

ANUDIP FOUNDATION

ITPR Project

Project title

“Hotel Management System”

By

	Name	Enrollment No.	
	Asmita Pandharinath Pande	AF0481957	

Under Guidance Of

Rajshri Thete

ABSTRACT

The Hotel Management System is a web-based application developed using the Django framework, aimed at streamlining the management of hotel operations. This system allows hotel administrators to efficiently manage room bookings, customer details, and overall operations through a secure, scalable, and user-friendly interface. Customers can browse available rooms, make reservations, and manage their bookings online. The system employs Django's built-in authentication features to ensure secure access for both administrators and users. It also features a modular design that separates concerns using Django's Model-View-Template (MVT) architecture. This project demonstrates how modern web technologies can be leveraged to automate manual hotel management tasks, enhance customer experience, and reduce administrative workload. The system is suitable for small to medium-sized hotels and provides a solid foundation for further enhancements such as online payments and mobile integration.

Interactive GUI and the ability to manage various hotel bookings and rooms make this system very flexible and convenient. The hotel manager is a very busy person and does not have the time to sit and manage the entire activities manually on paper. This application gives him the power and flexibility to manage the entire system from a single online system. Hotel Management project provides room booking, staff management and other necessary hotel management features.

The system allows the manager to post available rooms in the system. Customers can view and book room online. Admin has the power of either approving or disapproving the customer's booking request. Other hotel services can also be viewed by the customers and can book them too. The system is hence useful for both customers and managers to portably manage the hotel activities.

ACKNOWLEDGEMENT

The project “**Hotel Management System**” is the Project work carried out by

Name	Enrollment No
Rajshri Thete	AF0481957

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project

Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.



TOPIC OF THE PROJECT	<i>Page No</i>
1. <i>Title of the Project</i>	6
2. <i>Introduction/Objective</i>	7-8
2.1. <i>Introduction</i>	
2.2. <i>Objective</i>	
3. <i>System Analysis</i>	9-29
3.1. <i>Problem Definition</i>	
3.2. <i>Preliminary Investigation</i>	
3.3. <i>Feasibility Study</i>	
3.4. <i>Project Planning</i>	
3.5. <i>Project Scheduling</i>	
3.6. <i>Software Requirement Specification</i>	
3.9 <i>Functional Requirements</i>	
3.10 <i>Software Engineering Paradigm</i>	
3.11 <i>Data model description</i>	
4. <i>System Design</i>	30-45
4.1. <i>Modularization Details</i>	
4.2. <i>Database Design</i>	
4.3. <i>Procedural Design</i>	
4.4. <i>User Interface Design</i>	
4.5. <i>Outputs of the Report</i>	
5. <i>Coding</i>	46-65
5.1. <i>Complete Project Coding</i>	
5.2. <i>Error Handling</i>	

5.3. Validation Check

6. Testing

66-67

6.1. Testing Strategies

68-69

7. System Security

7.1. Security Compliance

7.2. Security Measures in Place

70

8. Reports

71

9. Future Scope of the Project

72

10. Bibliography



HOTEL MANAGEMENT SYSTEM



INTRODUCTION

The Hotel Management System is a web-based application developed using the Django framework, designed to streamline and automate the core operations of hotel management. This system provides a centralized platform for managing reservations, customer check-ins and check-outs, room allocations, billing, and administrative functions. By replacing traditional manual processes with a digital solution, the system enhances efficiency, reduces human error, and improves the overall customer experience.

Built using Python and Django on the backend, with HTML, CSS, and JavaScript for the frontend interface, the application supports both staff and customer modules. Staff can manage room availability, handle guest records, and generate invoices, while customers can browse rooms, make bookings, and receive booking confirmations online.

This project aims to demonstrate the implementation of a real-world hotel management scenario using modern web technologies and follows best practices in database design, user authentication, and web security. The system is scalable, user-friendly, and suitable for small to mid-sized hotel businesses seeking to digitalize their operations.

a hotel reservation site script where site users will be able to search rooms availability with an online booking reservations system. Site users can also browse hotels, view room inventory, check availability, and book reservations in real-time.

OBJECTIVES

The purpose of hotel booking system is to automate the existing manual system by the help of computerized equipment's and full-fledged computer software, fulfilling their requirement, so that their valuable or information can be stored for a longer period with easy accessing and manipulating of the same. The required software and hardware are easily available and easy to work with. This proposes that efficiency of hotel organizations could be improved by integrating service-oriented operations service-oriented operations with project management principles.

The primary objectives of the Hotel Management System project are:

1. **To automate hotel operations:** Replace manual processes with a digital system for managing bookings, check-ins, check-outs, and payments.
2. **To improve efficiency:** Minimize the time and effort required for staff to perform daily hotel operations through a centralized management interface.
3. **To enhance customer experience:** Allow guests to view available rooms, make online reservations, and receive booking confirmations conveniently.
4. **To maintain accurate records:** Store and manage guest data, room information, and transaction history in a secure and structured manner using a relational database.
5. **To generate reports:** Provide features for generating booking reports, occupancy summaries, and revenue insights to assist in business decision-making.



➤ PROBLEM DEFINITION:

Managing hotel operations manually is time-consuming, error-prone, and inefficient, especially as customer demands and operational complexities increase. Traditional hotel management relies heavily on paperwork, human memory, and face-to-face interactions, which can lead to issues such as overbookings, data loss, billing mistakes, and poor customer service.

There is a clear need for a digital solution that can handle the key aspects of hotel management — such as room bookings, guest check-in/checkout, payment processing, and record keeping — in an organized and automated manner. A lack of centralized data and real-time updates can result in operational delays and customer dissatisfaction.

The problem this project addresses is the **inefficiency and unreliability of manual hotel management systems** and the need for a **centralized, automated, and user-friendly web application** that can streamline the daily operations of a hotel. The proposed system will serve as a solution by digitizing operations, improving data accuracy, enabling online reservations, and enhancing the overall customer experience.

➤ PRELIMINARY INVESTIGATION:

Purpose:

- To develop a web-based Hotel Management System that automates core hotel operations such as room booking, check-in/check-out, and billing.
- To create a centralized and secure system for managing hotel data and transactions.
- To improve user experience by offering online services to customers and efficient tools to hotel staff.

Benefits:

- **Automation of processes** reduces manual effort and operational time.
- **Real-time room availability** prevents overbooking and enhances customer satisfaction.
- **Improved data management** ensures accuracy and reduces chances of data loss.
- **Role-based access** increases security and organizes permissions (admin, and user).
- **Billing and report generation** becomes fast and reliable.
- **Scalability** allows the system to grow with hotel operations.
- **Customer convenience** improves with access to booking history and confirmations online.

Proposed System:

The proposed Hotel Management System is a **Django-based web application** designed to handle the day-to-day activities of a hotel. It offers:

- **User Registration & Authentication:** Secure login for administrators, staff, and customers.

- **Room Management:** Admins can add, edit, or remove room types, availability, and pricing.
- **Booking Module:** Customers can view available rooms and make online bookings.
- **Check-in/Check-out System:** Staff can update the status of guests and assign rooms.
- **Billing System:** Automated bill generation based on services availed.
- **Dashboard for Admins:** Real-time insights into room status, occupancy, and revenue.
- **Database Integration:** Uses a relational database (e.g., MySQL or PostgreSQL) to store all hotel data securely.
- **Responsive UI:** A clean interface built with HTML, CSS, and Bootstrap (or Tailwind) for usability across devices.

The system aims to **eliminate inefficiencies** of manual systems by offering a **centralized, scalable, and user-friendly solution** for hotel management.

Specifications:

The Hotel Management System is a web-based application developed using Python and the Django framework, with HTML, CSS, and JavaScript for the frontend and MySQL/PostgreSQL as the backend database. It is designed to run on any modern operating system (Windows) and supports all major web browsers such as Chrome, Firefox, and Edge. The system requires a minimum of 4 GB RAM, an Intel i3 or equivalent processor, and 100 GB of storage. Key functionalities include user authentication, room management, online booking, check-in/check-out handling, billing, and report generation, all accessible via role-based dashboards for admins, staff, and customers. The system is secure,

scalable, responsive across devices, and capable of handling multiple users simultaneously without performance degradation.

➤ **REQUIREMENT SPECIFICATIONS:**

The requirements that are to be satisfied from the proposed system are:

- **To ensure Reliability:** The system must consistently perform its projected functions over time, ensuring that hotel operations like booking, check-in, check-out, and billing work seamlessly without interruptions.
- **To ensure Good Organization:** The system should be wellorganized, performing its intended purpose of automating hotel management tasks like room management, guest registration, and report generation in a structured and efficient manner.
- **To ensure Correctness:** The system must meet the specifications defined in the project and fulfill user objectives by ensuring that all features, such as bookings, payment calculations, and customer data, are accurate and function correctly.
- **To ensure Usability:** The system should be easy to understand and operate for both staff and customers, with an intuitive interface for managing rooms, making bookings, and generating reports without the need for extensive training.
- **To ensure Maintainability:** The system must be designed so that errors can be detected, logged, and easily removed. Regular updates and bug fixes should be simple to implement, ensuring continuous operation of the hotel management tasks.
- **To ensure Expandability:** The system should be built in a way that allows for future growth and the addition of new features without disrupting the current operations. For example, the system should

easily accommodate new types of rooms, services, or additional user roles.

- **To ensure Portability:** The system should be easy to deploy across various hardware configurations and operating systems. This includes the flexibility to run on different servers or cloud platforms, ensuring the system can be used by various hotel locations or businesses.
- **To ensure Accurateness:** The system must ensure accurate input, editing, calculations, and productivity tracking. For example, booking confirmations, billings, and report data should always reflect precise and error-free information.
- **To ensure Error Handling:** The system should effectively detect errors and allow for quick correction. It must also prevent errors from occurring wherever possible, especially in critical areas like booking, billing, and room management.
- **To ensure Communication:** The system's inputs and outputs must be compatible with existing hotel operations. For instance, integrating with the hotel's payment gateway, or communicating with email services for booking confirmations, should work smoothly with the existing infrastructure.

➤ **FEASIBILITY STUDY:**

A feasibility study for the proposed **Hotel Management System** is an evaluation designed to determine the practicality and potential challenges of developing and implementing the system. Typically, the feasibility study precedes the technical development phase, providing a solid foundation for project implementation. In the context of the Hotel Management System, this study analyzes the impact of automating and streamlining hotel operations, such as room bookings, check-ins/outs, billing, and customer management.

The feasibility study for this project will involve a detailed analysis of the hotel industry and current operational processes. It will help predict the outcomes of adopting a new digital system that will enhance efficiency, reduce errors, and improve customer service. The feasibility study is a vital step in the software development process for this Hotel Management System. It will examine aspects such as:

- **Cost Analysis:** The financial resources required for the development, deployment, and maintenance of the system.
- **Time Analysis:** The estimated time required to develop each phase of the system, including design, development, testing, and deployment.
- **System Requirements:** Understanding the hardware and software requirements to ensure smooth operation of the system.

If these critical factors are not carefully analyzed, it could have a significant negative impact on the hotel's operations and the development of the system, leading to potential system failures or underperformance. The feasibility study also provides objective information to assess the strengths of the hotel business and evaluate the competition within the industry. It will help hotel managers understand the impact of implementing a digital solution and how it can align with their strategic goals. This study will reveal how the system can improve operations, optimize resource allocation, and enhance customer experiences, enabling better decision-making about the system's scope and design.

By conducting this feasibility study, hotel managers will gain a clear understanding of what the system can achieve, what resources are necessary for its success, and how to leverage the hotel's strengths to fully benefit from the new system? Ultimately, this study will provide insight into how to effectively plan and implement the Hotel Management System, ensuring a smooth transition and long-term success.

Steps in Conducting a Feasibility Study:

- The primary objective of conducting a feasibility study for the **Hotel Management System** is to evaluate whether the proposed system is practical and whether its development and implementation are economically viable. The feasibility study begins by analyzing the overall situation of the hotel and its existing operational processes. For example, it will assess the current challenges faced by hotel management, such as inefficient room booking systems, manual guest check-in processes, and difficulties in generating reports. The study will evaluate how the proposed system can address these challenges, streamline operations, and improve efficiency.
- The second part of the feasibility study will focus on estimating the costs and benefits of the proposed **Hotel Management System**. This will include analyzing the development costs, potential savings from automating processes, and the expected improvement in customer satisfaction. The study will also examine whether there are simpler or more cost-effective alternatives to the proposed system, such as improving the existing manual processes rather than implementing a complex software solution. If the proposed system is deemed both feasible and beneficial, the feasibility study will help in setting up a strategic implementation plan, including setting measurable goals and concrete steps for the system's development.
- Typically, a feasibility study for this project should be conducted by a qualified consultant with expertise in the hospitality industry and software development. The consultant would gather the necessary information and data to ensure that the feasibility study is accurate and objective. It is also crucial for the hotel's internal team to be involved in the process, providing access to the necessary data about current operations, staff roles, and business requirements.

Advantages of making Feasibility study:

- **Comprehensive System Analysis:** The feasibility study helps in analyzing the complete requirements of the system, ensuring that all aspects, such as room booking, guest management, billing, and report generation, are thoroughly evaluated.
- **Risk Identification:** The study helps identify potential risks involved in the development, deployment, and operation of the system, allowing the hotel to plan for risk management and mitigation strategies.
- **Cost/Benefit Analysis:** The feasibility study provides a clear analysis of the costs and benefits, helping the hotel determine whether the investment in the system will result in sufficient returns through operational efficiency, reduced errors, and improved customer satisfaction.
- **Training and Implementation Plans:** The study highlights the need for training developers and staff members, ensuring that they are well-prepared for the system's implementation.

□

Different Types of Feasibility:

1. **Technical Feasibility:** This examines whether the necessary technology is available and suitable for the project. The study will analyze the technical requirements for building and running the hotel management system, including server configuration, database management systems, and the compatibility of technologies used in Django, MySQL/PostgreSQL, and front-end tools (HTML, CSS, JavaScript).
2. **Schedule Feasibility:** This evaluates whether the project can be developed within the proposed timeline. The feasibility study will outline a realistic schedule for the design, development, testing, and

deployment phases of the Hotel Management System, ensuring that the system can be implemented within the required time frame.

3. **Social Feasibility:** This considers whether the proposed system will be accepted by the hotel's staff and customers. It will involve gauging how employees and customers will react to the new system and whether they will find it easy to use and beneficial. This can include user training for staff and ensuring the user interface is intuitive for customers making bookings.
4. **Legal Feasibility:** This assesses whether the system complies with relevant legal requirements, such as data protection laws (e.g., GDPR) and financial regulations. The system must ensure secure data handling and proper consent collection from users.
5. **Marketing Feasibility:** This examines the market forces that might affect the adoption and success of the system. The study will look at the competitive landscape in the hospitality industry, understanding how other hotels manage operations and identifying how this system can provide a competitive advantage.

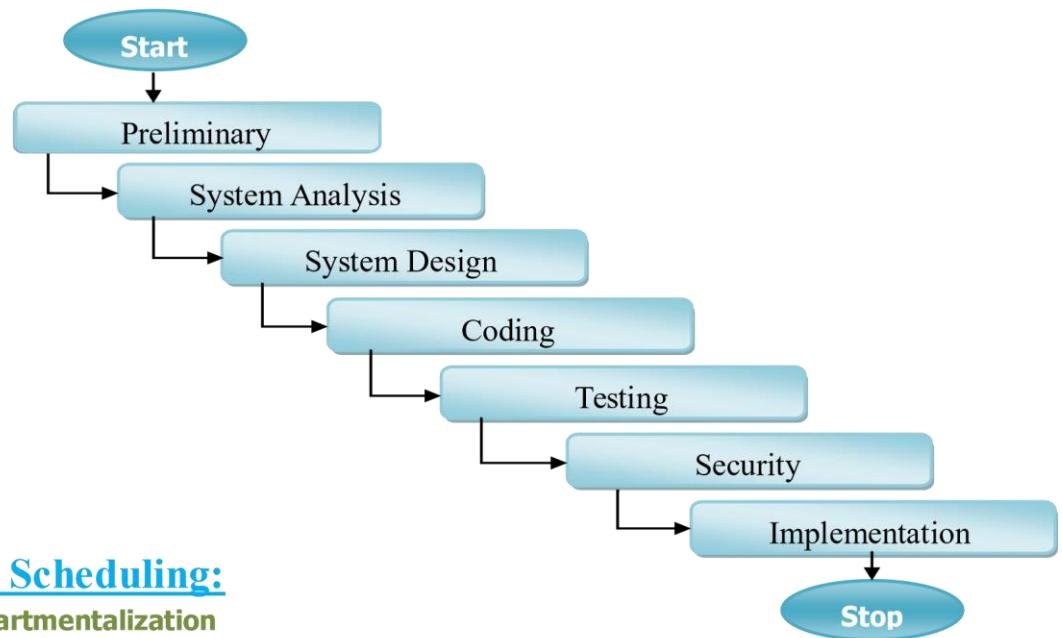
➤ **PROJECT PLANNING AND SCHEDULING:**

PROJECT PLANNING

■ **Project Planning**

■ **Phase to Cover**

1. Preliminary Investigation
2. System Analysis
3. System Design
4. Coding
5. Security
6. Testing
7. Implementation



Project Scheduling:

■ Compartmentalization

Activity Diagram:

Hotel Management System	
Preliminary Investigation	Identification of Need
	Requirement Specifications
	Feasibility Study
System Analysis	Project Planning & Scheduling
	SRS Preparations
	Software Engineering Paradigm
System Design	Module Description
	Database Design
	Procedural Design
	User Interface Design
Coding	Error free Coding
	Debugging
Testing	Unit Testing
	Integration Testing
	System Testing
Security	Application Security
	Database Security

Implementation	Implementation
	Documentation

➤ **SOFTWARE REQUIREMENT SPECIFICATION (SRS):**

Requirement analysis is conducted to fully understand the problem that needs to be solved. This is a vital step in the development of a database-driven application like the Hotel Management System. The process includes identifying both **data requirements** and **functional requirements**.

- **Data requirements** define the structure of data stored in the system.
- **Functional requirements** define the operations performed by users (like customers and administrators).

▪ **Data Requirements**

User Module

User Registration:

- Users can register by providing:
 - Full Name
 - Username (for login)
 - Password
 - Phone Number
 - Email Address

Room Search and Booking:

- Users can:
 - Check room availability by date and room type
 - View room details and prices
 - Book available rooms by providing necessary personal and payment information

- Select check-in and check-out dates, and the number of guests
- Receive booking confirmation via email or SMS

Admin Module Admin

Login:

- Admins can log in with a secure username and password.

Room Management:

- Add, update, or delete room categories (e.g., Deluxe, Suite, Standard)
- Manage room details: pricing, availability, amenities

Booking Management:

- View and manage current, upcoming, and past bookings
- Cancel or reschedule bookings

User Management:

- View registered users
- Remove inactive or fraudulent accounts
- Monitor booking activity

Reports:

- Generate reports on occupancy, income, and user activity

▪ **Functional Requirements**

1. Room Availability Search:

Users can search for rooms by date and room type using a query to check database availability.

2. Authentication System:

Both users and admins must log in to perform their specific Operations securely.

3. Point-and-Click Interface:

A user-friendly GUI will allow users to select room options, dates, and payment method easily.

4. Data Validation:

The system will verify user inputs and payment data before confirming any transactions.

Detailed Functional Modules

- a) **User Registration** – Enables one-time registration for users to simplify future bookings and access loyalty benefits.
- b) **Hotel Branch Information** – Provides information about different hotel branches, including location and facilities.
- c) **Room Details** – Displays room categories with pricing, photos, and included amenities.
- d) **Booking Slot Selection** – Allows users to select check-in/checkout dates and number of guests.
- e) **Room Selection** – After selecting dates, users choose the preferred room type from available options.
- f) **Booking Summary** – Displays all selected information before final booking confirmation.
- g) **Payment Module** – Handles payment processing and transaction records.

Software and Hardware Requirements

User Interface

The system will use a **Graphical User Interface (GUI)** designed with HTML, CSS, and Django templates, ensuring intuitive navigation and accessibility.

Software Constraints

The application will run on any operating system that supports Python, Django, and a web browser.

Hardware Constraints

The system is intended to run on Intel/AMD-based desktops or laptops with basic hardware specifications:

Development/Test Environment:

- 2 Desktop Systems with:
 - Intel Pentium 4 / AMD 2.0 GHz ◦ 128 MB DDR RAM (or higher) ◦ 40 GB Hard Disk ◦ 32 MB Video Card ◦ LAN Card
 - Internet connection (wired or wireless)
 - Peripherals: Keyboard, Mouse

Deployment/Production Environment:

- Server/Desktop with:
 - Intel Pentium 4 / AMD 2.0 GHz or higher
 - 512 MB RAM or higher ◦ 20 GB+ Hard Disk ◦ 16 MB+ Video Card
 - LAN card / Internet access

Software Requirements

For Development and Testing:

- Operating System: Windows 10/11, Linux, or macOS
- Python 3.x
- Django Framework
- PostgreSQL or MySQL Server
- Code Editor: VS Code / PyCharm
- Web Browsers: Google Chrome, Mozilla Firefox □ Office Suite: Microsoft Office / LibreOffice

For Deployment:

- OS: Windows Server / Ubuntu Server
- Django Web Server (Gunicorn/Apache)
- PostgreSQL / MySQL (Server-side)
- Nginx or Apache for reverse proxy
- SSL Certificate for secure HTTPS access
- Web Browser (for client access): Chrome, Edge, Firefox

➤ **FUNCTIONAL REQUIREMENTS:**

a) **User Registration** – Customers can register themselves on the hotel booking platform to receive loyalty discounts, exclusive offers, and faster future bookings. Upon registration, user details (like name, email, phone number, and address) are stored securely.

The user will not have to re-enter their details each time due to a unique user profile.

b) **Hotel Information** – This section displays a list of available hotels along with their branch locations, amenities (like Wi-Fi, pool, parking), ratings, and contact information. Each hotel will have its own detail page.

c) **Room Information** – This section provides a thumbnail and detailed view of various types of rooms (single, double, suite) available in each hotel. It includes room photos, pricing, facilities, and occupancy limits.

d) **Room Availability & Date Selection** – This section allows users to check available rooms for their selected dates and filter based on room type, hotel branch, and price range.

e) **Room Selection** – After checking availability, users can choose their desired room(s). They can specify check-in and check-out dates, number of guests, and special requirements (e.g., smoking/non-smoking).

f) **Booking Summary** – This section compiles all booking-related information such as selected hotel, room type, stay duration, price, user details, and any additional services like meals or transport.

g) **Payment and Transaction Process** – The system integrates with secure payment gateways (e.g., Razor pay, Stripe) to handle bookings. It maintains a record of all transaction details, provides booking confirmation, and sends an email/SMS receipt to the customer.

➤ **SOFTWARE ENGINEERING PARADIGM:**

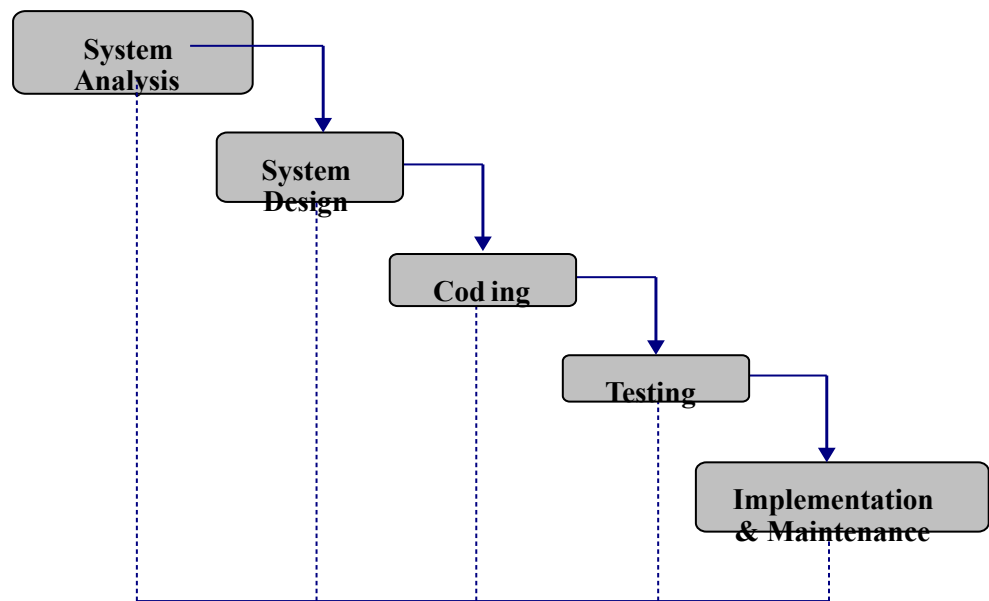
The conventional and mostly used software engineering paradigm till now the Waterfall Model is applied on this project. But according to project complexity and nature slight deviation from the proposed original waterfall model is taken into consideration. The three characteristics of Waterfall Model as linear, rigid, and monolithic are diverted here.

The reasons and assumptions are explained below:

The linearity of Waterfall Model is based on the assumption that software development may proceed linearly from analysis down to coding. But, in this project feedback loops are applied to the immediately preceding stages.

Another underlying assumption of the Waterfall Model is phase rigidity – i.e., that the results of each phase are frozen before proceeding to the next phase. But, this characteristic is not strictly maintained throughout the project. Therefore overlapping of two or more phases is allowed according to needs and judgments.

Finally, the Waterfall Model is monolithic in the sense that all planning is oriented to single delivery date. But since this project has deadline imposed by the organization, planning is done into successive phases. This project is the term-end project and obviously need to submit as soon as possible calendar date. With such considerations is selected the Waterfall model as the project development paradigm.



Water-fall Model with feedback mechanism for the proposed application as Software Engineering paradigm:

ANALYSIS DIAGRAMS DFD (Data Flow Diagram):

In the late 1970s *data-flow diagrams (DFDs)* were introduced and popularized for structured analysis and design (Game and Sarsen 1979). DFDs show the flow of data from external entities into the system, showed how the data moved from one process to another, as well as its logical storage.

A **data flow diagram (DFD)** is a significant modeling technique for analyzing and constructing information processes. DFD literally means an illustration that explains the course or movement of information in a process. DFD illustrates this flow of information in a process based on the inputs and outputs. A DFD can be referred to as a Process Model. Data

flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest.

As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating DFDs using easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.

Additionally, a **DFD** can be utilized to visualize data processing or a structured design. A DFD illustrates technical or business processes with the help of the external data stored, the data flowing from a process to another, and the results.

A designer usually draws a context-level DFD showing the relationship between the entities inside and outside of a system as one single step. This basic DFD can be then disintegrated to a lower level diagram demonstrating smaller steps exhibiting details of the system that is being

modeled. Numerous levels may be required to explain a complicated system.

Process

The process shape represents a task that handles data within the application. The task may process the data or perform an action based on the data.

Multiple Processes

The multiple process shape is used to present a collection of sub processes. The multiple processes can be broken down into its sub processes in another DFD. **External Entity**

The external entity shape is used to represent any entity outside the application that interacts with the application via an entry point. **Data Flow**

The data flow shape represents data movement within the application. The direction of the data movement is represented by the arrow. **Data Store**

The data store shape is used to represent locations where data is stored.

Data stores do not modify the data, they only store data.

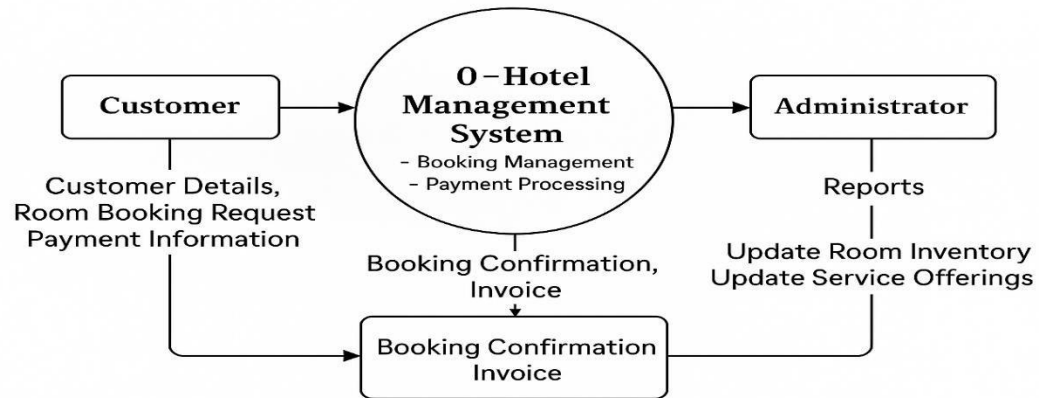
Privilege Boundary

The privilege boundary shape is used to represent the change of privilege levels as the data flows through the application.

Examples of Data Flow Diagrams - These examples demonstrate how to draw data flow diagram.

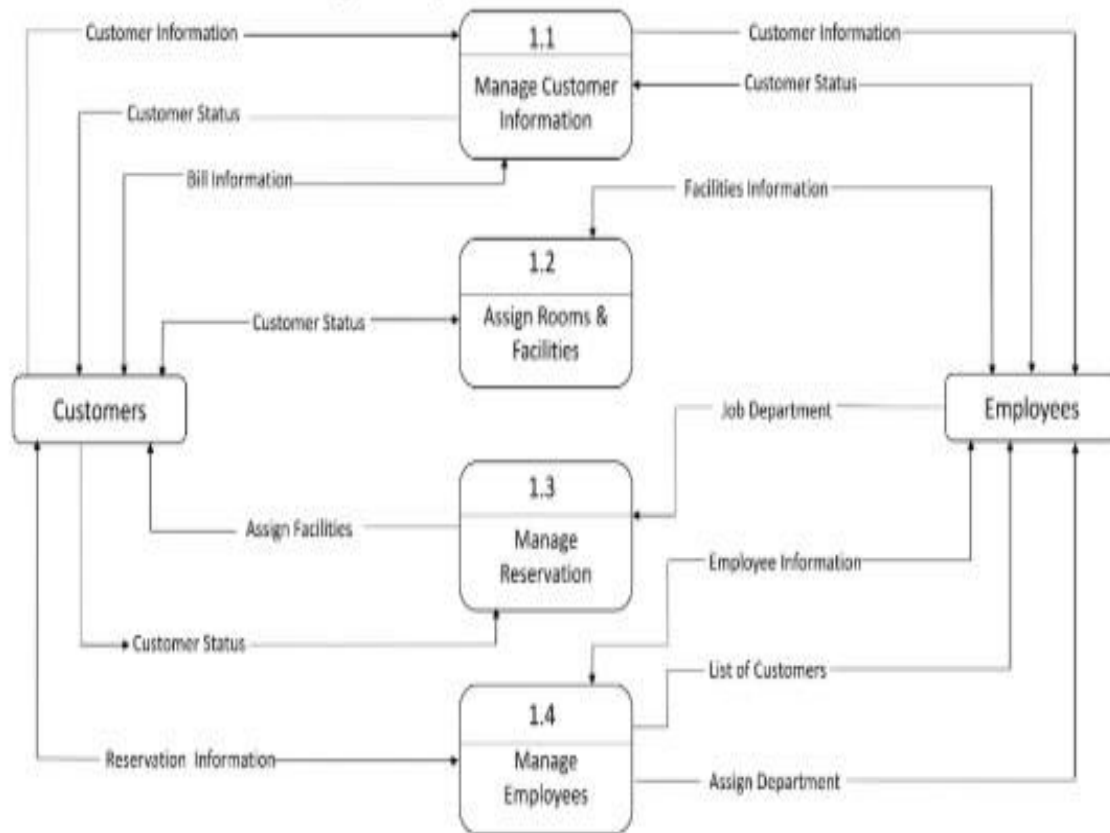
Before it was eventually replaced, a copy machine suffered frequent paper jams and became a notorious troublemaker. Often, a problem could be cleared by simply opening and closing the access panel. Someone observed the situation and flowcharted the troubleshooting procedure used by most people.

CONTEXT LEVEL DIAGRAM

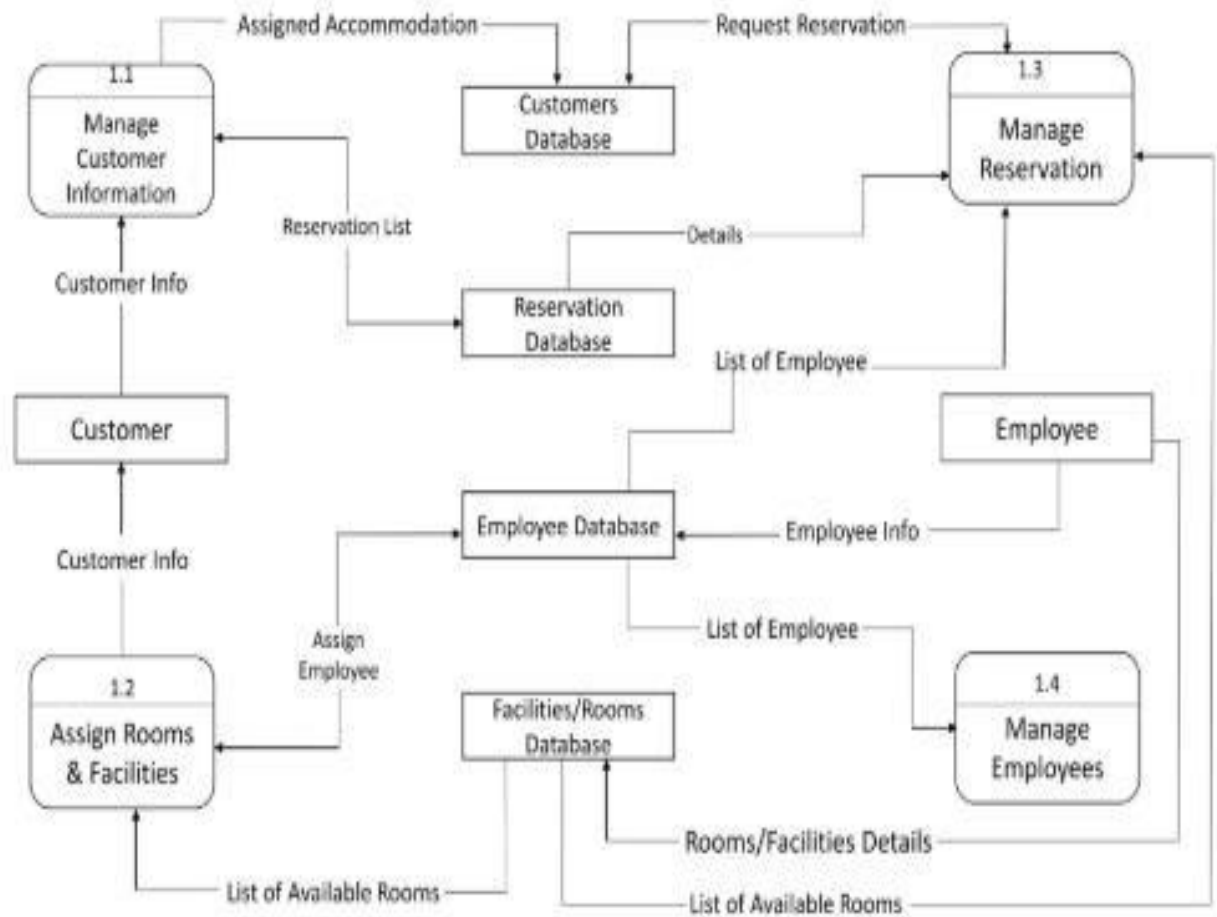


Level1 DFD Diagram:

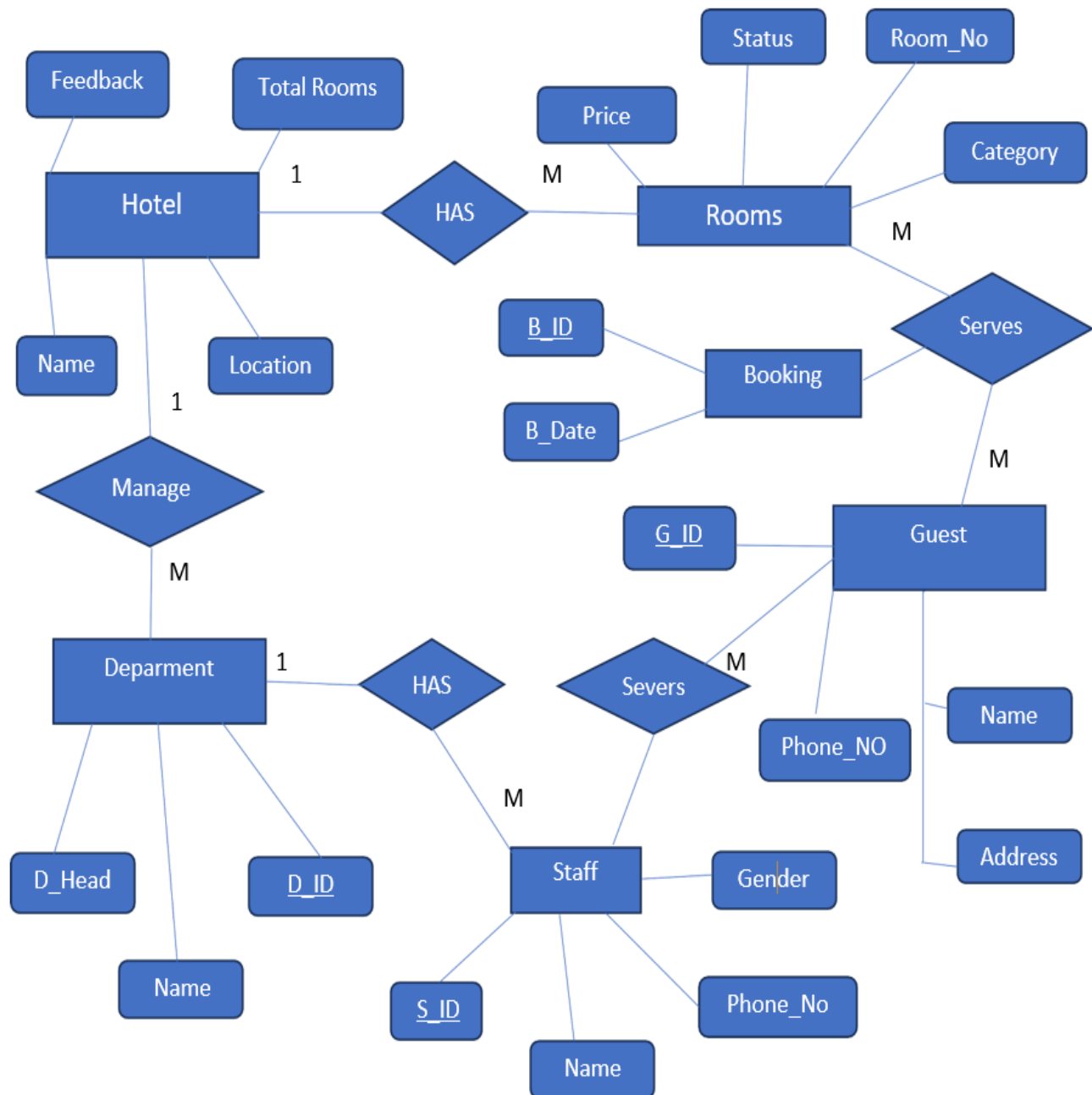
Level 1 DFD for Hotel Management System



Level 2 DFD Diagram:



Entity Relationship Diagram (ERD):





➤ **MODULES AND THEIR DESCRIPTION**

The various modules used in the project are as follows:
MODULES AND THEIR DESCRIPTION

- a) **User Registration** – Customers can register themselves to make future bookings easier and more personalized. Once registered, users do not need to enter their personal details repeatedly, as their profile and one-time registration credentials will automatically be used for subsequent bookings.
- b) **Hotel Information** – This section displays a list of hotels along with their branch locations. Users can browse through different hotels, view amenities offered, ratings, and general information to help them decide on their stay.
- c) **Room Information** – This section provides a thumbnail and brief view of the available room types (e.g., Deluxe, Suite, Standard) along with details like price per night, capacity, features, and images.
- d) **Room Availability and Date Selection** – Users can explore the available rooms in a selected hotel by selecting check-in and check-out dates. The system shows real-time availability for the selected period.
- e) **Room Selection** – After selecting the dates and hotel, users can choose their preferred room type based on their needs and availability.

- f) **Booking Information** – This section compiles all the booking-related details including user information, hotel and room selected, duration of stay, and the total cost. It allows users to confirm and finalize the booking.
- g) **Transaction Process** – This module manages the payment process and stores the transaction details securely. It supports various payment methods and generates receipts/invoices for each booking.

➤ **DATA STRUCTURE OF ALL MODULES:**

```
mysql> show tables;
+-----+
| Tables_in_hotel_db |
+-----+
| auth_group          |
| auth_group_permissions |
| auth_permission     |
| auth_user           |
| auth_user_groups    |
| auth_user_user_permissions |
| django_admin_log     |
| django_content_type  |
| django_migrations    |
| django_session       |
| homepage_billing     |
| homepage_bookedservice |
| homepage_booking     |
| homepage_booking_services |
| homepage_feedback    |
| homepage_guest       |
| homepage_room        |
| homepage_roomtype    |
| homepage_service     |
| homepage_staff       |
+-----+
```

```
mysql> desc homepage_feedback;
```

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
name	varchar(255)	NO		NULL	
email	varchar(254)	YES		NULL	
feedback_text	longtext	NO		NULL	
message	longtext	NO		NULL	
submitted_at	datetime(6)	NO		NULL	

```
6 rows in set (0.00 sec)
```

```
mysql> desc homepage_guest;
```

Field	Type	Null	Key	Default	Extra
GuestID	int	NO	PRI	NULL	auto_increment
FirstName	varchar(100)	NO		NULL	
LastName	varchar(100)	NO		NULL	
Email	varchar(100)	NO		NULL	
CNIC	varchar(100)	NO		NULL	
Password	varchar(100)	NO		NULL	
Address	varchar(100)	NO		NULL	
ConfirmPassword	varchar(100)	NO		NULL	
ContactNo	varchar(100)	NO		NULL	

```
9 rows in set (0.00 sec)
```

```
mysql> desc homepage_booking;
```

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
check_in_date	date	NO		NULL	
check_out_date	date	NO		NULL	
booking_date	datetime(6)	NO		NULL	
guest_id	int	NO	MUL	NULL	
room_id	varchar(10)	NO	MUL	NULL	

```
6 rows in set (0.00 sec)
```

```
mysql> desc homepage_booking_services;
```

Field	Type	Null	Key	Default	Extra
id	bigint	NO	PRI	NULL	auto_increment
booking_id	bigint	NO	MUL	NULL	
service_id	int	NO	MUL	NULL	

```
3 rows in set (0.00 sec)
```

```
mysql> desc homepage_room;
```

Field	Type	Null	Key	Default	Extra
room_number	varchar(10)	NO	PRI	NULL	
occupancy	varchar(10)	NO		NULL	
room_status	varchar(10)	NO		NULL	
room_type_id	varchar(100)	NO	MUL	NULL	

```
4 rows in set (0.00 sec)
```

```
mysql> desc homepage_roomtype;
```

Field	Type	Null	Key	Default	Extra
room_type	varchar(100)	NO	PRI	NULL	
room_desc	varchar(100)	NO		NULL	
roomtype_price	decimal(10,2)	NO		NULL	
room_image	varchar(200)	NO		NULL	

```
4 rows in set (0.00 sec)
```

```
mysql> desc homepage_service;
```

Field	Type	Null	Key	Default	Extra
service_id	int	NO	PRI	NULL	auto_increment
service_name	varchar(100)	NO		NULL	
description	varchar(100)	NO		NULL	
service_image	varchar(200)	NO		NULL	
service_price	decimal(10,2)	NO		NULL	

```
5 rows in set (0.00 sec)
```


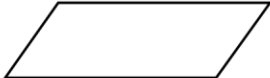
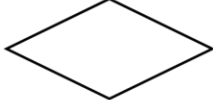


```
mysql> desc homepage_staff;
```

Field	Type	Null	Key	Default	Extra
staff_id	int	NO	PRI	NULL	auto_increment
first_name	varchar(100)	NO		NULL	
last_name	varchar(100)	NO		NULL	
role	varchar(100)	NO		NULL	
gender	varchar(100)	NO		NULL	
contact_no	varchar(100)	NO		NULL	

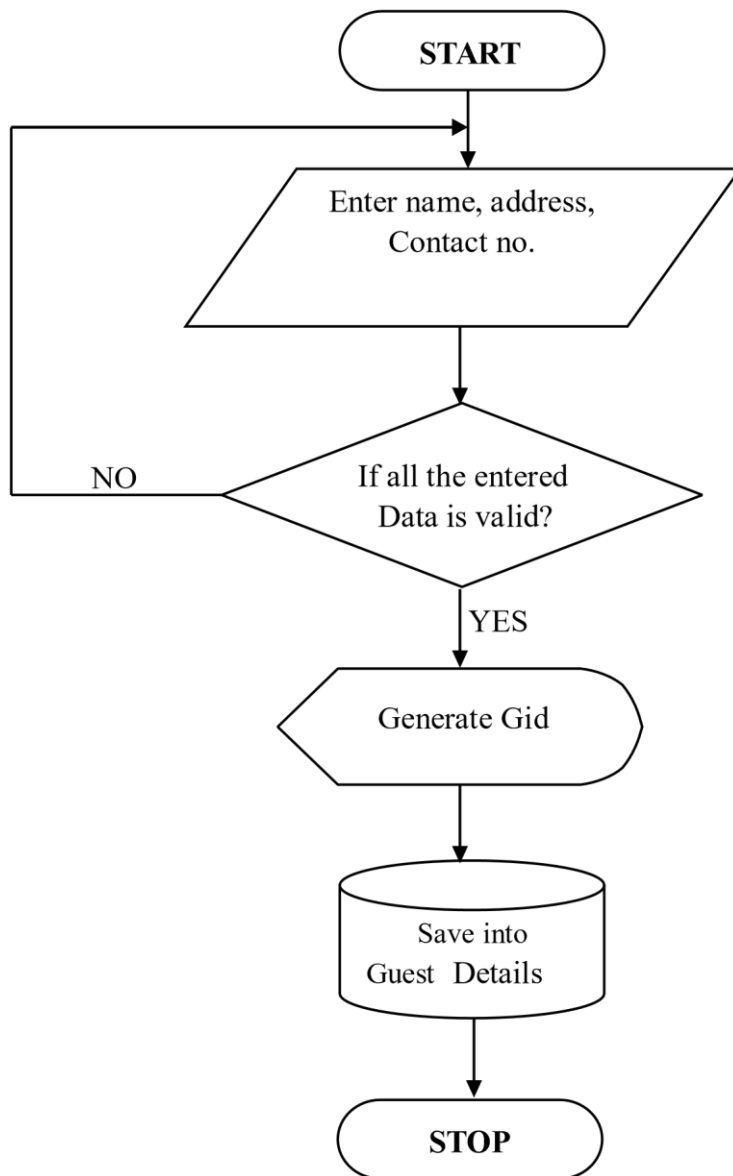
```
6 rows in set (0.00 sec)
```

PROCEDURAL DESIGN:

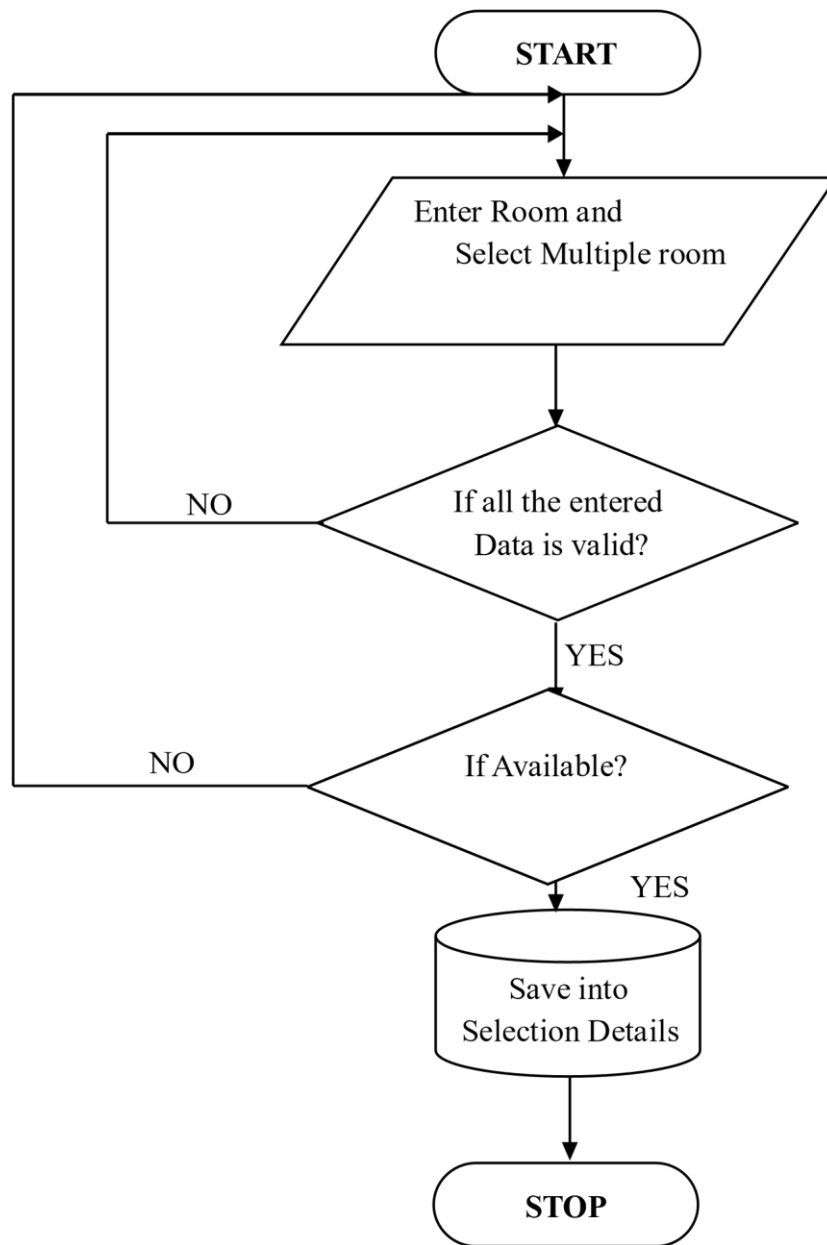
PROCESS LOGIC (FLOW CHART) OF EACH MODULE

<u>Notations Used For FLOW CHART</u>		
NOTATIONS		USED FOR
	Known as TERMINATOR	Used for START / STOP
	Known as DATA	Used for ACCEPTING INPUTS FROM USER AND also DISPLAYING THE ERROR MESSAGES GENERATED BY THE SYSTEM
	Known as DECISION	Used for DECISION MAKING
	Known as DISPLAY	Used for OUTPUT GENERATED BY THE SYSTEM
	Known as MAGNETIC DISK	Used for STORING INPUTS GIVEN TO THE SYSTEM

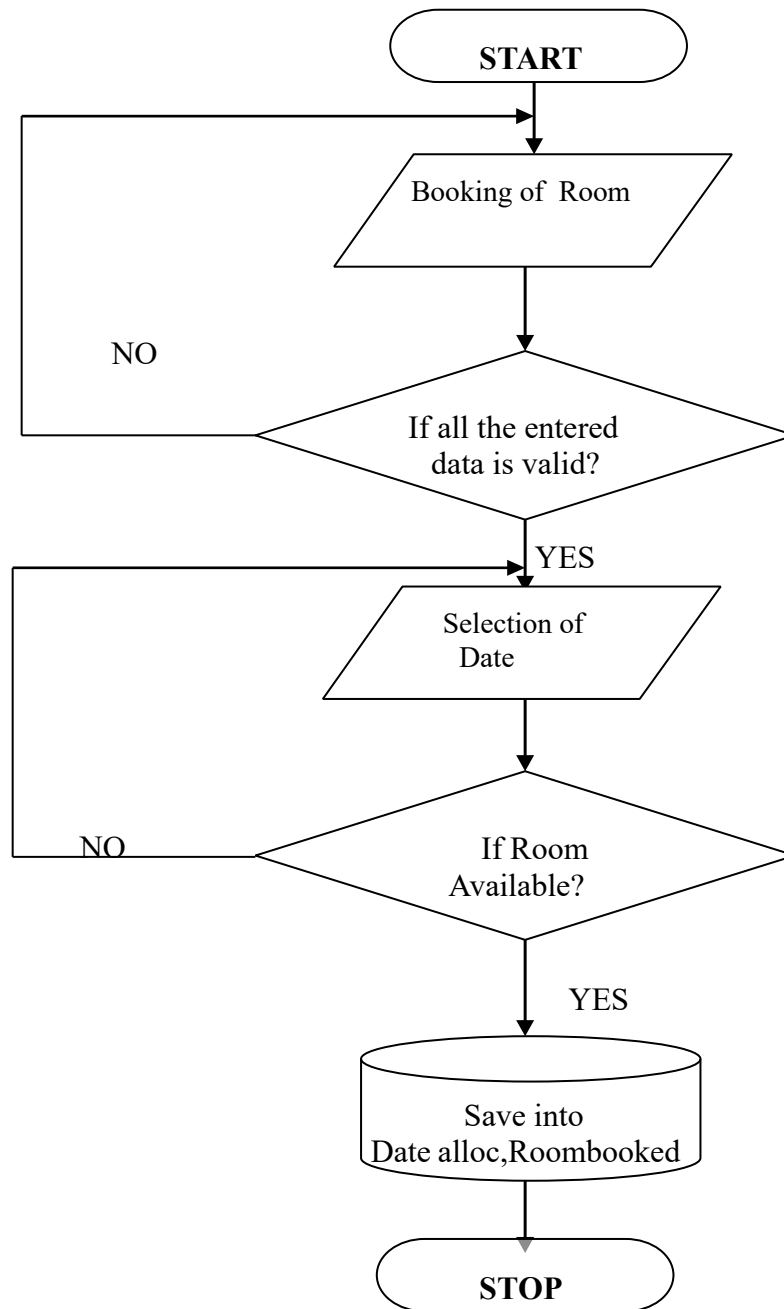
1. Flow chart for “Guest Registration”



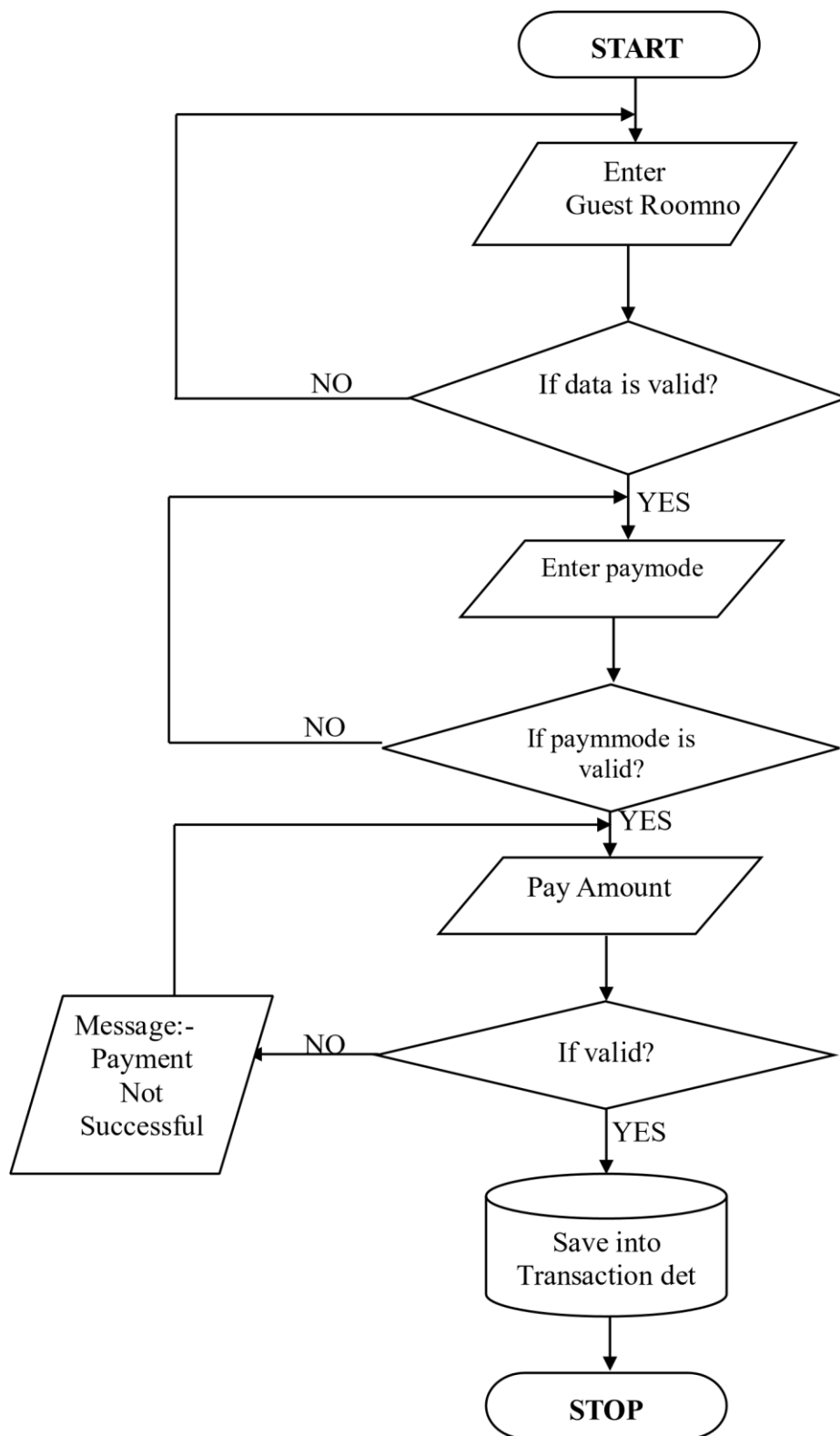
2. Flow chart for “Room Selection”



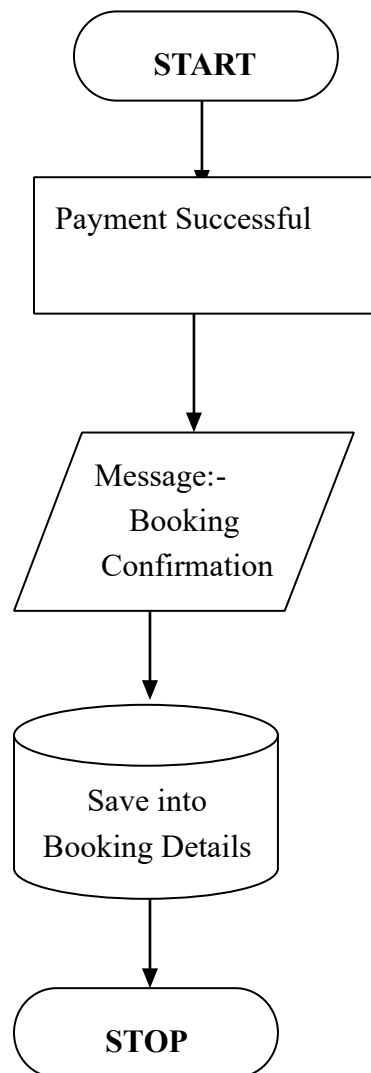
3. Flow chart for “Room Booking”



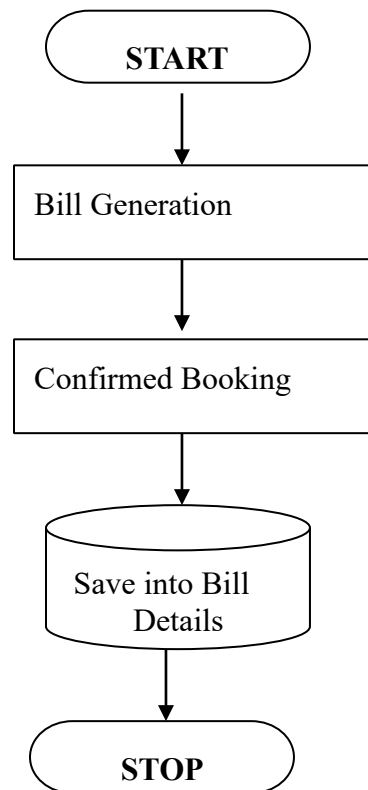
4. Flow chart for “Transaction Process”



5. Flow chart for “Booking Confirmation”

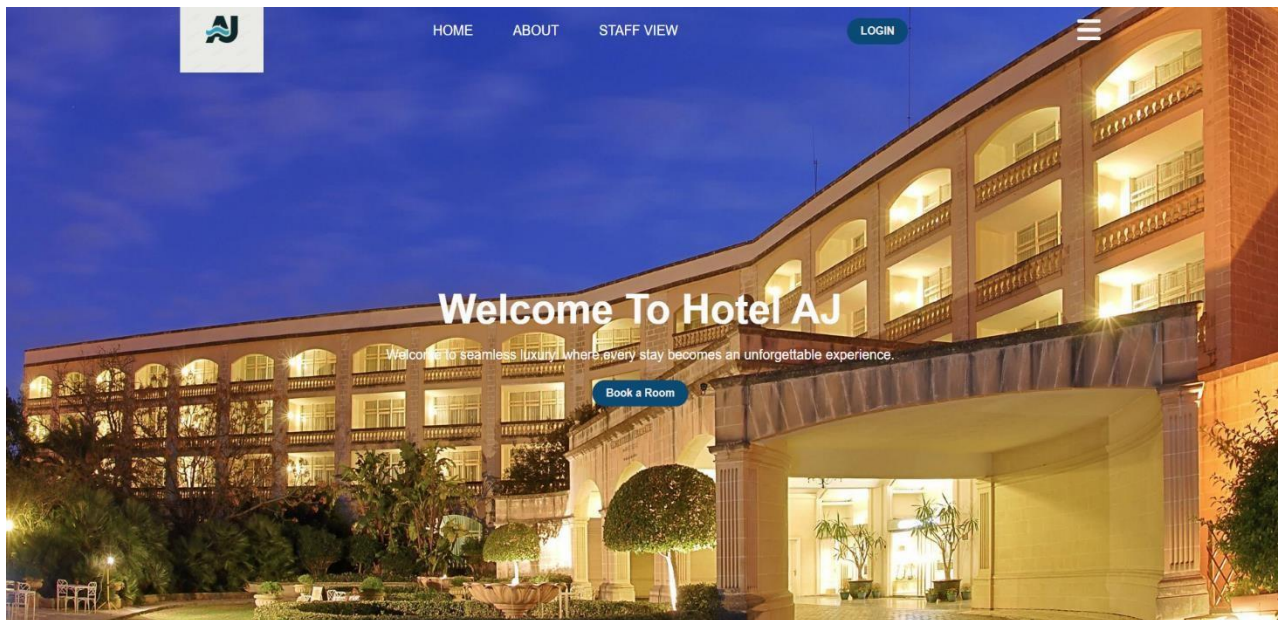


6. Flow chart for “Bill Generation”



INPUT SCREENS:

Home page:



Bookings Page:

Bookings | [Home](#) > [Homepage](#) > Bookings

Go

0 of 8 selected

Guest

Room

Check in date

Check out date

[Guest object \(20\)](#)

Room 102 - booked

May 15, 2025

May 17, 2025

[Guest object \(13\)](#)

Room 101 - available

May 16, 2025

May 17, 2025

[Guest object \(13\)](#)

Room 103 - available

May 7, 2025

May 10, 2025

[Guest object \(14\)](#)

Room 102 - booked

May 15, 2025

May 16, 2025

[Guest object \(14\)](#)

Room 101 - available

May 14, 2025

May 15, 2025

[Guest object \(13\)](#)

Room 102 - booked

May 7, 2025

May 9, 2025

[Guest object \(12\)](#)

Room 103 - available

May 7, 2025

May 9, 2025

[Guest object \(5\)](#)

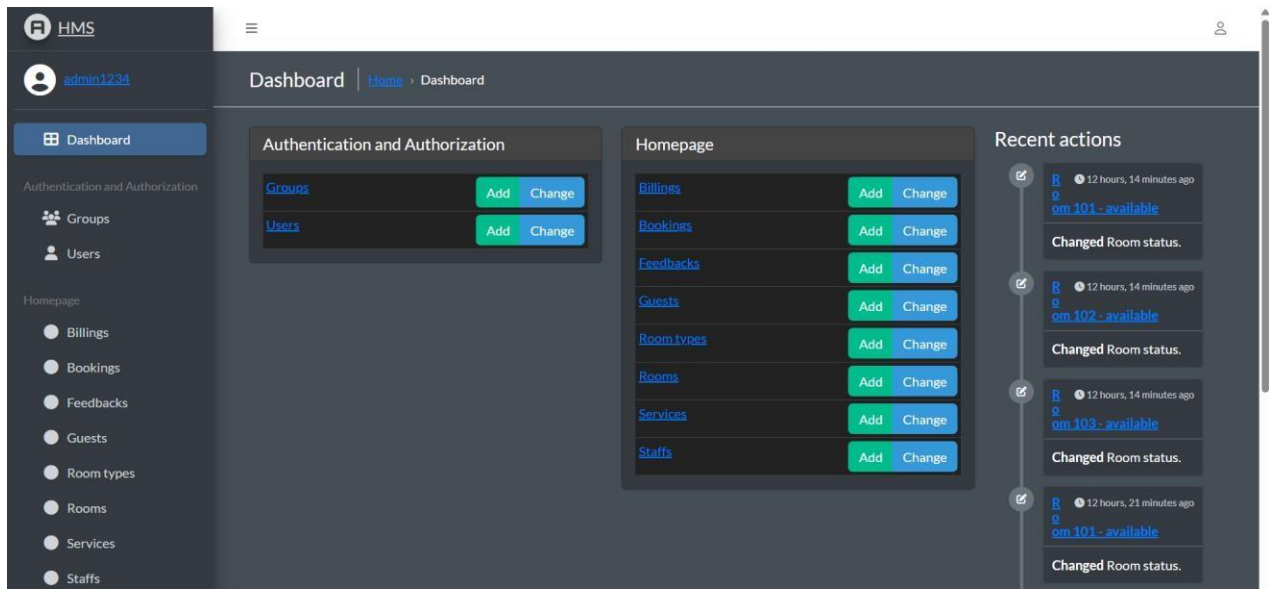
Room 102 - booked

May 5, 2025

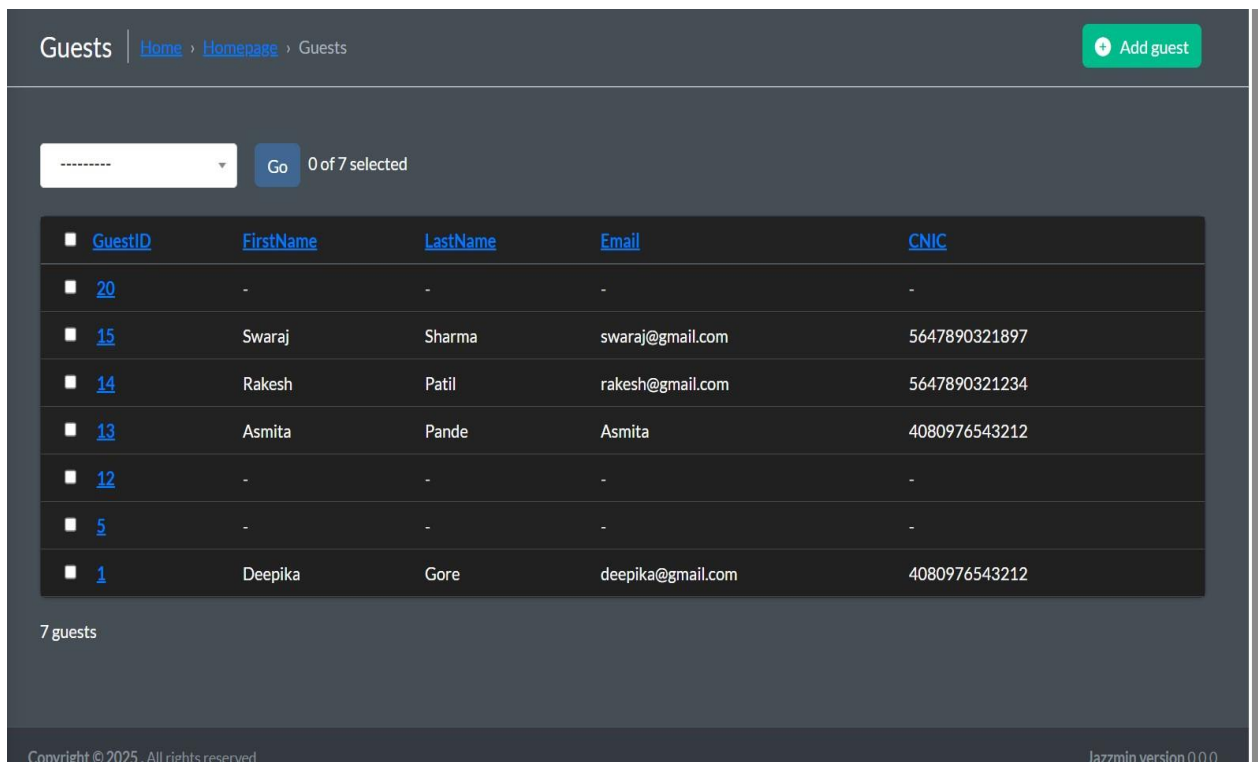
May 6, 2025

8 bookings

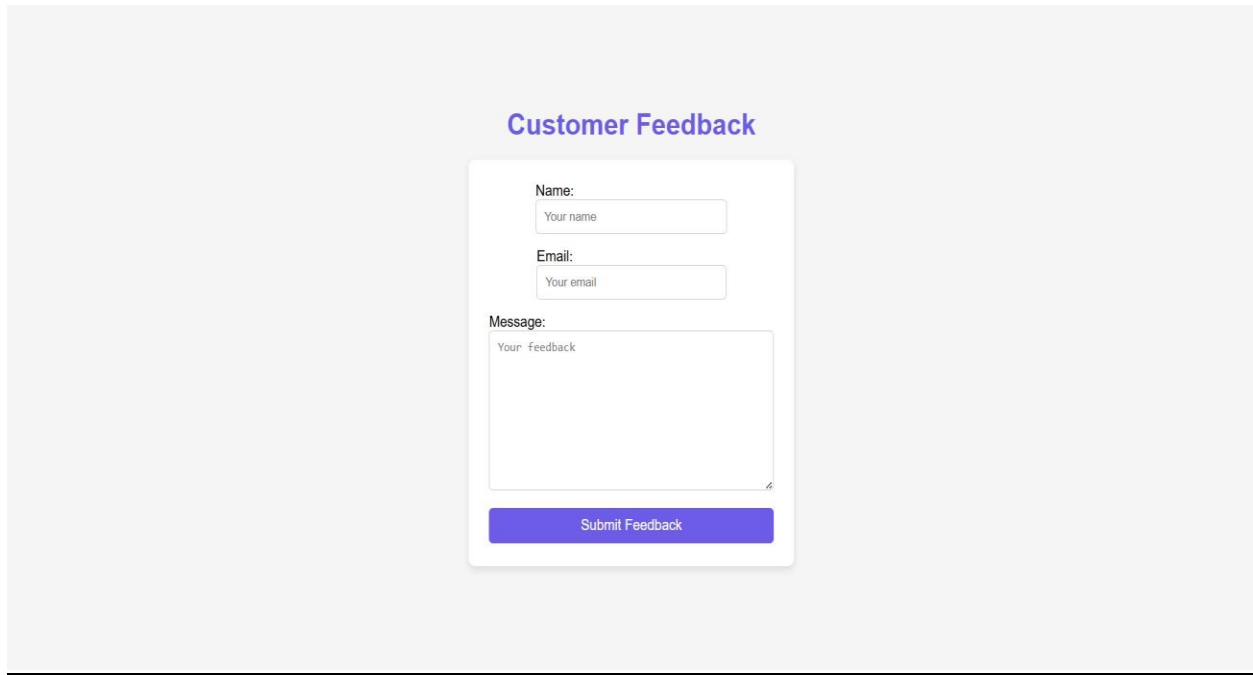
Dashboard page:



Guest's page:



Customer Feedback:



The image shows a web form titled "Customer Feedback" in a purple font. The form is centered on a light gray background. It contains three input fields: "Name:" with a placeholder "Your name", "Email:" with a placeholder "Your email", and "Message:" with a placeholder "Your feedback". Below these fields is a purple button labeled "Submit Feedback".

Customer Feedback

Name:
Your name

Email:
Your email

Message:
Your feedback

Submit Feedback

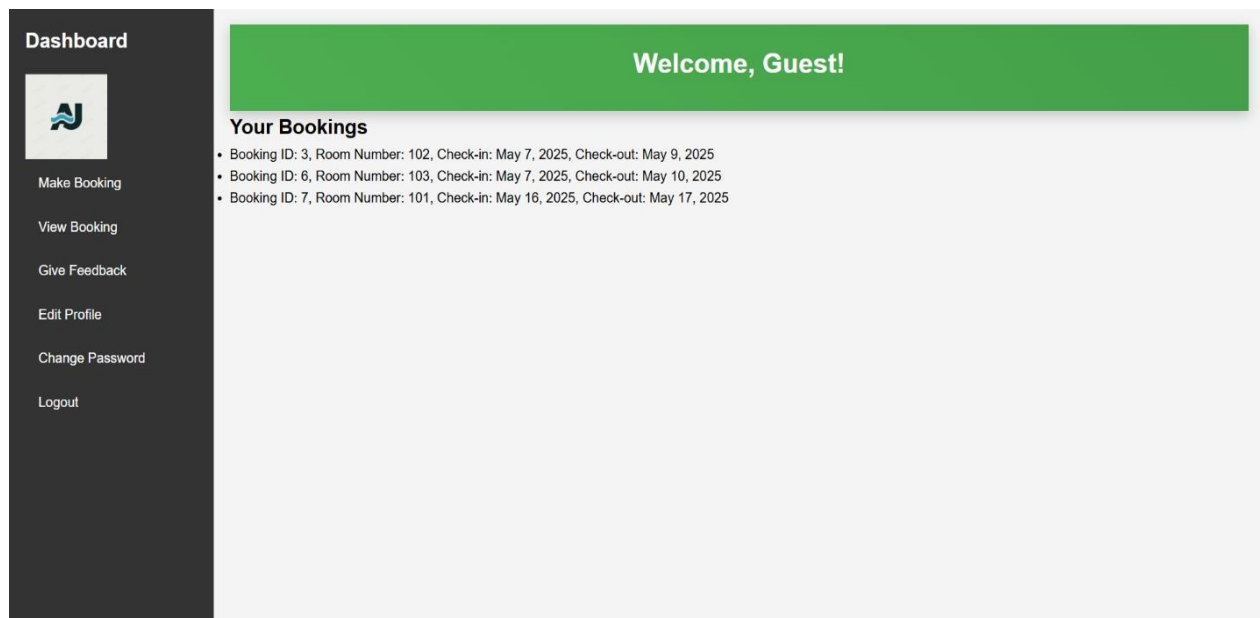
Room booking-Type page:

Book Room - Single

Available Single Rooms

Room Number	Occupancy	Status	Check-In Date	Check-Out Date	Action
101	1	available	<input type="text" value="dd-mm-yyyy"/>	<input type="text" value="dd-mm-yyyy"/>	Book Now

Welcome Guest page:



➤ **LIST OF REPORTS THAT ARE LIKELY TO BE GENERATED:**

The types of reports that can be generated include:

✓ **Room Category-wise Report**

- Generates lists of rooms based on their category (e.g., Deluxe, Standard, Suite, Executive).

✓ **Hotel Branch-wise and Brand-wise Report**

- Displays performance data of different hotel branches under a specific brand or chain.

✓ **Daily / Weekly / Monthly / Yearly Booking Trends**

- Analyzes booking patterns over time to help in understanding peak seasons and customer behavior.

✓ **Booking Statistics on Special Occasions and Holidays**

- Highlights room occupancy and revenue during holidays, festivals, and other high-traffic dates.
- ✓ **Profit Reports from Room Bookings and Services**
 - Calculates profits generated from room bookings and additional services like dining, spa, etc.
- ✓ **Room-Type Wise Booking Trends**
 - Compares occupancy rates among different types of rooms over selected time periods.
- ✓ **Payment Mode-wise Booking Statistics**
 - Summarizes how customers are paying (e.g., Credit Card, Debit Card, UPI, Net Banking, Cash).
- ✓ **Customer Satisfaction and Service Ratings**
 - Ranks rooms, amenities, or services based on customer reviews and feedback.
- ✓ **Location-wise Guest Traffic Analysis**
 - Shows statistics of guest origins (local, domestic, international) and most-visited branches.
- ✓ **Branch-wise Revenue and Profit Reports**
 - Tracks individual hotel branches for their financial performance and growth.



```
Homepage      Urls.py      from
django.contrib import admin from
django.shortcuts import redirect from
django.urls import path, include from
HomePage import views
from django.contrib.auth import views as auth_views

urlpatterns = [
    path("", views.home, name="home"),          path('register',
    views.register,                             name="register"),
    path('login/', views.login_view, name="login"),
    path('booking_page', views.booking_page, name='booking'),
    path('detail/<str:selected_room_type>', views.detail_page,
    name='detail'),
    #path('contact/', contact_us, name='contact_us'),
    path('billing_page/<int:booking_id>', views.billing_page,
    name='billing_page'),
    path('confirm_booking/', views.confirm_room,
    name='confirm_booking'),
    path('service-selection/', views.service_selection,
    name='service_selection'),
    path('about', views.about, name='about'),
    path('afterlogin/', views.after_login, name="after_login"),
```

```

    #path('service-selection/<int:service_id>/', views.service_selection,
name='service_detail'),
    path('book-service/<int:booking_id>/', views.book_service,
name='book_service'),
    path('staff/', views.staff_viewer, name='Staff_Viewer'), # Define the
URL pattern for staff viewer
    path('dashboard/', views.dashboard, name="dashboard"),
path('change-password/',
auth_views.PasswordChangeView.as_view(), name='passwordChange'),
path('accounts/login/', lambda request: redirect('/login/')),
path('change-password/',
auth_views.PasswordChangeView.as_view(success_url='/changepasswor
d-done/'), name='change_password'), path('change-password-done/',
auth_views.PasswordChangeDoneView.as_view(),
name='password_change_done'),
path('password_change/',
auth_views.PasswordChangeView.as_view(), name='password_change'),
path('password_reset/', auth_views.PasswordResetView.as_view(),
name='password_reset'), path('password_reset/done/',
auth_views.PasswordResetDoneView.as_view(),
name='password_reset_done'),
path('reset/<uidb64>/<token>/',
auth_views.PasswordResetConfirmView.as_view(),
name='password_reset_confirm'),
path('reset/done/',
auth_views.PasswordResetCompleteView.as_view(),
name='password_reset_complete'),

```

```

    path('profileChange/', views.profileChange, name='profileChange'),
    path('logoutview',      views.logoutview,      name='logoutview'),
    path('regComplete',    views.regComplete,    name='regComplete'),
    path('checkout/',      views.create_checkout_session,
name='create_checkout_session'),
    path('success/', views.success, name='success'), path('cancel/',
views.cancel,      name='cancel'),    path('incorrectDetails',
views.incorrectDetails,
name='incorrectDetails'),
    path('billing/<int:booking_id>/',      views.billing_page,
name='billing_page'),    path('receipt/<int:booking_id>/',
views.receipt,    name='receipt'),    path('download-
receipt/<int:billing_id>/',      views.download_receipt,
name='download_receipt'),
    path('feedback/', views.feedback_view, name='feedback'),
    path('thankyou/', views.feedback_thankyou, name='feedback_thankyou'),
]

```

[Homepage views.py](#)

```

from django.http import HttpResponseRedirect from
.models import Guest
from django.contrib.auth.models import User from
django.contrib.auth import update_session_auth_hash from
django.contrib.auth import authenticate, login, logout from
.models import *
from django.shortcuts import render, redirect from
django.contrib.auth.decorators import login_required
from django.shortcuts import render, get_object_or_404

```

```

import os import stripe from django.conf import settings
from django.http import HttpResponseRedirect from
django.template.loader import get_template from
xhtml2pdf import pisa
from .models import Billing, Booking, Guest from
.forms import FeedbackForm
# Create your views here.

def home(request):
    return render(request,"homePage.html")
from django.contrib import messages from
django.shortcuts import render, redirect from
django.core.validators import validate_email from
django.core.exceptions import ValidationError
from django.contrib.auth.models import User from
.models import Guest
import re def register(request):    if request.method ==
'POST':        # Extract form data        first_name =
request.POST.get('FirstName', "").strip()        last_name =
request.POST.get('LastName', "").strip()        email =
request.POST.get('Email', "").strip()        cnic =
request.POST.get('CNIC', "").strip()        password =
request.POST.get('Password', "")
        confirm_password = request.POST.get('ConfirmPassword',
")        address = request.POST.get('Address', "").strip()
contact_no = request.POST.get('ContactNo', "").strip()

```



```

        # Field Validations
errors = []
if not first_name:
    errors.append("First name is required.")
if not last_name:
    errors.append("Last name is required.")
if not email:
    errors.append("Email is required.")
else:
    try:
        validate_email(email)
    except User.objects.filter(email=email).exists():
        errors.append("Email already exists.")
    except ValidationError:
        errors.append("Invalid email format.")
if not cnic or not re.fullmatch(r'\d{13}', cnic):
    errors.append("Valid 13-digit CNIC is required.")
if not password or len(password) < 6:
    errors.append("Password must be at least 6 characters.")
if password != confirm_password:
    errors.append("Passwords do not match.")
if not address:
    errors.append("Address is required.")
if not contact_no or not re.fullmatch(r'\d{10,11}', contact_no):
    errors.append("Contact number must be 10 or 11 digits.") # If
there are errors, re-render form with messages
    if errors:
        for error in errors:
            messages.error(request, error)
        return render(request, 'registration.html')
        # Create user and guest record

```

```

        myUser = User.objects.create_user(email, email,
password)      myUser.first_name = first_name
myUser.last_name = last_name      myUser.username = email
        myUser.save()
new_guest = Guest(
FirstName=first_name,
        LastName=last_name,
        Email=email,
        CNIC=cnic,
        Password=password,
        ConfirmPassword=confirm_password,
        Address=address,
        ContactNo=contact_no
    )      new_guest.save()      return
redirect('regComplete')      return
render(request, 'registration.html')  def
booking_page(request):
    # Retrieve room types from the database
    room_types = RoomType.objects.all() # Fetch all room types (this
could be more specific as needed)
    # Pass room types data to the template as context
    return render(request, 'booking.html', {'room_types': room_types})
def login_view(request):  if request.method == 'POST':      email
= request.POST.get('email')
        password = request.POST.get('password')

```

```

        # Authenticate using email as username            user =
authenticate(request, username=email, password=password)    if user
is not None:          # Log in the user            login(request, user)

        # Redirect to the booking page or any desired URL
return redirect('dashboard')    else:

        # Invalid login, display an error message
return redirect('incorrectDetails')    return
render(request, 'login.html') def
incorrectDetails(request):    return
render(request, "incorrectDetails.html") def
about(request):    return render(request,
'aboutUs.html') #def contact_us(request):
    return HttpResponseRedirect("Contact page placeholder")
def    detail_page(request,    selected_room_type):
available_rooms =
Room.objects.filter(room_type__room_type=selected_room_type)
    context = {
        'selected_room_type': selected_room_type,
        'available_rooms': available_rooms,
    }
    return render(request, 'roomDetails.html', context)

def    service_selection(request):
services = Service.objects.all()
    return render(request, 'serviceSelection.html', {'services': services}) def
billing_page(request):

```

```

    return render(request, 'billing.html') def
after_login(request):    return render(request,
'afterlogin.html') def confirm_booking(request):
return render(request, 'booking_confirmation.html')
from django.shortcuts import render from .models
import Guest def dashboard(request):    if
request.user.is_authenticated:        try:
        # Fetch all Guest instances associated with the logged-in user's
email
        guest_instances = Guest.objects.filter(Email=request.user.email)
if guest_instances.exists():
        # Assuming you want to use the first Guest instance found
guest_instance = guest_instances.first()
        user_bookings = guest_instance.booking_set.all() # Access
bookings using related name

        return render(request, "dashboard.html", {'bookings':
user_bookings})
else:
        return HttpResponseRedirect("Guest information not found.")
except Guest.DoesNotExist:        return HttpResponseRedirect("Guest
information not found.")    else:        return HttpResponseRedirect("Please log in
to view the dashboard.") def profileChange(request):    guest =
Guest.objects.get(Email=request.user.email) #left side guest is variable,
right side Guest is model name    if request.method == "POST":
username = request.POST.get('username')        first_name =
request.POST.get('first_name')        last_name =
request.POST.get('last_name')        email = request.POST.get('email')

```

```

        user = request.user
user.username = username
user.first_name = first_name
user.last_name = last_name
user.email = email      user.save()
        # Update voter information
        guest.First_Name = request.POST.get('first_name')
guest.Last_Name = request.POST.get('last_name')
guest.Email = request.POST.get('email')      guest.CNIC
= request.POST.get('cnicNum')      guest.Address =
request.POST.get('address')
        guest.Phone_Number      =      request.POST.get('phone')
guest.save()
        return redirect('dashboard') else:
        return render(request, "changeProfile.html", {'guest': guest})
def logoutview(request):  logout(request)  return
redirect('home') from django.contrib.auth.decorators
import login_required
@login_required def confirm_room(request):  if
request.method == 'POST':      room_number =
request.POST.get('room_number')      check_in_date =
request.POST.get('check_in_date')      check_out_date =
request.POST.get('check_out_date')      try:
        room = Room.objects.get(room_number=room_number)
        room.room_status      =      'booked'
room.save()

```

```

        # Check if a Guest object exists for the logged-in user
        guest, created = Guest.objects.get_or_create(pk=request.user.id)
        booking = Booking.objects.create(
            guest=guest,
            room=room,
            check_in_date=check_in_date,
            check_out_date=check_out_date
        )
        return redirect("book_service", booking_id=booking.id) #
        Redirect to billing page with booking_id          except
        Room.DoesNotExist:
            return HttpResponseRedirect("Room does not exist!")
    except Exception as e:
        return HttpResponseRedirect(f"Booking failed. Error: {e}")
    return HttpResponseRedirect("Invalid request")
def
book_service(request, booking_id):    if request.method
== 'GET':
        booking = get_object_or_404(Booking, pk=booking_id)
        available_services = Service.objects.all() # Fetch all available
services        return render(request, 'serviceSelection.html', {'booking':
booking,
'services': available_services})    elif request.method == 'POST':
selected_services = request.POST.getlist('selected_services')
try:
        booking = Booking.objects.get(pk=booking_id)
        services = Service.objects.filter(pk__in=selected_services)
        booking.services.set(services) # Set the selected services for the booking

```

```

        return redirect("billing_page", booking_id=booking.id)
except Booking.DoesNotExist:
    return
HttpResponse("Booking does not exist!")
except Exception
as e:
    return HttpResponse(f"Service booking failed. Error:
    {e}")
    return HttpResponse("Invalid request")
def
billing_page(request, booking_id):
    try:
        booking = get_object_or_404(Booking, pk=booking_id)
        total_amount = booking.calculate_total()
        if
        request.method == 'POST':
            try:
                # Create Billing instance (cash payment assumed)
                billing = Billing.objects.create(booking=booking,
                total_amount=total_amount, payment_method='Cash')
                return redirect('receipt', booking_id=booking.id)
            except
            Exception as e:
                return HttpResponse(f"Failed to process payment.
                Error: {e}")
                return render(request, 'billing.html', {'total_amount':
                total_amount, 'booking': booking})
            except Booking.DoesNotExist:
                return HttpResponse("Booking does not exist!")
            except
            Exception as e:
                return HttpResponse(f"Error: {e}")
    def
    receipt(request, booking_id):
        booking = get_object_or_404(Booking, pk=booking_id)
        guest =
        booking.guest # Assuming Booking model has a foreign key to Guest
        # Fetch the guest details
        guest_first_name
        = guest.FirstName
        guest_last_name =
        guest.LastName
        room_number =
        booking.room.room_number
        check_in_date

```

```

= booking.check_in_date    check_out_date =
booking.check_out_date
    total_amount = booking.calculate_total() # Assuming you have a
method to calculate the total
    # Get the HTML content for the receipt
    template = get_template('receipt_pdf.html') # Your HTML template for
receipt
    html = template.render({
        'guest_first_name': guest_first_name,
        'guest_last_name': guest_last_name,
        'room_number': room_number,
        'check_in_date': check_in_date,
        'check_out_date': check_out_date,
        'total_amount': total_amount,
    })
    # Create a PDF from the HTML content
    response = HttpResponse(content_type='application/pdf')
response['Content-Disposition'] = f'attachment;
filename="receipt_{booking_id}.pdf"'
    # Use xhtml2pdf to convert HTML to PDF pisa_status =
pisa.CreatePDF(html, dest=response) if pisa_status.err:    return
HttpResponse('We had some errors <pre>' + html + '</pre>')    return
response def download_receipt(request, billing_id):    try:
    # Retrieve the billing instance based on the provided ID
billing = Billing.objects.get(id=billing_id)
    # Load the template for the PDF (make sure the path is
correct)    template = get_template('homepage/billing_pdf.html')

```



```

html = template.render({'billing': billing})      # Create an HTTP
response to return the PDF      response =
HttpResponse(content_type='application/pdf')
response['Content-Disposition'] = f'attachment;
filename="receipt_{billing_id}.pdf"      # Generate the PDF from
HTML content      pisa_status = pisa.CreatePDF(html,
dest=response)      if pisa_status.err:
        return HttpResponse(f'We had some errors while generating the
PDF: {pisa_status.err}')      return response
except Billing.DoesNotExist:      return
HttpResponse("Billing not found.")      except
Exception as e:
        return HttpResponse(f'An error occurred: {e}')

def create_checkout_session(request, booking_id):
try:
        booking      =      Booking.objects.get(pk=booking_id)
total_amount = booking.calculate_total()

        # Create a new Checkout Session
session = stripe.checkout.Session.create(
payment_method_types=['card'],
        line_items=[
                {
                        'price_data': {
                                'currency': 'usd',
                                'product_data': {
                                        'name': 'Room and Services',

```

```

        },
        'unit_amount': int(total_amount * 100), # Convert to cents
    },
    'quantity': 1,
},
],
mode='payment',
success_url='http://127.0.0.1:8000/success',
cancel_url='http://127.0.0.1:8000/cancel',
)
return redirect(session.url)
except Booking.DoesNotExist:
    return HttpResponse("Booking does not exist!")
except Exception as e:
    return HttpResponse(f"Failed to create checkout session. Error: {e}")
def success(request):
    return render(request, 'success.html')
def cancel(request):
    return render(request, 'cancel.html')
def regComplete(request):
    return render(request, "regComplete.html")
def staff_viewer(request):
    # Fetch all staff members from the database
    all_staff = Staff.objects.all() # Replace 'Staff' with your actual model name

```

```

    # Pass the staff data to the template for rendering
    context = {
        'all_staff': all_staff,
    }
    return render(request, 'staff_viewer.html', context)
def feedback_view(request):
    if request.method == 'POST':
        form = FeedbackForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('feedback_thankyou')
        else:
            form = FeedbackForm()
    return render(request, 'feedback.html', {'form': form})
def feedback_thankyou(request):
    return render(request, 'thankyou.html', {'message': 'Thank you for your feedback!'})

```

Django settings.py

```

from pathlib import Path
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-ld7g_z@s01c4e*s5=jr&vw1=sfxrhc_9perd$!5%orkd=*%ze'
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

```

```

ALLOWED_HOSTS = []
# Application definition
INSTALLED_APPS = [
    'jazzmin',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'HomePage',
]
JAZZMIN_UI_TWEAKS={
    "theme":"darkly",
}
JAZZMIN_SETTINGS={
    "site_title":"HMS Admin",
    "site_brand":"HMS",
    "welcome_sign":"Welcome",
}
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',

```

```

'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
ROOT_URLCONF = 'HotelManagmentSystem.urls'
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
                # Existing context processors
            ],
        },
    ],
]
WSGI_APPLICATION = 'HotelManagmentSystem.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'hotel_db',
        'USER': 'root',
        'PASSWORD': 'Security',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}

```

```

    }
}
# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-
passwordvalidators
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValida
t or',    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'

```

```

USE_I18N = True
USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static'),
]
STATIC_ROOT = 'H:\HMS\HotelManagmentSystem\static_root'
# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
STRIPE_PUBLISHABLE_KEY =
'pk_test_51OLi3ALYWee0pOU1kWHMuhXAoNhW5gAARHBgBVO6
x9H5aYWwPC7WFQrlF3pC61SPYVL1AYVQyI1Kq0K01a7blhT200s
L6U4pXX'
STRIPE_SECRET_KEY =
'sk_test_51OLi3ALYWee0pOU1I27z39rheeMmzKMzIxcBrvJydYY6N
BU32OCE6x7EvuwKJmHYeBkidCWUSNqiMPSWn0Kp8njP00AWn3n
p7h'
# Email configuration for Gmail

```

[Django Urls.py](#)

```

from django.contrib import admin
from django.urls import path, include
from HomePage import views
from django.conf import settings

```

```
from django.conf.urls.static import static

urlpatterns = [    path('admin/',
admin.site.urls),    path("",
include('HomePage.urls')), ] +
static(settings.STATIC_URL,
document_root=settings.STATIC_ROOT)
```

➤ **ERROR HANDLING:**

Errors are indicators of issues within the project implementation. A project is considered successful only when it runs with minimal or no errors. Hence, identifying and addressing errors is a critical part of developing a Hotel Management System using Django. Errors encountered during the development and deployment of this project can be classified into the following categories:

Syntax Errors:

These errors occur when the developer violates the grammatical rules of Python or Django syntax. For instance, missing colons, incorrect indentation, or improper use of Python or Django-specific commands (e.g., incorrect model or view syntax).

Such errors are caught at the time of code writing or compilation.

Logical Errors:

Logical errors arise when the system executes without crashing but produces incorrect or unintended output.

These errors are not flagged by the Django framework or Python interpreter and can only be found through rigorous testing of application functionality. Thorough testing of views, forms, and templates is essential to uncover and fix logical flaws in the system.

Runtime Errors:

These are the most complex errors to detect as they occur only during the execution of the application. These could be caused by unexpected user input, server-side exceptions, or environmental issues such as database connection failures.

In Django, runtime errors can include issues like:

- Form validation failures due to missing or incorrect user input.
- Server errors (HTTP 500) when unhandled exceptions occur in views.

➤ **VALIDATION CHECKS**

Software testing is a vital part of developing a robust Hotel Management System. It often falls under the broader concept of **Verification and Validation (V & V)**.

- **Verification** ensures that the system is built according to the specifications and performs the intended functions correctly.
- **Validation** ensures the product meets the business needs and customer requirements.

Boehm's states this another way:

- **Verification:** "Are we building the product right?"
- **Validation:** "Are we building the right product?"

The concept of V & V includes several practices that contribute to Software Quality Assurance (SQA), which is integral to any Hotel Management System.

Validation in Django Forms

Validation is commonly handled in Django using **forms** and **model field validators**. In this project, we use the following types of validation:

- Empty Field Validation
- Numeric Field Validation



TESTING STRATEGY

Testing is an essential phase in the development of the **Hotel Management System using Django**, where the primary goal is to ensure that each module—such as room booking, guest management, payment handling, and admin panel—works as intended.

Testing provides a clear, unbiased view of the system's reliability and performance and helps verify that the software meets business requirements and user expectations.

The objective is to:

1. Ensure the system meets hotel business and technical requirements.
2. Confirm all modules perform as expected.
3. Guarantee consistent and reproducible performance across similar use cases.

Software testing methods applied during the Django project development include:

Unit Testing

Unit testing is used to test individual modules in isolation—like the `booking()`, `check_availability()`, and `calculate_total()` functions in the backend views.

- Each function is tested to ensure it returns the correct response, handles exceptions, and validates user input properly.
- Django's built-in Test Case class is used for writing these tests.
- This ensures each core logic block, like calculating room rates or checking availability, works independently.

Example:

Test that the booking function rejects a request when a room is already occupied during the requested date range.

Integration Testing

Integration testing validates how modules interact—like how the booking page communicates with the database and payment gateway.

- Modules like the frontend room booking form, backend processing logic, and database models are integrated and tested.
- This ensures that data flows correctly from form input through Django views to the database.
- Django's testing framework and tools like `pytest-django` or Selenium (for UI flows) are used.

Example:

Test the interaction between the user room booking form, room model, and payment module.

System Testing

System testing checks the entire hotel management system to ensure it behaves correctly as a whole.

- This includes testing all features end-to-end: user login, room search, booking confirmation, and payment generation.
- Tests are conducted to ensure outputs match expected outcomes from real user scenarios.

Example:

Create a test user, simulate a complete room booking from login to confirmation, and verify system responses at each step.



In the **Hotel Management System developed using Django**, system security is a crucial aspect to ensure that sensitive data—such as customer information, booking details, and payment transactions—is protected from unauthorized access.

The system ensures that only authenticated users (e.g., hotel staff or registered customers) can access specific features. Users must create an account and log in with valid credentials to access any personalized functionality such as booking a room or viewing reservations. This authentication layer adds a strong level of security to the platform.

All user data handled through the system is stored securely in the database, using Django's built-in security mechanisms. Data transmissions can be encrypted using HTTPS to prevent data interception or tampering.

Security Compliance (Adapted from ISO 74982):

Authentication

The system verifies each user's identity before granting access to protected features. This is implemented using Django's built-in authentication system, which uses sessions, hashed passwords, and user login management.

Access Control

Role-based access is implemented:

- **Admin users** can manage rooms, view all bookings, and handle payments.
- **Regular users** can only view and manage their own bookings. This ensures users only interact with the data and features relevant to them.

Confidentiality

Data such as guest personal information and booking details is encrypted or protected using Django's ORM and middleware. Only authorized users can access or modify this data.

Data Integrity

Data entered by users—like booking dates, payment records, and guest details—is validated at multiple levels. Django forms and model validations prevent invalid data from being stored, while the admin panel allows staff to manage and update entries securely.

Non-repudiation

- The system keeps a log of actions by specific users (audit trails can be added).
- Security policies clearly define who has access to sensitive actions like booking cancellation, room modification, or payment processing.

Security Measures in Place

The Django-based Hotel Management System includes:

- Authentication
- Access Control
- Confidentiality
- Data Integrity □ User Role Management



Guest's Page:

Guests Home > Homepage > Guests					+ Add guest
<div><div>----- ▾</div><div>Go 0 of 7 selected</div></div>					
<input type="checkbox"/>	GuestID	FirstName	LastName	Email	CNIC
<input type="checkbox"/>	20	-	-	-	-
<input type="checkbox"/>	15	Swaraj	Sharma	swaraj@gmail.com	5647890321897
<input type="checkbox"/>	14	Rakesh	Patil	rakesh@gmail.com	5647890321234
<input type="checkbox"/>	13	Asmita	Pande	Asmita	4080976543212
<input type="checkbox"/>	12	-	-	-	-
<input type="checkbox"/>	5	-	-	-	-
<input type="checkbox"/>	1	Deepika	Gore	deepika@gmail.com	4080976543212
7 guests					
Copyright © 2025 - All rights reserved					lazzmin version 0.0.0

Booking Page:

Bookings Home > Homepage > Bookings					+ Add booking
<div><div>----- ▾</div><div>Go 0 of 8 selected</div></div>					
<input type="checkbox"/>	Guest	Room	Check in date	Check out date	
<input type="checkbox"/>	Guest object (20)	Room 102 - booked	May 15, 2025	May 17, 2025	
<input type="checkbox"/>	Guest object (13)	Room 101 - available	May 16, 2025	May 17, 2025	
<input type="checkbox"/>	Guest object (13)	Room 103 - available	May 7, 2025	May 10, 2025	
<input type="checkbox"/>	Guest object (14)	Room 102 - booked	May 15, 2025	May 16, 2025	
<input type="checkbox"/>	Guest object (14)	Room 101 - available	May 14, 2025	May 15, 2025	
<input type="checkbox"/>	Guest object (13)	Room 102 - booked	May 7, 2025	May 9, 2025	
<input type="checkbox"/>	Guest object (12)	Room 103 - available	May 7, 2025	May 9, 2025	
<input type="checkbox"/>	Guest object (5)	Room 102 - booked	May 5, 2025	May 6, 2025	
8 bookings					



FUTUR SCOPE OF THE PROJECT

The **Hotel Management System** developed using the **Django web framework** is designed to streamline and automate the day-to-day operations of a hotel. The system offers a secure user authentication module, allowing users to register, log in, and access features based on their roles—such as Admin, Staff, or Guest. One of the key features is the **Room Management** module, where admins can add, update, or delete room details, as well as manage their availability and status. Guests can view and book available rooms in real-time through the **Online Booking System**, with their booking history and status tracked through their account.

The system also includes a **Customer Management** feature that stores and manages customer profiles and identification details. Check-in and check-out operations are efficiently handled, with automatic updates to room availability. A built-in **Billing and Invoicing** system calculates charges based on booking details and generates downloadable invoices. An optional **Payment Integration** module can be implemented to allow online payments. The **Admin Dashboard** provides an overview of hotel operations, including statistics on room availability, bookings, and revenue.

Optional Enhancements

- Integration with third-party APIs (e.g., Google Maps, payment gateways).
- Advanced analytics and reports for hotel performance.

- Mobile app version using Django REST API.
- Loyalty or reward point system for frequent guests.
- Multi-language support.



1. Django Software Foundation. (n.d.). *Django Documentation*. Retrieved from <https://docs.djangoproject.com>
2. Python Software Foundation. (n.d.). *Official Python Documentation*. Retrieved from <https://docs.python.org>
3. W3Schools. (n.d.). *HTML, CSS, and JavaScript Tutorials*. Retrieved from <https://www.w3schools.com>
4. Geeks for Geeks. (n.d.). *Django Tutorials and Examples*. Retrieved from <https://www.geeksforgeeks.org>
5. TutorialsPoint. (n.d.). *Django and Python Programming Guides*. Retrieved from <https://www.tutorialspoint.com>
6. Stack Overflow. (n.d.). *Community Discussions and Developer Solutions*. Retrieved from <https://stackoverflow.com>
7. MDN Web Docs. (n.d.). *HTML and CSS Reference*. Retrieved from <https://developer.mozilla.org>

8. MySQL Documentation. (n.d.). *MySQL 8.0 Reference Manual*. Retrieved from <https://dev.mysql.com/doc>
9. Real Python. (n.d.). *Practical Django Projects and Python Tips*. Retrieved from <https://realpython.com>