**Name:** Parsley Kabuthi Njoroge

**Course:** Computer Science

**Reg No:** Sct 211-0007/2023

**Unit:** Internet Application Programming

**Unit code:** ICS 2203

**Project: Hospital Web**

**Project Overview**

The Hospital Web Application is a complete project built with Django that offers full information about dental services and makes managing user accounts easier. This program offers a dynamic interface to showcase doctors and the range of services they offer, as well as a platform for users to register, log in, and manage their appointments. By making it easier to obtain dental care information and make appointments, the project seeks to improve the user experience.

**Distinctiveness and Complexity**

**Uniqueness of the Project**

*Specialization*: Unlike more broad healthcare applications, this project is only focused on dental services. It offers features that are specifically designed to manage dental treatment, meeting the needs of dental clinics and making it a useful tool for both patients and healthcare professionals.   
*User-Centric Design*: The user experience is a major focus of the application's design. It provides simple navigation and quick access to key functions including appointment scheduling, login, and registration. Users can interact with the application with ease thanks to the user interface's simple and aesthetically pleasing design.  
*The integration of Features*: The program incorporates a number of features, such as email alerts, appointment scheduling, and user authentication. Users may easily manage their appointments and accounts in one location thanks to this integration.

**Complexities Involved**

*User Authentication*: The project becomes much more complex when secure user authentication and session management are implemented. The program makes use of Django's integrated authentication mechanism, which necessitates cautious user data management, password hashing, and security precautions to safeguard private data.   
*Dynamic Content Management*: In order to properly manage data, a strong backend structure is required because the application dynamically displays information about physicians and services. This entails developing templates, views, and models that cooperate to show users information in real time.   
*Email Notifications*: An additional level of complexity is introduced by integrating email functions for delivering confirmations and notifications. Error handling must be put in place to handle any problems that may occur during the email sending process, and proper configuration is necessary to guarantee that emails are sent consistently.   
*Database Management*: Since PostgreSQL is the project's database backend, it is necessary to set up the database structure, oversee migrations, and guarantee data integrity. This makes database administration and design more difficult.

**Design Approach**

*Model-View-Template (MVT):* A fundamental Django design principle, the project adheres to the MVT design pattern. Using this method, the application is divided into three interrelated parts:   
*Models*: Specify the data structure, such the Doctor model, which has fields for job, description, name, and image. Data handling and management are made simple by this division.   
*Views*: Manage user requests and business logic. To create or authenticate users, for instance, the views for registration and login handle user input and communicate with the models.   
*Templates*: Control the presentation layer by rendering user interfaces with HTML files. Scalability and maintainability are improved by this division of responsibilities.   
*Design that is responsive*: The responsive design of the frontend guarantees interoperability with a range of devices, including smartphones, tablets, and PCs. The user interface is improved by using CSS frameworks and bespoke styles, which give it a unified appearance and feel.   
*Mechanisms for User Feedback*: The program has features for user feedback, like notifications for successful registrations or unsuccessful login attempts. By giving users instant feedback on their activities, this improves the user experience.

**File Contents**

**Core Files**

\_\_init\_\_.py:   
The directory is identified as a Python package by this file. Although it is usually empty, Python needs it to identify the directory as a package.   
Asgi.py:   
enables the program to manage several requests at once by configuring the ASGI interface for asynchronous support.   
allows ASGI servers to launch the application by exposing the ASGI callable as an application.   
makes sure the application is using the correct configuration by specifying hospital\_web.settings as the default settings module.   
wsgi.py:   
sets up synchronous support for the WSGI interface, which is the industry standard for launching Python web apps.   
enables WSGI servers to execute the application by exposing the WSGI callable as an application.   
Additionally, hospital\_web.settings is specified as the default settings module.

**Settings**

All of the Django project's configuration parameters are contained in settings.py, including:   
*Security Settings*: To improve security, define the SECRET\_KEY, specify DEBUG mode, and list permitted hosts.   
Installed Applications: This list includes programs that are necessary for the project's operation, like accounts and services.   
*Middleware*: Sets up middleware for authentication, session management, and other features.   
Database Configuration: Indicates the connection information and PostgreSQL as the database backend.  
*Static and Media Files*: Specifies the directories for supplying media files (user-uploaded content) and static files (CSS, JavaScript), guaranteeing that the application can handle and display these resources appropriately.   
*Email Setup*: Sets up email parameters, such as SMTP server information, which are essential for user correspondence.

**URL Routing**

urls.py: Oversees the project's URL routing, sending incoming requests to the relevant views in accordance with the specified URL patterns.   
Enables users to access a range of features, including viewing services, registering, and logging in, by forwarding requests to views in the service and accounts apps.   
Includes the Django admin interface, which gives administrators the ability to control the data and users of the application.   
Makes sure user-uploaded material is available by configuring static file serving for media files.

**Applications**

***accounts:***   
Essential functionality for user registration and login are provided by this application, which also handles user authentication and account management.   
*admin.py*: Enables administrators to effortlessly maintain doctor records by registering the Doctor model with the Django admin interface.   
The accounts application's name and any special parameters are defined via the *apps.py* file.   
The Doctor model is defined by *models.py*, which also has fields for the doctor's name, picture, job title, and description. Data pertaining to doctors is stored using this model as the basis.   
Views for user registration, login, and logout are contained in *views.py*. These views render the relevant templates, manage user input, and communicate with the models.   
*urls.py*: Connects user actions to the appropriate views by routing account management-related URLs.   
*tests.py*: A placeholder for unit tests, which can be used to make sure the program works.

***service:***   
The hospital's primary service offerings are managed via this program, which also gives users information about the services they can access.   
*admin.py*: Enables simple administration of service records by registering models associated with services with the admin interface.   
The service application's name and settings are defined by the apps.py file.   
*models.py*: A stand-in for service-related models that can be extended to incorporate other hospital services.   
*Views.py*: Provides views for showcasing doctors and services, enabling users to peruse their options.   
*urls.py*: Connects user queries to the relevant views by routing service-related URLs.   
*tests.py*: A placeholder for unit tests that can be used to make sure the functionalities connected to the service work.

**Templates**

***HTML Files***

The primary landing page that shows a summary of services and physicians is *home.html*. For users, it acts as their initial point of contact.   
*contact.html*: A contact form that improves user engagement by enabling users to send questions or comments.   
A user login form called *login.html* allows registered users to safely access their accounts.   
*register.html*: A user registration form with fields for required information that enables new users to create an account.

**Static Files**

***CSS and JS Files***

Contains stylesheets and JavaScript files that enhance the frontend design and functionality. These files are crucial for creating a visually appealing and interactive user interface.

**How to Run the Application**

To successfully run the Hospital Web Application, follow these steps:

***Prerequisites:***

Ensure that Python and Django are installed on your system.

Install PostgreSQL and create a new database for the project. Make sure you have the necessary credentials (username and password) to access the database.

***Clone the Repository****:*

Use Git to clone the project repository to your local machine. Open your terminal and run:

git clone <repository-url>

cd hospital\_web

***Install Dependencies****:*

Navigate to the project directory and install the required Python packages using pip.

***Configure Database****:*

Open the *settings.py* file and update the **DATABASES** section with your PostgreSQL credentials. Ensure that the database name, user, and password are correctly specified.

***Run Migrations****:*

Apply the database migrations to set up the necessary tables and schema. Run the following command:

python manage.py migrate.

***Create a Superuser****:*

To access the Django admin interface, create a superuser account by running:

python manage.py createsuperuser.

***Run the Development Server****:*

Start the Django development server to run the application locally. Execute the following command:

python manage.py runserver

The application will be accessible at ***http://127.0.0.1:8000/****.*