**Hotel Management System (HMS) - Project Documentation**

**1. Project Overview**

**Project Name:** Hotel Management System (HMS)  
**Objective:** To automate and manage hotel operations, including hotel/room management, reservations, customers, payments, and housekeeping.

**Key Goals:**

* Efficiently manage hotels, rooms, and reservations.
* Track customer information and history.
* Handle payments and invoices.
* Manage housekeeping and staff assignments.
* Provide a centralized, web-based system with a clean UI.

**2. Actors**

| **Actor** | **Description** |
| --- | --- |
| Admin | System administrator, manages hotels, rooms, users, and configuration. |
| Receptionist | Manages reservations, check-in/check-out, and payments. |
| Customer | Makes reservations and views booking details (optional self-service portal). |
| Housekeeping Staff | Updates housekeeping tasks and room cleaning status. |
| System | Automated notifications, alerts, and reports. |

**3. Use Case Diagram**

+------------------------+

| Admin |

+------------------------+

| | |

v v v

[Manage Hotels] [Manage Rooms] [Manage Users]

|

v

[View Reports]

+------------------------+

| Receptionist |

+------------------------+

| | |

v v v

[Manage Reservations] [Check-In/Check-Out] [Process Payments]

|

v

[Generate Invoice]

+------------------------+

| Customer |

+------------------------+

|

v

[Make Reservation] [View Reservation]

+------------------------+

| Housekeeping Staff |

+------------------------+

|

v

[Update Housekeeping Tasks]

*This is a textual representation; for diagrams you can use tools like Lucidchart, Draw.io, or MS Visio.*

**4. Functional Requirements**

**4.1 User Management**

* Add/Edit/Delete users (Admin only).
* Assign roles: Admin, Receptionist, Housekeeping.
* Login/Logout.
* Password reset.

**4.2 Hotel Management**

* Add/Edit/Delete hotels.
* Store hotel details: name, address, phone, timezone.
* View hotel list with pagination/search/filter.

**4.3 Room Management**

* Add/Edit/Delete rooms.
* Assign room types (Standard, Deluxe, Suite, etc.) and rates.
* Track room status (available, occupied, maintenance).

**4.4 Customer Management**

* Add/Edit/Delete customers.
* Store personal details: name, contact, email, address.
* View customer history (past reservations, payments).

**4.5 Reservation Management**

* Create/Edit/Delete reservations.
* Assign rooms to customers.
* Track reservation status: Booked, Checked-In, Checked-Out, Cancelled.
* Calculate total amount automatically.
* Generate reservation reports.

**4.6 Check-In / Check-Out**

* Update reservation status during check-in/check-out.
* Update room status accordingly.

**4.7 Payment Management**

* Record payments for reservations.
* Support multiple payment methods (Cash, UPI, Credit/Debit, PayPal).
* Generate invoice and transaction ID.

**4.8 Housekeeping Management**

* Assign tasks to staff.
* Track room cleaning and maintenance status.
* Mark tasks as complete.

**4.9 Reporting**

* Generate reports for reservations, revenue, customer activity.
* Filter by date range, hotel, room type, or status.

**4.10 UI / Frontend**

* Responsive web interface.
* Separate views for Admin, Receptionist, Customer, and Staff.
* Navigation menu highlighting active modules.

**5. Non-Functional Requirements**

* **Performance:** Handle multiple concurrent users.
* **Security:** JWT authentication, role-based access control.
* **Scalability:** Modular services (backend microservices optional).
* **Maintainability:** Clean code structure, reusable UI components.
* **Compatibility:** Web application compatible with modern browsers.

**6. Technology Stack**

| **Layer** | **Technology** |
| --- | --- |
| Frontend | Angular / React / Thymeleaf (for Spring Boot) |
| Backend | Spring Boot (Java), Spring Security, Spring Data JPA |
| Database | MySQL / PostgreSQL |
| Caching | Redis (optional) |
| Messaging | Kafka / RabbitMQ (optional for real-time notifications) |
| Containerization | Docker |
| Deployment | AWS / GCP / On-premise |

**7. System Architecture**

**High-Level Modules:**

1. **User Service:** Authentication, roles, and profiles.
2. **Hotel Service:** Hotel CRUD operations.
3. **Room Service:** Room management and availability tracking.
4. **Reservation Service:** Bookings, check-in/out, and status updates.
5. **Customer Service:** Customer data and history.
6. **Payment Service:** Payment processing and invoice generation.
7. **Housekeeping Service:** Task management and room status updates.
8. **Reporting Module:** Reservation reports, revenue, analytics.

**Data Flow:**

* Frontend sends requests → Backend REST APIs → Database operations → Response.
* Optional WebSocket / Kafka for real-time updates (e.g., room status, reservations).

**8. Database Design (ER Diagram)**

**Key Tables:**

1. **Users** (id, name, email, password, role)
2. **Hotels** (id, name, address, phone, timezone)
3. **Rooms** (id, hotel\_id, room\_number, room\_type, status, rate)
4. **Customers** (id, first\_name, last\_name, email, phone, address)
5. **Reservations** (id, customer\_id, hotel\_id, room\_id, check\_in\_date, check\_out\_date, status, total\_amount)
6. **Payments** (id, reservation\_id, payment\_method, transaction\_id, amount, payment\_date)
7. **HousekeepingTasks** (id, room\_id, staff\_id, task\_date, status)

**9. Sample Use Cases**

| **Use Case** | **Actor** | **Description** |
| --- | --- | --- |
| Create Reservation | Receptionist | Add a new reservation with check-in/out dates and assign a room. |
| Edit Reservation | Receptionist | Modify existing reservation details. |
| Check-In | Receptionist | Mark customer as checked-in and update room status. |
| Process Payment | Receptionist | Record payment, generate invoice, assign transaction ID. |
| Manage Rooms | Admin | Add, update, or remove rooms and their types. |
| Assign Housekeeping Task | Housekeeping Staff | Assign cleaning or maintenance tasks. |

This documentation gives a **full overview of the project**, from actors, use cases, functional and non-functional requirements, to database and architecture.

let’s add **role-based login** so that:

* Users can log in with username/password.
* Each user has a ROLE (e.g., ADMIN, MANAGER, CUSTOMER, HOUSEKEEPER).
* After login, users are redirected to the correct dashboard/page based on their role.
* Access to certain URLs is restricted by role.

**xtend your User entity with roles**

If you don’t already have it, add a Role entity and a relation to User.

@Entity

@Table(name = "roles")

public class Role {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(unique = true, nullable = false)

private String name; // e.g., ROLE\_ADMIN, ROLE\_MANAGER

}

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(unique = true, nullable = false)

private String username;

private String passwordHash;

@ManyToMany(fetch = FetchType.EAGER)

@JoinTable(

name = "user\_roles",

joinColumns = @JoinColumn(name = "user\_id"),

inverseJoinColumns = @JoinColumn(name = "role\_id"))

private Set<Role> roles = new HashSet<>();

}

**2. Implement UserDetailsService**

This lets Spring Security fetch your user + roles.

@Service

public class CustomUserDetailsService implements UserDetailsService {

private final UserRepository userRepository;

public CustomUserDetailsService(UserRepository userRepository) {

this.userRepository = userRepository;

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepository.findByUsername(username)

.orElseThrow(() -> new UsernameNotFoundException("User not found"));

return new org.springframework.security.core.userdetails.User(

user.getUsername(),

user.getPasswordHash(),

user.getRoles().stream()

.map(role -> new SimpleGrantedAuthority(role.getName()))

.toList()

);

}

}

**3. Update SecurityConfig**

Enable form login, role-based access, and redirects.

@Configuration

@EnableWebSecurity

public class SecurityConfig {

private final CustomUserDetailsService userDetailsService;

public SecurityConfig(CustomUserDetailsService userDetailsService) {

this.userDetailsService = userDetailsService;

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http

.csrf(csrf -> csrf.disable()) // enable CSRF if you use forms

.authorizeHttpRequests(auth -> auth

.requestMatchers("/", "/home", "/css/\*\*", "/js/\*\*", "/images/\*\*").permitAll()

.requestMatchers("/admin/\*\*").hasRole("ADMIN")

.requestMatchers("/manager/\*\*").hasRole("MANAGER")

.requestMatchers("/customer/\*\*").hasRole("CUSTOMER")

.requestMatchers("/housekeeping/\*\*").hasRole("HOUSEKEEPER")

.anyRequest().authenticated()

)

.formLogin(form -> form

.loginPage("/login") // custom login page

.permitAll()

.successHandler(successHandler()) // custom redirect logic

)

.logout(logout -> logout.permitAll());

return http.build();

}

@Bean

public AuthenticationManager authManager(HttpSecurity http, PasswordEncoder encoder) throws Exception {

return http.getSharedObject(AuthenticationManagerBuilder.class)

.userDetailsService(userDetailsService)

.passwordEncoder(encoder)

.and()

.build();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationSuccessHandler successHandler() {

return (request, response, authentication) -> {

String role = authentication.getAuthorities().iterator().next().getAuthority();

if (role.equals("ROLE\_ADMIN")) {

response.sendRedirect("/admin/dashboard");

} else if (role.equals("ROLE\_MANAGER")) {

response.sendRedirect("/manager/dashboard");

} else if (role.equals("ROLE\_CUSTOMER")) {

response.sendRedirect("/customer/dashboard");

} else if (role.equals("ROLE\_HOUSEKEEPER")) {

response.sendRedirect("/housekeeping/dashboard");

} else {

response.sendRedirect("/home");

}

};

}

}

**4. Create role-based dashboards**

In src/main/resources/templates/:

* /templates/admin/dashboard.html
* /templates/manager/dashboard.html
* /templates/customer/dashboard.html
* /templates/housekeeping/dashboard.html

Each should be a Thymeleaf page. Example:

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Admin Dashboard</title>

</head>

<body>

<h2>Welcome, Admin!</h2>

<a th:href="@{/logout}">Logout</a>

</body>

</html>

**5. Login page**

/templates/login.html:

<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<title>Login</title>

</head>

<body>

<h2>Please login</h2>

<form th:action="@{/login}" method="post">

<div>

<label>Username:</label>

<input type="text" name="username"/>

</div>

<div>

<label>Password:</label>

<input type="password" name="password"/>

</div>

<button type="submit">Login</button>

</form>

</body>

</html>

✅ Now when users log in:

* Admins go to /admin/dashboard.
* Managers → /manager/dashboard.
* Customers → /customer/dashboard.
* Housekeepers → /housekeeping/dashboard.