Repository Design Pattern

The Repository Design Pattern is a structural pattern that mediates data access by providing an abstraction over the data layer. It allows you to decouple the data access logic and business logic by encapsulating the data access logic in a separate repository class. This pattern provides a collection-like interface for accessing domain objects.

### **Pros**

1. **Decoupling:**Separates the data access logic from business logic, promoting cleaner code organization and separation of concerns.
2. **Testability:**Makes the application easier to unit test by allowing mocking of the repository.
3. **Consistency:** Centralizes data access logic, ensuring consistency across different parts of the application.
4. **Flexibility:**Makes it easier to switch out data access technology (e.g., from Entity Framework to another ORM) without changing the business logic.

## mplementation in .NET Core

To implement the Repository Pattern in .NET Core, you need to follow these steps.

### **1. Create a Model**

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

C#

Copy

### **2. Create an Interface for the Repository**

public interface IProductRepository

{

IEnumerable<Product> GetAll();

Product GetById(int id);

void Add(Product product);

void Update(Product product);

void Delete(int id);

}

C#

Copy

### **3. Implement the Repository**

public class ProductRepository : IProductRepository

{

private readonly AppDbContext \_context;

public ProductRepository(AppDbContext context)

{

\_context = context;

}

public IEnumerable<Product> GetAll()

{

return \_context.Products.ToList();

}

public Product GetById(int id)

{

return \_context.Products.Find(id);

}

public void Add(Product product)

{

\_context.Products.Add(product);

\_context.SaveChanges();

}

public void Update(Product product)

{

\_context.Products.Update(product);

\_context.SaveChanges();

}

public void Delete(int id)

{

var product = \_context.Products.Find(id);

if (product != null)

{

\_context.Products.Remove(product);

\_context.SaveChanges();

}

}

}

C#

Copy

### **4. Register the Repository in the Dependency Injection Container**

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<AppDbContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

services.AddScoped<IProductRepository, ProductRepository>();

services.AddControllersWithViews();

}

C#

Copy

### **5. Use the Repository in a Controller**

public class ProductsController : Controller

{

private readonly IProductRepository \_productRepository;

public ProductsController(IProductRepository productRepository)

{

\_productRepository = productRepository;

}

public IActionResult Index()

{

var products = \_productRepository.GetAll();

return View(products);

}

}