Project Name: Claims Processing System

Client Name: Statewide Insurance Group (In USA)

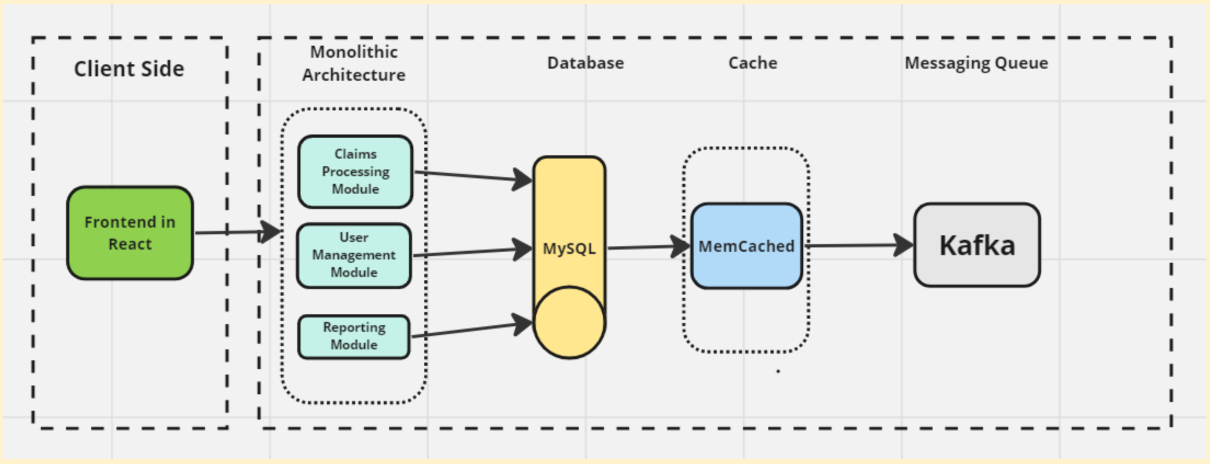
Disclaimer: This document uses fictional names for the project and client for illustrative and educational use only. Any resemblance to real entities is coincidental.

1) Introduction

Claims Processing System for Statewide Insurance Group is a centralized platform designed to manage and process insurance claims efficiently. Situated in the United States, this system aims to streamline the claims handling process by consolidating data from various internal and external sources such as client databases and insurance database.

The system is designed to provide real-time updates on claim status, automate the evaluation of claims through preset rules, and provide detailed reporting features for insurance agents and clients. The design is based on a monolithic architecture to facilitate ease of deployment, simpler scalability options, and straightforward maintenance.

2) Project Architecture



i) Client Side:

We are using angular 10 in the frontend part of this project and we have a separate frontend developers' team

ii) Monolithic Backend:

This application utilizes a single, unified codebase where all functionalities are managed and deployed together. Here are some key components:

- **Claims Processing Module**: It Manages the validation, processing, and status updates of insurance claims.

- **User Management Module**: It Handles authentication, user account management, and role-based access control.
- **Reporting Module**: It Generates detailed claim reports and analytics.

iii) Database:

A **MySQL** database is used for robust data management and complex queries required by insurance data handling.

iv) Cache Mechanism:

We are using Redis cache as an in-memory cache solution for storing frequently accessed data, such as insurance policy details, user profiles, or claim statuses

v) Logging Tools:

Using **Log4j2,** it provides powerful logging capabilities for Java applications and integrated **Splunk**, it is a powerful platform for searching, analysing, and visualizing machine-generated data, including logs, in real-time

vi) Security:

Implemented **OAuth** for secure access control. In Spring Boot, it is a way to let people sign into an app using their accounts from different places.

vii) Deployment and Operations:

- **Containerization:** Using Docker to containerize the microservices, making them easy to deploy and scale.
- **Orchestration:** Using Kubernetes to manage these containers, handling deployment, scaling, and operation of application containers across clusters of hosts.
- **CI/CD:** Continuous Integration and Continuous Deployment are achieved through Jenkins, automating the build, test, and deployment processes.
- **Git Collaboration Tool**: Using GitLab

viii) Messaging system:

- Using **Kafka** for operations that require real time updating such as sending notifications to users or systems about the status of claims, such as when a claim is received, processed, approved, or denied.

3) Project Flow

Step 1: Claim Submission and Authentication

- **Action**: An insurance agent logs into the system to submit a new claim. User cannot access this application, only insurance agent serves the user and share their details.
- **Process**:
    - **User management module** handles the login request. It authenticates the user credentials against the database.
    - Upon successful authentication, it generates an access token (using OAuth) which the user will use for subsequent requests, ensuring secure sessions.
    - For any request hit (In the form of API) by an agent, the same access token will be sent to the server for authorization

Step 2: Claim Validation and Processing

- **Action**: Automatic validation and processing of the submitted claim.
- **Process**:
    - The Claims Processing Module checks the validity of the claim based on predefined rules and past data.
    - Valid claims are forwarded for further processing and calculation of settlements.

Step 3: Reporting and Feedback

- **Action**: Generation of final claim reports and feedback to stakeholders.
- **Process**:
    - The Reporting Module compiles detailed reports on the claim, which can include decision reasoning and settlement amounts.
    - Reports are made available to agents and can be sent to clients via email.

Common activities across all the steps:

- Throughout all steps, all data transactions and user actions are logged and can be seen on **Splunk (Logging Tool)**
- For all the operations that require real time data updating, such as status of claims, **Kafka** is being used.

4) Use of Kafka in this Project:

While the main project is based on a monolithic architecture, **Apache Kafka** is employed for managing data streams and real-time event processing:

- **Description**: Kafka is used to handle real-time notifications and communications between different parts of the system, especially for events that require immediate action, such as updates to claim status or alerts.
- **Implementation**: Kafka topics are set up to segregate different types of messages, such as `claim-updates`, where notifications about changes in the claim processing

stages are published. This allows for asynchronous processing and reduces the load on the main application server by handling real-time data efficiently.

5) Use Log4j2 in the Project .

Log4j2 provides powerful logging capabilities for Java applications, offering various features such as asynchronous logging, structured logging, and dynamic configuration.

- First, we excluded the default logging dependency (**Spring Boot Starter Logging**) that comes with Spring Boot, which includes **Logback**, and then we added dependencies for **Log4j2** in our POM file.
- Then, we configured the **Yml file** in the resources

6) Integration with Splunk

Splunk is a powerful platform for searching, analysing, and visualizing machine-generated data, including logs, in real-time.

Sending Logs to Splunk:

- **Log4j2** is configured to send logs directly to Splunk using Splunk Universal Forwarder.
- Configured Log4j2 to include additional metadata in log messages, such as host information, to provide context for log analysis in Splunk.

7) Use of Redis cache:

Redis serves as an in-memory cache solution for the Claims Processing System, significantly enhancing the system's efficiency by storing frequently accessed data. This reduces the need to repeatedly query the database, thereby speeding up data retrieval and decreasing server load.

Where are we using Redis Cache:

- **Claim Status Updates:** Store the latest statuses of claims in the Redis cache to provide real-time updates to the frontend without querying the database repeatedly. This is particularly useful during high-volume periods when numerous users check claim statuses simultaneously.
- **Frequently Accessed Reports:** Cache generated reports that are frequently accessed, such as daily claims summaries or high-priority claim details. This ensures that

repeated requests for the same reports can be served quickly without regenerating them from scratch each time.

8) Use of CI/CD Pipeline in this project:

I. **Version Control System**: We used Git, which is a system that helps developers manage and keep track of changes in their code. It's hosted on websites like GitHub or GitLab. This system allows developers to save different versions of their work (called "commits") and use branches to work on new features or fix bugs without affecting the main code.
II. **Continuous Integration Tools**: We chose Jenkins, an automation tool, to help manage the process of automatically checking and merging the code developers submit. Jenkins uses a special file called a Jenkinsfile to manage these tasks, which makes it easier to update and maintain the process.
III. **Build Automation**: Whenever someone submits new code (a commit), Jenkins starts a series of actions: it compiles the code, runs tests to check if the code works properly, and analyzes the code for any potential errors. We use tools like Maven or Gradle, which are specifically designed for Java applications, to handle these tasks and manage any software libraries the code depends on.
IV. **Artifact Repository**: After Jenkins successfully builds the code, it produces files (like JAR files for Java applications). These files are stored in a special storage area called an artifact repository (like Nexus or Artifactory). This repository keeps all the different versions of the software, which helps if we need to go back to a previous version.
V. **Continuous Deployment**:
   a. **Containerization**: We use Docker to package the application with everything it needs (like libraries and other dependencies) into a container. This ensures the application works the same way in different computing environments.
   b. **Orchestration**: Kubernetes is used to manage these containers across different servers. It helps with balancing the load, scaling up or down as needed, and managing any failures.
   c. **Deployment Strategies**: To update the application with minimal disruption, we use techniques like blue-green deployments or canary releases. These methods allow us to test new versions in the real environment without affecting the existing system and switch back easily if something goes wrong.

9) Data Layer:

We are using **MySQL** for our data management needs, which supports complex queries essential for insurance data processing. The major tables and their primary fields include:

I. Claims Table:

- `claim_id` (Primary Key)
- `user_id` (Foreign Key)
- `claim_date`
- `claim_amount`
- `claim_type` (e.g., auto, home, life)
- `claim_status` (e.g., submitted, processed, approved, denied)
- `last_updated`

II. Users Table:

- `user_id` (Primary Key)
- `username`
- `password_hash`
- `email`
- `role` (e.g., agent, adjuster, administrator)
- `created_at`
- `updated_at`
- `last_login`
- `status` (e.g., active, inactive)

III. Transactions Table:

- `transaction_id` (Primary Key)
- `claim_id` (Foreign Key)
- `transaction_date`
- `amount`
- `transaction_type` (e.g., payout, adjustment)
- `description`

IV. Audit Logs Table:

- `log_id` (Primary Key)
- `action`
- `timestamp`
- `user_id` (Foreign Key)
- `details` (description of the action taken, changes made, etc.)

V. Payment Details Table:

- `payment_id` (Primary Key)
- `claim_id` (Foreign Key)
- `payment_date`
- `amount`
- `payment_method` (e.g., check, bank transfer)
- `status` (e.g., pending, completed)

10) REST APIs:

Claims Management

- POST /claims - Submit a new insurance claim.
- GET /claims/{claimId} - Retrieve details about a specific claim using its ID.
- PUT /claims/{claimId} - Update an existing claim (e.g., change status or update claim details).
- DELETE /claims/{claimId} - Delete a specific claim.
- GET /claims/user/{userId} - Retrieve all claims submitted by a specific user.

User Authentication and Management

- POST /users/register - Register a new user account.
- POST /users/login - Authenticate a user and return a JWT.
- GET /users/{userId} - Get details of a specific user.
- PUT /users/{userId} - Update user profile information.
- DELETE /users/{userId} - Deactivate or delete a user account.

Reporting

- GET /reports/claims/status - Generate a report on claims based on their status (e.g., pending, approved, denied).
- GET /reports/claims/summary - Generate a summary report of claims over a specified period.

Notifications and Real-time Updates

- POST /notifications/subscribe - Subscribe a user or device to real-time updates via WebSocket or other means.
- POST /notifications/unsubscribe - Unsubscribe from real-time notifications.

Administrative and Support

- GET /admin/logs - Access system logs for maintenance and auditing purposes.

11) What to add in your resume/CV?

Claims Processing System, Statewide Insurance Group USA

Description: It is a centralized platform designed to efficiently manage and process insurance claims, incorporating real-time updates and automation to streamline the claims handling process.

Responsibilities:

- Worked as a backend developer with springboot and java as a core technology
- Developed real-time claim status updates to enhance transparency and client satisfaction.
- Working on the REST APIs and developing business features.
- Implemented Oauth for secure user authentication, improving system security.
- Integrated Redis cache for efficient data caching, significantly boosting application performance and reducing database load.
- Contributed to the development of a monolithic architecture, also deployment and maintenance processes.

12) What to speak in front of your interviewer?

Recently I worked on Claims Processing System by Statewide Insurance Group situated in USA, it is a centralized platform designed to efficiently manage and process insurance claims, also real-time updates and automation to the claims handling process.

Here, we used Spring Boot and Java to add features like instant claim status updates, so clients got quick assessments. we also made sure different parts of our system could talk to each other smoothly with REST APIs. Security was important, so we set up safe logins to protect user data. And to make everything faster, we used redis cache, which stores important data in memory.