

Tarea Semana 6

Rene Euceda 41251020

Facultad de Ing. En informática, Centro Universitario Tecnológico | Universidad Ceutec

233 INGENIERÍA DE SOFTWARE II 2025Q4

Ing. María Salinas

26 de noviembre de 2025

Resumen Técnico – Ciclo de Vida de Servicios en .NET Core

1. Definiciones claras

- **Transient (AddTransient):** Se crea una nueva instancia cada vez que se solicita. Ideal para servicios livianos y sin estado.
- **Scoped (AddScoped):** Se crea una instancia por cada request HTTP. Garantiza consistencia de datos dentro de un mismo flujo.
- **Singleton (AddSingleton):** Se crea una única instancia durante toda la vida de la aplicación. Adecuado para servicios globales y reutilizables.

2. Cuándo usar cada lifetime

- **Transient:** Para servicios livianos/stateless como validadores, formateadores o conversores.
Justificación: Bajo costo de creación y no requieren mantener estado.
- **Scoped:** Para servicios que deben compartir datos dentro de un request, como DbContext, repositorios o unidades de trabajo.
Justificación: Garantiza consistencia y evita conflictos de concurrencia en un mismo flujo.
- **Singleton:** Para servicios globales como caches de configuración, clientes thread-safe (HttpClient) o proveedores de logging.
Justificación: Evita recreación costosa y mantiene estado compartido en toda la aplicación.

3. Implicaciones de concurrencia y recursos

- **Singleton:** Debe ser *thread-safe*, ya que múltiples hilos acceden a la misma instancia. Mal diseño puede causar condiciones de carrera.
- **Scoped:** Riesgo de uso indebido fuera del request; puede provocar fugas de memoria o excepciones si se comparte entre hilos.
- **Transient:** Cada solicitud crea una nueva instancia. Seguro en concurrencia, pero abusar de servicios pesados puede generar sobrecarga de recursos.

4. Ciclo de vida y disposición (IDisposable)

- El contenedor de DI maneja automáticamente la disposición de objetos que implementan IDisposable.
- **Transient y Scoped:** Se liberan al finalizar el request o scope.
- **Singleton:** Se libera al finalizar la aplicación.
- Importante: no forzar disposiciones manuales, ya que el contenedor controla el ciclo de vida.

5. Antipatrones y errores comunes

- Usar **DbContext como Singleton:** provoca inconsistencias y problemas de concurrencia.
- Abusar de **Transient en servicios pesados:** genera sobrecarga de CPU y memoria.
- Usar **Scoped fuera de su contexto (ej. en Singletons):** causa excepciones y fugas de memoria.
- No considerar *thread-safety* en Singletons: riesgo de corrupción de datos.