LAB-1:

## Understanding ORM and Entity Framework Core in Inventory Management

## 1. What is ORM?

**Object-Relational Mapping (ORM)** is a programming technique that allows developers to interact with a relational database using the constructs of an object-oriented language like C#. Instead of writing SQL queries for every database operation, you work with C# classes and objects, and the ORM handles the translation to SQL and the mapping to database tables behind the scenes.

## How ORM Maps C# Classes to Database Tables

Each C# class (e.g., Product, Category) corresponds to a table in the database.

Each property of the class (e.g., Name, Price, StockLevel) maps to a column in the table.

Each instance of the class represents a row in the table.

Relationships (like one-to-many between Category and Product) are mapped using navigation properties and foreign keys[1](https://education.launchcode.org/csharp-web-dev-curriculum/orm-part-1/reading/orm-intro/index.html" \t "https://www.perplexity.ai/search/_blank)[2](https://www.devart.com/entitydeveloper/what-is-orm.html" \t "https://www.perplexity.ai/search/_blank)[3](https://reviewnprep.com/blog/object-relational-mapping-orm-a-beginners-guide/" \t "https://www.perplexity.ai/search/_blank).

## Benefits of Using ORM

**Productivity:** Developers can focus on business logic rather than repetitive SQL code.

**Maintainability:** Changes in the data model are reflected in one place (the C# classes), making updates easier.

**Abstraction from SQL:** Most CRUD (Create, Read, Update, Delete) operations are handled in C#, reducing the need to write raw SQL and minimizing errors[4](https://www.telerik.com/blogs/dotnet-basics-orm-object-relational-mapping" \t "https://www.perplexity.ai/search/_blank)[2](https://www.devart.com/entitydeveloper/what-is-orm.html" \t "https://www.perplexity.ai/search/_blank).

## 2. EF Core vs. Entity Framework (EF6)

|  |  |  |
| --- | --- | --- |
| **Feature** | **EF Core** | **Entity Framework 6 (EF6)** |
| Platform Support | Cross-platform (.NET Core, .NET 5/6/7/8, Linux, macOS, Windows) | Windows-only (.NET Framework) |
| Performance | Lightweight, faster, optimized | Mature, but heavier and slower |
| Modern Features | LINQ, async queries, compiled queries, better bulk operations | Limited async, no compiled queries |
| Flexibility | Modular, extensible, supports NoSQL and relational databases | Relational databases only |
| Maturity | Newer, evolving rapidly | Stable, feature-rich, long history |
| Design Tools | No visual designer | Visual designer (EDMX) available |

**EF Core** is designed for modern .NET development, offering better performance, cross-platform support, and advanced features like LINQ, async/await, and compiled queries.

**EF6** is more mature and stable, but limited to Windows and lacks some of the modern features and flexibility of EF Core[5](https://github.com/dotnet/EntityFramework.Docs/blob/main/entity-framework/efcore-and-ef6/index.md" \t "https://www.perplexity.ai/search/_blank)[6](https://learn.microsoft.com/en-us/ef/ef6/?redirectedfrom=MSDN" \t "https://www.perplexity.ai/search/_blank)[7](https://riptutorial.com/entity-framework-core/example/24791/side-by-side-comparison" \t "https://www.perplexity.ai/search/_blank).

## 3. EF Core 8.0 Key Features

EF Core 8.0 introduces several enhancements that are particularly useful for building robust inventory management systems:

**JSON Column Mapping:** Store and query JSON data directly in database columns, making it easier to handle complex or dynamic data structures without extra tables.

**Improved Performance with Compiled Models:** Faster query execution and reduced startup time by pre-compiling the data model, which is especially beneficial for large applications.

**Interceptors:** Add custom logic to database operations (e.g., logging, security checks) by intercepting commands at runtime.

**Better Bulk Operations:** Enhanced support for efficient bulk updates and deletes, improving performance when processing large datasets[8](https://blog.devgenius.io/whats-new-in-ef-core-8-ce5f947a7358?gi=6aef8f842f48" \t "https://www.perplexity.ai/search/_blank)[9](https://devonblog.com/continuous-delivery/exploring-the-latest-features-of-entity-framework-core-in-net-8/" \t "https://www.perplexity.ai/search/_blank)[10](https://learn.microsoft.com/en-us/ef/core/what-is-new/ef-core-8.0/whatsnew" \t "https://www.perplexity.ai/search/_blank).

## Summary

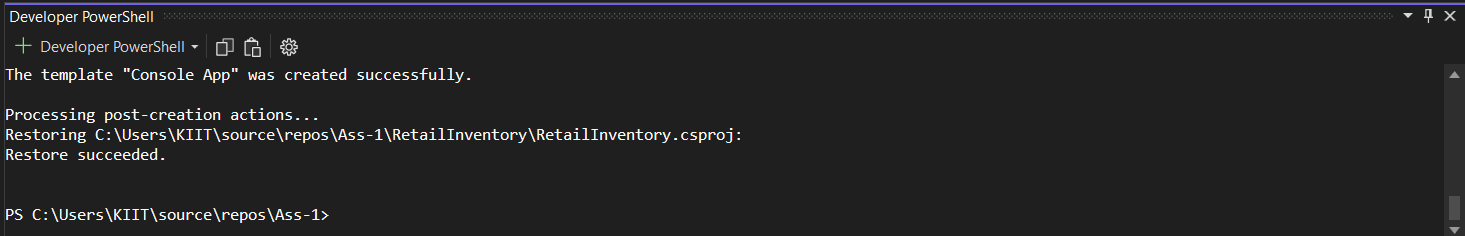
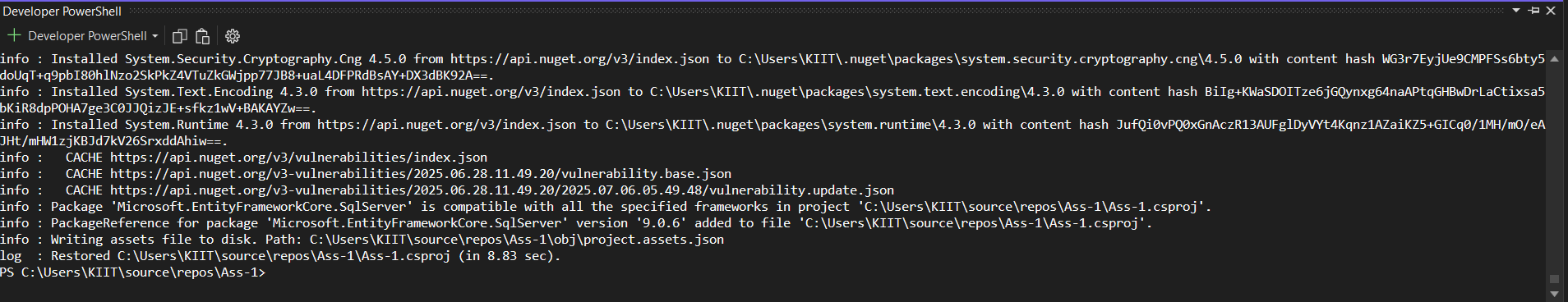
Using an ORM like **Entity Framework Core** in your inventory management system allows you to:

Model your products, categories, and stock levels as C# classes.

Automatically map these classes to SQL Server tables.

Benefit from increased productivity, maintainability, and abstraction from SQL.

Leverage modern features and performance improvements, especially with EF Core 8.0, to build scalable, efficient, and cross-platform applications[1](https://education.launchcode.org/csharp-web-dev-curriculum/orm-part-1/reading/orm-intro/index.html" \t "https://www.perplexity.ai/search/_blank)[2](https://www.devart.com/entitydeveloper/what-is-orm.html" \t "https://www.perplexity.ai/search/_blank)[8](https://blog.devgenius.io/whats-new-in-ef-core-8-ce5f947a7358?gi=6aef8f842f48" \t "https://www.perplexity.ai/search/_blank).



LAB-2:

1.public class Category

{

public int Id { get; set; }

public string Name { get; set; }

public List<Product> Products { get; set; } = new();

}

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

2.using Microsoft.EntityFrameworkCore;

public class AppDbContext : DbContext

{

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

public DbSet<Category> Categories { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Your\_Connection\_String\_Here");

}

}

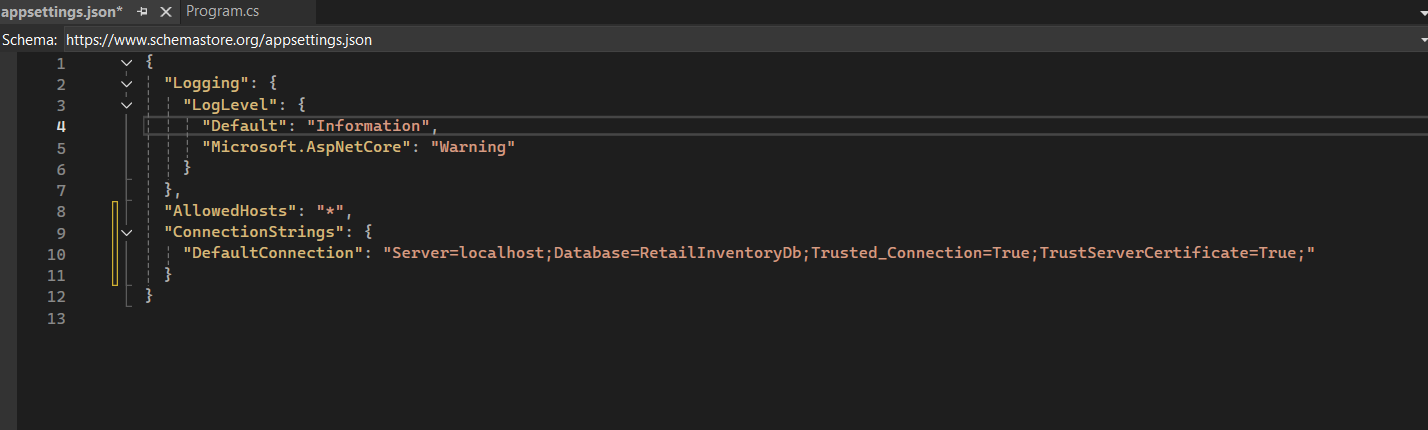
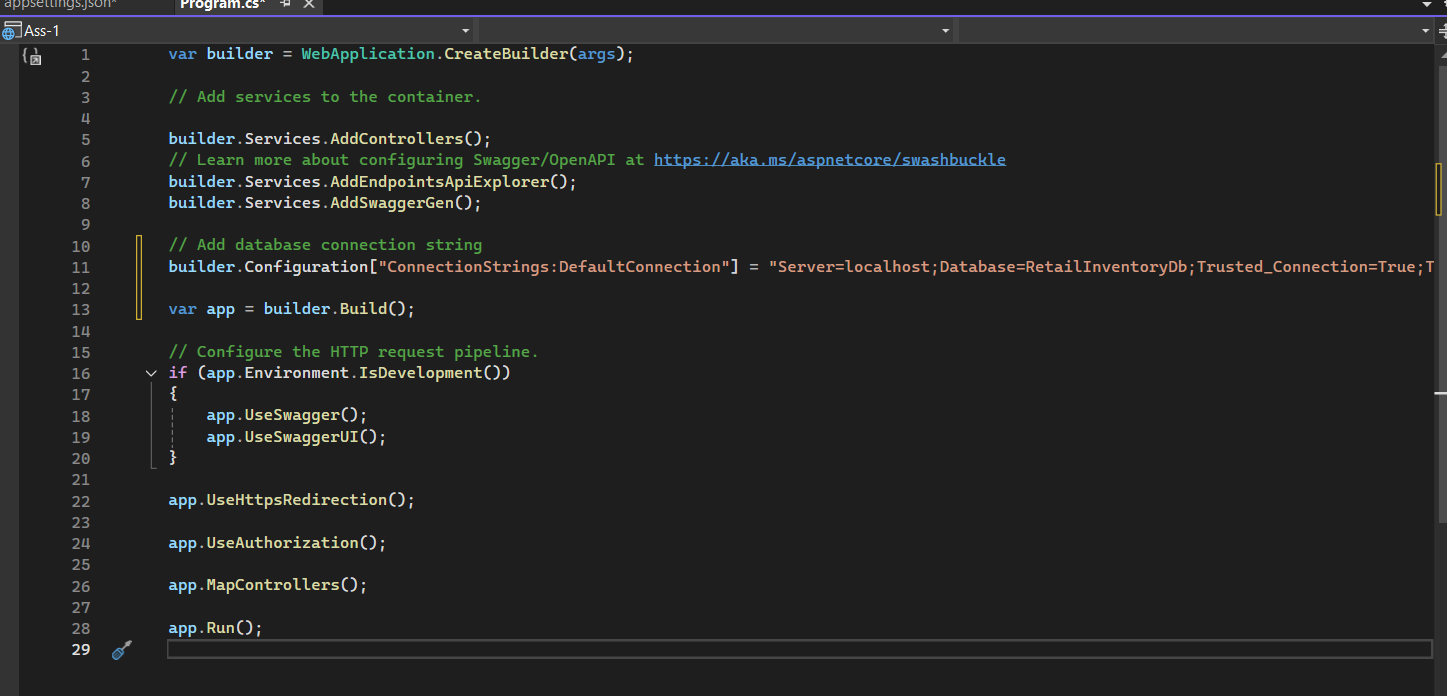
3.{

"ConnectionStrings": {

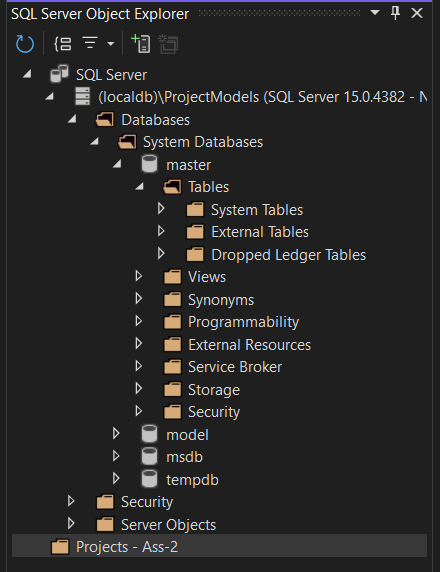
"DefaultConnection": "Server=localhost;Database=RetailInventoryDb;Trusted\_Connection=True;TrustServerCertificate=True;"

}

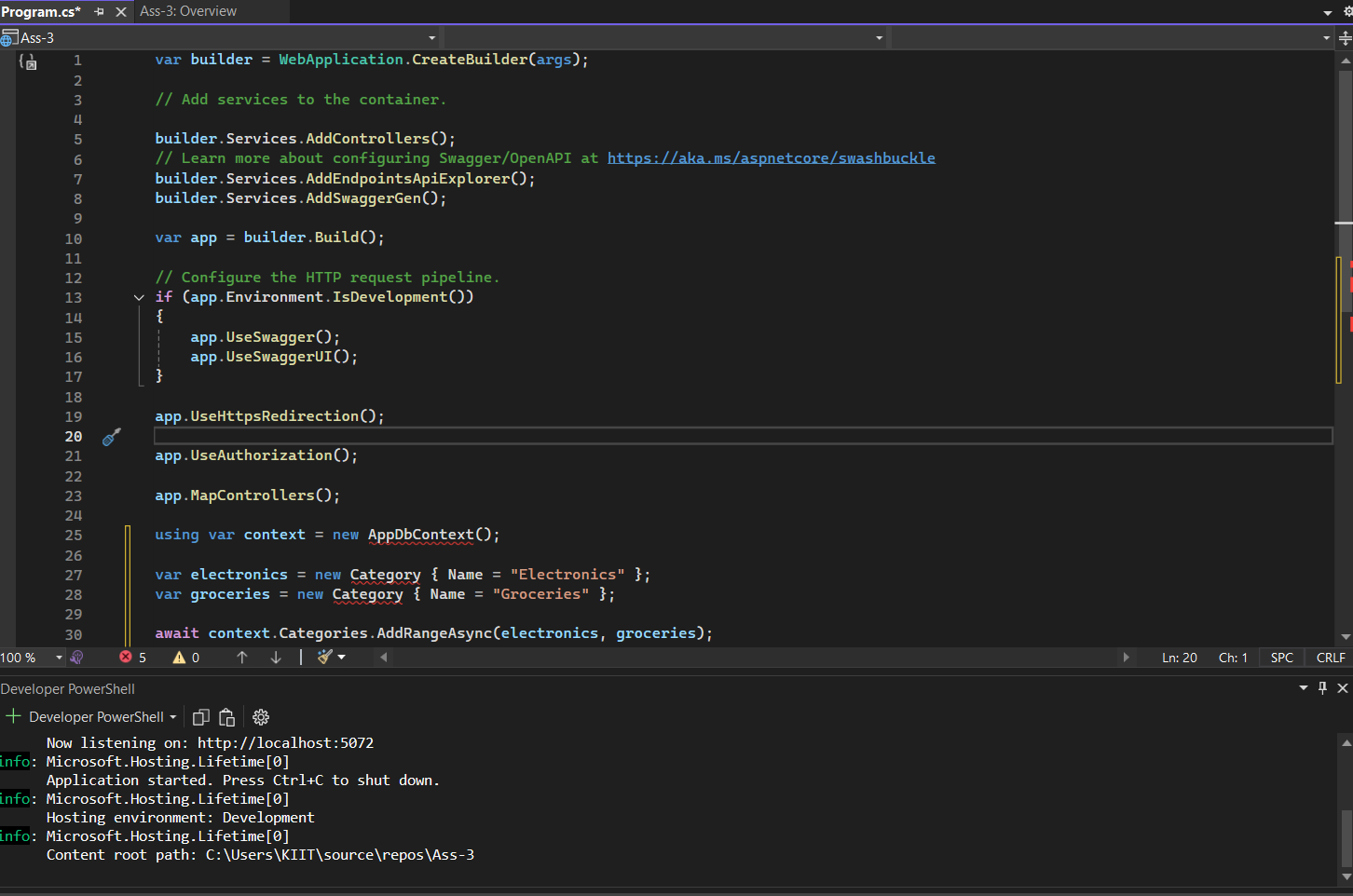
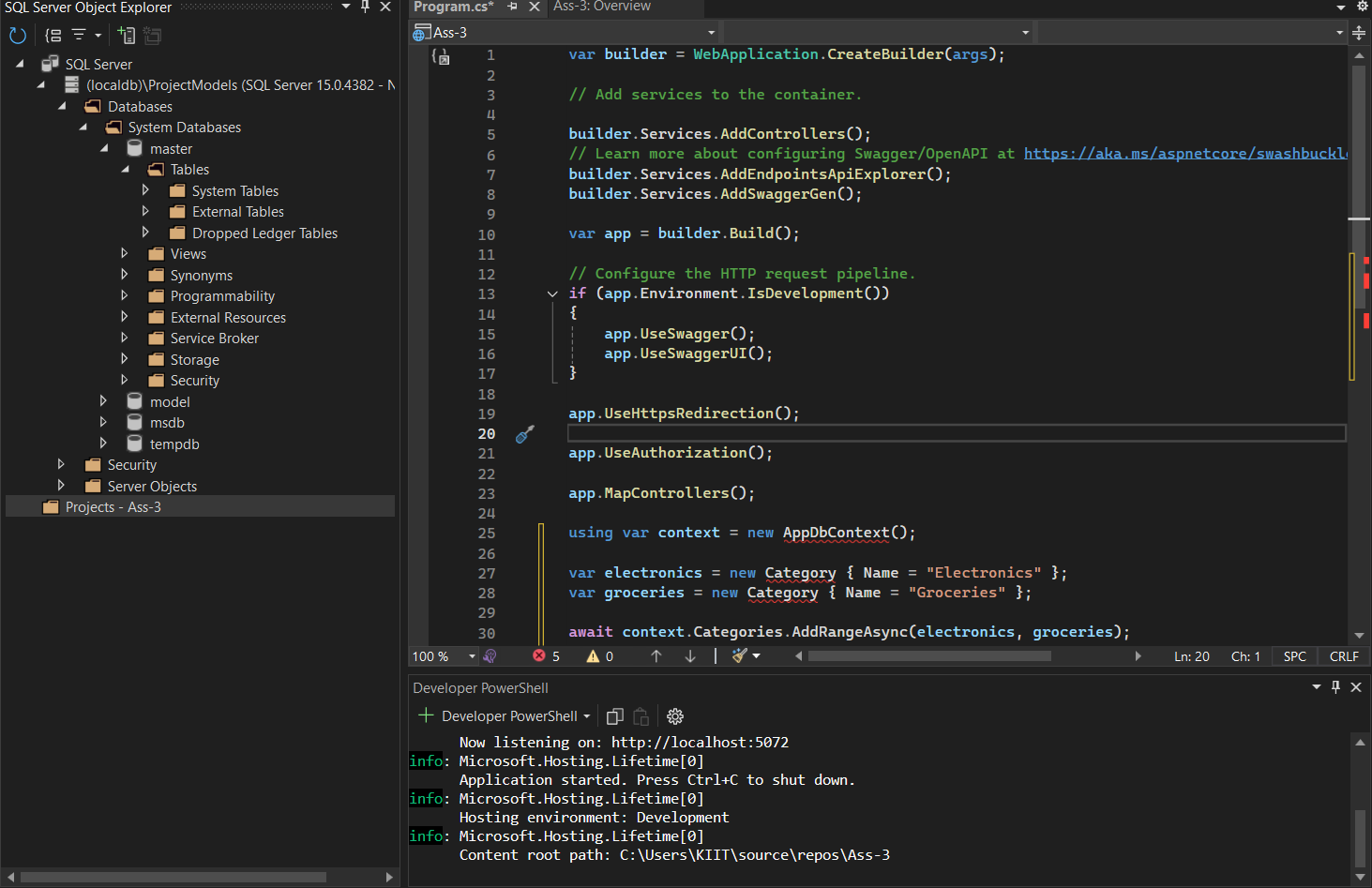
}



LAB-3:



LAB-4:



using var context = new AppDbContext();

var electronics = new Category { Name = "Electronics" };

var groceries = new Category { Name = "Groceries" };

await context.Categories.AddRangeAsync(electronics, groceries);

var product1 = new Product { Name = "Laptop", Price = 75000, Category = electronics };

var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };

await context.Products.AddRangeAsync(product1, product2);

await context.SaveChangesAsync();

LAB-5:

using Microsoft.EntityFrameworkCore;

using var context = new AppDbContext();

// 1. Retrieve All Products

var products = await context.Products.ToListAsync();

foreach (var p in products)

Console.WriteLine($"{p.Name} - ₹{p.Price}");

// 2. Find by ID

var product = await context.Products.FindAsync(1);

Console.WriteLine($"Found: {product?.Name}");

// 3. FirstOrDefault with Condition

var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

Console.WriteLine($"Expensive: {expensive?.Name}");

