**Low-Level Design for Insurance Aggregator Application**

**1.Introduction:**

Insurance aggregator applications is an online platform that helps users for comparing and purchasing insurance policies to your needs. With our user-friendly interface and powerful search capabilities, finding the right coverage for users. Whether you're looking for auto, home, health, or life insurance, our platform allows you to compare quotes from top providers, saving your time and money.

**1.1: Purpose**

The purpose of this document is to define the requirements for the development of a web-based insurance aggregator application. The application aims to provide users with a centralized platform to compare, choose, and purchase insurance policies from various providers.

**1.2: Scope**

The web-based insurance aggregator application will cover a range of insurance types, including but not limited to health insurance, auto insurance, home insurance, and travel insurance. It will facilitate the comparison of policy features, premiums, and terms from multiple insurance providers.

**1.3: Overview of the Module/Component**

**1.3.1 User Authentication Module:**

**Purpose:** This module handles user authentication and authorization Processes.

**Features**:

*User registration:* Allows users to create an account.

*Login/logout:* Enables users to securely log in and out of their accounts.

*Password management:* Provides functionality for password reset and Change.

Role-based access control: Assigns roles to users to control access to various parts of the system.

**1.3.2 Insurance Provider Integration Component**:

**Purpose**: Facilitates integration with various insurance providers' systems to fetch quotes and policy details.

**Features**:

*API Integration:* Connects to insurance providers' APIs to retrieve insurance products, quotes, and policy information.

*Data mapping:* Maps data between the aggregator's system and different providers' systems for seamless integration.

*Error handling:* Manages errors and exceptions during integration to ensure system stability.

**1.3.3 Quote Generation Module**:

**Purpose**: Generates insurance quotes based on user input and data retrieved from insurance providers.

**Features**:

*User input processing:* Collects user information such as age, location, coverage requirements, etc.

*Quote calculation:* Utilizes algorithms to calculate insurance premiums based on input data and insurance provider rates.

*Presentation:* Displays generated quotes to users in a clear and understandable format.

**1.3.4 Policy Management Module**:

**Purpose**: Handles the management of insurance policies purchased by users.

**Features**:

*Policy issuance:* Generates insurance policies for users based on their selected quotes.

*Policy storage:* Stores policy details securely for future reference and management.

*Renewal reminders:* Sends notifications to users for policy renewals and updates.

**1.3.5 User Dashboard Component**:

**Purpose**: Provides users with a centralized interface to manage their insurance-related activities.

**Features**:

*Policy overview*: Displays summary information about active insurance policies.

*Quote history:* Shows a history of quotes generated by the user.

*Policy documents:* Allows users to access and download policy documents.

**1.3.6 Admin Panel Module**:

**Purpose**: Offers administrators tools to manage users, insurance providers, and system settings.

**Features**:

*User management:* Enables administrators to create, update, and delete user accounts.

*Provider management:* Allows administrators to add, edit, or remove insurance provider integrations.

*System configuration:* Provides options to configure system settings such as API keys, rate limits, etc.

**1.3.7 Reports and Analytics Module:**

**Purpose**: Generates reports and analytics to provide insights into user behaviour, popular insurance products, etc.

**Features**:

*Data collection:* Gathers relevant data points from user interactions and system activities.

*Report generation:* Creates reports on key metrics such as quote requests, policy purchases, etc.

*Analytics dashboard:* Presents data in visual formats like charts and graphs for easy analysis.

**1.3.8 Notification System Component**:

**Purpose**: Sends out notifications to users regarding important events such as policy renewals, quote updates, etc.

**Features**:

*Email/SMS notifications:* Sends notifications via email or SMS based on user preferences.

*Customizable templates:* Allows customization of notification content for different types of events.

*Scheduling:* Supports scheduling of notifications for specific times or events.

**2.Low Level Design:**

* LLD provides detailed technical specifications for individual components or modules within a software system.
* It translates the High-Level Design (HLD) into smaller, manageable units that can be implemented by developers.
* It specifies how different modules interact with each other and defines interfaces, inputs, and outputs for seamless communication.
* LLD serves as a blueprint or roadmap for developers, guiding them through the implementation phase with clear instructions and design decisions.

**2.1 Entity Relationship Diagram:**

Entity-Relationship (ER) modelling is a method used to visualize and describe the logical structure of a database.

* User - Quotes: one-to-many (one user can request multiple quotes)
* Users - Policyholders: one-to-many (One user can be associated with multiple policyholders)
* Users -Transactions: one-to-many (one user can have multiple transaction)
* Users - Reviews and Ratings: one-to-many (Users can provide reviews and ratings for multiple insurance policies)
* Insurance Companies - Insurance Policies: one-to-many (Each insurance company can offer multiple insurance policies)
* Insurance Policies - Quotes: many-to-many (Many insurance policies can be included in quotes offered by different companies)
* Policyholders - Claims: one-to-many (Each policyholder can submit multiple claims)
* Users - Notifications: one-to-many (Users can receive multiple notifications)
* Admin - Insurance Companies: one-to-many (One admin may be associated with multiple insurance companies)
* Insurance Policies - Admin: one-to-many (one admin may manage multiple insurance policies)
* Quotes - Admin: one-to-many (One admin might oversee the generation of multiple quotes within the system)
* Policyholders - Admin: one-to-many (Each admin may manage multiple policyholders' details)
* Transactions - Admin: one-to-many (One admin may oversee multiple transactions within the system)
* Claims - Admin: one-to-many (one admin might handle multiple insurance claims)
* Reviews\_and\_Ratings - Admin: one-to-many (One admin may manage multiple reviews and ratings within the system)
* Notifications - Admin: one-to-many (Each admin may send notifications to multiple users within the system)

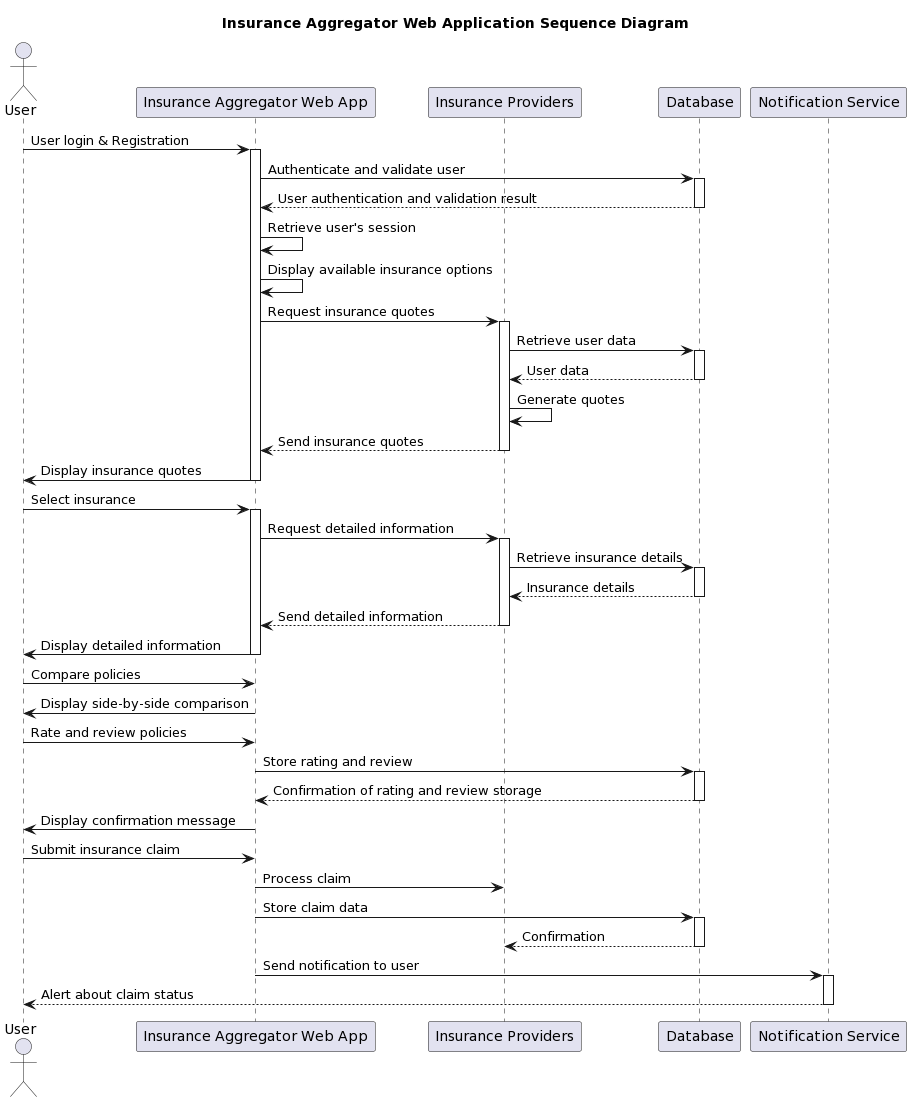
A computer screen shot of a computer

Description automatically generated

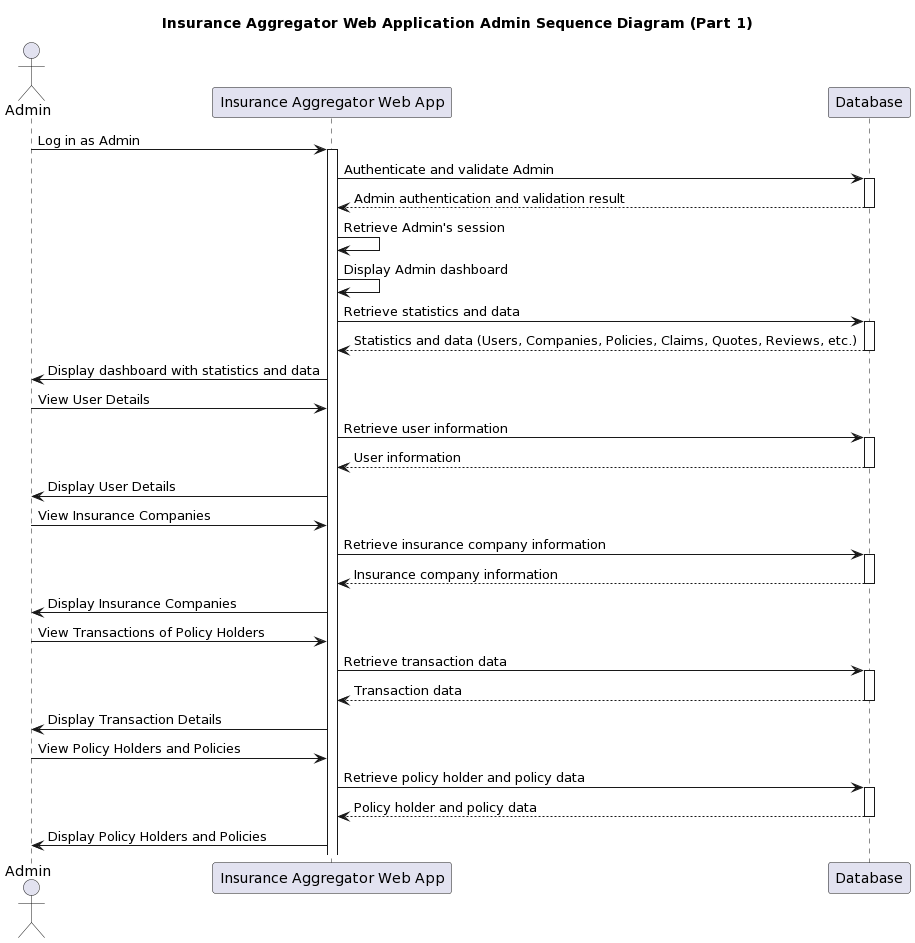
**2.2. Sequence diagram:**

A sequence diagram is a type of interaction diagram that illustrates the flow of interactions between objects or components in a system over time. Including how objects collaborate to accomplish a specific task or scenario.

**1.USER**

****

**2.Admin**

****

**A screenshot of a computer screen

Description automatically generated**

**4.Component Design:**

**4.1. Frontend (UI) Components:**

**Dashboard:** Provides users with an overview of their account information, policies, claims, and notifications.

**Policy Comparison Tool:** Allows users to compare different insurance policies based on various parameters.

**Claim Submission Form**: Enables users to submit insurance claims with necessary details.

**Notification Centre:** Displays real-time notifications related to policy updates, claim status changes, etc.

**Admin Panel:** Interface for administrators to manage users, policies, claims, and perform administrative tasks.

**4.2. Backend (Server-side) Components:**

**Authentication Service:** Handles user authentication and authorization for secure access to the application.

**Policy Management Service:** Manages insurance policies, including CRUD operations and policy-related functionalities.

**Claim Management Service:** Processes insurance claims, manages claim submissions, and updates claim statuses.

**Notification Service**: Sends notifications to users based on predefined events and triggers.

**Data Access Layer (DAL):** Provides a layer for interacting with the database, including CRUD operations and data retrieval.

**4.3. Integration Components:**

**Third-Party APIs Integration:** Integrates with external insurance providers' APIs for fetching policy quotes, details, etc.

**Payment Gateway Integration:** Integrates with payment gateways for processing premium payments, claim reimbursements, etc.

**4.4. Communication Components**:

**RESTful APIs:** Facilitates communication between the frontend and backend components using REST architecture.

**WebSocket:** Enables real-time bidirectional communication between clients and servers for interactive features.

**4.5. Testing Components:**

**Unit Tests:** Verify the functionality of individual components in isolation to ensure correctness.

**Integration Tests:** Validate the interaction between different components to ensure seamless integration.

**End-to-End (E2E) Tests:** Test the application's functionality from end to end to ensure it meets user requirements.

**5. Performance Optimization:**

**5.1 Optimization Techniques:**

Optimization techniques that can be implemented to enhance system performance across different layers and components, including database, backend, frontend, testing, and infrastructure.

**Frontend Optimization (Angular):** Implement lazy loading, bundle optimization, and server-side rendering to improve frontend performance.

**Backend Optimization (Java/Spring Boot):** Utilize caching, asynchronous processing, and optimized algorithms/data structures to enhance backend performance.

**Database Optimization:** Improve query performance through proper indexing, query optimization, database sharing, and connection pooling.

**Testing and Deployment Optimization:** Employ automated testing, CI/CD pipelines, and containerization for efficient testing and deployment.

**Infrastructure Optimization (AWS):** Use auto-scaling, CDN, and resource optimization techniques to optimize AWS infrastructure.

**5.2 Performance Testing Results:**

Performance testing conducted on various aspects of the system, such as database response time, backend request handling time, frontend page load time, and infrastructure scalability.

**Frontend (Angular):** Measure page load time and optimize DOM manipulation for improved rendering performance.

**Backend (Java with Spring Boot):** Monitor request handling time and resource utilization to identify and address bottlenecks.

**Database (MySQL):** Analyse response time, throughput, and concurrency to optimize database performance.

**Testing and Deployment:** Optimize build and test execution times for efficient testing and deployment processes.

**Infrastructure (AWS):** Validate auto-scaling and load balancing configurations to optimize resource utilization and improve performance.

**6. Security Consideration:**

**Encryption:** Utilizes encryption algorithms to secure sensitive data such as user passwords and payment information.

**Authorization:** Enforces role-based access control to restrict user access to authorized functionalities.

**Input Validation:** Validates user input to prevent injection attacks and other security vulnerabilities.

**HTTPS:** Ensures secure communication between the client and server using HTTPS protocol.

**7.Deployment Architecture**

**Environment Setup:** Prepare development, testing, and production environments mirroring each other to ensure consistency and minimize issues.

**Version Control and CI/CD:** Use Git for version control and Jenkins for CI/CD to automate building, testing, and deploying code changes.

**Cloud Deployment:** Hosts the application on cloud platforms such as AWS Cloud for scalability and reliability.

**Containerization:** Uses containerization technologies like Docker for packaging and deploying the application components.

**Load Balancing:** Implements load balancing mechanisms to distribute incoming traffic across multiple servers and improve scalability and fault tolerance.

**Post-Deployment Testing and Validation:** Conduct tests and monitor performance post-deployment to ensure stability and meet service objectives.

**10. Conclusion**

The Insurance Aggregator Web Application leverages modern technologies to streamline the insurance-buying process for users. Through a user-friendly interface powered by Angular on the frontend, users can easily compare, purchase, and manage insurance policies from multiple providers. The backend, developed in Java, employs robust components such as Docker for containerization, Jenkins for continuous integration, and Selenium for automated testing. This technology stack ensures the reliability, scalability, and security of the application, providing users with a seamless and efficient platform for their insurance needs.