PROJECT REPORT

ON

HOTEL MANAGEMENT SYSTEM

ADVANCED INTERNET PROGRAMMING

Subject Code: 24CAP-652

SubmittedBy

Name: Shubham Thakur

UID: 24MCI10139

SubmittedTo

Mrs.BanitaMandal

Github Link: https://github.com/shubham122w/Hotel-Management-System

INDEX

Acknowledgement

I express my deepest gratitude to my mentor, professors, and peers who have guided me throughout this project. Their invaluable insights, encouragement, and support have played a crucial role in the successful completion of this study.

I am especially thankful to my faculty for providing me with the opportunity to work on such an interesting and interdisciplinary project. Their continuous guidance helped me explore different aspects of handwriting analysis and its connection to personality prediction.

Furthermore, I would like to acknowledge the authors of various research papers and articles that provided me with useful knowledge and inspiration. Their work laid the foundation for my understanding of graphology and machine learning techniques.

My sincere thanks to my friends and classmates for their constant motivation and constructive feedback, which significantly contributed to refining my project. Their discussions and shared knowledge enhanced my approach to problem-solving and technical development.

Lastly, I extend my heartfelt gratitude to my family for their unwavering support, patience, and encouragement throughout my academic journey. Their belief in my capabilities has been a driving force in achieving this milestone.

This project would not have been possible without the collective efforts and support of all the aforementioned individuals. I am deeply grateful for their contributions.

Thank you

Abstract

The *Hotel Management System* is a simplified yet efficient application designed to simulate the daily operations of hotel administration. Built using **HTML**, **CSS**, and **JavaScript** for the frontend and **Java with file-based storage** for the backend (without a database), this project offers an accessible platform for learning and managing core hospitality management tasks. It facilitates the seamless management of hotel staff and guest records, providing a user-friendly interface for hotel administrators.

The system begins with a secure **login interface**, ensuring only authorized users can access the dashboard. Once logged in, the user is presented with an interactive **Hotel Management Dashboard**, featuring two major sections — **Staff Management** and **Guest Management**. Each module allows administrators to add, view, and delete records efficiently.

In the **Staff Management** module, users can register staff members by entering basic information such as name and email. This data is stored persistently using localStorage, ensuring that it remains intact even after refreshing or closing the browser.

The **Guest Management** module includes fields for capturing key details such as guest name, room number, bill amount, and the availability of **food** and **cab services**. This allows the administrator to monitor guest needs, manage amenities, and maintain organized billing details.

The system is designed to operate entirely on the client side using browser localStorage and JavaScript logic, eliminating the need for any external database or server. This makes it ideal for small-scale, offline simulations or educational purposes. The backend component (written in Java using file handling with serialization) is responsible for similar tasks in a desktop version of the system.

Overall, this Hotel Management System demonstrates a comprehensive understanding of frontend design, user interface development, and data persistence using file-based and browser-based methods. It is a valuable educational project for students and beginners looking to explore the integration of UI/UX with backend data logic in real-world applications.

.

Introduction

The **Hotel Management System (HMS)** is designed to provide an efficient solution for managing the various operations of a hotel, including employee management, guest services, and overall hotel administration. The **Employee Management Module** plays a crucial role in this system by allowing hotel management to handle employee-related tasks, such as adding new staff, searching for employees, deleting records, viewing employee details, and saving data for future use. This system is built using Java, which provides a strong foundation for the backend operations, and employs file-based storage using serialization to persist employee data between sessions. By providing a simple, text-based interface, the system allows hotel administrators to manage their employee data effectively and ensures that all operations can be carried out smoothly without relying on a database.

The **Employee Management Module** offers various functionalities essential for maintaining employee records in a hotel. These include adding new employees by entering their details (ID, name, department, and salary), viewing a list of all employees, searching employees by their unique ID, and deleting an employee's record if necessary. The module also provides a way to save employee data to a file and load it upon system restart, ensuring that the data persists even after the program is closed. With these functionalities, this system aims to simplify employee management for hotel administrators, making it more organized and efficient.

Key Features of the Employee Management Module

1. **Add Employee**: Administrators can add new employees by entering their ID, name, department, and salary details. This information is stored in memory and can be saved to a file.

2. **View Employees**: Users can view all employee records in a clear, readable format, which includes their ID, name, department, and salary.

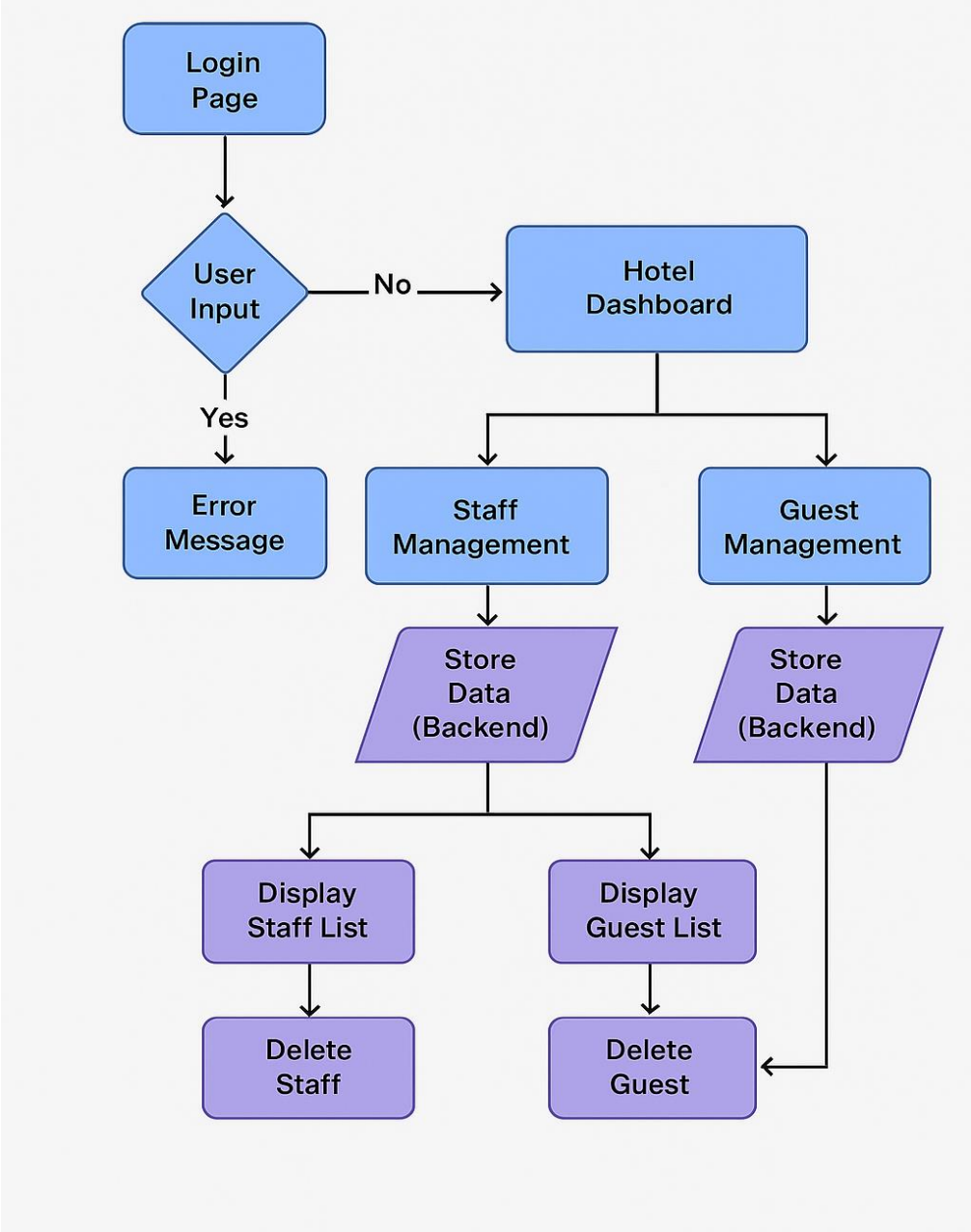
3. **Search Employee by ID**: Employees can be searched by their unique ID, allowing for quick access to specific employee details.

4. **Delete Employee by ID**: The system provides functionality to remove an employee's record by searching for their ID and deleting their information.

5. **Save and Load Data**: Employee data is saved to a file using Java's serialization, ensuring data persistence. The system can load data from the file upon startup, enabling continuous usage across multiple sessions.

These features ensure that the **Employee Management Module** is both practical and efficient for managing hotel staff, contributing to a smooth operational workflow.

Design flow of project



Code

EmployeeManager.java

```
package EM;

import java.io.*;
import java.util.*;

public class EmployeeManager {
    private List<Employee> employees = new ArrayList<>();
    private final String fileName = "employees.dat";

    public void addEmployee(Employee emp) {
        employees.add(emp);
        System.out.println("Employee added.");
    }

    public void viewEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
            return;
        }
        for (Employee e : employees) {
            System.out.println(e);
        }
    }

    public void searchById(int id) {
        for (Employee e : employees) {
            if (e.getId() == id) {
                System.out.println("Employee Found: " + e);
                return;
            }
        }
        System.out.println("Employee not found.");
    }

    public void deleteById(int id) {
        Iterator<Employee> iterator = employees.iterator();
        while (iterator.hasNext()) {
            if (iterator.next().getId() == id) {
                iterator.remove();
                System.out.println("Employee deleted.");
                return;
```

```
            }
        }
        System.out.println("Employee not found.");
    }

    public void saveToFile() {
        try (ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream(fileName))) {
            out.writeObject(employees);
            System.out.println("Data saved to file.");
        } catch (IOException e) {
            System.out.println("Error saving to file.");
        }
    }

    public void loadFromFile() {
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(fileName))) {
            employees = (List<Employee>) in.readObject();
            System.out.println("Data loaded from file.");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("No data file found. Starting fresh.");
        }
    }
}
```

Main.java

```
package EM;


import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        EmployeeManager manager = new EmployeeManager();
        manager.loadFromFile();

        Scanner scanner = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\n--- Employee Management System ---");
            System.out.println("1. Add Employee");
            System.out.println("2. View Employees");
            System.out.println("3. Search Employee by ID");
            System.out.println("4. Delete Employee by ID");
```

```
            System.out.println("5. Save to File");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    System.out.print("ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine(); // consume newline
                    System.out.print("Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Department: ");
                    String dept = scanner.nextLine();
                    System.out.print("Salary: ");
                    double salary = scanner.nextDouble();
                    manager.addEmployee(new Employee(id, name, dept, salary));
                    break;
                case 2:
                    manager.viewEmployees();
                    break;
                case 3:
                    System.out.print("Enter ID: ");
                    int searchId = scanner.nextInt();
                    manager.searchById(searchId);
                    break;
                case 4:
                    System.out.print("Enter ID to delete: ");
                    int delId = scanner.nextInt();
                    manager.deleteById(delId);
                    break;
                case 5:
                    manager.saveToFile();
                    break;
                case 0:
                    manager.saveToFile();
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice.");
            }
        } while (choice != 0);
    }
}
```

Employee.java

```
package EM;

import java.io.Serializable;

public class Employee implements Serializable {
    private int id;
    private String name;
    private String department;
    private double salary;

    public Employee(int id, String name, String department, double salary) {
        this.id = id;
        this.name = name;
        this.department = department;
        this.salary = salary;
    }

    public int getId() { return id; }
    public String getName() { return name; }
    public String getDepartment() { return department; }
    public double getSalary() { return salary; }

    public String toString() {
        return "ID: " + id + ", Name: " + name +
               ", Department: " + department + ", Salary: " + salary;
    }
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Hotel Management Dashboard</title>
  <style>
    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      margin: 0;
      padding: 0;
      background: linear-gradient(145deg, #0d1117, #1a1f27);
      color: #f0f6fc;
      display: flex;
      justify-content: center;
```

```
  align-items: center;
  height: 100vh;
  overflow: hidden;
}

.login-container {
  background-color: #161b22;
  padding: 30px;
  border-radius: 10px;
  width: 320px;
  text-align: center;
  border: 1px solid #30363d;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.4);
}

h2 {
  margin-bottom: 20px;
  color: #58a6ff;
}

input {
  width: 100%;
  padding: 10px;
  margin: 10px 0;
  border-radius: 5px;
  border: none;
  font-size: 1rem;
  background-color: #21262d;
  color: #f0f6fc;
}

button {
  width: 100%;
  padding: 10px;
  background-color: #58a6ff;
  border-radius: 5px;
  border: none;
  font-size: 1rem;
  color: white;
  cursor: pointer;
  transition: 0.3s;
}

button:hover {
```

```
  background-color: #3b8fd3;
}

.error {
  color: #f28d8d;
  font-size: 0.9rem;
  margin-top: 10px;
}

#dashboardPage {
  display: none;
  flex-direction: column;
  width: 100%;
  height: 100%;
}

header {
  background-color: #161b22;
  padding: 20px;
  text-align: center;
  font-size: 2rem;
  color: #58a6ff;
  border-bottom: 1px solid #30363d;
}

.dashboard {
  display: flex;
  height: calc(100vh - 60px);
}

.sidebar {
  width: 250px;
  background-color: #161b22;
  padding: 20px;
  height: 100%;
  border-right: 1px solid #30363d;
}

.sidebar h3 {
  color: #58a6ff;
  margin-bottom: 20px;
}

.sidebar button {
  width: 100%;
```

```
  background-color: #21262d;
  color: white;
  padding: 10px;
  margin: 10px 0;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: 0.3s;
}

.sidebar button:hover {
  background-color: #58a6ff;
}

.main-content {
  flex: 1;
  padding: 30px;
  overflow-y: auto;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 20px;
  color: #f0f6fc;
}

th, td {
  padding: 12px;
  border: 1px solid #30363d;
  text-align: center;
}

th {
  background-color: #21262d;
}

tr:nth-child(even) {
  background-color: #161b22;
}

.add-section {
  margin-top: 20px;
}
```

```
    .action-buttons button {
      margin: 0 5px;
      padding: 5px 10px;
    }

    input::placeholder {
      color: #8b949e;
    }
  </style>
</head>
<body>

  <!-- Login Page -->
  <div class="login-container" id="loginPage">
    <h2>Hotel Login</h2>
    <input type="text" id="username" placeholder="Username" />
    <input type="password" id="password" placeholder="Password" />
    <button onclick="login()">Login</button>
      <div id="errorMessage" class="error" style="display: none;">Invalid username or password</div>
  </div>

  <!-- Hotel Dashboard -->
  <div id="dashboardPage">
    <header>🏨 Hotel Management Dashboard</header>

    <div class="dashboard">
      <div class="sidebar">
        <h3>Navigation</h3>
        <button onclick="showStaff()">Staff</button>
        <button onclick="showGuests()">Guests</button>
      </div>

      <div class="main-content">
        <!-- Staff Section -->
        <div id="staffSection" style="display: none;">
          <h2>Staff Management</h2>
          <div class="add-section">
            <input type="text" id="staffName" placeholder="Name">
            <input type="email" id="staffEmail" placeholder="Email">
            <button onclick="addStaff()">➕ Add Staff</button>
          </div>
          <table>
            <thead>
              <tr>
```

```
            <th>Name</th>
            <th>Email</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody id="staffTable"></tbody>
      </table>
    </div>

    <!-- Guest Section -->
    <div id="guestSection" style="display: none;">
      <h2>Guest Management</h2>
      <div class="add-section">
        <input type="text" id="guestName" placeholder="Name">
        <input type="text" id="guestRoom" placeholder="Room Number">
        <input type="number" id="guestBill" placeholder="Bill Amount">
        <input type="text" id="guestFood" placeholder="Food Service (Yes/No)">
        <input type="text" id="guestCab" placeholder="Cab Service (Yes/No)">
        <button onclick="addGuest()">➕ Add Guest</button>
      </div>
      <table>
        <thead>
          <tr>
            <th>Name</th>
            <th>Room No.</th>
            <th>Bill</th>
            <th>Food</th>
            <th>Cab</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody id="guestTable"></tbody>
      </table>
    </div>
  </div>
 </div>
</div>

<script>
 function login() {
  const username = document.getElementById("username").value.trim();
  const password = document.getElementById("password").value.trim();
  if (username === "admin" && password === "hotel123") {
   sessionStorage.setItem("loggedIn", true);
   document.getElementById("loginPage").style.display = "none";
```

```
    document.getElementById("dashboardPage").style.display = "flex";
  } else {
    document.getElementById("errorMessage").style.display = "block";
  }
}

if (sessionStorage.getItem("loggedIn")) {
  document.getElementById("loginPage").style.display = "none";
  document.getElementById("dashboardPage").style.display = "flex";
}

function showStaff() {
  document.getElementById("staffSection").style.display = 'block';
  document.getElementById("guestSection").style.display = 'none';
}

function showGuests() {
  document.getElementById("staffSection").style.display = 'none';
  document.getElementById("guestSection").style.display = 'block';
}

function addStaff() {
  const name = document.getElementById("staffName").value.trim();
  const email = document.getElementById("staffEmail").value.trim();
  if (name && email) {
    const data = JSON.parse(localStorage.getItem("staff")) || [];
    data.push({ name, email });
    localStorage.setItem("staff", JSON.stringify(data));
    loadStaff();
  } else {
    alert("Please fill both fields.");
  }
}

function loadStaff() {
  const data = JSON.parse(localStorage.getItem("staff")) || [];
  const tbody = document.getElementById("staffTable");
  tbody.innerHTML = '';
  data.forEach((staff, idx) => {
    const row = `
      <tr>
        <td>${staff.name}</td>
        <td>${staff.email}</td>
        <td class="action-buttons">
          <button onclick="deleteStaff(${idx})">❌</button>
```

```
        </td>
      </tr>
    `;
    tbody.innerHTML += row;
  });
}

function deleteStaff(index) {
  const data = JSON.parse(localStorage.getItem("staff")) || [];
  data.splice(index, 1);
  localStorage.setItem("staff", JSON.stringify(data));
  loadStaff();
}

function addGuest() {
  const name = document.getElementById("guestName").value.trim();
  const room = document.getElementById("guestRoom").value.trim();
  const bill = parseFloat(document.getElementById("guestBill").value.trim());
  const food = document.getElementById("guestFood").value.trim();
  const cab = document.getElementById("guestCab").value.trim();

  if (name && room && !isNaN(bill) && food && cab) {
    const data = JSON.parse(localStorage.getItem("guests")) || [];
    data.push({ name, room, bill, food, cab });
    localStorage.setItem("guests", JSON.stringify(data));
    loadGuests();
  } else {
    alert("Please fill all guest fields.");
  }
}

function loadGuests() {
  const data = JSON.parse(localStorage.getItem("guests")) || [];
  const tbody = document.getElementById("guestTable");
  tbody.innerHTML = '';
  data.forEach((guest, idx) => {
    const row = `
      <tr>
        <td>${guest.name}</td>
        <td>${guest.room}</td>
        <td>$${guest.bill}</td>
        <td>${guest.food}</td>
        <td>${guest.cab}</td>
        <td class="action-buttons">
```

```
            <button onclick="deleteGuest(${idx})">❌</button>
          </td>
        </tr>
      `;
      tbody.innerHTML += row;
    });
  }

  function deleteGuest(index) {
    const data = JSON.parse(localStorage.getItem("guests")) || [];
    data.splice(index, 1);
    localStorage.setItem("guests", JSON.stringify(data));
    loadGuests();
  }

  window.onload = () => {
    loadStaff();
    loadGuests();
  };
 </script>
</body>
</html>
```

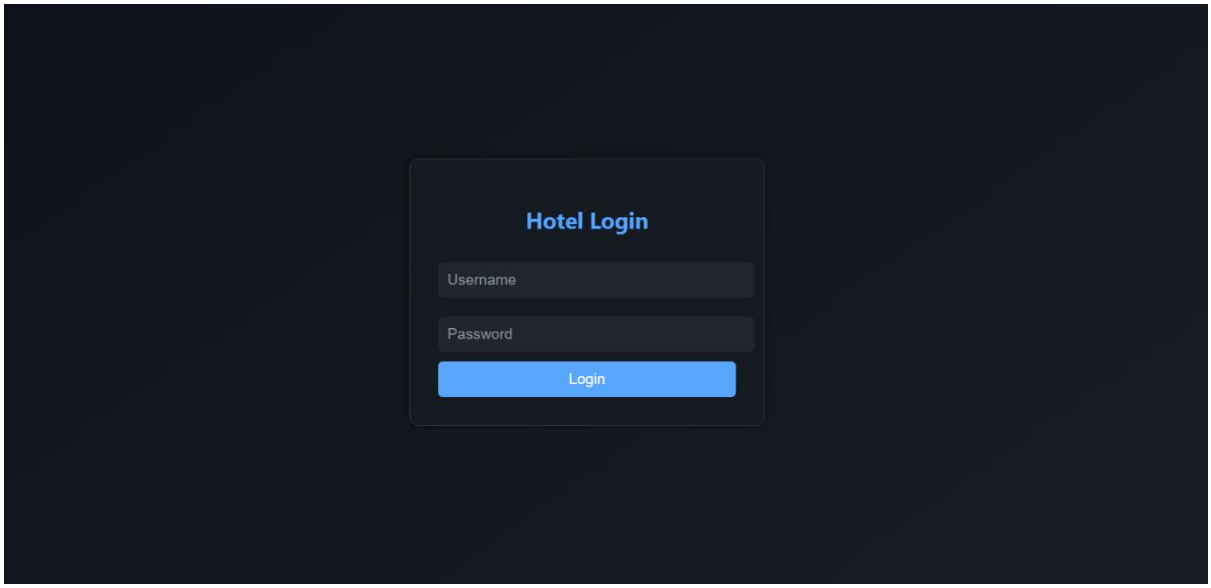
Pom.xml

```
<projectxmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany</groupId>
  <artifactId>HM</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>HM-1.0-SNAPSHOT</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <jakartaee>10.0.0</jakartaee>
  </properties>

  <dependencies>
```

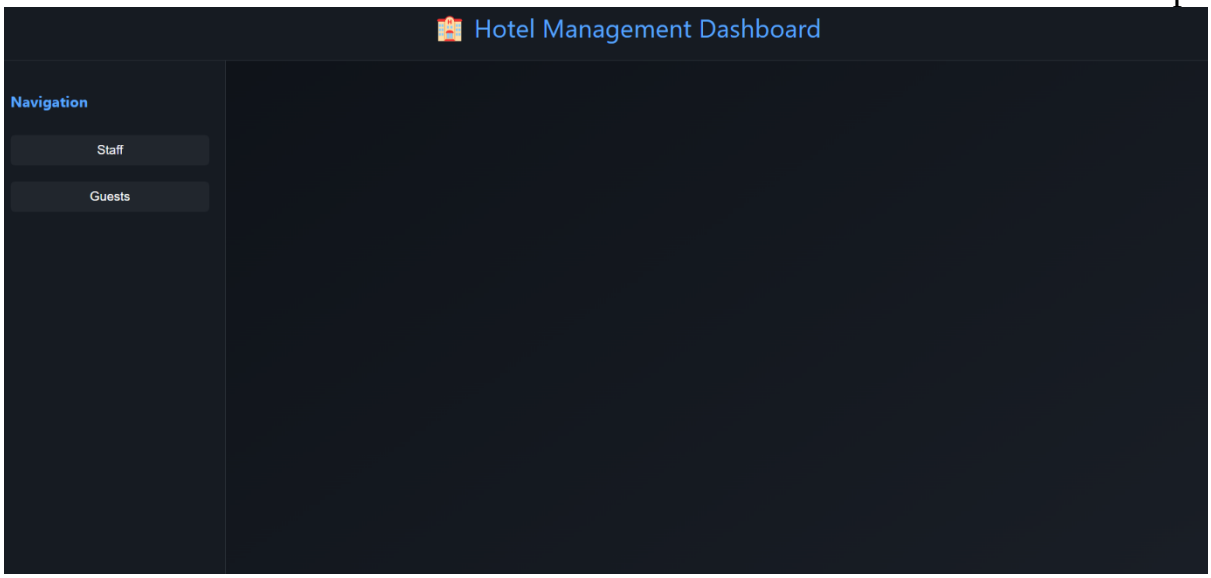
```
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-api</artifactId>
            <version>${jakartaee}</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>11</source>
                    <target>11</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.2</version>
            </plugin>
        </plugins>
    </build>
</project>
```

Output

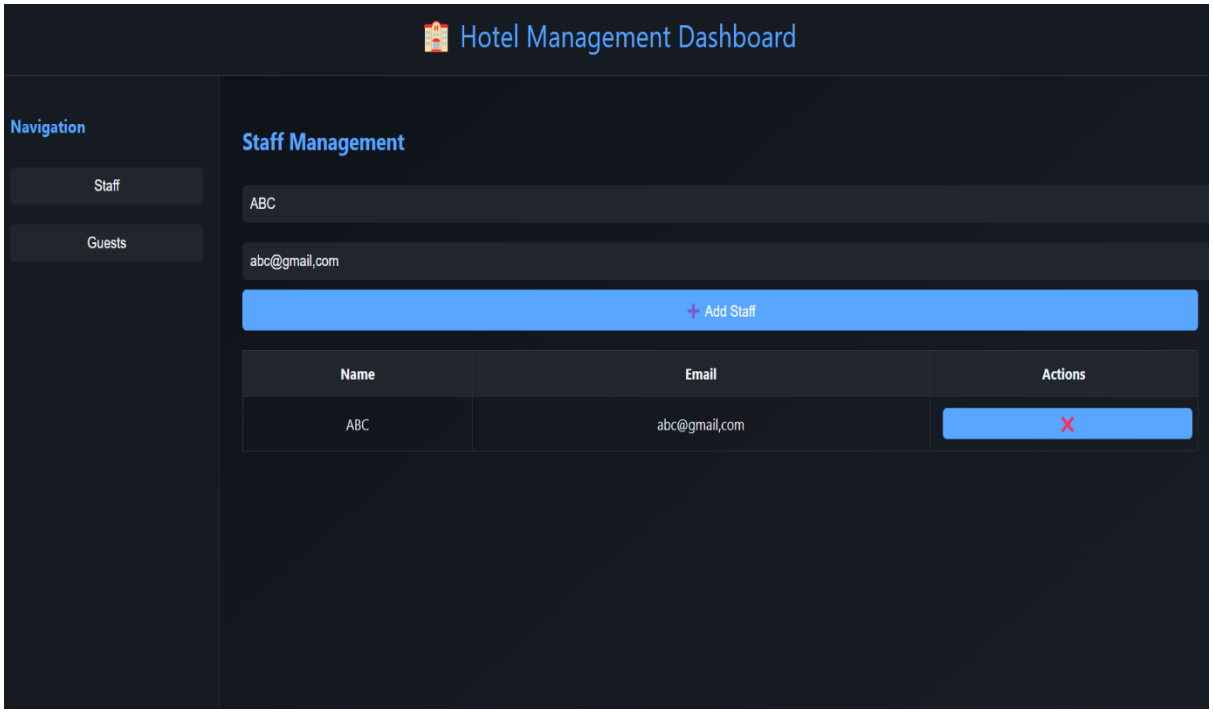
Login page:-



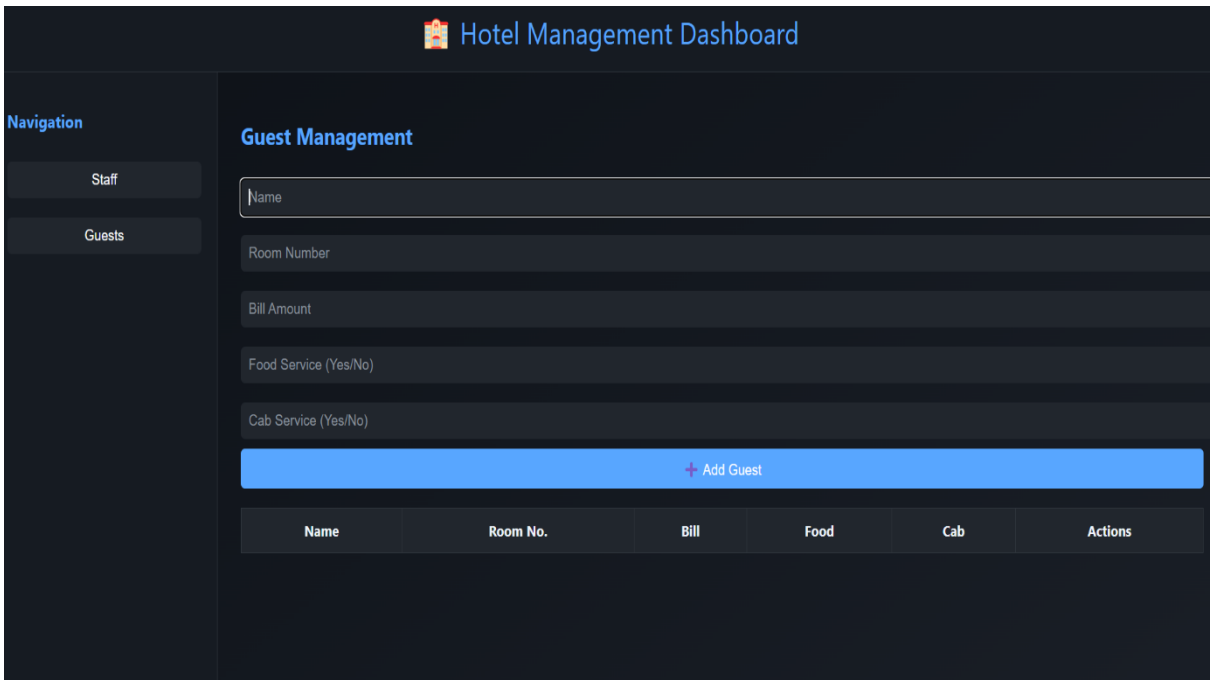
Dashboard:- pa



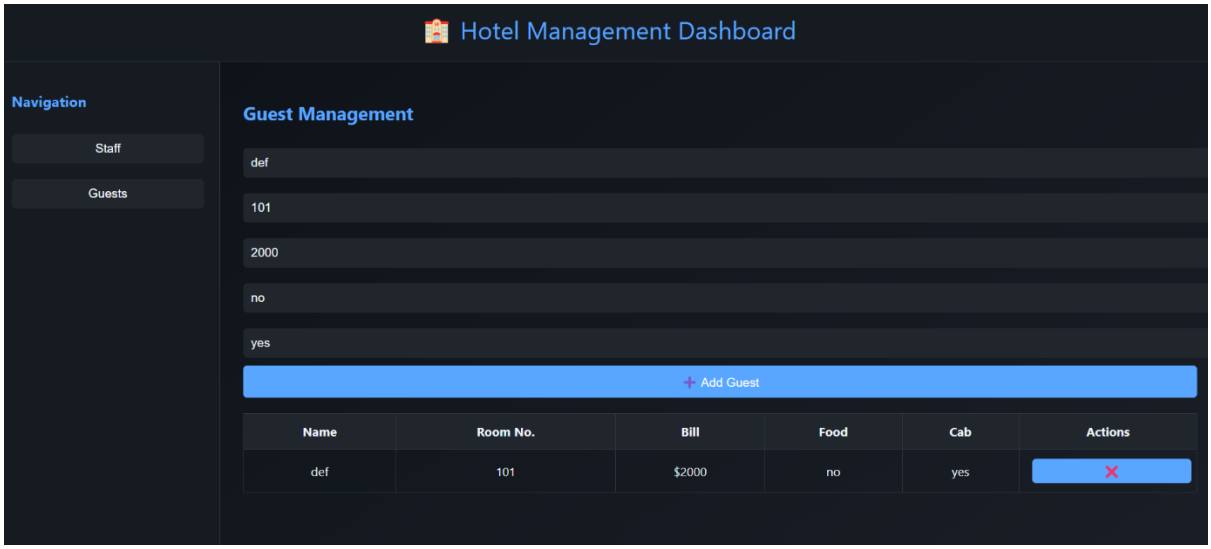
Staff Management:-



Guest management(interface):-



Guest Entries:-



Result analysis

This project provides a functional Hotel Management Dashboard with key features for staff and guest management. Here's a detailed analysis of the project's features, user experience, functionality, and potential improvements:

1. User Authentication (Login Page)
Functionality:

- Users are prompted to enter their username and password.
- On successful login (username: "admin", password: "hotel123"), the user is granted access to the dashboard.
- An error message is displayed if incorrect credentials are entered.

Analysis:

- **Strengths:**
  - Simple and intuitive login interface.
  - The login system is straightforward and works well for basic demonstration purposes.
- **Weaknesses:**
  - The login system is hardcoded, making it unsuitable for real-world use. For production, you'd need proper user authentication mechanisms.
  - Using localStorage for login tracking is not secure, as users can easily manipulate the local storage data.

Suggestions:

- Implement real authentication mechanisms, possibly connecting with a backend for secure login management.
- Add more robust session management to enhance security.

2. Hotel Dashboard (Navigation and Layout)
Functionality:

- Once logged in, the user is presented with a dashboard containing two main sections: "Staff Management" and "Guest Management."
- The dashboard layout includes a sidebar with buttons to toggle between these sections.

Analysis:

- **Strengths:**
  - The layout is user-friendly and well-organized, with easy navigation between different sections.
  - The use of a sidebar for navigation provides a clear structure and makes it easy to switch between staff and guest management.
- **Weaknesses:**
  - While the layout is functional, it can be enhanced with more dynamic content or more complex data visualizations for better insight.

Suggestions:

- Enhance the sidebar with active state indications or icons.
- Consider adding more sections or data visualizations, such as a summary of hotel revenue, occupancy rate, etc.

3. Staff Management

Functionality:

- Users can add staff details (name, email) and view the list of staff in a table format.
- The staff list is persisted in the localStorage and can be deleted.

Analysis:

- **Strengths:**
  - The staff management section allows for basic CRUD functionality (create, read, delete) in a simple table format.
  - Data is persisted in localStorage, ensuring the data is saved even after the page is refreshed.
- **Weaknesses:**
  - The deletion process removes the staff entry immediately, with no confirmation message, which can lead to accidental deletions.
  - The system doesn't handle email validation or checks for duplicate entries (e.g., two staff members with the same email).

Suggestions:

- Implement a confirmation prompt when deleting staff members to prevent accidental deletion.
- Add form validation for email fields to ensure proper formatting.
- Ensure staff entries have unique identifiers (e.g., employee IDs) to avoid duplicates.

Conclusion

The **Hotel Management Dashboard** project effectively demonstrates the basic functionality required for managing staff and guest data in a hotel setting. It provides an intuitive, user-friendly interface for managing hotel operations such as adding and deleting staff and guest records, displaying data in an organized table format, and offering a simple login mechanism for access control.

Key Highlights:

- **Staff Management:** The project allows users to add, view, and delete staff information, with data being stored in localStorage for persistence across sessions.
- **Guest Management:** Similar to the staff section, guests can be added with key details such as room number, bill amount, and services availed. The system also allows guests to be removed from the list.

Strengths:

- **User-Friendly Design:** The dark theme and clear layout contribute to a pleasant user experience. Navigation is simple, with a sidebar for switching between sections.

Areas for Improvement:

- **Authentication and Security:** The hardcoded login system and reliance on localStorage for storing sensitive data are not secure. A real-world application should use secure authentication mechanisms and store data in a backend database.
- **Validation and Error Handling:** Form validation for input fields (e.g., checking for empty fields, ensuring valid email format) is lacking, which could lead to errors in data entry.
- .

Future work

While the current version of the **Hotel Management Dashboard** serves as a functional prototype for hotel operations, there are several areas for future work and enhancements to make the system more scalable, secure, and feature-rich. Below are some potential improvements and additional features that could be implemented:

- **Database Integration**: Implement server-side database (e.g., MySQL, MongoDB) for persistent data storage instead of using localStorage, ensuring data consistency and scalability.

- **User Authentication and Roles**: Enhance the login system with different user roles (e.g., admin, staff) and implement more secure authentication methods (e.g., JWT, OAuth).

- **Advanced Reporting**: Add detailed reporting features, such as generating monthly revenue reports, guest occupancy rates, and staff performance summaries.

- **Booking System**: Introduce a booking module to allow guests to reserve rooms, including features like check-in/check-out dates, room availability, and payment processing.

- **Mobile App Development**: Develop a mobile version of the hotel management system to provide easy access to hotel staff and managers on the go.

- **API Integration**: Build RESTful APIs for external integrations (e.g., third-party booking sites, payment gateways) to streamline operations and enhance service offerings.

References

1. Hotel Management Systems and Web Development

- **Book**:
  - *"Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5"* by Robin Nixon – A great resource for full-stack development with a focus on PHP, MySQL, JavaScript, and related web technologies.
- **Article**:
  - *"How to Build a Hotel Management System using PHP and MySQL"* - Tutorials and guides available on sites like W3Schools and GeeksforGeeks.

2. Frontend Development (HTML, CSS, JavaScript)

- **MDN Web Docs**:
  - *HTML: Introduction* – MDN Web Docs
  - *CSS: Introduction* – MDN Web Docs
  - *JavaScript: Introduction* – MDN Web Docs
- **CSS Frameworks**:
  - *Bootstrap 5 Documentation* – Bootstrap Docs
  - *Tailwind CSS Documentation* – Tailwind Docs

3. Backend Development and Databases

- **MySQL Documentation**:
  - *MySQL Official Documentation* – MySQL Documentation
  - *SQL for Web Developers* – A guide for integrating MySQL into web development, available through books or online platforms like Udemy and Coursera.
- **Node.js and Express**:
  - *"Node.js Design Patterns"* by Mario Casciaro – A great book for understanding Node.js and scalable web application patterns.
- **Java (Spring Boot or Other Frameworks)**:
  - *Spring Framework Documentation* – Spring Docs
  - *"Spring in Action"* by Craig Walls – A great resource for working with Spring-based web applications.

4. Authentication and Security

- **OAuth and JWT Authentication**:
  - *JWT.io Introduction* – JWT.io
  - *OAuth 2.0 and OpenID Connect* – Auth0 Blog
- **Password Hashing**:
  - *bcrypt Documentation* – bcrypt NPM