Universidad de las Fuerzas Armadas - ESPE.

Departamento de Ciencias de la Computación.

Informe Final - Sistema de Mensajería Instantánea con Websockets en Java.

Aplicaciones Distribuidas.

Estudiante:
Shirley Stefania Otuna Rojano.

Docente:
Ing. Darío Javier Morales Caiza.

Agosto 2024.

Departamento de Ciencias de la Computación.
Aplicaciones Distribuidas.
Universidad de las Fuerzas Armadas - ESPE.

Ingeniería de Software.

Sistema de Mensajería Instantánea con Websockets en Java.

Otuna Rojano Shirley Stefania.

Resumen

Este informe detalla el desarrollo de una aplicación de chat en tiempo real utilizando Websockets y Spring Boot. Se abordan la descripción del problema, los objetivos del proyecto, la metodología utilizada, la arquitectura del sistema, los diagramas UML, los resultados obtenidos y las posibles mejoras futuras.

1. Descripción del Proyecto.

En el entorno actual de comunicación digital, la necesidad de interacciones instantáneas y en tiempo real entre los usuarios se ha vuelto fundamental. Las aplicaciones de mensajería instantánea, las redes sociales y las plataformas de colaboración en línea dependen en gran medida de la capacidad para intercambiar mensajes rápidamente y sin interrupciones. Sin embargo, las tecnologías tradicionales basadas en el protocolo HTTP presentan limitaciones significativas para este tipo de comunicación.

El protocolo HTTP está diseñado para solicitudes-respuestas simples, lo que implica que cada interacción requiere una nueva conexión al servidor. Esto resulta en una latencia considerable y una falta de bidireccionalidad, haciendo que las aplicaciones basadas en HTTP no sean adecuadas para la comunicación en tiempo real. Estas limitaciones se traducen en retrasos en la entrega de mensajes, una carga innecesaria en los servidores debido a la apertura y cierre constante de conexiones, y una experiencia de usuario subóptima.

Websockets surge como una solución ideal para estos problemas. Este protocolo permite una comunicación dúplex completa, lo que significa que tanto el cliente como el servidor pueden enviar datos en cualquier momento sin necesidad de volver a establecer la conexión. Esto no solo reduce la latencia, sino que también optimiza el uso de recursos del servidor, proporcionando una experiencia de usuario mucho más fluida y eficiente.

A pesar de las ventajas de Websockets, la implementación de esta tecnología presenta sus propios desafíos. Requiere una configuración adecuada en el servidor para manejar múltiples conexiones simultáneas, y la interfaz de usuario debe ser capaz de gestionar y mostrar mensajes en tiempo real de manera eficiente. Además, la seguridad de las comunicaciones debe ser garantizada para proteger la privacidad de los usuarios.

2. Objetivo del Proyecto.

Desarrollar una aplicación de chat en tiempo real que permita la comunicación instantánea entre múltiples usuarios utilizando Websockets con Spring Boot para el backend y Bootstrap para la interfaz de usuario. La aplicación debe proporcionar una experiencia de usuario fluida, segura y eficiente, aprovechando las capacidades de comunicación bidireccional de Websockets para minimizar la latencia y optimizar el rendimiento del sistema.

3. Metodología Utilizada para el Desarrollo.

Fases del Proyecto.

1. **Análisis de Requisitos:** Identificación de las necesidades de comunicación en tiempo real y selección de tecnologías adecuadas (Websockets y Spring Boot). Recolección de requerimientos funcionales y no funcionales del sistema de chat.
2. **Diseño del Sistema:** Creación de diagramas de arquitectura y diseño de la estructura del frontend utilizando Bootstrap. Definición de los componentes del backend y su interacción.
3. **Desarrollo del Backend:** Configuración de Websockets en Spring Boot y desarrollo de APIs para la gestión de mensajes. Implementación de controladores y servicios necesarios para manejar las conexiones y la lógica de negocio.
4. **Desarrollo del Frontend:** Implementación de la interfaz de usuario utilizando Bootstrap para garantizar una presentación responsive y amigable. Integración del frontend con las APIs del backend.
5. **Pruebas:** Realización de pruebas unitarias y de integración para garantizar la funcionalidad y estabilidad del sistema. Pruebas de carga para asegurar que el sistema puede manejar múltiples conexiones simultáneas.
6. **Despliegue:** Implementación de la aplicación en un entorno de producción. Configuración del servidor para manejar conexiones Websocket y asegurar un rendimiento óptimo.

Herramientas Utilizadas.

- **Backend:** Spring Boot, Maven, Lombok.
- **Frontend:** HTML, CSS, JavaScript, Bootstrap.
- **IDE:** IntelliJ IDEA.
- **Control de Versiones:** 1.0.

4. Descripción del Diseño y Arquitectura del Sistema.

Arquitectura del Sistema.

El sistema se basa en una arquitectura cliente-servidor donde el servidor maneja la lógica de negocio y las conexiones de Websockets, y el cliente proporciona la interfaz de usuario.

Componentes Principales.

- **Servidor Web:** Implementado con Spring Boot, maneja la lógica de Websockets y la gestión de usuarios.
- **Cliente Web:** Implementado con HTML, CSS, Bootstrap y JavaScript, proporciona la interfaz de usuario para la aplicación de chat.

Configuración del Websocket en Spring Boot.

- **WebSocketConfig:** Clase de configuración para habilitar Websockets y definir puntos finales.

- **WebSocketEventListener:** Clase para escuchar eventos de conexión y desconexión de Websockets.
- **ChatController:** Controlador para gestionar las solicitudes de mensajes y la gestión de usuarios.

Interacción Cliente-Servidor.

- **Cliente:** El cliente inicia una conexión Websocket con el servidor y puede enviar y recibir mensajes en tiempo real.
- **Servidor:** El servidor maneja las conexiones de Websockets, procesa los mensajes entrantes y distribuye los mensajes a todos los clientes conectados.

5. Diagramas.

Diagrama de Clases.

El diagrama de clases muestra las principales clases y sus relaciones en el sistema de chat.

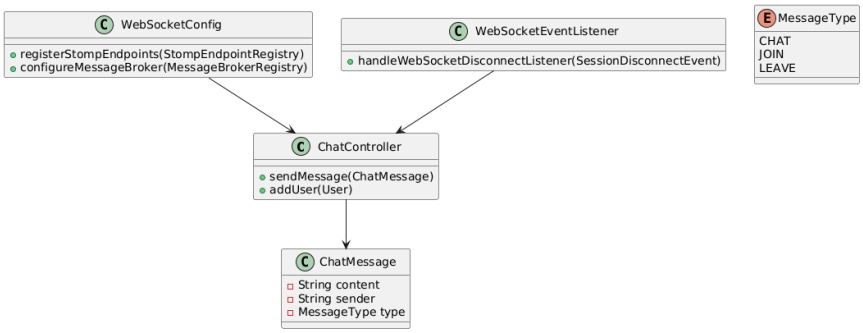

Figura 1: Diagrama de Clases.

- **ChatController:** Maneja las solicitudes de mensajes y la adición de usuarios.
- **WebSocketConfig:** Configura los endpoints de Websockets y el broker de mensajes.
- **WebSocketEventListener:** Escucha eventos de desconexión de Websockets.
- **ChatMessage:** Representa un mensaje de chat con contenido, remitente y tipo de mensaje.
- **MessageType:** Enumera los tipos de mensajes (CHAT, JOIN, LEAVE).

Diagrama de Secuencia.

El diagrama de secuencia detalla la secuencia de interacción entre el cliente y el servidor al enviar un mensaje de chat.

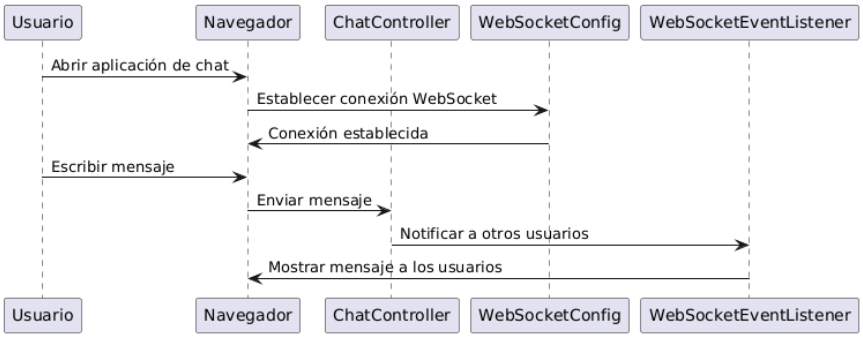

Figura 2: Diagrama de Secuencia.

- **Usuario:** Interactúa con la aplicación de chat.
- **Navegador:** Cliente que maneja la interfaz de usuario y la conexión Websocket.
- **ChatController:** Procesa los mensajes enviados por el usuario.
- **WebSocketConfig:** Configura y establece la conexión Websocket.
- **WebSocketEventListener:** Notifica a otros usuarios cuando se envía un mensaje.

Diagrama de Arquitectura del Sistema.

El diagrama muestra la arquitectura general del sistema, destacando los componentes principales y cómo interactúan entre sí.

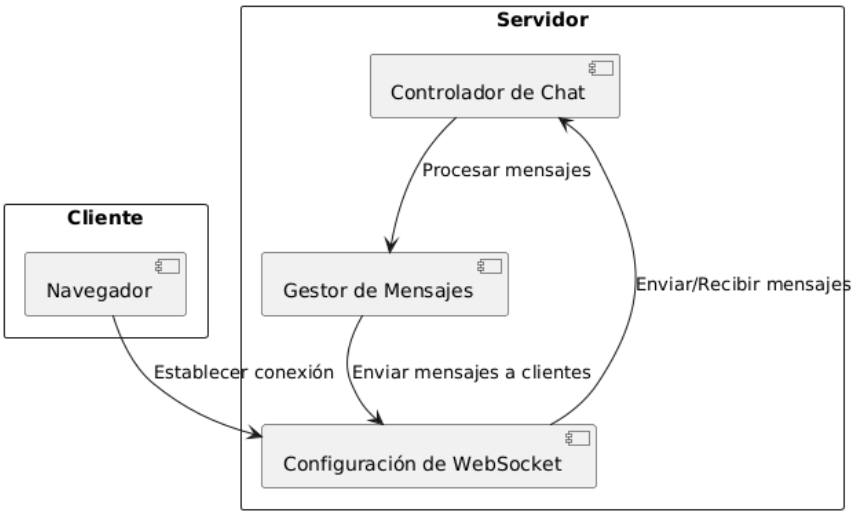

Figura 3: Diagrama de Arquitectura del Sistema.

- **Cliente (Navegador):** La interfaz de usuario que se comunica con el servidor.
- **Servidor:** Maneja la lógica de negocio y las conexiones Websocket.
  - **Controlador de Chat:** Gestiona la lógica de los mensajes.
  - **Configuración de WebSocket:** Configura la conexión Websocket.
  - **Gestor de Mensajes:** Procesa y distribuye los mensajes.

Diagrama de Componentes.

El diagrama detalla los componentes internos del servidor y cómo se relacionan entre ellos.

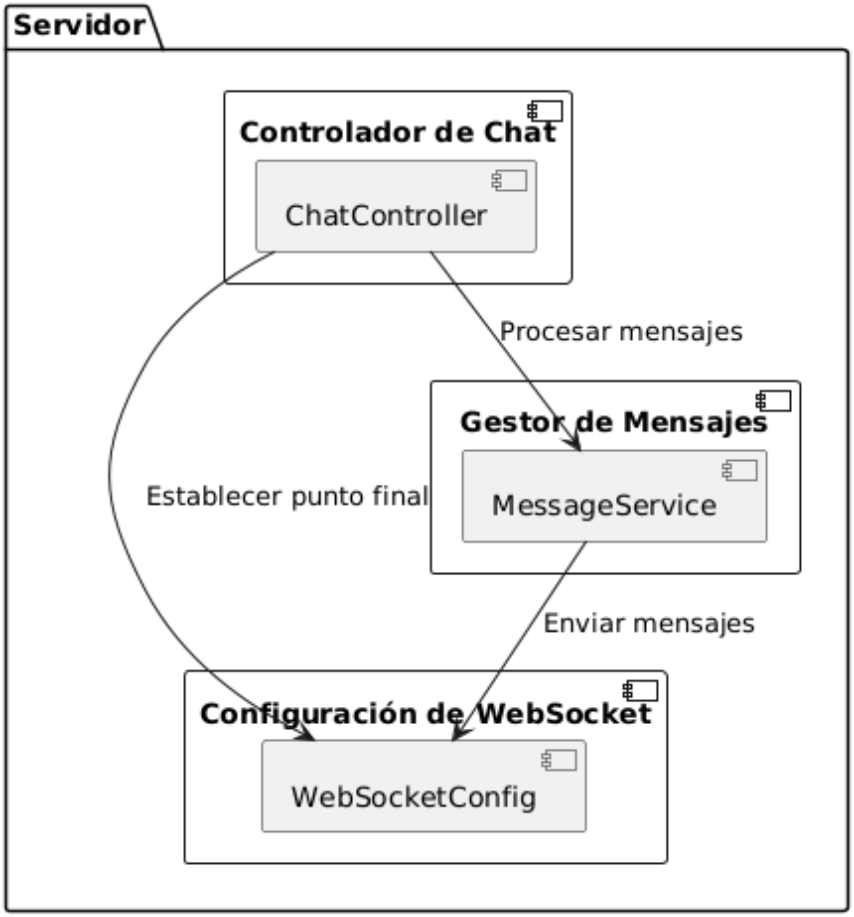

Figura 4: Diagrama de Componentes.

- **Controlador de Chat:** Gestiona la lógica de los mensajes.
- **Configuración de WebSocket:** Configura la conexión Websocket.
- **Gestor de Mensajes:** Procesa y distribuye los mensajes.

Diagrama de Actividades.

El diagrama muestra el flujo de actividades desde que un usuario se conecta hasta que envía un mensaje.

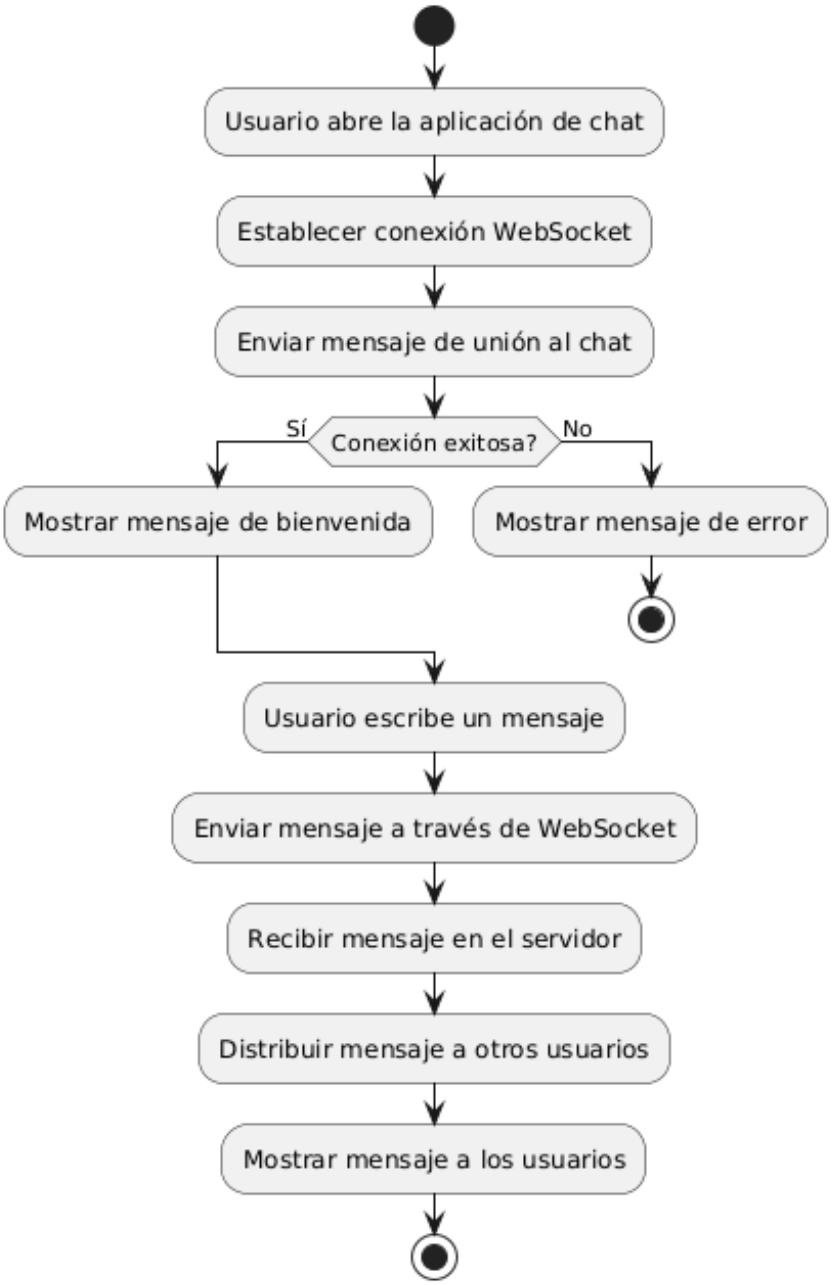

Figura 5: Diagrama de Actividades.

- **Establecer conexión WebSocket:** El navegador establece una conexión WebSocket con el servidor.
- **Enviar mensaje de unión:** El usuario envía un mensaje de unión al chat.
- **Enviar mensaje:** El usuario escribe y envía un mensaje a través del WebSocket.
- **Distribuir mensaje:** El servidor distribuye el mensaje a otros usuarios conectados.

Diagrama de Despliegue.

El diagrama muestra la configuración física del sistema, incluyendo los servidores y cómo están conectados.

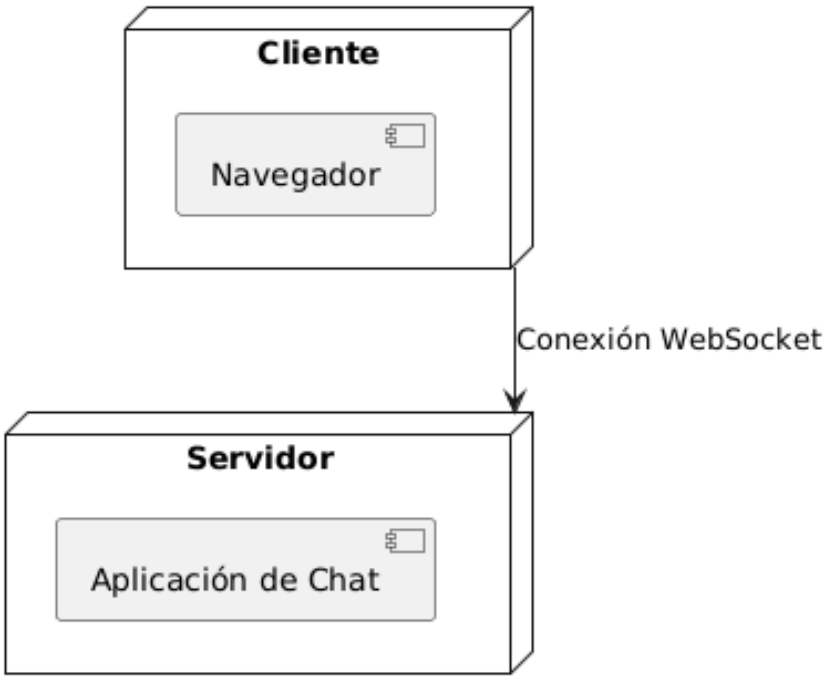

Figura 6: Diagrama de Despliegue.

- **Servidor:** Hospeda la aplicación de chat implementada con Spring Boot.
- **Cliente (Navegador):** Los usuarios acceden a la aplicación de chat a través de un navegador web.

Diagrama de Casos de Uso.

El diagrama de casos de uso muestra las interacciones entre los actores (usuarios y administradores) y los casos de uso (funcionalidades) del sistema de chat en tiempo real.

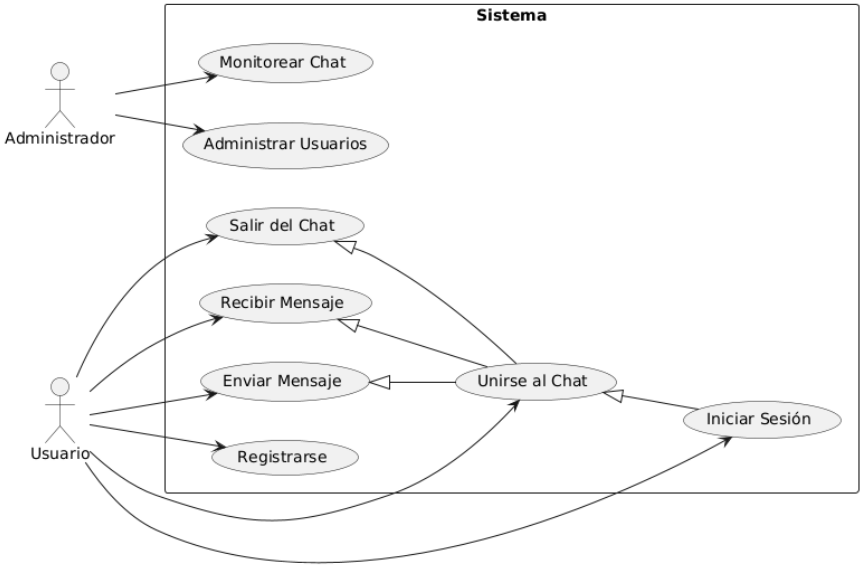

Figura 7: Diagrama de Casos de Uso.

6. Posibles Mejoras Futuras.

Mejoras Técnicas.

- Implementar técnicas de escalabilidad para manejar un mayor número de usuarios.
- Añadir autenticación y cifrado de mensajes para asegurar la comunicación.
- Implementar un sistema de almacenamiento de mensajes persistente para permitir la recuperación de conversaciones pasadas (opcional).

Mejoras Funcionales.

- Permitir el envío de imágenes y archivos.
- Integrar notificaciones push para alertar a los usuarios de nuevos mensajes cuando no están activos en la aplicación.
- Mejorar la interfaz de usuario para una mejor experiencia y usabilidad.

7. Resultados Obtenidos.

- La aplicación permite la comunicación en tiempo real entre múltiples usuarios sin latencia perceptible.
- La interfaz de usuario es intuitiva y responsive gracias a Bootstrap.
- El sistema maneja de manera eficiente múltiples conexiones simultáneas sin pérdida de mensajes.
- Las pruebas de carga demostraron que el sistema puede soportar un gran número de usuarios conectados simultáneamente.

8. Conclusiones.

El proyecto demostró que es posible implementar una aplicación de chat en tiempo real utilizando Websockets y Spring Boot sin necesidad de una base de datos persistente. La combinación de estas tecnologías permite una comunicación eficiente y de baja latencia, ideal para aplicaciones que requieren interacción instantánea entre usuarios.