Justificación del uso de los patrones MVC y Repositorio en la API de Node.js con SQL Server y Stored Procedures

Nota: pude haber usado un ORM, pero decidí no hacerlo para demostrar conocimiento en SQL

PATRONES USADOS MVC Y Patrón Repositorio

Introducción

La implementación de una API utilizando patrones de diseño es una práctica común en el desarrollo de software que tiene como objetivo mejorar la estructura, mantenibilidad, escalabilidad y reutilización del código. En este documento, se justificará el uso de los patrones Modelo-Vista-Controlador (MVC) y Repositorio en una API de Node.js que interactúa con una base de datos SQL Server utilizando Stored Procedures.

1. Patrón Modelo-Vista-Controlador (MVC)

Justificación:

Separación de responsabilidades: El patrón MVC divide la lógica de la aplicación en tres componentes principales: Modelo, Vista y Controlador. Esto permite una clara separación de responsabilidades, lo que facilita el mantenimiento y extensibilidad del código.

Reutilización de código: La separación de la lógica de la aplicación en capas permite reutilizar componentes de manera más efectiva. Por ejemplo, si se necesita cambiar la interfaz de usuario (Vista), la lógica de negocio (Modelo) se mantendrá intacta.

Escalabilidad: El patrón MVC facilita la escalabilidad al permitir que cada componente se enfoque en una tarea específica. Esto permite agregar nuevas funcionalidades o realizar cambios sin afectar otras partes del sistema.

Facilita el testing: El MVC promueve el desarrollo de pruebas unitarias y de integración más eficaces al separar la lógica de negocio del código de presentación. Esto permite probar cada componente de manera independiente.

2. Patrón Repositorio

Justificación:

Abstracción de la capa de datos: El patrón Repositorio permite abstraer el acceso a los datos y oculta los detalles de la implementación subyacente, como SQL Server y Stored Procedures. Esto facilita futuros cambios en la base de datos o tecnologías de almacenamiento sin afectar el resto de la aplicación.

Mantenibilidad y extensibilidad: El Repositorio proporciona una capa de abstracción entre la lógica de negocio y el acceso a los datos. Esto hace que el código de negocio sea independiente de la tecnología de almacenamiento utilizada, lo que facilita el mantenimiento y permite cambiar la fuente de datos sin afectar el resto del sistema.

Testing: Al separar la lógica de negocio del acceso a datos, es más fácil realizar pruebas unitarias y de integración, ya que se pueden crear implementaciones de repositorio falsas o en memoria para simular la interacción con la base de datos durante las pruebas.

Seguridad: El uso de un Repositorio ayuda a prevenir vulnerabilidades comunes, como inyecciones SQL, ya que se utilizan parámetros y procedimientos almacenados para interactuar con la base de datos.