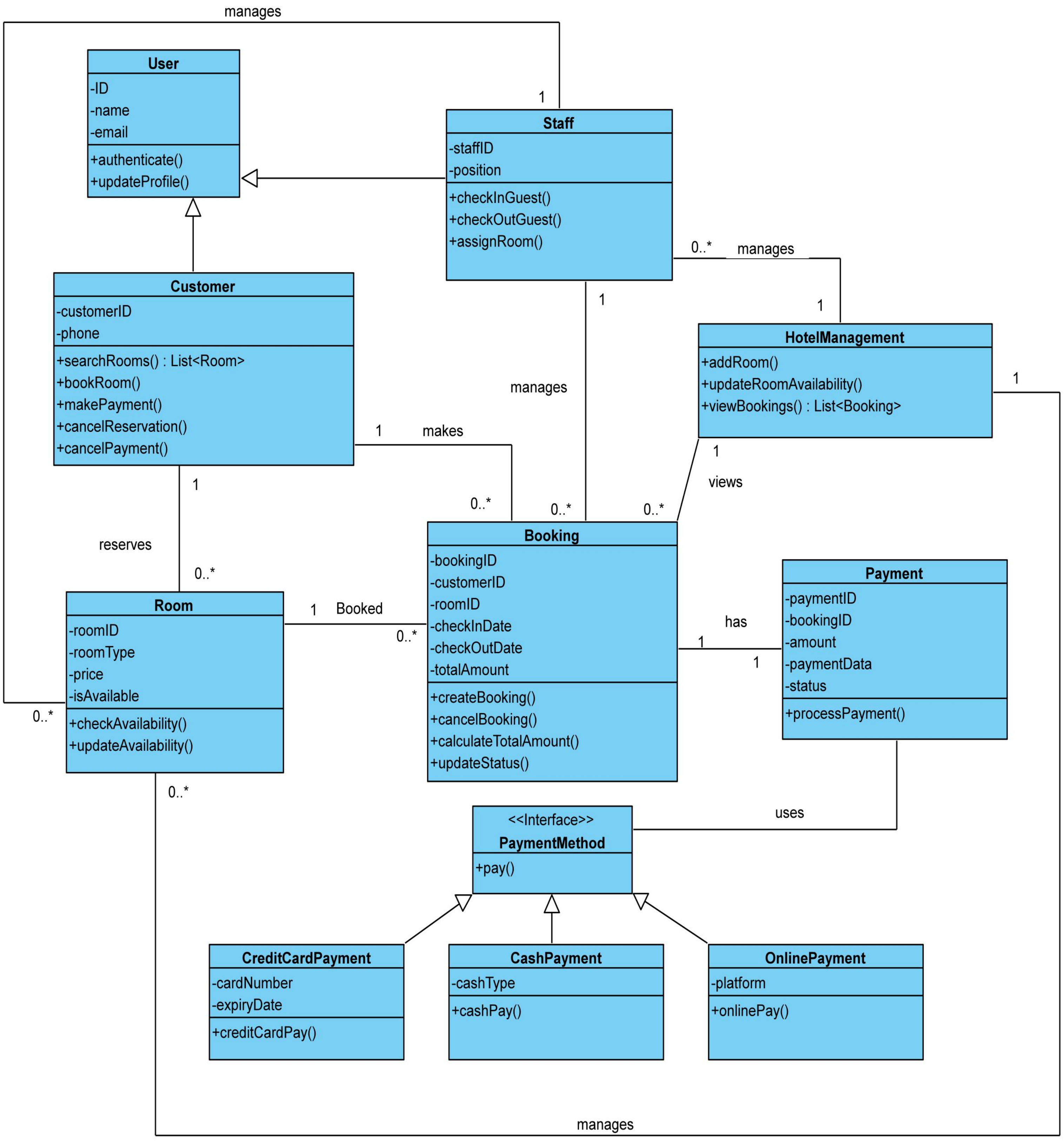
CPT203
Coursework 2

2024/2025 Semester 1
<2024.12.12>

Group number: 4
Student 1 Name: Yize Liu Student 1 ID: 2254472 who submits this coursework on the learning mall

Student 2 Name: Shengtian Huang Student 2 ID: 2254461
Student 3 Name: Qing Qin Student 3 ID: 2254084
Student 4 Name: Xu Chen Student 4 ID: 2257453
Student 5 Name: Zichen Qiu Student 5 ID: 2252705

Q1. You are supposed to draw the class diagram for the case (10 marks).



Q2. Evaluate the design principles in the class diagram. (15 marks).

1. Abstraction Principle

The abstraction principle simplifies system design and improves flexibility, maintainability, and scalability by extracting core features and hiding details.

1.1 Data Abstraction

Data abstraction extracts the core characteristics of data while hiding implementation details, allowing developers to focus on “what” rather than “how.” The attributes within various classes represent the essential data required by each object, concealing the specific implementation details. This reduces coupling between modules, making the system more modular.

1.2 Procedural Abstraction

Procedural abstraction describes system behavior by defining functionalities through methods rather than detailing their implementations. This approach enhances maintainability by allowing developers to concentrate on using the abstract interfaces provided by classes without worrying about underlying implementations. For example, when calling the checkAvailability() method, one only needs to know that it returns the room’s status, without understanding how it’s achieved internally. Such methods encapsulate specific operations and hide concrete implementation details. Furthermore, it increases reusability; for instance, both the PaymentMethod and Booking modules can easily reuse and extend functionalities by implementing a unified interface, eliminating the need to modify the existing system.

2. Modularity Principle

Class diagrams embody the modularity principle by breaking down the system into different classes (such as Customer, Room, Booking), each representing an independent module responsible for a specific functionality. This makes the system easier to understand, maintain, and expand.

The hotel management system exemplifies modular design through specialized modules like User, Staff, Customer, Room, Booking, and Payment. This structure reduces complexity, improves efficiency, and enhances scalability. Modular communication interfaces are evident in interactions between modules, such as Booking interacting with Room and Payment modules through method calls and interfaces. The Customer class interacts with Room and Booking interfaces for reservations. This approach reduces coupling, increases flexibility, and improves extensibility. New functionalities can be added without extensive modifications to the existing system, making the overall design more manageable and comprehensible.

3. Functional Independence Principle

The principle of functional independence emphasizes the ability of modules or components to operate autonomously, achieved through low coupling and high cohesion.

3.1 Coupling

Coupling measures module interdependence, and this design demonstrates low coupling through interfaces and method-based communication. Examples include Booking interacting with payments via the PaymentMethod interface, Room communicating with Booking through methods, and the PaymentMethod interface with its implementations. This approach allows for independent module development and modification, improving maintainability, extensibility, and stability while reducing maintenance costs and the impact of changes across modules.

3.2 Cohesion

Cohesion refers to the level of relatedness and focus within a module’s internal elements. High-cohesion modules concentrate on a single function. For example, the Room class manages room properties and states, while the Customer class focuses on customer-related behaviors. High cohesion makes modules more focused, making the code easier to understand, maintain, and reuse. High cohesion improves readability and reusability of the code since module functionalities are clear and related tasks are grouped within a single module.

4. Object-Oriented Design Principles

Object-oriented design principles leverage encapsulation, inheritance, and polymorphism to organize systems, enhancing modularity, reusability, and extensibility.

4.1 Encapsulation

Encapsulation binds data and methods together and controls their access, hiding implementation details. For example, the Room class encapsulates room information through its attributes and methods, while the Payment class encapsulates payment data and logic. Encapsulation hides internal implementations, reduces external coupling, and protects data integrity. By employing encapsulation, system modularity improves, with each module focusing on its own responsibilities and thus lowering overall complexity.

4.2 Inheritance

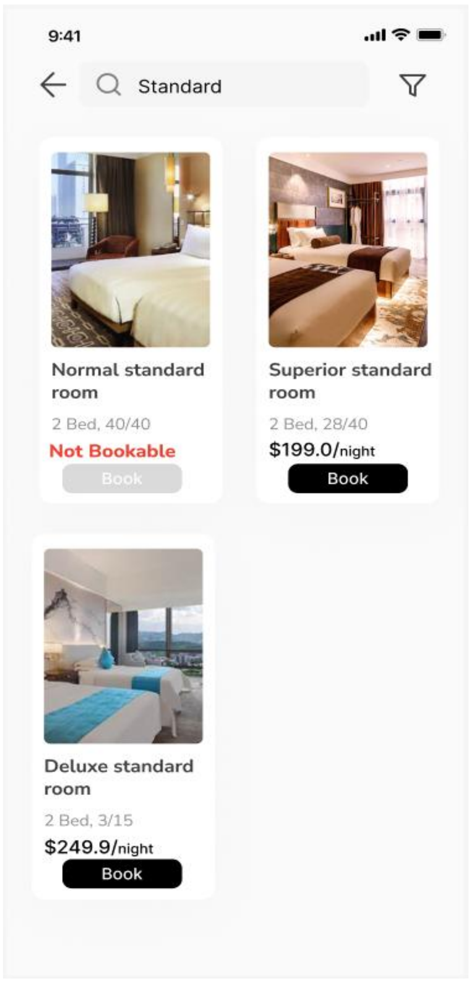
Inheritance allows one class to be created based on another, reusing code and extending or overriding behaviors. For example, the PaymentMethod interface is inherited by multiple payment classes, and User is inherited by Staff and Customer. Inheritance reduces code duplication, increases code reusability, and enhances system scalability, making it simple to introduce new entity types.

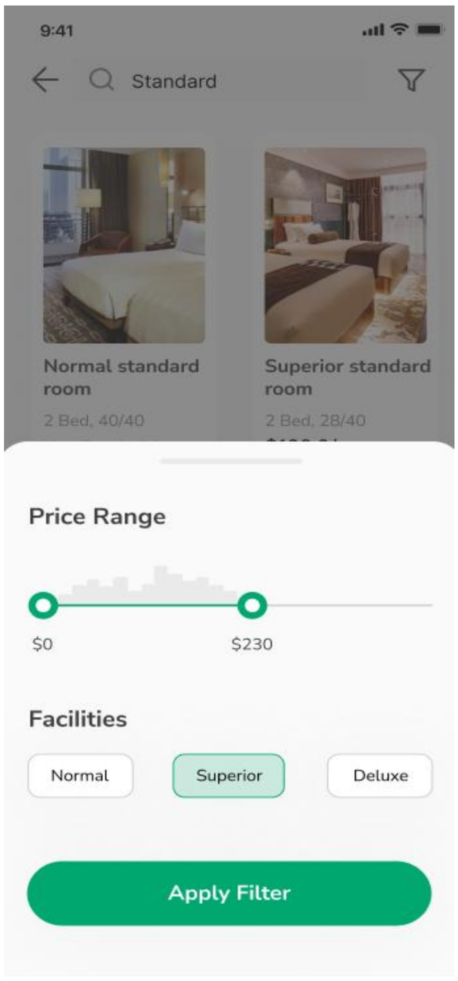
4.3 Polymorphism

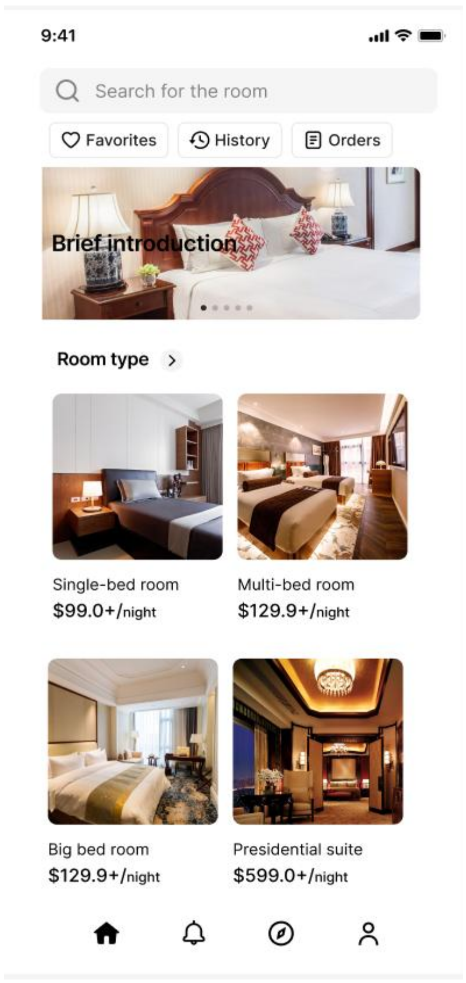
Polymorphism allows different objects to respond in the same way to the same message, enabling dynamic method selection. For instance, the Booking module completes payment through the PaymentMethod interface's pay() method without worrying about the specific payment method details. Polymorphism increases system flexibility by allowing the appropriate behavior to be selected at runtime, reducing code complexity since callers don't need to know the object's exact type.

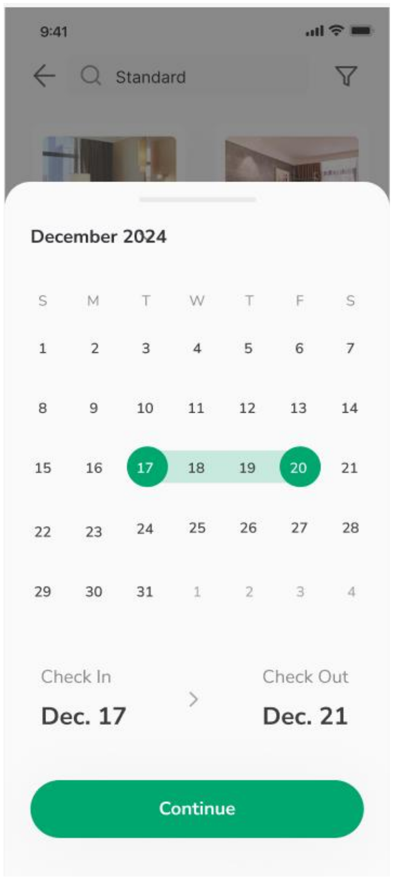
Q3. You are required to create the UI pages of these functions. (15 marks)

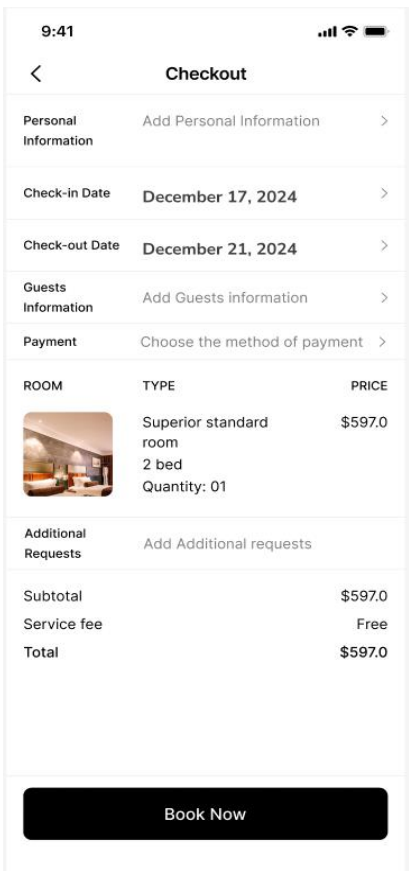
Customer Room Search and Booking:









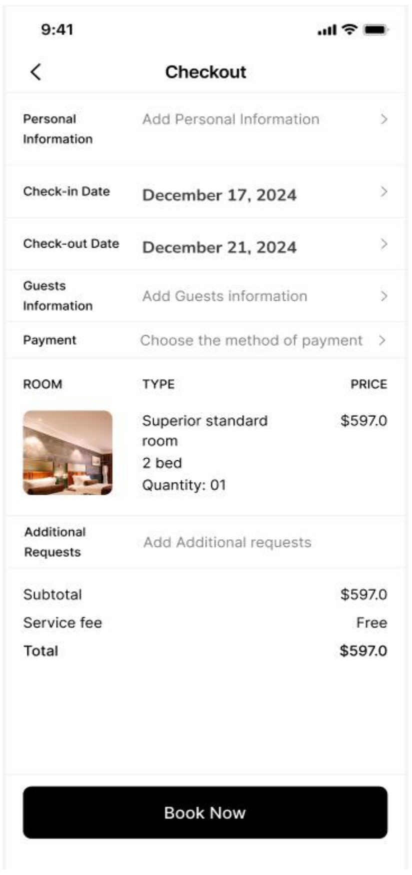


Personal Information	Add Personal Information
Check-in Date	December 17, 2024
Check-out Date	December 21, 2024
Guests Information	Add Guests information
Payment	Choose the method of payment

ROOM	TYPE	PRICE
	Superior standard room 2 bed Quantity: 01	$597.0

Additional Requests	Add Additional requests
Subtotal	$597.0
Service fee	Free
Total	$597.0

Payment Processing:


9:41

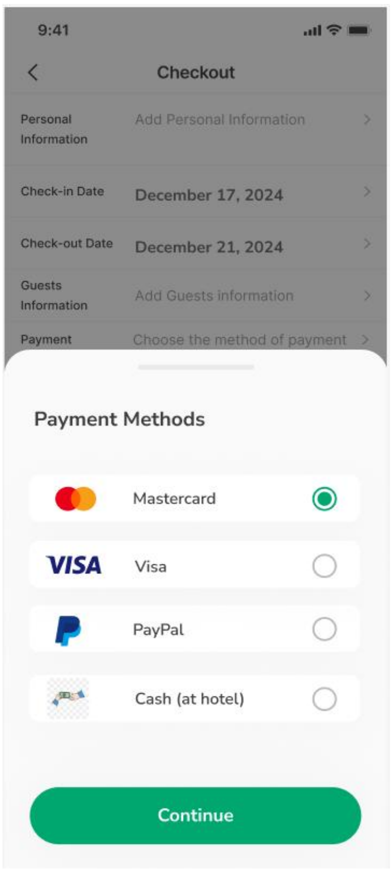
Checkout

Personal Information	Add Personal Information
Check-in Date	December 17, 2024
Check-out Date	December 21, 2024
Guests Information	Add Guests information
Payment	Choose the method of payment

ROOM	TYPE	PRICE
	Superior standard room 2 bed Quantity: 01	$597.0

Additional Requests	Add Additional requests

Subtotal	$597.0
Service fee	Free
Total	$597.0


9:41

Checkout

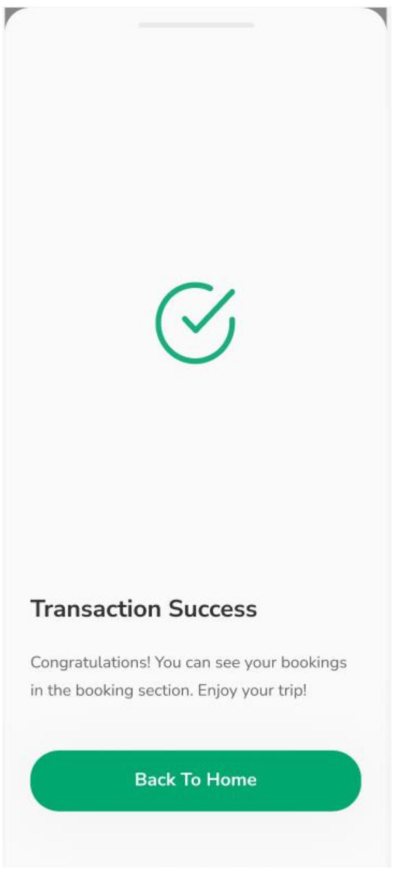
Payment Methods

[✓ Mastercard]

Visa

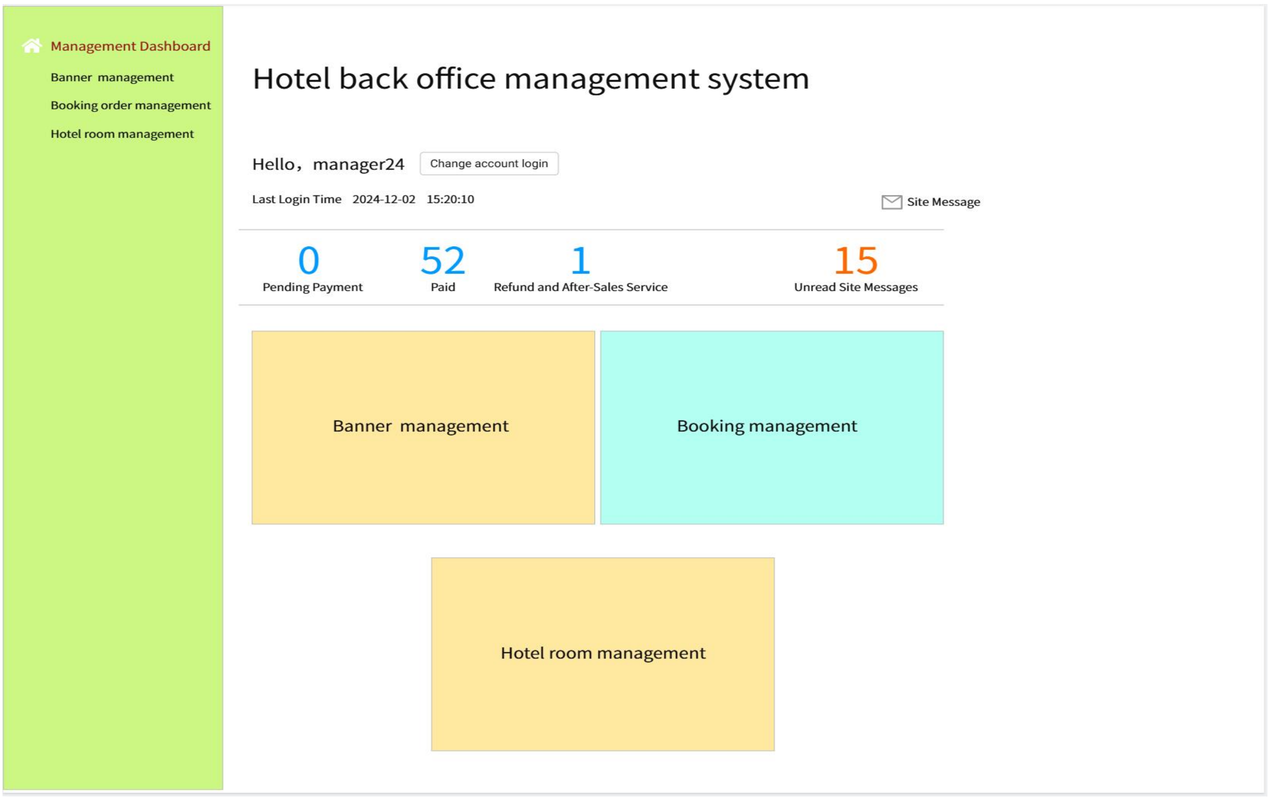
PayPal

Cash (at hotel)


Transaction Success

Congratulations! You can see your bookings in the booking section. Enjoy your trip!

Room Booking Management:



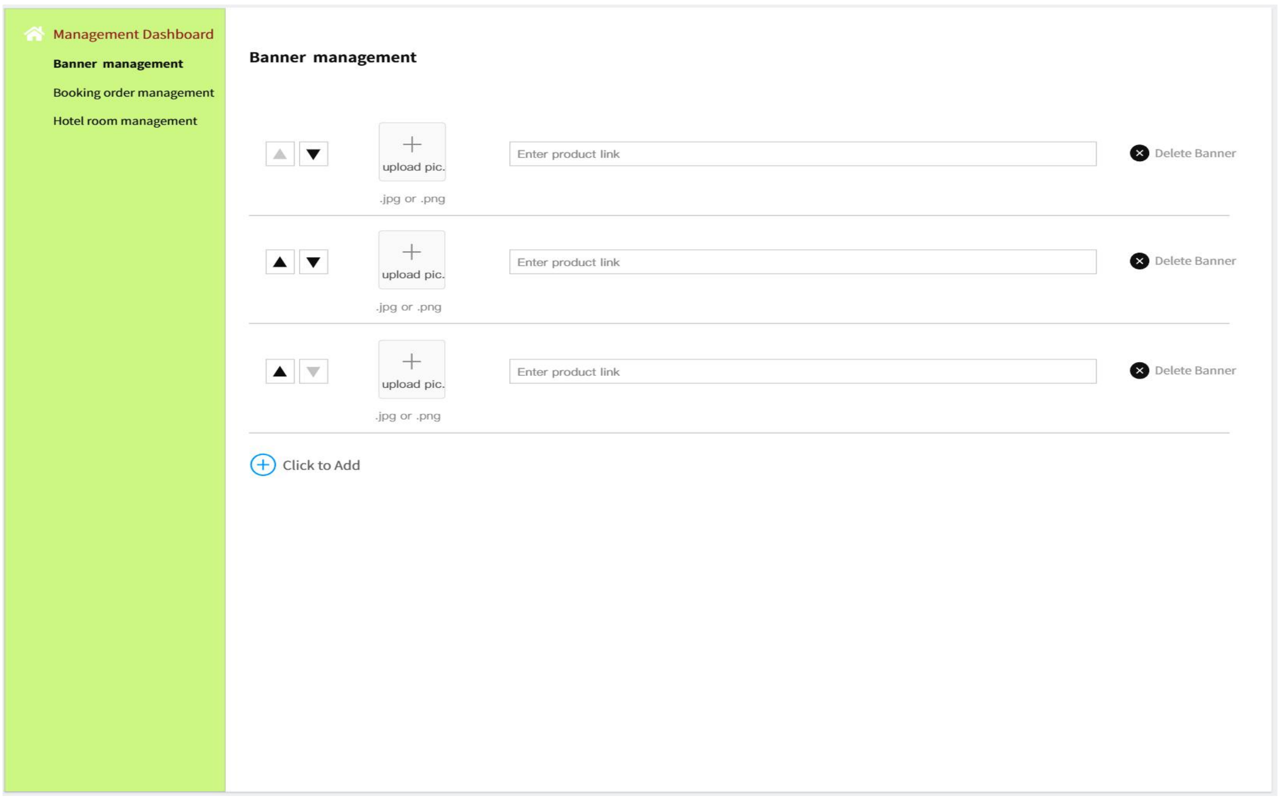
Hotel back office management system

Hello，manager24 Change account login

Last Login Time 2024-12-02 15:20:10

Site Message

0	52	1	15
Pending Payment	Paid	Refund and After-Sales Service	Unread Site Messages


Management Dashboard

Banner management

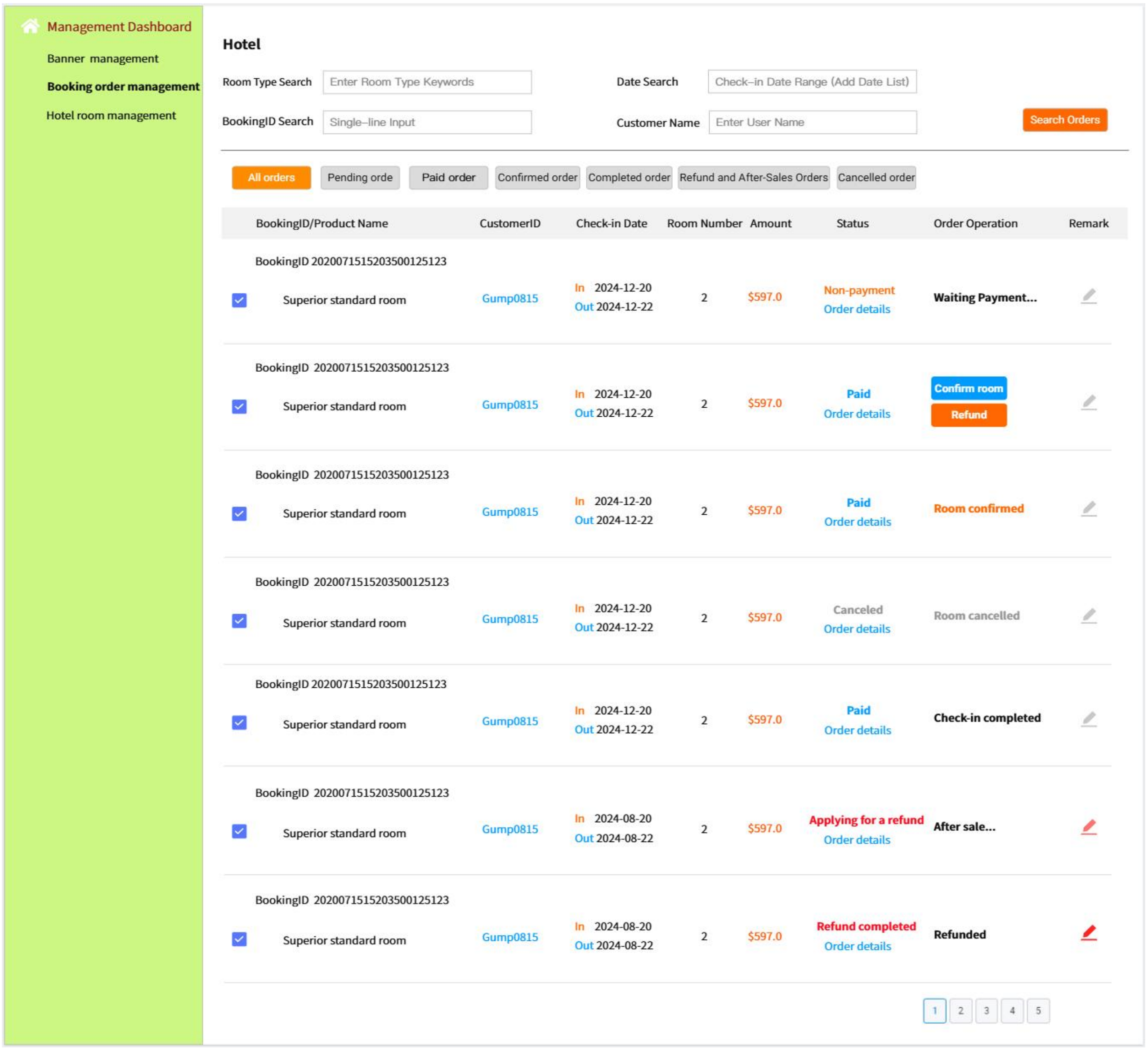
Booking order management

Hotel room management

Banner management

▲ ▼	upload pic. .jpg or .png	Enter product link	Delete Banner
▲ ▼	upload pic. .jpg or .png	Enter product link	Delete Banner
▲ ▼	upload pic. .jpg or .png	Enter product link	Delete Banner

Click to Add


Management Dashboard

Banner management

Booking order management

Hotel room management

Hotel

Room Type Search: Enter Room Type Keywords

Date Search: Check–in Date Range (Add Date List)

BookingID Search: Single–line Input

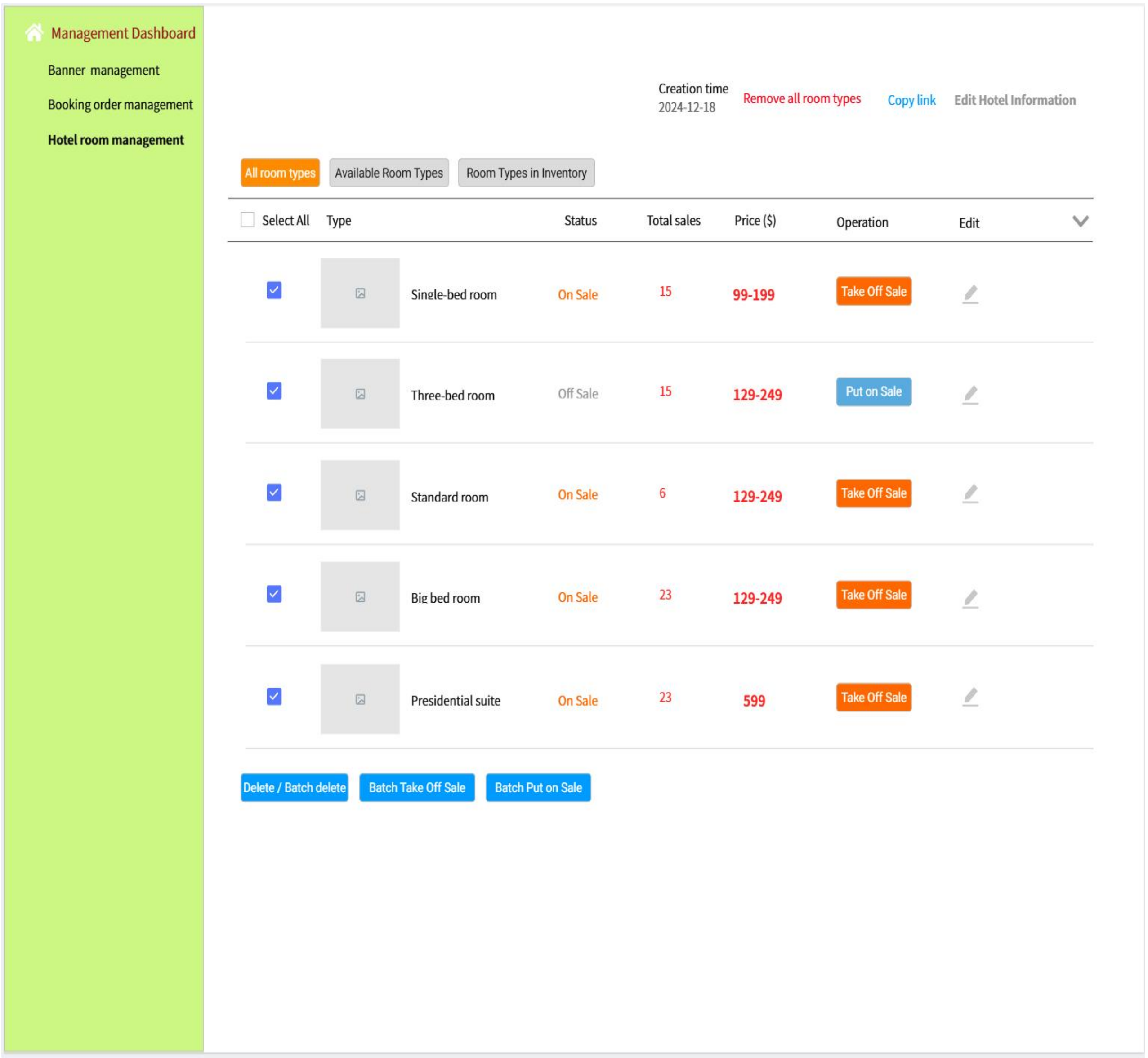
Customer Name: Enter User Name

Search Orders

All orders | Pending orde | Paid order | Confirmed order | Completed order | Refund and After-Sales Orders | Cancelled order

BookingID/Product Name	CustomerID	Check-in Date	Room Number	Amount	Status	Order Operation	Remark
BookingID 2020071515203500125123 ✓ Superior standard room	Gump0815	In 2024-12-20 Out 2024-12-22	2	$597.0	Non-payment Order details	Waiting Payment...	✎
BookingID 2020071515203500125123 ✓ Superior standard room	Gump0815	In 2024-12-20 Out 2024-12-22	2	$597.0	Paid Order details	Confirm room Refund	✎
BookingID 2020071515203500125123 ✓ Superior standard room	Gump0815	In 2024-12-20 Out 2024-12-22	2	$597.0	Paid Order details	Room confirmed	✎
BookingID 2020071515203500125123 ✓ Superior standard room	Gump0815	In 2024-12-20 Out 2024-12-22	2	$597.0	Canceled Order details	Room cancelled	✎
BookingID 2020071515203500125123 ✓ Superior standard room	Gump0815	In 2024-12-20 Out 2024-12-22	2	$597.0	Paid Order details	Check-in completed	✎
BookingID 2020071515203500125123 ✓ Superior standard room	Gump0815	In 2024-08-20 Out 2024-08-22	2	$597.0	Applying for a refund Order details	After sale...	✎
BookingID 2020071515203500125123 ✓ Superior standard room	Gump0815	In 2024-08-20 Out 2024-08-22	2	$597.0	Refund completed Order details	Refunded	✎

1 2 3 4 5


Management Dashboard

Banner management

Booking order management

Hotel room management

Creation time 2024-12-18 | Remove all room types | Copy link | Edit Hotel Information

All room types | Available Room Types | Room Types in Inventory

Select All		Type	Status	Total sales	Price ($)	Operation	Edit
✓		Single-bed room	On Sale	15	99-199	Take Off Sale	✎
✓		Three-bed room	Off Sale	15	129-249	Put on Sale	✎
✓		Standard room	On Sale	6	129-249	Take Off Sale	✎
✓		Big bed room	On Sale	23	129-249	Take Off Sale	✎
✓		Presidential suite	On Sale	23	599	Take Off Sale	✎

Delete / Batch delete | Batch Take Off Sale | Batch Put on Sale

Q4. For each of the UI page, (1) illustrate what interface design principles are used, (2)how they are applied with specific examples, and (3)how they improve the interaction between the system and users (e.g., equality, diversity and inclusion). (20 marks)

For Customer Room Search and Booking UI:

1. **Empowering the User:** 1.1 Defining Interaction Modes: Example: Users choose "Favorites," "History," or "Orders" from tabs. Improvement: Users control what they see based on their needs, increasing autonomy and satisfaction. 1.2 Flexible Interaction Methods: Example: Users select "Normal," "Superior," or "Deluxe" to filter rooms. Improvement: Tailored results meet varying preferences, enhancing flexibility and personalization. 1.3 Allowing Interruptions/Undo: Example: Users can go back a step or cancel a reservation anytime. Improvement: Users can correct errors or change their minds easily, reducing anxiety. 1.4 Direct On-Screen Interaction: Example: Clicking a room image shows details or initiates booking. Improvement: Direct interaction lowers cognitive load and streamlines the process. 1.5 Visible System Status: Example: Users see room availability (e.g., "Not Bookable"). Improvement: Immediate feedback clarifies choices and reduces uncertainty.

2. **Reducing Memory Load:** 2.1 Minimize Short-Term Memory Needs: Example: Room types/prices appear under images. Improvement: Less memory demand speeds decision-making and supports inclusivity. 2.2 Meaningful Defaults: Example: A price slider defaults to common ranges. Improvement: A sensible default speeds searching and reduces cognitive effort. 2.3 Real-World Metaphors: Example: A calendar interface resembles a real calendar. Improvement: Familiar layouts lower the learning curve and increase accessibility.

3. **Maintaining Interface Consistency:** 3.1 Consistent Visual Design: Example: Uniform buttons, icons, and colors. Improvement: Consistency helps users quickly recognize controls, reducing confusion. 3.2 Consistent Navigation/Layout: Example: A fixed navigation bar across pages. Improvement: Users navigate smoothly without relearning layouts,

improving efficiency. 3.3 Consistent Feedback/Prompts: Example: Dates highlight upon selection. Improvement: Uniform feedback reassures users and builds trust, especially for novices.

For Payment Processing UI:
1. **Empowering the User:** 1.1 Defining Interaction Modes: Example: Users review all booking details to ensure correctness. Improvement: Clear summaries let users confirm choices, increasing control, trust, and satisfaction. 1.2 Flexible Interaction Methods: Example: Users pick their preferred payment method. Improvement: Multiple options increase control and accommodate different payment habits, enhancing inclusivity. 1.3 Visibility of System Status: Example: Immediate feedback after a successful transaction. Improvement: Prompt confirmation reassures users and boosts confidence, especially for those less comfortable with online transactions.
2. **Reducing Users' Memory Load:** 2.1 Minimizing Short-Term Memory: Example: Booking details (room type, price, dates) are shown directly. Improvement: Less to remember speeds confirmation, increasing efficiency and accuracy. 2.2 Meaningful Defaults: Example: Previously used Mastercard is automatically selected. Improvement: Intelligent defaults reduce choices, simplifying and speeding up the payment process. 2.3 Real-World Metaphors: Example: Selecting payment icons resembles choosing in a store. Improvement: Familiar metaphors lower the learning curve, improving intuitiveness and usability.
3. **Maintaining Interface Consistency:** 3.1 Consistent Visual Organization: Example: Uniform fonts, colors, and layouts match the rest of the application. Improvement: Visual consistency helps users recognize and understand elements more easily. 3.2 Consistent Feedback: Example: Selected Mastercard highlights like other chosen elements. Improvement: Uniform feedback confirms user actions, building trust in system responsiveness.

For Room Booking Management UI:
1. **Empowering the User:** 1.1 Interruptions and Undo: Example: Users can click "Change account login" to modify account info or log out at any time. Improvement: Direct control over login enhances user autonomy and security. 1.2 Flexible Interaction Methods: Example: Users upload or delete banners via "upload pic" or "Delete Banner." Improvement: Intuitive options let users easily manage banners, improving flexibility and convenience. 1.3 Direct On-Screen Interaction: Example: Users click "Order details" to view specific booking info directly. Improvement: Direct links reduce steps needed to find details, improving the experience. 1.4 Visibility of System Status: Example: Real-time room statuses and sales data are shown. Improvement: Immediate insights help users make informed decisions, like adjusting prices.
2. **Reducing Users' Memory Load:** 2.1 Minimizing Short-Term Memory:Example: Key metrics display directly, so no memorization is needed. Improvement: Less to remember means faster, more accurate decisions. 2.2 Meaningful Defaults:Example: Frequently used banners appear at the top automatically. Improvement: Intelligent defaults reduce searching and cognitive load. 2.3 Real-World Metaphors: Example: Bookings resemble real-world tables or lists. Improvement: Familiar layouts lower the learning curve, enhancing usability.
3. **Maintaining Interface Consistency:** 3.1 Consistent Visual Organization: Example: Navigation bar and sidebar stay in the same place across pages. Improvement: Predictable layouts help users find what they need quickly. 3.2 Consistent Navigation and Layout: Example: The same sidebar menu is used in booking management as on the main interface.Improvement: Uniform navigation reduces learning time and confusion. 3.3 Consistent Information Presentation: Example: Button labels in room management match styles used elsewhere.Improvement: Uniform presentation improves familiarity and comprehension.

Q5. Please describe how would you test the function in cases that it is working and NOT working (e.g., boundary input). For each of the test cases, please also specify what can be the solution to the problem. (10 marks)

1. Normal Operation Test Cases

1.1 Standard Operation Test
- Input: Check-in date: 2024-12-17, Check-out date: 2024-12-20
- Expected Output: Dates accepted, booking process continues
- Rationale: These are valid dates in the correct order

1.2 Same-Day Check-in and Check-out
- Input: Check-in date: 2024-12-18, Check-out date: 2024-12-18
- Expected Output: Dates accepted, booking process continues
- Rationale: Some hotels allow same-day check-in and check-out

1.3 Year Transition Boundary Case

- Input: Check-in date: 2024-12-31, Check-out date: 2025-01-01
- Expected Output: Dates accepted, booking process continues
- Rationale: Valid input, but a boundary case to ensure correct handling of year transitions

2. Abnormal Operation Test Cases

2.1 Logical Violation Test
- Input: Check-in date: 2024-12-20, Check-out date: 2024-12-18
- Expected Output: System rejects input and displays error message: "Check-in date cannot be later than check-out date"
- Solution: Compare check-in and check-out dates to ensure check-in is not later than check-out. Set the minimum value of the check-out date selector to be the same as or later than the check-in date. Display an error message when the check-out date precedes the check-in date.

2.2 Past Date Test
- Input: Current date is 2024-11-25, Check-in date: 2024-11-01
- Expected Output: System rejects input and displays error message: "Cannot book dates in the past"
- Solution: Implement a restriction in the date picker to allow selection of only current or future dates as check-in dates. Set up a system that updates the date in real-time and set the current date as the lower limit for date selection. Reject input and display an error message when the check-in date is earlier than the current date.

2.3 Upper Limit Date Test
- Input: Check-in date: 10000-01-01, Check-out date: 10000-01-14
- Expected Output: System rejects input and displays error message indicating the date is beyond the acceptable range
- Solution: Set a reasonable upper limit in the date picker to restrict user selection. Reject input and display an error message when the check-out date exceeds the upper limit.

2.4 Null Pointer Test
- Input: Check-in date: [empty], Check-out date: [empty]
- Expected Output: System rejects input and displays error message prompting user to enter valid dates
- Solution: Implement validation to ensure neither check-in nor check-out dates are null. Reject null input and display an error message when either date is null.

2.5 Invalid Date Format
- Input: Check-in date: 01/07/2025, Check-out date: 05/07/2025 (assuming YYYY-MM-DD format is expected)
- Expected Output: Error message indicating incorrect date format
- Solution: Implement strict date format validation. Provide clear instructions on the expected format and offer date picker UI components to minimize input errors.

2.6 Non-existent Dates
- Input: Check-in date: 2025-02-30, Check-out date: 2025-03-01
- Expected Output: Error message indicating invalid date
- Solution: Utilize a robust date parsing library or function capable of validating date existence, accounting for leap years and varying days in months.

2.7 Non-numeric Input
- Input: Check-in date: "abc", Check-out date: "def"
- Expected Output: Error message indicating invalid input
- Solution: Implement input validation to ensure only numeric and allowed special characters (e.g., hyphens) are entered. Provide clear error messages and examples of the correct format.

Q6. Fill in the following code first and provide test cases verifying if the exceptions are thrown successfully. (10 marks)

```
public class Payment {
   private int paymentID;
   private int bookingID;
   private double amount;
   private String paymentMethod; // Example: "Alipay", "Wechat"
   private boolean isProcessed;

   public Payment(int paymentID, int bookingID, double amount, String paymentMethod) {
      this.paymentID = paymentID;
      this.bookingID = bookingID;
```

```
        this.amount = amount;
        this.paymentMethod = paymentMethod;
        this.isProcessed = false;
    }

    public boolean processPayment() throws IllegalArgumentException {
        if (amount < 0) {
            throw new IllegalArgumentException("Payment amount cannot be negative.");
        }

        if(!paymentMethod.equals("Alipay")&& !paymentMethod.equals("Wechat")) {
            throw new IllegalArgumentException("Invalid payment method. Only Alipay and Wechat are
        accepted.");
        }

        this.isProcessed = true;
        return true;
    }

    public boolean isProcessed() {
        return isProcessed;
    }

    public double getAmount() {
        return amount;
    }
}
```

//Q6.2 - Test cases

```
class PaymentTest {

    @Test
    public void testProcessPaymentWithNegativeAmount() {
        Payment payment = new Payment(1, 101, -100.0, "Alipay");
        assertThrows(IllegalArgumentException.class, () -> payment.processPayment());
    }

    @Test
    public void testProcessPaymentWithInvalidPaymentMethod() {
        Payment payment = new Payment(1, 102, 100.0, "MasterCard");
        assertThrows(IllegalArgumentException.class, () -> payment.processPayment());
    }

    @Test
    public void testProcessPaymentWithWechat() {
        Payment payment = new Payment(1, 1, 100.0, "Wechat");
        assertTrue(payment.processPayment());
        assertTrue(payment.isProcessed());
    }
}
```

Q7. Please describe what would you organize this process as a team. (20 marks)

1. Organization Process

1.1 Establish Testing Goals:

Define a clear goal for user testing, and the entire team must commit to that goal.

1.2 Team Division of Labor

Our five-member team has defined clear roles to ensure an efficient testing process. The Test Coordinator oversees the entire testing lifecycle, ensuring smooth execution and timely completion. The Alpha Test Leader manages user participation in early testing phases, gathering initial feedback. The Beta Test Manager focuses on performance issues and detailed user feedback in later stages, addressing system bottlenecks. The Acceptance Test Specialist verifies that the system meets all requirements and communicates with stakeholders. The Team Coordinator ensures overall team coordination, decision-making, and smooth communication. This division of responsibilities enables efficient workflows and helps achieve project goals.

1.3 Prompt communication and feedback

Maintaining steady progress relies on robust reporting and effective communication. We will hold daily stand-up meetings to provide concise updates, promptly address issues, and align team efforts. Additionally, comprehensive post-testing review sessions will enable us to analyze findings, share insights, and plan next steps collaboratively. All communications will be documented and accessible through a centralized repository using project management tools, ensuring transparency for the entire team. Regular updates will also be shared via newsletters and interactive dashboards, offering stakeholders real-time visibility into project metrics and key performance indicators.

1.4 Consider team and selecting test participants

When considering internal team and selecting test participants, our approach is multifaceted: technically proficient members tackle complex coding challenges and system architecture, while those with excellent communication skills coordinate idea exchange, mediate discussions, and maintain team cohesion. For user testing, we have curated a diverse pool of participants that reflects our target demographics, including different ages, levels of technical proficiency, and travel frequencies. By pairing tech-savvy testers with less experienced users, we can gain deep insights into advanced feature requirements and basic usability issues. Additionally, we have convened staff from various hotel departments to evaluate the system's efficiency from an operational perspective. This comprehensive selection process not only enhances the quality and breadth of our feedback but also fosters a culture of continuous learning and improvement within our team.

1.5 Alpha Testing

During Alpha Testing, the internal development team thoroughly evaluates the Simple Hotel Booking System to identify and fix defects or inconsistencies. This involves coordinating roles, setting up a controlled environment, and creating comprehensive test cases covering all functionalities such as room search, booking, payment processing, and management operations. Systematic execution of these tests uncovers bugs and performance issues, which are tracked and resolved through iterative cycles to ensure the system's stability before moving to external testing.

1.6 Beta Testing

In the Beta Testing phase, the system is released to a broader group of external users, including actual hotel customers and management staff, to gather real-world feedback and identify issues not found during Alpha Testing. Beta testers are encouraged to use all features extensively, and their feedback is collected through surveys, forms, and direct communication. The team analyzes this feedback to prioritize enhancements and fixes, refining the system’s performance and user experience based on genuine user interactions and needs.

1.7 Acceptance Testing

Acceptance Testing is the final validation stage where the system is assessed against predefined requirements to ensure readiness for deployment. Stakeholders and key users verify that all functionalities, such as room management, booking processes, and payment handling, meet their expectations and business needs. The system’s performance is evaluated under expected load conditions to confirm reliability. Final adjustments are made based on stakeholder feedback, and formal approval is obtained to ensure the system aligns with business objectives and is ready for production deployment.

2. Potential problems and solutions

2.1 Lack of Mobile Optimization

Users may face difficulties navigating the booking system on mobile devices, leading to lower conversion rates. The mobile interface may be poorly optimized or non-responsive; Buttons and text may be too small, leading to navigation errors. Solution: Mobile-first design: Continuously test and refine the mobile user experience. Ensure that the system is fully responsive and optimized for mobile devices, with easy-to-click buttons and clear text; Focus on performance: Optimize the loading speed and smoothness of mobile interactions to improve user retention and conversion rates; User testing on mobile: Conduct ongoing mobile usability tests to uncover issues and gather feedback directly from users to inform improvements.

2.2 Unclear Task Descriptions and Ambiguous Requirements Causing Delays During process, the team faced delays in completing testing activities due to insufficiently detailed task descriptions and unclear requirements. Ambiguous task breakdowns led to communication issues and a lack of consistency in expectations for each task, resulting in untimely progress and overall inefficiency in task execution. Moreover, differing understandings of priorities among team members exacerbated this issue. Solution: Implementing Agile Methodology and Using JIRA for Task Management To address this problem, we adopted Agile methodology, particularly Scrum, to enhance our task management and team collaboration. Through sprint retrospective meetings, we regularly evaluated our performance and identified areas for improvement. Utilizing JIRA, we ensured clear task descriptions and well-defined requirements for each sprint. We introduced the concepts of User Stories and Acceptance Criteria to better capture and communicate requirements. Additionally, we implemented Daily Stand-ups to facilitate communication between team members and quick problem-solving. This approach fostered a better understanding and execution of tasks, reduced delays, improved overall productivity, while also enhancing team cohesion and collaboration efficiency.

2.3 The integration and deployment process is flawed

During process, manual code integration and deployment created workflow bottlenecks, delaying update releases and slowing down user feedback collection and issue resolution. Additionally, manual processes introduced human errors, further hindering the development cycle. Solution: Adopting Continuous Integration and Continuous Delivery (CI/CD) We implemented a CI/CD pipeline to automate the integration and deployment processes. By frequently integrating code changes into a shared repository and automating builds and tests, we ensured that new changes wouldn't break the existing codebase. Moreover, automated deployment to staging environments allowed for faster testing and feedback loops. We used Jenkins as our CI/CD tool, coupled with Docker containerization technology, to achieve consistency in environments and repeatability in deployments. We also introduced a blue-green deployment strategy to minimize deployment risks and enable quick rollbacks. These measures not only accelerated deployment speed but also significantly improved deployment reliability and overall system stability, enabling us to respond more quickly to user feedback and address performance issues.

Section 4. Peer review form template

CPT203 Coursework
Peer review
Individual Contribution for Group Report

Group Number: <4>

Name	ID Number	**Contribution (%)** Please enter an integer, for example 15% contribution, please enter 15. The sum of this column should be 100	Signature
1. Yize liu	2254472	20	Yize Liu
2. Shengtian Huang	2254461	20	Shengtian Huang
3. Qing qin	2254084	20	Qing qin
4. Xu Chen	2257453	20	Xu Chen
5. Zichen Qiu	2252705	20	Zichen Qiu

END