Hotel Booking System Documentation

1. Overview of the System Architecture

The Hotel Booking System is designed using the MVC (Model-View-Controller) architecture pattern, which divides the application into three interconnected components:

- Model: Manages the data and business logic, representing hotel rooms, bookings, pricing, and clients.
- View: Displays the UI components and responds to the user's interactions with the application.
- Controller: Manages the communication between the Model and View, interpreting user inputs to update the View or modify the Model.

This separation of concerns allows for easy scaling, testing, and maintenance, as each component can be independently modified without impacting the others directly.

---

2. Explanations of Each Implemented Design Pattern and Its Role in the Project

The system uses several design patterns, which enable flexible and efficient handling of the hotel booking process. Here is a list of the patterns with their roles:

2.1 Creational Patterns

- Singleton:
  - Usage: Ensures that only one instance of the DatabaseConnection class exists across the application.
  - Role: Manages the centralized database connection, improving performance and resource management by reusing the same connection instance.
- **Builder**:
  - **Usage**: Provides a step-by-step construction of Booking objects with optional fields such as additional services.
  - **Role**: Simplifies the creation of complex booking objects, enhancing code readability and flexibility in defining optional booking details.

2.2 Structural Patterns

- **Adapter**:
  - **Usage**: The PayPalAdapter integrates with the external payment service, converting the PaymentProcessor interface to be compatible with PayPal.
  - **Role**: Facilitates seamless payment processing with an external provider without changing the core system, allowing flexibility in adding new payment providers.
- **Facade**:
  - **Usage**: The HotelManagementFacade simplifies interactions with subsystems, such as displaying available rooms, booking management, and client interactions.

- **Role**: Reduces system complexity by providing a unified interface for booking functionalities, streamlining client code and reducing dependencies on internal subsystems.

2.3 Behavioral Patterns

- **Observer**:
  - **Usage**: The Order class notifies subscribed Client instances of booking status updates (e.g., confirmed, on the way).
  - **Role**: Improves client experience by providing real-time updates on booking status, enhancing interactivity.
- **Strategy**:
  - **Usage**: Different pricing strategies (DistanceBasedPricing, TimeBasedPricing, and FixedPricePricing) are implemented to calculate trip costs based on various factors.
  - **Role**: Enables flexible price calculation by switching between strategies without modifying the core code, accommodating different pricing models.