Design Rationale

Design Principle - Single Responsibility Principle

Based on the class diagram, it is observed that most classes are doing just one task that is assigned to them. For instance, the LoginPage class's objective is only to authenticate a user, the PostOfferMessage class's objective is only to post the tutor's offer details to the POST /message endpoint.

Some of the more complex functionalities are broken down into multiple classes. For instance, the StudViewOffers class's objective is to display offers to the logged in student, and it has an association onto BidCardItem class where it uses its attributes to store as arraylist objects. So we tried our best to follow the single responsibility system as this made it quite easier for us to work on the system without worrying about understanding each other's code and just asking for the relevant data needed to work on the next Class. This also made it easier to divide the work fairly and efficiently.

Design Principle - DRY Principle

Moreover, many classes have been reused instead of having similar codes at different places within the application. For instance, the BidCardItem class is reused multiple times by other classes to instantiate objects to be added to arraylist. This avoids duplication of code.

We have also used polymorphism through overriding methods in order to reuse certain methods such as the onOptionsItemSelected method, onCreate method, and so on. Therefore, different classes override the methods to implement different behaviours. This improves reusability.

Design Principle - Open Closed Principle

The code is made extensible (open for extension) by making use of interfaces in the form of adapters for android. Therefore, more functionalities can be added without breaking the code.

Design Principle - Dependency Injection Principle

The design rationale includes dependency injection which is part of the inversion control principle where we are removing internal dependencies and instead using android intent class to inject dependencies externally. As evident in the class diagram we are mostly using intent class to pass dependencies. This helps in coordination, testing and makes the code easier to read by making it more modularized, thus avoiding tightly coupled classes.

Design Principle - Interface Segregation Principle

A big part of our project involves showing a list of bids and tutors in a way that can be easily navigated and acted upon. We implemented four different types of interfaces (BidCardItemAdapter, OffersCardAdapter, StudViewOffersAdapter, RecyclerAdapter) for these classes. The interfaces are in the form of adapters since we are using android. The interfaces used are narrow enough so other classes don't implement behaviours they don't need. Thus we have applied the Interface Segregation Principle on some aspects of the project.

Design Principle - Acyclic Dependencies Principle
Acyclic dependencies principle is also included where our classes do not have an acyclic graph of dependencies. Most classes depend on each other in a way that ends at some point for example for student bidding accepting. This is to ensure that the system runs in a smooth manner and improves maintainability.

Design Principle - Stable Dependencies Principle
The design also uses stable dependencies principle where our main package is only dependent on more stable packages provided by the Android framework. This will help us later on in assignment 3 where we might have to do some refactoring and changing aspects of the system will be much easier.

Creational Design Pattern
The creational design patterns that we have included in our design include, factory method during the login process. There can be a scenario where a user is both a student as well as a tutor. So the login class uses the factory method and based on the user input details creates the class that is required. For example a student logged in will be redirected to StudentLoggedIn class and a tutor logged in will be redirected to TutorLoggedIn class and if a user is both they will be redirected to TutorStudentBoth which again uses the factory method and decides on the user input to make new classes.

Behavioural Design Pattern
We also incorporated behavioural patterns into our design where we use adaptors that are used to give a known interface unknown objects. As we are using an API we created adaptors so that we can convert those interfaces to known interfaces. This has been used in the RecyclerView where we take data from the API but make use of known interfaces to make the RecyclerView work properly.

Observational Design Pattern
Lastly our design includes an observational design pattern which is being implemented in the bidding class which observes any changes in the dependencies and notifies the class that something has changed. We use this to implement a timer for the bidding procedure.

References

Kanjilal, J. (2015, August 21). Exploring the dependency injection principle. Retrieved May 02, 2021, from

https://www.infoworld.com/article/2974298/exploring-the-dependency-injection-principle.html#:~:text=The%20DI%20principle%20is%20a,a%20separate%20framework%20or%20component.