Candidate: Abdallah DAHMOU

Question 1: Is putting all functions in the same service the recommended approach?

No, putting all functions inside the same Service class is not the recommended approach from a software design perspective. This approach violates the Single Responsibility Principle (SRP), which states that a class should have only one reason to change. Currently, the Service class is handling multiple responsibilities including room management, user management, booking management, and data presentation through printing functions. This creates a monolithic class that becomes difficult to maintain, test, and extend as the system grows.

I think a better approach would be to implement the Repository pattern or separate service classes for each domain entity.

For example, we could have a RoomService for room-related operations, a UserService for user management, a BookingService for reservation handling, and separate repository classes for data access. This would make the code more modular, easier to test with unit tests for each service, and more maintainable. Each service would have a clear, focused responsibility, and changes to one area of functionality wouldn't impact others. Additionally, this separation would make it easier to implement features like dependency injection, caching, or different data storage mechanisms in the future.

Question 2: Alternative to setRoom() not impacting previous bookings - recommendations and justification

The current design where setRoom() doesn't impact previous bookings is actually a good approach for maintaining data integrity and historical accuracy. However, there are alternative approaches worth considering. One alternative would be to allow room updates to affect existing bookings, automatically recalculating costs and updating user balances accordingly. Another approach would be to implement versioning for rooms, where each room change creates a new version while maintaining references to the version used at booking time.

However, I recommend keeping the current approach where setRoom() doesn't impact previous bookings, and here's my justification:

First, it maintains historical accuracy and audit trails, which is important for financial systems where users have already paid based on specific room rates.

Second, it prevents unexpected charges or refunds that could confuse users and create customer service issues.

Third, it ensures legal compliance, as changing the terms of completed transactions could raise contractual issues.

Fourth, it provides system stability by preventing cascading updates that could introduce bugs or data inconsistencies. If room updates need to affect future revenue calculations or

analytics, this can be handled through separate reporting mechanisms that account for room changes without modifying historical booking data. This will treat bookings as immutable contracts, which is the standard practice in most real-world reservation systems.