# **Name- Sayandeep Dey (SupersetID:** **6363427)**

# **WEEK – 3 (Handson- Exercises)**

1. **Entity Framework Core 8.0:**

**LAB-1: Understanding ORM with a RetailInventory System:-**

**Code:**

**Creating the Project named RetailInventory:**

dotnet new console -n RetailInventory

cd RetailInventory

**Downloading all this Packages into the (.csproj):**

* dotnet add package Microsoft.EntityFrameworkCore
* dotnet add package Microsoft.EntityFrameworkCore.SqlServer
* dotnet add package Microsoft.EntityFrameworkCore.Tools
* dotnet add package Microsoft.EntityFrameworkCore.Design
* dotnet add package Microsoft.Extensions.Configuration
* dotnet add package Microsoft.Extensions.Configuration.Json
* dotnet add package Microsoft.Extensions.DependencyInjection
* dotnet add package Microsoft.Extensions.Hosting

**Downloading all this Packages into the (.csproj):**

**Creating Models file:**

**In Product.cs:**

namespace RetailInventory.Models;

public class Product

{

    public int ProductId { get; set; }

    public string Name { get; set; }

    public int Stock { get; set; }

    public int CategoryId { get; set; }

    public Category Category { get; set; }

}

**In Category.cs:**

namespace RetailInventory.Models;

public class Category

{

    public int CategoryId { get; set; }

    public string Name { get; set; }

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

}

**Creating Data folder:**

**In AppDbcontext.cs:**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory.Data;

public class AppDbContext : DbContext

{

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

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

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

    {

        optionsBuilder.UseSqlServer("Server=localhost;Database=RetailDB;Trusted\_Connection=True;TrustServerCertificate=True;");

    }

}

**Creating Migrations:**

dotnet tool install --global dotnet-ef

dotnet ef migrations add InitialCreate

dotnet ef database update

**In Program.cs:**

using RetailInventory.Data;

using RetailInventory.Models;

using Microsoft.EntityFrameworkCore;

var context = new AppDbContext();

context.Database.EnsureCreated();

// Seed categories if none exist

if (!context.Categories.Any())

{

    context.Categories.AddRange(

        new Category { Name = "Electronics" },

        new Category { Name = "Groceries" },

        new Category { Name = "Clothing" }

    );

    context.SaveChanges();

}

bool running = true;

while (running)

{

    Console.WriteLine("\n Retail Inventory Menu");

    Console.WriteLine("1. View Products");

    Console.WriteLine("2. Add Product");

    Console.WriteLine("3. Update Product");

    Console.WriteLine("4. Delete Product");

    Console.WriteLine("5. Exit");

    Console.Write("Select an option: ");

    var choice = Console.ReadLine();

    switch (choice)

    {

        case "1":

            ViewProducts();

            break;

        case "2":

            AddProduct();

            break;

        case "3":

            UpdateProduct();

            break;

        case "4":

            DeleteProduct();

            break;

        case "5":

            running = false;

            break;

        default:

            Console.WriteLine(" Invalid option. Try again.");

            break;

    }

}

//  View Products

void ViewProducts()

{

    var products = context.Products.Include(p => p.Category).ToList();

    if (!products.Any())

    {

        Console.WriteLine("No products found.");

        return;

    }

    Console.WriteLine("\n Product List:");

    foreach (var p in products)

        Console.WriteLine($"{p.ProductId}. {p.Name} ({p.Category.Name}) - Stock: {p.Stock}");

}

//  Add Product

void AddProduct()

{

    Console.Write("Enter product name: ");

    var name = Console.ReadLine();

    Console.Write("Enter stock: ");

    if (!int.TryParse(Console.ReadLine(), out int stock))

    {

        Console.WriteLine("Invalid stock value.");

        return;

    }

    Console.Write("Enter category name: ");

    var categoryName = Console.ReadLine();

    var category = context.Categories.FirstOrDefault(c => c.Name.ToLower() == categoryName.ToLower());

    if (category == null)

    {

        category = new Category { Name = categoryName };

        context.Categories.Add(category);

        context.SaveChanges();

        Console.WriteLine($" New category '{categoryName}' created.");

    }

    var product = new Product

    {

        Name = name,

        Stock = stock,

        CategoryId = category.CategoryId

    };

    context.Products.Add(product);

    context.SaveChanges();

    Console.WriteLine(" Product added.");

}

//  Update Product

void UpdateProduct()

{

    ViewProducts();

    Console.Write("Enter Product ID to update: ");

    if (!int.TryParse(Console.ReadLine(), out int id))

    {

        Console.WriteLine("Invalid ID.");

        return;

    }

    var product = context.Products.Find(id);

    if (product == null)

    {

        Console.WriteLine("Product not found.");

        return;

    }

    Console.Write("New name (leave blank to keep current): ");

    var name = Console.ReadLine();

    if (!string.IsNullOrWhiteSpace(name))

        product.Name = name;

    Console.Write("New stock (leave blank to keep current): ");

    var stockInput = Console.ReadLine();

    if (int.TryParse(stockInput, out int newStock))

        product.Stock = newStock;

    context.SaveChanges();

    Console.WriteLine(" Product updated.");

}

//  Delete Product

void DeleteProduct()

{

    ViewProducts();

    Console.Write("Enter Product ID to delete: ");

    if (!int.TryParse(Console.ReadLine(), out int id))

    {

        Console.WriteLine("Invalid ID.");

        return;

    }

    var product = context.Products.Find(id);

    if (product == null)

    {

        Console.WriteLine("Product not found.");

        return;

    }

    context.Products.Remove(product);

    context.SaveChanges();

    Console.WriteLine(" Product deleted.");

}

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**A screen shot of a computer

AI-generated content may be incorrect.**

**In SSMS:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab-2: Setting Up the Database Context for a Retail Store:**

**Code: (In Product.cs):**

**Code:**

namespace RetailInventory.Models;

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; }

}

**In Category.cs:**

namespace RetailInventory.Models;

public class Category

{

    public int Id { get; set; }

    public string Name { get; set; }

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

}

**In Program.cs:**

using RetailInventory.Data;

using RetailInventory.Models;

var context = new AppDbContext();

context.Database.EnsureCreated();

if (!context.Categories.Any())

{

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

    context.Categories.Add(electronics);

    context.Products.Add(new Product

    {

        Name = "Laptop",

        Price = 59999,

        Category = electronics

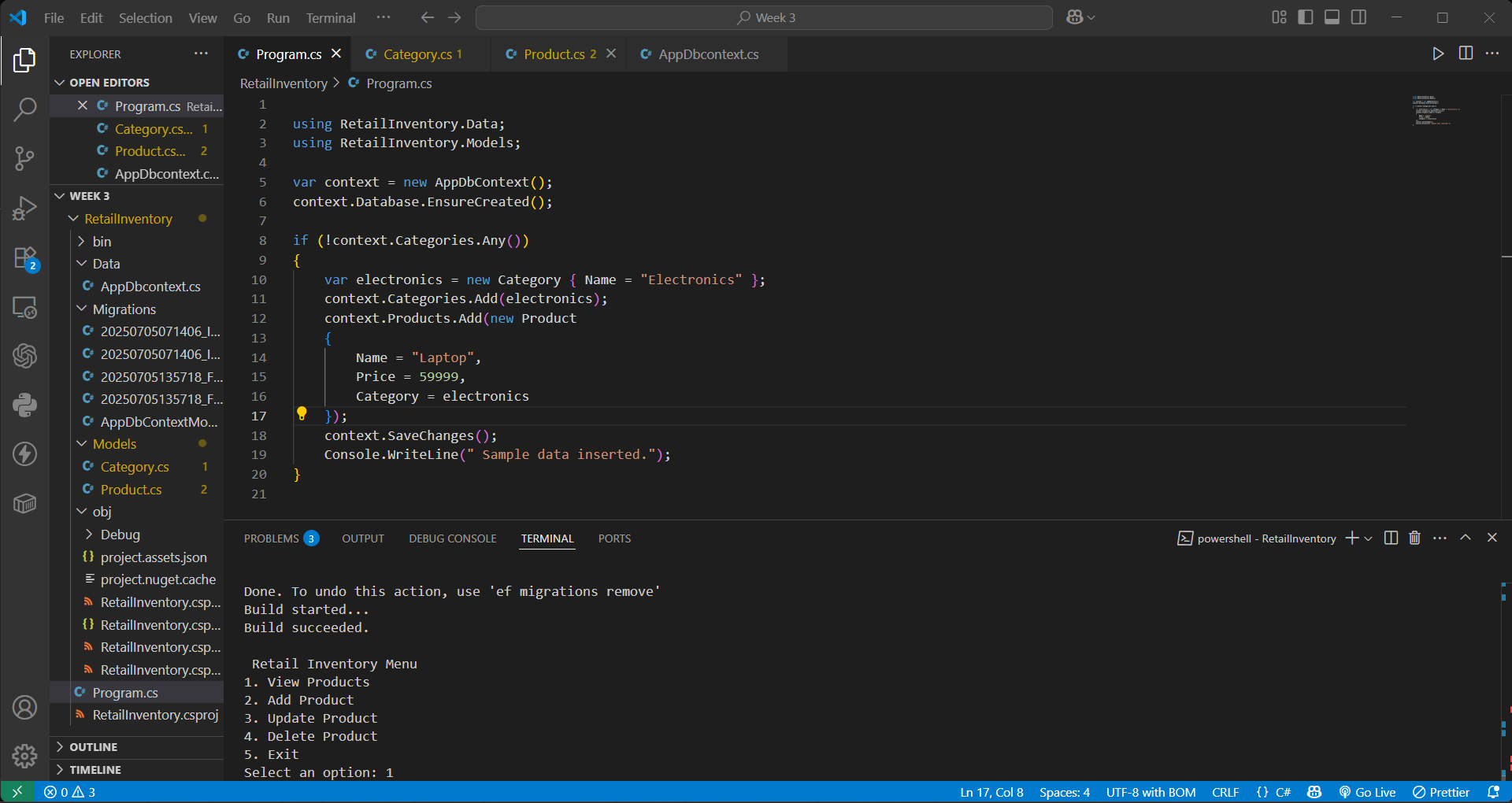
    });

    context.SaveChanges();

    Console.WriteLine(" Sample data inserted.");

}

**Output:**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab: 3: Using EF Core CLI to Create and Apply Migrations:**

**Code:**

dotnet tool install --global dotnet-ef

dotnet ef migrations add InitialCreate

dotnet ef database update

**Output:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

* **Opened SSMS**
* **Connected to: localdb**
* **Navigated to RetailDB → Tables**
* **Verified the presence of:**
* **Products**
* **Categories**
* **EFMigrationsHistory**

**Output:**

INSERT INTO Categories (Name) VALUES ('Electronics'), ('Groceries');

INSERT INTO Products (Name, Price, CategoryId) VALUES ('Laptop', 59999, 1);

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab-4 - Inserting Initial Data into the Database**

**Code:**

**In Program.cs:**

using RetailInventory.Data;

using RetailInventory.Models;

using RetailInventory.Models;

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 };

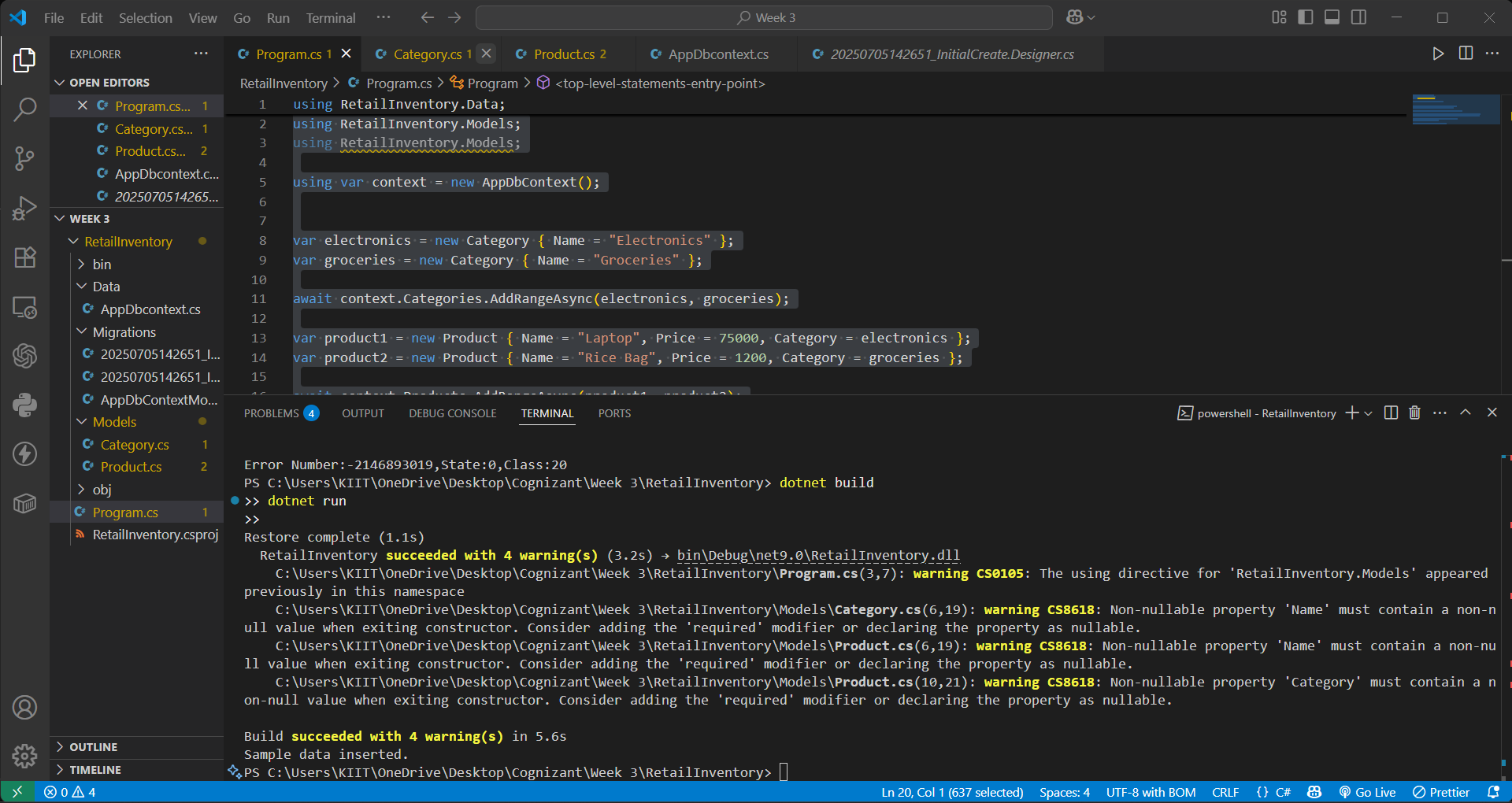
var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };

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

await context.SaveChangesAsync();

Console.WriteLine("Sample data inserted.");

**Output:**

****

**In SSMS:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab- 5: Retrieving Data from the Database**

**Code:**

**In Program.cs:**

using System;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using RetailInventory.Models; // make sure this is not duplicated

using var context = new AppDbContext();

// Step 1: Retrieve All Products

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

Console.WriteLine("All Products:");

foreach (var p in products)

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

Console.WriteLine(); // just spacing

// Step 2: Find by ID

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

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

Console.WriteLine(); // spacing

// Step 3: FirstOrDefault with Condition

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

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

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab- 6 : Updating and Deleting Records:**

**Code:**

**In Program.cs:**

using RetailInventory.Models;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using var context = new AppDbContext();

// Update Product Price

var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");

if (product != null)

{

    product.Price = 70000;

    await context.SaveChangesAsync();

    Console.WriteLine("Laptop price updated to ₹70,000");

}

else

{

    Console.WriteLine("Laptop not found");

}

// Delete Product

var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");

if (toDelete != null)

{

    context.Products.Remove(toDelete);

    await context.SaveChangesAsync();

    Console.WriteLine("Rice Bag deleted from database");

}

else

{

    Console.WriteLine("Rice Bag not found");

}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**In SSMS:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Lab- 7 : Writing Queries with LINQ:**

**Code:**

**In Program.cs:**

using RetailInventory.Models;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using var context = new AppDbContext();

// 1. Filter and Sort Products with Price > 1000 (Descending Order)

var filtered = await context.Products

    .Where(p => p.Price > 1000)

    .OrderByDescending(p => p.Price)

    .ToListAsync();

Console.WriteLine("Filtered & Sorted Products (Price > ₹1000):");

foreach (var product in filtered)

{

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

}

// 2. Project into Anonymous DTO

var productDTOs = await context.Products

    .Select(p => new { p.Name, p.Price })

    .ToListAsync();

Console.WriteLine("\nProduct DTOs (Name & Price only):");

foreach (var dto in productDTOs)

{

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

}

**Output:**

**A screen shot of a computer

AI-generated content may be incorrect.**