Iteration 2

Project Team: Naveen Prabakar, Ozair Nurani, Nicholas Wang, Brendan Kraft, Pranava Sai Maganti

Use Case Names

1. Guest calls room service → Name: Call Room service (Brendan)
2. MiniBar and Room Service Setup + Calculations → Name: Mini-Bar/Room Service Usage Tracking (Ozair)
3. Guest writes reviews → Name: Write Reviews (Nicholas)
4. Analytics team views data → Name: Analyze past data (Naveen)
5. Executives set hotel prices → Name: Set Hotel Prices (Naveen)
6. System Calculates Payment → Name: Calculate bills (Pranava)
7. HR reviews Guest feedback → Name: Review feedback (Nicholas)
8. Hotel Worker Clocks-in/out → Name: Clock in & out (Ozair)
9. Guest cancels reservation → Name: Cancel the reservation (Brendan)
10. Hotel staff should be able to login and check appropriate tasks → Name: Login for Workers (Pranava)

Use Case Cards

Use Case: Analyze Past Data (Naveen)	
Primary Actor	Data Team, Executive Staff
Triggering Event	The executive team requests performance insights to support their decisions making
Success Guarantee	The Data Team generates a report based off the executive staff needs
Preconditions	1. There is historical data stored somewhere to use for reports 2. Data Team has the proper access to get the historical data
Main Success Scenario	1. Data Team logs into the system 2. The team selects the time period for

Use Case: Analyze Past Data (Naveen)	
	analysis 3. The System retrieves the correct data 4. Data team writes a report based off the data 5. Report is saved 6. Executive team is notified off the finished report
Alternative/Extensions	1. If the data cannot answer the question, then no report will be generated and Executive team will be informed 2. If Historical data does not exist for mentioned time period, then no report will be generated and executive team will be informed

Use Case: Set Prices (Naveen)	
Primary Actor	Executive Staff
Triggering Event	A new pricing season is started or sudden damage changes
Success Guarantee	The price is updated and reflected on the price of the rooms
Preconditions	1. Reports are available to review to help set the prices 2. The data team has completed their analysis of the data 3. Executive staff has access to change the prices
Main Success Scenario	1. Executive team requests reports for the data team 2. Executive team reviews the reports from the data team 3. Executive team logs to panel to change the prices 4. The executive team sets prices based on the analysis (demands, etc.)
Alternative/Extensions	1. If Data team cannot generate report,

Use Case: Set Prices (Naveen)	
	then prices cannot be set 2. If prices are set to be below invalid numbers, then prices won't be set

Use Case: Mini-Bar and Room Service setup (Ozair)	
Primary Actor	Guest
Triggering Event	- Guest consumes an item from the mini bar, or - Guest places a room service order via the system.
Success Guarantee	- The consumed mini bar item or room service order is successfully added to the guest's bill. - Guest's total charges are updated correctly.
Preconditions	- The guest must have an active reservation/check-in for the room. - The room and mini bar inventory / room service menu must exist in the system.
Main Success Scenario	1) Guest selects mini bar item or room service item to consume/order. 2) System verifies guest's room and reservation are active. 3) System checks availability of the item (in mini bar or room service menu). 4) System records the item consumption/order. 5) System updates the guest's bill to

Use Case: Mini-Bar and Room Service setup (Ozair)	
	include the cost. 6) Confirmation is shown to the guest (or logged by staff).
Alternative/Extensions	1) Item not available: System informs the guest that the item is out of stock. 2) Guest not checked in: System prevents adding charges and prompts check-in first. 3) Billing failure: If updating the bill fails, system logs the error and informs staff to manually adjust.

Use Case: Clock in & Out (Ozair)	
Primary Actor	Hotel Worker
Triggering Event	Hotel worker chooses to clock in at the start of a shift or clock out at the end of a shift
Success Guarantee	The system successfully records the worker's clock-in or clock-out timestamp.
Preconditions	Worker is logged into the system
Main Success Scenario	Clock In - Worker navigates to the "Clock In/Out" page. - Worker selects Clock In. - System records the clock-in time. - System confirms successful clock-in. Clock Out - At the end of the shift, worker returns to the same page. - Worker selects Clock Out. - System records the clock-out time. - System confirms successful clock-out.

Use Case: Clock in & Out (Ozair)	
Alternative/Extensions	1) If Worker has already clocked in, and the worker tries to clock back in the system tells them they are already clocked in 2) If the worker tries to clock out without having clocked in, system shows: You cannot clock out before clocking in

Use Case: Login for Workers (Pranava)	
Primary Actor	All Hotel Staff
Triggering Event	When someone tries to use the application, they're asked to choose whether they're a guest or a worker.
Success Guarantee	The staff member is successfully authenticated and taken to the worker dashboard with the correct permissions.
Preconditions	1. The worker has an active account in the system. 2. The worker's role and permissions are already configured. 3. The application is connected to the authentication service.
Main Success Scenario	1. The staff member selects the Worker login option. 2. The system displays the worker login screen. 3. The staff member enters valid credentials. 4. The system verifies the credentials. 5. The worker is authenticated. 6. The system loads the appropriate worker dashboard based on their role.
Alternative/Extensions	1. **Invalid credentials:** The system shows an error and allows the worker to retry. 2. **Forgotten password:** Worker selects

Use Case: Login for Workers (Pranava)	
	"Forgot Password" and follows the recovery flow.

Use Case: Calculate Bills (Pranava)	
Primary Actor	Hotel Guests
Supporting Actors	Hotel Staff, Billing System
Triggering Event	A guest checks out or requests a bill, or staff initiates bill generation from the system.
Success Guarantee	The system produces an bill that reflects all charges, discounts, taxes, and current room rates.
Preconditions	1. Guest stay details, room rates, and service usage are already recorded in the system. 2. Pricing data and tax rules are up to date. 3. The billing system is available and connected to the reservation database.
Main Success Scenario	1. The guest requests a bill or the staff initiates bill calculation at checkout. 2. The system retrieves all stay details, including room type, duration, and services used. 3. The system fetches current pricing, tax rates, and any applicable discounts. 4. The system calculates the total amount due. 5. The calculated bill is displayed to staff or directly to the guest. 6. Guest reviews and confirms the bill. 7. The system finalizes and saves the bill.
Alternative/Extensions	1. System error during calculation: System logs the issue and prompts

Use Case: Calculate Bills (Pranava)	
	staff to retry or contact support. 2. The system alerts staff that required information is incomplete.

Use Case: Call House Keeping(Brendan)	
Primary Actor	Hotel Guest
Triggering Event	A checked-in guest wants to request room cleaning, schedule a cleaning time, or request additional amenities (e.g., towels, toiletries, extra pillows)
Success Guarantee	The guest's housekeeping request is captured, routed to the correct staff, scheduled/queued appropriately, and a confirmation is provided.
Preconditions	Guest is checked in and has an active room assigned in the system. Housekeeping services (cleaning types, amenity catalog, service hours) are configured and available. The housekeeping request channel (app/in-room tablet/TV/phone integration) is online.
Main Success Scenario	1. Guest opens the hotel app, in-room tablet/TV, or housekeeping interface. 2. Guest navigates to "Housekeeping" / "Room Cleaning & Amenities". 3. Guest's room is auto-detected or the guest confirms the room number. 4. Guest selects the service type: e.g., "Full Cleaning," "Quick Tidy," "Turn-Down," or "Amenity Request

Use Case: Call House Keeping(Brendan)	
	Only." 5. If scheduling is allowed, guest selects a preferred time window (e.g., "As soon as possible," "10:00–12:00," "Later today"). 6. Guest selects any additional amenities (e.g., extra towels, toiletries, linens, pillows). 7. Guest reviews the request summary (services, time window, amenities) and submits the request. 8. System validates the request (service hours, staff availability, room status such as Do Not Disturb). 9. System creates a housekeeping work order linked to the guest's room and stay. 10. System notifies housekeeping staff/management system with request details and scheduled time. 11. System displays a confirmation to the guest including estimated time of service and any notes (e.g., "Please remove Do Not Disturb sign at scheduled time").
Alternative/Extensions	If housekeeping is unavailable at the requested time (outside service hours or fully booked), system proposes alternative time slots and lets the guest select one. If the room is currently flagged Do Not Disturb, system informs the guest that housekeeping cannot enter and offers to schedule a time after DND is removed.

Use Case: Call House Keeping(Brendan)	
	If a requested amenity is unavailable or out of stock, system informs the guest and suggests alternatives (e.g., different pillow type, different toiletry brand). If the guest's room cannot be validated as checked-in or active, system shows an error and suggests contacting the front desk. If there is a system error sending the request to housekeeping staff, system logs the issue, informs the guest that there was a problem, and prompts staff to follow up manually (e.g., via fallback notification to front desk).

Use Case: Call Room Service(Brendan)	
Primary Actor	Hotel Guest
Supporting Actors	Room Service Staff, Kitchen/Housekeeping System
Triggering Event	A checked-in guest wants to order food or request room-service items (e.g., towels, toiletries).
Success Guarantee	The guest's room service request is captured, routed to the correct staff, and a confirmation.
Preconditions	Guest is checked in and has an active room assigned in the system. Room service features (menu, item catalog, hours) are configured and available. The room service ordering channel (app/TV interface/phone integration) is online.
Main Success Scenario	1. Guest opens the hotel app/in-room

Use Case: Call Room Service(Brendan)	
	tablet/TV or room-service interface. 2. Guest navigates to “Room Service”. 3. Guest selects the room (if needed) or the system auto-detects the room. 4. Guest browses the menu/items and adds desired items to the request (food, drinks, or amenities). 5. Guest reviews the order, sees estimated charges, and submits the request. 6. System validates the order (availability, service hours, room status). 7. System creates a room service order linked to the guest’s room and stay. 8. System notifies room service staff/kitchen/housekeeping with order details. 9. System displays an order confirmation the guest.
Alternative/Extensions	If room service is closed (outside service hours), system informs the guest that room service is unavailable and, if applicable, shows alternative options. If a selected item is unavailable or out of stock, system prompts the guest to choose a replacement or remove the item. If the guest’s room is not recognized as checked-in or active, system shows an error and suggests contacting the front desk. If there is a system error sending the order to staff, system logs the issue, informs the guest, and prompts staff to follow up

Use Case: Call Room Service(Brendan)	
	manually.

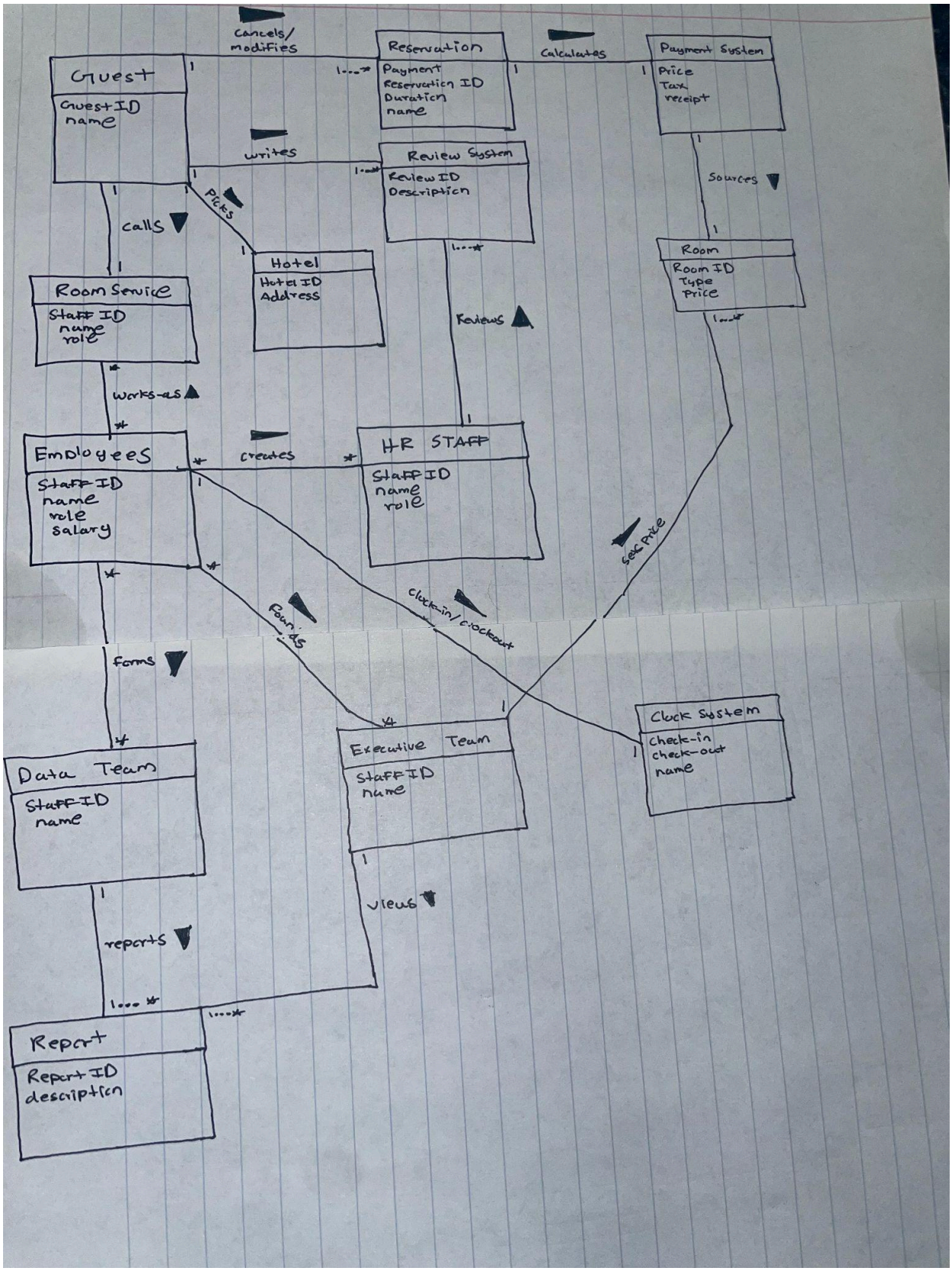
Use Case: Write Reviews(Nicholas)	
Primary Actor	Hotel Guest
Triggering Event	A guest wants to write reviews for their stay
Success Guarantee	The guest's stay or booking is matched, and the feedback is recorded in the system.
Preconditions	Guest has a valid booking Guest contact info is accessible Review system is online
Main Success Scenario	1. Guest opens the hotel app/in-room tablet/TV. 2. Guest navigates to the “Write Review” section. 3. System identifies the guest’s past stay. 4. Guest writes and submits their review. 5. System validates the review and links it to the guest’s stay. 6. Review is recorded and stored in the system. 7. Confirmation is displayed to the guest.
Alternative/Extensions	If the guest’s stay cannot be found, the system shows an error, and prompts the guest to contact support

Use Case: Write Reviews(Nicholas)	
	If the review system is down, system logs the issue and informs the guest to try later. If inappropriate content is detected, the system flags it for moderation.

Use Case: Review Feedback (Nicholas)	
Primary Actor	HR Management
Triggering Event	A guest submits a review that requires a response or acknowledgement
Success Guarantee	Guest receives feedback or acknowledgment from hotel staff
Preconditions	Guest reviews are recorded in the system Staff has access to the reviews Staff has appropriate permissions to respond
Main Success Scenario	1. Staff logs into the hotel management system 2. Staff navigates to the “Guest Reviews” section 3. System displays new or pending reviews 4. Staff selects a review to respond to. 5. Staff writes feedback or acknowledgement 6. System records the staff response and links it to the guest 7. Guest is notified that feedback has been provided
Alternative/Extensions	If the review is inappropriate, staff can flag it for moderation

Use Case: Review Feedback (Nicholas)	
	If staff response failed, system logs, and prompts retry If guest contact info is missing, notification cannot be sent

Domain Model


Guest
GuestID
name
Cancels/
modifies
Reservation
Payment
Reservation ID
Duration
name
Calculates
Payment System
Price
Tax
receipt
writes
Review System
ReviewID
Description
Sources
Picks
calls
Hotel
Hotel ID
Address
Room
Room ID
Type
Price
Room Service
Staff ID
name
role
Reviews
works-as
Employees
Staff ID
name
role
salary
creates
HR STAFF
Staff ID
name
role
Sets Price
Clock-in/clockout
Reports
forms
Clock System
check-in
check-out
name
Executive Team
Staff ID
name
Data Team
Staff ID
name
views
reports
Report
Report ID
description

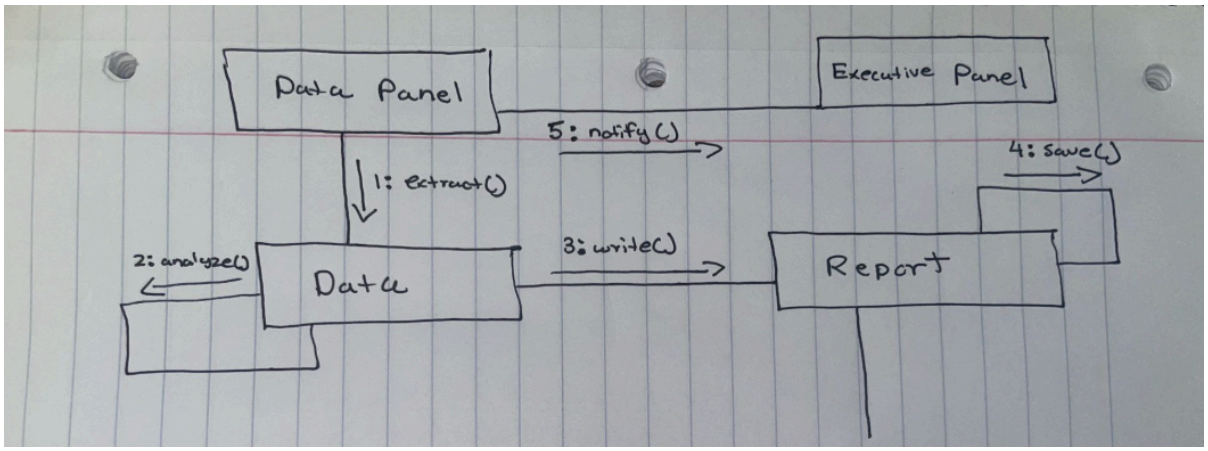
Justification for Design Choices

Classes	Grasp Principle	Explanation
Calling Room Service	Creator	Room service is called by the guest. This needs to create a room service ticket, for the staff to be able to respond to
Picking Hotel Spot/Descriptions	Information Expert	In order for the guest to pick a hotel location, information needs to be provided about the hotel. Hotel Class would need to hold the information/descriptions of the location.
Writing Reviews	Creator	The reviews need to be saved to a file somewhere. This needs a review object to be created to do so.
Analyzing/Creating reports	Creator, Controller	The Analytics team needs to analyze the data and make reports. This would involve making reports so the executive team can review them. This would also involve coordinating with the database files to be able to pull data, then analyze it.
Setting Hotel Prices	Information Expert	In order to set the room prices, the class would require information about all the rooms about a particular hotel they selected.
Calculate payment	Information Expert	For the system to calculate the payment, it needs access to the reservation files
Reviewing Feedback	Controller	The HR department would look at the views by the Customers and analyze them. It's controller because it's the bridge between Customer's

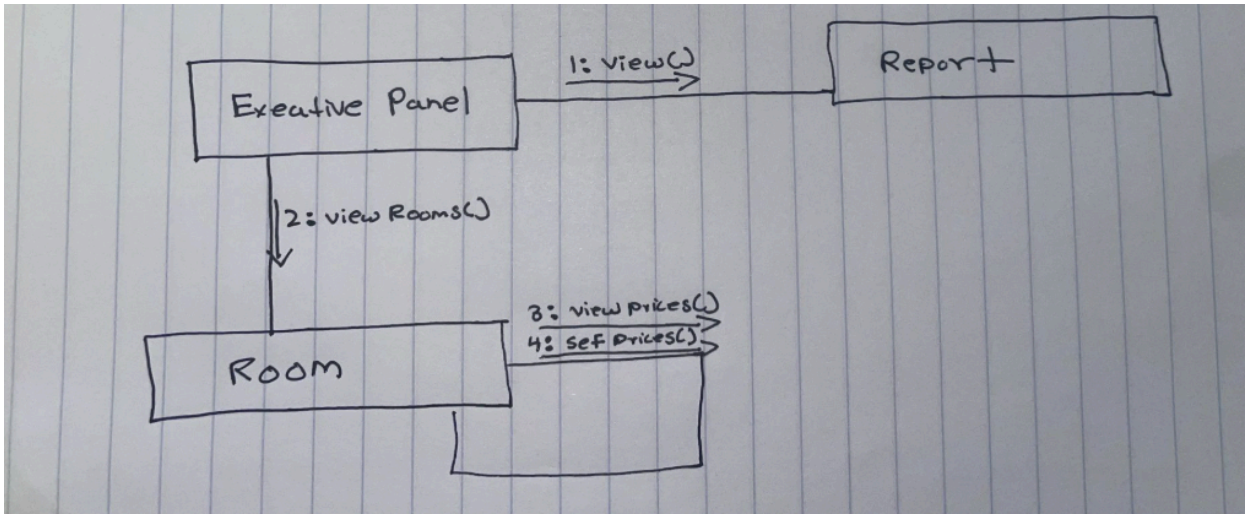
		and the Hotel
Clocking In & Out	Creator	When workers come in, they need to clock in and clock out so correct permissions are granted to the workers. This would involve creating check in and check out times
Calling Housekeeping	Creator	The housekeeper is called by the guest. This needs to create a cleaning ticket, for the staff to be able to respond to
Login for Workers	Controller	Handles the worker login process by validating credentials and directing the user to the correct dashboard.

Partial Communication Diagrams

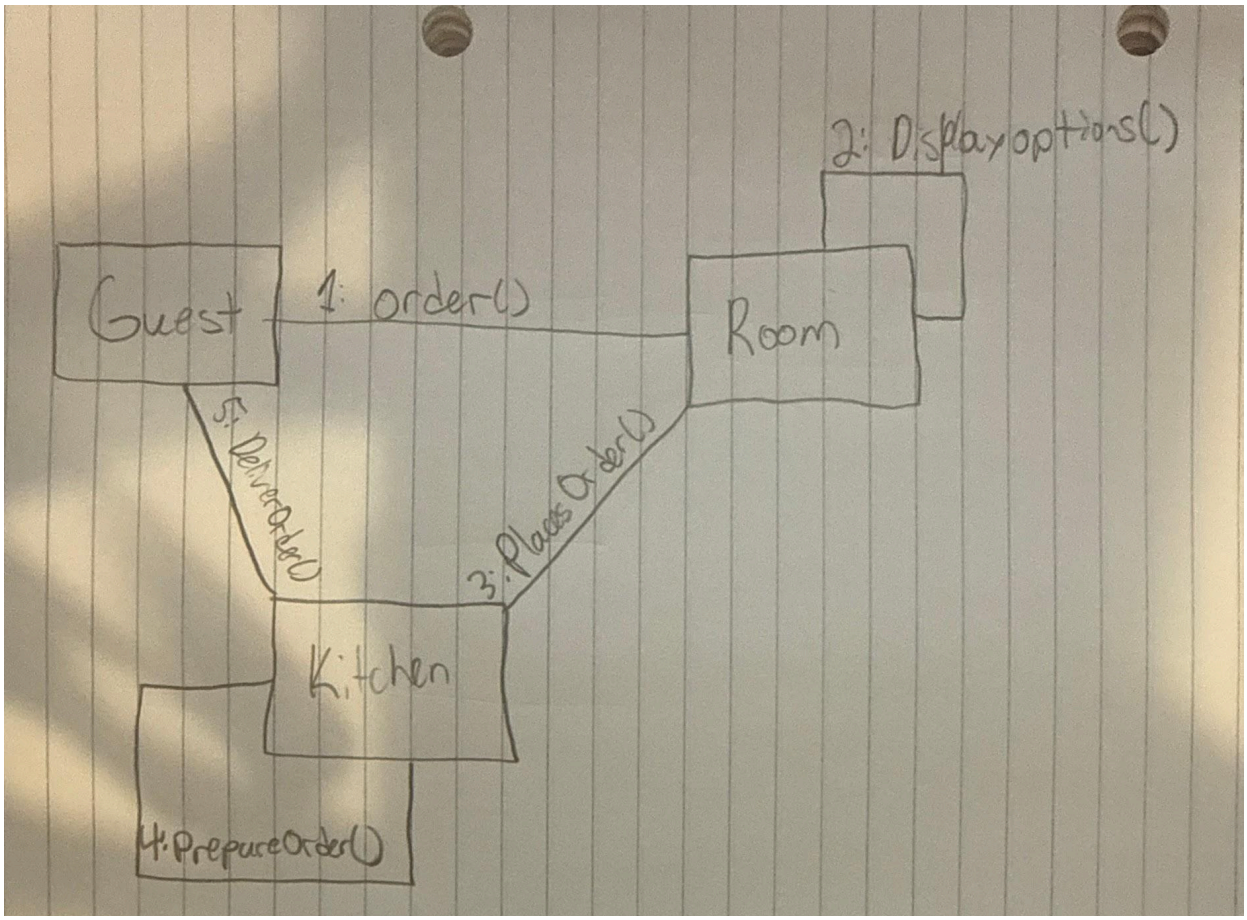
Communication Diagram for the Analyze Data Use case (Naveen)



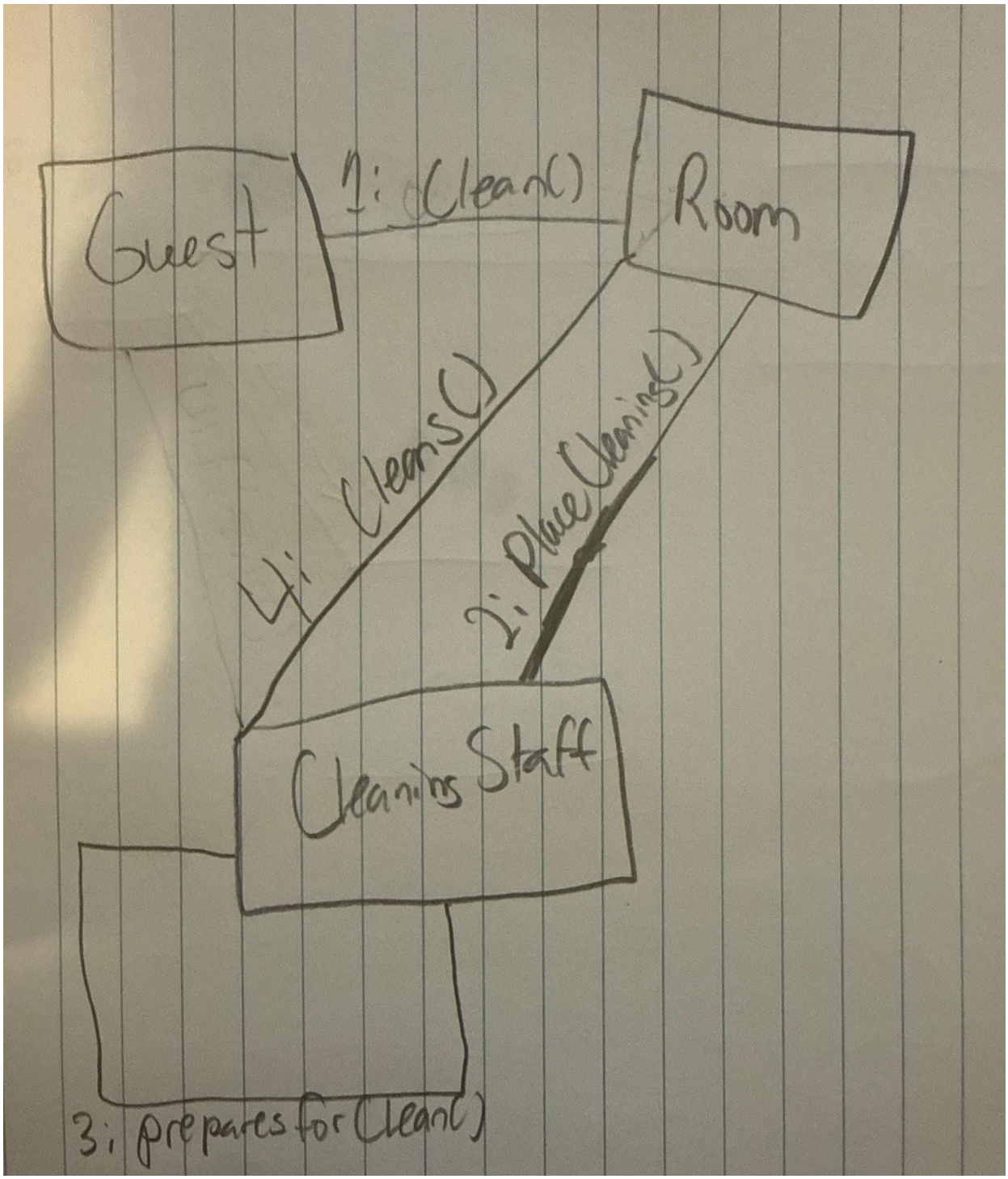
Communication Diagram for the Set price use case (Naveen)



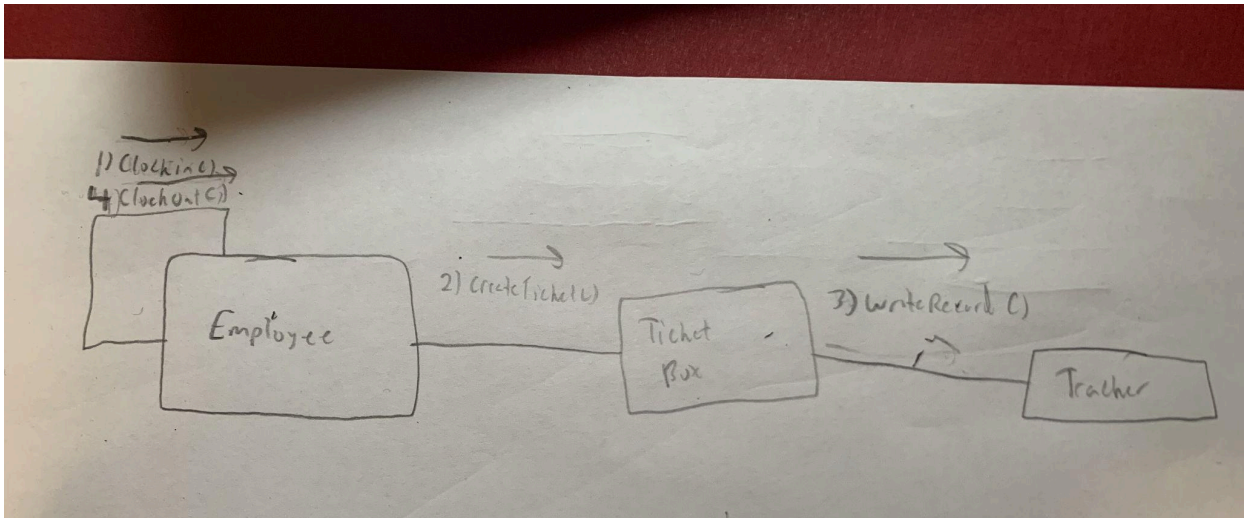
Communication diagram for calling room service (Brendan)



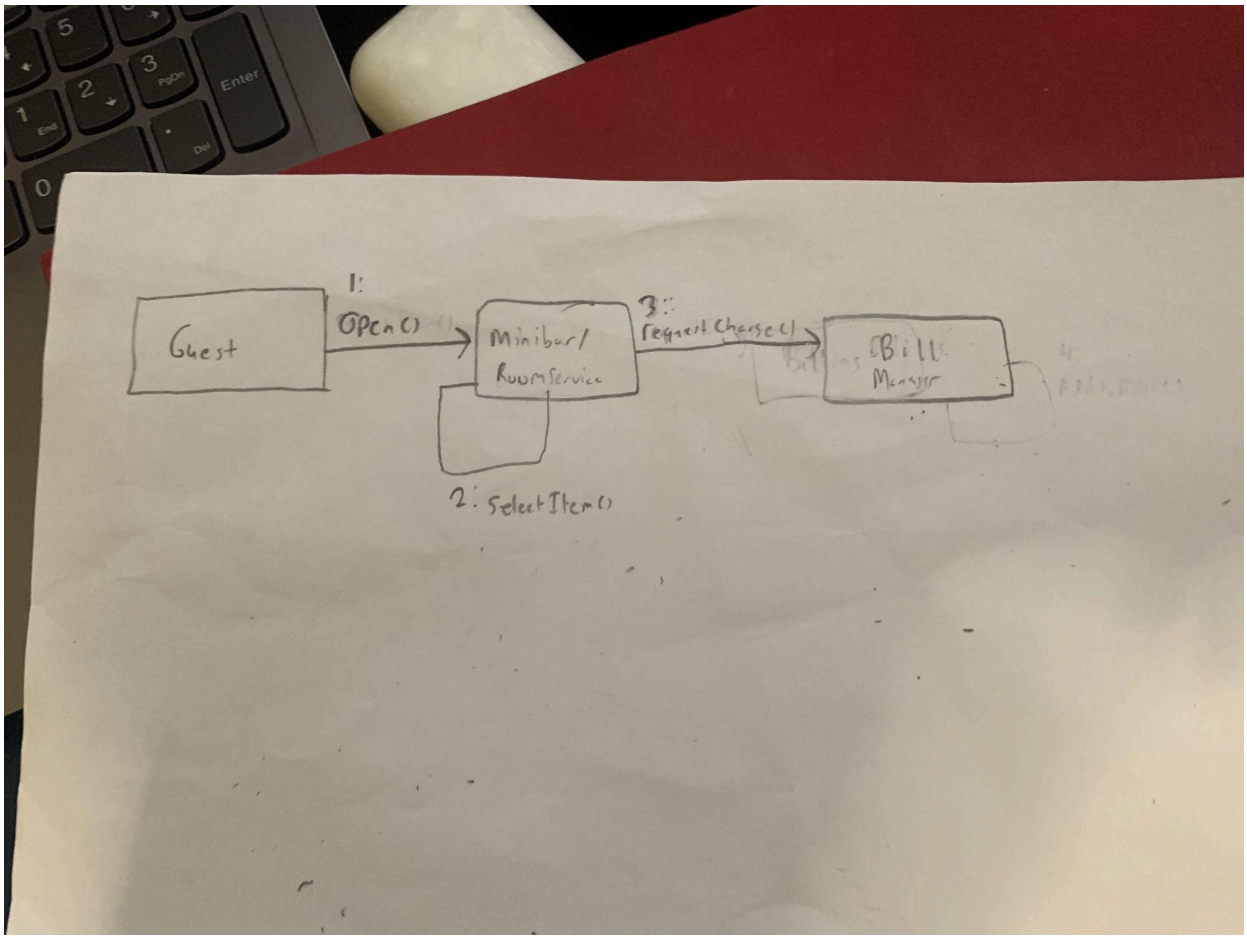
Communication diagram for calling Housekeeping (Brendan)



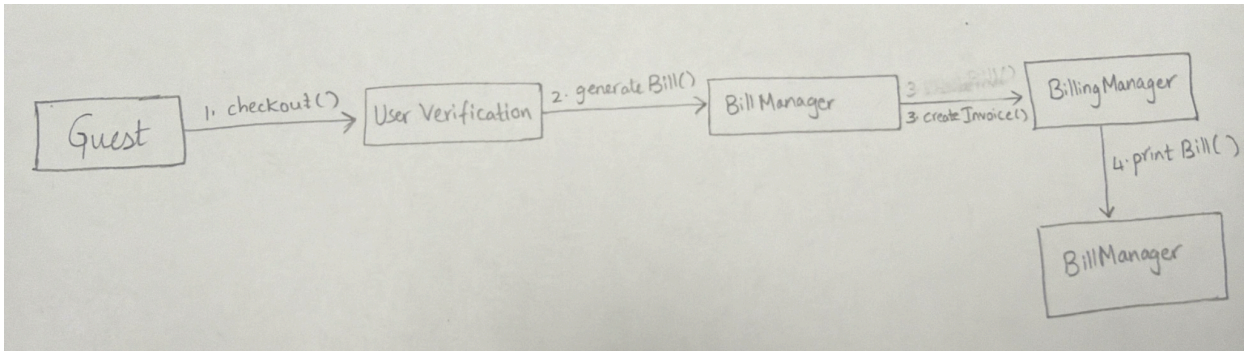
Communication diagram for clock in/clock out (Ozair)



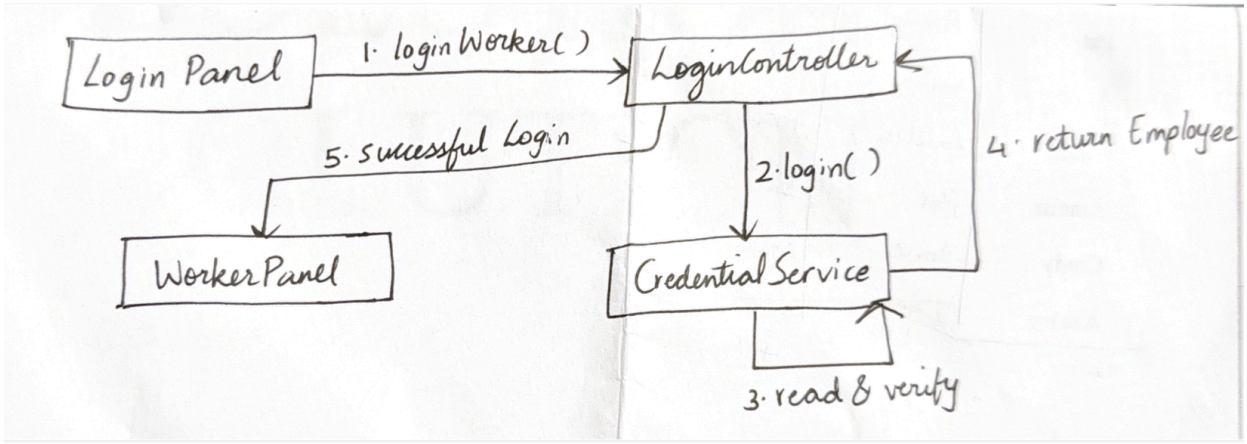
Communication diagram for MiniBar/ RoomService setup (Ozair)



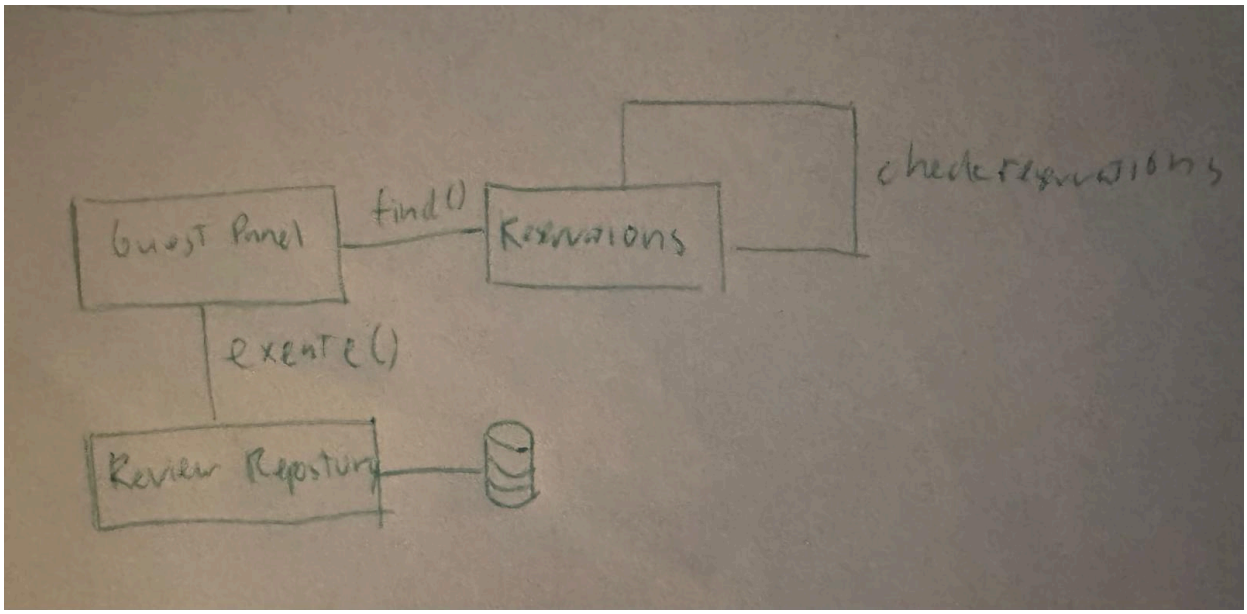
Communication Diagram for Billing upon Checkout (Pranava)



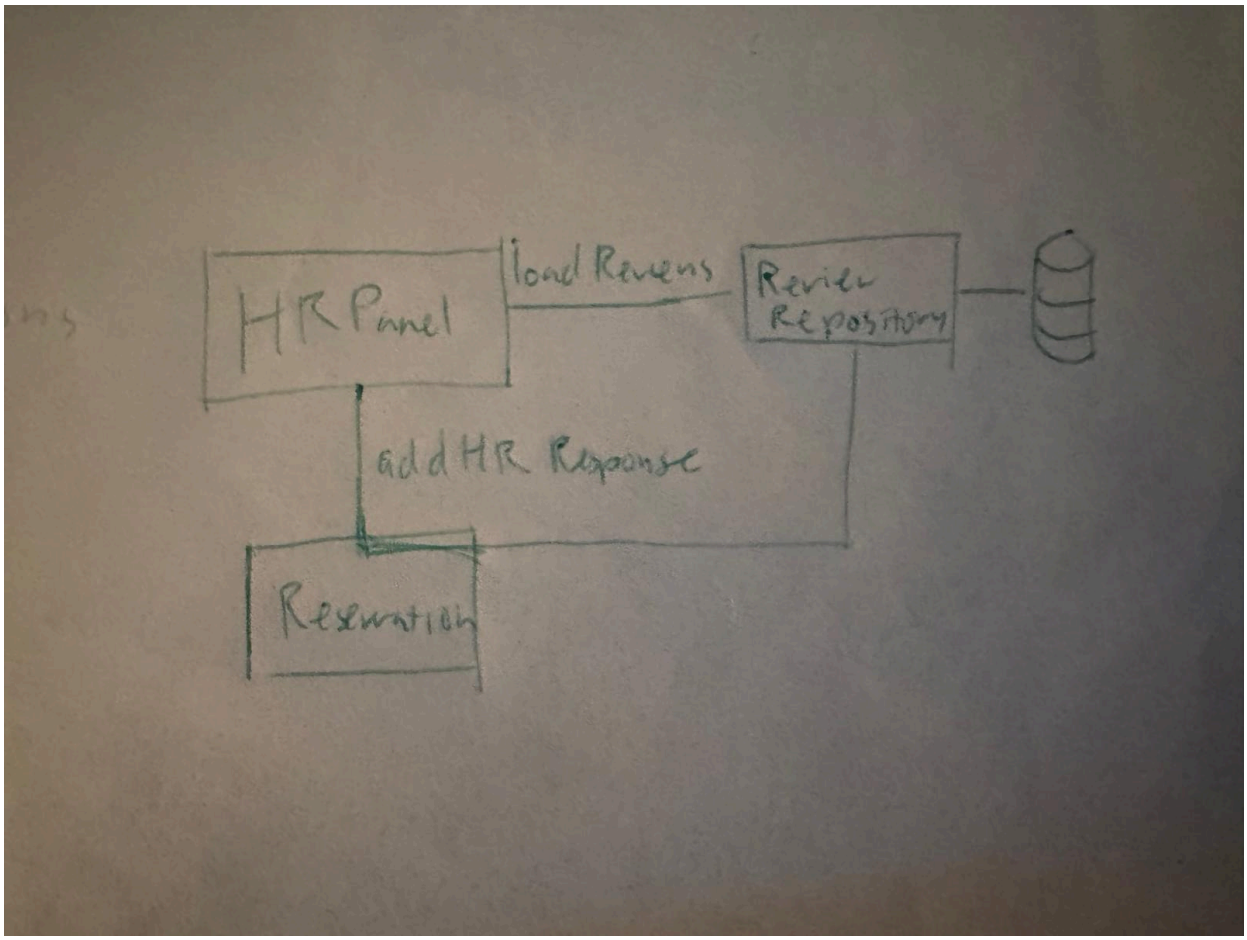
Communication Diagram for the Login for Workers use case (Pranava)



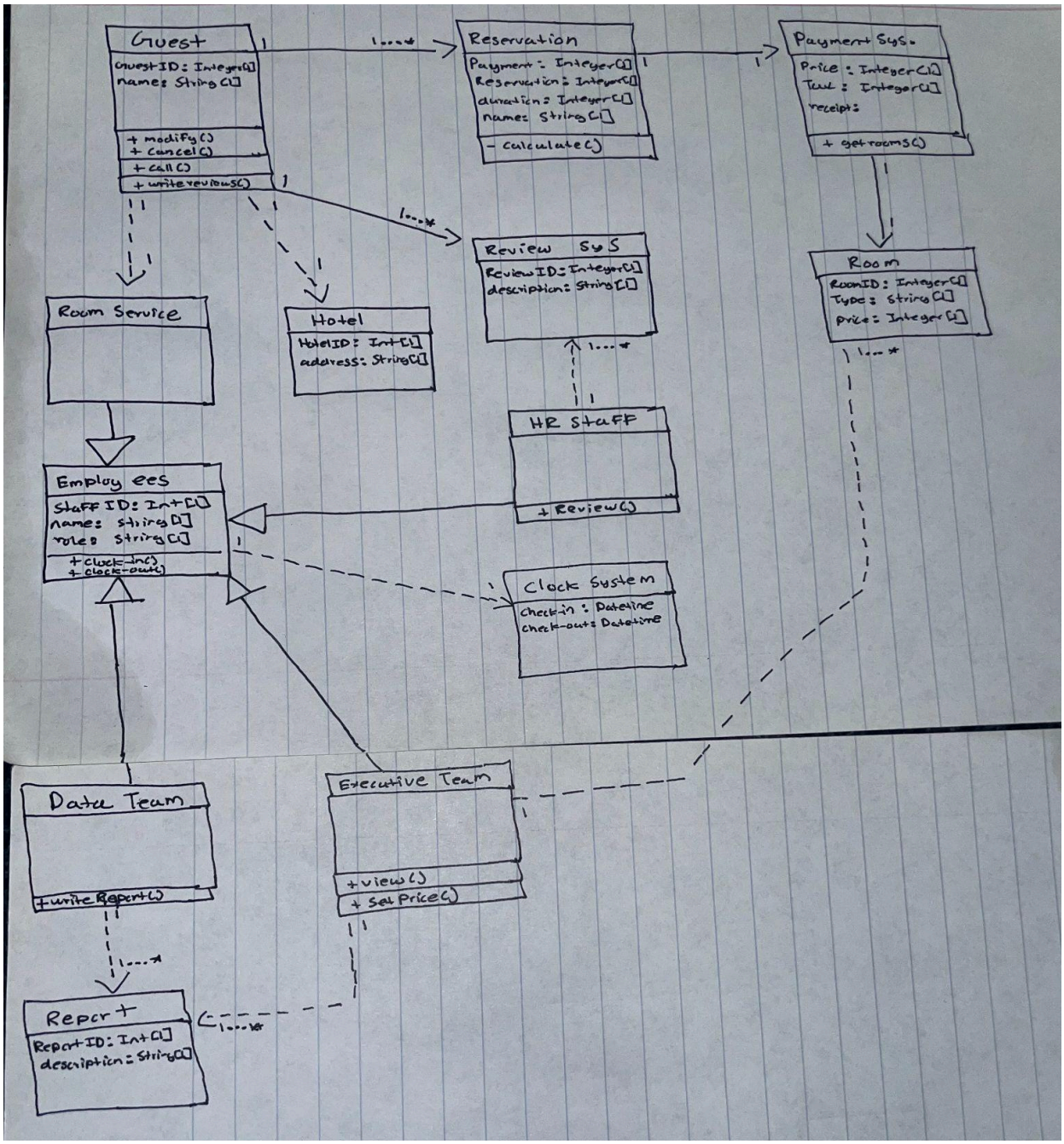
Communication diagram for Writing Reviews (Nicholas)



Communication diagram for Replying to Reviews (Nicholas)


HR Panel
load Reviews
Review Repository
add HR Response
Reservation

Partial Class Diagrams



Project Structure

Structure of the java files and packages:

```
|    Main.java
|
```

```
├───Booking
|       Booking.java
|       BookingInterface.java
|       Reservation.java
|       reviewInterface.java
|       ReviewProcess.java
|
├───Controller
|       CheckSystemController.java
|       DataTeamController.java
|       EmployeeLoginService.java
|       EmployeePortal.java
|       ExecutiveController.java
|       WorkerLoginController.java
|       HRController.java
|       PasswortUtils.java
|       WorkerLoginController.java
|
├───Data
|       DataAnalysis.java
|       DataRepository.java
|       FileDataRepository.java
|       ReportWriter.java
|       ReviewRepository.java
|
├───Employee
|       CleaningStaff.java
|       ClockSystem.java
|       DataPanel.java
|       DataTeam.java
|       Employee.java
|       ExecutivePanel.java
|       ExecutiveTeam.java
|       FrontDesk.java
|       frontdeskteam.java
|       Housekeeping.java
|       HousekeepingRequest.java
|       KitchenOrder.java
|       KitchenPanel.java
|       HRPanel.java
|       HRTeam.java
|       KitchenOrder.java
|       KitchenPanel.java
|
├───Guest
```

```
|       BillingManager.java
|       BillManager.java
|       CheckingProcess.java
|       DeskTask.java
|       Guest.java
|       GuestSession.java
|       KeyCard.java
|       KeyCardInterface.java
|       RoomAccessResult.java
|
└───Room
        Hotel.java
        room.java
        RoomInterface.java
```

Structure of the Database files

```
<address of hotel>
|
|   ClockData.txt
|   Employee.txt
|   Guest.txt
|   Keycard.txt
|   Past_Reservation.txt
|   Reservation.txt
|   Room.txt
|   RoomPrices.txt
|   review.txt
|
├───Bills
|       Room101_Bill.txt
|       Room104_Bill.txt
|       Room105_Bill.txt
|       Room106_Bill.txt
|
└───Reports
        Bills.txt.txt
        Guest_analysis.txt
        Guest_Report.txt
        Room_price_analysis.txt
```

Interface List

Interface for Analyze Use Case:

```
package Main.Employee;

public interface DataTeam {
    public void pullData();
    public void analyzeData();
    public void writeReport();
    public void notify_exec();
}
```

Interface for Set Price Use Case:

```
package Main.Employee;

public interface ExecutiveTeam {
    public void viewReports();
    public void viewPrices();
    public void setPrices();
}
```

Interface for Data Repository (to pull data):

```
package Main.Data;

import java.util.List;

public interface DataRepository {
    List<String[]> loadGuests();
    List<String[]> loadReservations();
    List<String[]> loadPastReservations();
    List<String[]> loadRooms();
    List<Employee> loadEmployees();
    boolean addEmployee(Employee emp);
    int getNextEmployeeId();

}
```

Interface for Data Analysis (to analyze the data):

```
package Main.Data;
```

```
public interface DataAnalysis {
    void createReport(String title, List<String> content);
    void editReport(String filename);
    void deleteReport(String filename);
}
```

```
Interface for Reply to Review use case
```

```
package Main.Employee;

public interface HRTeam {
   public void viewReviews();
   public void hireWorkers(JobApplication application);
   public void fireWorkers(Employee employee);
   public void massiveLayoffs();
   public void viewNotifications();
}
```

Interface for Write Review use case

```
package Main.Booking;

import java.util.Scanner;

public interface reviewInterface {
   public void execute(Scanner input);
}
```

Implementation

Analyze Data Use Case:

FileDataRepository Class (Information expert): (used by Pranava uses cases as well)

```
package Main.Data;

import Main.Booking.Reservation;
import Main.Guest.Guest;
import Main.Room.room;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
```

```
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

public class FileDataRepository implements DataRepository {

    private final Path basePath;

    public FileDataRepository(Path hotelPath) {
        this.basePath = hotelPath;
    }

    @Override
    public List<String[]> loadGuests() {
        try {
            return
loadFile(basePath.resolve("Guest.txt").toString());
        } catch (IOException e) {
            System.out.println("Error loading Guests: " +
e.getMessage());
            return new ArrayList<>();
        }
    }

    @Override
    public List<String[]> loadReservations() {
        try {
            return
loadFile(basePath.resolve("Reservation.txt").toString());
        } catch (IOException e) {
            System.out.println("Error loading Reservations: " +
e.getMessage());
            return new ArrayList<>();
        }
    }

    @Override
    public List<String[]> loadPastReservations() {
        try {
            return
loadFile(basePath.resolve("Past_Reservation.txt").toString());
        } catch (IOException e) {
            System.out.println("Error loading Past Reservations: " +
e.getMessage());
            return new ArrayList<>();
```

```
        }
    }

    @Override
    public List<String[]> loadRooms() {
        try {
            return loadFile(basePath.resolve("Room.txt").toString());
        } catch (IOException e) {
            System.out.println("Error loading Rooms: " +
e.getMessage());
            return new ArrayList<>();
        }
    }

    private List<String[]> loadFile(String filePath) throws
IOException {
        List<String[]> data = new ArrayList<>();
        List<String> lines = Files.readAllLines(Paths.get(filePath));

        for (String line : lines) {
            if (!line.trim().isEmpty()) {
                data.add(line.split(","));
            }
        }
        return data;
    }

    @Override
    public List<Employee> loadEmployees() {
        List<Employee> employees = new ArrayList<>();

        Path employeeFile = basePath.resolve("Employee.txt");

        try {
            List<String> lines = Files.readAllLines(employeeFile);

            for (String line : lines) {
                if (line.trim().isEmpty()) continue;

                employees.add(parseEmployee(line));
            }

        } catch (IOException e) {
            System.out.println("Error loading Employees: " +
e.getMessage());
```

```
        }

        return employees;
    }

    private Employee parseEmployee(String line) {
        String[] parts = line.trim().split(" ");

        int id = Integer.parseInt(parts[0]);

        // role, username, password are always last 3 tokens
        String role = parts[parts.length - 3];
        String username = parts[parts.length - 2];
        String password = parts[parts.length - 1];

        // name is everything between id and role
        StringBuilder nameBuilder = new StringBuilder();
        for (int i = 1; i < parts.length - 3; i++) {
            nameBuilder.append(parts[i]);
            if (i < parts.length - 4) nameBuilder.append(" ");
        }

        String name = nameBuilder.toString();

        return new Employee(id, name, role, username, password);
    }

    @Override
    public boolean addEmployee(Employee emp) {
        Path employeeFile = basePath.resolve("Employee.txt");

        try {
            String line = "\n" + emp.getId() + " "
                    + emp.getName() + " "
                    + emp.role() + " "
                    + emp.getUsername() + " "
                    + emp.getPassword();

            Files.write(employeeFile, line.getBytes(),
java.nio.file.StandardOpenOption.APPEND);

            return true;

        } catch (IOException e) {
```

```
            System.out.println("Error writing employee: " +
e.getMessage());
            return false;
        }
    }

    @Override
    public int getNextEmployeeId() {
        List<Employee> list = loadEmployees();
        if (list.isEmpty()) return 1;
        return list.get(list.size() - 1).getId() + 1;
    }
}
```

ReportWriter Class (Creator):

```
package Main.Data;

import Main.Main;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;
import java.util.Scanner;

public class ReportWriter implements DataAnalysis {
    @Override
    public void createReport(String title, List<String> content) {
        try {
            File file = new File(title + ".txt");
            try (PrintWriter writer = new PrintWriter(file)) {


                writer.println("====== Data Report ======");
                writer.println("Report Title: " + title);
                writer.println("================================\n");


                for (String line : content) {
                    writer.println(line);
                }
```

```
                System.out.println("Report saved to: " +
file.getAbsolutePath());
            }

        } catch (IOException e) {
            System.out.println("Error writing report: " +
e.getMessage());
        }
    }

    @Override
    public void editReport(String filename) {
        Scanner sc = new Scanner(System.in);
        File file = new File(filename);

        if (!file.exists()) {
            System.out.println("File does not exist.");
            return;
        }

        try {
            List<String> lines = Files.readAllLines(file.toPath());

            while (true) {
                System.out.println("\n===== Current Report =====");
                for (int i = 0; i < lines.size(); i++) {
                    System.out.println((i + 1) + ": " +
lines.get(i));
                }
                System.out.println("==========================");

                System.out.println("\nEdit Options:");
                System.out.println("1. Edit a Line");
                System.out.println("2. Delete a Line");
                System.out.println("3. Add a New Line");
                System.out.println("4. Save & Exit");
                System.out.print("Choose: ");

                int choice = sc.nextInt();
                sc.nextLine();

                if (choice == 1) {
                    System.out.print("Enter line number to edit: ");
                    int lineNum = sc.nextInt();
                    sc.nextLine();
```

```
                    if (lineNum < 1 || lineNum > lines.size()) {
                        System.out.println("Invalid line.");
                        continue;
                    }

                    System.out.println("Current: " +
lines.get(lineNum - 1));
                    System.out.print("New text: ");
                    lines.set(lineNum - 1, sc.nextLine());

                } else if (choice == 2) {
                    System.out.print("Enter line number to delete:
");
                    int lineNum = sc.nextInt();
                    sc.nextLine();

                    if (lineNum < 1 || lineNum > lines.size()) {
                        System.out.println("Invalid line.");
                        continue;
                    }

                    lines.remove(lineNum - 1);
                    System.out.println("Line removed.");

                } else if (choice == 3) {
                    System.out.print("Enter new line to append: ");
                    lines.add(sc.nextLine());

                } else if (choice == 4) {
                    Files.write(file.toPath(), lines);
                    System.out.println("Report updated
successfully!");
                    break;

                } else {
                    System.out.println("Invalid choice.");
                }
            }

        } catch (IOException e) {
            System.out.println("Error editing report: " +
e.getMessage());
        }
    }
```

```
    @Override
    public void deleteReport(String filename) {
        Scanner sc = new Scanner(System.in);
        File file = new File(filename);

        if (!file.exists()) {
            System.out.println("File does not exist.");
            return;
        }

        System.out.print("Are you sure you want to delete '" +
filename + "'? (y/n): ");
        String confirm = sc.nextLine().trim().toLowerCase();

        if (confirm.equals("y")) {
            if (file.delete()) {
                System.out.println("Report deleted successfully: " +
filename);
            } else {
                System.out.println("Failed to delete report.");
            }
        } else {
            System.out.println("Deletion cancelled.");
        }
    }
}
```

DataTeamController Class (Controller):

```
package Main.Controller;

import Main.Employee.DataPanel;

import java.util.Scanner;

public class DataTeamController {
    public static void runDataAnalysis(String analystName) {
        Scanner sc = new Scanner(System.in);
        DataPanel analyst = new DataPanel(2001, analystName, "Data
Team");

        System.out.println("\n===== DATA TEAM PANEL =====");
        System.out.println("Welcome, " + analystName + "!");
```

```
        System.out.println("----------------------------");

        boolean running = true;

        while (running) {
            System.out.println("\n--- Data Team Options ---");
            System.out.println("1. Pull Data");
            System.out.println("2. Analyze Data");
            System.out.println("3. Write Report");
            System.out.println("4. View Tickets");
            System.out.println("5. Mark Ticket as Completed");
            System.out.println("6. Delete Reports");
            System.out.println("7. Exit Data Panel");
            System.out.print("Enter your choice: ");

            String choice = sc.nextLine().trim();

            if (choice.equals("1")) {
                analyst.pullData();
            } else if (choice.equals("2")) {
                analyst.analyzeData();
            } else if (choice.equals("3")) {
                analyst.writeReport();
            } else if (choice.equals("4")) {
                DataPanel.viewRequests();
            } else if (choice.equals("5")) {
                DataPanel.viewRequests();
                System.out.print("Enter ticket number to mark
complete: ");
                int index = Integer.parseInt(sc.nextLine());
                DataPanel.completeRequest(index);
            } else if (choice.equals("7")) {
                running = false;
            } else if (choice.equals("6")){
                analyst.deleteReport();
            }
            else {
                System.out.println("Invalid choice. Please enter
1-6.");
            }
        }
    }
}
```
———

DataPanel Class (Information Expert):

```
package Main.Employee;

import Main.Data.DataRepository;
import Main.Main;
import java.io.*;
import java.nio.file.*;
import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;
import Main.Data.FileDataRepository;
import Main.Data.ReportWriter;


public class DataPanel extends Employee implements DataTeam {
    private static final String REPORT_FOLDER = Main.HOTEL_PATH +
"/Reports/";
    private final DataRepository repo;
    private final ReportWriter reportWriter = new ReportWriter();


    private List<String[]> guests = new ArrayList<>();
    private List<String[]> reservations = new ArrayList<>();
    private List<String[]> pastReservations = new ArrayList<>();
    private List<String[]> rooms = new ArrayList<>();
    private Map<String, Integer> guestVisitCount = new HashMap<>();

    private static final LinkedBlockingQueue<String> requestQueue =
new LinkedBlockingQueue<>();

    public DataPanel(int staffID, String name, String role) {

        super(staffID, name, role);
        this.repo = new
FileDataRepository(Paths.get(Main.HOTEL_PATH));
    }

    public static void addRequest(String request) {
        try {
            requestQueue.put(request);
            System.out.println("New report request added: " +
request);
        } catch (InterruptedException e) {
            System.out.println(" Failed to add request: " +
e.getMessage());
```

```
        }
    }

    public static void viewRequests() {
        if (requestQueue.isEmpty()) {
            System.out.println("No pending report requests.");
            return;
        }

        System.out.println("\n===== PENDING REPORT REQUESTS =====");
        int i = 1;
        for (String req : requestQueue) {
            System.out.println(i++ + ". " + req);
        }
    }

    public static void completeRequest(int index) {
        Scanner sc = new Scanner(System.in);

        if (index <= 0 || index > requestQueue.size()) {
            System.out.println("Invalid request number.");
            return;
        }

        List<String> list = new ArrayList<>(requestQueue);
        String done = list.get(index - 1);

        requestQueue.remove(done);
        System.out.println("Request completed: " + done);

        System.out.print("Enter a closing note to send to executives:
");
        String note = sc.nextLine().trim();

        if (note.isEmpty()) {
            note = "(No additional notes provided.)";
        }

        ExecutivePanel.notifications.add("Request Completed: " + done
+ " | Note: " + note);

        System.out.println("Closing note sent to executives.");
    }
```

```
    @Override
    public void pullData() {
        guests = repo.loadGuests();
        reservations = repo.loadReservations();
        pastReservations = repo.loadPastReservations();
        rooms = repo.loadRooms();

        System.out.println("Data pulled successfully from all
sources.");
    }

    @Override
    public void analyzeData() {
        if (guests.isEmpty() || (reservations.isEmpty() &&
pastReservations.isEmpty())) {
            System.out.println("Please pull data first before
analysis.");
            return;
        }

        Scanner sc = new Scanner(System.in);
        System.out.println("\n--- Choose Data Analysis Type ---");
        System.out.println("1. Guest Data Analysis");
        System.out.println("2. Reservation Data Analysis");
        System.out.println("3. Past Reservation Analysis");
        System.out.print("Enter choice (1-3): ");

        int choice = sc.nextInt();
        sc.nextLine();

        if (choice == 1) {
            analyzeGuestData();
        } else if (choice == 2) {
            analyzeReservationData();
        } else if (choice == 3) {
            analyzePastReservationData();
        } else {
            System.out.println("Invalid choice. Please enter 1-3.");
        }
    }

    @Override
    public void writeReport() {
        Scanner sc = new Scanner(System.in);
```

```
        System.out.println("\n--- Report Options ---");
        System.out.println("1. Create New Report");
        System.out.println("2. Edit Existing Report");
        System.out.print("Choose (1-2): ");
        int choice = sc.nextInt();
        sc.nextLine();

        if (choice == 2) {
            editExistingReport();
            return;
        }

        System.out.print("Enter report file name (without extension):
");
        String filename = sc.nextLine().trim();

        System.out.println("\n--- Write Your Report Below ---");
        System.out.println("(Blank line finishes writing)");
        System.out.println("--------------------------------");

        List<String> content = new ArrayList<>();

        while (true) {
            String line = sc.nextLine();
            if (line.trim().isEmpty()) break;
            content.add(line);
        }

        reportWriter.createReport(REPORT_FOLDER + filename, content);
    }


    private void editExistingReport() {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the report filename (including .txt):
");
        String filename = sc.nextLine().trim();

        reportWriter.editReport(REPORT_FOLDER + filename);
    }

    private void analyzeGuestData() {
        System.out.println("\n--- Guest Data Analysis ---");
        System.out.println("Total Guests: " + guests.size());
```

```
        Set<String> uniqueGuests = new HashSet<>();
        int duplicates = 0;

        for (String[] g : guests) {
            String guestName = g[0].trim();
            if (!uniqueGuests.add(guestName)) {
                duplicates++;
            }
        }

        System.out.println("Unique Guests: " + uniqueGuests.size());
        System.out.println("Duplicate Entries Found: " + duplicates);
    }

    private void analyzeReservationData() {
        System.out.println("\n--- Reservation Data Analysis ---");
        System.out.println("Active Reservations: " +
reservations.size());

        Map<String, Integer> roomBookings = new HashMap<>();
        for (String[] record : reservations) {
            if (record.length >= 3) {
                String roomNumber = record[2].trim();
                roomBookings.put(roomNumber,
roomBookings.getOrDefault(roomNumber, 0) + 1);
            }
        }

        System.out.println("Bookings per Room:");
        for (Map.Entry<String, Integer> entry :
roomBookings.entrySet()) {
            System.out.println("Room " + entry.getKey() + ": " +
entry.getValue() + " active reservations");
        }

        if (!rooms.isEmpty()) {
            Map<String, Integer> typeCount = new HashMap<>();
            for (String[] r : rooms) {
                if (r.length >= 2) {
                    String type = r[1].trim();
                    typeCount.put(type, typeCount.getOrDefault(type,
0) + 1);
                }
            }
```

```
            System.out.println("\nRoom Types Available:");
            for (Map.Entry<String, Integer> entry :
typeCount.entrySet()) {
                System.out.println(entry.getKey() + ": " +
entry.getValue() + " rooms");
            }
        }
    }

    private void analyzePastReservationData() {
        System.out.println("\n--- Past Reservation Data Analysis
---");
        System.out.println("Total Past Reservations: " +
pastReservations.size());

        guestVisitCount.clear();
        for (String[] record : pastReservations) {
            if (record.length >= 2) {
                String guestName = record[1].trim();
                guestVisitCount.put(guestName,
guestVisitCount.getOrDefault(guestName, 0) + 1);
            }
        }

        System.out.println("Guest Visit Counts:");
        for (Map.Entry<String, Integer> entry :
guestVisitCount.entrySet()) {
            System.out.println(entry.getKey() + " - " +
entry.getValue() + " visits");
        }
    }

    public void deleteReport() {
        System.out.println("\n===== DELETE REPORT =====");

        File folder = new File(REPORT_FOLDER);
        if (!folder.exists() || !folder.isDirectory()) {
            System.out.println("No reports folder found.");
            return;
        }

        File[] files = folder.listFiles();
        if (files == null || files.length == 0) {
            System.out.println("No reports available to delete.");
```

```
            return;
        }

        System.out.println("\nAvailable Reports:");
        for (int i = 0; i < files.length; i++) {
            System.out.println((i + 1) + ". " + files[i].getName());
        }

        Scanner sc = new Scanner(System.in);
        System.out.print("\nEnter the number of the report to DELETE:
");
        int choice = sc.nextInt();
        sc.nextLine();

        if (choice < 1 || choice > files.length) {
            System.out.println("Invalid choice.");
            return;
        }

        String filename = files[choice - 1].getName();
        reportWriter.deleteReport(REPORT_FOLDER + filename);
    }

}
```

Set Prices use case:

Executive Panel (Information Expert, Creator)

```
package Main.Employee;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.LinkedBlockingQueue;
import Main.*;

import Main.Room.*;

public class ExecutivePanel extends Employee implements ExecutiveTeam
{
```

```
    public static final List<String> notifications = new
ArrayList<>();

    public ExecutivePanel(int StaffID, String name, String role) {
        super(StaffID, name, role);
    }

    @Override
    public void viewReports() {
        System.out.println("\n===== EXECUTIVE REPORT VIEWER =====");

        File folder = new File(Main.HOTEL_PATH + "/Reports/");
        if (!folder.exists() || !folder.isDirectory()) {
            System.out.println("No reports folder found.");
            return;
        }

        File[] files = folder.listFiles();
        if (files == null || files.length == 0) {
            System.out.println("No reports available.");
            return;
        }

        System.out.println("\nAvailable Reports:");
        for (int i = 0; i < files.length; i++) {
            System.out.println((i + 1) + ". " + files[i].getName());
        }

        System.out.print("\nEnter report number to view: ");
        Scanner sc = new Scanner(System.in);
        int choice = sc.nextInt();
        sc.nextLine();

        if (choice < 1 || choice > files.length) {
            System.out.println("Invalid choice.");
            return;
        }

        File selected = files[choice - 1];
        System.out.println("\n--- Viewing: " + selected.getName() + "
---");

System.out.println("-----------------------------------------");

        try (Scanner reader = new Scanner(selected)) {
```

```
            while (reader.hasNextLine()) {
                System.out.println(reader.nextLine());
            }
        } catch (FileNotFoundException e) {
            System.out.println("Error: " + e.getMessage());
        }


System.out.println("-----------------------------------------");
        System.out.println("End of Report.");
    }


    public void askDataTeam() {
        Scanner sc = new Scanner(System.in);

        System.out.println("\n===== REQUEST DATA TEAM REPORT =====");
        System.out.print("Enter what you need a report on: ");
        String request = sc.nextLine().trim();

        if (request.isEmpty()) {
            System.out.println("Invalid request. Try again.");
            return;
        }
        DataPanel.addRequest("Executive " + getName() + " requested:
" + request);
    }

    @Override
    public void viewPrices() {
        Hotel hotel = new Hotel(Main.HOTEL_PATH);
        hotel.loadPricesFromFile();
        hotel.viewPrices();
    }

    @Override
    public void setPrices() {
        Hotel hotel = new Hotel(Main.HOTEL_PATH);
        hotel.loadPricesFromFile();
        hotel.setPrices();
        hotel.savePricesToFile();
    }

    public void viewNotifications() {
        Scanner sc = new Scanner(System.in);
```

```
        while (true) {
            System.out.println("\n===== EXECUTIVE NOTIFICATIONS
=====");

            if (notifications.isEmpty()) {
                System.out.println("No new notifications.");
            } else {
                for (int i = 0; i < notifications.size(); i++) {
                    System.out.println((i + 1) + ". " +
notifications.get(i));
                }
            }


System.out.println("==================================");
            System.out.println("Options:");
            System.out.println("1. Clear ALL notifications");
            System.out.println("2. Clear a specific notification");
            System.out.println("3. Exit");
            System.out.print("Choose: ");

            int choice = sc.nextInt();
            sc.nextLine();

            if (choice == 1) {
                notifications.clear();
                System.out.println("All notifications cleared.");

            } else if (choice == 2) {

                if (notifications.isEmpty()) {
                    System.out.println("There are no notifications to
clear.");
                    continue;
                }

                System.out.print("Enter notification number to clear:
");
                int index = sc.nextInt();
                sc.nextLine();

                if (index < 1 || index > notifications.size()) {
                    System.out.println("Invalid number.");
                    continue;
```

```
                }

                notifications.remove(index - 1);
                System.out.println("Notification removed.");

            } else if (choice == 3) {
                System.out.println("Exiting Notification Panel.");
                break;

            } else {
                System.out.println("Invalid option.");
            }
        }
    }
}
```

ExecutiveController (Controller)

```
package Main.Controller;

import Main.Employee.ExecutivePanel;

import java.util.Scanner;

public class ExecutiveController {
    public static void runExecutivePanel(String execName) {
        Scanner sc = new Scanner(System.in);
        ExecutivePanel executive = new ExecutivePanel(3001, execName,
"Executive");

        System.out.println("\n===== EXECUTIVE PANEL =====");
        System.out.println("Welcome, " + execName + "!");
        System.out.println("--------------------------");

        boolean running = true;

        while (running) {
            System.out.println("\n--- Executive Options ---");
            System.out.println("1. View Reports");
            System.out.println("2. Ask Data Team for New Report");
            System.out.println("3. View Current Room Prices");
            System.out.println("4. Set / Update Room Prices");
            System.out.println("5. Exit Executive Panel");
            System.out.println("6. Notifications");
```

```
            System.out.print("Enter your choice: ");

            String choice = sc.nextLine().trim();

            if (choice.equals("1")) {
                executive.viewReports();
            } else if (choice.equals("2")) {
                executive.askDataTeam();
            } else if (choice.equals("3")) {
                executive.viewPrices();
            } else if (choice.equals("4")) {
                executive.setPrices();
            } else if (choice.equals("5")) {
                System.out.println("Logging out of Executive
Panel...");
                running = false;
            }else if(choice.equals("6")){
                executive.viewNotifications();
            } else {
                System.out.println("Invalid choice. Please enter
1-5.");
            }
        }
    }
}
```

Order Room Service Use Case

BillingManager.java (Pranava and Brendan both used for uses cases)

```
package Main.Guest;

import Main.Booking.Reservation;
import Main.Main;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
```

```
public class BillingManager {

    private static final String ROOM_FILE = Main.HOTEL_PATH + "/Room.txt";
    private static final String PRICE_FILE = Main.HOTEL_PATH + "/RoomPrices.txt";
    private static final double SECURITY_DEPOSIT = 50.0;

    /**
     * Called at check-in.
     * Creates bill file, then writes:
     * - security deposit
     * - daily room charge for each night of stay
     */
    public static void initBillingForStay(Reservation reservation, String guestName, String
location) {
        int roomNumber = reservation.getRoomNumber();

        // Fetch room type
        String roomType = getRoomType(roomNumber);

        // Fetch daily rate
        double dailyRate = getRoomRate(roomType);

        // Prepare line entries
        List<String> entries = generateInitialCharges(reservation, dailyRate);

        // Write to bill file
        appendCharges(roomNumber, guestName, entries);
    }

    private static String getRoomType(int roomNumber) {
        try (Scanner sc = new Scanner(new File(ROOM_FILE))) {

            while (sc.hasNextLine()) {
                String line = sc.nextLine().trim();

                if (line.startsWith("roomNumber")) continue;

                String[] parts = line.split(",");

                int num = Integer.parseInt(parts[0]);
                String type = parts[1];
```

```
            if (num == roomNumber) {
                return type;
            }
        }

    } catch (Exception e) {
        System.out.println("Error reading Room.txt: " + e.getMessage());
    }
    return null;
}

private static double getRoomRate(String type) {
    try (Scanner sc = new Scanner(new File(PRICE_FILE))) {

        while (sc.hasNextLine()) {
            String line = sc.nextLine().trim();
            if (line.startsWith("Hotel") || line.startsWith("=====")) continue;
            String[] parts = line.split(",");

            String roomType = parts[0];
            double price = Double.parseDouble(parts[1]);

            if (roomType.equalsIgnoreCase(type)) {
                return price;
            }
        }

    } catch (Exception e) {
        System.out.println("Error reading RoomPrices.txt: " + e.getMessage());
    }
    return 0.0;
}

private static List<String> generateInitialCharges(Reservation r, double dailyRate) {
    List<String> lines = new ArrayList<>();

    // Security Deposit
    lines.add("SECURITY_DEPOSIT (RECEIVED AT CHECK-IN)," + SECURITY_DEPOSIT);
    lines.add("SECURITY_DEPOSIT (REFUNDED AT CHECK-OUT)," +
SECURITY_DEPOSIT);

    // Daily room charges
```

```
        LocalDate date = r.getStartDate();
        LocalDate end = r.getEndDate();

        while (date.isBefore(end)) {
            lines.add("ROOM_CHARGE," + date + "," + dailyRate);
            date = date.plusDays(1);
        }

        return lines;
    }

    private static String billFileForRoom(int roomNumber) {
        return Main.HOTEL_PATH + "/Bills/Room" + roomNumber + "_Bill.txt";
    }

    public static void appendCharges(int roomNumber, String guestName, List<String>
entries) {
        try (PrintWriter out = new PrintWriter(new FileWriter(billFileForRoom(roomNumber),
true))) {
            for (String line : entries) {
                out.println(line);
            }
        } catch (IOException e) {
            System.out.println("Error writing bill: " + e.getMessage());
        }
    }
}
```

KitchenOrder.java-

```
package Main.Employee;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.StringJoiner;
import java.util.UUID;

public class KitchenOrder {
    private static final DateTimeFormatter TS_FORMAT =
DateTimeFormatter.ofPattern("MMM dd HH:mm");
```

```
private final String id;
private final int roomNumber;
private final String guestName;
private final String hotelPath;
private final LinkedHashMap<String, Integer> items;
private final double totalAmount;
private final LocalDateTime createdAt;

public KitchenOrder(String hotelPath,
                    int roomNumber,
                    String guestName,
                    Map<String, Integer> orderedItems,
                    double totalAmount) {
   this.id = UUID.randomUUID().toString();
   this.hotelPath = hotelPath;
   this.roomNumber = roomNumber;
   this.guestName = guestName;
   this.items = new LinkedHashMap<>(orderedItems);
   this.totalAmount = totalAmount;
   this.createdAt = LocalDateTime.now();
}

public String getId() {
   return id;
}

public int getRoomNumber() {
   return roomNumber;
}

public String getGuestName() {
   return guestName;
}

public String getHotelPath() {
   return hotelPath;
}

public Map<String, Integer> getItems() {
   return new LinkedHashMap<>(items);
}
```

```
    public double getTotalAmount() {
        return totalAmount;
    }

    public LocalDateTime getCreatedAt() {
        return createdAt;
    }

    public boolean belongsToHotel(String hotelPath) {
        return this.hotelPath != null && this.hotelPath.equalsIgnoreCase(hotelPath);
    }

    public String formatLineItem() {
        StringJoiner joiner = new StringJoiner(", ");
        for (Map.Entry<String, Integer> entry : items.entrySet()) {
            joiner.add(entry.getKey() + " x" + entry.getValue());
        }
        return "Room " + roomNumber + " | " + guestName + " | " + joiner +
                String.format(" | Total $%.2f | %s", totalAmount,
createdAt.format(TS_FORMAT));
    }
}
```

KitchenPanel.java-

```
package Main.Employee;

import Main.Main;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.concurrent.LinkedBlockingQueue;

public class KitchenPanel extends Employee {

    private static final LinkedBlockingQueue<KitchenOrder> kitchenQueue = new
LinkedBlockingQueue<>();

    public KitchenPanel(int staffID, String name) {
        super(staffID, name, "Kitchen");
    }
```

```
    public static void submitOrder(KitchenOrder order) {
        if (order == null) {
            return;
        }
        kitchenQueue.add(order);
        System.out.println("🍽 Room service order queued for Room " +
order.getRoomNumber());
        ExecutivePanel.notifications.add("Room " + order.getRoomNumber() + " placed a
room service order.");
    }

    public static void processOrders(String staffName) {
        if (Main.HOTEL_PATH == null || Main.HOTEL_PATH.isEmpty()) {
            System.out.println("Hotel location not set. Please log in through the worker portal
first.");
            return;
        }

        Scanner sc = new Scanner(System.in);
        boolean running = true;

        while (running) {
            System.out.println("\n===== KITCHEN PANEL =====");
            System.out.println("Logged in as " + staffName + " at " + Main.HOTEL_PATH);
            System.out.println("1. View pending orders");
            System.out.println("2. Complete an order");
            System.out.println("3. Exit Kitchen Panel");
            System.out.print("Choose an option: ");

            String choice = sc.nextLine().trim();

            switch (choice) {
                case "1":
                    viewOrders();
                    break;
                case "2":
                    completeOrder(sc, staffName);
                    break;
                case "3":
                    running = false;
                    break;
                default:
```

```
                System.out.println("Invalid option. Choose 1-3.");
            }
        }
    }

    private static void viewOrders() {
        List<KitchenOrder> orders = collectOrdersForHotel();
        if (orders.isEmpty()) {
            System.out.println("No pending kitchen orders for this hotel.");
            return;
        }

        System.out.println("\n--- Pending Kitchen Orders ---");
        for (int i = 0; i < orders.size(); i++) {
            System.out.println((i + 1) + ". " + orders.get(i).formatLineItem());
        }
    }

    private static void completeOrder(Scanner sc, String staffName) {
        List<KitchenOrder> orders = collectOrdersForHotel();
        if (orders.isEmpty()) {
            System.out.println("No pending orders.");
            return;
        }

        System.out.println("Select the order to complete: ");
        for (int i = 0; i < orders.size(); i++) {
            System.out.println((i + 1) + ". " + orders.get(i).formatLineItem());
        }
        System.out.print("Enter number (or 0 to cancel): ");

        String input = sc.nextLine().trim();
        int idx;
        try {
            idx = Integer.parseInt(input);
        } catch (NumberFormatException e) {
            System.out.println("Invalid number.");
            return;
        }

        if (idx == 0) {
            return;
```

```
        }

        if (idx < 1 || idx > orders.size()) {
            System.out.println("Selection out of range.");
            return;
        }

        KitchenOrder selected = orders.get(idx - 1);
        kitchenQueue.remove(selected);
        System.out.println("Order for Room " + selected.getRoomNumber() + " completed by
" + staffName + ".");
    }

    private static List<KitchenOrder> collectOrdersForHotel() {
        List<KitchenOrder> result = new ArrayList<>();
        for (KitchenOrder order : kitchenQueue) {
            if (order.belongsToHotel(Main.HOTEL_PATH)) {
                result.add(order);
            }
        }
        return result;
    }
}
```

Call House Keeping Use Case

Housekeeping.java

```
package Main.Employee;

import Main.Room.room;
import Main.Main;

import java.util.*;
import java.util.concurrent.LinkedBlockingQueue;

public class Housekeeping extends Employee implements CleaningStaff {

    // Each cleaning request is stored as a "ticket"
    private static final LinkedBlockingQueue<HousekeepingRequest> cleaningQueue =
new LinkedBlockingQueue<>();

    private static boolean active = true;
    private static String currentHotelAddress = null;
```

```
    public Housekeeping(int id, String name) {
        super(id, name, "CleaningStaff");
    }

    // Add a cleaning task (ticket)
    public void addToCleanQueue(room roomToClean) {
        if (roomToClean == null) {
            return;
        }

        HousekeepingRequest autoTurnover = new HousekeepingRequest(
                cloneRoom(roomToClean),
                "Turnover Cleaning",
                "Next Available",
                List.of(),
                "Auto-generated after checkout",
                false,
                "System"
        );

        if (enqueueRequest(autoTurnover)) {
            System.out.println("Cleaning request added for Room "
                    + roomToClean.getRoomNumber()
                    + " (" + safeAddress(roomToClean) + ")");
        }
    }

    public static boolean submitGuestRequest(HousekeepingRequest request) {
        boolean accepted = enqueueRequest(request);
        if (accepted) {
            System.out.println("Guest-initiated housekeeping request queued for Room "
                    + request.getRoomInfo().getRoomNumber() + ".");
        }
        return accepted;
    }

    private static boolean enqueueRequest(HousekeepingRequest request) {
        if (request == null || request.getRoomInfo() == null) {
            System.out.println("Unable to queue housekeeping request without room
information.");
            return false;
```

```
        }

        for (HousekeepingRequest pending : cleaningQueue) {
            if (sameRoom(pending.getRoomInfo(), request.getRoomInfo())
                    && pending.getServiceType().equalsIgnoreCase(request.getServiceType())
                    && pending.getTimeWindow().equalsIgnoreCase(request.getTimeWindow())) {
                System.out.println("A similar housekeeping request is already queued for Room
"
                        + request.getRoomInfo().getRoomNumber() + ".");
                return false;
            }
        }

        try {
            cleaningQueue.put(request);
            return true;
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            System.out.println("Failed to queue housekeeping request.");
            return false;
        }
    }

    private static room cloneRoom(room original) {
        return new room(
                original.getAddress(),
                original.getRoomNumber(),
                original.getType(),
                original.getAvailability(),
                original.getStartDate(),
                original.getEndDate()
        );
    }

    private static boolean sameRoom(room a, room b) {
        if (a == null || b == null) {
            return false;
        }
        String addressA = safeAddress(a);
        String addressB = safeAddress(b);
        return a.getRoomNumber() == b.getRoomNumber() &&
addressA.equalsIgnoreCase(addressB);
```

```
    }

    private static String safeAddress(room r) {
        return r.getAddress() == null ? "Unknown" : r.getAddress();
    }

    public static void processCleaning(String cleanerName) {
        if (!active) {
            System.out.println("Housekeeping is currently inactive.");
            return;
        }

        if (Main.HOTEL_PATH == null || Main.HOTEL_PATH.isEmpty()) {
            System.out.println("Hotel location not set. Please log in through your assigned
hotel.");
            return;
        }

        currentHotelAddress = Main.HOTEL_PATH;

        System.out.println("\n===== HOUSEKEEPING PANEL =====");
        System.out.println("Welcome, " + cleanerName + " (" + currentHotelAddress + ")");
        boolean running = true;
        Scanner sc = new Scanner(System.in);

        while (running) {
            System.out.println("\n--- Options ---");
            System.out.println("1. View Cleaning Tickets");
            System.out.println("2. Clean a Room");
            System.out.println("3. Exit Housekeeping Panel");
            System.out.print("Select an option: ");

            String choice = sc.nextLine().trim();

            if (choice.equals("1")) {
                viewTickets();
            }
            else if (choice.equals("2")) {
                handleCleaning(sc);
            }
            else if (choice.equals("3")) {
                System.out.println("Logging out of housekeeping...");
```

```
            running = false;
        }
        else {
            System.out.println("Invalid choice. Please select 1–3.");
        }
    }

}


    private static void viewTickets() {
        if (cleaningQueue.isEmpty()) {
            System.out.println("No cleaning requests at this time.");
            return;
        }

        System.out.println("\nCleaning Tickets for " + currentHotelAddress + ":");
        List<HousekeepingRequest> matchingRequests = new ArrayList<>();
        int index = 1;

        for (HousekeepingRequest request : cleaningQueue) {
            room ticketRoom = request.getRoomInfo();
            if (ticketRoom != null &&
safeAddress(ticketRoom).equalsIgnoreCase(currentHotelAddress)) {
                System.out.println(index + ". " + request.formatTicketLine());
                matchingRequests.add(request);
                index++;
            }
        }

        if (matchingRequests.isEmpty()) {
            System.out.println("No cleaning tasks for this hotel.");
        }
    }

    private static void handleCleaning(Scanner sc) {
        if (cleaningQueue.isEmpty()) {
            System.out.println("No cleaning requests at this time.");
            return;
        }

        List<HousekeepingRequest> available = new ArrayList<>();
```

```
        for (HousekeepingRequest request : cleaningQueue) {
            room ticketRoom = request.getRoomInfo();
            if (ticketRoom != null &&
safeAddress(ticketRoom).equalsIgnoreCase(currentHotelAddress)) {
                available.add(request);
            }
        }

        if (available.isEmpty()) {
            System.out.println("No cleaning requests for " + currentHotelAddress);
            return;
        }

        System.out.println("\nSelect a room to clean:");
        for (int i = 0; i < available.size(); i++) {
            HousekeepingRequest req = available.get(i);
            System.out.println((i + 1) + ". " + req.formatTicketLine());
        }

        System.out.print("Enter room number to clean (or 0 to cancel): ");
        String input = sc.nextLine().trim();

        int choice;
        try {
            choice = Integer.parseInt(input);
        } catch (NumberFormatException e) {
            System.out.println("Invalid input.");
            return;
        }

        if (choice == 0) return;
        if (choice < 1 || choice > available.size()) {
            System.out.println("Invalid selection.");
            return;
        }

        HousekeepingRequest selected = available.get(choice - 1);
        cleaningQueue.remove(selected);
        performCleaning(selected);
    }

    private static void performCleaning(HousekeepingRequest request) {
```

```
        room roomToClean = request.getRoomInfo();
        synchronized (roomToClean) {
            System.out.println("Cleaning started for Room " + roomToClean.getRoomNumber()
+ "...");
            System.out.println("Service: " + request.getServiceType() + " | Window: " +
request.getTimeWindow());
            if (!request.getAmenities().isEmpty()) {
                System.out.println("Amenities to deliver: " + String.join(", ",
request.getAmenities()));
            }
            if (request.getNotes() != null && !request.getNotes().isEmpty()) {
                System.out.println("Notes: " + request.getNotes());
            }
            try {
                Thread.sleep(1500); // Simulate cleaning
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
            System.out.println("Cleaning complete for Room " +
roomToClean.getRoomNumber());
        }
    }

    @SuppressWarnings("unused")
    private static void fireAllWorkers() {
        active = false;
        System.out.println("All housekeeping operations halted.");
    }
}
```

HousekeepingRequest.java

```
package Main.Employee;

import Main.Room.room;

import java.time.LocalDateTime;
import java.util.Collections;
import java.util.List;
import java.util.UUID;

public class HousekeepingRequest {
    private final UUID id;
    private final room roomInfo;
```

```
private final String serviceType;
private final String timeWindow;
private final List<String> amenities;
private final String notes;
private final boolean guestInitiated;
private final LocalDateTime requestedAt;
private final String requestedBy;

public HousekeepingRequest(room roomInfo,
                           String serviceType,
                           String timeWindow,
                           List<String> amenities,
                           String notes,
                           boolean guestInitiated,
                           String requestedBy) {
   this.id = UUID.randomUUID();
   this.roomInfo = roomInfo;
   this.serviceType = serviceType;
   this.timeWindow = timeWindow;
   this.amenities = amenities == null ? List.of() : List.copyOf(amenities);
   this.notes = notes;
   this.guestInitiated = guestInitiated;
   this.requestedAt = LocalDateTime.now();
   this.requestedBy = requestedBy;
}

public UUID getId() {
   return id;
}

public room getRoomInfo() {
   return roomInfo;
}

public String getServiceType() {
   return serviceType;
}

public String getTimeWindow() {
   return timeWindow;
}
```

```
    public List<String> getAmenities() {
        return Collections.unmodifiableList(amenities);
    }

    public String getNotes() {
        return notes;
    }

    public boolean isGuestInitiated() {
        return guestInitiated;
    }

    public LocalDateTime getRequestedAt() {
        return requestedAt;
    }

    public String getRequestedBy() {
        return requestedBy;
    }

    public String formatTicketLine() {
        StringBuilder sb = new StringBuilder();
        sb.append("Room ").append(roomInfo.getRoomNumber()).append(" | ")
          .append(serviceType).append(" @ ").append(timeWindow);
        if (!amenities.isEmpty()) {
            sb.append(" | Amenities: ").append(String.join("; ", amenities));
        }
        if (guestInitiated) {
            sb.append(" | Guest request");
        }
        return sb.toString();
    }
}
```

Clock in/Clock out

EmployeePortal.java:

```
package Main.Controller;
```

```
import Main.Employee.ClockSystem;
import Main.Employee.Employee;

import java.util.Scanner;

public class EmployeePortal {
    private ClockSystem clockSystem;

    public EmployeePortal(String dataFilePath) {
        clockSystem = new ClockSystem(dataFilePath);
    }

    public void runPortal(Scanner sc, Employee employee) {
        boolean running = true;
        int empId = employee.getId();

        while (running) {
            System.out.println("\n╔═══════════════════════════════════╗");
            System.out.println("║ EMPLOYEE PORTAL        ║");
            System.out.println("╠═══════════════════════════════════╣");
            System.out.println("║ 1. Clock In            ║");
            System.out.println("║ 2. Clock Out           ║");
            System.out.println("║ 3. Show Hours Worked   ║");
            System.out.println("║ 0. Back to Worker Portal║");
            System.out.println("╚═══════════════════════════════════╝");
            System.out.print("Select an option: ");

            String choice = sc.nextLine().trim();

            switch (choice) {
                case "1":
                    clockSystem.clockIn(empId);
                    break;
                case "2":
                    clockSystem.clockOut(empId);
                    break;
                case "3":
                    clockSystem.showHoursWorked(empId);
                    break;
                case "0":
                    running = false; // go back to Worker Portal
                    break;
```

```
            default:
                System.out.println("Invalid option. Choose 0-3.");
        }
    }
  }
}
```

Employee.java: (also worked/used by Pranava for his uses cases)

```
package Main.Employee;

import java.time.LocalDateTime;

public class Employee {
   private int StaffID;
   private String name;
   private String role;
   private LocalDateTime clockInTime;
   private LocalDateTime clockOutTime;
   private double totalHoursWorked;
   private String username;
   private String password;

   public Employee(int StaffID, String name, String role, String username, String
password){
      this.StaffID = StaffID;
      this.name = name;
      this.role = role;
      this.totalHoursWorked = 0.0;
      this.username = username;
      this.password = password;
   }

   public Employee(int id, String name, String role) {
      this.StaffID = StaffID;
      this.name = name;
      this.role = role;
   }

   public int getId(){
      return StaffID;
   }
```

```
    public String getName(){
        return name;
    }

    public String role() {
        return role;
    }

    public LocalDateTime getClockInTime() {
        return clockInTime;
    }

    public void setClockInTime(LocalDateTime clockInTime) {
        this.clockInTime = clockInTime;
    }

    public LocalDateTime getClockOutTime() {
        return clockOutTime;
    }

    public void setClockOutTime(LocalDateTime clockOutTime) {
        this.clockOutTime = clockOutTime;
    }

    public double getTotalHoursWorked() {
        return totalHoursWorked;
    }

    public void addHoursWorked(double hours) {
        this.totalHoursWorked += hours;
    }

    public String getUsername() { return username; }
    public String getPassword() { return password; }

    public void setUsername(String username) { this.username = username; }
    public void setPassword(String password) { this.password = password; }
}
```

ClockSystem.java

```
package Main.Employee;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.time.Duration;
import java.util.*;
import java.io.*;

public class ClockSystem {
    private HashMap<Integer, LocalDateTime> clockInTimes = new HashMap<>();
    private HashMap<Integer, Long> totalWorkedMinutes = new HashMap<>();
    private DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm");
    private File file;

    public ClockSystem(String filePath) {
        file = new File(filePath);
        loadFromFile();
    }

    private void loadFromFile() {
        if (!file.exists()) return;

        try (Scanner sc = new Scanner(file)) {
            while (sc.hasNextLine()) {
                String[] parts = sc.nextLine().split(",");
                if (parts.length != 3) continue;

                int empId = Integer.parseInt(parts[0]);
                String clockInStr = parts[1];
                long minutes = Long.parseLong(parts[2]);

                totalWorkedMinutes.put(empId, minutes);

                if (!clockInStr.equals("null")) {
                    clockInTimes.put(empId, LocalDateTime.parse(clockInStr, formatter));
                }
            }
        } catch (Exception e) {
            System.out.println("Error loading clock data: " + e.getMessage());
        }
    }
```

```
    private void saveToFile() {
        try (PrintWriter pw = new PrintWriter(file)) {
            for (int empId : totalWorkedMinutes.keySet()) {
                LocalDateTime inTime = clockInTimes.getOrDefault(empId, null);
                String clockInStr = inTime == null ? "null" : inTime.format(formatter);
                pw.println(empId + "," + clockInStr + "," + totalWorkedMinutes.get(empId));
            }
        } catch (Exception e) {
            System.out.println("Error saving clock data: " + e.getMessage());
        }
    }

    public void clockIn(int employeeId) {
        if (clockInTimes.containsKey(employeeId)) {
            System.out.println("Already clocked in at: " +
clockInTimes.get(employeeId).format(formatter));
            return;
        }

        LocalDateTime now = LocalDateTime.now();
        clockInTimes.put(employeeId, now);
        saveToFile();
        System.out.println("Clocked in at: " + now.format(formatter));
    }

    public void clockOut(int employeeId) {
        if (!clockInTimes.containsKey(employeeId)) {
            System.out.println("Employee hasn't clocked in yet.");
            return;
        }

        LocalDateTime inTime = clockInTimes.get(employeeId);
        LocalDateTime now = LocalDateTime.now();
        long minutesWorked = Duration.between(inTime, now).toMinutes();

        totalWorkedMinutes.put(employeeId, totalWorkedMinutes.getOrDefault(employeeId,
0L) + minutesWorked);
        clockInTimes.remove(employeeId);
        saveToFile();
```

```
        System.out.println("Clocked out at: " + now.format(formatter) + " | Session minutes: "
+ minutesWorked);
    }

    public void showHoursWorked(int employeeId) {
        long totalMinutes = totalWorkedMinutes.getOrDefault(employeeId, 0L);
        long hours = totalMinutes / 60;
        long minutes = totalMinutes % 60;
        System.out.println("Total hours worked: " + hours + "h " + minutes + "m");
    }
}
```

MiniBar/RoomService Setup

BillManager.java: (also worked on and used by Pranava for his uses cases)

```
package Main.Guest;

import Main.Main;
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Path;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class BillManager {
    private static final DateTimeFormatter TS_FMT =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    private static String billsDirForHotel() {
        return Main.HOTEL_PATH + "/Bills";
    }

    private static String billFileForRoom(int roomNumber) {
        return billsDirForHotel() + "/Room" + roomNumber + "_Bill.txt";
    }

    private static void ensureBillsDir() throws IOException {
```

```
        Path dir = Path.of(billsDirForHotel());
        if (!Files.exists(dir)) {
            Files.createDirectories(dir);
        }
    }

    // Create a new bill session for a guest
    public static void createBillForRoom(int roomNumber, String guestName) {
        try {
            ensureBillsDir();
            File billFile = new File(billFileForRoom(roomNumber));
            boolean isNew = billFile.createNewFile();

            try (PrintWriter out = new PrintWriter(new FileWriter(billFile, true))) {
                String ts = LocalDateTime.now().format(TS_FMT);
                if (isNew) {
                    out.println("# Bill file for Room " + roomNumber);
                    out.println("# Format: DESCRIPTION,AMOUNT");
                }
                out.println("# Guest: " + guestName + " @ " + ts);
            }
        } catch (IOException e) {
            System.out.println("Error creating bill file: " + e.getMessage());
        }
    }

    // Add charges for a session
    public static void writeSessionBill(int roomNumber, String guestName, Map<String,
Double> sessionCharges) {
        try {
            ensureBillsDir();
            File billFile = new File(billFileForRoom(roomNumber));
            if (!billFile.exists()) {
                createBillForRoom(roomNumber, guestName);
            }
            try (PrintWriter out = new PrintWriter(new FileWriter(billFile, true))) {
                for (Map.Entry<String, Double> entry : sessionCharges.entrySet()) {
                    String desc = entry.getKey().replaceAll(",", " ");
                    double amt = entry.getValue();
                    if (amt > 0) {
                        out.printf(Locale.US, "%s,%.2f%n", desc, amt);
                    }
```

```
            }
        }
    } catch (IOException e) {
        System.out.println("Error writing session bill: " + e.getMessage());
    }
}

// Reads only the last session for the given guest
public static List<String[]> readLastSessionForGuest(int roomNumber, String
guestName) throws IOException {
    List<String[]> items = new ArrayList<>();
    File billFile = new File(billFileForRoom(roomNumber));
    if (!billFile.exists()) return items;

    List<String> lines = Files.readAllLines(billFile.toPath());
    boolean inGuestSession = false;

    // Scan from top to bottom to find the last session for the given guest
    for (int i = 0; i < lines.size(); i++) {
        String line = lines.get(i).trim();
        if (line.startsWith("# Guest:")) {
            inGuestSession = line.toLowerCase().contains(guestName.toLowerCase());
            continue;
        }
        if (inGuestSession && !line.startsWith("#") && !line.isEmpty()) {
            String[] parts = line.split(",");
            if (parts.length == 3) {
                items.add(new String[]{parts[0].trim(), parts[1].trim(), parts[2].trim()});
            } else if (parts.length == 4) {
                items.add(new String[]{parts[0].trim(), parts[1].trim() + " " + parts[2].trim(),
parts[3].trim()});
            }
            else {
                items.add(new String[]{parts[0].trim(), parts[1].trim()});
            }
        }
    }
    return items;
}

public static void printBillForGuestByName(String guestName) {
    int room = lookupRoomNumberFromPastReservation(guestName);
```

```
        if (room == -1) {
            System.out.println("No past reservation found for " + guestName + ". Could not
locate bill.");
            return;
        }

        try {
            List<String[]> items = readLastSessionForGuest(room, guestName);
            if (items.isEmpty()) {
                System.out.println("No charges recorded for this guest.");
                return;
            }

            final int WIDTH = 37;

            // ---------- HEADER ----------
            System.out.println("=".repeat(WIDTH+5));
            String header1 = "FINAL BILL - ROOM " + room;
            String header2 = "Guest: " + guestName;

            int pad1 = (WIDTH - header1.length()) / 2;
            int pad2 = (WIDTH - header2.length()) / 2;

            if (pad1 < 0) pad1 = 0;
            if (pad2 < 0) pad2 = 0;

            System.out.println(" ".repeat(pad1) + header1);
            System.out.println(" ".repeat(pad2) + header2);
            System.out.println("=".repeat(WIDTH+5));
            System.out.println();

            // ---- Group charges ----
            double securityDeposit = 0;
            List<String[]> roomCharges = new ArrayList<>();
            List<String[]> minibarCharges = new ArrayList<>();
            List<String[]> roomServiceCharges = new ArrayList<>();

            for (String[] it : items) {
                switch (it[0]) {
                    case "SECURITY_DEPOSIT":
                        securityDeposit += Double.parseDouble(it[1]);
                        break;
```

```
                case "ROOM_CHARGE":
                    roomCharges.add(it);
                    break;

                case "Water":
                case "Soda":
                case "Chips":
                case "Chocolate":
                    minibarCharges.add(it);
                    break;

                case "Sandwich":
                case "Salad":
                case "Pizza":
                case "Coffee":
                    roomServiceCharges.add(it);
                    break;
            }
        }

        double subtotal = 0.0;

        // ---------- SECURITY DEPOSIT ----------
        int padding = (WIDTH - "SECURITY DEPOSIT".length() - 10) / 2;
        if (padding < 0) padding = 0;
        System.out.println("=".repeat(padding+4) + "==== SECURITY DEPOSIT " +
"=".repeat(padding + 6));

        System.out.printf("%12s %28s%n",
                "Deposit", String.format("$%7.2f", securityDeposit));

        subtotal += securityDeposit;

        // ---------- ROOM CHARGES ----------
        if (!roomCharges.isEmpty()) {
            int pad = (WIDTH - "ROOM CHARGES".length() - 10) / 2;
            if (pad < 0) pad = 0;
            System.out.println("=".repeat(pad+4) + "==== ROOM CHARGES " +
"=".repeat(pad+6));

            for (String[] rc : roomCharges) {
```

```
                double amt = Double.parseDouble(rc[2]);
                System.out.printf("%12s %28s%n",
                        rc[1], String.format("$%7.2f", amt));
                subtotal += amt;
            }
        }

        // ---------- MINI BAR ----------
        if (!minibarCharges.isEmpty()) {
            int pad = (WIDTH - "MINI BAR".length() - 10) / 2;
            if (pad < 0) pad = 0;
            System.out.println("=".repeat(pad+4) + "==== MINI BAR " + "=".repeat(pad+6));

            for (String[] mb : minibarCharges) {
                double amt = Double.parseDouble(mb[1]);
                System.out.printf("%12s %28s%n",
                        mb[0], String.format("$%7.2f", amt));
                subtotal += amt;
            }
        }

        // ---------- ROOM SERVICE ----------
        if (!roomServiceCharges.isEmpty()) {
            int pad = (WIDTH - "ROOM SERVICE".length() - 10) / 2;
            if (pad < 0) pad = 0;
            System.out.println("=".repeat(pad+4) + "==== ROOM SERVICE " +
"=".repeat(pad+6));

            for (String[] rs : roomServiceCharges) {
                double amt = Double.parseDouble(rs[1]);
                System.out.printf("%12s %28s%n",
                        rs[0], String.format("$%7.2f", amt));
                subtotal += amt;
            }
        }

        // ---------- SUBTOTAL ----------
        System.out.println("-".repeat(WIDTH+5));
        System.out.printf("%12s %28s%n",
                "SUBTOTAL", String.format("$%7.2f", subtotal));

        // ---------- TAX ----------
```

```
        double taxRate = 0;

        if (Main.HOTEL_PATH.equals("Chicago")) {
            taxRate = 0.06 + 0.045 + 0.06 + 0.025 + 0.02 + 0.01;
        } else if (Main.HOTEL_PATH.equals("DesMoines")) {
            taxRate = 0.05 + 0.07;
        }

        double taxAmount = subtotal * taxRate;

        System.out.printf("%12s %28s%n",
                "TAX", String.format("$%7.2f", taxAmount));

        System.out.println("-".repeat(WIDTH+5));

        // ---------- GRAND TOTAL ----------
        double grandTotal = subtotal + taxAmount;
        System.out.printf("%12s %28s%n",
                "GRAND TOTAL", String.format("$%7.2f", grandTotal));

        System.out.println("=".repeat(WIDTH+5));
        System.out.println();

    } catch (Exception e) {
        System.out.println("Error reading bill: " + e.getMessage());
    }
}

// Lookup room number logic unchanged
public static int lookupRoomNumberFromPastReservation(String guestName) {
    String pastPath = Main.HOTEL_PATH + "/" + "Past_Reservation.txt";
    File pastFile = new File(pastPath);
    if (!pastFile.exists()) return -1;

    int foundRoom = -1;
    String foundDate = null;

    try (BufferedReader br = new BufferedReader(new FileReader(pastFile))) {
        String line;
        while ((line = br.readLine()) != null) {
            if (line.trim().isEmpty() || line.startsWith("reservationId")) continue;
            String[] parts = line.split(",");
```

```
            if (parts.length >= 6) {
                String rGuest = parts[1].trim();
                if (rGuest.equalsIgnoreCase(guestName.trim())) {
                    String actualOut = parts[5].trim();
                    if (foundDate == null || actualOut.compareTo(foundDate) >= 0) {
                        foundDate = actualOut;
                        try {
                            foundRoom = Integer.parseInt(parts[2].trim());
                        } catch (NumberFormatException ex) {
                            foundRoom = -1;
                        }
                    }
                }
            }
        } catch (IOException e) {
            return -1;
        }
        return foundRoom;
    }

    public static boolean deleteBillForRoom(int roomNumber) {
        File f = new File(billFileForRoom(roomNumber));
        return f.exists() && f.delete();
    }
}
```

GuestSession.java: (also worked on and used by Brendan and Pranava for there uses cases)

```
package Main.Guest;

import Main.Booking.Reservation;
import Main.Employee.Employee;
import Main.Employee.FrontDesk;
import Main.Employee.Housekeeping;
import Main.Employee.HousekeepingRequest;
import Main.Employee.KitchenOrder;
import Main.Employee.KitchenPanel;
import Main.Main;
import Main.Room.room;
```

```
import java.io.IOException;
import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

public class GuestSession implements CheckingProcess {

    private static final List<String> SERVICE_TYPES = List.of(
        "Full Cleaning",
        "Quick Tidy",
        "Turn-Down",
        "Amenity Request Only"
    );

    private static final List<String> TIME_WINDOWS = List.of(
        "As soon as possible",
        "08:00 - 10:00",
        "10:00 - 12:00",
        "14:00 - 16:00",
        "18:00 - 20:00"
    );

    private static final List<String> AMENITY_OPTIONS = List.of(
        "Extra Towels",
        "Toiletries",
        "Extra Pillows",
        "Fresh Linens",
        "Coffee Pods",
    );

    @Override
    @SuppressWarnings("resource")
    public void checkin(Employee frontDesk) {
        Scanner scnr = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = scnr.nextLine();
```

```
        System.out.println("Okay, and could you provide your id please?");
        int id = scnr.nextInt();
        scnr.nextLine(); // consume newline

        Guest guest = new Guest(name, id);

        boolean check = ((FrontDesk) frontDesk).verifyIdentity(guest);

        if (check) {
            System.out.println("Thank you. Give me a moment to verify your reservation");
            Reservation reserve = ((FrontDesk) frontDesk).verifyCheckIn(guest);

            if (reserve != null) {
                System.out.println("\nThanks for waiting, you have a reservation from "
                        + reserve.getStartDate() + " to " + reserve.getEndDate() + "\n");

                // Create a new session bill for this guest
                BillManager.createBillForRoom(reserve.getRoomNumber(), guest.getName());
                String location = Main.HOTEL_PATH;
                BillingManager.initBillingForStay(reserve, guest.getName(), location);

                ((FrontDesk) frontDesk).provideKeyCard(reserve.getRoomNumber(), guest);
                System.out.println("Here is your keycard for room " +
reserve.getRoomNumber());
            } else {
                System.out.println("No reservation found for your details.");
            }
        } else {
            System.out.println("Sorry, we couldn't verify your identity.");
        }
    }

    @Override
    @SuppressWarnings("resource")
    public void checkout(Employee frontDesk) {
        Scanner scnr = new Scanner(System.in);

        LocalTime now = LocalTime.now();
        String greeting;

        if (now.isBefore(java.time.LocalTime.NOON))
```

```
            greeting = "Good morning";
        else if (now.isBefore(java.time.LocalTime.of(17, 0)))
            greeting = "Good afternoon";
        else if (now.isBefore(java.time.LocalTime.of(21, 0)))
            greeting = "Good evening";
        else
            greeting = "Good night";

        System.out.println(greeting + "! Checking out today?");

        System.out.print("Can I get the name on the reservation? ");
        String name = scnr.nextLine();

        System.out.print("And the ID you used during check-in? ");
        int id = scnr.nextInt();
        scnr.nextLine(); // consume leftover newline

        Guest guest = new Guest(name, id);

        boolean verified = ((FrontDesk) frontDesk).verifyIdentity(guest);

        if (!verified) {
            System.out.println("Hmm, I'm not finding a match with that name and ID. Could you
double-check them?");
            return;
        }

        boolean checkedOut = ((FrontDesk) frontDesk).verifyCheckOut(guest);

        if (checkedOut) {
            System.out.println("Alright, " + guest.getName() + ", I found your reservation.");
            System.out.println("Your checkout is all set. I'll take your keycard—thank you!");

            // Print final bill for **this guest only**
            System.out.println("\nCalculating final bill...");
            BillManager.printBillForGuestByName(guest.getName());

            System.out.println("\nWe hope you had a pleasant stay. Safe travels!");
        } else {
            System.out.println("It looks like you don't have any active reservations right now.");
        }
    }
```

```
    /**
     * Called when guest accesses room via keycard.
     * Allows them to add MiniBar or RoomService charges.
     */
    @SuppressWarnings("resource")
    public static void roomAccess(int roomNumber, String guestName, room
accessedRoom) {
        Scanner scnr = new Scanner(System.in);

        System.out.println("===== ROOM ACCESS =====");
        System.out.println("Welcome to your room! You can add charges for MiniBar or
RoomService,");
        System.out.println("and you can request Housekeeping directly from here.");
        System.out.println("Type 'DONE' when finished.\n");

        // Define items and prices
        Map<String, Double> miniBarItems = new HashMap<>();
        miniBarItems.put("Chips", 2.0);
        miniBarItems.put("Chocolate", 3.0);
        miniBarItems.put("Soda", 1.5);
        miniBarItems.put("Water", 1.0);

        Map<String, Double> roomServiceItems = new HashMap<>();
        roomServiceItems.put("Sandwich", 5.0);
        roomServiceItems.put("Salad", 6.0);
        roomServiceItems.put("Pizza", 8.0);
        roomServiceItems.put("Coffee", 2.0);

    Map<String, Double> sessionCharges = new HashMap<>();
    Map<String, Integer> roomServiceCounts = new LinkedHashMap<>();
    double roomServiceTotal = 0.0;
    boolean housekeepingRequested = false;

        while (true) {
            System.out.print("Enter type (MiniBar / RoomService / Housekeeping) or DONE: ");
            String type = scnr.nextLine().trim();

            if (type.equalsIgnoreCase("DONE")) break;

            if (type.equalsIgnoreCase("MiniBar")) {
                System.out.println("Available MiniBar items:");
```

```
            miniBarItems.forEach((item, price) -> System.out.printf("- %s ($%.2f)\n", item,
price));

            System.out.print("Select an item: ");
            String item = scnr.nextLine().trim();

            if (miniBarItems.containsKey(item)) {
                double price = miniBarItems.get(item);
                sessionCharges.put(item, sessionCharges.getOrDefault(item, 0.0) + price);
                System.out.printf("%s charge of $%.2f added to your bill.\n", item, price);
            } else {
                System.out.println("Invalid item.");
            }

        } else if (type.equalsIgnoreCase("RoomService")) {
            System.out.println("Available RoomService items:");
            roomServiceItems.forEach((item, price) -> System.out.printf("- %s ($%.2f)\n",
item, price));

            System.out.print("Select an item: ");
            String item = scnr.nextLine().trim();

            if (roomServiceItems.containsKey(item)) {
                double price = roomServiceItems.get(item);
                sessionCharges.put(item, sessionCharges.getOrDefault(item, 0.0) + price);
                roomServiceCounts.put(item, roomServiceCounts.getOrDefault(item, 0) + 1);
                roomServiceTotal += price;
                System.out.printf("%s charge of $%.2f added to your bill.\n", item, price);
            } else {
                System.out.println("Invalid item.");
            }

        } else if (type.equalsIgnoreCase("Housekeeping")) {
            if (accessedRoom == null) {
                System.out.println("Room context unavailable. Try again later or contact the
front desk.");
                continue;
            }

            boolean queued = handleHousekeepingFlow(scnr, accessedRoom, guestName);
            if (queued) {
                housekeepingRequested = true;
```

```
            }
        } else {
            System.out.println("Invalid type. Choose MiniBar, RoomService, Housekeeping,
or DONE.");
        }
    }

    // Write session charges to file under this guest's header
    BillManager.writeSessionBill(roomNumber, guestName, sessionCharges);

    if (!roomServiceCounts.isEmpty()) {
        KitchenOrder order = new KitchenOrder(
                Main.HOTEL_PATH,
                roomNumber,
                guestName,
                roomServiceCounts,
                roomServiceTotal
        );
        KitchenPanel.submitOrder(order);
    }

    if (housekeepingRequested) {
        System.out.println("Housekeeping has been notified for your room. Someone will
visit shortly.");
    }

    // Display current totals based on persisted file entries
    try {
        List<String[]> recorded = BillManager.readLastSessionForGuest(roomNumber,
guestName);
        if (recorded.isEmpty()) {
            System.out.println("\nNo bill entries found for this guest yet.");
            return;
        }

        System.out.println("\n================ CURRENT BILL FOR ROOM " +
roomNumber + " ============================");

        double runningTotal = 0.0;

// Column widths
        final int LABEL_WIDTH = 22;
```

```
        final int DATE_WIDTH  = 14;
        final int AMOUNT_WIDTH = 10;

        for (String[] line : recorded) {
            if (line.length < 2) continue;

            if (line[0].equals("SECURITY_DEPOSIT (REFUNDED AT CHECK-OUT)"))
                continue;

            String label = line[0];
            String date  = "";
            String amountStr;

            if (line.length >= 3) {
                date = line[1];
                amountStr = line[2];
            } else {
                amountStr = line[1];
            }

            double amount = 0;
            try { amount = Double.parseDouble(amountStr); } catch (Exception ignored) {}

            runningTotal += amount;

            if (line[0].startsWith("SECURITY_DEPOSIT")) {
                int width = 6;
                System.out.printf(
                        "%-" + LABEL_WIDTH + "s %-" + DATE_WIDTH + "s %" + width +
"s$%.2f%n",
                        label, date, "", amount
                );
            } else {
                int width = 23;
//                System.out.printf(
//                        "%-" + LABEL_WIDTH + "s %" + "s$%.2f%n",
//                        label, date, amount
//                );
                System.out.printf(
                        "%-" + LABEL_WIDTH + "s %-" + DATE_WIDTH + "s %" + width +
"s$%.2f%n",
                        label, date, "", amount
```

```
                );
            }
        }
        int width = 22;

System.out.println("------------------------------------------------------------------------")
;
        System.out.printf(
                "%-" + (LABEL_WIDTH + DATE_WIDTH) + "s %" + width + "s$%.2f%n",
                "TOTAL", "", runningTotal
        );
      } catch (IOException e) {
        System.out.println("Unable to read current bill: " + e.getMessage());
      }
   }

   private static boolean handleHousekeepingFlow(Scanner sc,
                                      room accessedRoom,
                                      String guestName) {
      System.out.println("\n===== HOUSEKEEPING REQUEST =====");
      System.out.println("We'll gather a few details to schedule your service.");

      String serviceType = promptFromList(sc, SERVICE_TYPES, "Select a service type (1-"
+ SERVICE_TYPES.size() + "): ");
      if (serviceType == null) {
        return false;
      }

      String timeWindow = promptFromList(sc, TIME_WINDOWS, "Preferred time window
(1-" + TIME_WINDOWS.size() + "): ");
      if (timeWindow == null) {
        return false;
      }

      boolean dndActive = askYesNo(sc, "Is your room currently set to 'Do Not Disturb'?
(y/n): ");
      timeWindow = adjustTimeWindow(timeWindow, dndActive);

      List<String> amenities = promptAmenities(sc);

      String additionalNotes = "";
      if (askYesNo(sc, "Any additional notes or requests? (y/n): ")) {
```

```
            System.out.print("Enter notes: ");
            additionalNotes = sc.nextLine().trim();
        }

        System.out.println("\n--- Request Summary ---");
        System.out.println("Room: " + accessedRoom.getRoomNumber());
        System.out.println("Service: " + serviceType);
        System.out.println("Window: " + timeWindow);
        if (!amenities.isEmpty()) {
            System.out.println("Amenities: " + String.join(", ", amenities));
        } else {
            System.out.println("Amenities: None");
        }
        if (!additionalNotes.isEmpty()) {
            System.out.println("Notes: " + additionalNotes);
        }

        if (!askYesNo(sc, "Submit this housekeeping request? (y/n): ")) {
            System.out.println("Request cancelled.");
            return false;
        }

        String finalNotes = buildNotes(additionalNotes, dndActive);
        HousekeepingRequest request = new HousekeepingRequest(
                cloneRoomForTicket(accessedRoom),
                serviceType,
                timeWindow,
                amenities,
                finalNotes,
                true,
                guestName
        );

        boolean accepted = Housekeeping.submitGuestRequest(request);
        if (accepted) {
            System.out.println("Confirmed: Housekeeping has your request. You'll receive
service " + timeWindow + ".");
        }
        return accepted;
    }

    private static String promptFromList(Scanner sc, List<String> options, String prompt) {
```

```
        for (int i = 0; i < options.size(); i++) {
            System.out.println((i + 1) + ". " + options.get(i));
        }
        System.out.print(prompt);
        String input = sc.nextLine().trim();
        try {
            int idx = Integer.parseInt(input) - 1;
            if (idx >= 0 && idx < options.size()) {
                return options.get(idx);
            }
        } catch (NumberFormatException ignored) {
        }
        System.out.println("Invalid selection.");
        return null;
    }

    private static boolean askYesNo(Scanner sc, String prompt) {
        while (true) {
            System.out.print(prompt);
            String input = sc.nextLine().trim().toLowerCase();
            if (input.equals("y") || input.equals("yes")) return true;
            if (input.equals("n") || input.equals("no")) return false;
            System.out.println("Please enter 'y' or 'n'.");
        }
    }

    private static String adjustTimeWindow(String selection, boolean dndActive) {
        LocalTime now = LocalTime.now();
        LocalTime serviceStart = LocalTime.of(7, 0);
        LocalTime serviceEnd = LocalTime.of(21, 0);

        if (selection.equalsIgnoreCase("As soon as possible")) {
            if (dndActive) {
                System.out.println("⚠ Room is set to Do Not Disturb. Scheduling the earliest
afternoon slot instead.");
                return "14:00 - 16:00 (after DND)";
            }
            if (now.isBefore(serviceStart) || now.isAfter(serviceEnd.minusMinutes(30))) {
                System.out.println("Housekeeping is unavailable right now. Scheduling for 08:00
- 10:00 next service window.");
                return "08:00 - 10:00 (next service window)";
            }
```

```
            return "As soon as possible (within 30 mins)";
        }

        Map<String, LocalTime[]> bounds = timeWindowBounds();
        LocalTime[] windowBounds = bounds.get(selection);
        if (windowBounds != null && now.isAfter(windowBounds[1])) {
            System.out.println("That time slot has passed. Scheduling the next available slot
instead.");
            return "14:00 - 16:00";
        }

        if (now.isAfter(serviceEnd)) {
            System.out.println("Service hours have ended. Scheduling for tomorrow morning
08:00 - 10:00.");
            return "08:00 - 10:00 (next day)";
        }

        return selection;
    }

    private static Map<String, LocalTime[]> timeWindowBounds() {
        Map<String, LocalTime[]> bounds = new HashMap<>();
        bounds.put("08:00 - 10:00", new LocalTime[]{LocalTime.of(8, 0), LocalTime.of(10,
0)});
        bounds.put("10:00 - 12:00", new LocalTime[]{LocalTime.of(10, 0), LocalTime.of(12,
0)});
        bounds.put("14:00 - 16:00", new LocalTime[]{LocalTime.of(14, 0), LocalTime.of(16,
0)});
        bounds.put("18:00 - 20:00", new LocalTime[]{LocalTime.of(18, 0), LocalTime.of(20,
0)});
        return bounds;
    }

    private static List<String> promptAmenities(Scanner sc) {
        System.out.println("Optional amenities (separate multiple choices with commas, or
press Enter for none):");
        for (int i = 0; i < AMENITY_OPTIONS.size(); i++) {
            System.out.println((i + 1) + ". " + AMENITY_OPTIONS.get(i));
        }
        System.out.print("Amenities: ");
        String input = sc.nextLine().trim();
        if (input.isEmpty()) {
```

```
            return Collections.emptyList();
        }

        String[] tokens = input.split(",");
        List<String> selections = new ArrayList<>();
        for (String token : tokens) {
            String trimmed = token.trim();
            try {
                int idx = Integer.parseInt(trimmed) - 1;
                if (idx >= 0 && idx < AMENITY_OPTIONS.size()) {
                    selections.add(AMENITY_OPTIONS.get(idx));
                }
            } catch (NumberFormatException e) {
                if (AMENITY_OPTIONS.contains(trimmed)) {
                    selections.add(trimmed);
                }
            }
        }
        return selections;
    }

    private static String buildNotes(String userNotes, boolean dndActive) {
        List<String> notes = new ArrayList<>();
        if (dndActive) {
            notes.add("Guest indicated Do Not Disturb was active. Ensure DND sign is cleared
at arrival.");
        }
        if (userNotes != null && !userNotes.isEmpty()) {
            notes.add(userNotes);
        }
        return String.join(" | ", notes);
    }

    private static room cloneRoomForTicket(room original) {
        return new room(
                original.getAddress(),
                original.getRoomNumber(),
                original.getType(),
                original.getAvailability(),
                original.getStartDate(),
                original.getEndDate()
        );
```

```
    }
}
```

Write a review

Booking/Reservation.java

```
package Main.Booking;

import Main.Main;
import Main.Data.FileDataRepository;
import java.nio.file.Path;
import java.time.LocalDate;

public class Reservation {
    private String reservationId;
    private String guestName;
    private int roomNumber;
    private LocalDate startDate;
    private LocalDate endDate;
    private boolean complete = false;

    private static final String RESERVATION = Main.HOTEL_PATH + "/Reservation.txt";
    private static final String PAST_RESERVATIONS = Main.HOTEL_PATH +
"/Past_Reservation.txt";

    public Reservation(String reservationId, String guestName, int roomNumber, LocalDate
startDate, LocalDate endDate) {
        this.reservationId = reservationId;
        this.guestName = guestName;
        this.roomNumber = roomNumber;
        this.startDate = startDate;
        this.endDate = endDate;
        This.complete = true;
    }

    public Reservation(String name, String resId) {
        String[] customerInfo = checkReservations(RESERVATION, name, resId);
        if (customerInfo == null) {
            customerInfo = checkReservations(PAST_RESERVATIONS, name, resId);
        }
        if (customerInfo != null) {
```

```
            this.reservationId = customerInfo[0];
            this.guestName = customerInfo[1];
            this.roomNumber = Integer.parseInt(customerInfo[2]);
            this.startDate = LocalDate.parse(customerInfo[3]);
            this.endDate = LocalDate.parse(customerInfo[4]);
            this.complete = true;
        } else {
            System.out.println("Reservation not found for the given name and reservation ID.");
            this.complete = false;
        }
    }

    private String[] checkReservations(String path, String nameInput, String resIdInput) {
        java.util.List<String[]> reservations;
        FileDataRepository fileDataRepository = new
FileDataRepository(Path.of(Main.HOTEL_PATH));

        if (path.equals(RESERVATION)) {
            reservations = fileDataRepository.loadReservations();
        } else {
            reservations = fileDataRepository.loadPastReservations();
        }

        for (String[] reservation : reservations) {
            // Skip header row
            if (reservation[0].equals("reservationId")) {
                continue;
            }
            if (reservation[0].equals(resIdInput) && reservation[1].equals(nameInput)) {
                return reservation;
            }
        }

        return null;
    }

    public String getReservationId() {
        return reservationId;
    }

    public String getGuestName() {
        return guestName;
```

```
    }

    public int getRoomNumber() {
        return roomNumber;
    }

    public LocalDate getStartDate() {
        return startDate;
    }

    public LocalDate getEndDate() {
        return endDate;
    }

    @Override
    public String toString() {
        return reservationId + "," + guestName + "," + roomNumber + "," + startDate + "," +
endDate;
    }
}
```

Booking/ReviewProcess.java

```
package Main.Booking;

import Main.Data.ReviewRepository;

import java.util.Scanner;

public class ReviewProcess extends Reservation implements reviewInterface{
    private Scanner input;
    private ReviewRepository reviewRepository;

    public ReviewProcess(Reservation reservation) {
        super(reservation.getReservationId(), reservation.getGuestName(),
reservation.getRoomNumber(), reservation.getStartDate(), reservation.getEndDate());
        this.input = null;
        this.reviewRepository = new ReviewRepository();
    }

    public void execute(Scanner input) {
        System.out.println("Thank you for leaving a review!");
        this.input = input;
```

```
        int stars = getStarRating();
        System.out.print("Please enter your review (optional): ");
        String review = input.nextLine();
        reviewRepository.addReview(this, stars, review);
        System.out.println("Thank you for submitting a review! A team member will be in
touch shortly.");
    }

    private int getStarRating() {
        while (true) {
            System.out.print("How would you rate your stay? (1–5): ");

            if (input.hasNextInt()) {
                int stars = input.nextInt();
                input.nextLine();

                if (stars >= 1 && stars <= 5) {
                    return stars;
                }
                System.out.println("Invalid number. Please enter a value between 1 and 5.");
            } else {
                System.out.println("Invalid input. Please enter a number between 1 and 5.");
                input.nextLine();
            }
        }
    }
}
```

Data/ReviewRepository,java

```
package Main.Data;

import Main.Booking.Reservation;
import Main.Main;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
```

```
public class ReviewRepository {
    private static final String REVIEWS_FILE = Main.HOTEL_PATH + "/reviews.txt";

    public ReviewRepository() {
    }

    /**
     * Adds a new review to the reviews file
     */
    public void addReview(Reservation reservation, int stars, String reviewText) {
        List<String[]> allReviews = new ArrayList<>();
        String[] header = null;
        boolean found = false;

        // Read all reviews
        File reviewFile = new File(REVIEWS_FILE);
        if (reviewFile.exists()) {
            try (Scanner scanner = new Scanner(reviewFile)) {
                while (scanner.hasNextLine()) {
                    String line = scanner.nextLine().trim();
                    if (line.isEmpty()) {
                        continue;
                    }

                    String[] parts = line.split(",");

                    if (parts[0].startsWith("reservationId")) {
                        header = parts;
                        continue;
                    }

                    // Check if this reservation already has a review
                    if (parts[0].equals(reservation.getReservationId()) && !found) {
                        String reviewDate = java.time.LocalDate.now().toString();
                        parts[4] = String.valueOf(stars);
                        parts[5] = escapeCSV(reviewText);
                        parts[6] = reviewDate;
                        parts[7] = "PENDING";
                        parts[8] = "PENDING";
                        parts[9] = "PENDING";
                        parts[10] = "PENDING";
                        found = true;
```

```
                }

                allReviews.add(parts);
            }
        } catch (IOException e) {
            System.out.println("Error reading reviews: " + e.getMessage());
        }
    }

    // If not found, add new review
    if (!found) {
        String reviewDate = java.time.LocalDate.now().toString();
        String[] newReview = {
            reservation.getReservationId(),
            reservation.getGuestName(),
            String.valueOf(reservation.getRoomNumber()),
            reservation.getEndDate().toString(),
            String.valueOf(stars),
            escapeCSV(reviewText),
            reviewDate,
            "PENDING",
            "PENDING",
            "PENDING",
            "PENDING"
        };
        allReviews.add(newReview);
    }

    // Write all reviews back
    writeAllReviews(header, allReviews);

    if (found) {
        System.out.println("Review updated successfully!");
    } else {
        System.out.println("Review added successfully!");
    }
}

/**
 * Loads all reviews from the file
 */
public List<String[]> loadAllReviews() {
```

```
        List<String[]> reviews = new ArrayList<>();
        File reviewFile = new File(REVIEWS_FILE);

        if (!reviewFile.exists()) {
            return reviews;
        }

        try (Scanner scanner = new Scanner(reviewFile)) {
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine().trim();
                if (line.isEmpty()) {
                    continue;
                }

                String[] parts = line.split(",");
                reviews.add(parts);
            }
        } catch (IOException e) {
            System.out.println("Error reading reviews: " + e.getMessage());
        }

        return reviews;
    }

    /**
     * Updates a review with HR response
     */
    public void addHRResponse(Reservation reservationId, String hrName, String
response) {
        List<String[]> allReviews = new ArrayList<>();
        String[] header = null;
        boolean found = false;

        // Read all reviews
        File reviewFile = new File(REVIEWS_FILE);
        try (Scanner scanner = new Scanner(reviewFile)) {
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine().trim();
                if (line.isEmpty()) {
                    continue;
                }
```

```
            String[] parts = line.split(",");

            if (parts[0].startsWith("reservationId")) {
                header = parts;
                continue;
            }

            // Check if this is the review we want to reply to
            if (parts[0].equals(reservation.getReservationId()) && !found) {
                parts[7] = escapeCSV(response);
                parts[8] = hrName;
                parts[9] = java.time.LocalDate.now().toString();
                parts[10] = "REPLIED";
                found = true;
            }

            allReviews.add(parts);
        }
    } catch (IOException e) {
        System.out.println("Error reading reviews: " + e.getMessage());
        return;
    }

    // Write all reviews back
    writeAllReviews(header, allReviews);

    if (found) {
        System.out.println("Reply saved successfully!");
    } else {
        System.out.println("Review not found for reservation ID: " +
reservation.getReservationId());
    }
}

/**
 * Writes all reviews back to the file
 */
private void writeAllReviews(String[] header, List<String[]> reviews) {
    File reviewFile = new File(REVIEWS_FILE);
    try (PrintWriter writer = new PrintWriter(new FileWriter(reviewFile, false))) {
        if (header != null) {
            writer.println(String.join(",", header));
```

```
            } else {

writer.println("reservationId,guestName,roomNumber,checkoutDate,stars,review,reviewD
ate,hrResponse,hrRespondent,responseDate,status");
            }

            for (String[] review : reviews) {
                writer.println(String.join(",", review));
            }
        } catch (IOException e) {
            System.out.println("Error writing reviews: " + e.getMessage());
        }
    }

    /**
     * Escapes CSV special characters
     */
    private String escapeCSV(String value) {
        if (value == null || value.isEmpty()) {
            return "";
        }
        if (value.contains(",") || value.contains("\"") || value.contains("\n")) {
            return "\"" + value.replace("\"", "\"\"") + "\"";
        }
        return value;
    }
}
```

Reply to Reviews

Controller/HRController.java

```
package Main.Controller;

import Main.Employee.ExecutivePanel;
import Main.Employee.HRPanel;

import java.util.Scanner;
```

```
public class HRController {
    public static void runHRPanel(String hrName) {
        Scanner sc = new Scanner(System.in);
        HRPanel hrPanel = new HRPanel(3001, hrName, "HR");

        System.out.println("\n===== HR PANEL =====");
        System.out.println("Welcome, " + hrName + "!");
        System.out.println("--------------------------");

        boolean running = true;

        while (running) {
            System.out.println("\n--- HR Options ---");
            System.out.println("1. View Reviews");
            System.out.println("2. Hire Workers");
            System.out.println("3. Fire Workers");
            System.out.println("4. Massive Layoffs");
            System.out.println("5. Exit HR Panel");
            System.out.println("6. Notifications");
            System.out.print("Enter your choice: ");

            String choice = sc.nextLine().trim();

            if (choice.equals("1")) {
                hrPanel.viewReviews();
            } else if (choice.equals("2")) {
                hrPanel.hireWorkers(null);;
            } else if (choice.equals("3")) {
                hrPanel.fireWorkers(null);
            } else if (choice.equals("4")) {
                hrPanel.massiveLayoffs();
            } else if (choice.equals("5")) {
                System.out.println("Logging out of Executive Panel...");
                running = false;
            }else if(choice.equals("6")){
                hrPanel.viewNotifications();
            } else {
                System.out.println("Invalid choice. Please enter 1–5.");
            }
        }
    }
```

```
}
```

Employee/HRPanel.java

```
package Main.Employee;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import Main.*;
import Main.Data.ReviewRepository;

public class HRPanel extends Employee implements HRTeam {

    private ArrayList<String[]> reviews = new ArrayList<>();
    private ReviewRepository reviewRepository;

    public HRPanel(int StaffID, String name, String role) {
        super(StaffID, name, role);
        this.reviewRepository = new ReviewRepository();
    }

    @Override
    public void viewReviews() {
        // Implementation for viewing reviews
        System.out.println("\n--- Viewing: " + this.getName() + " ---");
        System.out.println("-----------------------------------------");

        List<String[]> allReviews = reviewRepository.loadAllReviews();
        int reviewCount = 0;

        for(String[] line : allReviews) {
            if (line.length == 0) {
                continue; // Skip empty lines
            }
            if (line[0].startsWith("reservationId")) {
                System.out.println("HEADER || " + String.join(" | ", line));
                continue; // Skip adding header to reviews list
            }
            reviews.add(line);
            reviewCount++;
            System.out.println(reviewCount + " || " + String.join(" | ", line));
```

```
        }
        if (reviews.size() == 0) {
            System.out.println("No reviews available.");
            return;
        }
        replytoReviews();
        reviews.clear();
    }

    private void replytoReviews() {
        System.out.println("\nTotal Reviews: " + reviews.size());
        System.out.println("-----------------------------------------");
        System.out.println("Select which one to review (1-" + (reviews.size()) + "): ");
        System.out.println("Enter 0 to exit.");

        Scanner sc = new Scanner(System.in);
        int choice;
        if (sc.hasNextInt()) {
            choice = sc.nextInt();
        } else {
            System.out.println("Invalid input. Please enter a number.");
            sc.next(); // consume invalid input
            return;
        }
        sc.nextLine();
        if (choice == 0) {
            System.out.println("Exiting...");
            return;
        }
        if (choice < 1 || choice > reviews.size()) {
            System.out.println("Invalid choice.");
            return;
        }
        System.out.println("Enter your reply: ");
        String reply = sc.nextLine();

        // Get the selected review
        String[] selectedReview = reviews.get(choice - 1);
        Reservation reservationId = new
Reservation(selectedReview[0],selectedReview[1],Integer.parseInt(selectedReview[2]),
LocalDate.parse(selectedReview[3]),LocalDate.parse(selectedReview[3]));
```

```
        System.out.println("------------------");
        System.out.println("Replying to: \n" + selectedReview[1] + ": " +  selectedReview[5]);
        System.out.println("Your Reply: " + reply);
        System.out.println("------------------");

        // Use repository to save the reply
        reviewRepository.addHRResponse(reservationId, this.getName(), reply);
    }

    @Override
    public void hireWorkers(JobApplication application) {
        // Implementation for hiring workers
        System.out.println("Hiring new workers...");
    }

    @Override
    public void fireWorkers(Employee employee) {
        // Implementation for firing workers
        System.out.println("Firing workers...");
    }

    @Override
    public void massiveLayoffs() {
        // Implementation for massive layoffs
        System.out.println("Conducting massive layoffs...");
    }

    @Override
    public void viewNotifications() {
        // Implementation for viewing notifications
        System.out.println("Viewing notifications...");
    }

}
```

Login for Workers Use Case (Pranava)

EmployeeLoginService.java:

```
package Main.Controller;
```

```
import Main.Data.DataRepository;
import Main.Employee.*;
import Main.Main;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class EmployeeLoginService {

    private static DataRepository repo;

    public static void initialize(DataRepository repository) {
        repo = repository;
    }

    public static Employee login(String username, String password) throws
FileNotFoundException {
        File file = new File(Main.HOTEL_PATH + "/" + "Employee.txt");

        if (!file.exists()) {
            System.out.println("Employee file not found for hotel: " + Main.HOTEL_PATH);
            return null;
        }

        Scanner sc = new Scanner(file);

        while (sc.hasNextLine()) {
            String[] tokens = sc.nextLine().split("\\s+");
            if (tokens.length < 6) continue;

            int id = Integer.parseInt(tokens[0]);
            String fullName = tokens[1] + " " + tokens[2];
            String role = tokens[3];
            String storedUsername = tokens[4];
            String storedPassword = tokens[5];

            if (storedUsername.equals(username) && storedPassword.equals(password)) {
                // Build correct employee class
                switch (role) {
                    case "FrontDesk":
                        return new frontdeskteam(id, fullName);
                    case "Cleaner":
```

```
                return new Housekeeping(id, fullName);
            case "Kitchen":
                return new KitchenPanel(id, fullName);
            case "Executive":
                return new ExecutivePanel(id, fullName, role);
            case "DataTeam":
                return new DataPanel(id, fullName, role);
            case "HR":
                return new HRPanel(id, fullName, role);
        }
      }
    }

    return null;
  }

  public static Employee createEmployee(String name, String role,
                                        String username, String password) {

    int newId = repo.getNextEmployeeId();

    Employee e = new Employee(newId, name, role, username, password);

    boolean saved = repo.addEmployee(e);

    if (!saved) return null;

    return e;
  }
}
```

PasswordUtils.java:

```
package Main.Controller;

public class PasswordUtils {
  public static boolean isValid(String password) {
    if (password.length() < 8) return false;
    if (!password.matches(".*[A-Z].*")) return false;       // uppercase
    if (!password.matches(".*[a-z].*")) return false;       // lowercase
    if (!password.matches(".*\\d.*")) return false;         // digit
    if (!password.matches(".*[!@#$%^&*()_+\\-={}:;<>?,.].*")) return false; // special
char
```

```
        return true;
    }

    public static String strength(String password) {
        int score = 0;

        if (password.length() >= 8) score++;
        if (password.length() >= 12) score++;
        if (password.matches(".*[A-Z].*")) score++;
        if (password.matches(".*[a-z].*")) score++;
        if (password.matches(".*\\d.*")) score++;
        if (password.matches(".*[!@#$%^&*()_+\\-={}:;<>?,.].*")) score++;

        if (score <= 2) return "Weak";
        if (score <= 4) return "Moderate";
        return "Strong";
    }
}
```

WorkerLoginController.java

```
package Main.Controller;

import java.io.FileNotFoundException;
import java.util.Random;
import java.util.Scanner;
import Main.Employee.Employee;

public class WorkerLoginController {

    public static Employee loginWorker(Scanner sc) throws FileNotFoundException {

        System.out.println("\n===== WORKER LOGIN =====");

        System.out.print("Username: ");
        String username = sc.nextLine().trim();

        System.out.print("Password: ");
        String password = sc.nextLine().trim();

        Employee emp = EmployeeLoginService.login(username, password);
```

```
        if (emp == null) {
            System.out.println("Invalid username or password.");
            System.out.println("===== WORKER LOGIN UNSUCCESSFUL =====");
            return null;
        }

        System.out.println("✔ Login successful. Welcome, " + emp.getName() +
                " (" + emp.role() + ")");
        System.out.println("===== WORKER LOGIN SUCCESSFUL =====");
        return emp;
    }

    // ============================================
    // NEW: REGISTER NEW WORKER
    // ============================================
    public static void registerNewWorker(Scanner sc) {
        System.out.println("\n===== CREATE NEW WORKER =====");

        System.out.print("Full Name: ");
        String name = sc.nextLine().trim();

        System.out.print("Role (FrontDesk / Cleaner / Kitchen / Executive / DataTeam / HR):
");
        String role = sc.nextLine().trim();

        System.out.print("Username: ");
        String username = generateUsername(name);
        System.out.println(username);

        String password = "";

        while (true) {
            System.out.print("Password: ");
            password = sc.nextLine();

            String strength = PasswordUtils.strength(password);
            System.out.println("Password Strength: " + strength);

            if (!PasswordUtils.isValid(password)) {
                System.out.println("!! Password must be at least 8 characters and contain:");
                System.out.println("   - Uppercase letter");
```

```
            System.out.println("   - Lowercase letter");
            System.out.println("   - Digit");
            System.out.println("   - Special character (!@#$%^&* etc.)");
            continue;
        }

        System.out.println("Password accepted.");
        break;
    }

    Employee emp = EmployeeLoginService.createEmployee(
            name, role, username, password
    );

    if (emp != null) {
        System.out.println("Worker created successfully: ID " + emp.getId());
    } else {
        System.out.println("Failed to save worker.");
    }
    System.out.println("\n===== NEW WORKER CREATED =====");
  }

  public static String generateUsername(String fullName) {
    String[] parts = fullName.trim().toLowerCase().split("\\s+");
    Random r = new Random();

    StringBuilder base = new StringBuilder();

    if (parts.length == 1) {
        String name = parts[0];

        int pattern = r.nextInt(4); // 4 patterns for single-word
        switch (pattern) {
            case 0:
                base = new StringBuilder(slice(name, 5) + (100 + r.nextInt(900)));      //
rihanna341
                break;
            case 1:
                base = new StringBuilder(slice(name, 3) + slice(name, 2) + r.nextInt(99)); //
adele23
                break;
            case 2:
```

```
                base = new StringBuilder(slice(name, 4) + "x" + r.nextInt(9999));       //
platox889
                break;
            case 3:
                base = new StringBuilder(slice(name, 2) + slice(name, 3) + r.nextInt(1000)); //
rihanna → rirhan314
                break;
        }

        // ensure minimum length 8
        while (base.length() < 8) {
            base.append(r.nextInt(10));
        }

        return base.toString();
    }

    String first = parts[0];
    String middle = parts.length > 2 ? parts[1] : "";
    String last = parts[parts.length - 1];

    int pattern = r.nextInt(7); // 7 patterns
    switch (pattern) {
        case 0:
            base = new StringBuilder(slice(first, 4) + slice(last, 3));
            break;
        case 1:
            base = new StringBuilder(slice(first, 3) + slice(last, 4));
            break;
        case 2:
            base = new StringBuilder(slice(first, 3) + slice(middle, 1) + slice(last, 3));
            break;
        case 3:
            base = new StringBuilder(slice(first, 1) + slice(last, 3 + r.nextInt(3))); // 3–5
letters
            break;
        case 4:
            base = new StringBuilder(slice(last, 3) + slice(first, 3));
            break;
        case 5:
            base = new StringBuilder(slice(first, 4) + slice(last, 4));
            break;
```

```
            case 6:
                base = new StringBuilder(slice(first, 2) + slice(last, 4));
                break;
        }

        if (r.nextInt(100) < 40) {
            base.append(r.nextInt(100));
        }

        while (base.length() < 8) {
            base.append(r.nextInt(10));
        }

        return base.toString();
    }

    // ONLY helper allowed
    private static String slice(String s, int n) {
        if (s == null || s.isEmpty()) return "";
        return s.substring(0, Math.min(s.length(), n));
    }
}
```

Minor Modifications to Iteration 1 Use Case:

Booking.java (Pranava & Ozair)

```
package Main.Booking;

import Main.Room.room;
import java.io.*;
import java.time.LocalDate;
import java.util.*;
import Main.*;

public class Booking implements BookingInterface {
    private static String ROOM_FILE = Main.HOTEL_PATH+ "/" + "Room.txt";
    private static String RESERVATION_FILE = Main.HOTEL_PATH+ "/" + "Reservation.txt";
    private static String GUEST_FILE = Main.HOTEL_PATH+ "/" +"Guest.txt";

    // ==================== MAIN MENU ====================
    public static void handleBooking() {
        Scanner sc = new Scanner(System.in);
```

```
        // STEP 1: Hardcoded list of supported hotel cities
        List<String> hotelCities = new ArrayList<>();
        hotelCities.add("Chicago");
        hotelCities.add("DesMoines");

        System.out.println("\n===== SELECT HOTEL LOCATION =====");
        for (int i = 0; i < hotelCities.size(); i++) {
            System.out.println((i + 1) + ". " + hotelCities.get(i));
        }

        System.out.print("Type your city: ");
        String cityInput = sc.nextLine().trim().toLowerCase();

        String selectedCity = null;
        for (String c : hotelCities) {
            if (c.toLowerCase().equals(cityInput)) {
                selectedCity = c;
                break;
            }
        }

        if (selectedCity == null) {
            System.out.println("Invalid city. Returning to menu.");
            return;
        }

        Main.HOTEL_PATH = selectedCity;
        ROOM_FILE = Main.HOTEL_PATH+ "/" + "Room.txt";
        RESERVATION_FILE = Main.HOTEL_PATH+ "/" + "Reservation.txt";
        GUEST_FILE = Main.HOTEL_PATH+ "/" +"Guest.txt";

        System.out.println("Selected hotel: " + selectedCity);

        // STEP 3: Proceed with booking flow for that location
        Booking bookingSystem = new Booking();
        boolean running = true;

        while (running) {
            System.out.println("\n===== HOTEL BOOKING SYSTEM (" + selectedCity + ")
=====");
            System.out.println("1. Book a Room");
```

```
        System.out.println("2. Modify a Reservation");
        System.out.println("3. Cancel a Reservation");
        System.out.println("4. Exit");
        System.out.print("Select an option: ");

        String choice = sc.nextLine();

        if (choice.equals("1")) {
            bookingSystem.bookRoomFlow();
        } else if (choice.equals("2")) {
            bookingSystem.modifyReservationFlow();
        } else if (choice.equals("3")) {
            bookingSystem.cancelReservationFlow();
        } else if (choice.equals("4")) {
            System.out.println("Returning to main menu...");
            running = false;
        } else {
            System.out.println("Invalid choice. Please try again.");
        }
    }
}

// ====================== BOOK FLOW ======================
private void bookRoomFlow() {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter Guest Name: ");
    String guestName = sc.nextLine();

    System.out.print("Enter Guest ID: ");
    int guestId = Integer.parseInt(sc.nextLine());

    System.out.print("Enter Check-in Date (YYYY-MM-DD): ");
    LocalDate checkIn = LocalDate.parse(sc.nextLine());

    System.out.print("Enter Check-out Date (YYYY-MM-DD): ");
    LocalDate checkOut = LocalDate.parse(sc.nextLine());

    List<room> availableRooms = getAvailableRooms(checkIn, checkOut);

    if (availableRooms.isEmpty()) {
        System.out.println("No rooms available for the selected dates.");
```

```
            return;
        }

        // Map room types to available rooms
        Map<String, List<room>> typeToRooms = new LinkedHashMap<>();
        for (room r : availableRooms) {
            typeToRooms.computeIfAbsent(r.getType(), k -> new ArrayList<>()).add(r);
        }

        // Display room types
        System.out.println("\nAvailable Room Types:");
        int index = 1;
        List<String> roomTypeMenu = new ArrayList<>();
        for (Map.Entry<String, List<room>> entry : typeToRooms.entrySet()) {
            System.out.println(index + ". " + entry.getKey() + " (" + entry.getValue().size() + "
available)");
            roomTypeMenu.add(entry.getKey());
            index++;
        }

        // FIXED INPUT SECTION: accept number OR text
        System.out.print("\nSelect a room type to book (1-" + roomTypeMenu.size() + " or
type name): ");
        String choice = sc.nextLine().trim().toLowerCase();

        String selectedType = null;

        // Case 1: user typed a number
        if (choice.matches("\\d+")) {
            int selectedIndex = Integer.parseInt(choice);
            if (selectedIndex >= 1 && selectedIndex <= roomTypeMenu.size()) {
                selectedType = roomTypeMenu.get(selectedIndex - 1);
            }
        } else {
            // Case 2: user typed the room type name
            for (String type : roomTypeMenu) {
                if (type.toLowerCase().equals(choice)) {
                    selectedType = type;
                    break;
                }
            }
        }
```

```
        if (selectedType == null) {
            System.out.println("Invalid input.");
            return;
        }

        // Pick the first available room of that type
        room selectedRoom = typeToRooms.get(selectedType).get(0);
        selectedRoom.setStartDate(checkIn);
        selectedRoom.setEndDate(checkOut);

        if (createReservation(guestName, guestId, selectedRoom)) {
            System.out.println("\nReservation created successfully for a " + selectedType +
"!");
        } else {
            System.out.println("Could not create reservation.");
        }
    }

    // ====================== MODIFY FLOW ======================
    private void modifyReservationFlow() {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Reservation ID to modify: ");
        String reservationId = sc.nextLine();

        if (!modifyReservation(reservationId)) {
            System.out.println("Reservation not found or could not be modified.");
        } else {
            System.out.println("Reservation updated successfully!");
        }
    }

    // ====================== CANCEL FLOW ======================
    private void cancelReservationFlow() {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter Reservation ID to cancel: ");
        String reservationId = sc.nextLine();

        if (!cancelReservation(reservationId)) {
            System.out.println("Reservation not found or could not be cancelled.");
```

```
        } else {
            System.out.println("Reservation cancelled successfully!");
        }
    }

    // ====================== INTERFACE IMPLEMENTATION
======================
    @Override
    public boolean createReservation(String guestName, int guestId, room room) {
        try {
            // Generate a unique reservation ID
            String reservationId = UUID.randomUUID().toString();

            // Append reservation to Reservation.txt (without guestId)
            try (BufferedWriter bw = new BufferedWriter(new FileWriter(RESERVATION_FILE,
true))) {
                bw.write(reservationId + "," + guestName + "," + room.getRoomNumber() + "," +
room.getStartDate() + "," + room.getEndDate());
                bw.newLine();
            }

            // Add guest to Guest.txt if they are new
            addGuestIfNew(guestName, guestId);

            return true;
        } catch (IOException e) {
            System.out.println("Error creating reservation: " + e.getMessage());
            return false;
        }
    }

    @Override
    public boolean modifyReservation(String reservationId) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter new Check-in Date (YYYY-MM-DD): ");
        LocalDate newCheckIn = LocalDate.parse(sc.nextLine());

        System.out.print("Enter new Check-out Date (YYYY-MM-DD): ");
        LocalDate newCheckOut = LocalDate.parse(sc.nextLine());

        List<String> updatedReservations = new ArrayList<>();
```

```
        boolean found = false;

        try (BufferedReader br = new BufferedReader(new FileReader(RESERVATION_FILE)))
{
            String header = br.readLine();
            updatedReservations.add(header);

            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(",");
                if (parts[0].equals(reservationId)) {
                    line = parts[0] + "," + parts[1] + "," + parts[2] + "," + newCheckIn + "," +
newCheckOut;
                    found = true;
                }
                updatedReservations.add(line);
            }
        } catch (IOException e) {
            System.out.println("Error modifying reservation: " + e.getMessage());
            return false;
        }

        if (found) {
            try (BufferedWriter bw = new BufferedWriter(new FileWriter(RESERVATION_FILE)))
{
                for (String r : updatedReservations) {
                    bw.write(r);
                    bw.newLine();
                }
            } catch (IOException e) {
                System.out.println("Error saving modified reservation: " + e.getMessage());
                return false;
            }
        }

        return found;
    }

    @Override
    public boolean cancelReservation(String reservationId) {
        List<String> updatedReservations = new ArrayList<>();
        boolean found = false;
```

```
        try (BufferedReader br = new BufferedReader(new FileReader(RESERVATION_FILE)))
{
            String header = br.readLine();
            updatedReservations.add(header);

            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(",");
                if (parts[0].equals(reservationId)) {
                    found = true;
                    continue; // skip
                }
                updatedReservations.add(line);
            }
        } catch (IOException e) {
            System.out.println("Error cancelling reservation: " + e.getMessage());
            return false;
        }

        if (found) {
            try (BufferedWriter bw = new BufferedWriter(new FileWriter(RESERVATION_FILE)))
{
                for (String r : updatedReservations) {
                    bw.write(r);
                    bw.newLine();
                }
            } catch (IOException e) {
                System.out.println("Error updating reservations: " + e.getMessage());
                return false;
            }
        }

        return found;
    }

    @Override
    public boolean checkAvailability(room room) {
        LocalDate start = room.getStartDate();
        LocalDate end = room.getEndDate();
        return isRoomAvailable(room.getRoomNumber(), start, end);
    }
```

```
    // ====================== HELPER METHODS ======================
    private List<room> getAvailableRooms(LocalDate requestedStart, LocalDate
requestedEnd) {
        List<room> availableRooms = new ArrayList<>();

        try (BufferedReader br = new BufferedReader(new FileReader(ROOM_FILE))) {
            br.readLine(); // skip header

            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(",");
                int roomNumber = Integer.parseInt(parts[0]);
                String type = parts[1];

                if (isRoomAvailable(roomNumber, requestedStart, requestedEnd)) {
                    availableRooms.add(new room(null, roomNumber, type, "Available", null,
null));
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading rooms: " + e.getMessage());
        }

        return availableRooms;
    }

    private boolean isRoomAvailable(int roomNumber, LocalDate requestedStart, LocalDate
requestedEnd) {
        try (BufferedReader br = new BufferedReader(new FileReader(RESERVATION_FILE)))
{
            br.readLine(); // skip header

            String line;
            while ((line = br.readLine()) != null) {
                String[] parts = line.split(",");
                if (Integer.parseInt(parts[2]) == roomNumber) {
                    LocalDate existingStart = LocalDate.parse(parts[3]);
                    LocalDate existingEnd = LocalDate.parse(parts[4]);

                    if (!requestedEnd.isBefore(existingStart) &&
!requestedStart.isAfter(existingEnd)) {
```

```
                    return false;
                }
            }
        }
    } catch (IOException e) {
        System.out.println("Error reading reservations: " + e.getMessage());
    }

    return true;
}

private void addGuestIfNew(String guestName, int guestId) {
    boolean exists = false;

    try (BufferedReader br = new BufferedReader(new FileReader(GUEST_FILE))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] parts = line.split(",");
            if (parts.length >= 2 && parts[0].trim().equalsIgnoreCase(guestName) &&
Integer.parseInt(parts[1].trim()) == guestId) {
                exists = true;
                break;
            }
        }
    } catch (IOException ignored) {
    }

    if (!exists) {
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(GUEST_FILE, true))) {
            bw.write(guestName + "," + guestId);
            bw.newLine();
        } catch (IOException e) {
            System.out.println("Error writing guest: " + e.getMessage());
        }
    }
}
}
```

File used by everyone:

Main.java:

```
package Main;

import java.io.FileNotFoundException;
import java.nio.file.Paths;
import java.util.*;
import java.io.File;
import Main.Booking.Booking;
import Main.Booking.Reservation;
import Main.Booking.ReviewProcess;
import Main.Controller.*;
import Main.Data.DataRepository;
import Main.Data.FileDataRepository;
import Main.Employee.*;
import Main.Guest.*;
import Main.Room.room;

public class Main {
    public static String HOTEL_PATH = "";

    public static void main(String[] args) throws FileNotFoundException,
InterruptedException {
        Random random = new Random();
        int randomBannerNumber  = random.nextInt(4) + 1;
        switch (randomBannerNumber) {
            case 1:
                printBanner1();
                break;
            case 2:
                printBanner2();
                break;
            case 3:
                printBanner3();
                break;
            case 4:
                printBanner4();
                break;
        }
        Scanner sc = new Scanner(System.in);
        boolean running = true;
```

```
while (running) {
    System.out.println("====================================");
    System.out.println("  Welcome to the HOTEL OAK STONE");
    System.out.println("====================================");
    System.out.print("Are you a Guest or a Worker? (G/W) or Q to quit: ");
    String userType = sc.nextLine().trim().toUpperCase();

    if (userType.equals("G")) {
        runGuestPortal(sc);
    }
    else if (userType.equals("W")) {
        System.out.print("Enter the Hotel Address you're working at: ");
        String hotelAddress = sc.nextLine().trim();
        HOTEL_PATH = hotelAddress.isEmpty() ? "123 Main St, Iowa" : hotelAddress;

        // Initialize repository for this hotel
        DataRepository repo = new FileDataRepository(Paths.get(HOTEL_PATH));
        EmployeeLoginService.initialize(repo);

        boolean workerMenu = true;

        while (workerMenu) {
            System.out.println("\n===== WORKER ACCESS =====");
            System.out.println("1. Login");
            System.out.println("0. Back");
            System.out.print("Select: ");

            String choice = sc.nextLine().trim();

            if (choice.equals("1")) {
                Employee worker = WorkerLoginController.loginWorker(sc);
                if (worker != null) {
                    initializeEmployeeManagers();
                    runWorkerPortal(sc, HOTEL_PATH, worker);
                }
            }
            else if (choice.equals("0")) {
                workerMenu = false;
            }
            else {
                System.out.println("Invalid option.");
```

```
                }
            }
        }

        else if (userType.equals("Q")) {
            System.out.println("Exiting system... Goodbye!");
            running = false;
        } else {
            System.out.println("Invalid selection. Please choose Guest (G), Worker (W), or
Quit (Q).");
        }
    }

    sc.close();

  }

  private static void initializeEmployeeManagers() throws FileNotFoundException {
    File file = new File(Main.HOTEL_PATH + "/" + "Employee.txt");
    Scanner scnr = new Scanner(file);

    frontdeskteam FDManager = null;
    Housekeeping HKManager = null;

    while (scnr.hasNextLine()) {
        String line = scnr.nextLine();
        String[] e = line.split(" ");
        if (e.length < 4) continue;

        int id = Integer.parseInt(e[0]);

        // role is always third-from-last
        String role = e[e.length - 3];

        // reconstruct name from tokens between ID and role
        StringBuilder nameBuilder = new StringBuilder();
        for (int i = 1; i < e.length - 3; i++) {
            nameBuilder.append(e[i]);
            if (i < e.length - 4) nameBuilder.append(" ");
        }

        String name = nameBuilder.toString();
```

```
            if (role.equals("FrontDesk")) {
                FDManager = new frontdeskteam(id, name);
            } else if (role.equals("Cleaner")) {
                HKManager = new Housekeeping(id, name);
            }
        }

        if (FDManager != null && HKManager != null) {
            FDManager.addHouseKeepingManager(HKManager);
        } else {
            System.out.println("WARNING: Could not link managers.");
        }
    }

    private static void runGuestPortal(Scanner sc) throws FileNotFoundException,
InterruptedException {
        boolean running = true;

        while (running) {
            System.out.println("\n╔═══════════════════════════════════════════════╗");
            System.out.println("║          GUEST PORTAL                 ║");
            System.out.println("╠═══════════════════════════════════════════════╣");
            System.out.println("║  1. Book a Room                       ║");
            System.out.println("║  2. Check In                          ║");
            System.out.println("║  3. Check Out                         ║");
            System.out.println("║  4. Access Room (Key Card)            ║");
            System.out.println("║  5. Leave a review                    ║");
            System.out.println("║  0. Exit                              ║");
            System.out.println("╚═══════════════════════════════════════════════╝");
            System.out.print("Select an option: ");

            String choice = sc.nextLine().trim();

            if (choice.equals("1")) {
                Booking.handleBooking();
            } else if (choice.equals("2")) {
                System.out.println("Check In option selected.");
                CheckSystemController.RunCheckin(null);
            } else if (choice.equals("3")) {
                System.out.println("Check Out option selected.");
                CheckSystemController.RunCheckOut(null);
```

```
            } else if (choice.equals("4")) {
                handleRoomAccess(sc);
            } else if(choice.equals("5")){
                leaveReview(sc);
            } else if (choice.equals("0")) {
                System.out.println("Thank you for visiting our hotel. Goodbye!");
                running = false;
            } else {
                System.out.println("Invalid option. Please choose 0–4.");
            }
        }
    }

    // ============================================================
    // Worker Portal (Employees, Data Team, Executives, etc.)
    // ============================================================

    private static void runWorkerPortal(Scanner sc, String hotelAddress, Employee
loggedIn) throws FileNotFoundException, InterruptedException {
        System.out.println("Logged in as: " + loggedIn.getName() + " (" + loggedIn.role() +
")");

        boolean running = true;

        while (running) {
            printWorkerMenu(loggedIn);
            System.out.print("\nSelect an option: ");
            String choice = sc.nextLine().trim();
            String role = loggedIn.role();

            // Logout
            if (choice.equals("0")) {
                System.out.println("Logging out..." + loggedIn.getName() + " from " +
loggedIn.role() + " Team");
                running = false;
                loggedIn = null;
                continue;
            } else if(choice.equals("9")){
                EmployeePortal portal = new EmployeePortal(Main.HOTEL_PATH +
"/ClockData.txt");
                portal.runPortal(sc, loggedIn);
                continue; // ← go back to Worker Portal menu immediately
```

```
}

// Role-based handling
switch (role) {
    case "FrontDesk":
        if (choice.equals("1")) {
            frontdeskteam.processQueue(loggedIn.getName());
        } else {
            System.out.println("Invalid option for your role.");
        }
        break;

    case "CleaningStaff":
        if (choice.equals("1")) {
            Housekeeping.processCleaning(loggedIn.getName());
        } else {
            System.out.println("Invalid option for your role.");
        }
        break;

    case "Kitchen":
        if (choice.equals("1")) {
            KitchenPanel.processOrders(loggedIn.getName());
        } else {
            System.out.println("Invalid option for your role.");
        }
        break;

    case "DataTeam":
        if (choice.equals("1")) {
            DataTeamController.runDataAnalysis(loggedIn.getName());
        } else {
            System.out.println("Invalid option for your role.");
        }
        break;

    case "Executive":
        if (choice.equals("1")) {
            frontdeskteam.processQueue(loggedIn.getName());
        } else if (choice.equals("2")) {
            Housekeeping.processCleaning(loggedIn.getName());
        } else if (choice.equals("3")) {
```

```
                KitchenPanel.processOrders(loggedIn.getName());
            } else if (choice.equals("4")) {
                ExecutiveController.runExecutivePanel(loggedIn.getName());
            } else {
                System.out.println("Invalid option for your role.");
            }
            break;
        case "HR":
            if (choice.equals("1")) {
                HRController.runHRPanel(loggedIn.getName());
            }
            else if (choice.equals("2")) {
                WorkerLoginController.registerNewWorker(sc);
            }
            else {
                System.out.println("Invalid option for your role.");
            }
            break;

        default:
            System.out.println("Unknown role.");
    }
  }
}

private static void printWorkerMenu(Employee user) {
  String role = user.role();

  System.out.println("╔═══════════════════════════════════════════════════════╗");
  System.out.println("║          WORKER PORTAL                ║");
  System.out.println("╠═══════════════════════════════════════════════════════╣");

  switch (role) {
    case "FrontDesk":
      System.out.println("║  1. Front Desk Panel                  ║");
      break;
    case "CleaningStaff":
      System.out.println("║  1. Cleaning Panel                    ║");
      break;
    case "Kitchen":
      System.out.println("║  1. Kitchen Panel                     ║");
      break;
```

```
            case "DataTeam":
                System.out.println("╟  1. Data Team Panel                  ╢");
                break;
            case "Executive":
                System.out.println("╟  1. Front Desk Panel                 ╢");
                System.out.println("╟  2. Cleaning Panel                   ╢");
                System.out.println("╟  3. Kitchen Panel                    ╢");
                System.out.println("╟  4. Executive Panel                  ╢");
                break;
            case "HR":
                System.out.println("╟  1. HR Panel                         ╢");
                System.out.println("╟  2. Register New Worker              ╢");
                break;
        }

        System.out.println("╟  0. Logout                           ╢");
        System.out.println("╟  9. Clock In/Out                     ╢");
        System.out.println("╚═══════════════════════════════════════════════════════════════════════════╝");
    }

    private static void handleRoomAccess(Scanner sc) {
        System.out.print("Enter the hotel location you're accessing (e.g., Chicago,
DesMoines): ");
        String hotel = sc.nextLine().trim();

        if (hotel.isEmpty()) {
            System.out.println(“ No location entered. Returning to menu.");
            return;
        }

        Main.HOTEL_PATH = hotel;
        new RoomAccessFlow(sc).execute();
    }

    private static void leaveReview(Scanner sc){
        System.out.print("Enter the hotel location you're accessing (e.g., Chicago,
DesMoines): ");
        String hotel = sc.nextLine().trim();

        if (hotel.isEmpty()) {
            System.out.println("No location entered. Returning to menu.");
            return;
```

```
        }

        Main.HOTEL_PATH = hotel;

        System.out.println("What's your name?");
        String name = sc.nextLine().trim();
        System.out.println("What's your reservation ID?");
        String resId = sc.nextLine().trim();
        Reservation reservation = new Reservation(name, resId);
        if (reservation.getStatus()) {
            System.out.println("I have located your booking! You stayed with us until " +
reservation.getEndDate());
            ReviewProcess reviewProcess = new ReviewProcess(reservation);
            reviewProcess.execute(sc);
        }
    }

    private static void printBanner1() {
        System.out.println();
        System.out.println("    $$$$$$\\  $$$$$$\\  $$\\   $$\\ $$$$$$\\ $$$$$$$$\\
$$$$$$\\  $$\\   $$\\ $$$$$$$$\\");
        System.out.println("   $$ __$$\\ $$ __$$\\ $$ | $$  |$$ __$$\\\\__$$ __|$$ __$$\\
$$$\\  $$ |$$  ____|");
        System.out.println("   $$ /  $$ |$$ /  $$ |$$ |$$  / $$ /  \\__|  $$ |   $$ /  $$ |$$$$\\
$$ |$$ |      ");
        System.out.println("   $$ |  $$ |$$$$$$$$ |$$$$$  /  \\$$$$$$\\    $$ |   $$ |  $$ |$$
$$\\$$ |$$$$$\\    ");
        System.out.println("   $$ |  $$ |$$  __$$ |$$  $$<    \\____$$\\   $$ |   $$ |  $$ |$$
\\$$$$ |$$  __|");
        System.out.println("   $$ |  $$ |$$ |  $$ |$$ |\\$$\\  $$\\   $$ |  $$ |   $$ |  $$ |$$
|\\$$$ |$$ |");
        System.out.println("    $$$$$$  |$$ |  $$ |$$ | \\$$\\ \\$$$$$$  |  $$ |    $$$$$$  |$$
| \\$$ |$$$$$$$$\\");
        System.out.println("    \\______/ \\__|  \\__|\\__|  \\__| \\______/   \\__|    \\______/ \\__|
\\__|\\________|");
        System.out.println("\n                    O   A   K   S   T   O   N   E   \n
H   O   T   E   L\n");
    }

    private static void printBanner2() {
        System.out.println();
```

```
    System.out.println("
==========================================================================
====================");
    System.out.println("  ===  ======= ===== ==== ===   ===    ====  ====
======= ==     =");
    System.out.println("  == == =====  ==== === === ==== ===== ====== ==
===  ====== == =======");
    System.out.println("  = ==== === == === == ==== ==== ===== =====
==== ==  ===== == =======");
    System.out.println("  = ==== == ==== == = ====== ========== =====
==== == == === == =======");
    System.out.println("  = ==== == ==== ==   ======== ======== =====
==== == === == ==   ===");
    System.out.println("  = ==== ==    == == ========= ====== ===== ====
== ==== = == =======");
    System.out.println("  = ==== == ==== == === === ==== ===== =====
==== == =====  == =======");
    System.out.println("  == == === ==== == ==== == ==== ===== ====== ==
=== ======  == =======");
    System.out.println("  ===  ==== ==== == ==== ===   ====== =======
==== ======= ==     =");
    System.out.println("
==========================================================================
====================");
    System.out.println("\n               O  A  K  S  T  O  N  E  \n
H  O  T  E  L\n");
  }

  private static void printBanner3() {
    System.out.println();
    System.out.println("
:'#######:::::'###::::'##:::'##::'######::'########::'#######::'##::: ##:'########:");
    System.out.println("  '##.... ##:::'## ##::: ##::'##::'##... ##:... ##..::'##.... ##: ###::
##: ##.....::");
    System.out.println("   ##:::: ##::'##:. ##:: ##:'##::: ##:::..::::: ##:::: ##:::: ##: ####: ##:
##:::::::");
    System.out.println("   ##:::: ##:'##:::. ##: #####:::. ######::::: ##:::: ##:::: ##: ## ##
##: ######:::");
    System.out.println("   ##:::: ##: #########: ##. ##:::..... ##:::: ##:::: ##:::: ##: ##.
####: ##...::::");
    System.out.println("   ##:::: ##: ##.... ##: ##:. ##::'##::: ##:::: ##:::: ##:::: ##: ##:.
###: ##:::::::");
```

```
        System.out.println("    . #######:: ##:::: ##: ##::. ##:. ######::::: ##::::. #######::
##::. ##: ########:");
        System.out.println("   :.....:::..:::::..::..::::..:::.....::::::..::::::.....:::..:::::..::.....::");987
        System.out.println("\n                    O   A   K   S   T   O   N   E   \n
H   O   T   E   L\n");
    }

    private static void printBanner4() {
        System.out.println();
        System.out.println("
");
        System.out.println("
");
        System.out.println("        OOOOOOOOO              AAA             KKKKKKKKK
KKKKKKK   SSSSSSSSSSSSSSS TTTTTTTTTTTTTTTTTTTTTTT     OOOOOOOOO
NNNNNNNN        NNNNNNNNEEEEEEEEEEEEEEEEEEEEE");
        System.out.println("       OO:::::::::OO            A:::A            K:::::::K    K:::::K
SS:::::::::::::::ST:::::::::::::::::::::T   OO:::::::::OO   N:::::::N       N::::::NE::::::::::::::::::::E");
        System.out.println("     OO:::::::::::::OO          A:::::A           K:::::::K
K:::::KS:::::SSSSSS::::::ST:::::::::::::::::::::T OO:::::::::::::OO N::::::::N      N::::::NE::::::::::::::::::::E");
        System.out.println("    O:::::::OOO:::::::O        A:::::::A          K:::::::K   K::::::KS:::::S
SSSSSSST:::::TT:::::::TT:::::TO:::::::OOO:::::::ON::::::::N     N::::::NEE::::::EEEEEEEEE::::E");
        System.out.println("    O::::::O   O::::::O       A:::::::::A         KK::::::K  K:::::KKKS:::::S
TTTTTT  T:::::T  TTTTTTO::::::O   O::::::ON:::::::::N    N::::::N  E:::::E       EEEEEE");
        System.out.println("    O:::::O     O:::::O      A:::::A:::::A          K:::::K K:::::K   S:::::S
T:::::T       O:::::O     O:::::ON:::::::::::N   N::::::N  E:::::E             ");
        System.out.println("    O:::::O     O:::::O     A:::::A A:::::A         K::::::K:::::K     S::::SSSS
T:::::T       O:::::O     O:::::ON:::::::N::::N  N::::::N  E::::::EEEEEEEEEE   ");
        System.out.println("    O:::::O     O:::::O     A:::::A   A:::::A        K:::::::::::K
SS::::::SSSSS          T:::::T       O:::::O     O:::::ON::::::N N::::N N::::::N  E::::::::::::::E   ");
        System.out.println("    O:::::O     O:::::O    A:::::A     A:::::A       K:::::::::::K
SSS::::::::SS         T:::::T       O:::::O     O:::::ON::::::N  N::::N:::::::N  E::::::::::::::E   ");
        System.out.println("    O:::::O     O:::::O   A:::::AAAAAAAAA:::::A       K::::::K:::::K
SSSSSS::::S        T:::::T       O:::::O     O:::::ON::::::N   N:::::::::::N  E::::::EEEEEEEEEE   ");
        System.out.println("    O:::::O     O:::::O  A:::::::::::::::::::::A      K:::::K K:::::K
S:::::S       T:::::T       O:::::O     O:::::ON::::::N    N::::::::::N  E:::::E             ");
        System.out.println("    O::::::O   O::::::O A:::::AAAAAAAAAAAAA:::::A    KK::::::K
K:::::KKK          S:::::S       T:::::T       O::::::O   O::::::ON::::::N     N:::::::::N  E:::::E
EEEEEE");
        System.out.println("    O:::::::OOO:::::::OA:::::A           A:::::A   K:::::::K   K::::::KSSSSSSS
S:::::S      TT:::::::TT      O:::::::OOO:::::::ON::::::N      N::::::::NEE::::::EEEEEEEE:::::E");
```

```
      System.out.println("     OO:::::::::::::OOA:::::A              A:::::A  K:::::::K
K:::::KS::::::SSSSSS:::::S      T:::::::::T       OO:::::::::::::OO N::::::N       N:::::::NE::::::::::::::::::::E");
      System.out.println("       OO:::::::::OO A:::::A               A:::::A K:::::::K
K:::::KS:::::::::::::::SS       T:::::::::T         OO:::::::::OO   N::::::N       N::::::NE::::::::::::::::::::E");
      System.out.println("         OOOOOOOOO  AAAAAAA
AAAAAAAKKKKKKKKK    KKKKKKK SSSSSSSSSSSSSSS        TTTTTTTTTTT
OOOOOOOOO     NNNNNNNN        NNNNNNNNEEEEEEEEEEEEEEEEEEEEE");
      System.out.println("
");
      System.out.println("
");
      System.out.println("\n                                                              O   A   K   S   T
O   N   E   \n                                                        H   O   T   E   L\n");
   }


   private static final class RoomAccessFlow {
      private static final String DATA_ROOT = "Backend/Hotel/untitled/src/Main/";
      private static final String FALLBACK_ROOT = "src/Main/";
      private final Scanner input;

      RoomAccessFlow(Scanner input) {
         this.input = input;
      }

      void execute() {
         System.out.println("===== ROOM ACCESS AUTHENTICATION =====");

         List<KeyCard> issuedCards = loadIssuedKeyCards();

         if (issuedCards.isEmpty()) {
            System.out.println("No key cards are currently issued. Please visit the front
desk.");
            return;
         }

         System.out.print("Room number: ");
         int roomNumber = parseNumber(input.nextLine().trim());

         System.out.print("Name on key card: ");
         String guestName = input.nextLine().trim();
```

```
        if (roomNumber <= 0 || guestName.isEmpty()) {
            System.out.println("Access denied: incomplete information.");
            return;
        }

        Optional<KeyCard> keyCard = findMatchingCard(issuedCards, roomNumber,
guestName);

        if (keyCard.isEmpty()) {
            System.out.println(RoomAccessResult.INVALID_GUEST.getMessage());
            return;
        }

        Optional<room> targetRoom = loadRoom(roomNumber);

        if (targetRoom.isEmpty()) {
            System.out.println(RoomAccessResult.SYSTEM_ERROR.getMessage());
            return;
        }

        Guest guest = new Guest(guestName, -1);
        RoomAccessResult result = guest.scanKeyCard(keyCard.get(), targetRoom.get());

        System.out.println(result.getMessage());

        if (result == RoomAccessResult.ACCESS_GRANTED) {
            GuestSession.roomAccess(roomNumber, guestName, targetRoom.get());
        }
    }

    private List<KeyCard> loadIssuedKeyCards() {
        File source = resolveDataFile(Main.HOTEL_PATH +"/"+"Keycard.txt");
        List<KeyCard> cards = new ArrayList<>();

        if (!source.exists()) return cards;

        try (Scanner scanner = new Scanner(source)) {
            while (scanner.hasNextLine()) {
                String[] tokens = scanner.nextLine().trim().split("\\s+");

                if (tokens.length >= 3) {
                    int recordedRoom = parseNumber(tokens[0]);
```

```
                boolean isActive = Boolean.parseBoolean(tokens[1]);
                String ownerName = rebuildName(tokens);

                cards.add(new KeyCard(recordedRoom, isActive, new Guest(ownerName,
-1)));
            }
          }
        } catch (FileNotFoundException e) {
          cards.clear();
        }

        return cards;
    }

    private Optional<KeyCard> findMatchingCard(List<KeyCard> cards, int roomNumber,
String guestName) {
        return cards.stream()
                .filter(c -> c.getRoomnumber() == roomNumber &&
c.getOwner().getName().equalsIgnoreCase(guestName))
                .findFirst();
    }

    private Optional<room> loadRoom(int roomNumber) {
        File source = resolveDataFile(Main.HOTEL_PATH +"/"+"Room.txt");

        if (!source.exists()) return Optional.empty();

        try (Scanner scanner = new Scanner(source)) {
          if (scanner.hasNextLine()) scanner.nextLine();

          while (scanner.hasNextLine()) {
            String[] parts = scanner.nextLine().split(",");

            if (parts.length >= 2) {
              int recordedRoom = parseNumber(parts[0].trim());

              if (recordedRoom == roomNumber) {
                String type = parts[1].trim();
                room entry = new room(Main.HOTEL_PATH, recordedRoom, type,
"Reserved", null, null);
                entry.lock();
                return Optional.of(entry);
```

```
                }
            }
        }
    } catch (FileNotFoundException e) {
        return Optional.empty();
    }

    return Optional.empty();
}

private File resolveDataFile(String fileName) {
    String[] prefixes = {"", DATA_ROOT, FALLBACK_ROOT, "Backend/Hotel/untitled/",
"Backend/Hotel/"};

    for (String prefix : prefixes) {
        File candidate = prefix.isEmpty() ? new File(fileName) : new File(prefix +
fileName);
        if (candidate.exists()) return candidate;
    }

    return new File(DATA_ROOT + fileName);
}

private int parseNumber(String value) {
    try {
        return Integer.parseInt(value);
    } catch (NumberFormatException ex) {
        return -1;
    }
}

private String rebuildName(String[] tokens) {
    return String.join(" ", Arrays.copyOfRange(tokens, 2, tokens.length));
}
  }
}
```