Design Patterns:

1) Strategy Pattern:
We have used the strategy design pattern for the item control classes. We will have one PosterController and CustomerController class for each user(poster or seller explicitly). Also, these two controller classes have an attribute, itemController. According to the use case of the poster or customer, the attribute switches between the following: SaleItemController, RentItemController, LostItemController, FoundItemController, PrivateLessonItemController and CourseItemController. This is helpful because in the run-time, a customer can search in any type of items or a poster can call a method related with the item type. With the attribute of the interface, switching the attribute is enough for handling all types of items. So, the strategy design pattern provides a solution which requires much less code. If we were not using it, we would have to consider all possible cases with if-else statements but now, just switching the attribute handles all the jobs. For example, if a customer wants to search a sale item, the attribute of CustomerController class becomes SaleItemController and when the searchItem() method is called, it works compatible with the SaleItemController(searchItem method calls createAndAddItemToDb() method for example, when the attribute is initialized to SaleItemController, the searchInItems method of SaleItemController is called).

2) MVC Pattern
Our project adopts MVC (model-view-controller) design pattern, which is essential for the maintainability of the Project since it allows us to differentiate between different logic and pinpoint certain errors of the Project. MVC design pattern separates our application into three interconnected components, each with its own responsibilities:

Model:

Represents the application's data.

Responsible for managing the data, logic, and rules of the application.

Responds to requests for information, updates its state, and notifies observers (typically views) about any changes.

View:

Represents the user interface and presentation of the application.

Displays information to the user and forwards user input to the controller.

Receives data from the model and renders it to the user.

Controller:

Acts as an intermediary between the model and the view.

Handles user input and updates the model accordingly.

Listens for events triggered by the view and updates the view or model as needed.

The key idea behind MVC is to separate the concerns of an application, making it easier to manage and maintain. Changes in one component (e.g., the user interface) do not directly affect the others (e.g., data management or business logic). This separation enhances modularity, flexibility, and scalability in software development. Additionally, MVC facilitates the development of different user interfaces for the same application logic.

For example; the models include, user model, chat model, message model, item model etc. Which define the model of the data which will be stored inside our databases. The structure of these models are fitted into the application fort he most practical and efficient data flow.

Controllers include; user controller, chat controller, item controller etc. Which essentially define how the modelized data will be able to communicate and move within the system. That is, e.g; the users will be able to send messages to each other, this functionality will be captured within the controller, which uses the models such as message and user.

In terms of view, the frontend will create the look of the application upon which the users will communicate with the model and controller parts of the application. View is essentially the boundary object from which the users are executing the given functionalities of the system. The execution in place, is abstracted away from the user, which is done with the explained controllers and models.