



PRD-VER-COD

Documento de Arquitectura

Chat Web Grupal - Chatly

Versión 1.0.0

Área de Desarrollo
Dirección de Tecnología

CONFIDENCIAL


| | | |
|---|---|---------------|
|  | CHATLY Chat Web Grupal | |
| | Documento de Arquitectura | Versión: 1.0 |
| | | Página 1 de 9 |

Tabla de Contenido

| | |
|---|----------|
| 1. Introducción | 2 |
| 1.1. Objetivo | 2 |
| 1.2. Justificación | 2 |
| 2. Antecedentes | 2 |
| 3. Alcance | 3 |
| 4. Componentes del sistema | 3 |
| 5. Patrón de la Arquitectura | 4 |
| 6. Diseño de la arquitectura | 6 |
| 6.1 Diagramas de la Arquitectura | 6 |
| 7. Conclusión y Recomendación | 6 |
| 8. Firma de Participantes | 7 |



Documento de Arquitectura

Chat Web Grupal

1. Introducción

Este documento describe la arquitectura utilizada para un sistema de comunicación en tiempo real entre el cliente y el servidor, empleando tecnologías modernas como WebSocket, Spring Boot, y Angular. La solución está diseñada para manejar la autenticación y autorización mediante JWT, garantizar la comunicación bidireccional eficiente, y realizar operaciones seguras en una base de datos PostgreSQL.

1.1. Objetivo

El propósito de este documento es detallar la arquitectura del sistema presentado en el diagrama. Este sistema se enfoca en proporcionar un entorno seguro, escalable y de alta disponibilidad para aplicaciones que requieren comunicación en tiempo real, manejo de transacciones en la base de datos y autenticación robusta basada en estándares modernos.

1.2. Justificación

La arquitectura propuesta ha sido diseñada para cumplir con los siguientes objetivos clave:

- **Escalabilidad:** Permite un crecimiento eficiente, garantizando el manejo de un alto volumen de conexiones en tiempo real.
- **Seguridad:** Integra mecanismos avanzados de autenticación y autorización mediante JWT.
- **Interoperabilidad:** Facilita la integración con sistemas externos a través de servicios REST.
- **Desempeño:** Optimiza las consultas y transacciones con PostgreSQL, asegurando tiempos de respuesta bajos incluso bajo alta carga.

2. Antecedentes

El desarrollo de un sistema de chat en tiempo real responde a la necesidad de mejorar la comunicación entre usuarios en entornos digitales mediante el uso de tecnologías modernas como WebSockets, Angular y Spring Boot. Actualmente, las soluciones existentes carecen de una integración robusta que permita manejar usuarios concurrentes, garantizar la seguridad de datos y ofrecer notificaciones en tiempo real.



3. Alcance

El sistema permitirá una comunicación en tiempo real entre múltiples usuarios, la gestión de salas de chat grupales, autenticación segura mediante tokens JWT y la Visualización de usuarios conectados y notificaciones instantáneas. No incluye: Integración con redes sociales o herramientas de videoconferencia.

4. Componentes del sistema

4.1. Módulo de Comunicación WebSocket


- Función: Gestionar la comunicación en tiempo real entre el cliente (Frontend) y el backend, permitiendo una interacción fluida.
- Responsabilidades:
 - Establecer conexiones WebSocket seguras.
 - Manejar eventos en tiempo real entre el cliente y el servidor.
 - Enviar y recibir mensajes bidireccionales.
- Interfaces:
 - WebSocket API para la comunicación cliente-servidor.
 - Conexión con el backend para interactuar con la lógica de negocio.
- Tecnologías:
 - Protocolo WebSocket.
 - Integración con Spring Boot para gestionar sesiones WebSocket.

4.2. Módulo Backend (Spring Boot)

- Función: Implementar la lógica de negocio del sistema, gestionar transacciones con la base de datos y manejar la autenticación de usuarios.
- Responsabilidades:
 - Procesar las solicitudes recibidas desde el frontend.
 - Ejecutar operaciones CRUD en la base de datos.
 - Coordinar la comunicación con el módulo de seguridad para validar tokens JWT.
- Interfaces:
 - API REST para consultas y operaciones transaccionales.
 - Integración con el módulo de comunicación WebSocket.
- Tecnologías:
 - Spring Boot.
 - Java 17.

4.3. Módulo de Seguridad (JWT y Encriptación)

- Función: Garantizar que las transacciones y comunicaciones sean seguras mediante la autenticación y autorización de usuarios.
- Responsabilidades:
 - Emitir y validar tokens JWT para las sesiones de los usuarios.
 - Proteger las rutas críticas del backend mediante filtros de seguridad.
 - Encriptar datos sensibles antes de almacenarlos en la base de datos.
- Interfaces:
 - Middleware de autenticación para las rutas del backend.

| | | |
|---|---|---------------|
|  | CHATLY Chat Web Grupal | |
| | Documento de Arquitectura | Versión: 1.0 |
| | | Página 4 de 9 |

- Gestión de sesiones seguras con los tokens JWT.
- Tecnologías:
 - JWT (JSON Web Tokens).
 - Spring Security para la implementación de políticas de autorización.

4.4. Módulo de Base de Datos (PostgreSQL)

- **Función:** Almacenar y gestionar los datos del sistema de manera estructurada y segura.
- **Responsabilidades:**
 - Manejar las transacciones de lectura y escritura de datos.
 - Garantizar la integridad y consistencia de los datos almacenados.
 - Proporcionar consultas eficientes para el backend.
- **Interfaces:**
 - Conexión con el backend para ejecutar transacciones.
- **Tecnologías:**
 - PostgreSQL como base de datos relacional.
 - Hibernate como ORM para interactuar con la base de datos.

5. Patrón de la Arquitectura

5.1. Modelo MVVM en el Frontend

El frontend sigue el patrón Model-View-ViewModel (MVVM), donde se establece una clara separación entre la lógica de negocio y la interfaz de usuario. Este patrón facilita la organización del código, el mantenimiento y la escalabilidad.

- **Componentes del MVVM:**
 - **Model:** Representa los datos y la lógica del negocio, que generalmente provienen del backend.
 - **View:** Es la interfaz gráfica que interactúa con el usuario.
 - **ViewModel:** Actúa como intermediario entre el modelo y la vista, manejando la lógica de presentación y la interacción con los datos.
- **Ventajas del MVVM:**
 - Permite un vínculo directo entre la vista y el modelo mediante el enlace de datos (*data binding*).
 - Facilita la reutilización de componentes.
 - Hace que las pruebas unitarias sean más efectivas al desacoplar la lógica de presentación de la interfaz.
- **Tecnología utilizada:** Angular es el framework implementado en el frontend, ya que soporta de manera nativa el modelo MVVM con su sistema de componentes, servicios y



data binding.

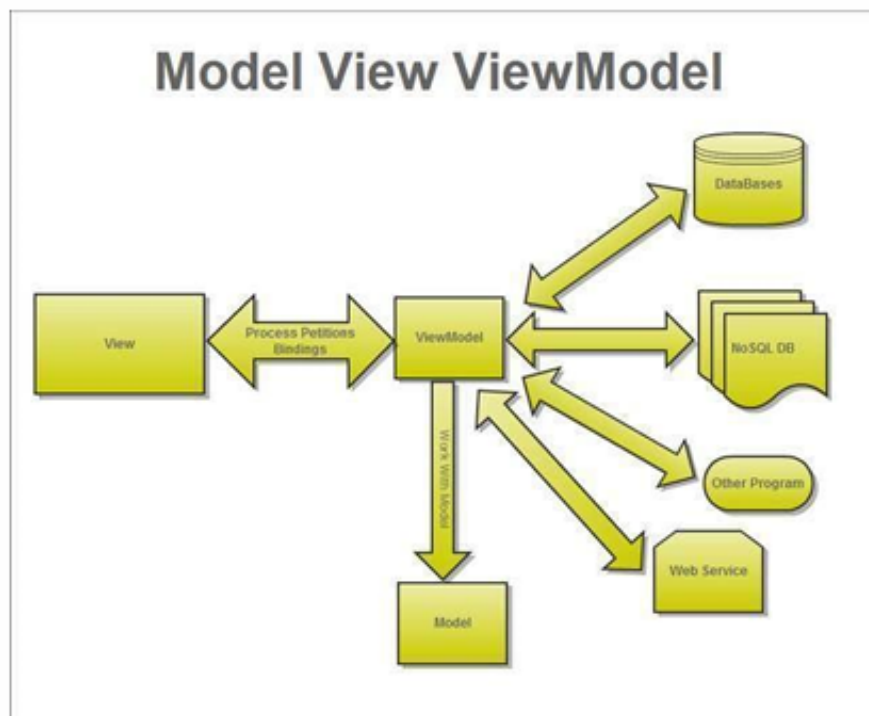


Figura 1: Modelo vista vista modelo (Ecured, 2021)

5.2. Arquitectura n-Capas en el Backend

El backend utiliza una arquitectura **n-capas**, implementada con **MVC (Model-View-Controller)** en combinación con un **servicio** para la lógica de negocio. Esta estructura modular permite un diseño escalable, mantenible y eficiente.

- **Capas Principales:**
 - **Capa de Presentación (Controller):**
 - Gestiona las solicitudes HTTP provenientes del frontend o servicios externos.
 - Invoca a los servicios necesarios para procesar la solicitud.
 - Devuelve las respuestas en formato JSON, listas para ser consumidas por el cliente.
 - **Capa de Servicios (Business Logic Layer):**
 - Contiene la lógica de negocio del sistema.
 - Procesa las reglas específicas y transforma los datos entre las capas de presentación y persistencia.
 - Ofrece métodos reutilizables que encapsulan las operaciones de negocio.
 - **Capa de Persistencia (Model y DAO):**
 - Gestiona la interacción con la base de datos mediante el uso de Hibernate o JPA.
 - Define las entidades y operaciones CRUD que permiten la persistencia de los datos.
- **Modelo MVC con Servicio:**
 - **Model:** Representa la estructura de los datos y los objetos persistentes.

- **View:** En este caso, no es una vista gráfica, sino la respuesta que se envía al cliente (generalmente en formato JSON).
- **Controller:** Maneja las solicitudes HTTP, invoca los servicios y envía las respuestas al cliente.

6. Diseño de la arquitectura

6.1. Diagrama de la Arquitectura

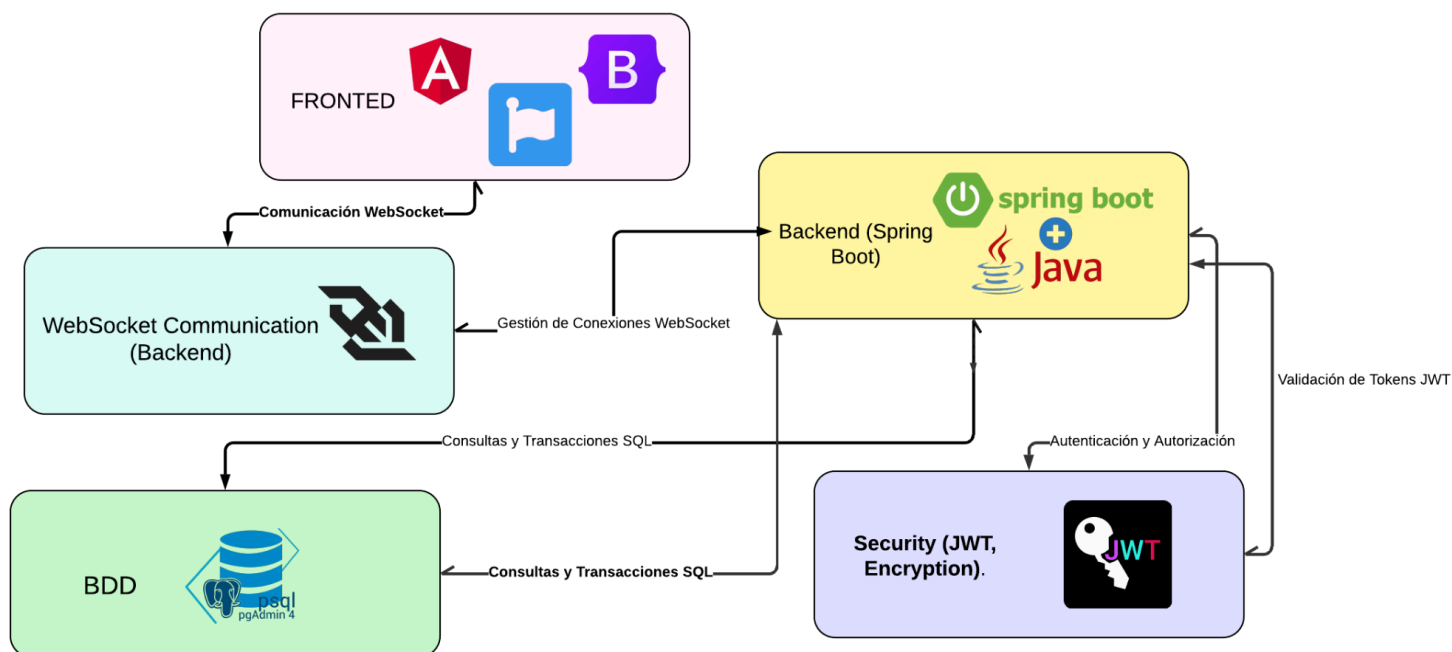



Figura 2: Diagrama de Arquitectura (Autoría Propia, 2024)

7. Conclusión y Recomendación

Conclusiones

- El uso de Angular y MVVM en el frontend asegura una interfaz interactiva y fácil de mantener.
- Spring Boot y la arquitectura n-capas en el backend permiten la gestión eficiente de la lógica de negocio, la seguridad y las operaciones con la base de datos.
- La integración de WebSocket garantiza la comunicación en tiempo real, esencial para un sistema de chat moderno.
- PostgreSQL, con su enfoque en la consistencia y el desempeño, proporciona una base sólida para manejar datos estructurados.

| | | |
|---|---|---------------|
|  | CHATLY Chat Web Grupal | |
| | Documento de Arquitectura | Versión: 1.0 |
| | | Página 7 de 9 |

Recomendaciones

- Implementar un sistema de balanceo de carga y escalar horizontalmente el backend para manejar mayores volúmenes de tráfico.
- Realizar pruebas exhaustivas de seguridad, como pruebas de penetración, para garantizar la robustez del sistema.
- Monitorear y optimizar continuamente las consultas a la base de datos y el desempeño del servidor WebSocket.



8. Firma de Participantes

APROBACIÓN DEL CRONOGRAMA

| Elaborado y Aprobado | | |
|---------------------------------|--------------------------------|-------|
| Unidad de Negocio | Responsable | Firma |
| Equipo de Desarrollo | Grupo 11 | |
| Docente | Ing. Ángel Marcelo Rea Guamán. | |
| FECHA DE APROBACIÓN: 21/11/2024 | | |