CPT203
Coursework 2

2024/2025 Semester 1
2024.12.12

Group number: 54
Student 1 Name: Pengkai Chen Student 1 ID: 2251486
Student 2 Name: Jinhong Jiang Student 2 ID: 2251601
Student 3 Name: Yilin Li Student 3 ID: 2255705
Student 4 Name: Rui Sang Student 4 ID: 2251576
Student 5 Name: Peiling Tu Student 5 ID: 2251487

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

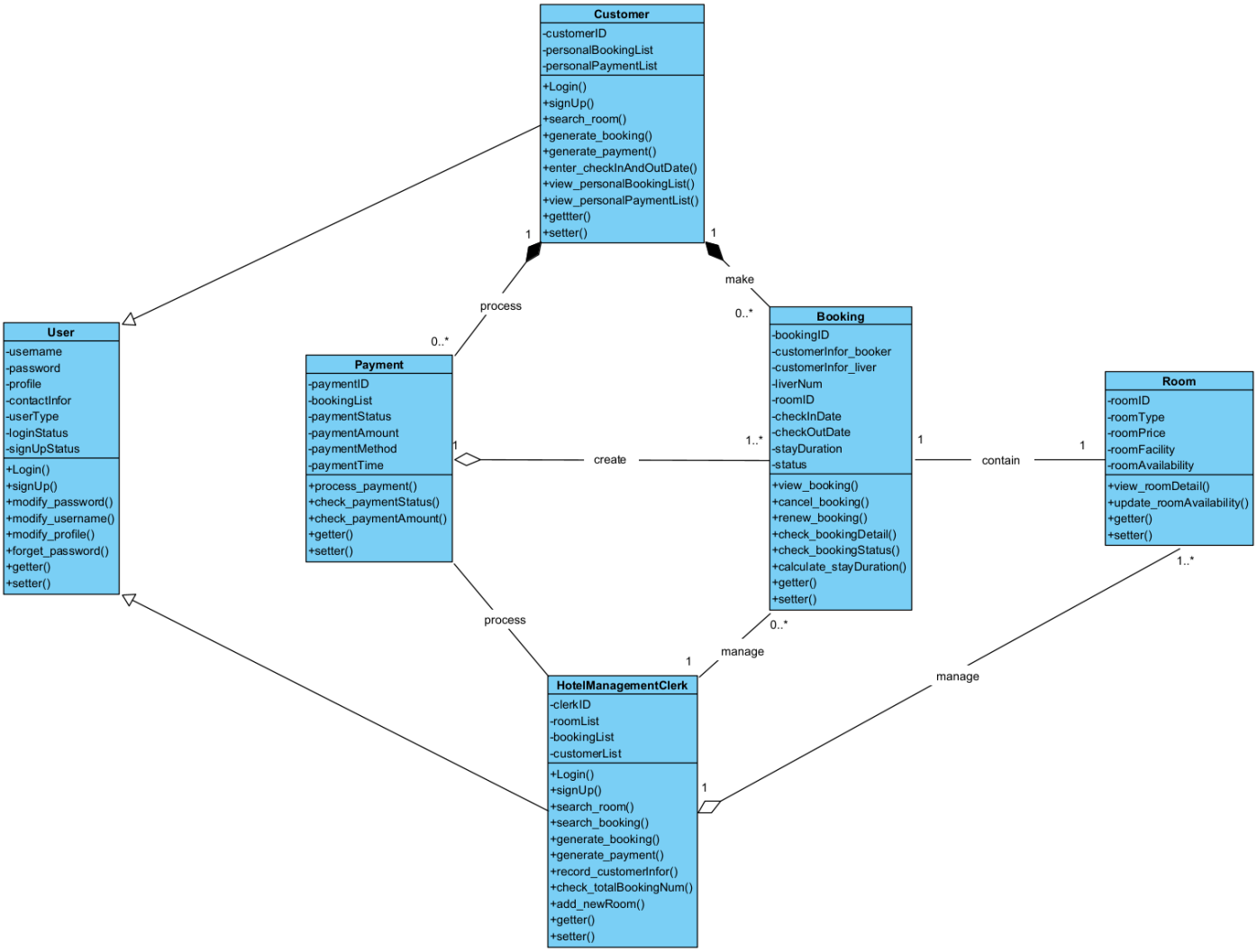


Figure 1: Class Diagram of the Hotel Booking System

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

1. **Abstraction**
   Data abstraction and procedural abstraction design principles are utilized to optimize the system design. Through data abstraction, classes such as 'Booking' and 'Payment' encapsulate specific data attributes, which allow users to interact only with the essential characteristics of these objects without being burdened by their underlying complexities. Similarly, procedural abstraction is implemented by concealing the internal logic of methods, such as 'Login()' and 'signUp()' within the User class, and exposing only the method signatures. This enables users to utilize these functions directly without delving into their implementation details. Collectively, these abstraction techniques contribute to reducing code complexity, improving system maintainability, enhancing readability and usability, and fostering code reusability.

2. **Modularity**
   Using the modularity design principle, the hotel booking system is systematically decomposed into distinct classes, each addressing specific functionalities, and further segmented into dedicated class functions. For instance, the 'Customer' class is modularized as an independent module responsible exclusively for customer-related operations. Within this module, essential functions such as 'search_room()', 'book_room()', and 'process_payment()' are meticulously defined and modularized to streamline these operations. This structured methodology significantly enhances system flexibility, reduces overall complexity, optimizes code organization, improves fault tolerance and system stability, and facilitates subsequent optimization efforts, ensuring a robust and maintainable architecture.

3. **Function independence**
   The system design adheres to the functional independence principle, ensuring low coupling between different classes. This is achieved by incorporating 'getter()' and 'setter()' methods within each class to facilitate safe data modification operations between them, thereby minimizing interdependencies, which could effectively reduce system complexity and optimize its testability and maintainability. Additionally, the high cohesion principle is emphasized, with each class and its associated methods designed to fulfill a specific, well-defined purpose. For instance, the functionalities related to booking and payment are handled by distinct classes, 'Booking' and 'Payment', respectively. Within these classes, the responsibilities are further subdivided into specialized methods such as 'update_bookingDetail()', 'check_bookingDetail()', and 'process_payment()', each addressing a particular aspect of their respective operations.

4. **Object-Orient Design**
   The encapsulation and inheritance design principles are utilized to optimize the system design. Initially, based on the class diagram, data attributes of the 'Customer' class will be set as 'private' in the system to avoid direct modification of the object state by the external code. Meanwhile, the 'getter()' and 'setter()' methods are included in every system class to realize safe data access operations. In addition, The 'Customer' class and 'HotelManagementClerk' class inherit from the 'User' class. The inheritance relationship allows child classes to directly access the data attributes and class methods of the parent class, thereby improving overall system logic and enhancing code reusability.

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

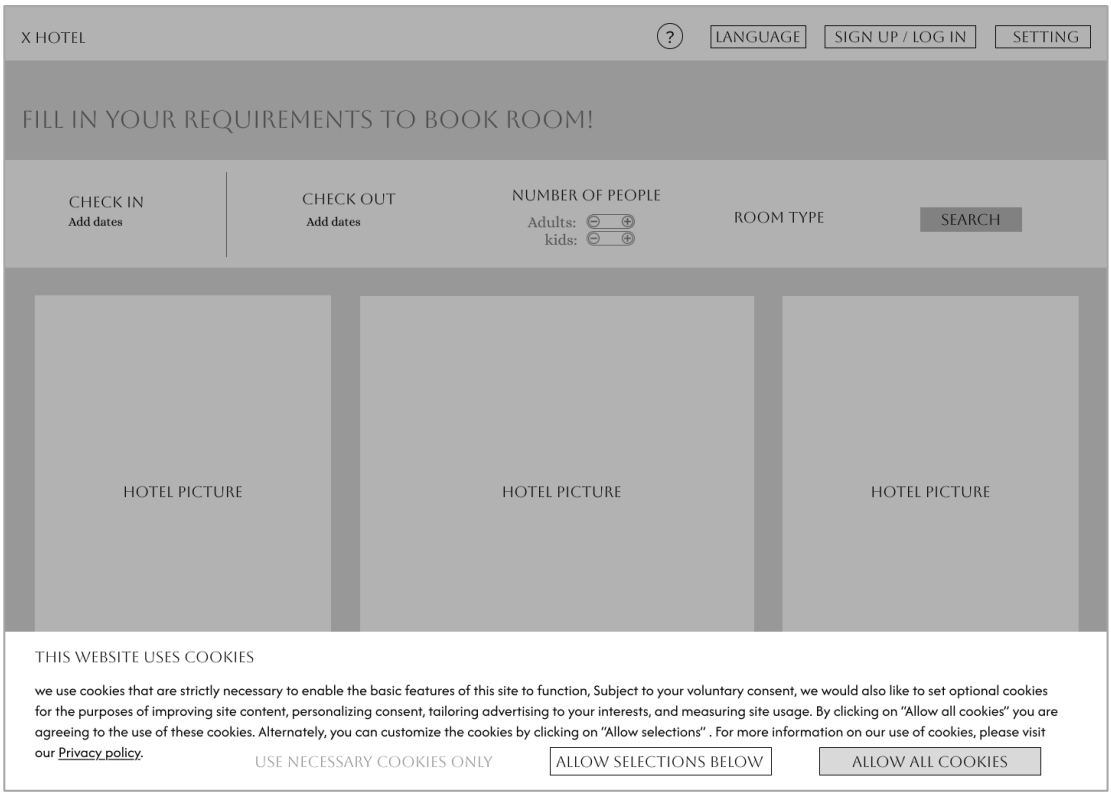


Figure 2: Cookies page

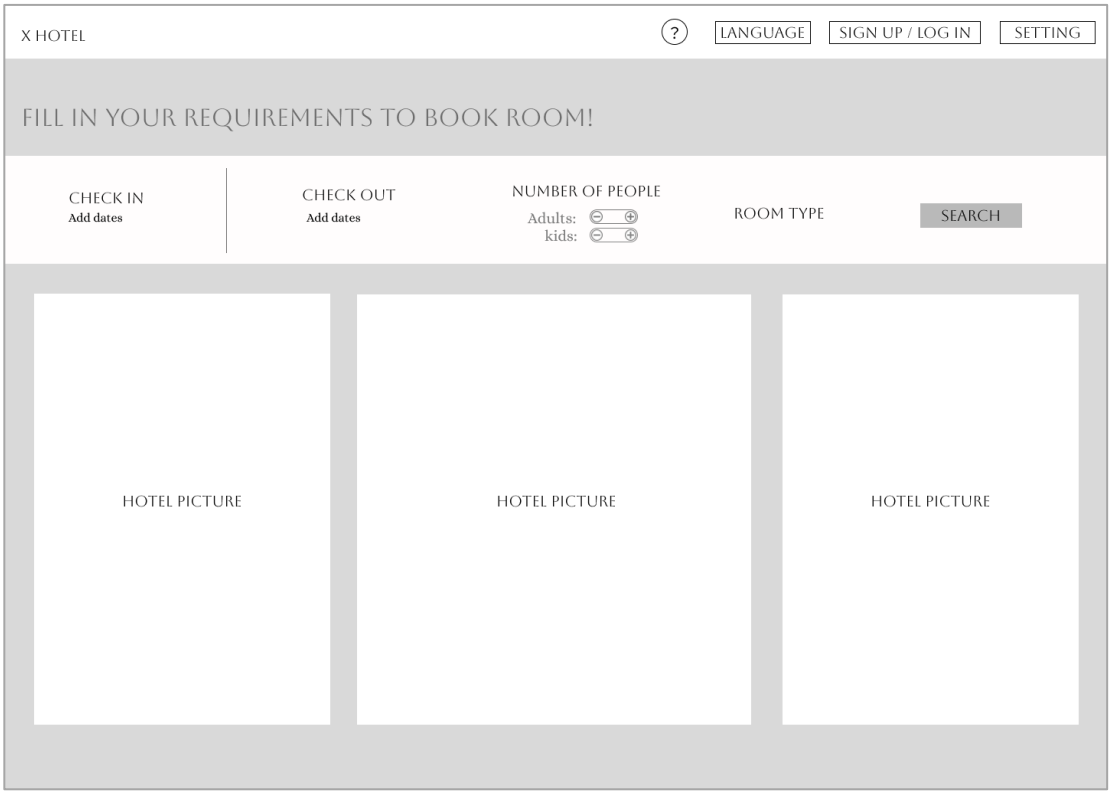


Figure 3: Main home page

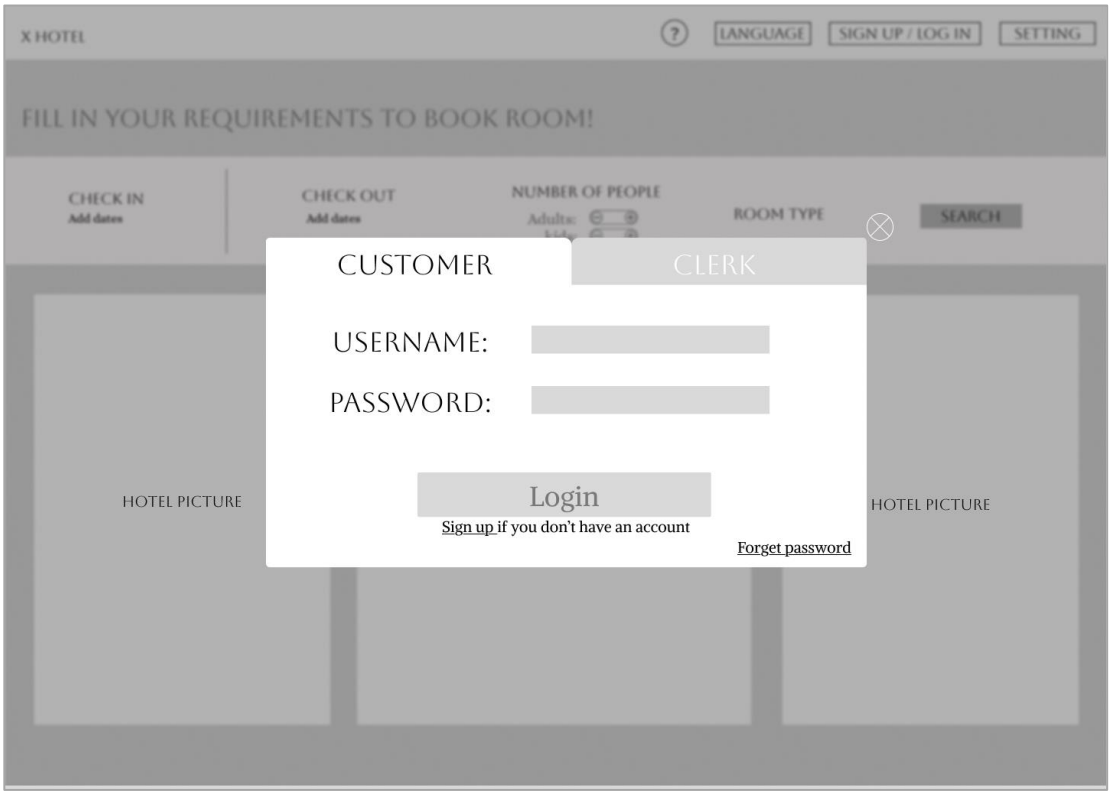


Figure 4: Login page (customer)

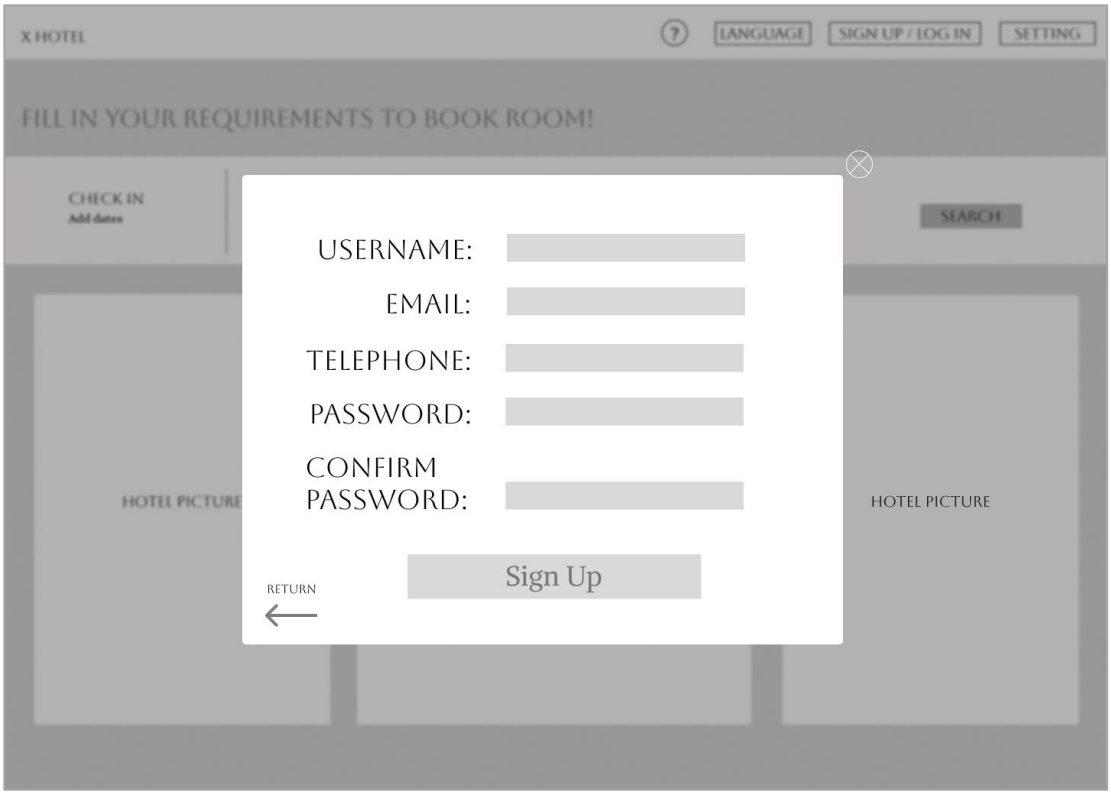


Figure 5: Signup page (customer)

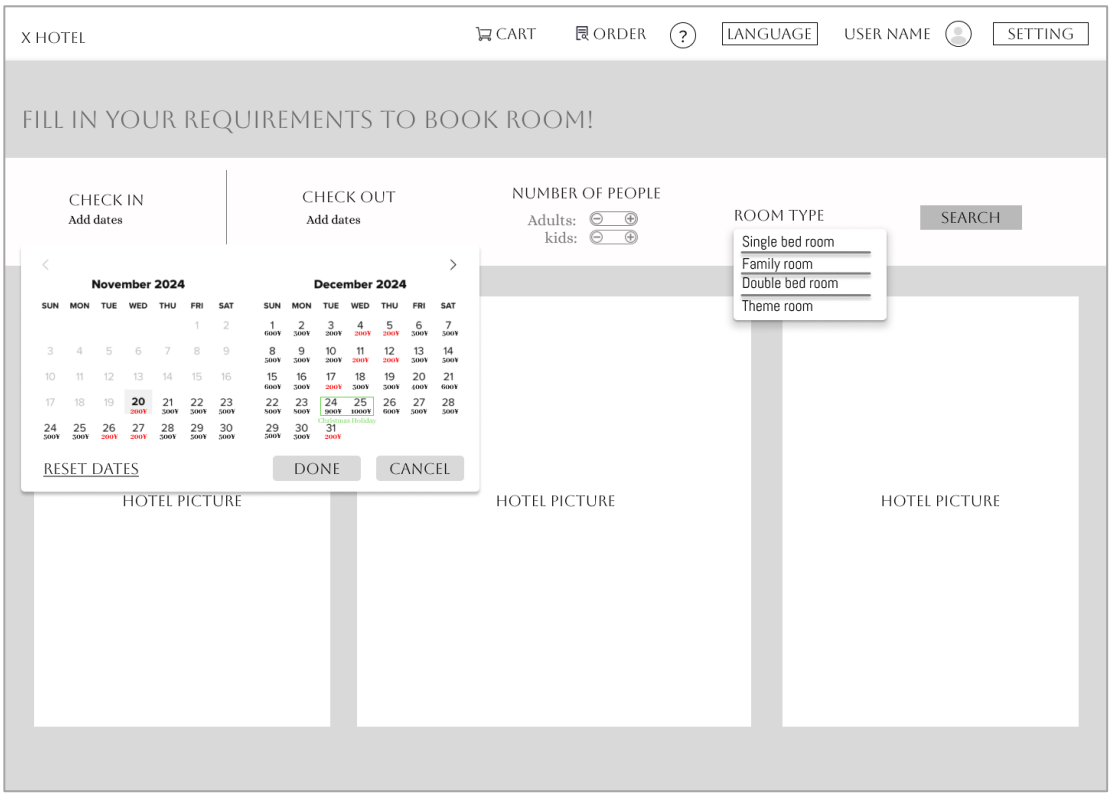


Figure 6: Search instruction page

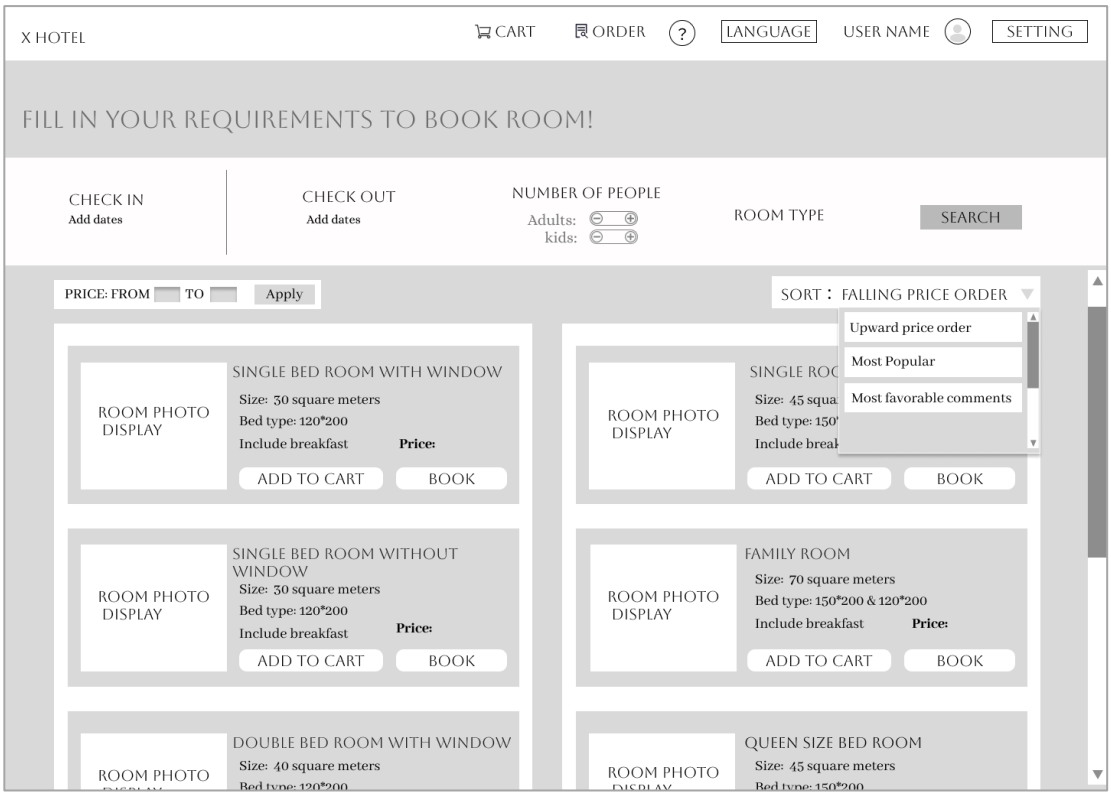


Figure 7: Available room information page

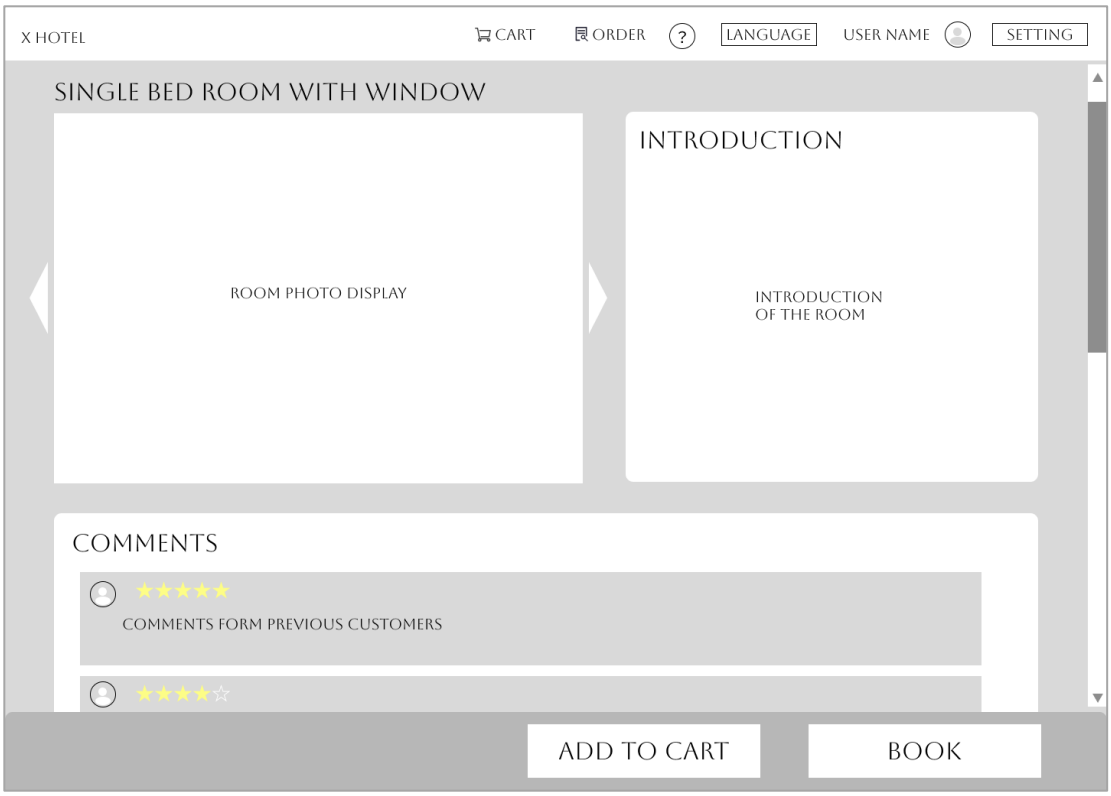


Figure 8: Room detail information page

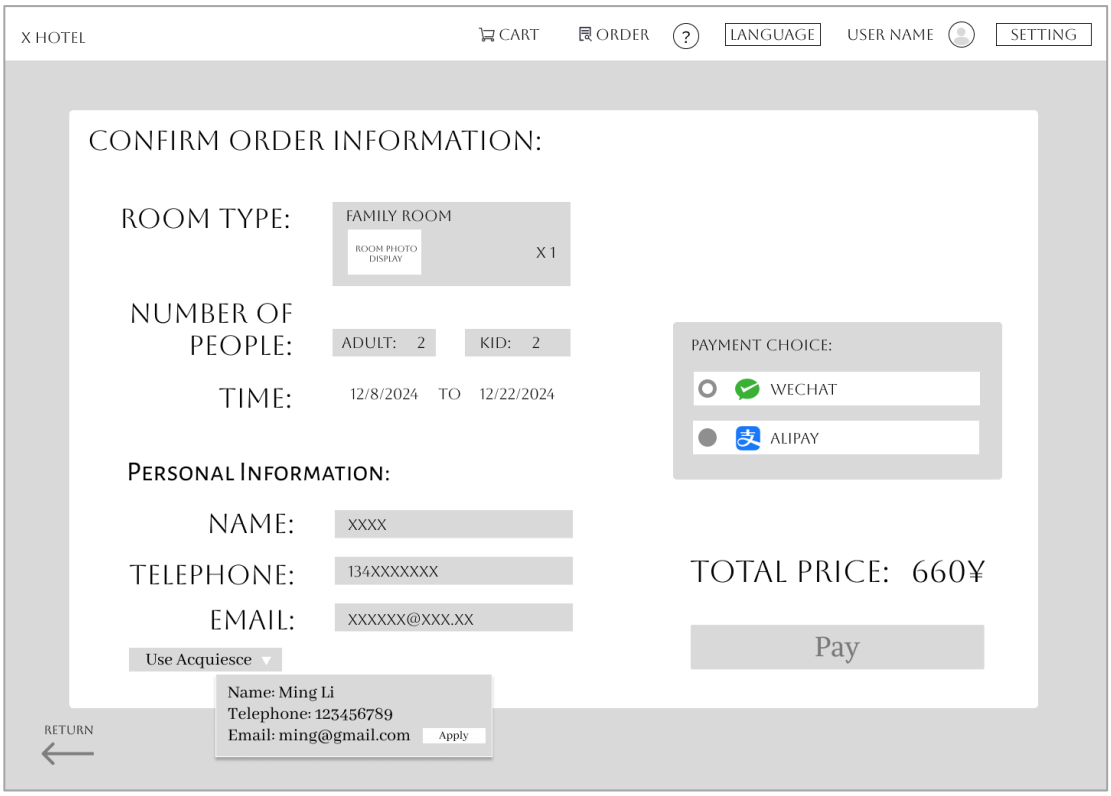


Figure 9: Payment process page (customer)

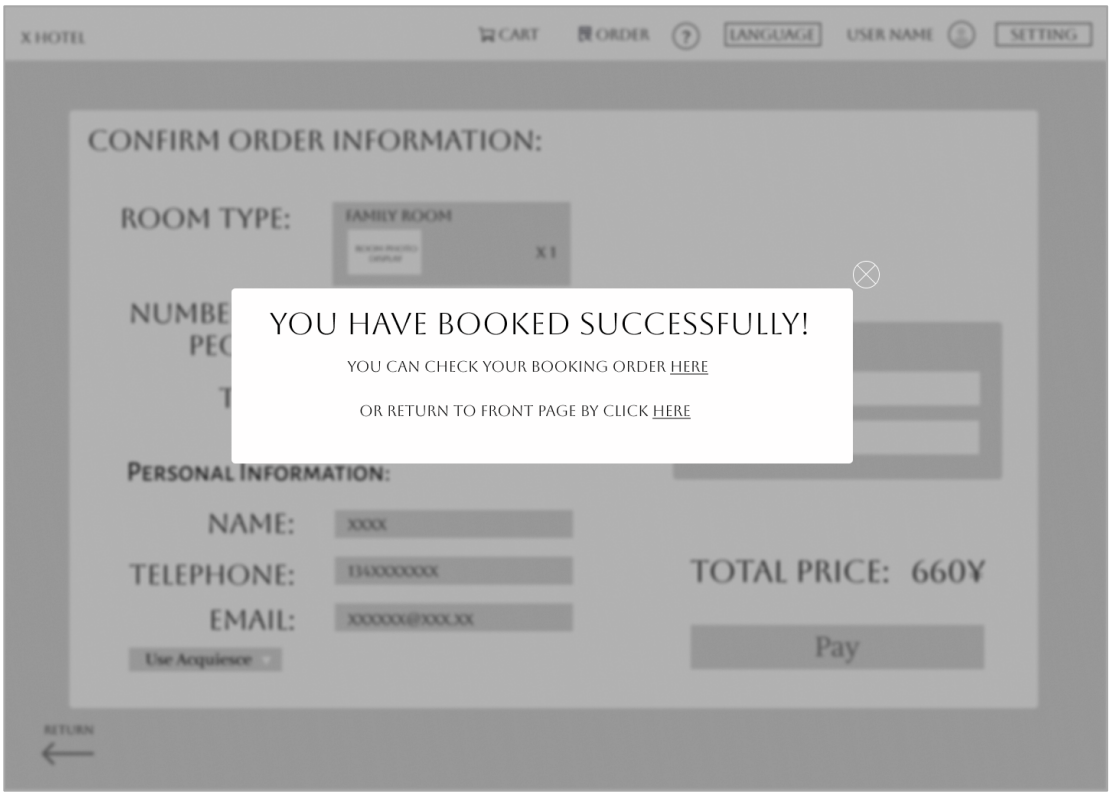


Figure 10: Message of book/payment results

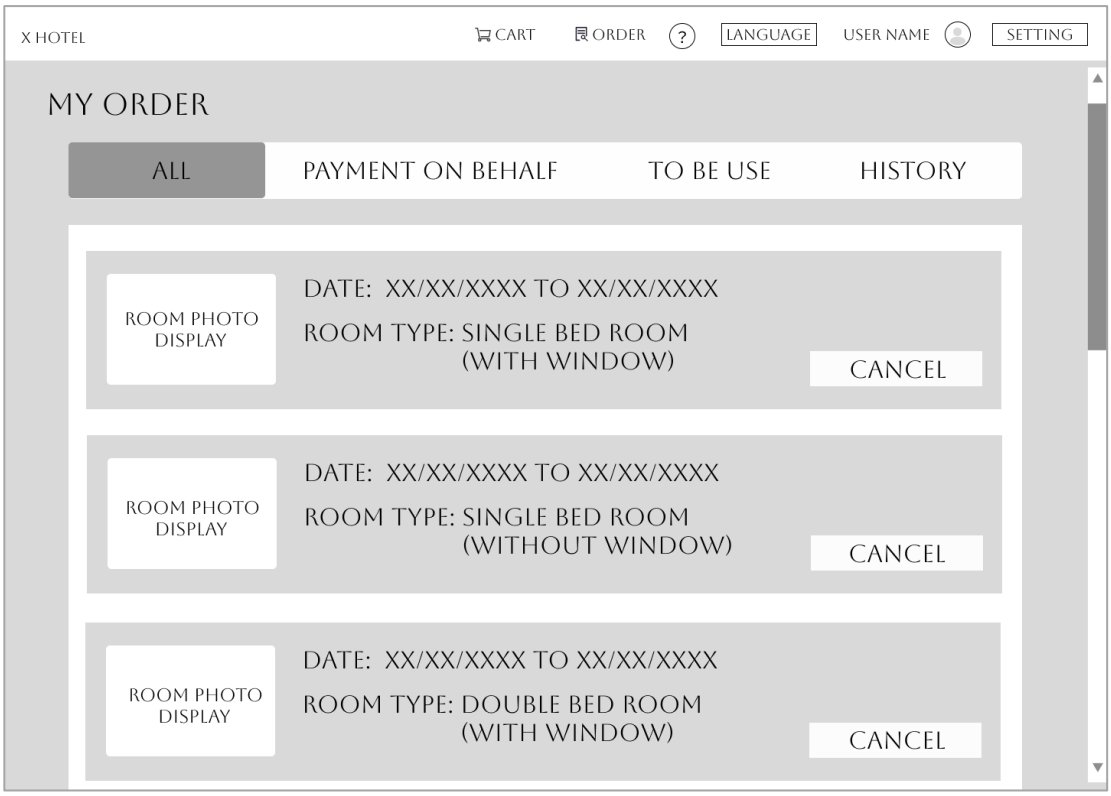


Figure 11: Order details page

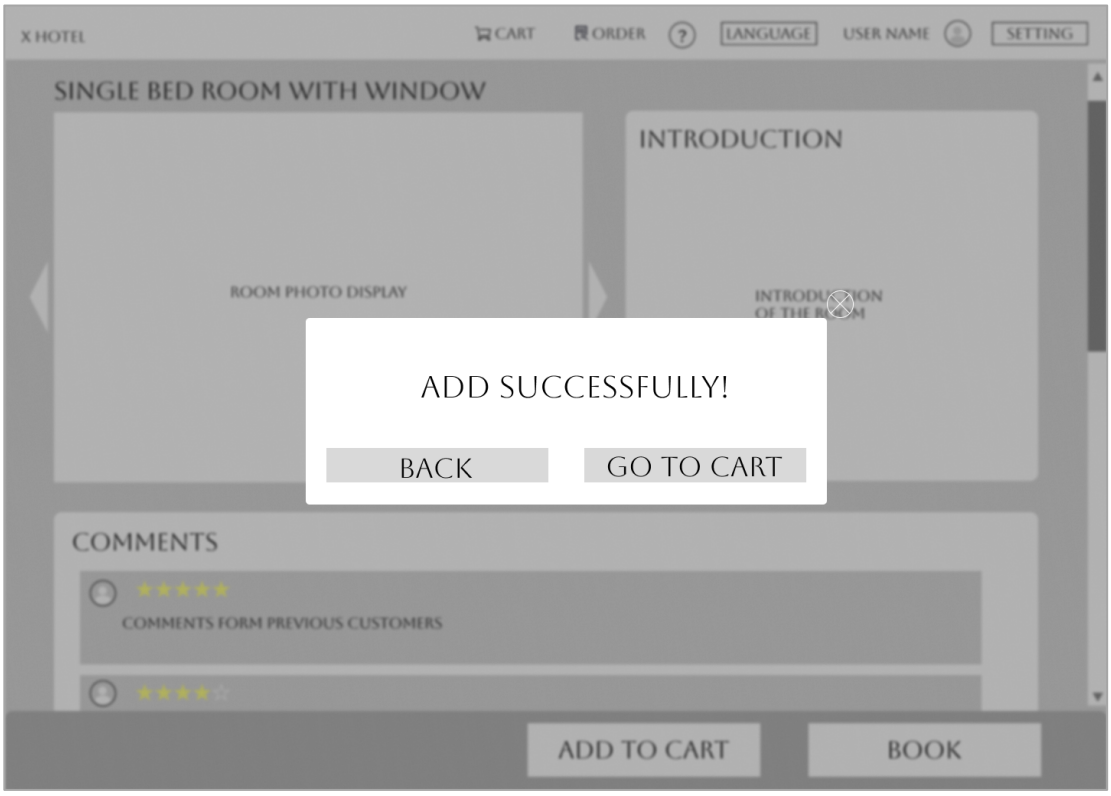


Figure 12: Adding result message

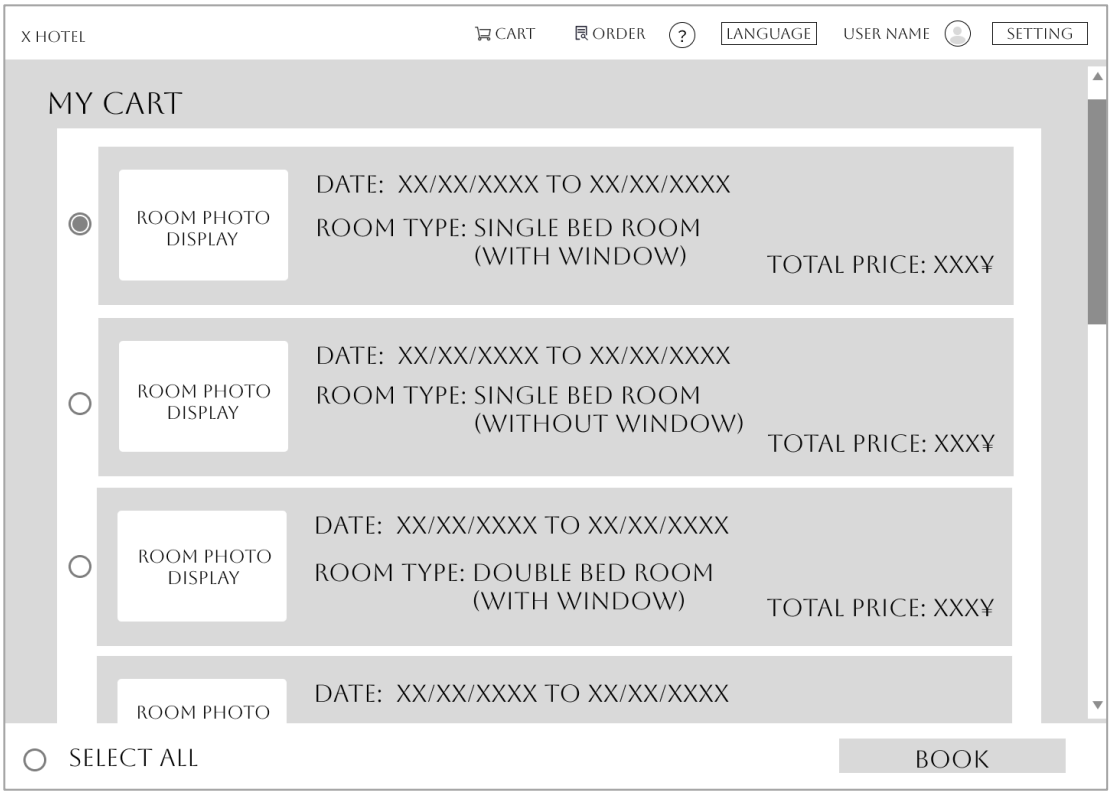


Figure 13: Cart details page

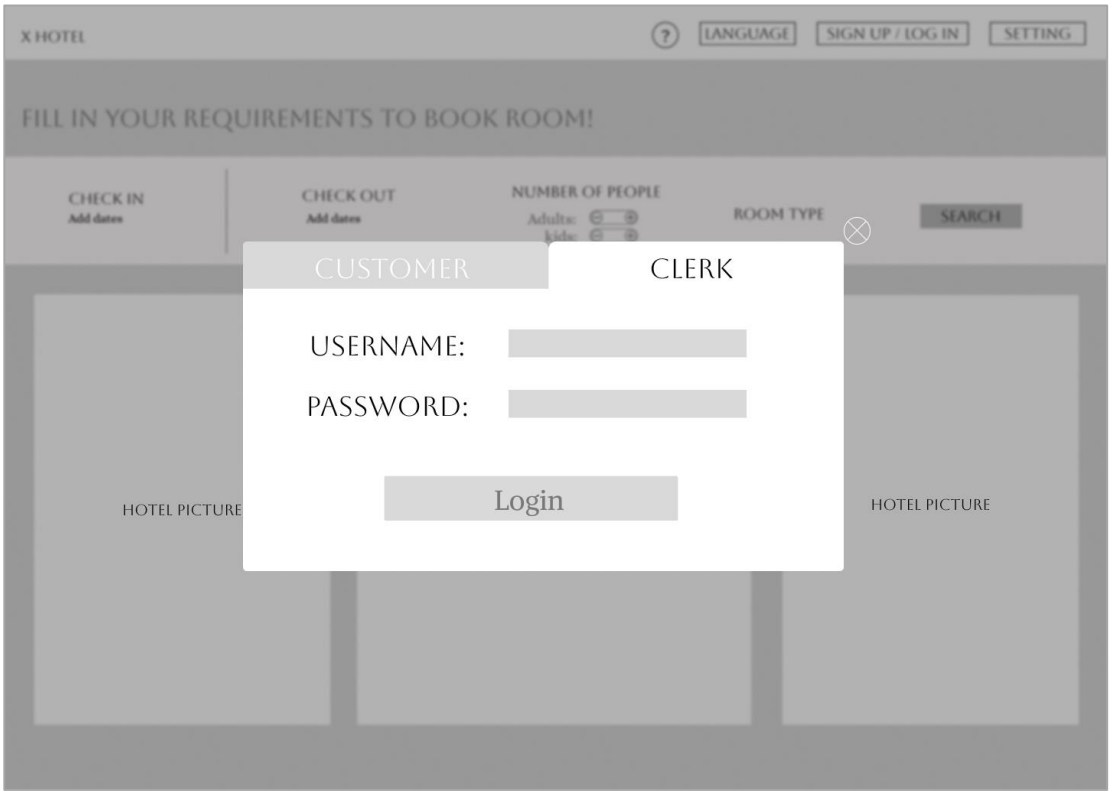


Figure 14: Login page (clerk)

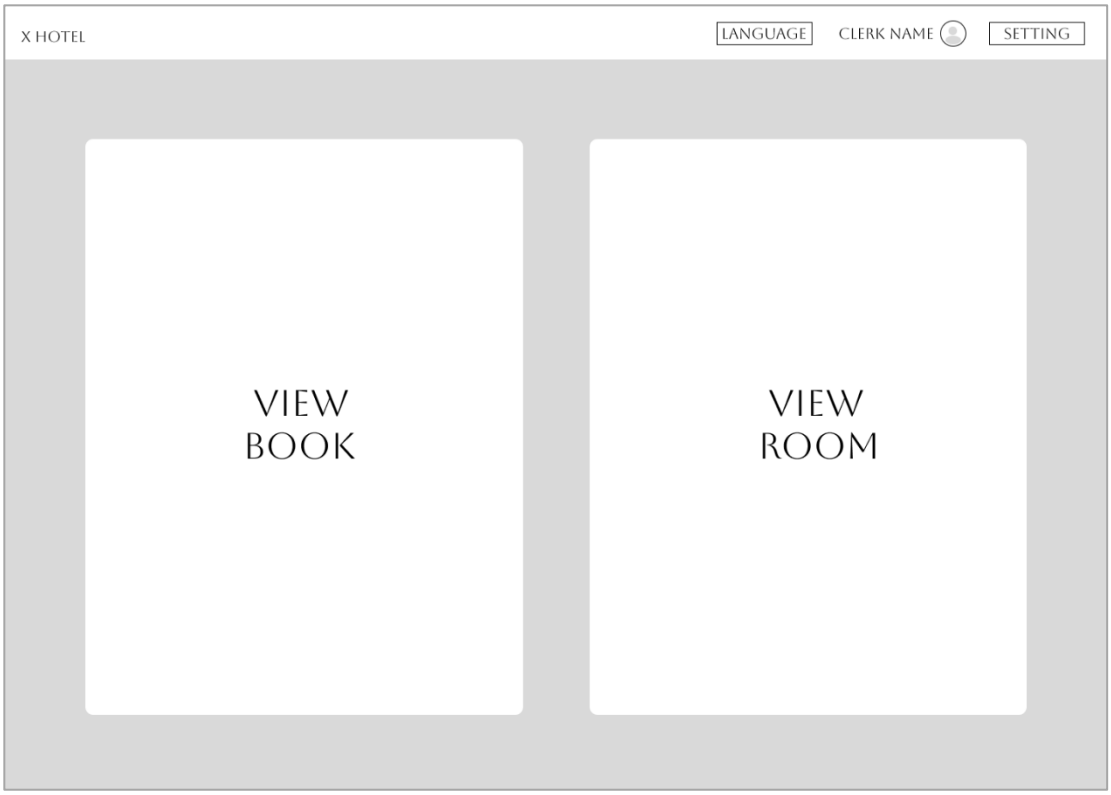


Figure 15: Functional partition page

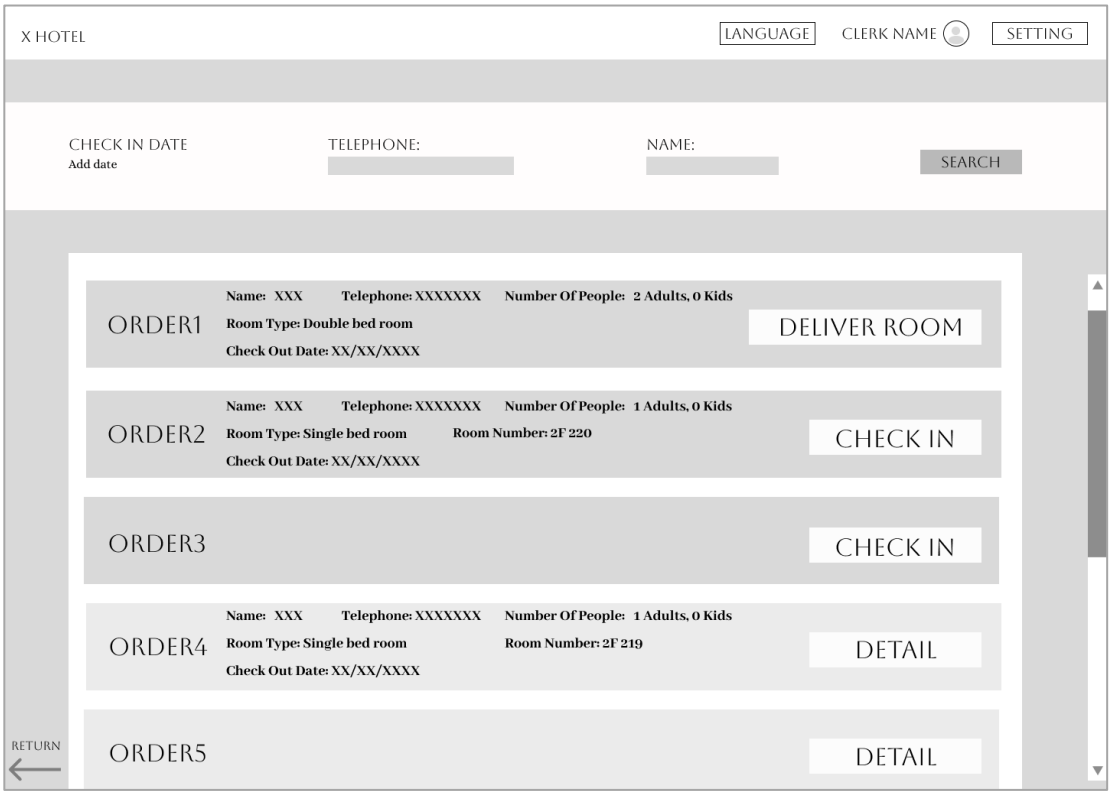


Figure 16: Order list page

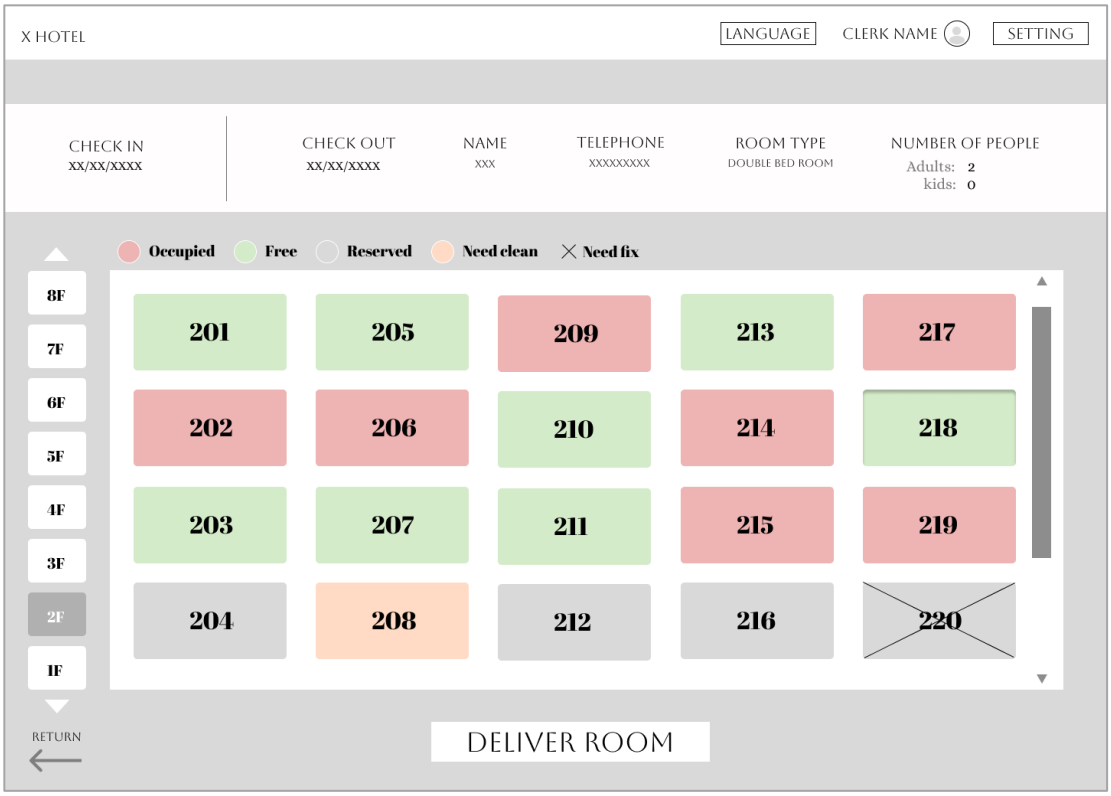


Figure 17: Room viewing page1

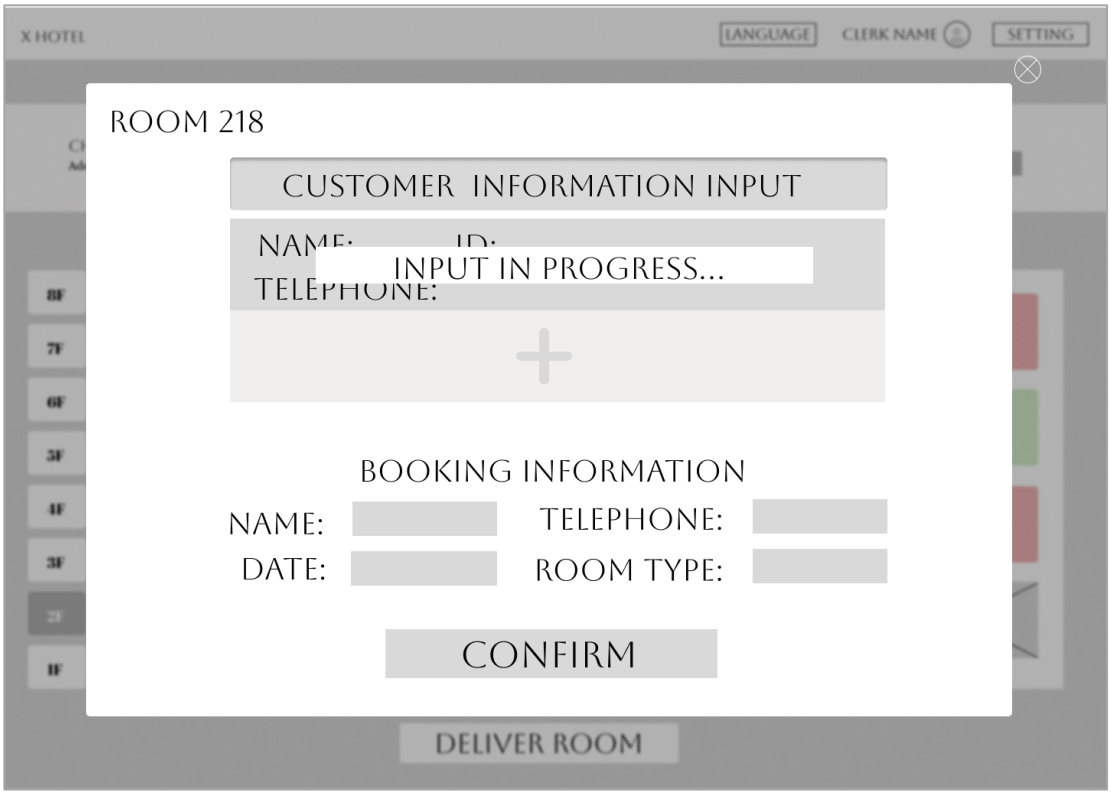


Figure 18: Check-in page

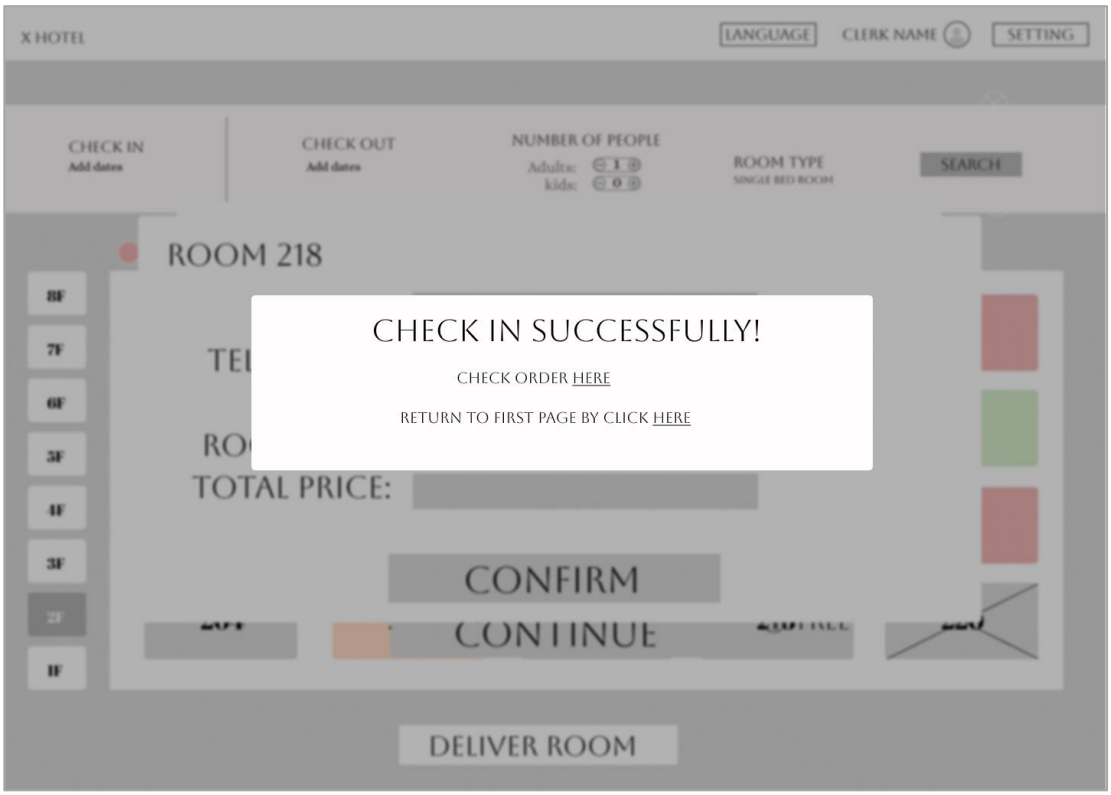


Figure 19: Check-in successfully message

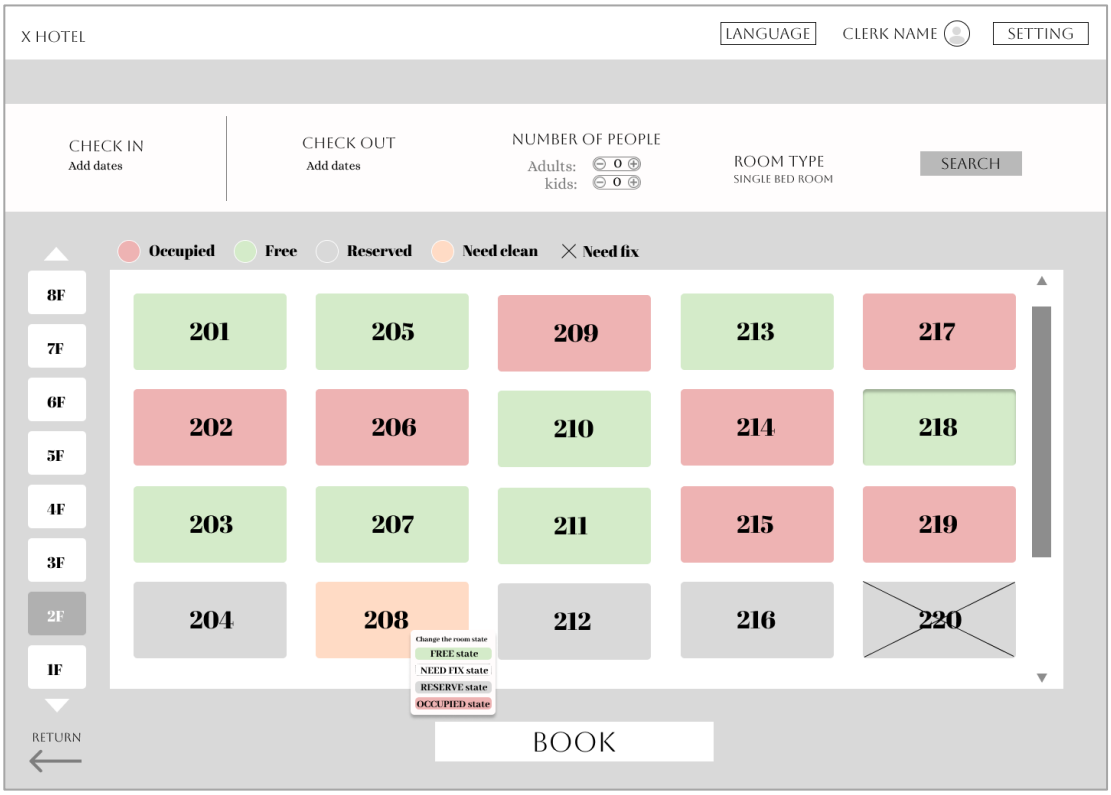


Figure 20: Room viewing page2

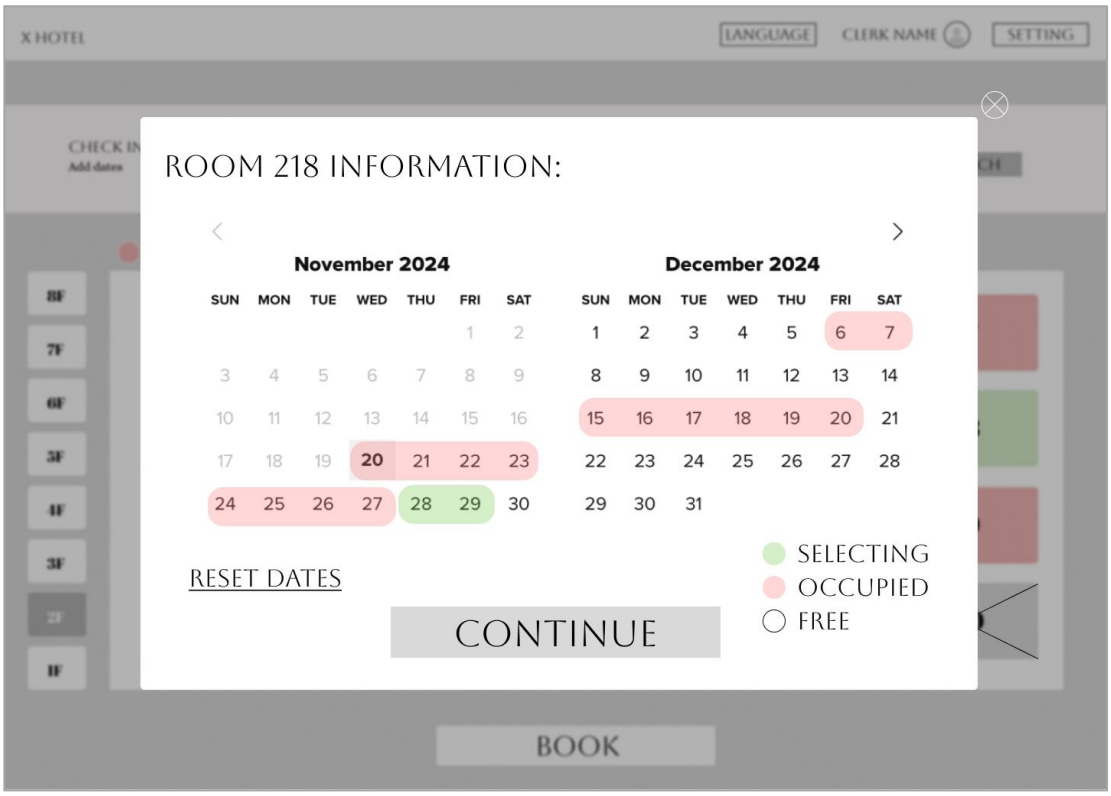


Figure 21: Room status detail page (last 60 days)

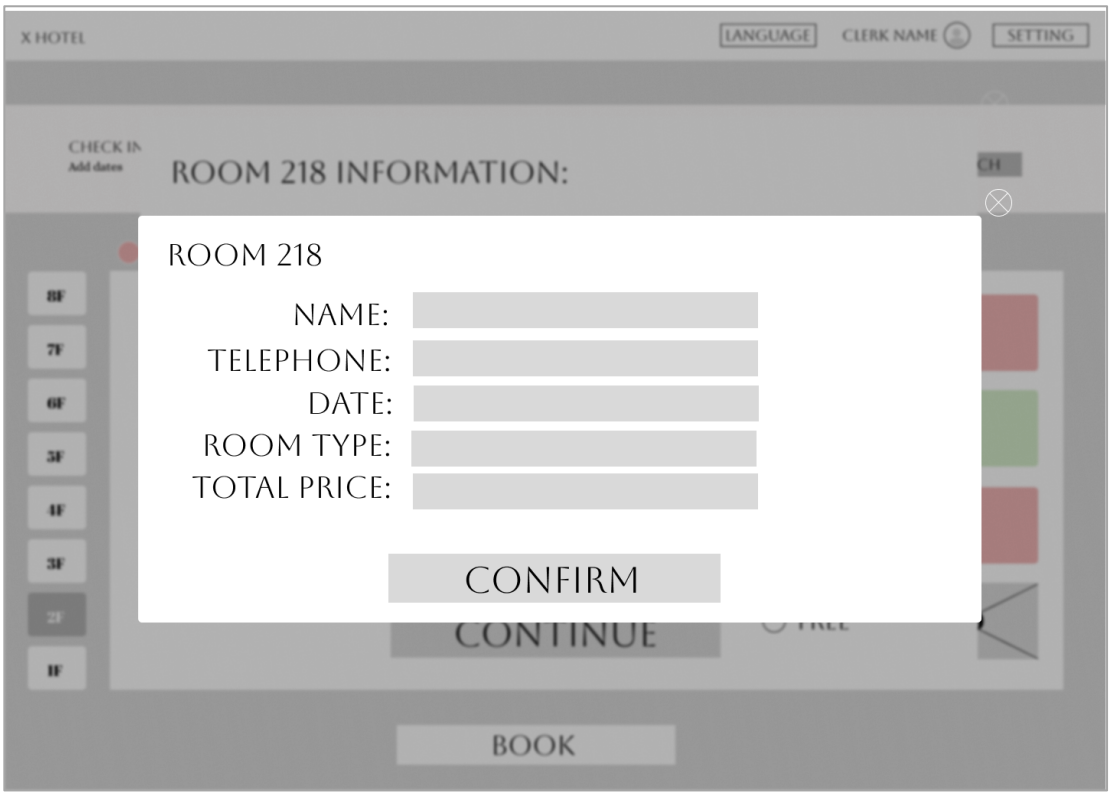


Figure 22: Reservation information page

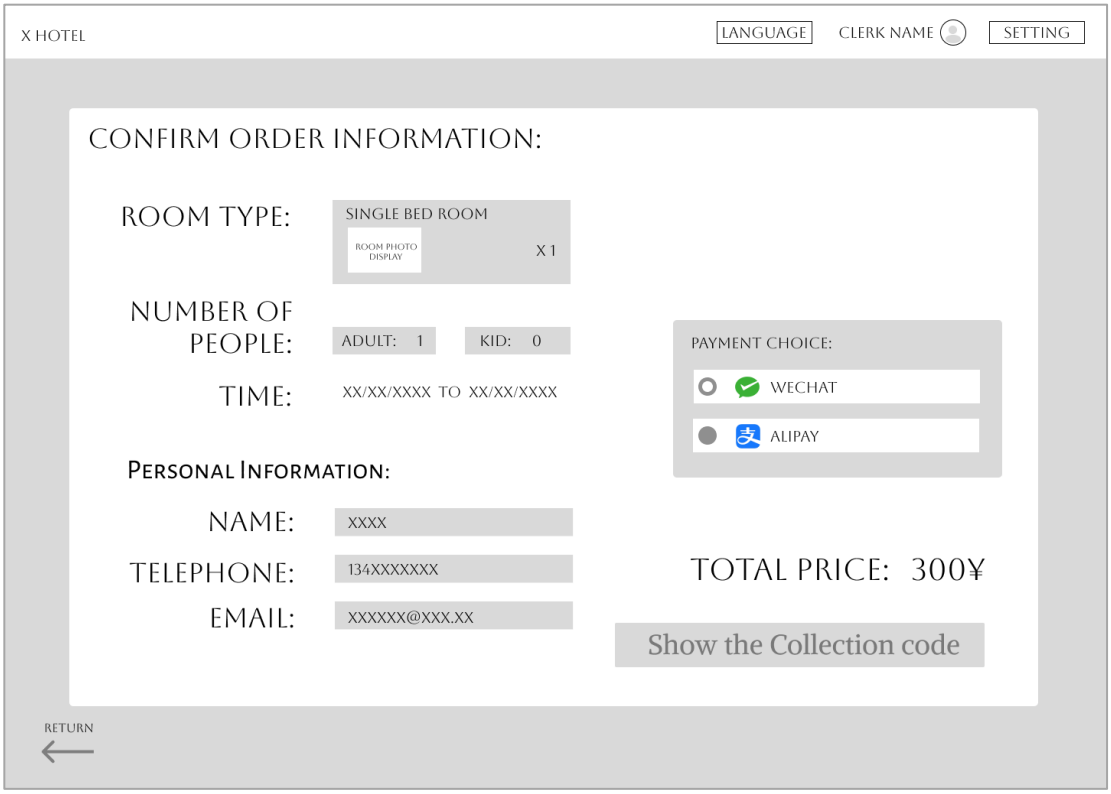


Figure 23: Payment process page (clerk)

Figure 24: Payment QR code

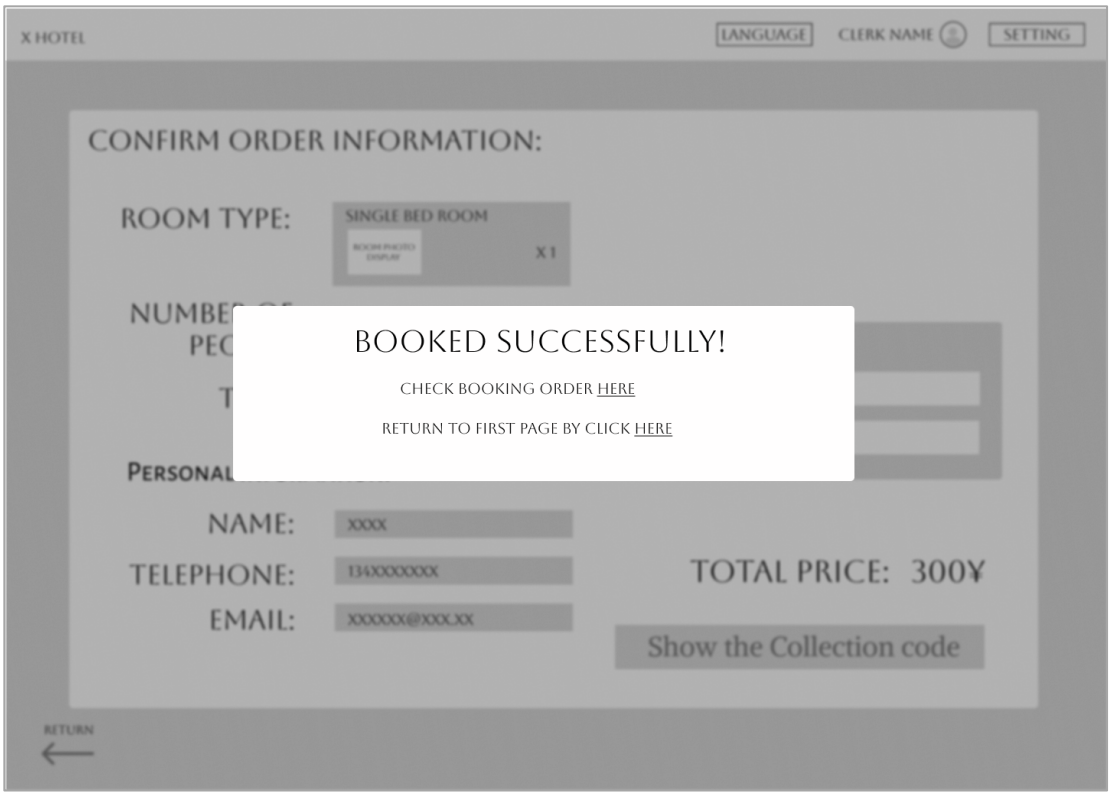


Figure 25: Book successfully message

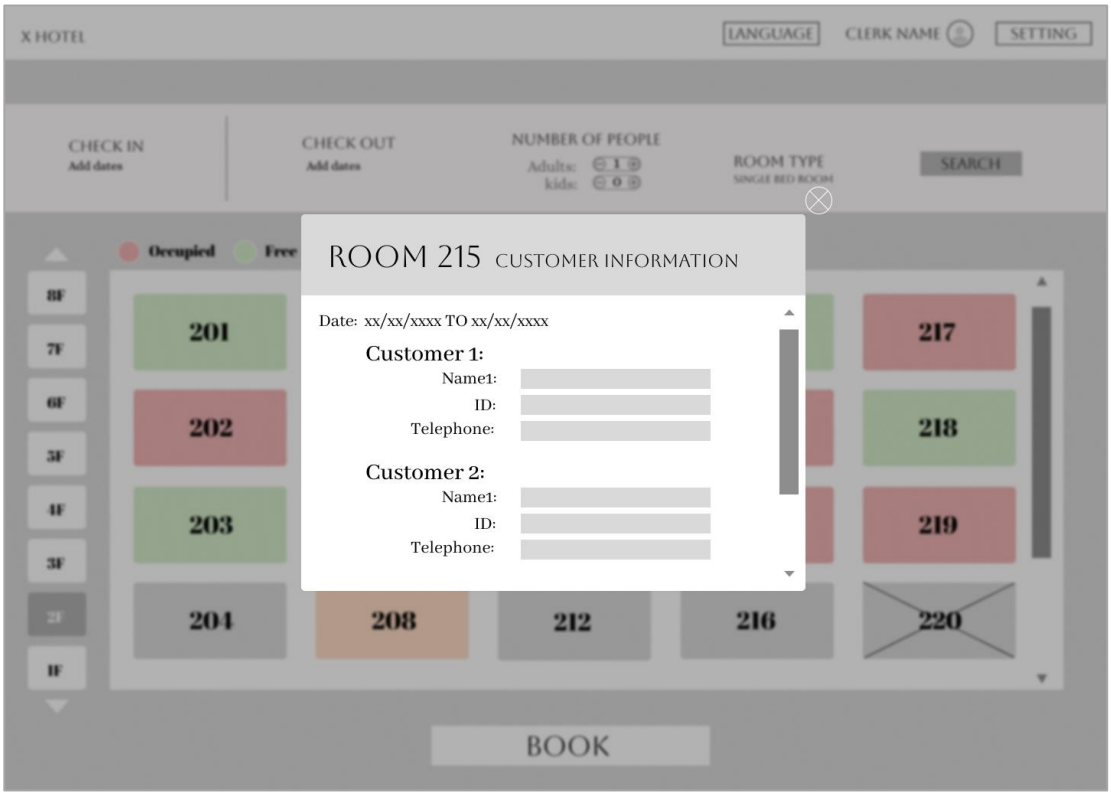


Figure 26: Occupied Room Detail Page

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)

A. Overall process framework

Due to the complexity of the UI design logic, an overall processing **framework** is provided to demonstrate the workflow of the UI in specific.

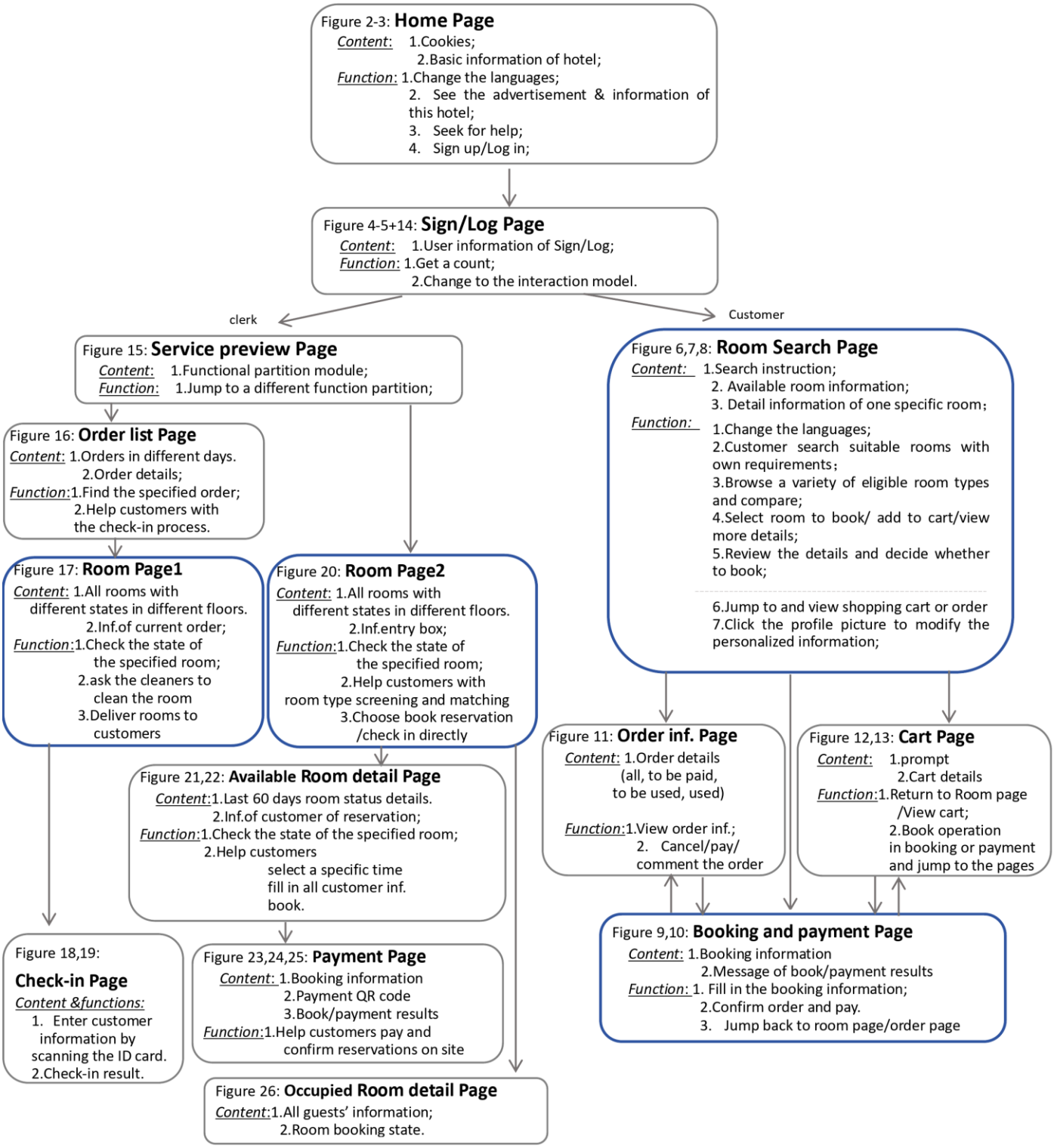


Figure 27: Workflow of the UI design

B. How we design our UI with "The Golden Rules"

1. Place the User in Control

1.1. Customer Room Search and Booking:

- Implement search and filter functions, such as filtering by room type, price (with a range slider), date, and discounts, with daily prices displayed for each date to help users analyze and identify off-peak and peak seasons.
- Let user choose the sort (a. Price (low to high), b. Number of bookings, c. Positive review ratio.).
- Users can promptly modify, return to, and reset filter conditions.
- Provide various prompts (e.g., 'Add successfully,' 'Loading,' 'Book successfully') to help users understand the system's status.
- Allow users to view images directly by clicking on them and access detailed information by clicking on the room name, enhancing efficiency and interactivity.

1.2. Room Booking Management:

- Display prompt windows during customer and room information inquiries, allowing interruption, cancellation, and modification.
- Visualize room status: Administrators can view the current status of all rooms in real-time and manually update them as needed.

1.3. Payment Processing:

- Add batch operations to the payment to enhance interaction efficiency.
- Enable users to return and modify their order during payment confirmation. Display an error message such as 'Payment failed, please check your network connection,' and allow users to go back to the previous page or retry.

2. Reduce the User's Memory Load

2.1. Customer Room Search and Booking:

- The page automatically saves the user's filter criteria, allowing for quick adjustments and repeated searches, while ensuring that other conditions remain unaffected when changes are made.
- Provide customers with their own booking history for easy reference.
- Display search results intuitively by including features such as price and square footage directly on the search interface, helping users avoid repetitive memory and comparison efforts.

2.2. Room Booking Management:

- Display a visual room status panel with distinct colors clearly marking different room statuses, reducing the memory burden on administrators

2.3. Payment Processing

- Set default options to retain the customer's preferred payment method for direct selection.

3. Make the Interface Consistent:

3.1. Customer Room Search and Booking:

- Ensure consistent layouts across multiple room search interfaces, making minimal changes to retain visual uniformity.

3.2. Room Booking Management:

- Display booking information for all rooms in a tabular format with consistent field order.

3.3. Payment Processing:

- Direct all payment processes from different paths to the same interface, avoiding overly complex designs.

C. Out Extension and Principles

1. Containment and Diversity:

- Provide multi-language support to cater to users from diverse linguistic backgrounds.
- Offer font size adjustment options to meet the needs of users across different age groups.
- Allow users to customize default preferences, priorities, and page themes in the personal settings interface according to their own preferences.

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)

To test whether the function is working correctly, it is essential to verify its performance within the valid range by using normal inputs and boundary values. Moreover, testing should include invalid data, such as illegal characters and out-of-range values. Also, the system could handle errors effectively and display error prompts accurately.

Focus Area: check-in and check-out dates

Test Case:

1. **Working cases (Valid Inputs):**

   Testing Case 1: The check-in date and check-out date are valid. The check-in date is before the check-out date.

   Expected Result: Dates are accepted, and the system allows proceeding with booking.

   Testing Case 2: The check-in and check-out dates are within a valid range and the room is available at that period of time.

   Expected Result: A message is displayed: "Booking has been done successfully."

2. **Non-working Cases (Invalid Input):**

   Testing Case 3: The check-out date is earlier than the check-in date.

   Expected Result: A message is displayed: "Check-out date cannot be earlier than check-in date."

   Solution: Adding a calendar component to prevent users from selecting invalid dates. The users could click the exact date on the calendar.

   Testing Case 4: The check-in date is earlier than today (in case the users will check in in advance).

   Expected Result: A message is displayed: "Check-in date cannot be earlier than today."

   Solution: Implement a validation check in the system to compare the selected check-in date with the current date.

   Testing Case 5: The check-in and check-out dates are the same.

   Expected Result: A message is displayed: "Check-in and check-out date cannot be the same."

   Solution: Add a calendar component to prevent users from selecting the dates as both check-in and check-out. Add a verification code to ensure the check-out date is at least 1 day after the check-in date.

   Testing Case 6: No dates have been selected.

   Expected Result: A message is displayed: "Please select both check-in date and check-out date."

   Solution: Implement a validation check in the system to ensure the user cannot input null.

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 {
        // Q6.1 - Fill in the Code
        // 1. amount verification
        If (amount < 0) {
            throw new IllegalArgumentException("Your payment amount is negative!");
        }

        // 2. payment method verification
        if (!paymentMethod.equals("Alipay") && !paymentMethod.equals("Wechat")) {
            throw new IllegalArgumentException("Your payment method is invalid!");
        }

        // process payment
        else {
            System.out.println("Start to process your payment");
            System.out.println("Payment complete.");
            return isProcessed = true;
        }
    }

    public boolean isProcessed() {
        return isProcessed;
    }

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

//Q6.2 - Test cases

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class PaymentTest {
    @Test // amount valid, payment method valid
    public void Test_1() {
        Payment payment = new Payment(492327, 156940, 10000, "Alipay");
        assertDoesNotThrow(() → payment.processPayment());
    }

    @Test // amount not valid, payment method valid
    public void Test_2() {
        Payment payment = new Payment(492327, 156940, -1, "Alipay");
        IllegalArgumentException e = assertThrows(IllegalArgumentException.class,
                                                  () → payment.processPayment());
        assertEquals("Your payment amount is negative!", e.getMessage());
    }

    @Test // amount valid, payment method not valid
    public void Test_3() {
        Payment payment = new Payment(492327, 156940, 10000, "Visa");
        IllegalArgumentException e = assertThrows(IllegalArgumentException.class,
                                                  () → payment.processPayment());
        assertEquals("Your payment method is invalid!", e.getMessage());
    }
}
```

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

Alpha Testing

1. Define Testing Goals

The testing ensures all core functionalities (e.g., room search, booking, and payment) satisfy users' needs. The team members should also identify pivotal areas for improvement, focusing on technical functionality and initial usability problems.

2. Create a Testing Plan

The testing plan covers core modules, including customer and management interfaces. The team leader assigns tasks to team members, such as creating test data, setting up environments, and guiding users. Team members record testing results, errors, and feedback for analysis.

3. Prepare and Execute the Testing Process

The development team recruits internal team members as the primary testers during this stage and guides internal users to complete specific tasks. When processing the testing proceeds, the members will record the problems that the users encounter.

Beta Testing

1. User Recruitment and Preparation

The development team recruits testers from diverse backgrounds and provides clear instructions. Also, real-time support should be available for any issues, with different user groups assigned to different team members.

2. Execute the Testing Process

The development team allows external users to test the software in real-world conditions and encourages them to explore different scenarios. The members of the team record the real-world users' experiences and suggestions which will help improve system performance, interface design, and usability.

3. Data Analysis

After testing, the team should analyze results such as task success rates, completion times, and errors, categorizing issues into functional, design, and performance aspects.

Acceptance Testing

1. Define Acceptance Criteria

The team leader holds discussions with stakeholders to understand business, functional, and performance requirements. The technical members review the criteria to ensure they are testable and suitable for the project goals.

2. Plan Acceptance Testing

The team leader defines the test scope, objectives, and timeline. Each group is assigned customers to engage and ensure their participation, with a representative selected from each group.

3. Derive and Run Acceptance Tests

The technical team creates and categorizes acceptance test cases, covering key functionality, edge cases, and boundary conditions. The team verifies test availability and then executes the tests while customers follow guidelines. Results and feedback are collected to identify which case is successful or failed.

4. Negotiate Test Results

The whole team collaborates with stakeholders to review test results and determine solutions. The technical members explain technical limitations and propose feasible solutions based on the requirements of the stakeholders.

5. Accept or Reject the System

The team leader oversees the final decision-making process and ensures all stakeholders have reached an agreement. The customer representatives evaluate and approve or reject the system based on the testing results. If the system is accepted, the final version will be released. If the system is rejected, they collect feedback for further development.

Potential Problems and Solutions for Continuous Improvement

User Interface Issues:

Issue: Displaying page adaptability and responsiveness problems may occur.

Problem: Different sizes or the resolution of different devices may cause the interface to display inaccurately. Whenever the displaying page is zoomed in or out by the user, the interface may not be demonstrated completely.

Solution: Make the interface scalable to dynamically adjust and fit different devices and screens. To achieve this goal more efficiently, some strategies could be followed.

Applying Agile Methodologies

To conduct an iterative development, the team could divide the whole task into small, incremental improvements through circles called sprints. Also, a Scrum approach could be used to better conduct the sprints due to a retrospective meeting for evaluating team performance (e.g., JIRA could track tasks). For example, the team could first solve the zoom-in and out question and hold a meeting to evaluate the performance. After the first sprint is finished, a second one such as coping with different resolution problems could have proceeded. Until the last sprint is completed, the entire iterative development has been finished.

Using Continuous Integration and Continuous Delivery (CI/CD)

The technical developers in the team could integrate their code changes into a shared repository. And timely test and check to ensure that the new changes do not break the existing code. Continuously polish responsive design to cater to novel devices and screen resolutions. Delivery of the modified code to a staging environment for further detection. For example, the modified code should be implemented on different devices and different resolutions for constant testing before the final delivery.

Monitoring and Feedback

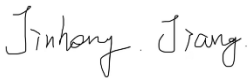
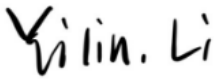
Regularly tests different types of devices, especially some of the newer ones that have come on the market, to ensure that the system interfaces are still compatible on devices with different display conditions. Devices without adaptation issues need to be tested and the tested details delivered to the development team.

Metrics for Continuous Improvement:

- For Development: The completion time and the frequency of the submitted code applied to the stage environment. The whole time is taken from commence to end. The development speed should be limited to 36 hours while 90% of the problems could be solved.
- For Product Quality: The defect density should be below an accepted rate (e.g., defects per 500 lines of code). Ensure at least 95% of the devices with different sizes and resolutions could be displayed successfully and the zoom in and out would not influence the display function.
- For Team Productivity: The team members should finish the task before the deadline. To be more efficient, several tasks could be processed concurrently. The team should complete nearly 98% of the assigned tasks within a sprint time.
- For Customer and User: After the bugs and improvements have been fixed and finished, some scoring and appraising interfaces could be displayed on the released version within a period to collect some feedback from the customers. Nearly 70% of satisfaction should be reached, otherwise, the team should improve again.

CPT203 Coursework
Peer review
Individual Contribution for Group Report

Group Number: 54

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. Pengkai Chen	2251486	20	
2. Jinhong Jiang	2251601	20	
3. Yilin Li	2255705	20	
4. Rui Sang	2251576	20	
5. Peilin Tu	2251487	20