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:

The front-end part of this project is developed using **React 17**, managed by a dedicated team of frontend developers.

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**: Manages the validation, processing, and status updates of insurance claims.
- **User Management Module**: Handles authentication, user account management, and role-based access control.
- **Reporting Module**: 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:

Memcached is employed as an in-memory caching solution to enhance performance by storing session data and frequently accessed user data.

v) Logging Tools:

Logback is used for logging, given its seamless integration with the Spring framework, complemented by **Elasticsearch** for log storage and analysis.

vi) Security:

JWT (JSON Web Tokens) is used for secure authentication and authorization within the system.

vii) Deployment and Operations:

- **Containerization**: Utilizing **Docker** to package the application, making it portable and consistent across different environments.
- **CI/CD**: Using **GitHub Actions** for continuous integration and deployment, automating the code integration and deployment pipeline.
- **Version Control**: Managed with **GitHub**, facilitating collaboration and version control.

3) Project Flow

Step 1: Claim Submission and Authentication

- **Action**: An insurance agent logs into the system to submit a new claim.
- **Process**:
  - Authentication via JWT ensures secure access.
  - The agent submits the claim through the user interface.

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.

4) Use of Memcached in this Project:

Memcached is a tool that helps websites run faster by storing bits of information that are used often. This means the website doesn't have to keep asking the database for this information, making everything much quicker. It's like keeping your most-used tools on top of your desk instead of in a drawer.

Memcached provides a fast, in-memory caching layer:

- **Description**: Used to store session data and quick-access data such as user profiles and frequent queries.
- **Implementation**: Integrated directly with the application code to reduce database load and improve response times.

5) Integration with Elasticsearch:

Elasticsearch allows for robust searching, analyzing, and visualizing of logs in real time:

- **Sending Logs to Elasticsearch**: Configured within the application to forward logs directly to Elasticsearch for real-time analysis and visualizations.

6) 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)

7) Deployment Strategy:

Containerization and Orchestration:

- **Docker** is used to containerize the entire application, ensuring that each deployment is consistent and isolated.
- For local development and testing, **Docker Compose** is used to manage multi-container setups and simplify the orchestration process.

CI/CD:

- **GitHub Actions** is utilized to automate the continuous integration and deployment pipeline. It manages the testing, building, and deployment of the application whenever changes are pushed to the repository.
- Deployment strategies such as **rolling updates** are implemented to ensure minimal downtime and seamless deployment of new versions.

8) Security Measures:

Security is a critical aspect of the Claims Processing System, especially concerning data privacy and protection:

- **JWT (JSON Web Tokens)**: Used for secure and stateless authentication across the system.
- **Role-Based Access Control (RBAC)**: Ensures that users only have access to functionalities relevant to their roles.
- **Data Encryption**: Sensitive data such as user credentials and payment information are encrypted using industry-standard encryption algorithms.

9) 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.

10) Use of Logback in the Project:

Logback is chosen for logging due to its native integration with the Spring framework, which provides robust logging capabilities:

- **Configuration**: Unlike other logging frameworks that may require extensive setup, Logback is directly supported by Spring Boot, facilitating easier configuration and management.
- **Performance**: It offers high-speed logging capabilities and minimal memory usage, making it ideal for a system where reliability and performance are critical.
- **Integration with Elasticsearch**: Although Elasticsearch is used for log analysis, Logback ensures efficient logging and management of log files. Logs generated by Logback can be pushed to Elasticsearch where they are indexed and made searchable for more detailed analysis and real-time monitoring.

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 JWT for secure user authentication, improving system security.
- Integrated Memcached for efficient data caching, significantly boosting application performance and reducing database load.
- Contributed to the development of a monolithic architecture, streamlining 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, incorporating real-time updates and automation to streamline the claims handling process.

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

13) Project Related All Possible Interview Questions and Answers

Which Java version are you using in this project?

We are using Java 14 for this project.

Why did you choose Java 14 for this project?

Java 14 offers several advanced features that enhance productivity and system performance, such as new APIs for the JDK, better memory management, and improved performance features, making it ideal for handling the complexities of a claims processing system.

Can you name some significant features that were introduced in Java 14?

Some of the noteworthy features introduced in Java 14 include the Records feature, which simplifies the modeling of data as it reduces boilerplate code, Pattern Matching for instanceof, which simplifies conditional extraction of components from objects, and the Helpful NullPointerExceptions that provide better exception messages.

Which Spring Boot version are you using in this project?

We are using Spring Boot version 2.3.1 in this project.

What improvements does Spring Boot version 2.3.1 offer?

Version 2.3.1 includes several enhancements such as graceful shutdown support, improved Docker container integration, layering for Docker images, and liveness/readiness probes for better Kubernetes integration.

Can you explain the monolithic architecture you used in the Claims Processing System? Why did you choose this architecture?

We chose a monolithic architecture for the Claims Processing System because it simplifies deployment, debugging, and testing processes. This architecture is particularly beneficial for our application as it simplifies the interaction between different components involved in claims processing, such as claim validation, user management, and report generation.

How do you ensure data consistency in your monolithic architecture?

Data consistency is ensured by using transactions within our MySQL database, which maintains ACID properties across different data manipulations. Additionally, we use Spring Data JPA's transaction management capabilities to handle transactional integrity.

What databases did you choose for this project and why? How did you handle data modeling?

We chose MySQL for its robustness and its ability to handle complex queries, which are essential for claims data processing. Data modeling was managed by clearly defining database schemas that support efficient querying and data integrity.

Can you describe how you implemented security measures in the Claims Processing System? Detail the authentication and authorization strategies.

Security is implemented using JWTs for both authentication and authorization. JWTs provide a compact, URL-safe means of representing claims to be transferred between two parties, helping to ensure that only authorized users can access the system and that their interactions are securely managed.

What role does Kafka play in this project, and what are its primary functions?

In this project, Kafka is used primarily for handling asynchronous messaging and notifications related to claim status updates. This allows for real-time communication within the system without impacting the performance of the main application.

Explain a scenario where real-time processing is critical in your application and how Kafka supports this requirement.

Real-time processing is critical during the claim status update process. For example, when a claim is approved or rejected, Kafka facilitates real-time notifications to the relevant parties

such as claims adjusters and clients, ensuring that they are promptly informed about the status changes.

What are some of the best practices you follow while using Kafka in your projects?

Best practices include defining clear topic strategies, ensuring proper partitioning and replication of topics for fault tolerance, and monitoring Kafka performance continuously to handle scalability and throughput effectively.

Can you explain the use of React in this project? Why was it chosen for the frontend?

We chose React for the frontend due to its efficient handling of updates with the Virtual DOM, which is particularly beneficial for our dynamic user interfaces that involve real-time data, such as claim status updates. React's component-based architecture also helps in reusing UI components across the application, promoting a clean and maintainable codebase.

What are the benefits and drawbacks of using Memcached in your system?

Benefits: Memcached offers a simple yet powerful caching mechanism that reduces database load by caching frequently accessed data, thereby improving response times significantly.

Drawbacks: Memcached's cache is volatile, meaning data can be lost if it crashes, and it doesn't support data persistence. Additionally, managing cache consistency across multiple instances can be challenging.

How do you handle session management in your monolithic architecture?

Session management is handled using server-side sessions stored in Memcached. This allows us to maintain user state across multiple requests while ensuring quick access to session data due to Memcached’s performance.

Describe the security measures taken to protect data within MySQL.

Security measures for MySQL include encryption at rest using built-in MySQL features, enforcing SSL/TLS for data in transit, regular updates and patches to the MySQL system, and strict access controls with role-based permissions to protect sensitive data from unauthorized access.

Can you describe how logging is implemented in your project? What makes Logback suitable for this project?

Logging in our project is implemented using Logback because of its high performance, flexibility in log management (like file rolling and archiving), and ease of configuration.

Logback allows for fine-grained control over log levels and outputs, which is crucial for monitoring and debugging the application effectively.

What strategies are used to handle large data loads and maintain performance in your system?

To handle large data loads and maintain performance, we use techniques such as database indexing, query optimization, and load balancing across multiple server instances. Additionally, heavy computations are offloaded to background processes where possible to keep the user interface responsive.

How is the CI/CD pipeline configured for this project? Which tools and practices are integral to this process?

The CI/CD pipeline is configured using GitHub Actions for continuous integration and deployment. Key practices include automated testing, code reviews, and automated deployments to staging environments. This setup ensures that new changes are thoroughly tested and deployed smoothly without disrupting the existing functionality.

Explain how JWT authentication enhances the security of the Claims Processing System.

JWT authentication enhances security by ensuring that tokens are signed and verifiable, meaning the server can confirm the authenticity of tokens without needing to query the database constantly. This stateless authentication mechanism is not only secure but also scales well with our application's architecture.

Can you discuss any specific challenges you faced with this project and how you addressed them?

One challenge was managing complex business logic related to claims processing, which required numerous rules and exceptions. We addressed this by implementing a domain-driven design, encapsulating business logic in the domain model, and using service layers to handle business operations, which improved maintainability and testing.

How do you ensure the application's resilience and fault tolerance, especially considering its monolithic architecture?

We ensure resilience and fault tolerance by implementing extensive error handling, using retries for transient failures, and fallback methods for critical processes. Regular stress testing and capacity planning also help us prepare for unexpected spikes in load or failures.

What methodologies are employed to ensure quality assurance throughout the development lifecycle?

Quality assurance methodologies include automated testing (unit, integration, and system tests), code reviews, and manual testing stages. We also utilize static code analysis tools to detect potential issues early in the development cycle.

How do you manage changes to the database schema without affecting existing data?

xChanges to the database schema are managed using migration scripts with tools like Flyway. These scripts are carefully designed to alter the database structure without data loss and are tested in staging environments before being applied to production.

Describe the lifecycle of a claim in the Claims Processing System.

The lifecycle begins when an insurance claim is submitted through the front end. It then goes through initial validation checks for completeness and accuracy. If validated, the claim is processed where calculations and assessments are performed based on the policy details. Following approval or denial, the result is communicated back to the submitting party, and relevant financial transactions are initiated if the claim is approved.

How do you handle rollback operations in case of a process failure during claim processing?

Rollback operations are managed through transaction management in MySQL. If a process fails at any stage, the transaction is rolled back to ensure data integrity and consistency. Additionally, we use compensatory transactions to reverse any intermediate steps if a complete rollback isn't feasible.

What measures do you take to ensure the scalability of the Claims Processing System?

We ensure scalability by designing the system with scalable components from the start, such as using load balancers to distribute incoming traffic and employing database partitioning to manage large datasets efficiently. Scalability tests are conducted regularly to identify bottlenecks and address them proactively.

Can you discuss the integration of third-party services in the system? How do you manage and monitor these integrations?

Third-party services, such as payment gateways and external data providers, are integrated using APIs. We manage these integrations through dedicated service classes that handle API communications, and we monitor the integrations using centralized logging and performance metrics to quickly identify and resolve issues.

How do you manage data privacy and compliance in the Claims Processing System?

Data privacy and compliance are managed by adhering to regulatory requirements like GDPR and HIPAA where applicable. This involves implementing data encryption, securing

data transmissions, and ensuring that data access is logged and auditable. We also conduct regular compliance audits and privacy assessments.

Explain how feature updates are handled in your monolithic application. How do you minimize downtime?

Feature updates are carefully planned and implemented using strategies like feature flags to decouple deployment from release, allowing us to deploy new features without exposing them immediately. For updates requiring downtime, we use blue-green deployments to minimize impact, ensuring that there is always a live version available to users.

How is user feedback incorporated into the development cycle of the Claims Processing System?

User feedback is gathered through multiple channels like direct user interviews, support tickets, and usability testing. This feedback is prioritized and incorporated into the product backlog, where it is considered during sprint planning to ensure continuous improvement of the system based on user needs.

Discuss a major technical challenge you overcame in this project and the lessons learned.

A major challenge was handling the high availability requirements for the system. We initially faced issues with service disruptions during peak load times. By implementing robust load balancing and failover strategies, along with performance optimizations, we learned the importance of proactive capacity planning and stress testing in critical system design.

What are the protocols in place for disaster recovery in the Claims Processing System?

We have a comprehensive disaster recovery plan that includes regular backups, failover mechanisms to secondary data centers, and rigorous testing of recovery procedures. These measures ensure that we can quickly restore service in case of a major incident with minimal data loss.

What was the critical feature you worked on in this project?

I worked on adding a feature that lets insurance agents and clients see updates on their insurance claims as soon as they happen. This helps everyone stay informed and makes the process smoother and more transparent. It utilized Kafka for this real time updates.

How do you ensure continuous improvement in your project management practices?

Continuous improvement is ensured by regularly reviewing our project management practices against industry standards and feedback from project retrospectives. We also invest in training for our team members on the latest project management techniques and tools to keep improving our processes,

Which methodology do you use for your project management?

We use the Agile methodology, which allows for flexible planning, progressive development, early deployment, and continuous improvement.

How does the Agile approach help in managing the development and deployment of microservices in your project?

Agile provides a framework that supports the dynamic and complex nature of managing multiple microservices, allowing for regular feedback, iterative development, and rapid adjustments as needed.

Describe how you structure sprints in your project. What is the typical duration of a sprint, and how are tasks prioritized?

Sprints are typically two weeks long. Tasks are prioritized based on their business impact and urgency, guided by the product owner in collaboration with the team.

Which Agile framework (Scrum, Kanban, etc.) do you use in your project, and why was it chosen?

We use Scrum because of its structured approach to managing large projects, its emphasis on regular updates, and its ability to handle complexity through roles like Scrum Master and Product Owner.

How are user stories created and maintained for your project? Who is responsible for writing these stories?

User stories are created by the product owner with input from stakeholders and the development team. They are maintained in a product backlog, which is regularly groomed and prioritized.

How is testing integrated into your Agile process? Are there dedicated sprints for testing, or is it continuous?

Testing is continuous throughout the development process. Each sprint includes development and testing tasks, ensuring that new features are both developed and tested within the same sprint.

How did you send the mails by using java mail service ?

First, I ensure the Spring Boot Starter Mail dependency is in my project's pom.xml. Next in application.properties, I set up my mail server details, like host, port, username, and password. Then I write a service class that uses JavaMailSender to send emails.

In this service, I craft the email content and use the send method to dispatch emails. And finally, I call my mail service from within the registration logic to send the email.

Can you tell me How did you integrate a MySQL in your project?

First we have added the MySQL dependency in our project's POM file. Then, in our application properties, we set up the connection details for MySQL, like the database URL and credentials.

Then we created repository interfaces in our code using Spring Data, which helps in interacting with MySQL.

Finally, we use these repositories in our services to save, retrieve, and manage data in our MySQL database.