Check list de codigo

Equipo: Vm´s

Nombre del verificador: _____

Fecha (D/M/Y) :_____

Criterios	Si	No
Caso de uso		
¿Has probado el sistema con un usuario que inicie sesión con Google utilizando NextAuth y se asegure de que puede acceder a las funciones del CRUD?		
Calidad del código		
¿El backend está organizado en rutas y controladores bien definidos? (Ej. **`/api/users.js`** para las operaciones de usuarios).		
¿Las funciones y variables en el backend tienen nombres descriptivos y claros? (Ej. **`createUser`**, **`findUserByEmail`**)		
¿Se usan middlewares (como el de autenticación) para verificar el acceso a rutas protegidas y no repetir código en cada endpoint?		
Eficiencia y rendimiento		
¿Las consultas MySQL son específicas y solo devuelven los datos necesarios (por ejemplo, usar **SELECT** solo para las columnas necesarias en lugar de **SELECT ***)?		

¿Se está utilizando **`async`** y **`await`** correctamente para manejar operaciones asincrónicas y evitar bloqueos en el servidor?		
¿Se ha implementado la paginación o algún tipo de optimización para evitar cargar grandes volúmenes de datos en una sola petición?		
Seguridad		
¿Estás validando los datos del usuario antes de enviarlos a la base de datos (por ejemplo, asegurando que los campos no estén vacíos o que los emails sean válidos)?		
¿Estás gestionando de manera segura los tokens de sesión de NextAuth (por ejemplo, asegurando que los tokens no se guardan en localStorage)?		
Documentación		
¿Cada función clave del backend está bien comentada, explicando su propósito y parámetros de entrada?		
¿Se ha documentado cómo configurar el proyecto, qué librerías estás utilizando (NextAuth, MySQL, etc.), y cómo se configuran las variables de entorno (ej. **`.env.local`**)?		
¿Se ha creado un documento o archivo README que explique cómo un desarrollador puede clonar el proyecto, instalar las dependencias, configurar la base de datos y probar el CRUD?		
Mantenibilidad		
¿Cada componente de Next.js tiene responsabilidades claras y limitadas, lo que permite que cambios en un componente no afecten a los		

demás?		
¿Cada función tiene una única responsabilidad? Por ejemplo, ¿los controladores de usuarios solo manejan la lógica de usuarios, sin mezclar otras funcionalidades?		
Portabilidad		
¿El backend utiliza variables de entorno (ej. **.env**) para la configuración de la base de datos, permitiendo cambiar entre entornos (desarrollo, producción) sin modificar el código?		
Cumplimiento de normativas		
¿Las credenciales de acceso a la base de datos y claves de API están almacenadas de manera segura en variables de entorno?		

Nombre del verificador: _____

Fecha (D/M/Y) :_____

Criterios	Si	No
Caso de uso		
Los botones tienen íconos con atributos aria-hidden para mejorar la accesibilidad.		
¿La interfaz responde adecuadamente a la interacción del usuario (clics, navegación, entradas de datos)?		
Se proporciona texto significativo que puede ser leído por tecnologías de asistencia.		
¿Se está validando correctamente el tipo de datos de las props recibidas?		
Calidad del código		
Se evita la duplicación de código innecesario.		
El código es fácil de leer y seguir, con nombres descriptivos y claros.		
¿Se aplican los principios de React correctamente, usando componentes funcionales, hooks, y separando la lógica de la presentación?		
Eficiencia y rendimiento		
¿Se manejan adecuadamente los cambios de estado para evitar renders innecesarios?		
¿Se optimiza la carga de recursos como imágenes y efectos visuales (p. ej., evitar la carga innecesaria de imágenes)?		

Seguridad		
¿Los enlaces y redirecciones (Link de Next.js) están bien gestionados para prevenir comportamientos inesperados?		
¿Las entradas del usuario están validadas correctamente para evitar ataques de inyección de datos?		
Documentación		
¿Cada componente tiene comentarios que describen su propósito y el uso de props?		
¿Se documentan adecuadamente los componentes, interfaces y funciones para facilitar su comprensión por otros desarrolladores?		
¿Se documenta correctamente el uso de APIs externas o servicios?		
Mantenibilidad		
¿Los componentes son lo suficientemente pequeños y enfocados en una tarea específica, evitando componentes demasiado grandes o complejos?		
¿Los componentes son reutilizables y desacoplados de otros módulos específicos?		
Portabilidad		
¿Se utiliza una combinación de tecnologías que facilita la portabilidad (React, Next.js, TypeScript)?		
¿Se limita el uso de dependencias externas no esenciales para reducir problemas de		

compatibilidad en diferentes entornos?		
Cumplimiento de normativas		
¿El código incluye buenas prácticas para accesibilidad (p. ej., etiquetas “alt” descriptivas, soporte para navegadores con lector de pantalla)?		
¿El código sigue las normativas de seguridad, como el uso de HTTPS y la gestión segura de tokens y sesiones?		