COMSATS UNIVERSITY ISLAMABAD ABBOTTABAD CAMPUS

SOFTWARE DESIGN AND ARCITECTURE

Submitted to: Sir Mukhtiar Zamin

LAB MID

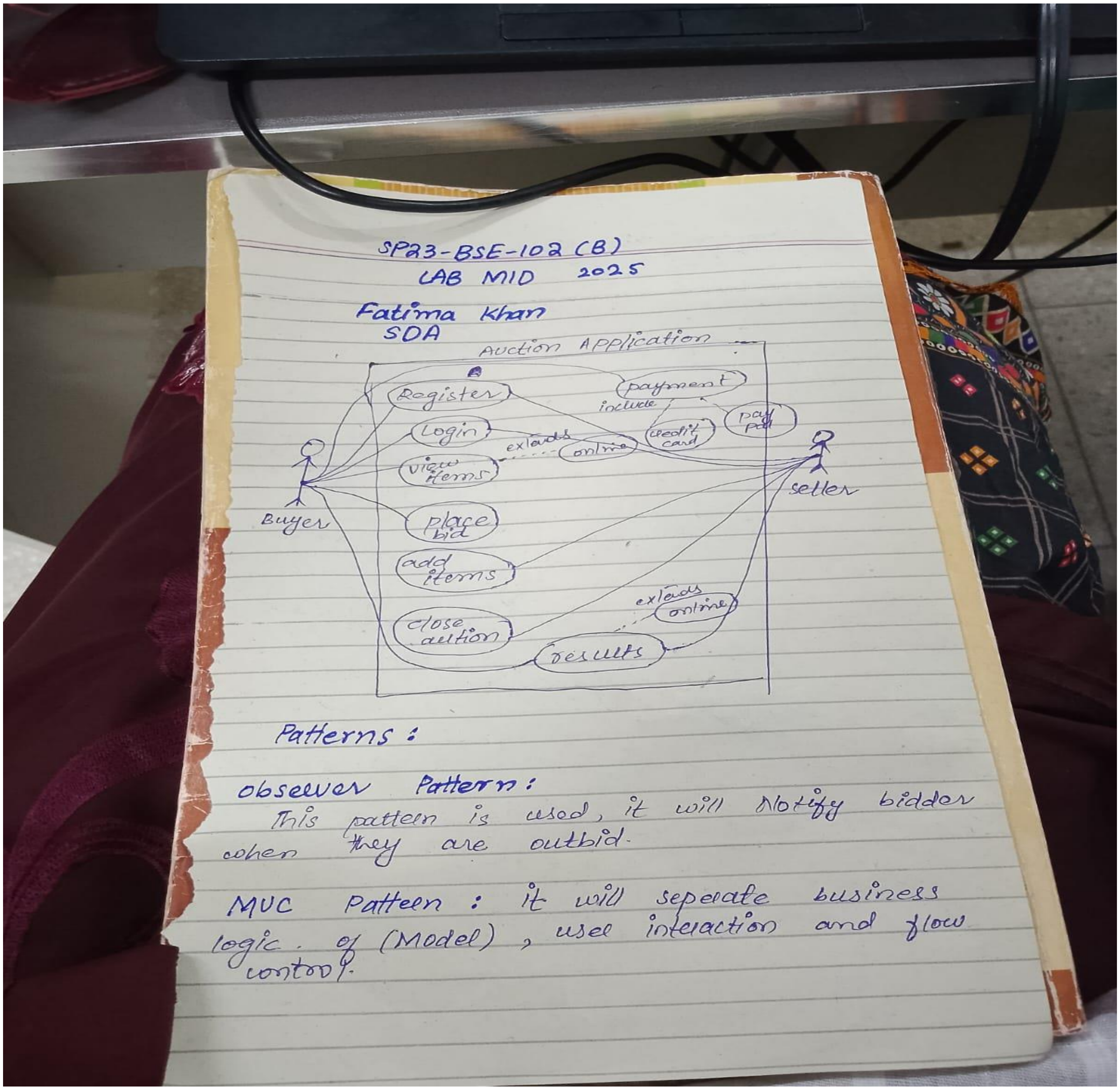
Submitted by: Fatima Khan

Roll no: SP23-BSE-102

Date: 22ND MAY 2025

COMSATS UNIVERSITY ISLAMABAD ABBOTTABAD CAMPUS

Use case diagram



SP23-BSE-102 (B)
LAB MID 2025

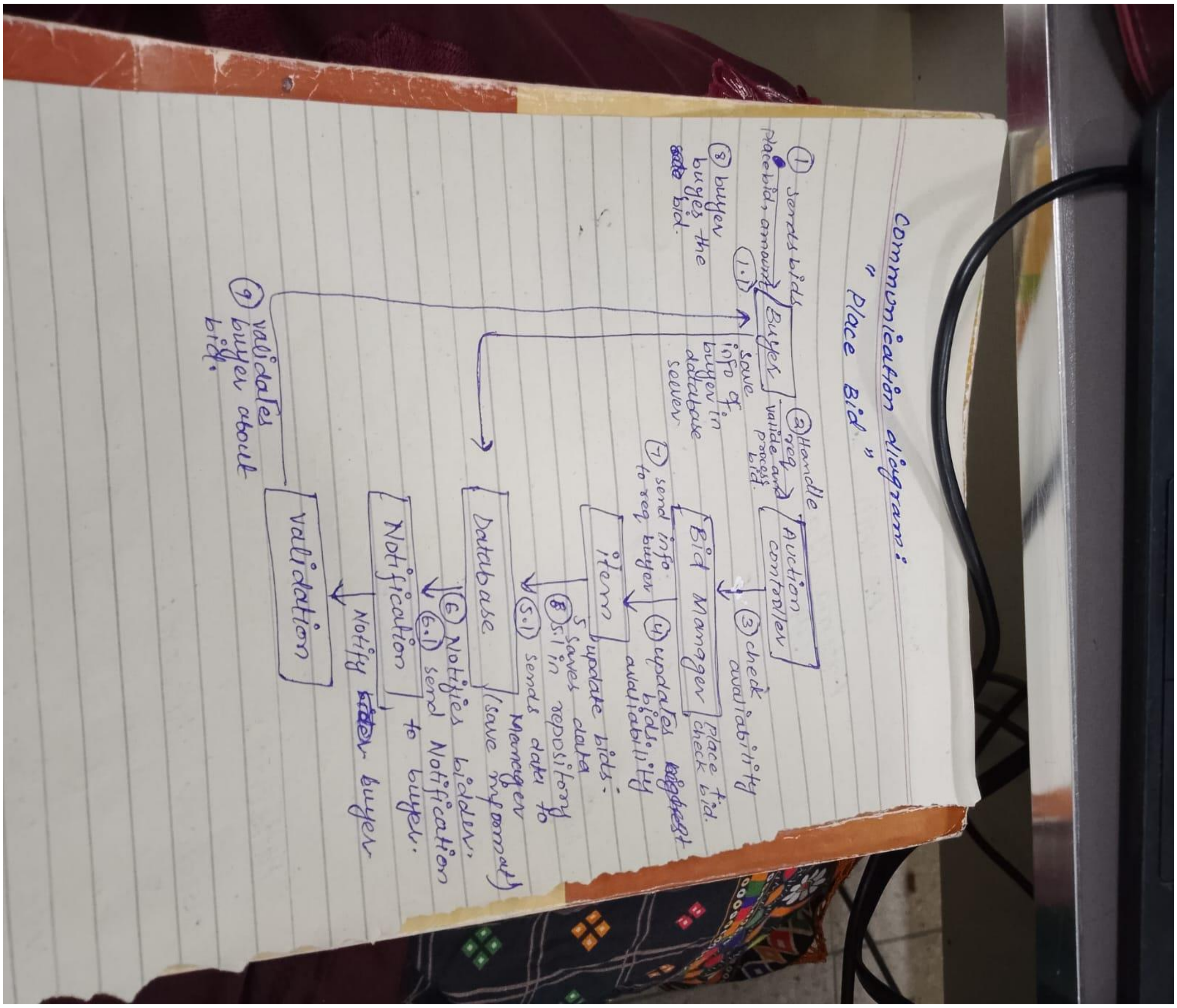
Fatima Khan
SDA

Patterns :

obseever Pattern :
This patten is used, it will Notify bidder when they are outbid.

MVC Patten : it will seperate business logic. of (Model), usee interaction and flow control.

Communication diagram



PRINCIPLES

Following are the principles that I have used in my diagram

1: information expert:

I have used this GRASP principle because it is used to assign responsibility to the class that has the necessary information. Since Item knows the current highest bid and bidder, it makes sense for it to manage bid updates.

2: controller:

Auction Controller acts as a mediator between external events and the entire system, using this, it will make the system easy to maintain without effecting the other functionality od the code.

3: creator:

the class that has the right information creates the object, Since Bid Manager knows about Items, it makes sense for it to handle creating or managing them. This keeps the code organized and easier to manage.

4: low coupling:

low coupling reduces the dependencies between the classes, if we change one class that change does not affect other classes, this will improve the maintainability of the system, for example, if Buyer just calls Auction Controller it handles coordination: it talks to Bid Manager, which updates the Item and triggers Notification Service doesn’t know how bidding works or how notifications happen it only knows Auction Controller.

4: high cohesion:

High Cohesion is used in the auction application by ensuring each class has a single, clear responsibility. For example, Item manages bid updates, Bid Manager handles bid processing, Notification Service deals with notifications, and Auction Controller coordinates the request flow. This focused design makes the system easier to understand, maintain, and extend, as each class does one job well without mixing unrelated logic.

5: Polymorphism

Polymorphism allows the system to support multiple types of bids or notifications (e.g., manual bid, auto-bid, SMS, email) using common interfaces. This makes the system flexible and easily extendable without modifying existing classes.

6: Pure Fabrication

Notification Service is a pure fabrication   it’s not part of the core domain model, but it handles notifications as a separate concern. This keeps the core logic focused and the system design cleaner.

7: Indirection

The Auction Controller introduces indirection by acting as a middle layer between the user and business logic. This decouples components, making the system more modular and easier to modify or replace parts independently.

8: Protected Variations

Protected Variations are used by hiding variability (like how notifications are sent) behind interfaces. For example, changes in the notification method (e.g., from email to push) don’t affect bidding logic, ensuring stability in the rest of the system.

DESIGN PATTERNS

Observer Pattern

If we commit any change observer pattern will notify all the observers.

- **Notification Service observes bidding events**
- When a higher bid is placed, it notifies the previous bidder

Singleton Pattern (optional)

Insuring a single instance of services like `Notification Service` or `Bid Manager`

ARCHITECTURE

Model-View-Controller (MVC)

Separating concerns in the application

- **Model:** Item, Bid, Bid Manager handle business logic and data
- **View:** UI layer
- **Controller:** Auction Controller handles user input and coordinates with the model

COMSATS UNIVERSITY ISLAMABAD ABBOTTABAD CAMPUS