PCS: Diseño Chat con JAutomata Y Modificación Algoritmo Berkeley (ZeroMQ)

Programación en sistemas Cloud

Alumno

José Javier Gutiérrez Gil

jogugi@posgrado.upv.es| jogugil@gmail.com

27 novembre 2024

Profeso/i

Bataller Mascarell, Jordi (batallerdsic.upv.es)

Índice

CAPÍTULO 1

Sistema Chat

1.1 introducción

Objetivos

El objetivo de este proyecto es diseñar e implementar un sistema de chat funcional y sencillo, permitiendo a los usuarios interactuar en tiempo real con un enfoque en simplicidad y comodidad tanto en el cliente como en el servidor. El presente documento detalla las especificaciones funcionales para el desarrollo de este sistema, el cual permitirá a los usuarios comunicarse mediante mensajes de texto en tiempo real, asegurando una experiencia de usuario óptima y accesible.

Alcance

El sistema de chat permitirá a los usuarios iniciar sesión, enviar y recibir mensajes en tiempo real. En esta primera versión solo existirá una única sala, un único tipo de usuario y no existirá comprobaciones de sesión. La pantalla principal sólo mostrará el historial de mensajes y la posibilidad de desconectarse de la sesión de usuario.

1.2 Descripción de las especificaciones del sistema

Para entrar al sistema chat, los usuarios introducen un 'nickname' que es el identificativo dentro del sistema chat. Por tanto, este 'nickname' será único en todo el sistema.

La interfaz del cliente será lo más sencilla posible y constará de dos zonas principales. La primera zona incluirá un formulario para que el usuario pueda ingresar al chat. La segunda zona mostrará el historial de mensajes y un campo de entrada donde el usuario podrá escribir los mensajes que desee enviar al chat. Además, habrá un botón para desconectarse.

Por tanto, el sistema deberá gestionar notificaciones en tiempo real para avisar sobre nuevos mensajes y cambios de estado, como la conexión y desconexión de usuarios. En caso de pérdida de conexión, el cliente intentará reconectarse automáticamente un número determinado de veces, superado el número de vece emitirá un error de conexión. Para facilitar la interacción, la implementación debe permitir utilizar herramientas como la línea de comandos (CLI) o interfaces gráficas, como el cliente chat que se entregará con la implementación del sistema.

El cliente interactuará con el servidor a través de solicitudes al servidor mediante un mecanismo de comunicación, bien sea un API REST o a través de sockets o broadcast.

La comunicación entre cliente y servidor se gestionará eficientemente a través de APIs, garantizando la distribución de mensajes y la actualización de la lista de usuarios activos. Los mensajes emitidos al servidor y recibidos mantendrán un formato válido como json o xml, serializando y/o descerealizando al emitir o recibir los mensajes.

El servidor del sistema de chat se diseñará de manera modular. Para optimizar el rendimiento y evitar la acumulación descontrolada de mensajes en la base de datos, se implementará una capa intermedia de buffer, que funcionará como un "área de almacenamiento temporal". Este buffer almacenará los mensajes más recientes y gestionará su tamaño de acuerdo con parámetros configurables, lo que permitirá definir el número máximo de mensajes o el límite de bytes que se mantendrán en memoria antes de ser enviados a la base de datos para su persistencia. Así, se evitará el crecimiento desmesurado de la base de datos y se garantizará un acceso más rápido a los mensajes recientes. El buffer interactuará directamente con el módulo de comunicación, que será el encargado de gestionar el polling y de distribuir los mensajes

desde el buffer hacia los clientes, proporcionando una interacción eficiente y en tiempo real. Además, el servidor gestionará la lista de usuarios activos, registrando sus conexiones y desconexiones, y notificando a los clientes sobre estos cambios.

En esta implementación sólo se tendrá una sala de chat.

Módulos del sistema:

Frontend de Usuarios al Chat:

- Interfaz gráfica de usuario donde los usuarios pueden interactuar con el sistema de chat.
- Permite a los usuarios iniciar sesión, unirse a la sala de chat y enviar mensajes ¡.
- Los usuarios pueden ver las listas de usuarios activos.

Servidor:

- Controla la lógica del mecanismo de interacción de un usuario con el sistema y el intercambio de mensajes a la sala del chat.
- **Gestión de usuarios registrados/libres:** El servidor gestiona la entrada y salida de usuarios del sistema. Los usuarios pueden iniciar sesión, y el servidor valida si el nickname introducido es único en todo el sistema.
- **Control de sala:** El servidor gestiona la creación y eliminación de la sala donde los usuarios envían mensajes. LLeva el control de la entrada y salida de usuarios en la sala y de los mensajes emitidos a la misma.
- **Modularidad del servidor:** El diseño del servidor se basa en módulos funcionales, que permiten una estructura flexible y escalable:

  - **Módulo de comunicación e interacción con clientes:** Responsable de gestionar la comunicación entre el servidor y los clientes. En un principio tenemos varios mecanismos de comunicación que no podemos descartar y que referencian a patrones como react, polling y/o broadcast.
  - **Módulo de lógica del chat:** Encargado de la administración de usuarios activos en el sistema.
  - **Módulo de gestión de mensajes y persistencia:** Actúa como intermediario entre el módulo de comunicación y la base de datos. Se ocupa de almacenar mensajes, registros de usuarios y datos de forma segura y eficiente.

Mensajes:

- Los mensajes pueden ser:

  - **Mensajes generales:** Son enviados a una sala y visibles para todos los miembros de esa sala.

- Todos los mensajes están asociados a una sala específica, lo que asegura un manejo organizado de las comunicaciones y facilita su administración.
- El servidor utiliza una **capa intermedia** entre el módulo de comunicación y la base de datos para gestionar los mensajes almacenados, previniendo un aumento desmesurado de mensajes en la base de datos.

- Se implementará un mecanismo cíclico o programado para limpiar mensajes antiguos en la base de datos del buffer y/o base de datos. De esta manera conservaremos solo los más recientes o relevantes según las políticas del sistema.
- Los mensajes se procesan en tiempo real, asegurando una entrega rápida y sin retrasos hacia su destino correspondiente.

Salas:

- En esta versión existirá una única sala donde los usuarios intercambian mensajes a través de ella.

Módulo de Base de Datos

El módulo de base de datos es responsable de almacenar de manera estructurada y segura la información del sistema de chat. Incluye el manejo de datos de usuarios, mensajes, salas y registros de auditoría. Diseñado para adaptarse tanto a sistemas relacionales (MySQL,Postgres,...) como a sistemas NoSQL (MongoDB, Cassandra,...), este módulo permite flexibilidad en la elección según los requisitos de rendimiento y escalabilidad.

Gestión de Usuarios

- Mantener el estado de los usuarios (activo, desconectado, último acceso, etc.).
- Historial de actividad en la sala para auditorías y reportes.

Gestión de Mensajes

- Almacenar mensajes, con metadatos como:
  - Usuario emisor.
  - Sala asociada.
  - Marca de tiempo.
- Implementar políticas de limpieza de mensajes antiguos para evitar sobrecarga.
- Indexación de mensajes para búsquedas rápidas.

Auditoría y Monitoreo

- Almacenar logs del sistema para análisis de métricas y control de errores.

1.3 Requisitos del Sistem Chat

1.3.1 Requisitos del módulo cliente del sistema chat

Requisitos Funcionales de la Interfaz de Usuario (IU)

- **CL-IU-001: Página de Inicio**
  El sistema debe tener una página de inicio con un campo para ingresar el nombre de usuario (nickname).
- **CL-IU-002: Página de Chat**
  Debe incluir una zona de solo lectura para el historial de mensajes, una zona para escribir mensajes, listado de usuarios en la sala.

- **CLI-IU-03:** El usuario debe ingresar un nombre de usuario único para unirse al chat.
  El sistema validará si el nombre de usuario está en uso y notificará al usuario en caso de conflicto.
- **CLI-IU-04:** Los usuarios podrán enviar mensajes al servidor, que los distribuirá a los demás participantes de la sala por un mecanismo de comunicación. El historial de mensajes de la sala activa se actualizará de forma permanente, mostrando los mensajes ordenados en fecha y forma.
- **CLI-IU-05:**Cada mensaje enviado debe incluir el nombre de usuario (nick) y la hora de envío, para que los demás usuarios puedan identificar al emisor y cuándo se envió el mensaje. Internamente elvarña también el identificador de la sala que en esta implementación será única.
- **CLI-IU-06:** El sistema debe recibir mensajes recientes del servidor, mostrarse de manera ordenada y cronológica, respetando el orden en que fueron enviados.
- **CLI-IU-07:** El cliente puede solicitar y mostrar la lista de usuarios activos en la sala activa. La lista de usuarios activos debe incluir el nombre de usuario y su estado (por ejemplo, activo, inactivo).
- **CLI-IU-08:** Los usuarios inactivos se mostrarán y permanecerán un tiempo en el sistema hasta que son eliminados por un proceso en el servidor, momento en el cual se emite una comunicación al cliente para actuañizar dicha lista.

Requisitos de los usuarios

- **CLI-GU-01:** El sistema debe permitir el inicio de sesión de los usuarios utilizando su nombre de usuario.
- **CLI-GU-02:** El sistema debe validar la unicidad del nombre de usuario durante el registro y mostrar un mensaje de error si ya está en uso.

Requisitos de los mensajes

- **CLI-IUM-01:** El sistema debe permitir el envío y la recepción de mensajes en tiempo real entre los usuarios mediante un sistema de notificación.
- **CLI-IUM-02:** El cliente debe actualizar la interfaz de mensajes en tiempo real sin necesidad de que el usuario realice ninguna acción adicional.
- **CLI-IUM-03:** El sistema debe mostrar al usuario todos los mensajes actualizados cuando reciba un nuevo mensaje en una sala activa.
- **CLI-IUM-04:** Las actualizaciones de los mensajes deben ser claras y visibles para el usuario, sin interrumpir la conversación actual.
- **CLI-IUM-05:** El historial de mensajes debe ser accesible y debe mostrarse de manera ordenada, permitiendo a los usuarios navegar fácilmente por los mensajes anteriores.

Requisitos de las comunicaciones

- **CLI-SS-01:** Los operadores del sistema deben poder supervisar el estado del servidor en tiempo real, visualizando estadísticas clave sobre el rendimiento y uso del chat (número de usuarios conectados, actividad del servidor, etc.).
- **CLI-SS-02:** El cliente debe permitir a los operadores visualizar logs de actividad para monitorear eventos y posibles incidencias.
- **CLI-GE-01:** El sistema debe permitir a los operadores acceder a logs de errores y eventos del sistema para ayudar en la resolución de problemas técnicos.
- **CLI-GE-02:** El cliente debe proporcionar herramientas para detectar y manejar errores automáticamente (como la reconexión en caso de fallo de conexión).
- **CLI-RP-01:** Los administradores deben poder generar reportes de actividad del chat, como el

número de usuarios activos, mensajes enviados y eliminados, y cualquier otra acción relevante.

- **CLI-RP-02:** Los reportes generados deben ser claros y exportables para su análisis posterior.
- **CLI-CG-01:** Los administradores deben poder activar o desactivar funcionalidades del chat, como la opción de compartir archivos o habilitar ciertos comandos.
- **CLI-CG-02:** Los administradores deben poder modificar las reglas del sistema, como las configuraciones de privacidad, la duración de las sesiones o los límites de tamaño de los mensajes.

1.3.2 Requisitos del módulo servidor del sistema chat

Lógica de Usuario

- **SV-LU-01:** El servidor debe registrar y mantener usuarios temporales al sistema. Son aquellos que entran al chat con un nombre de usuario ('nickname').
- **SV-LU-02:** El servidor debe validar la unicidad de los nombres de usuario (nickname) al entrar al sistema chat.
- **SV-LU-03:** El servidor debe gestionar los estados de los usuarios (activo, desconectado).
- **SV-LU-04:** El servidor debe controlar cuando el usuario entra o sale de una sala. En esta implementación sólo existe una única sala.
- **SV-LU-05:** El servidor debe controlar la desconexión de usuarios del sistema chat. Si loa usuarios pierden la conexión, se mantiene un tiempo antes de ser eliminado.

Lógica de Mensajes

- **SV-LM-01:** El servidor debe recibir mensajes de los clientes a través de una API. Cada mensaje estará asociado con la sala de origen, el usuario que lo envía, la hora y fecha de envío.
- **SV-LM-02:** El servidor debe almacenar los mensajes en la base de datos si es necesario. Cada mensaje almacenado debe incluir la información de la sala, el usuario que lo envió, la fecha y hora.
- **SV-LM-03:** El servidor debe mantener una cola de mensajes circular por cada sala existente, con un tamaño máximo definido para evitar el desbordamiento de memoria. Esta cola actúa como buffer de almacenamiento de mensajes para la sala.
- **SV-LM-04:** El servidor debe almacenar los mensajes en la base de datos en la sala correspondiente, liberando espacio en la cola cuando se almacenan correctamente para evitar el desbordamiento de memoria. Si la cola llega a su límite, antes de sobrescribir, los mensajes más viejos son almacenados en la base de datos.
- **SV-LM-05:** El servidor debe utilizar un mecanismo de comunicación para que el cliente pueda actualizar el histórico de mensajes en la sala activa y la lista de usuarios activos en ella.
- **SV-LM-06:** Cada mensaje debe estar asociado con los siguientes atributos:
  - **Sala:** La sala de origen del mensaje.
  - **Usuario:** El usuario que envió el mensaje.
  - **Fecha y hora:** El momento en que se envió el mensaje.

Módulo de Seguridad y Control de Acciones

- **SV-SG-01 / control de logs**: Cada operación base se debe registrar en un log para la creación de métricas y auditorias base.

- **SV-SG-02 /control del mecanismo de comunicación**: El sistema debe monitorizar y controlar el buen funcionamiento del mecanismo de comunicación con el cliente. Además debe tener un control mínimo de los mensajes emitidos por el cliente al servidor para evitar problemas. Por ejemplo, el formato valido de los mensajes.

Requisitos de Lógica y Gestión de Salas

- **SV-GS-01:** El sistema debe controlar el acceso a las salas, permitiendo que los usuarios puedan unirse o abandonar las salas según deseen.

Requisitos del Sistema: Arranque

- **SV-AC-01:** Al iniciar el módulo servidor, se debe garantizar que el módulo de comunicación esté activo y operativo, permitiendo que los clientes puedan establecer comunicación sin retrasos o interrupciones desde el momento de su conexión.

Requisitos del Sistema: Comunicación y Mensajes

Módulo Cliente

- **SYS-COM-CL-01:** el cliente debe permitir enviar mensajes nuevos al servidor mediante un api o mecanismo de comunicación, Por ejemplo podríamos usar un 'endpoit' abierto en el servidor y emitir mensajes nuevos mediante POST /newmesssage.
- **SYS-COM-CL-02:** El cliente debe permitir la actualización del historial de mensajes emitidos a la sala del chat por parte de los demás usuarios. Para ello podríamos usar mecanismos de polling, donde para obtener los mensajes nuevos emitimos solicitudes GET en el endpoint $/messages, lastMessageId =< id >$.
- **SYS-COM-CL-03:** El cliente debe permitir la actualización de a kista de usaurios activos. Para ello podríamos usar mecanismos de polling, donde para obtener los usuarios emitimos solicitudes GET en el endpoint $/users$

Módulo Servidor

- **SYS-COM-sv-01:** El servidor recibirá los mensajes de los clientes a través de una API.
- **SYS-COM-sv-02:** Los mensajes serán procesados por el módulo de procesamiento de mensajes antes de ser almacenados.
- **SYS-COM-sv-03:** El servidor almacenará los mensajes en colas cíclicas por sala, garantizando que solo se almacenen los últimos N mensajes por sala. Sólo existe una única sala.
- **SYS-COM-sv-05:** El servidor entregará los mensajes en tiempo real a los clientes registrados en la sala, asegurando que lleguen de manera eficiente y sin latencia significativa.
- **SYS-COM-sv-07:** El servidor devolverá solo los mensajes nuevos que no hayan sido entregados previamente a través del patrón de polling, utilizando el endpoint `/messages?lastMessageId=<id>`.
- **SYS-COM-sv-08:** El servidor gestionará un historial de mensajes por cada sala, accesible solo para los clientes activos en la sala.
- **SYS-COM-sv-09:** El servidor gestionará las colas cíclicas de mensajes para asegurar que, cuando el número de mensajes exceda el límite máximo, los más antiguos sean almacenados en la base de datos y eliminados automáticamente del buffer.

1.3.3 Requisitos Generales del Sistema Chat

- **SYS-GI-01:** El sistema debe permitir registrar y supervisar errores técnicos y de usuarios.

- **SYS-AU-01:** El sistema debe permitir la generación de reportes de actividad, incluyendo mensajes eliminados, sanciones y cambios en las salas.
- **SYS-AU-02:** Los mensajes y registros de actividad deben almacenarse para recuperación o auditoría. iempo real.

1.4 Diseño del sistema y especificaciones técnicas

1.4.1 Características de los Mensajes

- **Longitud máxima de un mensaje:** El sistema debe permitir mensajes de hasta 500 caracteres. Este tamaño puede ser ajustado dependiendo de los requisitos de almacenamiento y procesamiento del servidor y de los ficheros de configuración del cliente chat. Mensajes mayores pueden requerir optimización o compresión.
- Los mensajes enviados desde el servidor deben tener un tamaño máximo de 1 MB configurable, adecuado para múltiples mensajes y contenido adicional como logs o eventos del sistema.
- **Formato de los mensajes:** Los mensajes deben ser en texto plano, incluyendo elementos como el nombre de usuario, timestamp (fecha y hora exacta del envío).
- **Envío/recepción de los mensajes:** Los mensajes antes de su envío se procesarán para ver que no contienen código malicioso o que pueda producir algún error al mostrarse en pantalla o al tratarse en el servidor. Además, antes del envío se serializará en un formato JSON y en la recepción se desserializará del formato JSON dejándolo en texto plano antes de su presentación por pantalla.

1.4.2 Tiempos de Polling

- Si se decanta por la utilización de un mecanismo de comunicación mediante polling. El sistema realizará las peticiones de actualización entre cliente y servidor cada 5 segundos por defecto. Esta frecuencia es configurable según la carga de trabajo o requisitos de latencia.

1.4.3 Control de errores

- **Logs de eventos y errores:** El sistema debe registrar toda acción importante (entrada/salida de usuarios, cambios de estado, mensajes enviados, errores, etc.). Este control se llevará tanto en el cliente como en el servidor. Cada traza de log mantendrá un formato inicial fijo: <fecha> <modulo> : <usuario> <sala> : <situación>

1.4.4 Módulos del cliente

FrontEnd de la UI

Cada pantalla, zona y acción de la misma se transforma en una función necesaria del cliente chat asociada al frontend:

- **Pantalla de inicio:** Incluirá un formulario de conexión al chat donde el usuario puede introducir su <nickname> deseado. Cuando el usuario se desconecta del chat vuele a esta pantalla.
- **Pantalla de chat:** Incluirá 3 zonas específicas y 3 acciones.
  - **Zona de historial de mensajes:** Para mostrar los mensajes de la sala seleccionada.
  - **Zona de usuarios activos en al sala focal:** Para mostrar los usuarios de la sala seleccionada.
  - **Modo oscuro y claro:** Configurable por el usuario.
  - **Botón salida sala seleccionada:** Para salir del chat.

Control de Acceso

Se realizará una autenticación débil. Cualquier persona puede entrar al chat con un nickname. El único control es la utilización de un token jwt de sesión, que servirán para que un usuario pueda enviar mensajes a la sala del chat. Con esto evitamos que cualquier persona que no haya accedido al sistema pueda enviar mensajes.

Envío de Mensajes y actualización de históricos en Tiempo Real

El cliente debe ser capaz de recibir y mostrar nuevos mensajes en tiempo real sin necesidad de recargar la página. Una posible solución es el uso de **API REST** para la autenticación inicial y para enviar mensajes. Al inicio, el cliente realiza una solicitud `POST` al servidor con las credenciales del usuario. El servidor valida la información y, si es correcta, envía un **token JWT** (JSON Web Token) que el cliente almacenará para autenticar las siguientes interacciones. Este token será utilizado en todas las solicitudes posteriores, asegurando que el cliente permanezca autenticado durante su sesión, permitiéndole, por ejemplo, mantener su *nickname* y foto de perfil.

Para la actualización del historial de mensajes o la lista de usuarios activos en la sala, se puede utilizar una solución híbrida que combine **API REST** y **WebSockets**. La **API REST** se puede usar para solicitudes iniciales o para cargar información estática, como el historial de mensajes al ingresar a una sala. Sin embargo, para la interacción en tiempo real, como la recepción de nuevos mensajes, **WebSockets** es más adecuado. Con **WebSockets**, el cliente mantiene una conexión persistente con el servidor, lo que permite el intercambio de mensajes de manera continua y eficiente. Esto garantiza que los mensajes se entreguen en tiempo real, sin la necesidad de recargar la página o hacer solicitudes repetitivas.

Otra opción que se puede considerar es el uso de **Server-Sent Events (SSE)** para la transmisión de mensajes en tiempo real. **SSE** es una tecnología eficiente y ligera que permite al servidor enviar actualizaciones al cliente de manera continua a través de una única conexión. Es especialmente adecuada para aplicaciones en las que el servidor necesita enviar datos de forma unidireccional hacia el cliente, como cuando se actualiza la lista de usuarios activos o se carga el historial de mensajes. Aunque **WebSockets** es más adecuado para aplicaciones de alta interacción, donde el cliente y el servidor intercambian mensajes bidireccionalmente, **SSE** es ideal para aplicaciones con menor latencia y actualizaciones unidireccionales.

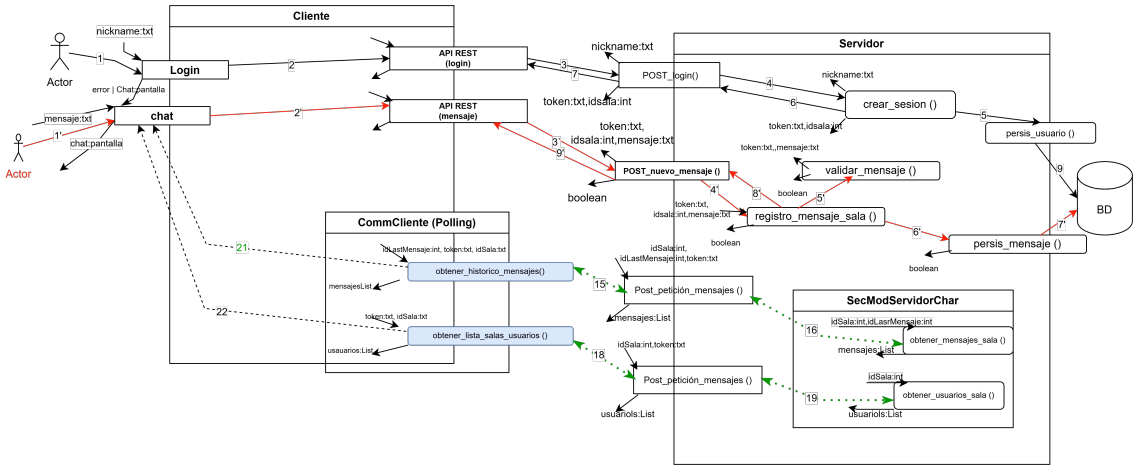

Figura 1.1. Flujo Autenticación y comunicación APIREST + Sockets (Polling)

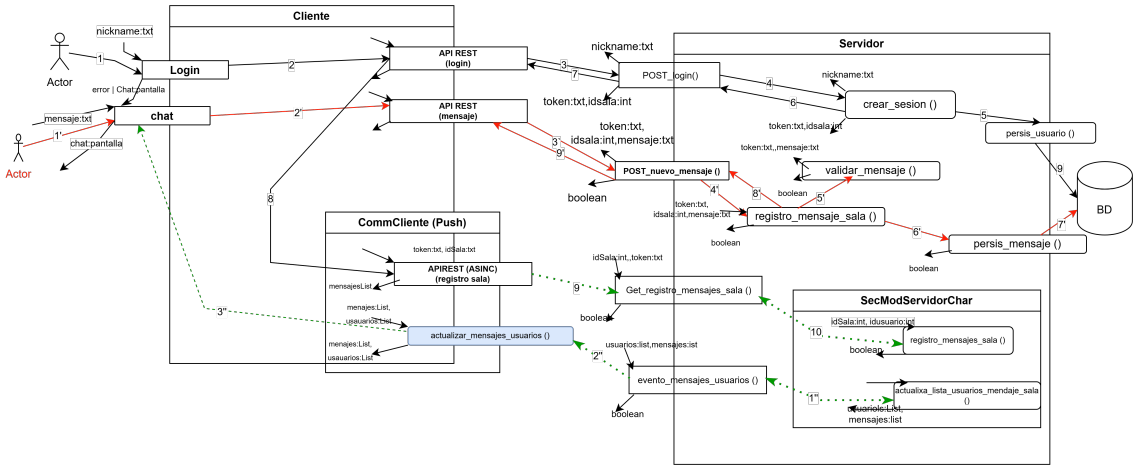

Figura 1.2. Flujo Autenticación y comunicación APIREST + SSE

1.4.5 Módulos del Servidor

Módulo de Gestión de Usuarios (User Management Module)

Responsabilidades:

- Registrar usuarios en el sistema (temporales y externos) mediante la generación de token's JWT de sesión.
- Validar la unicidad de los *nicknames*.
- Supervisar la conexión y desconexión de usuarios mediante los token's de sesión jwt y actividad de los mismos. Se almacena la fecha de la última actividad si pasa cierto tiempo sin actualizarse es desconectado del chat.

Interfaces:

- `POST /users/login`: Registra acceso de usuarios al chat.
- `PATCH /users/{id}/disconnect`: Gestionar desconexiones.

Funciones para la Lógica de Usuario

- **token:txt registrar_usuario (nickname:txt)**: El servidor debe registrar y mantener usuarios en el sistema.
- **boolean verificar_unicidad_nickname(nickname:txt)**: El servidor debe validar la unicidad de los nombres de usuario (nicknames).
- **token:txt crear_token_sesion(nickname:txt)**: El servidor debe crear un token de sesión con un tiempo máximo de permanencia en el chat.
- **boolean registro_ultima_accion(nickname:txt)**: El servidor ¡registra la fecha de la última acción del cliente para que un proceso en el servidor pueda comprobar usuarios posiblemente desconectados y limpiar y actualizar el listado de usuarios activos.
- **date obtener_ultima_accion(nickname:txt)**: ¡Devuelve la fecha de la última acción ejecutado por el usaurio.
- **boolean cambiar_estado_usuario(nickname:txt, estado:boolean)** : El servidor debe gestionar los estados de los usuarios (activo, libre, bloqueado).
- **boolean opt_user_sala (nick:txt, sala:int, opt:txt)**: registrar o sacar al usuario de una sala.

Sólo existe una única sala.

Módulo de Gestión de Mensajes (Message Management Module)

Responsabilidades:

- Recibir y procesar los mensajes enviados por los usuarios a través de una API. Cuando los recibe, crea un objeto mensaje en el servidor con una marca de fecha y hora. Añade el id de usuario a partir del token de sesión y saca su nickname.
- Validar el token de sesión jwt asociada a la petición de mensaje.
- Validar y verificar mensajes (asegurarse de que no contengan código malicioso).

Interfaces:

- `POST /messages/{tokenjwt}`: Permite recibir un mensaje nuevo.
- `GET /messages/`?$idMensaje = id$: Permite recuperar el texto del mensajes.

Funciones internas:

- `boolean validar_token(token:txt)`: Valida el token de sesión jwt del mensaje recibido.
- `boolean verificar_mensaje(mensaje:txt,token:txt)`: procesar mensajes antes de almacenarlo y reenviarlo al cliente. Si el mensaje contiene contenido que puede provocar algún error en el cliente, entonces se desecha sin más.
- `<mensaje> POST_Recibir_mensaje(mensaje:txt, token:txt)`: Crea un objeto mensaje, añadiendo no sólo el texto del mensaje, sino el nickname e id asociado al token de sesión.Además se crea un id para el nuevo mensaje.
- `int almacenar_mensaje(mensaje:<Mensaje>)`: Almacena mensajes por sala en la BD.

Módulo de Gestión de Salas

Responsabilidades:

- Crear y gestionar salas. Sólo existe una única sala pero crearemos la entidad sala.
- Gestionar usuarios en salas (agregar, eliminar).
- Mantener cola de mensajes por sala. Sólo existe una única sala. El tamaño es configurable y se puede modificar al arrancar el servidor.
- Devolver listado de mensajes nuevos desde el último dado.
- Devolver listado de mensajes en la cola d mensajes de la sala.
- Persistencia de los mensajes en la BD por sala. Invoca módulo de persistencia de mensajes.
- Almacenar mensajes en colas cíclicas asociada a la sala donde se envía el mensaje. Invoca módulo de gestión de salas.
- Proveer un historial de mensajes para clientes registrados en la sala.

Interfaces:

- `POST /rooms`: Crear una sala.
- `DELETE /rooms/{id}`: Eliminar una sala.
- `GET /rooms/{id}`: Obtener detalles de una sala.
- `POST /messages/`?$idSala = id\&tokenjwt$: Permite recibir un mensaje nuevo.

- `POST /rooms/{id}/users`: Agregar usuarios a una sala.
- `DELETE /rooms/{id}/users/{user_id}`: Expulsar usuarios de una sala.

Funciones internas:

- **boolean crear_sala(nombre_sala:txt)**:Se crea una única sala al arrancar el servidor, el nombre es configurable en el arranque del servidor.
- **gestion_acceso_sala(id_sala;int, nickname:txt, opt:txt)**: El sistema debe controlar el acceso o salida de um usuario en la sala.
- **boolean eliminar_sala ()**: Eliminar sala, que permite vaciar colas de mensajes y usuarios asociados a la sala. Se invoca en un apagado del servidor ordenado.
- **List obteber_usuarios_sala(id_sala:int)**: Devuelve listado de usuarios activos en la sala.
- **void almacenar_mensajes_sala (mensaje:txt, id_sala;int, nickname;txt)**: Almacena mensaje en la cola de mensajes de la sala. Comprueba si llega a su límite, en este caso almacena los mensajes en la base de datos antes de sobrescribir os mensajes viejos.
- **List obtener_nuevos_mensajes_sala(id_sala,id)**: Listar los mensajes nuevos a partir del último dado.
- **List obtener_mensajes_sala(id_sala)**: Listar los N últimos mensajes de la sala.

Módulo de Seguridad y adminsitración

Responsabilidades:

- Autenticar usuarios y gestionar tokens de sesión.
- Comprobar tokens de sesión asociados a las peticiones del cliente.
- Comprobar contenido de un mensaje
- Comprobar usuarios desconectados de forma abrupta
- Registrar auditorías de acciones críticas (envío de mensajes, gestión de usuarios, etc.).

Interfaces:

- `POST /auth/token`: Generar tokens de sesión.
- `GET /auth/validate`: Validar tokens de sesión.

Proceso segundo plano:

`validar_conexion_usuarios ()`: Validar si los usuarios activos están realizando acciones en intervalos aconsejables, Si no es así se elimina el usuario porque se piensa que ha sido desconectado de forma abrupta.

`validar_sistema ()`: Validar tamaño de la base de datos para que no crezca demasiado por el almacenamiento de mensajes. Otras acciones.

Funciones Internas:

- **boolean validar_token_accion(token:txt, nickname:txt,opt:txt)**: Gestiona las acciones de usuarios, envia nuevo mensaje, petición mensajes nuevos, listado de usuarios activos, etc..
- **token:txt crear_token_sesion(nickname)**: e encarga de crear los tokens jwt de sesion.

Módulo de Comunicación

Responsabilidades:

- Establecer y mantener las conexiones entre el cliente y el servidor.
- Proveer notificaciones en tiempo real (usando WebSockets o eventos del servidor al cliente).
- Permitir que el cliente emita nuevos mensajes
- Permitir que el cliente pueda actualizar listado de mensajes en el chat y listado de usuarios activos.

Interfaces:

- `POST /newmessage/`: Endpoint para recibir nuevos mensajes. (si se usa APIREST para nuevos mensajes)
- `WebSocket /ws/chat`: Canal para notificaciones en tiempo real. (si usamos polling)
- `POST /ws/register`: Registrar un cliente para recibir notificaciones.(si usamos broadcast)
- `DELETE /ws/disconnect`: Desconectar un cliente del sistema.

Funciones Internas:

- **boolean registrar_usuario_en_sala(nickname:txt, id_sala:int)**: Registra un usuario en una sala para que pueda recibir mensajes, si se usa broadcast.
- **boolean recibir_mensaje_cliente(nickname:txt, id_sala:int, mensaje:txt,tocken:txt)**: Envía un mensaje desde un cliente a la sala correspondiente.
- **List consultar_mensajes_sala_polling(id_sala:int)**: Consulta los mensajes nuevos en la sala mediante polling.
- **List obtener_historial_mensajes(id lastMessage:int, id_sala:int)**: Gestiona el historial de mensajes de una sala para mantener un registro.
- **List consultar_listado_usuarios_en_sala_polling(, id_sala:int)**: Consulta el listado de usuarios presentes en la sala mediante polling.

1.5 Primer tentativa del diseño del sistema chat

1.5.1 Diagrama de clases

Se muestra la primera versión del diagrama de clases principales que conlleva el diseño e implementación del chat, tanto el módulo cliente como servidor. Notar que no se han añadido las clases asociadas al control de formularios y gestión de sesión del usuario. Se incluye en el esquema las clases base de usuario, sala y mensaje que son las principales. Además se añaden las clases y módulos presentes en el servidor. Notar que no se han añadido en el esquema las funciones get/set para los atributos de las clases. Un resumen del esquema de clases para la implementación de nuestro sistema chat se puede ver en la imagen 1.3

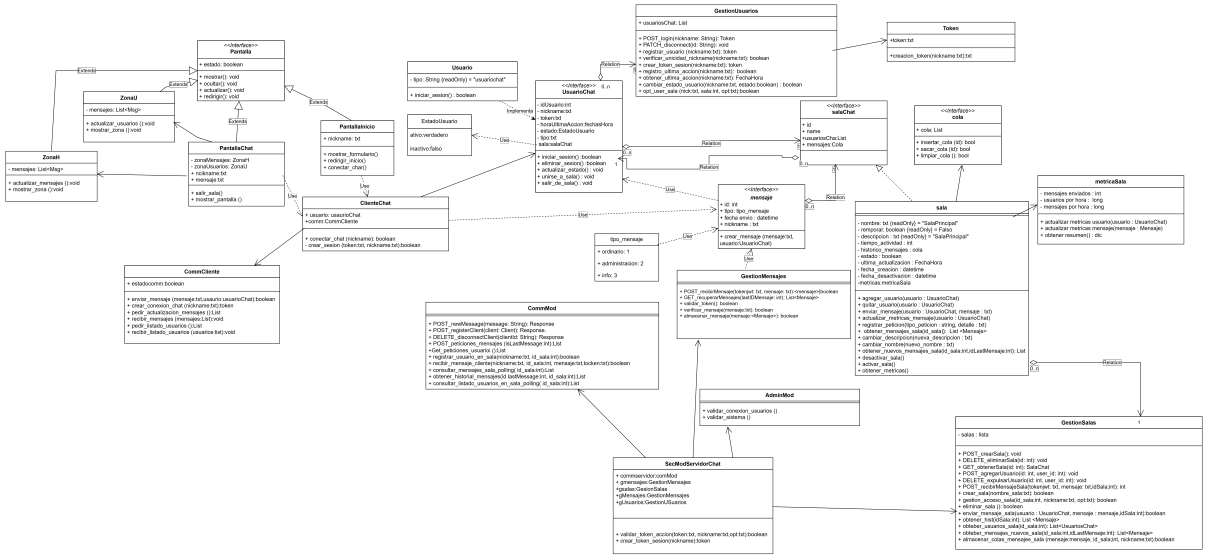

Figura 1.3. Diagrama Clases principales

1.5.2 Diagramas de Interacción

Inicio de Sesión

En la figura 1.4 podemos ver un diagrama de interacción el proceso de entrada de un usuario al chat. Un resumen de los pasos son:

1. El usuario introduce su nombre.
2. El cliente envía la solicitud al servidor para validarlo y crea un token de sesión JWT.
3. El servidor registra el usuario en la sala principal y responde con éxito o error.
4. Si responde con éxito. El cliente crea un usuario y lo registra en el polling para la sala principal. Envía peticiones a través del polling para refrescar la pantalla del chat (histórico de mensajes en al sala, usuarios activos en la sala).

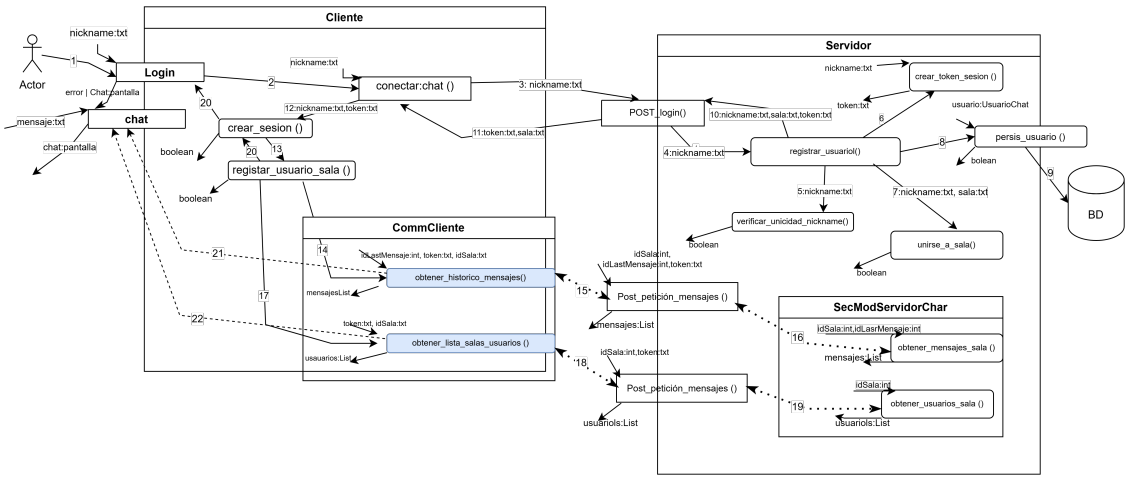

Figura 1.4. Inicio sesión en el cliente chat

Envío de Mensajes

En la figura 1.5 podemos ver un diagrama de interacción el proceso de envío de mensajes de un usuario a la sala del chat. Un resumen de los pasos son:

1. El usuario escribe un mensaje.
2. El cliente envía el mensaje al servidor.
3. El servidor comprueba y valida el token, el formato del mensaje, crea un objeto mensaje y lo almacena en la base de datos si todo ha ido bien.
4. El servidor almacena el objeto mensaje en al cola de mensajes de la sala correspondiente (sólo existe una única sala)
5. El servidor devuelve true o false se se ha podido enviar correctamente el mensaje
6. El cliente refresca la pantalla chat Si ha ido mal le sale un mensaje de error que sólo puede ver él.
7. El cliente cada cierto tiempo refresca la pantalla del chat con el histórico de mensajes de la sala ordenado por fecha d envío y el listado de usuarios presentes. Esto lo realiza bien pro petición al servidor o si el servidor emite un evento de actualización de datos.

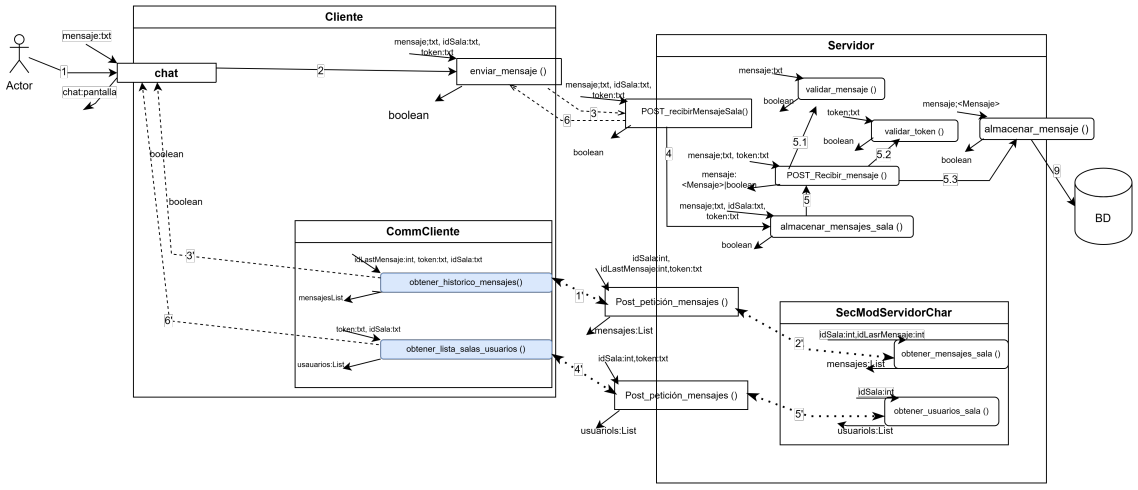

Figura 1.5. Flujo envío mensaje del usuario a la sala del chat

Mecanismos Comunicación

A continuación, se presentan dos diagramas de interacción que ilustran los procesos de comunicación en el sistema de chat. El primer diagrama describe el uso de API REST para gestionar la conexión al chat y el envío de mensajes nuevos por parte del usuario, complementado con sockets para recuperar el historial de mensajes de la sala y la lista de usuarios presentes en ella, implementando el patrón polling (1.6).

El segundo diagrama muestra una alternativa basada en API REST combinada con tecnología SSE (Server-Sent Events), que implementa el patrón push para una transmisión más eficiente de datos en tiempo real (1.7).

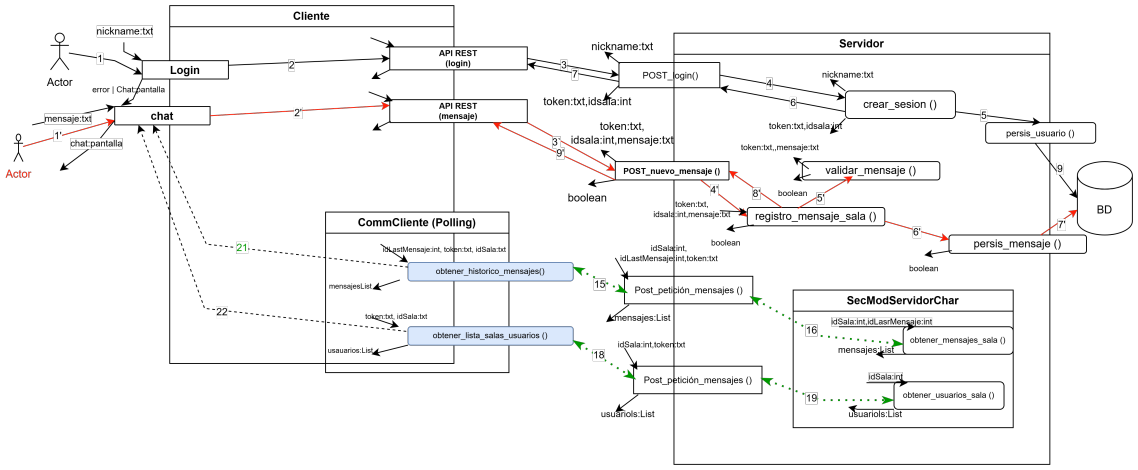

Figura 1.6. Flujo Autenticación y comunicación APIREST + Sockets (Polling)

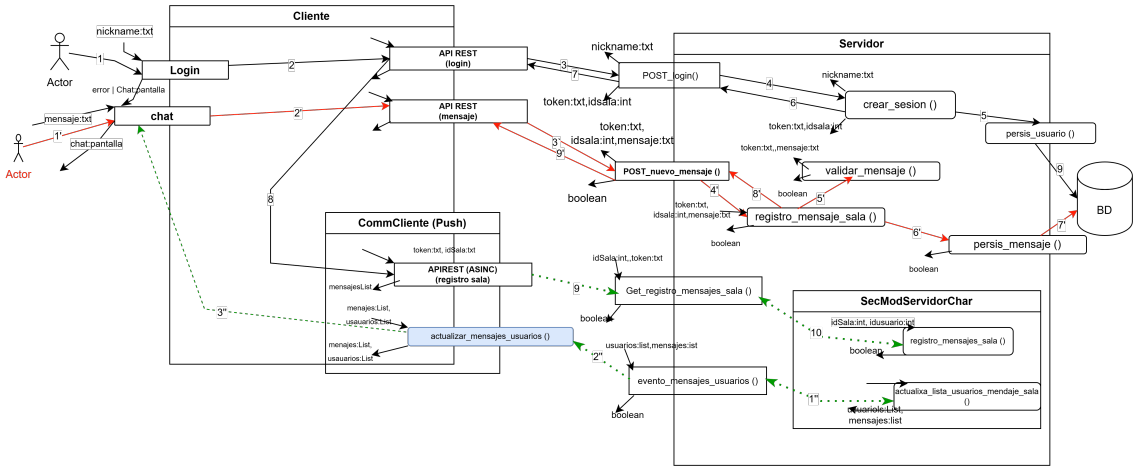

Figura 1.7. Flujo Autenticación y comunicación APIREST + SSE

1.6 Implementación del sistema Chat

Tecnologías Utilizadas

- **Frontend:** Node.js, typescript con React.js o React Native (según la plataforma deseada: web o móvil). El diseño será responsive utilizando styled-components o Material-UI.
- **Backend:** Go. Ambos soportan APIs RESTful y autenticación basada en JWT.

1.6.1 Diseño de las Comunicaciones

1.6.2 Detalles del Patrón Polling

El patrón de polling se implementa mediante las siguientes acciones y requisitos:

Solicitud de Nuevos Mensajes (Polling del Cliente)

Los clientes deben hacer una solicitud periódica para obtener los nuevos mensajes de una sala. Cada vez que un cliente hace una solicitud GET, el servidor debe devolver únicamente los mensajes que hayan

sido enviados después del último mensaje conocido por el cliente. Esto se logra mediante el parámetro `lastMessageId` que los clientes incluirán en la solicitud para indicar hasta qué mensaje se ha leído.

Colas Cíclicas para Mensajes

Cada sala tiene su propia cola cíclica que almacena los mensajes. El tamaño máximo de la cola está limitado a N mensajes. Cuando se envía un nuevo mensaje y la cola está llena, el mensaje más antiguo es eliminado para dar espacio al nuevo. Esto permite que las salas no sobrecarguen la memoria con mensajes antiguos que ya no son relevantes.

Actualización de la Cola Cíclica

Cada vez que un mensaje nuevo es recibido, se agrega a la cola cíclica de la sala correspondiente. Los mensajes deben ser entregados a los clientes registrados en la sala, garantizando que todos los mensajes sean distribuidos en orden de llegada.

Manejo de Sesiones Activas y Registro de Clientes

Los clientes deben estar registrados en una sala específica para recibir mensajes. Si un cliente no está registrado en la sala, no podrá recibir los mensajes. El servidor debe mantener un listado de usuarios y salas donde se encuentran activas cada usuario.

Lista d esalas y usuarios

Al igual que el histórico de mensajes, el cliente chat debe mantener un listado de salas activas y existentes en el chat. cuando el usuario entra al sistema chat, es el cliente quién obtendrá del servidor la salas existentes y los usuarios presentes en cada sala. Al igual que con el histórico de mensajes, el cliente utiliza el polling para actualizar dicha información. Si se opta por no usar polling y si SSE, es el cliente el que debe emitir una petición de actualización mediante un APIREST.

La diferencia entre usar sockets dedicados sobre los cuales realizar peticiones de actualización del histórico de mensajes y usar tecnología SSE es quién mantiene el listado de salas activas del usuario. En el caso de usar SSE, al ser una comunicación unidireccional, el servidor debe mantener un registro de qué clientes están activos en cada sala. Por tanto, cada vez que el usuario sala o entra de una sala debe enviar una petición de registro o de salida de sala al servidor, esto se realizará mediante un apirest (endpoint /joinRoom<salas>). Si por el contrario se usan sockets dedicados, es el mismo cliente chat quien mantiene la salas donde el usuario está activo. Es en la petición de actualización de histórico de mensajes donde el cliente le envía junto al token de sesión JWT, el listado de salas activas.

Tiempo de Polling y Control de Solicitudes

El servidor debe definir un intervalo de tiempo mínimo entre cada solicitud de polling para evitar sobrecargar el sistema. Los clientes deben realizar las solicitudes de polling de manera eficiente y no con demasiada frecuencia para evitar una carga innecesaria en el servidor.

En el caso de usar SSE, el cliente refresca cada cierto tiempo pero no realiza ninguna petición al servidor. Es el servidor quien actualiza los históricos de mensajes en el cliente, cuando en ese tiempo refresque el frontend, le aparecería al cliente el listado de mensajes actualizado.

Respuestas a Solicitudes de Polling

El servidor debe responder a las solicitudes de polling de manera rápida, enviando solo los mensajes nuevos y asegurándose de que los mensajes se entreguen correctamente a los clientes correspondientes.

Integración con el Módulo de Procesamiento de Mensajes

El servidor debe recibir los mensajes de los clientes a través de la API REST (endpoint: /messagesnew?<mensaje>) y una vez comprobado si el token de sesión es válido, pasarlos al módulo de procesamiento de mensajes, donde se validarán, procesarán, y luego almacenarán en las colas cíclicas de la sala. Una vez procesado el mensaje, se distribuye a los clientes registrados en la sala mediante el patrón de polling o el sistema SSE actualizará las estructuras del cliente que mantenga el histórico de mensajes.

Eliminación Automática de Mensajes Antiguos

El servidor debe eliminar automáticamente los mensajes antiguos cuando se alcance el límite de N mensajes en la cola cíclica, lo que garantiza que la memoria no se llene innecesariamente y que solo los mensajes recientes sean accesibles.

CAPÍTULO 2

Algoritmo de Berkeley

El algoritmo de Berkeley es un método de sincronización de relojes en sistemas distribuidos. Se basa en la existencia de un líder (coordinador) que ajusta los relojes de los nodos en el sistema para mantener la coherencia temporal.

2.1 Objetivos

Refinar el diseño del "Algoritmo de Berkeley" que aparece en `"Apuntes_NotacionDiseño.pdf"` basándolo en sockets de ZeroMQ. A continuación, se explica paso a paso el algoritmo:

- **Inicio del proceso de sincronización:** El líder toma la iniciativa para sincronizar los relojes en el sistema y envía una petición a todos los nodos seguidores solicitando sus marcas temporales actuales.
- **Seguidores responden al líder:** Cada seguidor responde al líder con su marca temporal actual y, opcionalmente, con el tiempo que tardó en procesar la solicitud (latencia de ida y vuelta).
- **Líder calcula los ajustes:** Una vez que el líder recibe todas las marcas temporales, calcula el tiempo promedio ajustado del sistema (considerando posibles retrasos) y determina la diferencia de tiempo entre el promedio y el reloj de cada nodo.
- **Líder envía ajustes:** El líder comunica a cada nodo un ajuste temporal que especifica cuánto adelantar o retrasar su reloj.
- **Seguidores aplican ajustes:** Cada seguidor ajusta su reloj local según el valor recibido. En algunos casos, los ajustes se aplican de forma gradual para evitar cambios bruscos.
- **Finalización:** Los relojes de los nodos quedan sincronizados dentro de los límites de error permitidos por la red.

El algorimo de berkeley representado en IOAutomata se muestra en al figura 2.1. Adaptaremos eta representación usando zeroMQ para las comunicaciones.

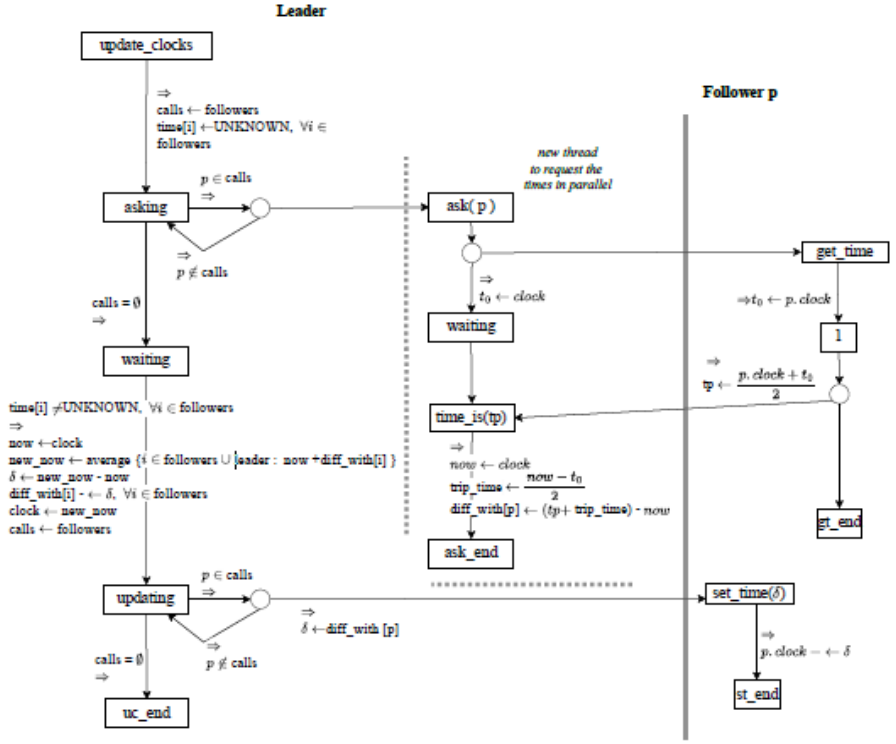

Figure 8: Berkeley Algorithm

Figura 2.1. Algoritmo Berkeley JAutomata (ref. Jordi Ballester «notacion.pdf»)

2.1.1 Cambios en el diagrama para representar ZeroMQ

Se proponen los siguientes cambios en el diagrama del algoritmo Berkeley para integrar sockets ZeroMQ:

- **Líder (Request/Reply):**

  El lider mantiene una nueva variable $req[i]$ que permite verificar que un nodo seguidor ha respondido correctamente a la solicitud de tiempo del líder. Dicha variable se inicializa a false para cada follower ($req[i] = false$ y $time[i] = UNKNOWN$)

  En el flujo para cada hilo donde el lider obtiene el tiempo de cada nodo p se realia lso sigueitnes cambios:

  - Cambiar el nodo *asking* para reflejar el envío de un mensaje ZeroMQ con el patrón REQ (Request) para pedir los tiempos de los seguidores. Notar que cada nodo p tiene un REP para esperar las peticiones del lider. (El líder abrirá un socket REQ para cada nodo y enviará un mensaje de solicitud de tiempo.)
  - Eliminar el nodo *waiting* ya que en el anterior ya espera la respuesta REP (Reply) del nodo p. Además, si se llega el tiempo t_p del nodo p actualizamos el valor de $req[p] = True$ para controlar que followers devuelven correctamente su tiempo.

  Una vez el lider obtiene todos los tiempos de cada follower, modificamos las sigueintes operaciones;

- Calculamos el tiempo promedio pero sólo con el número de followers que han contestado correctamente, es decir, aquellos que tienen true en la variable req. Por tanto, modificamos el cálculo:

$$\texttt{new\_now} \leftarrow \text{average}\,(i \in \text{followers} \wedge \text{req}[i] = \text{true} \cup \text{leader} : \text{now} + \text{diff\_with}[i])$$

- Modificamos la variable `diff_with` solo para los nodos que han contestado con su tiempo:

$$\texttt{diff\_with}[i] \leftarrow \delta, \forall i \in \text{followers}, \text{req}[i] = \text{true}$$

- Creamos el conjunto `call` sólo con estos nodos que tienen la variable `req` a true:

$$\texttt{call} \leftarrow \{i \in \text{followers} : \text{req}[i] = \text{true}\}$$

  De esta forma, el líder sólo enviará el delta para que modifique su tiempo local solo a los nodos que han contestado.

- **Seguidores (Request/Reply):**
  - Cambiar el nodo *get_time)* para representar la recepción de una solicitud REQ del líder.
  - Añadir un nodo para que cada seguidor envíe una respuesta REP con el tiempo calculado o el ajuste solicitado.
  - Modificar el nodo set_time añadiendo un nodo en cada seguidor que indique que está esperando la solicitud de ajuste y que recibirá el ajuste del líder mediante una respuesta REP.

Con esta corrección, el cálculo de `new_now` ahora solo toma en cuenta los seguidores que han respondido correctamente (aquellos donde `req[i] = true`) y el líder, asegurando que solo se promedien los tiempos válidos. Esto es crucial para sincronizar los relojes de manera correcta en un sistema distribuido. Esta nueva variable ayuda a filtrar los nodos que han respondido correctamente. Si el líder envía una solicitud de tiempo (`REQ`) y no recibe respuesta (por algún fallo en la red o en el nodo), entonces `req[i]` se mantiene como `false`, y ese nodo no se incluye en el cálculo del promedio de tiempo (`new_now`). Mejora la robustez del algoritmo y facilita la comprensión y el manejo de errores. Asegura que solo los nodos que han respondido exitosamente sean tomados en cuenta para el cálculo del nuevo tiempo ajustado. Esto ayuda a evitar errores de sincronización si algún nodo no puede ser alcanzado por el líder.

El esquema del algoritmo, adaptado a ZeroMQ utilizando REQ/REP y con la utilización de la variable $req[i]$ se refleja en la imagen 2.2.

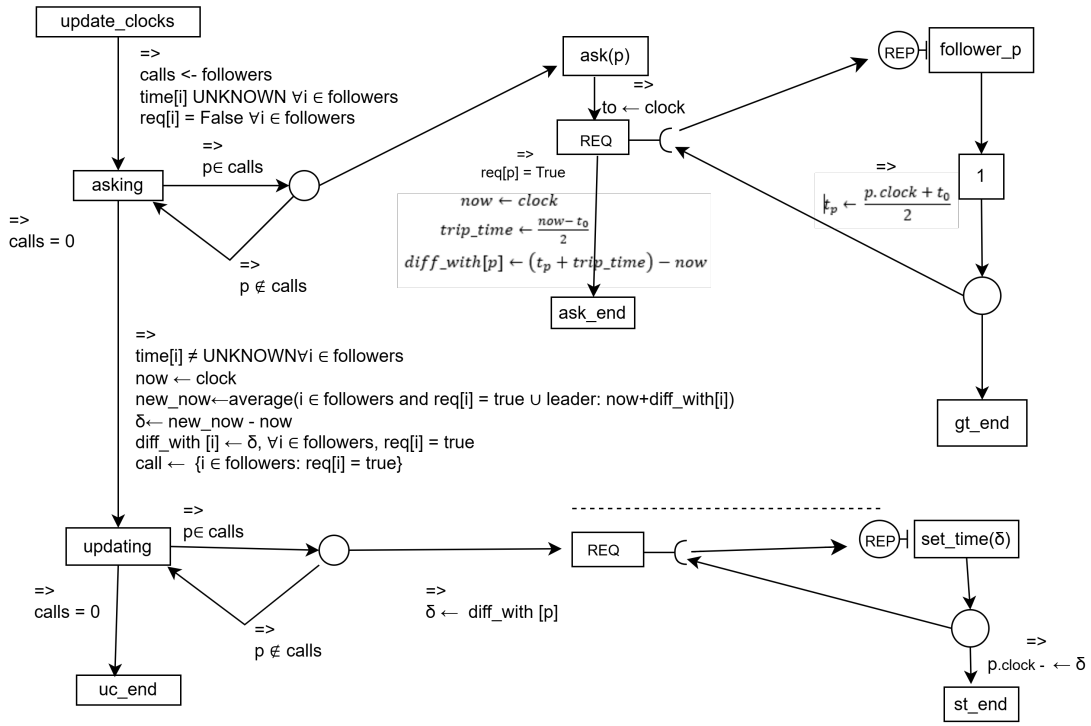

Figura 2.2. Algoritmo Berkeley con Sockets REQ/REP ZeroMQ