Insurance Management System

Sreya Vallabhaneni, Abhinav Palem, Venkata Abhishek Tiruchunapalli, Venkatesh Gopinath

Software Engineering Systems, Northeastern University

Data Analytics Engineering, Northeastern University

{vallabhaneni.s, palem.a, tirchunapalli.a, bogem.v}@northeastern.edu

***Abstract*—**

**The aim of insurance management system is to automate the current manual method with the use of computerized tools and comprehensive computer software, meeting their needs, in order to preserve their important data and information for a longer**

**The software and hardware that are needed are readily available and simple to use. An error-free, secure, dependable, and quick management system may be achieved with the help of the insurance management system mentioned above. Instead of concentrating on record keeping, the user may find it easier to focus on their other tasks with its help. The firm will thereby benefit from greater resource use. The company has digital records management capabilities. Thus, when accessing the material, one need not be sidetracked by irrelevant information.**

**This is done using Apache Netbeans 15 with Java, Java FX and FXML. Maven dependencies to inject MySQL jar files, MVC Framework with Java. All database operations are created in a separate file jdbcDao which takes care of all CRUD operations. Operations and design of this project can be classified into 4 major parts which are- registration/ login, Insurance planning, Activity Log and Admin operations.**

***Keywords— JavaFX, Object Oriented Programming, Insurance Management, MVC Framework, JAVA, SQL,***

# **I. Problem Description**

Creating an Insurance Management System for users with different functionalities depending upon user role (Customer/Admin).

The main objective of this project is to manage user data and insurance policy. The application is built for both user ease and administration purposes, though the functions for user and admin are different.

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

Compared to the traditional method of insurance filing and registration, building a web application has a lot of advantages. User doesn’t have to visit an agent or an insurance company to purchase the policies. User can acquire the required plan from the convenience of your home or business. Online insurance is simple and hassle-free throughout, which saves time and effort. To make it easy for you to purchase the coverage right away, the businesses provide a user-friendly interface.

Online portals help you to compare and comprehend different policies offered by various insurers. User can explore different plans and make an informed selection. You can be certain that the insurance coverage you purchase online will meet your specific needs. Additionally, when you buy online, you avoid agents' attempts to offer you insurance that don't fit your needs.

Most big insurance now have an online portal or application for customers. This project was for us to build and understand the lifecycle of developing an application from scratch.

# **III. System Design**

Diagram, schematic

Description automatically generated

Figure 1. System Architecture

# **IV. Implementation**

*Brief Overview*

Model View Controller or MVC, is a software design pattern for developing web applications.

A Model view Controller pattern is made up of the following three parts:-

* Model- The lowest level of the pattern which is responsible for maintaining data.
* View- This is responsible for displaying all or a portion of the data to the user.
* Controller- Software   Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user int interface layer and supports the separation of concerns. Here the Controller receives all requests for the application then works with the Model to prepare any data needed by the view. The View then uses the data prepared by the Controller to generate a final presentable response.

*A. Registration/Login*

The login and registration buttons are located on the application's landing page. New users must first register, and on submit button on the registration form prompts the java file to connect to DB and add a customer to User DB, however logged-in users can just login with their credentials.

If the user id supplied does not finish with @northeastern.edu, we receive an alert about failed login. The user's credentials are looked for and compared in our SQL database; if they match, the user is considered to have successfully signed in and is redirected to the application home page.

*B. User Home/ Post Login*

After a user logs in, the controller implements the Initializable method, which is used to display all of the dynamic data that was pulled from the database and displayed in the application. This method is overridden in the code so that labels, buttons, and text are displayed when the page loads.

The user can see the add plan button on the display if he doesn't already have insurance; nevertheless, if the user's id already has insurance, they can see the update plan and delete plan buttons on the display.

The database is updated when a plan is added or updated, but when a user clicks the delete button, they are removed from the database system. Data conversion and display are done during page load using the list operator.

*C. Activity Log*

All user activity for that session is displayed on the screen when an user clicks the show log button. Two static variables with the datatypes Stack and Dequeue were used to accomplish this. We call the push log function, which puts the string into the stack, each time an action is performed.

The pop() method is used to show the log in reverse order, with the most recent activity being displayed on top because the Stack datatype is built on first-in-last-out.

We utilize the double-ended queue (deque) datatype and push information into the queue before popping the element from the stack to print in order to save the order of the elements in the stack.

Finally, the deque is cleared and all information is pushed when the user presses the back button.

*D. Admin*

Currently in the application, sysadmin is being used, and only the administrator has an ID without a @northeastern.edu. As soon as we click "Submit," we are directed to the admin's home page, where they may search and remove database records.

In order to check the input text with our stored data and return all the information, such as insurance plan details if the email matches, we utilized the LIKE query in our jdbcDao file.

The specific insurance plan is removed from the database if the admin clicks the delete button. Consequently, admin has the right to delete any plan from the system. The user cannot create a new admin since we have hardcoded the admin login information.

To do this, we must either manually add new admin credentials to the database or update the code. Finally, the admin is returned to the login page when they click the sign out button.

*E. Database Connection*

All the database connection and SQL queries are written in the jdbcDAO file. Whenever a controller is trying to access the database it creates of the JdbcDAO class. This class has several functions like addRecord, setRecord, getRecord, searchRecord two of them with different parameters, searchAdminRecord and DeleteRecord.

We are using a class called Driver Manager to make a connection with the MySQL using the port 3306 with localhost.

This helps us write prepared Statements i.e. write statements which takes parameters dynamically without writing a new query every time we call that function.

After logging in for the application to remember the user throughout his session we create a class call User Session. As soon as we login we run the getInstance() function. This function creates an instance of the UserSession class, and it remembers the Username until we run the setInstance() function. This function sets the value of the instance to null i.e. user has logged out. This action keeps repeating every time we login.

The searchAdminRecord function in JdbcDAO class does not make any call to the database. The name and password for admin are stored as static values in the code and does not need a database. However, for a general user login there is a database check, the register function adds new rows to the data table user\_registration. During login we check if the select query in SQL returns at least one row of data with the user and password we have given it.

Our database ood\_project has two tables user\_registration and insurance\_plan. The two tables are not connected and do not have foreign keys that link them. User registration table only has three rows Username, email and password. All the other insurance information is stored in the insurance plan. The one common row between these two tables is email id. That is how our Post login page can display the user insurance information as soon as the user logs in. Both of the id’s in these tables are auto increment values and do not need to be given a value every time a new row has added. The varchar values in MySQL are typecast to String in our java code. This encapsulates the database information we have used in the project.

# **V. Evaluation**

Graphical user interface, text, application, chat or text message

Description automatically generated

1. Registration page

Graphical user interface, text, application

Description automatically generated

1. Login form

Graphical user interface, text, application, email, website

Description automatically generated

1. User Home

Graphical user interface, text, application

Description automatically generated

1. Insurance Plan

Graphical user interface

Description automatically generated

1. Add new plan

Graphical user interface, website

Description automatically generated

1. Update Plan

Graphical user interface, text, application

Description automatically generated

1. User Log

Graphical user interface, website

Description automatically generated

1. Admin Login

Graphical user interface, text, application

Description automatically generated

1. Admin Home

# **VII. Discussion (Reflection)**

On completion of this project, we have accomplished building a working model for insurance management.

While working on this project we gained deeper knowledge of Object orientation which helps a developer to achieve a modular, extensible, maintainable, and reusable system.

To write good-quality programs, a programmer must have a firm command of OOP concepts. Object-oriented programming revolves around the concept of an object, which encapsulates data and behavior acting on the data together.

An object provides its services through a well-defined interface. This interface specifies "what" the object offers, abstracting "how" (actual implementation).

Object orientation provides support for modeling solutions at a higher level of abstraction in terms of classes, a hierarchy of related classes (inheritance), association among classes, and dynamic binding (dynamic polymorphism).

You can construct a GUI with Scene Builder without manually entering any code. You are thus spared the laborious effort of adding "children" items to the user interface. Scene Builder allows you to easily layout JavaFX UI controls, charts, shapes, and containers, so that you can quickly prototype user interfaces. Animations and effects can be applied seamlessly for more sophisticated UIs.

FXML, an XML-based markup language that enables users to create an application's user interface independently of the application logic, is produced by Scene Builder. You may also access and edit FXML files that have already been created by other users.

Any Java IDE may be used in conjunction with Scene Builder, although NetBeans IDE has a closer integration. Run your program in NetBeans, and any changes to the FXML there will also be reflected in your Scene Builder project. This allows you to easily link the user interface (UI) to the source code that will handle the events and actions done on each element. You can view a real-time preview of your project's user interface at any time while it is being created, free from the constraints of the tool's menus and palettes.

Spring MVC provides a lot of ease for developing an application. The container makes use of a lightweight servlet for the creation and deployment of applications. It allows for quick and concurrent development. he application is developed quickly. Collaboration amongst numerous developers is simple. Updates to the program are simpler.

# **VIII. Conclusions and Future Work**

An error-free, secure, dependable, and quick management system may be achieved with an insurance management system. It can help users and administrators retain records. improves data organization and makes it easier to use resources.

The goal is to automate its current manual method with the use of computerized hardware and comprehensive computer software, fulfilling their needs and allowing for easier access to and manipulation of the data for longer periods of time.

The problem of keeping records in an orderly way will be resolved with the adoption of a computerized system.

The ability to get knowledge at the click of a mouse is the best of all. Thus, the suggested approach facilitates time savings.

Due to a lack of time (experiments using actual data are typically quite time demanding, taking several days to complete a single run), numerous modifications, testing, and experiments have been shelved.

Future work includes a closer examination of certain systems, fresh suggestions to attempt new techniques, and adding several new features to the application. There are some ideas we had, but the implementation is not done yet.

Some features are adding a page called "Make Claims" where consumers may submit insurance claims. including a chat window where people may communicate with the administrator. adding a payment option for consumers online.

Further, UI enhancements and the addition of dynamic material on FXML pages, adding the capability for clients to add several insurance plans. Using style sheets, you may give your GUI layout the appearance and feel of your choice. It only requires choosing a GUI component and using the Properties Panel to direct the GUI component to the desired CSS file. The CSS analyzer enables you to comprehend how particular CSS rules may impact JavaFX component features.

Personalized insurance for each customer based on their lifestyle and prior medical conditions. We can integrate a model trained engine that predicts and find the best insurance plan for a customer based on the data they enter.

# **IX. Job Assignment**

* Abhinav Palem: Building our database schema, code for Search, Register and login fxml and controllers, GitHub Project Report I, VII
* Sreya Vallabhaneni: Building the jdbcDAO class and addplan and update plan fxml and java controller and project presentation. Project Report IV, IX
* Tiruchunapalli Venkata Abhishek: System design of the application and the App.java page. Building the PostLogin and log fxml and controllers files. Project Report V, VI
* Venkatesh Gopinath: Building the data models User, Insurance and UserSession java files. Project Report II, III and VIII.

##### **References**

1. https://www.oracle.com/technicalresources/articles/javase/mvc.html : This article has shown how to implement a variation of the MVC design using Java SE and the Swing libraries.
2. https://docs.oracle.com/javaee/6/ Java Platform, Enterprise Edition (Java EE) Technical Documentation
3. https://docs.oracle.com/javafx/2/api/ JavaFX fxml: Contains classes for loading an object hierarchy from markup.