# Repository Pattern in ASP.NET Core Web API

## 📌 What is the Repository Pattern?

The Repository Pattern is a design pattern used to separate the data access logic from the business logic. It provides an abstraction layer between the data source (database) and the application.  
  
✅ Centralizes data access logic in a single place.  
✅ Reduces code duplication.  
✅ Makes code easier to maintain and test.  
✅ Supports the \*\*Dependency Injection\*\* principle.

## 📌 Why is the Repository Pattern Important?

Without the Repository Pattern, business logic and data access logic are tightly coupled, making it hard to maintain and test.  
  
The Repository Pattern helps in:  
✅ \*\*Separation of Concerns\*\*: Keeps business logic separate from data access.  
✅ \*\*Improving Testability\*\*: Enables unit testing by allowing mock implementations.  
✅ \*\*Flexibility\*\*: Easily switch between different data sources (e.g., SQL Server, MongoDB).  
✅ \*\*Code Maintainability\*\*: Reduces code duplication and keeps data access in one place.

### 📝 Example Without Repository Pattern

Create new Webapi project 🡺 Ef\_Code\_First\_API

Right click on project in solution Explorer 🡺Add New Folder 🡺Add new Class as Product.cs

Install 3 packages from Nuget package solution

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

**Product.cs**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

Add Another Folder 🡺Data🡺Add new class (AppDbContext.cs)

public class AppDbContext:DbContext

{

public AppDbContext(DbContextOptions<AppDbContext> options):base(options)

{

}

public DbSet<Product> Products { get;set; }

}

Appsettings.json

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"myconnection": "Server=(localdb)\\MSSQLLocalDB;Database=cmrdb;Integrated Security=true;TrustServerCertificate=True;"

}

}

Open Nuget package console 🡺

Add-migration “table creation”

Update-databse

🡺 will add tables in your databse

Check your sqlserver 🡺 Database 🡺 Tables🡺 Product and MigrationHistory table will be available or not

Create New Folder 🡺Repositories 🡺

Add new Interface IProductRepository.cs

public interface IProductRepository

{

Task<IEnumerable<Product>> GetProductsAsync();

Task<Product> GetProductByIdAsync(int id);

Task<Product> AddProductAsync(Product product);

Task<Product> UpdateProductAsync(int id,Product product);

Task<string> DeleteProductAsync(int id);

}

Add New class in Repositories Folder ProductRepository.cs

public class ProductRepository:IProductRepository

{

private readonly AppDbContext \_context;

public ProductRepository(AppDbContext context)

{

\_context = context;

}

public async Task<IEnumerable<Product>> GetProductsAsync()

{

return await \_context.Products.ToListAsync();

}

public async Task<Product> GetProductByIdAsync(int id)

{

return await \_context.Products.FindAsync(id);

}

public async Task<Product> AddProductAsync(Product product)

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

return product;

}

public async Task<Product> UpdateProductAsync(int id, Product product)

{

var updateProduct = await \_context.Products.FindAsync(id);

if (updateProduct == null)

{

return null; // Return null if product not found

}

updateProduct.Name = product.Name;

updateProduct.Price = product.Price;

await \_context.SaveChangesAsync();

return updateProduct; // Return updated product

}

public async Task<string> DeleteProductAsync(int id)

{

var prodcut = await \_context.Products.FindAsync(id);

if (prodcut != null)

{

\_context.Remove(prodcut);

await \_context.SaveChangesAsync();

return "Product Removed Successfully";

}

else

return "Product not found";

}

}

}

**Program.cs**

using EF\_Code\_first\_Demo1.Repositories;

using Microsoft.Data.SqlClient;

using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddDbContext<AppDbContext>(

options => options.UseSqlServer(builder.Configuration.GetConnectionString("myconnection")));

builder.Services.AddScoped<IProductRepository, ProductRepository>();

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Right click on Controllers folder 🡺Add new Empty Webapi Controller 🡺ProductsController**

using EF\_Code\_first\_Demo1.Data;

using EF\_Code\_first\_Demo1.Models;

using EF\_Code\_first\_Demo1.Repositories;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

namespace EF\_Code\_first\_Demo1.Controllers

{

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

[ApiController]

public class ProductsController : ControllerBase

{

private readonly IProductRepository \_productRepository;

public ProductsController(IProductRepository repo)

{

\_productRepository = repo;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Product>>> GetAllProducts()

{

return Ok(await \_productRepository.GetProductsAsync());

}

[HttpGet("{id}")]

public async Task<ActionResult<Product>> GetProduct(int id)

{

var product= await \_productRepository.GetProductByIdAsync(id);

if(product == null)

{

return NotFound();

}

return product;

}

[HttpPost]

public async Task<ActionResult<Product>> AddProduct(Product product)

{

var newProduct = await \_productRepository.AddProductAsync(product);

return CreatedAtAction(nameof(GetProduct), new { id = newProduct.Id }, newProduct);

}

[HttpPut("{id}")]

public async Task<ActionResult> UpdateProduct(int id, [FromBody] Product updatedProduct)

{

var product = await \_productRepository.UpdateProductAsync(id, updatedProduct);

if (product == null)

{

return NotFound("Product not found"); // Return 404 Not Found

}

return NoContent(); // Return 204 No Content if successful

}

[HttpDelete("{id}")]

public async Task<ActionResult> DeleteProduct(int id)

{

string result = await \_productRepository.DeleteProductAsync(id);

if (result == "Product not found")

{

return NotFound(result); // Return 404 Not Found if product does not exist

}

return Ok(result); // Return 200 OK with success message

}

}

}