RMIT International University Vietnam

Assignment 2 Build a Backend

Subject Code:	**COSC230**
Subject Name:	**Further Programming**
Title of Assignment:	**Build a backend**
Student name and student ID:	**Dong Dang Khoa - s3986281** **Nguyen Tran Tri An - s3974923** **Nguyen Trung Tin - s3988148**
Teachers Name:	**Mr. Vu Minh Thanh**
Number of pages including this one:	**6**

Git Link: https://github.com/khoabronze/Furtherprog_ASM2

1. **Introduction**

Our insurance claims management system has been designed with precision to simplify and improve the process of managing insurance claims for a diverse range of users. From policyholders to insurance managers, this sophisticated software solution prioritizes efficiency and user-friendliness, offering many features to navigate the claims process from start to finish effortlessly.

Claims are the cornerstone of the system. They contain comprehensive information for each claim, including claim ID, claim date, insured individual's information, agency contact details, inspection date, required documents, claim amount, status (new, in progress, or completed), and recipient's banking information. These claims can be easily managed, edited, deleted, or accessed through an intuitive interface.

2. Key features of the system

Role-Based Access: Various user roles are allocated distinct functionalities to enhance efficiency and effectiveness. Policy holders have the ability to oversee their claims and personal details, dependents can access their claims, policy owners can manage their beneficiaries and claims, insurance surveyors can suggest and request information regarding claims, insurance managers can authorize or decline claims, and system administrators have the authority to conduct CRUD operations on all entities and produce statistical reports.

Document Management: Allow users to easily upload required documents in image format when submitting a claim, with automatic file renaming to maintain a consistent and organized structure.

Real-time updates: The system facilitates seamless instant updates, providing the latest data to all users and ensuring access to the newest information.

Responsive GUI: Created using JavaFX, the visually appealing user interface is designed to provide seamless responsiveness and easy navigation, ensuring a flawless user experience without glitches or interruptions.

Security and authentication: Sophisticated input validation and exception handling mechanisms have been implemented to ensure data integrity and maintain high system performance.

Our comprehensive reporting feature: Allow users to effortlessly organize entities, visually present data, and generate reports that can be saved for future reference. This streamlines the data analysis process and facilitates informed decision-making.

Our sophisticated insurance claims management system is designed to offer a robust, intuitive, and streamlined solution for handling insurance claims, elevating user satisfaction and operational productivity.

3. Class description

The system architecture has been meticulously designed to incorporate multiple classes and interfaces to efficiently manage various entities such as customers, claims, and requests. The DAO<T>, PolicyOwnerDAO<PO>, PolicyHolderDAO <PH>, etc. interface has been implemented to provide standard CRUD operations for all entity types, ensuring consistent data access methods. For example, the Request_DAO<Request> interface is explicitly tailored for request entities, including a specialized process for handling request-specific operations. The Db_function class is responsible for managing database connections and ensuring a reliable connection to the database. The ClaimDAO and RequestDAO classes have been created to handle CRUD operations for claims and requests, respectively, utilizing SQL statements and in-memory storage. Service classes like ClaimService and DependentService have been developed to handle business logic for their respective entities, working closely with DAOs to manage entity operations. This architecture follows the DAO pattern to abstract data access and the Service pattern to provide an API for CRUD operations and business logic. Core object-oriented programming principles such as encapsulation, abstraction, inheritance, and polymorphism are evident throughout the design, with each class encapsulating its data and methods, interfaces defining abstract methods, and DAO implementations showcasing polymorphism. Robust error handling, specifically for SQLExceptions, has been implemented within DAO methods using try-catch blocks to manage database-related errors effectively. This well-structured architecture ensures a clear separation of concerns, ease of maintenance, and scalability.

4. Limitations and future updates

While effective, the current insurance claims management system has areas that could be enhanced for a more seamless user experience. Dependents need help accessing and providing additional information, and there may be some overlap in responsibilities between policy owners and holders. The system's scalability may require optimization for larger datasets and increased transaction volumes. The JavaFX interface, though responsive, lacks the advanced features found in modern web and mobile applications, such as drag-and-drop document uploads and real-time notifications. Security measures are basic and could benefit from enhancements like two-factor authentication and data encryption for sensitive information. Real-time collaboration is not supported, hindering multiple users from working on claims simultaneously with live updates. Future enhancements should focus on enhancing document management, optimizing scalability, developing web or mobile interfaces, enhancing security measures, enabling collaborative real-time, and improving reporting and analytics.

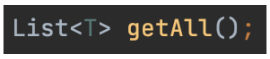
These updates will improve its robustness, user-friendliness, and ability to adapt to changing user needs.

5. DAO pattern and its layers (class explain)

Methods

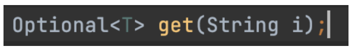
The DAO.java file defines an interface with several methods for handling data access operations in our system.

These methods include getAll(), which retrieves all records of a specified types from the database, and get(String i), which retrieves a single record using an identifier and returns an Optional object.



```
List<T> getAll();
```

-->



```
Optional<T> get(String i);
```

--> return as an 'Optional object'

The **getOne()** method also retrieves a single record but it could throw an exception if the record is not found. The **add()** method adds a new record to the database, while the **update()** method updates an existing record.

The **delete()** method removes a record from the database, and the **switchStatus (String id, String status)** method changes the status of a record identified by its ID.

The above methods are part of the **Data Access Object (DAO)** pattern, which provides an abstract interface to the database, separating the business logic from the data access logic. So to make the system a cleaner and more maintainable codebase by summarizing the details of data persistence and retrieval. It also enhances testability, as the mock implementation of the DAO interface can be used in unit tests and promotes reusability of data access operations.

Our systems use a layered architecture pattern, organizing the system into distinct layers, each with specific responsibilities.

Layered Architecture Pattern

The insurance claims management system uses a layered architecture pattern, which organizes the system into distinct layers, each with specific responsibilities.

These layers include the **Presentation Layer (UI)**, implemented using **JavaFX,** which handles the user interface and interaction; the **Application Layer (Service)**, containing the business logic and rules, which processes input from the interface layer and makes calls to the data access layer; the **Data Access Layer**, managed by the DAO pattern, which performs **CRUD** operations on the database; and the Database Layer, which is the actual **PostgreSQL** database hosted on **Supabase**.

The layered architecture pattern provides several advantages, including modularity, maintainability, testability, and scalability. Each layer can be developed and updated independently, making the system easier to maintain and update. It also facilitates unit testing by isolating the layers and enhances scalability by allowing layers to be distributed across multiple servers. Combining the DAO and layered architecture patterns ensures a robust, modular, and maintainable insurance claims management system.

6. Class Diagram