# **Project Report**

# **Hotel Silverpeak Booking System**

A Web-Based Hotel Reservation Platform

## **Abstract / Project Overview**

### **i. Brief Summary of the Project's Purpose and Key Features**

The Hotel Silverpeak Booking System is a secure, user-friendly web application that revolutionizes hotel reservations by enabling real-time room availability checks, instant bookings (Standard/Family/Suite), and hassle-free cancellations. Developed with PHP and MySQL for robust backend operations, and enhanced with responsive HTML5/CSS3 for seamless mobile/desktop access, the system implements a smart two-phase reservation process with automated pricing calculations and session-based transaction security. Its optimized database architecture ensures data integrity while advanced security protocols (prepared statements, input validation) protect against threats. Designed for scalability, the platform serves as an all-in-one digital solution for modern hotels, with potential for expansion to include online payments and customer portals.

**Introduction**

**i.Background and purpose of the project**

In today's digital age, hotels need efficient online booking systems to replace outdated manual processes. The Hotel Silverpeak Booking System was developed to provide a modern solution that simplifies room reservations for both guests and hotel staff. This web-based platform allows customers to easily check room availability, book their preferred room type (Standard, Family, or Suite), and manage cancellations - all through a simple, intuitive interface. For hotel management, the system automates the entire reservation process, reduces paperwork, minimizes errors, and maintains organized records of bookings and payments. By digitizing these operations, the project aims to enhance customer satisfaction while streamlining hotel

administration.

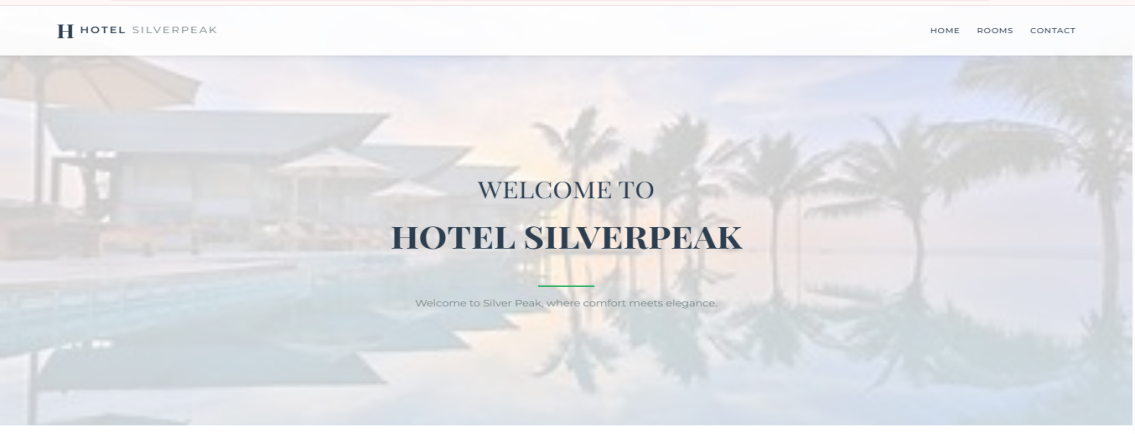
**ii.Tools & technologies used (HTML, CSS, PHP, MySQL, XAMPP, etc.)**

The system was built using a combination of fundamental web technologies. The frontend interface was created with HTML5 and CSS3 to ensure an attractive, responsive design that works across devices. PHP powers the backend logic, handling all booking operations, calculations, and data processing. MySQL serves as the database management system, securely storing all room information, guest details, and payment records. During development, XAMPP provided the local server environment for testing and debugging. Security was prioritized through the implementation of prepared statements to prevent SQL injection and input validation to ensure data integrity. These technologies were selected for their reliability, compatibility, and effectiveness in creating a fully-functional web application. Together, they form a robust foundation for the hotel booking system while allowing room for future enhancements and scalability.

**Project Screenshots**

**i.Screenshots of website pages (home page, forms, etc.)**

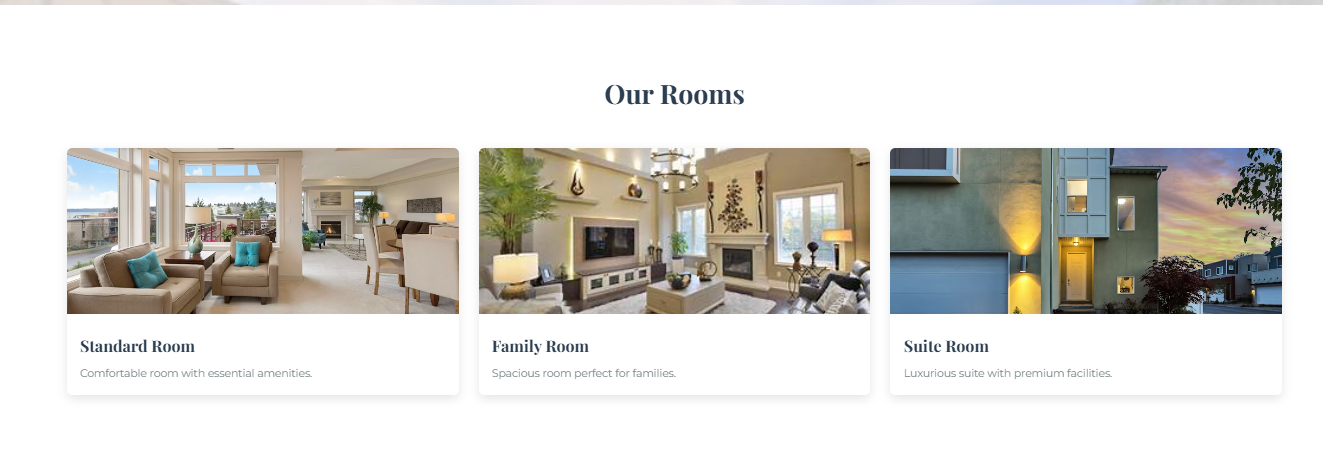
**ii.Brief description of each screenshot.**

****

**Website Header & Banner Description:**

The image shows Hotel Silverpeak's clean, elegant website header featuring:

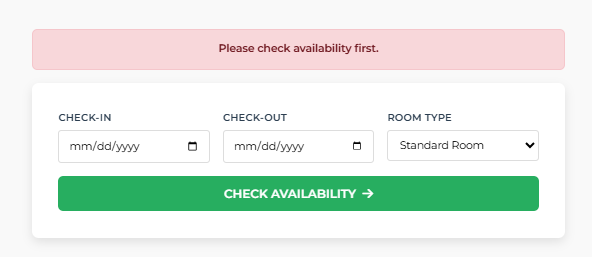
* A bold "H HOTEL SILVERPEAK" logo
* Simple navigation (Home | Rooms | Contact)
* A centered welcome banner with:  
  ✧ Main heading: "WELCOME TO"  
  ✧ Prominent hotel name: "HOTEL SILVERPEAK"  
  ✧ Tagline: *"Where comfort meets elegance"*

**

**Rooms Section Description:**

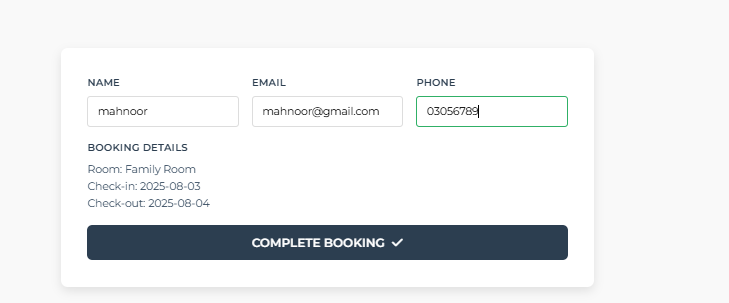
This image displays Hotel Silverpeak's room offerings in a clean, structured format:

1. Main Heading
   * "# Our Rooms" (clearly identifying the section)
2. Room Categories
   * Standard Room → "Comfortable room with essential amenities"
   * Family Room → "Spacious room perfect for families"
   * Suite Room → "Luxurious suite with premium facilities".

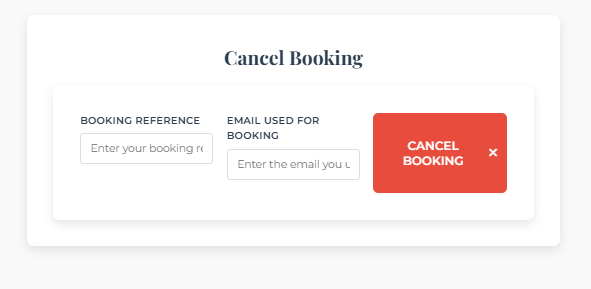
**

**Availability Check Screen:**

* Error message: "Please check availability first"
* Booking form with 3 fields:  
  ✓ CHECK-IN (date picker)  
  ✓ CHECK-OUT (date picker)  
  ✓ ROOM TYPE dropdown (default: Standard Room)
* "CHECK AVAILABILITY" call-to-action button (→ indicator)
* Minimalist design focusing on core booking functionality

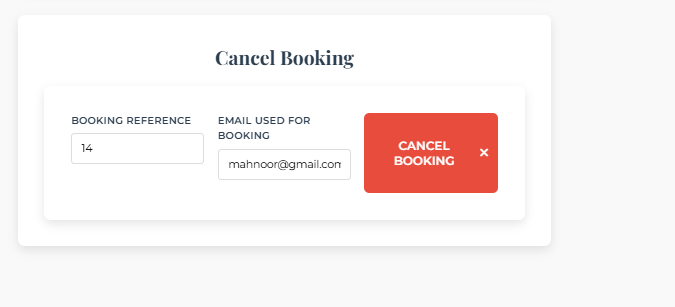
**

* Displays guest details (Name: Mahnoor, Email: [mahnoor@gmail.com](https://mailto:mahnoor@gmail.com/), Phone: 03056789)
* Shows booking summary:  
  ✓ Room Type: Family Room  
  ✓ Dates: August 3-4, 2025
* Features a prominent "COMPLETE BOOKING" button (✔ indicator).



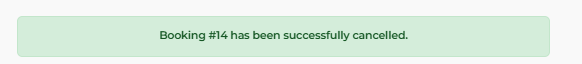
**Cancellation Form (Empty State)**

* Heading: "# Cancel Booking"
* Two input fields:  
  • BOOKING REFERENCE (placeholder: "Enter your booking rt")  
  • EMAIL USED FOR BOOKING (placeholder: "Enter the email you t")
* Red "CANCEL BOOKING" button (warning color for destructive action)
* Minimalist form design with clear labels

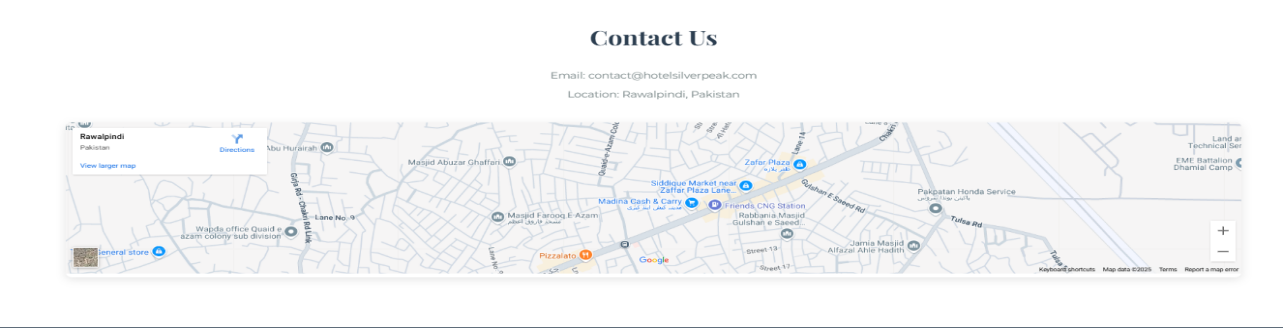


**Cancellation Form (Filled Example)**

* Shows real cancellation attempt:  
  • Booking #14  
  • Registered email: [mahnoor@gmail.com](https://mailto:mahnoor@gmail.com/)
* Maintains same structure as empty state but with user-provided data
* Demonstrates the actual data format required

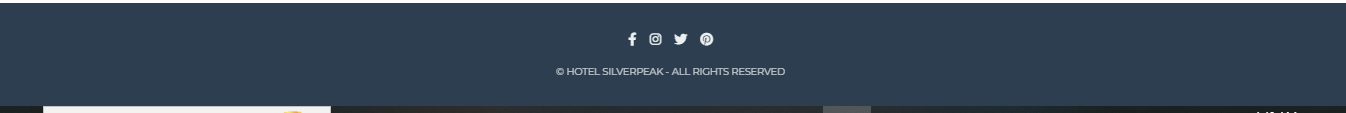
 **Success Message**

* Confirmation: "Booking #14 has been successfully cancelled."



This section provides essential contact information for **Hotel Silverpeak** and includes an embedded map for easy navigation.

* **Heading**: *Contact Us*
* **Email**: contact@hotelsilverpeak.com
* **Location**: Rawalpindi, Pakistan

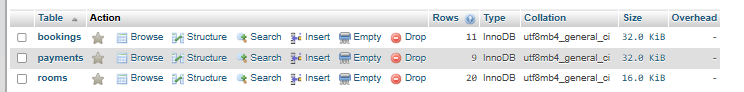


The footer of Hotel Silverpeak features social media icons (Facebook, Instagram, Twitter, Pinterest) centered on a dark blue background. Below them, a copyright notice states: “© HOTEL SILVERPEAK - ALL RIGHTS RESERVED.”

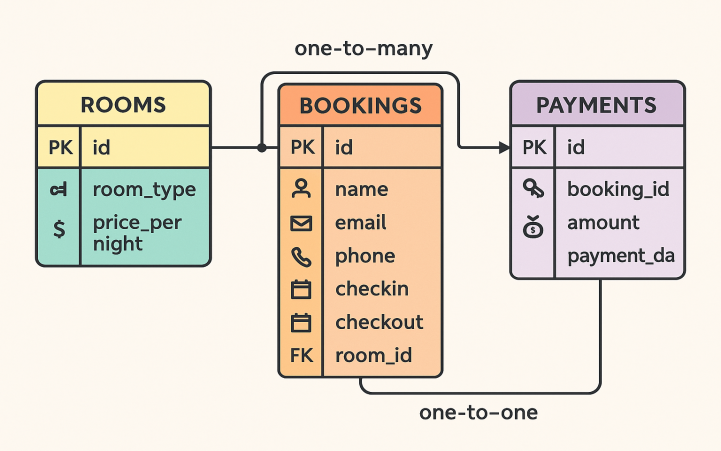
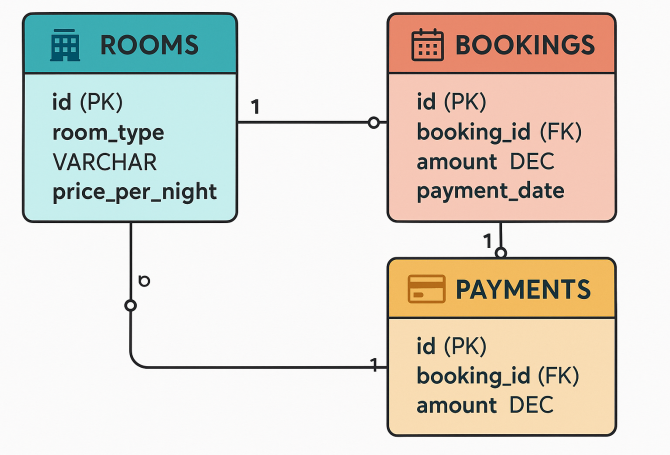
**Database Design**

**I. List of tables created (table names, columns, primary keys)**

1. Tables Created:  
   **a) rooms table:**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * room\_type (VARCHAR(50), NOT NULL)
   * price\_per\_night (DECIMAL(10,2), NOT NULL)
2. **b) bookings table:**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * name (VARCHAR(100), NOT NULL)
   * email (VARCHAR(100), NOT NULL)
   * phone (VARCHAR(20), NOT NULL)
   * checkin (DATE, NOT NULL)
   * checkout (DATE, NOT NULL)
   * room\_id (INT, NOT NULL, FOREIGN KEY REFERENCES rooms(id))
3. **c) payments table:**
   * id (INT, PRIMARY KEY, AUTO\_INCREMENT)
   * booking\_id (INT, NOT NULL, FOREIGN KEY REFERENCES bookings(id))
   * amount (DECIMAL(10,2), NOT NULL)
   * payment\_date (TIMESTAMP, DEFAULT CURRENT\_TIMESTAMP)



**Ii .ER Diagram or relational diagram (if applicable)**

****

**Relationship Explanation:**

1. ROOMS → BOOKINGS (One-to-Many):  
   One room can have many bookings over time (but only one active booking at a time)
2. BOOKINGS → PAYMENTS (One-to-One):  
   Each booking has exactly one payment record.

**Database Connection**

**i.Code snippet for the PHP database connection.**

****

**ii.Explanation of connection parameters (host, username, password, db name).**

$servername = "localhost"; // Server where MySQL is running

$username = "root"; // MySQL username (default is "root")

$password = ""; // MySQL password (empty for XAMPP/WAMP by default)

$dbname = "silver\_peak"; // Name of your database

**SQL Queries & Integration**

**i.Code snippets in PHP where SQL queries are used (SELECT, INSERT, UPDATE,**

**DELETE, JOINs, etc.).**

**ii.Explanation of what each query does and where it’s used in the website.**

## **1. Check Room Availability Query**

SELECT id FROM rooms WHERE room\_type = ?

AND id NOT IN (

SELECT room\_id FROM bookings

WHERE NOT (checkout <= ? OR checkin >= ?)

)

LIMIT 1

**Explanation:**

* This query checks for a room of the requested type (room\_type = ?) that is not already booked during the requested date range.
* The subquery selects room\_ids from bookings where the requested dates overlap with existing bookings (i.e., bookings where the checkout date is after the requested check-in date and the check-in date is before the requested check-out date).
* The NOT IN ensures the selected room is not among those booked in the overlapping period.
* LIMIT 1 returns only one available room ID if any exist.
* This query is prepared and executed with parameters for room type, check-in, and check-out dates to prevent SQL injection.

## **2. Insert Booking Record Query**

INSERT INTO bookings (name, email, phone, checkin, checkout, room\_id) VALUES (?, ?, ?, ?, ?, ?)

Explanation:

* Inserts a new booking record with customer details, booking dates, and the assigned room ID.
* Uses prepared statements with bound parameters for security.

## **3. Retrieve Room Price for Payment Calculation**

SELECT r.price\_per\_night

FROM bookings b

JOIN rooms r ON b.room\_id = r.id

WHERE b.id = ?

**Explanation:**

* Joins the bookings and rooms tables to fetch the price per night of the booked room.
* Uses the newly inserted booking ID to get the correct price.
* This price is used to calculate the total payment amount based on the number of nights.

## **4. Insert Payment Record Query**

INSERT INTO payments (booking\_id, amount) VALUES (?, ?)

**Explanation:**

* Inserts a payment record linked to the booking with the calculated total amount.
* Uses prepared statements to securely insert data.

## **5. Verify Booking for Cancellation**

SELECT b.id, p.id as payment\_id

FROM bookings b

LEFT JOIN payments p ON b.id = p.booking\_id

WHERE b.id = ? AND b.email = ?

**Explanation:**

* Checks if a booking exists with the given booking ID and email address.
* Joins bookings with payments to retrieve related payment information.
* Used to verify the booking before cancellation.

## **6. Delete Payment Record for Cancellation**

DELETE FROM payments WHERE booking\_id = ?

**Explanation:**

* Deletes the payment record associated with the booking to maintain referential integrity (likely due to foreign key constraints).

**7. Verify booking exists with this email using JOIN**

$verify\_sql = "SELECT b.id, p.id as payment\_id

FROM bookings b

LEFT JOIN payments p ON b.id = p.booking\_id

WHERE b.id = ? AND b.email = ?";

$verify\_stmt = $conn->prepare($verify\_sql);

$verify\_stmt->bind\_param("is", $booking\_id, $email);

$verify\_stmt->execute();

$verify\_result = $verify\_stmt->get\_result();

**Explanation:**

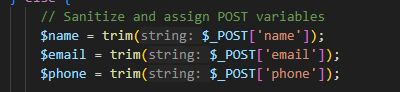
* Make sure the booking really exists and belongs to the person requesting cancellation.
* Find and handle any payment linked to the booking.
* Allow safe and complete cancellation of both the booking and its payment, if present.

**Security Measures**

**i.Steps taken to prevent SQL injection or security issues.**

SQL injection is a common security risk where attackers can manipulate SQL queries by injecting malicious input. The code takes the following steps to prevent this:

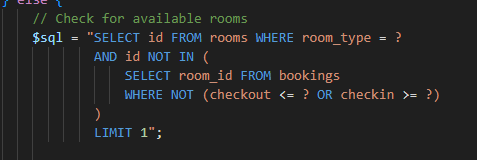
* No direct insertion of user input into SQL queries:  
  All database queries that use input from forms ($\_POST) are executed using prepared statements, not by directly appending user input to SQL strings.
* Input validation and sanitization:  
  The code trims and checks user input (e.g., dates, email, phone) before using it in queries. For example:



**ii.Use of prepared statements (if applicable)**

Prepared statements are used throughout the code to securely interact with the database. Here’s how:

## **Example : Checking Room Availability**



* The ? placeholders are replaced with actual values only after the SQL is parsed by the database, making it impossible for attackers to inject malicious SQL.

**Validation & Error Handling**

**i.How form inputs are validated before inserting into the database.**

1. Date Validation:

* Ensures check-in is not in the past.
* Ensures check-out is after check-in.

2. Required Fields:

* Checks that all necessary fields (name, email, phone) are provided before inserting a booking.

3. Email and Phone:

* The code trims whitespace, but does not do strict format validation (e.g., regex for email/phone).
* You could add further validation for stricter requirements.

4. Step Validation:

* Ensures the user checked availability before completing a booking.

**ii.Code snippets for input validation.**

**Date Validation Example:**

$today = date('Y-m-d');

if ($checkin < $today) {

$message = "Check-in date cannot be in the past.";

$messageClass = "error";

} elseif ($checkin >= $checkout) {

$message = "Check-out date must be after check-in date.";

$messageClass = "error";

} else {

// Proceed with availability check

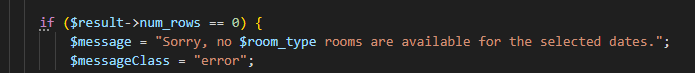
}

**iii.Error handling examples (if any)**

Setting Error Messages:

* If any validation fails, a user-friendly error message is set and displayed on the form.

**Example: No Room Available**



**Challenges & Learnings**

**i.Brief summary of challenges faced during the project.**

**First-Time Database Connection Setup**

* Struggled with configuring the backend to communicate with the database (e.g., MySQL/PostgreSQL).
* Issues included connection timeouts, authentication errors, and incorrect driver setups.

**Frontend-Backend Integration**

* Uncertainty about what to display on the frontend (e.g., how to structure forms, tables, or error messages).

**ii.Key learnings**

* Optimize database queries (indexes, caching, pagination).
* Always use prepared statements to prevent SQL injection.

**Conclusion**

**Summary of project outcomes**

This project was a valuable learning experience that highlighted the importance of security, performance optimization, and scalability in software development. Initially, we faced challenges like slow database queries and vulnerabilities to SQL injection, but by implementing prepared statements, query optimization, and proper input validation, we significantly improved both security and efficiency. Moving to cloud hosting with auto-scaling capabilities helped handle increased traffic, while robust error handling and monitoring reduced unexpected downtime. The key takeaway was adopting a security-first mindset—ensuring that every line of code not only works but is also resistant to attacks. Looking ahead, integrating Redis for caching and setting up CI/CD pipelines will further enhance performance and deployment efficiency. Overall, this project reinforced the need for proactive optimization, thorough testing, and scalable architecture in building reliable applications.