✅ What You Will Learn

1. What is Model, Repository & Repository Pattern
2. Create Model class
3. Create Repository Interface & Implementation
4. Configure Repository in **Program.cs** (Dependency Injection)
5. Create Controller with Full CRUD Methods
6. Complete runnable example (GET, POST, PUT, DELETE)

🔷 1. Understanding the Layers

✓ Model

Represents a **data structure** (table in DB). Example: **Product**.

✓ Repository

A class that handles **all database operations** for a model:

- Get all data
- Get by id
- Create
- Update
- Delete

✓ Repository Pattern

This pattern helps to:

- Separate DB logic from Controller
- Make code reusable
- Easy to unit test
- Follows clean architecture

🔷 2. Create Model Class (Product.cs)

```
public class Product
{
    public int Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public int Quantity { get; set; }
}
```

🔷 3. Create Repository Interface (IProductRepository.cs)

```
public interface IProductRepository
{
    IEnumerable<Product> GetAll();
    Product GetById(int id);
    Product Add(Product product);
    Product Update(Product product);
    bool Delete(int id);
}
```

🔷 4. Create Repository Implementation (ProductRepository.cs)

➡️ For simplicity, we use a **List<Product>** as a fake database.

```
public class ProductRepository : IProductRepository
{
    private readonly List<Product> _products = new List<Product>()
    {
        new Product(){ Id = 1, Name = "Laptop", Price = 55000, Quantity = 10},
```

```
    new Product(){ Id = 2, Name = "Mouse", Price = 500, Quantity = 50}
};

public IEnumerable<Product> GetAll()
{
    return _products;
}

public Product GetById(int id)
{
    return _products.FirstOrDefault(p => p.Id == id);
}

public Product Add(Product product)
{
    product.Id = _products.Max(p => p.Id) + 1;
    _products.Add(product);
    return product;
}

public Product Update(Product product)
{
    var existing = _products.FirstOrDefault(x => x.Id == product.Id);
    if (existing == null) return null;

    existing.Name = product.Name;
```

```
        existing.Price = product.Price;
        existing.Quantity = product.Quantity;

        return existing;
    }

    public bool Delete(int id)
    {
        var existing = _products.FirstOrDefault(x => x.Id == id);
        if (existing == null)
            return false;

        _products.Remove(existing);
        return true;
    }
}
```

🔷 5. Register Repository in Program.cs

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

This enables **Dependency Injection** so the controller can use the repository.

🔷 6. Create Controller (ProductController.cs)

With complete CRUD (GET, POST, PUT, DELETE)

```
[ApiController]
[Route("api/[controller]")]
public class ProductController : ControllerBase
```

```
{
    private readonly IProductRepository _repo;

    public ProductController(IProductRepository repo)
    {
        _repo = repo;
    }

    // GET: api/product
    [HttpGet]
    public IActionResult GetAll()
    {
        var products = _repo.GetAll();
        return Ok(products);
    }

    // GET: api/product/2
    [HttpGet("{id}")]
    public IActionResult GetById(int id)
    {
        var product = _repo.GetById(id);

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

        return Ok(product);
```

```
}

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

// PUT: api/product/2
[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);
}

// DELETE: api/product/2
```

```
    [HttpDelete("{id}")]
    public IActionResult Delete(int id)
    {
        var result = _repo.Delete(id);

        if (!result)
            return NotFound();

        return NoContent(); // 204
    }
}
```

🎯 7. CRUD Summary

Operation	Method	URL	Description
Create	POST	/api/product	Add new product
Read All	GET	/api/product	Get all products
Read One	GET	/api/product/{id}	Get product by id
Update	PUT	/api/product/{id}	Update product
Delete	DELETE	/api/product/{id}	Delete product

⭐ Your Project is Ready!

You now have:

✓ Model
✓ Repository Interface
✓ Repository Implementation

✓ Dependency Injection

✓ Full CRUD Controller