# ASP.NET CORE CON C#

## 7. SERVICIOS

En el último ejercicio de la Práctica_1 en el que pasamos un objeto persona y una lista de proyectos a una vista, ambos objetos los creamos en el controlador. Esta no es una buena opción, los controladores deben dedicarse únicamente a renderizar vistas (**Principio de responsabilidad única),** y en todo caso a pedir datos a otras clases. Lo más lógico es que esos datos deban ser accesibles desde otros controladores u otras clases y por tanto no deben ser "propiedad" de un controlador. No parece que ese sea el sitio para "almacenar". Los datos los almacenamos en una clase (servicio)

**Ejemplo**: Pasando listas con servicios

1. Declaramos una clase en la carpeta servicios que sea el repositorio de mis datos, ésto nos permite quitar ese código del controlador y sólo debemos instanciar una variable de ese tipo

2. Nuestro controlador

```
0 referencias
public IActionResult Index()
{
    RepositorioPersonas r=new RepositorioPersonas();
    HomeIndexViewModel m = new HomeIndexViewModel
    {
        Listado = r.getPersonas()
    };
    return View(m);
}
```

**Inyeccion de dependencias**

Instanciar los objetos que tendremos que pasar a las vistas en un controlador no es la mejor manera de trabajar. Si esos objetos a su vez contienen a otros tendremos que ir haciendo instancias de instancias…. Por otro lado, puede ocurrir que unas veces el proveedor de datos sea una base de datos y otras veces los datos los obtenemos de otro "almacén". Debemos intentar que ni las clases ni los controladores haya que cambiarlos en función de ésto. Para trabajar de forma más automáticas usaremos **la inyección de dependencias con interfaces.**
Veamos primero qué es la inyección y luego veremos como automatizar con interfaces.

**Ejemplo**: Pasando listas con inyección

No queremos que el controlador instancias los objetos de los que depende, queremos inyectar una variable en su constructor y a través de ella utilizamos ese objeto. ¿Dónde se instancia entonces ese objeto?.

3. En el archivo Program.css

```
using PasandoListasDependencias.servicios;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddTransient<RepositorioPersonas>();
```

Esto significa que cada vez que yo inyecte una variable de este tipo en el constructor de una clase, automáticamente se instancia la misma y se genera el objeto.

```
1 referencia
public class HomeController : Controller
{
    private readonly RepositorioPersonas r;
    0 referencias
    public HomeController(RepositorioPersonas r)
    {
        this.r = r;
    }
```

```
0 referencias
public IActionResult Index()
{
    //RepositorioPersonas r=new RepositorioPersonas();
    HomeIndexViewModel m = new HomeIndexViewModel
    {
        Listado = r.getPersonas()
    };
    return View(m);
```

Podemos mejorar ésto añadiendo interfaces

**Ejemplo**: Pasando listas con interfaces

4. Defino una interfaz

```
using PasandoListasDependencias.Models;
namespace PasandoListasDependencias.Interfaces
{
    2 referencias
    public interface IRepositorioAlumnos
    {
        1 referencia
        List<Alumno> GetAlumnos();
    }
}
```

5. La clase servicio tiene que implementar esta interfaz y definir ese método. Por convenio el método debe empezar por mayúsculas

```
4 referencias
public class RepositorioAlumnos : IRepositorioAlumnos
{

    1 referencia
    public List<Alumno> GetAlumnos()
   // public List<Alumno> getAlumnos()
    {
        return new List<Alumno>
        {
            new Alumno
            {
                Nombre="Pepe Pérea",
                DNI="11111111A",
                Telefono="677878788"
```

6. Por último inyectamos en Program.cs

```
builder.Services.AddControllersWithViews();
builder.Services.AddTransient<RepositorioAlumnos>();
builder.Services.AddTransient<IRepositorioAlumnos, RepositorioAlumnos>();
```

Esto significa que cuando una clase pida un IRepositorioAlumnos se le envía una instancia de tipo RepositorioAlumnos.

En nuestro controlador sólo cambia el tipo del atributo

```
1 referencia
public class HomeController : Controller
{
    private readonly IRepositorioAlumnos r;
    0 referencias
    public HomeController(IRepositorioAlumnos repositorio)
    {
        this.r = repositorio;
    }
```

Parece poca cosa pero lo cierto es que tenemos mucha flexibilidad. Mi controlador realmente no sabe de que tipo estamos instanciando el objeto, él sólo recibe datos de tipo IRepositorioAlumno. Es decir, podría recibir una instancia de cualquier clase que implemente esta interfaz ( siempre que hayamos añadido la dependencia en Program.cs). Esto nos abre muchas posibilidades, por ejemplo recibir datos desde una lista, un json, una base de datos…… y utilizar la dependencia que me interese en cada momento.
A esto se le llama **Principio de inversión de dependencias**, nuestros controladores no dependen de otras clases sino de tipos abstractos.

Los servicios son clases o interfaces que tenemos registradas en el sistemas de inyección de dependencias. Es decir, podemos utilizarlas inyectando una variable en el constructor de un controlador.
No todos los servicios son iguales, en función de su tiempo de vida los tenemos de tres tipos:

- Transitorios (transient): Los que menos viven, cada vez que se inyecta en una clase se genera una nueva instancia. **Siempre diferentes**

- Delimitados(scope): su ciclo de vida está delimitado por una solicitud http. Dentro de una misma solicitud no se genera una nueva instancia cada vez que se inyecta

en un contructor pero si lo hace con cada petición htttp diferente.

- Únicos(singleton): es el que más vive, sólo se renueva si la aplicación se reinicia. Siempre es el mismo cuándo se inyecta incluso en diferentes peticiones http. **Siempre los mismos.**

El método que usaremos para añadir la dependencia en Program.cs dependerá del tipo de servicio que queramos usaremos

builder.Services.AddTransient, builder.Services.AddScoped, builder.Services.AddSingleton