Assessment Task 2 - Code and project bundle

Name: Amit Munjal
Student ID: s3925455

Analyse and justify your design choice in building this desktop-based application by covering the following questions:

1. How did you apply MVC design pattern to build this application?
2. How does your code adhere to SOLID design principles?
3. What other design patterns does your code follow? Why did you choose these design patterns?

1. MVC Design Pattern Implementation:

Model: The Model class is responsible for managing data and business logic. It interacts with the database (model.Model) to handle user-related data and operations.

View: The view is represented by FXML files (LoginView.fxml, SignupView.fxml) and JavaFX controller classes (LoginController, SignupController). These files and classes handle the user interface and user interactions.

Controller: The controller classes (Main, LoginController, SignupController) act as intermediaries between the model and the view. They handle user input, update the model, and update the view accordingly. For example, LoginController manages the login process, interacts with the model to validate user credentials, and updates the view to show login status.

2. Adherence to SOLID Design Principles:

Single Responsibility Principle (SRP): Each class has a single responsibility. For instance, LoginController is responsible for handling login-related operations, SignupController for signup-related operations, and Main for initializing the application.

Open/Closed Principle (OCP): The code is open for extension (e.g., adding new controllers or views) but closed for modification. Existing classes can be extended without altering their core functionalities.

Liskov Substitution Principle (LSP): There are no direct violations of LSP. Subclasses (LoginController, SignupController) can be substituted for their base class (Application) without affecting the program's behavior.

Interface Segregation Principle (ISP): Interfaces are not explicitly defined in the provided code, but the separation of concerns between classes ensures that each class has a focused set of responsibilities.

Dependency Inversion Principle (DIP): Dependencies are injected rather than hardcoded. For example, in Main, the controller instance is customized using a Callback, allowing for flexible controller instantiation.

3. **Other Design Patterns:**

Factory Method Pattern: In Main, a factory method pattern is implemented using Callback<Class<?>, Object> to customize the controller instance (LoginController).

Singleton Pattern (Potentially): The Model class may follow the Singleton pattern, ensuring there is only one instance of the model throughout the application's lifecycle. However, this isn't explicitly shown in the provided code snippets.

Strategy Pattern (Potentially): The use of callback functions (Callback<Class<?>, Object>) in Main can be seen as a form of the Strategy pattern, where different strategies (controller instances) can be selected at runtime based on conditions.

Overall, the design choice appears to follow best practices by separating concerns, promoting modularity, and allowing for extensibility and flexibility in the application's architecture.