



Insurance Broker Management System

J2EE Web Components – ITE 5332 – 0NA

Assignment – 1

Hitesh Kamalkant Jha – N01610330

Neel Kachhia – N01609700

Abdul Mubeen Mohammed – N01611677

Prof. Vitalii Bohudskyi

Table of Contents

Project Description	1
Overview	1
Features	1
Design Decisions	1
Architecture	1
File Storage Format	2
Repository Design Pattern	2
Screenshots	3
UML Diagrams.....	10
Collaboration Details	12

Project Description

Overview

The Insurance Broker Management System (IBMS) is designed to ensure that the brokers remain organized while efficiently managing the data pertaining to the consumers and the insurance policies offered to them in a systematic orderly manner. The system provides data persistence through the use of JSON files and also allows creation, updating, and deletion of consumers and policies. The system improves both maintainability and scalability by ensuring the application of the Repository Design Pattern where there exists a clear separation of concerns between the business components and data components.

The system allows brokers to handle activities like customer registration, policy handling, and organization of insurance records which burnishes operation management along with ensuring the data consistency and integrity.

Features

- Customer Management:
 - Add, update, delete, and view customer records.
 - Comprehensive customer profiles that store important information.
- Policy Management:
 - Create, edit, and assign insurance policies to customers.
 - Delete outdated policies and manage ongoing ones.
- Data Persistence:
 - Store customer and policy data in JSON files for easy management and durability.

Design Decisions

System Architecture

The system is structured with a modular approach to simplify the management of customers, insurance policies and future extensions. The key layers of the system are:

Model Layer:

- Classes like Customer and Policy represent the domain entities of Insurance Broker Management System.

View Layer:

- Developed using Nord Design library, this layer provides simple yet modern interface for brokers to manage customers and insurance policies.

Controller Layer:

- This handles the HTTP requests from the client and interacts with the repository layer to fetch or manipulate data.

Repository Layer (Data Access Layer):

- This is responsible for accessing the data storage. In this case, it's interacting with JSON files for reading and writing customer or policy data.

File Storage Format (JSON)

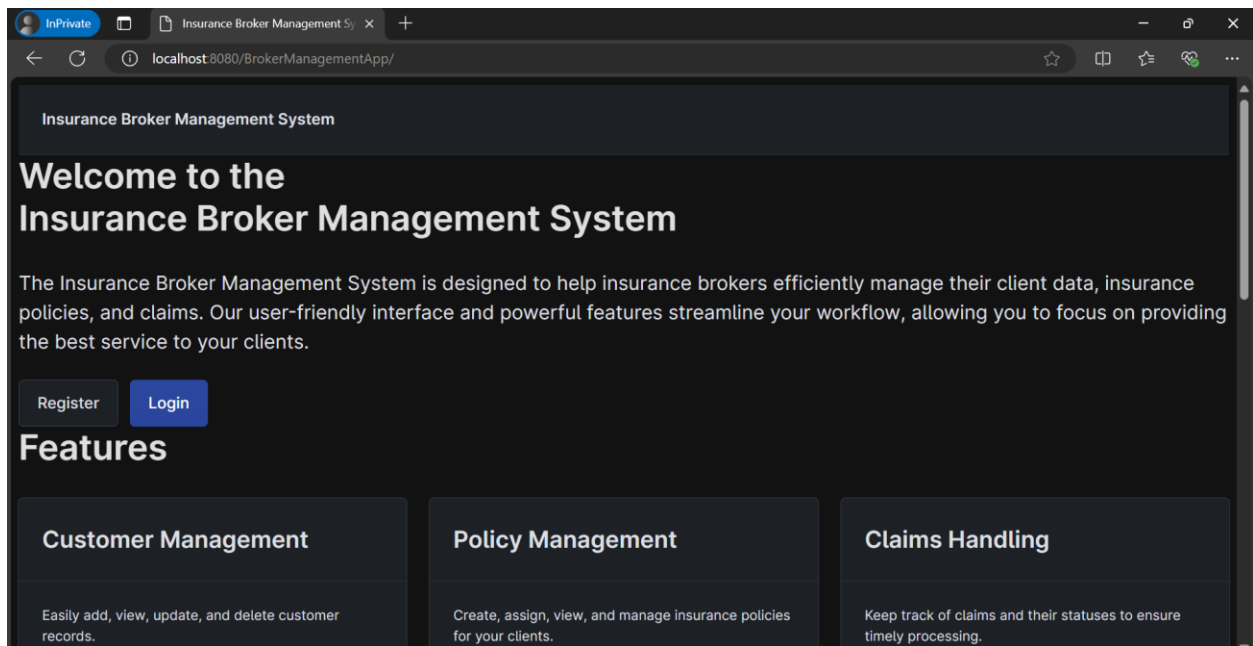
- The data in this architecture is stored in JSON format. The JSON files (customers.json and policies.json) contain serialized data about customers and policies.
- Jackson is used as external jar to manage json files from java code.

Repository Design Pattern

- The Repository pattern is used in application to abstract and encapsulate the logic for accessing data sources. It provides a way to separate the data access logic from the business logic for loose coupling.
- The Repository Design Pattern provides a clear abstraction for data access, making the system flexible and maintainable. This architecture could be extended to use databases or other storage mechanisms in the future, without needing to change the business logic.

Screenshots

Welcome Page



Login

Log In as a Broker

Username

enter your email address

Password

enter your password

Log In

New User? [Register](#).

Register

Register as a Broker

Username

enter your username

Email Address

enter your email address

Password

enter your password

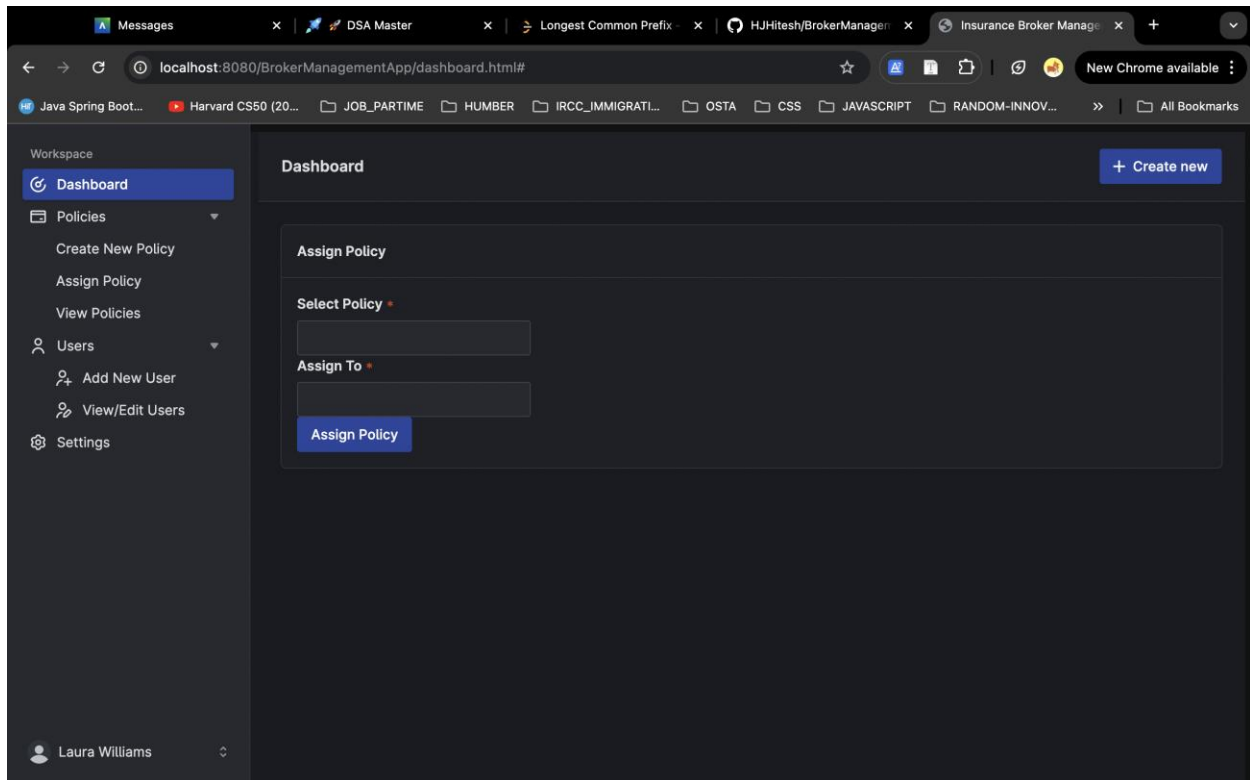
Confirm Password

confirm your password

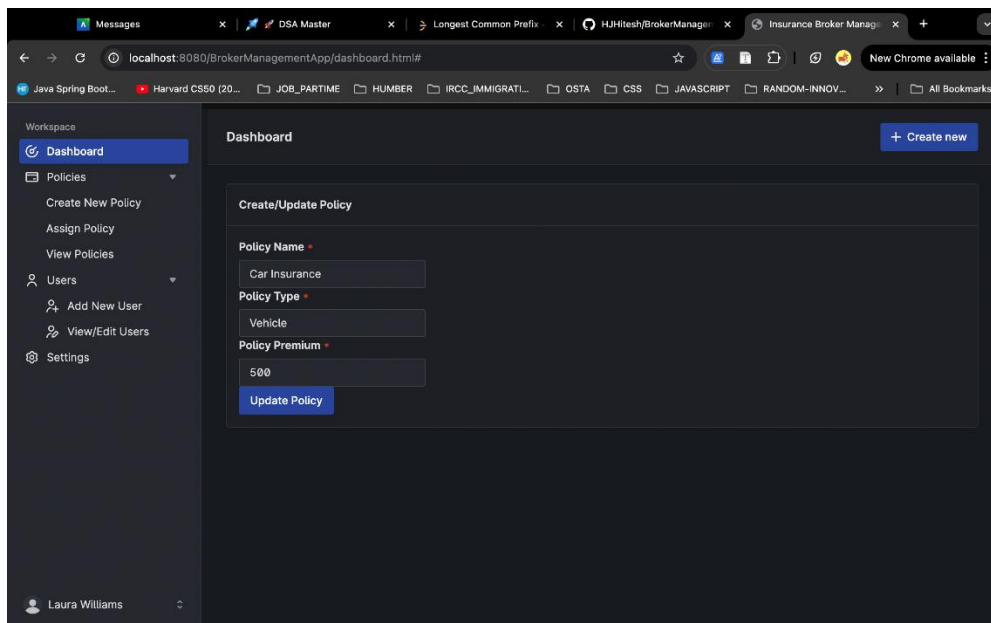
Register

Already an user? [Login](#).

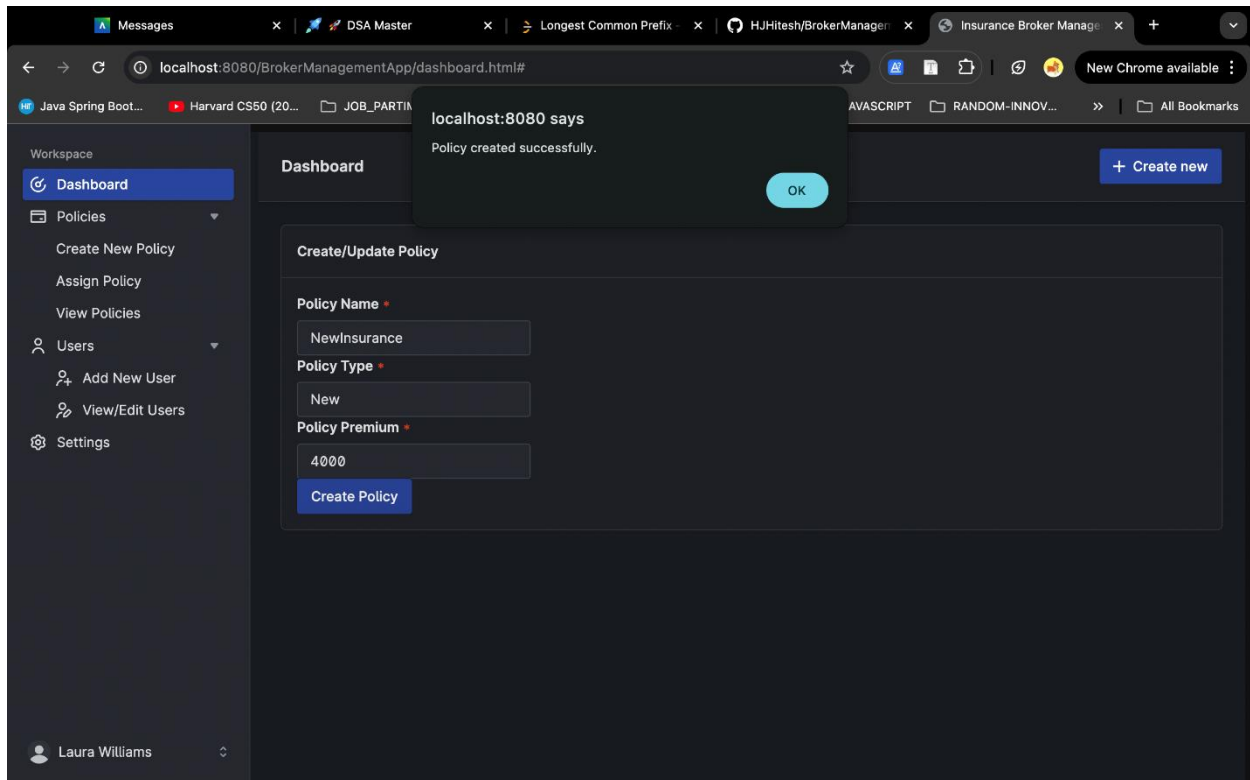
Assign policy



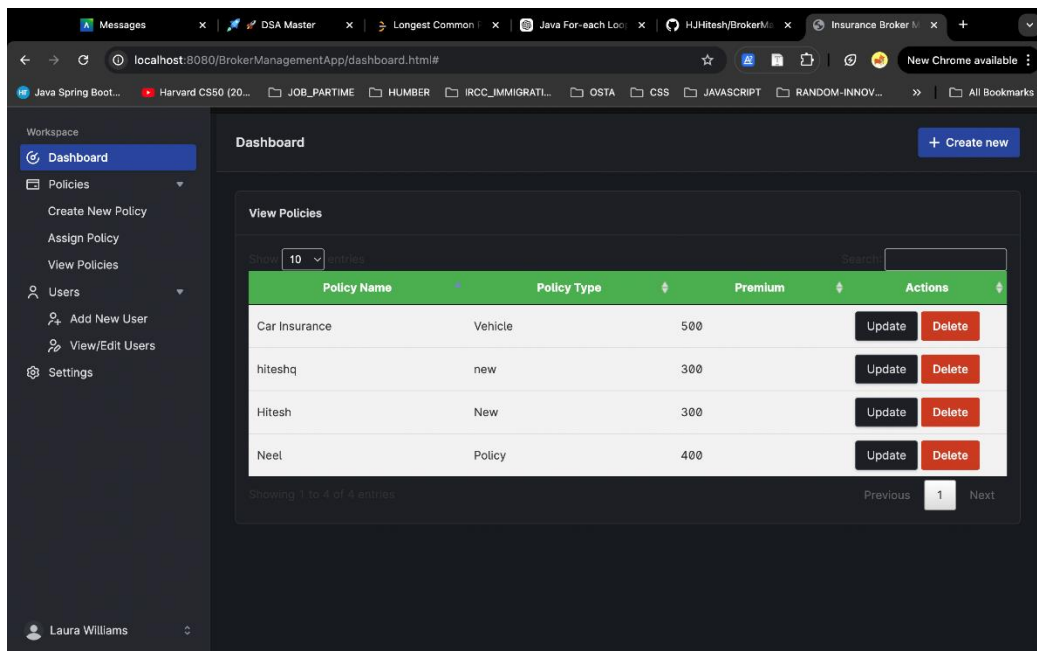
Create policy



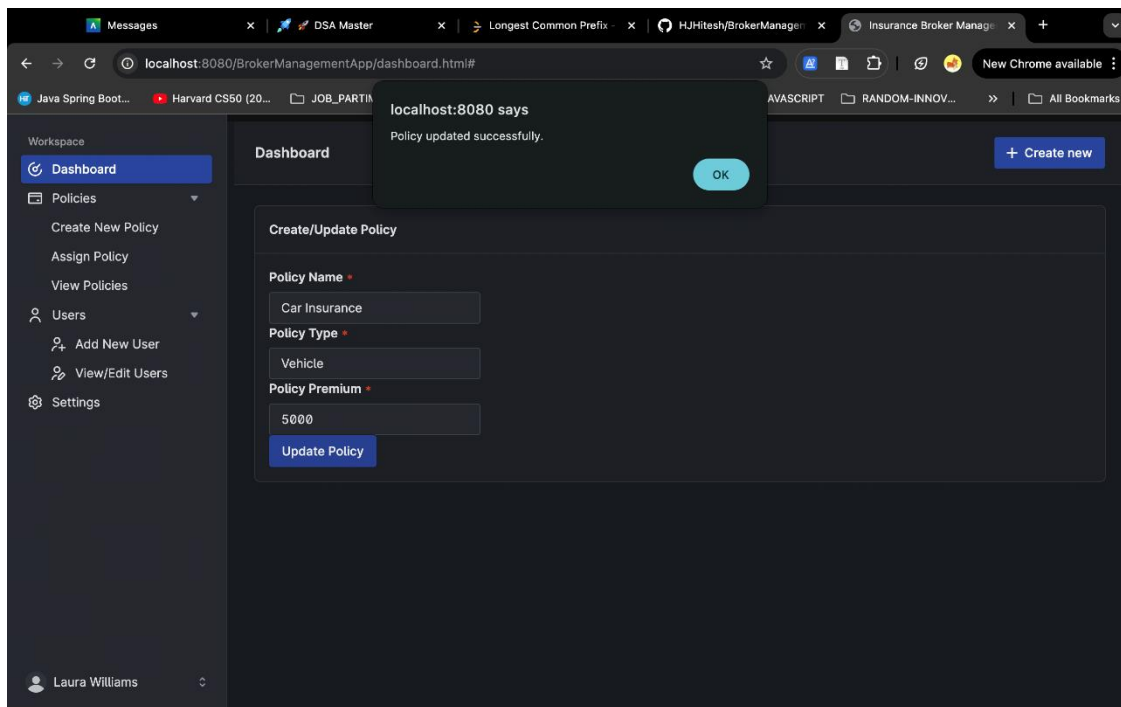
Policy create success



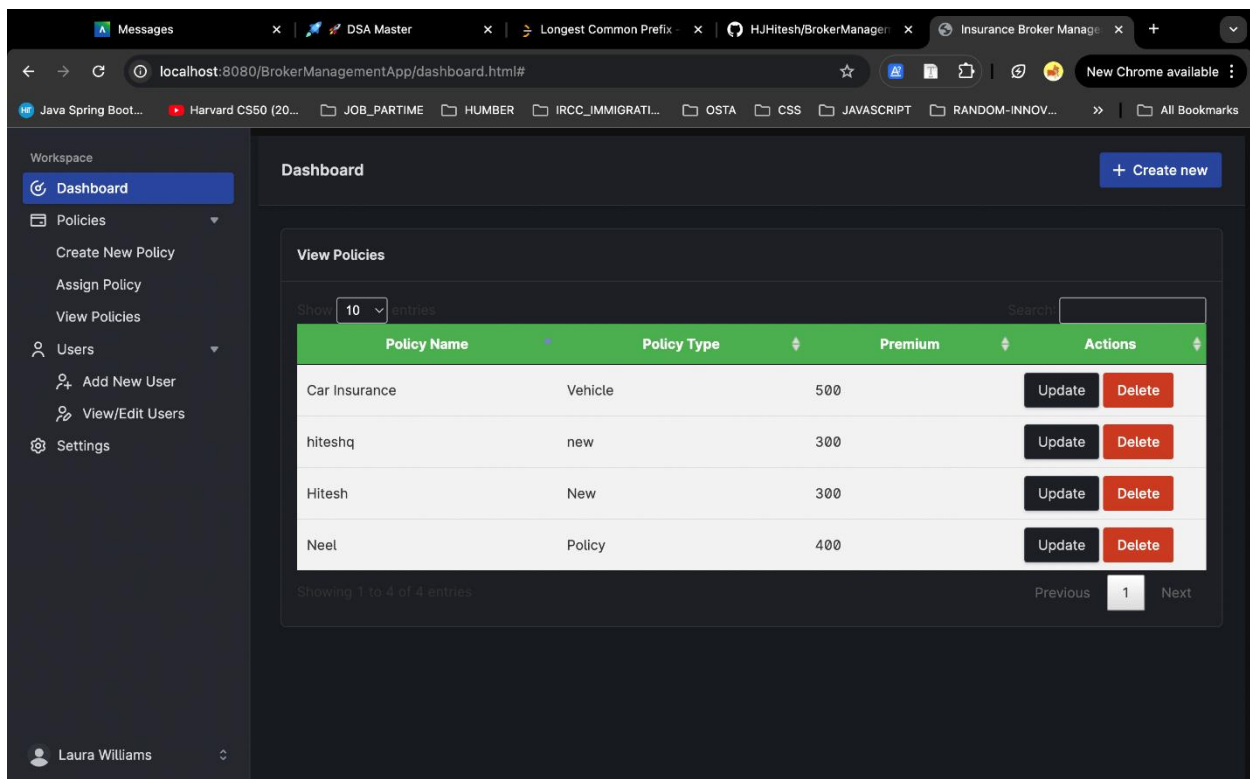
List Policy Delete



Update policy



View policy



Add new user

The screenshot shows the 'Add New User' form in the Insurance Broker Management App dashboard. The form is located in the main content area, titled 'Add New User'. It contains three input fields: 'Username' with the value 'Riya', 'Email' with the value 'riya@gmail.com', and 'Phone Number' with the value '66666666'. Below the input fields is a blue 'Add User' button. The dashboard sidebar on the left shows the 'Users' menu expanded, with 'Add New User' selected. The top navigation bar includes a 'Create new' button.

Dashboard

+ Create new

Add New User

Username *

Riya

Email *

riya@gmail.com

Phone Number

66666666

Add User

View User

The screenshot shows the 'View/Edit Users' table in the Insurance Broker Management App dashboard. The table is located in the main content area, titled 'View/Edit Users'. It displays a list of users with columns for Name, Email, Phone, and Actions. The table includes a search bar and a pagination control at the bottom. The sidebar on the left shows the 'Users' menu expanded, with 'View/Edit Users' selected. The top navigation bar includes a 'Create new' button.

Dashboard

+ Create new

View/Edit Users

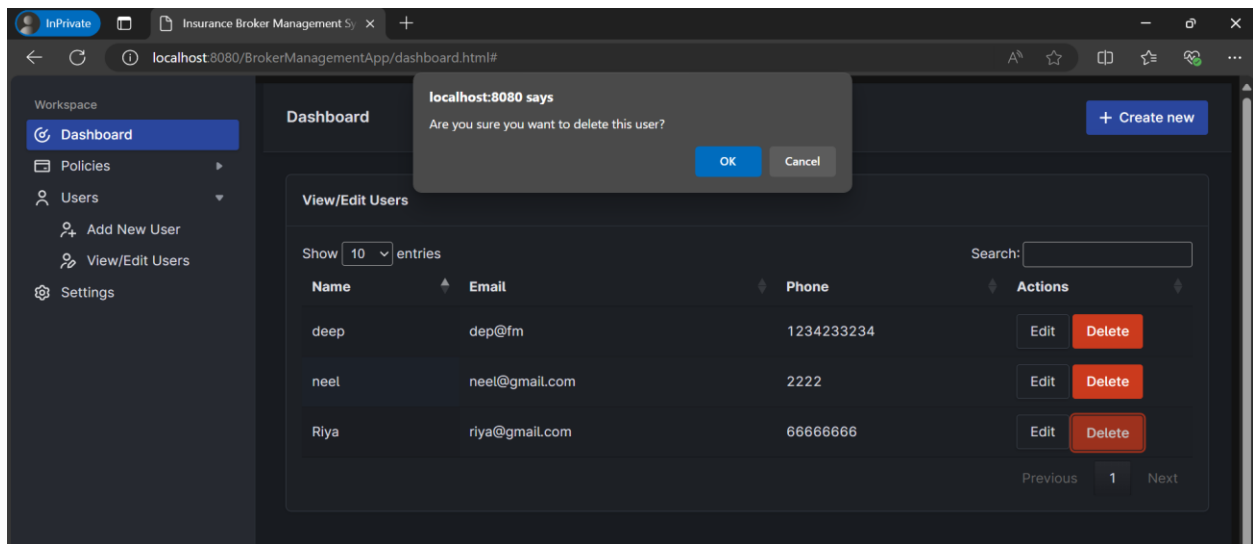
Show 10 entries

Search:

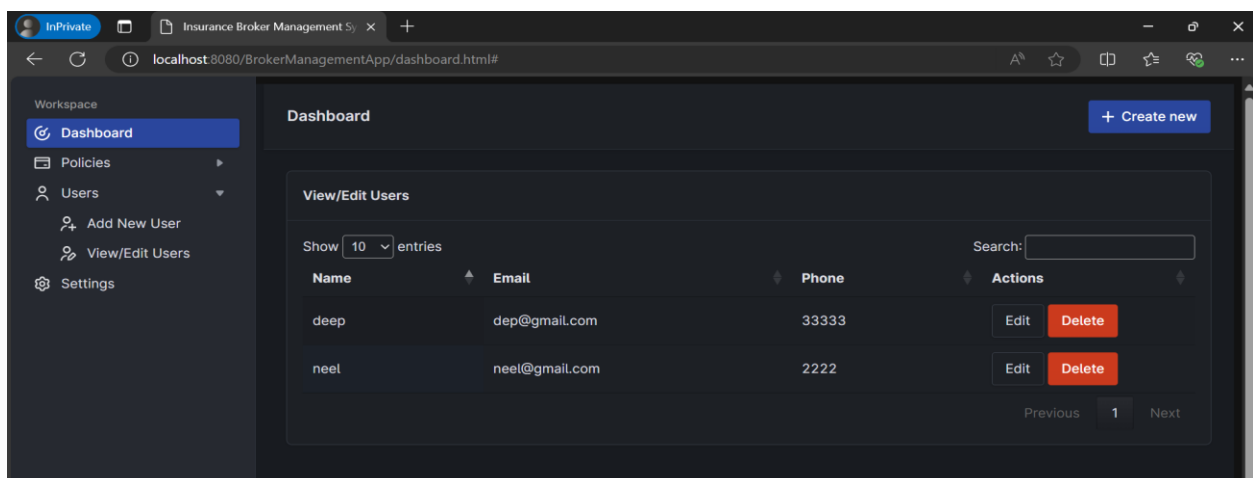
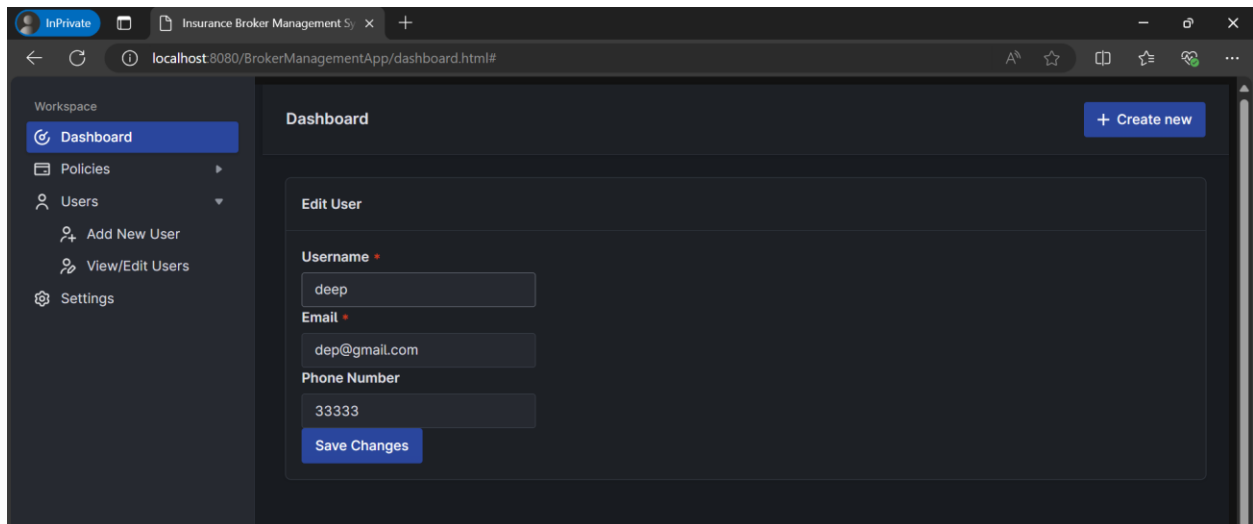
Name	Email	Phone	Actions
deep	dep@fm	1234233234	Edit Delete
neel	neel@gmail.com	2222	Edit Delete
Riya	riya@gmail.com	66666666	Edit Delete

Previous 1 Next

Delete User



Edit User

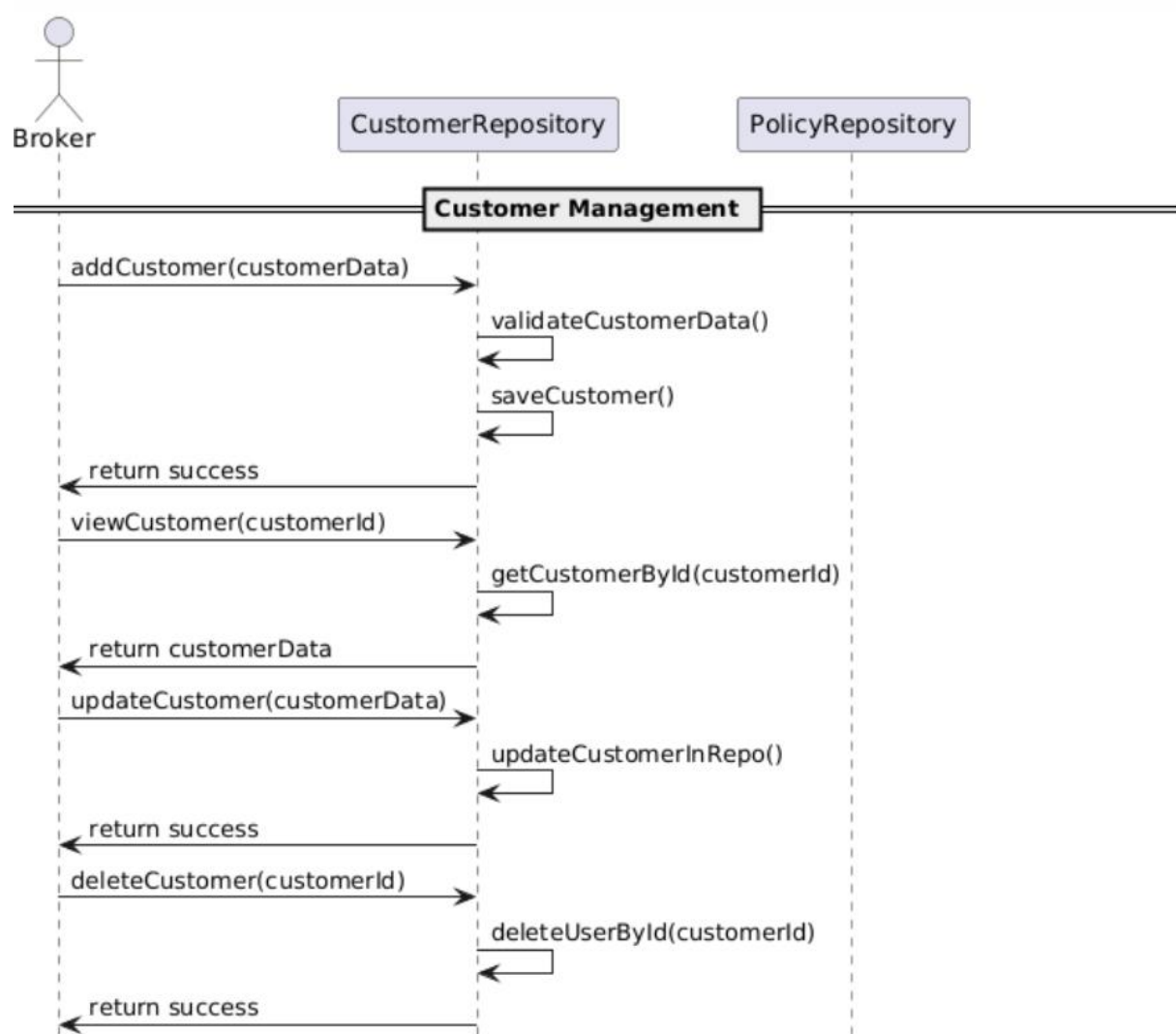


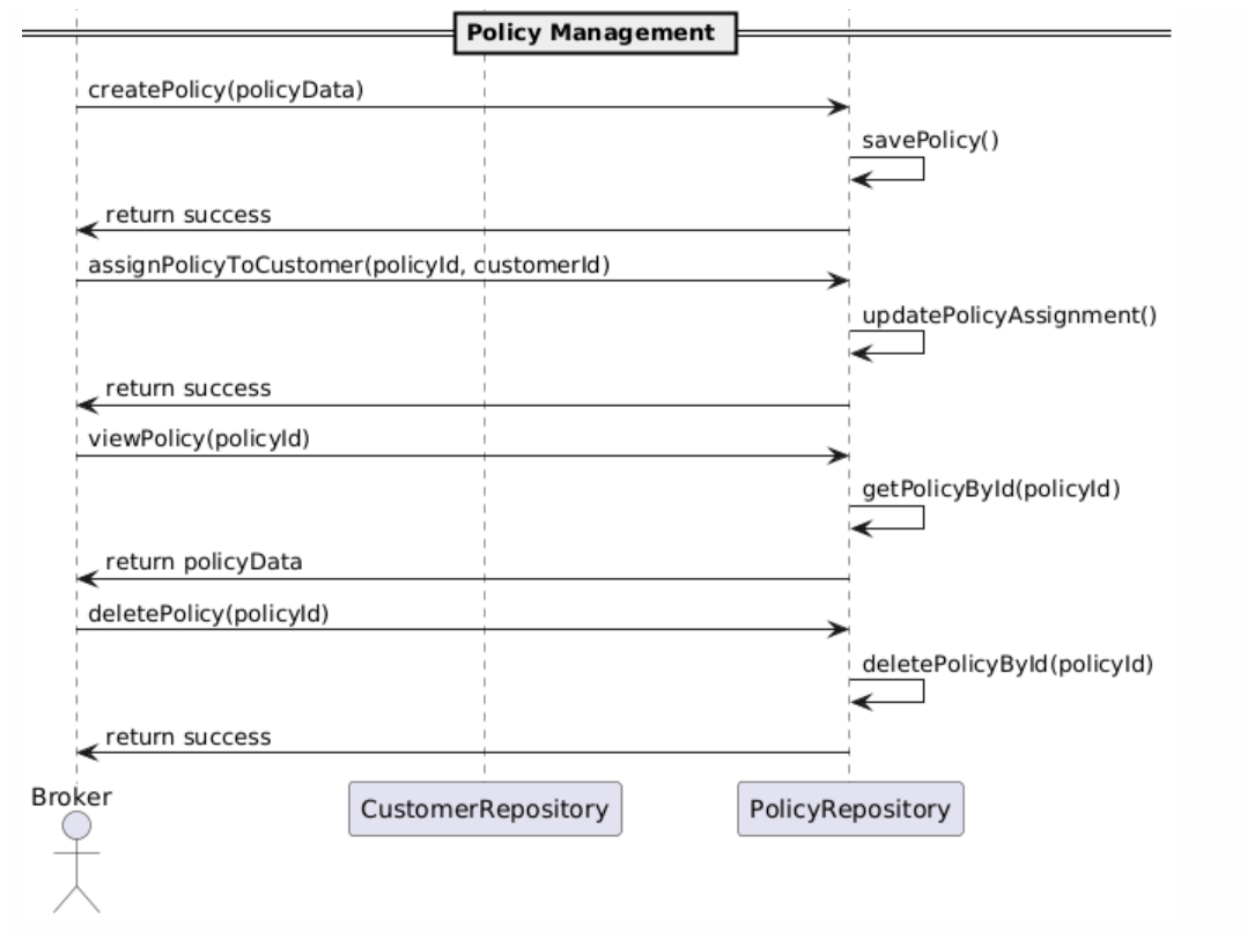
UML Diagrams

Class Diagram



Sequence Diagram





Collaboration Details

The team effectively collaborated using GitHub for version control and task management. Each team member was responsible for specific modules within the system, ensuring a clear division of responsibilities and smooth integration of the various components:

- Neel worked on the user management module.
- Hitesh developed the policy management features.
- Abdul Mubeen was responsible for the front-end development.

To ensure parallel development, we adopted a feature-based branching strategy. Each team member worked on their own designated branch:

- Neel: feature/neel-branch
- Hitesh: feature/hitesh-code
- Abdul: feature/front-end

This branching approach allowed us to work on individual features without interfering with others' work. Once a feature was completed, team members submitted pull requests to merge their code into the main branch. The pull request process included a thorough code review to ensure smooth integration and maintain high-quality standards. This workflow minimized conflicts, streamlined code merging, and maintained a consistent codebase.