DESARROLLO DE SOFTWARE FRONTEND III

MARTINEZ COSIO JOSE ALFREDO

Aarón Hernández García

3.1 Investigación sobre servicios e inyección de dependencias

01/06/2025

1. ¿Qué es la Inyección de Dependencias?

La **inyección de dependencias** es un patrón de diseño utilizado en la programación orientada a objetos donde un objeto no se encarga de crear sus propias dependencias, sino que se le entregan (o "inyectan") desde el exterior. Es decir, en lugar de que una clase cree sus propios recursos (como servicios, repositorios o utilidades), alguien más (generalmente un contenedor o framework) se los proporciona.

2. ¿Por qué utilizar DI?

Usar DI tiene varias ventajas:

- **Desacoplamiento:** Las clases no dependen directamente de otras clases específicas, sino de interfaces o abstracciones. Esto facilita los cambios y el mantenimiento.
- **Facilidad de pruebas:** Al poder inyectar dependencias, es fácil sustituirlas por versiones falsas o simuladas (mocks) en pruebas unitarias.
- **Reutilización de código:** Las dependencias pueden ser compartidas o reutilizadas entre múltiples clases.
- **Mayor legibilidad y organización del código:** Se separan claramente las responsabilidades y se mantiene un código más limpio.

3. Ejemplo de uso de DI en un proyecto

Proyecto: API de productos para una tienda en línea.

Funcionalidad: Un controlador de productos necesita acceder a un servicio que obtiene los productos desde una base de datos.

¿Cómo ayuda la DI?:

- En lugar de crear el servicio directamente dentro del controlador (new ProductService()), se inyecta como una dependencia.
- Esto permite cambiar la implementación del servicio (por ejemplo, una versión simulada en pruebas) sin modificar el controlador.
- También facilita escalar el proyecto a largo plazo y mantenerlo más limpio y organizado.

4. Práctica: Proyecto simple usando DI (en C# / ASP.NET Core)

Objetivo: Crear una API que devuelve un saludo personalizado usando un servicio inyectado.

Estructura del proyecto:

- IGreetingService (interfaz)
- GreetingService (implementación)
- GreetingController (controlador)

```
// IGreetingService.cs
public interface IGreetingService
{
    string GetGreeting(string name);
}
```

```
// GreetingService.cs
public class GreetingService : IGreetingService
{
    public string GetGreeting(string name)
    {
        return $"Hola, {name}! Bienvenido al sistema.";
    }
}
```

```
// GreetingController.cs
[ApiController]
[Route("[controller]")]
public class GreetingController : ControllerBase
{
    private readonly IGreetingService _greetingService;

    public GreetingController(IGreetingService greetingService)
    {
        _greetingService = greetingService;
    }

    [HttpGet("{name}")]
    public IActionResult GetGreeting(string name)
    {
        var message = _greetingService.GetGreeting(name);
        return Ok(message);
    }
}
```

Resultado esperado:

Al acceder a /Greeting/Juan, se obtendrá:

```
"Hola, Juan! Bienvenido al sistema."
```