****

**COLLEGE OF VOCATIONAL STUDIES**

(University of delhi)

**Hotel Management System**

*(Software engineering project report)*

Group description

**Avika singh 22013570010**

**Shristy 22013570025**

**Riddhi luthra 22013570053**

**Anuradha 22013570036**

Under the supervision of : **Mrs. Parul Chachra**

Date – 26-11-2024

**ACKNOWLEDGEMENT**

On the successful completion of our project HOTEL MANAGEMENT SERVICE SYSTEM we would like to express our sincere gratitude to everyone who helped us in the completion of this project.

We are sincerely thankful to our project guide Mrs. Parul Chachra, for her interest, guidance and suggestions throughout the course of the project. We feel honored and privileged to work under her. She shared her vast pool of knowledge with us that helped us steer through all the difficulties with ease. This project would not have been possible without her guidance.

**CERTIFICATE**

This is to certify that the project titled "HOTEL MANAGEMENT SERVICE SYSTEM " is submitted by AVIKA SINGH , RIDDHI LUTHRA, SHRISTY and ANURADHA under the supervision of Mrs. Parul Chachra in the academic year 2024-2025.

This project is submitted for paper Software Engineering of BSc (Hons.) Computer Science, Semester V.

Supervisor

**Mrs. Parul Chachra**

(Assistant Professor)

(Department: Computer Science)

**TABLE OF CONTENTS**

**1. Introduction**

1.1 Problem Statement

1.2 Objectives

1.3 Intended Audience and Reading Suggestions

1.4 Process Model :Iterative Development

**2. Overall Description**

2.1 Product Perspective

2.2 Product Functions

2.3 User Characteristics

2.4 General Constraints

2.5 Assumptions and Dependencies

2.6 Operating Environment

**3. Data Framework**

3.1 Data Dictionary

3.2 Definitions, Acronyms, and Abbreviations

**4. Requirement Analysis**

4.1 Functional Requirements

4.2 Non-Functional Requirements

4.3 External Interface Requirements

**5.Risk Management**

**6.Project Management**

6.1 Timeline (Gantt Chart)

6.2 Function Point (FP) Calculation

6.3 Effort Estimation

6.4 Cost Estimation

**7 Design Engineering**

7.1 Architectural Design

7.2 Component Diagram

7.3 Benefits of MVC and Modular Design

**8.Coding**

8.1 Pseudocode for a Module

8.2 Code of a Module in PYTHON

**9. Testing**

9.1. Compute Basis Path for the Room Booking Module

9.2. Generate Test Cases

9.2.1 Using white box

9.2.2 Using black box

**10.Analysis Models**

10.1 Data Flow Diagram (DFD)

10.2 Database Design

10.3 Use case diagram

**11. References**

**12. Conclusion**

**1. Introduction**

**1.1 Problem Statement-**

The ‘Hotel Management System (HMS)’ is designed to streamline hotel operations by automating room bookings, customer management, and room tracking processes. The system addresses inefficiencies in manual processes, ensuring faster, error-free handling of bookings and customer details. Key functionalities include:

* Simplified room booking based on real-time availability.
* Management of customer details such as contact information and stay preferences.
* Automated calculation of room charges, taxes, and total bills.
* Generation of detailed invoices for customers.

**1.2 Objectives**

* Provide a robust, user-friendly platform for hotel staff to manage daily operations efficiently.
* Enable customers to experience seamless room booking and billing.
* Ensure accurate data handling and minimize errors in bookings.
* Create a scalable and maintainable system for future feature integration.

**1.3 Intended Audience and Reading Suggestions**

This document is intended for:

Project Managers: To understand the overall design and architecture of the HMS Developers: To comprehend the technical details of the system, including database management, API usage, and system modules.

Testers: To gain insight into the test cases and system behavior to ensure quality and reliability.

End Users: To provide guidance on how to interact with the system, particularly those responsible for daily operations like customer management, room bookings , generating invoices. It is suggested that readers first familiarize themselves with the System Overview and then proceed to the more technical sections like System Features and External Interface Requirements for deeper insights into the HMS.

**1.4 Process Model: Iterative Development**

The ITERATIVE MODEL was selected for its flexibility and feedback-driven approach. Each iteration results in a functional version of the system, which undergoes evaluation and refinement.

Benefits include:

* EARLY FEEDBACK : Stakeholders can provide input at each iteration, ensuring alignment with requirements.
* REDUCED RISKS : Issues are identified and resolved early in the development cycle.
* CONTINUOUS IMPROVEMENT: The system evolves incrementally, with functionality and usability enhancements in each cycle.

**2. Overall Description**

**2.1Product Perspective**

* The HMS replaces traditional manual systems with a modern, automated solution. It is a standalone system integrated with a central database to manage customer data, room information, and transaction records. The system uses a web-based or desktop interface for users to interact with the functionalities.

The system consists of the following key components:

* Customer Management: Handles customer details and booking preferences.
* Room Availability Check: Ensures rooms are booked only if available.
* Booking and Payment Processing: Calculates charges and generates invoices.
* Admin Operations: Allows staff to update room statuses and view reports.

**2.2 Product Functions**

The HMS provides the following functionalities:

Customer Management:

* Collect and validate customer details.
* Store customer preferences and booking history.

Room Booking:

* Allow customers to book rooms based on real-time availability.
* Confirm bookings and generate invoices.

Admin Operations:

* Update room availability (e.g., maintenance or booking status).
* Generate and view reports on booking trends and revenues.

Billing:

* Calculate total charges, including taxes.
* Generate and store invoices for customer records.

**2.3 User Characteristics**

The system is designed for the following types of users:

Customers:

* Individuals booking rooms for their stay(via admin) .
* Minimal technical knowledge required to interact with the system.

Admins:

* Hotel staff responsible for managing bookings and room statuses.
* Requires basic knowledge of hotel operations and system usage.

**2.4 General Constraints**

* The system must operate within the constraints of the hotel’s existing IT infrastructure.
* Data processing must comply with data protection regulations such as GDPR.
* The interface should be optimized for desktop systems and compatible with modern browsers.
* The system should support up to 500 simultaneous users without performance degradation.

**2.5 Assumptions and Dependencies**

* The system assumes accurate initial data input regarding room availability and pricing.
* The hotel will provide a reliable network infrastructure to support the system.
* Integration with third-party payment gateways may be required in future versions.

**2.6 Operating Environment**

* Software: The HMS will run on a modern web browser (e.g., Chrome, Firefox) with a responsive design for desktop and mobile compatibility.
* Server: The system can run on any standard web server capable of supporting Python web frameworks such as Flask or Django.
* Database: The system uses MySQL for managing data related to customers, rooms, employees and invoice.
* Platform: The system is accessible through web-based interfaces on Windows,macOS , and Linux platforms

**3.Data Framework**

**3.1 Data Dictionary –**

The data dictionary is a centralized repository of information about data. It provides a detailed description of the data, including its meaning, relationship to other data, usage, and format.

|  |  |  |
| --- | --- | --- |
| **Sno** | **Term** | **Description** |
| 1 | Customer details | Includes name, gender, phone number, and email , address. |
| 2 | Booking Request | Room type, check-in/check-out dates. |
| 3 | Room Data Store | Stores room details, availability, pricing |
| 4 | Invoice details | Includes room charges, taxes and total bill amount. |
| 5 | Generated invoice | Save the booking invoice for future generation. |

**3.2 Definitions, Acronyms, and Abbreviations**

|  |  |  |
| --- | --- | --- |
| **Sno** | **Term** | **Description** |
| 1 | HMS | Hotel Management System |
| 2 | Admin | Hotel staff responsible for managing bookings and rooms. |
| 3 | Invoice | A detailed bill containing room charges, taxes, and fees |
| 4 | Booking request | A customer’s request to reserve a room |
| 5 | Availability check | A process to verify if a room is available for booking |

**4. Requirement Analysis -**

**4.1 Functional Requirements**

Functional requirements are the desired operations of a software program or system, and they define how the system must respond to inputs. They are a key part of requirements analysis, which is the engineering field that deals with the design and maintenance of complex systems.

Customer Management:

* The system shall allow customers to enter personal details and booking requests.
* The system shall validate customer input for accuracy.

Room Availability Check:

* The system shall query the database for room availability based on user input.
* The system shall display feedback on room availability status.

Booking and Payment Processing:

* The system shall calculate charges based on room type, stay duration, and taxes.
* The system shall generate a detailed invoice upon booking confirmation.
* The system shall update the room status in the database after successful booking.

Admin Operations:

* The system shall allow admins to update room statuses (e.g., available, booked, under maintenance).
* The system shall provide reports on booking trends and revenue generation.

**4.2 Non-Functional Requirements**

Non-functional requirements (NFRs) are specifications that describe how a system should operate, focusing on quality attributes like performance, security, reliability, and scalability.

Performance:

* The system shall respond to user actions within 2 seconds under normal conditions.
* The system shall support up to 500 concurrent users.

Security:

* The system shall encrypt sensitive customer data, including personal and payment information.
* Admin operations shall require secure login credentials.
* Only authorized admins should be able to perform actions like deleting records or adding new rooms.

Scalability:

* The system shall be scalable to accommodate new features such as online payments and loyalty programs.

Usability:

* The interface shall be intuitive, with clear navigation and error handling.
* All text shall be displayed in English for international use.

Portability:

* The system should be compatible with the most commonly used browsers (Chrome, Firefox, Edge) and responsive on both desktop and mobile Devices.

**4.3 External Interface Requirements**

An “external interface requirement” refers to the specifications defining how a software system will interact with elements outside of itself, such as other systems, hardware devices, or users, including details like data formats, communication protocols, and user interface elements that are visible to the end-user.

User Interface:

* A web-based GUI for customers and admins to interact with the system.

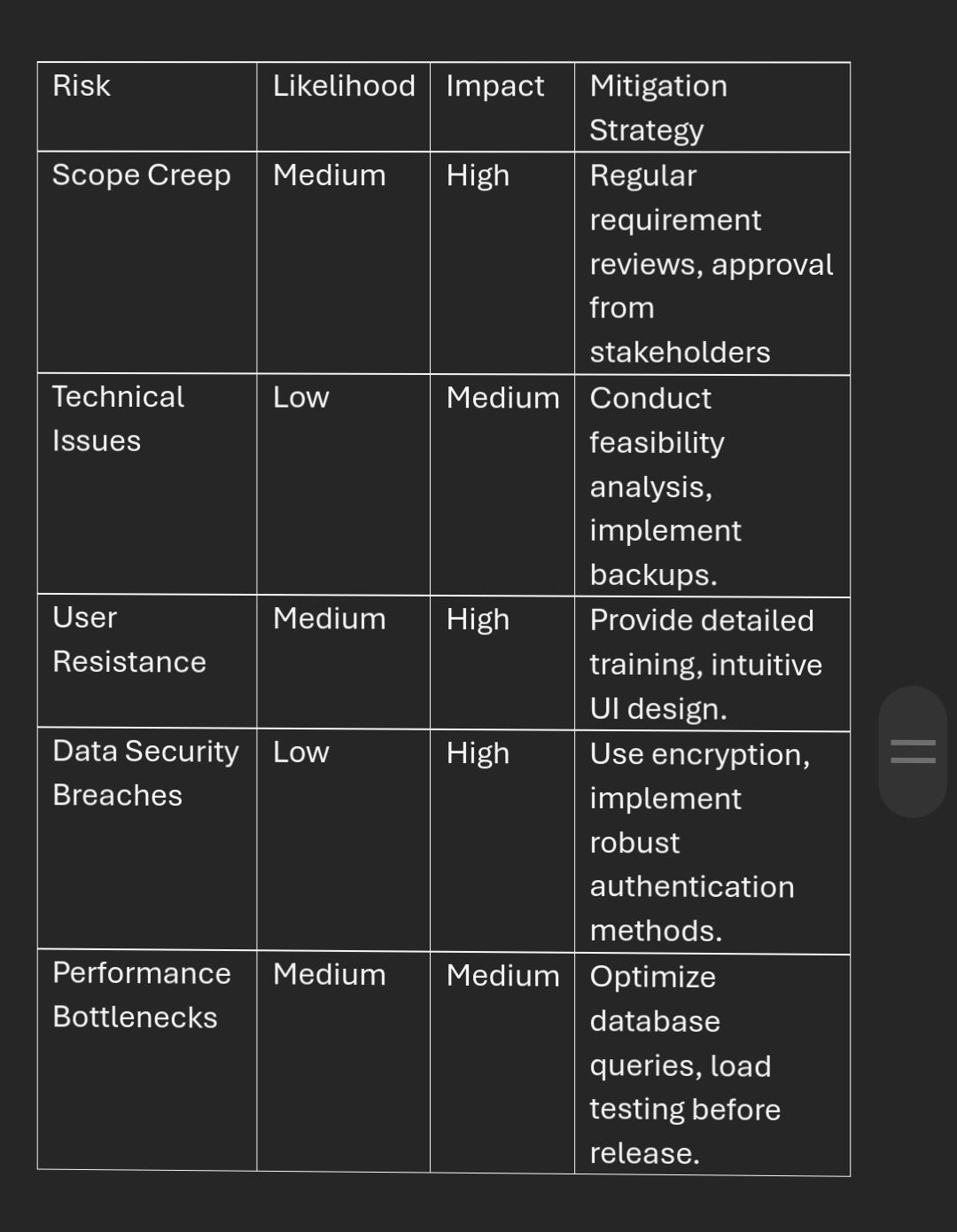
Hardware Interface:

* Compatible with standard desktop computers and printers for invoice generation.

Software Interface:

* Integrated with a central database for storing and retrieving data.

**5.Risk Management**

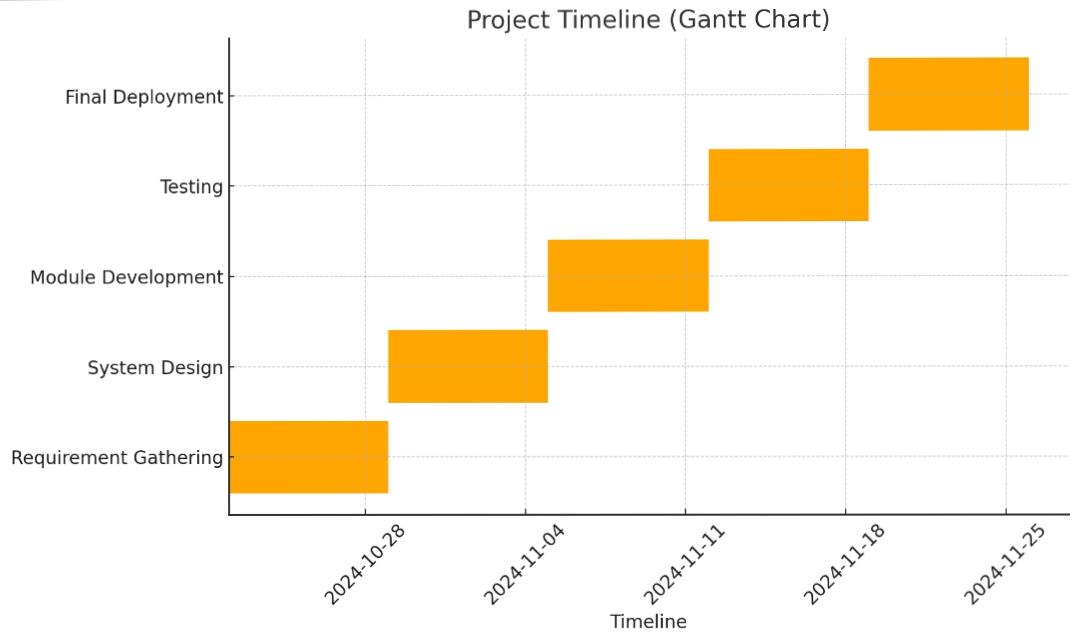
****

**6.Project Management**

**6.1 Timeline (Gantt Chart)-**

A popular type of timeline chart that shows a project’s schedule using a horizontal bar chart. Each task is represented by a bar that runs from the start to the end date of the task. Gantt charts are useful for managing complex projects with dependencies.

|  |  |
| --- | --- |
| Phase | Activities |
| Requirement Gathering | Identify stakeholders, gather system requirements, create SRS. |
| System Design | Design DFDs, ER diagrams, and finalize architecture. |
| Development | Implement features: Room booking, billing, admin panel. |
| Testing | Conduct functional, integration, and user acceptance testing. |
| Deployment and Handover | Deploy system, train staff, and finalize documentation. |



**6.2 Function Point (FP) Calculation-**

FP calculation uses the Function Point Analysis (FPA) method based on complexity weightings for different system components.

*Step 1: Assign Components –*

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Type | Count | Weight (Simple, Avg, Complex) |
| External Inputs (EI) | Customer booking details | 5 | Simple (3) |
| External Outputs (EO) | Invoice generation | 3 | Average (4) |
| External Inquiries (EQ) | Room availability checks | 5 | Simple (3) |
| Internal Logical Files (ILF) | Customer and room data | 4 | Average (10) |
| External Interface Files (EIF) | Existing hotel data | 2 | Simple (7) |

*Step 2: Compute Unadjusted Function Points (UFP)-*

Using the formula:

UFP=∑(count×weight)

UFP=∑(count×weight)

UFP=(5×3)+(3×4)+(5×3)+(4×10)+(2×7)

UFP=15+12+15+40+14=96

*Step 3: Adjusted Function Points (AFP)-*

Apply the Complexity Adjustment Factor (CAF). Assume an average system complexity with a CAF of 1.2:

AFP=UFP×CAF

AFP=96×1.2=115.2

Final FP: 115 (rounded)

**6.3 Effort Estimation-**

Effort estimation is based on the relationship:

Effort (Person-Months)=FP×Productivity (Hours/FP)

Assumptions:

* + Average productivity = 10 hours/FP.
  + Team works 160 hours/month.

Effort=(FP×Hours/FP)/(Hours/Month)

Effort=115×10/160

=7.1875 Person-Months (approx. 7 months)

Example:

If we have a team of 3 developers, the actual time required is:

7.18/3=2.39 months (approximately 10 weeks).

**6.4 Cost Estimation-**

Cost estimation uses the formula:

Cost=Effort (Person-Months)×Cost per Month

Assumptions:

* Average developer salary = $4,000/month.
* Cost=7.18×4,000=$28,720

Example:

For a team of 3 developers:

Cost=Developer Salary×Duration per Developer

Cost=(3×4,000)×2.39=$28,680

**7 Design Engineering**

**7.1 Architectural Design**

*System Architecture: MVC (Model-View-Controller)*

The 'Model-View-Controller (MVC)' pattern is used to organize the system into three interconnected layers for scalability and maintainability:

Model:

* Handles business logic, calculations, and database operations.
* Manages data related to rooms, bookings, and customers.

Example:

* Storing room details (availability, type, rate).
* Storing customer and booking information.

View:

* Represents the user interface (UI).
* Displays booking forms, availability status, and invoices.
* Ensures user-friendly interaction for both admins and customers.

Controller:

* Processes user inputs from the View and interacts with the Model.
* Ensures data flow between the UI and backend logic.

Example:

* Handling booking requests and forwarding them to the Model.
* Triggering invoice generation.

**7.2 Component Diagram-**

|  |  |
| --- | --- |
| **Component** | **Description** |
| Login System | Handles user authentication (Admin). |
| Booking Management | Handles room availability checks and booking creation. |
| Billing Module | Calculates total cost (room charge, taxes, extras). |
| Room Management | Admin updates room details (availability, rates). |
| Database | Stores customer, booking, and room details securely. |

**7.3 Benefits of MVC and Modular Design**

*1. Scalability:*

- Each layer (Model, View, Controller) can be updated independently.

- Easier integration of new features, like online payments.

*2. Maintainability:*

- Modular code reduces complexity.

- Clear separation of concerns ensures efficient debugging and updates.

*3. Reusability:*

- Components like `RoomBooking` can be reused for other hotel-related features.

**8.Coding**

**8.1 Pseudocode for a Module-**

Below is the pseudocode for a Room Booking Module, based on the previously provided structure and functions. The goal is to automate the booking process, validate availability, and calculate costs.

|  |
| --- |
| class RoomBooking:  def \_\_init\_\_(self):  self.rooms = {"Single": 5, "Double": 3, "Luxury": 2} # Sample room data  self.room\_rates = {"Single": 100, "Double": 150, "Luxury": 250} # Room rates  self.tax\_rate = 0.10 # 10% tax  def check\_availability(self, room\_type):  if room\_type in self.rooms and self.rooms[room\_type] > 0:  return True  return False  def calculate\_cost(self, room\_type, num\_days):  if room\_type in self.room\_rates:  subtotal = self.room\_rates[room\_type] \* num\_days  tax = subtotal \* self.tax\_rate  total\_cost = subtotal + tax  return {"subtotal": subtotal, "tax": tax, "total\_cost": total\_cost}  return None  def book\_room(self, customer\_name, room\_type, num\_days):  if self.check\_availability(room\_type):  self.rooms[room\_type] -= 1 # Decrease available room count  bill = self.calculate\_cost(room\_type, num\_days)  print(f"Booking Successful for {customer\_name}!")  print(f"Room Type: {room\_type}")  print(f"Number of Days: {num\_days}")  print(f"Subtotal: ${bill['subtotal']}")  print(f"Tax: ${bill['tax']}")  print(f"Total Cost: ${bill['total\_cost']}")  else:  print(f"Sorry, {room\_type} rooms are unavailable!")  # Example usage  booking\_system = RoomBooking()  booking\_system.book\_room("Alice", "Luxury", 3) |

**8.2 Code of a Module in PYTHON**

The Room Booking System is implemented in Python using the Tkinter library for the graphical user interface (GUI).This implementation aligns with the core functionality described, including room availability checks, booking management, and invoice generation.

|  |
| --- |
| class RoomBookingSystem:  def \_\_init\_\_(self):  self.rooms = {  "Single": {"available": 10, "price": 2000},  "Double": {"available": 5, "price": 3500},  "Luxury": {"available": 3, "price": 8000},  }  self.tax\_rate = 0.12  self.bookings = []  def check\_availability(self, room\_type, num\_rooms):  if room\_type in self.rooms and self.rooms[room\_type]["available"] >= num\_rooms:  return True  return False  def calculate\_cost(self, room\_type, num\_rooms, num\_days):  room\_price = self.rooms[room\_type]["price"]  subtotal = room\_price \* num\_rooms \* num\_days  tax = subtotal \* self.tax\_rate  total\_cost = subtotal + tax  return {"subtotal": subtotal, "tax": tax, "total\_cost": total\_cost}  def book\_room(self, customer\_name, room\_type, num\_rooms, num\_days):  if not self.check\_availability(room\_type, num\_rooms):  return f"Sorry, {room\_type} rooms are unavailable for your request."  self.rooms[room\_type]["available"] -= num\_rooms  cost\_details = self.calculate\_cost(room\_type, num\_rooms, num\_days)  booking = {  "customer\_name": customer\_name,  "room\_type": room\_type,  "num\_rooms": num\_rooms,  "num\_days": num\_days,  "cost\_details": cost\_details,  }  self.bookings.append(booking)  confirmation\_message = (  f"Booking Successful for {customer\_name}!\n"  f"Room Type: {room\_type}\n"  f"Number of Rooms: {num\_rooms}\n"  f"Number of Days: {num\_days}\n"  f"Subtotal: ₹{cost\_details['subtotal']:.2f}\n"  f"Tax (12%): ₹{cost\_details['tax']:.2f}\n"  f"Total Cost: ₹{cost\_details['total\_cost']:.2f}\n"  )  return confirmation\_message  def get\_booking\_summary(self):  if not self.bookings:  return "No bookings have been made yet."  summary = "Booking Summary:\n"  for idx, booking in enumerate(self.bookings, 1):  summary += (  f"{idx}. {booking['customer\_name']} - {booking['room\_type']} - "  f"{booking['num\_rooms']} room(s) for {booking['num\_days']} days.\n"  )  return summary  if \_\_name\_\_ == "\_\_main\_\_":  booking\_system = RoomBookingSystem()  print(booking\_system.book\_room("Avika", "Luxury", 1, 3))  print(booking\_system.book\_room("Riddhi", "Single", 2, 5))  print(booking\_system.book\_room("Anu", "Double", 6, 2))  print("\n" + booking\_system.get\_booking\_summary()) |

Explanation of the Code

*1. Class Definition:*

- `RoomBookingSystem` encapsulates all functionalities related to room booking, including inventory management, cost calculation, and booking storage.

*2. Key Methods:*

- `check\_availability`: Checks if the requested room type and number are available.

- `calculate\_cost`: Computes the total cost, including room charges and taxes.

- `book\_room`: Handles booking, updates inventory, calculates costs, and generates confirmation messages.

- `get\_booking\_summary`: Retrieves a summary of all bookings for review.

*3. Room Inventory:*

- Maintains room availability and prices using a dictionary structure.

- Updates room availability dynamically upon successful bookings.

*4. Example Usage:*

- Demonstrates successful and failed bookings based on availability.

- Outputs a detailed confirmation message for successful bookings.

*5. Scalability:*

- The code is modular and can be extended to include additional features, such as:

- Online payments.

- Customer authentication.

- Integration with a database.

**Sample Output**

*Case 1: Successful Booking*

|  |
| --- |
| Booking Successful for Avika!  Room Type: Luxury  Number of Rooms: 1  Number of Days: 3  Subtotal: ₹24000.00  Tax (12%): ₹2880.00  Total Cost: ₹26880.00 |

|  |
| --- |
| Booking Successful for Riddhi!  Room Type: Single  Number of Rooms: 2  Number of Days: 5  Subtotal: ₹20000.00  Tax (12%): ₹2400.00  Total Cost: ₹22400.00 |

*Case 2: Room Unavailable*

|  |
| --- |
| Sorry, Double rooms are unavailable for your request. |

*Booking Summary*

|  |
| --- |
| Booking Summary:  1. Avika - Luxury - 1 room(s) for 3 days.  2. Riddhi - Single - 2 room(s) for 5 days. |

**9. Testing**

**9.1. Compute Basis Path for the Room Booking Module**

*Control Flow Graph (CFG) Representation of `book\_room` Method:*

1. Start.

2. Input customer details, room type, number of rooms, and number of days.

3. Check room availability (`check\_availability` method).

*Decision 1* : If room is available, continue. Else, output “Room Unavailable” and terminate.

4. Calculate room cost (`calculate\_cost` method).

5. Update room availability.

6. Generate invoice and confirmation message

7. End.

*Independent Paths:*

Using Cyclomatic Complexity (V(G)), the number of independent paths is calculated as:

V(G) = E – N + 2P

= Number of edges = 7,

= Number of nodes = 7,

= Number of connected components = 1.

V(G) = 7 – 7 + 2 = 2

Basis Paths:

1. Path 1: 1 → 2 → 3 → 4 → 5 → 6 → 7 (Successful booking).

2. Path 2: 1 → 2 → 3 → 7 (Room unavailable).

**9.2. Generate Test Cases**

*9.2.1 Using white box*

White box testing is a software testing method that involves examining a software system’s internal structure, code, and logic. Testers have complete access to the application’s design documents and source code, allowing them to understand how the software works and how it achieves its results.

***Test Case 1: Successful Booking Path (Path 1)***

Input:

Customer Name: “Avika”.

Room Type: “Luxury”.

Number of Rooms: 1.

Number of Days: 3.

Execution:

1. System checks availability (check\_availability returns True).

2. Calculates total cost (₹8000/room \* 3 days + 12% tax).

3. Updates inventory.

4. Generates booking confirmation.

Expected Output:

Booking Successful for Avika!

Room Type: Luxury

Number of Rooms: 1

Number of Days: 3

Subtotal: ₹24000.00

Tax (12%): ₹2880.00

Total Cost: ₹26880.00

***Test Case 2: Room Unavailability Path (Path 2)***

Input:

Customer Name: “Riddhi”.

Room Type: “Double”.

Number of Rooms: 6.

Number of Days: 2.

Execution:

1. System checks availability (check\_availability returns False).

2. Booking process terminates with an unavailability message.

Expected Output:

Sorry, Double rooms are unavailable for your request.

***Test Case 3: Invalid Input (Boundary Testing)***

Input:

Customer Name: “Anu”.

Room Type: “Single”.

Number of Rooms: 0 (invalid input).

Number of Days: 3.

Execution:

1. Input validation identifies an invalid room count.

2. Process terminates with an error message.

Expected Output:

Error: Invalid input for number of rooms. Please enter a positive value.

***Test Case 4: Extreme Values***

Input:

Customer Name: “Riddhi”.

Room Type: “Luxury”.

Number of Rooms: 3 (maximum available).

Number of Days: 365 (long stay).

Execution:

1. System checks availability (check\_availability returns True).

2. Calculates total cost for an extended stay.

3. Updates inventory.

4. Generates confirmation message.

Expected Output:

Booking Successful for Riddhi!

Room Type: Luxury

Number of Rooms: 3

Number of Days: 365

Subtotal: ₹8760000.00

Tax (12%): ₹1051200.00

Total Cost: ₹9811200.00

*9.2.2 Using black box*

Black box testing is a software testing technique that assesses a system’s functionality without knowing how it’s coded or designed. Testers provide inputs and observe the system’s outputs to evaluate how it responds to user actions, its performance, and other aspects.

***Test Case 1: Valid Room Booking***

Input:

Room Type: “Single”.

Number of Rooms: 2.

Check-in Date: “2024-12-01”.

Check-out Date: “2024-12-05”.

Expected Behavior:

1. System verifies availability for two single rooms for 4 nights.

2. Calculates the total cost (₹2000/room \* 4 nights + 12% tax).

3. Generates a confirmation message.

Expected Output:

Booking Successful!

Room Type: Single

Number of Rooms: 2

Total Cost: ₹17920.00

***Test Case 2: Booking Conflict***

Input:

Room Type: "Luxury".

Number of Rooms: 1.

Check-in Date: "2024-12-03".

Check-out Date: "2024-12-08".

Expected Behavior:

1. System checks availability but finds the room booked for overlapping dates.

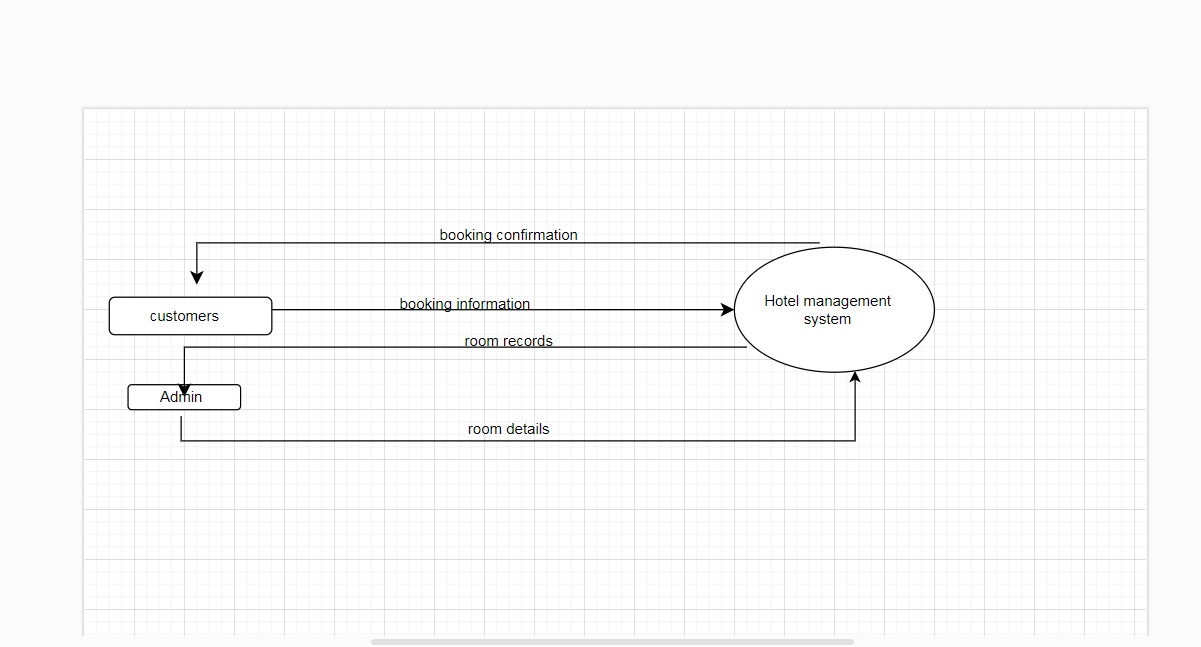
2. Returns a conflict message: "Selected room type is unavailable for the given dates."

**10.Analysis Models**

**10.1 Data Flow Diagram (DFD)**

* Level 0 DFD: Context Diagram

The Level 0 DFD represents the overall process as a single system, interacting with external entities. It showcases the primary external data sources and sinks for the system.

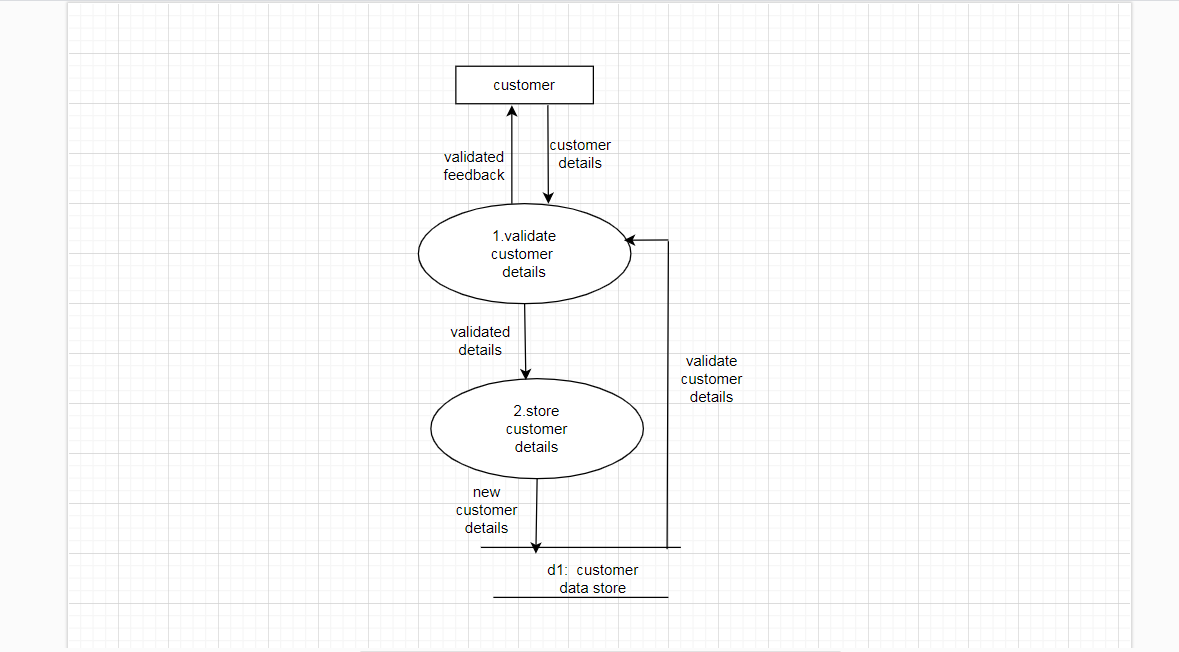


* Level 1 DFD –

In the Level 1 DFD , the single "Hotel Management System" process is expanded into multiple numbered subprocesses to illustrate its core functionalities.

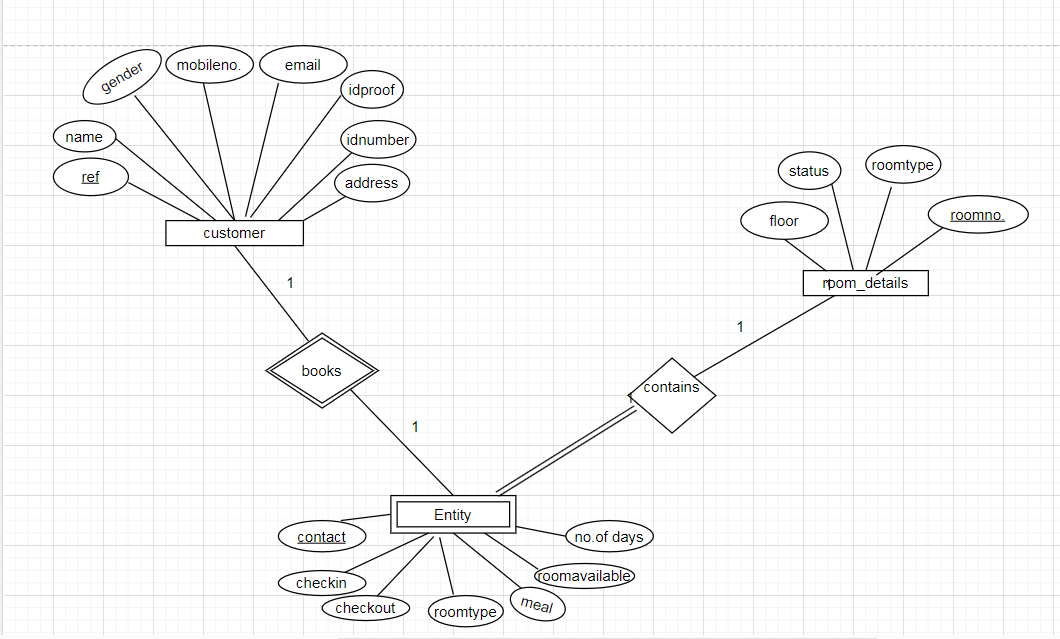
* Level 2 DFD

The Level 2 DFD breaks down individual processes from Level 1 into finer subprocesses. For simplicity, we’ll focus on Processes 1.

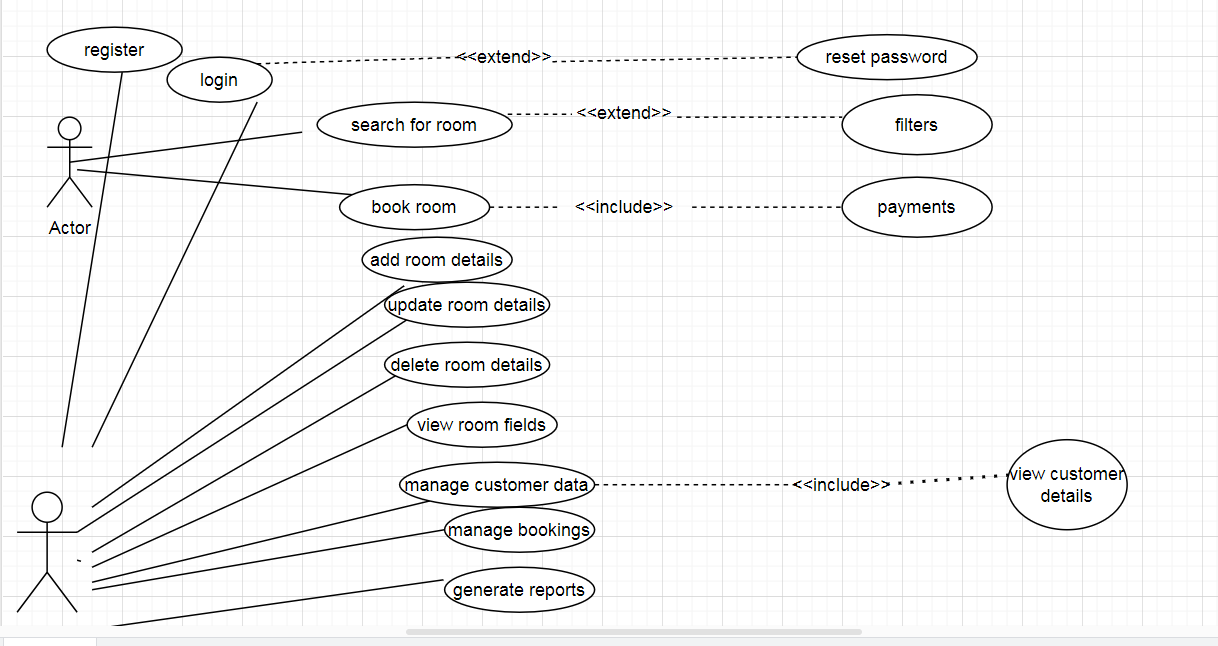


**10.2 Database Design**

ER Diagram Entities:



**10.3 Use case diagram**

****

* use-case diagrams model the behavior of a system.
* used to illustrate the functional requirements of the system and its interaction with external agents (actors).
* A use case diagram gives us a high level view of the system without going into implementation details.

Note – User can search for room and book room via admin. It means user just providing information to the admin for booking room.

**11. References**

1. IEEE Standard 830-1998 for Software Requirements Specification.
2. Existing manual hotel booking workflows and processes.
3. Python Software Foundation. (2024). Python documentation. Retrieved from <https://www.python.org/doc/>
4. GeeksforGeeks. (2024). Hotel management system project in Python. Retrieved from <https://www.geeksforgeeks.org/>
5. W3Schools. (2024). Python Object-Oriented Programming. Retrieved from <https://www.w3schools.com/python/>
6. Online resources and tutorials related to Python’s tkinter library and its use in building GUI applications.

**12. Conclusion –**

The “Hotel Management System” is a robust and user-friendly solution designed to streamline the management of hotel operations, particularly room bookings. It effectively handles key functionalities such as checking room availability, calculating costs with applicable taxes, and managing customer bookings. This system ensures efficiency by providing accurate booking details, real-time room availability updates, and an organized summary of reservations. By automating these processes, the system minimizes manual errors and enhances customer satisfaction, making it an essential tool for modern hospitality management.

The implementation leverages Python’s capabilities, utilizing object-oriented programming principles for scalability and maintainability. It demonstrates how even a straightforward application can significantly enhance operational efficiency in a hotel environment. With further expansion and integration of additional modules such as customer management, staff management, or online payment systems, this system has the potential to evolve into a comprehensive hotel management suite.