Utilizar primero una interfaz en el paquete de servicios (service) y luego llamar a una implementación concreta es una práctica común en el diseño de aplicaciones basadas en microservicios y sigue el principio de inversión de dependencias (Dependency Inversion Principle) y el principio de segregación de interfaces (Interface Segregation Principle), ambos principios del diseño de software.

Aquí hay algunas razones por las que es beneficioso seguir este enfoque:

1. **Desacoplamiento:** Al definir una interfaz en el paquete de servicios y luego llamar a una implementación concreta, estás desacoplando tu código de la implementación real. Esto significa que el código que llama al servicio no necesita conocer los detalles internos de cómo se implementa el servicio, lo que facilita el cambio y la evolución del sistema con el tiempo.
2. **Flexibilidad:** Al utilizar una interfaz, tienes la flexibilidad de cambiar la implementación subyacente del servicio en el futuro sin modificar el código que llama al servicio. Esto es útil si necesitas cambiar la implementación del servicio debido a cambios en los requisitos, mejoras de rendimiento, etc.
3. **Pruebas unitarias:** Utilizar interfaces facilita la escritura de pruebas unitarias, ya que puedes crear implementaciones de prueba para las interfaces y utilizarlas en lugar de las implementaciones reales durante las pruebas. Esto te permite simular el comportamiento del servicio en un entorno controlado y probar el código de manera más efectiva.
4. **Separación de responsabilidades:** Al definir interfaces claras en el paquete de servicios, estás separando claramente las responsabilidades de tu código. Las interfaces definen el contrato entre el código que llama al servicio y la implementación real del servicio, lo que hace que tu código sea más modular y mantenible.

En resumen, utilizar interfaces en el paquete de servicios y luego llamar a una implementación concreta proporciona beneficios como desacoplamiento, flexibilidad, facilita las pruebas unitarias y promueve una mejor separación de responsabilidades en tu código. Esto es especialmente importante en arquitecturas basadas en microservicios, donde la modularidad y la flexibilidad son fundamentales.