**PATRON REPOSITORY**

Es un patrón que **abstrae la lógica de acceso a datos** y centraliza todas las operaciones sobre una entidad específica en una clase o interfaz. Su objetivo es separar la **lógica del dominio** de la **lógica de persistencia.**

**Objetivo general del taller.**

Diseñar y construir una API RESTful en ASP.NET Core (.NET 8) aplicando el patrón Repositorio, utilizando almacenamiento simulado en memoria, como base para comprender los principios fundamentales del desarrollo de servicios web y la separación de responsabilidades en aplicaciones empresariales.

**Objetivos específicos.**

* Crear un proyecto Web API en Visual Studio 2022 utilizando .NET 8, configurando correctamente el entorno de desarrollo.
* Implementar el patrón Repositorio, separando la lógica de acceso a datos de la lógica de presentación (controladores).
* Simular la persistencia de datos mediante una estructura en memoria, permitiendo pruebas y comprensión del flujo de datos sin usar una base de datos real.
* Construir endpoints RESTful para operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre la entidad Empleado, aplicando buenas prácticas en el uso de controladores.
* Inyectar dependencias usando el contenedor de servicios de ASP.NET Core, aprendiendo a registrar e instanciar clases de forma controlada.
* Probar y validar el funcionamiento de los endpoints usando Swagger, aprovechando las herramientas integradas de .NET para documentación y pruebas.

**Ventajas:**

* Facilita el mantenimiento y pruebas (por ejemplo, se puede hacer mock del repositorio).
* Promueve la **separación de responsabilidades.**
* Encapsula la lógica de acceso a datos.
* Permite cambiar la tecnología de persistencia sin afectar el resto de la aplicación.

**Crear una Web API RESTful en .NET 8, con:**

* Patrón Repositorio
* Datos en memoria (sin base de datos)
* Sin DTOs, sin MediatR, sin UnitOfWork
* Estilo tradicional con controladores
* Estructura final:

Estructura de directorios:

EmpleadoApi/

│

├── Controllers/

│ └── EmpleadoController.cs

├── Models/

│ └── Empleado.cs

├── Repositories/

│ ├── IEmpleadoRepository.cs

│ └── EmpleadoRepository.cs

├── Program.cs

1. Crear un Proyecto en Visual Studio 2022 y .Net 8.

* Abre Visual Studio 2022.
* Selecciona "Crear un nuevo proyecto".
* Elige “ASP.NET Core Web API” y haz clic en Siguiente.
* Nombre: EmpleadoApi
* Ubicación: donde quieras guardarlo
* Click en Siguiente

1. Configuración del proyecto

* Framework: Elige .NET 8

Marca solo la opción:

* "Utilizar controladores"
* "Habilitar compatibilidad con OpenAPI (Swagger)"

1. Desmarcar

* "Usar autenticación"
* "Habilitar HTTPS (opcional)"

1. Haz clic en Crear
2. Crear la carpeta Models y la clase Empleado

Crea una carpeta Models en la raíz del proyecto y dentro de esta crea la clase Empleado.

namespace EmpleadoApi.Models;

public class Empleado

{

public int Id { get; set; }

public string Nombre { get; set; }

public string Cargo { get; set; }

public decimal Salario { get; set; }

}

1. Crear la interfaz del repositorio

Crea una carpeta Repositories en la raíz del proyecto y dentro de esta crea las Interfaz.

using EmpleadoApi.Models;

namespace EmpleadoApi.Repositories;

public interface IEmpleadoRepository

{

IEnumerable<Empleado> GetAll();

Empleado? GetById(int id);

void Add(Empleado empleado);

void Update(Empleado empleado);

void Delete(int id);

}

1. Implementar la interfaz Repositorio

En la carpeta Repositories crea una nueva clase.

using EmpleadoApi.Models;

namespace EmpleadoApi.Repositories;

public class EmpleadoRepository : IEmpleadoRepository

{

private readonly List<Empleado> \_empleados = new();

private int \_nextId = 1;

public IEnumerable<Empleado> GetAll() => \_empleados;

public Empleado? GetById(int id) => \_empleados.FirstOrDefault(e => e.Id == id);

public void Add(Empleado empleado)

{

empleado.Id = \_nextId++;

\_empleados.Add(empleado);

}

public void Update(Empleado empleado)

{

var existente = GetById(empleado.Id);

if (existente is null) return;

existente.Nombre = empleado.Nombre;

existente.Cargo = empleado.Cargo;

existente.Salario = empleado.Salario;

}

public void Delete(int id)

{

var empleado = GetById(id);

if (empleado is not null)

{

\_empleados.Remove(empleado);

}

}

}

1. Registrar el repositorio

En la clase Program se deben registrar las dependencias.

using EmpleadoApi.Repositories;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

**// Registro del repositorio como singleton**

**builder.Services.AddSingleton<IEmpleadoRepository, EmpleadoRepository>();**

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseAuthorization();

app.MapControllers();

app.Run();

1. Crear la carpeta Controllers y el controlador

Crea una carpeta Controllers en la raíz del proyecto y dentro de esta crea el controlador.

using Microsoft.AspNetCore.Mvc;

using EmpleadoApi.Models;

using EmpleadoApi.Repositories;

namespace EmpleadoApi.Controllers;

[ApiController]

[Route("api/[controller]")]

public class EmpleadoController : ControllerBase

{

private readonly IEmpleadoRepository \_repo;

public EmpleadoController(IEmpleadoRepository repo)

{

\_repo = repo;

}

[HttpGet]

public ActionResult<IEnumerable<Empleado>> GetAll()

{

return Ok(\_repo.GetAll());

}

[HttpGet("{id}")]

public ActionResult<Empleado> GetById(int id)

{

var empleado = \_repo.GetById(id);

if (empleado == null) return NotFound();

return Ok(empleado);

}

[HttpPost]

public ActionResult<Empleado> Create(Empleado empleado)

{

\_repo.Add(empleado);

return CreatedAtAction(nameof(GetById), new { id = empleado.Id }, empleado);

}

[HttpPut("{id}")]

public IActionResult Update(int id, Empleado empleado)

{

if (id != empleado.Id) return BadRequest();

var existente = \_repo.GetById(id);

if (existente is null) return NotFound();

\_repo.Update(empleado);

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var existente = \_repo.GetById(id);

if (existente is null) return NotFound();

\_repo.Delete(id);

return NoContent();

}

}

1. Probar la API

Ejecuta el proyecto. Se abrirá **Swagger, o usa Postman**

Puedes probar directamente en Swagger los endpoints como:

GET /api/empleado

POST /api/empleado

PUT /api/empleado/{id}

DELETE /api/empleado/{id}

**¿Qué aprendimos?**

* Crear un proyecto Web API desde Visual Studio 2022 con .NET 8
* Aplicar el Patrón Repositorio
* Registrar dependencias con inyección de dependencias
* Crear un controlador RESTful básico

**Nota aclaratoria.**

### CreatedAtAction(...)

Esto es un helper de ASP.NET Core que **devuelve un 201 Created** y genera automáticamente la URL del nuevo recurso.

CreatedAtAction(nameof(GetById), new { id = empleado.Id }, empleado)

* nameof(GetById) → es el nombre del método que devuelve un empleado por id (típicamente GetById(int id))
* new { id = empleado.Id } → define el valor de ruta necesario para generar el link (/api/empleado/{id})
* empleado → se devuelve el objeto recién creado como cuerpo de la respuesta.