MVC Design Pattern

1 Model **:** stores & manages data

Of the three parts of MVC, the model has the most processing tasks, and I use JDBC objects to process the database. The data returned by the model is neutral, meaning that the model is independent of the data format, so that a model can provide data for multiple views, reducing code duplication.

I create a database call ass2,I five tables in “ass2”, which are users, projects, columns, tasks, and items.

I create UserDBUtils, TaskDBUtils,ProjrctDBUtils,ColumnDBUtils to link java database and execute SQL statement.

2 View

A view is an interface that the user sees and interacts with. A view is an interface composed of FXML elements

I have created a lot of FXML files that are all about interface design.

3 Controller

The controller takes input from the user and invokes models and views to fulfill the user's requirements. So when you click a button in a Java page and send an FXML form, the controller itself does nothing and does nothing. It simply receives the request and decides which model artifact to call to process the request, and then determines which view to use to display the data returned by the model process.

I create a lot of Controller class. Each FXML file has a corresponding controller class. These controllers receive the user's request and decide which model should be invoked for processing

MCV

I will show the processing process of MVC is summarized. Firstly, the controller class receives the user's request and decides which model(database) should be invoked for processing and return data. Finally, the controller formats the data returned by the model with the corresponding view and present to the user through the View(fxml file).

SOLID Design Principle

Open to extension here means that adding new code allows the program to expand to meet changing requirements; Closed to modification means that you do not modify existing code while extending the behavior of the program to avoid affecting existing functionality.

In SmartBoardController class I use Open-Closed Principle, The basic definition is that objects (classes) in software should be open for extension, but closed for modification.

I also use single responsibility principle, There should be only one reason for a class to change. The reason for the change represents the responsibility of the class. Such as One for creating test and one for rewriting test. The reason for these two changes is actually two separate functions that should be separated in separate classes.