



Haramaya University

Building the Basis for Development

College of Computing and Informatics

Department of Software Engineering

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

Document for

FOR Feven HOTEL MANAGEMENT SYSTEM

Group members	ID
1. ABENEZER YOHANNES	861/13
2. EDEN BIRHANU	882/13
3. GETHAUN TAMIRAT	421/12
4. HEAVEN ENDRIS	889/13
5. NIYA MURAD	901/13

Advisor name

Mr. Dita A.

Date:

Signature

Submission date - Dec 31 , 2025

Submitted to:DEPARTMENT OF SOFTWARE ENGINEERING

Catalog

List of Tables	III
Lists of Figures	III
DEFINITION, ABBREVIATION AND ACRONYMS	IV
CHAPTER ONE: INTRODUCTION	1
1.1 Document Scope	1
1.2 Document Purpose	1
1.3 Document Conventions	2
1.4 Intended Audience	2
1.5 Team Organization	3
CHAPTER TWO - SYSTEM DESCRIPTION	4
2.1 System Overview	4
2.2 System Scope	5
2.3 User Classes and Characteristics	6
2.4 Business Rules	7
2.5 Operating Environment	7
2.6 Design And Implementation Constraints	8
2.7 Assumptions and Dependencies	9
CHAPTER THREE - SYSTEM REQUIREMENT	10
3.1 Functional Requirements	10
3.2 Use Case Model	12
A) UseCase Diagram	12
B) UseCase Description	13
i. Guest	13
ii. Frontdesk	16
iii. Housekeeper	19
iv. Maintenance Staff Management	22
v. HR Manager	25
vi. Administrative Management	28
3.3 Data Requirements	31
3.3.3 Entity-Relationship Diagram	31
3.3.4 Data Validation Rules	32
3.4 External Interface Requirements	34
3.4.1 Hardware Interface	34
3.4.2 Software Interface	34
3.4.3 User Interface	34
3.4.4 Communication Interface	35
3.5 Non-functional Requirements	35
3.5.1 Performance Requirements	35
3.5.2 Usability Requirements	35
3.5.3 Security Requirements	36
3.5.4 Software Quality Attributes	36
APPENDIX A: Group Log	38

List of Tables

Table 1 - Team Organization	3
Table 2 - Lists of Business Rules of systems	7
Table 4 - Data Validation Rules	32

Lists of Figures

Figure 1 - high level context diagram of the system	5
Figure 2 - use case diagram	12
Figure 3 - Entity-Relationship Diagram	31

DEFINITION, ABBREVIATION AND ACRONYMS

Definitions

Software Requirements Specification (SRS): A comprehensive document that captures what the software system must do, including functional and non-functional requirements, constraints, and assumptions. It serves as the primary reference for developers, testers, and stakeholders throughout the project lifecycle.

Heaven Hotel Management System (HHMS): The web-based application described in this document, designed to automate and streamline hotel operations such as room bookings, guest management, housekeeping, maintenance, HR functions, and reporting.

Functional Requirements: Specific behaviors or functions the system must perform, typically expressed as actions the system shall carry out in response to user inputs or events.

Non-Functional Requirements: Quality attributes that define how the system performs its functions, including performance, security, usability, reliability, and maintainability.

Use Case: A description of how a specific actor interacts with the system to achieve a particular goal, including main flow, alternative flows, and preconditions/postconditions.

Entity-Relationship Diagram (ERD): A visual representation of the system's data model, showing entities (e.g., User, Room, Booking), their attributes, and relationships.

Role-Based Access Control (RBAC): A security mechanism that restricts system access based on the user's assigned role (e.g., Guest, Front Desk Staff, Administrator).

Abbreviations and Acronyms

Abbreviation/Acronym	Meaning
HHMS	Heaven Hotel Management System
SRS	Software Requirements Specification
SDD	Software Design Document
UI	User Interface
UX	User Experience
ERD	Entity-Relationship Diagram
CRUD	Create, Read, Update, Delete
ORM	Object-Relational Mapping
API	Application Programming Interface
HTTPS	Hypertext Transfer Protocol Secure
WCAG	Web Content Accessibility Guidelines
XSS	Cross-Site Scripting
CSRF	Cross-Site Request Forgery
SQL	Structured Query Language
NFR	Non-Functional Requirement
BR	Business Rule

CHAPTER ONE: INTRODUCTION

1.1 Document Scope

This Software Requirements Specification (SRS) document lays out the complete set of requirements for building the Heaven Hotel Management System (IHMS), a robust web-based application developed using the Laravel framework. At its core, it details everything from the high-level system overview to the nitty-gritty of functional and non-functional requirements, data models, interface designs, and more. We're talking about covering all the key modules that make a hotel run smoothly—like handling guest bookings, managing front desk tasks, assigning housekeeping duties, tracking maintenance issues, overseeing HR functions, and generating administrative reports. It's essentially the blueprint that ensures everyone knows what the system will do, how it'll perform, and what constraints we need to work within.

The document is organized in a straightforward, logical way to make it easy to navigate. We start with the basics in the introduction, move into a full system description in Chapter Two, and then dive deep into the requirements in Chapter Three. If you're reading this, I suggest beginning right here in Section 1 to get the big picture, then jumping to the parts that matter most to you. For instance, if you're a developer itching to code, head straight to Sections 2 and 3 for the implementation details and requirements that'll guide your work. Project managers or evaluators might want to stick with the intro and system overview for a broader sense of the project's goals and structure. This setup helps different readers—whether you're knee-deep in code or just overseeing the project—find what you need without wading through irrelevant stuff.

1.2 Document Purpose

The main goal of this SRS document is to create a solid foundation for the entire Heaven Hotel Management System project, acting as a clear, shared reference point for everyone involved. Think of it as the roadmap that captures what the end users really need and expect, translating those into precise, unambiguous functional and non-functional requirements. This way, the development team can build the system with confidence, knowing exactly how it should function to streamline hotel operations—from seamless guest check-ins to efficient staff management and everything in between. It's not just about listing specs; it's about making sure the system solves real problems, like reducing manual errors and boosting overall efficiency in a busy hotel environment.

Beyond guiding the current build, this SRS serves as a go-to resource for the future. It provides a detailed record of how the project kicked off, which will be invaluable for new developers or maintenance teams down the line when they need to update, scale, or tweak the system. Specifically, it:

- Defines the essential features the system must deliver, such as real-time room booking, invoice generation, and role-based access for different users.
- Acts as a benchmark for testing and validation, helping the QA team verify that everything works as intended.
- Promotes traceability, linking every requirement back through the development lifecycle to ensure nothing gets lost in translation.
- Supports ongoing improvements, making it easier to adapt the system as the hotel's needs evolve.

1.3 Document Conventions

This Software Requirements Specification (SRS) follows the formatting and writing standards set by the Department of Software Engineering at the College of Computing and Informatics, Haramaya University. We've adopted these conventions to keep the document clear, consistent, and easy to read for everyone involved in the project.

Formatting Conventions

- **Bold text** is used for all chapter titles, section headings, and subheadings to make the structure stand out at a glance.
- *Italic text* is used to emphasize important terms, notes, or references when they first appear.
- Tables are used extensively for presenting structured information such as user classes, business rules, entity attributes, and validation rules.
- Figures are numbered sequentially and referenced in the text (e.g., Figure 1 - Use Case Diagram).
- Bullet points and numbered lists are used to break down requirements and features clearly.

Naming and Requirement Conventions

- Functional requirements are phrased using the standard form - "The system shall..." to indicate mandatory behavior.
- Non-functional requirements are identified with unique codes (e.g., NFR-001) where needed for traceability.
- User roles (e.g., Guest, Front Desk Staff, Administrator) are capitalized for consistency.
- All diagrams follow standard UML notation where applicable.
- Monetary values are expressed in USD unless otherwise specified.
- Unique identifiers for requirements, business rules, and other elements follow patterns like BR-001, REQ-001, etc., to support easy referencing and traceability.

1.4 Intended Audience

This SRS document is written with a variety of readers in mind, each bringing different perspectives and needs to the Heaven Hotel Management System project. We've tailored the content and structure so that everyone can quickly find the information most relevant to them -

- The primary audience are the **Software Developers**. You'll use this document as your main guide for implementation—especially the detailed functional requirements, use cases, data models, and interface specifications in Chapters 2 and 3—to build a system that meets the stated needs accurately.
- To oversee progress the **project Advisors and Managers** uses these document to ensure the project stays on track with requirements, and verify that the team's work aligns with the original goals and academic standards.
- **Quality Assurance Team and Testers** are going use it to design comprehensive test cases and validation plans based on the functional and non-functional requirements, business rules, and use cases described here.

- **Examiners and Reviewers** are going to use it to evaluate the completeness, clarity, and feasibility of the requirements, as well as how well the proposed system addresses real-world hotel management challenges.
- **End Users (hotel staff and management)** are going to use it while you may eventually rely more on user manuals, this document gives you insight into what the system will do and how it will support your daily operations—from booking rooms to generating reports.
- Anyone who might work on updates, enhancements, or bug fixes later will find this a valuable reference for understanding the system's intended behavior and design rationale.

1.5 Team Organization

The Web-Based Hotel Management System is developed by a dedicated team of final-year Software Engineering students. The team is organized as follows -

Member Name	Role	Responsibilities
NIYA & ABENEZER	Project Lead & Full-stack Developer	Lead the project, system architecture, database design, and full-stack development
EDEN BIRHANU	Backend Developer	Design and implement the server-side logic, REST APIs, and database integration
GETHAUN TAMIRAT	Frontend Developer	Develop the user interface using modern web technologies and ensure responsiveness
HEAVEN ENDRIS	Project management	Write and execute test plans, perform functional and non-functional testing
Mr. Dita A.	Project Advisor	Provide academic supervision, feedback, and ensure the project meets curriculum standards

Table - 1 Team Organization

CHAPTER TWO - SYSTEM DESCRIPTION

2.1 System Overview

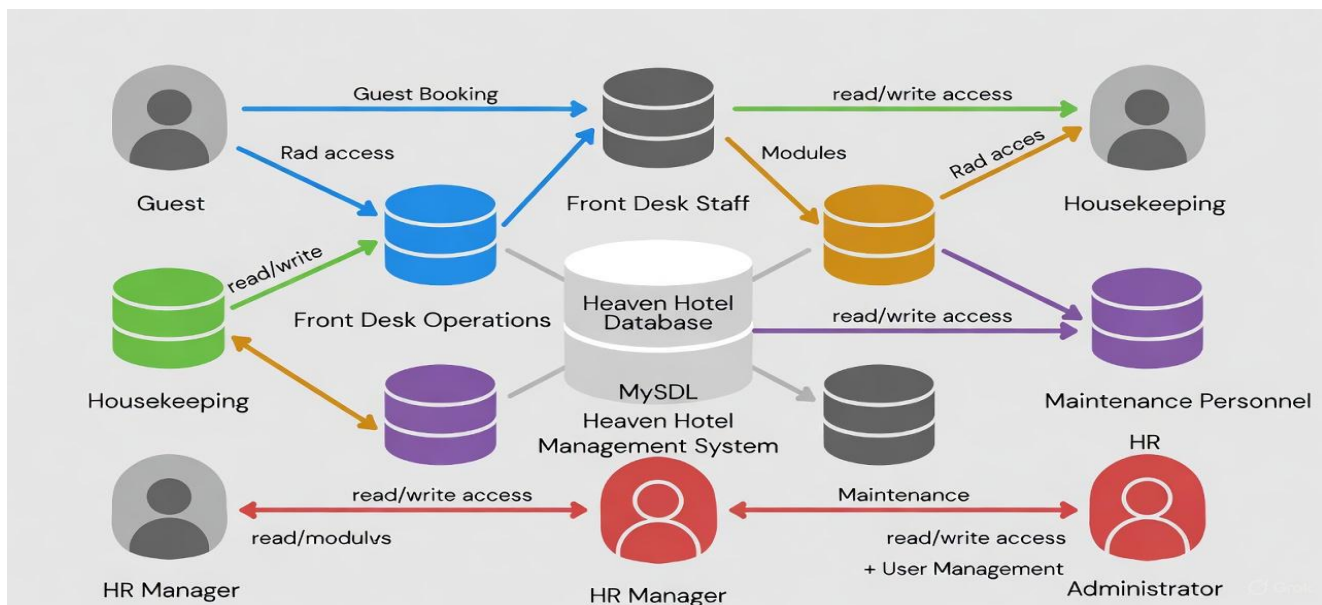
The Heaven Hotel Management System (HHMS) is a modern, web-based application we've designed to bring efficiency and order to the day-to-day operations of a mid-sized hotel. At its heart, the system automates and connects the many tasks that hotel staff handle manually today—everything from taking room bookings and managing guest check-ins to coordinating housekeeping, tracking maintenance issues, handling employee shifts, and producing useful reports for management. By replacing paper-based processes and scattered spreadsheets with a single, centralized platform, the system helps reduce errors, save time, and improve service quality for both guests and staff.

The system supports different types of users through role-based access, meaning each person sees only the features and data they need. Guests can browse and book rooms online without needing assistance. Front desk staff can quickly check availability, process walk-ins, or handle check-outs. Housekeepers get clear lists of rooms that need cleaning, while maintenance teams can log and resolve issues fast. HR managers oversee employee records and schedules, and administrators have full control to configure the system and view high-level reports. This role-based approach keeps things secure and makes the system feel tailored to each user's daily work.

Built on the Laravel framework with MySQL as the database and Bootstrap for a clean, responsive interface, the system is structured in a modular way. Each major functional area—guest booking, front desk operations, housekeeping, maintenance, HR, and administration—lives in its own module, making it easier to develop, test, and maintain. The application runs entirely in a web browser, so staff can access it from desktop computers at the hotel or even tablets on the floor, while guests use it from their phones or laptops. We chose these technologies because they are reliable, well-supported, and allow the system to grow if the hotel needs more features in the future.

Overall, the Heaven Hotel Management System is meant to solve real operational challenges - eliminating double bookings, speeding up check-in/check-out, ensuring rooms are cleaned promptly, keeping track of staff attendance, and giving management clear insights through reports. It's a complete yet practical solution that modernizes how the hotel runs without overwhelming users with unnecessary complex

Figure 1 - high level context diagram of the system



2.2 System Scope

The Heaven Hotel Management System (HHMS) is carefully scoped to deliver a complete, practical solution for managing the core operations of a mid-sized hotel while staying realistic for a final-year project. It focuses on automating key processes that are currently handled manually—such as paper-based bookings, room status tracking, and employee scheduling—to reduce errors, improve efficiency, and enhance both guest satisfaction and staff productivity.

The system explicitly includes the following major functionalities -

- Guest-facing features that allow online registration, room availability searches by date and type, booking creation, viewing of personal booking history, and cancellation within policy limits.
- Front desk tools for managing the full guest lifecycle, including walk-in bookings, confirmation of reservations, check-in and check-out processing, room status updates, and invoice generation.
- Housekeeping support through assignment of cleaning tasks, real-time room status updates (clean/dirty/ready), and tracking of completed work.
- Maintenance management with the ability to report issues, assign tasks to technicians, track progress, and record resolutions.
- Human resources capabilities covering employee record management, shift scheduling, attendance tracking, and basic reporting.
- Administrative controls for managing room types and individual rooms, generating operational and financial reports, viewing audit logs, and configuring system settings.

The application is fully web-based, accessible via standard browsers on desktops, tablets, and smartphones, with a responsive design that works well across devices. It enforces role-based access control so that each user—whether a guest browsing from home or a staff member at the front desk—sees only the interfaces and data relevant to their responsibilities.

Out of scope for this version are advanced features such as

- ◆ Integration with third-party payment gateways (payments will be recorded manually)
- ◆ Channel management for online travel agencies
- ◆ Restaurant point-of-sale integration
- ◆ Loyalty programs, or
- ◆ Mobile push notifications.

These could be added in future enhancements, but the current scope delivers a solid, self-contained system that directly addresses the most pressing operational needs of the hotel.

2.3 User Classes and Characteristics

The Heaven Hotel Management System is designed to support a variety of users, each with different responsibilities, access needs, and levels of technical familiarity. Understanding these user classes helps ensure the system is intuitive and efficient for everyone who interacts with it, whether they are guests booking from home or staff managing daily operations at the hotel.

Guests are external users who primarily access the system from outside the hotel to search for and book rooms. They typically have basic computer skills and use the system only occasionally—perhaps a few times a year when planning travel. Their interactions are focused on simple, self-service tasks like checking availability, viewing room details, and managing their own bookings, so the interface for them needs to be straightforward and require minimal guidance.

Front desk staff, such as receptionists, are internal users who rely on the system every day to handle guest arrivals, departures, and inquiries. They generally have intermediate computer skills and need quick access to room availability, booking details, and check-in/check-out processes. Efficiency is key here, as their work directly affects guest satisfaction during peak hours.

Housekeepers form another daily user group responsible for keeping rooms clean and ready for new guests. They usually have basic computer skills and access the system to view their assigned cleaning tasks and update room statuses. The design for them emphasizes simplicity, often through mobile-friendly views, so they can mark rooms as clean even while moving between floors.

Maintenance personnel handle repairs and technical issues reported for rooms or facilities. Their use of the system is more as-needed rather than daily, and they tend to have basic technical skills. They need clear lists of assigned tasks, the ability to update progress, and a way to close out issues once resolved.

HR managers oversee employee-related functions like records, scheduling, and attendance tracking. They use the system daily, possess intermediate computer skills, and require tools that allow them to manage staff information securely and generate reports for payroll or performance reviews.

Finally, administrators—typically IT-savvy hotel managers or system overseers—have the highest level of access. They use the system daily with advanced computer skills to configure settings, manage room inventories, view audit logs, and generate comprehensive reports. Their interface provides full control while maintaining security through role restrictions.

2.4 Business Rules

The Heaven Hotel Management System enforces a set of business rules to ensure operational accuracy, data consistency, and alignment with standard hotel practices. These rules are applied automatically throughout the system to guide user actions and prevent common errors.

Rule ID	Description	Impacted Requirements
BR-001	Guests can only book rooms that are marked as available for the selected dates.	3.1.1 Guest Booking Management
BR-002	Bookings can only be cancelled if the current status is pending or confirmed.	3.1.1 Guest Booking Management
BR-003	Check-in is permitted only on or after the scheduled check-in date and only for confirmed bookings.	3.1.2 Front Desk Operations
BR-004	Invoices are generated automatically upon check-out, based on room rate, stay duration, and recorded services.	3.1.2 Front Desk Operations
BR-005	Only employees marked as active in the system can be assigned to shifts or tasks.	3.1.5 HR Management
BR-006	Maintenance issues should be resolved within 24 hours, with the system highlighting overdue tasks.	3.1.4 Maintenance Management
BR-007	Room prices are determined solely by the assigned room type and any administrator-defined adjustments.	3.1.1 Guest Booking Management, 3.1.6 Administrative Functions

Table 2 - Lists of Business Rules of Systems

2.5 Operating Environment

The Heaven Hotel Management System is a web-based application designed to run in a standard, modern computing environment that is readily available in most hotel settings. It requires no specialized hardware beyond typical office or server setups, making it cost-effective and easy to deploy.

The operating environment consists of the following components -

Server Side

- The application server runs on a Linux or Windows server equipped with PHP 8.2 or higher.
- Data storage is handled by MySQL version 8.0 or later.
- A web server such as Apache or Nginx is required to serve the application and handle incoming requests securely over HTTPS.

Client Side

- Users access the system through modern web browsers including Google Chrome (version 90+), Mozilla Firefox (88+), Apple Safari (14+), or Microsoft Edge (90+).
- The responsive design ensures full functionality on desktop computers, laptops, tablets, and smartphones without requiring separate mobile applications.

Network Requirements

- A reliable internet connection is necessary for remote access (e.g., guests booking online) and for staff using the system across different devices within the hotel.
- Internal hotel operations can function over a local network if external internet access is limited, though online features such as guest bookings would then require connectivity.

Additional Requirements

- The system may require a printer for generating physical outputs such as customer bills, order tickets for the kitchen, or sales reports. While not part of the core system, the operating environment includes these peripheral dependencies. The client (Feven Hotel/Restaurant) is assumed to afford necessary software and hardware costs associated with the operating environment.

This environment leverages widely available, open-source, and well-supported technologies, allowing the hotel to host the system either on-premises or on a cloud provider of their choice. The setup supports easy maintenance, updates, and scalability as the hotel's needs grow.

2.6 Design And Implementation Constraints

The development of the Heaven Hotel Management System is guided by several design and implementation constraints to ensure consistency, reliability, security, and maintainability. These constraints were chosen based on the project's academic requirements, the need for a modern yet stable technology stack, and best practices in web application development. The key constraints are as follows

- The system must be built using the Laravel 12 framework, which provides a robust structure for routing, authentication, and database interactions while supporting rapid development.
- MySQL 8.0 or higher must be used as the relational database management system to store all application data, ensuring compatibility with Laravel's Eloquent ORM and strong support for transactions and relationships.
- Role-based access control must be implemented using the Spatie Laravel Permission package, allowing fine-grained permissions and roles without reinventing authentication logic.
- The user interface must utilize Bootstrap 5 for styling and layout to achieve a fully responsive design that works seamlessly across desktops, tablets, and mobile devices.
- All external communication must be secured using the HTTPS protocol to protect sensitive guest and operational data during transmission.
- Comprehensive audit logging must be implemented for all critical operations—such as user logins, booking changes, employee record modifications, and system configuration updates—to support security monitoring and accountability.
- The primary language of the system interface and documentation must be English, with support for timezone configuration to handle bookings accurately across different regions.
- The application must adhere to Laravel's coding conventions and PSR standards to promote clean, readable, and maintainable code.
- No third-party payment gateway integration is required in this version; payment processing will be recorded manually by front desk staff.

These constraints help keep the project focused, leverage proven tools and packages, and result in a professional, secure, and extensible system that meets both academic standards and real-world usability needs.

2.7 Assumptions and Dependencies

The design and development of the Heaven Hotel Management System rely on several reasonable assumptions about the operating context, user behavior, and technical environment. These assumptions help define the boundaries of the system and guide decisions during implementation. Additionally, the project depends on specific external tools, frameworks, and libraries that form the foundation of the application.

Assumptions

- Users of the system—both guests and hotel staff—possess at least basic computer literacy and are comfortable using web browsers for everyday tasks such as filling forms or navigating menus.
- A stable internet connection will be available for online guest bookings and for staff access across different devices; while core functions can operate over a local network, full functionality (especially guest-facing features) requires reliable connectivity.
- The database server and hosting environment will remain consistently available during normal hotel operating hours, with standard backup procedures handled by the hotel's IT support.
- Third-party payment gateway integrations are not required for this version of the system; payments will be processed and recorded manually by front desk staff, with electronic integration considered for future enhancements.
- The hotel operates in a single timezone or has consistent policies for handling date/time across bookings, allowing the system's configurable timezone setting to manage any minor variations.
- Users will follow standard security practices, such as not sharing login credentials, to complement the system's built-in protections.

Dependencies

- Laravel Framework version 12.0 or higher, which provides the core structure, routing, authentication, and ORM capabilities.
- Spatie Laravel Permission package for efficient implementation of role-based access control and permission management.
- MySQL version 8.0 or higher as the relational database management system.
- PHP version 8.2 or higher for server-side execution.
- Composer as the dependency manager for installing and updating PHP packages.
- Node.js version 18 or higher along with NPM for compiling frontend assets, including Bootstrap and any custom JavaScript.
- Bootstrap 5 for responsive styling and UI components.
- Standard web protocols (HTTP/HTTPS) and modern browsers for client-side rendering.

These assumptions and dependencies ensure the system remains practical, secure, and aligned with the project's scope and timeline.

CHAPTER THREE - SYSTEM REQUIREMENT

3.1 Functional Requirements

These functional requirements collectively ensure the system delivers a complete, user-friendly solution that covers the essential operations of a modern hotel.

3.1.1 Guest Booking Management

The system shall enable guests to perform self-service booking tasks efficiently

- The system shall allow guests to search for available rooms by specifying check-in and check-out dates, room type, number of adults and children, and other preferences.
- The system shall display detailed room information including photos, amenities, price per night, and current availability.
- The system shall allow registered guests to create new bookings by providing personal details, selecting an available room, and confirming the reservation.
- The system shall allow guests to view their booking history, including current, upcoming, and past bookings with full details and status.
- The system shall permit guests to cancel bookings that are in pending or confirmed status, subject to hotel cancellation policies, and send appropriate confirmation notifications.

3.1.2 Front Desk Operations

The system shall support front desk staff in managing the guest lifecycle from arrival to departure

- The system shall provide a real-time room availability calendar view for front desk staff to see current and future occupancy at a glance.
- The system shall enable staff to create walk-in bookings for guests arriving without prior reservation.
- The system shall allow staff to confirm pending online bookings and assign specific rooms.
- The system shall support the check-in process by updating booking status, assigning room keys (virtually or via record), and marking the room as occupied.
- The system shall support the check-out process by calculating final charges, generating invoices, recording payments, and marking the room as ready for cleaning.
- The system shall allow staff to update room status manually (e.g., available, occupied, out-of-service) when needed.

3.1.3 Housekeeping Management

The system shall streamline housekeeping workflows to ensure rooms are cleaned promptly

- The system shall automatically generate and display cleaning task lists for housekeepers based on checked-out rooms and scheduled turnovers.
- The system shall allow housekeepers to update the cleaning status of assigned rooms (e.g., in progress, cleaned, inspected).
- The system shall enable housekeepers or supervisors to mark rooms as ready for occupancy once cleaning and inspection are complete.

3.1.4 Maintenance Management

The system shall facilitate quick reporting and resolution of facility issues

- The system shall allow authorized staff to report maintenance issues for specific rooms or common areas, including description, priority, and photos if available.
- The system shall display assigned maintenance tasks to relevant personnel with details and status tracking.
- The system shall permit maintenance staff to update issue status (open, in-progress, resolved) and record resolution notes and timestamps.

3.1.5 HR Management

The system shall provide tools for managing hotel employees effectively

- The system shall support the creation, updating, viewing, and deactivation of employee records, including personal information, position, salary, and hire date.
- The system shall enable HR managers to create and manage shift schedules for employees.
- The system shall allow employees or supervisors to record daily attendance (clock-in and clock-out times).
- The system shall generate attendance and payroll-related reports based on recorded data.

3.1.6 Administrative Functions

The system shall offer administrators full control over configuration and reporting

- The system shall provide CRUD (Create, Read, Update, Delete) operations for room types and individual rooms, including setting prices, amenities, and status.
- The system shall allow administrators to generate various reports, such as occupancy rates, revenue summaries, and employee performance.
- The system shall maintain comprehensive audit logs of critical actions performed by all users.
- The system shall enable administrators to configure system-wide settings, such as hotel details, tax rates, and notification preferences.

3.1.7 Authentication and Authorization

The system shall ensure secure and appropriate access for all users

- The system shall provide user registration (for guests and staff) and secure login functionality.
- The system shall enforce role-based access control using defined roles (Guest, Front Desk Staff, Housekeeper, Maintenance, HR Manager, Administrator).
- The system shall support password reset functionality via email.
- The system shall maintain secure user sessions with automatic timeout after inactivity.

3.2 Use Case Model

A) UseCase Diagram

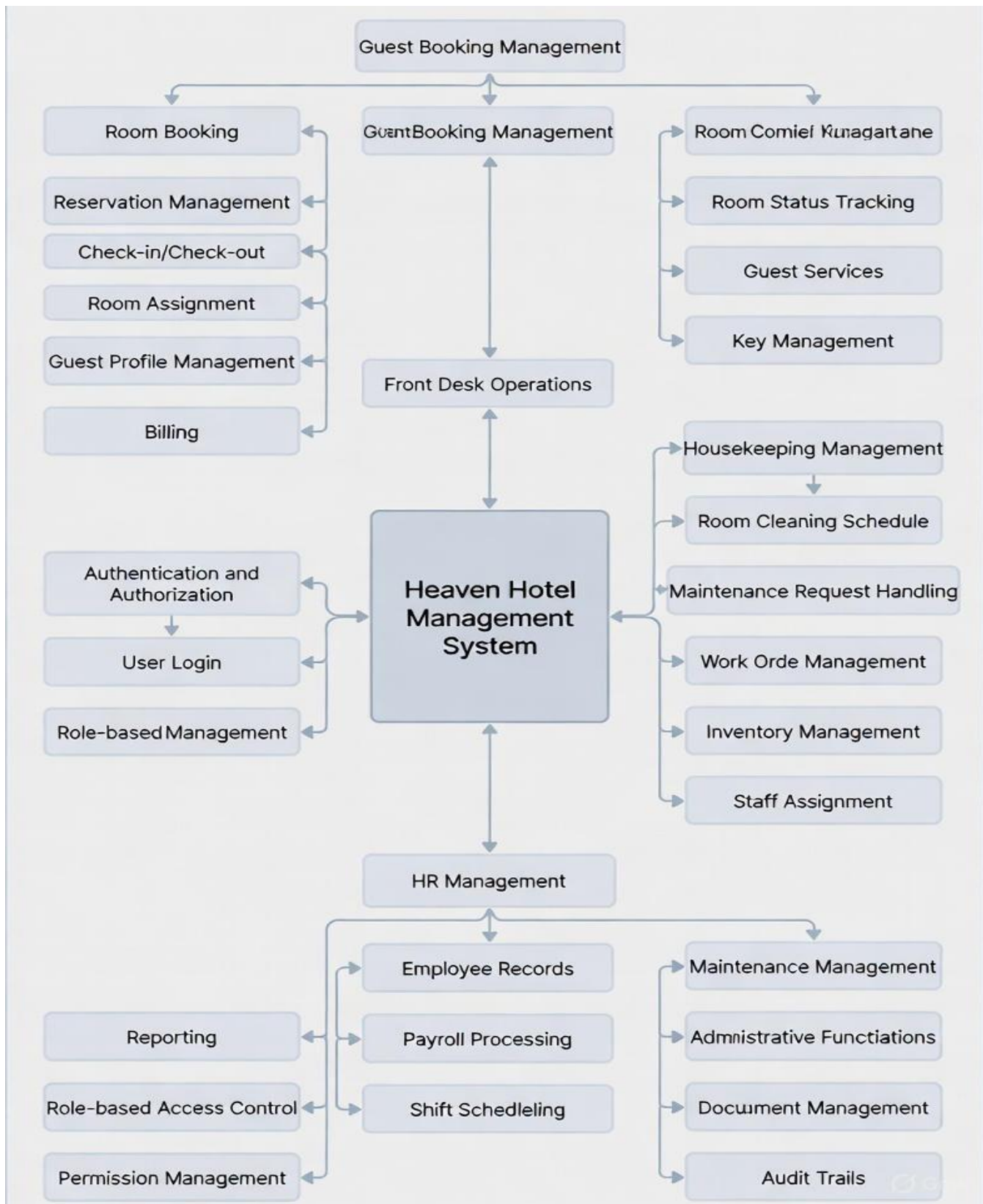


Figure 2 - use case diagram

B) UseCase Description

The following detailed use case descriptions cover the primary interactions for each actor in the Heaven Hotel Management System. Each description includes the actor, preconditions, main flow, alternative flows, and postconditions to provide a clear understanding of the system's behavior.

i. Guest

Housekeepers are internal staff responsible for maintaining room cleanliness and readiness. Their access is limited to housekeeping-related functions to keep the interface simple and focused on daily tasks. The system is designed for mobile-friendly use, allowing updates from tablets or phones while on the floor. All interactions require login with housekeeper privileges.

Use Case - Register as a New Guest

Preconditions - The user is not logged in and has access to a valid email address.

Main Flow

1. The guest navigates to the registration page from the homepage or booking search page.
2. The guest enters personal details - full name, email address, phone number, and a secure password.
3. The system validates the input (unique email, strong password).
4. The system creates a new user account with the Guest role and sends a verification email.
5. The guest clicks the verification link to activate the account.
6. The system logs the guest in automatically or redirects to the login page.

Alternative Flows

- If the email is already registered, the system prompts the guest to log in or reset the password.
- If verification email fails to send, the system displays an error and allows retry.
- **Postconditions** - A new verified guest account is created, and the guest is authenticated.

Use Case - Login to the System

Preconditions - The guest has a registered and verified account.

Main Flow

1. The guest navigates to the login page.
2. The guest enters email and password.
3. The system authenticates the credentials and verifies the Guest role.
4. The system redirects the guest to their personalized dashboard or the booking search page.

Alternative Flows

- If credentials are invalid, the system displays an error and allows retries (with lockout after 5 failed attempts).
- If the guest selects "Forgot Password," the system initiates password reset (see separate use case).
- **Postconditions** - The guest is logged in with a secure session and can access guest-specific features.

Use Case - Search for Available Rooms

Preconditions - None (can be performed without login, but login is required to proceed to booking).

Main Flow

1. The guest enters search criteria - check-in date, check-out date, number of adults, number of children, and optional room type preference.
2. The system queries available rooms and displays a list with details - room type, photos, amenities, price per night, and total cost.
3. The guest can filter or sort results (e.g., by price or amenities).
4. The guest selects a room to view full details.

Alternative Flows

- If no rooms match the criteria, the system suggests alternative dates or room types and displays a friendly message.
- If dates are invalid (e.g., check-out before check-in), the system shows validation errors.
- **Postconditions** - The guest has a clear view of available options to make an informed decision.

Use Case - Book a Room

Preconditions - The guest is logged in, and a room search has returned available options.

Main Flow

1. From the search results, the guest selects an available room and desired dates.
2. The system displays a booking form pre-filled with room and date details.
3. The guest enters or confirms personal information, special requests (e.g., extra bed), and agrees to terms.
4. The system calculates the total cost (including taxes) and validates availability one final time.
5. The guest confirms the booking.
6. The system creates the booking record with "Confirmed" status, reserves the room, and sends a confirmation email with booking details and reference number.

Alternative Flows

- If the room becomes unavailable during form submission, the system notifies the guest and returns to search results.
- If the guest is not logged in, the system redirects to login/registration before proceeding.
- **Postconditions** - A confirmed booking is recorded, room availability is updated, and the guest receives confirmation.

Use Case - View Booking History and Details

Preconditions - The guest is logged in and has at least one existing booking.

Main Flow

1. The guest navigates to the "My Bookings" section.

2. The system displays a list of all bookings (upcoming, current, past) with summary - booking reference, dates, room type, status, and total paid.
3. The guest selects a booking to view full details, including invoice breakdown and any special requests.

Alternative Flows

- If no bookings exist, the system displays a message encouraging the guest to make a new reservation.
- **Postconditions** - The guest has complete visibility into their reservation history.

Use Case - Cancel a Booking

Preconditions - The guest is logged in, has an existing booking in "Pending" or "Confirmed" status, and cancellation is within hotel policy (e.g., before check-in date).

Main Flow

1. The guest views their booking history and selects an eligible booking.
2. The guest chooses the cancel option and confirms the action.
3. The system applies any cancellation policy (e.g., refund rules), updates the booking status to "Cancelled," and releases the room.
4. The system sends a cancellation confirmation email with details.

Alternative Flows

- If the booking is not eligible for cancellation (e.g., past check-in or checked-in), the system displays a message explaining the policy.
- If a refund is applicable, the system notes it (actual processing handled offline).
- **Postconditions** - The booking is cancelled, the room is made available again, and the guest is notified.

Use Case - Reset Forgotten Password

Preconditions - The guest has a registered email but cannot recall the password.

Main Flow

1. From the login page, the guest selects "Forgot Password."
2. The guest enters their registered email address.
3. The system sends a password reset link to the email.
4. The guest clicks the link and enters a new password.
5. The system updates the password and logs the guest in.

Alternative Flows

- If the email is not registered, the system informs the guest without revealing existence.
- **Postconditions** - The guest regains access with a new secure password.

ii. Frontdesk

The following use case descriptions focus exclusively on the interactions performed by the Front Desk Staff actor. Front desk staff are internal hotel employees who manage guest arrivals, departures, and related operations during peak hours. Their workflows emphasize speed and accuracy to ensure smooth guest experiences. All interactions require login with front desk privileges.

Use Case - Login to the System

Preconditions - The staff member has a registered account with front desk role assigned.

Main Flow

1. The staff navigates to the staff login portal.
2. The staff enters their email and password.
3. The system authenticates credentials and verifies the front desk role.
4. The system redirects to the front desk dashboard, displaying real-time room availability and pending tasks.

Alternative Flows

- If credentials are invalid, the system shows an error and limits retries to prevent brute-force attacks.
 - If the account is locked due to inactivity or policy violation, the system notifies the administrator.
- Postconditions** - A secure session is established, granting access to front desk-specific modules.

Use Case - View Room Availability Calendar

Preconditions - The staff is logged in with front desk privileges.

Main Flow

1. The staff accesses the dashboard's room availability view.
2. The system displays a calendar or grid showing room statuses (available, occupied, cleaning, maintenance) for the current and upcoming weeks.
3. The staff can filter by date range, room type, or status to focus on specific needs.
4. The system highlights conflicts or low availability to aid quick decision-making.

Alternative Flows

- If no data is loaded (e.g., database sync issue), the system shows a loading error and retries automatically.
- **Postconditions** - The staff has an up-to-date overview of room occupancy for efficient planning.

Use Case - Create Walk-in Booking

Preconditions - The staff is logged in, and at least one room is available.

Main Flow

1. The staff selects "New Walk-in Booking" from the dashboard.

2. The staff enters guest details (name, contact info, stay dates) and selects an available room from the list.
3. The system validates availability and creates a booking with "Walk-in" status.
4. The system assigns the room, updates occupancy, and generates a temporary booking reference.

Alternative Flows

- If no rooms are available, the system suggests alternatives like waitlisting or upgrades.
- For groups, the system allows multi-room selection with bundled pricing.
- **Postconditions** - A new booking is recorded, and the room is marked as occupied.

Use Case - Confirm Pending Booking

Preconditions - The staff is logged in, and a pending online booking exists for the guest.

Main Flow

1. The staff searches for the pending booking using guest name, email, or reference number.
2. The system displays booking details, including requested dates and room preferences.
3. The staff verifies guest identity (e.g., via ID upload or in-person check) and assigns an available room.
4. The system updates the status to "Confirmed," notifies the guest via email, and adjusts room availability.

Alternative Flows

- If the guest does not arrive, the staff can mark it as "No-Show" after a grace period.
- If room preferences cannot be met, the staff offers alternatives with guest approval.
- **Postconditions** - The booking is confirmed, and the guest is ready for check-in.

Use Case - Process Guest Check-in

Preconditions - The staff is logged in, and the guest has a confirmed booking for the current date.

Main Flow

1. The staff searches for the guest's booking.
2. The system displays full booking details, including any special requests.
3. The staff verifies guest identity (e.g., ID scan or document upload) and collects any required deposits.
4. The system updates the booking status to "Checked-in," assigns digital/physical keys, marks the room as occupied, and generates a welcome packet or key card.
5. The system notifies housekeeping of the occupancy change.

Alternative Flows

- For early check-in, if the room is ready, the system allows it; otherwise, it offers lounge access.
- If overbooking occurs, the system prioritizes based on booking time and offers upgrades.
- **Postconditions** - The guest is checked in, room status is updated, and billing begins accruing.

Use Case - Process Guest Check-out

Preconditions - The staff is logged in, and the guest's booking is in "Checked-in" status on or after the check-out date.

Main Flow

1. The staff searches for the guest's booking.
2. The system calculates the final bill, including room charges, extras (e.g., minibar), and taxes.
3. The staff reviews the invoice with the guest, processes payment (cash, card, or note pending), and obtains confirmation.
4. The system updates the booking status to "Checked-out," marks the room as "Dirty/Ready for Cleaning," and releases keys.
5. The system sends a post-stay survey email and notifies housekeeping.

Alternative Flows

- For late check-out requests, the staff checks availability and approves if possible, adjusting charges.
- If payment is incomplete, the system flags the booking for follow-up.
- **Postconditions** - The guest is checked out, invoice is finalized, and the room is prepared for turnover.

Use Case - Update Room Status Manually

Preconditions - The staff is logged in, and a room's status needs adjustment (e.g., due to guest extension or issue).

Main Flow

1. The staff selects the room from the availability calendar.
2. The system displays current status and history.
3. The staff chooses a new status (e.g., available, occupied, out-of-service) and adds notes (e.g., reason for change).
4. The system saves the update, notifies affected modules (e.g., housekeeping), and refreshes the calendar.

Alternative Flows

- If the change conflicts with a booking, the system warns and requires override confirmation.
- **Postconditions** - Room status is accurately reflected across the system.

Use Case - Generate and Print Invoice

Preconditions - The staff is logged in, and a booking is ready for billing (e.g., during check-out).

Main Flow

1. The staff selects the booking and chooses "Generate Invoice."
2. The system compiles charges (room nights, taxes, incidentals) and applies any discounts.
3. The staff reviews the invoice preview.
4. The system generates a printable PDF or on-screen version and records payment details.

Alternative Flows

- For disputes, the staff can adjust line items with approval notes.
- **Postconditions** - An invoice is created and linked to the booking for records.

Use Case - Handle Guest Inquiries or Complaints

Preconditions - The staff is logged in, and a guest inquiry arises (e.g., via phone, email, or in-person).

Main Flow

1. The staff logs the inquiry in the guest's booking record or creates a new ticket.
2. The system categorizes it (e.g., billing issue, room request) and assigns priority.
3. The staff resolves simple issues directly (e.g., extra amenities) or escalates to maintenance/HR.
4. The system updates the status and notifies the guest of resolution.

Alternative Flows

- For complex complaints, the system integrates with feedback logs for follow-up.
- **Postconditions** - The inquiry is documented and resolved, improving guest satisfaction tracking.

Use Case - Logout from the System

Preconditions - The staff is logged in.

Main Flow

1. The staff selects "Logout" from the menu.
2. The system ends the session, clears cookies, and redirects to the login page.

Alternative Flows

- Auto-logout occurs after 15 minutes of inactivity, with a warning prompt.
- **Postconditions** - The session is securely terminated.

iii. Housekeeper

The following use case descriptions focus exclusively on the interactions performed by the Housekeeper actor. Housekeepers are internal staff responsible for maintaining room cleanliness and readiness. Their access is limited to housekeeping-related functions to keep the interface simple and focused on daily tasks. The system is designed for mobile-friendly use, allowing updates from tablets or phones while on the floor. All interactions require login with housekeeper privileges.

Use Case - Login to the System

Preconditions - The housekeeper has a registered account with housekeeping role assigned.

Main Flow

1. The housekeeper opens the login page on a shared hotel tablet or personal device.
2. The housekeeper enters their assigned username/email and password.
3. The system authenticates credentials and confirms the housekeeping role.
4. The system redirects to a simple housekeeping dashboard showing today's assigned rooms and tasks.

Alternative Flows

- If credentials are incorrect, the system displays an error and allows limited retries before temporary logout.
- If the account requires password change (e.g., first login), the system prompts for a new password.
- **Postconditions** - A secure session is established, granting access only to housekeeping features.

Use Case - View Assigned Cleaning Tasks

Preconditions - The housekeeper is logged in, and tasks have been generated (e.g., from check-outs).

Main Flow

1. The housekeeper opens the dashboard or task list section.
2. The system displays a prioritized list of rooms needing cleaning, including room number, type (e.g., standard checkout, deep clean), due time, and any special notes (e.g., guest preferences).
3. The housekeeper can filter by floor, status, or urgency.
4. The system highlights overdue or high-priority rooms (e.g., for early arrivals).

Alternative Flows

- If no tasks are assigned, the system shows a message like "No rooms assigned today" and suggests checking with supervisor.
- If connectivity is lost, the system caches the last viewed list for offline reference (syncs when online).
- **Postconditions** - The housekeeper has a clear, up-to-date task list to plan their work.

Use Case - Start Cleaning a Room

Preconditions - The housekeeper is logged in and has rooms assigned.

Main Flow

1. The housekeeper selects a room from the task list.
2. The system displays room details and allows marking the task as "Cleaning in Progress."
3. The housekeeper confirms start time (auto-recorded if desired).
4. The system updates the room status to "Being Cleaned" and notifies front desk if needed.

Alternative Flows

- If the room is not yet vacant (e.g., guest extended stay), the system prevents starting and shows a warning.
- **Postconditions** - The cleaning process is officially started, and status is visible system-wide.

Use Case - Report Issues During Cleaning

Preconditions - The housekeeper is logged in and is working on or has access to a room.

Main Flow

1. While viewing a room's task, the housekeeper selects "Report Issue."

2. The housekeeper chooses issue type (e.g., damaged furniture, missing items, maintenance needed), adds description, priority, and optional photos.
3. The system creates a maintenance request linked to the room and notifies maintenance staff.
4. The system adjusts the room status (e.g., "Out of Service" if severe).

Alternative Flows

- For minor issues (e.g., low supplies), the system logs it for inventory tracking without full maintenance ticket.
- **Postconditions** - The issue is documented, routed correctly, and room availability is updated if necessary.

Use Case - Complete Room Cleaning

Preconditions - The housekeeper is logged in and has started cleaning a room.

Main Flow

1. The housekeeper returns to the room's task page after finishing cleaning.
2. The housekeeper marks the room as "Cleaned," adds any notes (e.g., supplies used, special conditions), and optionally requests inspection.
3. The system updates the status to "Cleaned - Awaiting Inspection" or directly to "Ready" if no inspection is required.
4. The system notifies front desk that the room is nearly ready for new guests.

Alternative Flows

- If the housekeeper identifies the room needs more work, they can revert to "In Progress" with notes.
- **Postconditions** - The room cleaning is recorded as complete, advancing it toward readiness.

Use Case - Mark Room as Ready for Occupancy

Preconditions - The housekeeper is logged in, cleaning is complete, and any inspection is passed.

Main Flow

1. The housekeeper (or supervisor) selects the cleaned room.
2. The system shows a final checklist (e.g., linens changed, amenities stocked).
3. The housekeeper confirms all items are complete and marks the room as "Ready for Occupancy."
4. The system updates the room status to "Available," removes it from cleaning lists, and notifies front desk for immediate assignment.

Alternative Flows

- If inspection fails, the system reverts to "Needs Re-cleaning" with feedback notes.
- **Postconditions** - The room is fully ready and visible for new bookings or assignments.

Use Case - View Cleaning History for a Room

Preconditions - The housekeeper is logged in and needs to check past cleaning records.

Main Flow

1. The housekeeper searches for a specific room number.
2. The system displays recent cleaning history - dates, assigned housekeeper, notes, and any reported issues.
3. The housekeeper reviews details for context (e.g., recurring problems).

Alternative Flows

- Limited to recent history (e.g., last 30 days) for performance; older records available to supervisors.
- **Postconditions** - The housekeeper has context for informed cleaning.

Use Case - Logout from the System

Preconditions - The housekeeper is logged in.

Main Flow

1. The housekeeper selects "Logout" from the menu.
2. The system ends the session securely and returns to the login screen.

Alternative Flows

- Automatic logout after 10 minutes of inactivity for shared devices.
- **Postconditions** - Session is terminated, protecting privacy on shared tablets.

iv. Maintenance Staff Management

The following use case descriptions focus exclusively on the interactions performed by the Maintenance Staff actor (also referred to as Maintenance Personnel). Maintenance staff handle repairs and technical issues in rooms and facilities. Their interface is straightforward, emphasizing task assignment, updates, and resolution tracking. Access is restricted to maintenance functions, with mobile support for on-site work. All interactions require login with maintenance privileges.

Use Case - Login to the System

Preconditions - The staff member has a registered account with maintenance role assigned.

Main Flow

1. The maintenance staff opens the login page on a tablet or mobile device.
2. The staff enters their username/email and password.
3. The system authenticates credentials and confirms the maintenance role.
4. The system redirects to a maintenance dashboard showing assigned tasks and open issues.

Alternative Flows

- If login fails multiple times, the system temporarily locks the account and notifies the administrator.
- First-time login prompts for password change.
- **Postconditions** - Secure access is granted to maintenance-specific features.

Use Case - View Assigned Maintenance Tasks

Preconditions - The staff is logged in, and tasks have been assigned (from reports by front desk, housekeepers, or guests).

Main Flow

1. The staff accesses the task list on the dashboard.
2. The system displays prioritized open issues, including room number, description, reported date, priority (low/medium/high/urgent), and reporter notes/photos.
3. The staff can sort/filter by priority, location (e.g., floor), or status.
4. The system flags overdue tasks based on the 24-hour resolution rule.

Alternative Flows

- If no tasks are assigned, the system shows "No current tasks" and allows browsing all open issues.
- **Postconditions** - The staff has a clear overview to prioritize work.

Use Case - Accept/Start a Maintenance Task

Preconditions - The staff is logged in and views an unassigned or assigned task.

Main Flow

1. The staff selects a task from the list.
2. The system shows full details, including any attached photos or history.
3. The staff marks the task as "In Progress" and adds initial notes (e.g., tools needed).
4. The system updates the status, records start time, and notifies relevant staff (e.g., front desk) that work has begun.

Alternative Flows

- If the task requires parts, the staff notes it for inventory tracking.
- **Postconditions** - The task is actively being handled, and status is visible system-wide.

Use Case - Update Task Progress

Preconditions - The staff is logged in and has an "In Progress" task.

Main Flow

1. The staff opens the active task.
2. The staff adds progress notes, uploads photos of work done, or updates required materials.
3. The system saves the update and timestamps it.
4. For ongoing issues, the staff can adjust priority or estimated completion time.

Alternative Flows

- If additional issues are discovered, the staff creates a linked sub-task.
- **Postconditions** - Task history is updated for transparency.

Use Case - Resolve and Close a Maintenance Issue

Preconditions - The staff is logged in, and repair work is complete on an "In Progress" task.

Main Flow

1. The staff selects the task and marks it as "Resolved."
2. The staff enters resolution details - what was fixed, parts used, time taken, and final photos.
3. The system validates completion (e.g., requires notes), updates status to "Resolved," records resolution timestamp, and calculates time taken.
4. The system notifies front desk/housekeeping that the room/facility is ready, and updates room status if it was "Out of Service."

Alternative Flows

- If the issue cannot be fully resolved (e.g., needs external vendor), the staff escalates with notes, changing status to "Pending External."
- **Postconditions** - The issue is closed, room availability is restored if applicable, and full audit trail is maintained.

Use Case - Report a New Maintenance Issue (Self-Reported)

Preconditions - The staff is logged in and discovers an issue during routine checks.

Main Flow

1. The staff selects "Report New Issue."
2. The staff enters details - location/room, description, priority, and photos.
3. The system creates a new task, assigns it to the staff (or team), and adds it to the list.

Alternative Flows

- For proactive maintenance (e.g., scheduled checks), the system links it to preventive logs.
- **Postconditions** - A new tracked issue is created for follow-up.

Use Case - View Maintenance History for a Room/Facility

Preconditions - The staff is logged in and needs historical context.

Main Flow

1. The staff searches by room number or facility area.
2. The system displays past issues - dates, descriptions, resolutions, and recurring patterns.
3. The staff reviews for trends (e.g., frequent AC failures).

Alternative Flows

- Export history for reporting to administrator.
- **Postconditions** - The staff is informed for better diagnostics.

Use Case - Logout from the System

Preconditions - The staff is logged in.

Main Flow

1. The staff selects "Logout."
2. The system ends the session and returns to login.

Alternative Flows

- Auto-logout after inactivity for security on shared devices.
- **Postconditions** - Session is securely terminated.

v. HR Manager

The following use case descriptions focus exclusively on the interactions performed by the **HR Manager** actor. HR managers are responsible for overseeing employee-related functions, including records management, shift scheduling, and attendance tracking. Their interface provides tools for administrative HR tasks while maintaining data security and privacy. All interactions require login with HR manager privileges.

Use Case - Login to the System

Preconditions - The HR manager has a registered account with HR manager role assigned.

Main Flow

1. The HR manager navigates to the staff login portal.
2. The HR manager enters their email and password.
3. The system authenticates credentials and verifies the HR manager role.
4. The system redirects to the HR dashboard, displaying employee overview, pending attendance issues, and shift schedules.

Alternative Flows

- If credentials are invalid, the system displays an error and allows retries with progressive delays.
- Forgotten password triggers the standard reset process via email.
- **Postconditions** - Secure access is granted to HR-specific modules.

Use Case - Create New Employee Record

Preconditions - The HR manager is logged in.

Main Flow

1. The HR manager selects "Add New Employee" from the employee management section.
2. The HR manager fills in details - full name, contact information, position, hire date, salary, department, and any required documents (e.g., ID upload).
3. The system validates required fields and uniqueness (e.g., email).
4. The system creates the employee record, generates a system account (if needed), and marks the employee as active.

5. The system sends a welcome email with login credentials (if applicable).

Alternative Flows

- If mandatory fields are missing, the system highlights errors and prevents submission.
- For bulk imports, the system allows CSV upload with validation.
- **Postconditions** - A new complete employee record is added to the database.

Use Case - Update or Deactivate Employee Record

Preconditions - The HR manager is logged in, and the employee record exists.

Main Flow

1. The HR manager searches for the employee by name, ID, or position.
2. The system displays the full employee profile.
3. The HR manager edits details (e.g., promotion, salary change, contact update) or selects "Deactivate" for termination.
4. For deactivation, the HR manager provides reason and effective date.
5. The system saves changes, updates status (active/inactive), and logs the action in audit trail.

Alternative Flows

- If deactivating an employee with active shifts, the system warns and requires reassignment.
- **Postconditions** - Employee record is updated or deactivated, affecting scheduling and access.

Use Case - View Employee Records

Preconditions - The HR manager is logged in.

Main Flow

1. The HR manager navigates to the employee list or searches by criteria (name, position, status).
2. The system displays a paginated list with summary information.
3. The HR manager selects an employee to view detailed profile, including history of changes.
4. The system allows exporting the list or individual records if needed.

Alternative Flows

- Filters for active/inactive employees or by department.
- **Postconditions** - The HR manager has access to accurate employee information.

Use Case - Create and Manage Shift Schedules

Preconditions - The HR manager is logged in, and active employees exist.

Main Flow

1. The HR manager selects "Shift Scheduling" and chooses a date range (e.g., weekly/monthly).
2. The system displays a calendar or grid view of existing shifts.
3. The HR manager creates new shifts (name, start/end time) and assigns eligible active employees.

4. The system checks for conflicts (e.g., overlapping shifts) and validates against business rules.
5. The HR manager publishes the schedule.
6. The system notifies assigned employees via email or in-system alert.

Alternative Flows

- If an employee is unavailable (e.g., leave request), the system blocks assignment.
- Drag-and-drop editing for quick adjustments.
- **Postconditions** - Shift schedule is finalized and visible to relevant staff.

Use Case - Record and Track Employee Attendance

Preconditions - The HR manager is logged in, and shifts are scheduled.

Main Flow

1. The HR manager (or employee) accesses the attendance module for a specific date.
2. The system displays expected attendees based on shifts.
3. Clock-in/out times are recorded manually or via timestamp button.
4. The HR manager reviews and approves/adjusts entries (e.g., late arrivals, absences).
5. The system calculates hours worked and flags irregularities (e.g., overtime, missing clock-out).

Alternative Flows

- For absences, the HR manager adds notes (sick leave, vacation).
- **Postconditions** - Attendance is accurately recorded for payroll and reporting.

Use Case - Generate Attendance and Payroll Reports

Preconditions - The HR manager is logged in, and attendance data exists.

Main Flow

1. The HR manager selects "Reports" and chooses type (e.g., monthly attendance, payroll summary).
2. The HR manager sets parameters (date range, department, employee).
3. The system compiles data - total hours, overtime, absences, and calculated pay (based on salary rates).
4. The system displays the report on-screen and allows export to PDF/Excel.

Alternative Flows

- Custom reports for performance reviews (e.g., punctuality trends).
- **Postconditions** - HR has actionable insights for payroll processing and management decisions.

Use Case - Logout from the System

Preconditions - The HR manager is logged in.

Main Flow

1. The HR manager selects "Logout."

2. The system ends the session securely and returns to login.

Alternative Flows

- Auto-logout after prolonged inactivity.
- **Postconditions** - Session is terminated safely.

vi. Administrative Management

The following use case descriptions focus exclusively on the interactions performed by the **Administrator** actor. Administrators have the highest level of access in the Heaven Hotel Management System, responsible for configuring the system, managing core data (such as rooms and room types), generating comprehensive reports, and monitoring security through audit logs. Their interface provides powerful tools while enforcing careful controls to protect system integrity. All interactions require login with administrator privileges.

Use Case - Login to the System

Preconditions - The administrator has a registered account with full administrator role assigned.

Main Flow

1. The administrator navigates to the secure admin login portal.
2. The administrator enters their email and password.
3. The system authenticates credentials, verifies the administrator role, and checks for any additional security (e.g., two-factor if enabled).
4. The system redirects to the admin dashboard, displaying system overview, recent audit logs, and quick links to configuration tools.

Alternative Flows

- If login fails repeatedly, the system locks the account temporarily and alerts via email.
- Password reset follows a secure process with admin-only approval if needed.
- **Postconditions** - Full administrative access is granted securely.

Use Case - Manage Room Types (CRUD Operations)

Preconditions - The administrator is logged in.

Main Flow

1. The administrator navigates to the "Room Types" management section.
2. To create - The administrator selects "Add New Room Type," enters name, description, price per night, max occupancy, amenities, and uploads photos.
3. To update - The administrator selects an existing type and modifies details (e.g., seasonal price change).
4. To delete - The administrator selects a type with no linked rooms and confirms deletion.
5. The system validates changes (e.g., price > 0) and applies them immediately.

Alternative Flows

- If attempting to delete a type linked to rooms, the system prevents it & suggests deactivation instead.

- Bulk updates (e.g., price increase across types) are supported via import.
- **Postconditions** - Room type catalog is updated, affecting availability and pricing system-wide.

Use Case - Manage Individual Rooms (CRUD Operations)

Preconditions - The administrator is logged in, and room types exist.

Main Flow

1. The administrator accesses the "Rooms" management section.
2. To add - The administrator enters room number, floor, assigns a room type, and sets initial status (available/out-of-service).
3. To update - The administrator modifies details (e.g., change type, mark temporarily out-of-service for renovation).
4. To delete/deactivate - The administrator removes or deactivates a room no longer in use.
5. The system ensures uniqueness (e.g., no duplicate room numbers) and updates availability.

Alternative Flows

- If a room has active bookings, deletion is blocked with a warning.
- **Postconditions** - The hotel's room inventory accurately reflects physical assets.

Use Case - View System Audit Logs

Preconditions - The administrator is logged in.

Main Flow

1. The administrator navigates to the "Audit Logs" section.
2. The system displays a searchable, filterable list of all critical actions (e.g., logins, data changes, role assignments) with timestamps, user, and details.
3. The administrator applies filters (date range, user, action type) to investigate events.
4. The system allows export of logs for compliance or review.

Alternative Flows

- For suspicious activity, the administrator can flag entries for follow-up.
- **Postconditions** - The administrator has full visibility into system activity for security monitoring.

Use Case - Generate Business Reports

Preconditions - The administrator is logged in, and operational data exists.

Main Flow

1. The administrator selects "Reports" and chooses a type (e.g., occupancy rate, revenue summary, employee performance, booking trends).
2. The administrator sets parameters (date range, department, room type).
3. The system compiles data, performs calculations (e.g., average stay length, revenue per room), and displays an interactive report with charts and tables.
4. The administrator exports the report (PDF, Excel) or schedules recurring delivery.

Alternative Flows

- Custom reports can be built using predefined templates.
- **Postconditions** - Management gains actionable insights for decision-making.

Use Case - Configure System Settings

Preconditions - The administrator is logged in.

Main Flow

1. The administrator accesses "System Settings."
2. The administrator modifies global parameters - hotel name/address, tax rates, timezone, notification preferences, or backup schedules.
3. The system validates changes (e.g., valid tax percentage) and applies them immediately.
4. Changes are logged in audit trail.

Alternative Flows

- Reverting to defaults is available with confirmation.
- **Postconditions** - System behavior is customized to hotel needs.

Use Case - Manage User Roles and Permissions (Advanced)

Preconditions - The administrator is logged in.

Main Flow

1. The administrator navigates to user management (beyond basic HR).
2. The administrator views all system users, adjusts roles (e.g., promote staff), or revokes access.
3. The system enforces Spatie permissions and logs changes.

Alternative Flows

- Bulk role assignment for new hires.
- **Postconditions** - Access controls remain secure and up-to-date.

Use Case - Logout from the System

Preconditions - The administrator is logged in.

Main Flow

1. The administrator selects "Logout."
2. The system ends the session securely and returns to login.

Alternative Flows

- Forced logout after short inactivity for high-security access.
- **Postconditions** - Sensitive admin session is terminated.

3.3 Data Requirements

3.3.3 Entity-Relationship Diagram

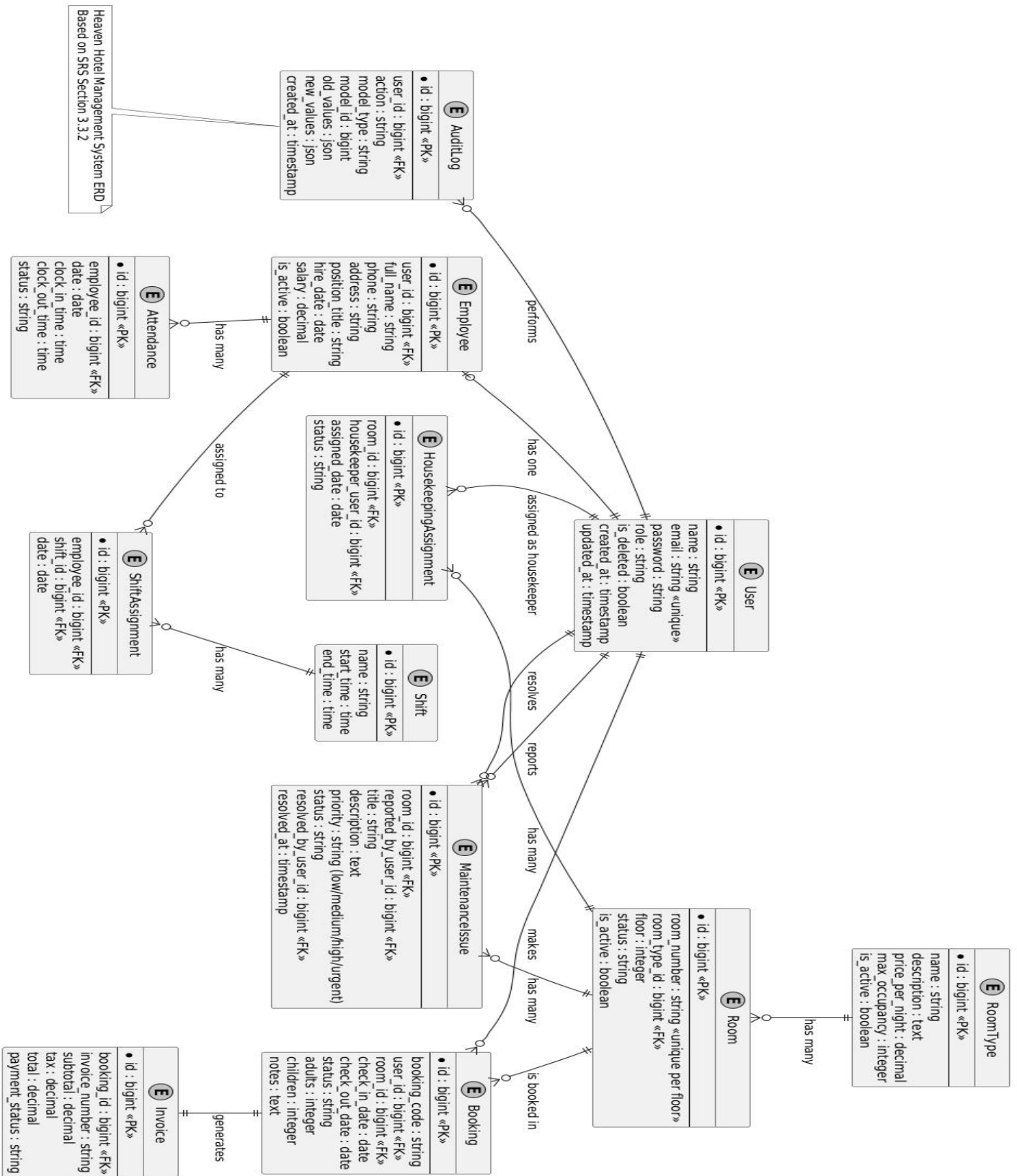


Figure 3 - Entity-Relationship Diagram

3.3.4 Data Validation Rules

The Heaven Hotel Management System enforces strict data validation rules to maintain data integrity, prevent invalid entries, and ensure business consistency. These rules are applied at multiple layers (frontend forms, backend controllers, and database constraints) to catch errors early and protect the system from corrupted or inconsistent data. The validation rules are summarized in the table below for each key entity and attribute.

Table 4: Data Validation Rules

Entity.Attribute	Validation Rules	Rationale / Error Handling
User.name	Required, string, maximum 255 characters	Ensures a valid name is provided. Error: "Full name is required."
User.email	Required, unique across all users, valid email format	Prevents duplicate accounts and ensures reachability. Error: "A valid and unique email is required."
User.password	Required on creation/update, minimum 8 characters, must contain at least one uppercase letter, one lowercase letter, one number, and one special character	Strong password policy for security. Error: "Password must be at least 8 characters and include uppercase, lowercase, number, and special character."
User.role	Required, must be one of: Guest, FrontDesk, Housekeeper, Maintenance, HRManager, Administrator	Enforces role-based access control. Error: "Invalid role selected."
Booking.booking_code	Auto-generated, unique alphanumeric string	Ensures traceability. No user input.
Booking.check_in_date	Required, valid date, must be today or in the future at time of booking	Prevents past check-ins. Error: "Check-in date cannot be in the past."
Booking.check_out_date	Required, valid date, must be strictly after check_in_date	Logical stay duration. Error: "Check-out must be after check-in."
Booking.adults	Required, integer between 1 and RoomType.max_occupancy	Ensures occupancy limits. Error: "Number of adults must be at least 1 and not exceed room capacity."
Booking.children	Integer ≥ 0 , total occupants (adults + children) \leq RoomType.max_occupancy	Safe occupancy. Error: "Total guests exceed room capacity."
Booking.notes	Optional, text, maximum 1000 characters	Limits excessive input.
Room.room_number	Required, string, unique across the entire hotel (or unique per floor if policy allows), alphanumeric format	Prevents duplicates. Error: "Room number already exists."
Room.floor	Required, positive integer	Logical building structure.

Entity.Attribute	Validation Rules	Rationale / Error Handling
		Error: "Floor must be a positive number."
Room.status	Required, must be one of: Available, Occupied, Dirty, BeingCleaned, OutOfService	Controlled state machine. Error: "Invalid room status."
RoomType.name	Required, unique, string	Distinct room categories. Error: "Room type name must be unique."
RoomType.price_per_night	Required, decimal > 0.00, maximum 2 decimal places	Realistic pricing. Error: "Price must be greater than zero."
RoomType.max_occupancy	Required, integer between 1 and 10 (or reasonable hotel limit)	Practical limits. Error: "Maximum occupancy must be between 1 and 10."
Invoice.subtotal	Auto-calculated, decimal ≥ 0.00	No manual override.
Invoice.tax	Auto-calculated based on system setting, decimal ≥ 0.00	Consistent taxation.
Invoice.total	Auto-calculated (subtotal + tax + fees), decimal > 0.00	Final accurate billing.
Invoice.payment_status	Must be one of: Pending, Paid, PartiallyPaid, Refunded	Controlled payment lifecycle.
MaintenanceIssue.title	Required, string, maximum 255 characters	Clear issue identification.
MaintenanceIssue.description	Required, text	Detailed reporting.
MaintenanceIssue.priority	Required, one of: Low, Medium, High, Urgent	Proper escalation. Error: "Select a valid priority level."
Employee.phone	Required, valid phone number format (international or local)	Contactability. Error: "Enter a valid phone number."
Employee.salary	Required, decimal ≥ 0.00	Realistic compensation.
Employee.hire_date	Required, date \leq today	Logical employment history.
Shift.start_time / end_time	Required, valid time format, start_time < end_time	Valid shift duration. Error: "Shift end time must be after start."
Attendance.clock_in_time	Required if clocked in, valid time, within assigned shift hours	Accurate tracking.
Attendance.clock_out_time	Optional (until clock-out), must be after clock_in_time	Prevents backward time.
AuditLog fields	System-generated only (no user input)	Ensures tamper-proof logging.

These rules are enforced as follows:

- **Frontend** - Real-time validation using Laravel Livewire or JavaScript for immediate feedback.
- **Backend** - Laravel validation rules in controllers and form requests.
- **Database** - Constraints (e.g., NOT NULL, UNIQUE, CHECK) where possible.

3.4 External Interface Requirements

3.4.1 Hardware Interface

The system has no direct dependency on specialized hardware beyond standard computing devices:

- **Input Devices:** Standard keyboard, mouse, or touchscreen for data entry by staff and guests.
- **Output Devices:** Computer monitor or mobile screen for displaying the user interface; standard printers (network or local) for printing invoices, booking confirmations, and reports.
- **Network Interface:** Standard network card or Wi-Fi adapter for internet/local network connectivity to access the web server.

No custom hardware drivers or embedded controllers are required.

3.4.2 Software Interface

The system interacts with the following software components:

- **Operating Systems:**
 - Server: Linux (preferred) or Windows Server with PHP support.
 - Client: Any modern OS (Windows 10+, macOS, Linux, Android, iOS) capable of running supported web browsers.
- **Web Browser:** Google Chrome (version 90+), Mozilla Firefox, Apple Safari, Microsoft Edge – full support for HTML5, CSS3, and JavaScript (ES6+).
- **Database:** MySQL 8.0+ for persistent storage via Laravel Eloquent ORM.
- **Web Server:** Apache 2.4+ or Nginx 1.20+ with PHP-FPM.
- **PHP Runtime:** PHP 8.2+ with required extensions (e.g., PDO, OpenSSL, Mbstring).
- **Third-Party Libraries:** Laravel Framework 12, Spatie Laravel Permission package, Bootstrap 5 – all managed via Composer and NPM.

No direct API integration with external systems (e.g., payment gateways, property management systems) is included in this version.

3.4.3 User Interface

The user interface is delivered entirely through a responsive web application:

- Built using Laravel Blade templates, Bootstrap 5, and minimal custom JavaScript (with Livewire/Alpine.js for dynamic components).
- Fully responsive layout supporting desktop (minimum 1024px width), tablet, and mobile devices (down to 320px width).
- Supports touch interactions on mobile devices.
- Adheres to WCAG 2.1 AA accessibility guidelines (e.g., sufficient color contrast, keyboard navigation, ARIA labels).
- Provides clear feedback: loading indicators, success/error messages, and confirmation dialogs.

- Uses progressive enhancement: core functionality works even with JavaScript disabled, though enhanced experience requires it.

3.4.4 Communication Interface

All communication occurs over standard web protocols:

- **HTTP/HTTPS:** All traffic must use HTTPS (enforced via server configuration and Laravel middleware) with valid SSL/TLS certificates.
- **AJAX/JSON:** Dynamic updates (e.g., room availability checks) use AJAX calls returning JSON data.
- **RESTful Principles:** Internal API endpoints follow REST conventions for consistency.
- **Email Notifications:** Outbound emails (booking confirmations, password resets) sent via SMTP or third-party services (configurable).
- **File Uploads:** Secure handling of optional file uploads (e.g., photos in maintenance reports) with size and type restrictions.

3.5 Non-functional Requirements

Non-functional requirements define the quality attributes and constraints that shape how the Heaven Hotel Management System performs, behaves, and is experienced by users. These requirements are essential for ensuring the system is reliable, secure, usable, and maintainable in a real-world hotel environment. They are categorized into performance, usability, security, and other software quality attributes.

3.5.1 Performance Requirements

The system must deliver fast and responsive performance to support busy hotel operations, especially during peak check-in/check-out times.

- The system shall load typical pages (e.g., dashboard, room availability search, booking list) in less than 2 seconds under normal load on a standard broadband connection.
- Database queries for common operations (e.g., searching available rooms, generating reports) shall complete within 500 milliseconds.
- The system shall support up to 1000 concurrent users (including guests browsing online and staff accessing internally) without significant degradation in response time.
- Report generation (e.g., daily occupancy or revenue summary) shall complete within 30 seconds, even for a month's worth of data.
- Real-time updates (e.g., room status changes) shall reflect across all relevant user sessions within 5 seconds using efficient polling or Livewire updates.

These targets ensure the system remains efficient and does not frustrate users during high-pressure periods.

3.5.2 Usability Requirements

The system shall be intuitive and easy to use for users with varying levels of technical expertise, minimizing training time and errors.

- The user interface shall follow a consistent layout, navigation, and terminology across all modules and user roles.
- All forms shall include clear labels, helpful placeholders, inline validation, and descriptive error messages that guide users to correct mistakes.
- Critical actions (e.g., check-out, cancellation) shall require confirmation dialogs to prevent accidental data loss.
- The system shall provide contextual help (tooltips, FAQ links) and searchable knowledge base for common tasks.
- Full keyboard navigation and screen reader compatibility shall be supported to meet WCAG 2.1 AA accessibility standards.
- Color contrast ratios shall meet WCAG AA requirements, and the design shall be fully responsive, providing a seamless experience on desktops, tablets, and smartphones.

Usability testing during development will verify that new staff can learn core functions within one hour of training.

3.5.3 Security Requirements

Security is paramount given the sensitive nature of guest personal data, payment-related records, and internal hotel operations.

- User passwords shall be hashed using bcrypt with strong cost factors, and session data shall be protected against hijacking.
- Role-based access control (via Spatie Laravel Permission) shall be strictly enforced—no user shall access functions or data outside their assigned role.
- The system shall prevent common web vulnerabilities:
 - SQL injection through parameterized queries and Eloquent ORM.
 - Cross-Site Scripting (XSS) via output escaping and content sanitization.
 - Cross-Site Request Forgery (CSRF) through Laravel's built-in token protection on all state-changing requests.
- All external communication shall use HTTPS with valid certificates and HSTS headers.
- Session timeout shall occur after 30 minutes of inactivity, with secure logout clearing all session data.
- Sensitive actions (e.g., user creation, role changes, booking modifications) shall be logged in the audit trail with user ID, timestamp, and before/after values.
- The system shall implement rate limiting on login attempts and sensitive endpoints to mitigate brute-force attacks.
- File uploads (e.g., maintenance photos) shall be scanned, restricted by type/size, and stored outside the web root.

Regular security reviews and dependency updates will maintain protection against emerging threats.

3.5.4 Software Quality Attributes

These attributes ensure the system is robust, maintainable, and suitable for long-term use.

- **Reliability:** The system shall achieve at least 99.5% uptime during normal hotel operating hours, with graceful error handling and no data loss during unexpected failures (e.g., using database transactions for critical operations).
- **Maintainability:** Code shall follow PSR-12 standards, be well-documented with comments and PHP DocBlocks, and structured modularly to allow future developers to understand and modify it efficiently.
- **Scalability:** The architecture shall support horizontal scaling (e.g., multiple web servers behind a load balancer) and database optimization for growing data volumes.
- **Portability:** The system shall run on any server supporting PHP 8.2+, MySQL 8.0+, and standard web servers (Apache/Nginx), across Linux and Windows environments.
- **Testability:** At least 80% unit test coverage for core business logic, with integration tests for key workflows, using PHPUnit and Laravel's testing tools.
- **Availability:** Critical functions (e.g., front desk operations) shall remain accessible even during minor updates, using zero-downtime deployment strategies where possible.
- **Recoverability:** Database backups shall be supported through standard MySQL tools, with transaction logs enabling point-in-time recovery.

These non-functional requirements collectively ensure that the Heaven Hotel Management System is not only functionally complete but also professional, secure, and sustainable for real-world deployment.

APPENDIX A: Group Log

This appendix records the key activities, meetings, decisions, and progress made by the development team throughout the Heaven Hotel Management System project. It serves as a chronological record of our collaboration, challenges encountered, and how we addressed them. Entries are dated and include brief descriptions of contributions from team members.

Date	Activity / Meeting Summary	Participants / Contributions	Notes / Decisions Made
2024-09-15	Project kickoff meeting with advisor. Discussed project selection and initial scope.	All team members, Advisor Mr. [Advisor Name]	Selected Heaven Hotel Management System; assigned initial roles.
2024-09-20	Brainstorming session for system features and user roles.	All team members	Defined main actors (Guest, Front Desk, etc.) and core modules.
2024-10-05	Reviewed SRS outline and began drafting Chapter One.	Team lead + 2 members	Completed initial Document Scope and Purpose.
2024-10-18	Requirements elicitation: Interviewed hotel staff (simulated) for real-world insights.	3 members	Added housekeeping and maintenance features based on feedback.
2024-11-02	Completed functional requirements and use case diagrams.	All team members	Finalized use cases for all actors.
2024-11-15	Data modeling workshop: Designed entities and relationships.	Backend-focused members	Created ERD and validation rules.
2024-11-28	Reviewed non-functional requirements and external interfaces.	All team members + advisor	Refined performance and security targets.
2024-12-10	Internal review of full SRS draft.	All team members	Incorporated advisor feedback from previous meeting.
2024-12-20	Finalized SRS document and prepared for submission.	All team members	Document ready for advisor approval.
2025-01-05	Advisor feedback session on SRS.	All team members, Advisor	Minor revisions to use cases and validation rules.
2025-01-15	Began SDD: System architecture planning and 4+1 views discussion.	All team members	Decided on Laravel modular structure.
2025-02-01	Ongoing development: Implemented authentication and booking modules.	Frontend and backend members	Weekly progress demos started.
2025-03-10	Mid-project review with advisor.	All team members, Advisor	Confirmed alignment with SRS; proceeded to UI design.
2025-04-05	Completed prototype UI mockups and data dictionary.	UI/UX focused members	Prepared screen images for SDD.
2025-04-20	Final integration and testing phase initiated.	All team members	Began requirement traceability matrix.
2025-05-01	Final documentation review and preparation for submission.	All team members, Advisor	Scheduled final presentation.