UTM

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING

SEMESTER 2 /20242025

REPORT GROUP PROJECT

(SECJ2154 OBJECT ORIENTED PROGRAMMING)

SECTION 05

TOPIC:

SMART ROOM BOOKING SYSTEM

LECTURER NAME:

Dr LIZAWATI

GROUP 5	
NAME	**MATRIC NUMBER**
CORNELIA LIM ZHI XUAN	A23CS5044
FOUAD MAHMOUD FOUAD BESHIR	A23CS0017
LIM EN DHONG	A23CS0239
NG JIN EN	A23CS0146

Table of Contents

1.0 DESCRIPTION OF THE PROJECT

1.1 INTRODUCTION

The Smart Room Booking System is an advanced, object-oriented desktop application designed to automate and streamline room management and booking processes across university branches such as PSZ and PRZS at UTMJB. The system empowers customers to book smart rooms (Small or Large) for specific timeslots while providing administrators with comprehensive tools to manage users, branches, rooms, and bookings.

Key features include:

- Role-based access: Different functionalities for Admins (full system control) and Customers (booking management).
- Real-time availability checks: Prevents double-booking and ensures efficient resource allocation.
- Scalability: Supports multiple branches and room types with customizable capacity.
- User-friendly interface: Console-based menus for easy navigation.

1.2 OBJECTIVE AND SCOPES OF THE SYSTEM

- To design and implement an object-oriented solution to a multi-user room booking system.
- To demonstrate basic OOP principles such as inheritance, polymorphism, encapsulation,
- and exception handling.
- To provide administrators and customers with access to the system through simple-to-use
- menus.
- To provide a convenient and efficient platform for users to book rooms in various
- branches.
- To streamline the booking process for customers and provide administrators with the tools to manage users, facilities, and bookings effectively

1.3 PURPOSE OF THE SYSTEM

The main purposes of the system are:

- Customers can register for an account, browse available rooms across different branches, book a room for a specific date and time, view their booking history, and cancel their bookings.
- Administrators have full control over the system. They can manage user accounts (view and delete), manage branches/buildings (add, delete, view), and manage rooms within those branches (add, delete, view). They also have a global view of all bookings in the system.

2.0 ANALYSIS & DESIGN

2.1 WORKFLOW

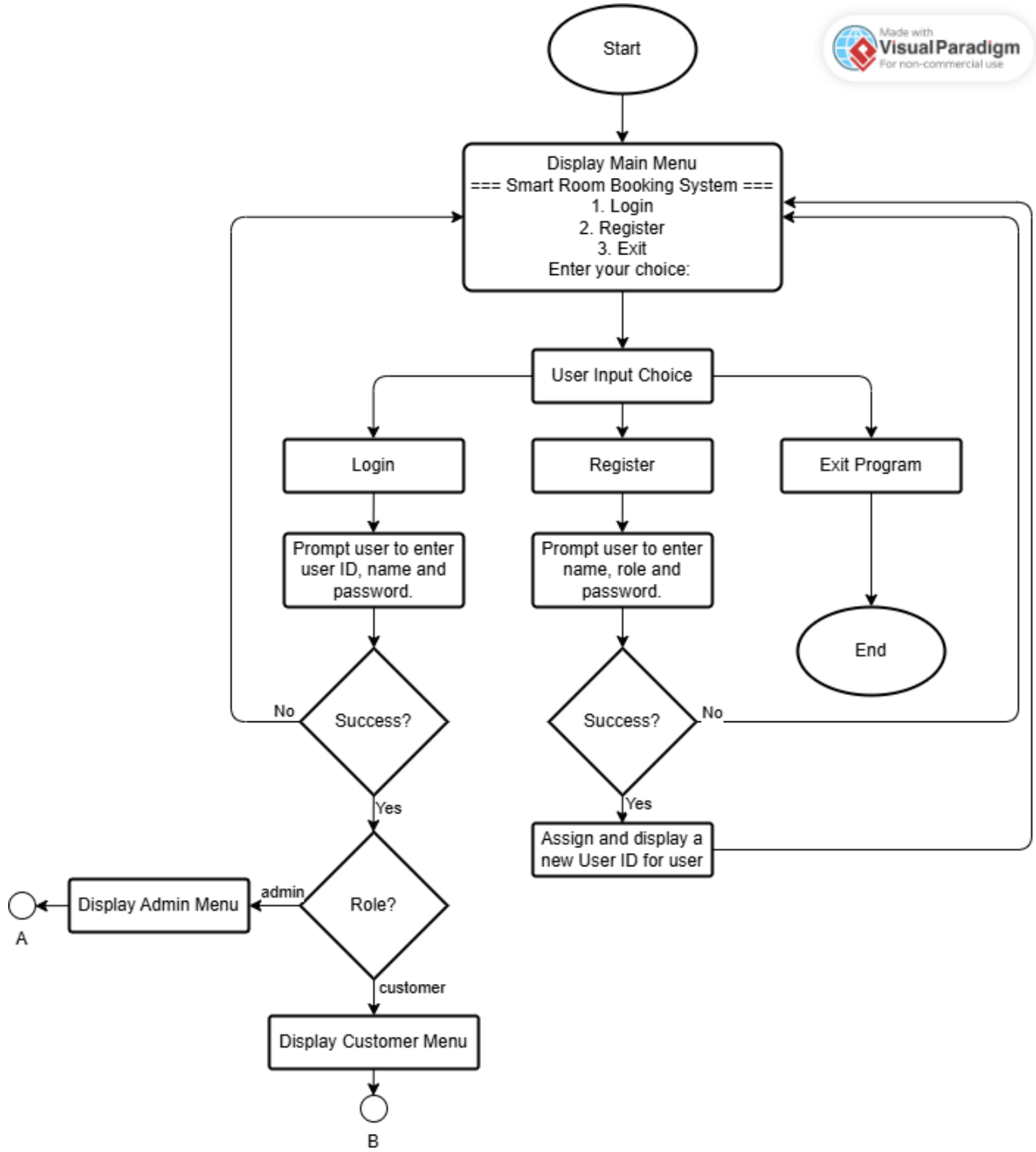


Figure 2.1.1 Flowchart for Main Menu

The Main Menu flowchart above described the initial interaction process for the Smart Room Booking System. The system displays a menu which includes “Login”, “Register” and “Exit” options. The system asks users to provide their User ID and name and password after they choose Login. The system will either loop back to the previous step or display an error message when the user enters incorrect credentials. After a successful login the system determines the user’s role (e.g. Customer) to direct them to their appropriate menu. The system generates a fresh User ID after users select Register while they provide their name and role and password information. The system will ask users to attempt registration again when registration fails because of duplicate details. The Exit option terminates the program.

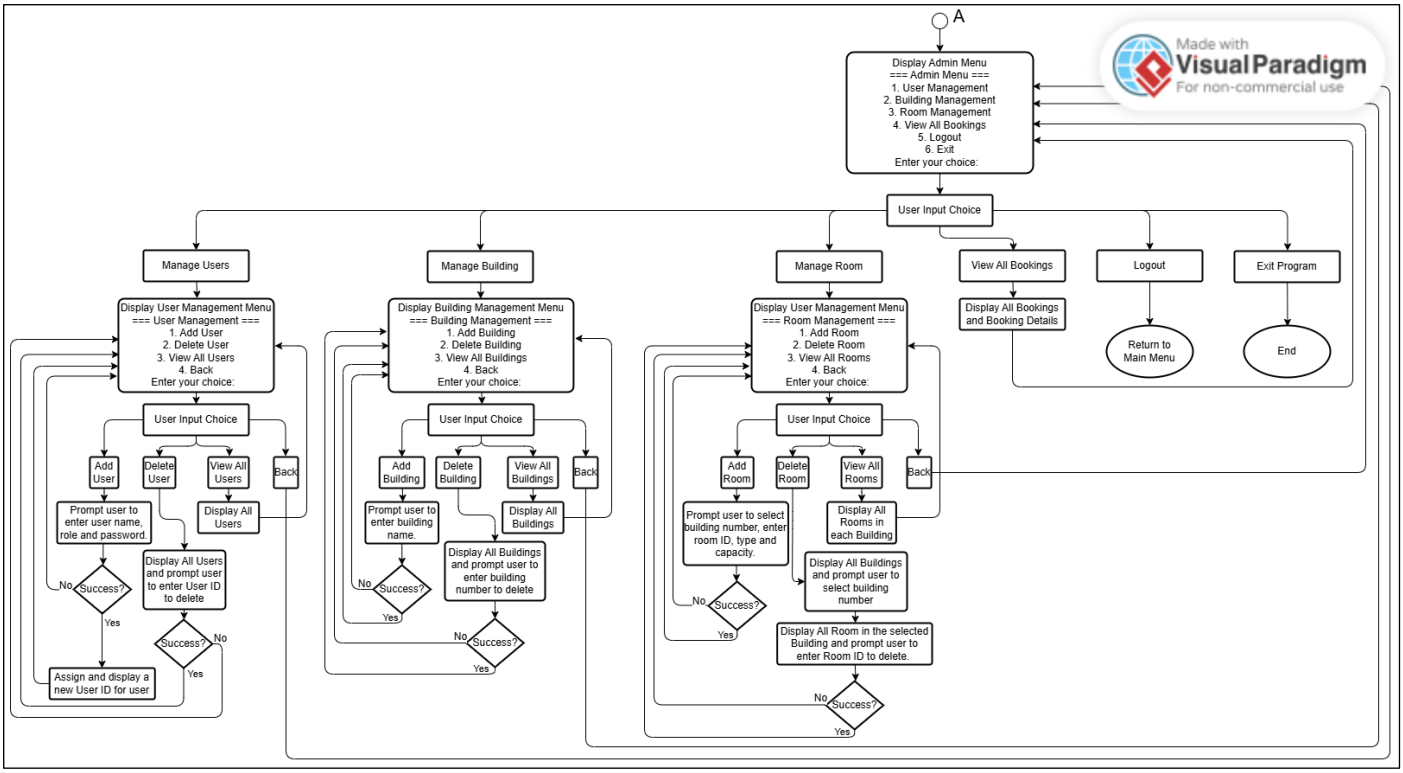


Figure 2.1.2 Flowchart for Admin Menu

The Admin Menu flowchart above highlighted the primary administrative operations available in the Smart Room Booking System, enabling system administrators to efficiently manage users, buildings, rooms and bookings. Upon accessing the Admin Menu, six main options are presented: User Management, Building Management, Room Management, View All Bookings, Logout and Exit. Choosing User Management leads to a submenu with options to add, delete or view users. Adding a user prompt for the name, role and password, followed by assignment of a unique User ID upon success. Deleting a user involves viewing the user list and confirming deletion by entering a valid User ID. Viewing all users simply displays the current list. The Building Management option functions similarly, allowing the admin to add buildings by entering a name, delete existing ones via ID input or view all registered buildings. In the Room Management section, the admin can add rooms by selecting a building and inputting room details like ID, type and capacity or delete rooms by navigating through buildings and selecting room IDs. The View All Rooms option provides a categorized display of rooms under each building. The View All Bookings function lists all reservation records along with their details for review. Choosing Logout returns the admin to the main menu without closing the session while Exit terminates the program entirely.

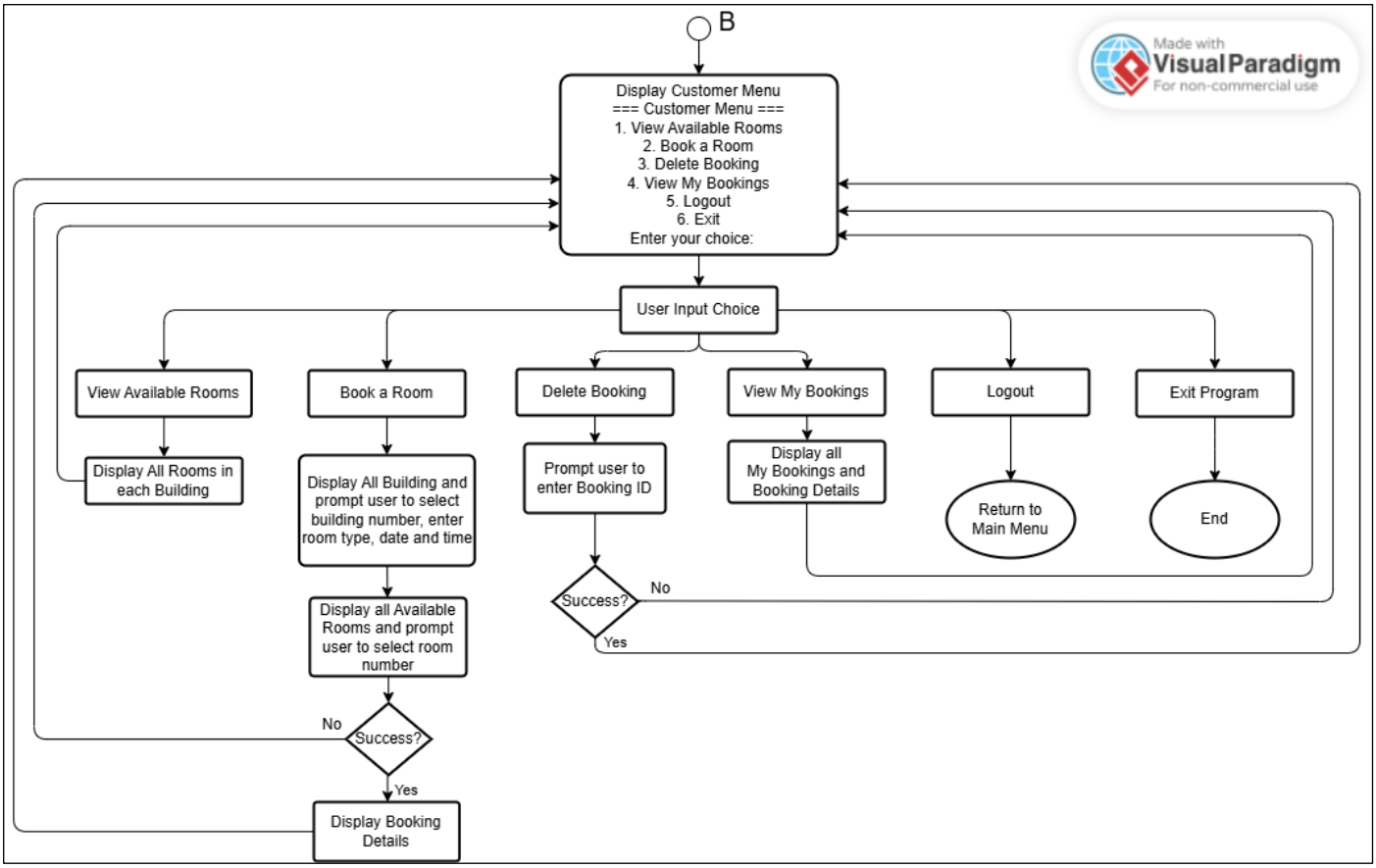


Figure 2.1.3 Flowchart for Customer Menu

The Customer Menu flowchart above explained the functionalities available to users after logging into the Smart Room Booking System. The menu presents six options: View Available Rooms, Book a Room, Delete Booking, View My Bookings, Logout and Exit. Selecting View Available Rooms displays a list of all rooms across buildings. To Book a Room, the user selects a building, specifies the room type, date and time and chooses from available rooms. Successful bookings display confirmation details while failures may prompt retries. The Delete Booking option requires a Booking ID and confirms deletion upon success. View My Bookings lists all reservations linked to the user's account. Logout returns the user to the Main Menu and Exit closes the program.

2.2 UML CLASS DIAGRAM

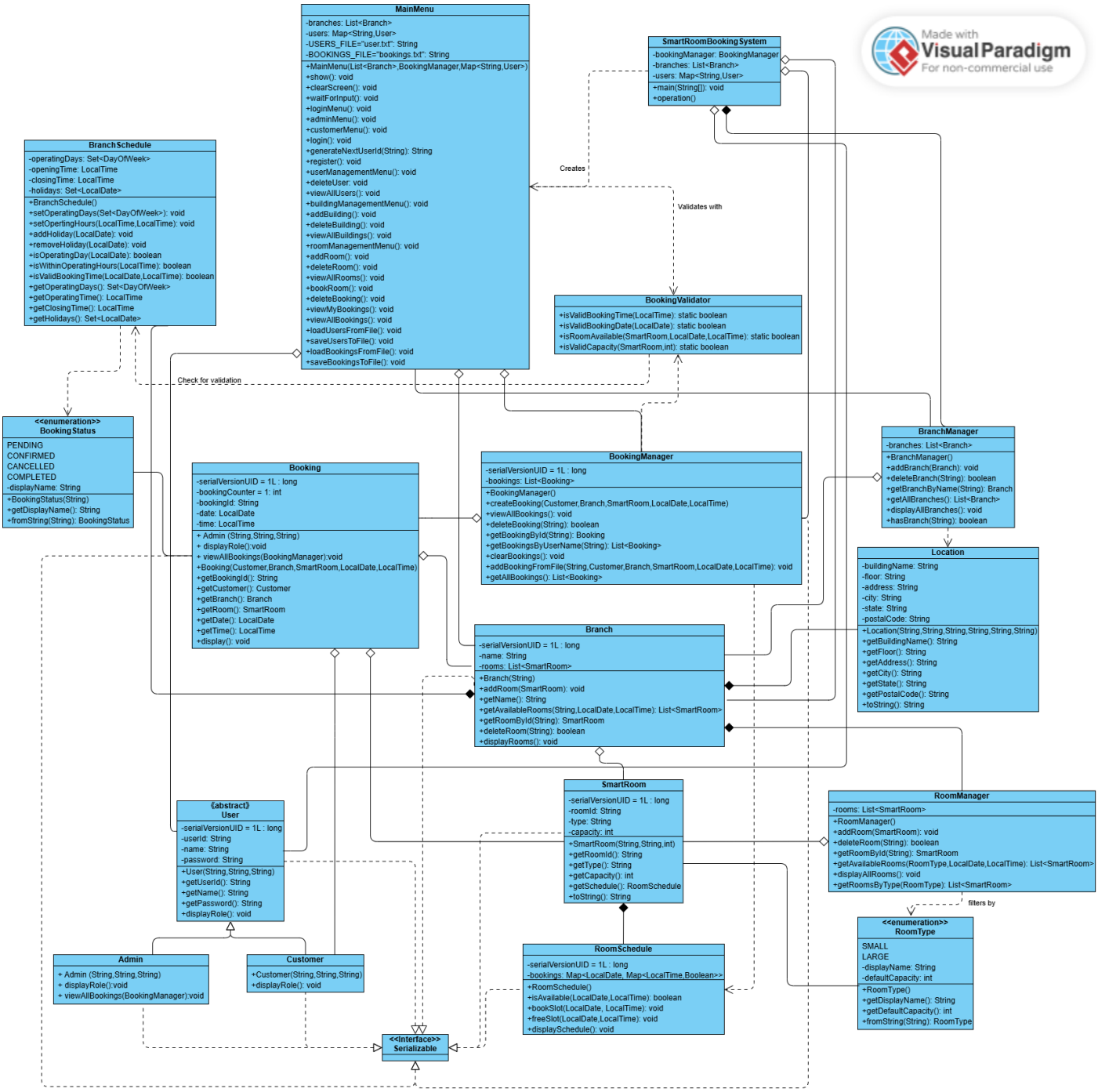


Figure 2.2.1 UML CLass Diagram

User Management Module:

Class User

Attributes	Description
userId (String)	A unique identifier for the user (e.g. “A001”, “C001”).
name (String)	The name of the user.
password (String)	The password used for authentication.
Methods	**Description**
User	A constructor of class User.
getUserId	A public getter method to retrieve the user’s ID.
getName	A public getter method to retrieve the user’s name.
getPassword	A public getter method to retrieve the user’s password.
displayRole	A n abstract method that forces subclasses (Admin, Customer) to provide their own implementation for displaying their specific role.

Class Admin

Methods	Description
Admin	A constructor of class Admin.
displayRole	(Overrides User) Implements the abstract method to print “Role: Admin” to the console identifying the user’s role.
viewAllBookings	This is a unique operation available only to Admin objects. It takes a BookingManager instance as a parameter and calls the manager’s viewAllBookings() method to display every booking currently in the system. It acts as a bridge allowing an Admin to trigger the system-wide booking report.

Class Customer

Methods	Description
Customer	A constructor of class Customer.
displayRole	(Overrides User) Implements the abstract method to print “Role: Customer” to the console identifying the user’s role.

Room Management Module:

Class SmartRoom

Attributes	Description
roomId (String)	A unique identifier for the room (e.g. “P101”).
type (String)	The type of the room, such as “Small” or “Large”. This aligns with the RoomType enum.
capacity (int)	The maximum number of people the room can accommodate.
schedule (RoomSchedule)	A composed object that holds the booking schedule for this specific room. Each room has its own RoomSchedule instance.
Methods	**Description**
SmartRoom	A constructor of class SmartRoom.
getRoomId	A public getter method to retrieve the room’s ID.
getType	A public getter method to retrieve the room’s type.
getCapacity	A public getter method to retrieve the room’s capacity.
getSchedule	A public getter method to access the room’s RoomSchedule object.
toString	(Overrides Object) Returns a concise, user-friendly string representation of the room’s state.

Class RoomSchedule

Attributes	Description
bookings (Map)	A data structure that stores the booked time slots. The keys are the LocalDate and LocalTime and the value indicates if the slot is booked.
Methods	**Description**
RoomSchedule	A constructor of class RoomSchedule.
isAvailable	Checks if the room is free at a specific date and time by looking up the entry in the bookings map. Returns true if available, false otherwise.
bookSlot	Marks a specific time slot as booked by adding an entry to the bookings map.
freeSlot	Marks a specific time slot as available again by removing its entry from the bookings map.
displaySchedule	A method to print the schedule of the room to the console.

Class RoomManager

Attributes	Description
rooms (List)	A collection that holds all the SmartRoom objects this manager is responsible for.
Methods	**Description**
RoomManager	A constructor of class RoomManager.
addRoom	Adds a new SmartRoom instance to the rooms list.
deleteRoom	Finds and removes a SmartRoom from the rooms list based on its unique roomId.
getRoomById	Searches the rooms list and returns the SmartRoom object that matches the given roomId.
getAvailableRooms	Filters the rooms list to find all rooms of a specific type that are available at the requested date and time.
displayAllRooms	Iterates through the rooms list and prints the details of each room to the console.
getRoomsByType	A new operation that filters and returns a list of all rooms that match a specific RoomType enum regardless of their availability.

Class RoomType

Attributes	Description
SMALL, LARGE	The constant, fixed values that represent the available room types.
displayName (String)	A user-friendly string representation for each enum constant (e.g. “Small”, “Large”).
defaultCapacity (int)	A new attribute that stores the default capacity associated with each room type (e.g. Small = 4, Large = 8).
Methods	**Description**
RoomType	A constructor of class RoomType.
getDisplayName	A public getter method to retrieve the user-friendly name of the room type.
getDefaultCapacity	A public getter method to retrieve the default seating capacity for this room type.
fromString	A static utility method that converts a string (like “Small”) into its corresponding RoomType enum constant ignoring case.

Booking Management Module:

Class Booking

Attributes	Description
bookingCounter (int)	A static variable that keeps track of the total number of bookings created across the entire system. It is used to auto-generate unique bookingId values (e.g. “BK0001”, “BK0002”).
bookingId (String)	A unique identifier for this specific booking, automatically generated upon

	creation (e.g. “BK0001”).
customer (Customer)	A reference to the Customer object who made the booking.
branch (Branch)	A reference to the Branch where the booked room is located.
room (SmartRoom)	A reference to the specific SmartRoom that has been booked.
date (LocalDate)	The date for which the room is booked.
Time (LocalTime)	The time for which the room is booked.
Methods	**Description**
Booking	A constructor of class Booking.
getBookingId	Public getter to retrieve the booking’s unique ID.
getCustomer	Public getter to retrieve the associated Customer.
getBranch	Public getter to retrieve the associated Branch.
getRoom	Public getter to retrieve the associated SmartRoom.
getDate	Public getter to retrieve the booking LocalDate.
getTime	Public getter to retrieve the booking LocalTime.
display	Prints all the details of this booking to the console in a formatted, user-friendly way.

Class BookingManager

Attributes	**Description**
bookings (List)	A collection that holds all the Booking objects in the system.
Methods	**Description**
BookingManager	A constructor of class BookingManager.
createBooking	The core method for creating a new booking. It validates the request (using BookingValidator), creates a new Booking object, adds it to the bookings list and updates the room’s schedule.
viewAllBookings	Iterates through the entire bookings list and calls the display() method for each one showing all bookings in the system.
deleteBooking	Finds a booking by its ID then removes it from the bookings list and frees up the corresponding time slot in the room’s schedule.
getBookingById	Searches the bookings list and returns the Booking object that matches the given ID.
getBookingsByUserName	Filters the bookings list and returns a new list containing all bookings made by a specific user.
clearBookings	A utility method that removes all Booking objects from the bookings list. This is typically used when loading fresh booking data from a file at system startup to ensure the in-memory list matches the file contents.
addBookingFromFile	Helper methods added to support loading booking data from a persistent file at system startup.
getAllBookings	A getter method that returns the entire list of bookings, often used for saving all bookings to a file.

Class BookingStatus

Attributes	**Description**
PENDING, CONFIRMED, CANCELLED, COMPLETED	The constant, fixed values that represent the possible states of a booking.
displayName (String)	A user-friendly string representation for each enum constant.
Methods	**Description**

BookingStatus	A constructor of class BookingStatus.
getDisplayName	A public getter method to retrieve the user-friendly name of the status.
fromString	A static utility method that converts a string like “Confirmed” into its corresponding BookingStatus enum constant.

Class BookingValidator

Methods	**Description**
isValidBookingTime	Checks if the given time is within the business operating hours (e.g. 8 AM to 8 PM).
isValidBookingDate	Checks if the given date is not in the past.
isRoomAvailable	Checks if the specified room is available at the requested date and time by querying its RoomSchedule.
isValidCapacity	Checks if the room’s capacity is sufficient for the number of people requested.

Branch Management Module:

Class Branch

Attributes	**Description**
name (String)	The name of the branch (e.g. “PSZ, UTMJB”).
rooms (List)	A collection of SmartRoom objects that belong to this branch.
Methods	**Description**
Branch	A constructor of class Branch.
addRoom	Adds a new SmartRoom to the branch’s room collection.
getName	Public getter to retrieve the branch name.
getAvailableRooms	Returns a list of rooms that are available at a specific date, time and type.
getRoomById	Finds and returns a specific SmartRoom by its ID.
deleteRoom	Removes a SmartRoom from the branch based on its room ID.
displayRooms	Prints information about all rooms in the branch.

Class BranchManager

Attributes	**Description**
branches (List)	A collection of all Branch objects in the system.
Methods	**Description**
BranchManager	A constructor of class BranchManager.
addBranch	Creates a new Branch with the given name and location and adds it to the branches list.
deleteBranch	Removes a Branch from the system based on its name or ID.
getBranchByName	Finds and returns a specific Branch by its name.
getAllBranches	Returns a list of all branches in the system.
displayAllBranches	Prints information about all branches in the system.
hasBranch	A utility method that quickly checks if a branch with the given name or ID already exists in the system. Returns true if it exists, false otherwise.

Class Location

Attributes	**Description**
buildingName (String)	The building name of the branch.
floor (String)	The specific floor number or level within the building where the branch is located.
address (String)	The address where the branch is located.
city (String)	The city where the branch is located.
state (String)	The state or province where the branch is located.
postalCode (String)	The postal or zip code of the branch location.
Methods	**Description**
Location	A constructor of class Location.
getBuildingName	Public getter to retrieve the building name.
getFloor	Public getter to retrieve the floor number.
getAddress	Public getter to retrieve the address.
getCity	Public getter to retrieve the city.
getState	Public getter to retrieve the state.
getPostalCode	Public getter to retrieve the postal code.
toString	Returns a formatted string representation of the complete address.

Class BranchSchedule

Attributes	**Description**
operatingDays (Set)	A collection of DayOfWeek enum values that defines which days the branch is open.
operatingTime (LocalTime)	The time when the branch opens each operating day.
closingTime (LocalTime)	The time when the branch closes each operating day.
holidays (Set)	A collection of LocalDate objects representing specific dates when the branch is closed.
Methods	**Description**
BranchSchedule	A constructor of class BranchSchedule.
setOperatingDays	Sets the operating days of the week for the branch.
setOperatingHours	Sets the opening and closing times for the branch.
addHoliday	Adds a specific date to the holidays list.
removeHoliday	Removes a specific date from the holidays list.
isOperatingDay	Checks if the given date is an operating day (not a holiday and within operating days).
isWithinOperatingHours	Checks if the given time is within the opening and closing hours.
isValidBookingTime	Combines the above checks to determine if a booking can be made at the given date and time.
getOperatingDays	Returns a copy of the operating days set.
getOpeningTime	Returns the opening time.
getClosingTime	Returns the closing time.
getHolidays	A public getter that returns a copy of the set of holidays allowing other parts of the system to view scheduled closing dates without being able to modify the original list.

Final Compilation and Output Formatting:

Class MainMenu

Attributes	Description
branches (List)	A reference to the list of all Branch objects, passed from SmartRoomBookingSystem.
bookingManager (BookingManager)	A reference to the application's BookingManager used to handle all booking-related actions.
users (Map)	A reference to the map of all Users used for authentication and user management.
currentUser (User)	Holds the User object of the currently logged-in user. It is null if no one is logged in.
USERS_FILE (String)	A constant that defines the filename for storing user data ("users.txt").
BOOKINGS_FILE (String)	A constant that defines the filename for storing booking data ("bookings.txt").
Methods	**Description**
MainMenu	A constructor of class MainMenu.
show	The main loop of the UI. It continuously checks if a user is logged in and calls either loginMenu() or the appropriate adminMenu()/ customerMenu().
clearScreen	A utility method to clear the console screen for a cleaner user experience.
waitForInput	A utility method that pauses the application and waits for the user to press Enter before continuing.
loginMenu	Displays the initial menu for non-logged-in users (Login, Register, Exit).
adminMenu	Displays the menu with all actions available to an Admin.
customerMenu	Displays the menu with all actions available to a Customer.
login	Prompts for user credentials, validates them against the users map and sets currentUser upon success.
generateNextUserId	A utility method that automatically generates the next available user ID for a given role. It scans existing user IDs and increments the number.
register	Prompts for new user details, creates a User object and saves it to the users map and users.txt file.
userManagementMenu	Displays the sub-menu for admin user management (Add, Delete, View).
deleteUser	Prompts for a user ID, removes the user from the users map and updates users.txt. Includes protection to prevent deletion of the default admin.
viewAllUsers	Iterates through the users map and displays information about all registered users in a formatted table.
buildingManagementMenu	Displays the sub-menu for admin building management.
addBuilding	Prompts for a building name, creates a new Branch object, and adds it to the branches list.
deleteBuilding	Displays all buildings, prompts for selection, and removes the chosen building from the branches list.
viewAllBuildings	Iterates through the branches list and displays the name of each building with a numbered list.
roomManagementMenu	Displays the sub-menu for admin room management.
addRoom	Prompts for building selection, room details (ID, type, capacity), creates a new SmartRoom and adds it to the selected building.
deleteRoom	Prompts for building selection, displays rooms in that building, prompts for room ID and removes the room from the building.
viewAllRooms	Iterates through all branches and calls each branch's displayRooms()

	method to show all rooms in the system.
bookRoom	Guides a customer through the process of booking a room interacting with the BookingManager.
deleteBooking	Prompts for a bookingId and asks the BookingManager to delete the corresponding booking.
viewMyBookings	Asks the BookingManager for all bookings associated with the currentUser.
viewAllBookings	Asks the BookingManager to display all bookings in the system (Admin only).
loadUsersFromFile	Reads user data from users.txt and populates the users map at startup.
saveUsersToFile	Writes the current state of the users map to users.txt whenever a user is added or deleted.
loadBookingsFromFile	Reads booking data from bookings.txt and populates the BookingManager at startup.
saveBookingsToFile	Writes all current bookings from the BookingManager to bookings.txt whenever a booking is created or deleted.

]Class SmartRoomBookingSystem

Attributes	**Description**
bookingManager (BookingManager)	A static instance of BookingManager that will be shared across the entire application to manage bookings.
branches (List)	A static list that holds all Branch objects. This collection represents all the physical locations in the system.
users (Map)	A static map that holds all User objects, keyed by their userId. This serves as the in-memory user database.
Methods	**Description**
main	The main method where the Java application begins execution. It coordinates the system's startup sequence.
initializeSystem	A helper method called by main to set up the initial state of the application. It creates the bookingManager, initializes the branches and users collections and can be used to add default data like a default admin user or branches.

3.0 OO CONCEPT IN THE SYSTEM

3.1 ASSOCIATION: AGGREGATION

Aggregation is a special form of association that represents a “has-a” relationship. It implies a weaker ownership, where the lifetime of the contained object is not dependent on the container object. Aggregated objects can exist independently and can be shared across different owners.

In our system, aggregation is used to model relationships like a MainMenu having a list of Branch objects, or a Branch containing a collection of SmartRoom objects. These objects are not tightly bound — they can exist and be used independently elsewhere in the system.

In MainMenu class, MainMenu holds these objects but does not own their lifecycles.

```
public class MainMenu {
    private List<Branch> branches;
    private BookingManager bookingManager;
    private Map<String, User> users;
```

Figure 3.1.1 Aggregation Concept in MainMenu Class

In Branch class, Branch aggregates SmartRoom objects to represent rooms in a building.

```
public class Branch implements Serializable {
    private static final long serialVersionUID = 1L;
    private String name;
    private List<SmartRoom> rooms;
```

Figure 3.1.2 Aggregation Concept in Branch Class

In BookingManager class, the BookingManager aggregates booking objects and manages them.

```
public class BookingManager implements Serializable {
    private static final long serialVersionUID = 1L;
    private List<Booking> bookings;
```

Figure 3.1.3 Aggregation Concept in BookingManager Class

3.2 ASSOCIATION: COMPOSITION

Composition is a strong form of association that implies ownership — when the container object is destroyed, so are the objects it contains. The contained object cannot exist without the container. This concept is used when a class needs to create and control the lifecycle of the contained object.

In our system, composition is clearly illustrated in how a SmartRoom contains a RoomSchedule, and how bookings are managed exclusively by the BookingManager. These child objects are not shared elsewhere and are dependent on the parent.

In SmartRoom class, the RoomSchedule is created and owned by the SmartRoom.

```
private RoomSchedule schedule;

public SmartRoom(String roomId, String type, int capacity) {
    this.roomId = roomId;
    this.type = type;
    this.capacity = capacity;
    this.schedule = new RoomSchedule();
}
```

Figure 3.2.1 Composition Concept in SmartRoom class

In RoomSchedule class, the schedule manages its own internal data structure, not accessible elsewhere.

```
public class RoomSchedule implements Serializable {
    private static final long serialVersionUID = 1L;
    private Map<LocalDate, Map<LocalTime, Boolean>> bookings;
```

Figure 3.2.2 Composition Concept in RoomSchedule class

In BookingManager class, all bookings are managed solely by the BookingManager. Bookings do not exist outside its context.

```
public class BookingManager implements Serializable {
    private static final long serialVersionUID = 1L;
    private List<Booking> bookings;

    public BookingManager() {
        this.bookings = new ArrayList<>();
    }
```

Figure 3.2.3 Composition Concept in BookingManager class

3.3 INHERITANCE

Inheritance allows one class (subclass/child) to inherit the properties and methods of another class (superclass/parent). It promotes code reuse and helps in modeling hierarchical relationships between classes.

In our system, inheritance is used to generalize common behavior into the User class, which is extended by the Admin and Customer classes. These subclasses share the structure of User but override behavior as needed.

Abstract superclass User class:

```java
public abstract class User implements Serializable {
    private static final long serialVersionUID = 1L;
    private String userId;
    private String name;
    private String password;

public abstract void displayRole();
```

Figure 3.3.1 Inheritance Concept in User class

Subclass Admin class:

```java
public class Admin extends User {
    public Admin(String userId, String name) {
        super(userId, name, password:"");
    }

    public Admin(String userId, String name, String password) {
        super(userId, name, password);
    }

    @Override
    public void displayRole() {
        System.out.println(x:"Role: Admin");
    }
```

Figure 3.3.2 Inheritance Concept in Admin class

Subclass Customer class:

```java
public class Customer extends User {
    public Customer(String userId, String name) {
        super(userId, name, password:"");
    }

    public Customer(String userId, String name, String password) {
        super(userId, name, password);
    }

    @Override
    public void displayRole() {
        System.out.println(x:"Role: Customer");
    }
}
```

Figure 3.3.3 Inheritance Concept in Customer class

Usage in MainMenu class:

```
if (currentUser instanceof Admin) {
    adminMenu();
} else {
    customerMenu();
}
```

Figure 3.3.4 Inheritance Concept in MainMenu class

3.4 POLYMORPHISM

Polymorphism allows objects of different subclasses to be treated as objects of a common superclass, and enables method overriding. It supports flexibility and extensibility in code by allowing one interface to serve many implementations.

In our system, polymorphism is used through method overriding (displayRole()), and dynamic method dispatch is applied when calling overridden methods via a User reference.

Superclass User class:

```
public abstract class User implements Serializable {
    private static final long serialVersionUID = 1L;
    private String userId;
    private String name;
    private String password;
```

```
public abstract void displayRole();
```

Figure 3.4.1 Polymorphism Concept in User class

Subclass Admin class:

```
public class Admin extends User {
    public Admin(String userId, String name) {
        super(userId, name, password:"");
    }

    public Admin(String userId, String name, String password) {
        super(userId, name, password);
    }

    @Override
    public void displayRole() {
        System.out.println(x:"Role: Admin");
    }
```

Figure 3.4.2 Polymorphism Concept in Admin class

Subclass Customer class:

```
public class Customer extends User {
    public Customer(String userId, String name) {
        super(userId, name, password:"");
    }

    public Customer(String userId, String name, String password) {
        super(userId, name, password);
    }

    @Override
    public void displayRole() {
        System.out.println(x:"Role: Customer");
    }
}
```

Figure 3.4.3 Polymorphism Concept in Customer class

MainMenu class:

- Uses instanceof checks:

```
if (currentUser instanceof Admin) {
    adminMenu();
} else {
    customerMenu();
}
```

Figure 3.4.4 Polymorphism Concept in MainMenu class

- Calling overridden method polymorphically:

```
private void viewAllUsers() {
   System.out.println(x:"=== All Users ===");
    int index = 1;
    for (User u : users.values()) {
        System.out.println("User " + index + ":");
        index++;
        System.out.printf(format:"ID: %-10s Name: %-25s\t", u.getUserId(), u.getName());
        u.displayRole();
        System.out.println(x:"");
    }
}
```

Figure 3.4.5 Polymorphism Concept in MainMenu class

- File saving logic uses polymorphism:

```
for (User user : users.values()) {
    String role = (user instanceof Admin) ? "Admin" : "Customer";
    bw.write(user.getUserId() + "," + user.getName() + "," + role + "," + user.getPassword());
    bw.newLine();
}
```

Figure 3.4.6 Polymorphism Concept in MainMenu class

3.5 EXCEPTION HANDLING

Exception handling is the mechanism to handle runtime errors using try, catch, and finally blocks. It promotes robustness by preventing application crashes due to unexpected events such as invalid inputs or file errors.

In our system, various forms of exception handling are implemented — for input mismatches, date/time parsing errors, and file I/O operations. These are used to provide clear error messages and maintain a smooth user experience.

MainMenu class:

- Handling input mismatch from user:

```
int choice = -1;
try {
    choice = scanner.nextInt();
} catch (InputMismatchException e) {
    System.out.println(x:"Invalid input! Please enter a number.");
    scanner.nextLine();
    waitForInput();
    return;
}
scanner.nextLine(); // Consume newline
```

Figure 3.5.1 Exception Handling Concept in MainMenu class

- Catching invalid date format:

```
System.out.print(s:"Enter date (yyyy-MM-dd): ");
String dateStr = scanner.nextLine();
LocalDate date;
try {
    date = LocalDate.parse(dateStr);
} catch (DateTimeParseException e) {
    System.out.println(x:"Invalid date format!");
    waitForInput();
    return;
}
```

Figure 3.5.2 Exception Handling Concept in MainMenu class

- File reading exceptions:

```
try (BufferedReader br = new BufferedReader(new FileReader(USERS_FILE))) {
    String line;
    while ((line = br.readLine()) != null) {
        String[] parts = line.split(regex:",");
        if (parts.length == 4) {
            String userId = parts[0];
            String name = parts[1];
            String role = parts[2];
            String password = parts[3];
            User user = role.equalsIgnoreCase(anotherString:"Admin") ? new Admin(userId, name, password) : new Customer(userId, name, passwo
            users.put(userId, user);
        }
    }
} catch (FileNotFoundException e) {
    // File not found, ignore (will be created on save)
} catch (IOException e) {
    System.out.println("Error reading users file: " + e.getMessage());
}
```

Figure 3.5.3 Exception Handling Concept in MainMenu class

- Booking conflicts:

```
try {
    Booking booking = bookingManager.createBooking((Customer) currentUser,
        selectedBranch, availableRooms.get(roomIndex), date, time);
    saveBookingsToFile();
    System.out.println(x:"Booking successful!");
    booking.display();
} catch (IllegalStateException e) {
    System.out.println("Error: " + e.getMessage());
}
waitForInput();
```

Figure 3.5.4 Exception Handling Concept in MainMenu class

3.6 ENCAPSULATION AND DATA HIDING

Encapsulation is the process of wrapping data and methods that operate on the data into a single unit (class). It ensures that internal representation of an object is hidden from the outside. Data hiding means restricting access to internal object details, allowing access only through public getters/setters.

In our system, all fields are kept private in our classes, and public methods (getX()) are used to access them. This controls how data is accessed and protects the integrity of the objects.

User class:

```
private String userId;
private String name;
private String password;

public User(String userId, String name, String password) {
    this.userId = userId;
    this.name = name;
    this.password = password;
}

public String getUserId() {
    return userId;
}

public String getName() {
    return name;
}

public String getPassword() {
    return password;
}
```

Figure 3.6.1 Encapsulation and Data Hiding Concept in User Class

SmartRoom class:

```java
private String roomId;
private String type; // "Small" or "Large"
private int capacity;
private RoomSchedule schedule;

public SmartRoom(String roomId, String type, int capacity) {
    this.roomId = roomId;
    this.type = type;
    this.capacity = capacity;
    this.schedule = new RoomSchedule();
}

public String getRoomId() {
    return roomId;
}

public String getType() {
    return type;
}

public int getCapacity() {
    return capacity;
}
```

Figure 3.6.2 Encapsulation and Data Hiding Concept in SmartRoom Class

RoomSchedule class:

```java
private Map<LocalDate, Map<LocalTime, Boolean>> bookings;

public RoomSchedule() {
    this.bookings = new HashMap<>();
}

public boolean isAvailable(LocalDate date, LocalTime time) {
    if (!bookings.containsKey(date)) {
        return true;
    }
    return !bookings.get(date).getOrDefault(time, defaultValue:false);
}
```

Figure Encapsulation and Data Hiding Concept in SmartRoom Class

The internal booking map is completely hidden from outside access — only controlled methods allow interaction.

4.0 USER MANUAL

The Smart Room Booking System is designed to manage room reservations across multiple buildings. The system supports two types of users:

- **Admin**: Full system management capabilities
- **Customer**: Room booking and personal booking management

(1) Room Types

- Small Room: 4-person capacity
- Large Room: 8-person capacity

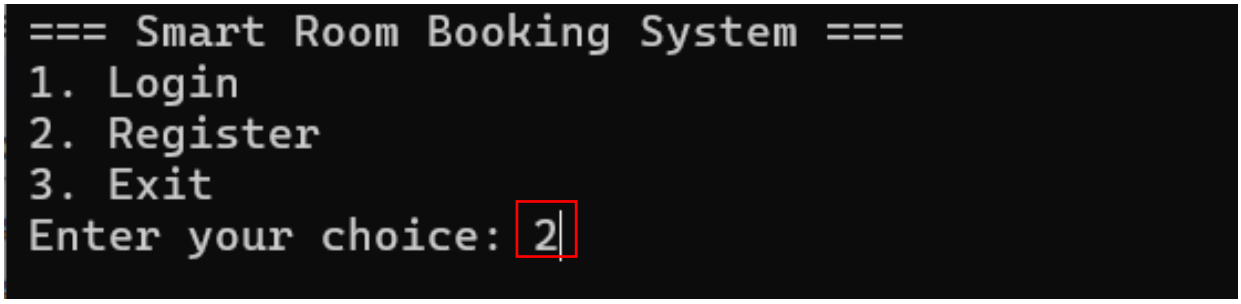
(2) Default Buildings

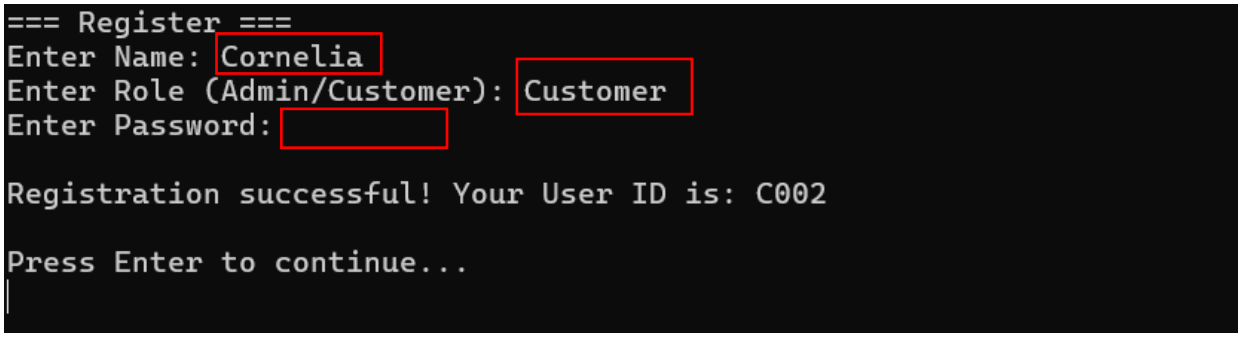
- PSZ, UTMJB: Rooms P101, P102 (Small), P201, P202 (Large)
- PRZS, UTMJB: Rooms R101, R102 (Small), R201, R202 (Large)

4.1 INITIALIZATION

***Note: The text surrounded with a red rectangle represents user input.**

1. At the beginning of the program, users have to register themselves by entering name, role and password.
2. After the information stored, the program will show the message *"Registration successful! "*.
3. Then, user can login the system with their user id, name, and password. All of the user id name, and password must be correct in order to login the system otherwise the system will show the message *"Invalid credentials! "*.
4. After successfully login into system, the system will show the menu by the user's role.





```
=== Login ===
Enter User ID: A000
Enter Name: Admin
Enter Password:

Login successful!

Press Enter to continue...
```

```
=== Smart Room Booking System ===
1. Login
2. Register
3. Exit
Enter your choice: 1
```

```
=== Login ===
Enter User ID: A001
Enter Name: Jin En
Enter Password:

Invalid credentials!

Press Enter to continue...
```

4.2 ADMIN VIEW

1. Admin Menu is shown.

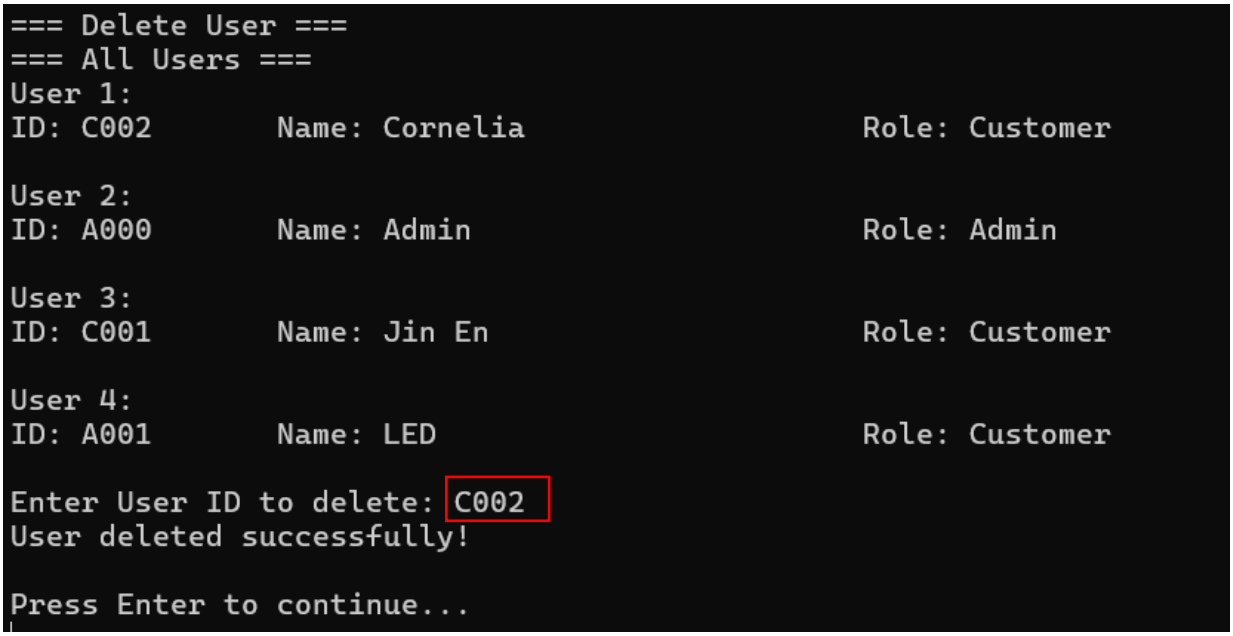
```
=== Admin Menu ===
1. User Management
2. Building Management
3. Room Management
4. View All Bookings
5. Logout
6. Exit
Enter your choice:
```

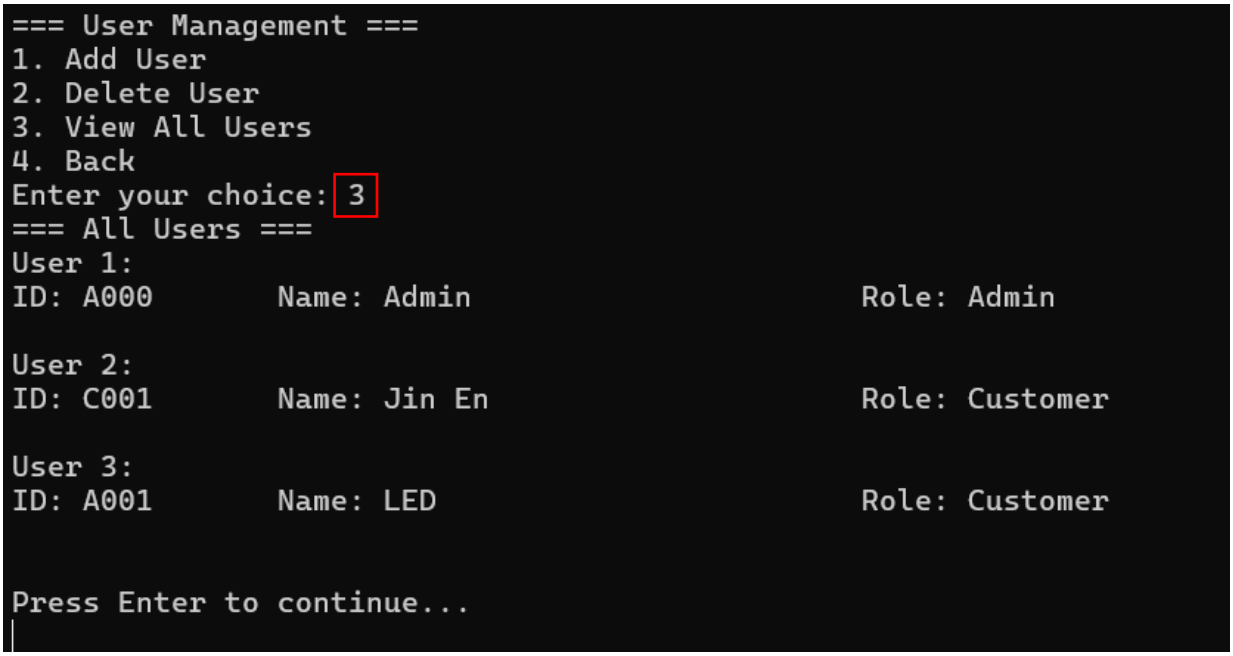
2. If **User Management** is chosen, then proceed to User Management's interface. Admin can choose to add, delete or view users, and back to main admin menu.
3. If **Add User** is chosen, proceed to register interface. Admin can register a new user.
4. If **Delete User** is chose, system will show admin the list of user and then admin can delete a user by entering user id. However, default admin cannot be deleted otherwise system will prompt a message *"Cannot delete default admin!"*.
5. If **View All Users** is chosen, the system will display list of active users.
6. If **Back** is chosen, proceed to admin main menu interface.

```
=== Admin Menu ===
1. User Management
2. Building Management
3. Room Management
4. View All Bookings
5. Logout
6. Exit
Enter your choice: 1
```

```
=== User Management ===
1. Add User
2. Delete User
3. View All Users
4. Back
Enter your choice: 1
```

```
=== Register ===
Enter Name: Cornelia
Enter Role (Admin/Customer): Customer
Enter Password:

Registration successful! Your User ID is: C002

Press Enter to continue...
```





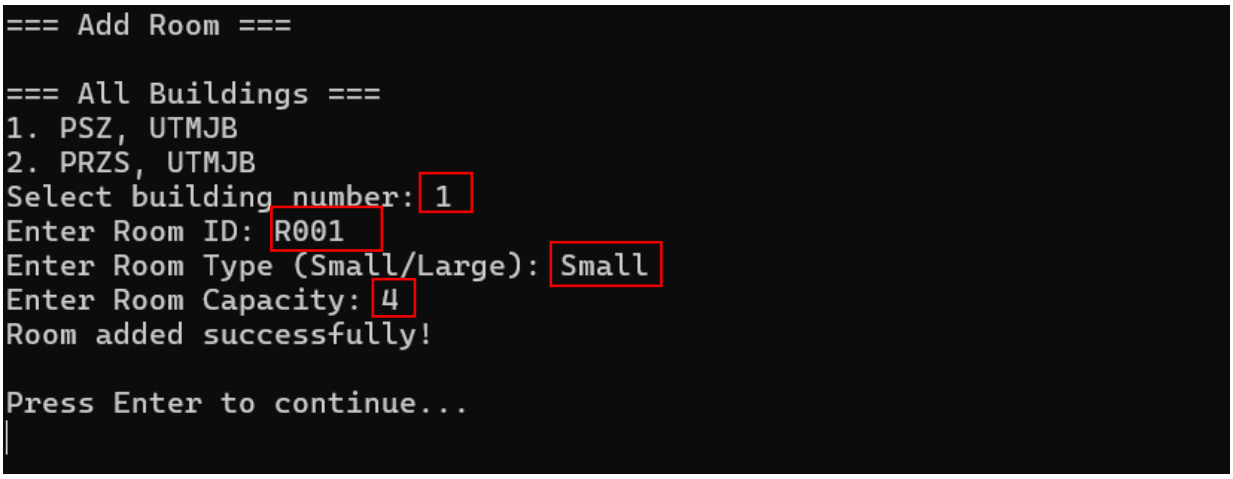
7. If **Building Management** is chosen, then proceed to Building Management's interface.
8. If **Add Building** is chosen, admin can add the place for the smart room by entering the building's name. Message *"Building added successfully! "* will show to user if the process of adding building is success.
9. If **Delete Building** is chosen, the system will show the list of buildings to user and admin can chose the number to delete the building.

10. If **View All Buildings** is chosen, the system will display list of buildings.
11. If **Back** is chosen, proceed to admin main menu interface.

```
=== Admin Menu ===
1. User Management
2. Building Management
3. Room Management
4. View All Bookings
5. Logout
6. Exit
Enter your choice: 2

=== Building Management ===
1. Add Building
2. Delete Building
3. View All Buildings
4. Back
Enter your choice: 1

=== Add Building ===
Enter Building Name: Library
Building added successfully!

Press Enter to continue...

=== Building Management ===
1. Add Building
2. Delete Building
3. View All Buildings
4. Back
Enter your choice: 2
=== Delete Building ===

=== All Buildings ===
1. PSZ, UTMJB
2. PRZS, UTMJB
3. Library
Enter building number to delete: 3
Building deleted successfully!

Press Enter to continue...

=== Building Management ===
1. Add Building
2. Delete Building
3. View All Buildings
4. Back
Enter your choice: 3

=== All Buildings ===
1. PSZ, UTMJB
2. PRZS, UTMJB

Press Enter to continue...
```

12. If **Room Management** is chosen, then proceed to Room Management's interface.

13. If **Add Room** is chosen, admin can add the smart rooms by entering the building's number and then the room id, type, and capacity that want to create. System will prompt message *"Room added successfully!"* after the room added.
14. If **Delete Room** is chosen, the system will show the list of buildings to user and ask user to input the building number which the room that want to delete located at. After that, system will show the list of rooms in the selected buildings and admin can enter room id to delete the room. Message *"Room deleted successfully!"* will prompt to user after the room is successfully deleted.
15. If **View All Rooms** is chosen, the system will display list of rooms.
16. If **Back** is chosen, proceed to admin main menu interface.



```
=== Delete Room ===

=== All Buildings ===
1. PSZ, UTMJB
2. PRZS, UTMJB
Select building number: 1

=== Rooms in PSZ, UTMJB ===
Room ID: P101, Type: Small, Capacity: 4
Room ID: P102, Type: Small, Capacity: 4
Room ID: P201, Type: Large, Capacity: 8
Room ID: P202, Type: Large, Capacity: 8
Room ID: R001, Type: Small, Capacity: 4
Enter Room ID to delete: R001
Room deleted successfully!

Press Enter to continue...
```

```
=== Room Management ===
1. Add Room
2. Delete Room
3. View All Rooms
4. Back
Enter your choice: 3

=== All Rooms ===

=== Rooms in PSZ, UTMJB ===
Room ID: P101, Type: Small, Capacity: 4
Room ID: P102, Type: Small, Capacity: 4
Room ID: P201, Type: Large, Capacity: 8
Room ID: P202, Type: Large, Capacity: 8

=== Rooms in PRZS, UTMJB ===
Room ID: R101, Type: Small, Capacity: 4
Room ID: R102, Type: Small, Capacity: 4
Room ID: R201, Type: Large, Capacity: 8
Room ID: R202, Type: Large, Capacity: 8

Press Enter to continue...
```

17. If **View All Bookings** is chosen, the system will show the list of bookings. However if there is no any bookings in the system, the system will show the message *"No bookings found."*.

```
=== Admin Menu ===
1. User Management
2. Building Management
3. Room Management
4. View All Bookings
5. Logout
6. Exit
Enter your choice: 4
```

```
=== All Bookings ===

=== Booking Details ===
Booking ID: BK0001
Branch: PSZ, UTMJB
Room: Room ID: P101, Type: Small, Capacity: 4
Date: 2025-06-23
Time: 12:30
Customer ID: C001

=== Booking Details ===
Booking ID: BK0002
Branch: PRZS, UTMJB
Room: Room ID: R202, Type: Large, Capacity: 8
Date: 2025-06-15
Time: 12:30
Customer ID: C001

=== Booking Details ===
Booking ID: BK0003
Branch: PSZ, UTMJB
Room: Room ID: P202, Type: Large, Capacity: 8
Date: 2025-06-23
Time: 16:30
Customer ID: C001

=== Booking Details ===
Booking ID: BK0004
Branch: PSZ, UTMJB
Room: Room ID: P102, Type: Small, Capacity: 4
Date: 2025-06-23
Time: 11:30
Customer ID: C001

Press Enter to continue...
```

18. If **Logout** is chosen, proceed to Main Menu interface.
19. If **Exit** is chosen, the system will log out. *Message "Thank you for using the Smart Room Booking System!"* will prompt to user.

```
=== Admin Menu ===
1. User Management
2. Building Management
3. Room Management
4. View All Bookings
5. Logout
6. Exit
Enter your choice: 6
Thank you for using the Smart Room Booking System!
PS C:\OOP_Y2S2\OOP_Y2S2\Project>
```

4.3 CUSTOMER VIEW

1. Customer Main Menu is shown.

```
=== Customer Menu ===
1. View Available Rooms
2. Book a Room
3. Delete Booking
4. View My Bookings
5. Logout
6. Exit
Enter your choice:
```

2. If **View Available Rooms** is chosen, then proceed to View Available Rooms' interface. The system will show the list of available rooms.

```
=== Customer Menu ===
1. View Available Rooms
2. Book a Room
3. Delete Booking
4. View My Bookings
5. Logout
6. Exit
Enter your choice: 1

=== All Rooms ===

=== Rooms in PSZ, UTMJB ===
Room ID: P101, Type: Small, Capacity: 4
Room ID: P102, Type: Small, Capacity: 4
Room ID: P201, Type: Large, Capacity: 8
Room ID: P202, Type: Large, Capacity: 8

=== Rooms in PRZS, UTMJB ===
Room ID: R101, Type: Small, Capacity: 4
Room ID: R102, Type: Small, Capacity: 4
Room ID: R201, Type: Large, Capacity: 8
Room ID: R202, Type: Large, Capacity: 8

Press Enter to continue...
```

3. If **Book a Room** is chosen, customer can make a smart room booking by selecting building number first, then choose the room type, date and time, and room number. After that, a message *"Booking successful!"* and the booking details will show to customer. If the time, date, and the room type selected is not available, *"No available rooms found!"* will shown.

```
=== Book a Room ===

=== All Buildings ===
1. PSZ, UTMJB
2. PRZS, UTMJB
Select building number: 1
```

```
=== Book a Room ===

=== All Buildings ===
1. PSZ, UTMJB
2. PRZS, UTMJB
Select building number: 1
Enter room type (Small/Large): Small
Enter date (yyyy-MM-dd): 2025-06-18
Enter time (HH:mm): 10:30

Available Rooms:
1. Room ID: P101, Type: Small, Capacity: 4
2. Room ID: P102, Type: Small, Capacity: 4
Select room number: 1
Booking successful!

=== Booking Details ===
Booking ID: BK0001
Branch: PSZ, UTMJB
Room: Room ID: P101, Type: Small, Capacity: 4
Date: 2025-06-18
Time: 10:30
Customer ID: C001

Press Enter to continue...
```

4. If **Delete Booking** is chosen, system will prompt user to input the booking id. If the booking id entered is not match with the existing booking id, *"Booking not found!"* will shown, otherwise *"Booking deleted successfully!"* will shown.

```
=== Customer Menu ===
1. View Available Rooms
2. Book a Room
3. Delete Booking
4. View My Bookings
5. Logout
6. Exit
Enter your choice: 3
```

```
=== Delete Booking ===
Enter Booking ID: BK0002
Booking deleted successfully!

Press Enter to continue...
```

5. If **View My Bookings** is chosen, system will display a list of the bookings done by customer.

```
=== Customer Menu ===
1. View Available Rooms
2. Book a Room
3. Delete Booking
4. View My Bookings
5. Logout
6. Exit
Enter your choice: 4
```

```
=== My Bookings ===

=== Booking Details ===
Booking ID: BK0001
Branch: PSZ, UTMJB
Room: Room ID: P101, Type: Small, Capacity: 4
Date: 2025-06-18
Time: 10:30
Customer ID: C001

Press Enter to continue...
```

6. If **Logout** is chosen, proceed to Main Menu interface.
7. If **Exit** is chosen, the system will log out. *Message "Thank you for using the Smart Room Booking System!"* will prompt to user.

```
=== Customer Menu ===
1. View Available Rooms
2. Book a Room
3. Delete Booking
4. View My Bookings
5. Logout
6. Exit
Enter your choice: 6
Thank you for using the Smart Room Booking System!
PS C:\OOP_Y2S2\OOP_Y2S2\Project>
```

5.0 CONCLUSION

The Smart Room Booking System represents a significant advancement in resource management for multi-branch environments. By integrating user-centric design with robust backend functionality, the system addresses critical challenges in room allocation, scheduling efficiency, and administrative oversight. Its dual-role architecture, which distinguishing between administrative control and customer autonomy. This creates a balanced ecosystem where operational oversight coexists with intuitive self-service capabilities. This duality not only streamlines institutional workflows but also empowers end-users to manage reservations independently, reducing administrative bottlenecks.

Beyond immediate efficiency gains, the system fosters sustainable resource management by minimizing idle room time and maximizing accessibility. Its success illustrates how targeted technological interventions can transform routine administrative tasks into strategic assets. Future iterations could enhance predictive analytics for demand forecasting or integrate IoT sensors for real-time occupancy verification. Ultimately, this project demonstrates that thoughtfully engineered systems not only solve logistical challenges but also elevate user experiences, setting a precedent for next-generation facility management solutions.

All in all, the Smart Room Booking System transcends conventional scheduling paradigms, offering a scalable, user-empowered framework that redefines efficiency in shared-space utilization.

6.0 TASK DISTRIBUTION

Category	Task	Person In-Charge
Coding Implementation	User Management Module – User.java – Admin.java – Customer.java	CORNELIA LIM ZHI XUAN
	Room Management Module – SmartRoom.java – RoomSchedule.java – RoomManager.java – RoomType.java	LIM EN DHONG
	Booking Management Module – Booking.java – BookingManager.java – BookingStatus.java – BookingValidator.java	NG JIN EN
	Branch Management Module – Branch.java – BranchManager.java – Location.java – BranchSchedule.java	FOUAD MAHMOUD FOUAD BESHIR
	Final Compilation and Output Formatting – MainMenu.java – SmartRoomBookingSystem.java	NG JIN EN
Group Report	Description of the project – Introduction – Objective and scope of the system – Purpose of the system	FOUAD MAHMOUD FOUAD BESHIR
	Analysis & Design – Workflow (flowchart)	LIM EN DHONG & FOUAD MAHMOUD FOUAD BESHIR
	– UML Class Diagram	LIM EN DHONG
	OO Concept	CORNELIA LIM ZHI XUAN
	User Manual	NG JIN EN

	Conclusion	
Presentation Video	Presentation slide – Project description – Analysis and design – OO concept – System demo	ALL MEMBERS
Progress Video	– Introduction of Team Members and Roles – Evidence of Collaboration – Demonstration of Project Progress – Team Reflection	ALL MEMBERS

7.0 REFERENCES

1. Progress Video: https://youtu.be/SpF_7l7-Y7s
2. Presentation Video: https://youtu.be/1GfC5GUsAvU