Introduction

In this assignment, we made quite a lot of changes and we tried to follow some design principles where applicable. The biggest system wide change we made was the way how things were abstracted and reusing those elements instead of creating similar code all over. We Created an abstract class called WidgetAbstract and its child classes are the UI/views that inherit from the WidgetAbstract. We also applied the MVC architecture which is applied to the system wherever the View needs to talk to the Model. This architecture was applied to our dashboard which required different views depending on the type of user and the MVC architecture deemed perfect for it.

Widget Abstract Class

This abstract class was created to improve the code in our User interface so that all the UI widgets, frames etc can be created using this one class instead of having repetitive code of creating a new widget for every UI.

Advantages

One of the biggest advantages of having such a class was that, when a user logs in to the system, the username is stored in the widget class. So if we wanted to get access to the current user details, we could just access the user details using the self object rather than having to pass the username into every UI as a parameter which was done with our previous implementation of assignment 2.

This class had simple methods that could be called easily and the methods were much smaller. Therefore any minor change to the methods in this class did not affect the code at all because the minor change that could have potentially have arisen would be changing the font size, colour, etc.

Disadvantages

When an abstract class is created, it needs to be carefully created. This is because, this class will be used by future classes and there is a good chance that this abstract class could change if there are any major changes to the design of the UI

In our design, we did not allow the font of the widget to be changed externally because this would mean that too many parameters need to be passed and it could complicate the code. So we managed the font and its size internally.

MVC Architecture

The MVC architecture is applied to our system as our system has a Model and a view component. So for the interaction to take place without too much coupling between the components, we added the controller package.

Advantages

One of the main advantages of using MVC pattern is that it is easy to add multiple views. In our system, we had to have different views because there are 2 types of users in our system namely being student and tutor. Both of them will have different purposes to use the app so they should have different views. This is where the MVC pattern comes to help manage multiple views. The MVC pattern would also help in adding a third type of user in the future if we wish to.

The MVC pattern also separates the business logic from the UI. This means that the User interface can act as a template where inputs are fed from the controller and displayed to the user based on the user. So in our system, if the user login as a tutor, they can view all the bids whereas for the student, the student can create a bid, view their bids etc.

Another advantage that this pattern brought to us was that most of the changes we made in our tutor and student view controller did not really affect the rest of the system. We were able to modify how the dashboard looks and add more buttons and this would not affect the system at all. This would also be a greater advantage in the future if there would be any need to add more features because we can customize the dashboard and not affect the system at all.

Disadvantages

The only disadvantage that comes with implementing the MVC pattern is that it adds too many classes to the system which increases the complexity and readability of the code. This also made our class diagram more complicated and thus making it harder to understand the design as well. So if there were to be any new potential developers who wish to join the project, they would be frustrated to read the code.

Refactoring techniques

In our system, we did a lot of basic refactoring like renaming variables, methods, classes etc. The code in our loginUI was previously coupled with the logic of the program and this made it quite confusing as to where we needed to make the changes. So we used the extract composing technique in our code to extract the logic from the UI and created a call in the UI to do the login process.

We also used the extract variable technique to refactor a few things which check if the user is a tutor or student and we also refactored some methods like the isOpen() method and isAvailable() methods to call the methods directly instead of creating a variable.