

Django Lab Manual

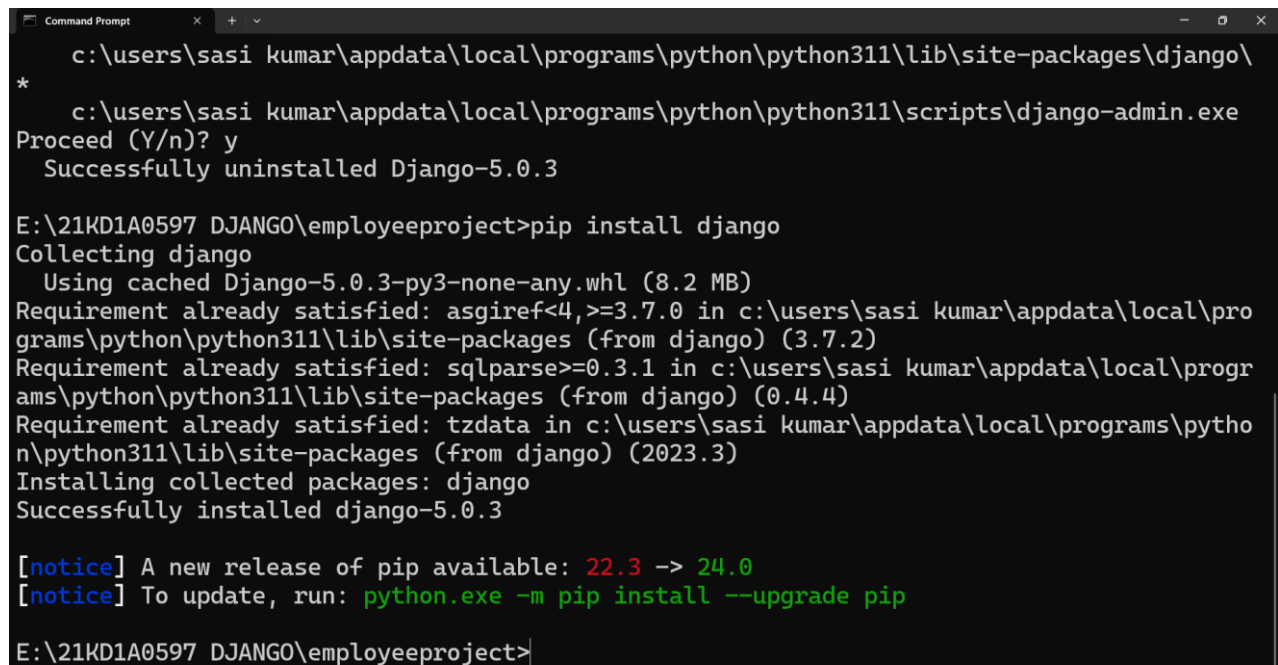
MODULE-1

1) Create Django environment setup and installation in windows/Linux

Method -1: Using Python & pip

1. First go to search bar and search CMD
2. Move to E drive by using command **>E:**
3. Then we get E:\>
4. Create folder with your rollno(21KD1A0597 DJANGO) in E drive by using the command:
>mkdir 21KD1A0597 DJANGO and enter into the folder using command:
>cd 21KD1A0597 DJANGO
5. Output: E:\ 21KD1A0597 DJANGO >
6. Installation of django : In the directory E:\ 21KD1A0597 DJANGO give the following command to install django

E:\ 21KD1A0597 DJANGO > pip install Django



```
Command Prompt
c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages\django\
*
c:\users\sasi kumar\appdata\local\programs\python\python311\scripts\django-admin.exe
Proceed (Y/n)? y
Successfully uninstalled Django-5.0.3

E:\21KD1A0597 DJANGO\employeeproject>pip install django
Collecting django
  Using cached Django-5.0.3-py3-none-any.whl (8.2 MB)
Requirement already satisfied: asgiref<4,>=3.7.0 in c:\users\sasi kumar\appdata\local\pro
grams\python\python311\lib\site-packages (from django) (3.7.2)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\sasi kumar\appdata\local\progr
ams\python\python311\lib\site-packages (from django) (0.4.4)
Requirement already satisfied: tzdata in c:\users\sasi kumar\appdata\local\programs\pytho
n\python311\lib\site-packages (from django) (2023.3)
Installing collected packages: django
Successfully installed django-5.0.3

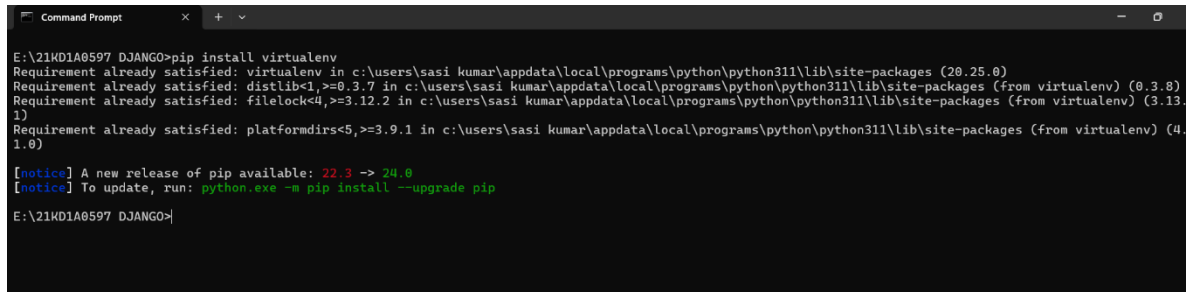
[notice] A new release of pip available: 22.3 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

E:\21KD1A0597 DJANGO\employeeproject>
```

Method -2: Using Virtual Environment

1: installing virtual environment in a particular folder by giving the command as:

Pip install virtualenv



```

E:\21KD1A0597 DJANGO>pip install virtualenv
Requirement already satisfied: virtualenv in c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages (20.25.0)
Requirement already satisfied: distlib<1,>=0.3.7 in c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages (from virtualenv) (0.3.8)
Requirement already satisfied: filelock<4,>=3.12.2 in c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages (from virtualenv) (3.13.1)
Requirement already satisfied: platformdirs<5,>=3.9.1 in c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages (from virtualenv) (4.1.0)

[notice] A new release of pip available: 22.3 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
E:\21KD1A0597 DJANGO>

```

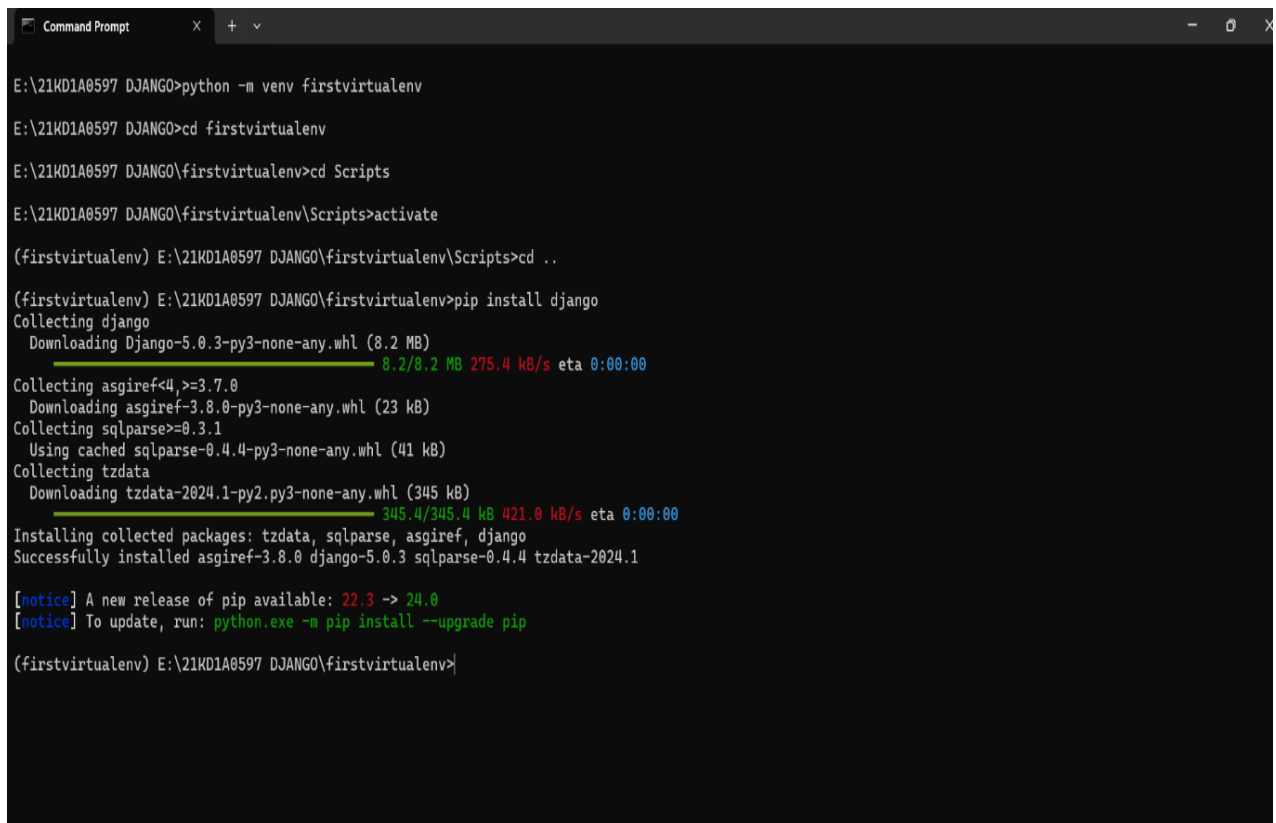
2: After installing the virtual environment we should give name to our virtual environment by giving command as:

python -m venv “virtualenvironment name”

E:\21KD1A0597 DJANGO>python -m venv firstvirtualenv

3: After creating virtual environment then install the Django in the virtual environment by command as:

Pip install django



```

E:\21KD1A0597 DJANGO>python -m venv firstvirtualenv
E:\21KD1A0597 DJANGO>cd firstvirtualenv
E:\21KD1A0597 DJANGO\firstvirtualenv>cd Scripts
E:\21KD1A0597 DJANGO\firstvirtualenv\Scripts>activate
(firstvirtualenv) E:\21KD1A0597 DJANGO\firstvirtualenv\Scripts>cd ..
(firstvirtualenv) E:\21KD1A0597 DJANGO\firstvirtualenv>pip install django
Collecting django
  Downloading Django-5.0.3-py3-none-any.whl (8.2 MB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.2/8.2 MB 275.4 kB/s eta 0:00:00
Collecting asgiref<4,>=3.7.0
  Downloading asgiref-3.8.0-py3-none-any.whl (23 kB)
Collecting sqlparse>=0.3.1
  Using cached sqlparse-0.4.4-py3-none-any.whl (41 kB)
Collecting tzdata
  Downloading tzdata-2024.1-py2.py3-none-any.whl (345 kB)
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 345.4/345.4 kB 421.0 kB/s eta 0:00:00
Installing collected packages: tzdata, sqlparse, asgiref, django
Successfully installed asgiref-3.8.0 django-5.0.3 sqlparse-0.4.4 tzdata-2024.1

[notice] A new release of pip available: 22.3 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
(firstvirtualenv) E:\21KD1A0597 DJANGO\firstvirtualenv>

```

2) Create DJANGO project and app structure with django-admin commands

To know about commands of django use the command: **django-admin**

E:\21KD1A0597 DJANGO>django-admin

To know commands of django , use cmd : **>django-admin**

Output : **C:\ DjangoProjects >django-admin**

If django-admin is not working, uninstall python software and install latest version of the python.

Type 'django-admin help <subcommand>' for help on a specific subcommand.

Available subcommands: [django]

- Check
- createcachetable
- diffsettings
- flush
- loaddata
- makemigrations
- optimizemigration
- sendtestemail
- showmigrations
- sqlmigrate
- squashmigrations
- startproject
- testserver
- compilemessages
- dbshell
- dumpdata
- inspectdb
- makemessages
- migrate
- runserver
- shell
- sqlflush
- sqlsequencereset
- startapp
- test

These are the commands available in django.

Note that only Django core commands are listed as settings are not properly configured (error: Requested setting `INSTALLED_APPS`, but settings are not configured. You must either define the environment variable `DJANGO_SETTINGS_MODULE` or call `settings.configure()` before accessing settings.).

STEPS TO CREATE A PROJECT IN DJANGO:

1:To create a project give the following command:

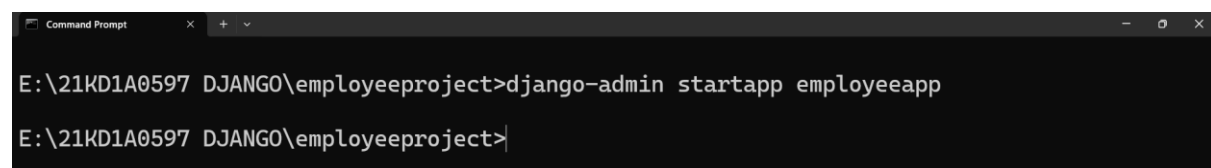
E:\21KD1A0597 DJANGO>django-admin startproject employeeproject

2.Change directory:

E:\21KD1A0597 DJANGO>cd employeeproject

3.To create app:

E:\21KD1A0597 DJANGO\employeeproject>django-admin startapp employeeapp



```

Command Prompt
E:\21KD1A0597 DJANGO\employeeproject>django-admin startapp employeeapp
E:\21KD1A0597 DJANGO\employeeproject>
  
```

3. Deployment of project to the server

1. To see the response, run the development server using the following command: **python manage.py runserver**.

E:\21KD1A0597 DJANGO\employeeproject>python manage.py runserver

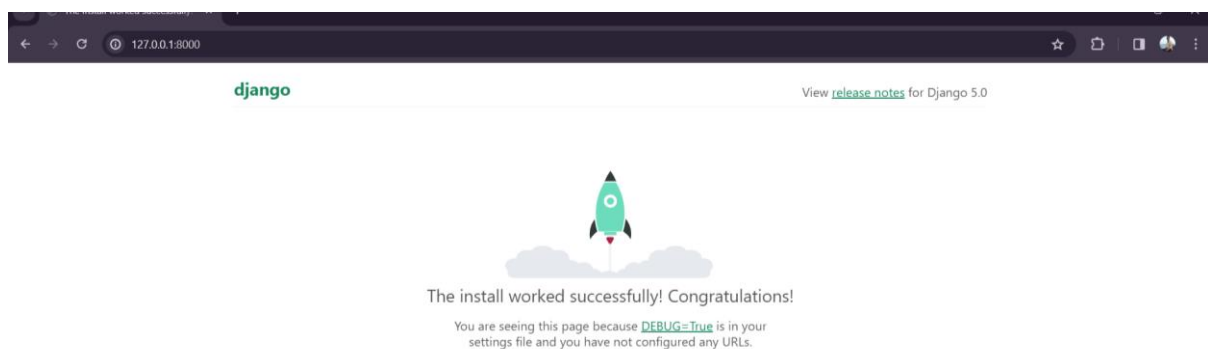
```
Command Prompt - python r x + v
E:\21KD1A0597 DJANGO\employeeproject>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
March 24, 2024 - 12:43:30
Django version 5.0.3, using settings 'employeeproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[24/Mar/2024 12:46:36] "GET / HTTP/1.1" 200 10629
```

2. Then, open your browser and visit with the URL (**http://127.0.0.1:8000/**) to see the response.



4) Implement a simple HTTP response using Django.

In Django , the `HttpResponse` class is used to construct an HTTP response. It takes one required argument , which is the content of the response. The content can be a string , a file-like object, or a callable that returns either of those things.

Here's an example of how you can create a simple response using Django.

1.Set up your Django project and app:

a)Create a new Django project: **`django-admin startproject employeeproject`**

b)Create a new Django app within the project: **`django-admin startapp employeeapp`**

2.Open the "**views.py**" file and add the following code:

```
from django.shortcuts import render
from django.http import HttpResponse
from django.template import loader
def root(request):
return HttpResponse("<h1 style='color:red'>Hello world!</h1>")
```

3.Configure the URL:

In your app's directory (**employeeapp**), create a new file called "**urls.py**" if it doesn't exist.

In "**urls.py**" file, add the following code:

```
from django.urls import path
from . import views
urlpatterns = [
path('', views.root, name='root'),
]
```

4.Configure the project's main URL:

In your project's "**urls.py**" file, add the following code:

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
path('admin/', admin.site.urls),
path('', include('employeeapp.urls')),
]
```

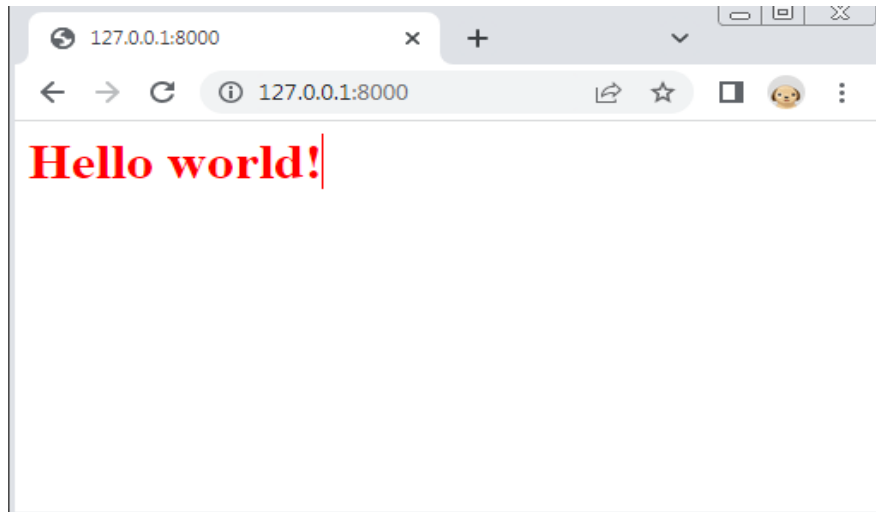
5.Add the following line under `INSTALLED_APPS` in "**settings.py**" file
'employeeapp'

6.You've created a `HttpResponse` using Django.

To see the response, run the development server using the following command: **`python manage.py runserver`**.

Command for deployment:

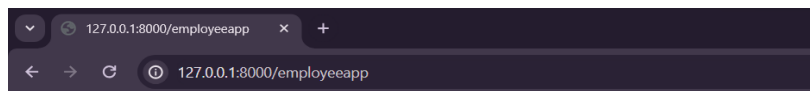
```
E:\21KD1A0597 DJANGO\employeeproject>python manage.py runserver
```



7. Open the "urls.py" file of your app's directory (**myapp**) and add the following code:

```
#path("", views.root, name='root'),  
path('employeeapp', views.root, name='root'),
```

Now, open your browser and visit the URL **<http://localhost:8000/myapp>** to see the response.



MODULE-2

5) Implement template inheritance with views and images.

Template inheritance in Django is a powerful feature that allows you to create reusable templates that can be extended by child templates. This can be very useful for creating consistent layouts across your website, as well as for avoiding code duplication.

To use template inheritance, you first need to create a base template. This template will contain all of the common elements of your website, such as header, footer and navigation bar.

Here's an example of how you can create a simple web page using Django templates and images.

1. Set up your Django project and app:

a) Create a new Django project: **django-admin startproject imageproject**

b) Create a new Django app within the project: **django-admin startapp imageapp**

2. Configure static files:

In your project's "**settings.py**", add the following code:

```
import os
BASE_DIR=Path(__file__).resolve().parent.parent
STATIC_URL = '/static/'
MEDIA_URL='/images/'
STATICFILES_DIRS=[
    os.path.join(BASE_DIR,'static')
]
```

In your project's "**settings.py**", add '**imageapp**' in installed apps

3. Create the HTML template:

In your app's directory (**imageapp**), create a new directory called "**templates**" if it doesn't exist.

Inside the **templates** directory, create a new HTML file called "**index.html**" with the following content:

```

<!DOCTYPE html>
{% load static %}
<html>
<head>
<title>My Web Page</title>
</head>
<body> <h1>Welcome to my web page!</h1>
<p>This is a simple web page created using Django templates and images.</p>

</body>
</html>

```

4. Create static files:

In your app's directory (**myapp**), create a new directory called "**static**" if it doesn't exist.

Inside the "**static**" directory, create a subdirectory: "**images**".

Place your image file (**3.jpg**) inside the "**images**" directory.

5. Create a view in "**views.py**":

In your app's directory (**myapp**), open "**views.py**" and add the following code

```

from django.shortcuts import render

def index(request):
    return render(request, 'index.html')

```

6. Configure the URL: In project **urls.py** add the following code:

```

from django.contrib import admin
from django.urls import path
from imageapp.views import *
from django.conf.urls.static import static
from django.conf import settings

urlpatterns = [
    path("", index, name="index"),
]

```

7 .You've created a simple web page using Django templates and static files. To see the page in action, run the development server using the following command:

>python manage.py runserver.

welcome to the webpage.



6) Create a simple web page using django templates and static files.

Here's an example of how you can create a simple web page using Django templates and static files.

1. Set up your Django project and app:

Create a new Django project: **django-admin startproject myproject**

Create a new Django app within the project: **django-admin startapp myapp**

2. Configure static files:

In your project's settings.py, add the following code:

```
from pathlib import Path

STATIC_URL = '/static/'
```

In your project's "settings.py", add 'myapp' in installed apps

3. Create the HTML template:

In your app's directory (myapp), create a new directory called "templates" if it doesn't exist.

Inside the templates directory, create a new HTML file called "index.html" with the following content:

```
<!DOCTYPE html>
{% load static %}
<html>
<head>
    <title>My Web Page</title>
    <link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}">
</head>
<body>
    <h1>Welcome to my web page!</h1>
    <p>This is a simple web page created using Django templates and static files.</p>
    
</body>
</html>
```

5. Create static files:

In your app's directory (myapp), create a new directory called "static" if it doesn't exist.

Inside the static directory, create two subdirectories: "css" and "images".

Place your CSS file (style.css) inside the "css" directory and your image file (myimage.jpg) inside the "images" directory.

6. Create a view in views.py:

In your app's directory (myapp), open views.py and add the following code

```
from django.shortcuts import render
def index(request):
    return render(request, 'index.html')
```

7. Configure the URL:

In your app's directory (myapp), create a new file called "**urls.py**" if it doesn't exist. Inside "**urls.py**", add the following code:

```
from django.urls import path
from . import views
urlpatterns = [
    path("", views.index, name='index'),
]
```

8. Configure the project's main URL:

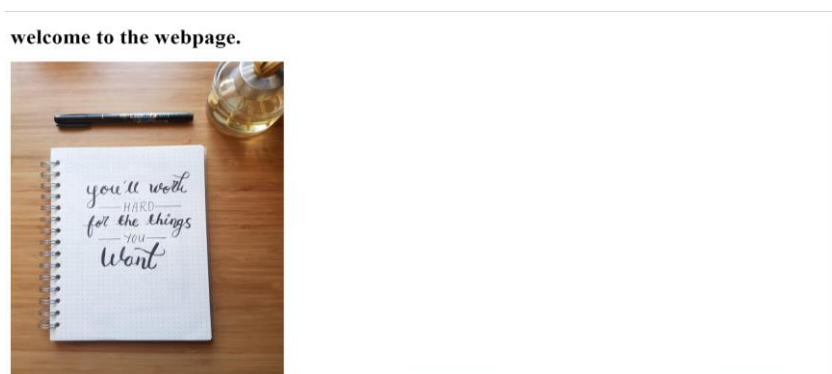
In your project's "**urls.py**" file, add the following code:

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('myapp.urls')),
]
```

9. You've created a simple web page using Django templates and static files.

To see the page in action, run the development server using the following command: **python manage.py runserver**.

Then, open your browser and visit <http://localhost:8000> to see the web page.



7) Create a django web page to render templates to multiple routes.

To render a template to multiple routes in Django, you can use the `include()` function. The `include()` function takes a URL pattern as its argument and returns a callable that can be used to render the template.

Here's an example of how you can create a Django web page that renders templates for multiple routes.

1. Set up your Django project and app:

Create a new Django project: **`django-admin startproject myproject`**

Create a new Django app within the project: **`python manage.py startapp myapp`**

2. Create the HTML templates:

In your app's directory (myapp), create a new directory called "templates" if it doesn't exist. Inside the templates directory, create a new HTML file called "home.html" with the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>Home</title>
</head>
<body>
  <h1>Welcome to the Home Page!</h1>
</body>
</html>
```

3. Create another HTML file called "about.html" with the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>About</title>
</head>
<body>
  <h1>About Us</h1>
  <p>We are a company that provides amazing services.</p>
</body>
</html>
```

4. Create another HTML file called "productst.html" with the following content:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Products</title>
  </head>
  <body>
    <h1>Our Products</h1>
    <ul>
      <li>Mobile Phones</li>
      <li>Beverages</li>
      <li>Cosmetics</li>
      <li>Electronics</li>
    </ul>
  </body>
</html>
```

5. Create views in views.py:

In your app's directory (myapp), open views.py and add the following code:

```
from django.shortcuts import render
def home(request):
    return render(request, 'home.html')
def about(request):
    return render(request, 'about.html')
def products(request):
    return render(request, 'products.html')
```

6. Configure the URLs:

In your app's directory (myapp), create a new file called urls.py if it doesn't exist.

Inside urls.py, add the following code:

```
from django.urls import path
from . import views
urlpatterns = [
    path('home/', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('products/', views.products, name='products'),
]
```

7. Configure the project's main URL:

In your project's urls.py file, add the following code:

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

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

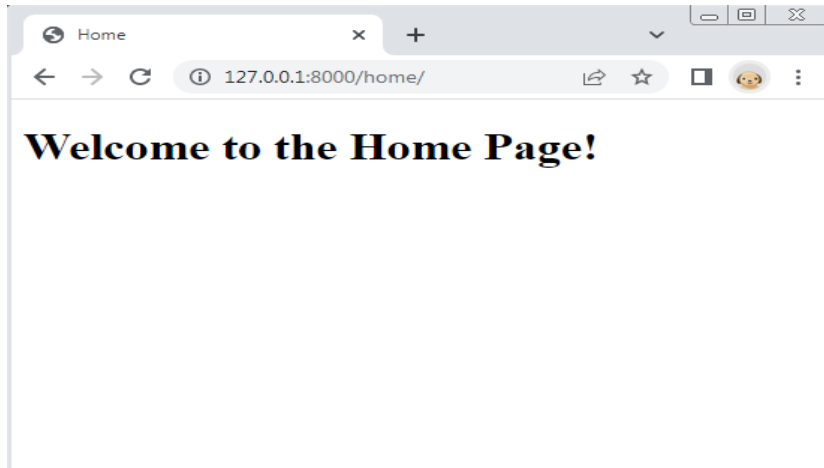
You've created a Django web page that renders templates for multiple routes.

8. In your project's "settings.py", add 'myapp' in installed apps

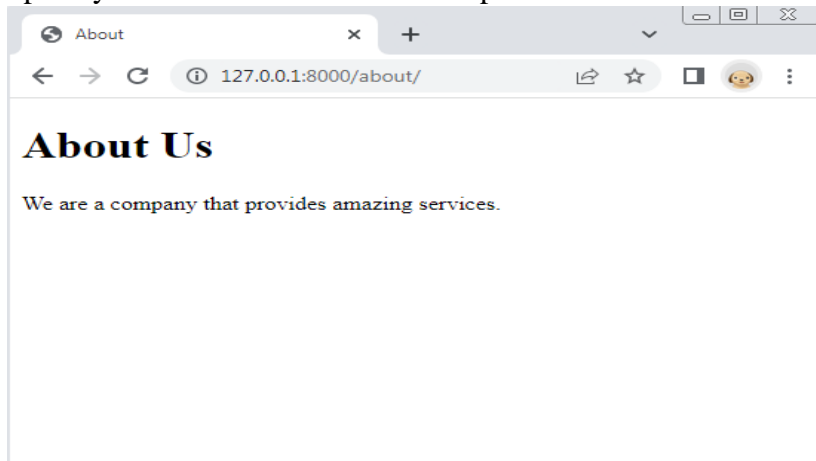
9. To see the pages in action, run the development server using the following command:

python manage.py runserver.

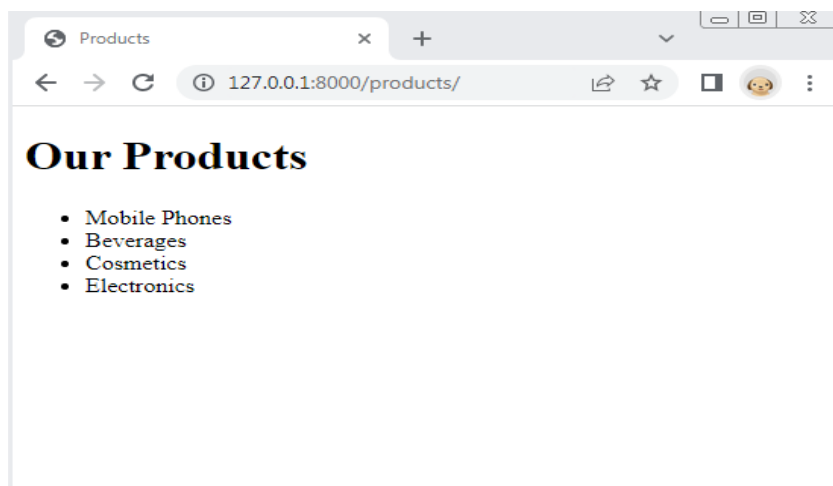
open your browser and visit <http://127.0.0.1:8000/home/> for the home page



open your browser and visit [http:// 127.0.0.1:8000/about/](http://127.0.0.1:8000/about/) for the about page.



open your browser and visit [http:// 127.0.0.1:8000/products/](http://127.0.0.1:8000/products/) for the products page.



MODULE-3

8) Create a Django model named **customer** having fields **name**, **age**, **phone number**, **address** and print the customer details in web page.

SQLite Database:

When we created the Django project, we got an empty SQLite database.

It was created in the root folder, and has the filename db.sqlite3.

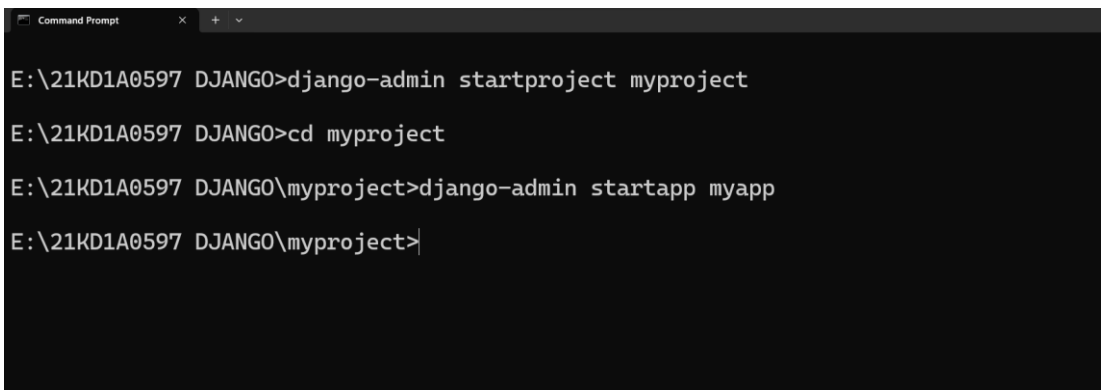
By default, all Models created in the Django project will be created as tables in this database.

Here's an example of how you can create a Django model named **Customer** with fields **name**, **age**, **phone_number**, and **address**, and then print the customer details on a web page.

1. Set up your Django project and app:

Create a new Django project: **django-admin startproject myproject**

Create a new Django app within the project: **python manage.py startapp myapp**



```

E:\21KD1A0597 DJANGO>django-admin startproject myproject

E:\21KD1A0597 DJANGO>cd myproject

E:\21KD1A0597 DJANGO\myproject>django-admin startapp myapp

E:\21KD1A0597 DJANGO\myproject>
  
```

2. Define the **Customer** model:

In your app's directory (myapp), open models.py and add the following code

```

from django.db import models
class Customer(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    phone_number = models.CharField(max_length=15)
    address = models.CharField(max_length=200)
    def __str__(self):
        return self.name
  
```

3. In your app's directory(myapp), open admin.py and add the following code:

```

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

4. Create a view in views.py to retrieve customer details:

In your app's directory (myapp), open views.py and add the following code:

```
from django.shortcuts import render
from .models import Customer
def customer_details(request):
    customers = Customer.objects.all()
    return render(request, 'customer_details.html', {'customers': customers})
```

5. Create a template to display customer details:

In your app's directory (myapp), create a new directory called "templates" if it doesn't exist. Inside the templates directory, create a new HTML file called "customer_details.html" with the following content

```
<!DOCTYPEhtml>
<html>
<head>
  <title>Customer Details</title>
</head>
<body>
  <h1>Customer Details</h1>
  <table>
    <thead>
      <tr>
        <th>Name</th>
        <th>Age</th>
        <th>Phone Number</th>
        <th>Address</th>
      </tr>
    </thead>
    <tbody>
      {% for customer in customers %}
      <tr>
        <td>{{ customer.name }}</td>
        <td>{{ customer.age }}</td>
        <td>{{ customer.phone_number }}</td>
        <td>{{ customer.address }}</td>
      </tr>
      {% endfor %}
    </tbody>
  </table>
</body>
</html>
```

6. Configure the URLs:

In your app's directory (myapp), open urls.py or create a new file called urls.py if it doesn't exist. Inside urls.py, add the following code:

```
from django.urls import path
from . import views
urlpatterns = [
    path('customer-details/', views.customer_details, name='customer_details'), ]
```

7. Configure the project's main URL:

In your project's `urls.py` file, add the following code:

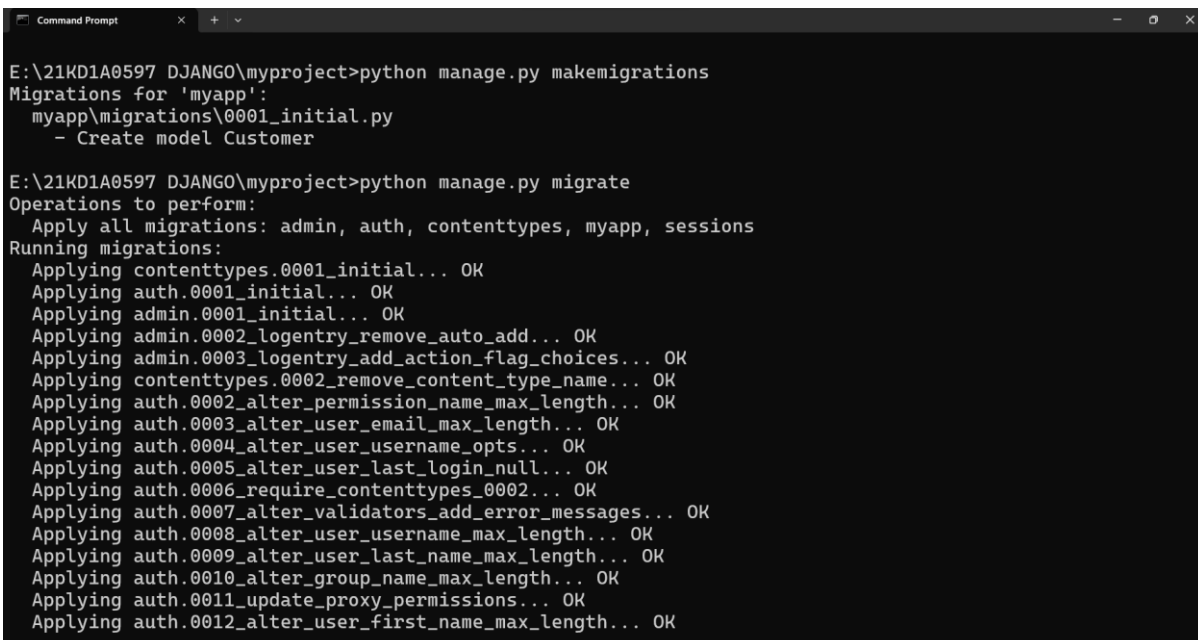
```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('myapp.urls')),
]
```

8. In your project's `settings.py`, add the following code:

```
import os
```

In your project's "`settings.py`", add `'myapp'` in installed apps

9. After defining the model, run the following command to create the necessary database tables: **`python manage.py makemigrations`** and then **`python manage.py migrate`**



```
E:\21KD1A0597 DJANGO\myproject>python manage.py makemigrations
Migrations for 'myapp':
  myapp\migrations\0001_initial.py
    - Create model Customer

E:\21KD1A0597 DJANGO\myproject>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, myapp, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
```

We created a Django model named **Customer** with fields **name**, **age**, **phone_number**, and **address**.

To add customer data, you can use the Django admin interface or create a form to handle.

10. Create admin user

run the following command in cmd

`python manage.py createsuperuser`

Then it will ask details like 'username', 'email', 'password', & 'confirm password'


```

Command Prompt

E:\21KD1A0597 DJANGO\myproject>python manage.py createsuperuser
Username (leave blank to use 'sasikumar'): admin
Email address: admin@gmail.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.

E:\21KD1A0597 DJANGO\myproject>

```

11. Once the user is created run server. In command prompt run the following command for running the server.

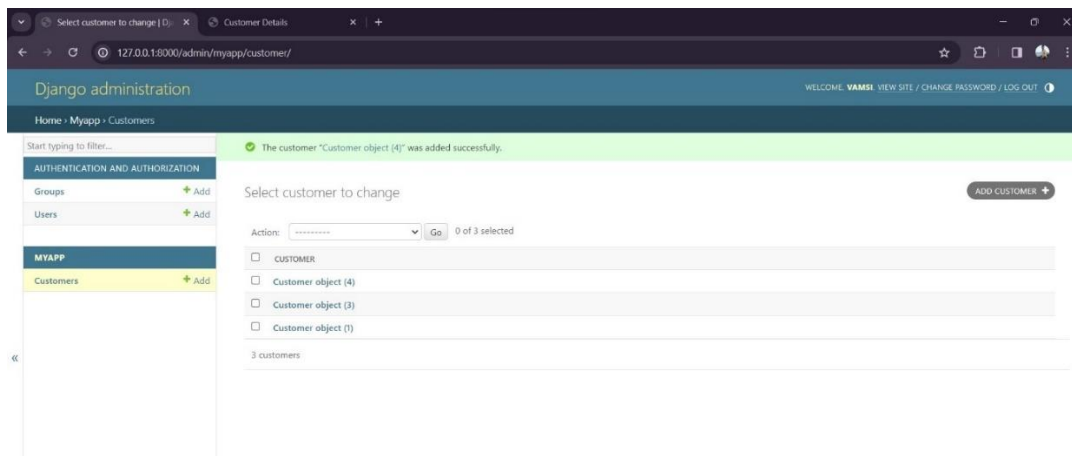
python manage.py runserver

12. Open the browser and type the URL

<http://127.0.0.1:8000/admin/>

Login using above admin user credentials.

After logging in we are directed to site administration.



The order details of the customer will be displayed on a web page when you visit <http://localhost:8000/customer-orders/1/>



Customer Details

Name	Age	Phone Number	Address
vamsi	20	9347164792	vzm
sasi	19	9347164792	svt
ojaswini	20	7993548574	rjy

9) Create a customer and order models and map them using one to many relationships. Print all the orders made by customer in a web page.

Here's an example of how you can create two Django models named **Customer** and **Order** and establish a one-to-many relationship between them. You can then print all the orders made by a customer on a web page.

1. Set up your Django project and app:

Create a new Django project: **django-admin startproject myproject**

Create a new Django app within the project: **python manage.py startapp myapp**

2. Define the **Customer** and **Order** models:

In your app's directory (myapp), open models.py and add the following code:

```
from django.db import models
class Customer(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    phone_number = models.CharField(max_length=15)
    address = models.CharField(max_length=200)
    def __str__(self):
        return self.name
class Order(models.Model):
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    order_number = models.CharField(max_length=20)
    product = models.CharField(max_length=100)
    quantity = models.IntegerField()
    price = models.CharField(max_length=10)
    def __str__(self):
        return self.order_number
```

3. In your app's directory (myapp), open admin.py and add the following code:

```
from django.contrib import admin
from .models import Customer
from .models import Order
admin.site.register(Customer)
admin.site.register(Order)
```

4. Create a view in views.py to retrieve customer orders:

In your app's directory (myapp), open views.py and add the following code:

```
from django.shortcuts import render
from .models import Customer
def customer_orders(request, customer_id):
    customer = Customer.objects.get(id=customer_id)
    orders = customer.order_set.all()
    return render(request, 'customer_orders.html', {'customer': customer, 'orders': orders})
```

5. Create a template to display customer orders:

In your app's directory (myapp), create a new directory called "templates" if it doesn't exist.

Inside the templates directory, create a new HTML file called "customer_orders.html" with the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>Customer Orders</title>
</head>
<body>
  <h1>Orders for {{ customer.name }}</h1>
  <table>
    <thead>
      <tr>
        <th>Order Number</th>
        <th>Product</th>
        <th>Quantity</th>
      </tr>
    </thead>
    <tbody>
      {% for order in orders %}
      <tr>
        <td>{{ order.order_number }}</td>
        <td>{{ order.product }}</td>
        <td>{{ order.quantity }}</td>
      </tr>
      {% endfor %}
    </tbody>
  </table>
</body>
</html>
```

6. Configure the URLs:

In your app's directory (myapp), open urls.py or create a new file called urls.py if it doesn't exist.

Inside urls.py, add the following code:

```
from django.urls import path
from . import views
urlpatterns = [
    path('customer-details/', views.customer_details, name='customer_details'),
    path('customer-orders/<int:customer_id>', views.customer_orders, name='customer_orders'),
]
```

7. Configure the project's main URL:

In your project's urls.py file, add the following code:

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]
```

8. Add the following in 'settings.py' file

```
import os
```

In your project's "settings.py", add 'myapp' in installed apps

9. After defining the models, run the following command to create the necessary database tables:

```
python manage.py makemigrations
```

```
python manage.py migrate
```

10. Once the user is created run server. In command prompt run the following command for running the server.

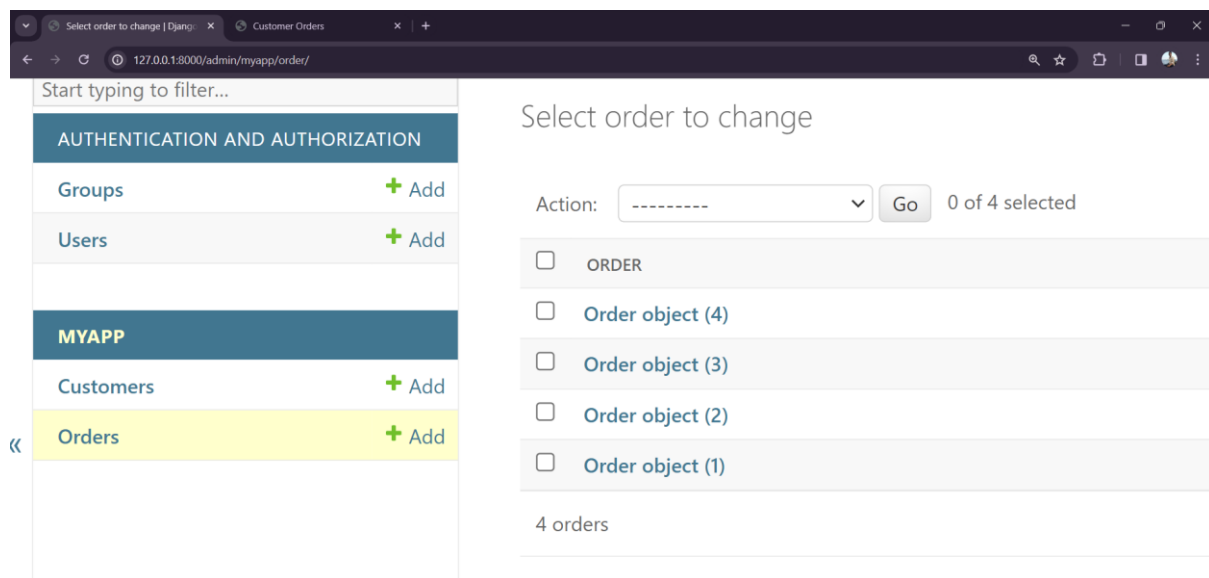
```
python manage.py runserver
```

12. Open the browser and type the URL

<http://127.0.0.1:8000/admin/>

Login using above admin user credentials.

After logging in we are directed to site administration.



The order details of the customer will be displayed on a web page when you visit <http://localhost:8000/customer-orders/1/>



Orders for vamsi

Order Number	Product	Quantity
564	pen	12
140	blending stamp	10

MODULE-4

10) Create a registration form to store details of a customer into database.

1. Set up your Django project and app:

Create a new Django project: **django-admin startproject myproject**

Create a new Django app within the project: **django-admin startapp myapp1**

2. Define the **Customer** model:

In your app's directory (myapp1), open models.py and add the following code:

```
from django.db import models
class Customer(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    phone_number = models.CharField(max_length=15)
    address = models.CharField(max_length=200)
    username=models.CharField(max_length=100)
    password=models.CharField(max_length=100)
def __str__(self):
    return self.name
```

3. Create a registration form in forms.py:

In your app's directory (myapp1), open forms.py and add the following code:

```
from django import forms
from .models import Customer
class CustomerForm(forms.ModelForm):
    class Meta:
        model = Customer
        fields = ['name', 'age', 'phone_number', 'address', 'username', 'password']
```

4. Create a view in views.py to handle the registration form:

In your app's directory (myapp1), open views.py and add the following code:

```
from django.shortcuts import render, redirect
from .forms import CustomerForm
def register_customer(request):
    if request.method == 'POST':
        form = CustomerForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('success')
    else:
        form = CustomerForm()
        return render(request, 'register_customer.html', {'form': form})
def registration_success(request):
    return render(request, 'registration_success.html')
```

5. Create templates for the registration form and success message:

In your app's directory (myapp1), create a new directory called "templates" if it doesn't exist. Inside the templates directory, create a new HTML file called "register_customer.html" with the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>Customer Registration</title>
</head>
<body>
  <h1>Customer Registration</h1>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <button type="submit">Register</button>
  </form>
</body>
</html>
```

Create another HTML file called "registration_success.html" with the following content::

```
<!DOCTYPE html>
<html>
<head>
<title>Registration Success</title>
</head>
<body>
<h1>Registration Successful</h1>
<p>Thank you for registering as a customer.</p>
</body>
</html>
```

6. Configure the URLs:

In your app's directory (myapp1), open urls.py or create a new file called urls.py if it doesn't exist. Inside urls.py, add the following code:

```
from django.urls import path
from . import views
urlpatterns = [
    path('register-customer/', views.register_customer, name='register_customer'),
    path('success/', views.registration_success, name='success'),
]
```

7. Configure the project's main URL: In your project's urls.py file, add the following code:

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('myapp.urls')), ]
```

8. Add the following in 'settings.py' file

```
import os
```

In your project's "settings.py", add 'myapp1' in installed apps

9. In your app's directory(myapp), open admin.py and add the following code:

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

10. After defining the models, run the following command to create the necessary database tables:

python manage.py makemigrations

python manage.py migrate

```

Command Prompt
E:\21KD1A0597 DJANGO\myproject>python manage.py makemigrations
Migrations for 'myapp1':
  myapp1\migrations\0001_initial.py
    - Create model Customer

E:\21KD1A0597 DJANGO\myproject>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, myapp1, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK

```

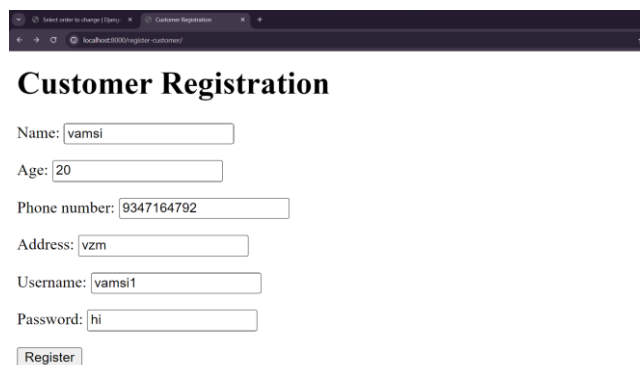
11. Once the user is created run server. In command prompt run the following command for running the server.

python manage.py runserver

12. Open the browser and type the URL

<http://127.0.0.1:8000/register-customer/>

Then the Customer Registration form will open as below:



Customer Registration

Name:

Age:

Phone number:

Address:

Username:

Password:

On clicking on Register the customer is successfully registered page open as below:



Registration Successful

Thank you for registering as a customer.

11) Create a login page for the customers who have registered in the database

1. Set up your Django project and app:

Create a new Django project: **django-admin startproject myproject**

Create a new Django app within the project: **python manage.py startapp myapp**

2. Define the **Customer** model:

In your app's directory (myapp), open models.py and add the following code:

```
from django.db import models
class Customer(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    phone_number = models.CharField(max_length=15)
    address = models.CharField(max_length=200)
    username = models.CharField(max_length=100, unique=True)
    password = models.CharField(max_length=100)
    def __str__(self):
        return self.name
```

3. Create a login form in forms.py:

In your app's directory (myapp), open forms.py and add the following code:

```
from django import forms
class LoginForm(forms.Form):
    username = forms.CharField(max_length=100)
    password = forms.CharField(max_length=100, widget=forms.PasswordInput)
```

4. Create a view in views.py to handle the login page:

In your app's directory (myapp), open views.py and add the following code:

```
from django.shortcuts import render, redirect
from .forms import LoginForm
from .models import Customer
def login(request):
    if request.method == 'POST':
        form = LoginForm(request.POST)
        if form.is_valid():
            username = form.cleaned_data['username']
            password = form.cleaned_data['password']
            try:
                customer = Customer.objects.get(username=username, password=password)
                # Perform the necessary login actions here, e.g., setting session variables
                return redirect('dashboard') # Replace 'dashboard' with your actual dashboard URL name
            except Customer.DoesNotExist:
                form.add_error(None, 'Invalid username or password')
        else:
            form = LoginForm()
    return render(request, 'login.html', {'form': form})
def dashboard(request):
    return render(request, 'dashboard.html')
```


5. Create a template for the login page:

In your app's directory (myapp), create a new directory called "templates" if it doesn't exist. Inside the templates directory, create a new HTML file called "login.html" with the following content:

```
<!DOCTYPE html>
<html>
<head>
  <title>Login</title>
</head>
<body>
  <h1>Login</h1>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    {% if form.errors %}
      <p>{{ form.errors }}</p>
    {% endif %}
    <button type="submit">Login</button>
  </form>
</body>
</html>
```

Create another HTML file called "dashboard.html" with the following content::

```
<!DOCTYPE html>
<html>
<head>
  <title>Customer DashBoard</title>
</head>
<body>
  <h1>Login Successful</h1>
  <p>Customer Dashboard</p>
</body>
</html>
```

6. Configure the URLs:

In your app's directory (myapp), open urls.py or create a new file called urls.py if it doesn't exist.

Inside urls.py, add the following code:

```
from django.urls import path
from . import views
urlpatterns = [
    path('login/', views.login, name='login'),
    path('dashboard/', views.dashboard, name='dashboard'),
]
```

7. Configure the project's main URL:

In your project's `urls.py` file, add the following code:

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('myapp.urls')),
]
```

8. Add the following in 'settings.py' file

```
import os
```

In your project's "`settings.py`", add '`myapp`' in installed apps

9. In your app's directory(myapp), open `admin.py` and add the following code:

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

10. After defining the models, run the following command to create the necessary database tables:

`python manage.py makemigrations`

`python manage.py migrate`

11. Once the user is created run server. In command prompt run the following command for running the server.

`python manage.py runserver`

12. Open the browser and type the URL

<http://127.0.0.1:8000/login/>

Then the Customer login form will open as below:



Login

Username:

Password:

Login

On clicking on login the customer is successfully registered page open as below



login Successful

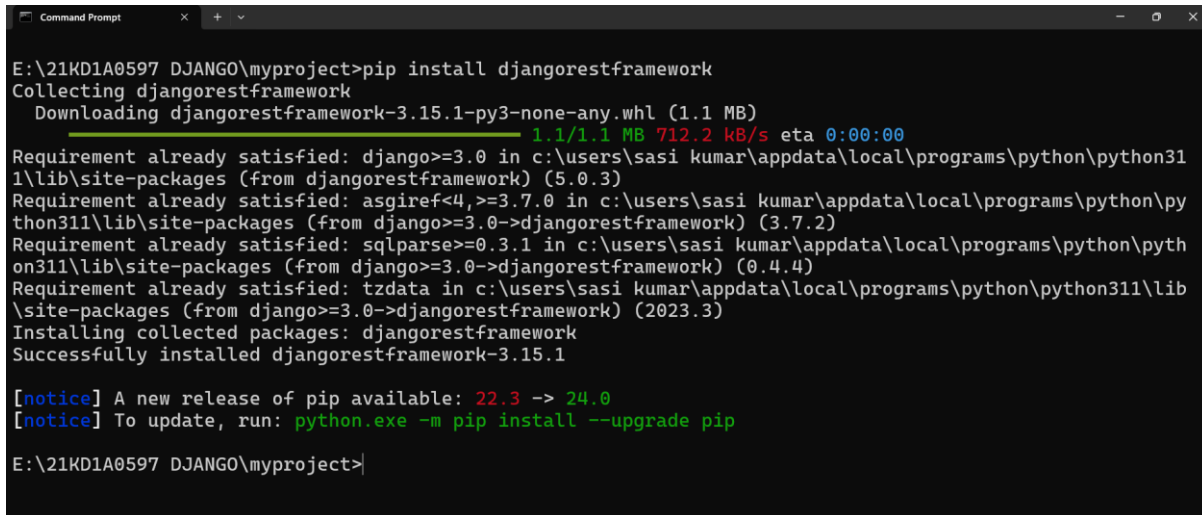
Thank you for logging as a customer.

MODULE-5

12) Create a rest API call for getCustomers to get (GET method) customer records from database using django rest framework.

1. Install Django Rest Framework:

Make sure you have Django Rest Framework installed. If not, you can install it using pip:
pip install djangorestframework



```

E:\21KD1A0597 DJANGO\myproject>pip install djangorestframework
Collecting djangorestframework
  Downloading djangorestframework-3.15.1-py3-none-any.whl (1.1 MB)
    1.1/1.1 MB 712.2 kB/s eta 0:00:00
Requirement already satisfied: django>=3.0 in c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages (from djangorestframework) (5.0.3)
Requirement already satisfied: asgiref<4,>=3.7.0 in c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages (from django>=3.0->djangorestframework) (3.7.2)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages (from django>=3.0->djangorestframework) (0.4.4)
Requirement already satisfied: tzdata in c:\users\sasi kumar\appdata\local\programs\python\python311\lib\site-packages (from django>=3.0->djangorestframework) (2023.3)
Installing collected packages: djangorestframework
Successfully installed djangorestframework-3.15.1

[notice] A new release of pip available: 22.3 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
E:\21KD1A0597 DJANGO\myproject>

```

2. Set up your Django project and app:

Create a new Django project: **django-admin startproject myproject**

Create a new Django app within the project: **python manage.py startapp myapp**

3. Define the Customer model:

In your app's directory (myapp), open models.py and add the following code:

```

from django.db import models
class Customer(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()
    phone_number = models.CharField(max_length=15)
    address = models.CharField(max_length=200)
    def __str__(self):
        return self.name

```

4. Create a serializer in serializers.py:

In your app's directory (myapp), open serializers.py and add the following code:

```

from rest_framework import serializers
from .models import Customer
class CustomerSerializer(serializers.ModelSerializer):
    class Meta:
        model = Customer
        fields = ['id', 'name', 'age', 'phone_number', 'address']

```

5. Create a view in views.py for the getCustomers API:

In your app's directory (myapp), open views.py and add the following code:

```
from rest_framework import generics
from .models import Customer
from .serializers import CustomerSerializer
class CustomerListAPIView(generics.ListAPIView):
    queryset = Customer.objects.all()
    serializer_class = CustomerSerializer
```

6. Configure the URLs:

In your app's directory (myapp), open urls.py or create a new file called urls.py if it doesn't exist.

Inside urls.py, add the following code:

```
from django.urls import path
from . import views
urlpatterns = [
    path('api/customers/', views.CustomerListAPIView.as_view(), name='customer-list'),
]
```

7. Configure the project's main URL:

In your project's urls.py file, add the following code:

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('myapp.urls')),
]
```

8. After defining the model, run the following command to create the necessary database tables:

```
python manage.py makemigrations
python manage.py migrate
```

9. Add the following in 'settings.py' file

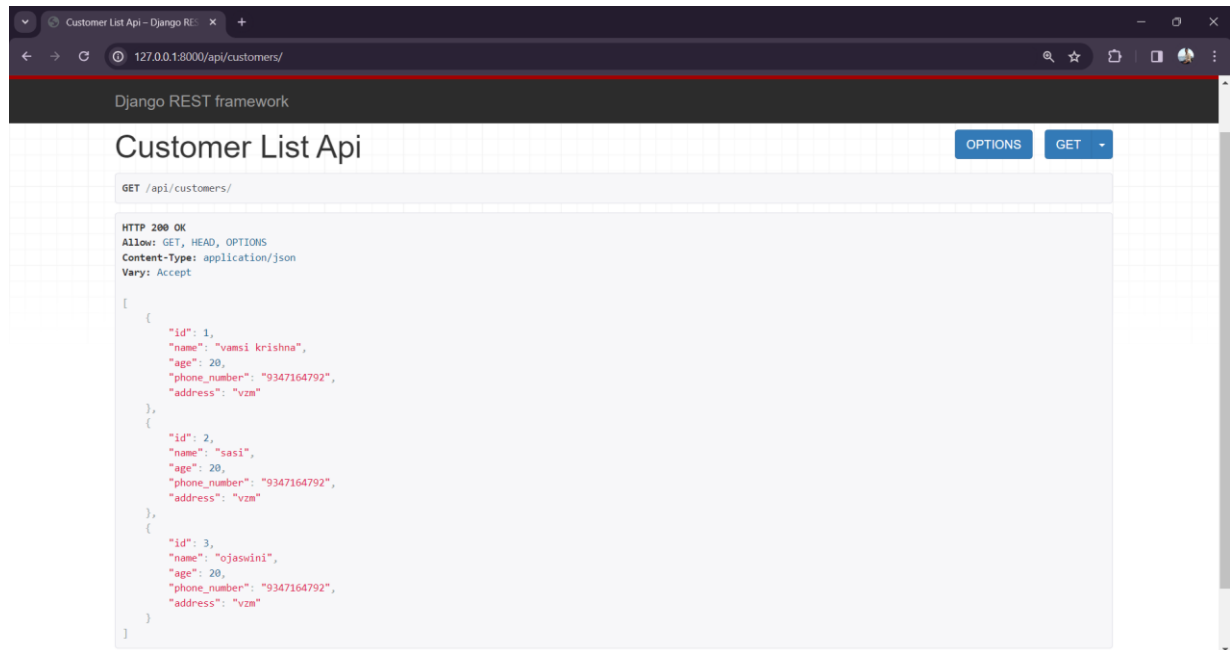
In your project's "settings.py", add 'myapp', 'rest_framework' in installed apps

10. Once the user is created run server. In command prompt run the following command for running the server.

```
python manage.py runserver
```

11. Open the browser and type the URL: <http://127.0.0.1:8000/api/customers/>

Then the Customer list API will open as below:



13) Create a rest API call for saveCustomer to save (POST method) customer records into database using django rest framework.

1. Follow steps 1-5 from the previous response to set up your Django project, app, model, serializer, and views.

2. Update your serializer in serializers.py:

Add a new serializer class to handle the creation of a customer record:

```
class CustomerCreateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Customer
        fields = ['name', 'age', 'phone_number', 'address']
```

3. Update your views.py to handle the saveCustomer API:

Add a new class-based view to handle the creation of a customer record:

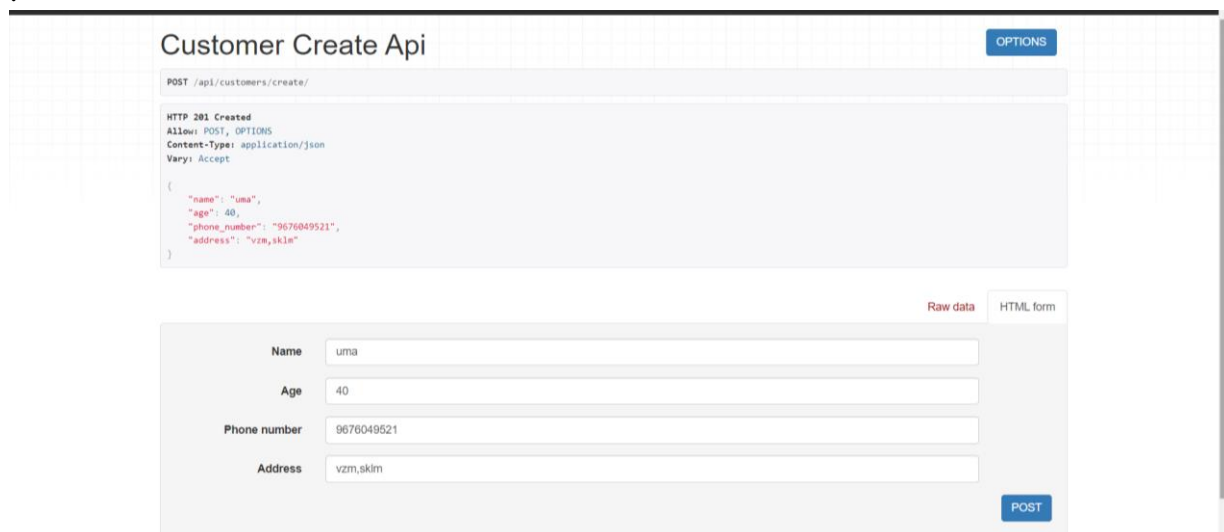
```
from .serializers import CustomerCreateSerializer
class CustomerCreateAPIView(generics.CreateAPIView):
    queryset = Customer.objects.all()
    serializer_class = CustomerCreateSerializer
```

4. Update your urls.py to include the new API endpoint:

```
from django.urls import path
from . import views
urlpatterns = [
    path('api/customers/', views.CustomerListAPIView.as_view(), name='customer-list'),
    path('api/customers/create/', views.CustomerCreateAPIView.as_view(), name='customer-create'),
]
```

5. Update the project's main URLs in urls.py as mentioned before.

Now, you can use the POST method to make a request to the `/api/customers/create/` endpoint to save customer records into the database



Customer Create Api OPTIONS

POST /api/customers/create/

HTTP 201 Created
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "name": "uma",
  "age": 40,
  "phone_number": "9676049521",
  "address": "vzm,skim"
}
```

Raw data HTML form

Name

Age

Phone number

Address

POST

14) Create a rest API call for updateCustomer to update (PUT method) customer records into database using django rest framework.

1. Follow steps 1-5 from the previous responses to set up your Django project, app, model, serializer, and views.

2. Update your serializer in serializers.py:

Add a new serializer class to handle the update of a customer record:

```
class CustomerUpdateSerializer(serializers.ModelSerializer):
    class Meta:
        model = Customer
        fields = ['name', 'age', 'phone_number', 'address']
```

3. Update your views.py to handle the updateCustomer API:

Add a new class-based view to handle the update of a customer record:

```
from .serializers import CustomerUpdateSerializer
class CustomerUpdateAPIView(generics.UpdateAPIView):
    queryset = Customer.objects.all()
    serializer_class = CustomerUpdateSerializer
    lookup_field = 'id' # Specify the lookup field (in this case, 'id')
```

4. Update your urls.py to include the new API endpoint:

```
from django.urls import path
from . import views
urlpatterns = [
    path('api/customers/update/<int:id>/', views.CustomerUpdateAPIView.as_view(), name='customer-update'),
]
```

5. Update the project's main URLs in urls.py as mentioned before.

Now, you can use the PUT method to make a request to the `/api/customers/update/{id}/` endpoint to update customer records in the database.

Replace `{id}` with the actual ID of the customer you want to update.

Customer Update Api OPTIONS

GET /api/customers/update/1/

HTTP 405 Method Not Allowed
Allow: PUT, PATCH, OPTIONS
Content-Type: application/json
Vary: Accept

```
{
  "detail": "Method \"GET\" not allowed."
}
```

Raw data HTML form

Name: M.VAMSI KRISHNA

Age: 20

Phone number: 9347164792

Address: vzm

PUT

Django REST framework

Customer Update Api

OPTIONS

PUT /api/customers/update/1/

HTTP 200 OK
Allow: PUT, PATCH, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  
  "name": "M.VAMSI KRISHNA",  
  "age": 20,  
  "phone_number": "9347164792",  
  "address": "vzm"  
}
```

Raw data HTML form

Name M.VAMSI KRISHNA

Age 20

Phone number 9347164792

Address vzm

PUT

The **lookup_field** attribute is set to 'id', specifying that the ID field should be used to look up the customer record to update.

Remember to include the necessary data in the request payload, following the fields specified in the **CustomerUpdateSerializer**.

15) Create a rest API call for deleteCustomer to delete (DELETE method) customer records from database using django rest framework.

1. Follow steps 1-5 from the previous responses to set up your Django project, app, model, serializer, and views.

2. Update your views.py to handle the deleteCustomer API:

Add a new class-based view to handle the deletion of a customer record:

```
class CustomerDeleteAPIView(generics.DestroyAPIView):
    queryset = Customer.objects.all()
    serializer_class = CustomerSerializer
    lookup_field = 'id' # Specify the lookup field (in this case, 'id')
```

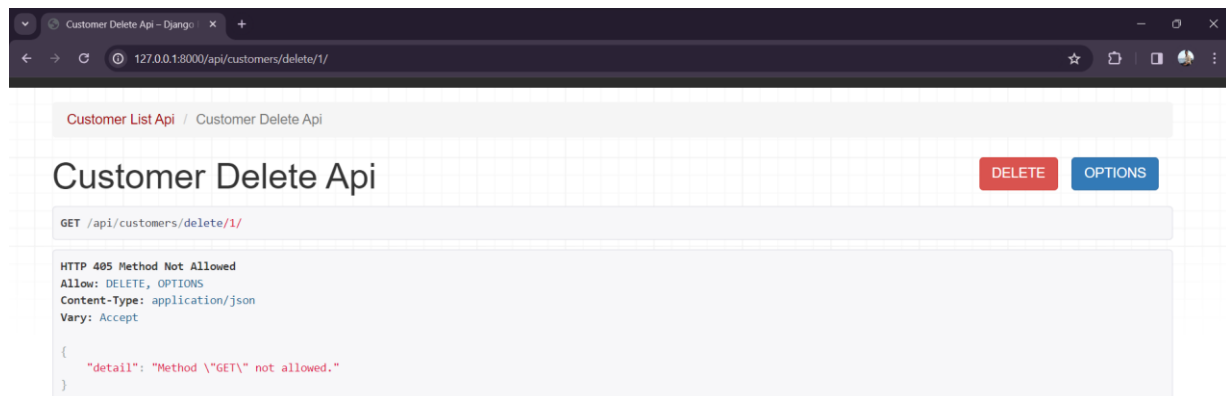
3. Update your urls.py to include the new API endpoint:

```
from django.urls import path
from . import views
urlpatterns = [
    path('api/customers/delete/<int:id>/', views.CustomerDeleteAPIView.as_view(), name='customer-delete'),
]
```

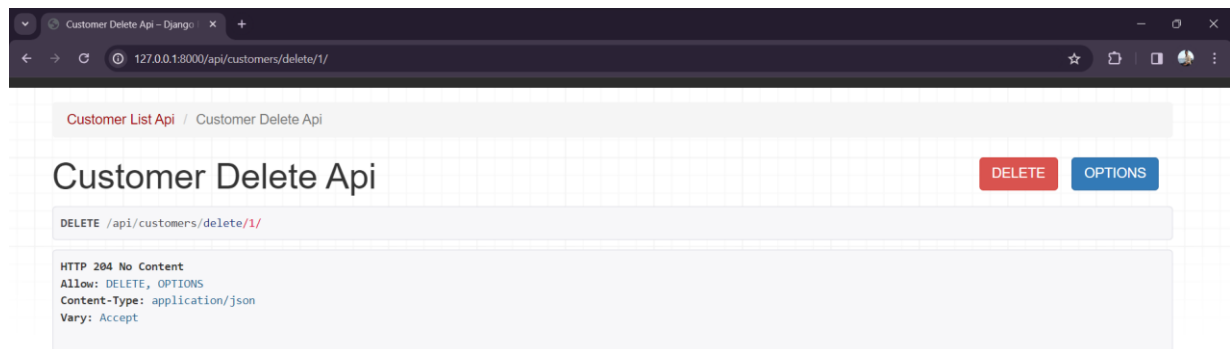
4. Update the project's main URLs in urls.py as mentioned before.

Now, you can use the DELETE method to make a request to the `/api/customers/delete/{id}/` endpoint to delete customer records from the database.

Replace `{id}` with the actual ID of the customer you want to delete.



Clicking the delete button to delete



Please note that the **DestroyAPIView** provided by Django Rest Framework handles the deletion operation automatically.

The **lookup_field** attribute is set to 'id', specifying that the ID field should be used to look up the customer record to delete.

When making a DELETE request to this endpoint, the corresponding customer record will be deleted from the database.

Remember to handle the appropriate authentication and authorization mechanisms to ensure that only authorized users can delete customer records.