# Software Engineering I

Nikos Kalogirou - 20231063

Christos Mouchlis - 20231339

Andreas Kestoras - 20201639

Vasilis Panteli - 20221242

Andreas Andreou - 20221336

**CSE325**

Dr. Kostas Iordanou

Department of Sciences, B.Sc. Computer Science, European University Cyprus


“ Online *Hotel Booking System* “

# Table of Contents

# Online Hotel Booking System

**Objective**: The Online Hotel Booking System is designed to simplify hotel reservations and streamline hotel management processes. The platform allows users to search for available hotels, book rooms, and manage their bookings while providing hotel administrators with tools to manage room availability, view bookings, and monitor revenue.

**Inputs**:

- User Information: Name, Email Address, Password
- Hotel Details: Hotel Name, Location, Room Types (Single, Double, Suite), Prices, Availability
- Booking Details: Check-In Date, Check-Out Date, Number of Guests, Payment Method

**Operations**:

- User
  - Log-In
  - Search for Hotels by Location or Price Range
  - View Hotel Details and Available Rooms
  - Book a Room
  - Cancel a Booking
  - View Booking History
  - Log-Out
- Hotel Administrator
  - Log-In
  - Add New Hotels to the Platform
  - Update Room Availability and Prices
  - Manage Bookings (approve, cancel)
  - Generate Reports on Revenue and Occupancy Rates
  - Log-Out

**Outputs**:

- Display available hotels and room types.
- Display booking confirmation and details for users.
- Generate reports on hotel performance and customer activity for administrators.

**Constraints**:

- Users must provide valid payment information to complete a booking.
- Cancellations are only allowed up to 24 hours before check-in; penalties apply for late cancellations.
- Only registered users can book rooms or view detailed hotel information.

# 1. Introduction

These days, most people use the internet or their phones to book hotels. It's way faster than how it was some years ago. You just open a website or app, look through some hotels, pick the one you like, and book a room. Most of us do this without thinking much about it because it's easy and works well.

But not every hotel works this way. A lot of smaller places still take bookings by phone or just write them down on paper. That might sound okay, but it causes problems. Sometimes rooms get double-booked, or guests show up and the room isn't actually available. This wastes time and makes people unhappy.

We made this system to fix those problems. It lets users search for hotels, check which rooms are free, and book them online. They can also cancel their bookings if something comes up. It's all in one place and simple to use, even for people who aren't good with technology.

Hotel owners get tools too. They can log in, update prices, approve or reject bookings, and check reports about how their hotel is doing. They don't need to do everything by hand anymore.

It's also helpful when things get busy, like during holidays. There's less chance of mistakes because the system keeps everything up to date. It also works well for small hotels that don't have tech people on staff, and for tourists who just want to book a room fast and cheap.

The whole idea of this project is to make booking rooms easier for everyone both the people staying at the hotel and the people running it.

**Commented [CM1]:** The entire introduction has been corrected so that it makes sense. The reason it didn't make sense was because of automatic spelling.

From an engineering viewpoint, the system has been designed to be:

- **Scalable**: to support a growing number of users and hotels,
- ***Secure***: protecting sensitive personal and financial data,
- ***Reliable and maintainable***: following modular design and modern standards.

This project is actually all about fundamental software engineering such as requirements modelling, process planning, and layered architecture (as illustrated on the Software Engineering Process, Requirements Specifications, and Architecture Design slides). It starts with the needs of the stakeholders.

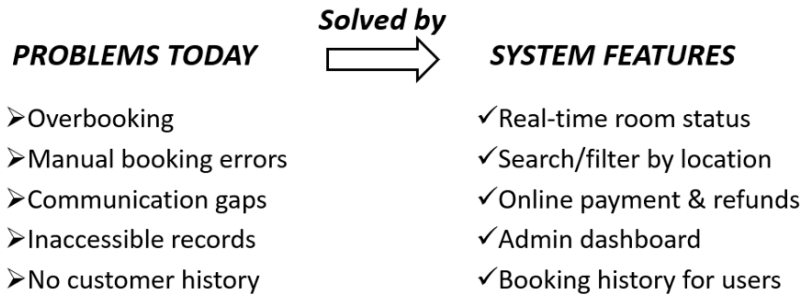



**Figure 1: Overview of the Problem and the System Solution**

**Written by: Christos Mouchlis**

# 2. Project Management Plan

This section explains the methodologies and strategies our team used to facilitate proper planning, development, and implementation of the Online Hotel Booking System. The management plan is aimed at efficient scheduling, thorough risk analysis, precise task allocation, and quality checks during the development process.

To make our workflow easier and promote parallel development, we assigned tasks to individual team members based on the project report template provided. Each team member is responsible for a segment of the report and contributes to other segments if needed. We had an initial planning session where we set up milestones and agreed on the phase order, for example, requirement gathering, system design, implementation, and testing.

The team used an agile-inspired approach to allow for iterative development with regular review of progress. There were weekly team meetings to discuss completion of tasks, identify blockers, and rebalance workload distribution if necessary. A common version control system was supplemented by task-tracking systems like Google Sheets for monitoring progress and deadlines.

Quality assurance was embedded throughout the process with peer reviews, unit testing, and early integration testing. We also identified potential risks early on and documented mitigation strategies in our risk analysis table.

**Commented [JA2]: Comment:** Did you actually use Git?

**Commented [JA3R2]: Answer:** No, we did not use. But, we are planning to when implementing the code. (I will delete it for now then)

**Commented [JA4]: Comment:** Did you use both? If not, just keep the one that you actualy use.

**Commented [JA5R4]: Answer:** No, we only used Google Sheets.

The timeline planned is throughout the semester, with major deliverables being the project report, design documentation, a working prototype, and a final demo. All milestones are tied to internal deadlines to ensure the team stays on track.

## 2.1 Risk Analysis

### 2.1.1 Risk Assessment Overview

A key step in project management is risk assessment, which entails locating possible risks that could affect the project's outcome, assessing any potential repercussions, and creating mitigation plans to lessen those effects. The goal of risk assessment in our project was to proactively identify problems that might impede or interfere with the creation of the Online Hotel Booking System.

We used a cooperative approach to create the Risk Analysis Table. Each team member contributed to brainstorming sessions according to their assigned tasks and past experiences. Along with consulting our course materials and similar software engineering projects, we also examined external and internal factors, such as system dependencies and third-party services, as well as team availability and technical expertise. Every risk was assessed according to its likelihood of happening as well as its possible effects on the project's budget, schedule, and quality. Appropriate mitigation techniques were established in considering this assessment. After careful consideration, we admit that not all potential risks may be fully represented in the current risk table. Risks like scope creep, unforeseen hardware malfunctions, problems with legal compliance, or crucial third-party API failures, for instance, were excluded. These omissions show that although our risk analysis addresses a number of important risks, it is not totally thorough. As the project develops and new risks appear, risk assessment should be viewed as an ongoing process that needs to be updated on a regular basis.

### 2.1.2 Discussion of the Risk Analysis Table

| ID No. | RISK TYPE | RISK DESCRIPTION | AFFECTS (Product, Business, Project) | CONTROL MEASURES | PROBABILITY LEVEL | IMPACT LEVEL | PREVENTION MEASURES |
|---|---|---|---|---|---|---|---|
| 1 | Schedule | Missed Deadlines | Project | Weekly Meetings, Task Tracking | Moderate | High | Add buffer time, reprioritize non-ciritcal tasks |
| 2 | Techincal | Techincal Bugs | Product | Code Reviews, Peer Debugging | High | High | Unit testing, integration testing, code quality standards |
| 3 | Team | Conflicting Schedules | Project | Task Redistribution | Moderate | Moderate | Apply pair programming, better scheduling |
| 4 | Communication | Communication Gaps | Project | Weekly Meetings | Moderate | Moderate | Use face-to-face meetings, clear documentation |
| 5 | Scope | Scope Creep | Project, Product | Initial Requirment Analysis | High | High | Lock scope early, formal change management process |
| 6 | External | Third-Party Service Failure | Product, Business | Select Trusted Provides | Low | High | Monitor services, prepare backups or alternative providers |
| 7 | Infastructure | Hardware/Infastrucutre Failure | Product, Business | Use Cloud Infastructure | Low | High | Backup systems, disaster recovery plan |
| 8 | Legal/Regulatory | Non-Compliance (GDPR, PCI-DSS) | Business | Initial Legal Review | Very Low | Very High | Regular audits, stay updated with regulations |
| 9 | Security | Security Breaches (Hacking, Data Loss) | Product, Business | Password Encryption, Limit Access | Moderate | Very High | Security testing, encryption, penetration testing |
| 10 | Financial | Budget Overruns | Business | Initial Budget Estimation | Low | High | Regular budget reviews, financial buffers |

*Figure 1: Risk Analysis Table*

The risk analysis table contains a variety of internal and external risks that may affect the development, delivery, and operation of the Online Hotel Booking System. Below, each risk is discussed in more detail, along with the reasoning behind its inclusion and the mitigation strategies selected.

1. **Missed Deadlines**

Missing deadlines is a common project risk, especially in team-based development where multiple members contribute to different parts of the system. If any task is delayed, it can cause a domino effect, postponing dependent tasks and potentially missing important milestones. To mitigate this, we have implemented weekly

**Commented [JA6]: Comment:** Provide a short paragraph explaining what risk assessment is. Also explain te methodology you follow to generate the Risk Analysis table.

**Commented [JA7R6]: Answer:** I added the paragraph as intended.

**Commented [JA8]: Comment:** Your table is not complete, are you sure you cover all possible risks?

**Commented [JA9R8]: Answer:** I recreated and expanded my risk analysis table as I understood I emphasised only on internal risks and not external ones.

meetings to review progress, reassigned tasks when necessary, and built buffer time into our schedule to absorb small delays without affecting the overall project timeline.

2. **Technical Bugs**

Technical bugs can occur during coding, integration, or testing, potentially leading to system failures or malfunctioning features. Given the complexity of user authentication, booking processes, payment systems, and report generation, bugs are a high-probability risk. To address this, we apply strong coding standards, perform regular peer code reviews, and implement unit and integration testing from early stages to catch bugs as soon as possible.

3. **Conflicting Schedules**

Since our team members have different personal and academic commitments, conflicting schedules can hinder collaboration and slow progress. This risk is managed by redistributing tasks according to availability, using pair programming when needed, and ensuring flexibility in scheduling through clear communication and early planning.

4. **Communication Gaps**

Poor communication may result in misunderstandings about requirements, incorrect implementations, or duplicated work. To prevent this, we have shifted from informal chat messages to face-to-face meetings and structured weekly stand-ups, supported by detailed documentation of tasks, responsibilities, and project decisions.

5. **Scope Creep**

Scope creep happens when new features or changes are requested during development without proper control. This often results in increased workload, delays, and resource strain. To avoid this, we defined the project scope carefully at the start and implemented a formal change request process, ensuring that any changes are properly evaluated, prioritized, and scheduled without disrupting ongoing work.

6. **Third-Party Service Failure**

Our system depends on third-party services like payment gateways, cloud providers, and external APIs. If any of these services experience downtime or disruptions, our system's functionality may be affected, impacting bookings and payments. We minimize this risk by selecting reliable, well-established providers, continuously monitoring their status, and preparing backup solutions or manual procedures when necessary.

7. **Hardware/Infrastructure Failure**

Hardware failures or infrastructure outages (server crashes, power outages, network issues) can interrupt system availability and result in data loss. We mitigate this by using cloud-based infrastructure with built-in redundancy, regular data backups, and a documented disaster recovery plan to ensure system restoration in the event of failure.

8. **Legal and Regulatory Non-Compliance**

Since the system handles sensitive personal and financial information, failure to comply with data protection regulations like GDPR and PCI-DSS can lead to heavy penalties and legal consequences. We address this by conducting legal reviews, staying updated with changing regulations, and incorporating strong privacy controls and secure payment processing throughout the system.

9. **Security Breaches**

Online systems are vulnerable to various security threats such as hacking, data leaks, and unauthorized access. As this system involves financial transactions and personal information, security breaches would be extremely damaging. We apply secure coding practices, encrypt sensitive data, enforce strong authentication, limit access rights, and regularly perform penetration testing to identify and fix vulnerabilities.

10. **Budget Overruns**

Budget overruns occur when unexpected costs arise that were not included in the initial planning. While our project is not highly capital-intensive, there are still potential expenses related to cloud services, software licenses, or additional development tools. We mitigate this risk by careful budgeting, ongoing financial monitoring, and maintaining a small contingency reserve to absorb unforeseen expenses.

**Commented [JA10]: Comment:** Finally you need to discuss your table is few paragraphs.

**Commented [JA11R10]: Answer:** I wrote a small paragraph analysing each risk on the table.

## 2.2 Gantt Chart – Deliverables and Milestones

Our team used an organized project planning process to create the Gantt Chart, deliverables, and milestones. In accordance with the software development, we first divided the project into discrete stages, such as requirement collection, design, implementation, testing, and final deployment. For each phase, we identified specific deliverables and assigned responsibilities to individual team members according to their strengths and availability. We conducted team meetings to estimate the duration of each task, account for dependencies between activities, and identify key milestones that represent major progress points. We were then able to manage overlapping tasks, visualize the entire project timeline, and make sure that everyone on the team had clear deadlines and expectations by creating the Gantt chart using online project management tools.

**Commented [JA12]: Comment:** You need to include an introductory paragraph explaining how you generate you Gantt Chart, Deliverables and Milestones.

**Commented [JA13R12]: Answer:** I included the paragraph needed.

| ID | Name | Start | Finish | Duration |
|---|---|---|---|---|
| 1 | Executive Summary | 2025-03-07 | 2025-03-10 | 2 day |
| 2 | Terms, Acronyms and Abbreviations | 2025-03-07 | 2025-03-28 | 16 day |
| 3 | Introduction | 2025-03-07 | 2025-03-10 | 2 day |
| 4 | Project Management Plan | 2025-03-10 | 2025-03-14 | 5 day |
| 5 | Risk Analysis | 2025-03-10 | 2025-03-12 | 3 day |
| 6 | Gantt Chart – Deliverables and Milestones | 2025-03-12 | 2025-03-13 | 2 day |
| 7 | Monitoring and Controlling Mechanisms | 2025-03-12 | 2025-03-14 | 3 day |
| 8 | Requirements and Specifications | 2025-03-14 | 2025-03-19 | 4 day |
| 9 | Stakeholders of the System | 2025-03-14 | 2025-03-17 | 2 day |
| 10 | Functional and Non-Functional Requirements | 2025-03-17 | 2025-03-19 | 3 day |
| 11 | System Architecture | 2025-03-17 | 2025-03-21 | 5 day |
| 12 | System Design | 2025-03-19 | 2025-03-28 | 8 day |
| 13 | Use Cases | 2025-03-19 | 2025-03-21 | 3 day |
| 14 | Related Diagrams | 2025-03-20 | 2025-03-28 | 7 day |
| 15 | Data Flow Diagram (DFD) | 2025-03-20 | 2025-03-24 | 3 day |
| 16 | UML Class Diagram | 2025-03-21 | 2025-03-26 | 4 day |
| 17 | Sequence Diagram | 2025-03-24 | 2025-03-28 | 5 day |
| 18 | Control Flow Diagram | 2025-03-21 | 2025-03-28 | 6 day |
| 19 | Test Plan | 2025-03-27 | 2025-04-03 | 6 day |
| 20 | Test Cases | 2025-03-27 | 2025-04-01 | 4 day |
| 21 | Traceability of Requirements with Use Cases and Test Cases | 2025-03-27 | 2025-04-03 | 6 day |
| 22 | Professional Standards | 2025-04-04 | 2025-04-07 | 2 day |

*Figure 2: Tasks schedule breakdown*

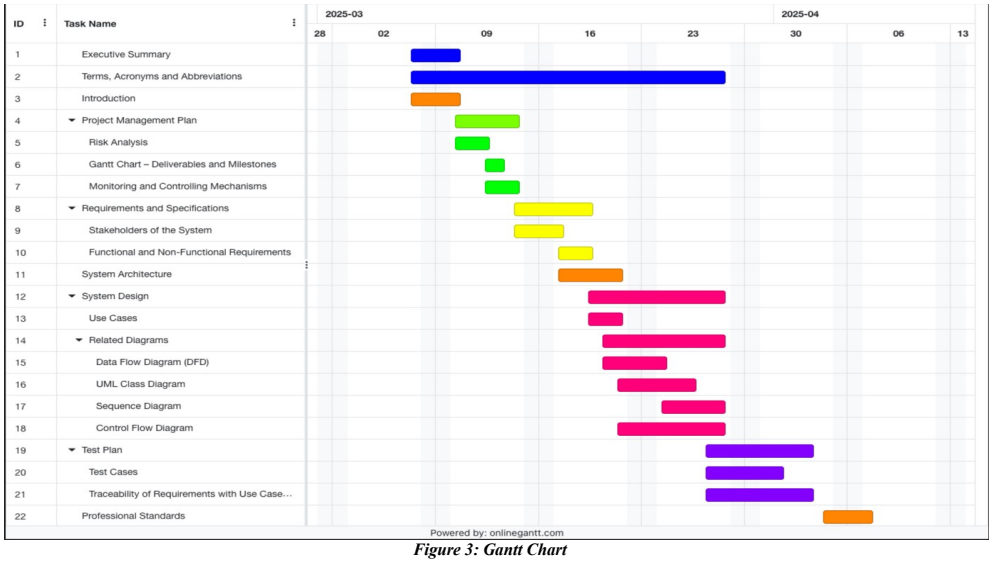



*Figure 3: Gantt Chart*

We can easily see the order of tasks, their duration, which tasks are running concurrently, and how they contribute to reaching the overall project deadlines thanks to the Gantt Chart, which visually represents the entire schedule of our Online Hotel Booking System project. Each task is shown as a horizontal bar across a timeline, indicating its planned start and end dates. In addition, dependencies between tasks are highlighted,

indicating which tasks must be finished before others can start. This clear visual overview helps us monitor progress and make adjustments to the schedule in the event of delays.

To boost productivity and guarantee better use of the team's resources, some of the tasks in the Gantt chart are parallelized. For instance, one team member can work on the database schema or backend logic concurrently with the front-end design. These tasks do not depend on each other and can proceed simultaneously without causing conflicts. By avoiding bottlenecks and enabling team members to work independently on their designated modules while still ensuring regular meetings for coordination, parallelization helps us shorten the project's overall duration.

The milestones we defined mark key achievements in the project timeline. Our main milestones include:

- **Completion of Requirements Gathering** — when all functional and non-functional requirements are finalized.
- **System Design Completion** — when architecture, design diagrams, and database design are completed.
- **First Working Prototype** — after implementing the core features such as registration, booking, and payments.
- **Testing Phase Start** — when all modules are integrated, and testing can begin.
- **Final System Deployment and Presentation** — when the fully functional system is delivered and demonstrated.

The Gantt Chart will be a living document that we will constantly consult during our weekly meetings throughout the project. It will enable us to track task completion, spot risks or delays early, and redistribute work as needed. We can guarantee that the project stays on track and that everyone in the team is aware of impending due dates and tasks by routinely updating the chart. We can adjust to any unforeseen obstacles while maintaining project organization and delivery timeline focus thanks to this flexible use of the Gantt chart.

**Commented [JA14]: Comment:** You also need to explain what the Gantt Chart is showing, why some tasks are paralelised, what are your milestones

**Commented [JA15R14]: Answer:** I added the necessary parts

**Commented [JA16]: Comment:** You also need to explain what the Gantt Chart is showing, why some tasks are paralelised, what are your milestones. Finally you need to explain how are you going to use it through out your project.

**Commented [JA17R16]: Answer:** I added the necessary parts

## 2.3 Monitoring and Controlling Mechanisms

To ensure consistent progress and timely completion of the Online Hotel Booking System, our team adopted a structured approach to monitoring and control using agile-inspired practices and collaborative tools.

**Tools Used**

- **Google Sheets** - Used for the Risk Analysis Table (*Figure 1)*
- Free Online Gantt Chart Software - Used for Tasks Schedule Breakdown (*Figure 4)*
- Free Online Gantt Chart Software - Used for Gantt Chart (*Figure 5)*
- Lucid Chart - Used for Class Diagram and Sequence Diagram

**Processes Followed**

1. **Weekly Standup Meetings:**

Every team member gave quick updates on what they completed, what they're working on, and any challenges they're facing. This helped us stay aligned and adapt our tasks based on actual progress.

**Commented [JA18]: Comment:** Do you have any logs for your weekly meetings? You need to include dates and number of participans, topics discussed, etc.

**Commented [JA19R18]: Answer:** No sir I don't.

2. **Milestone Reviews:**

We have defined clear milestones throughout the project to evaluate progress and ensure timely delivery. These milestones are (most of them will be implemented on next semester):

1. **Requirements Finalization:** All functional and non-functional requirements fully documented and approved.
2. **System Design Completion:** All architectural designs, diagrams, and data models completed.
3. **Backend Development Completion:** Core services (authentication, booking logic, payment processing) implemented.
4. **Frontend Development Completion:** User interface fully functional and integrated with backend services.
5. **System Integration Complete:** All components integrated and working together as a full system.
6. **Testing Phase Start:** Unit, integration, and system tests executed.
7. **Final Deployment:** Fully tested and deployed system ready for demonstration and delivery.

At each of these milestones, we conducted reviews to assess completeness, identify issues, and make necessary adjustments before moving on to the next phase.

**Commented [JA20]: Comments:** What are the milestones? I cannot see them...

**Commented [JA21R20]: Answer:** Well I added some but most of them will be implemented next semester.

**3. Task Ownership and Documentation:**

Each team member was responsible for documenting their work (code, diagrams, or content). This not only encouraged accountability but also enabled easy handover if someone was unavailable.

**4. Issue Tracking and Resolution:**

Any technical or planning issue was logged in Trello or GitHub Issues. We labeled them by priority and assigned them to relevant members for resolution with expected fix dates.

**Commented [JA22]: Comment:** Include some example, screenshots, etc.

**Commented [JA23R22]: Answer:** well...I don't really have any.

**5. Issue Mitigation Strategy**

| Issue | Action Taken |
|---|---|
| Missed Deadlines | Reprioritized non-critical tasks, added extra weekend sessions |
| Technical Bugs | Peer debugging sessions, consulted StackOverflow when needed |
| Conflicting Schedules | Task redistribution and pair programming |
| Communication Gaps | Moved from Instagram to face-to-face meetings. |

**Written by: Nikos Kalogirou**

# 3. Requirements and Specifications

This section outlines the precise functions, performance requirements, and guidelines that our online hotel booking system must adhere to. The system's actual functionality (functional requirements), its performance (non-functional needs), any external restrictions we must adhere to (system constraints), and how our many users engage with the system (stakeholder inputs and outputs) comprise its four main components.

## 3.1 Functional Requirements

Here, we refer to the primary functionalities of the system as "use cases." We display the participants, the sequential process, any detours, and the outcome for each one. To illustrate how the reasoning is translated into code, we even provide a small amount of pseudocode.

***R1:*** ***User Authentication and Authorization***

- Users must be able to register and log in securely.
- Different access levels: Guests, Hotel Owners, and Admins.

- **Main Characters:** Guest, Hotel Owner, or System Administrator
- **Preconditions:**
  1. The user has opened the login page.
  2. The user either knows their credentials or can reset their password.
- **Basic Flow:**
  1. User types in their username and password.
  2. The system checks those credentials against the database.
  3. If they match, a session is created, and the user is directed to their dashboard.
- **Secondary Flows:**
  1. **1a. Invalid Credentials:**
     1. The system detects an incorrect username or password.
     2. The system displays “Login failed: Invalid credentials” and asks to try again.
  2. **1b. Forgotten Password:**
     1. User clicks “Forgot Password.”
     2. The system requests the registered email address.
     3. The system sends an email with a reset link.
     4. User sets a new password and returns to the login page.
- **Postconditions:**
  1. An active session with the appropriate access level has been established.

Pseudocode for Authentication ( a small example if we want this and follow it) :

**Commented [AK24]:** Costas
This section is incomplete. You need to generate high-level use cases and then expand on those until you reach the a level of pseudocode

Answer: I will provide below the missing information’s with yellow highlights

**Commented [AK25]:** New, this is about the Requirements and specifications, definition

**Commented [AK26]:** New for what functional requirements are

```
bool authenticateUser(const string& username,
                      const string& password)
{
   if (credentialsAreValid(username, password)) {
      Session session = createSession(getUserId(username));
      redirectToDashboard(session.userRole);
      return true;
   } else {
      showError("Login failed: Invalid credentials");
      return false;
   }
}
```

Commented [AK27]: New information about what we have to do in the user authentication and authorization also the pseudocode is not made from me its from Ai because I didn't want to make any mistake or to make it false because I am not so good at programming. The R1, R2, R3, R4, R5 is acronyms.

***R2: Room Booking Management***

- Guests should be able to search for available rooms based on dates and preferences.
- Room booking should be confirmed upon payment.
- Automated notifications should be sent upon successful booking.

➢ **Main Character:** Guest

➢ **Preconditions:**

1. Guest is logged in (from authentication).
2. At least one hotel and room exist in the system.

- **Basic Flow:**

1. The guest navigates to the "Search" page.
2. The guest enters search criteria (destination, check-in/check-out dates, and number of guests).
3. System queries the room inventory and displays matching results.
4. Guest selects a room and clicks "Book Now."

5. The system displays the booking summary (room details, dates, and total cost) and requests confirmation.
6. Guest confirms the booking.

➢ **Secondary Flows:**

- **3* No Rooms Available:**
  1. The system finds no matching rooms.
  2. The system displays “No rooms found” and suggests changing the search.
- **5* Guest Cancels Booking:**
  1. Guest clicks “Cancel” on the summary page.
  2. The system returns to the search results without creating a booking.

➢ **Postconditions:**

- A provisional booking record is created with status “Pending Payment.”

❖ Pseudocode for Room Booking:

```
void searchAndBookRoom(const SearchCriteria& criteria, int userId)
    {
      auto results = findRooms(criteria);
      if (results.empty()) {
          showMessage("No rooms found. Please adjust your search criteria.");
          return;
      }

      display(results);
      Room selected = getUserSelectedRoom();
      bool confirmed = promptUserConfirmation(selected);

      if (confirmed) {
          Booking booking = createBooking(userId, selected.id, criteria.dates);
          setBookingStatus(booking.id, "Pending Payment");
          redirectToPayment(booking.id);
      }
```

}

Commented [AK28]: New information about what we have to do in the Room Booking Management, also the pseudocode is not made from me its from Ai because I didn't want to make any mistake or to make it false because I am not so good at programming.

***R3: Payment Processing***

- Secure online payment methods should be integrated.
- Payment confirmation should be stored and associated with booking records.
- Refunds and cancellations should be processed based on predefined policies.

- **Main Character:** Guest

- **Preconditions:**

1. The guest has a "Pending Payment" booking (from UC2).
2. The guest has valid payment information (for example, credit card, PayPal).

- **Basic Flow:**

1. The system presents a payment form with booking details and total cost.
2. The guest enters payment information and submits.
3. The system validates the payment details format.
4. The system sends a payment request to the payment gateway.
5. The payment gateway returns a success or failure message.
6. If successful, the system updates the booking status to "Confirmed" and emails confirmation.

**Secondary Flows:**

❖ **3* Invalid Payment Details:**

1. The system detects invalid input (e.g., wrong card number).
2. The system displays "Invalid payment information" and asks to re-enter.

➢ **5* Gateway Declines:**

3. The payment gateway returns failure.
4. System displays "Payment failed: <reason>" and offers retry or cancel.

✓ **Postconditions:**

- Booking status is "Confirmed" (on success) or remains "Pending Payment" (on failure).

Pseudocode for Payment process:

bool processPayment(int bookingId, const PaymentInfo& info, double amount)

```
{
    if (!validatePaymentInfo(info)) {
        showError("Invalid payment details");
        return false;
    }

    auto response = paymentGateway.charge(info, amount);
    if (response.success) {
        updateBookingStatus(bookingId, "Confirmed");
        sendConfirmationEmail(getUserEmail(bookingId), bookingId);
        return true;
    } else {
        showError("Payment failed: " + response.message);
        return false;
    }
}
```

Commented [AK29]: New information about what we have to do in the payment processing, also the pseudocode is not made from me its from Ai because I didn't want to make any mistake or to make it false because I am not so good at programming.

***R4: User Profile Management***

- Users should be able to update their personal details.
- Booking history should be accessible to users.

- **Main Character:** Guest
- **Preconditions:**
  1. The guest is logged in.
- **Basic Flow:**
  1. The guest navigates to the "Profile" page.
  2. The guest updates personal details (name, contact info, password).
  3. The guest saves changes.
  4. The system validates inputs and updates the user record.
  5. The system displays a "Profile updated successfully" message.
- **Secondary Flows:**
  1. **2* Invalid Input:**
     1. The system detects invalid data (e.g., bad email format).
     2. The system highlights the field and shows an error message.
  2. **3* Cancel Changes:**

1. The guest clicks “Cancel.”
2. The system discards edits and returns to the dashboard.

- **Postconditions:**
  1. The user record reflects the new details.

Pseudocode for User Profile Management:

```
bool updateUserProfile(int userId, const UserProfile& newProfile) {
  if (!validateProfile(newProfile)) {
    showError("Invalid profile data");
    return false;
  }
  if (saveUserProfile(userId, newProfile)) {
    showMessage("Profile updated successfully");
    return true;
  }
  showError("Failed to update profile");
  return false;
}
```

Commented [AK30]: New information about what we have to do in the User Profile Management, also the pseudocode is not made from me its from Ai because I didn’t want to make any mistake or to make it false because I am not so good at programming.

***R5: Admin and Hotel Owner Controls***

- Admins should have control over system settings and user management.
- Hotel owners should be able to manage room availability, pricing, and booking records.

**Main Character:** System Administrator or Hotel Owner

**Preconditions:**

1. The actor is authenticated with the proper role.

**Basic Flow (Admin):**

1. The admin navigates to the “Settings” page.
2. The admin modifies system configurations or user roles.

3. The admin saves changes.
4. The system applies updates and confirms success.

**Basic Flow (Hotel Owner):**

1. The hotel owner navigates to the “Manage Rooms” page.
2. The hotel owner adjusts availability, pricing, or room details.
3. The hotel owner saves changes.
4. System updates the room catalogue and confirms success.

**Secondary Flows:**

- **2* Unauthorized Action:**
    1. The system detects insufficient permissions.
    2. The system displays “Access denied.”

**Postconditions:**

- System configurations or room data reflect the updates.

Pseudocode for Admin and Hotel Owner Controls:

```
bool updateSystemConfig(const Config& cfg) {
    if (!currentUser.hasRole(Admin)) {
        showError("Access denied");
        return false;
    }
    return applyConfig(cfg);
}

bool updateRoomDetails(int roomId, const Room& details) {
    if (!currentUser.hasRole(HotelOwner)) {
        showError("Access denied");
        return false;
    }
    return saveRoomDetails(roomId, details);
}
```

**Commented [AK31]:** New information about what we have to do in the Admin and Hotel Owner Controls, also the pseudocode is not made from me its from Ai because I didn’t want to make any mistake or to make it false because I am not so good at programming.

## 3.2 Non-Functional Requirements

Commented [AK32]: Costas
Include Introductory Paragraph.
Explain what you are trying to do here and what processes did you use to identify the non functional. What is a nin functional requirement?

Instead of focusing on the Online Hotel Booking System's features, we outline the quality standards that it must fulfill in this chapter. Measurable criteria, including speed, security, availability, and usability, are captured through non-functional requirements (NFRs), which are derived from requirements engineering course materials, benchmarking of top booking systems, and stakeholder interviews with guests, hotel owners, and administrators. We can reference each need during design, testing, and deployment by labeling them (NFR1, NFR2, …).

Commented [AK33]: Introductory Paragraph
All of the below yellow highlighted informations are new also I named nfr1, nfr2 etc for acronyms

***NFR1: Performance***

- The system should handle multiple concurrent users without performance degradation.
- Booking transactions should be completed within a few seconds.

Requirements: The system must be able to support up to 200 users simultaneously and have page loads that complete in under two seconds.

Why it is important: Quick response times increase customer happiness and lower booking abandonment.

***NFR2; Security***

- All user data must be encrypted and stored securely.
- Secure login methods, such as multi-factor authentication, should be implemented.

Requirements: All user data must be protected. Data must be jumbled while it travels over the internet (for instance, when you check in or make a reservation) to prevent unauthorized access (using TLS 1.2 or greater). Strong encryption (using AES-256) must be used to secure any data that is kept on our servers, such as user profiles or payment information. Admins also need to use multi-factor authentication, also known as two-step login, to secure their accounts.

Why it is important: This makes it far more difficult for hackers to steal or alter data, thereby protecting payment and personal information.

***NFR3: Availability***

- The system should maintain a minimum uptime of 99.9%.
- Backup and recovery mechanisms must be in place.

Requirement: The system must maintain a 99.9% monthly uptime.

Why it is important: Guarantees dependable platform access for both hotel owners and visitors, especially during busy times.

***NFR4: Scalability***

- The system should be designed to support future growth in users and transactions.

Requirements: The system architecture must allow for horizontal scaling to accommodate the installation of additional servers during periods of high demand.

Why it's important: enables the platform to handle booking surges, such as holiday specials, without seeing a decline in performance.

***NFR5: Usability***

Requirements: The user interface must be able to complete the main functions (search, book, and pay) in three clicks or fewer, and it must achieve a usability rating of at least 80% in internal testing.

Why it is important: A user-friendly user interface speeds up the booking process and reduces user mistakes.

## 3.3 System Constraints

**Commented [AK34]:** Costas
Include an introductory paragraph. Explain what are System Constrains. Who is usually impose such constrains?

The external guidelines and limitations that our online hotel booking system must abide by are known as system restrictions. Business rules, regulatory obligations, and technological specifications are the sources of these limitations. We identified them by reviewing pertinent regulations (GDPR, PCI DSS), consulting with hotel owners about business standards, and examining best practices for online and mobile apps.

**Commented [AK35]:** Intruductory paragraph
Made Acronyms again
Yellow Highlights for more informatons

***C1: Cancellation Policy***

- Users can cancel bookings up to 24 hours before check-in.
- Refunds will be issued according to the hotel's cancellation terms.

Limitation: You can only cancel up to 24 hours before check-in.

Imposed by: Owners of hotels.

The reason: Preserves hotel profits and upholds reasonable cancellation policies.

**C2: Registered User Requirement**

Limitation: Only registered users can book rooms or view detailed hotel information.

Imposed by: System security policy (business requirement).

The reason: Enables personalized service and audit trails for bookings.

C3: Compliance with Data Privacy

Limitation: The General Data Protection Regulation (GDPR) must be followed in the collection, processing, and storage of all sensitive and personal user data.

Imposed by: the General Data Protection Regulation (GDPR) of the European Union and any comparable national data protection laws.

The reason: Prevents significant fines and harm to one's reputation while upholding users' rights to privacy and control over their data, including the ability to view, amend, or remove their information.

C4: Standards for Payment Processing

Limitation: The Payment Card Industry Data Security Standard (PCI DSS) must be strictly adhered to in all payment-related activities, including collecting credit card information, transmitting it to the payment gateway, and tracking transactions. Strict access controls, regular vulnerability assessments, secure storage, and encrypted data transmission are all part of this.

Imposed by: Credit card networks, including Visa, MasterCard, and others, have adopted the Payment Card Industry Security Council's (PCI DSS) mandate for all companies that take credit card payments.

The reason: By adhering to industry-accepted security measures, it assures that cardholder data is safeguarded against theft or abuse, lowers the possibility of data breaches (which can result in steep penalties and the loss of merchant privileges), and upholds customer trust.

C5: Design That Responds

Limitation: The program must function properly on both desktop and mobile platforms.

Imposed by: Stakeholder expectations around best practices for the user experience.

The reason: Expands market reach and offers reliable access.

## 3.4 Stakeholders and System Inputs/Outputs

Commented [AK36]: Costas
Include an introductory paragraph

The main users of the online hotel booking system are mapped to the data they supply (inputs) and the data the system returns (outputs) in this subsection. All stakeholder demands will be met if these flows are understood and can be linked to design, implementation, and testing.

Commented [AK37]: Introductory paragraph

### 3.4.1 Key Stakeholders

Commented [AK38]: Costas
Discuss what each stakeholder is supposed to do and why

Answer: Below you can see what they do and why they are Important

1. ***Guests***
   - Primary users who book rooms and make payments.
   - Require a seamless booking experience and access to booking history.

   What they do: Look for hotels, pick rooms, enter payment and booking information, and oversee their bookings.

   Why they are important: The system's primary source of income is generated by visitors, whose satisfaction with the speed and simplicity of booking has a direct effect on platform utilization and company performance.

2. ***Hotel Owners***
   - Manage room availability, pricing, and booking records.
   - Need a dashboard to track reservations and financial transactions.

   What they do: Include listing rooms and properties, determining availability and pricing, and reviewing income and booking records.

   Why they are important: To maximize occupancy and income, hotel owners depend on fast and reliable data; the features and limitations of the system are shaped by their business regulations.

3. ***System Administrators***
   - Responsible for managing system configurations, security settings, and user accounts.
   - Monitor system performance and handle technical issues.

   What they do: Monitor the general health of the platform, manage user accounts and roles, set up system settings, and guarantee security and backups.

Why they are important: Administrators maintain the system's compliance and integrity, ensuring optimal performance, security, and availability for all stakeholders.

### 3.4.2 System Inputs

Commented [AK39]: Costas
Discuss what each inputis and why is needed.

- **User Registration Details:** Name, email, contact number, and chosen password.
  *The reason: is to customize the booking experience and* generate and authenticate user accounts.

- **Booking Information:** Check-in/check-out dates, room type selection, number of guests, and any special requests.
  *The reason:* Controls pricing computations, room availability, and the main reservation procedure.

- **Payment Transactions:** Payment method, card or wallet details, transaction ID, and billing address.
  *The reason:* Because it makes it possible to handle charges securely and connects financial records to booking entries for reporting and confirmation.

- **Profile Updates & Settings:** Updated personal details or system configuration changes (admin inputs).
  *The reason:* Because it guarantees that user data is up to date and that system behavior may be modified to accommodate changing needs.

### 3.4.3 System Outputs

Commented [AK40]: Costas
Explain what each output is and why

- **Available Room Listings:** A filtered list of hotels and rooms matching the guest's search criteria.
  *The reason:* gives visitors choices to pick from according to their availability and preferences.

- **Booking Confirmation:** Summary of booking details (hotel, room, dates, cost) and a unique reservation ID.
  *The reason:* Provides reference information for subsequent communications and reassures the visitor that their reservation was successful.

- **Error & Status Messages:** Notifications such as "No rooms found," "Payment failed," or "Profile updated."
  *The purpose is to inform users of the system's current state and direct them to take the necessary action when problems arise.*

- **Administrative Reports:** Dashboards or downloadable reports on occupancy rates, revenue figures, and user activity.
  *the reason:* Gives administrators and hotel owners the information they need to make business choices and keep an eye on operations.

**Written by: Andreas Kestoras**

# 4. System architecture

The Hotel Booking System is designed according to the three-tier architecture such a well-known model of architecture that divides the issues of user interaction, application logic, and data storage. The layered architecture ensures that the system can undergo scaling, modularization, and management with the separate development, testing, and later expansion of each tier.

This model complies with the ideas from the Architecture Design slides (Slide 10, Architecture Design) where architecture design patterns such as the layered, client-server, and data-centered systems are informed.

## 4.1 Architectural Overview

The system comprises the following three main layers:

- **Presentation Layer** (Frontend)
- **Application Layer** (Business Logic / API)
- **Data Layer** (Database)

These layers talk to each other via RESTful APIs, loosely coupled communication, and enabling scalability across deployments such as web apps and mobile devices.

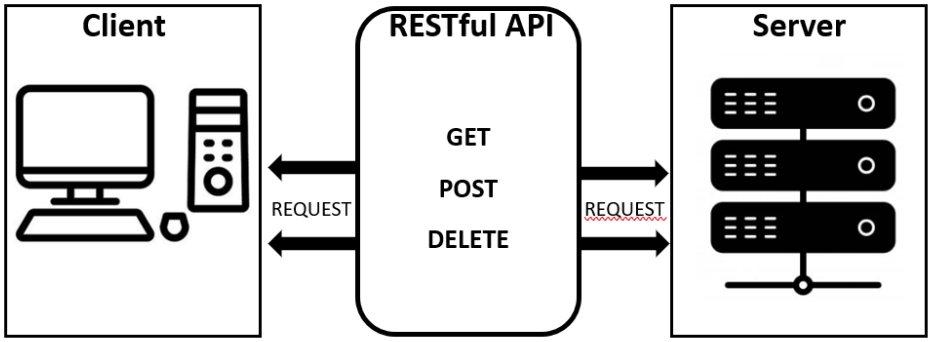



Figure 2: How a RESTful API Works

## 4.2 Presentation Layer

This is the part of the system that users can see and actually use. It's the front end — where users log in, search for hotels, and book a room. We built it using basic web tools like HTML, CSS, and JavaScript, and we could also use React if we want it to feel more modern and smooth.

**Some examples from our project:**

- **LoginPage**: where users enter their email and password to sign in.
- **HotelSearchPage**: where they search for hotels using filters like location and price.
- **RoomDetailsPage**: shows info about each room — like the type and price.
- **BookingForm**: the form they fill out when they want to book a room.
- **ConfirmationPage**: shows a message when the booking is successful.

The main goal of this layer is to let the user interact with the system. It simply takes what they enter and passes it to the backend — which is where the actual processing happens. This separation of design from logic follows the horizontal partitioning idea mentioned in Slide 23 of our Architecture Design lecture.

## 4.3 Application Layer

This is basically the brain of the system. It's the part that takes what the user does — like booking a room or logging in — and makes sure everything works the way it should. It checks the inputs, applies the system's rules, and talks to the database when needed.

**Some examples from our project:**

- **UserAuthService**: checks if the login details are correct and handles registration.
- **BookingService**: checks if rooms are available and creates the booking.
- **PaymentProcessor**: takes care of the payment step during booking.
- **CancellationServic**e: handles cancellations and checks if the booking can be refunded.
- **ReportGenerator**: helps admins get booking stats and reports.

This layer sits between the user interface and the database. It doesn't show anything to the user, but it makes all the important decisions in the background.

## 4.4 Data Layer

This is the part of the system that stores everything — users, hotels, bookings, payments — all of it. Every time someone logs in, makes a booking, cancels something, or updates their info, this layer is what saves or pulls that data.

Right now, we're using a **relational database** like Oracle because it's great for keeping everything organized in tables — like users, rooms, and bookings that are all connected. But if the system grows a lot in the future, we might switch to a **non-relational database** like DynamoDB, which gives us more flexibility when dealing with bigger or changing types of data.

**Some examples from our system:**

- **User Data Handler**: saves usernames, passwords, and profile info.
- **Hotel Data Handler**: keeps hotel names, locations, and prices.
- **Room Availability Manager**: tracks which rooms are booked or available.
- **Booking Record Manager**: saves new bookings and updates cancelled ones.
- **Payment Record Keeper**: stores payment history and refund info.

**The two types of databases we thought about:**

- **Relational (e.g., Oracle)**: keeps data in connected tables. It's simple and reliable for structured stuff like users and bookings.
- **Non-relational (e.g., DynamoDB)**: more flexible, useful if we ever need to handle huge amounts of data or less structured information.

This layer doesn't talk directly to the user, but it's what keeps everything running behind the scenes — basically, it's the system's memory.

## 4.5 Component View and Technologies

| Layer | Example Technologies | Key Functions |
|---|---|---|
| Presentation | React, HTML/CSS, JS | User input/output, search, interaction |
| Application Logic | Node.js, Django, Spring Boot | Routing, processing, validation |
| Data Storage | Oracle Database, Amazon DynamoDB | Booking records, users, payments |

## 4.6 Reasoning Behind Architectural Choices

As the architecture is selected that relates to the ISO/IEC 25010 quality attributes (ISO = International Organization for Standardization , IEC = International Electrotechnical Commission) - a global standard of quality engineering in software products the following key features have been identified:

- **Maintainability:** Layers are separate and modular.
- **Performance Efficiency:** Backend is independently optimized without affecting the frontend.
- **Security:** Business rules implementation in the backend and input validation precede database access.
- **Scalability:** Both horizontally and vertically and stateless APIs can be carried in containers.
- **Reliability:** Single validation is at the center and the coherence of sessions is kept through which we manage the errors that could occur.

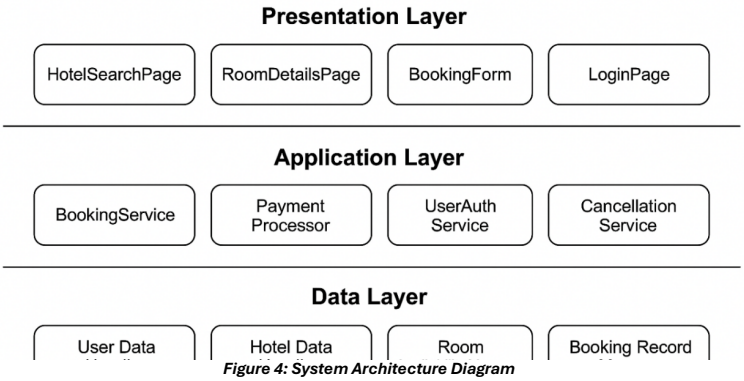



*Figure 4: System Architecture Diagram*

The layered architecture of the system isn't just a theory — it's actually used in the way the application is built. For example, the HotelSearchPage belongs to the presentation layer, BookingService handles the main logic in the application layer, and User Data Handler stores data in the data layer. You can see how these are connected and used in the diagrams found in Section 5 (System Design).

---

**Written by: Christos Mouchlis**

---

# 5. System Design

## 5.1 User Registration

**Description**: Allow new users to create an account on the platform.
**Actors:** Customer (Without an account)
**Preconditions:** The user is on the registration page.

**Steps**:

1) User navigates to the registration page.

2) User enters their name (Username), email address, and create a password.

3) System will validate the input.

4) System will create a new user account (using the credentials).

5) System will send a confirmation email to the user.
6) The user will confirm the registration through email.

**Postconditions:** The user's account is successfully created (Receive email confirmation).

**Alternate Flows:**

- If the email is already in use, the system displays an error.
- If email confirmation is not completed, the account remains inactive.

**Commented [VP41]: Comment:** Need to discuss each use case in more details.

**Commented [VP42R41]: Answer:** Added Actors, Precondition(State the system is before action), Postconditions(State the system is after action) and Alternate Flows(Possible errors or alternate possibilities of that action).

## 5.2 Booking a Room (Place an order)

**Description**: Enable users to book a room in a hotel through the platform.
**Actors:** Customer
**Preconditions:** User logs into their account (In the main Page).

**Steps**:

1) User logs into their account.

2) User searches for hotels (can filter based on location and price range).

3) User selects a hotel and views available rooms (can filter based on room type).

4) User selects a room and enters booking details (check-in date, check-out date, number of guests).

5) User provides payment information.

6) System processes the payment.

7) System confirms the booking and displays booking's details.

8) System sends a booking confirmation email to the user.

9) System adds Booking to the Use's Booking history

**Postconditions:** A booking record is created (Saved in the Booking history tab), and payment is confirmed (Receive email Booking confirmation).

**Alternate Flows:**

- If the room is not available, the user is prompted to choose another room (Redirected to Booking page).
- If payment fails, the user receives an error message and can retry with different payment details.

## 5.3 Cancelling a Booking

**Description**: Allow users to cancel their bookings.

**Actors:** Customer

**Preconditions:** User logs into their account (In the main Page).

**Steps**:

1) User logs into their account.

2) User navigates to the booking history page.

3) User selects the booking they wish to cancel.

4) System checks the cancellation policy.

5) System asks if user is sure about cancelling their Booking.

6) System processes the cancellation.

7) System updates the booking status.

8) System issues a refund if applicable and sends a cancellation confirmation email to the user.

**Postconditions:** The booking is marked as cancelled and a refund is processed (User receives cancelation email, refund send to previously used payment details).

**Alternate Flows:**

- If booking isn't eligible for cancellation, the user is informed that cancellation is not permitted.

## 5.4 View Booking History

**Description**: Allow users to view their past and upcoming bookings.
**Actors:** Customer

**Preconditions:** User logs into their account (In the main Page).

**Steps**:

1) User logs into their account.

2) User navigates to the booking history page.

3) System displays the user's past and upcoming bookings.

**Postconditions:** Booking history is displayed to the user.

## 5.5 Log-Out

**Description**: Allow users and hotel administrators to securely log out of their account.
**Actors:** User or Admin
**Preconditions:** User is logged in.

**Steps**:

1) User or administrator navigates to the log-out option on the platform.

2) User or administrator clicks on the log-out button.

3) System terminates the current session.

4) System redirects the user or administrator to the homepage or login page.
5) System displays a message indicating successful log-out.

**Postconditions:** Session is terminated.

## 5.6 Add New Hotels to the Platform (For Hotel Administrators)

**Description**: Allow hotel administrators to add new hotels to the platform.
**Actors:** Hotel Admin
**Preconditions:** Administrator is logged in.

**Steps**:

1) Administrator logs into their account.

2) Administrator navigates to the "Add Hotel" page.

3) Administrator enters hotel details (name, location, room types, prices, availability).

4)System validates the input.
5) System adds the new hotel to the platform.

**Postconditions:** New hotel is available in the system for customers to book and admins to modify.

**Alternate Flows:**

- If validation fails (e.g., missing fields) the system prompts for correction.

## 5.7 Generate Reports (For Hotel Administrators)

**Description**: Allow hotel administrators to generate reports on hotel performance and customer activity.
**Actors:** Hotel Admin
**Preconditions:** Administrator is logged in.

**Steps**:

1) Administrator logs into their account.

2) Administrator navigates to the reports page.

3) Administrator selects the type of report they want to generate.

4) System retrieves relevant data.

5) System generates the report and displays it.
6) Administrator downloads or prints the report if needed.

**Postconditions:** Report is generated (with the option to download the file).

**Alternate Flows:**

- If data is not available system displays an error message.

## Diagram Description:

The use cases demonstrate the most important user & admin interactions with the hotel booking system. They constructed a structure that is used as the blueprint for how the system should respond to various activities, ensuring consistency during software development. These use cases helped identify components such as identification, payment processing, and booking management.

Commented [VP43]: **Comment:** Need to discuss all you diagrams using a small paragraph

Commented [VP44R43]: **Answer:** A small paragraph explaining the diagram and its future implementation in the systems development

# 5.8 Diagrams:

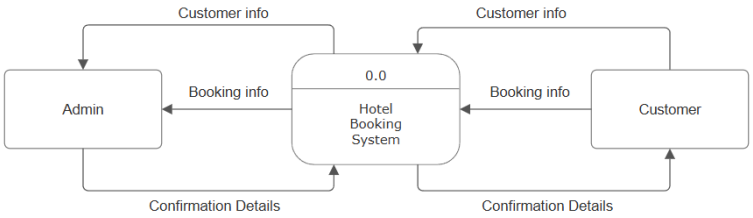



*Figure 5: DFD Level 0*

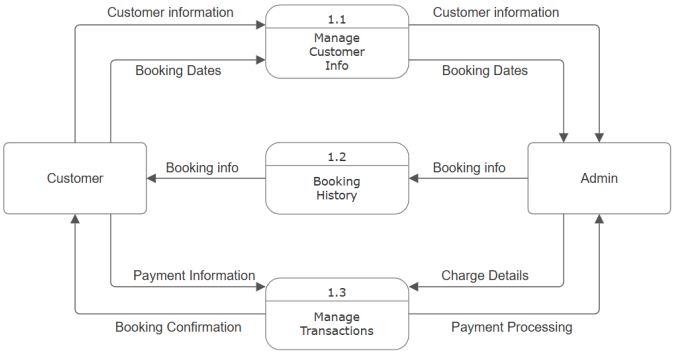



*Figure 6: DFD Level 1*

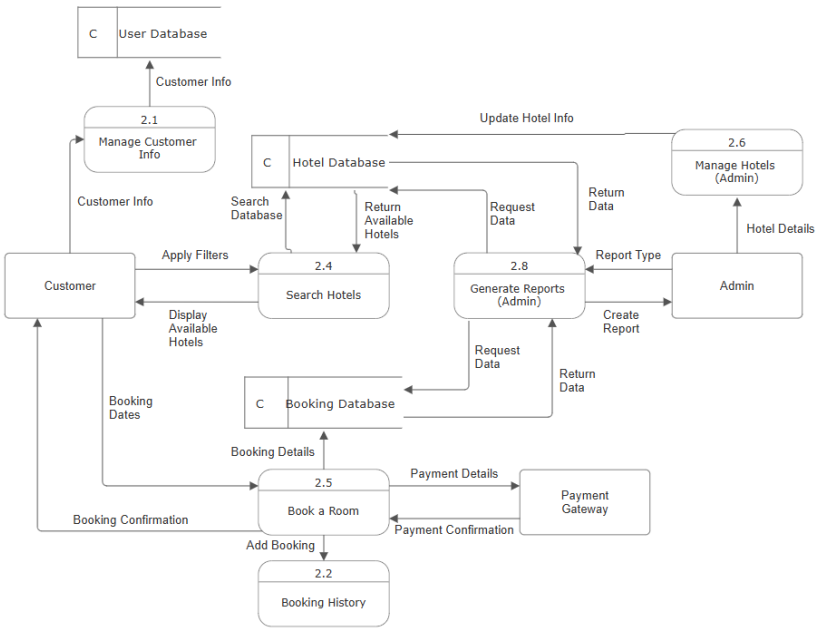



*Figure 7: DFD Level 2*

## Diagram Description:

These diagrams illustrate the core data flows between system components including customer info management, booking processing, and admin functions. It highlights how external systems like payment gateways integrate into internal databases. For implementation, this will guide our API endpoint design and database schema, ensuring proper data communication between bookings, hotels, and customer functions.

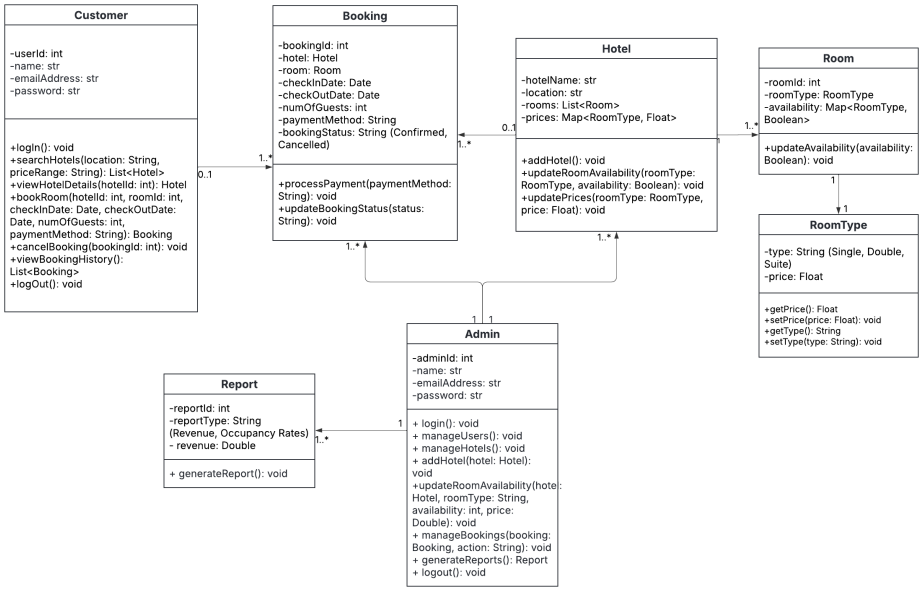



*Figure 8: Class Diagram*

## Diagram Description:

This diagram implements the use cases as system entities, with each class representing a core concept from the requirements, such as "Book Room" (Booking class) or "Manage Hotels" (Admin class). Attributes and methods correspond to specific use case stages like searchHotels(), which implements the "Search Hotels" feature. The structure ensures that all necessary activities from our use cases have supporting classes that interact with one another.

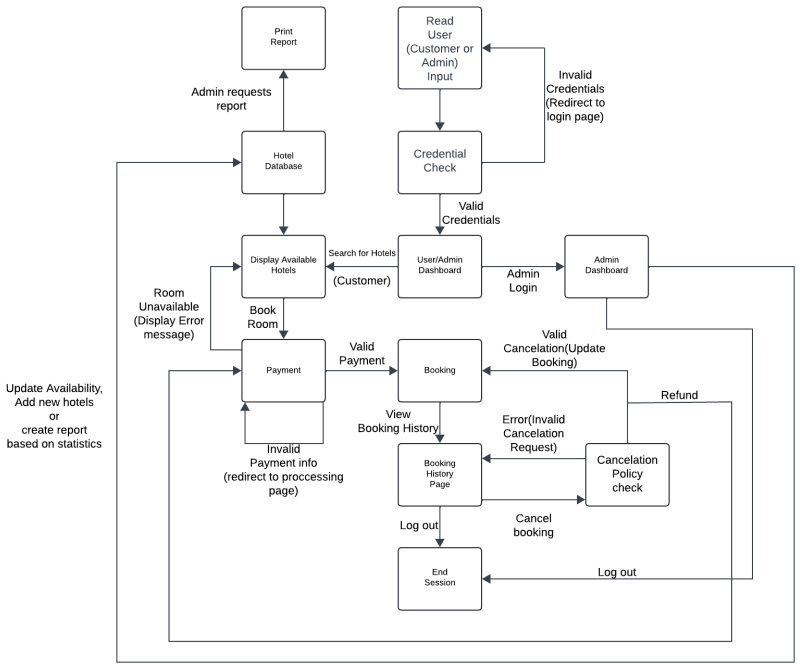



***Figure 9: Control Flow Diagram***

## Diagram Description:

This Diagram outlines the user's journey from login to booking and admin's workflows. It also highlights how the system responds to an error like wrong credentials through error messages. When actions are successful (like payments), the system saves the information and shows a confirmation message. This helps us program each step clearly, so users always know what's happening. When implementing the control flow diagram guides us by displaying the system's step-by-step logic and showcasing every possible user/admin action.

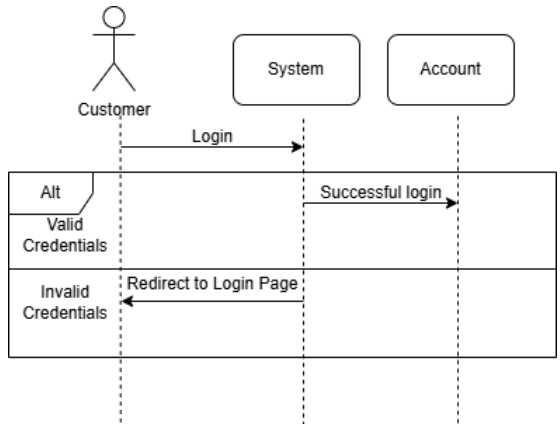



*Figure 10: Sequential Diagram (Login)*

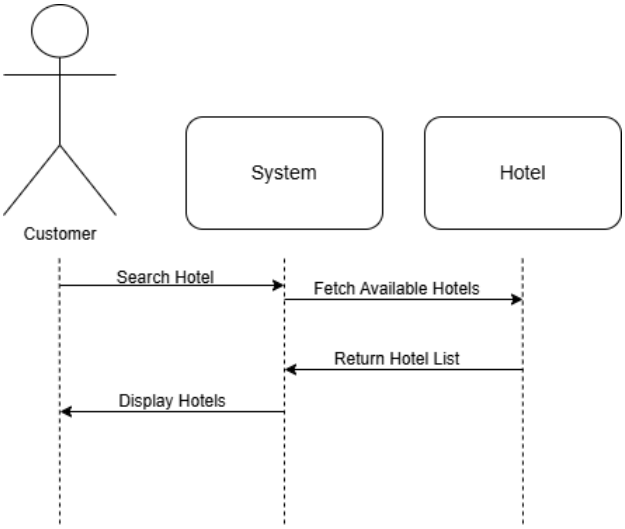



*Figure 11: Sequential Diagram (Search Hotel)*

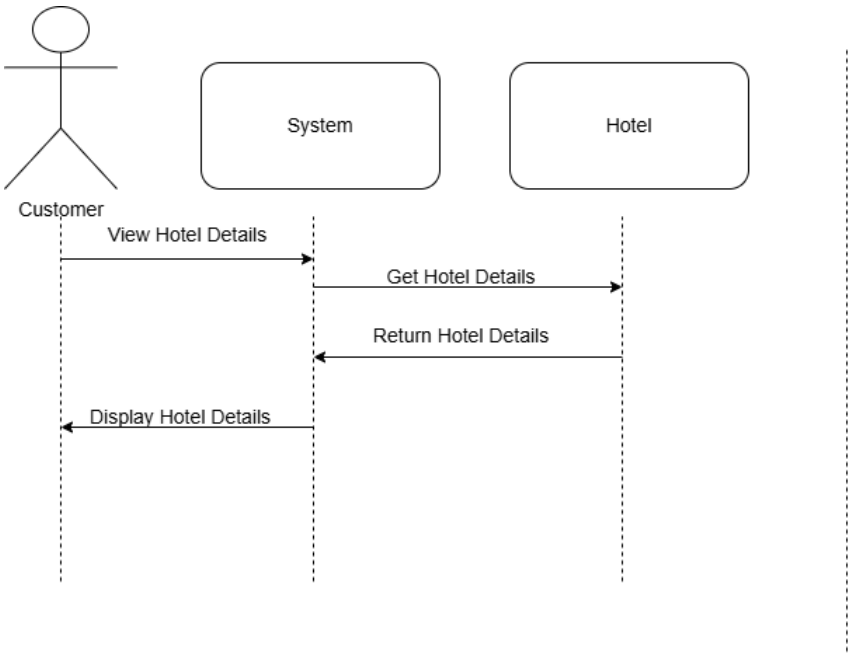



*Figure 12: Sequential Diagram (View Hotel Details)*

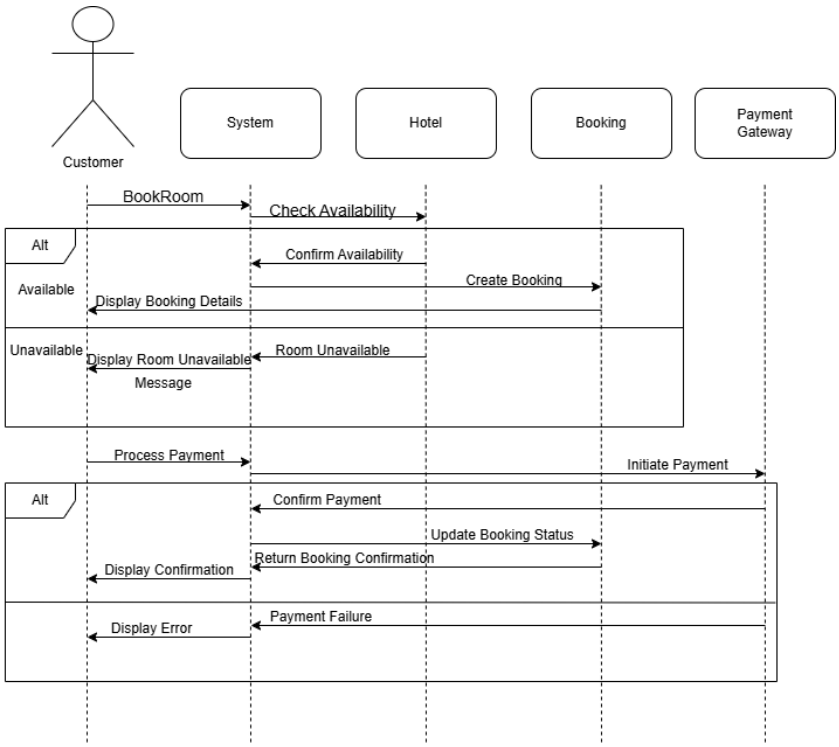



*Figure 13: Sequential Diagram (Booking & Payment)*

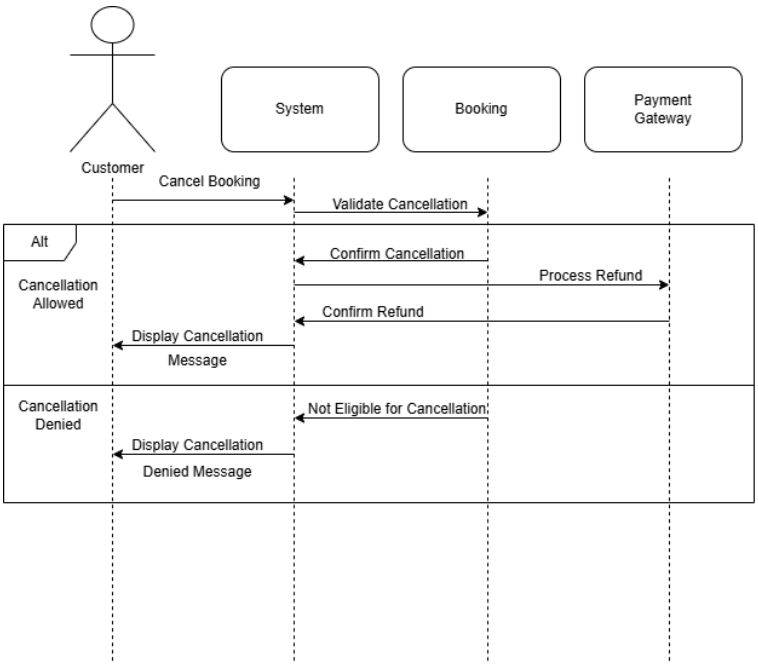



*Figure 14: Sequential Diagram (Cancellation & Refund)*

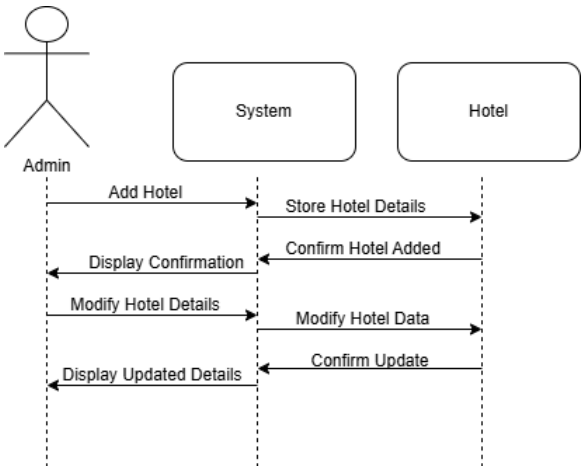



*Figure 15: Sequential Diagram (Add & Edit Hotel)*

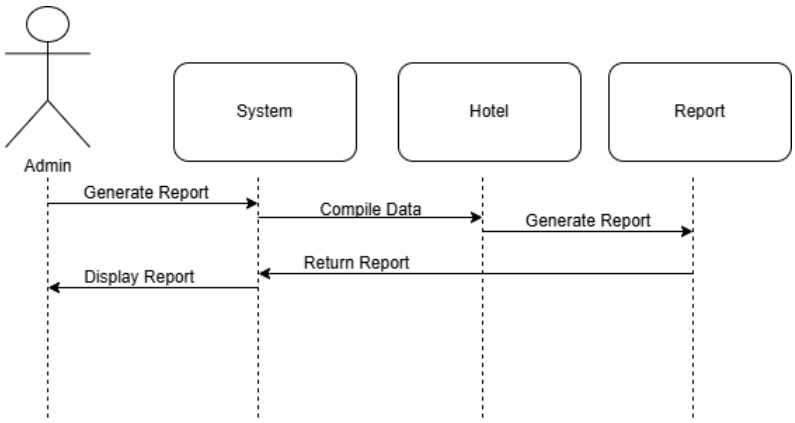



*Figure 16: Sequential Diagram (Create Report)*

## Diagram Description:

(Combining login, booking, cancellation, and admin flows)

These Sequential diagrams display the step-by-step process of the above actions and various possible outcomes of each (Wrong credentials, Cancellation eligibility etc.). This will help us figure out how a user and or an admin will interact with the system.

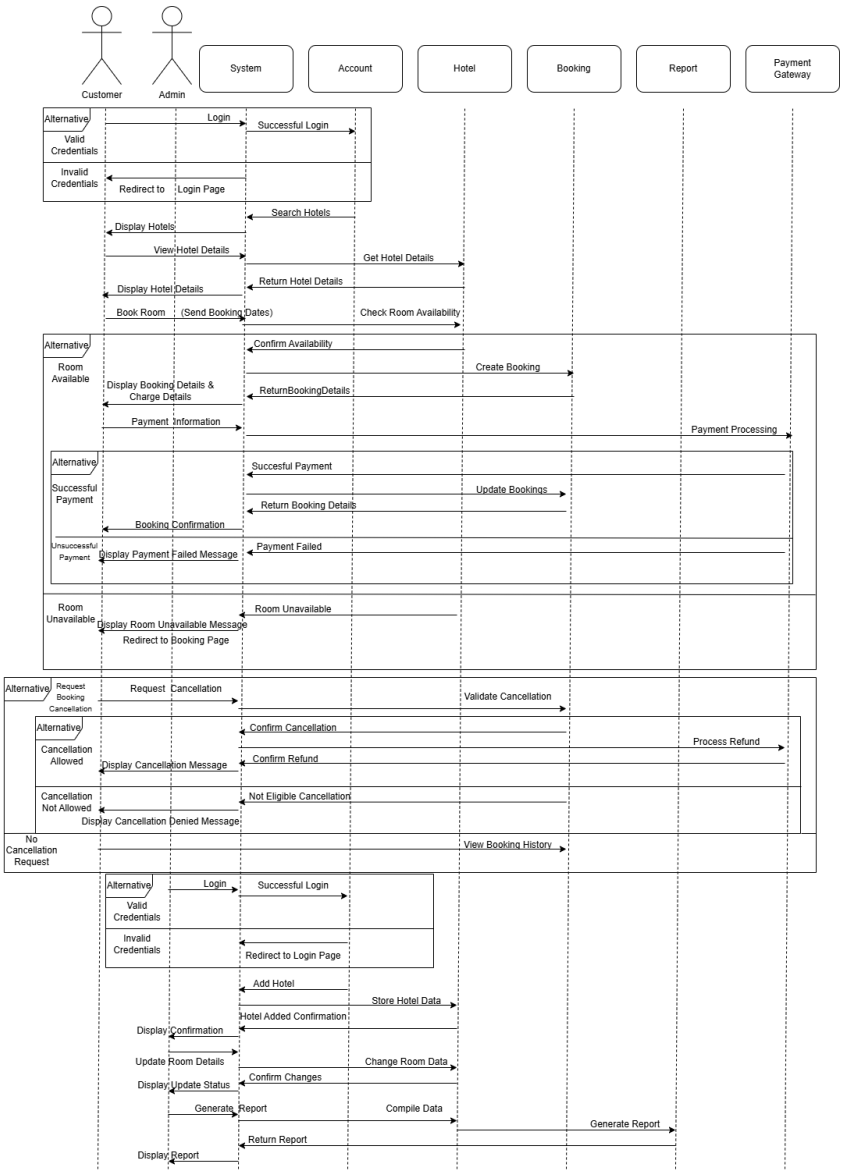



***Figure 17: Final Sequential Diagram (Complete process)***

# 6. Test Plan

## 6.1 Testing Strategy

We will test the hotel booking system using the following methods:
• Unit Testing: To check individual features like registration, login, and payment.
• Integration Testing: To see if different parts (like booking + payment) work together.
• System Testing: Full end-to-end testing of the whole flow (like signing up, booking a room, and paying).
• User Acceptance Testing (UAT): To make sure the system is easy to use and meets user expectations.
• Security Testing: To ensure data is secure, especially during login and payment.
• Performance Testing: To check how fast and stable the system is under multiple users.

To ensure thorough validation of the hotel booking system, our testing strategy combines both manual and automated methods across different levels of the application. Unit testing focuses on verifying that individual components like registration forms, login authentication, and payment processing behave as expected in isolation. Integration testing connects these components to ensure smooth interaction—for example, confirming that booking details are correctly passed to the payment gateway. System testing simulates real-world user scenarios to validate the entire workflow from start to finish. In User Acceptance Testing (UAT), we involve end users to evaluate Security and performance testing help ensure the application is robust, resistant to threats, and performs well under load. Whether the system meets functional requirements and provides a smooth, intuitive user experience.

### 6.2.1 Tools and Methods Used

• Manual Testing for most test cases.
• Postman for testing API calls.
• Selenium (if needed) for automated browser tests.
• Trello or Excel Sheet to track bugs and testing progress.

Manual testing involves executing test cases by hand without using automation tools. It allows testers to explore the application from a user's perspective and catch UI or usability issues that automated tests might miss. For backend testing, we use Postman, which helps us simulate API requests, validate responses, and ensure data is being exchanged correctly between the client and server. Selenium can be used where needed to automate repetitive browser actions like filling out forms or navigating pages, increasing efficiency during regression testing. To manage testing progress and track bugs, we rely on Trello boards or Excel sheets, where we record each test case, its current status (pass/fail), and any issues found.

## 6.2 Test Cases:

| Test Case ID | Requirement ID | Test Description | Input | Expected Result |
|---|---|---|---|---|
| TC1 | R1 | Try registering for a new account with valid details | Name, email, password | Account created and logged in |
| TC2 | R1 | Try logging in with a wrong password | Correct email, wrong password | Error message shown |
| TC3 | R2 | Search for rooms based on check-in and check-out dates | Dates and filters | List of available rooms appears |
| TC4 | R2 | Book a room after selecting dates and making payment | Room details, card info | Booking confirmed and email sent |
| TC5 | R3 | Try paying with incorrect card details | Fake card info | Payment fails and shows error |
| TC6 | R4 | Update your user profile info | New phone number, name | Profile updated successfully |
| TC7 | R5 | Admin disables a user account | Choose user from admin dashboard | User is deactivated |
| TC8 | R6 | Cancel a booking 30 hours before check-in | Booking ID | Booking is cancelled and refund initiated |
| TC9 | R7 | Request to delete user data | Click "Delete My Data" button | Data is deleted from the system |
| TC10 | R8 | Simulate 100 users booking rooms at once | Simulated data | System handles load with no errors and quickly |

| Requirement | Test Description | Input | Expected Result |
|---|---|---|---|
| User Authentication | Try registering for a new account with valid details | Name, email, password | Account created and logged in |
| User Authentication | Try logging in with a wrong password | Correct email, wrong password | Error message shown |
| Room Booking | Search for rooms based on check-in and check-out dates | Dates and filters | List of available rooms appears |

| Test Case ID | Requirement ID | Test Description | Input | Expected Result |
|---|---|---|---|---|
| Room Booking | Book a room after selecting dates and making payment | Room details, card info | | Booking confirmed and email sent |
| Payment System | Try paying with incorrect card details | Fake card info | | Payment fails and shows error |
| Profile Management | Update your user profile info | New phone number, name | | Profile updated successfully |
| Admin Panel | Admin disables a user account | Choose user from admin dashboard | | User is deactivated |
| Booking Cancellation | Cancel a booking 30 hours before check-in | Booking ID | | Booking is cancelled and refund initiated |
| GDPR | Request to delete user data | Click "Delete My Data" button | | Data is deleted from the system |
| Performance | Simulate 100 users booking rooms at once | Simulated data | | System handles load with no errors and within few seconds |

## 6.3 Traceability Table:

| Requirement ID | Test Case IDs Covered |
|---|---|
| R1 | TC1, TC2 |
| R2 | TC3, TC4 |
| R3 | TC5 |
| R4 | TC6 |
| R5 | TC7 |
| R6 | TC8 |
| R7 | TC9 |
| R8 | TC10 |

**Written by: Andreas Andreou**

# 7. Professional standards

The Hotel Booking System guarantees to professional standards in the following five wide categories: security, privacy, quality, engineering practices, and development methodology

First of all, these standards are drawn from the following sources:

- **International standards** (GDPR, ISO/IEC 25010, PCI DSS)
- **Best practices of modern software engineering**
- **Lecture notes of this course** (especially Software Engineering Process, Architecture Design, and Project Management slides)

To make it clearer how these standards relate to our system, here is a table where we matched each one with where exactly we applied it:

| Standard / Source | Where We Applied It in the System |
|---|---|
| **GDPR** | Used when the user signs up, edits their account, or deletes it. We only ask for info after consent. |
| **PCI DSS** | We don't handle card info ourselves. Instead, we use a third-party payment method that follows the rules. |
| **ISO/IEC 25010** | We tried to build everything clean and easy to change like using separate layers and keeping it modular. |
| **Modern Software Practices** | We followed software design practices by structuring the system Into clearly separated components each with its own purpose |
| **Lecture Notes** | Helped us decide how to organize the project, what features to build, and how to split the system. |

Thus the system becomes capable of assuredness, dependability, and readiness for the real deployment by following the above guidelines.

## 7.1 Data Security and User Privacy

On top of the list of concerns while constructing that particular system was a task of securing the privacy of the user and the data of a sensitive nature. As the platform involves the processing of personal and financial data, then we have guarded the protection of the data from the beginning by applying the current data protection practices. The question is about the availability and protection of the data.

### 7.1.1 GDPR Compliance

We made the system to be in line with the rules of GDPR  the regulation which stands for the protection of users' personal data in the EU. Our requirements from the users are limited to essential details and this happens only on their first visit to the site or at the time of booking, and we do not make them confused about the data usage issue. They can check, change or completely remove the data from our system. Our collection of data is solely based on the users' permission, which can be taken away any time they want.

We encrypt the passwords and never save them in plain text. Only an administrative staff who has access rights can trace the history of the bookings, and we did so deliberately, avoiding the storage of details such as credit card numbers that are sensitive.

### 7.1.2 Password and Data Encryption

Passwords are hashed with encryption algorithms like bcrypt that make the reversed finding of the original passwords almost impracticable. In addition, SSL/TLS and AES are the reliable and secure cryptographic protocols which being applied to ensure an information security transparency of the transactions. These technologies provide strong protection for the data whether it is stored or transmitted.

### 7.1.3 PCI-DSS Compliance

To undertake the amount related transactions in a safe and secure manner, we have embedded a fully compliant with the PCI-DSS standard third-party payment gateway. As a result of this move, the customers' private financial details will in no way be in our possession, which ensures the ultimate level of risk reduction for the consumers and the hoteliers.

## 7.2 Software Quality and Reliability

From the very outset of the project, our primary objective was not merely to develop a functioning system but to develop a system that is functioning well, a system, that is moreover, maintaining the functionality. To be the stepping stone, from this point of our path, we utilized ISO/IEC 25010 as the governing document that defines eight quality characteristics that mark software of high quality. These characteristics cover usability, reliability, performance, efficiency, maintainability, and even more.

In particular, we took the following aspects into account in our project:

- **Usability**, that is to say, the system is not only easy and convenient but also clear for people with no IT background. The system should be usable by consumers even not belonging to this domain of IT.

- **Reliability**, through which we made certain that all the central features of the system are the expected ones created by the normal use of the system.
- **Performance**, we made efforts in order to make web pages load quicker and the system run stably. In our approach, the capacity of the machine to respond efficiently to high loads without breakdowns is a priority and therefore treated the same way.
- **Efficiency**, the system has been designed in a manner where no computational resources are unnecessarily used. Take, for instance, a blog post that only requests necessary information. In addition, it is still possible for the system to be light and quick, when it is being accessed by many users, due to the restrictions in the number of operations being executed.
- **Maintainability**, to be exact, we were accessed by the effective and easily maintainable code that was soft and fuss-free and that could be easily updated or fixed later if there had been any reasons to do so.

These ways have assured us that the work that we have done is compliant with the requirements of ISO/IEC 25010. Thus, it is confirmed that the software we have developed not only has the quality we expect but it is also consistent and safe for users and hotel managers.

## 7.3 Software Engineering Practices

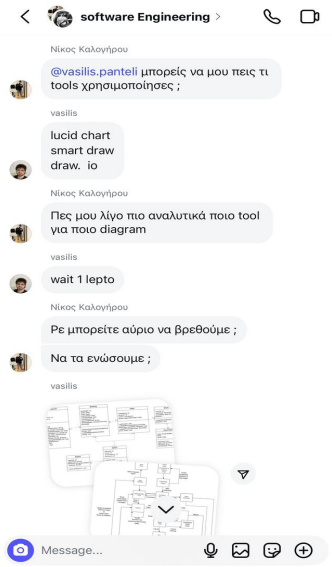
At the inception of the project, we were concentrated on developing a software product that was both usable and understandable for the customers as well as for developers. We did not completely stick to any traditional procedures, though, we were proud that we did practice good software engineering habits. On top of our list was always writing clean, user-friendly code as well as creating a system that would be simple to fix and expand later if the need arises.

### 7.3.1 Modularity & Separation of Concerns

That means we split the system into small pieces, with only 1 function per piece. For example, one part would be responsible for the authentication of the user, another one for the reservation and the other one for the preparation of the reports. It became much simpler to do each of the tasks without interfering with other parts of the system. If any part of the system was not working as expected, the first thing we did was to go directly to the specific module, get it fixed, and then continue running the rest of the system smoothly. This really was a great time-saver and kept the bugs from turning into bigger issues in the future.

### 7.3.2 Team Communication

We were keeping it simple in terms of project management tools. Our contacts remained mostly in the direct messages of Instagram. This sounds like a casual way to do it, but it worked well as everyone was constantly communicating, sharing updates, assigning tasks, and resolving problems in collaboration. In addition to that, we did not use GitHub or any other version control tool. Instead, we just shared the files with one another and then kept a record of who did what. It lacked sophistication, but it was a very powerful method for us.





### 7.3.3 Manual Testing

Once a segment of the system was completed, it was our turn to manually test it to ensure that everything worked. We all carefully went through the logins, filled in the forms, and step by step, made a reservation of a room.

## 7.4 Project Management & Methodology

Our team stuck with the basic idea of the Agile methodology from the very start and made the decision to simply keep things easy and changeable. We didn't use any official tools such as Jira or Trello, but we were still able to keep order and be flexible by being in the spirit of Agile which included numerous small, regular plans, communication, and being open to every possible change through regular review meetings.

The starting point for the hotel booking system functionalities discussion was a team discussion. The primary steps were to create a list of the core things that had to be done login, booking, admin panel, and reports and then to proceed with each feature one by one, as well as giving each one the confidence it deserved. Some of our team members were front-end developers, some were back-end, while others worked on the database.

Mainly, Instagram direct chat was our preferred communication channel. Despite no complicated workflows being created by us, still we communicated nearly on a daily basis to update ourselves and give each other support wherever it was necessary. Undoubtedly, the project never came to a halt, and we could sail smoothly to the finish line without facing any troubles.

We had a habit of holding a weekly meeting where we set targets to be achieved and discussed the accomplished work. A feature we were developing, in case of any changes or the task consuming more time than initially expected, that was the occasion to modify the plan and move on. By doing so, the whole process became both less stressful and more efficient.

Even when we were not exactly Agile, the method that we use to complete tasks and made us keep up with the schedule that of the essence of productivity.

## 7.5 Ethical and Sustainable Engineering

We have been a part of the project committed ourselves to the ethical standards of organizations such as ACM and IEEE.

For us, the codes were workable not just words. Every time we designed a new piece of the system, we tried to embed those codes in it. Our commitment has never been more certain.

Regarding our privacy and trust towards the users, we had their privacy and trust kept intact right from the beginning. Meaning that we not only never requested unnecessary information from the users but also were their data managers all the time so we were never influenced by the desire of seeking unnecessary information that we did not need about the users. We did not even remotely suggest the bad practice of saving the passwords in a plain text format or using hardcoded credentials. Instead it was our only option to use strong encryption and then, we provided authorization to people who needed it.

Transparency was the very first issue dealt with. We offered practical evidence that users were aware of their rights to their own data and their choices they were making. There is no such thing as deception or confusion in the designs and buttons. The policy of honesty was pursued. Our intention was that we would not want anyone to think they had been misled or to doubt the result if they had touched something.

Ultimately, we made sure that the system was extremely simple such that not only it was user-friendly and inviting to all the users but also to the ones who are not so familiar with the latest technology. The layout is clear, the steps are straightforward, and the language we used in the interface is simple. Our conviction is that the good available software is that which not only guards the information of the users but also ensures the system is friendly and secure for all.

**Written by: Christos Mouchlis**

**Commented [CM45]:** I just wanted to clarify something regarding the previous version of Part 7. If it gave the impression of being AI-generated, that might be because I used tools like Google Docs suggestions and AI-based assistants not to write the content, but to improve the clarity of the wording. The content was written entirely by me from the beginning. I only used those tools to improve the appearance and make the text easier to read and understand. I have now reviewed and updated the entire **Professional Standards** section, and it reads more naturally. Thank you very much for your understanding, and I apologize for any confusion...