✅ 1. What Are CRUD Operations?

CRUD stands for:

- **C**reate → Insert a new record
- **R**ead → Get one or many records
- **U**pdate → Modify an existing record
- **D**elete → Remove a record

These operations are implemented inside:

- **Controller** (API endpoints)
- **Repository/Service layer** (business logic)

✅ 2. Why Do We Use DTOs? (Data Transfer Objects)

DTOs are classes used to **send and receive data** from APIs.

💡 Why not directly use Entity/Model?

Problem if using Entity directly	How DTO solves it
Exposes internal database structure	DTO exposes only required fields
Sensitive fields get leaked (password, etc.)	DTO hides unnecessary fields
Entities become tightly coupled with API	DTO keeps layers independent
Update request might modify fields not allowed	DTO controls allowed update fields

Types of DTOs

- **CreateDTO** → For POST requests
- **UpdateDTO** → For PUT requests
- **ReadDTO** → For GET responses

✅ 3. Why AutoMapper?

AutoMapper is a library that **automatically maps data between Entity ↔ DTO**.

Without AutoMapper (Manual mapping):

```
var product = new Product {
    Id = dto.Id,
    Name = dto.Name,
    Price = dto.Price
};
```

With AutoMapper:

```
var product = _mapper.Map<Product>(dto);
```

✓ Benefits of AutoMapper

- Reduces boilerplate code
- Avoids repetitive mapping code
- Makes controller/service cleaner
- Reduces human errors in field mapping

✅ 4. CRUD Example using Entity + DTO + AutoMapper

🎯 Step 1: Entity / Model

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

🎯 Step 2: DTOs

Create DTO

```
public class ProductCreateDto
```

```
{
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

Update DTO

```
public class ProductUpdateDto
{
    public string Name { get; set; }
    public decimal Price { get; set; }
}
```

Read DTO

```
public class ProductReadDto
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

🎯 Step 3: AutoMapper Profile

```
using AutoMapper;

public class ProductProfile : Profile
{
    public ProductProfile()
    {
        CreateMap<Product, ProductReadDto>();
        CreateMap<ProductCreateDto, Product>();
```

```
        CreateMap<ProductUpdateDto, Product>();
    }
}
```

Register in **Program.cs**:

```
builder.Services.AddAutoMapper(typeof(Program));
```

🎯 Step 4: Repository (Business Layer)

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

Implementation (Simple In-Memory Example)

```
public class ProductRepository : IProductRepository
{
    private readonly List<Product> _products = new();

    public IEnumerable<Product> GetAll() => _products;

    public Product GetById(int id) => _products.FirstOrDefault(x => x.Id == id);

    public Product Add(Product product)
    {
```

```
        product.Id = _products.Count + 1;
        _products.Add(product);
        return product;
    }

    public void Update(Product product)
    {
        var existing = GetById(product.Id);
        if (existing == null) return;

        existing.Name = product.Name;
        existing.Price = product.Price;
    }

    public void Delete(int id)
    {
        var p = GetById(id);
        if (p != null) _products.Remove(p);
    }
}
```

🎯 Step 5: Controller (Using DTO + AutoMapper)

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

```
private readonly IProductRepository _repo;
private readonly IMapper _mapper;

public ProductsController(IProductRepository repo, IMapper mapper)
{
    _repo = repo;
    _mapper = mapper;
}

// GET ALL
[HttpGet]
public ActionResult<IEnumerable<ProductReadDto>> GetAll()
{
    var products = _repo.GetAll();
    return Ok(_mapper.Map<IEnumerable<ProductReadDto>>(products));
}

// GET BY ID
[HttpGet("{id}")]
public ActionResult<ProductReadDto> GetById(int id)
{
    var product = _repo.GetById(id);
    if (product == null) return NotFound();

    return Ok(_mapper.Map<ProductReadDto>(product));
}
```

```
// CREATE
[HttpPost]
public ActionResult<ProductReadDto> Create(ProductCreateDto dto)
{
    var product = _mapper.Map<Product>(dto);
    _repo.Add(product);

    var readDto = _mapper.Map<ProductReadDto>(product);
    return CreatedAtAction(nameof(GetById), new { id = product.Id }, readDto);
}

// UPDATE
[HttpPut("{id}")]
public IActionResult Update(int id, ProductUpdateDto dto)
{
    var product = _repo.GetById(id);
    if (product == null) return NotFound();

    _mapper.Map(dto, product);
    _repo.Update(product);

    return NoContent();
}

// DELETE
```

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

        _repo.Delete(id);
        return NoContent();
    }
}
```