**Entity Framework Core8.0**

**Lab01: Understanding ORM with a Retail Inventory system**

**What is ORM?**

ORM (Object Relational Mapping) is a technique that maps C# classes to database tables.

Instead of writing raw SQL, you interact with objects in your code.

Benefits:

Improved productivity.

Easier to maintain and refactor.

SQL abstraction using LINQ.

**EF Core vs EF Framework:**

EF Core:

Cross-platform.

Lightweight, modern, fast.

Supports LINQ, async queries, compiled queries.

EF Framework (EF6):

Windows-only.

Mature and stable, but lacks modern features.

**EF Core 8.0 Features:**

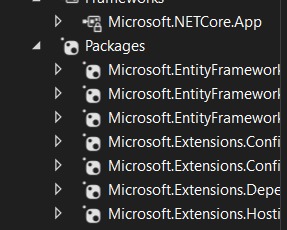
Native JSON column mapping.

Compiled models for better startup performance.

Interceptors for advanced scenarios.

Improved support for bulk operations.

**Creation of the packages**

****

**Lab02: Setting Up the Database Context for a Retail Store:**

**Models:**

1. **Product Model**

namespace EfCoreWk3Demo.Models

{

internal class Product

{

public int Id { get; set; };

public string Name { get; set; }

public string Price { get; set; }

public string CategoryId { get; set; }

public string Category Category { get; set; } //Navigation property to Category

}

}

1. **Category Model**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace EfCoreWk3Demo.Models

{

internal class Category

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**AppDbContext**

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using EfCoreWk3Demo.Models;

namespace EfCoreWk3Demo.Data

{

internal class AppDbContext:DbContext

{

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)

{

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

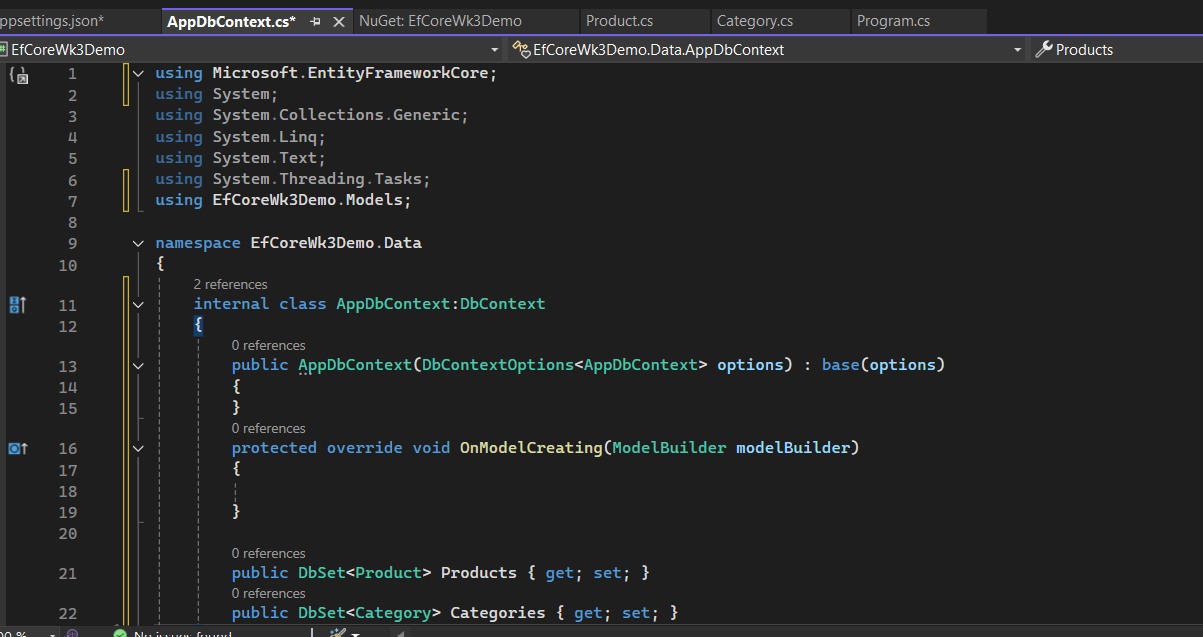
}

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

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

}

}



**appsettings,json**

{

"ConnectionStrings": {"DefaultConnection": "server=.\\SQLEXPRESS;database=EfCore3Db;trusted\_connection=yes;TrustServerCertificate=True"}

}

**Lab03: Using EF Core CLI to Create and Apply Migrations**

**install EF Core Tools**

dotnet tool install --global dotnet-ef

dotnet ef

**Enable Migrations in your project:**

Add-Migration InitialCreate

**Apply Migration to Create Databases:**

Update-Database



**Lab04: Inserting Intial Data into the Database:**

**Program.cs**

using System;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

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();

Console.WriteLine("Data inserted successfully.");

}

}

****

**Lab05: Retrieving Data from the Database:**

**program.cs**

using var context = new AppDbContext();

// Retrieve All Products

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

Console.WriteLine("\nAll Products:");

foreach (var p in products)

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

// Find by ID

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

Console.WriteLine($"\nProduct with ID=1: {productById?.Name}");

// FirstOrDefault with Condition

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

Console.WriteLine($"\nFirst product > ₹50,000: {expensiveProduct?.Name}");

