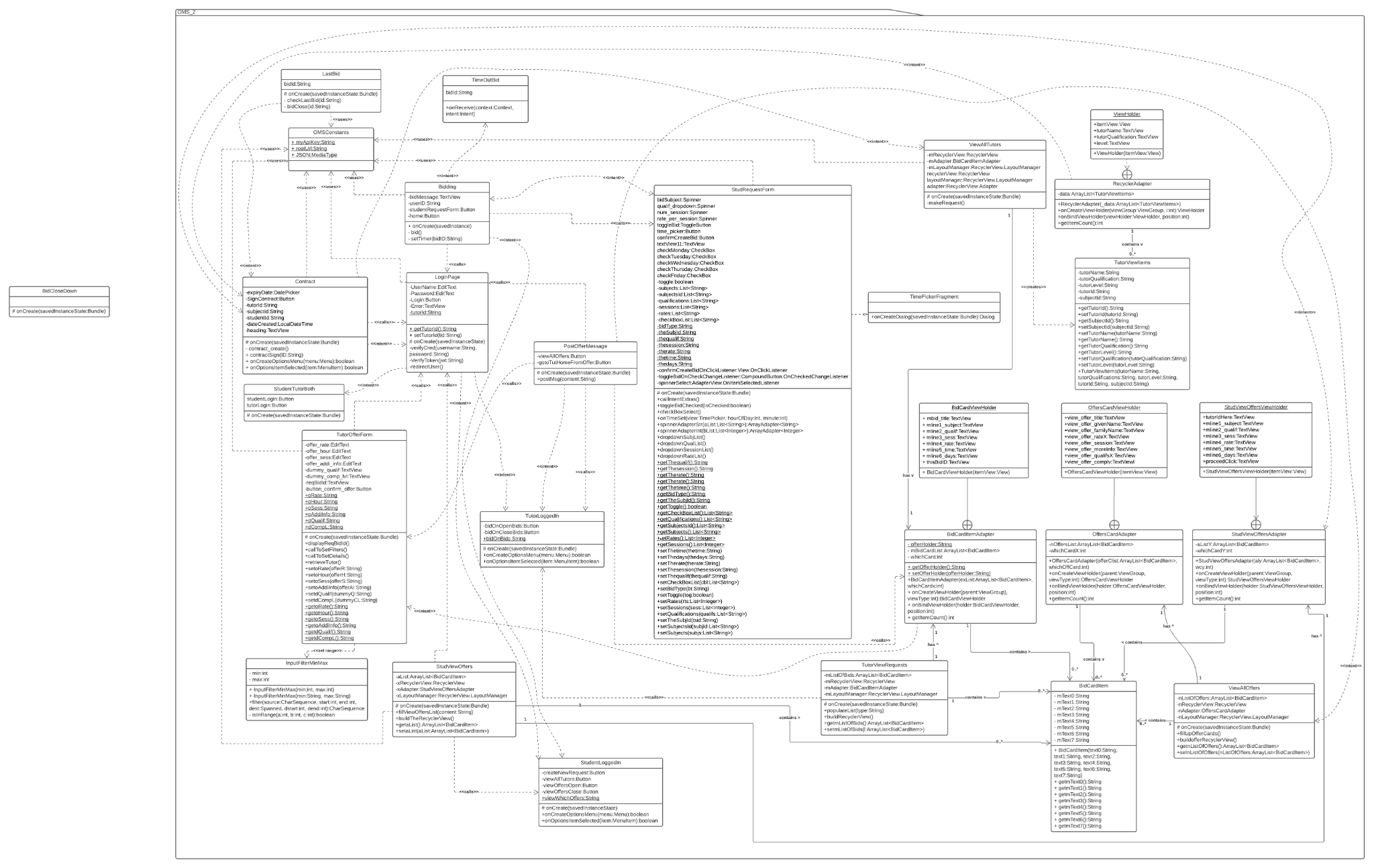
**Class Diagram** (please view the class diagram from the repo for clarity)

****

**Design Rationale**

Based on the class diagram, you can see that most classes are doing just one task that is assigned to them. Some of the more complex functionalities are broken down into multiple classes. 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.

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.

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 three different types of interfaces for these classes thus we have acted upon the Interface Segregation Principle on some aspects of the project.

Acyclic dependencies principle is also included where our classes do not have an acyclic graph of dependencies. Most classes depend on each 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.

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.

The 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.

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. 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.

Bibliography

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>.