Design Patterns for Sublime.

Requirement for M3:  
Describe at least two **design patterns** that will be implemented for your project. Clearly explain why your team has chosen those design patterns and how they will be implemented. You can add a class diagram for supporting your decision.  
  
Response:  
  
**Factory Method Pattern:** This pattern is useful for creating objects without specifying the exact class of object that will be created. It's particularly useful in our system for creating different types of user accounts (Administrator, Event Organizer, Customer) without binding our code to specific classes. This pattern promotes loose coupling and scalability, which is essential for managing various user roles with distinct permissions and functionalities.  
Chosen because it allows the system to introduce new user types in the future (like vendors, venue owners, etc.) without altering the existing code that creates users. It encapsulates the user creation process, making the system more maintainable and scalable.

**Observer Pattern:** This pattern is valuable for a system where changes to one object (subject) need to be reflected in multiple other objects (observers) without making the observers dependent on the subject. It's particularly applicable for notifying users (Event Organizers and Customers) about changes in event details, approval statuses, and ticket purchases. For example, when an event's details are updated and approved, all subscribed customers (observers) can receive notifications about the update.  
Selected because it efficiently manages notifications related to event updates, ticket purchases, and other real-time changes that need to be communicated to multiple users. This pattern ensures that the system remains flexible and loosely coupled, as it can easily add or remove observers (users) as needed without modifying the subject (event details, ticket inventory).

Uml Diagram for the Project