

**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING
VIZIANAGARAM ANDHRA PRADESH**



DJANGO FRAME WORK LAB

M. J. FRANKLIN	(23VV1A1231)
P. SAI VAMSI	(23VV1A1238)
D. H. V. VINAY	(24VV5A1269)
CH.MOUNIKA	(23VV1A1209)

**UNDER GUIDANCE OF
MRS.MADHUMITA CHANDA**

**DEPARTMENT OF INFORMATION
TECHNOLOGY**



**JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Regd.No : 23VV1A1231

CERTIFICATE

This is to certify that this is a bonafide record of practical work done by MR. M.J.FRANKLIN of IInd B.Tech IInd Semester Class in DJANGO FRAME WORK Lab during the year 2024-25.

No.of Tasks Completed and Certified:12

Lecture In-Charge

Head of The Department

Date:



DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM

Website: www.jntugvcev.edu.in

Subject Name: DJANGO FRAMEWORK

Subject Code: R232212SE01

Regulation: R23

Year: 2025

COURSE OUTCOMES

NBA Subject Code	Course Outcomes	
	CO1	Design and build static as well as dynamic web pages and interactive web-based applications .
	CO2	Web development using Django framework.
	CO3	Analyze and create functional website in Django and deploy Django Web Application on Cloud .

CO-PO Mapping

Mapping of Course Outcomes (COs) with Program Outcomes (POs)

Course Outcomes		Program Outcomes (POs)														
		PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PS O 1	PS O 2	PS O 3
	CO1	3	1	3	1	3	1	1	1	2	3	2	1	3	3	2
	CO2	3	2	3	1	3	1	1	1	2	2	2	2	3	3	3
	CO3	2	3	3	3	3	2	2	2	2	3	3	3	3	3	3

Enter correlation levels 1,2 and 3 as defined below:

1: Slight (Low) 2: Moderate (Medium) 3: Substantial (High) If there is no correlation, put “-”

Signature of course instructor

Table of contents

SNO	DATE	CONTENT	PAGE	MARKS	REMARKS
1	13-12-2024	Understanding Django and its Libraries	5-20		
2	20-12-2024	Introduction to Django Framework	21-22		
3	27-12-2024	Step-by-Step Guide to Installation of Django	23-24		
4	03-01-2025	Linking Views and URL Configurations	25-26		
5	24-01-2025	Exploring Django Views	27-33		
6	24-01-2025	Setting Up App-level URLs	34-36		
7	31-01-2025	Working with Templates in Django	37-82		
8	21-02-2025	Database Integration and Configuration SQLITE	83-85		
9	21-02-2025	Handling Forms in Django	86-87		
10	21-02-2025	Defining and Using Models	88-89		
11	07-03-2025	Migration Sync with the Database	90-91		
12	27-03-2025	Deploying Django Application on cloud platforms	92-93		
13	04-04-2025	Front End Web Developer Certification	94-95		

Date :

signature:



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.[it@intugvcev.edu.in](mailto:hod.it@intugvcev.edu.in)

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Understanding Django and its Libraries
7. Date of Experiment : 13-12-2024
8. Date of Submission of Report : 20-12-2024

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

LIBRARIES

1 PYTHON LIBRARIES

2 Python Collections - Container Datatypes:

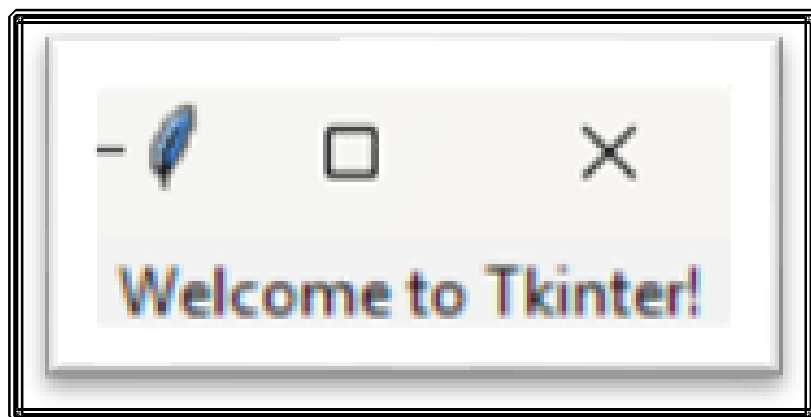
- Purpose: Provides specialized container datatypes that support efficient handling of data.
- **Key Types:**
 - List: Ordered, mutable, allows duplicates.
 - Tuple: Ordered, immutable, allows duplicates.
 - Set: Unordered, no duplicates, fast membership testing.
 - Dictionary: Unordered, key-value pairs, fast lookups.
- Common Use: Data manipulation, storing and accessing collections of data in web apps (like user data or API responses).

3 Tkinter

- 3.1 Purpose: Python's standard library for creating graphical user interfaces (GUIs).
- 3.2 Keyfeatures:
 - 3.2.1 Widgets: Buttons, labels, text boxes, etc.
 - 3.2.2 Event handling: Respond to user interactions like clicks or key presses.
 - 3.2.3 Simple layout management.

Code:

```
from tkinter import Tk, Label  
  
# Create a window  
  
root = Tk()  
  
root.title('Hello Window')  
  
# Add a label to display text  
  
Label(root, text='Welcome to Tkinter!').pack()  
  
# Run the application  
  
root.mainloop()
```

Output :

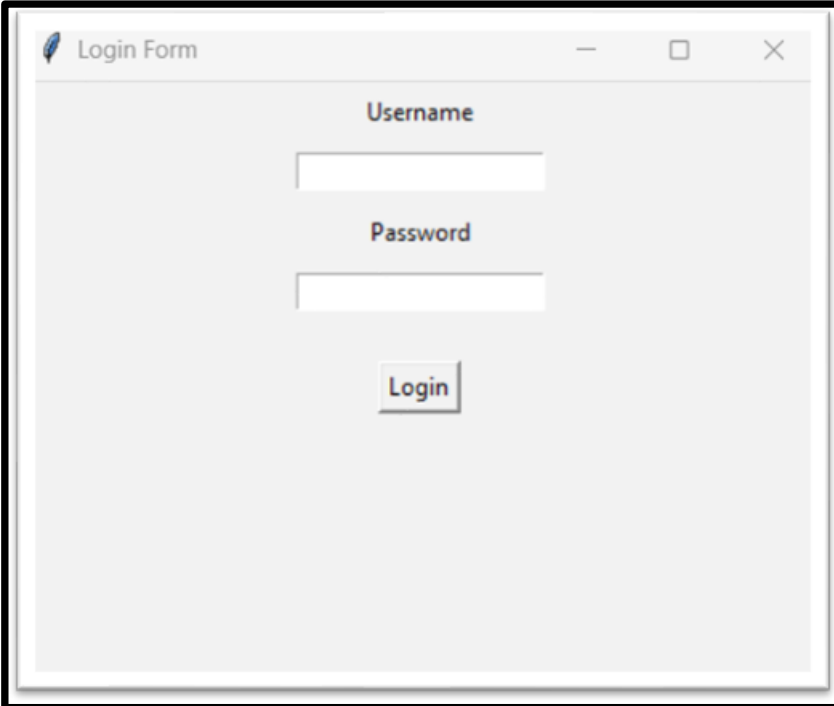
4 Requests - HTTP Requests:

- Purpose: Simplifies HTTP requests to interact with web APIs.
- Key Features:
 - Send GET, POST, PUT, DELETE requests easily.
 - Handle request parameters, headers, and cookies.
 - Simple error handling and response handling.
- Common Use: Interact with REST APIs, download content from the web.

Code:

```
from tkinter import Tk, Label, Entry, Button
def login():
    username = username_entry.get()
    password = password_entry.get()
    print(f'Username: {username}, Password: {password}') # Placeholder for real login logic
# Create main window
root = Tk()
root.title("Login Form")
root.geometry("300x200") # Set size of the window
# Username Label and Entry
Label(root, text="Username", font=('Arial', 10, 'bold')).pack(pady=(10, 0))
username_entry = Entry(root, width=30)
username_entry.pack(pady=(5, 10))
# Password Label and Entry
Label(root, text="Password", font=('Arial', 10, 'bold')).pack()
password_entry = Entry(root, show="*", width=30)
password_entry.pack(pady=(5, 10))
# Login Button
Button(root, text="Login", width=10, command=login).pack(pady=10)
# Run the application
root.mainloop()
```


Output:



5 Scrapy:

- a. Purpose: An open-source web crawling framework for large-scale web scraping.
- b. Key Features:
 - i. Fast, extensible, and asynchronous web scraping.
 - ii. Supports handling requests, data extraction, and storing results.
 - iii. Built-in handling for logging, retries, and sessions.
- c. Common Use: Web crawling and scraping projects that require high performance.

6 **BeautifulSoup4 - Web Scraping:**

- a) Purpose: Parses HTML and XML documents to extract data.
- b) Key Features:
 - c) Easy navigation and searching within HTML.
 - d) Supports different parsers like html.parser, lxml, and html5lib.
- e) Common Use: Extract data from websites for analysis, e.g., for building data-driven applications

Code:

```

import requests

from bs4 import BeautifulSoup

def scrape_quotes():

    base_url = "http://quotes.toscrape.com"

    next_page = "/"

    while next_page:

        response = requests.get(base_url + next_page)

        if response.status_code == 200:

            soup = BeautifulSoup(response.text, "html.parser")

            quotes = soup.find_all("span", class_='text')

            authors = soup.find_all("small", class_='author')

            for quote, author in zip(quotes, authors):

                print(f'{quote.text}' - {author.text}\n')

            next_btn = soup.find("li", class_='next')

            next_page = next_btn.a["href"] if next_btn else None

        else:

            print(f'Failed to fetch webpage. Status code: {response.status_code}')

            break

# Run the scraper

scrape_quotes()

```

Output:

```
(myenv) C:\Users\Lenovo>python -u "c:\Users\Lenovo\import requests.py"
```

"“The world as we have created it is a process of our thinking. It cannot be changed without changing our thinking.”” - Albert Einstein

"“It is our choices, Harry, that show what we truly are, far more than our abilities.”” - J.K. Rowling

"“There are only two ways to live your life. One is as though nothing is a miracle. The other is as though everything is a miracle.”” - Albert Einstein

"“The person, be it gentleman or lady, who has not pleasure in a good novel, must be intolerably stupid.”” - Jane Austen

"“Imperfection is beauty, madness is genius and it's better to be absolutely ridiculous than absolutely boring.”” - Marilyn Monroe

"“Try not to become a man of success. Rather become a man of value.”” - Albert Einstein

"“It is better to be hated for what you are than to be loved for what you are not.”” - André Gide

"“I have not failed. I've just found 10,000 ways that won't work.”” - Thomas A. Edison

"“A woman is like a tea bag; you never know how strong it is until it's in hot water.”” - Eleanor Roosevelt

"“A day without sunshine is like, you know, night.”” - Steve Martin

7 **Zappa:**

- a. Purpose: Deploy Python web applications to AWS Lambda and API Gateway.
- b. Key Features:
 - i. Supports frameworks like Flask and Django for serverless deployments.
 - ii. Manages serverless architecture and deployment configurations.
- c. Common Use: Build scalable, serverless web apps without maintaining servers.

8 **Dash:**

- a. Purpose: Web application framework for building interactive data visualization applications.
- b. Key Features:
 - i. Built on top of Flask, React, and Plotly.
 - ii. Integrates seamlessly with data science libraries (e.g., Pandas, Plotly).
- c. Common Use: Building dashboards and data-driven web applications.

b) Turbo Gears

- a. Purpose: Full-stack web framework built on top of WSGI.
- b. Key Features:
 - i. Modular: Mix and match components like SQLAlchemy, Genshi, and others.
 - ii. Focus on rapid development and scalability.
- c. Common Use: Develop scalable, enterprise-level web applications.

9 CherryPy:

- a. Purpose: Minimalistic web framework for building web applications.
- b. Key Features:
 - i. Provides a simple and fast HTTP server.
 - ii. Handles routing, cookies, sessions, and file uploads.
- c. Common Use: Building web applications with a lightweight framework.

Code:

```

b) import cherrypy
c) class HelloWorld:
d) @cherrypy.expose # Exposes this method as a web page
e) def index(self):
f) return "Hello, World! Welcome to CherryPy Web Server."
g) # Configure and start the CherryPy server
h) if __name__ == "__main__":
i) cherrypy.quickstart(HelloWorld(), '/', config={
j) 'global': {
    a. 'server.socket_host': '127.0.0.1', # Localhost
    b. 'server.socket_port': 8080,      # Port number
k) }
l) })

```

Output:

```
(myenv) C:\Users\Lenovo>python -u "c:\Users\Lenovo\import requests.py"
```

```
[10/Apr/2025:01:34:09] ENGINE Listening for SIGTERM.
```

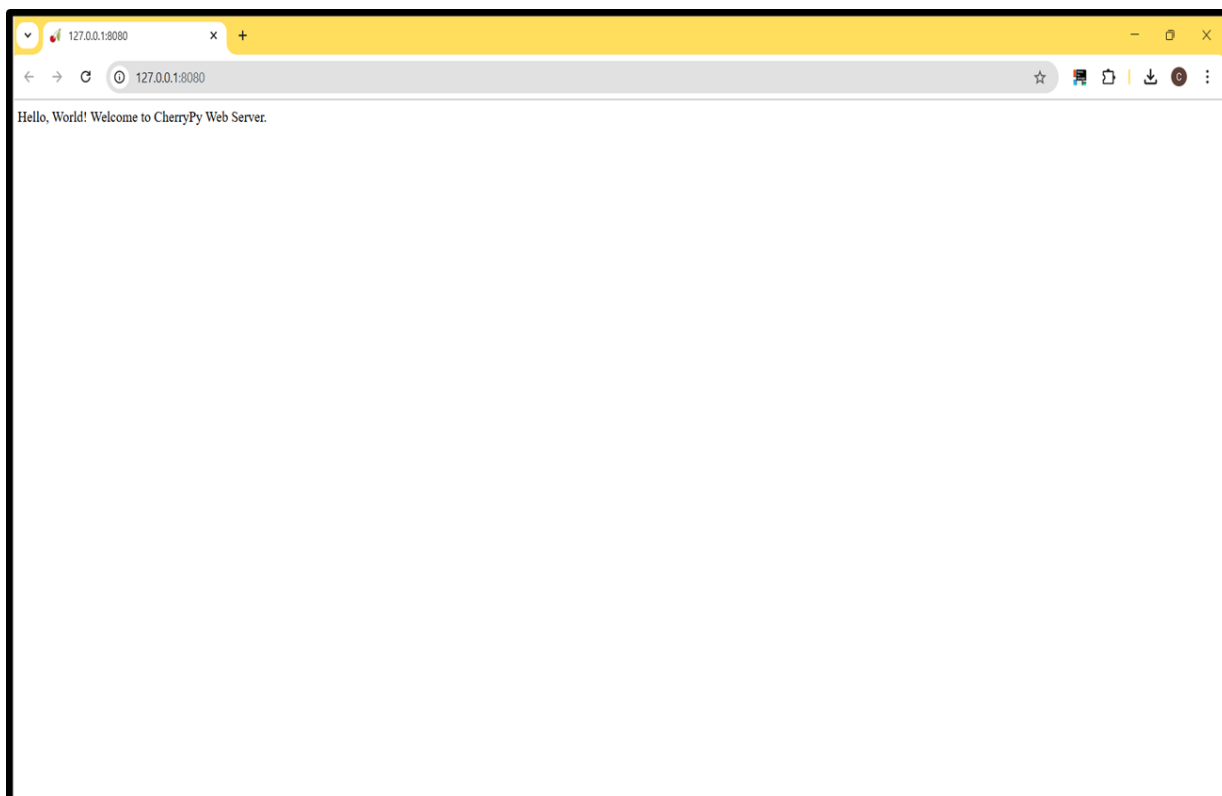
```
[10/Apr/2025:01:34:09] ENGINE Bus STARTING
```

```
[10/Apr/2025:01:34:09] ENGINE Started monitor thread 'Autoreloader'.
```

```
[10/Apr/2025:01:34:09] ENGINE Serving on http://127.0.0.1:8080
```

```
[10/Apr/2025:01:34:09] ENGINE Bus STARTED
```

After run the server :-



a) Flask:

- a. Purpose: Lightweight micro-framework for building web applications.
- b. Key Features:
 - i. Simple to learn and use, but highly extensible.
 - ii. Supports extensions for database integration, form handling, authentication, etc.
- c. Common Use: Small to medium web applications, APIs, or microservices.

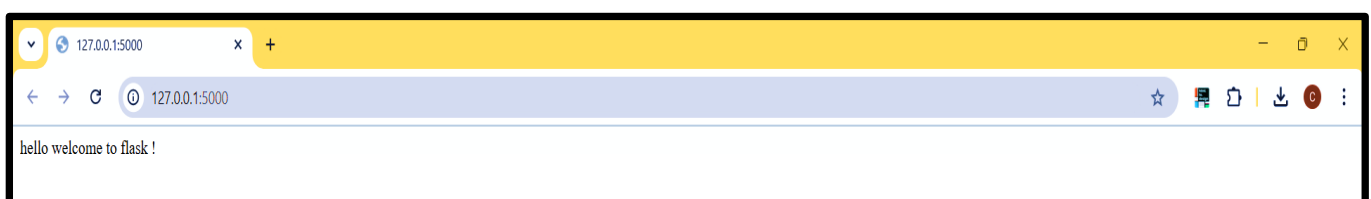
9.1.1 Code:

```
from flask import Flask
app = Flask(__name__)
@app.route('/', methods=['GET'])
def hellouser():
    return "Hello, welcome to Flask!"
if __name__ == '__main__':
    app.run(debug=True)
```

Output:

```
(myenv) C:\Users\Lenovo> * Serving Flask app 'import requests'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use
a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 134-121-940
```

After run the server:



10 Web2Py:

- a. Purpose: Full-stack framework for rapid web application development.
- b. Key Features:
 - i. Includes a web-based IDE for development.
 - ii. Built-in ticketing system and database integration.
- c. Common Use: Enterprise web applications with minimal setup.

11 Bottle:

- a. Purpose: Simple and lightweight WSGI micro-framework.
- b. Key Features:
 - i. Single-file framework, minimalistic, and fast.
 - ii. No dependencies, supports routing, templates, and form handling.
- c. Common Use: Small web applications, APIs, and prototypes.

Code:

```
from bottle import Bottle, run  
  
app = Bottle()  
  
@app.route('/')  
  
def home():  
  
    return "Hello, welcome to Bottle framework!"  
  
if __name__ == '__main__':  
  
    run(app, host='localhost', port=8080, debug=True)
```

Output:

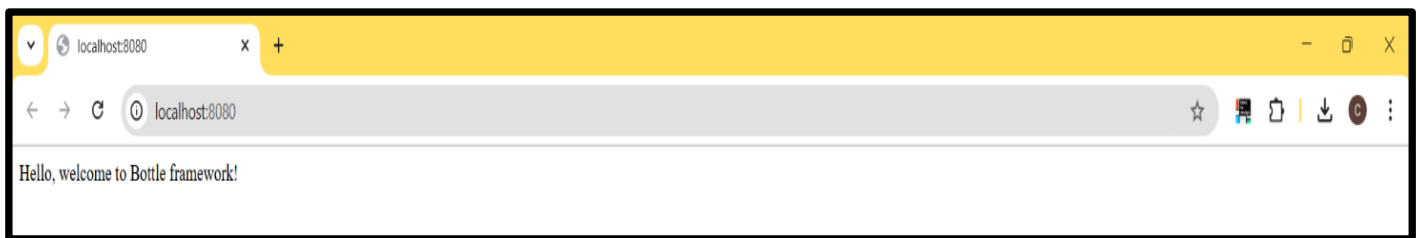
```
(myenv) C:\Users\Lenovo>python -u "c:\Users\Lenovo\import requests.py"
```

```
Bottle v0.13.2 server starting up (using WSGIRefServer()...
```

```
Listening on http://localhost:8080/
```

```
Hit Ctrl-C to quit.
```

After run the server:



a) Falcon:

- a. Purpose: High-performance framework for building APIs.
- b. Key Features:
 - i. Focuses on speed and minimalism.
 - ii. Supports RESTful API development and is optimized for large-scale deployments.
- c. Common Use: Building fast, high-performance APIs.

12 CubicWeb:

- a. Purpose: Web application framework based on an entity-relation model.
- b. Key Features:
 - i. Uses a highly modular architecture for development.
 - ii. Focus on building web apps with rich data models.
- c. Common Use: Semantic web applications or data-driven web apps.

13 Quixote:

- a. Purpose: A web framework designed for simplicity and scalability.
- b. Key Features:
 - i. Full support for Python's object-oriented programming.
 - ii. Easily extensible, with minimalistic core.
- c. Common Use: Scalable and customizable web applications.

14 **Pyramid:**

- a. **Purpose:** Full-stack web framework that can scale from simple to complex applications.
- b. **Key Features:**
 - i. Highly flexible with support for routing, templating, authentication, and authorization.
 - ii. Allows for small and large applications, with fine-grained control.
- c. **Common Use:** Building large, enterprise-grade web applications and REST APIs.

SUMMARY:

- a) **Flask, Django, Pyramid:** Popular web frameworks, each offering flexibility and scalability.
- b) **Scrapy, BeautifulSoup4:** Specialized for web scraping and data extraction.
- c) **Requests, Zappa, Dash:** Tools for making HTTP requests, serverless apps, and interactive data visualizations.
- d) **Tkinter, Bottle, CherryPy:** Libraries for building lightweight desktop and web applications.



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Introduction to Django Frame Work
7. Date of Experiment : 20-12-2024
8. Date of Submission of Report : 27-12-2024

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUATION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Django: A Web Framework for Python

- **Django:** Django is a high-level Python web framework that allows developers to build secure, scalable, and maintainable web applications quickly and efficiently. It follows the Model-View-Template (MVT) architectural pattern.
- **Key Features of Django:**
 - Fast Development – Comes with built-in features like authentication, database management, and an admin panel.
 - Scalability – Suitable for small projects to enterprise-level applications.
 - Security – Protects against common security threats (SQL Injection, CSRF, XSS, etc.).
 - ORM (Object-Relational Mapper) – Allows database interaction using Python instead of SQL.
 - Built-in Admin Panel – Auto-generates an admin interface for managing data.
 - Reusable App-Developers can create modular and reusable components.

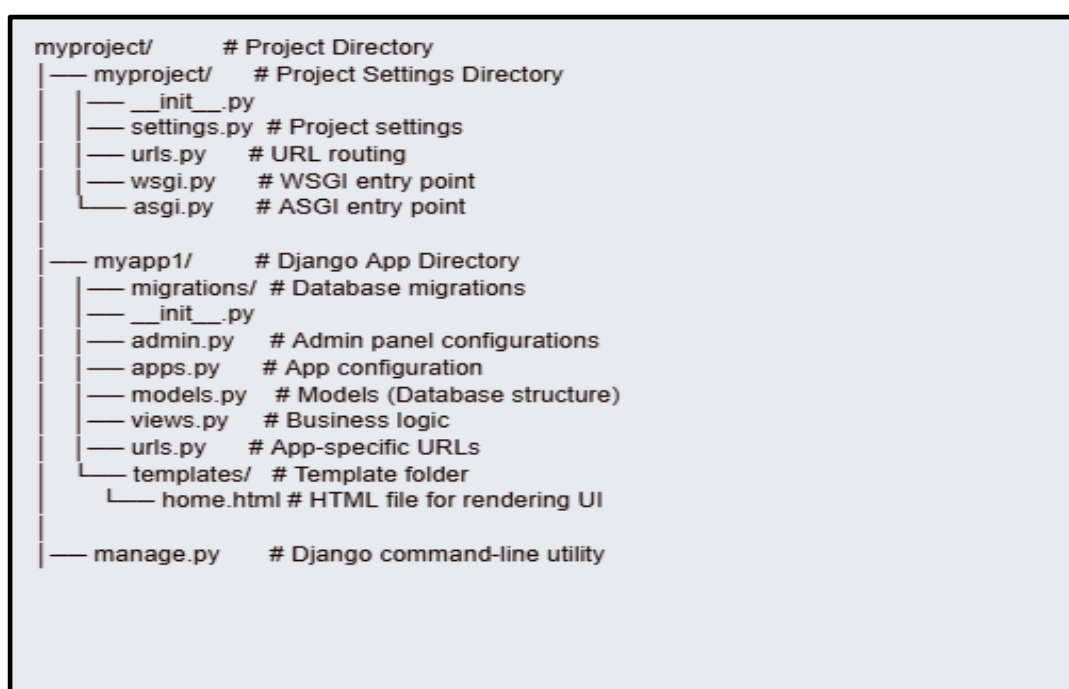
Django's MVT Architecture:

Model (M) – Handles database interactions (e.g., User, Booking).

View (V) – Manages business logic and connects models to templates.

Template (T) – Renders HTML pages dynamically.

Example MVT Folder Structure in Django





**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Step By Step Guide to Installing Django
7. Date of Experiment : 27-12-2024
8. Date of Submission of Report : 03-01-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUATION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Step By Step Guide to Installing Django

Steps to Create Django Project:

Step-1 : Checking the installation & version of Python & PIP `python --version` , `pip --version`

Step-2 : Installation of Virtual Environment `pip install virtualenvwrapper-win`

Step-3 : Creation of Virtual Environment `mkvirtualenv (name)`

Step-4 : Installation of Django in Virtual environment

`pip install Django`

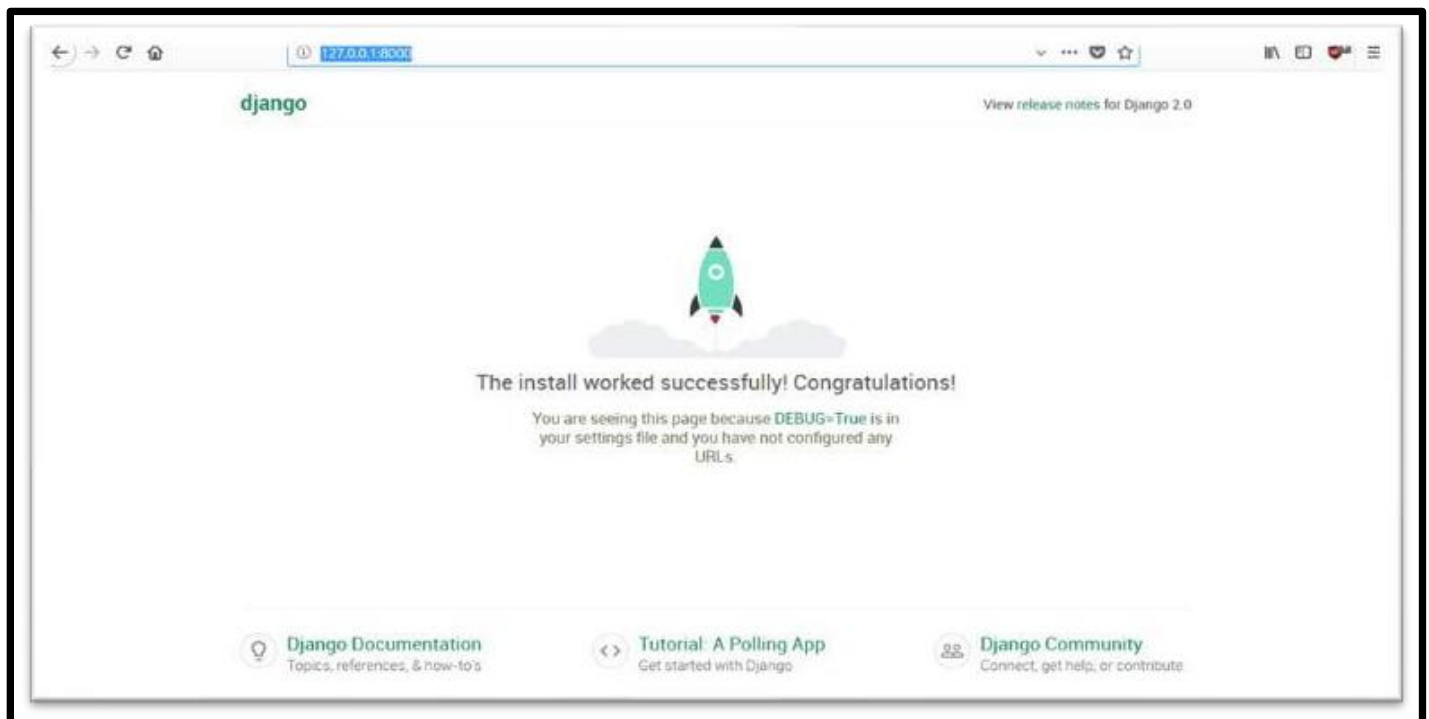
Step-5 : Create a folder to store all the projects `mkdir proj_folder_name`

Step-6 : Start new_project `django-admin startproject project_name` `django-admin startapp app_name`

Step-7 : Run the server `python manage.py runserver`

Step-8 : Open the browser and check the homepage of Django.

Output:





**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Linking Views and URL Configurations
7. Date of Experiment : 03-01-2025
8. Date of Submission of Report : 24-01-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUATION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Connecting Views and URLs:

Set Up URLs:

Project-level URL Configuration:

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path("", include('myapp1.urls')), # Include app URLs
]
```

App-level URL Configuration

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

Create a Sample View:

```
from django.http import HttpResponse
def home(request):
    return HttpResponse("<h1>Welcome to My Django App!</h1>")
```

Run Migrations:

```
python manage.py migrate
```

Run the Server and Test:

```
python manage.py runserver
```



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Exploring Django Views
7. Date of Experiment : 24-01-2025
8. Date of Submission of Report : 24-01-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Views.py:

In Django, views.py is the file where you define functions or classes that handle requests and return responses. Views act as the logic layer of a Django web application, controlling how data is processed and which HTML templates are displayed.

Code:

```
from django.shortcuts import render, redirect
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
from django.contrib.auth import login, logout
from django.contrib import messages
from django.contrib.auth.models import Group
from django.utils import timezone
from .models import Room, Booking
from django.core.mail import send_mail
from django.conf import settings
# Home Page View
def home(request):
    # Prevent students from accessing the home page
    if request.user.groups.filter(name='Student').exists():
        return redirect('student_dashboard') # Redirect students directly to their dashboard
    return render(request, 'home.html')
# Login View
def login_view(request):
    if request.user.is_authenticated:
        return redirect_based_on_role(request)
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            user = form.get_user()
            login(request, user)
            return redirect_based_on_role(request)
        else:
            messages.error(request, "Invalid username or password.")
    else:
        form = AuthenticationForm()

    return render(request, 'login.html', {'form': form})
# Redirect Based on Role
def redirect_based_on_role(request):
    user = request.user
    if user.is_staff:
        messages.success(request, f'Welcome back, {user.username} (Admin)!')
        return redirect('admin_dashboard')
    elif user.groups.filter(name='Teacher').exists():
        messages.success(request, f'Welcome back, {user.username}
```

```

elif user.groups.filter(name='Student').exists():

    messages.success(request, f'Welcome back, {user.username} (Student)!')
    return redirect('student_dashboard')
else:
    messages.error(request, 'User has no assigned role.')

    return redirect('home')
# Logout View
def logout_view(request):
    logout(request)
    messages.success(request, "You have been logged out successfully.")
    return redirect('home')
# Room Booking View
from django.shortcuts import render, redirect
from django.contrib import messages
from .models import Room, Booking
from django.contrib.auth.decorators import login_required
from datetime import datetime
@login_required
def book_room(request):
    if request.method == "POST":
        room_id = request.POST.get("room_id")
        date = request.POST.get("date")
        start_time = request.POST.get("start_time")

        try:
            room = Room.objects.get(id=room_id)
            time_slot = datetime.strptime(start_time, "%H:%M").time()
            # Check if the room is already booked for the same time
            existing_booking = Booking.objects.filter(room=room, date=date, time_slot=time_slot).exists()
            if existing_booking:
                messages.error(request, "Room is already booked for the selected time slot.")
            else:
                booking = Booking.objects.create(
                    room=room,
                    date=date,
                    time_slot=time_slot,
                    booked_by=request.user.username,
                    user=request.user,
                    email=request.user.email,
                )
                booking.send_booking_email()
                messages.success(request, "Room booked successfully!")
                return redirect("home")

        except Room.DoesNotExist:
            messages.error(request, "Room not found.")
    rooms = Room.objects.filter(is_active=True)

```

```

    return render(request, 'book_room.html', {'rooms': rooms})
# View Schedule
def view_schedule(request):

    bookings = Booking.objects.filter(user=request.user)

    return render(request, 'view_schedule.html', {'bookings': bookings})

# Send Booking Confirmation Email
def send_booking_email(booking):
    subject = 'Room Booking Confirmation'
    message = f'Dear {booking.user.username},\n\nYour booking for {booking.room.name} on
{booking.date} is confirmed.'
    from_email = settings.EMAIL_HOST_USER
    to_email = [booking.user.email]
    send_mail(subject, message, from_email, to_email)

# Signup View
def signup_view(request):
    if request.user.is_authenticated:
        return redirect('home')

    if request.method == 'POST':
        form = UserCreationForm(request.POST)
        role = request.POST.get('role')

        if form.is_valid():
            user = form.save()

            if role:
                try:
                    group = Group.objects.get(name=role)
                    user.groups.add(group)
                except Group.DoesNotExist:
                    messages.error(request, "The selected role doesn't exist.")
                    return redirect('signup')
            else:
                messages.error(request, "Please select a role.")
                return redirect('signup')

        login(request, user)

        if role == 'Admin':
            messages.success(request, 'Account created successfully! You are now logged in as Admin.')
            return redirect('admin_dashboard')
        elif role == 'Teacher':

```

```

        messages.success(request, 'Account created successfully! You are now logged in as Teacher.')
        return redirect('teacher_dashboard')
    elif role == 'Student':
        messages.success(request, 'Account created successfully! You are now logged in as Student.')
        return redirect('student_dashboard')

    else:
        messages.error(request, 'Unknown role selected.')

        return redirect('home')

    else:
        messages.error(request, f'There was an error creating your account: {form.errors}')

    else:
        form = UserCreationForm()

        return render(request, 'signup.html', {'form': form})

# Admin Dashboard View
def admin_dashboard(request):
    return render(request, 'admin_dashboard.html')

# Teacher Dashboard View
def teacher_dashboard(request):
    return render(request, 'teacher_dashboard.html')

# Student Dashboard View
from django.contrib.auth.models import Group

from django.shortcuts import render
from .models import Booking, Room # Import Booking and Room models
from django.contrib.auth.models import Group

# Student Dashboard View
def student_dashboard(request):
    if not request.user.groups.filter(name='Student').exists():
        messages.error(request, "You do not have permission to view this page.")
        return redirect('home')

    # Fetch the student's own bookings
    student_bookings = Booking.objects.filter(user=request.user)

    # Fetch all bookings made by teachers
    teacher_group = Group.objects.get(name='Teacher')
    teacher_bookings = Booking.objects.filter(user__groups=teacher_group)

    # Fetch all available rooms
    available_rooms = Room.objects.all()

```

```

return render(request, 'student_dashboard.html', {
    'student_bookings': student_bookings,
    'teacher_bookings': teacher_bookings,
    'available_rooms': available_rooms, # Pass the available rooms to the template
})

```

```

from django.shortcuts import render
from .models import Booking # Ensure that you import the necessary model

```

```

# View to show all bookings (for admin)
def view_all_bookings(request):
    bookings = Booking.objects.all() # Fetch all bookings from the database
    return render(request, 'view_all_bookings.html', {'bookings': bookings})
# myapp1/views.py

```

```

from django.shortcuts import render
from .models import Room # Make sure to import your Room model

```

```

# View for managing rooms (for admin)
def manage_rooms(request):
    rooms = Room.objects.all() # Fetch all rooms from the database

```

```

    return render(request, 'manage_rooms.html', {'rooms': rooms})
from django.core.mail import send_mail
from django.conf import settings
import logging

```

```

def send_booking_email(booking):
    subject = 'Room Booking Confirmation'
    message = f'Dear {booking.user.username},\n\nYour booking for {booking.room.name} on
{booking.date} is confirmed.'
    from_email = settings.EMAIL_HOST_USER
    to_email = [booking.user.email]

```

```

    try:
        send_mail(subject, message, from_email, to_email)
        logging.info(f'Booking confirmation email sent to {booking.user.email}')
    except Exception as e:
        logging.error(f'Failed to send booking confirmation email: {e}')

```

```

# views.py
from django.shortcuts import render, get_object_or_404, redirect
from .models import Room
from .forms import RoomForm # Assuming you have a form for Room

```

```

def edit_room(request, room_id):
    room = get_object_or_404(Room, id=room_id)

```

```

    if request.method == "POST":

```



```

    form = RoomForm(request.POST, instance=room)
    if form.is_valid():
        form.save()
        return redirect('manage_rooms') # Redirect to the rooms management page
    else:
        form = RoomForm(instance=room)

    return render(request, 'edit_room.html', {'form': form})

# views.py
def delete_room(request, room_id):
    room = get_object_or_404(Room, id=room_id)

    room.delete()
    return redirect('manage_rooms') # Redirect to the rooms management page

```

In Django, the `urls.py` file is responsible for mapping URLs to views. It acts as the router of your application, directing user requests to the correct function in `views.py`.



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Setting Up App-Level URLs
7. Date of Experiment : 24-01-2025
8. Date of Submission of Report : 31-01-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUATION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

App Urls.Py :

Creating urls.py in a Django App:

Each Django app should have its own urls.py file to define app-specific routes.

Steps to Create urls.py in a Django App

- Inside your Django app folder (myapp1), create a file named urls.py.
- Define URL patterns to map URLs to views.

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.home, name='home'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
    path('signup/', views.signup_view, name='signup'),
    path('book-room/', views.book_room, name='book_room'),
    path('view-schedule/', views.view_schedule, name='view_schedule'),
    path('admin-dashboard/', views.admin_dashboard, name='admin_dashboard'),
    path('teacher-dashboard/', views.teacher_dashboard, name='teacher_dashboard'), # Teacher
Dashboard
    path('student-dashboard/', views.student_dashboard, name='student_dashboard'), # Add this for
student dashboard
    path('view-all-bookings/', views.view_all_bookings, name='view_all_bookings'),
    path('manage-rooms/', views.manage_rooms, name='manage_rooms'),
    path('rooms/edit/<int:room_id>', views.edit_room, name='edit_room'),
    path('rooms/delete/<int:room_id>', views.delete_room, name='delete_room'),
]
```

Connecting App urls.py to Project urls.py:

To use the app's URLs, include them in the project-level urls.py (myproject/urls.py)

```
from django.contrib import admin  
from django.urls import path, include  
from myapp1.views import *  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('', include('myapp1.urls')), # Include your app URLs  
]
```



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.[it@intugvcev.edu.in](mailto:hod.it@intugvcev.edu.in)

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Working with Templates in Django
7. Date of Experiment : 31-01-2025
8. Date of Submission of Report : 17-02-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Templates:

Templates in Django are HTML files that display dynamic content. They separate the frontend (UI) from the backend logic, following the MVT (Model-View-Template) architecture.

Where to Store Templates?

By default, Django looks for templates in a folder named `templates/` inside your app.

```
myproject/
├── myapp1/
│   ├── templates/          # Folder for HTML templates
│   │   ├── home.html
│   │   ├── login.html
│   │   ├── signup.html
│   │   ├── book_room.html
│   │   ├── view_schedule.html
│   │   ├── admin_dashboard.html
│   │   ├── teacher_dashboard.html
│   │   ├── student_dashboard.html
│   │   ├── manage_rooms.html
│   │   └── view_all_bookings.html
│   ├── views.py            # Handles logic
│   ├── urls.py             # URL mapping
│   └── models.py           # Database models
```

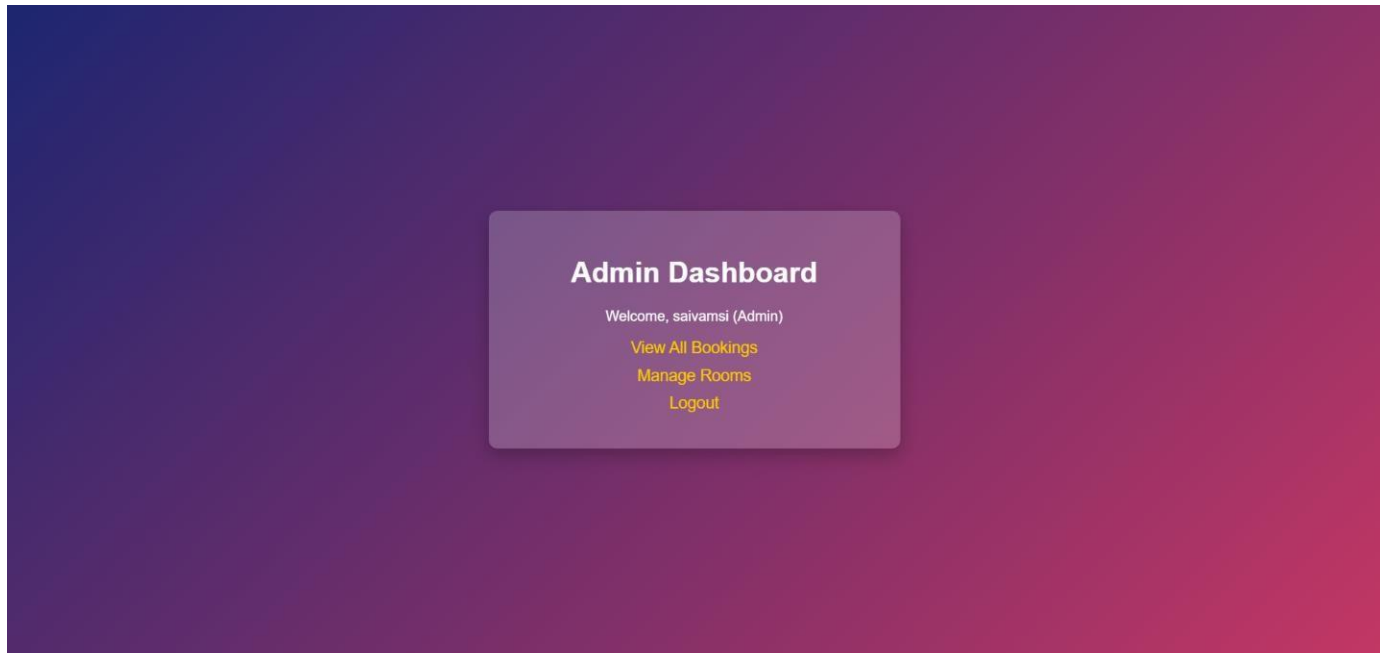
Admin_dashboard.html :

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Admin Dashboard</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background: linear-gradient(135deg, #1d2671, #c33764);
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      text-align: center;
      color: white;
    }
    .container {
      background: rgba(255, 255, 255, 0.2);
      padding: 30px;
      border-radius: 10px;
      box-shadow: 0 8px 20px rgba(0, 0, 0, 0.25);
      backdrop-filter: blur(10px);
      width: 400px;
    }
    a {
      display: block;
      margin: 10px 0;
      color: #ffcc00;
      text-decoration: none;
      font-size: 18px;
    }
    a:hover {
      color: #fff;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Admin Dashboard</h1>
    <p>Welcome, {{ user.username }} (Admin)</p>
    <a href="{% url 'view_all_bookings' %}">View All Bookings</a>
    <a href="{% url 'manage_rooms' %}">Manage Rooms</a>
    <a href="{% url 'logout' %}">Logout</a>
  </div>
</body>
</html>

```

OUTPUT:



Role of admin_dashboard.html in Django:

The admin_dashboard.html template is used to display the Admin Dashboard for users with admin privileges. This page provides management options for the administrator, such as:

- Viewing all bookings
- Managing rooms (adding/editing/deleting rooms)
- Managing users (teachers, students, other admins)
- Approving or rejecting booking requests

Book_room.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Book a Classroom</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: linear-gradient(135deg, #4facfe, #00f2fe);
      color: #fff;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      overflow: hidden;
      animation: gradientMove 5s ease infinite;
    }

    @keyframes gradientMove {
      0% { background: linear-gradient(135deg, #4facfe, #00f2fe); }
      50% { background: linear-gradient(135deg, #00f2fe, #4facfe); }
      100% { background: linear-gradient(135deg, #4facfe, #00f2fe); }
    }

    .container {
      background: rgba(0, 0, 0, 0.7);
      padding: 20px;
      border-radius: 12px;
      width: 400px;
      box-shadow: 0px 5px 15px rgba(0, 0, 0, 0.3);
      animation: slideIn 0.8s ease-in-out, fadeIn 0.8s ease-in-out;
    }

    @keyframes slideIn {
      from { transform: translateY(20px); opacity: 0; }
      to { transform: translateY(0); opacity: 1; }
    }

    h1 {
      text-align: center;
      font-size: 24px;
      margin-bottom: 15px;
      animation: fadeIn 1.5s ease-in-out;
    }

    label {

```

```

font-size: 16px;
display: block;
margin: 10px 0 5px;
}

```

```

select, input {
width: 100%;
padding: 12px;
border: 2px solid #ddd;
border-radius: 8px;
font-size: 16px;
background: #fff;
color: #333;
margin-bottom: 15px;
transition: all 0.3s ease-in-out;
box-sizing: border-box;
}

```

```

select:hover, input:hover {
transform: scale(1.05);
border-color: #ff9f43;
}

```

```

select:focus, input:focus {
outline: none;
border-color: #ff9f43;
background: #f9f9f9;
}

```

```

button {
width: 100%;
padding: 12px;
background: #ff9f43;
color: white;
font-size: 18px;
border: none;
border-radius: 5px;
cursor: pointer;
transition: transform 0.3s, background 0.3s, box-shadow 0.3s;
animation: pulse 1.5s infinite;
}

```

```

@keyframes pulse {
0% { transform: scale(1); }
50% { transform: scale(1.05); }
100% { transform: scale(1); }
}

```

```

button:hover {

```

```

    transform: scale(1.1);
    background: #ff6b2b;
    box-shadow: 0px 5px 15px rgba(255, 255, 255, 0.3);
}

.messages {
    margin-top: 15px;
    padding: 10px;
    background: rgba(255, 255, 255, 0.2);

    border-radius: 5px;
    font-size: 14px;
    animation: fadeIn 1s;
}

.back-link {
    display: block;
    text-align: center;
    margin-top: 10px;
    color: #ffcc00;
    text-decoration: none;
    transition: color 0.3s;
}

.back-link:hover {
    color: #fff;
}

@keyframes fadeIn {
    from { opacity: 0; transform: translateY(-10px); }
    to { opacity: 1; transform: translateY(0); }
}
</style>
</head>
<body>
<div class="container">
    <h1>Book a Classroom</h1>
    <form method="POST">
        {% csrf_token %}

        <!-- Classroom selection -->
        <label for="classroom">Select Classroom:</label>
        <select name="room_id" id="classroom" required>
            {% for room in rooms %}
                <option value="{{ room.id }}">{{ room.name }} (Capacity: {{ room.capacity }})</option>
            {% empty %}
                <option value="">No rooms available</option>
            {% endfor %}

```

```

</select>

<!-- Date selection -->
<label for="date">Select Date:</label>
<input type="date" name="date" id="date" required>

<!-- Start Time selection -->
<label for="start_time">Start Time:</label>
<input type="time" name="start_time" id="start_time" required>

<!-- End Time selection -->
<label for="end_time">End Time:</label>
<input type="time" name="end_time" id="end_time" required>

<!-- Submit Button -->

<button type="submit">Book Classroom</button>

<!-- Displaying Messages (success/failure) -->
{% if messages %}
    <div class="messages">
        {% for message in messages %}
            <p>{{ message }}</p>
        {% endfor %}
    </div>
{% endif %}
</form>

<!-- Back Link -->
<a class="back-link" href="{% url 'home' %}">Back to Home</a>
</div>
</body>
</html>

```

OUTPUT:

Role of book_room.html in Django:

The book_room.html template is used to allow users (teachers, students, or admins) to book available classrooms in the Classroom Booking System. It provides a form where users can select a room and a date for booking.

- Displays a list of available rooms for booking.
- Allows users to select a room and date from a form.
- Prevents double bookings (if a room is already booked for that date).
- Sends an email confirmation upon successful booking.
- Displays messages (success or error) after form submission.

Edit_room.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Edit Room</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background: linear-gradient(135deg, #ff6a00, #ee0979);
      margin: 0;
```

```

padding: 0;
display: flex;
justify-content: center;
align-items: center;
height: 100vh;
}

.container {
background: rgba(255, 255, 255, 0.9);
padding: 40px;
border-radius: 10px;
box-shadow: 0 8px 30px rgba(0, 0, 0, 0.2);

max-width: 600px;
width: 100%;
transition: transform 0.3s ease;
}

.container:hover {
transform: scale(1.05);
}

h1 {
font-size: 32px;
text-align: center;
color: #333;
margin-bottom: 20px;
}

form {
display: flex;
flex-direction: column;
}

form button {
padding: 12px 20px;
background-color: #ff6a00;
color: white;
font-size: 18px;
border: none;
border-radius: 5px;
cursor: pointer;
transition: background-color 0.3s;
}

form button:hover {
background-color: #ee0979;
}

```

```

form input,
form select {
    padding: 10px;
    margin: 10px 0;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

a {
    display: block;
    text-align: center;
    margin-top: 20px;
    color: #ff6a00;
    text-decoration: none;
    font-size: 18px;
    transition: color 0.3s;
}

a:hover {
    color: #ee0979;
}

.confirmation-message {
    display: none;

    padding: 10px;
    background-color: #4caf50;
    color: white;
    border-radius: 5px;
    margin-top: 20px;
    text-align: center;
}

</style>
</head>
<body>

<div class="container">
    <h1>Edit Room</h1>

    <form method="POST" id="edit-room-form">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit" id="submit-button">Save Changes</button>
    </form>

    <a href="{% url 'manage_rooms' %}">Back to Room Management</a>

```

```

<div class="confirmation-message" id="confirmation-message">
  Your changes have been saved successfully!
</div>
</div>

<script>
  // Handle form submission and display confirmation message
  const form = document.getElementById('edit-room-form');
  const submitButton = document.getElementById('submit-button');
  const confirmationMessage = document.getElementById('confirmation-message');

  form.addEventListener('submit', function(event) {
    event.preventDefault(); // Prevent form submission to simulate the save process

    // Show the confirmation message after form submission
    confirmationMessage.style.display = 'block';

    // Simulate a successful form submission with a slight delay
    setTimeout(function() {
      confirmationMessage.style.display = 'none';
      form.submit(); // Actually submit the form after the confirmation
    }, 2000); // Hide confirmation message after 2 seconds
  });
</script>

</body>
</html>

```

OUTPUT:

The screenshot shows a web application interface for editing a room. The form is titled "Edit Room" and is set against a light pink background with a subtle shadow. The form fields are as follows:

- Name:** A text input field containing "Room D1".
- Capacity:** A text input field containing "50".
- Teacher:** A dropdown menu with a downward arrow and a placeholder text "-----".
- Is active:** A checkbox that is checked, with a blue checkmark icon.

Below the form, there are two buttons:

- Save Changes:** A prominent orange button.
- Back to Room Management:** A smaller, light pink button.

Role of edit_room.html in **Django**:

The edit_room.html template is used for editing the details of an existing room in the Classroom Booking System. It allows admins to modify room information, such as the room name, capacity, or availability.

1. Displays a form with the current room details pre-filled.
2. Allows admins to update room details and save changes.
3. Validates form inputs before saving.
4. Redirects back to the manage rooms page after a successful update.

Home.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Classroom Booking</title>
  <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap"
rel="stylesheet">
  <style>
    :root {
      --primary-color: #673ab7;
      --secondary-color: #512da8;
      --background-color: #303f9f;
      --text-color: #fff;
      --container-background: rgba(255, 255, 255, 0.1);
      --font-family: 'Poppins', sans-serif;
    }

    body {
      font-family: var(--font-family);
      background-color: var(--background-color);
      color: var(--text-color);
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
      margin: 0;
      overflow: hidden;
      transition: background-color 0.5s ease;
    }

    .background-overlay {
      position: fixed;
      top: 0;
      left: 0;
      width: 100%;
      height: 100%;
      background-color: rgba(0, 0, 0, 0.4);
      z-index: -1;
      animation: backgroundPulse 10s infinite alternate;
    }

    @keyframes backgroundPulse {
      0% {
        background-color: rgba(0, 0, 0, 0.4);
      }
      100% {
        background-color: rgba(0, 0, 0, 0.6);
      }
    }
  </style>

```

```

    }
}

```

```

.container {
  background: var(--container-background);
  padding: 40px;
  border-radius: 15px;
  box-shadow: 0 0 20px rgba(0, 0, 0, 0.3);
  text-align: center;
  backdrop-filter: blur(5px);
  max-width: 400px;
  width: 90%;
  animation: fadeIn 1s ease-in-out, slideIn 0.5s ease forwards;
}

```

```

@keyframes slideIn {
  from {
    transform: translateX(-20px);
    opacity: 0;
  }
  to {
    transform: translateX(0);
    opacity: 1;
  }
}

```

```

h1, p {
  opacity: 0;
  animation: fadeInText 0.5s ease forwards;
}

```

```

h1 {
  animation-delay: 0.2s;
}

```

```

p {
  animation-delay: 0.4s;
}

```

```

@keyframes fadeInText {
  from {
    opacity: 0;
    transform: translateY(10px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

```

```

    }
}

.button {
    display: inline-block;
    padding: 15px 30px;
    border-radius: 10px;
    text-decoration: none;
    color: var(--text-color);
    background: var(--primary-color);

    font-size: 1.1rem;
    transition: background-color 0.3s ease, transform 0.2s ease, box-shadow 0.2s ease;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
    animation: buttonPop 0.5s ease forwards;
}

@keyframes buttonPop {
    from {
        transform: scale(0.8);
        opacity: 0;
    }
    to {
        transform: scale(1);
        opacity: 1;
    }
}

.button:hover {
    background: var(--secondary-color);
    transform: translateY(-2px) scale(1.05);
    box-shadow: 0 6px 10px rgba(0, 0, 0, 0.3);
    animation: buttonBounce 0.3s ease;
}

@keyframes buttonBounce {
    0% {
        transform: translateY(0);
    }
    50% {
        transform: translateY(-5px);
    }
    100% {
        transform: translateY(0);
    }
}

.logout-button {
    background-color: #d32f2f;

```

```

}

.logout-button:hover {
    background-color: #c62828;
}

@media (max-width: 500px) {
    .container {
        padding: 20px;
    }

    h1 {
        font-size: 2rem;
    }

    p {

        font-size: 1rem;
    }

    .button {
        padding: 12px 25px;
        font-size: 1rem;
    }
}
</style>
</head>
<body>
<div class="background-overlay"></div>
<div class="container">
<div class="content">
<h1>Classroom Booking System</h1>
{% if user.is_authenticated %}
<p id="greeting">Welcome, {{ user.username }}!</p>
<nav class="button-container">
<a href="{% url 'book_room' %}" class="button">Book a Room</a>
<a href="{% url 'view_schedule' %}" class="button">View Schedule</a>
<a href="{% url 'logout' %}" class="button logout-button" id="logoutButton">Logout</a>
</nav>
{% else %}
<p>Please log in to continue.</p>
<a href="{% url 'login' %}" class="button">Login</a>
{% endif %}
</div>
</div>

<script>
window.onload = function() {
    const greetingElement = document.getElementById('greeting');

```

```

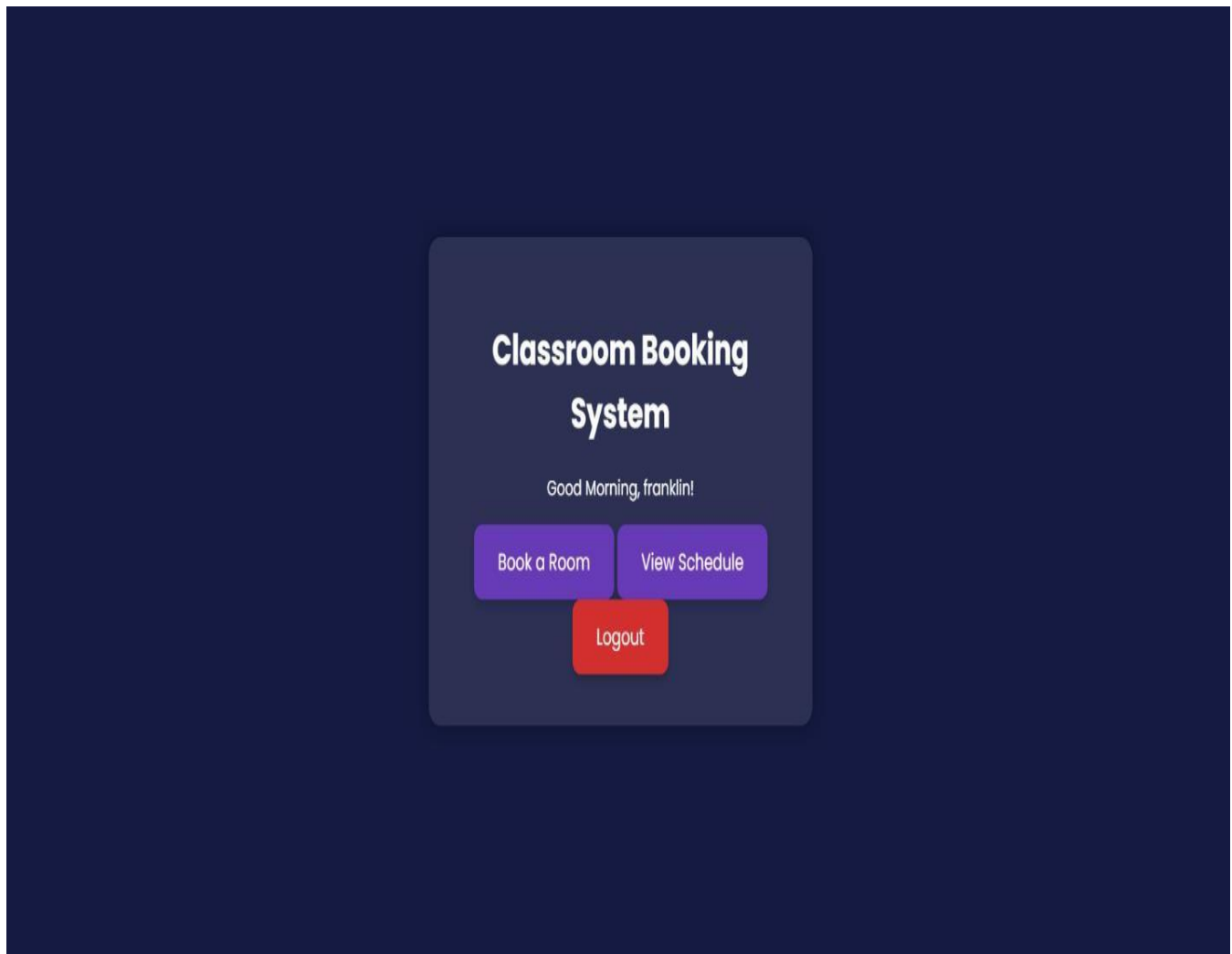
const logoutButton = document.getElementById('logoutButton');
const currentTime = new Date().getHours();

// Get the username dynamically from Django's context
const username = "{{ user.username }}"; // Rendered into the page

// Set greeting text based on time of day
if (currentTime < 12) {
    greetingElement.textContent = `Good Morning, ${username}!`;
} else if (currentTime < 18) {
    greetingElement.textContent = `Good Afternoon, ${username}!`;
} else {
    greetingElement.textContent = `Good Evening, ${username}!`;
}

// Add event listener for logout button
if (logoutButton) {
    logoutButton.addEventListener('click', function(event) {
        const confirmLogout = confirm("Are you sure you want to logout?");
        if (!confirmLogout) {
            event.preventDefault(); // Prevent logout if user cancels
        }
    });
}
};
</script>
</body>
</html>

```

OUTPUT:**Role of home.html in Django :**

The home.html template serves as the landing page of the Classroom Booking System. It is the first page users see when they visit the site.

1. Welcomes users to the system.
2. Displays navigation options (Login, Signup, Dashboard, etc.).
3. Redirects students directly to their dashboard if they try to access the home page.
4. Ensures unauthenticated users see the login/signup options.
5. May include basic system information (e.g., features, instructions).

Login.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background: linear-gradient(135deg, #6a11cb, #2575fc);
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }

    .container {
      background: rgba(255, 255, 255, 0.2);
      padding: 30px;
      border-radius: 10px;
      box-shadow: 0 8px 20px rgba(0, 0, 0, 0.25);
      backdrop-filter: blur(10px);
      width: 100%;
      max-width: 400px; /* Added max-width for better responsiveness */
      text-align: center;
    }

    h1 {
      color: #fff;
      margin-bottom: 20px;
    }

    label {
      color: #fff;
      font-size: 16px;
      display: block;
      margin-bottom: 8px;
    }

    input {
      width: 100%;
      padding: 12px;
      margin: 10px 0;
      border: none;
      border-radius: 8px;
      font-size: 16px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Login</h1>
    <div>
      <label>Email</label>
      <input type="text">
    </div>
    <div>
      <label>Password</label>
      <input type="password">
    </div>
    <button type="submit">Login</button>
  </div>
</body>
</html>

```



```

    background: rgba(255, 255, 255, 0.8);
}

```

```

button {
    background: #ff8c00;
    color: white;
    padding: 12px 20px;
    border-radius: 8px;
    cursor: pointer;
    transition: 0.3s;
    font-size: 16px;
}

```

```

button:hover {
    background: #e07b00;
}

```

```

.message {
    color: white;
    margin-top: 10px;
    font-weight: bold;
}

```

```

.link {
    color: white;
    font-size: 14px;
}

```

```

.link a {
    color: #ff8c00;
    text-decoration: none;
}

```

```

.link a:hover {
    text-decoration: underline;
}

```

```

/* Mobile responsiveness */
@media (max-width: 600px) {
    .container {
        padding: 20px;
        width: 90%;
    }
}

```

```

h1 {
    font-size: 24px;
}

```

```

label {

```

```

        font-size: 14px;
    }

    input {
        font-size: 14px;
    }

    button {
        font-size: 14px;
        padding: 10px 18px;
    }
}
</style>
</head>
<body>
    <div class="container">
        <h1>Login</h1>
        <form method="POST" onsubmit="return validateLogin()">
            {% csrf_token %}

            <label for="username">Username</label>
            <input type="text" name="username" id="username" required autofocus
autocomplete="username">

            <label for="password">Password</label>
            <input type="password" name="password" id="password" required autocomplete="current-
password">

            <button type="submit">Login</button>
        </form>

        {% if messages %}
        <div>
            {% for message in messages %}
            <p class="message">{{ message }}</p>
            {% endfor %}
        </div>
        {% endif %}

        <p class="link">Don't have an account? <a href="{% url 'signup' %}">Sign up here</a>.</p>
    </div>

    <script>
        function validateLogin() {
            const username = document.getElementById("username").value.trim();
            const password = document.getElementById("password").value.trim();

            if (username === "" || password === "") {
                alert("Please fill in all fields.");
                return false;
            }
        }
    </script>

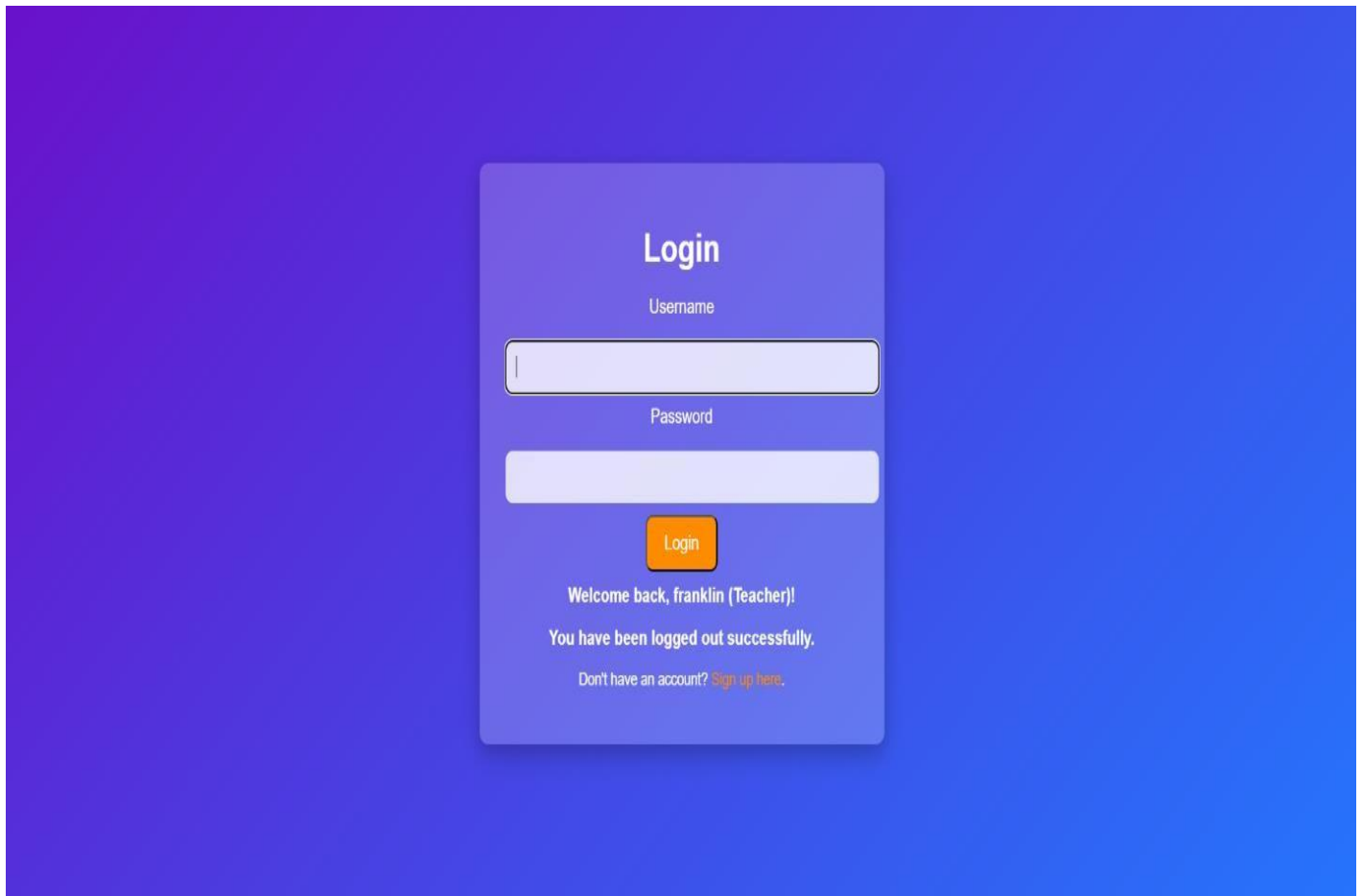
```

```

    }
    return true;
}
</script>
</body>
</html>

```

OUTPUT:



Role of login.html in Django:

The login.html template provides the user login interface for the Classroom Booking System. It allows users to enter their credentials and log in to their respective dashboards based on their role (Admin, Teacher, or Student).

- Displays a login form (username & password).
- Authenticates users using Django's built-in authentication system.

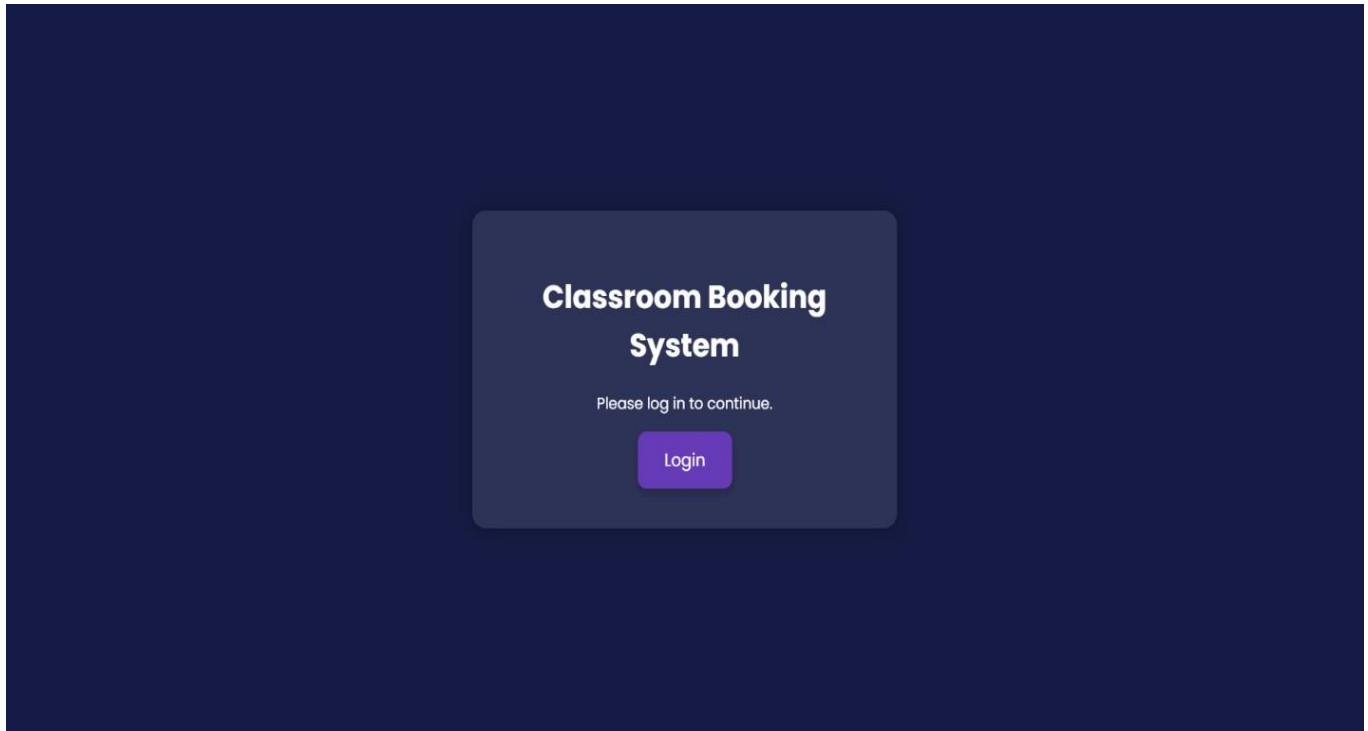
||nd btech ||nd

- Redirects users to their respective dashboards based on their role.
- Displays error messages if login fails.
- Provides a link to the signup page for new users

Logout.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Logged Out</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background: linear-gradient(135deg, #6a11cb, #2575fc);
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      text-align: center;
      color: white;
    }
    .container {
      background: rgba(255, 255, 255, 0.2);
      padding: 30px;
      border-radius: 10px;
      box-shadow: 0 8px 20px rgba(0, 0, 0, 0.25);
      backdrop-filter: blur(10px);
      width: 350px;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>You have been logged out</h1>
    <a href="{% url 'login' %}" style="color: white;">Login Again</a>
  </div>
</body>
</html>
```

OUTPUT:



Role of logout_view in Django:

The logout function in Django handles logging out users and redirecting them to the home page or login page.

- Logs out the current authenticated user.
- Redirects the user to the home page or login page.
- Displays a success message confirming the logout.
- Prevents access to protected pages after logging out.

Manage_rooms.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Manage Rooms</title>
  <style>
    /* Basic Styles for Room Management */
    body {
      font-family: Arial, sans-serif;
      background-color: #f0f0f0;
      padding: 20px;
    }

    h1 {
      text-align: center;
      margin-bottom: 20px;
    }

    table {
      width: 100%;
      border-collapse: collapse;
    }

    table, th, td {
      border: 1px solid #ddd;
    }

    th, td {
      padding: 10px;
      text-align: center;
    }

    th {
      background-color: #1d2671;
      color: white;
    }

    tr:nth-child(even) {
      background-color: #f2f2f2;
    }
  </style>

```

```

tr:hover {
    background-color: #ddd;
}

.back-link {
    display: block;
    text-align: center;
    margin-top: 20px;
    font-size: 16px;
}

.back-link a {
    text-decoration: none;
    color: #1d2671;
}
</style>
</head>
<body>
<h1>Manage Rooms</h1>
<table>
<thead>
<tr>
<th>Name</th>
<th>Capacity</th>
<th>Teacher</th>
<th>Status</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
{% for room in rooms %}
<tr>
<td>{{ room.name }}</td>
<td>{{ room.capacity }}</td>
<td>{{ room.teacher }}</td>
<td>{{ room.is_active|yesno:"Active,Inactive" }}</td>
<td><a href="{% url 'edit_room' room.id %}">Edit</a> |
    <a href="{% url 'delete_room' room.id %}" onclick="return confirm('Are you sure you
want to delete this room?');">Delete</a></td>
</tr>
{% empty %}
<tr>
<td colspan="5">No rooms available.</td>
</tr>
{% endfor %}
</tbody>
</table>

```

Output:

Manage Rooms				
Name	Capacity	Teacher	Status	Actions
Room D1	50	None	Active	Edit Delete
Room A1	30	None	Active	Edit Delete
Room B1	25	None	Active	Edit Delete
Room C1	20	None	Active	Edit Delete
Room D1	50	None	Active	Edit Delete
Room E1	40	None	Active	Edit Delete
Room F1	35	None	Active	Edit Delete
Room G1	45	None	Active	Edit Delete
Room H1	60	None	Active	Edit Delete
Room I1	55	None	Active	Edit Delete
Room J1	70	None	Active	Edit Delete

[Back to Dashboard](#)

```

<div class="back-link">
  <a href="{% url 'admin_dashboard' %}">Back to Dashboard</a>
</div>
</body>
</html>

```

Role of manage rooms in Django:

The manage rooms function (view) is responsible for displaying, adding, editing, and deleting rooms in a Django-based Classroom Booking System.

- Fetches all available rooms from the database.
- Displays rooms in a structured table format.
- Allows Admins to add, edit, or delete rooms.
- Ensures only Admin users can access the page.

Signup.html

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

  <!-- Link to Google Fonts for stylish text -->
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;600&family=Roboto:wght@300;400
&display=swap" rel="stylesheet">

<style>
  /* Global Styles */
  body {
    font-family: 'Poppins', sans-serif;
    background-color: #f5f7fa;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    color: #333;
  }

  /* Container Style */
  .container {
    background-color: #fff;
    padding: 40px;
    border-radius: 12px;
    box-shadow: 0 8px 25px rgba(0, 0, 0, 0.1);
    width: 100%;
    max-width: 400px;
    position: relative;
    transform: scale(0);
    animation: fadeInUp 0.8s ease-in-out forwards;
  }

  /* Heading Style */
  h2 {
```

```

    text-align: center;
    font-size: 30px;
    font-weight: 600;
    color: #4CAF50;
    text-transform: uppercase;
    letter-spacing: 1px;
    margin-bottom: 30px;
    animation: slideIn 1s ease-in-out;
}

/* Input Field Styling */
.input-field {

    margin-bottom: 25px;
    position: relative;
}

.input-field input,
.input-field select {
    width: 100%;
    padding: 18px 15px;
    border: 2px solid #ddd;
    border-radius: 10px;
    font-size: 16px;
    background-color: #fafafa;
    transition: border-color 0.3s ease;
    font-family: 'Roboto', sans-serif;
}

.input-field input:focus,
.input-field select:focus {
    border-color: #4CAF50;
    outline: none;
    box-shadow: 0 0 5px rgba(76, 175, 80, 0.5);
}

.input-field label {
    font-size: 14px;
    font-weight: 600;
    color: #777;
    position: absolute;
    left: 15px;
    top: 50%;
    transform: translateY(-50%);
    transition: all 0.3s ease;
    pointer-events: none;
    background-color: #fafafa;
    padding: 0 5px;

```

```

}

.input-field input:focus + label,
.input-field select:focus + label,
.input-field input:not(:placeholder-shown) + label,
.input-field select:not(:placeholder-shown) + label {
    top: -10px;
    font-size: 12px;
    color: #4CAF50;
}

/* Button Styling */
button {
    width: 100%;
    padding: 15px;
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 8px;

    cursor: pointer;
    font-size: 18px;
    font-weight: 600;
    text-transform: uppercase;
    letter-spacing: 1px;
    transition: transform 0.3s ease, background-color 0.3s ease;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
}

button:hover {
    background-color: #45a049;
    transform: scale(1.05);
}

button:active {
    background-color: #388e3c;
}

/* Error and Success Message */
.error, .success {
    font-size: 14px;
    font-weight: 600;
    text-align: center;
    margin-top: 10px;
}

.error {
    color: #f44336;
}

```

```

.success {
    color: #4CAF50;
}

/* Animations */
@keyframes slideIn {
    0% {
        transform: translateY(-20px);
        opacity: 0;
    }
    100% {
        transform: translateY(0);
        opacity: 1;
    }
}

@keyframes fadeInUp {
    0% {
        opacity: 0;
        transform: scale(0.9);
    }
    100% {
        opacity: 1;
        transform: scale(1);
    }
}

}

}

/* Responsive Design */
@media (max-width: 480px) {
    .container {
        padding: 30px;
    }
    h2 {
        font-size: 24px;
    }
    button {
        font-size: 16px;
    }
}
</style>
</head>
<body>

<div class="container">
    <h2>Sign Up</h2>

    <!-- Form Starts -->

```

```

<form method="POST" onsubmit="return validateForm()">
  {% csrf_token %}

  <!-- Username Field -->
  <div class="input-field">
    <input type="text" name="username" required>
    <label for="username">Username</label>
  </div>

  <!-- Email Field -->
  <div class="input-field">
    <input type="email" name="email" required>
    <label for="email">Email</label>
  </div>

  <!-- Password Field -->
  <div class="input-field">
    <input type="password" name="password1" required>
    <label for="password1">Password</label>
  </div>

  <!-- Confirm Password Field -->
  <div class="input-field">
    <input type="password" name="password2" required>
    <label for="password2">Confirm Password</label>
  </div>

  <!-- Role Selection -->
  <div class="input-field">
    <select name="role" required>
      <option value="">Select your role</option>

      <option value="Teacher">Teacher</option>
      <option value="Student">Student</option>
    </select>
    <label for="role">Role</label>
  </div>

  <!-- Submit Button -->
  <button type="submit">Sign Up</button>

  <!-- Error and Success Messages -->
  {% if messages %}
    <div class="messages">
      {% for message in messages %}
        <div class="{{ message.tags }}">{{ message }}</div>
      {% endfor %}
    </div>
  {% endif %}

```

```
</form>
</div>

<!-- JavaScript for real-time validation -->
<script>
    function validateForm() {
        var role = document.querySelector('select[name="role"]').value;
        if (!role) {
            alert("Please select a role.");
            return false;
        }
        return true;
    }
</script>

</body>
</html>
```

OUTPUT:

SIGN UP

Username

Email

Password

Confirm Password

Role

Select your role ▼

SIGN UP

Role of Signup (signup _view) in Django

The signup view allows new users to create an account in the Classroom Booking System. During signup, users can choose a role (Admin, Teacher, or Student), and their account is assigned to the corresponding group

- Allows new users to register.
- Uses Django's User Creation Form for account creation.
- Assigns a role (Admin, Teacher, Student) to the user.
- Logs in the user automatically after signup.
- Redirects users based on their role.
- Ensures authenticated users cannot access signup again.

Student_dashboard.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Student Dashboard</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background: linear-gradient(135deg, #2b5876, #4e4376);
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
      text-align: center;
      color: white;
    }

    .container {
      background: rgba(255, 255, 255, 0.2);
      padding: 30px;
      border-radius: 15px;
      box-shadow: 0 8px 20px rgba(0, 0, 0, 0.4);
      backdrop-filter: blur(10px);
      width: 500px;
      max-width: 90%;
    }

    h1, h2 {
      margin: 15px 0;
      font-size: 28px;
      color: #ffcc00;
    }

    p {
      margin: 8px 0;
      font-size: 18px;
    }

    table {
      width: 100%;
      border-collapse: collapse;
      margin: 15px 0;

```



```

}

th, td {
    padding: 10px;
    border: 1px solid #ddd;
    color: white;
}

th {
    background-color: #333;
    color: #ffcc00;
}

tr:nth-child(even) {
    background-color: rgba(255, 255, 255, 0.1);
}

.no-data {
    margin: 15px 0;
    font-size: 20px;
    color: #ffcc00;
}

a {
    display: inline-block;
    margin: 15px 10px;
    padding: 12px 20px;
    color: #ffffff;
    background-color: #ffcc00;
    text-decoration: none;
    border-radius: 8px;
    transition: background-color 0.3s ease;
}

a:hover {
    background-color: #e6b800;
}
</style>
</head>
<body>
<div class="container">
    <h1>Student Dashboard</h1>
    <p>Welcome, {{ user.username }} (Student)</p>

    <h2>Rooms Booked by Teachers</h2>
    {% if teacher_bookings %}
        <table>
            <tr>
                <th>Room</th>

```

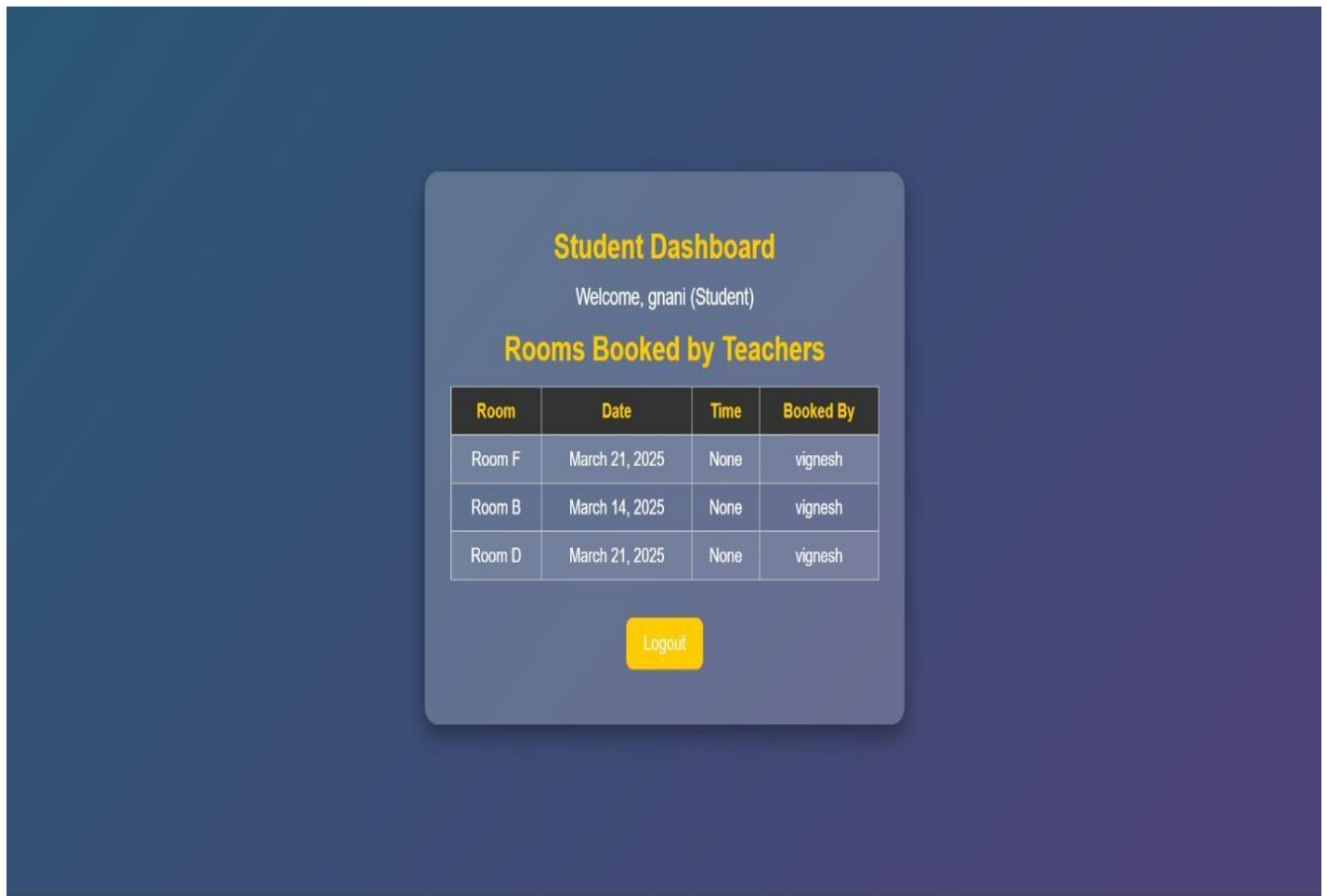
```

        <th>Date</th>
        <th>Time</th>
        <th>Booked By</th>
    </tr>
    {% for booking in teacher_bookings %}
        <tr>
            <td>{{ booking.room.name }}</td>
            <td>{{ booking.date }}</td>
            <td>{{ booking.time_slot }}</td>
            <td>{{ booking.user.username }}</td>
        </tr>
    {% endfor %}
</table>
{% else %}

    <p class="no-data">No rooms booked by teachers yet.</p>
{% endif %}

    <a href="{% url 'logout' %}">Logout</a>
</div>
</body>
</html>

```

_OUTPUT:**Role of student_dashboard.html in Django:**

The Student Dashboard (student_dashboard.html) is the homepage for students after they log in. It allows students to:

- See classroom bookings made by teachers
- Prevent unauthorized access (only students can view it)

Teacher_dashboard.html:

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

```

<title>Teacher Dashboard</title>
<style>
  body {
    font-family: 'Arial', sans-serif;
    background: linear-gradient(135deg, #ff6a00, #ee0979);
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
    text-align: center;
    color: white;
    padding: 0;
  }
  .container {
    background: rgba(255, 255, 255, 0.2);
    padding: 40px;
    border-radius: 15px;
    box-shadow: 0 8px 30px rgba(0, 0, 0, 0.4);
    backdrop-filter: blur(15px);
    width: 450px;
    transition: all 0.3s ease;
  }
  .container:hover {
    transform: scale(1.05);
  }
  h1 {
    font-size: 36px;
    margin-bottom: 20px;
    color: #fff;
  }
  a {
    display: block;
    margin: 12px 0;
    color: #ffcc00;
    text-decoration: none;
    font-size: 18px;
    padding: 10px;
    border-radius: 5px;
    background-color: rgba(255, 255, 255, 0.2);
    transition: all 0.3s ease;
  }
  a:hover {
    color: #fff;
    background-color: rgba(255, 255, 255, 0.4);
  }
  .welcome-message {

    margin-bottom: 25px;

```

```

        font-size: 22px;
        font-weight: bold;
    }
    .info-box {
        background-color: rgba(0, 0, 0, 0.5);
        padding: 15px;
        margin: 15px 0;
        border-radius: 8px;
        font-size: 16px;
    }
    .info-box p {
        margin: 0;
    }
</style>
</head>
<body>
    <div class="container">
        <h1>Teacher Dashboard</h1>

        {% if user.is_authenticated %}
            <p class="welcome-message">Welcome, {{ user.username }} (Teacher)</p>

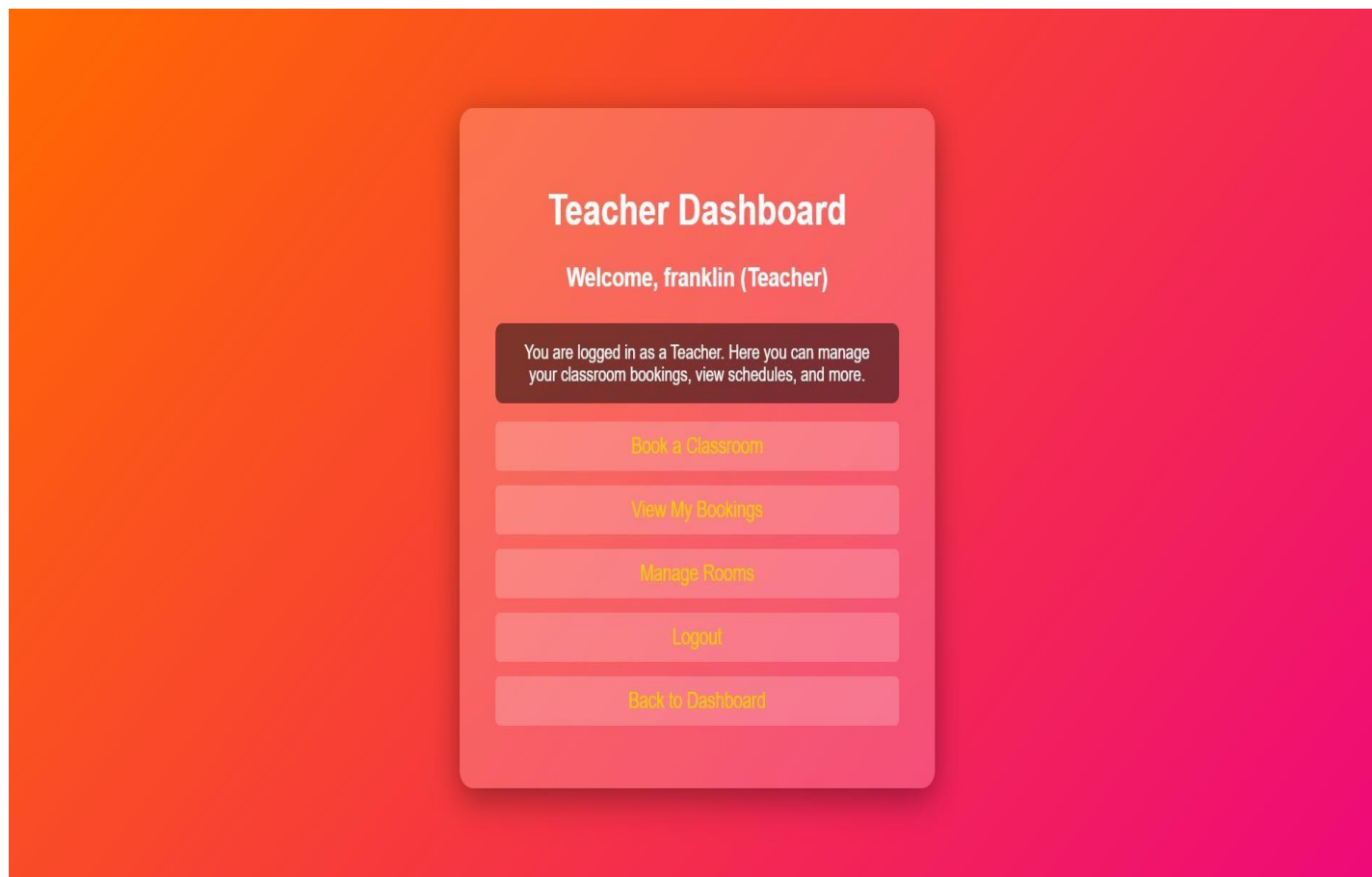
            <!-- Info Box to show the role and any important message -->
            <div class="info-box">
                <p>You are logged in as a Teacher. Here you can manage your classroom bookings, view
schedules, and more.</p>
            </div>

            <!-- Links for teacher actions -->
            <a href="{% url 'book_room' %}">Book a Classroom</a>
            <a href="{% url 'view_schedule' %}">View My Bookings</a>
            <a href="{% url 'manage_rooms' %}">Manage Rooms</a>
            <a href="{% url 'logout' %}">Logout</a>

            <!-- Back to Teacher Dashboard Link -->
            <a href="{% url 'teacher_dashboard' %}">Back to Dashboard</a>
        {% else %}
            <p>You must be logged in to access the Teacher Dashboard.</p>
            <a href="{% url 'login' %}">Login</a>
        {% endif %}
    </div>
</body>
</html>

```

OUTPUT:



Role of teacher_dashboard.html in Django

The Teacher Dashboard (teacher_dashboard.html) is the homepage for teachers after they log in. It allows teachers to:

- View their own bookings
- Manage classroom bookings (Book and Cancel)

View_schedule.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Schedule</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background: linear-gradient(135deg, #1e3c72, #2a5298);
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      flex-direction: column;
      margin: 0;
    }

    h1 {
      color: #fff;
      text-align: center;
      margin-bottom: 20px;
      animation: fadeIn 1.5s ease-in-out;
    }

    table {
      width: 80%;
      max-width: 800px;
      border-collapse: collapse;
      background: rgba(255, 255, 255, 0.2);
      border-radius: 10px;
      backdrop-filter: blur(10px);
      overflow: hidden;
      box-shadow: 0 4px 10px rgba(0, 0, 0, 0.2);
      animation: fadeIn 2s ease-in-out;
    }

    th, td {
      padding: 12px;

```

```

    text-align: center;
    color: white;
}

th {
    background: rgba(255, 255, 255, 0.3);
}

tr:hover {
    background: rgba(255, 255, 255, 0.1);
    transition: 0.3s ease-in-out;
}

```

```

.back-home {
    margin-top: 20px;
    padding: 10px 20px;
    background: #ff8c00;
    color: white;
    text-decoration: none;
    border-radius: 5px;
    transition: transform 0.3s ease-in-out, background 0.3s ease-in-out;
    animation: fadeIn 2.5s ease-in-out;
}

```

```

.back-home:hover {
    background: #ff6600;
    transform: scale(1.1);
}

```

```

/* Fade-in animation */
@keyframes fadeIn {
    from { opacity: 0; transform: translateY(-20px); }
    to { opacity: 1; transform: translateY(0); }
}

```

```

</style>
</head>
<body>
<h1>My Schedule</h1>
<table>
  <thead>
    <tr>
      <th>Classroom</th>
      <th>Date</th>
      <th>Booked By</th>
    </tr>
  </thead>
  <tbody>
    {% for booking in bookings %}

```



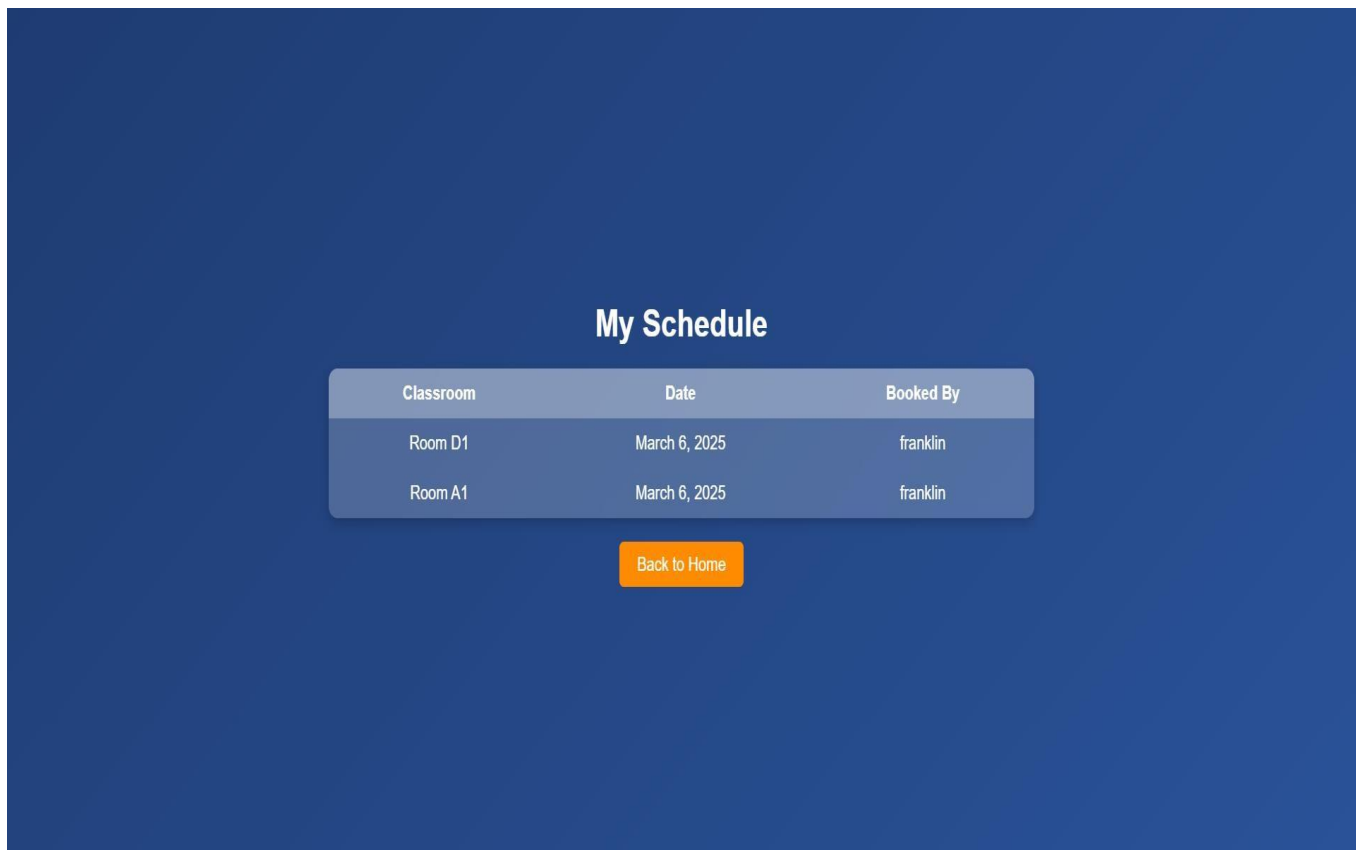
```

        <tr>
            <td>{{ booking.room.name }}</td>
            <td>{{ booking.date }}</td>
            <td>{{ booking.booked_by }}</td>
        </tr>
    {% endfor %}
</tbody>
</table>

<a href="{% url 'home' %}" class="back-home">Back to Home</a>
</body>
</html>

```

OUTPUT:



Role of view_schedule.html in Django:

- See their own booked classrooms
- Check the date and room details of their bookings
- Ensure only logged-in users can access it



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Database Integration and Configuration-SQL LITE
7. Date of Experiment : 17-02-2025
8. Date of Submission of Report : 21-02-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Database Integration and Configuration-SQL LITE

Databasse is an organized collection of data that is stored and managed in a way that makes it easy to retrieve, update, and manage information. Databases are used to store data in a structured format, allowing efficient access, management, and modification.

Step 1: Check if SQLite3 is Already Installed

`sqlite3 --version`

Step 2: Install SQLite3 (if not already installed)

On Windows:

1. Download the SQLite3 command-line tool from the official website: [SQLite Downloads](#)
2. Download the "Precompiled Binaries for Windows" (usually a ZIP file).
3. Extract the ZIP file and place `sqlite3.exe` in a folder (e.g., `C:\sqlite`).
4. Add the folder to your system's PATH environment variable:
 - Search for "Environment Variables" in the Start menu.
 - Click on "Environment Variables."
 - In "System variables," select "Path" and click "Edit."
 - Add the folder path (e.g., `C:\sqlite`) and click OK.

On macOS:

`sqlite3 --version`

Step 4: Using SQLite3 with Django

Since Django uses SQLite3 as the default database, you don't need to install any additional drivers. Just make sure your `settings.py` file has the following configuration:

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

- Step 5: Run Migrations to Create the Database
- After setting up, run:
 - `python manage.py migrate`
- Step 1: Open the Django DB Shell
 - Make sure your virtual environment is activated, then use the following command:
 - `python manage.py db shell`
- Step 2: Common SQLite Commands
 - Once you're inside the SQLite shell, you can use the following commands:
 - List All Tables:
 - `.tables`
 - Step3: View Table Schema:
 - `.schema table_name;`
 - Step4: Show All Data in a Table:
 - `SELECT * FROM table_name;`
 - Exit the SQLite Shell:
 - `.exit`



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Handling Forms in Django
7. Date of Experiment : 21-02-2025
8. Date of Submission of Report : 21-02-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Handling forms.py in Django:

In Django, forms.py is used to handle user input efficiently and securely. It allows developers to create and manage forms without manually writing HTML and validation logic.

Why Use forms.py:

- Simplifies form creation
- Handles input validation automatically
- Integrates with Django models
- Prevents security risks like SQL Injection & CSRF attacks

Types of Forms in Django:

- Django Forms (forms.Form) – Used for manually creating forms
- Model Forms (forms.ModelForm) – Used to create forms directly from a Django model

myapp1/forms.py

from django import forms

from .models import Room, Booking

from datetime import time

class RoomForm(forms.ModelForm):

class Meta:

model = Room

fields = ['name', 'capacity', 'teacher', 'is_active'] # Include the fields you need

class BookingForm(forms.ModelForm):

class Meta:

model = Booking

fields = ['room', 'date', 'time_slot', 'booked_by', 'user']

time_slot = forms.TimeField(required=True) # Make it mandatory to catch errors



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Defining and Using Models
7. Date of Experiment : 21-02-2025
8. Date of Submission of Report : 07-03-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUTION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Defining and Using Models:

In Django, models.py is where you define the database structure using Python code. Django models act as a bridge between the database and the application, allowing you to create, read, update, and delete records easily.

Why Use Django Models:

No need to write raw SQL queries, Automatically creates tables in the database

Apply Models:

Create the Model in models.py:

- Write your models inside the models.py file.

MODELS.PY:

```
from django.db import models
from django.contrib.auth.models import User
from django.core.mail import send_mail
from django.conf import settings

class Room(models.Model):
    name = models.CharField(max_length=100)
    capacity = models.IntegerField()
    teacher = models.ForeignKey(User, on_delete=models.CASCADE, related_name='rooms', null=True, blank=True)
    is_active = models.BooleanField(default=True)
    def __str__(self):
        return self.name

class Booking(models.Model):
    room = models.ForeignKey(Room, on_delete=models.CASCADE)
    date = models.DateField()
    time_slot = models.TimeField() # Single time field for booking
    booked_by = models.CharField(max_length=100)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
```



```
email = models.EmailField()
```

```
def __str__(self):
```

```
    return f'Booking for {self.room} on {self.date} at {self.time_slot} by {self.user.username}'
```

```
def send_booking_email(self):
```

```
    subject = 'Room Booking Confirmation'
```

```
message = f"Your room booking for {self.room.name} on {self.date} at {self.time_slot} is confirmed." email_from = settings.EMAIL_HOST
```



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Migrations: Sync with the Database
7. Date of Experiment : 07-03-2025
8. Date of Submission of Report : 27-03-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUATION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Migrations: Sync with the Database

Run Migrations to Create Database Tables:

After defining your models, run the following commands to apply them to the database:

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

Register Models in admin.py:

Admin.py:

```
from django.contrib import admin
from .models import Room, Booking
class RoomAdmin(admin.ModelAdmin):
    list_display = ('name', 'capacity', 'teacher', 'is_active') # Display relevant fields in the list
    search_fields = ('name', 'teacher__username') # Allow search by room name and teacher's
    username
    list_filter = ('is_active', 'teacher') # Add filters for active status and teacher

class BookingAdmin(admin.ModelAdmin):
    list_display = ('room', 'user', 'date', 'booked_by') # Display room, user, date, and who booked the
    room
    list_filter = ('room', 'user', 'date') # Add filters for room, user, and date
    search_fields = ('room__name', 'user__username') # Allow search by room name and user

# Register the models with custom admin classes
admin.site.register(Room, RoomAdmin)
admin.site.register(Booking, BookingAdmin)
```



**DEPARTMENT OF INFORMATION TECHNOLOGY
JNTU-GURAJADA VIZIANAGARAM
COLLEGE OF ENGINEERING VIZIANAGARAM (A)
VIZIANAGARAM**

Dr.Ch. Bindu Madhuri

Asst. Professor & HOD

Email: hod.it@intugvcev.edu.in

1. Name of the Laboratory : Django Frame Work Lab
2. Name of the Student : M.J.Franklin
3. Roll No : 23VV1A1231
4. Class : II B-Tech II Semester
5. Academic Year : 2024-25
6. Name of Experiment : Deploying Django Applications on Cloud Platforms
7. Date of Experiment : 27-03-2025
8. Date of Submission of Report : 04-04-2025

S,NO	ABILITY AND ACTIVITY	WEIGHTAGE OF MARKS	DAY TO DAY EVALUATION SCORE
1	Aim Objective, Tools required	3	
2	Theory, Algorithm and Observations	3	
3	Implementation	3	
4	Schematic diagrams, Architecture, workflow, Flowchart	3	
5	Tidiness of his/her working area, proper maintenance of system during and after experiment.	3	
	Total Score	15	

DATE:

Signature of Faculty

Deploying Django Web Application on Cloud

What is Deployment?

- Deployment is the process of making a Django web application live on the internet so users can access it. This involves hosting your app on a cloud server like AWS, Google Cloud, Digital Ocean, Heroku, or PythonAnywhere.
- Features:
 - Scalability – Handle more users without performance issues.
 - Security – Protect user data with SSL and secure databases.
 - Global Accessibility – Users can access your app from anywhere.
 - Continuous Deployment – Easily update your app with new features.
 - Here's a step-by-step guide to Register on GitHub, Create a Django website with login and registration pages, and Configure Django to handle static files.

Step 1: Register on GitHub

- Go to [GitHub](#) and click Sign up.
- Enter your Username, Email, and Password.
- Complete the verification and click Create Account.
- Verify your email by clicking the link in your inbox.

Step 2: Push to GitHub

- Initialize Git in your project: `git init`
- Connect to GitHub:
- `git remote add origin`
- `https://github.com/haridammu/Classroom_Booking_system.git`
- Add and commit changes: `git add .`
- `git commit -m "Initial Commit: Login and Registration App"`
- Push to GitHub: `git branch -M main`
- `git push -u origin main`
- You have successfully built a Django website with login, registration, and static file management.
- Your code is now available on GitHub.
- **GITHUB LINK:**
https://github.com/haridammu/Classroom_Booking_System.git



CERTIFICATE OF ACHIEVEMENT

The certificate is awarded to

JOSHUA FRANKLIN

for successfully completing

Front End Web Developer Certification

on February 23, 2025



Congratulations! You make us proud!



Issued on: Sunday, February 23, 2025

To verify, scan the QR code at <https://verify.onwingspan.com>

Thirumala Arohi
Executive Vice President and Global Head
Education, Training & Assessment (ETA)
Infosys Limited