

**Project Title: Software Testing for Hotel Booking System**

**Team Members:**

- Mostafa Mahmoud Salah
- Mohamed Ahmed Ramadan

**Section: cs**

# **Software Requirements Specification (SRS) for HOTEL**

## **1. Introduction**

The Hotel Booking System is a web application built using Flask, a Python web framework. It provides functionality for managing hotel rooms and bookings.

## **2. Functional Requirements**

### **2.1. Get Rooms all rooms.**

- The system should provide an API endpoint to retrieve the details of all available rooms.
- The API should respond with a JSON object containing information about each room, including the room type, price, and availability.

### **2.2. Get Room specific room**

- The system should provide an API endpoint to retrieve the details of a specific room based on the room number.
- The API should accept a room number as a parameter and respond with a JSON object containing information about the room if it exists.
- If the room number is not found, the API should respond with an error message and a status code of 404.

### **2.3. Get Bookings**

- The system should provide an API endpoint to retrieve the details of all bookings.
- The API should respond with a JSON object containing information about each booking, including the room number, start date, and end date.

### **2.4. Create Booking**

- The system should provide an API endpoint to create a new booking.
- The API should accept a JSON object containing the room number, start date, and end date for the booking.
- If any of the required fields (room number, start date, end date) are missing, the API should respond with an error message and a status code of 400.
- If the specified room number is not found, the API should respond with an error message and a status code of 404.
- If the specified room is not available for booking, the API should respond with an error message and a status code of 400.

- Upon successful creation of a booking, the API should respond with a JSON object containing the booking details and a status code of 201.
- The availability of the booked room should be updated to "not available" in the system.

### **3. Non-Functional Requirements**

#### **3.1. Usability**

- The system should be user-friendly and provide clear and concise error messages.
- The API responses should be in JSON format and well-structured for easy parsing and integration with other systems.

#### **3.2. Performance**

- The system should be able to handle multiple concurrent requests without significant performance degradation.
- The response time of the API endpoints should be kept minimal to ensure a smooth user experience.

### **4. Dependencies**

- The system relies on the Flask web framework and its associated libraries.
- Python version 3.x or above should be installed.

### **5. Constraints**

- The system assumes that the room numbers are unique identifiers for the hotel rooms.
- The start date and end date for bookings should be in a valid date format.

### **6. Future Enhancements**




- Implement update and delete functionality for bookings.
- Add authentication and authorization mechanisms to secure the API endpoints.
- Include validation checks for input data to ensure data integrity.

## **Test Scenario for HOTEL**

1. **test\_get\_rooms**: Tests the GET request for retrieving all rooms.
2. **test\_get\_room**: Tests the GET request for retrieving a specific room.
3. **test\_get\_room\_not\_found**: Tests the case where a specific room is not found.
4. **test\_get\_bookings**: Tests the GET request for retrieving all bookings.
5. **test\_create\_booking**: Tests the POST request for creating a new booking with different scenarios using parameterization.
6. **test\_booked\_room\_not\_available**: Tests the case where a room is already booked and not available for booking.
7. **test\_update\_booking**: Tests the PUT request for updating an existing booking.
8. **test\_update\_booking\_not\_found**: Tests the case where the booking to be updated is not found.
9. **test\_update\_booking\_invalid\_dates**: Tests the case where the updated booking has invalid dates.
10. **test\_delete\_booking**: Tests the DELETE request for deleting an existing booking.
11. **test\_delete\_booking\_not\_found**: Tests the case where the booking to be deleted is not found.
12. **test\_delete\_booking\_room\_available**: Tests the case where a booking is deleted and the room becomes available for booking again.
13. **test\_update\_booking\_room\_not\_available**: Tests the case where a booking is made, the room is not available, and then an update is attempted.
14. **test\_update\_booking\_invalid\_room**: Tests the case where an invalid room is specified for an update.
15. **test\_get\_room\_invalid\_id**: Tests the case where an invalid room ID is specified in the GET request.
16. **test\_create\_booking\_missing\_fields**: Tests the case where required fields are missing in the POST request for creating a booking.
17. **test\_create\_booking\_invalid\_dates**: Tests the case where invalid dates are specified in the POST request for creating a booking.

Entry Criteria
Code successfully compiles without errors
Code review has been conducted
Test environment is set up with dependencies
Initial test data is prepared
Test cases are defined and documented
Test plan is reviewed and approved
Test data is initialized to match the defined test data

### Execution Strategy:

Entry Criteria	Test Team	Technical Team	Notes
<i>Test environment(s) is available</i>			
<i>Test data is available</i>			
<i>Code has been merged successfully</i>			
<i>Development has completed unit testing</i>			
<i>Test scripts are completed, reviewed and approved by the Project Team</i>	