ variable }}
{% empty %}
   Empty
{% endfor %}
```

Syntax:-

```
{% for key, value in data.items %}
   {{ key }}: {{ value }}
{% endfor %}
```

Step1: python -m django startproject project4
Step2: cd project4
Step3: python manage.py startapp dtl
Step4: install app in setting.py
Step 5: Create a templates folder in dtl.after creating the dtl folder .afetr creating if.html and for.html in dtl folder. Also make a urls.py file in dtl app



Step 6 write for.html and if.html

For.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>for</title>
</head>
<body>
    <ul>
```

```
        {% for stu in student %}
            <li>{{ stu }}</li>
        {% endfor %}
        </ul>

</body>
</html>
```

If.html

```
<html>
   <body>
    <h2> Course Name:{{nm}} Duration:{{du}} and Total Seats: {{st}}</h2>
   <h1> Example:-if elif else</h1>

{% if nm == 'Django' %}
    </h1>Hello {{nm}}</h1>
{% elif st == 5 %}
    </h1>Seats {{st}}</h1>
{% else}
    <h1>No Course Available</h1>
{% endif %}

    </body>
</html>
```

Step 7: Rendering html files in view.py

Views.py

```
from django.shortcuts import render

# Create your views here.
def for_demo(request):
    return render(request,'dtl/for.html',{'student':[1,2,3,4,5]})
def if_demo(request):
    cname = 'Django'
    duration = '4 Months'
    seats = 10
    django_details = {'nm':cname, 'du':duration, 'st':seats}
    return render(request, 'dtl/if.html', django_details)
```

Step 8: make a url for each function in urls.py(in app)

```
from django.contrib import admin
from django.urls import path
from . import views
urlpatterns = [
    path('for/',views.for_demo),
    path('if/',views.if_demo),
]
```

Step 9: register urls.py (in app(dtl)) in Project4>urls.py

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

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

Step10: python manage.py runserver

Output:

Project 5

Template Inheritance/ Template Extending

- Template inheritance allows you to build a base "skeleton" template that contains all the common elements of your site and defines blocks that child templates can override.
- The *extends* tag is used to inherit template.
- extends tag tells the template engine that this template "extends" another template.
- When the template system evaluates this template, first it locates the parent let's assume, "base.html".
- At that point, the template engine will notice the block tags in base.html and replace those blocks with the contents of the child template.
- You can use as many levels of inheritance as needed.

extends Tag

{% extends %} – The *extends* tag is used to inherit template. It tells the template engine that this template "extends" another template. It has no end tag.

Syntax:-

{% extends 'parent_template_name' %}

{% extends variable %}

Example:-

{% extends "./base1.html" %}

{% extends "../base2.html" %}

{% extends "./my/base3.html" %}

{% extends somthing %}

block Tag

{% block %} – The *block* tag is used to for overriding specific parts of a template.

Syntax:-

{% block blockname %}….{% endblock %}

{% block blockname %}….{% endblock blockname %}

Example:-

{% block title %} ….. {% endblock %}

{% block content %}….. {% endblock content %}

base.html

```
<html>
<head>
<title>{% block title %} Other {% endbock %}</title>
</head>
<body>
{% block content %} {% endblock content %}
</body>
</html>
```

home.html

```
{% extends 'base.html' %}
{% block title %} {{block.super}}
Home{% endblock %}


{% block content %}
  <h1>Hello I am Home Page </h1>
{% endblock content %}
```

about.html

```
{% extends 'base.html' %}

{% block content %}
  </h1>Hello I am About Page</h1>
{% endblock content %}
```

Step1: python -m django startproject project5

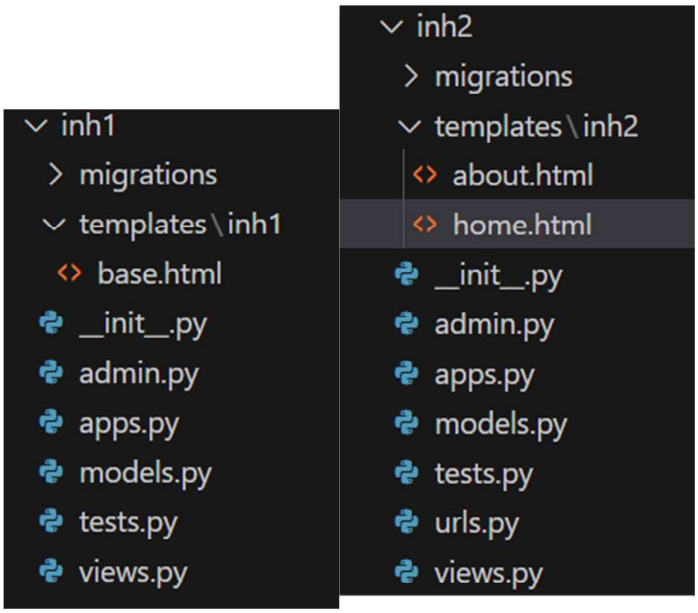
Step2: cd project5

Step3: python manage.py startapp inh1

Step4: python manage.py startapp inh2

Install apps in setting.py

Step5: create below structure



Step6: write html files

Base.html

```
<html>
<head>
<title> {% block title %}other{% endblock title %} </title>
</head>
<body>
{% block content %} {% endblock content %}
</body>
</html>
```

Home.html

```
{%  extends 'inh1/base.html' %}
{% block title %} Home{% endblock %}
{% block content %}
   <h1>Hello I am Home Page </h1>
{% endblock content %}
```

About.html

```
{%  extends 'inh1/base.html' %}
{% block content %}
   </h1>Hello I am About Page</h1>
{% endblock content %}
```

Step7: rendering html files by function in views.py(inh2)

Inh2>views.py

```
from django.shortcuts import render

# Create your views here.
def homes(request):
    return render(request,'inh2/home.html')
def abouts(request):
    return render(request,'inh2/about.html')
```

Step 8: write URL for each function

Inh2>urls.py

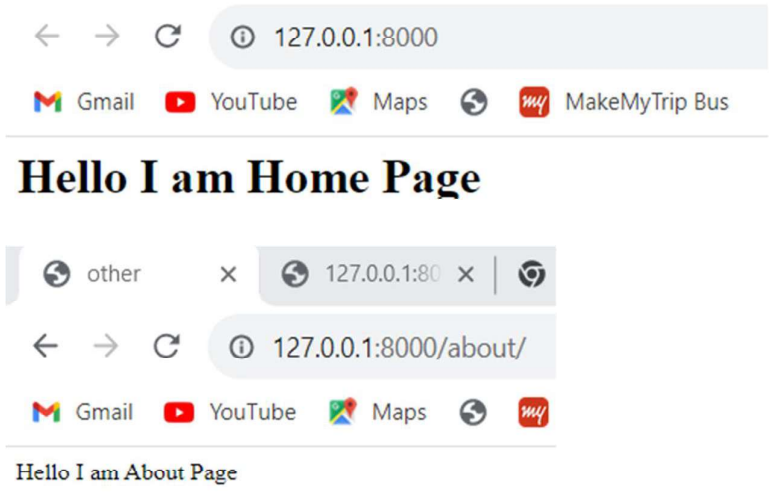
```
from django.contrib import admin
from django.urls import path,include
from . import views
urlpatterns = [

    path('',views.homes),
    path('about/',views.abouts)
]
```

Step 9: register urls.py using include

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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('inh2.urls')),
]
```

Step10: run server

Output:



Project 6

url Tag

{% url %} - It returns an absolute path reference (a URL without the domain name) matching a given view and optional parameters. Any special characters in the resulting path will be encoded using iri_to_uri().

Syntax:-

{% url 'urlname' %}

{% url 'urlname' as var %}

{% url 'urlname' arg1=value1 arg2=value2 %}

{% url 'urlname' value1 value2 %}

path(route, view, kwargs=None, name=None) - It returns an element for inclusion in urlpatterns.

Here we write the urlname replace with name value

Example

Urls.py

path(route, view, kwargs=None, name='vishal')

html files

{% url 'vishal' %}

Step1: python -m django startproject project6

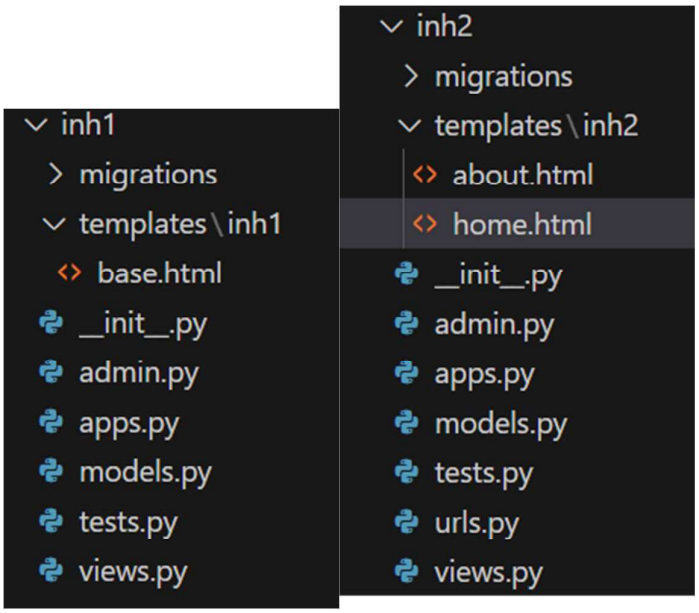
Step2: cd project5

Step3: python manage.py startapp inh1

Step4: python manage.py startapp inh2

Install apps in setting.py

Step5: create below structure



Step6: write html files

Base.html

```
<html>
<head>
<title> {% block title %}other{% endblock title %} </title>
</head>
<body>
    <nav>
    <a href="{% url 'home1' %}">HOME</a> 

    <a href="{% url 'about1' %}">ABOUT</a> 

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

Home.html

```
{%  extends 'inh1/base.html' %}
{% block title %} Home{% endblock %}
{% block content %}
   <h1>Hello I am Home Page </h1>
{% endblock content %}
```

About.html

```
{%  extends 'inh1/base.html' %}
{% block content %}
   </h1>Hello I am About Page</h1>
{% endblock content %}
```

Step7: rendering html files by function in views.py(inh2)
Inh2>views.py

```
from django.shortcuts import render

# Create your views here.
def homes(request):
    return render(request,'inh2/home.html')
def abouts(request):
    return render(request,'inh2/about.html')
```

Step 8: write URL for each function
Inh2>urls.py

```
from django.contrib import admin
```

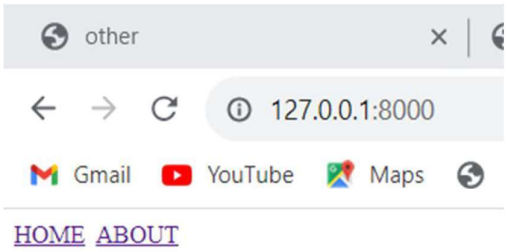
```
from django.urls import path,include
from . import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('',views.homes,name='home1'),
    path('about/',views.abouts,name='about1')
]
```

Step 9: register urls.py using include

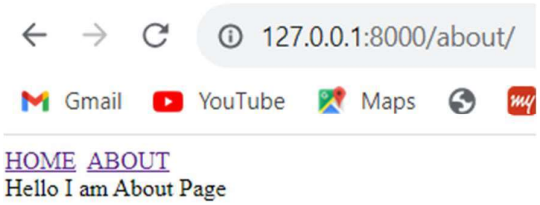
```
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('inh2.urls')),
]
```

Step10: run server

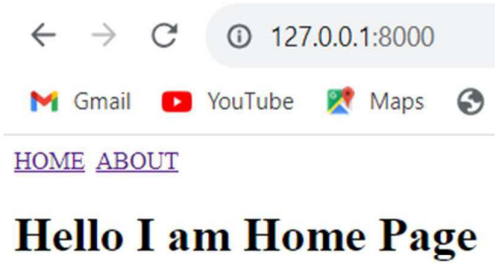
Output:



Click ABOUT



Click Home



Project 7

Step1: python -m django startproject project7
Step2: cd project7
Step3: python manage.py startapp myapp
Step4: Install myapp in setting.py
Step5: Generate a model in models.py in myapp
Myapp>models.py

```
from django.db import models
# Create your models here.
class Product(models.Model):
    name = models.CharField(max_length=100)
    description = models.TextField()

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

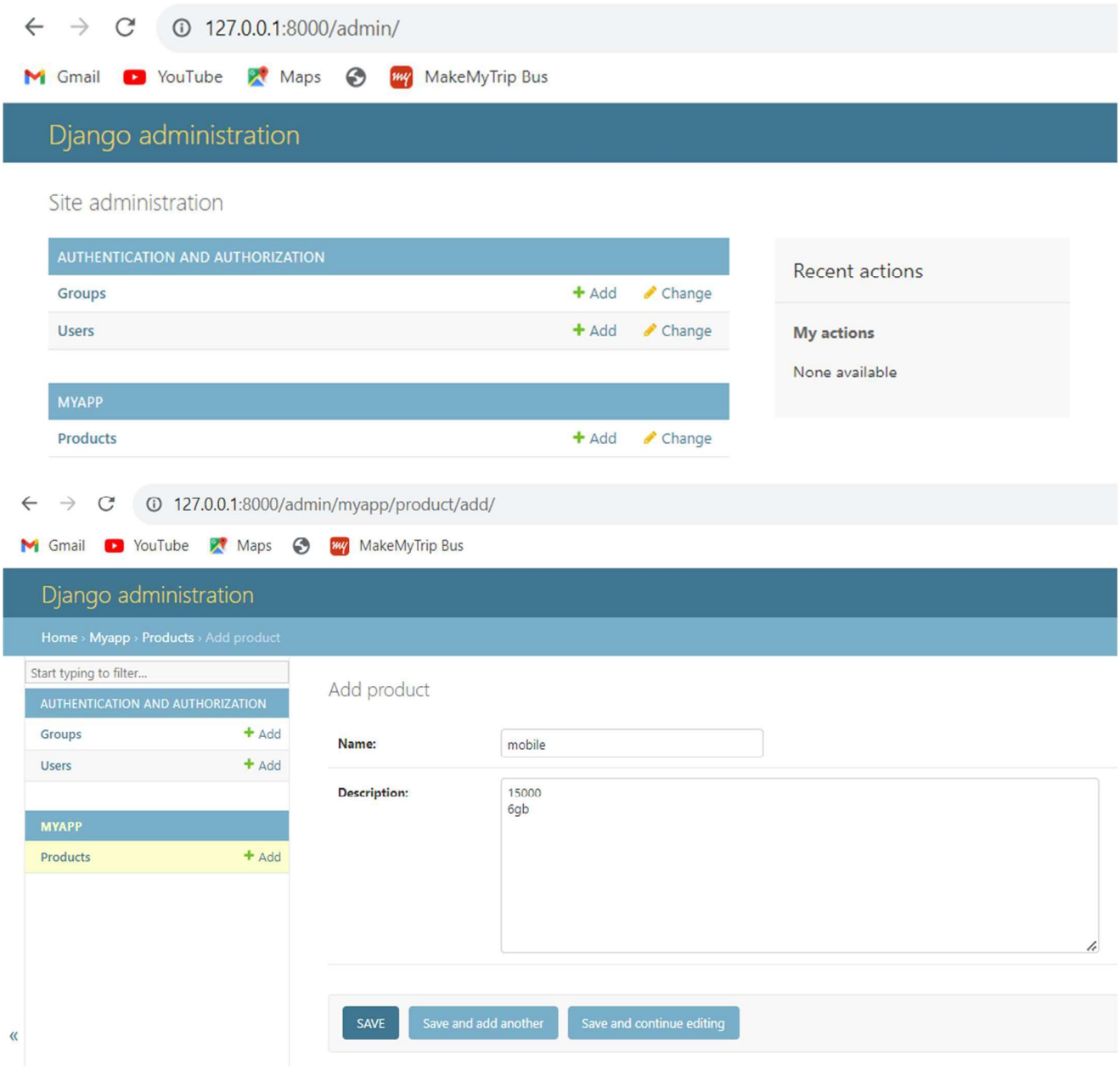
Step 6: Register the model in admin.py
Myapp>admin.py

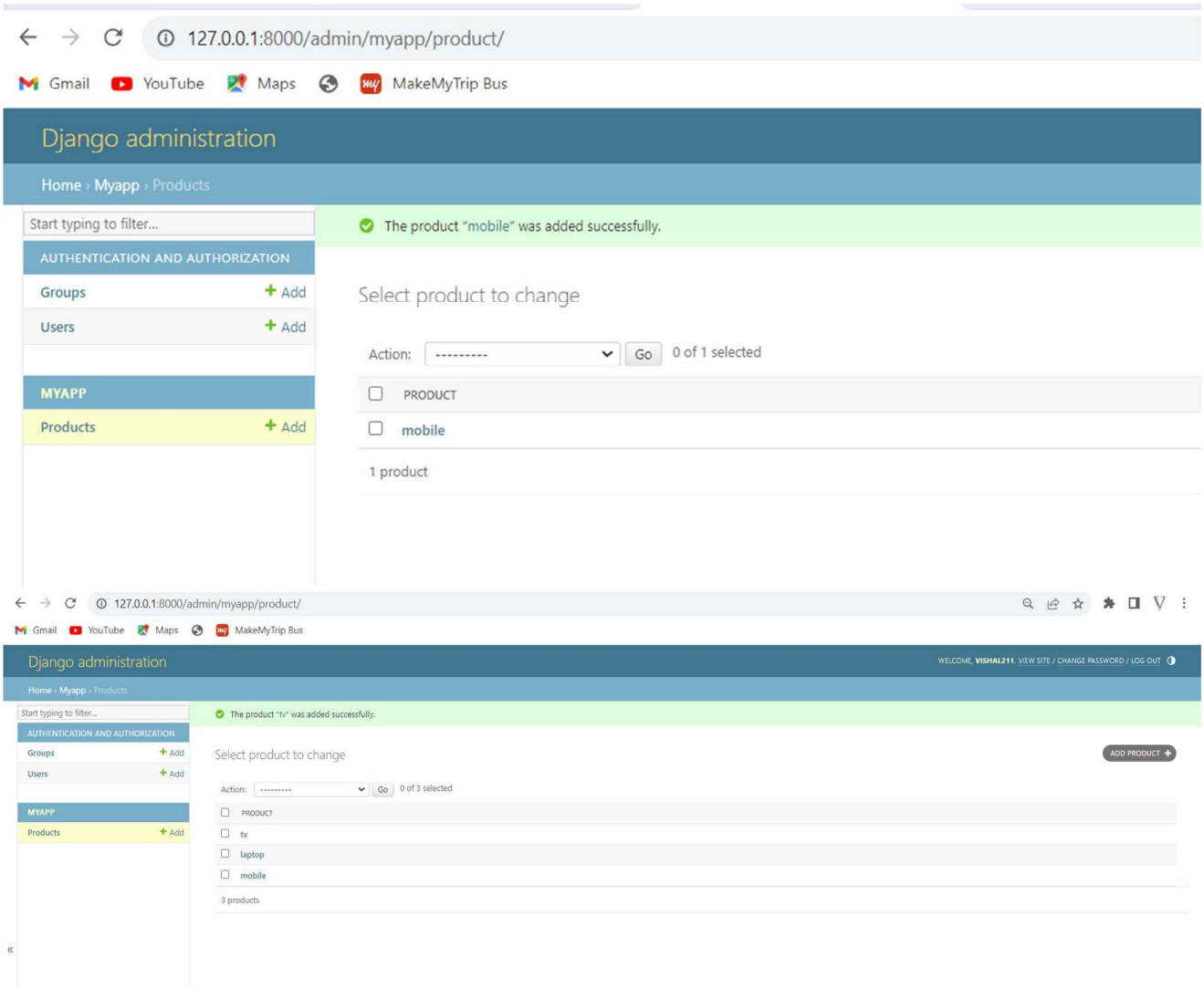
```
from django.contrib import admin
from myapp.models import Product
admin.site.register(Product)
```

Step7: for query generation
python manage.py makemigrations
Step 8: for query execute
python manage.py migrate
Step 9: for admin user

```
D:\django\project7\project7>python manage.py createsuperuser
Username (leave blank to use 'vishal'): vishal211
Email address: V@gmail.com
Password:
Password (again):
The password is too similar to the username.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

Step 10: python manage.py runserver
output:


127.0.0.1:8000/admin/
Gmail
YouTube
Maps
MakeMyTrip Bus
Django administration
Site administration
AUTHENTICATION AND AUTHORIZATION
Groups + Add Change
Users + Add Change
MYAPP
Products + Add Change
Recent actions
My actions
None available
127.0.0.1:8000/admin/myapp/product/add/
Gmail
YouTube
Maps
MakeMyTrip Bus
Django administration
Home › Myapp › Products › Add product
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
MYAPP
Products + Add
Add product
Name:
mobile
Description:
15000
6gb
SAVE
Save and add another
Save and continue editing



Display all product lists and dynamic urls

Step 11: first make urls.py in myapp

```
# myapp/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('products/', views.product_list, name='product_list'),
    path('products/<int:product_id>/', views.product_detail, name='product_detail'),
]
```

After register urls.py using include

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

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

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

Step 12: For display make product_list.html and dynamic url product_detail.html

Product_list.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Product List</title>
</head>
<body>
    <h1>Product List</h1>
    <ul>
        {% for product in products %}
            <li>
                <a href="{% url 'product_detail' product.id %}">
                    {{ product.name }}
                </a>
                - {{ product.description }}
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

Product_detail.html

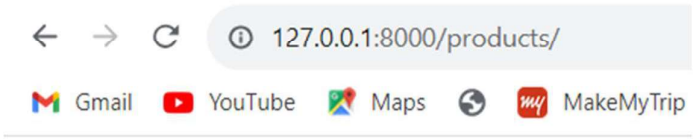
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Product Detail</title>
</head>
<body>
    <h1>Product Detail</h1>
    <h2>{{ product.name }}</h2>
    <p>{{ product.description }}</p>
    <a href="{% url 'product_list' %}">Back to Product List</a>
</body>
</html>
```

Step 13: Rendering HTML file using function-based view

```
# myapp/views.py
# Create your views here.
from django.shortcuts import render, get_object_or_404
```

```python
from .models import Product

def product_list(request):
    products = Product.objects.all()
    return render(request, 'myapp/product_list.html', {'products': products})

def product_detail(request, product_id):
    product = Product.objects.get(pk=product_id)
    #product = get_object_or_404(Product, pk=product_id)
    return render(request, 'myapp/product_detail.html', {'product': product})
```

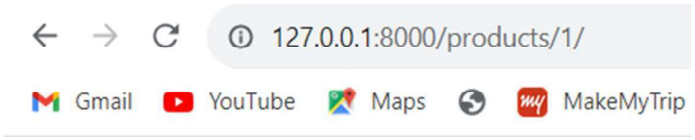
output:



Product List

- mobile - 15000 6gb
- laptop - 200002 8gb ram 521 gb
- tv - 300000 6 gb ram 12 gb

If I click mobile



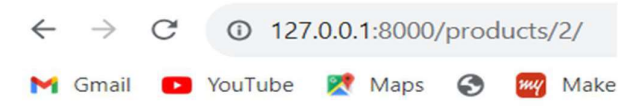
Product Detail

mobile

15000 6gb

Back to Product List

If I click laptop



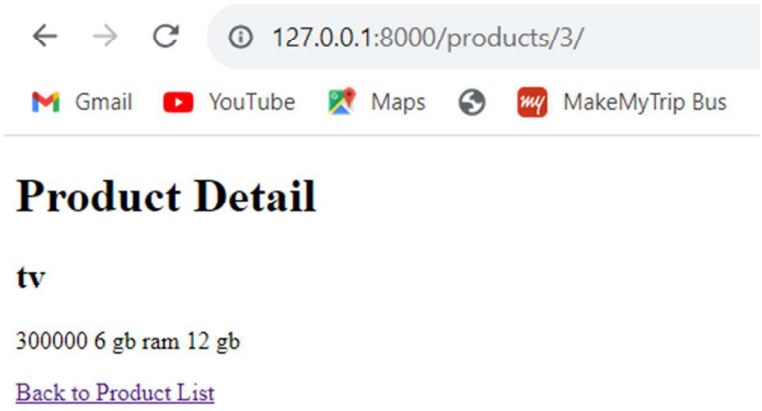
Product Detail

laptop

200002 8gb ram 521 gb

Back to Product List

If I click tv



Modify the project by using the Search button

For that modify product_list.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Product List</title>
</head>
<body>
    <h1>Product List</h1>

    <!-- Search Form -->
    <form method="GET" action="{% url 'product_list' %}">
        <input type="text" name="search" placeholder="Search by Product Name" value="{{
request.GET.search }}">
        <button type="submit">Search</button>
    </form>

    <ul>
        {% for product in products %}
            <li>
                <a href="{% url 'product_detail' product.id %}">
                    {{ product.name }}
                </a>
                - {{ product.description }}
            </li>
        {% empty %}
            <li>No products found.</li>
        {% endfor %}
    </ul>
</body>
</html>
```

Explain

1. We've added an HTML `<form>` element that sends a `GET` request to the same URL (`{% url 'product_list' %}`), which is the URL for the product list view.
2. Inside the form, there is an `<input>` field with the name `search`, which will hold the search term entered by the user. We've also added a `placeholder` attribute to provide a hint to the user.
3. To display any previously entered search term, we set the input field's `value` attribute to `{{ request.GET.search }}`. This will pre-fill the input field with the search term if it was provided in the URL query parameters.
4. We've added a `<button>` element with the type "submit" to create a button for submitting the search form.

For that modify views.py

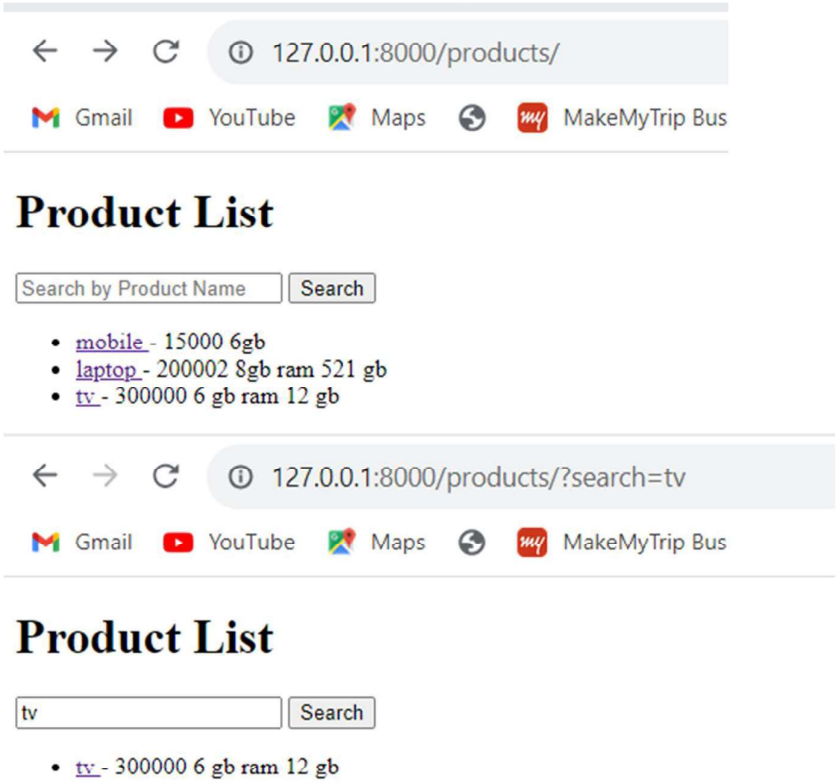
```python
# myapp/views.py
# Create your views here.
from django.shortcuts import render, get_object_or_404
from .models import Product

def product_list(request):
    # Get the search term from the URL query parameters
    search_term = request.GET.get('search')

    # If a search term is provided, filter the products
    if search_term:
        products = Product.objects.filter(name__icontains=search_term)
    else:
        # If no search term is provided, display all products
        products = Product.objects.all()

    return render(request, 'myapp/product_list.html', {'products': products})

def product_detail(request, product_id):
    product = Product.objects.get(pk=product_id)
    #product = get_object_or_404(Product, pk=product_id)
    return render(request, 'myapp/product_detail.html', {'product': product})
```

Explain:

1. We use `request.GET.get('search')` to retrieve the search term from the URL query parameters. The `get()` method returns `None` if the 'search' parameter is not present in the URL.

2. If a search term is provided (`search_term` is not `None`), we filter the products using `Product.objects.filter(name__icontains=search_term)`. This filters products whose names contain the search term in a case-insensitive manner.
3. If no search term is provided, we retrieve all products with `Product.objects.all()`.

output:



Project 8

What is Cross Site Request Forgery (CSRF/ XSRF)

A Cross-site request forgery hole is when a malicious site can cause a visitor's browser to make a request to your server that causes a change on the server. The server thinks that because the request comes with the user's cookies, the user wanted to submit that form

Django provides CSRF Protection with **csrf_token** which we need to add inside the form tag. This token will add a hidden input field with random value in form tag.

templates/enroll / userregistration.html

```
<!DOCTYPE html>
<html>
        <body>
                <form method="post"> {% csrf_token %}
                        {{form}}
                         <input type="submit" value="Submit">
                </form>
        </body>
</html>
```

Step1: python -m django startproject project8

Step2: cd project8

Step3: python manage.py startapp csrf_app

Step 4: install app in setting.py

Step5: create a file called form.html inside the csrf_app/templates directory:

Csrf_app>templates>form.html

```
<html>
<head>
    <title>CSRF Token Demo</title>
</head>
<body>
    <h1>CSRF Token Demo</h1>
    <form method="post">
        {% csrf_token %}
        <label for="input_text">Enter Text:</label>
        <input type="text" id="input_text" name="input_text">
        <button type="submit">Submit</button>
    </form>
</body>
</html>
```

Step 6: Create the views.py for rendering form.html

```
from django.shortcuts import render

# Create your views here.
def csrf_demo(request):
    return render(request, 'form.html')
```

Step7: write url for csrf_demo function in urls.py inside csrf_demo app

Csrf_demo>urls.py

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

urlpatterns = [
    path('', views.csrf_demo, name='csrf_demo'),
]
```

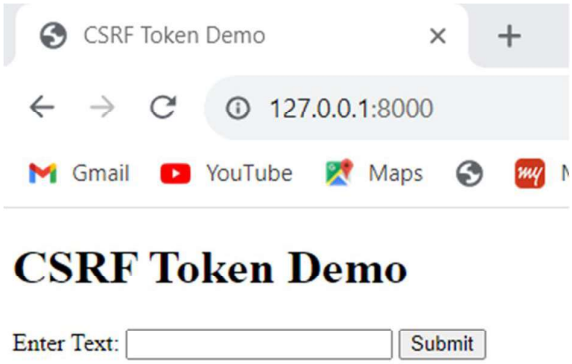
Step 8: register urls.py using include

Project8>urls.py

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

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

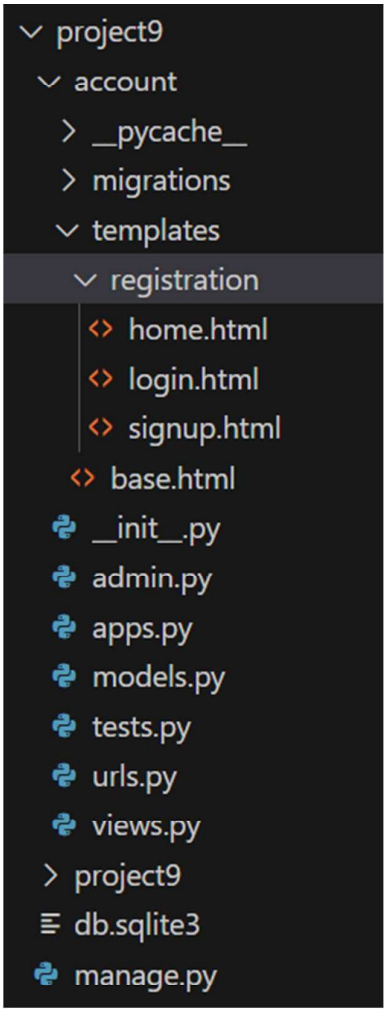
Step9: python manage.py runserver

Output:



Project 9

Step1: python -m django startproject project9
Step2: cd project9
Step3: python manage.py startapp account
Step4: install app in setting.py
Step5: create below structure



Step 6: create urls for signup,login,logout,home in account>urls.py

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

urlpatterns = [
    path('signup/', views.signup, name='signup'),
    path('login/', views.user_login, name='login'),
    path('logout/', views.user_logout, name='logout'),
    path('',views.home1,name='home')
]
```

Register account.urls using includef

Project9>include

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

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

Step7: create base.html for navbar after that extends another html file

```
<!-- base.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <!-- Your head content here -->
</head>
<body>
    <header>
        <!-- Your header content here -->
        <nav>
            <ul>

                {% if user.is_authenticated %}
                    <li><span>Welcome, {{ user.username }}</span></li>
                    <li><a href="{% url 'logout' %}">Logout</a></li>
                {% else %}
                    <li><a href="{% url 'login' %}">Login</a></li>
                    <li><a href="{% url 'signup' %}">Signup</a></li>
                {% endif %}
                    <li><a href="{% url 'home' %}">HOME</a></li>
            </ul>
        </nav>
    </header>

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

Explain in details

1. `{% if user.is_authenticated %}`:
    - This is an opening conditional tag in Django template syntax.
    - It checks whether the `user` object is authenticated (i.e., if the user is logged in).
2. `<li><span>Welcome, {{ user.username }}</span></li>`:

- If the user is authenticated, this line generates an HTML list item (`<li>`) with a welcome message that includes the user's username.
- It uses the `{{ user.username }}` template variable to display the username of the authenticated user.

3. `<li><a href="{% url 'logout' %}">Logout</a></li>`:
   - In the authenticated user block, this line creates an HTML list item with a link to the "Logout" action.
   - It uses the `{% url 'logout' %}` template tag to generate the URL for the logout view.
4. `{% else %}`:
   - This is the opening tag for the "else" block, which is executed if the user is not authenticated (i.e., the "else" part of the conditional statement).
5. `<li><a href="{% url 'login' %}">Login</a></li>`:
   - In the non-authenticated user block, this line generates an HTML list item with a link to the "Login" action.
   - It uses the `{% url 'login' %}` template tag to generate the URL for the login view.
6. `<li><a href="{% url 'signup' %}">Signup</a></li>`:
   - In the non-authenticated user block, this line creates an HTML list item with a link to the "Signup" action.
   - It uses the `{% url 'signup' %}` template tag to generate the URL for the signup view.

Write another html file code

Signup.html

```
{% extends 'base.html' %}

{% block content %}
  <h2>Sign up</h2>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Sign up</button>
  </form>
  <p>Already have an account? <a href="{% url 'login' %}">Login</a></p>
{% endblock %}
```

Login.html

```
{% extends 'base.html' %}

{% block content %}
  <h2>Login</h2>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Login</button>
  </form>
  <p>Don't have an account? <a href="{% url 'signup' %}">Sign up</a></p>
```

```
{% endblock %}
```

Home.html

```
{% extends "base.html" %}
{% block content %} <h1>welcome</h1>{% endblock %}
```

Step 8: writing function for signup,login,logout and home

```
from django.contrib.auth.forms import UserCreationForm,AuthenticationForm
from django.contrib.auth import login,logout,authenticate
from django.shortcuts import render, redirect

def signup(request):
    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        if form.is_valid():
            user = form.save()
            login(request, user)
            return redirect('home')  # Redirect to the home page or any other desired page
    else:
        form = UserCreationForm()
    return render(request, 'registration/signup.html', {'form': form})
def user_login(request):
    if request.method == 'POST':
        fm = AuthenticationForm(request=request, data=request.POST)
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('home')  # Redirect to the home page or any other desired page
    else:
        fm = AuthenticationForm()
    return render(request, 'registration/login.html',{'form':fm})

def user_logout(request):
    logout(request)
    return redirect('login')  # Redirect to the login page or any other desired page
def home1(request):
    return render(request,'registration/home.html')
```

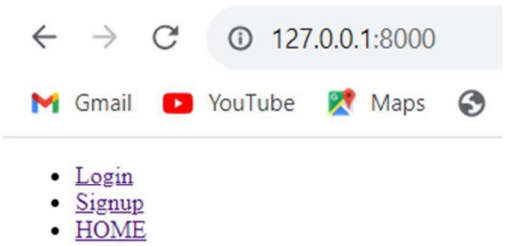
Explain in detail

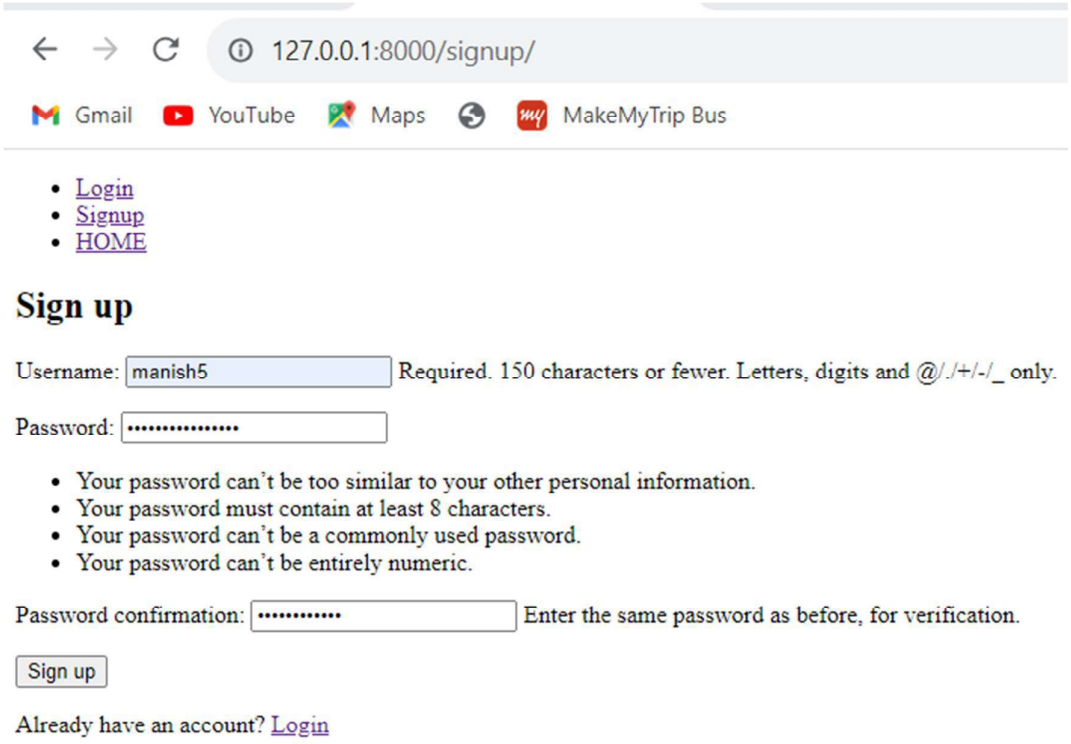
- The `signup` view handles user registration.
- It checks if the HTTP request method is POST. If it is, it creates a `UserCreationForm` instance with the POST data, validates the form, saves the user, logs them in, and redirects to the 'home' page upon successful registration.
- If the method is not POST (i.e., a GET request), it initializes an empty `UserCreationForm` for rendering the signup form.
- Finally, it renders the 'registration/signup.html' template, passing the form as context data.

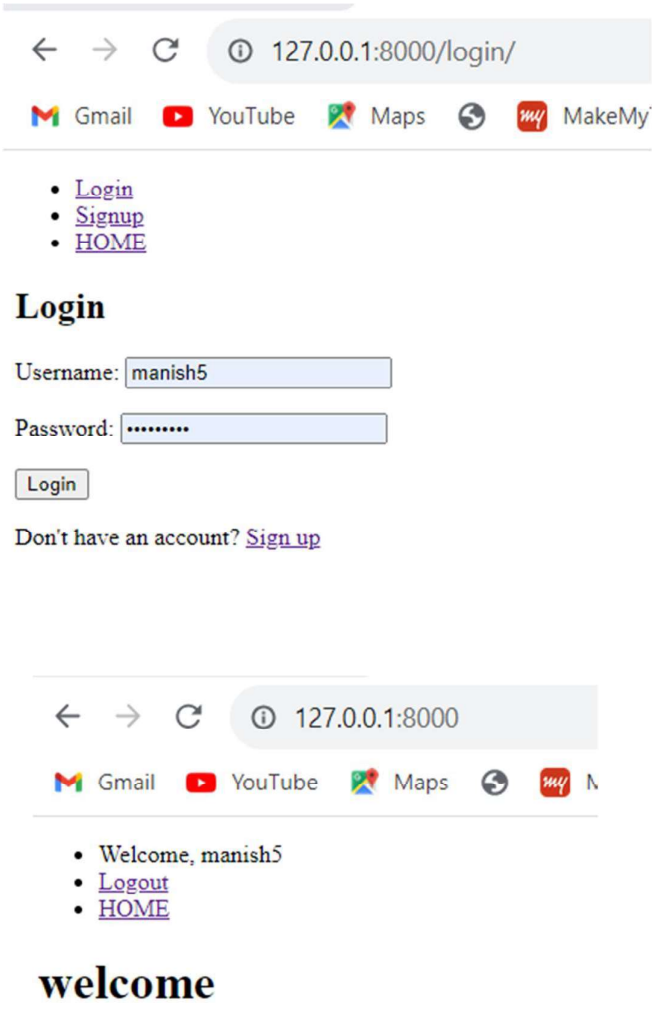
- The `user_login` view handles user login.
- It checks if the HTTP request method is POST. If it is, it creates an `AuthenticationForm` instance with the POST data, attempts to authenticate the user, logs them in, and redirects to the 'home' page upon successful authentication.
- If the method is not POST (i.e., a GET request), it initializes an empty `AuthenticationForm` for rendering the login form.
- Finally, it renders the 'registration/login.html' template, passing the form as context data.

- The `user_logout` view handles user logout.
- It logs the user out using the `logout` function provided by Django and redirects them to the 'login' page.

Step9: python manage.py makemigrations
Step10: Python manage.py migrate
Step11: python manage.py runserver
Output:


127.0.0.1:8000
Gmail
YouTube
Maps
Login
Signup
HOME
welcome

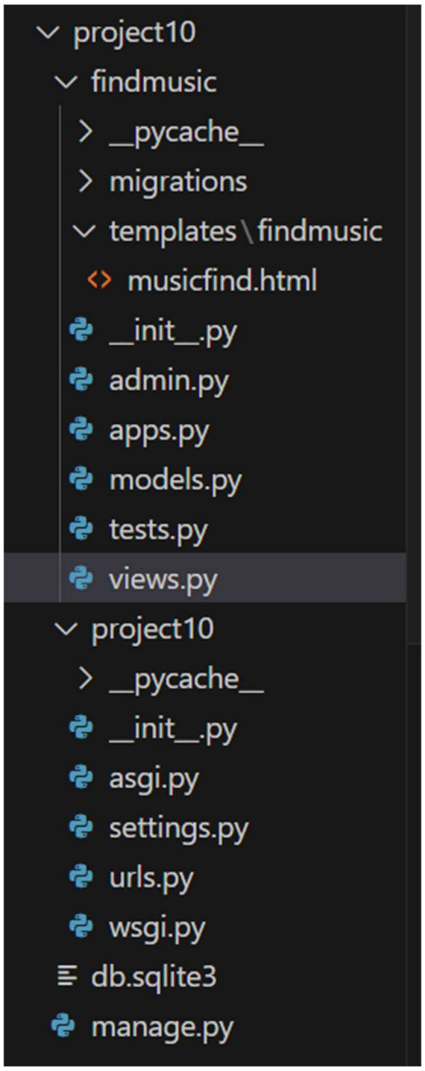

127.0.0.1:8000/signup/
Gmail
YouTube
Maps
MakeMyTrip Bus
Login
Signup
HOME
Sign up
Username: manish5
Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
Password:
Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.
Password confirmation:
Enter the same password as before, for verification.
Sign up
Already have an account? Login


127.0.0.1:8000/login/
Gmail
YouTube
Maps
MakeMy
Login
Signup
HOME
Login
Username: manish5
Password:
Login
Don't have an account? Sign up

127.0.0.1:8000
Gmail
YouTube
Maps
Welcome, manish5
Logout
HOME
welcome

Project 10

Step1: python -m django startproject project10
Step2: cd project10
Step3: python manage.py startapp findmusic
Step4: install the app in setting.py
Step5: create below structure



Step 6: Create a model for music
Findmusic>models.py

```python
from django.db import models

# Create your models here.
class Music(models.Model):
    song = models.CharField(max_length=255)
    artist = models.CharField(max_length=255)
    year = models.IntegerField()
    album = models.CharField(max_length=255)
```

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

Step7: Register Model in admin.py

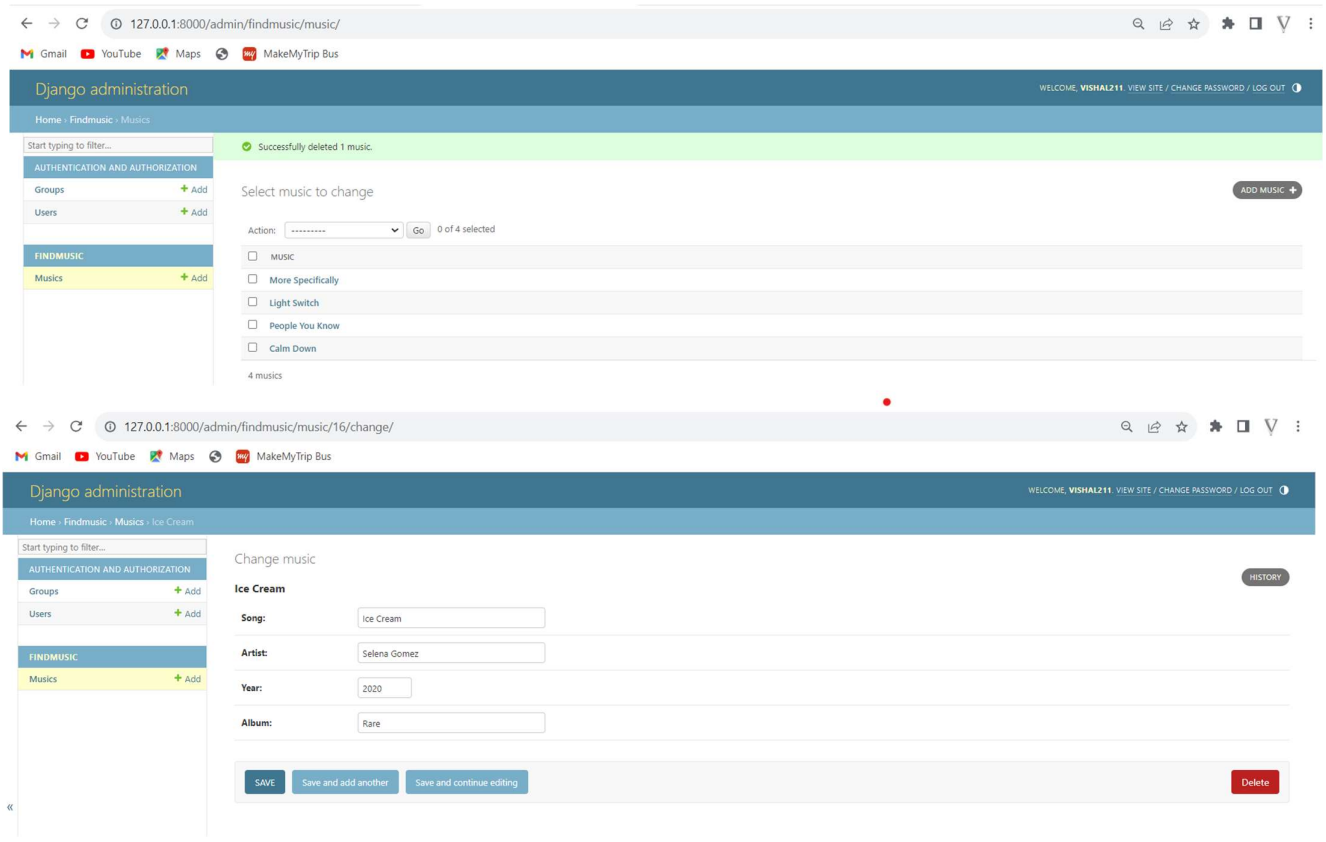
```
from django.contrib import admin
from .models import Music

admin.site.register(Music)
```

Step8: python manage.py makemigrations

Step9: python manage.py migrate

Step10: python manage.py createsuperuser

Output:



Require search music and show list of music

Step 11: make musicfind.html in templates in findmusic app

Findmusic>templates>findmusic>musicfind.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Music Finder</title>
</head>
<body>
    <form method="POST" action="">
```

```
        {% csrf_token %}
        <h1>Welcome</h1>
        <h2>Let's search your music!</h2>
        <label for="song">Song</label>
        <input id="song" class="input" type="text" name="song" placeholder="">
        <label for="artist">Artist</label>
        <input id="artist" type="text" name="artist" placeholder="">
        <label for="year">Year</label>
        <input id="year" type="text" name="year" placeholder="">
        <label for="album">Album</label>
        <input id="album" type="text" name="album" placeholder="">

        <button type="submit">Submit</button>
    </form>
    <table>
        <tr>
            <th>Song</th>
            <th>Artist</th>
            <th>Year</th>
            <th>Album</th>
        </tr>
        {% for music in music_records %}
        <tr>
            <td>{{ music.song }}</td>
            <td>{{ music.artist }}</td>
            <td>{{ music.year }}</td>
            <td>{{ music.album }}</td>
        </tr>
        {% endfor %}
    </table>
</body>
</html>
```

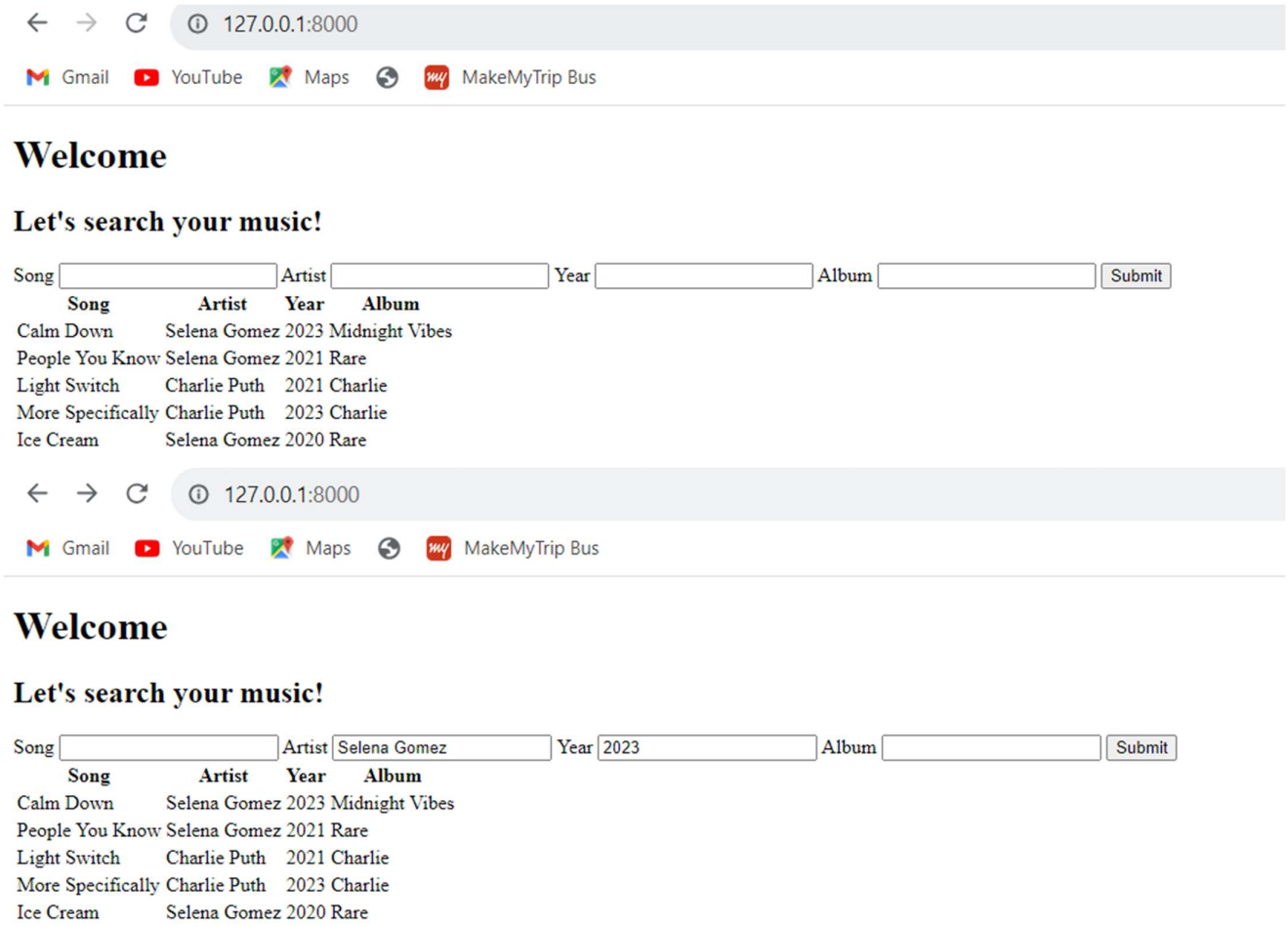
Step 12: write views.py

```
from django.shortcuts import render
from .models import Music
def home(request):
    music_records = Music.objects.all()
    # Apply filters if the form is submitted
    if request.method == "POST":
        song = request.POST.get("song")
        artist = request.POST.get("artist")
        year = request.POST.get("year")
        album = request.POST.get("album")
        if song:
            music_records = music_records.filter(song__icontains=song)
        if artist:
```

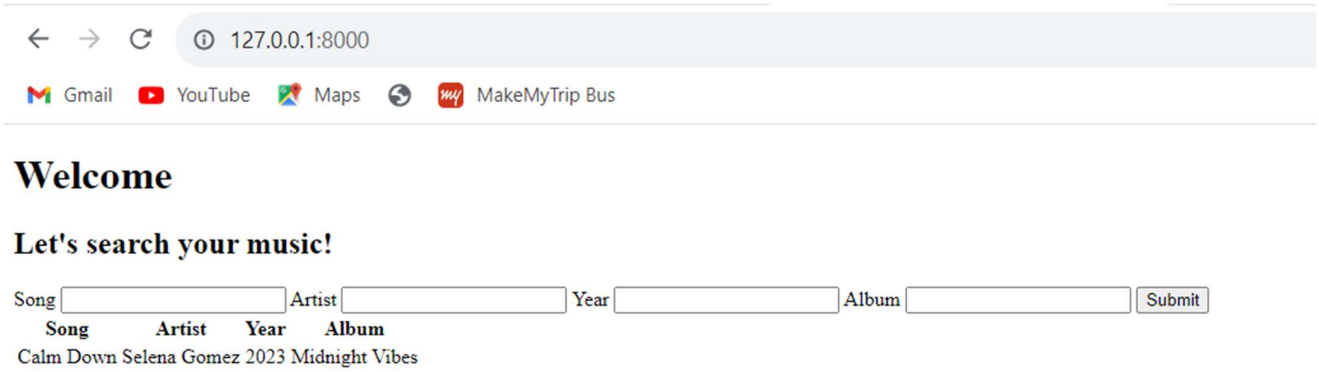
```
            music_records = music_records.filter(artist__icontains=artist)
        if year:
            music_records = music_records.filter(year__icontains=year)
        if album:
            music_records = music_records.filter(album__icontains=album)

    return render(request, 'findmusic/musicfind.html', {'music_records': music_records})
```

Step 13: python manage.py runserver

Output:



After that press submit



You also use get method