**DEPARTMENT OF INFORMATION TECHNOLOGY**

**JNTU-GURAJADA VIZIANAGARAM**

**COLLEGE OF ENGINEERING VIZIANAGARAM (A)**

**VIZIANAGARAM**

****

**DJANGO FRAMEWORK LAB**

**CAMPUS HEALTH AND WELLNESS PORTAL**

**DONE BY**

**P.SRAVAN KUMAR (23VV1A1240)**

Y.GHANA SATYA KARTHIK (23VV1A1263)

V.CHARU VRUDHA HASINI (23VV1A1254)

M.LOKESH (23VV1A1227)

**UNDER THE GUIDANCE OF**

**MRS.MADHUMITHA CHANDA**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

# Python Libraries

Python libraries are pre-written code that provides a set of functionalities, making it easier to perform specific tasks. They are reusable, well-tested, and widely adopted, saving developers time and effort.

## Tkinter - GUI:

* **Purpose:** Python’s standard library for creating graphical user interfaces (GUIs).
* **Key Features**:

1. Widgets: Buttons, labels, text boxes, etc.
2. Event handling: Respond to user interactions like clicks or key presses.
3. Simple layout management.

**Common Use**: Build desktop applications or tools for local interaction with a web app backend.

**implement the code we need to install the tinker library:**

***pip install tk***

***from tkinter import Tk, Label***

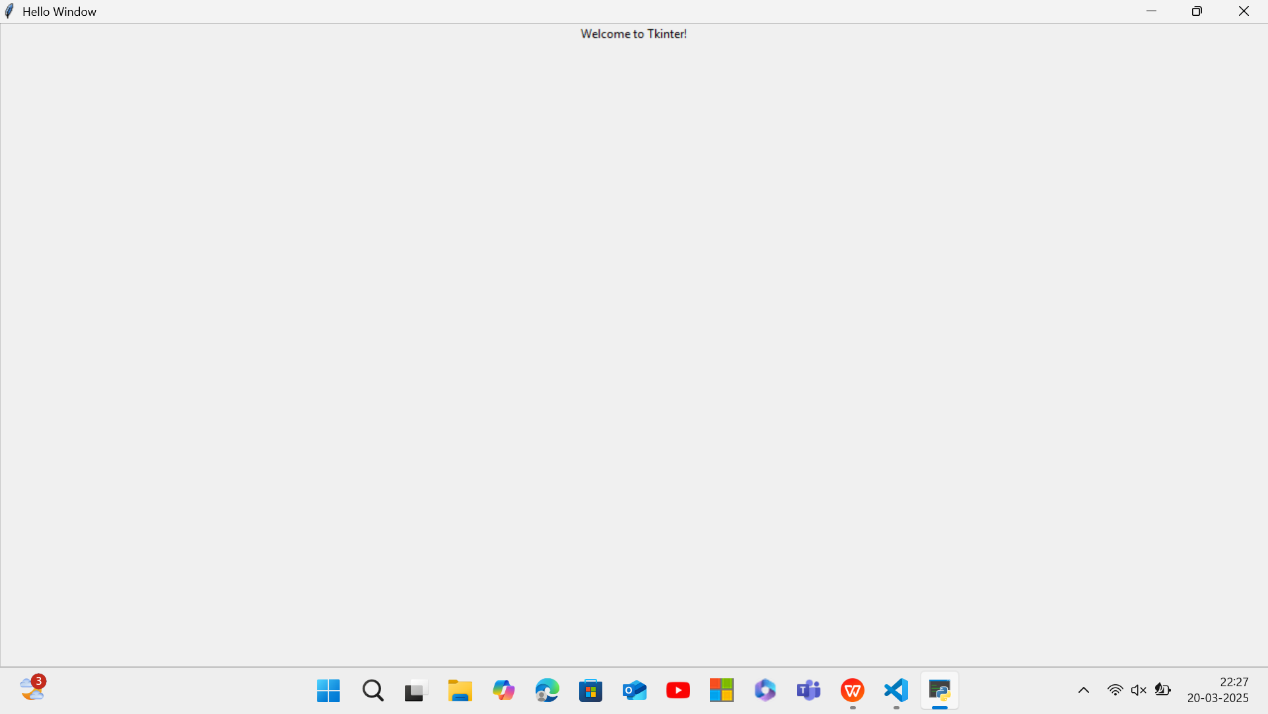
***root = Tk()***

***root.title("Hello Window")***

***Label(root, text="Welcome to Tkinter!").pack()***

***root.mainloop()***

**Output:**



**Requests - HTTP Requests**

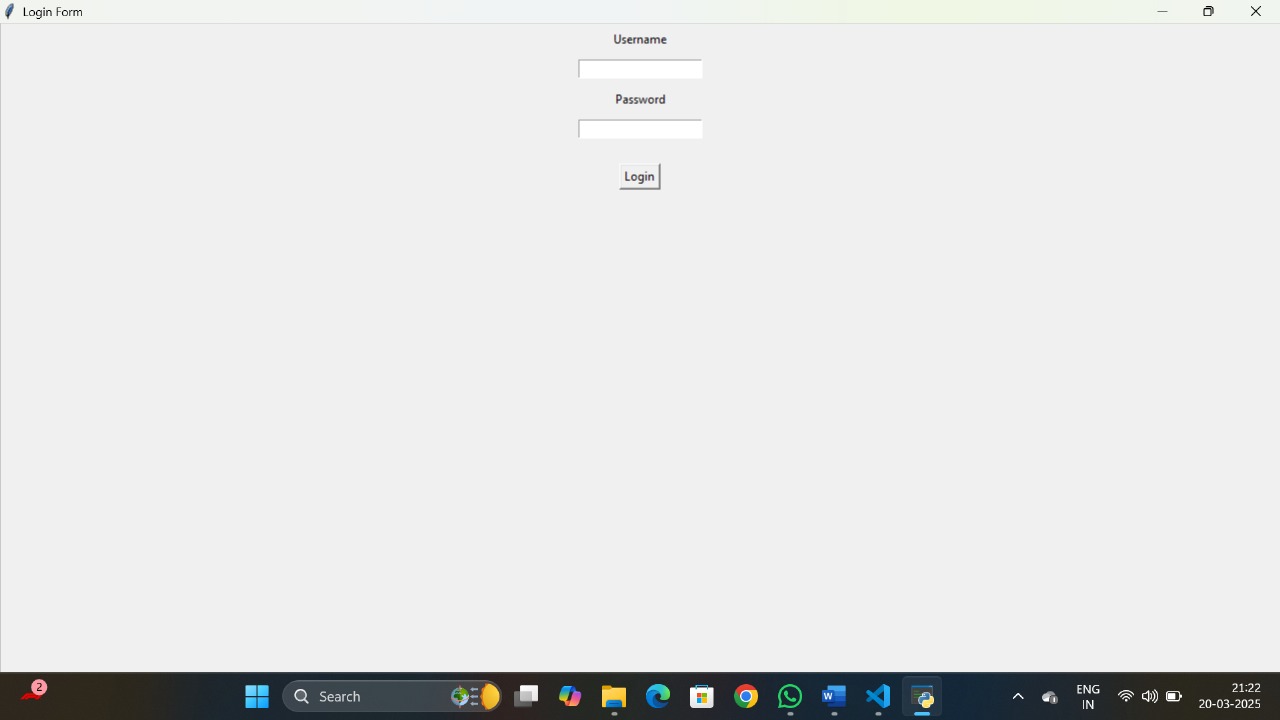
* **Purpose**: Simplifies HTTP requests to interact with web APIs.
* **Key Features**:
  + Send GET, POST, PUT, DELETE requests easily.
  + Handle request parameters, headers, and cookies.
  + Simple error handling and response handling.
* **Common Use**: Interact with REST APIs, download content from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

**To implement the code we need to install the tkinter library:**

|  |
| --- |
| ***pip install tk*** |

**Code:**

|  |
| --- |
| ***import tkinter as tk***  ***from tkinter import messagebox***  ***def validate\_login():***  ***username = username\_entry.get()***  ***p***  ***assword = password\_entry.get()***  ***# Example credentials***  ***if username == "user" and password == "password":***  ***messagebox.showinfo("Login Success", "Login Successful!")***  ***else:***  ***messagebox.showerror("Login Failed", "Invalid username or password")***  ***# Create the main window***  ***root = tk.Tk()***  ***root.title("Login Form")***  ***root.geometry("1080x720")***  ***# Create username and password labels and entry widgets***  ***username\_label = tk.Label(root, text="Username")***  ***username\_label.pack(pady=5)***  ***username\_entry = tk.Entry(root)***  ***username\_entry.pack(pady=5)***  ***password\_label = tk.Label(root, text="Password")***  ***password\_label.pack(pady=5)***  ***password\_entry = tk.Entry(root, show="\*") # 'show' hides the password characters***  ***password\_entry.pack(pady=5)***  ***# Create the login button***  ***login\_button = tk.Button(root, text="Login", command=validate\_login)***  ***login\_button.pack(pady=20)***  ***# Run the Tkinter event loop***  ***root.mainloop()*** |

**Output:**

## BeautifulSoup4 - Web Scraping

**Purpose**: Parses HTML and XML documents to extract data.

**Key Features**:

* + Easy navigation and searching within HTML.
  + Supports different parsers like html.parser, lxml, and html5lib.

**Common Use:** Extract data from websites for analysis, e.g., for building data-driven applications.

**To implement the code we need to install the beautifulsoup4 library:**

***pip install bs4***

***(or)***

***pip install beautifulsoup4***

**Code:**

***import requests***

***from bs4 import BeautifulSoup***

***# The URL of the website to scrape***

***url = 'https://example.com' # Replace with the website you want to scrape***

***# Send a GET request to fetch the raw HTML content***

***response = requests.get(url)***

***# Parse the raw HTML content with BeautifulSoup***

***soup = BeautifulSoup(response.text, 'html.parser')***

***# Find the title of the webpage***

***title = soup.title.string***

***print(f"Title of the page: {title}")***

***# Find all headings (e.g., <h1>, <h2>, <h3>, etc.)***

***headings = soup.find\_all(['h1', 'h2', 'h3'])***

***for heading in headings:***

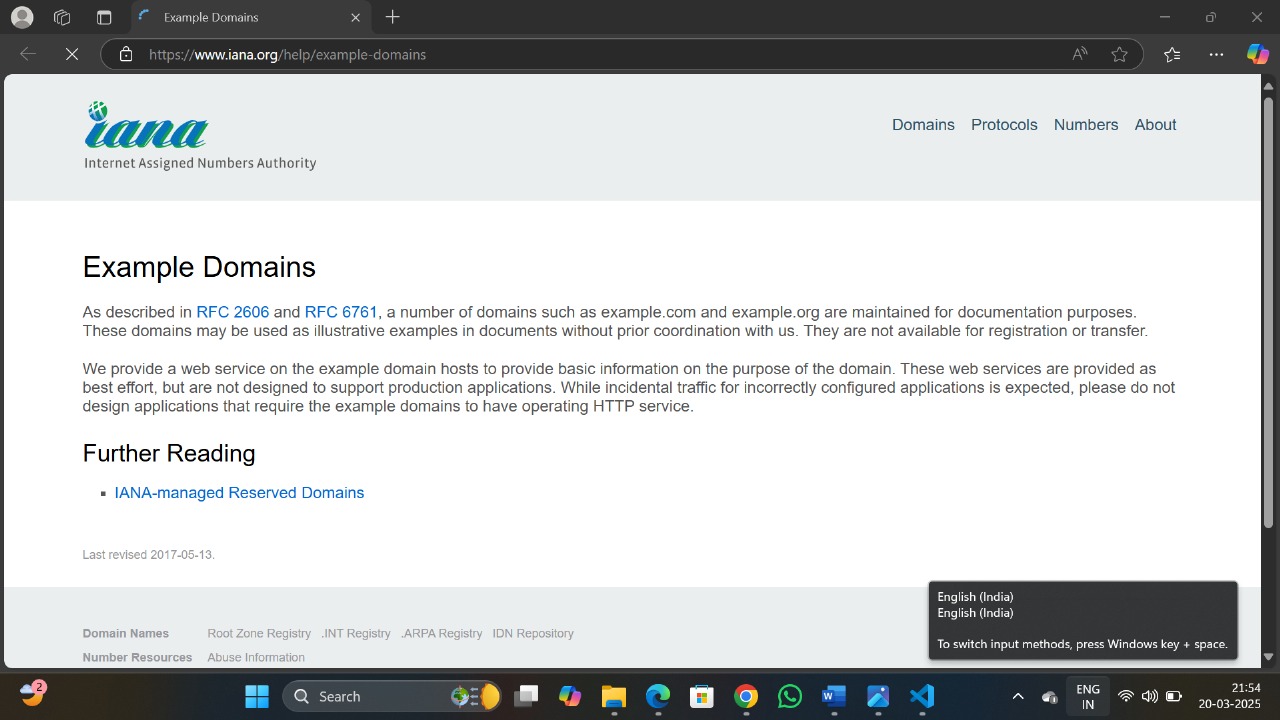
***print(heading.text.strip()) # Print the heading text***

***# Find all links (<a> tags) on the page***

***links = soup.find\_all('a', href=True)***

***for link in links:***

***print(f"Link: {link['href']}")***

**Output:**

## CherryPy

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

**To implement the code we need to install the cherry py library:**

***pip install cherrypy***

**Code:**

***import cherrypy***

***class HelloWorld(object):***

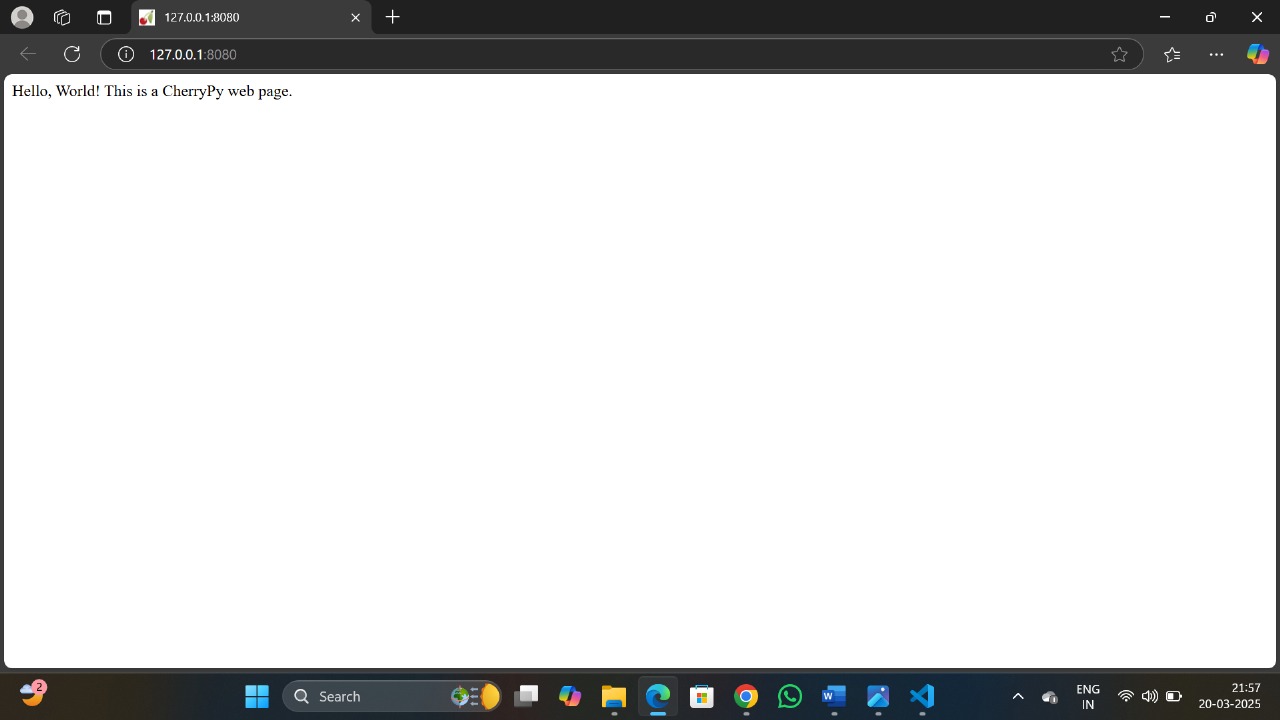
***@cherrypy.expose***

***def index(self):***

***return "Hello, World! This is a CherryPy web page."***

***if \_\_name\_\_ == '\_\_main\_\_':***

***cherrypy.quickstart(HelloWorld())***

**Output:**

## Flask

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

**To implement the code we need to install the flask library:**

***pip install flask***

**Code:**

***from flask import Flask***

***# Create a Flask application instance***

***app = Flask(\_\_name\_\_)***

***# Define a route for the root URL ("/")***

***@app.route('/')***

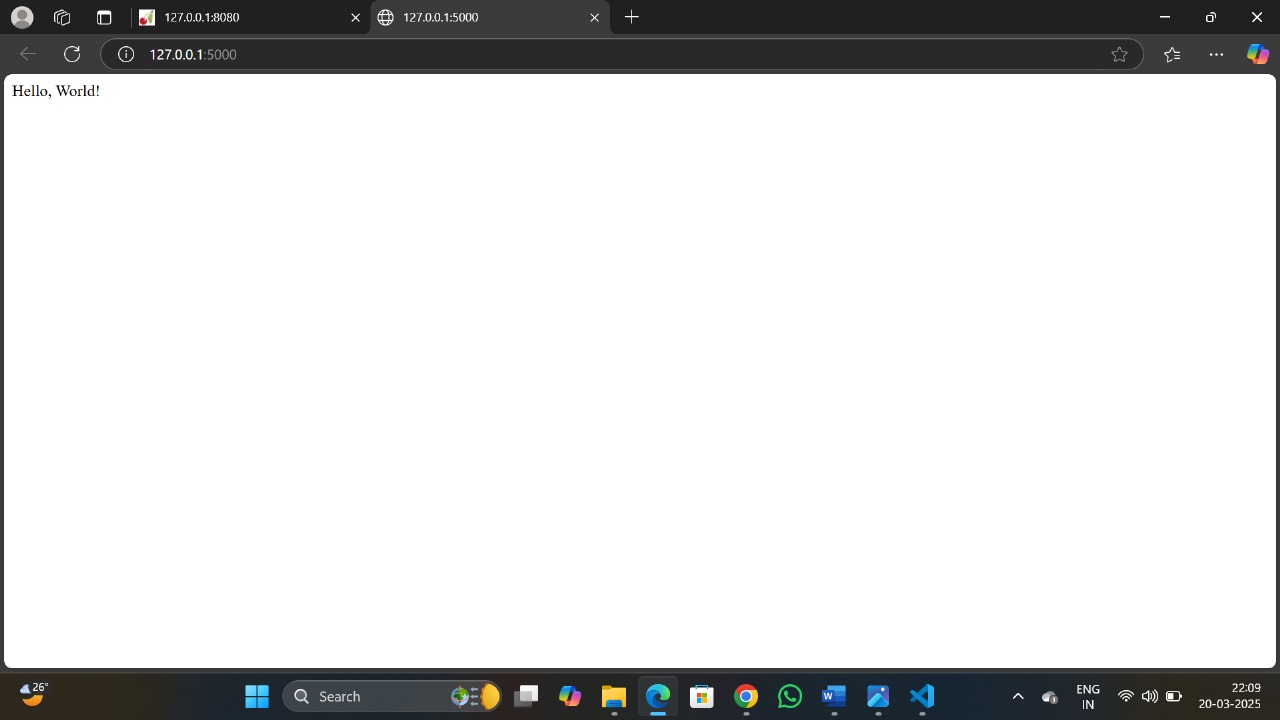
***def hello\_world():***

***return 'Hello, World!'***

***# Run the Flask application***

***if \_\_name\_\_ == '\_\_main\_\_':***

***app.run(debug=True)***

**Output:**

## Bottle

**Purpose**: Simple and lightweight WSGI micro-framework.

* **Key Features**:
  + Single-file framework, minimalistic, and fast.
  + No dependencies, supports routing, templates, and form handling.
* **Common Use**: Small web applications, APIs, and prototypes.

**To implement the code we need to install the bottle library:**

***pip install bottle***

**Code:**

***from bottle import route, run***

***@route('/')***

***def index():***

***return "Hello, World! This is a Bottle web page."***

***run(host='localhost', port=8080)***

**Output:**



# 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 Installation:

Creating a Django project involves a series of steps, from setting up your environment to running your initial server. Here's a breakdown of the process:

## Prerequisites:

* **Python:** Ensure you have Python 3.6 or later installed.
* **pip:** Python's package installer, pip, should also be installed.

1. **Create a Virtual Environment (Recommended):**

* Virtual environments isolate your project's dependencies. This prevents conflicts between different projects.
  + Open your terminal or command prompt.
  + Navigate to the directory where you want to create your project.
  + Create a virtual environment:
    - python -m venv venv (On Windows)
    - python3 -m venv venv (On macOS/Linux)
  + Activate the virtual environment:
    - venv\Scripts\activate (On Windows)
    - source venv/bin/activate (On macOS/Linux)

1. **Install Django:**

* With your virtual environment activated, install Django using pip:

pip install Django

1. **Create the Django Project:**

* Use the django-admin command to create your project:

django-admin startproject myproject (Replace "myproject" with your desired project name)

* + This creates a directory named "myproject" containing the project's initial files.

1. **Navigate to the Project Directory:**

* Change your current directory to the newly created project directory:

cd myproject

1. **Run Migrations:**

* Django uses migrations to manage database changes. Run the initial migrations:

python manage.py migrate

1. **Start the Development Server:**

* Start the Django development server:

python manage.py runserver

* + This will start a local server, and you can access your Django project in your web browser at http://127.0.0.1:8000/.

1. **Create a Django App:**

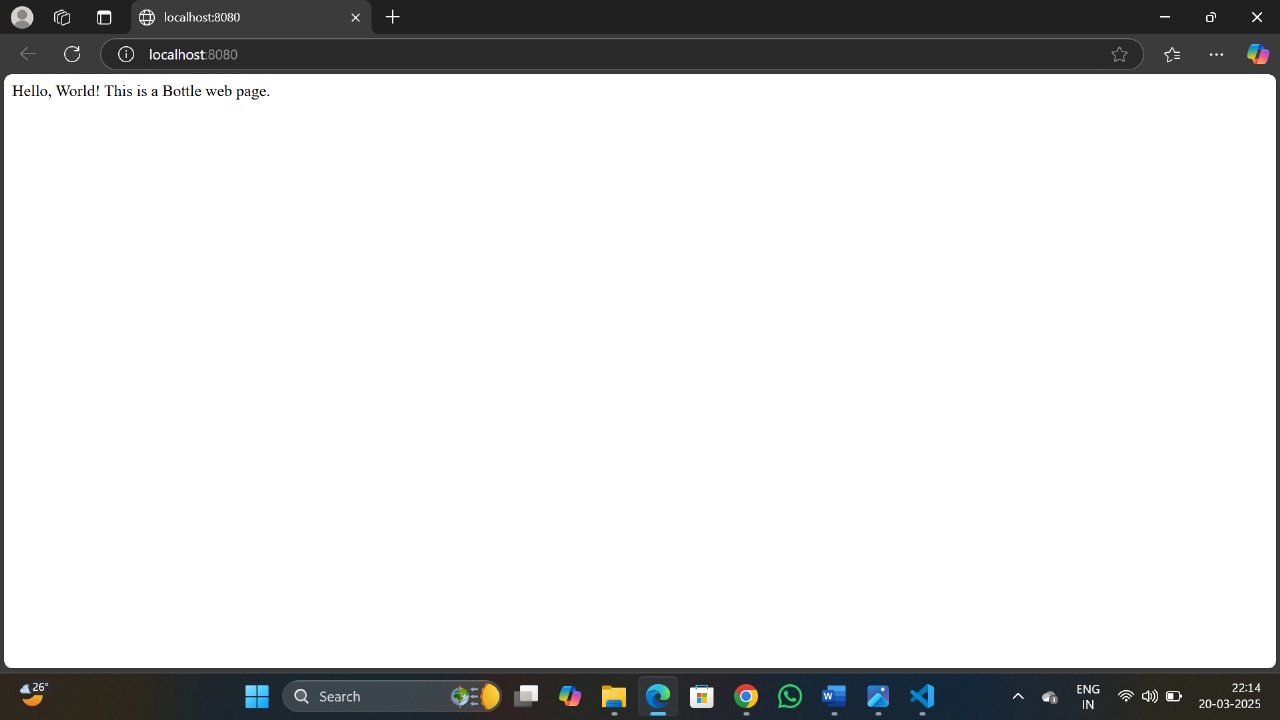
* A django project is made up of apps. to create an app within the project use this command.

python manage.py startapp myapp (Replace "myapp" with your desired app name)

## Key Files and Directories:

* **manage.py:** A command-line utility for administrative tasks.
* **myproject/:** The project's main directory, containing:
  + **\_\_init\_\_.py:** An empty file that tells Python this directory is a Python package.
  + **asgi.py:** Asynchronous Server Gateway Interface configuration.
  + **settings.py:** Project settings, including database configuration and installed apps.
  + **urls.py:** URL routing configuration.
  + **wsgi.py:** Web Server Gateway Interface configuration.
* **myapp:** the app directory, containing:
  + **\_\_init\_\_.py**
  + **admin.py**
  + **apps.py**
  + **models.py**
  + **tests.py**
  + **views.py**

By following these steps, you'll have a basic Django project up and running. From there, you can start building your web application by defining models, views, and templates.

****

## Django’s MVT Architecture:

Django follows the MVT (Model-View-Template**)** architecture, which is a variation of the traditional MVC (Model-View-Controller) pattern. It helps in separating concerns, making Django projects scalable, maintainable, and organized.

* **Model (M) – Data Layer:**

The **Model** handles **database interactions** and represents the data structure. It defines how data is stored, retrieved, and managed in the database.

* View (V) – Business Logic Layer

The **Model** handles **database interactions** and represents the data structure. It defines how data is stored, retrieved, and managed in the database.

* Template (T) – Renders HTML pages dynamically.

The Template is responsible for rendering the HTML and displaying data dynamically.

**Example MVT Folder Structure in Django**

myproject/ # Root project folder

│

├── manage.py # Django's command-line utility

├── db.sqlite3 # Default database (can vary)

│

├── myproject/ # Project configuration folder (same name as the project)

│ ├── \_\_init\_\_.py

│ ├── settings.py # Project settings

│ ├── urls.py # Project-level URL configurations

│ ├── asgi.py

│ └── wsgi.py

│

├── myapp/ # A Django app (can have multiple apps)

│ ├── \_\_init\_\_.py

│ ├── admin.py # Admin interface configuration

│ ├── apps.py

│ ├── models.py # Models (M in MVT)

│ ├── views.py # Views (V in MVT)

│ ├── urls.py # App-level URL configurations

│ ├── tests.py

│ ├── forms.py # Forms (optional, for user input handling)

│ ├── templates/ # Templates folder (T in MVT)

│ │ └── myapp/ # Match app name to avoid template name conflicts

│ │ └── home.html # Example HTML template

│ └── static/ # Static files (CSS, JS, images)

│ └── myapp/

│ └── style.css

│

└── templates/ # (Optional) Project-wide templates

└── base.html # Base template to extend

Some Key Points:

* Django follows the MVT architecture — Model, View, Template.
* The Model (models.py) defines the structure of the database using Python classes.
* The View (views.py) contains functions or classes that handle business logic and return responses.
* The Template (templates/) contains HTML files that define the UI shown to users.
* Each app has its own folder with files like models.py, views.py, urls.py, and templates/.
* Static files like CSS, JS, and images go inside the static/ folder.
* urls.py (in both project and app) defines URL patterns for routing requests.
* settings.py holds the configuration for apps, databases, templates, and more.
* manage.py is a command-line tool to interact with the project (runserver, migrate, etc.).

This structure keeps the project organized, scalable, and easy to manage.

# 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

# Exploring Django VIews:

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.

***from django.shortcuts import render***

***from django.http import HttpResponse***

***from django.template import loader***

***from HealthHive.models import Disease***

***from django.shortcuts import render***

***def home(request):***

***featured\_diseases = Disease.objects.filter(featured=True)[:4]***

***print("Featured Diseases:", featured\_diseases)***

***return render(request, 'Homepage.html', {'featured\_diseases': featured\_diseases})***

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.

## 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 django.conf.urls.static import static***

***from django.conf import settings***

***urlpatterns = [***

***path('', include('homepage.urls')),***

***path('admin/', admin.site.urls),***

***path('accounts/', include('accounts.urls')),***

***path('appointments/', include('Appointments.urls')),***

***path('healthhive/', include('HealthHive.urls')),***

***] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)***

In Django, the connection between app URLs and project URLs is a fundamental aspect of how URL routing works. It's designed to keep your project organized and maintainable, especially as it grows in complexity. Here's a breakdown of how it works:

Modularity:

* Apps can be developed and reused independently.

Organization:

* URL configurations are kept organized and manageable.You can use namespaces to avoid URL name conflicts between apps.

# 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.

App1 Urls.Py :

***from django.urls import path***

***from . import views***

***from django.contrib.auth import views as auth\_views***

***from .views import add\_disease***

***app\_name = 'accounts'***

***urlpatterns = [***

***path('login/', views.login, name='login'),***

***path('registration/', views.registration, name='registration'),***

***path('userdashboard/', views.userdashboard, name='userdashboard'),***

***path('logout/', auth\_views.LogoutView.as\_view(next\_page='homepage:home'), name='logout'),***

***path('doctordashboard/', views.doctor\_dashboard, name='doctordashboard'),***

***path('coordinatordashboard/', views.coordinator\_dashboard, name='coordinatordashboard'),***

***path('add\_disease/', add\_disease, name='add\_disease'),***

***]***

## App2 Urls.Py :

***from django.urls import path***

***from . import views***

***app\_name = 'Appointments'***

***urlpatterns = [***

***path('appointments\_form/', views.appointments\_form, name='appointments\_form'),***

***path('manage\_appointments/', views.manage\_appointments, name='manage\_appointments'),***

***path('appointment/<int:appointment\_id>/cancel/', views.cancel\_appointment, name='cancel\_appointment'),***

***path('appointment/<int:appointment\_id>/', views.appointment\_detail, name='appointment\_detail'),  # New route for appointment details***

***]***

## App3 Urls.Py :

***from django.contrib import admin***

***from django.urls import path***

***from . import views***

***app\_name = 'HealthHive'***

***urlpatterns = [***

***path('search/', views.search\_results, name='search\_results'),***

***path('<int:disease\_id>/', views.disease\_detail, name='disease\_detail'),***

***]***

In a Django project, app URLs play a crucial role in routing incoming web requests to the appropriate views within a specific application. Here's a breakdown of how app URLs work:

## App-Level urls.py:

* Each Django app can have its own urls.py file. This allows you to define URL patterns specific to that app.
* This promotes modularity and keeps your URL configurations organized.

## Including App URLs:

* To include an app's URLs in your project's URL configuration, you use the include() function.
* This allows you to delegate URL routing to the app's urls.py file.

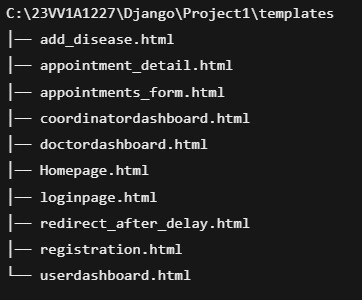
# Templates:

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.

## Folder Tree of the templates:



## Benefits:

* **Separation of Concerns:**
  + Keeps presentation logic separate from business logic.
* **Maintainability:**
  + Makes code easier to maintain and update.
* **Reusability:**
  + Templates can be reused across multiple views.
* **Flexibility:**
  + Allows for easy customization of the user interface

## Home\_page.html :

***1 {% load static %}***

***2***

***3 <!DOCTYPE html>***

***4 <html>***

***5 <head>***

***6 <title> Campus Health and Wellness Project</title>***

***7 <link rel="stylesheet" href="{% static 'style.css' %}">***

***8 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">***

***9***

***10 {% if messages %}***

***11 <script>***

***12 window.onload = function() {***

***13 {% for message in messages %}***

***14 alert("{{ message }}");***

***15 {% endfor %}***

***16 };***

***17 </script>***

***18 {% endif %}***

***19***

***20 <style>***

***21 /\* Hero Image \*/***

***22 .Rel-img {***

***23 position: relative;***

***24 margin-bottom: 90px;***

***max-width: 100%;***

***26 text-align: center;***

***27 }***

***28***

***29 .Rel-img img {***

***30 width: 100%;***

***31 height: auto;***

***32 display: block;***

***33 border-radius: 12px;***

***34 }***

***35***

***36 .overlaytext {***

***37 position: absolute;***

***38 top: 50%;***

***39 left: 50%;***

***40 transform: translate(2%, -50%);***

***41 background: rgba(0, 0, 0, 0.6);***

***42 color: white;***

***43 padding: 10px 20px;***

***44 font-size: 24px;***

***45 border-radius: 8px;***

***46 }***

***47***

***48 /\* Bento Grid Layout \*/***

***49 .bento-grid {***

***50 display: grid;***

***51 grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));***

***52 gap: 20px;***

***53 padding: 20px;***

***54 }***

***55***

***56 .bento-item {***

***57 position: relative;***

***58 height: 200px;***

***59 border-radius: 12px;***

***60 box-shadow: 2px 2px 10px rgba(0, 0, 0, 0.2);***

***61 transition: transform 0.3s ease-in-out;***

***62 overflow: hidden;***

***63 background-size: cover;***

***64 background-position: center;***

***65 }***

***66***

***67 .bento-item:hover {***

***68 transform: scale(1.05);***

***69 }***

***70***

***71 .overlay {***

***72 position: absolute;***

***73 bottom: 0;***

***74 background: rgba(0, 0, 0, 0.6);***

***75 color: white;***

***76 width: 100%;***

***77 padding: 10px;***

***78 text-align: center;***

***79 }***

***80***

***81 .btn {***

***82 display: inline-block;***

***83 padding: 5px 10px;***

***84 background: #0078D7;***

***85 color: white;***

***86 text-decoration: none;***

***87 border-radius: 5px;***

***88 font-size: 14px;***

***89 }***

***90***

***91 .btn:hover {***

***92 background: #005aa7;***

***93 }***

***94***

***95 /\* Search Bar \*/***

***96 .search-container {***

***97 display: flex;***

***98 justify-content: center;***

***99 margin-top: 20px;***

***100 }***

***101***

***102 .search-container input {***

***103 width: 60%;***

***104 padding: 10px;***

***105 font-size: 16px;***

***106 border: 1px solid #ccc;***

***107 border-radius: 5px;***

***108 }***

***109 </style>***

***110 </head>***

***111 <body>***

***112***

***113 <!-- Navbar -->***

***114 <section class="navbar">***

***115 <img src="{% static 'images/jntugv.png' %}" alt="jntugv img">***

***116 <nav>***

***117 <ul>***

***118 <li><a href="{% url 'Appointments:appointments\_form' %}">Request Appointment</a></li>***

***119 {% if user.is\_authenticated %}***

***120 <li>***

***121 <h4><a href="{% url 'accounts:userdashboard' %}">***

***122 Welcome, {{ user.username }}&nbsp;&nbsp;<i class="fa-solid fa-user"></i>***

***123 </a></h4>***

***124 </li>***

***125 {% else %}***

***126 <li><a href="{% url 'accounts:login' %}">Login&nbsp;&nbsp;<i class="fa-solid fa-user"></i></a></li>***

***127 {% endif %}***

***128 <li><a href="services.html"><i class="fa-solid fa-magnifying-glass"></i></a></li>***

***129 </ul>***

***130 </nav>***

***131 </section>***

***132***

***133 <!-- Hero Section -->***

***134 <section class="Rel-img">***

***135 <img src="{% static 'images/bg.jpg' %}" alt="Campus Image">***

***136 <div class="overlaytext">Campus Health is Top Priority</div>***

***137 </section>***

***138***

***139 <!-- Shortcut Section -->***

***140 <div class="shortcut">***

***141 <div class="firsthalf">***

***142 <h2>Healing Starts Here</h2>***

***143 <h3>We are here to help you</h3>***

***144 </div>***

***145 <ul>***

***146 <hr>***

***147 <li><a href="{% url 'Appointments:appointments\_form' %}">Request an Appointment<i class="fas fa-arrow-right"></i></a></li>***

***148 <hr>***

***149 <li><a href="Guide.html">Patient & Visitor Guide<i class="fas fa-arrow-right"></i></a></li>***

***150 <hr>***

***151 <li><a href="Condition&Diseases.html">Diseases and Conditions<i class="fas fa-arrow-right"></i></a></li>***

***152 <hr>***

***153 <li><a href="contactDoctor.html">Find a Doctor<i class="fas fa-arrow-right"></i></a></li>***

***154 <hr>***

***155 </ul>***

***156 </div>***

***157***

***158 <!-- Featured Diseases -->***

***159 <section class="bento-grid">***

***160 {% if featured\_diseases %}***

***161 {% for disease in featured\_diseases %}***

***162 <div class="bento-item" style="background-image: url('{{ disease.image.url }}');">***

***163 <div class="overlay">***

***164 <h3>{{ disease.name }}</h3>***

***165 <p>{{ disease.description|truncatewords:10 }}</p>***

***166 <a href="{% url 'HealthHive:disease\_detail' disease.id %}" class="btn">View More</a>***

***167 </div>***

***168 </div>***

***169 {% endfor %}***

***170 {% else %}***

***171 <p>No diseases found.</p>***

***172 {% endif %}***

***173 </section>***

***174***

***175 <!-- Search Bar (Moved outside Bento Grid) -->***

***176 <div class="search-container">***

***177 <form action="{% url 'HealthHive:search\_results' %}" method="GET">***

***178 <input type="text" name="search" placeholder="Search for diseases...">***

***179 </form>***

***180 </div>***

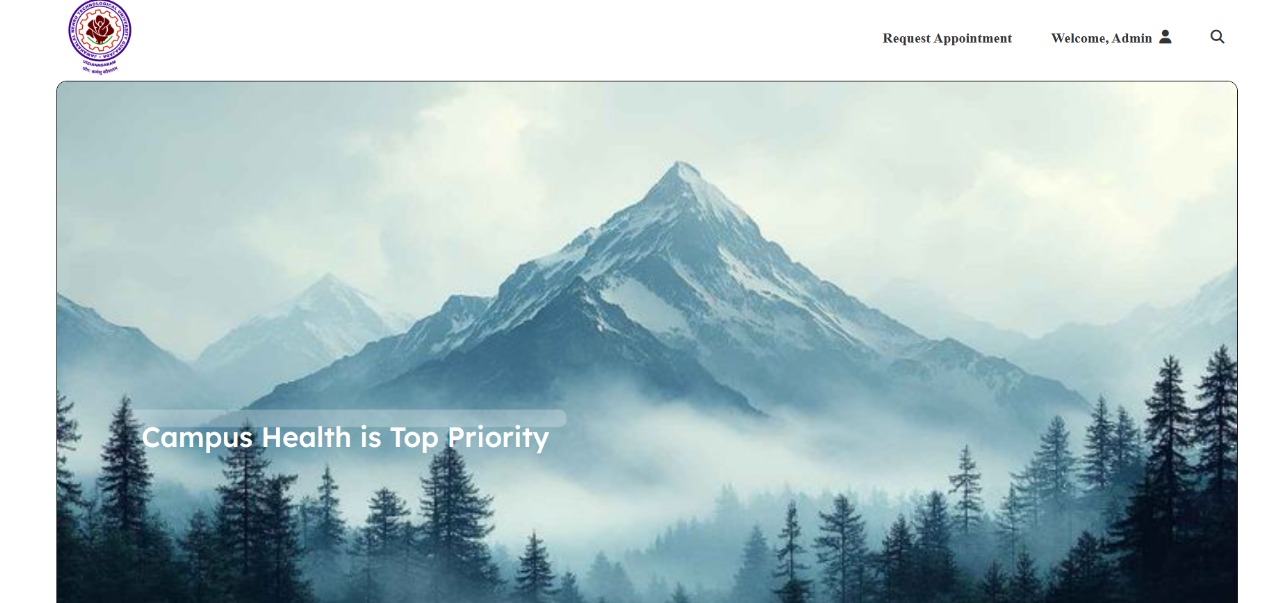
***181***

***182 </body>***

***183 </html>***

***184***

**OUTPUT:**



## Login\_page.html:

***<!DOCTYPE html>***

***<html lang="en">***

***<head>***

***<meta charset="UTF-8">***

***<meta name="viewport" content="width=device-width, initial-scale=1.0">***

***<title>Login Page</title>***

***<style>***

***body {***

***text-decoration: none;***

***font-family: Arial, sans-serif;***

***display: flex;***

***justify-content: center;***

***align-items: center;***

***height: 100vh;***

***background-color: #f4f4f4;***

***}***

***.login-container {***

***background: white;***

***padding: 20px;***

***box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);***

***border-radius: 5px;***

***width: 400px;***

***text-align: center;***

***}***

***input {***

***width: 90%;***

***padding: 10px;***

***margin: 10px 0;***

***border: 1px solid #ccc;***

***border-radius: 5px;***

***}***

***#button {***

***width: 96%;***

***padding: 10px;***

***background-color: #007bff;***

***color: white;***

***border: none;***

***border-radius: 5px;***

***cursor: pointer;***

***}***

***#button:hover {***

***background-color: #0056b3;***

***}***

***.registration-link {***

***margin-top: 20px;***

***font-size: small;***

***}***

***.registration-link a{***

***color: #007bff;***

***text-decoration: none;***

***}***

***.registration-link a :hover{***

***color: #000000;***

***}***

***</style>***

***</head>***

***<body>***

***<div class="login-container">***

***<h2>Login</h2>***

***<form action="{% url 'accounts:login' %}" method="post">***

***{% csrf\_token %}***

***<input type="text" name="username" id="username" placeholder="Username" required>***

***<input type="password" name="password" id="password" placeholder="Password" required>***

***<input type="submit" id="button">***

***</form>***

***<form>***

***<div class="registration-link">***

***<a href="{% url 'accounts:registration' %}">Don't have an account? Register here</a>***

***</div>***

***<div error>***

***{% for message in messages %}***

***<p style="color: red;">{{ message }}</p>***

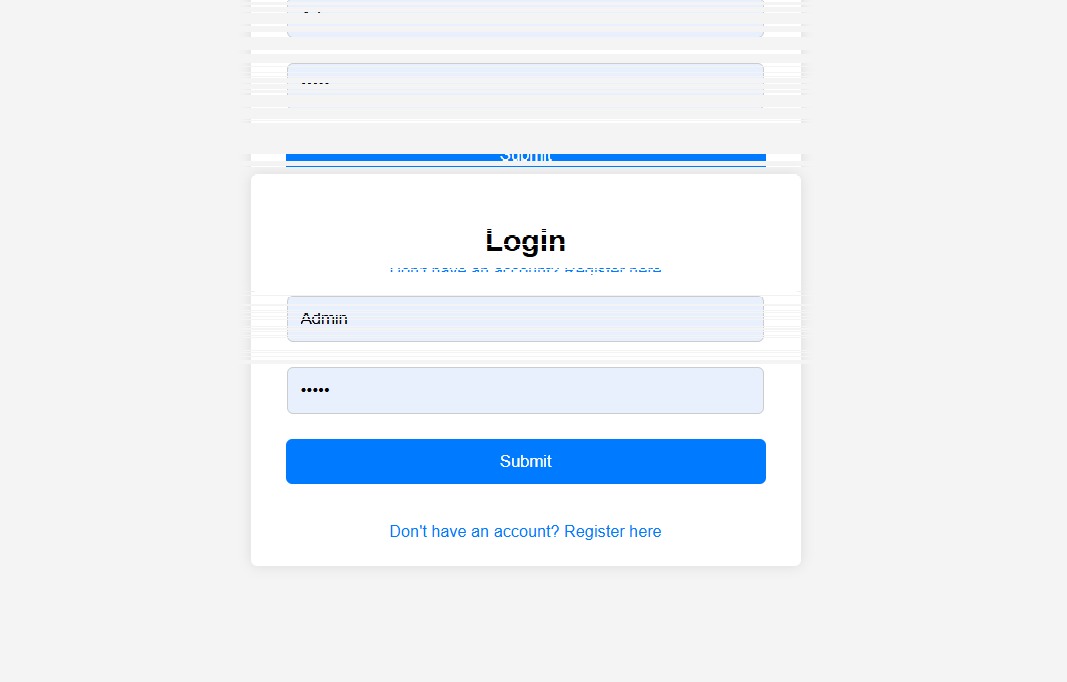
***{% endfor %}***

***</div>***

***</div>***

***</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.
* **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.

## Registration\_page.html:

***<!DOCTYPE html>***

***<html lang="en">***

***<head>***

***<meta charset="UTF-8">***

***<meta name="viewport" content="width=device-width, initial-scale=1.0">***

***<title>Registration Page</title>***

***<style>***

***body {***

***font-family: Arial, sans-serif;***

***background-color: #f4f4f4;***

***display: flex;***

***justify-content: center;***

***align-items: center;***

***height: 100vh;***

***margin: 0;***

***}***

***.registration-form {***

***background-color: #fff;***

***padding: 20px;***

***border-radius: 8px;***

***box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);***

***width: 350px;***

***}***

***.registration-form h2 {***

***margin-bottom: 20px;***

***font-size: 24px;***

***color: #333;***

***}***

***.registration-form input[type="text"],***

***.registration-form input[type="email"],***

***.registration-form input[type="password"] {***

***width: 90%;***

***padding: 10px;***

***margin: 10px 0;***

***border: 1px solid #ccc;***

***border-radius: 4px;***

***}***

***.registration-form input[type="submit"] {***

***width: 100%;***

***padding: 10px;***

***background-color: #28a745;***

***border: none;***

***border-radius: 4px;***

***color: #fff;***

***font-size: 16px;***

***cursor: pointer;***

***}***

***.registration-form input[type="submit"]:hover {***

***background-color: #218838;***

***}***

***.registration-form select {***

***width: 96%;***

***padding: 10px;***

***margin: 10px 0;***

***border: 1px solid #ccc;***

***border-radius: 4px;***

***}***

***</style>***

***</head>***

***<body>***

***<div class="registration-form">***

***<h2>Register</h2>***

***<form action="" method="post">***

***{% csrf\_token %}***

***<input type="text" name="username" placeholder="User Name" required>***

***<input type="text" name="firstname" placeholder="First Name" required>***

***<input type="text" name="lastname" placeholder="Last Name" required>***

***<input type="email" name="email" placeholder="Email" required>***

***<input type="password" name="password" placeholder="Password" required>***

***<label for="role" >Select Role:</label>***

***<select name="role" required>***

***<option value="">Select Role</option>***

***<option value="Citizen">Citizen</option>***

***<option value="Student">Student</option>***

***<option value="Doctor">Doctor</option>***

***<option value="Coordinator">Coordinator</option>***

***</select>***

***<input type="submit" value="Register">***

***</form>***

***<div class="error">***

***{% for message in messages %}***

***<p style="color:red">{{message}}</p>***

***{% endfor %}***

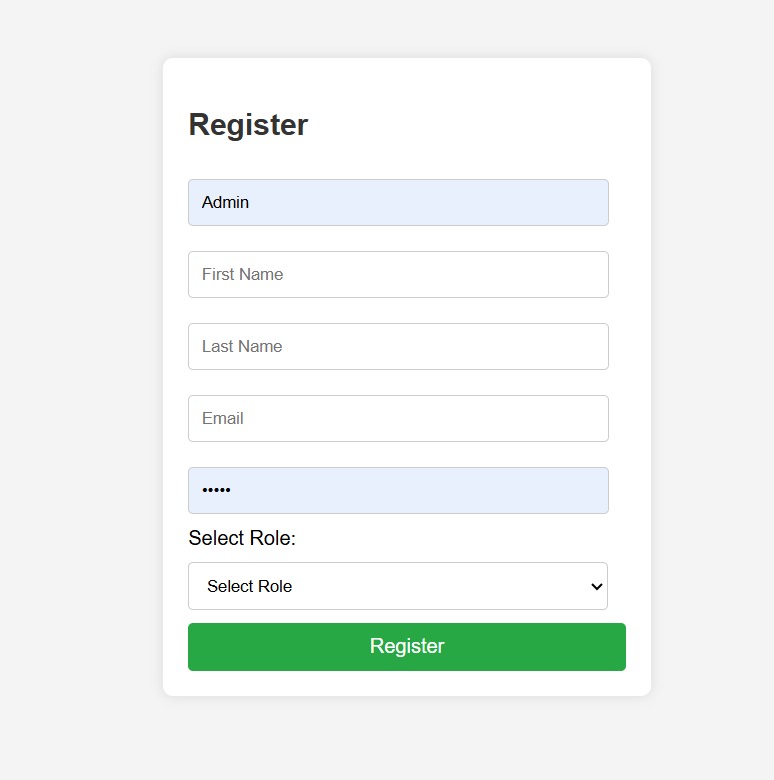
***</div>***

***</div>***

***</body>***

***</html>***

**OUTPUT:**



## User\_dashboard.html:

***{% load static %}***

***<!DOCTYPE html>***

***<html lang="en">***

***<head>***

***<meta charset="UTF-8">***

***<meta name="viewport" content="width=device-width, initial-scale=1.0">***

***<title>Dashboard</title>***

***<style>***

***body {***

***font-family: Arial, sans-serif;***

***margin: 0;***

***display: flex;***

***}***

***.sidebar {***

***width: 250px;***

***background: #f4f4f4;***

***padding: 20px;***

***height: 100vh;***

***}***

***.main-content {***

***flex-grow: 1;***

***padding: 20px;***

***}***

***.header {***

***display: flex;***

***justify-content: space-between;***

***}***

***.box {***

***border: 1px solid #ccc;***

***padding: 15px;***

***margin-top: 10px;***

***}***

***.logout {***

***color: red;***

***text-decoration: none;***

***}***

***</style>***

***</head>***

***<body>***

***<!-- Sidebar -->***

***<div class="sidebar">***

***<img src="{% static 'images/profile.jpeg' %}" alt="Profile Pic" width="80" height="80">***

***<h3>{{ user.username }}</h3>***

***<p>Status: Active</p>***

***<a href="#">Complete Your Profile</a>***

***</div>***

***<!-- Main Content -->***

***<div class="main-content">***

***<div class="header">***

***<p>Date: {{ date }} | Time: {{ time }}</p>***

***<form action="{% url 'accounts:logout' %}" method="post">***

***{% csrf\_token %}***

***<button type="submit" class="logout">Logout</button>***

***</form>***

***</div>***

***<div class="box">***

***<h3>Upcoming Appointments</h3>***

***{% if appointments %}***

***<ul>***

***{% for appointment in appointments %}***

***<li>{{ appointment.date }} - {{ appointment.description }} ({{ appointment.status }})</li>***

***{% endfor %}***

***</ul>***

***{% else %}***

***<p>No upcoming appointments.</p>***

***{% endif %}***

***</div>***

***<div class="box">***

***<h3>Health Records</h3>***

***<p>No records available.</p>***

***</div>***

***<div class="box">***

***<h3>Feedback</h3>***

***<textarea rows="4" cols="50" placeholder="Write your feedback here..."></textarea>***

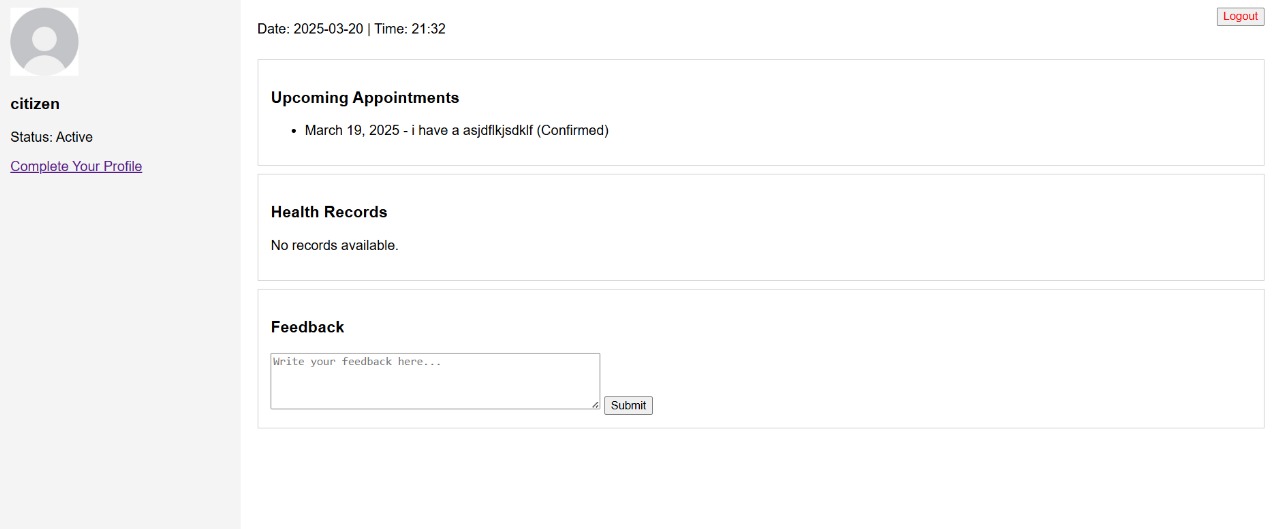
***<button>Submit</button>***

***</div>***

***</div>***

***</body>***

***</html>***

**OUTPUT:**

Key Considerations:

Security:

* Protect user data by implementing proper authentication and authorization.
* Sanitize user input to prevent security vulnerabilities.

Performance:

* Optimize database queries and template rendering to ensure a fast and responsive dashboard.
* Use caching where possible.

User Experience (UX):

* Design a dashboard that is easy to use and provides a positive user experience.
* Gather user feedback and iterate on your design.

## Doctor\_dashboard.html

***{% load static %}***

***<!DOCTYPE html>***

***<html lang="en">***

***<head>***

***<meta charset="UTF-8">***

***<meta name="viewport" content="width=device-width, initial-scale=1.0">***

***<title>Doctor Dashboard</title>***

***<style>***

***body {***

***font-family: Arial, sans-serif;***

***margin: 0;***

***display: flex;***

***}***

***.sidebar {***

***width: 250px;***

***background: #f4f4f4;***

***padding: 20px;***

***height: 100vh;***

***}***

***.main-content {***

***flex-grow: 1;***

***padding: 20px;***

***}***

***.header {***

***display: flex;***

***justify-content: space-between;***

***}***

***.box {***

***border: 1px solid #ccc;***

***padding: 15px;***

***margin-top: 10px;***

***}***

***.logout {***

***color: red;***

***text-decoration: none;***

***}***

***</style>***

***</head>***

***<body>***

***<!-- Sidebar -->***

***<div class="sidebar">***

***<img src="{% static 'images/profile.jpeg' %}" alt="Profile Pic" width="80" height="80">***

***<h3>{{ user.username }}</h3>***

***<p>Role: Doctor</p>***

***<a href="#">Manage Profile</a>***

***</div>***

***<!-- Main Content -->***

***<div class="main-content">***

***<div class="header">***

***<p>Date: {{ date }} | Time: {{ time }}</p>***

***<form action="{% url 'accounts:logout' %}" method="post">***

***{% csrf\_token %}***

***<button type="submit" class="logout">Logout</button>***

***</form>***

***</div>***

***<div class="box">***

***<h3>Appointments to Review</h3>***

***<ul>***

***{% for appointment in appointments %}***

***<li>***

***<a href="{% url 'Appointments:appointment\_detail' appointment.id %}">***

***{{ appointment.date }} - {{ appointment.patient\_name }} ({{ appointment.status }})***

***</a>***

***</li>***

***{% empty %}***

***<li> No appointments to review.</li>***

***{% endfor %}***

***</ul>***

***</div>***

***<div class="box">***

***<h3>Patient Reports</h3>***

***<p>No reports available.</p>***

***</div>***

***<div class="box">***

***<h3>Messages</h3>***

***<textarea rows="4" cols="50" placeholder="Send a message to a patient..."></textarea>***

***<button>Send</button>***

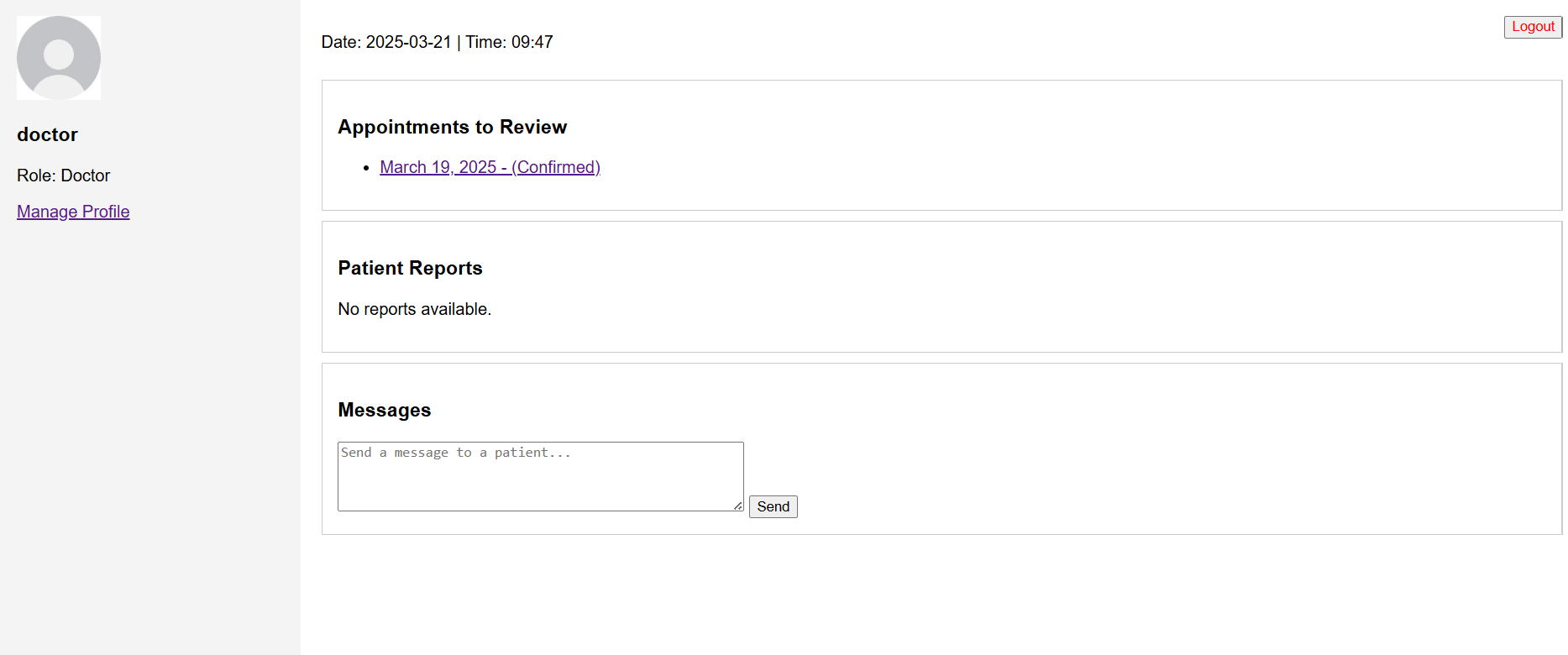
***</div>***

***</div>***

***</body>***

***</html>***

**OUTPUT:**

****

A doctor's dashboard within a Django application would be a specialized user dashboard tailored to the needs of medical professionals. Here's a breakdown of key features and considerations for building such a dashboard:

**Key Features to Emphasize:**

* Efficiency: Doctors are busy, so the dashboard should be designed for speed and ease of use.
* Accuracy: Medical information must be accurate and reliable.
* Security: Patient privacy is paramount.

## Coordinator\_dashboard.html:

***{% load static %}***

***<!DOCTYPE html>***

***<html lang="en">***

***<head>***

***<meta charset="UTF-8">***

***<meta name="viewport" content="width=device-width, initial-scale=1.0">***

***<title>Coordinator Dashboard</title>***

***<style>***

***body {***

***font-family: Arial, sans-serif;***

***margin: 0;***

***display: flex;***

***}***

***.sidebar {***

***width: 250px;***

***background: #f4f4f4;***

***padding: 20px;***

***height: 100vh;***

***}***

***.main-content {***

***flex-grow: 1;***

***padding: 20px;***

***}***

***.header {***

***display: flex;***

***justify-content: space-between;***

***}***

***.box {***

***border: 1px solid #ccc;***

***padding: 15px;***

***margin-top: 10px;***

***}***

***.logout {***

***color: red;***

***text-decoration: none;***

***}***

***</style>***

***</head>***

***<body>***

***<!-- Sidebar -->***

***<div class="sidebar">***

***<img src="{% static 'images/profile.jpeg' %}" alt="Profile Pic" width="80" height="80">***

***<h3>{{ user.username }}</h3>***

***<p>Role: Coordinator</p>***

***<a href="#">Manage Schedule</a>***

***</div>***

***<!-- Main Content -->***

***<div class="main-content">***

***<div class="header">***

***<p>Date: {{ date }} | Time: {{ time }}</p>***

***<form action="{% url 'accounts:logout' %}" method="post">***

***{% csrf\_token %}***

***<button type="submit" class="logout">Logout</button>***

***</form>***

***</div>***

***<div class="box">***

***<h3>Manage Doctor Schedules</h3>***

***<ul>***

***{% for doctor in doctors %}***

***<li>{{ doctor.name }} - Available: {{ doctor.availability }}</li>***

***{% endfor %}***

***</ul>***

***</div>***

***<div class="box">***

***<h3>Appointment Requests</h3>***

***<p>Pending requests: {{ pending\_requests }}</p>***

***</div>***

***<div class="box">***

***<h3>Notifications</h3>***

***<p>No new notifications.</p>***

***</div>***

***</div><div class="box">***

***<h3>Diseases To Showcase</h3>***

***<a href="{% url 'accounts:add\_disease' %}"> Add New Disease</a>***

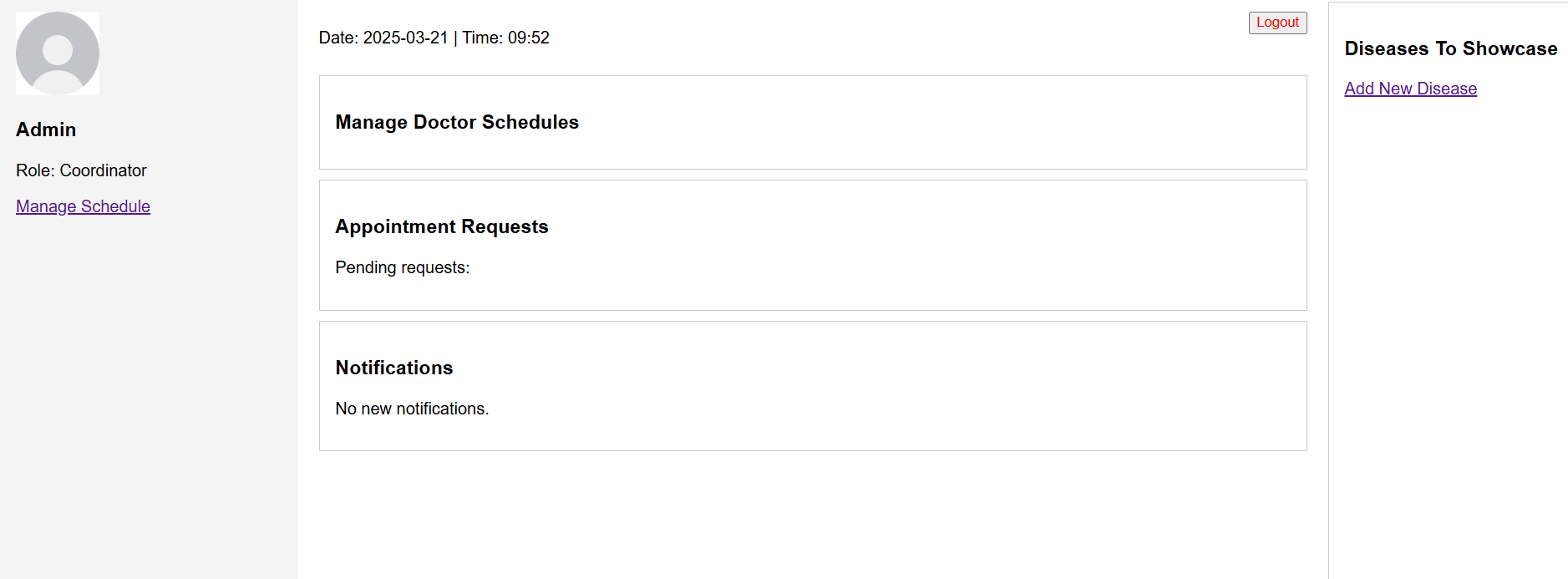
***</div>***

***</div>***

***</body>***

***</html>***

**OUTPUT:**

****

A coordinator dashboard, in a general sense, is designed to provide an overview and management tools for someone responsible for coordinating activities, resources, or people. This could apply to various contexts, such as event coordination, project management, logistics, or even educational programs. Here's a breakdown of key features and considerations for building a coordinator dashboard in Django:

Key Features to Emphasize:

* Organization: The dashboard should help coordinators stay organized and manage their tasks effectively.
* Efficiency: The dashboard should streamline coordination processes and save time.
* Communication: The dashboard should facilitate clear and efficient communication.
* Visibility: The dashboard should provide a clear overview of progress and key metrics.

By focusing on these key features, you can build a coordinator dashboard that empowers coordinators to manage their responsibilities effectively.

## 

## Appointment\_form.html:

***<!DOCTYPE html>***

***<html lang="en">***

***<head>***

***<meta charset="UTF-8">***

***<meta name="viewport" content="width=device-width, initial-scale=1.0">***

***<title>Doctor Appointment Form</title>***

***<style>***

***body {***

***font-family: Arial, sans-serif***

***background-color: #f4f4f9;***

***margin: 0;***

***padding: 0;***

***display: flex;***

***justify-content: center;***

***align-items: center;***

***height: 100vh;***

***}***

***.form-container {***

***background: #fff;***

***padding: 20px 30px;***

***border-radius: 8px;***

***box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);***

***width: 100%;***

***max-width: 400px;***

***}***

***.form-container h2 {***

***margin-bottom: 20px;***

***color: #333;***

***text-align: center;***

***}***

***.form-group {***

***margin-bottom: 15px;***

***}***

***.form-group label {***

***display: block;***

***margin-bottom: 5px;***

***color: #555;***

***}***

***.form-group input, .form-group textarea {***

***width: 100%;***

***padding: 10px;***

***border: 1px solid #ccc;***

***border-radius: 4px;***

***font-size: 14px;***

***}***

***.form-group textarea {***

***resize: none;***

***height: 80px;***

***}***

***.form-group button {***

***width: 100%;***

***padding: 10px;***

***background-color: #007bff;***

***color: #fff;***

***border: none;***

***border-radius: 4px;***

***font-size: 16px;***

***cursor: pointer;***

***}***

***.form-group button:hover {***

***background-color: #0056b3;***

***}***

***</style>***

***</head>***

***<body>***

***<div class="form-container">***

***<h2>Doctor Appointment Form</h2>***

***<form method="post">***

***{% csrf\_token %}***

***<div class="form-group">***

***<label for="name">Full Name</label>***

***<input type="text" id="name" name="name" placeholder="Enter your full name" required>***

***</div>***

***<div class="form-group">***

***<label for="email">Email Address</label>***

***<input type="email" id="email" name="email" placeholder="Enter your email" required>***

***</div>***

***<div class="form-group">***

***<label for="phone">Phone Number</label>***

***<input type="tel" id="phone" name="phone" placeholder="Enter your phone number" required>***

***</div>***

***<div class="form-group">***

***<label for="date">Appointment Date</label>***

***<input type="date" id="date" name="date" required>***

***</div>***

***<div class="form-group">***

***<label for="message">Additional Notes</label>***

***<textarea id="message" name="message" placeholder="Enter any additional notes"></textarea>***

***</div>***

***<div class="form-group">***

***<button type="submit">Book Appointment</button>***

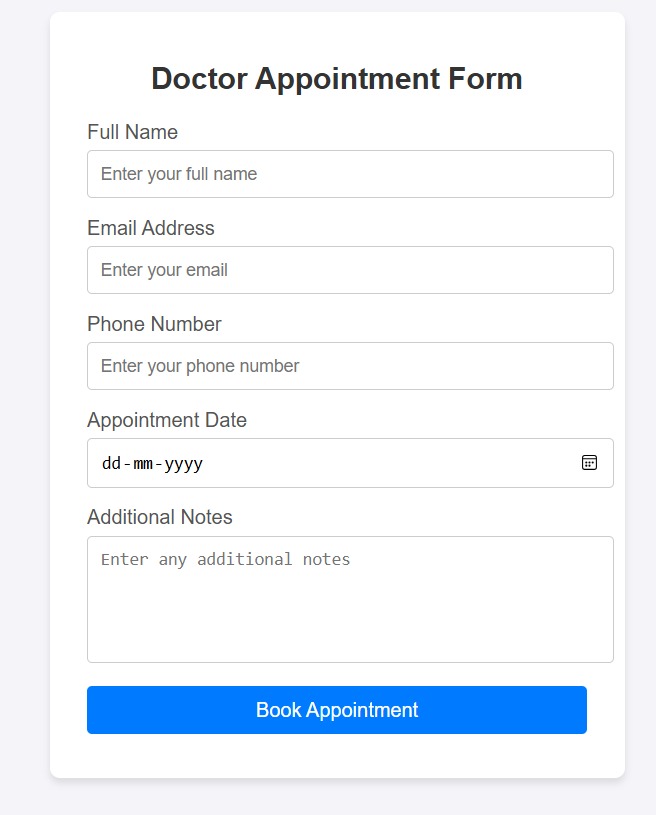
***</div>***

***</form>***

***</div>***

***</body>***

***</html>***

**OUTPUT:**

## Appointment\_details.html

***{% load static %}***

***<!DOCTYPE html>***

***<html lang="en">***

***<head>***

***<meta charset="UTF-8">***

***<meta name="viewport" content="width=device-width, initial-scale=1.0">***

***<title>Appointment Details</title>***

***<style>***

***body {***

***font-family: Arial, sans-serif;***

***margin: 20px;***

***}***

***.container {***

***width: 50%;***

***margin: auto;***

***}***

***.box {***

***border: 1px solid #ccc;***

***padding: 15px;***

***margin-top: 10px;***

***}***

***.back-link {***

***display: inline-block;***

***margin-top: 10px;***

***text-decoration: none;***

***color: blue;***

***}***

***</style>***

***</head>***

***<body>***

***<div class="container">***

***<h2>Appointment Details</h2>***

***<div class="box">***

***<p><strong>Patient:</strong> {{ appointment.user.username }}</p>***

***<p><strong>Date:</strong> {{ appointment.date }}</p>***

***<p><strong>Description:</strong> {{ appointment.description }}</p>***

***<p><strong>Status:</strong> {{ appointment.status }}</p>***

***<!-- Form to update the status -->***

***<form method="POST">***

***{% csrf\_token %}***

***<label for="status">Change Status:</label>***

***<select name="status">***

***<option value="Pending" {% if appointment.status == "Pending" %}selected{% endif %}>Pending</option>***

***<option value="Confirmed" {% if appointment.status == "Confirmed" %}selected{% endif %}>Confirmed</option>***

***<option value="Cancelled" {% if appointment.status == "Cancelled" %}selected{% endif %}>Cancelled</option>***

***</select>***

***<button type="submit">Update</button>***

***</form>***

***</div>***

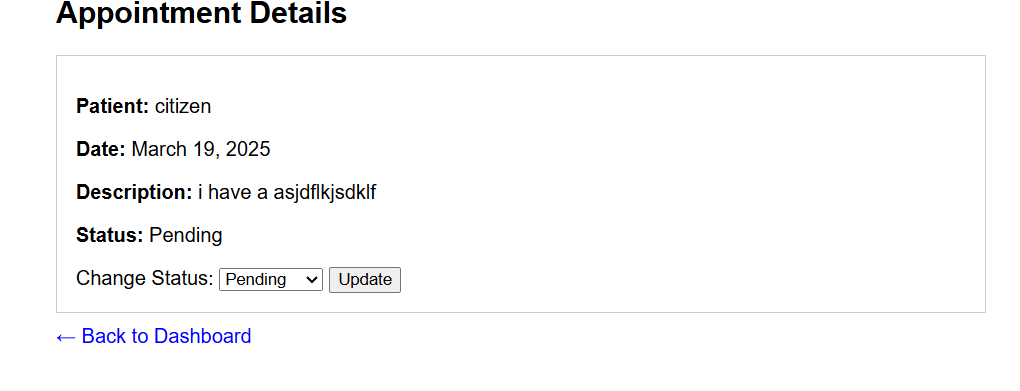
***<a href="{% url 'accounts:userdashboard' %}" class="back-link">← Back to Dashboard</a>***

***</div>***

***</body>***

***</html>***

**OUTPUT:**

****

Connection Database with the Project:

To connect MySQL with a Django project, you essentially configure Django to use MySQL as its database backend instead of the default SQLite. Here's a breakdown of the steps involved:

## Install MySQL and Required Packages

First, make sure MySQL server is installed and running.

Then, install the MySQL client library that Django uses:

pip install mysqlclient

## Configure Django Settings

Open your Django project’s settings.py, and change the DATABASES section to use MySQL:

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'chwp\_db',

        'USER': 'root',

        'PASSWORD': '4D4C50\_2005',

        'HOST': 'localhost',

        'PORT': '3306',

    }

}

# What is models.py in Django:

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:

Django models offer a structured and efficient way to handle database interactions without manually writing SQL queries. They provide several advantages that make web development faster, safer, and more maintainable.

## How to Apply Models:

Create the Model in models.py:

* Write your models inside the models.py file.

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..why?:

Registering models in admin.py is an essential step to leverage Django’s built-in admin panel for easy database management. It eliminates the need for writing manual queries and provides a secure, user-friendly interface for handling data.

## Accounts.py:

***from django.db import models***

***from django.contrib.auth.models import User***

***# Create your models here.***

***class UserProfile(models.Model):***

***ROLE\_CHOICES = (***

***('Citizen', 'Citizen'),***

***('Student', 'Student'),***

***('Doctor', 'Doctor'),***

***('Coordinator', 'Coordinator'),***

***)***

***user = models.OneToOneField(User, on\_delete=models.CASCADE)***

***role = models.CharField(max\_length=20, choices=ROLE\_CHOICES, default='Citizen')***

***def \_str\_(self):***

***return f"{self.user.username} - {self.role}"***

## Appointment.py

***from django.db import models***

***# Create your models here.***

***from django.db import models***

***from django.contrib.auth.models import User***

***import random***

***class Appointment(models.Model):***

***user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name="patient\_appointments") # Patient***

***doctor = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True, blank=True, related\_name="doctor\_appointments") # Assigned Doctor***

***date = models.DateField()***

***description = models.CharField(max\_length=255)***

***status = models.CharField(***

***max\_length=20,***

***choices=[("Confirmed", "Confirmed"), ("Pending", "Pending"), ("Cancelled", "Cancelled")],***

***default="Pending"***

***)***

***def assign\_random\_doctor(self):***

***"""Assigns a random doctor from the available doctors."""***

***doctors = User.objects.filter(userprofile\_\_role="Doctor")***

***if doctors.exists():***

***self.doctor = random.choice(doctors)***

***self.save()***

***def \_str\_(self):***

***return f"{self.user.username} - {self.description} on {self.date} (Doctor: {self.doctor.username if self.doctor else 'Not Assigned'})"***

# What is 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
* EXAMPLE:

***# models.py***

***from django.db import models***

***class Item(models.Model):***

***name = models.CharField(max\_length=50)***

***# forms.py***

***from django import forms***

***from .models import Item***

***class ItemForm(forms.ModelForm):***

***class Meta:***

***model = Item***

***fields = ['name']***

***# views.py***

***from django.shortcuts import render, redirect***

***from .forms import ItemForm***

***def add(request):***

***f = ItemForm(request.POST) if request.method == 'POST' else ItemForm()***

***if f.is\_valid():***

***f.save()***

***return redirect('ok')***

***return render(request, 'i.html', {'f': f})***

***def ok(request):***

***return render(request, 'ok.html')***

***# urls.py***

***from django.urls import path***

***from . import views***

***urlpatterns = [***

***path('', views.add, name='add'),***

***path('ok/', views.ok, name='ok'),***

***]***

***# templates/i.html***

***<form method="post">{% csrf\_token %}{{ f.as\_p }}<button>Add</button></form>***

***# templates/ok.html***

***<h1>Done!</h1>***

This structure keeps the project organized, scalable, and easy to manage.

## Migrations : Synchronizing Models with the Database

Now that the database is set up, apply migrations to create the necessary tables that are created in the models:

python manage.py makemigrations

## View your tables in the mysql command line client

Use the below code to view

Enter password: \*\*\*\*\*\*\*\*\*\*\*

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 329

Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

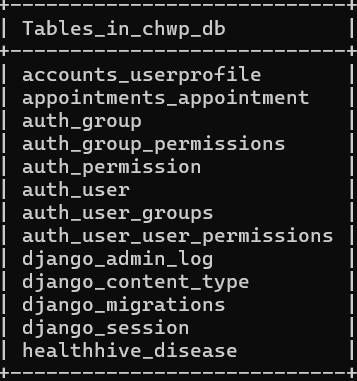
mysql> use yourdatabase

Database changed

mysql> show tables

-> ;

**Output:**

****

## 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:

1. Scalability – Handle more users without performance issues.
2. Security – Protect user data with SSL and secure databases. Global Accessibility – Users can access your app from anywhere.
3. 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*

1. Go to [GitHub](https://github.com/) and click Sign up.
2. Enter your Username, Email, and Password.
3. Complete the verification and click Create Account.
4. Verify your email by clicking the link in your inbox.

*Step 2: Push to GitHub*

Initialize Git in your project:

git init

1. Connect to GitHub:

git remote add origin <https://github.com/Lokesh-Madiri/Campus-Health-and-Wellness-Project>

Add and commit changes: git add *.*

git commit -m "Initial Commit "

1. 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.

CERTIFICATE FOR FRONTEND DEVELOPMENT:

