🎯 What You Will Learn

1. What is a Generic Repository
2. Why use a Generic Repository
3. How to implement a Generic Repository for CRUD
4. How to use it for **multiple models** (Product, Customer, etc.)
5. How to build a **specific repository** if needed
6. Full working example (no EF Core, no async)

🔷 1. What is a Generic Repository?

A **Generic Repository** is a single class that handles CRUD operations for **any type of model**.

For example:

✓ Product
✓ Customer
✓ Order
✓ Category

All will use the SAME repository class:

IGenericRepository<Product>

IGenericRepository<Customer>

IGenericRepository<Order>

🔷 2. Why use Generic Repository?

Benefits

Avoid duplicate CRUD code

Easy to maintain

Extensible

Benefits

Reduces boilerplate code

Integrates well with Unit of Work

Helps follow DRY principle

🔷 3. Create Models

We will use **two models**.

Product Model

Models/Product.cs

```
namespace MyApi.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; } = "";
        public decimal Price { get; set; }
    }
}
```

Customer Model

Models/Customer.cs

```
namespace MyApi.Models
{
    public class Customer
```

```
    {
        public int Id { get; set; }
        public string FullName { get; set; } = "";
        public string Email { get; set; } = "";
    }
}
```

🔷 4. Generic Repository Interface (No Async)

Repositories/IGenericRepository.cs

```
using System.Collections.Generic;

namespace MyApi.Repositories
{
    public interface IGenericRepository<T> where T : class
    {
        IEnumerable<T> GetAll();
        T? GetById(int id);
        T Add(T entity);
        T? Update(T entity);
        bool Delete(int id);
    }
}
```

✓ Works for any model
✓ CRUD defined once, used everywhere

🔷 5. Generic Repository Implementation

Using **in-memory List<T> as database** (no EF Core, no database).

Repositories/GenericRepository.cs

```
using System.Collections.Generic;
using System.Linq;

namespace MyApi.Repositories
{
    public class GenericRepository<T> : IGenericRepository<T> where T : class
    {
        private readonly List<T> _data = new List<T>();
        private int _nextId = 1;

        public IEnumerable<T> GetAll()
        {
            return _data;
        }

        public T? GetById(int id)
        {
            return _data.FirstOrDefault(x =>
            {
                var prop = x.GetType().GetProperty("Id");
                return (int)prop.GetValue(x) == id;
            });
        }
```

```
public T Add(T entity)
{
    var prop = entity.GetType().GetProperty("Id");
    prop.SetValue(entity, _nextId++);

    _data.Add(entity);
    return entity;
}

public T? Update(T entity)
{
    var idProp = entity.GetType().GetProperty("Id");
    int id = (int)idProp.GetValue(entity);

    var existing = GetById(id);
    if (existing == null) return null;

    _data.Remove(existing);
    _data.Add(entity);

    return entity;
}

public bool Delete(int id)
{
    var entity = GetById(id);
```

```
            if (entity == null) return false;


            _data.Remove(entity);
            return true;
        }
    }
}
```

🔥 **Key Points**

- Works for ANY model
- Uses reflection to access the "Id" property
- List<T> simulates a database
- No async methods (as requested)

🔷 **6. Register Generic Repository in Program.cs**

Program.cs

```
builder.Services.AddSingleton(typeof(IGenericRepository<>), typeof(GenericRepository<>));
```

✓ This line enables Dependency Injection
✓ Now you can inject IGenericRepository<Product> and IGenericRepository<Customer>

🔷 **7. Create Controllers for Multiple Models**

Product Controller

Controllers/ProductController.cs

```
using Microsoft.AspNetCore.Mvc;
using MyApi.Models;
using MyApi.Repositories;
```

```
namespace MyApi.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class ProductController : ControllerBase
    {
        private readonly IGenericRepository<Product> _repo;

        public ProductController(IGenericRepository<Product> repo)
        {
            _repo = repo;
        }

        [HttpGet]
        public IActionResult GetAll()
        {
            return Ok(_repo.GetAll());
        }

        [HttpGet("{id}")]
        public IActionResult GetById(int id)
        {
            var product = _repo.GetById(id);
            if (product == null) return NotFound();
            return Ok(product);
```

```
}

[HttpPost]
public IActionResult Create(Product product)
{
    var created = _repo.Add(product);
    return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);
}

[HttpPut("{id}")]
public IActionResult Update(int id, Product product)
{
    if (id != product.Id)
        return BadRequest("ID mismatch");

    var updated = _repo.Update(product);
    if (updated == null) return NotFound();

    return Ok(updated);
}

[HttpDelete("{id}")]
public IActionResult Delete(int id)
{
    var deleted = _repo.Delete(id);
    return deleted ? NoContent() : NotFound();
```

```
        }
    }
}
```

Customer Controller

Controllers/CustomerController.cs

```
using Microsoft.AspNetCore.Mvc;
using MyApi.Models;
using MyApi.Repositories;

namespace MyApi.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class CustomerController : ControllerBase
    {
        private readonly IGenericRepository<Customer> _repo;

        public CustomerController(IGenericRepository<Customer> repo)
        {
            _repo = repo;
        }

        [HttpGet]
        public IActionResult GetAll()
        {
```

```
    return Ok(_repo.GetAll());
}

[HttpGet("{id}")]
public IActionResult GetById(int id)
{
    var customer = _repo.GetById(id);
    if (customer == null) return NotFound();
    return Ok(customer);
}

[HttpPost]
public IActionResult Create(Customer customer)
{
    var created = _repo.Add(customer);
    return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);
}

[HttpPut("{id}")]
public IActionResult Update(int id, Customer customer)
{
    if (id != customer.Id)
        return BadRequest("ID mismatch");

    var updated = _repo.Update(customer);
    if (updated == null) return NotFound();
```

```
        return Ok(updated);
    }

    [HttpDelete("{id}")]
    public IActionResult Delete(int id)
    {
        var deleted = _repo.Delete(id);
        return deleted ? NoContent() : NotFound();
    }
  }
}
```

🔥 8. Summary — What Did We Achieve?

Feature	Achieved
Generic Repository	✓
Multiple Models (Product, Customer)	✓
Reusable CRUD Methods	✓
Clean Architecture	✓
Dependency Injection	✓