Prueba Técnica: API de Gestión de Tareas (Node.js & Express)

Objetivo:
Evaluar tus habilidades en Node.js, Express, manejo de bases de datos (puedes usar SQLite para simplificar) y WebSockets. No es necesario desplegar en Azure/AKS para esta prueba.
Contexto:
Imagina que estamos construyendo un sistema simple de "Lista de Tareas Pendientes" (To-Do List) en tiempo real. Los usuarios deben poder agregar tareas y ver cómo la lista se actualiza para todos los clientes conectados sin necesidad de recargar la página.

id: Identificador único (autogenerado).
titulo: String, obligatorio, máximo 100 caracteres.
descripcion: String, opcional, máximo 500 caracteres.
status: String, por defecto "pendiente".
fechaCreacion: Timestamp, autogenerado al crear la tarea.
fechaActualizacion: Timestamp, autogenerado al crear y se actualiza con cada modificación.

Requisitos de la Aplicación:

API Backend (Node.js y Express):
- POST /tasks: Endpoint para crear una nueva tarea. Deberá recibir un JSON con { "titulo": "string", "descripcion": "string" }. Debe guardar la tarea en la base de datos y asignarle un ID único y un estado inicial (ej. "pendiente").
- GET /tasks: Endpoint para obtener todas las tareas existentes.
- PUT /tasks/:id: Endpoint para actualizar el estado de una tarea (ej. de "pendiente" a "completada"). Deberá recibir un JSON con { "status": "nuevo_estado" }.
- DELETE /tasks/:id: Endpoint para eliminar una tarea.

Base de Datos:
- Utiliza SQLite (o cualquier otra base de datos SQL ligera que prefieras, como PostgreSQL si te es más cómodo, pero SQLite es ideal por su simplicidad de configuración).
- La tabla de tareas debe tener al menos: id (PK, autoincremental), description (texto), status (texto, ej. "pendiente", "completada").

WebSockets (usando socket.io o ws):
- Cuando se crea una nueva tarea (POST /tasks), el servidor debe emitir un evento a todos los clientes conectados con la información de la nueva tarea (ej. newTask, payload: la tarea creada).
- Cuando se actualiza el estado de una tarea (PUT /tasks/:id), el servidor debe emitir un evento a todos los clientes conectados con el ID de la tarea y su nuevo estado (ej. taskUpdated, payload: { id: tareaId, status: nuevoEstado }).
- (Opcional) Cuando se elimina una tarea, emitir un evento para que los clientes la eliminen de su vista.

Frontend (Opcional y muy básico):
- No es necesario crear una interfaz de usuario compleja. Si decides incluir un frontend básico (HTML simple con JavaScript), debe conectarse al servidor WebSocket y actualizar dinámicamente la lista de tareas cuando reciba eventos del servidor.
- Si no incluyes frontend, asegúrate de que la funcionalidad WebSocket sea comprobable (puedes usar herramientas como wscat o Postman para probar WebSockets, o simplemente explicar cómo lo probarías).

Entregables:

- Un repositorio Git (puede ser en GitHub, GitLab, Bitbucket, o un archivo .zip con el proyecto) que contenga:
  - Todo el código fuente del backend.
  - Un archivo README.md con:
    - Instrucciones claras sobre cómo configurar el entorno (instalación de dependencias).
    - Instrucciones sobre cómo ejecutar la aplicación.
    - Una breve explicación de tus decisiones de diseño o cualquier consideración importante.
    - (Si no hay frontend) Cómo probar la funcionalidad WebSocket.
- No es necesario escribir tests unitarios, pero es un plus si lo haces.

Criterios de Evaluación:

- Funcionalidad: ¿La aplicación cumple con los requisitos especificados?
- Calidad del Código: Código claro, bien estructurado, mantenible y legible.
- Diseño de API: Uso correcto de métodos HTTP, códigos de estado y formato de respuesta.
- Implementación de WebSockets: Correcta emisión y manejo de eventos.
- Manejo de Base de Datos: Consultas SQL correctas, estructura de tabla adecuada.
- Manejo de Errores: Consideración básica de errores.
- Documentación: Claridad de las instrucciones en el README.

———————————————————————