**RMIT University**

**COSC2391 – Further Programming**

**Assignment 2**

**Student Name:** Dean Papadofrangakis

**Student Number(s):** s3824418

**Submission Due Date:** 3/06/2022

**Submission Date:** 31/05/2022

**Analysis & Justification of Design Choices**

**MVC Design Patterns**

Using javafx alongside MVC to form the ‘Smart Canvas’ application enables the user to not only view the application but navigate and interact with its features. Using the view portion of MVC I was able to use javafx stages to display the user interface to the user, here the user could choose where to navigate next. To navigate between the different views (user interfaces) each model is connected to a controller class. This controller class defines the FXML elements, and their backend features shown in the related view. Connecting the view to the controller allows the user to interact with the elements shown in the current view and receive the correlated component updates, an example of this is the rotation value of a canvas element changing based on the current value of the rotation slider, in which the user can define through clicking and dragging said slider. This connection between view and controller is like the connection between model and controller, a controller communicating changes or requests to the current model for certain pieces of data. For example, in this application the controller can send or request data from the current model such as the current user and any of their relevant details which is stored in database only accessible using methods defined within the model.

**SOLID Design Principles**

In the design of ‘Smart Canvas’ I attempted to adhere to the SOLID principles as closely as possible. In relation to the single responsibility principle, when forming the features of the application each controller class is connected to and used to control a singular view. Meaning that each view has a dedicated controller class containing methods that define the back-end functionality of the related user interface functions. Both the open-closed principle and Liskov substitution principle were adhered to using proper segmentation and properly defined variables. In the case of interface segregation, the only methods used when navigating the application are those directly related to the relevant view’s controller class. Meaning that only methods required for the specific functions of that view are utilized. Dependency inversion is also followed where high level elements of the UI do not depend on low level methods to function.

**Other Design Patterns**

When designing the control class behind the canvas elements (Text, Images, Circle, Rectangles) the elements were separated into 3 categories, text, images, and shapes. This allowed for better code segmentation and less repetition in methods that relate to both circles as well as rectangles. This segmentation allowed for elements to be centred, moved, selected, and unselected using the same methods.