Customer Support System Development Report

1. Introduction

This report details the systematic development process of a real-time customer support system, designed to efficiently manage support requests, assign agents, and facilitate message and file exchanges between customers and support staff.

2. Initial Project Setup

The initial project structure was established using Spring Initializr (https://start.spring.io/), chosen for its rapid, reliable, and optimized approach to initializing Spring Boot applications with effective dependency management. The selected technical stack included Java 21, Maven, Spring Boot 3.4.3, and essential dependencies such as Spring Web, JPA, and the H2 database.

3. Development Environment

- **Programming Language:** Java 21 (LTS)
- **Framework:** Spring Boot 3.4.3
- **Database:** H2 (In-Memory)
- **Dependency Management & Build Tool:** Apache Maven 3.9.x
- **Integrated Development Environment (IDE):** IntelliJ IDEA
- **Technologies Employed:**
    - Spring Data JPA (Hibernate)
    - WebSockets
    - Spring Actuator
    - RESTful APIs via Spring Boot
    - Postman for API Testing

4. Database Selection and Implementation

The H2 database was strategically chosen due to its ease of installation, embedded in-memory capability, and seamless integration with Spring Boot. These attributes significantly accelerated the development and initial testing phases, eliminating the need for extensive configuration.

5. System Architecture and MVC Application

The application is developed following the Model-View-Controller (MVC) architectural pattern, enhancing modularity, maintainability, and scalability:

Models (Entities):

- **Customer:** Represents system users or clients.
- **SupportAgent:** Depicts available support representatives.
- **SupportRequest:** Stores individual support queries.
- **Message:** Facilitates communications between customers and agents.
- **FileAttachment:** Manages associated file uploads.

Repositories:

- Provide abstracted data persistence interactions using Spring Data JPA.

Services:

- **SupportService:** Encapsulates the business logic for managing support requests, embodying the service layer of the MVC structure.

Controllers:

- **SupportController:** Manages HTTP requests, defines RESTful API endpoints, and orchestrates interactions between models and services, aligning with the controller layer of MVC.

Real-time Communication Components:

- **WebSocketConfig:** Manages real-time WebSocket configurations.
- **SupportChatHandler:** Handles incoming WebSocket messages, ensuring real-time customer-agent interactions.

6. Development Process

Development was methodically tracked using Clockify, totaling 6.5 hours. Techniques utilized include:

- AI-driven tools for optimization and ideation.
- Comprehensive Google searches for adhering to best practices.
- Reuse and refactoring of code from prior projects.

7. Testing and Validation

Robust system testing was performed using Postman, curl, and the H2 Console, ensuring comprehensive coverage of functionalities:

- Retrieval of active requests (`GET /support/requests`).

- Agent availability updates (`POST /support/add-agent`).
- Assignment of agents (`POST /support/assign`).
- Submission of support queries (`POST /support/submit`).
- Communication via messages (`POST /support/message`).
- Retrieval of specific message threads (`GET /support/messages/{requestId}`).
- File uploads (`POST /support/upload`).

Additionally, the H2 Console provided essential insights into data storage accuracy and database integrity.

8. Security Considerations and Database Accessibility

Given that Spring Security is enabled by default, specific configurations were applied to facilitate access to the H2 Console, such as disabling CSRF protection and frame options. These adjustments were instrumental in streamlining data management and monitoring processes during development.

9. Future Architectural Evolution and Microservices

Currently, the application employs a monolithic design. However, transitioning to a microservices architecture is proposed to significantly enhance scalability, maintainability, and resilience.

Benefits of Microservices Architecture:

- **Scalability:** Facilitates independent service-level scalability.
- **Enhanced Maintainability:** Simplifies debugging and accelerates feature deployment.
- **Development Autonomy:** Allows development teams to independently manage individual services.
- **Fault Isolation:** Reduces systemic risks, isolating issues within modules.

Proposed Microservices Structure:

Potential microservices include User Service, Ticketing Service, Messaging Service, Notification Service, File Service, API Gateway, and Service Registry. Each service would maintain a dedicated database, communicating via REST APIs or asynchronous message brokers like Kafka.

Furthermore, deployment within Docker and Kubernetes environments is recommended to fully exploit microservices architecture benefits.