## Patrón de Diseño DTO (Data Transfer Object)

## Taller Práctico: DTO usando AutoMapper

## Objetivo del ejemplo

Este taller tiene como objetivo enseñar cómo usar AutoMapper en un proyecto para mapear entidades a objetos DTO (Data Transfer Object). En este ejemplo mapeamos colecciones.

## Estructura del proyecto

El proyecto está organizado en los siguientes espacios de nombres (namespaces):

* 01\_Dto.Models – contiene la clase de dominio Empleado
* 01\_Dto.DTOs – contiene el objeto de transferencia EmpleadoDTO
* 01\_Dto.Profiler – contiene el perfil de mapeo para AutoMapper
* 01\_Dto.Controllers – contiene el controlador EmpleadoController
* Program.cs – archivo de configuración principal de la aplicación

1. **Definición del modelo de dominio (Empleado.cs)**

namespace \_01\_Dto.Models  
{  
 public class Empleado  
 {  
 public int Id { get; set; }  
 public string FullName { get; set; } = string.Empty;  
 public string Role { get; set; } = string.Empty;  
 public decimal Salary { get; set; }  
 }  
}

1. **Definición del DTO (EmpleadoDTO.cs)**

namespace \_01\_Dto.DTOs  
{  
 public class EmpleadoDTO  
 {  
 public int Id { get; set; }  
 public string FullName { get; set; } = string.Empty;  
 public string Cargo { get; set; } = string.Empty;  
 }  
}

1. **Configuración del perfil de AutoMapper (EmpleadoProfile.cs)**

using \_01\_Dto.DTOs;  
using \_01\_Dto.Models;  
using AutoMapper;  
  
namespace \_01\_Dto.Profiler  
{  
 public class EmpleadoProfile : Profile  
 {  
 public EmpleadoProfile()  
 {  
 CreateMap<Empleado, EmpleadoDTO>()  
 .ForMember(dest => dest.Cargo, opt => opt.MapFrom(src => src.Role));  
 }  
 }  
}

1. **Controlador con el endpoint API (EmpleadoController.cs)**

using \_01\_Dto.DTOs;  
using \_01\_Dto.Models;  
using AutoMapper;  
using Microsoft.AspNetCore.Mvc;  
  
namespace \_01\_Dto.Controllers  
{  
 [ApiController]  
 [Route("api/[controller]")]  
 public class EmpleadoController : ControllerBase  
 {  
 private readonly IMapper \_mapper;  
  
 public EmpleadoController(IMapper mapper)  
 {  
 \_mapper = mapper;  
 }  
  
 [HttpGet]  
 public ActionResult<List<EmpleadoDTO>> ObtenerEmpleado()  
 {  
 var empleados = new List<Empleado>  
 {  
 new Empleado { Id = 1, FullName = "Carlos Pérez", Role = "Gerente de Proyectos", Salary = 9500000 },  
 new Empleado { Id = 2, FullName = "Aníbal Martínez", Role = "CEO", Salary = 35000000 },  
 new Empleado { Id = 3, FullName = "José Lopez", Role = "Ing. de Mantenimiento", Salary = 8500000 }  
 };  
  
 var listaDto = \_mapper.Map<List<EmpleadoDTO>>(empleados);  
 return Ok(listaDto);  
 }  
 }  
}

1. **Configuración del proyecto (Program.cs)**

using \_01\_Dto.Profiler;  
  
var builder = WebApplication.CreateBuilder(args);  
  
// Registro de AutoMapper  
builder.Services.AddAutoMapper(typeof(EmpleadoProfile));  
  
// Agregamos los controladores  
builder.Services.AddControllers();  
  
var app = builder.Build();  
  
app.MapControllers();  
app.Run();

## Resultado esperado

Al ejecutar el proyecto y acceder al endpoint:

GET http://localhost:{puerto}/api/empleado

Se debe obtener una respuesta JSON como:

[  
 { "id": 1, "fullName": "Carlos Pérez", "cargo": "Gerente de Proyectos" },  
 { "id": 2, "fullName": "Aníbal Martínez", "cargo": "CEO" },  
 { "id": 3, "fullName": "José Lopez", "cargo": "Ing. de Mantenimiento" }  
]

## Conclusión y aprendizajes

Este taller permite comprender:

* La separación entre entidades de dominio y DTOs.
* Cómo ocultar o renombrar propiedades sensibles o irrelevantes para el consumidor de una API.
* Cómo usar AutoMapper para automatizar ese proceso.