# Index

## Experiment 1 : First output in Django

- In the terminal, create virtual environment named **myenv** in the current directory using following command

```
python3 -m venv myenv
```

- Activate the virtual environment using below command

  For Mac: `source myenv/bin/activate`

  For Windows: `.\myenv\Scripts\activate`

- Below two commands are optional:

  Check python version using below command

```
python3 —version
```

  Check pip version using below command

```
pip —version
```

- Install Django using below command inside the virtual environment **myenv**. Once installed Django will not be available outside the virtual environment.

```
pip install django
```

- Create a django project named **MyFirstDjango** using below command.

```
django-admin startproject MyFirstDjango
```

- Create an app called **calc** inside **MyFirstDjango** folder.

```
python manage.py startapp calc
```

- Create an empty file in **calc** app named **urls.py**

- Place the below content in **urls.py** of **MyFirstDjango** folder

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

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

- Place the below content in **urls.py** of **calc** app folder

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

urlpatterns=[
    path(‘’, views.home, name='home')
]
```

- Include a home method in **views.py** and render to home.html

```
from django.shortcuts import render
def home(request):
    return render(request,’home.html’,{‘name’:’Raju’})
```

- Create a folder named **templates** inside the project folder where **manage.py** and **calc** app exists. Make the following changes in **settings.py.** Place 'calc' inside installed apps.

```
import os

'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

- Place the **home.html** file inside **templates** folder. Place the following content in **home.html**

```
<h1> Hello {{name}} </h1>
```

- Save all the files and run the below command

```
python manage.py runserver
```

Open browser and enter **127.0.0.1:8000** to see your output.

## Experiment 2 : HTML forms in Django

Include a second path in **urls.py** of **calc** app folder as given below. Place a comma after first path

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

Create a HTML form inside **home.html** to collect two numbers as input from user as shown below.

In VS code to get the default html template press !+Enter

```
<form action="add" method="post">
    {% csrf_token %}
    Enter First Number: <input type="text" name="num1"><br>
    Enter Second Number: <input type="text" name="num2"><br>

    <input type="submit">

</form>
```

In **views.py** add the below method

```
def add(request):
     val1=int(request.POST[‘num1'])
     val2=int(request.POST[‘num2'])
     val3=val1+val2
     return render(request,’result.html’,{'result':val3})
```

Create **results.html** in **templates** folder and place the below content

```
<h1>Result: {{result}} </h1>
```

In the **result.html** place the below content to create a back button to home page.

```
<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>
```

- Save all the files and run the below command

```
python manage.py runserver
```

- Open browser and enter 127.0.0.1:8000 to see your output.

## Experiment 3 : Template Inheritance

- Create **base.html** file inside **templates** folder. In VS Code editor press !+Enter to get default template of HTML. Change the body tag styling using inline style and change background colour as shown below. Add block content and endblock as shown below.

```
<body style="background-color: powderblue;">
    {% block content %}
    {% endblock %}
</body>
```

- In **home.html** and **result.html** place the following changes to inherit styling from **base.html**. Immediately after the body tags or in the top of above html pages place the below content:

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

- In the bottom of html page before the end body tags place the below content. Between them place the html content.
-

```
{% endblock %}
```

- Save all the files and run the below command

```
python manage.py runserver
```

- Open browser and enter 127.0.0.1:8000 to see your output.

## Experiment 4 : Bootstrap and Navigation Bar

- Google Bootstrap Navbar and go to https://getbootstrap.com/docs/4.0/components/navbar/. Create a new html page called **navbar.html** and copy one of the templates and place it in the **templates** folder. For example one of the templates look like below. (Note: don’t type the below code, copy and paste from above website).

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="#">Navbar</a>
  <button class=“navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNavDropdown" aria-controls="navbarNavDropdown" aria-expanded="false"
aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="collapse navbar-collapse" id="navbarNavDropdown">
    <ul class="navbar-nav">
      <li class="nav-item active">
        <a class="nav-link" href="#">Home <span class="sr-only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Features</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="#">Pricing</a>
      </li>
      <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
          Dropdown link
        </a>
        <div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
          <a class="dropdown-item" href="#">Action</a>
          <a class="dropdown-item" href="#">Another action</a>
          <a class="dropdown-item" href="#">Something else here</a>
        </div>
      </li>
    </ul>
  </div>
</nav>
```

- In **base.html** make the following change below the body tag as follows

```
<body style="background-color: powderblue;">

    {% include 'navbar.html' %}
```

- Place the below link in the head tag of **base.html**

```
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
crossorigin="anonymous">
```

- Place the below script command after the body’s closing tags. Refer to the last chapter to see the complete code.

```
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/
X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/dist/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
```

- In **navbar.html** activate the hyperlink tabs by changing the href content as shown below

```
<a class="nav-link" href="{% url 'dashboard' %}”>Dashboard</a>
```

- To make the above url working, we should add the path in **urls.py** as follows and define method in **views.py.**

```
path(‘dashboard/',views.dashboard, name=‘dashboard'),
```

- In **views.py** include the below dashboard method

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

- Create a **dashboard.html** file in **templates** folder with below content to return back to home page.

```
<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>
```

- Do all the above four steps for products page instead of dashboard. Refer to last chapter for complete code.
- Save all the files and run the below command

```
python manage.py runserver
```

- Open browser and enter **127.0.0.1:8000** to see your output.

## Experiment 5 : Managing Static Files (Images, JS, and CSS) in Django

- Create a folder named **static** in the project folder which contains templates, calc etc. Create a subfolder named **Images** insider static folder. Insert KLH logo image in Images folder.
- In **settings.py** file make the following changes:

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

- Inside the **navbar.html** place the below highlighted (in yellow) content.

```
{% load static %}

<nav class="navbar navbar-expand-lg navbar-light bg-light">
     <img src="{% static 'Images/klhlogo.png' %}"
     <a class="navbar-brand" href="#">Navbar</a>
```

- Save all the files and run the below command

```
python manage.py runserver
```

- Open browser and enter **127.0.0.1:8000** to see your output.

## Experiment 6 : Git Terminal Commands to Connect Local Repository with Remote Repository.

- Create a folder named GitTest. We attempt to transform this folder as a repository and connect it to the remote repository accessed through Github web account.
- Step 1: `cd GitTest`

  Enter the directory GitTest
- Step 2: `git init`

  Initialise a new git repository. It creates .git directory in the root of the project that contains all version control files.
- Step 3: `git add .`

  To stage all changes in the current and sub-directories for the next commit.
- Step 4: `git commit -m "Initial Commit"`

  Creates a new commit in your Git repository with a message describing changes.
- Step 5: `git branch -M main`

  Above command is used to rename the current branch to main.
- Step 6: `git branch`

  Above command checks your current branch

Step 7: `git remote add origin https://github.com/csreddy89/GitTest.git`

Creates a remote repository in GitHub named GitTest. Above command is used to add a remote repository to your local Git repository. It links local repository to remote repository in GitHub allowing to pull and push between local and remote repositories.

Step 8: `git push -u origin main`

Push your local branch main to remote repository. -u is needed in the beginning, later it becomes default. No need of using during second invoking of the same command unless there is a need for change of branch.

Make changes to any file and push using below commands.

1. `git add .`
2. `git commit -m “1st change”`
3. `git push origin main`

**Git commands for switching a branch:**

```
git checkout <branch-name>
```

or

```
git switch <branch-name>
```

Use above commands to switch to a new branch. If we use `-c` after switch it creates a new branch and switches to it. Also `-b` after checkout also creates or changes a branch.

**Git commands to clone a remote repository to local:**

```
git clone <repository-url>
```

`Ex: git clone` https://github.com/csreddy89/GitTest.git

**Cloning in VS Code:**

In home page of VS Code IDE click on Clone Git repository. Paste the copied Git repository link in the VS code search bar. Select a destination folder the local directory

**Git Pull :**

If you have changes in the remote repository then local repository can be synced using the following pull command.

```
git pull origin main
```

**General form:** `git pull <remote> <branch>`

`remote` is name of remote repository.

**Forking a repository:**

A fork is a personal copy of another user's repository that resides in your own GitHub account. Forking a repository allows you to freely experiment with changes without affecting the original project.

## Experiment 7 : Hosting Django Web Application using Render Cloud Platform

- Create a **requirements.txt** file by running following command in terminal.

```
pip3 freeze > requirements.txt
```

Include following package manually in to **requirements.txt** file.

```
gunicorn == 22.0.0
```

Make the following changes in **settings.py** file of your Django project.

```
ALLOWED_HOSTS = ['MyFirstDjango.onrender.com','127.0.0.1']
```

Note: Above given name should match with the name given in Render platform while hosting the web application.

In the Render platform following changes are needed:

Name - MyFirstDjango

Start Command - gunicorn MyFirstDjango.wsgi

Instance Type - Free

Click on create web service.

## Experiment 8 : Creating Tables through Model Classes and Rendering through Django

Django follows Object-Relational Mapping (ORM) where database tables can be created as model classes. These tables can be accessed through Django admin account that can be accessed using the link: **127.0.0.1:8000/admin**

Before we access Django admin page, we need to migrate and create a superuser as follows:

```
python manage.py migrate
python manage.py createsuperuser
```

Enter the username and password. Password will be asked twice. Press 'Y' if you want to proceed with small password. Now you can access Django admin account using above link.

Create a model class for database table in **models.py** as follows

```
class Customer(models.Model):
    name=models.CharField(max_length=300, null=True)
    age=models.CharField(max_length=300, null=True)
    date=models.DateTimeField(max_length=300, null=True)

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

```
class Tag(models.Model):
    name=models.CharField(max_length=300, null=True)

    def __str__(self):
        return self.name

class Product(models.Model):

    CATEGORY=(
        ('Indoor','Indoor'),
        ('Outdoor','Outdoor')
        )
    name=models.CharField(max_length=300,null=True)
    price=models.FloatField(max_length=300,null=True)
    category=models.CharField(max_length=300,choices=CATEGORY)
    tags=models.ManyToManyField(Tag)

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

In **admin.py** import models and tables as follows.

```
from .models import *

admin.site.register(Customer)
admin.site.register(Product)
admin.site.register(Tag)
```

In **views.py** create the following methods and also import models as follows

```
from .models import *


def dashboard(request):

    customers=Customer.objects.all()
    return render(request,’dashboard.html’,{'customers':customers})


def products(request):
    products=Product.objects.all()
    return render(request,’product.html',{'products':products})
```

In **templates** folder edit the **dashboard.html** as follows:

```
{% block content %}

<h1>Dashboard</h1>

&nbsp;

<table class="table">
    <tr>
       <th>Name</th>
        <th>Age</th>
    </tr>
    {% for customer in customers %}
    <tr>
       <td>{{customer.name}}</td>
        <td>{{customer.age}}</td>
    </tr>

    {% endfor %}

</table>

<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>

{% endblock %}
```

Create **product.html** as given below:

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

{% block content %}

<h1>Products</h1>

&nbsp;

<table class="table">
    <tr>
        <th>Name</th>
        <th>Price</th>
    </tr>
    {% for product in products %}
    <tr>
        <td>{{product.name}}</td>
        <td>{{product.price}}</td>
    </tr>

    {% endfor %}

</table>

<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>

{% endblock %}
```

Run the following commands in terminal

```
     python manage.py makemigrations
```

```
python manage.py migrate

python manage.py runserver
```

Now go to admin page using below link and add customers, products. Add tags and map tags to each product

    https://127.0.0.1:8000/admin

Run the following command and check the result in dashboard and product links.

```
python manage.py runserver
```

## Experiment 9 : Create Tables with ForeignKey through Model Classes in Django

Create a orders table in **models.py**

```
class Order(models.Model):

    STATUS=(
        ('Pending','Pending'),
        ('Out of Delivery','Out of Delivery'),
        ('Delivered','Delivered')
        )

    customer=models.ForeignKey(Customer, null=True, on_delete=models.SET_NULL)
    product=models.ForeignKey(Product, null=True, on_delete=models.SET_NULL)
    status=models.CharField(max_length=300, choices=STATUS)
```

In **admin.py** place the below line:

```
     admin.site.register(Tag)
```

Perform make migrations and migrate command in terminal as discussed earlier and go to admin page and place orders.

In **urls.py** add the below path to url_patterns list

```
path('customer/<str:pk_test>/',views.customer, name='customer'),
```

In **views.py** create **customer** method

```
def customer(request, pk_test):
    customer=Customer.objects.get(id=pk_test)
    customers=Customer.objects.all()
    orders=customer.order_set.all()
    order_count=orders.count()
    context={'customers':customers, 'cust':customer,'orders':orders,'ordcount':order_count}
    return render(request,'customer.html',context)
```

In **customer.html** include the following

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

{% block content %}

<h1>Customer Name: {{ cust}}</h1>


<h1>Orders</h1>
```

```
&nbsp;

<table class="table">

    {% for i in orders %}
    <tr>
        <td>{{i.product}}</td>
    </tr>

    {% endfor %}

</table>

<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>

{% endblock %}
```

In **dashboard.html** include the following inside table

```
<th>URL</th>

<td><a href="{% url 'customer' customer.id %}">View</a></td>
```

```
Run the following command and check the result:

     python manage.py runserver
```

## Experiment 10 : CRUD Operations in Django - Create Orders using Model Forms

Create a new file named **forms.py** in **calc** app folder. Import ModelForm from django.forms. Create a class named OrderForm inheriting ModelForm.

```
from django.forms import ModelForm

from .models import Order

class OrderForm(ModelForm):
    class Meta:
        model=Order
        fields="__all__"
```

```
In urls.py add the following path
```

```
    path(‘create_order/',views.createOrder, name='create_order'),
```

Create a method named createOrder

```python
def createOrder(request):
    form=OrderForm()

    if request.method=="POST":
        form=OrderForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('/')

    context={'form':form}

    return render(request,'order_form.html',context)
```

Create a html file **order_form.html** with the below content.

```html
<form action="" method="POST">

    {% csrf_token %}

    {{form}}

    <input type="submit" name="submit">

</form>
```

In **dashboard.html** you can place below reference to create order page

```html
<a href="{% url 'create_order' %}"><button type="button" class="btn btn-success">Create Order</
button></a>
```

## Experiment 11 : CRUD Operations in Django - Update Orders using Model Forms

Create a method named updateOrder in **views.py**

```
def updateOrder(request, pk):

    order=Order.objects.get(id=pk)
    form=OrderForm(instance=order)
    if request.method=="POST":
        form=OrderForm(request.POST, instance=order)
        if form.is_valid():
            form.save()
            return redirect('/')
    context={'form':form}

    return render(request,'order_form.html',context)
```

In **urls.py** add the following path

```
     path(‘update_order/<str:pk>/',views.updateOrder, name=‘update_order'),
```

Modify dashboard method as follows

```
def dashboard(request):
    customers=Customer.objects.all()
    orders=Order.objects.all()
    return render(request,'dashboard.html',{'customers':customers,'orders':orders})
```

Append the below code at the bottom of **dashboard.html**

```
<table class="table">
    <tr>
        <th>Update</th>
        <th>Delete</th>
        <th>Name</th>
        <th>Product</th>
        <th>Status</th>
    </tr>
    {% for order in orders %}
    <tr>
        <td><a href="{% url 'update_order' order.id %}"><button type="button" class="btn btn-
success">Update Order</button></a></td>
```

```
        <td>{{order.customer}}</td>
        <td>{{order.product}}</td>
        <td>{{order.status}}</td>
    </tr>

    {% endfor %}

</table>
```

Run the following command and check the result:

```
     python manage.py runserver
```

## Experiment 12 : CRUD Operations in Django - Delete Orders using Model Forms

In **urls.py** add the following path

```
    path('delete_order/<str:pk>/',views.deleteOrder, name='delete_order')
```

Add deleteOrder method in **views.py**

```
def deleteOrder(request, pk):

    order=Order.objects.get(id=pk)

    if request.method=="POST":
        order.delete()
        return redirect('/')

    context={'item':order}

    return render(request,'delete.html',context)
```

Create **delete.html** with below content

```
<p> Are you sure that you want to delete the order </p>

<form action="{% url 'delete_order' item.id %}" method="POST">
    {% csrf_token %}

    <a href="{% url 'home' %}"> Cancel </a>

    <input type="submit" name="Confirm">

</form>
```

Add the following rows in **dashboard.html** file

```
<th>Delete</th>

<td><a href="{% url 'delete_order' order.id %}"><button type="button" class="btn btn-success">Delete 
Order</button></a></td>
```

## Experiment 13 : Django Authentication : Setting up Registration Page

Add the following path to **urls.py**

```
path(‘register/',views.registerPage, name='register'),
```

In **views.py** add the following method

```
from django.contrib.auth.forms import UserCreationForm


def registerPage(request):
    form=UserCreationForm()
    if request.method=="POST":
        form=UserCreationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
        else:
            messages.success(request,"Password does not follow the rules")

    context={'form':form}

    return render(request, 'register.html', context)
```

Create a **register.html** file in **templates** folder

```
<form method="POST" action="">
    {% csrf_token %}
     {{form}}
     <input type="submit" name="Create User">

</form>
```

Run the following command and check the result:

```
     python manage.py runserver
```

## Experiment 14 : Django Authentication : Login and Logout Pages

Import the following packages

```
from django.contrib.auth import authenticate, login, logout

from django.contrib import messages

from django.contrib.auth.decorators import login_required

from django.views.decorators.cache import cache_control
```

In **urls.py** add the following

```
path(‘login/',views.loginPage, name='login'),
path(‘logout/',views.logoutPage, name=‘logout'),
```

Place the below two decorators up on every method in **views.py**. First one restricts unauthenticated users (those who are not logged in) from accessing pages. Second decorator prevents from accessing past pages when a back button is pressed. Except loginPage and register Page method, place it above every method in views.py. Refer to the last chapter for complete code.

```
@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
```

In **views.py** add the following loginPage method.

```
def loginPage(request):
    if request.user.is_authenticated:
        return redirect('home')
    else:
        if request.method=='POST':
            username=request.POST.get('username')
            password=request.POST.get('password')
            print(username, password)

            user = authenticate(request, username=username, password=password)

            if user is not None:
                login(request, user)
                return redirect('home')
            else:
                messages.success(request,"Username or Password is incorrect")

        context={}
        return render(request,'login.html',context)
```

Add the following content in **login.html**

```
<h1>Login</h1>

<form method="POST" action="">
    {% csrf_token %}

    Username: <input type="text" name="username"><br>
    Password: <input type="password" name="password"><br>

    <input type="submit" name="Login">
</form>

{% for message in messages %}

<p id="messages">{{message}} </p>

{% endfor %}

<a href="{% url 'register' %}"><button type="button" class="btn btn-success”>Register</button></a>
```

Create a logout page method in **views.py** as follows

```
@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def logoutPage(request):
    logout(request)
    return redirect('login')
```

Run the following command and check the result:

```
python manage.py runserver
```

## Experiment 15 : Django Authentication : Creating Customised Registration Pages

Import the following package:

```
from django.contrib import messages
```

In **forms.py** add the following:

```
from django.contrib.auth.forms import UserCreationForm
from django import forms
from django.contrib.auth.models import User

class CreateUserForm(UserCreationForm):
    class Meta:
        model=User
        fields=[“username","email","password1","password2"]
```

In **views.py** add the following

```
from .forms import CreateUserForm
```

Replace UserCreationForm to CreateUserForm in registerPage

```
def registerPage(request):
    form=CreateUserForm()
    if request.method=="POST":
        form=CreateUserForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
        else:
            messages.success(request,"Password does not follow the rules")
    context={'form':form}
    return render(request, 'register.html', context)
```

In **register.html** add following

```
<h1>Register</h1>

<form method="POST" action="">
    {% csrf_token %}

    {{form.username.label}}
    {{form.username}}
    <br>
    <br>
    {{form.email.label}}
    {{form.email}}
    <br>
    <br>
    {{form.password1.label}}
    {{form.password1}}
    <br>
    <br>
    {{form.password2.label}}
    {{form.password2}}
    <br>
    <br>

    <input type="submit" name="Create User">

</form>

{% for message in messages %}

<p id="messages">{{message}} </p>

{% endfor %}
```

Run the following command and check the result:

```
     python manage.py runserver
```

## Experiment 16 : Complete Django Application Codes

1. Final code in **admin.py** is as follows

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

admin.site.register(Customer)
admin.site.register(Product)
admin.site.register(Tag)
admin.site.register(Order)
```

2. Final code in **apps.py** is as follows

```
from django.apps import AppConfig

class CalcConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'calc'
```

3. Final code in **forms.py** is as follows

```
from django.forms import ModelForm

from django.contrib.auth.forms import UserCreationForm
from django import forms
from django.contrib.auth.models import User

from .models import Order

class OrderForm(ModelForm):
    class Meta:
        model=Order
        fields="__all__"

class CreateUserForm(UserCreationForm):
    class Meta:
        model=User
        fields=[“username”,”email","password1","password2"]
```

4. Final code in **models.py** is as follows

```
from django.db import models

class Customer(models.Model):
    name=models.CharField(max_length=300,null=True)
    age=models.CharField(max_length=300,null=True)
    date=models.DateTimeField(max_length=300,null=True)

    def __str__(self):
        return self.name

class Tag(models.Model):
    name=models.CharField(max_length=300,null=True)

    def __str__(self):
        return self.name

class Product(models.Model):

    CATEGORY=(
        ('Indoor','Indoor'),
        ('Outdoor','Outdoor')
        )

    name=models.CharField(max_length=300,null=True)
    price=models.FloatField(max_length=300,null=True)
    category=models.CharField(max_length=300,choices=CATEGORY)
    tags=models.ManyToManyField(Tag)

    def __str__(self):
        return self.name


class Order(models.Model):

    STATUS=(
        ('Pending','Pending'),
        ('Out of Delivery','Out of Delivery'),
        ('Delivered','Delivered')
        )

    customer=models.ForeignKey(Customer, null=True, on_delete=models.SET_NULL)
    product=models.ForeignKey(Product, null=True, on_delete=models.SET_NULL)
    status=models.CharField(max_length=300, choices=STATUS)
```

5. Final code in **urls.py** is as follows

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

urlpatterns=[
    path(‘register/',views.registerPage, name='register'),
    path(‘login/',views.loginPage, name='login'),
    path(‘logout/',views.logoutPage, name='logout'),
    path(‘',views.home, name='home'),
    path(‘add',views.add, name='add'),
    path(‘dashboard/',views.dashboard, name='dashboard'),
    path(‘products/',views.products, name='products'),
    path(‘customer/<str:pk_test>/',views.customer, name='customer'),
    path(‘create_order/',views.createOrder, name='create_order'),
    path(‘update_order/<str:pk>/',views.updateOrder, name='update_order'),
    path(‘delete_order/<str:pk>/',views.deleteOrder, name='delete_order')

]
```

6. Final code in **views.py** is as follows

```
from django.shortcuts import render, redirect
from .models import *
from .forms import OrderForm, CreateUserForm
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth import authenticate, login, logout
from django.contrib import messages
from django.contrib.auth.decorators import login_required
from django.views.decorators.cache import cache_control

@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def home(request):
    return render(request,’home.html',{'name':'Raju'})

@login_required(login_url='login')
def add(request):
    val1=int(request.POST['num1'])
    val2=int(request.POST['num2'])
    val3=val1+val2
    return render(request,'result.html',{'result':val3})

@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def dashboard(request):

    customers=Customer.objects.all()
    orders=Order.objects.all()
    return render(request,'dashboard.html',{'customers':customers,'orders':orders})

@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def products(request):
    products=Product.objects.all()
    return render(request,'product.html',{'products':products})

@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def customer(request,pk_test):
    customer=Customer.objects.get(id=pk_test)
    customers=Customer.objects.all()
    orders=customer.order_set.all()
    order_count=orders.count()

    context={'customers':customers, 'cust':customer,'orders':orders,'ordcount':order_count}

    return render(request,'customer.html',context)

@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def createOrder(request):
    form=OrderForm()

    if request.method=="POST":
        form=OrderForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('/')

    context={'form':form}
    return render(request,'order_form.html',context)
```

```
@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def updateOrder(request,pk):

    order=Order.objects.get(id=pk)
    form=OrderForm(instance=order)

    if request.method=="POST":
        form=OrderForm(request.POST,instance=order)
        if form.is_valid():
            form.save()
            return redirect('/')

    context={'form':form}

    return render(request,'order_form.html',context)

@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def deleteOrder(request,pk):

    order=Order.objects.get(id=pk)

    if request.method=="POST":
        order.delete()
        return redirect('/')

    context={'item':order}

    return render(request,'delete.html',context)
```

```
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def registerPage(request):
    form=CreateUserForm()
    if request.method=="POST":
        form=CreateUserForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
        else:
            messages.success(request,"Password does not follow the rules")


    context={'form':form}

    return render(request, 'register.html', context)

@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def loginPage(request):
    if request.user.is_authenticated:
        return redirect('home')
    else:
        if request.method=='POST':
            username=request.POST.get('username')
            password=request.POST.get('password')
            print(username,password)

            user = authenticate(request,username=username, password=password)

            if user is not None:
                login(request, user)
                return redirect(‘home')

            else:
                messages.success(request,"Username or Password is incorrect")

        context={}
        return render(request,'login.html',context)

@login_required(login_url='login')
@cache_control(no_cache=True, must_revalidate=True, no_store=True)
def logoutPage(request):
    logout(request)
    return redirect('login')
```

In the project folder following are the final files.

7. The content of **asgi.py** is as follows

```
import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'MyDjangoPractise.settings')

application = get_asgi_application()
```

8. Final code in **settings.py** is as follows

```
from pathlib import Path
import os

BASE_DIR = Path(__file__).resolve().parent.parent


SECRET_KEY = 'django-insecure-d4lp3__le-&6jg+ol!9qjqnkw64%pf18k_#w3jjuk5+hd1@(5$'

DEBUG = True

ALLOWED_HOSTS = ['MyDjangoPractise-5.onrender.com','127.0.0.1']

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

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'MyDjangoPractise.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'MyDjangoPractise.wsgi.application'


DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


STATIC_URL = ‘static/'

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

```
DEFAULT_AUTO_FIELD = ‘django.db.models.BigAutoField'
```

9. Final code in **urls.py** is as follows

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

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

10. Final code in **wsgi.py** is as follows

```
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'MyDjangoPractise.settings')

application = get_wsgi_application()
```

In template folders following html files are to be present.

11. Final code in **base.html** is as follows

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
crossorigin="anonymous">
</head>

<body style="background-color: powderblue;">

    {% include 'navbar.html' %}

    {% block content %}

    {% endblock %}

</body>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/
X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
```

```
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.14.7/dist/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin=“anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.3.1/dist/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>

</html>
```

12. Final code in **home.html** is as follows

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

    {% extends 'base.html' %}

    {% block content %}

<h1> Hello {{name}} </h1>

<form action="add" method="post">
    {% csrf_token %}
    Enter First Number: <input type="text" name="num1"><br>
    Enter Second Number: <input type="text" name="num2"><br>

    <input type="submit">

</form>
<br><br>

<td><a href="{% url 'register' %}"><button type="button" class="btn btn-success">Register</button></
a></td>

<a href="{% url 'logout' %}"><button type="button" class="btn btn-success">Logout</button></a>

{% endblock %}

</body>
</html>
```

13. Final code in **result.html** is as follows

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

{% block content %}


<h1>Result: {{result}} </h1>

<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>

{% endblock %}
```

14. Final code in **navbar.html** is as follows

```
{% load static %}

<nav class="navbar navbar-expand-lg navbar-light bg-light”>
     <img src="{% static 'Images/klhlogo.png' %}"
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarNavDropdown" aria-controls="navbarNavDropdown" aria-expanded="false" aria-
label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNavDropdown">
      <ul class="navbar-nav">
        <li class="nav-item active">
          <a class="nav-link" href="{% url 'dashboard' %}">Dashboard</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="{% url 'products' %}">Products</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Pricing</a>
        </li>
        <li class="nav-item dropdown">
```

```
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdownMenuLink" data-
toggle="dropdown" aria-haspopup="true" aria-expanded="false">
            Dropdown link
          </a>
          <div class="dropdown-menu" aria-labelledby="navbarDropdownMenuLink">
            <a class="dropdown-item" href="#">Action</a>
            <a class="dropdown-item" href="#">Another action</a>
            <a class="dropdown-item" href="#">Something else here</a>
          </div>
        </li>
      </ul>
    </div>
  </nav>
```

15. Final code in **customer.html** is as follows

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

{% block content %}

<h1>Customer Name: {{ cust}}</h1>


<h1>Orders</h1>

&nbsp;
```

```
<table class="table">

    {% for i in orders %}
    <tr>
        <td>{{i.product}}</td>
    </tr>

    {% endfor %}

</table>

<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>

{% endblock %}
```

16. Final code in **product.html** is as follows

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

{% block content %}

<h1>Products</h1>

&nbsp;

<table class="table">
    <tr>
        <th>Name</th>
        <th>Price</th>
    </tr>
    {% for product in products %}
    <tr>
        <td>{{product.name}}</td>
        <td>{{product.price}}</td>
    </tr>

    {% endfor %}

</table>

<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>

{% endblock %}
```

17. Final code in **order_form.html** is as follows

```
<form action="" method="POST">
    {% csrf_token %}
    {{form}}

    <input type="submit" name="submit">

</form>
```

18. Final code in **dashboard.html** is as follows

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

{% block content %}

<h1>Dashboard</h1>

&nbsp;

<table class="table">
    <tr>
        <th>URL</th>
        <th>Name</th>
        <th>Age</th>
    </tr>
    {% for customer in customers %}
    <tr>
        <td><a href="{% url 'customer' customer.id %}">View</a></td>
        <td>{{customer.name}}</td>
        <td>{{customer.age}}</td>
    </tr>

    {% endfor %}

</table>

<a href="{% url 'create_order' %}"><button type="button" class="btn btn-success">Create Order</
button></a>
```

```
<a href="{% url 'home' %}"><button type="button" class="btn btn-success">Home Window</button></a>

<table class="table">
    <tr>
        <th>Update</th>
        <th>Delete</th>
        <th>Name</th>
        <th>Product</th>
        <th>Status</th>
    </tr>
    {% for order in orders %}
    <tr>
        <td><a href="{% url 'update_order' order.id %}"><button type="button" class="btn btn-
success">Update Order</button></a></td>
        <td><a href="{% url 'delete_order' order.id %}"><button type="button" class="btn btn-
success">Delete Order</button></a></td>
        <td>{{order.customer}}</td>
        <td>{{order.product}}</td>
        <td>{{order.status}}</td>
    </tr>

    {% endfor %}

</table>
{% endblock %}
```

19. Final code in **delete.html** is as follows

```
<p> Are you sure that you want to delete the order </p>

<form action="{% url 'delete_order' item.id %}" method="POST">
    {% csrf_token %}

    <a href="{% url 'home' %}"> Cancel </a>

    <input type="submit" name="Confirm">

</form>
```

20. Final code in **register.html** is as follows

```
<h1>Register</h1>

<form method="POST" action="">
    {% csrf_token %}

    {{form.username.label}}
    {{form.username}}

    <br>
    <br>

    {{form.email.label}}
    {{form.email}}
    <br>
    <br>

    {{form.password1.label}}
    {{form.password1}}

    <br>
    <br>

    {{form.password2.label}}
    {{form.password2}}

    <br>
    <br>

    <input type="submit" name="Create User">

</form>

{% for message in messages %}

<p id="messages">{{message}} </p>

{% endfor %}

<a href="{% url 'login' %}"><button type="button" class="btn btn-success">Sign Up</button></a>
```

21. Final code in **login.html** is as follows

```
<h1>Login</h1>
```

```
<form method="POST" action="">
    {% csrf_token %}

    Username: <input type="text" name="username"><br>
    Password: <input type="password" name="password"><br>

    <input type="submit" name="Login">
</form>

{% for message in messages %}

<p id="messages">{{message}} </p>

{% endfor %}


<a href="{% url 'register' %}"><button type="button" class="btn btn-success">Register</button></a>
```

Run the following command and check the result:

```
python manage.py runserver
```

## Experiment 17 : Display tables in Django using Visual Studio extension for sqlite

By default DATABASES variable in **settings.py** is as follows.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Tables created using classes in **models.py** can be viewed in sqlite by installing following plugins in visual studio code: **SQLite** by Alex Covizzi, or **SQLite Viewer** by Florian Klampfer. We will use the first extension.

In Visual studio terminal you can run following commands to enter entries in to the table

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

```
>> from myDBApp.models import Customer
>> Customer.objects.create(name='Raju', age=50)
```

Run the following command to migrate and see the output as tables.

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

In the EXPLORER panel go to SQLITE EXPLORER and click on myDBApp_customer to view the table. (Note: my app name used here is myDBApp)

## Experiment 18 : Django connection to MYSQL Database

**Connecting to MYSQL after installation:**

**First approach:**

From your terminal or command prompt enter MYSQL using below command:

```
mysql -u root -p
```

Enter the password.

**Second Approach:**

In Windows, you can enter through MYSQL shell

Switch the MySQL shell from JS to SQL mode using the command below;

```
\sql
```

Connect to the server using the MySQL user and hostname specified during the installation process. Use the command below.

```
\connect root@localhost
```

If you want to create a database, the command to use is as below.

```
create database MYDB;
```

To connect your Django project to MYSQL, default DATABASES variable in **settings.py** has to be changed as follows:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'MYDB',
        'HOST': '127.0.0.1',
        'PORT': '3306',
        'USER': 'root',
        'PASSWORD': ‘******’,

    }
}
```

Also we need to install mysqlclient as follows:

```
pip install mysqlclient
```

Also import following packages in **settings.py**

1. `import pymysql`
2. `pymysql.install_as_MySQLdb()`

To use pymysql we need to install as follows in terminal

```
pip install pymysql
```

On Mac you may need to install mysql client using following command

```
brew install mysql-client
```

In Visual studio terminal you can run following commands to enter entries in to the table

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

```
python manage.py shell
```

```
>> from myDBApp.models import Customer
>> Customer.objects.create(name='Raju', age=50)
```

Now migrate and use runserver as follows

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

## Experiment 19 : Data Collection using HTML Forms, Storage in MYSQL Database and Render it

Following methods should be present in **views.py**

```
from django.shortcuts import render

from .models import *

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

def insertData(request):
    n=request.POST['nameVar']
    a=request.POST['ageVar']
    Customer.objects.create(name=n, age=a);
    customers=Customer.objects.all()
    return render(request,’home.html',{'customers':customers})
```

**urls.py** contain following data.

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

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

**models.py** contain following data.

```
from django.db import models

class Customer(models.Model):
    name=models.CharField(max_length=100, null=True)
    age=models.CharField(max_length=200, null=True)

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

**admin.py** contain following data.

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

DATABASES variable in **settings.py** has to be as follows (Note: values should be changed as per your database)

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'MYDB',
        'HOST': '127.0.0.1',
        'PORT': '3306',
        'USER': 'root',
        'PASSWORD': '*****',
    }
}
```

**home.html** contains rendering of all entries in table and also to collect data from user.

```
<table class="table">
    <tr>
        <th>URL</th>
        <th>Name</th>
        <th>Age</th>
</tr>
    {% for customer in customers %}
    <tr>
        <td>{{customer.name}}</td>
        <td>{{customer.age}}</td>
</tr>
    {% endfor %}
</table>

<form action="insertData" method="post">
    {% csrf_token %}
    name: <input type="text" name="nameVar"><br>
    age: <input type="text" name="ageVar"><br>

    <input type="submit">
</form>
```

## Experiment 20 : Import data from CSV file in to MYSQL/sqlite Database

Step 1: Prepare Your Django Model in **models.py** file

```
class Customer(models.Model):
    name=models.CharField(max_length=100, null=True)
    age=models.CharField(max_length=200, null=True)

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

Step 2: Create a CSV File say **customers.csv**. with customer details in the columns as per the above class definition. For example above class contains two attributes: name and age. So there should be two columns, one for name and other for age. Each row is an entry for a different customer.

Step 3: Write a Custom Django Management Command to Import the CSV

**Create a Management Command:** Django provides a way to create custom management commands. Create a directory management/commands inside your app folder if it doesn't exist yet.

```
myDBApp/ management/ commands/ import_customers.py
```

`import_customers.py` file contains following code:

```python
import csv
from django.core.management.base import BaseCommand
from myDBApp.models import Customer

class Command(BaseCommand):
    help = 'Import student data from a CSV file'

    def add_arguments(self, parser):
        parser.add_argument('csv_file', type=str, help='The path to the CSV file')

    def handle(self, *args, **kwargs):
        csv_file = kwargs['csv_file']

        # Open the CSV file
        with open(csv_file, 'r') as file:
            reader = csv.DictReader(file)

            # Iterate over each row in the CSV
            for row in reader:
                # Create a new Customer object and save it to the database
                Customer.objects.create(
                    name=row['name'],
                    age=row['age']
                )
        self.stdout.write(self.style.SUCCESS('Successfully imported student data'))
```

Step 4: Run the Custom Command using following command in your terminal

```
python manage.py import_customers customers.csv
```

Now migrate and use runserver as follows

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

If the DATABASE variable in settings.py is set to MYSQL attributes, check whether csv data has been loaded in to MYSQL table. If it is sqlite, check for the table in SQLITE EXPLORER in visual studio code.

## References:

https://www.youtube.com/playlist?list=PL-51WBLyFTg2vW-_6XBoUpE7vpmoR3ztO

https://www.youtube.com/playlist?list=PLKolgtjvFc5cpy7lwd6T-N2oMsnWlADg5

https://docs.djangoproject.com/en/5.0/

https://www.youtube.com/playlist?list=PLsyeobzWxl7r2ukVgTqIQcl-1T0C2mzau