Design Patterns:

1. **Model-View-Controller (MVC) Pattern:**
   The MVC pattern is a widely used architectural design pattern that separates an application into three interconnected components: Model, View, and Controller. In the context of our online ticketing app:

   - Model: Represents the data and business logic of the application. In the case of the ticketing app, this could include user information, event details, and transaction data.

   - View: Displays the user interface and interacts with the users. In the ticketing app, the view would be the web page where users browse events, select seats, make purchases, and sell their unwanted tickets. As well as where event organizers create and manage their event information and tickets.

   - Controller: Acts as an intermediary between the Model and View. It handles user input, processes requests, and updates the Model accordingly. For the ticketing app, the controller would manage actions like creating an event, processing a ticket purchase, and updating the user's transaction history

   When implemented, the backend of the ticketing app can be implemented using a server-side framework that follows the MVC pattern, such as Django (hence why we chose it in our techstack)

2. **Mediator pattern:**
   The mediator design pattern as its name suggests specializes in managing complex interactions between multiple independent components similar to the role of a control tower in an airport. In the case of our ticketing app, using the mediator pattern will allow us to manage the communication between different components without them being explicitly aware of each other. For example, the booking system, payment gateway, and user authentication module can communicate through a central mediator, reducing direct dependencies. Additionally the mediator pattern should simplify the addition of new components or functionalities without requiring changes to existing components.