LENDI

INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by A.I.C.T.E& Affiliated to JNTU GV, Vizianagaram)

JONNADA, VIZIANAGARAM DIST – 535005.

Certificate

This is to certify that this is the bonafide record of the work done by Mr. **T.S.V.Srikar** bearing Regd. No. **21KD1A05H8** in the **Django** Laboratory of **III** year **II** Semester of **B.Tech** Course in **CSE** Branch during the Academic Year 2023-2024.

Total number of
Experiments held .....

Total number of
Experiments held .....

LAB-IN-CHARGE

HEAD OF THE DEPARTMENT

INTERNAL EXAMINER

EXTERNAL EXAMINER

INDEX

Exp .No.	Date	Name of the Experiment	Page No.	Grade	Remarks
		Module-1			
1		Create Django environment setup and installation in windows/Linux.			
2		Create DJANGO project and app structure with django-admin commands.			
3		Deployment of project to the server			
4		Implement a simple HTTP response using DJango.			
		Module-2			
5		Implement template inheritance with views and images.			
6		Create a simple web page using django templates and static files.			
7		Create a django web page to render templates to multiple routes.			
		Module-3			
8		Create a Django model named customer having fields name, age, phone number, address and print the customer details in web page.			
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.			
		Module-4			
10		Create a registration form to store details of a customer into database.			
11		Create a login page for the customers who have registered in the database			
		Module-5			
12		Create a rest API call for getCustomers to get (GET method) customer records from database using django rest framework.			
13		Create a rest API call for saveCustomer to save (POST method) customer records into database using django rest framework.			
14		Create a rest API call for updateCustomer to update (PUT method) customer records into database using django rest framework.			
15		Create a rest API call for deleteCustomer to delete (DELETE method) customer records from database using django rest framework.			

Django Lab Manual

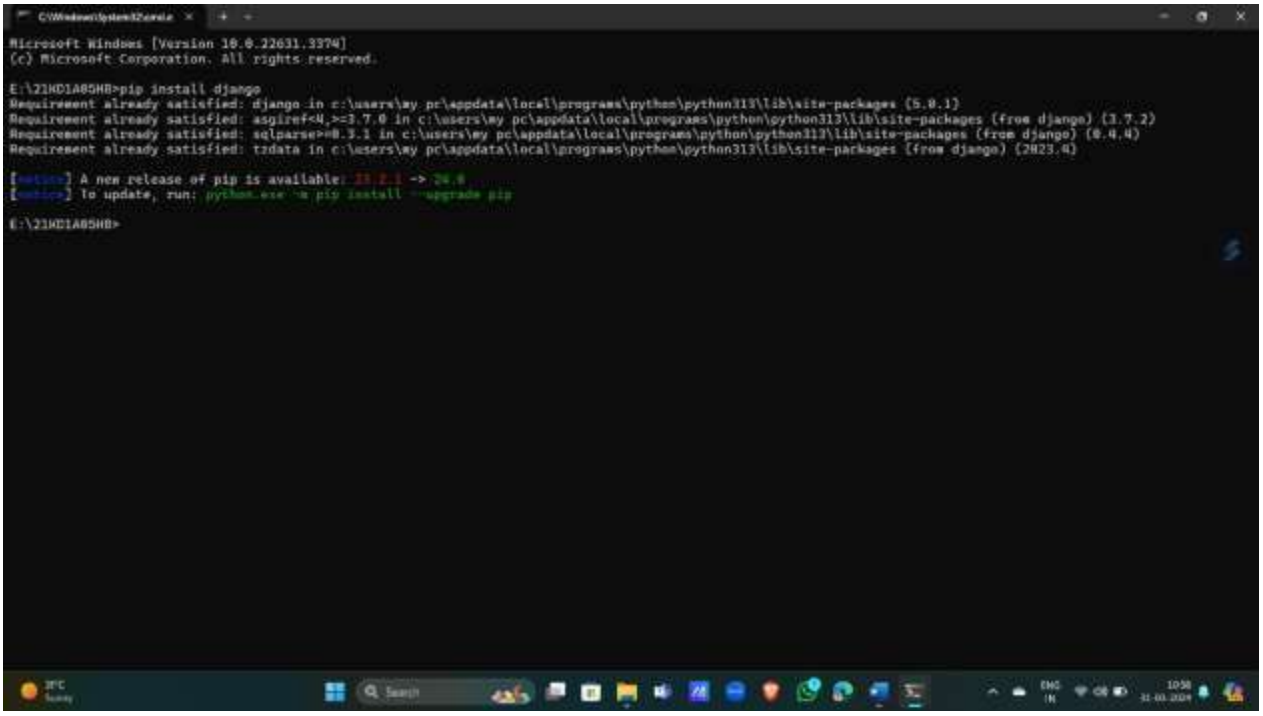
MODULE-1

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

Method -1: Using Python & pip

- Open **Command Prompt (cmd)**
- Change to E drive by using command e: enter
- Then we get E:\>
- Create a folder by using **cmd** : **mkdir <folder name> (press enter)**
- Change to folder using **cmd** : **cd <folder name> (press enter)**
- Ouput: **E:\<21KD1A05H8>**
- Installation of django : **pip install django**

Output:



Method -2: Using Virtual Environment

The name of the virtual environment is your choice, in this manual we will call it **myworld**.

python -m venv myworld

Then you have to activate the environment, by using the following command:

myworld\Scripts\activate.bat

Now, Django can be installed using pip, with the following command:

(myworld) C:\DjangoProjects>python -m pip install Django

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

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

C:\ DjangoProjects >django-admin

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

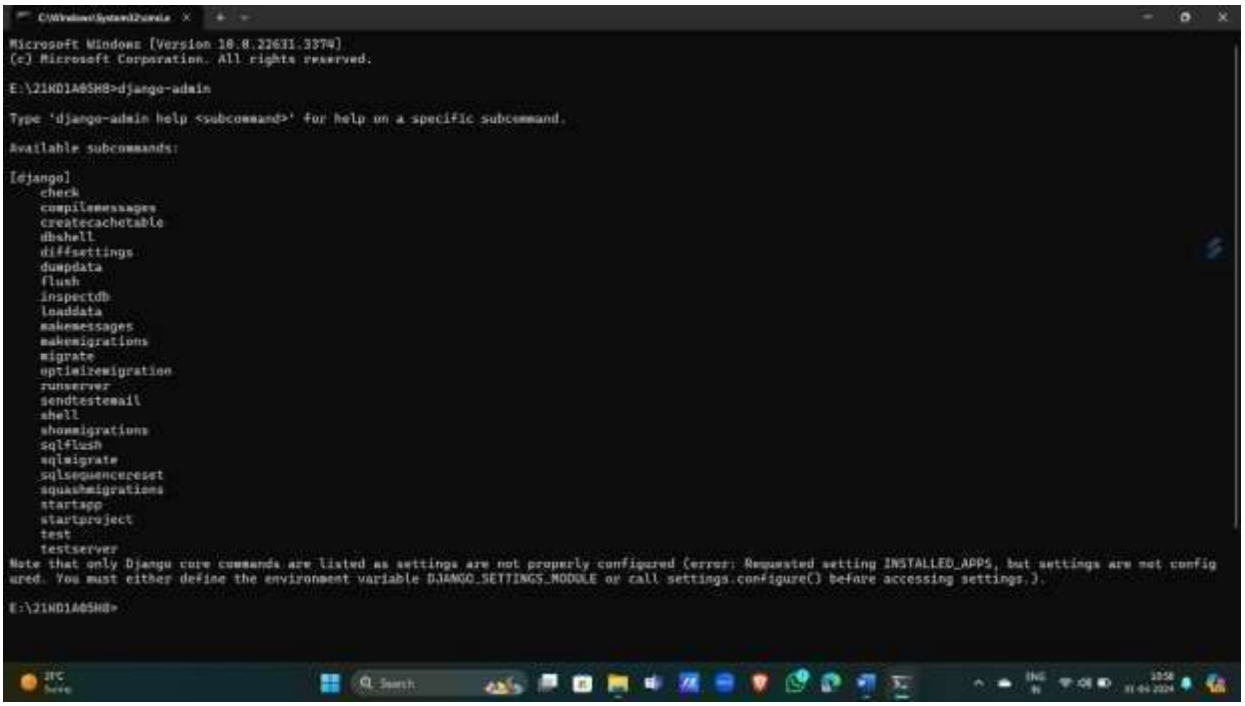
Available subcommands:

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

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.).

These are the commands available in django

Output:



a) Steps to create a project:

E:/<folder-name>**django-admin startproject <project name> (press enter)**

b) Steps for creating app inside project:

E:\<FOLDER-NAME> **cd <project name> (press enter)**

E:\<FOLDER-NAME>/<project name> **django-admin startapp <app name> (press enter)**

Output:

C:\Users\lendi>e:

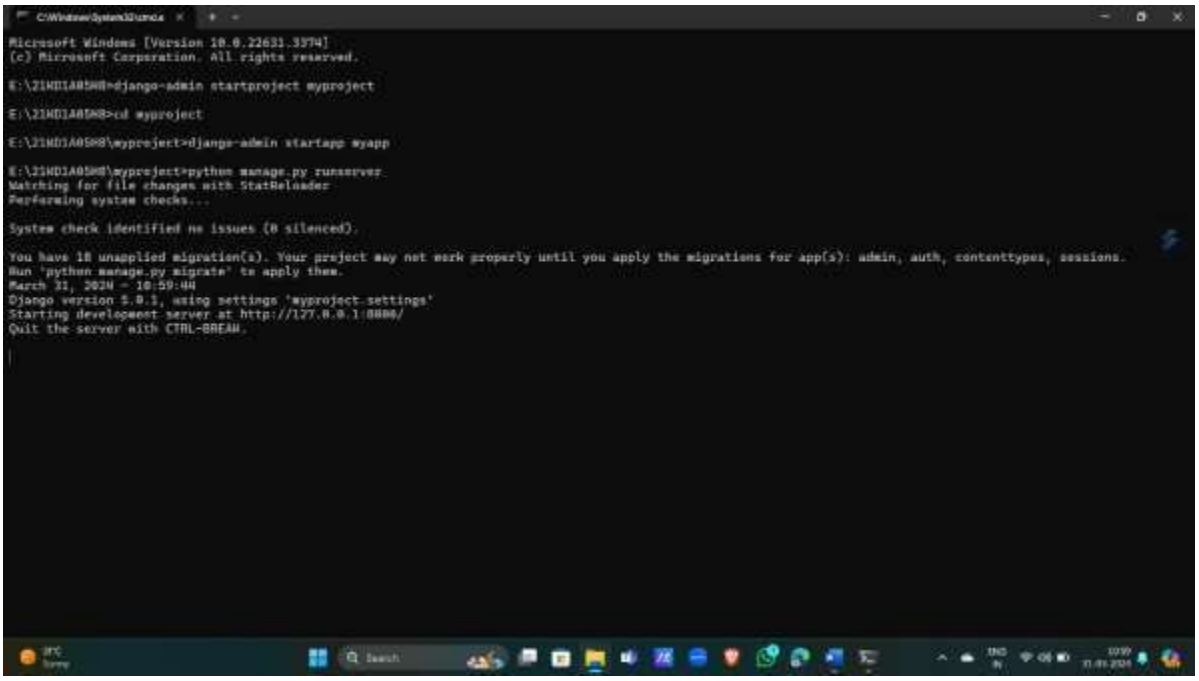
E:\>**mkdir djangoprojects**

E:\>**cd djangoprojects**

E:\21KD1A05H8>**django-admin startproject myproject**

E:\21KD1A05H8>**cd myproject**

E:\21KD1A05H8\myproject>**django-admin startapp myapp**
Output:



GO TO SUBLIME TEXT (OR) VISUAL STUDIO CODE:

Drag & drop the folder myproject in to sublime text or Visual Studio Code

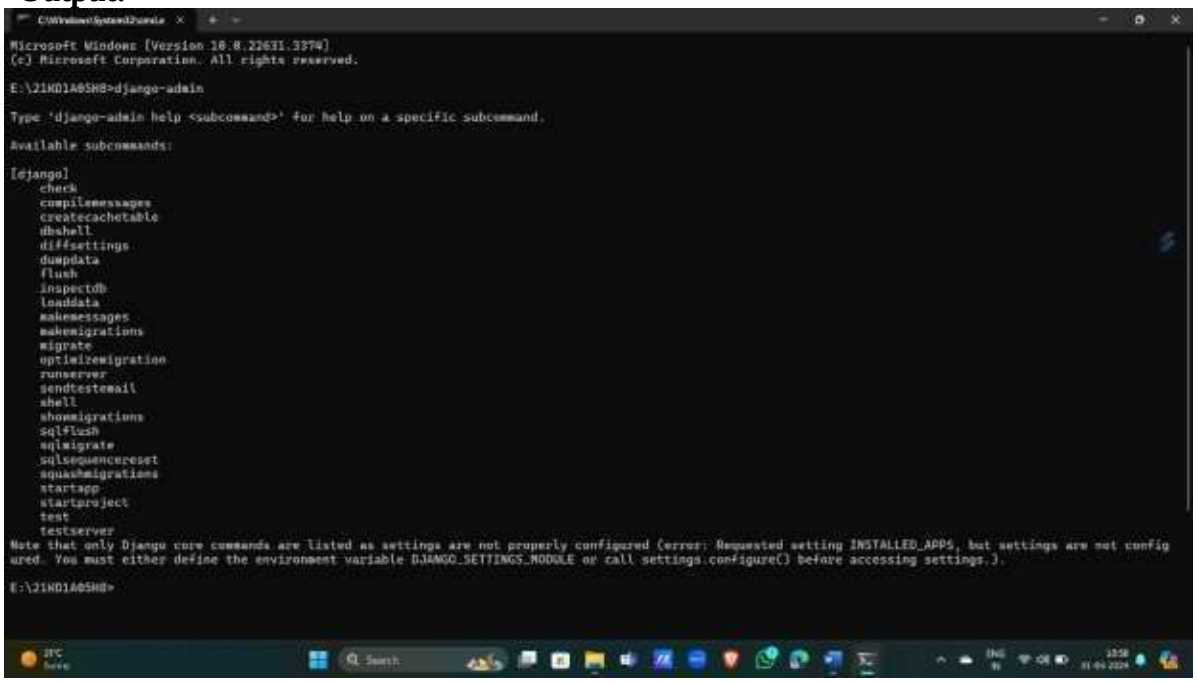
3) Deployment of project to the server

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

Command for deployment:

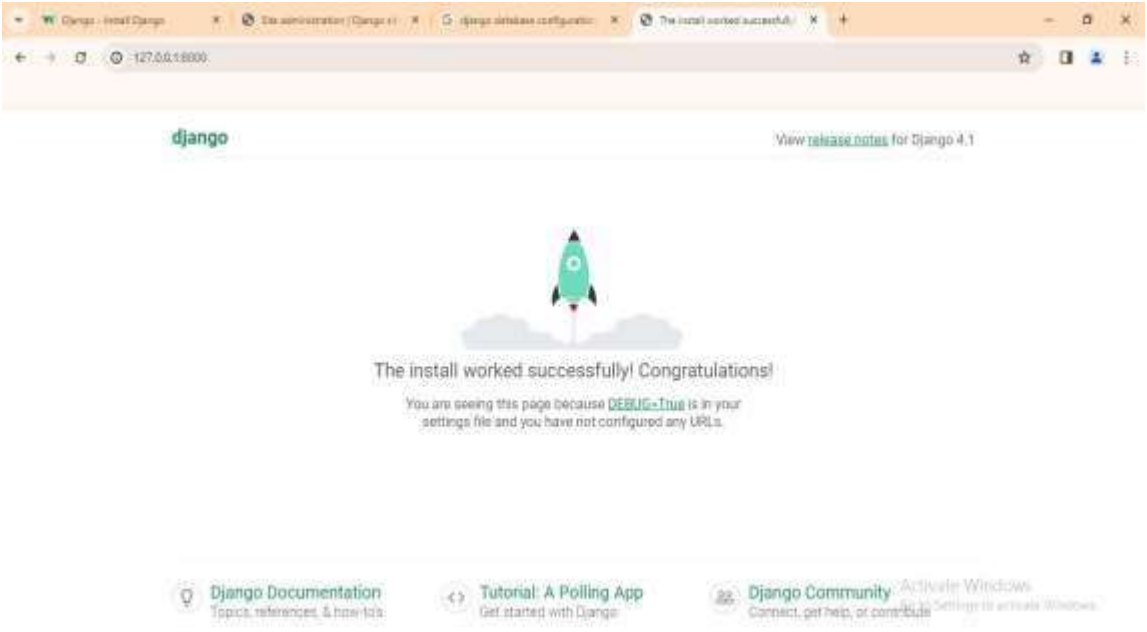
E:\djangoprojects\myproject>**python manage.py runserver**

Output:



Then, open your browser and visit the URL http://localhost:8000 to see the response.

Output:



4) Press the keys **Ctrl + C** for stopping the server.

4) Implement a simple HTTP response using DJango.

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

1. Set up your Django project and app:
   - Create a new Django project: **django-admin startproject myproject**

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

   a) 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>")
```

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

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

   c) 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')),
]
```

   d) Add the following line under INSTALLED_APPS in "**settings.py**" file
      'myapp'

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:

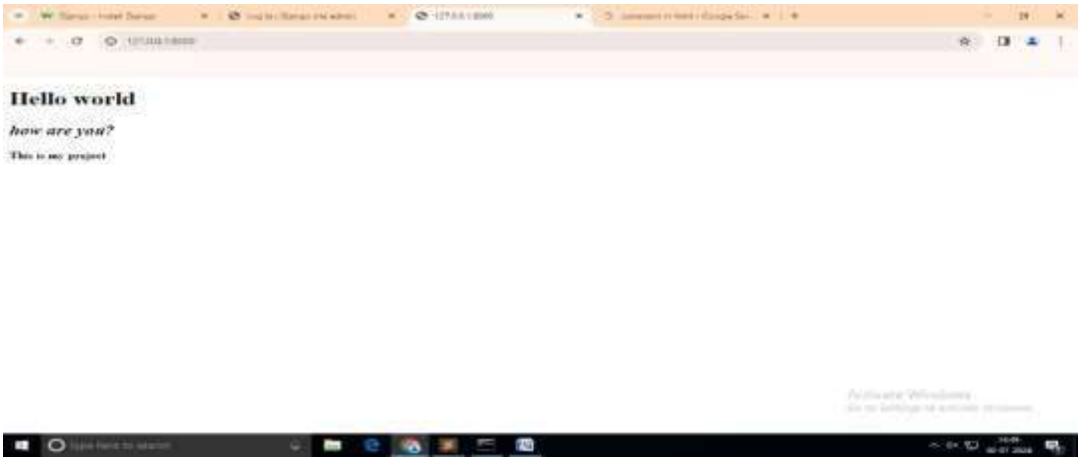
E:\djangoprojects\myproject>**python manage.py runserver**

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

path('', views.root, name='root'),

path('myapp', 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.

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

3. Set up your Django project and app:
   - Create a new Django project: **django-admin startproject myproject**

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

5. Configure static files:
   - In your project's "**settings.py**", add the following code:

```
import os

STATIC_URL = '/static/'
MEDIA_URL='/images/'

STATICFILES_DIRS=[
    os.path.join(BASE_DIR,'static')
]
```

6. 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>
</head>
<body>
<h1>Welcome to my web page!</h1>
<p>This is a simple web page created using Django templates and images.</p>
<img src="{% static 'images/myimage.jpg' %}" alt="My Image">
</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 a subdirectory: **"images"**.
- Place 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:

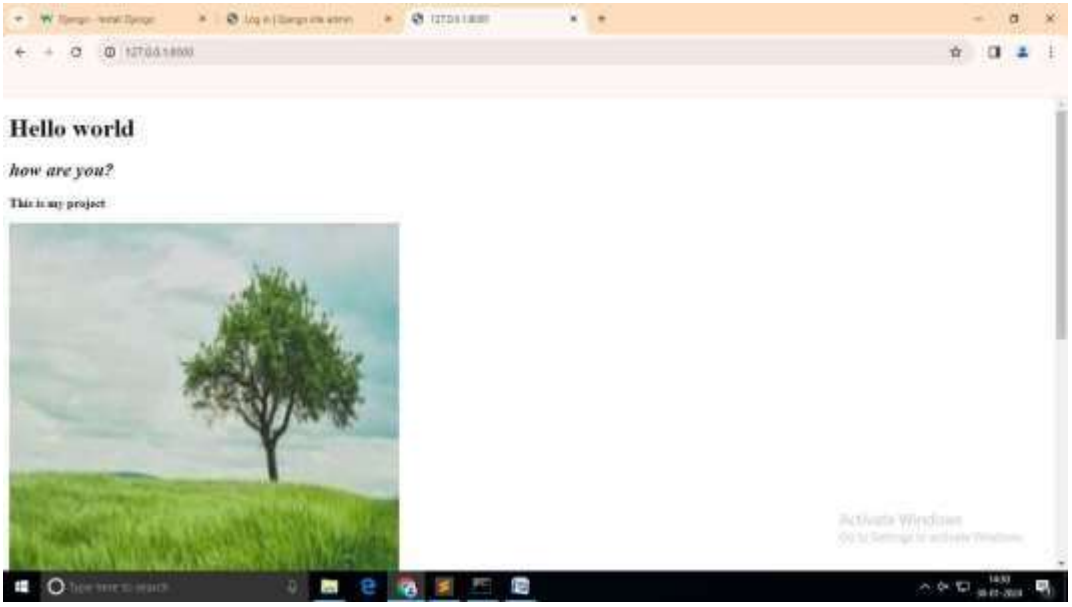
- In your project's "**urls.py**" file, add the following code:

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

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.



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.

7. Set up your Django project and app:
   - Create a new Django project: **django-admin startproject myproject**

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

9. Configure static files:
   - In your project's settings.py, add the following code:

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [
        BASE_DIR / 'static',
        '/var/www/static/',
]
```

10. 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>
<img src="{% static 'images/myimage.jpg' %}" alt="My Image">
</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:

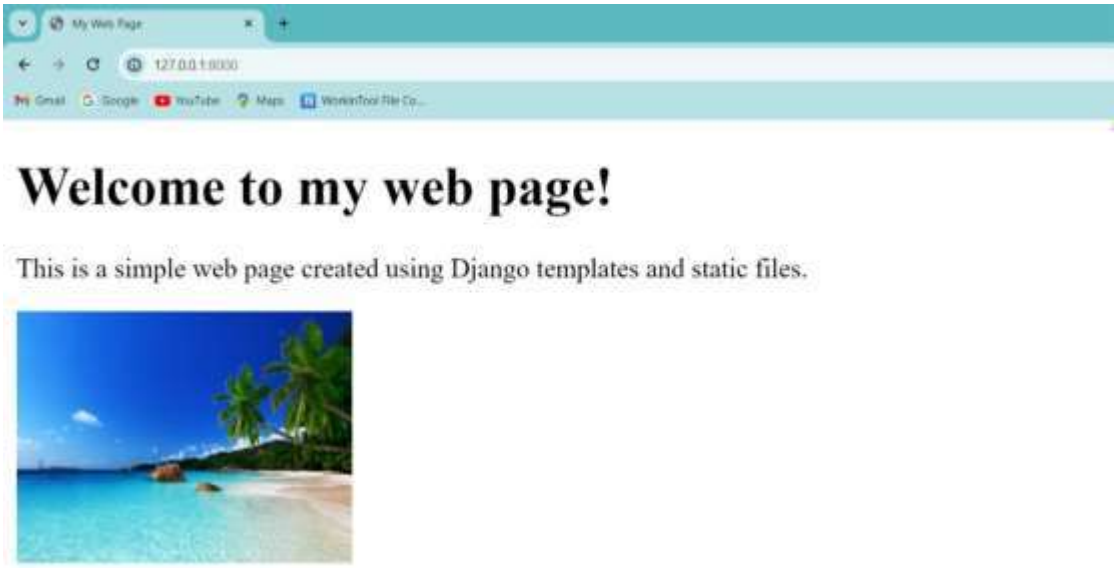
- In your project's "**urls.py**" file, add the following code:

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

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.

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 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')
```

5. 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('', views.home, name='home'),
        path('about/', views.about, name='about'),
]
```

6. 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.

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

Then, open your browser and visit http://localhost:8000 for the home page and http://localhost:8000/about for the about page.

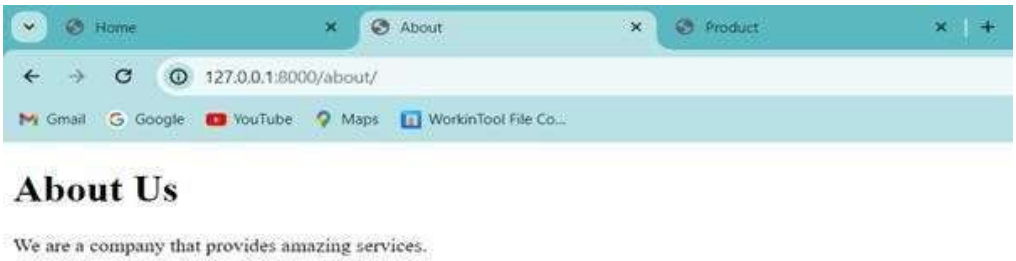
Similarly create multiple routes for the following.

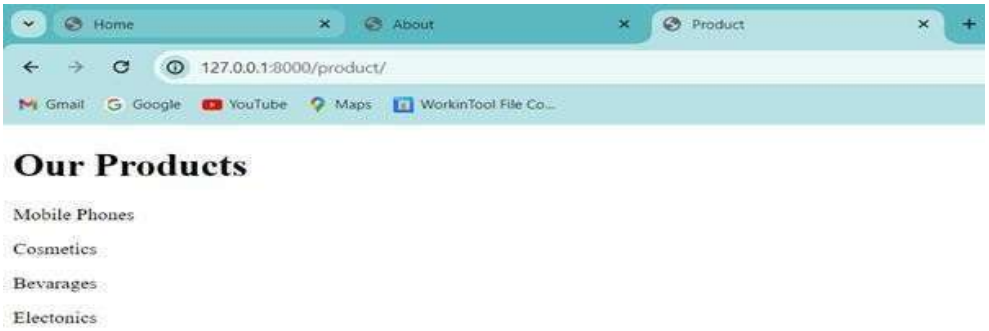
http://127.0.0.1:8000/index
http://127.0.0.1:8000/products
http://127.0.0.1:8000/contact

Create a new page products.html in 'templates' folder

Map the url to

http://127.0.0.1:8000/emp/products


Home
About
Product
127.0.0.1:8000/about/
Gmail
Google
YouTube
Maps
WorkinTool File Co...
About Us
We are a company that provides amazing services.


Home
About
Product
127.0.0.1:8000/product/
Gmail
Google
YouTube
Maps
WorkinTool File Co...
Our Products
Mobile Phones
Cosmetics
Bevarages
Electonics

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

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

5. 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)
```

6. 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})
```

7. 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

```
<!DOCTYPE html>
```

```
<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>
```

8. 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'),
]
```

9. 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')),
]
```

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'

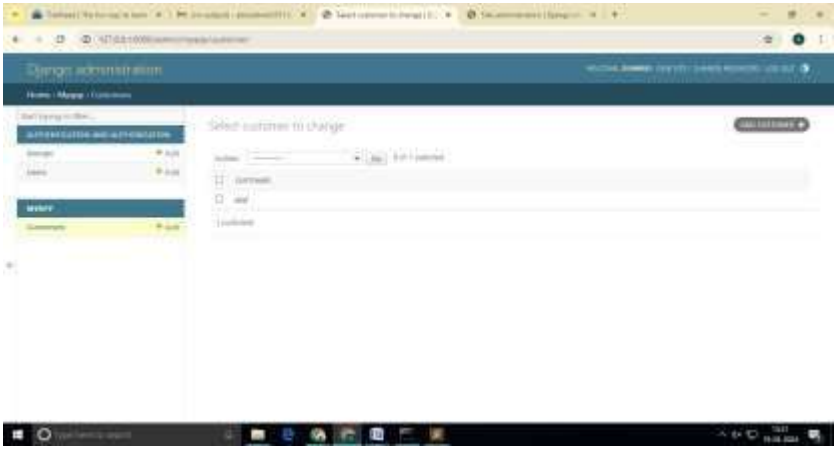
11. Once the user is created run server. In command prompt run the following command for running the server.
    python manage.py runserver

12. Open the browser and type the URL

13. http://127.0.0.1:8000/admin/

14. Login using above admin user credentials.

15. After logging in we are directed to site administration.



The customer details will be displayed on a web page when you visit **http://localhost:8000/customer-details/**

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:
    a. Create a new Django project: **django-admin startproject myproject**
2. Create a new Django app within the project: **python manage.py startapp myapp**
3. Define the **Customer** and **Order** models:
    a. 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
```

4. After defining the models, run the following command to create the necessary database tables:
   python manage.py makemigrations and then
   python manage.py migrate

5. In your app's directory(myapp), opne 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)
```

6. 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})
```

7. 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>
```

8. 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 = [
```

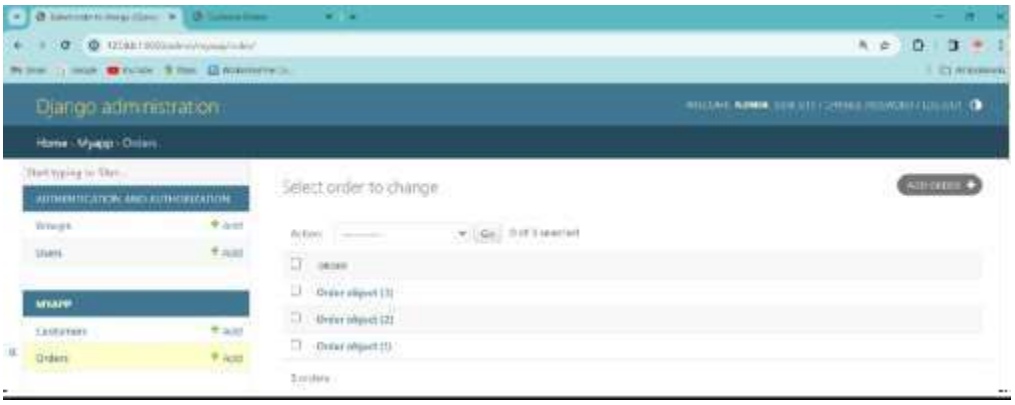
```
    path('customer-orders/<int:customer_id>/', views.customer_orders,
name='customer_orders'),
]
```

9. 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')),
]
```

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

http://localhost:8000/customer-orders/2/



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

1. Set up your Django project and app:
   a. Create a new Django project: **django-admin startproject myproject**
   b. Create a new Django app within the project: **python manage.py startapp myapp**
2. Define the **Customer** model:
   a. 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. After defining the model, run the following command to create the necessary database tables: **python manage.py makemigrations** and then **python manage.py migrate**.
4. Create a registration form in forms.py:
   a. In your app's directory (myapp), 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']
```

5. Create a view in views.py to handle the registration form:
   b. In your app's directory (myapp), 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')
```

6. Create templates for the registration form and success message:
    - 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 "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>
```

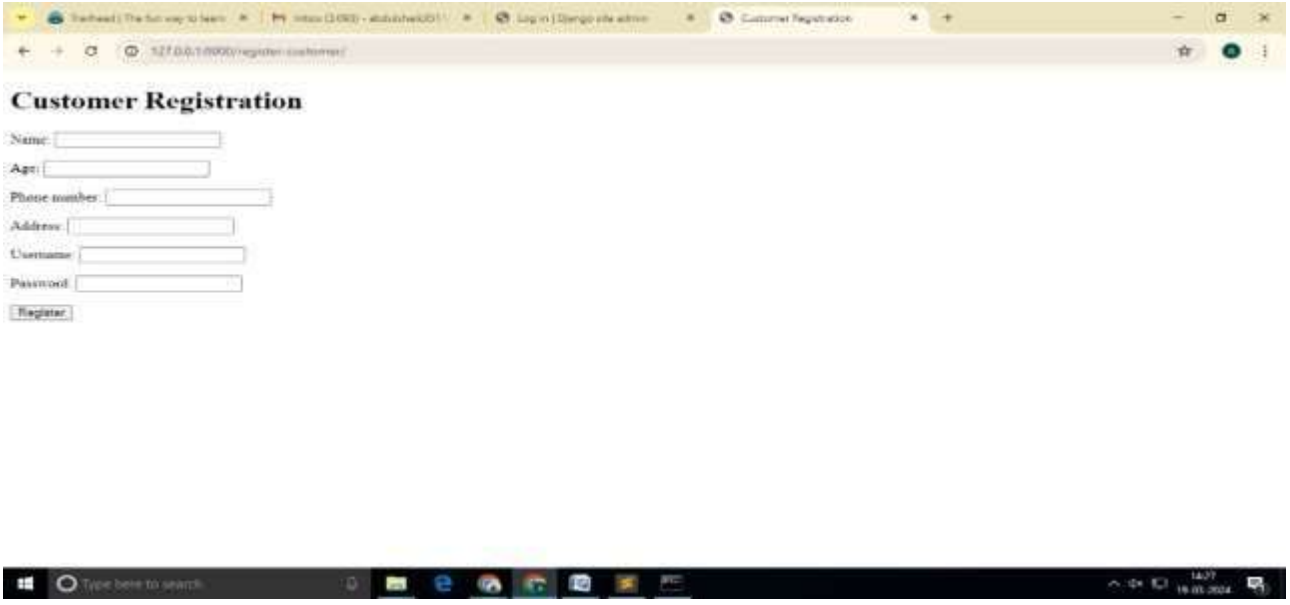
7. Configure the URLs:
    - In your app's directory (myapp), open urls.py or create a new file called urls.py if it doesn't exist.
    - Inside urls.py, add the following code:

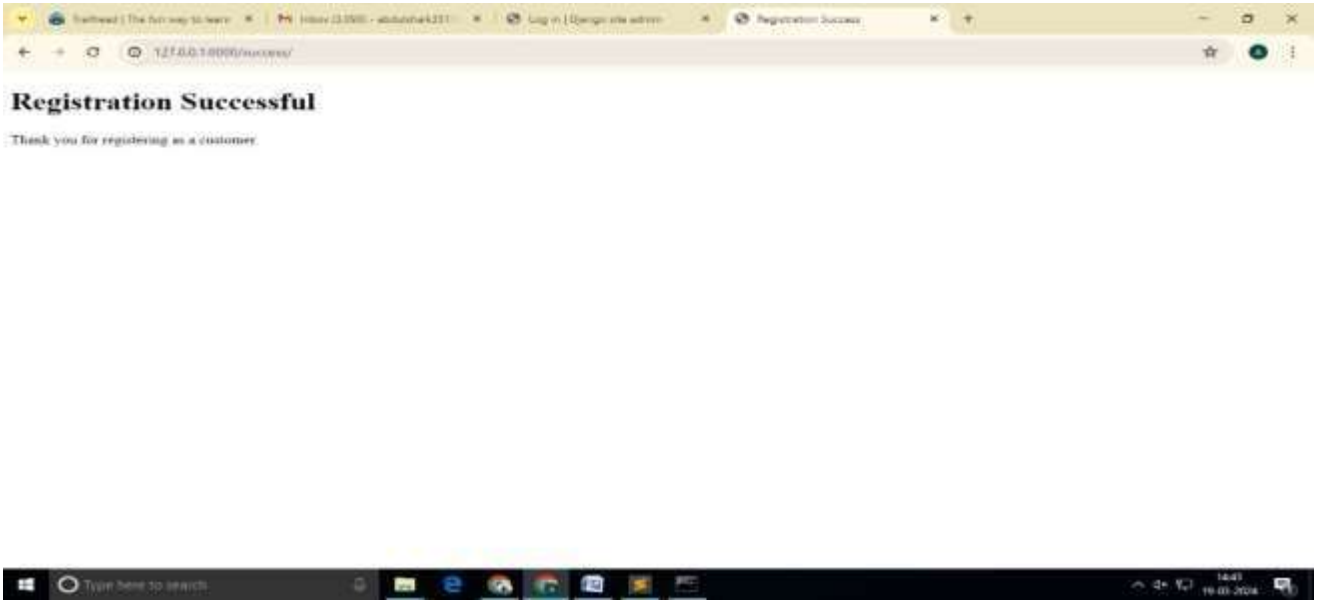
```
from django.urls import path
from . import views
urlpatterns = [
        path('register-customer/', views.register_customer, name='register_customer'),
        path('success/', views.registration_success, name='success'),
]
```

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





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**
2. 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)
username = models.CharField(max_length=100, unique=True)
password = models.CharField(max_length=100)
def___str__(self):
        return self.name
```

4. After defining the model, run the following command to create the necessary database tables: **python manage.py makemigrations** and then **python manage.py migrate**.
5. 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)
```

6. 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})
```

7. Create a template for the login page:
    - In your app's directory (myapp), create a new directory called "templates" if it doesn't exist.
8. 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>
```

9. 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'),
]
```

10. Configure the project's main URL:
11. 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')),
```

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:
   a. Make sure you have Django Rest Framework installed. If not, you can install it using pip: **pip install djangorestframework**
2. Set up your Django project and app:
   a. Create a new Django project: **django-admin startproject myproject**
   b. Create a new Django app within the project: **python manage.py startapp myapp**
3. Define the Customer model:
   a. 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. After defining the model, run the following command to create the necessary database tables: **python manage.py makemigrations** and then **python manage.py migrate**.
   a. In your app's directory (myapp), open serializers.py and add the following code:

5. Create a serializer in serializers.py:

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

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

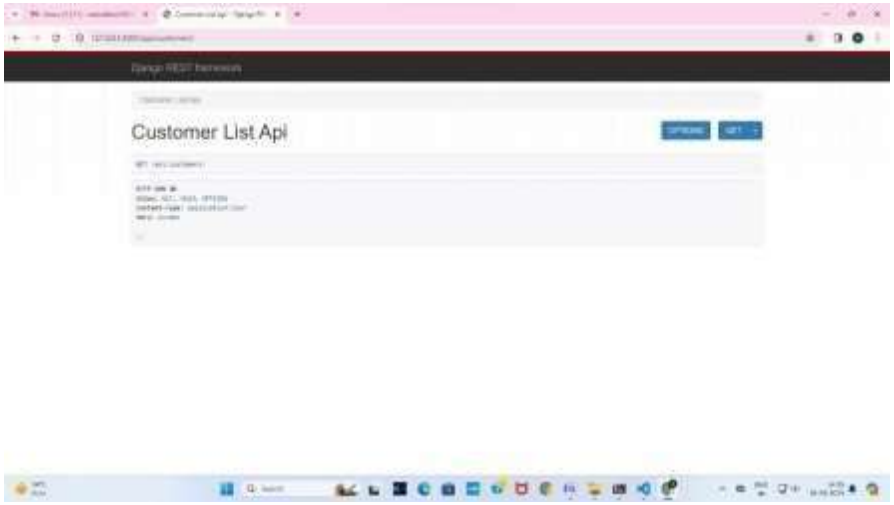
7. Configure the URLs:
   a. In your app's directory (myapp), open urls.py or create a new file called urls.py if it doesn't exist.

b. 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'),
]
```

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

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



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

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

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

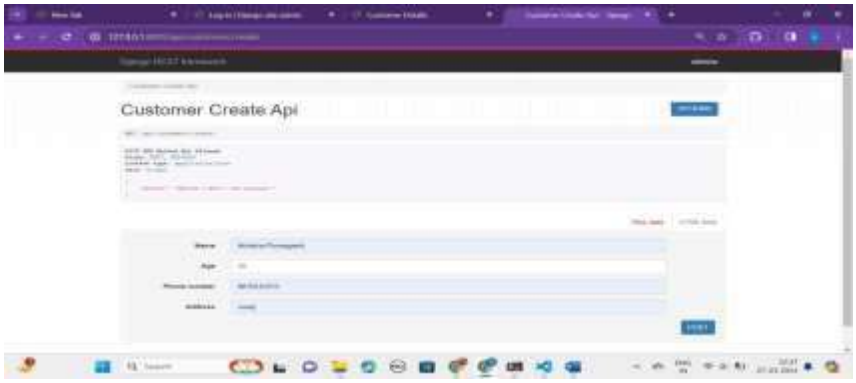
```
from django.urls import path
from . import views
urlpatterns = [
        path('api/customers/', views.CustomerListAPIView.as_view(),
name='customer-list'),
        path('api/customers/create/', views.CustomerCreateAPIView.as_view(),
        name='customer-create'),
]
```

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.

In this example, the **CreateAPIView** is used to handle the creation of customer records automatically.

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



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

```
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/', views.CustomerListAPIView.as_view(), name='customer-list'),
    path('api/customers/create/', views.CustomerCreateAPIView.as_view(),
name='customer-create'),
    path('api/customers/update/<int:pk>/', 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.

Please note that the **UpdateAPIView** provided by Django Rest Framework handles the update 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 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:
    a. 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
l       ookup_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/' views.CustomerListAPIView.as_view(), name='customer-list'),
    path('api/customers/create/', views.CustomerCreateAPIView.as_view(),
name='customer-create’),
    path('api/customers/update/<int:pk>/', views.CustomerUpdateAPIView.as_view(),
name='customer-update'),
    path('api/customers/delete/<int:pk>/', 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.

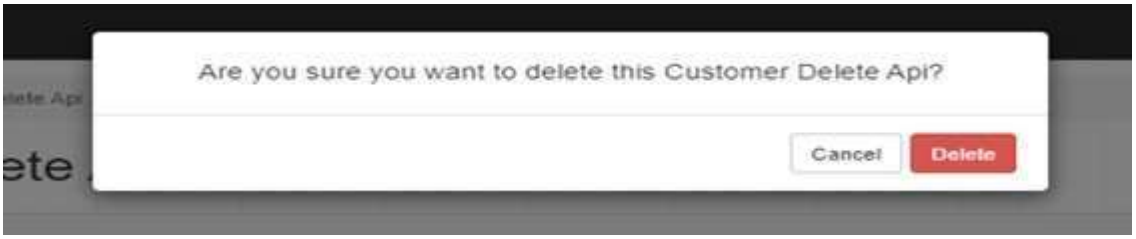
Please note that the **DestroyAPIView** provided by Django Rest Framework handles the deletion operation automatically.

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.


Django REST framework
admin
Customer List Api
Customer Delete Api
Customer Delete Api
DELETE
OPTIONS


Are you sure you want to delete this Customer Delete Api?
Cancel
Delete