ASR Insurance Management System

Ananthakrishna Kalle, Saranya Machavaram, Rucha Mangalekar

Information Systems Engineering, Northeastern University

{kalle.a@northeastern.edu, machavaram.s@northeastern.edu, mangalekar.r@northeastern.edu}

***Abstract*—**

The goal of an insurance management system is using advanced software and electronic technologies to automate current manual procedures. This solution securely stores insurance providers' information and crucial data for an extended period, aligning with their objectives. The hardware and software necessary for the system are available and user-friendly. Insurance providers can obtain reliable, error-free, and prompt management by utilizing this solution, enabling them to concentrate on other duties besides record-keeping. The technology used can manage digital records, reducing interruptions from irrelevant data when viewing information.

The project utilizes the Java, JavaFX, and FXML-enabled Eclipse IDE, along with dependencies of Maven to inject jar files of MySQL. The project's design and operations are organized using the Java MVC Framework. All database actions are handled by a second file, DB.java, which controls all CRUD activities. The project is segmented into four essential categories: activity log, admin operations, insurance planning, and registration/login.

Keywords: ***Object Oriented Programming, JavaFX, MVC Framework, Insurance Management , SQL, and JAVA.***

# **I. Problem Description**

The aim of this project is to build and develop an insurance management system that caters to two types of users - customers and admins - each with distinct roles and functions. The system aims to efficiently manage insurance plans and user data. The application's interface is designed to ensure user comfort and smooth administration. Both users and admins have access to a range of features and functions.

# **II. Analysis (Related Work)**

Creating a web application for registration and insurance filing provides several advantages over traditional methods. With online insurance, users can conveniently obtain necessary plans from their homes or workplaces, eliminating the need to visit brokers or insurance companies to purchase policies. This hassle-free method saves time and energy, providing a user-friendly platform to purchase coverage instantly.

Online portals allow users to compare and understand the diverse policies offered by different insurers, enabling informed decision-making. Purchasing insurance online ensures that users can obtain policies that fulfill their needs without the interference of salespeople who may try to sell unnecessary products.

Nowadays, many major insurance providers offer their customers access to an online application or portal. The aim of this research was to gain a comprehensive understanding of the application development lifecycle.

# **III. System Design**

Diagram

Description automatically generated

Figure 1. Architecture of Insurance Management System

# **IV. Implementation**

*Brief Overview*

The Model View Controller (MVC) paradigm is a widely used design pattern for building and developing online applications, consisting of three key components.

* The Model manages data.
* While the View displays it.
* The Controller acts as a mediator between the View and the Model, facilitating their interactions.

By separating logic of the application from the user interface layer, MVC is renowned for its ability to segregate concerns. The Controller receives the application requests and works in conjunction with Model to generate any required data for View. View then utilizes this processed data to produce a final presentable output.

*A. User’s Registration and Login*

The application's homepage features registration and login buttons. New users must fill out a registration form to create an account and gain access to the application. After clicking the submit button, the Java program establishes a connection to the database and adds the user's information to the User database.

Existing users can simply use their login credentials to access the system. However, if the ID of the user does not end with "@northeastern.edu", a failed login notice is sent. The credentials of user are then used to query the MySQL database. If the user's credentials match, they are redirected to the application's homepage and considered successfully logged in.

*B. Home page after Login*

After logging in, the application controller uses the Initializable method to display all the dynamic data retrieved from database and presented in application. The specific method is overridden by the code to display buttons, labels and text when the website loads.

Users who don't have insurance can see the “Add Plan" button, while those whose ID suggests they already have insurance can access “Update Plan", and "Delete Plan" buttons. Whenever a user adds or modifies their insurance plan, database is updated accordingly. However, if a user clicks the "Delete Plan" button, the plan is deleted from the database. During page loading, the list operator is used to convert and display data.

*C. Application Log*

When the user clicks the "Show Log" button, it activates two static variables, a Stack and a Dequeue. These data types display the user's activities during a session. The function push log adds the appropriate string to Stack every time an action is performed. Stack follows a first-in-last-out methodology, so the method pop() displays the log in reverse order, with the most latest action at the top. The Dequeue is used to maintain order of the components in the Stack. The data is added to the Deque before being popped from the Stack and displayed in sequence. When the user clicks the back button, the Dequeue is cleared, and all of the information is re-added.

*D. Functionality of Admin*

In the current system, only users with an ID ending with "@northeastern.edu" can log in as regular users. However, the system allows for an administrator to log in with different credentials. Once logged in, the admin is redirected to a webpage where they can search for and delete records from the database by clicking the "Submit" button.

To retrieve data such as plan details of insurance that match the entered email, the system uses a LIKE query in the DB.java file. The administrator has the authority to delete any insurance plan from the application by clicking the delete button, which removes the associated plan from the database. The admin credentials are stored in the database, and new admin accounts can be added through the application's interface.

Finally, after clicking the "Sign out" option, the admin is redirected back to login page.

*E. Connection of Database*

The DB.java file is responsible for handling all the database-related operations and SQL queries. It creates a new instance of the DB class each time a controller requires a database connection, and it provides various methods for adding, setting, getting, searching, and deleting records with different arguments. The DriverManager class is used to establish a localhost connection to the MySQL database on port 3306. Prepared statements are used to execute queries with parameters, eliminating the need to create a new query for each execution.

The UserSession class keeps track of the user's session by creating an instance of itself with the getsessionInstance() function when user logs in. setsessionInstance() function sets the instance value to null when the user logs out.

The function searchAdminRecord() in the DB class doesn’t make any database calls. Instead, the code stores the admin's name and password as static values, removing the need for a database connection. For regular user logins, the select SQL query is used to check whether the entered username and password match any rows in the user\_registration table.

The ood\_project database has two tables: insurance\_plan and user\_registration. The user\_registration table contains three columns: username, email address, and password. The insurance\_plan table contains all the insurance-related information, with the email ID being the field shared by both tables. This allows the user's insurance information to be displayed immediately upon logging in. The auto-incremented values for both table IDs can be used when adding new rows. The Java method converts the varchar values from MySQL to Strings for the project's database data.

# **V. Evaluation**

Graphical user interface, application

Description automatically generated

1. User’s Registration

Graphical user interface, application

Description automatically generated

1. User’s Login

Graphical user interface, application

Description automatically generated

1. Home Page After Login

Graphical user interface, application

Description automatically generated

4. Adding User’s Plan

Graphical user interface, text

Description automatically generated 5. Insurance Plan in Home page

Graphical user interface, application, Word

Description automatically generated

6. Updating User’s Plan

Graphical user interface, text, application

Description automatically generated

1. Application Log

Graphical user interface, application

Description automatically generated

1. Admin’s Login

Graphical user interface, application

Description automatically generated

1. Admin’s Home Page

# **VII. Discussion (Reflection)**

During the course of the project, we successfully developed a useful insurance management system. Through this process, we gained a deeper understanding of Object-Oriented Programming (OOP) principles and how they can be applied to create modular, extendable, maintainable, and reusable software systems.

OOP is based on the concept of objects, which combine data and behavior and provide services from a well-defined interface that hides implementation details. With features such as classes, inheritance, association, and dynamic polymorphism, OOP also allows modeling solutions at a abstraction of a higher level.

Scene Builder, a GUI design tool, enables the creation of graphical user interfaces without requiring any coding. It allows the quick arrangement of JavaFX UI shapes, controls, charts, and containers to prototype UIs. FXML, XML-based markup language, can be used to create the UI separately from the application logic, and it is possible to modify FXML files that are being created by others.

Scene Builder is compatible with any Java IDE, but Eclipse IDE offers tighter integration. Projects that use Scene Builder automatically update when FXML is modified in Eclipse, making it easy to connect the user interface to source code which manages events and actions for each element. The user interface of your project can be previewed in real-time without being limited by the palettes and tool's menus.

# **VIII. Conclusions and Future Work**

An insurance management system offers a reliable, secure, and error-free way for users and administrators to organize and access data and resources. The system automates manual processes using computerized hardware and extensive software, allowing for easier and longer-term data access for users and more effective data management for administrators.

Future work involves investigating new approaches, thoroughly examining specific systems, and adding new features such as a chat window for communication between users and administrators and an online payment option for users. However, some testing, modifications, and experiments were delayed due to time constraints.

To enhance the user interface and provide dynamic content on FXML pages, style sheets can be used, including adding multiple insurance plans for clients. Using Properties Panel and pointing a component of GUI to the specified CSS file, the required look and feel can be achieved. The CSS analyzer can also determine how certain CSS rules impact the functionality of JavaFX component features.

Using a model-trained engine, our insurance management system can provide each customer with personalized insurance plans based on their unique lifestyle and medical history. The engine forecasts and determines the best insurance plan for the customer depending on the information entered, creating a tailored insurance experience.

# **IX. Job Assignment**

Saranya Machavaram worked on creating the database schema, as well as developing the code for Searchview, primary, and Logininto fxml, data model for Insurance java file and controllers. Additionally, they contributed to GitHub Project Reports I, VII, and VIII.

Ananthakrishna Kalle was responsible for building the DB class, as well as developing the AddPlan, UpdatePlan fxml, data model for User java file and Java controller files. They also contributed to Project Presentation and authored Project Reports III, IV and IX.

Rucha Mangalekar focused on designing the system for the application and developed the App.java page. They also worked on building the AfterLogin and MessageLog fxml, data model for UserSession Java file and controllers files, and contributed to Project Reports II, V and VI.

##### **References**

1. <https://openjfx.io/openjfx-docs/>: a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms.
2. <https://docs.oracle.com/javase/8/javafx/api/toc.htm>: provides detailed information about the classes, methods, and interfaces in the JavaFX API.
3. <https://fxdocs.github.io/docs/html5/>: It includes tutorials, examples, and explanations of how to use various features of JavaFX, as well as best practices.