Design Choice Justification Report- MyHealth Application

Application Design with MVC Pattern:
In building the MyHealth desktop-based application, the MVC (Model-View-Controller) design pattern was applied to achieve a clear separation of concerns and to enhance maintainability and extensibility.
Model: The model represents the data and business logic of the application. It includes classes such as User and HealthRecord that handle data storage, retrieval, and manipulation.

View: The view is responsible for the user interface components. In this application, the views are implemented using JavaFX, including the UserBoard and Login screen. They provide a visual representation of the data and interact with the user.

Controller: The controller acts as an intermediary between the model and the view. It handles user interactions, updates the model, and updates the view accordingly. The UserController class manages user-related operations, such as user creation, login, and health record management.

Adherence to SOLID Design Principles:
The code in the MyHealth application adheres to the SOLID design principles, which promote clean code architecture and flexibility.
Single Responsibility Principle (SRP): Each class has a single responsibility and focuses on one aspect of the application. For example, the UserController handles user-related operations, the DatabaseManager deals with database interactions, and the HealthRecord class manages health record data.

Open/Closed Principle (OCP): The code is designed to be open for extension but closed for modification. This is evident in the use of interfaces and abstract classes that allow for easy extension of functionality without modifying existing code.

(**LSP**): The code uses inheritance and polymorphism correctly, ensuring that derived classes can be substituted for their base classes without affecting the correctness of the program.

(ISP): Interfaces are designed to be focused and specific to the needs of the implementing classes. This avoids forcing classes to depend on methods they do not need, promoting loose coupling and high cohesion.

Dependency Inversion Principle (DIP): The code relies on abstractions and interfaces, rather than concrete implementations, to promote loose coupling and easier maintenance. For example, the UserController depends on the DatabaseManager interface instead of a specific database implementation.

Other Design Patterns:

Apart from the MVC pattern, the code in the MyHealth application also follows other design patterns to improve code structure and solve specific problems:

Singleton Pattern: The UserController and DatabaseManager classes use the Singleton pattern to ensure that only one instance of these classes is created throughout the application's lifecycle. This pattern provides a centralized point of access and prevents unnecessary object creation.