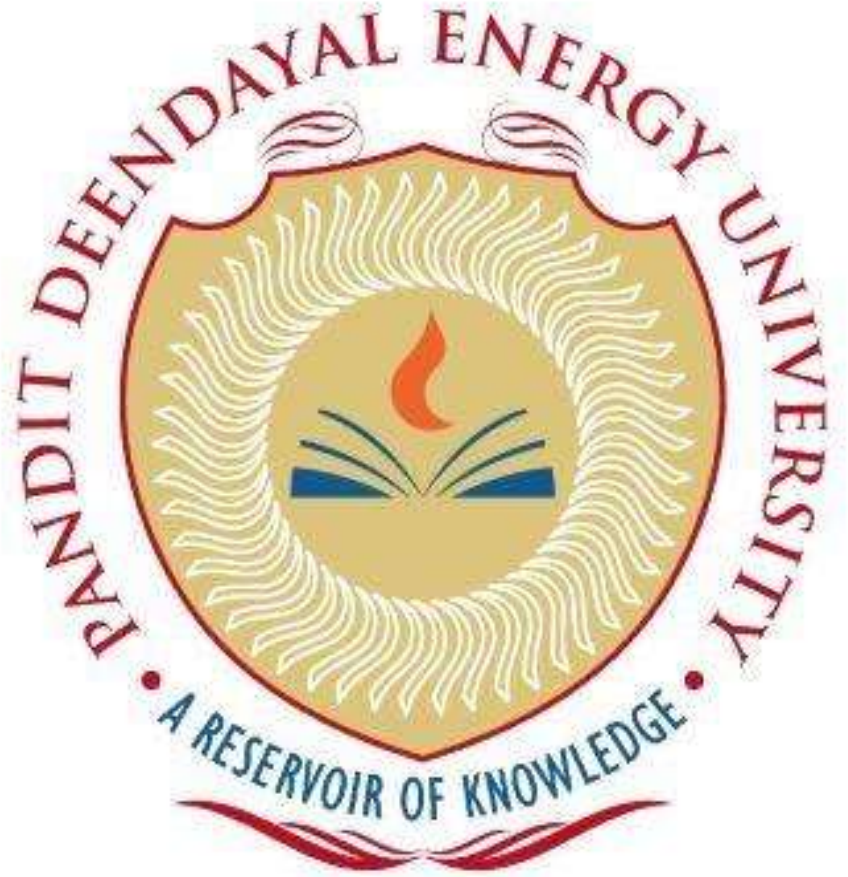
Pandit Deendayal Energy University, Gandhinagar

School of Technology



Department of Computer Science & Engineering

Advance Web Technology Lab (23CP308P)

B.Tech-Computer Science & Engineering (Sem-VI)

Name:Darshan Devani

Roll No: 21BCP180

Div-3/G6

Practical-11

Assignment: Building a Simple Django Web Application

Aim:

To create a simple Django web application that displays "Hello World!" on a web page.

Software and Hardware Requirements:

- **Software:**
  - Python (preferably version 3.9 or later)
  - Django framework (installed using pip)
  - Code editor (e.g., VSCode, PyCharm, Sublime Text)
- **Hardware:**
  - Computer system (Windows/Mac/Linux)
  - Internet connection for downloading required software

Views in Django:

In the context of Django, **views** represent the logic behind web applications. They receive HTTP requests from clients (browsers) and return HTTP responses. Views handle user interactions, process data, and determine what content to present to the user. Django supports both function-based views (FBVs) and class-based views (CBVs). Views are responsible for interacting with models to retrieve or manipulate data and render templates to generate HTML responses.

Templates in Django:

Templates in Django are used to generate HTML dynamically based on data provided by views. They separate the presentation layer (HTML/CSS) from the business logic (views). Templates contain placeholders (template variables) and logic (template tags) to dynamically render content. Template variables are replaced with actual data when rendering the template, and template tags allow for conditional logic, loops, and other control structures within templates. Django uses its template engine to parse templates and generate the final HTML sent to clients.

URLs in Django:

URLs in Django serve as a mapping between URL patterns and views. When a client sends an HTTP request to a Django application, Django's URL resolver matches the URL to a corresponding view based on the defined URL patterns. URL patterns are specified in the `urls.py` files of Django projects and apps. Django's URL routing mechanism allows

developers to create clean and meaningful URLs for different parts of the application, improving readability and maintainability.

Model-View-Template (MVT) Architecture:

Django follows the Model-View-Template (MVT) architectural pattern, which is similar to the more common Model-View-Controller (MVC) pattern. In MVT:

- **Model:** Represents the data structure of the application, typically defined using Django's ORM (Object-Relational Mapping) to interact with databases.
- **View:** Handles user requests, processes data, and determines the appropriate response. Views interact with models to access or modify data and render templates to generate HTML responses.
- **Template:** Contains the presentation logic and structure of web pages. Templates use template variables and tags to dynamically render content based on data provided by views.

CODE

Creating a simple Todo app in Django involves setting up models, views, templates, and URLs to manage tasks. Here's a basic example to get you started:

1. Define Models (`models.py`):

python

```
# In your app's models.py file
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=255)
    description = models.TextField(blank=True)
    completed = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

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

2. Create Views (`views.py`):

python

```
# In your app's views.py file
from django.shortcuts import render, redirect
```

```
from .models import Task
from .forms import TaskForm

def task_list(request):
    tasks = Task.objects.all()
    return render(request, 'task_list.html', {'tasks': tasks})

def add_task(request):
    if request.method == 'POST':
        form = TaskForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('task_list')
    else:
        form = TaskForm()
    return render(request, 'add_task.html', {'form': form})
```

3. Create Forms (`forms.py`):

python

```
# In your app's forms.py file
from django import forms
from .models import Task

class TaskForm(forms.ModelForm):
    class Meta:
        model = Task
        fields = ['title', 'description', 'completed']
```

4. Create Templates (`templates` folder):

`task_list.html`:

html

```
<!-- In your app's templates folder -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Task List</title>
</head>
<body>
    <h1>Task List</h1>
    <ul>
        {% for task in tasks %}
            <li>{{ task.title }}</li>
        {% endfor %}
    </ul>
    <a href="{% url 'add_task' %}">Add Task</a>
</body>
</html>
```

add_task.html:

html

```
<!-- In your app's templates folder -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Add Task</title>
</head>
<body>
    <h1>Add Task</h1>
    <form method="POST">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Save Task</button>
    </form>
    <a href="{% url 'task_list' %}">Back to Task List</a>
</body>
</html>
```

5. Configure URLs (`urls.py`):

python

```
# In your app's urls.py file
from django.urls import path
from . import views

urlpatterns = [
    path('', views.task_list, name='task_list'),
    path('add/', views.add_task, name='add_task'),
]
```