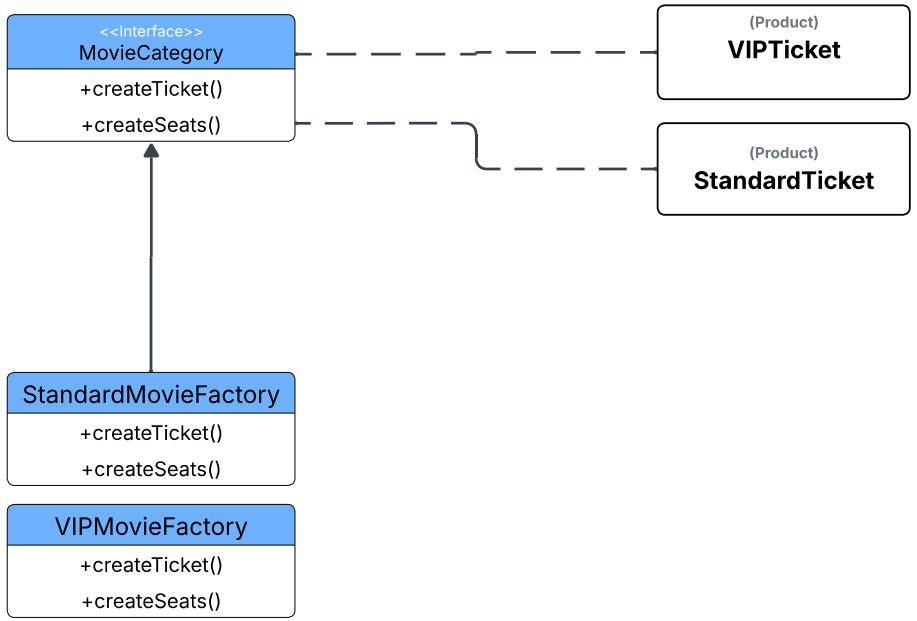
Selected Design Patterns & Justification

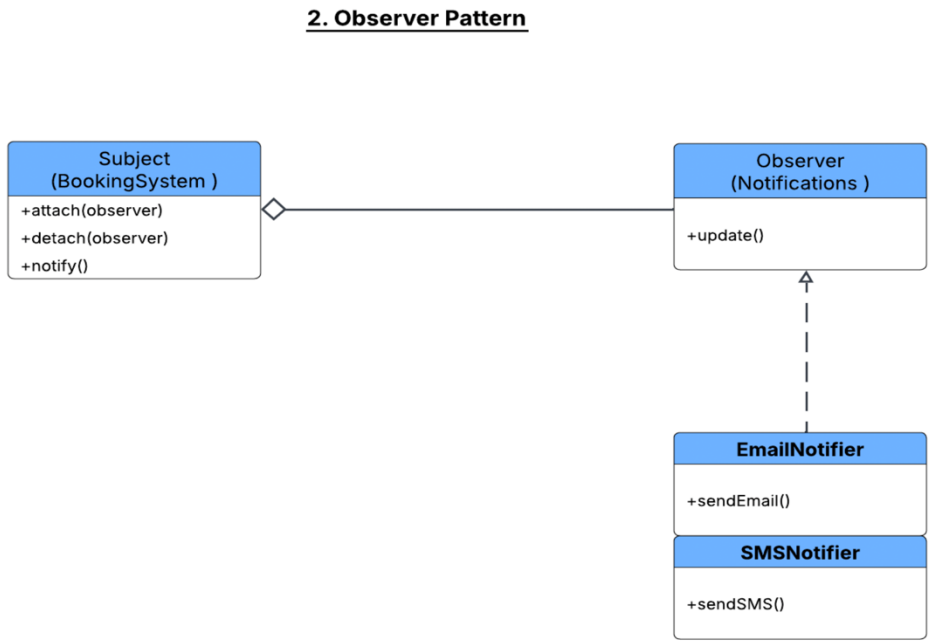
Pattern	Purpose	Justification	UML Diagram (Lucidchart Steps)	Deployment Diagram
Abstract Factory	Dynamically create movie categories (e.g., "Standard," "VIP," "3D") with different pricing/features.	Avoids hardcoding categories. Simplifies adding new types (e.g., "4DX") without modifying existing code.	**Component Diagram:**- Define **Factory interface** (MovieCategoryFactory).- Add **Concrete factories** (StandardMovieFactory, VIPMovieFactory).- Link to **Products** (Ticket, Seat).- Use **association arrows** from factories to products.	Factories deployed in the **backend (Node.js service)**, responsible for creating category instances.
Observer	Notify users about booking confirmations, payment status, and reminders.	Decouples notification logic from booking logic. Supports multiple channels (Email/SMS).	**Component Diagram:**- Define **Subject** (BookingSystem) with list of observers.- Add **Observers** (EmailNotifier, SMSNotifier).- Use **dependency arrows** to show observer updates via update() method.- Indicate event flow (e.g., booking triggers notification).	Observers run as **microservices** (e.g., AWS Lambda for SMS, Email microservice).
Decorator	Add optional features to tickets (e.g., "VIP (+$5)", "3D Glasses (+$3)").	Flexible pricing without subclass explosion. Features can be stacked (e.g., VIP + 3D).	**Component Diagram:**- Define **Base component** (BasicTicket).- Add **Decorators** (VIPDecorator, 3DDecorator) that **wrap** the base.- Show decorators implementing the same interface as the base.- Highlight stackability with layered arrows.	Decorators reside within the **booking service**, enriching base ticket instances.

Component Diagram:

1-Abstract Factory Pattern

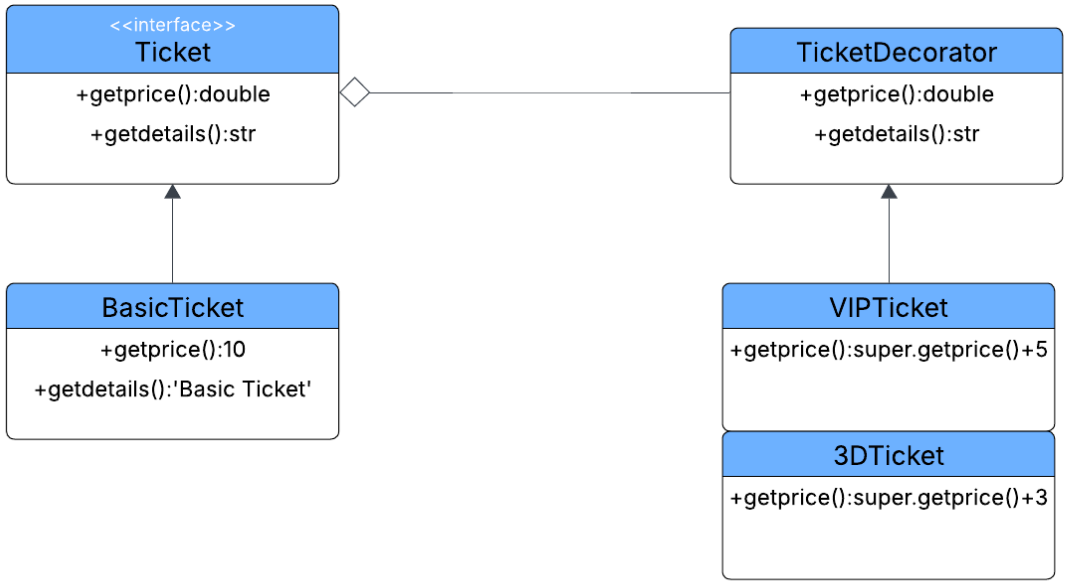


Component Diagram:



Component Diagram:

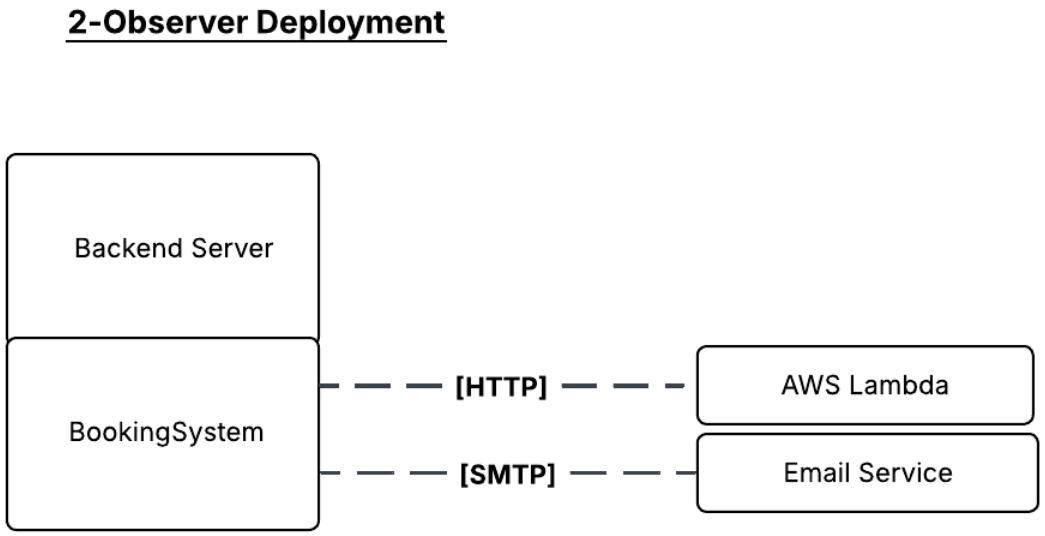
3. Decorator Pattern



Deployment Diagram:

1-Abstract Factory Deployment

Deployment Diagram:



Deployment Diagram: