**Resumen de la Arquitectura de la API**

La **Employee Management API** está diseñada siguiendo una arquitectura en capas que separa las responsabilidades en distintas capas, lo que mejora la mantenibilidad, escalabilidad y testabilidad del sistema.

**Capas Principales:**

1. **Controladores (Controllers):**
   * Los controladores actúan como la capa de presentación y son responsables de manejar las solicitudes HTTP, coordinar la lógica de negocio y devolver las respuestas adecuadas. Cada endpoint en la API está definido en un controlador específico, como EmployeesController, que maneja las operaciones CRUD y otros métodos específicos de los empleados.
   * **Decisión Técnica:** Elegí separar la lógica de presentación en controladores para mantener una clara separación de responsabilidades y facilitar la gestión de las rutas y acciones HTTP.
2. **Servicios (Services):**
   * La capa de servicios encapsula la lógica de negocio de la aplicación. Aquí se implementan las reglas de negocio, como la validación de datos y las operaciones que involucran múltiples pasos. Los servicios interactúan con los repositorios para acceder a los datos y son inyectados en los controladores.
   * **Decisión Técnica:** La introducción de una capa de servicios permite que la lógica de negocio sea fácilmente testeable y reutilizable, al tiempo que mantiene los controladores livianos y enfocados en la orquestación de las solicitudes HTTP.
3. **Repositorios (Repositories):**
   * La capa de repositorios es responsable de la comunicación directa con la base de datos. Se utiliza un patrón de repositorio para abstraer el acceso a los datos, lo que permite cambiar la implementación de acceso a la base de datos sin afectar el resto de la aplicación.
   * **Decisión Técnica:** Utilizar un patrón de repositorio facilita el manejo de la persistencia de datos y permite un acceso más estructurado y organizado a la base de datos. Esto es especialmente útil al trabajar con procedimientos almacenados, ya que permite encapsular la lógica de acceso a datos.
4. **Modelos (Models):**
   * Los modelos representan las entidades de datos, como Employee, y son utilizados en toda la aplicación para intercambiar datos entre las diferentes capas. Estos modelos también se utilizan para mapear los datos de las tablas de la base de datos.
   * **Decisión Técnica:** Al definir modelos claros y coherentes, la API mantiene una estructura de datos consistente, lo que facilita el mantenimiento y la comprensión del código.

**Acceso a Datos y Procedimientos Almacenados**

* **Entity Framework Core y Procedimientos Almacenados:**
  + Aunque Entity Framework Core se utiliza para manejar la mayor parte del acceso a datos y mapear las entidades del dominio a las tablas de la base de datos, se decidió implementar procedimientos almacenados para todas las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) relacionadas con los empleados. Esta combinación permite aprovechar la simplicidad y flexibilidad de Entity Framework Core para las operaciones generales mientras se garantiza un control detallado y optimización en las interacciones con la base de datos mediante los procedimientos almacenados.
  + **Decisión Técnica:** Al utilizar procedimientos almacenados para todas las operaciones CRUD, se mejora el rendimiento y la seguridad, asegurando que las operaciones en la base de datos sean eficientes y cumplan con las reglas de negocio definidas. Entity Framework Core sigue desempeñando un papel clave en la gestión de la conexión a la base de datos y en la abstracción del modelo de datos, lo que facilita el desarrollo y el mantenimiento del código.

Este enfoque proporciona lo mejor de ambos mundos: la facilidad de uso de Entity Framework Core para el desarrollo ágil y la potencia de los procedimientos almacenados para un control más preciso y optimizado de las operaciones de base de datos.

**Pruebas Unitarias**

* **xUnit y Moq:**
  + Las pruebas unitarias se implementan utilizando xUnit y Moq para asegurar que la lógica de negocio y las interacciones entre las capas funcionen como se espera. Las pruebas se enfocan en los servicios y controladores, garantizando que las principales funcionalidades de la API se comporten correctamente bajo diferentes escenarios.
  + **Decisión Técnica:** Al usar xUnit y Moq, se facilita la escritura de pruebas unitarias efectivas, promoviendo la calidad del código y permitiendo futuras modificaciones con menor riesgo.

**Conclusión**

La arquitectura de la API fue diseñada para ser modular, escalable y fácil de mantener. La separación en capas, el uso de patrones de diseño como repositorios y la implementación de pruebas unitarias permiten un desarrollo ágil y la posibilidad de escalar o modificar la aplicación con mayor facilidad en el futuro. Estas decisiones técnicas garantizan que la API no solo sea funcional, sino también robusta y preparada para cambios a largo plazo.