Introduction

Este documento describe el diseño tecnico para la API backend de una aplicacion de mensajeria en tiempo real (similar a Slack). La api maneja autenticacion y autorizacion, workspaces, canales, mensajes, attachments con un sistema que guarda los archivos en un servicio como S3 y un sistema de notificaciones centralizado.

Key Objectives

- Proveer de una API RESTful segura y escalable.
    - Proveer de un servicio de comunicación y notificaciones centralizado para un mejor manejo de los trabajadores.
        - Permitir comunicación en tiempo real gracias al servicio de websocket.
        - Manejar almacenamiento de archivos de forma desacoplada con un servicio como S3.

Use cases

Use Case 1: Team Channel Communication

Scenario: A software development team needs to discuss the progress of a sprint in a dedicated channel.

Flow:

1. A team member posts an update in the #development channel.
2. Other members comment and react to the message.
3. A thread is created to discuss a specific task without interrupting the general conversation.
4. The team receives notifications for new messages and relevant mentions.

Use Case 2: Direct Messages Between Employees

Scenario: A graphic designer needs to ask a developer for clarification about a specific functionality.

Flow:

1. The designer searches for the developer on the platform and sends a direct message.
2. The developer receives a notification and replies.
3. If necessary, they can escalate the conversation to a video call within the platform.

Use Case 3: Notifications and Mentions

Scenario: A manager mentions an employee in a message to request an update.

Flow:

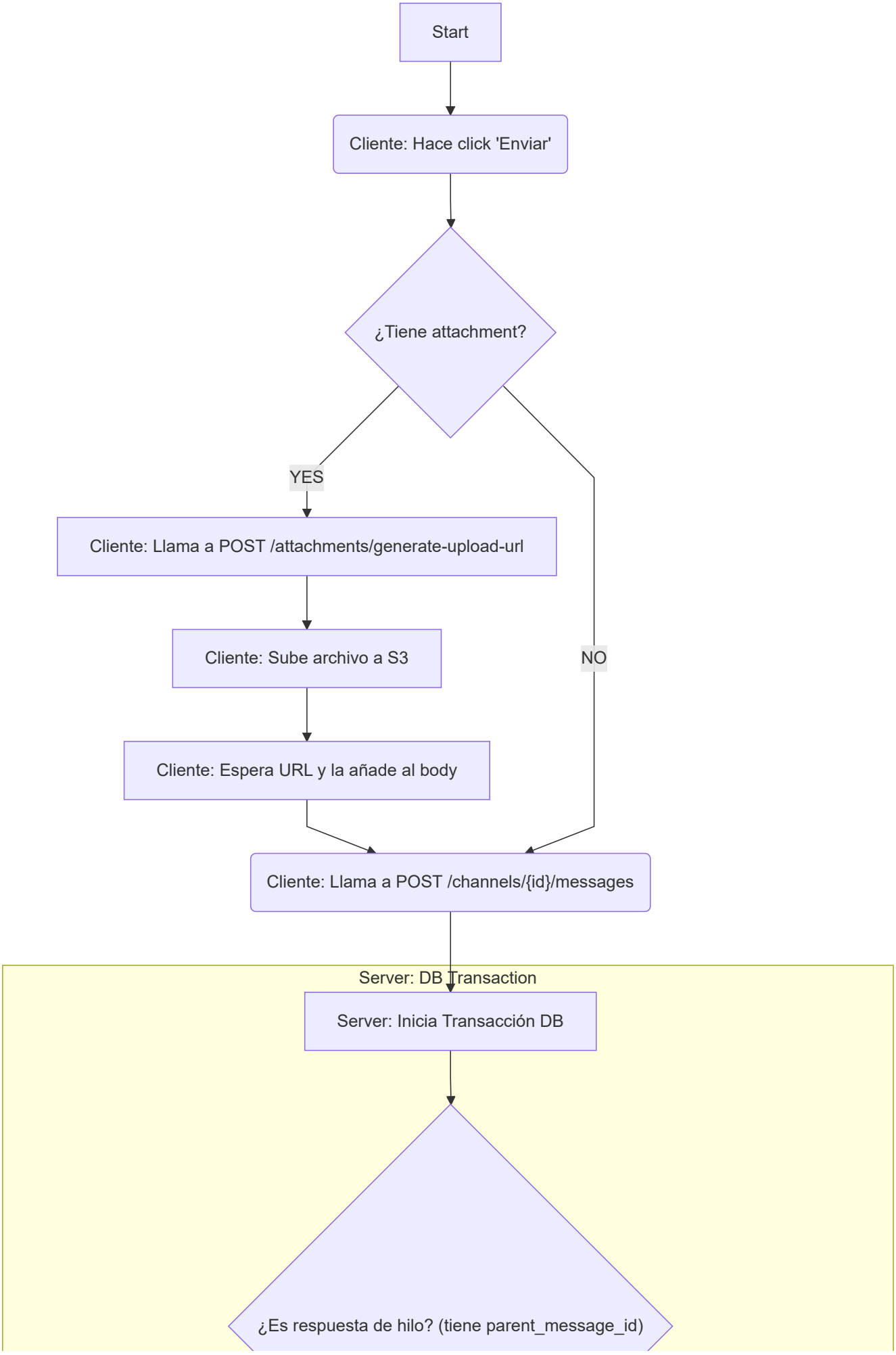
1. The manager writes a message in the #projects channel and mentions the employee with @name.
2. The employee receives a notification in their app.
3. The employee replies in the message thread.
4. The manager and other team members can follow and respond to the conversation as needed.

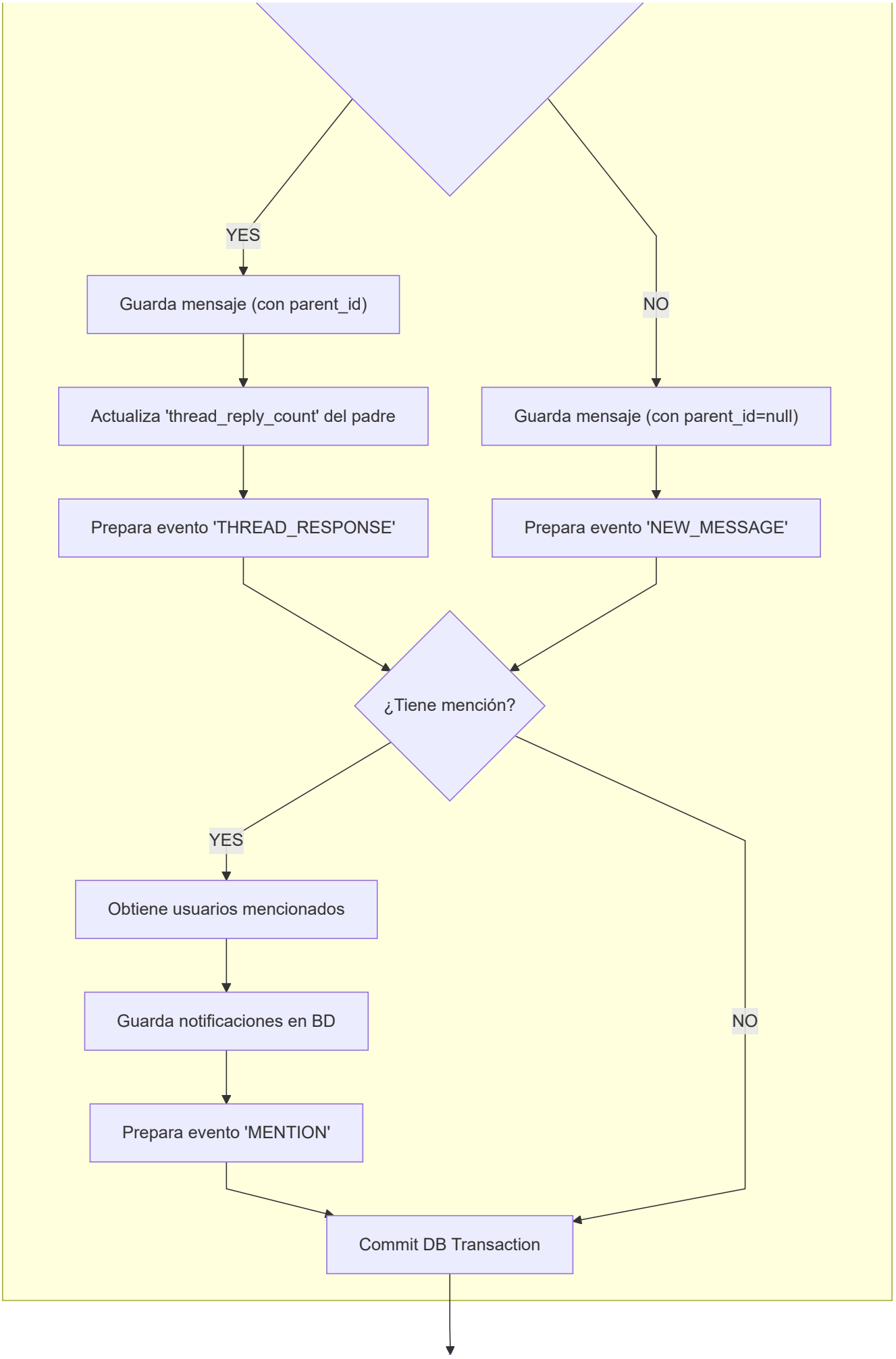
High level solution

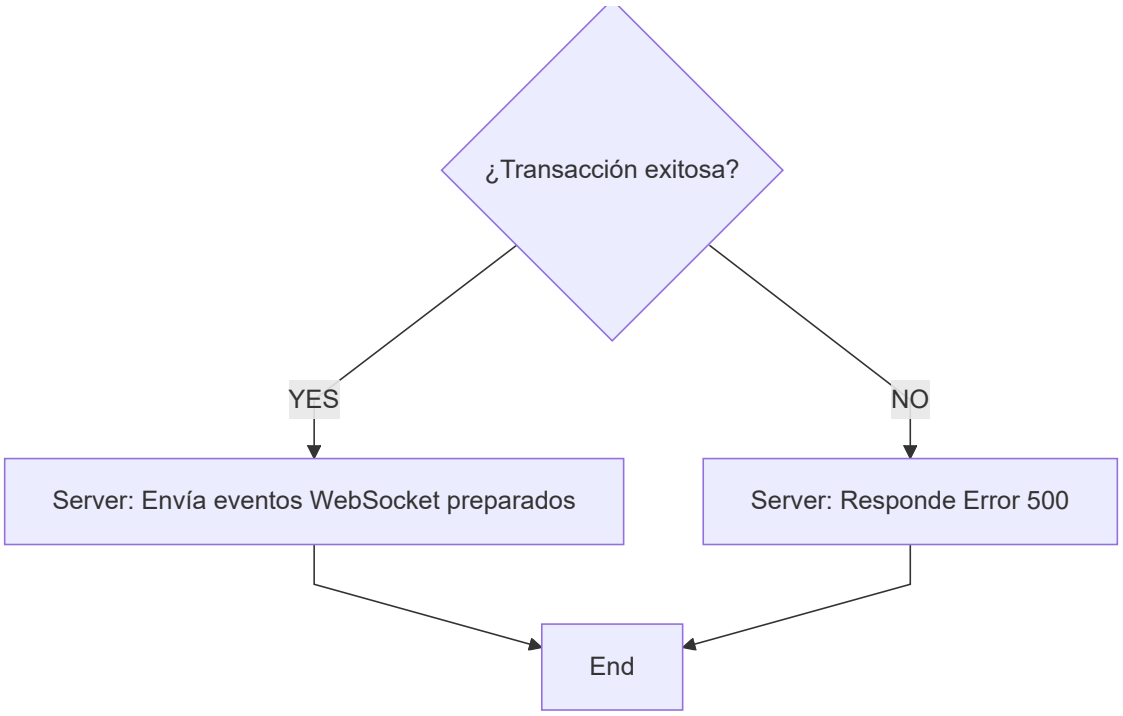
Se propone una solución de una API monolitica construida en PHP/Laravel, siguiendo una arquitectura RESTful. Se usará JWT Tokens para una autenticación segura y eficaz con Bcrypt para manejar las contraseñas, una base de datos relacional (Postgresql o Mariadb), y un servicio Websocket para el manejo de los eventos en tiempo real.

Flow diagram

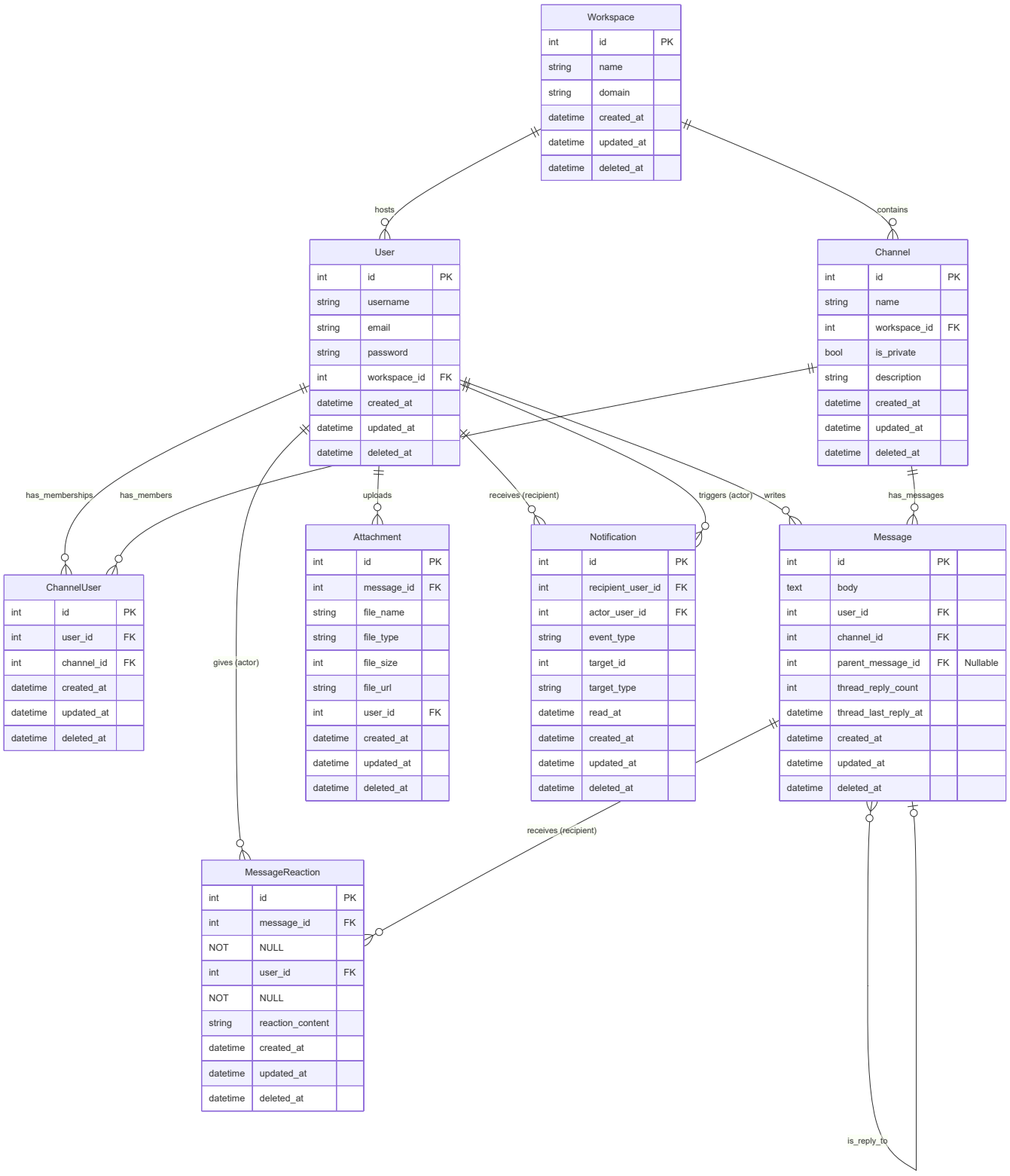
NEW MESSAGE


Start
Cliente: Hace click 'Enviar'
¿Tiene attachment?
YES
NO
Cliente: Llama a POST /attachments/generate-upload-url
Cliente: Sube archivo a S3
Cliente: Espera URL y la añade al body
Cliente: Llama a POST /channels/{id}/messages
Server: DB Transaction
Server: Inicia Transacción DB
¿Es respuesta de hilo? (tiene parent_message_id)


YES
Guarda mensaje (con parent_id)
Actualiza 'thread_reply_count' del padre
Prepara evento 'THREAD_RESPONSE'
NO
Guarda mensaje (con parent_id=null)
Prepara evento 'NEW_MESSAGE'
¿Tiene mención?
YES
Obtiene usuarios mencionados
Guarda notificaciones en BD
Prepara evento 'MENTION'
NO
Commit DB Transaction


¿Transacción exitosa?
YES
NO
Server: Envía eventos WebSocket preparados
Server: Responde Error 500
End

Entity diagram



Epics and Tasks

- EPIC: Módulo de Autenticación y gestión de usuarios
  Task1: Implementar endpoint `POST /register/` con validaciones y hashing de contraseña usando Bcrypt.
  Task2: Implementar lógica de creación de token JWT con secret_key.
  Task3: Implementar lógica de inicio de sesión devolviendo un JWT.

Task4: Implementar lógica de middleware para verificar estado de JWT y verificar autorización del servicio solicitado.
Task5: Implementar endpoint `GET /users/{me}` para obtener el perfil del usuario autenticado.
Task6: Implementar endpoint `PUT /users/{me}` para que el usuario actualice su propia información.

- EPIC: Módulo de Workspace
  Task1: Implementar endpoint `POST /wokspaces` para crear workspaces.
  Task2: Implementar endpoint `PUT /workspaces/{workspace_id}` para editar un workspace.
  Task3: Implementar endpoint `GET /workspaces` para listar todos los workspaces.
  Task4: Implementar endpoint `GET /workspaces/{wokspace_id}` para listar un workspace en específico.
  Task5: Implementar endpoint `DELETE /workspaces/id` para eliminar un workspace.
  Task6: Implementar endpoint `POST /workspaces/{workspace_id}/members` para añadir un usuario.
  Task7: Implementar endpoint `GET /workspaces/{workspace_id}/members` para listar todos los usuarios en un workspace.
  Task8: Implementar endpoint `DELETE /workspaces/{workspace_id}/members` para eliminar un usuario.

- EPIC: Módulo de Canales
  Task1: Implementar endpoint `POST /workspaces/{workspace_id}/channels` para crear un canal en el workspace.
  Task2: Implementar endpoints `POST /channels/{channel_id}/members` y `DELETE /channels/channel_id/members/user_id` para añadir y remover usuarios de un canal.
  Task3: Implementar endpoint `GET /workspaces/{workspace_id}/channels` para listar todos los canales en un workspace.
  Task4: Implementar endpoint `GET /channels/{channel_id}` para listar un canal.
  Task5: Implementar endpoint `PUT /channels/{channel_id}` para actualizar un canal.
  Task6: Implementar endpoint `DELETE /channels/{channel_id}` para eliminar un canal.

- EPIC: Módulo de mensajería
  Task1: Implementar lógica para la "creación" de threads usando los atributos de `parent_message_id` para saber a que mensaje pertenecen las respuestas, `thread_reply_count` y `thread_last_reply_at` para mejorar el agrupado de las respuestas.
  Task2: Implementar lógica en endpoint `POST /channels/{channel_id}/messages` (si este tiene un archivo debio hacer uso del servicio de `POST /attachments/generate-upload-url` para obtener un url pre-firmado para despues agregarlo al body) para enviar nuevo mensaje.
  Task3: Implementar lógica para publicar un evento al websocket para transmitirlo al cliente.
  Task4: Implementar endpoint `GET /channels/{channel_id}/messages` para obtener todos los mensajes de un canal.
  Subtask: Implementar paginación de los mensajes del canal (ej:?limit=50&cursor=timestamp).
  Task5: Implementar endpoint `GET /messages/{message_id}/replies` para obtener todas las respuestas de un mismo mensaje.
  Task6: Implementar endpoint `PUT /messages/{message_id}` para editar un mensaje.
  Task7: Implementar endpoint `DELETE /messages/{messaage_id}` para eliminar mensajes.
  Task8: Implementar endpoint `POST /dms/users/{user_id}/messages` para enviar mensajes directos entre usuarios.
  Subtask: La lógica del servicio debe primero identificar si un canal existe entre usuario1 y usuario2, si no crearlo y despues crear en bd el mensaje.
  Task9: Implementar endpoint `GET /dms/users/{user_id}/messages` para obtener todos los mensajes

directos entre usuario1 y usuario2.
Task10: Implementar endpoint `GET /dms` para obtener todos tus mensajes directos.

- EPIC: Módulo de Notificaciones
  Task1: Implementar lógica para crear notificaciones.
  Task2: Implementar lógica para obtener todas las notificaciones en orden y que aun no son leidas con el endpoint `GET /notifications` y utilizando el user_id desde el jwt.
  Task3: Implementar endpoint `PUT /notifications/{notification_id}` para "actualizar" notificaciones para poder saber cuando una notificacion a sido leida usando el atributo "read_at".
  Task4: Implementar lógica en los servicios de `POST /channels/{channelId}/messages` y `PUT /messages/{message_id}` para detectar las `@mention` del mensaje para enviar la notificacion personalizada de una mencion.
  Task5: Implementar lógica de publicar un evento al websocket para transmitirlo al cliente.
- EPIC: Módulo de Attachment
  Task1: Implementar endpoint `POST /attachments/generate-upload-url` para generar "urls pre-firmados" subiendo los archivos a un servidor de archivos como `s3` para poder usarlo en los mensajes con attachment antes de ser guardados en la base de datos.
- EPIC: Módulo de reacciones en mensajes
  Task1: Implementar endpoint `POST /messages/{message_id}/reactions` para anadir una reaccion al mensaje, puede ser mediante caracteres ascii, emojis o iconos personalizados.
  Task2: Implementar endpoint `DELETE:/message/{message_id}/reactions` para eliminar una reaccion especifica.
  Task3: Implementar endpoint `GET /messages/{message_id}/reactions` para obtener todas las reacciones que tiene un mensaje.
  Task4: Implementar lógica de publicar un evento al websocket para transmitir al cliente cuando se anade o elimina una reaccion.

Mejoras

. Búsqueda avanzada(mensajes, archivos)
. Estado (online/offline/ocupado/out of office)
. Añadir rol para "invitados" que puedan entrar a un workspace con acciones