Observer Design Pattern

The Observer pattern is used to:

- Trigger downstream processes (side effects) after a specific event occurs.

- Enable real-time notifications, such as chat updates and booking status changes.

- Maintain immutable audit logs, ensuring that every significant event is recorded.

- Synchronize multiple modules without requiring them to call each other directly, reducing tight coupling.

Implementation Observer: Scheduling & Sessions:

This is the most critical area because session state changes drive many other system behaviors.

Key Events:

SessionRequested

SessionCounterOffered

SessionAccepted

SessionCompleted

SessionDisputed

Suitable Observers:

- **NotificationObserver**

Sends notifications to teachers and students when a session is requested, counter-offered, or accepted.

- **CalendarObserver**

Blocks time slots to prevent double booking or updates user availability.

- **ChatObserver**

Creates or activates the chat channel and enables video link exchange once a session is accepted.

- **AuditObserver**

Records every session status change for traceability and compliance.

- **Benefit:**

The core scheduling flow remains clean and focused, while all side effects are handled automatically and independently by observers.

Implementation in a Modular Monolith:

A simple Domain Event approach can be used:

- **Publisher**

The module that performs the core action (e.g., Scheduling module).

- **Event Bus (in-memory)**

Dispatches events within the monolithic application.

- **Subscribers / Observers**

Other modules such as Wallet, Notification, Audit, Chat, and Reputation that react to those events.

This approach preserves modular boundaries while enabling coordinated behavior across the system.