**Week 3 Assignment**

Lab 1: Understanding ORM with a Retail Inventory System

Programs  
Models:

[Category.cs](http://category.cs)  
using System.ComponentModel.DataAnnotations;

namespace RetailInventory.Models

{

public class Category

{

public int CategoryId { get; set; }

[Required]

[MaxLength(100)]

public string Name { get; set; } = string.Empty;

[MaxLength(500)]

public string Description { get; set; } = string.Empty;

// Navigation property

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

}

}

[Product.cs](http://product.cs)  
using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace RetailInventory.Models

{

public class Product

{

public int ProductId { get; set; }

[Required]

[MaxLength(200)]

public string Name { get; set; } = string.Empty;

[MaxLength(1000)]

public string Description { get; set; } = string.Empty;

[Column(TypeName = "decimal(18,2)")]

public decimal Price { get; set; }

public int StockQuantity { get; set; }

// Foreign key

public int CategoryId { get; set; }

// Navigation property

public virtual Category Category { get; set; } = null!;

public DateTime CreatedDate { get; set; } = DateTime.Now;

public bool IsActive { get; set; } = true;

}

}

[RetailInventoryContext.cs](http://retailinventorycontext.cs)  
using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory.Data

{

public class RetailInventoryContext : DbContext

{

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

{

}

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

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

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Configure Category

modelBuilder.Entity<Category>(entity =>

{

entity.HasKey(e => e.CategoryId);

entity.Property(e => e.Name).IsRequired().HasMaxLength(100);

entity.HasIndex(e => e.Name).IsUnique();

});

// Configure Product

modelBuilder.Entity<Product>(entity =>

{

entity.HasKey(e => e.ProductId);

entity.Property(e => e.Name).IsRequired().HasMaxLength(200);

entity.Property(e => e.Price).HasPrecision(18, 2);

// Configure relationship

entity.HasOne(d => d.Category)

.WithMany(p => p.Products)

.HasForeignKey(d => d.CategoryId)

.OnDelete(DeleteBehavior.Restrict);

});

// Seed data

SeedData(modelBuilder);

}

private void SeedData(ModelBuilder modelBuilder)

{

// Seed Categories

modelBuilder.Entity<Category>().HasData(

new Category { CategoryId = 1, Name = "Electronics", Description = "Electronic devices and gadgets" },

new Category { CategoryId = 2, Name = "Clothing", Description = "Apparel and accessories" },

new Category { CategoryId = 3, Name = "Books", Description = "Books and educational materials" }

);

modelBuilder.Entity<Product>().HasData(

new Product { ProductId = 1, Name = "Laptop", Description = "High-performance laptop", Price = 999.99m, StockQuantity = 50, CategoryId = 1 },

new Product { ProductId = 2, Name = "Smartphone", Description = "Latest smartphone", Price = 699.99m, StockQuantity = 100, CategoryId = 1 },

new Product { ProductId = 3, Name = "T-Shirt", Description = "Cotton t-shirt", Price = 19.99m, StockQuantity = 200, CategoryId = 2 },

new Product { ProductId = 4, Name = "Programming Book", Description = "Learn C# programming", Price = 49.99m, StockQuantity = 75, CategoryId = 3 }

);

}

}

}

[Program.cs](http://program.cs)

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using RetailInventory.Models;

Console.WriteLine("=== EF Core 8.0 Retail Inventory Demo ===\n");

var connectionString = "Server=(localdb)\\MSSQLLocalDB;Database=RetailInventoryDB;Trusted\_Connection=true;";

var options = new DbContextOptionsBuilder<RetailInventoryContext>()

.UseSqlServer(connectionString)

.Options;

using var context = new RetailInventoryContext(options);

await context.Database.EnsureCreatedAsync();

await DemoOperations(context);

Console.WriteLine("\nPress any key to exit...");

Console.ReadKey();

static async Task DemoOperations(RetailInventoryContext context)

{

Console.WriteLine("1. All Categories:");

var categories = await context.Categories.ToListAsync();

foreach (var category in categories)

{

Console.WriteLine($" - {category.Name}: {category.Description}");

}

Console.WriteLine("\n2. Products with Categories:");

var productsWithCategories = await context.Products

.Include(p => p.Category)

.ToListAsync();

foreach (var product in productsWithCategories)

{

Console.WriteLine($" - {product.Name} ({product.Category.Name}): ${product.Price:F2} - Stock: {product.StockQuantity}");

}

Console.WriteLine("\n3. Adding new product...");

var newProduct = new Product

{

Name = "Wireless Headphones",

Description = "Bluetooth wireless headphones",

Price = 79.99m,

StockQuantity = 30,

CategoryId = 1 // Electronics

};

context.Products.Add(newProduct);

await context.SaveChangesAsync();

Console.WriteLine($" Added: {newProduct.Name}");

Console.WriteLine("\n4. Updating stock quantity...");

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

if (laptopProduct != null)

{

laptopProduct.StockQuantity += 10;

await context.SaveChangesAsync();

Console.WriteLine($" Updated {laptopProduct.Name} stock to: {laptopProduct.StockQuantity}");

}

Console.WriteLine("\n5. Electronics Products:");

var electronicsProducts = await context.Products

.Where(p => p.Category.Name == "Electronics")

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

.ToListAsync();

foreach (var product in electronicsProducts)

{

Console.WriteLine($" - {product.Name}: ${product.Price:F2} (Stock: {product.StockQuantity})");

}

Console.WriteLine("\n6. Low Stock Alert (< 60 items):");

var lowStockProducts = await context.Products

.Where(p => p.StockQuantity < 60)

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

.ToListAsync();

foreach (var product in lowStockProducts)

{

Console.WriteLine($" - {product.Name}: {product.StockQuantity} items remaining");

}

Console.WriteLine("\n7. Categories with Product Count:");

var categoryStats = await context.Categories

.Select(c => new

{

c.Name,

ProductCount = c.Products.Count(),

TotalValue = c.Products.Sum(p => p.Price \* p.StockQuantity)

})

.ToListAsync();

foreach (var stat in categoryStats)

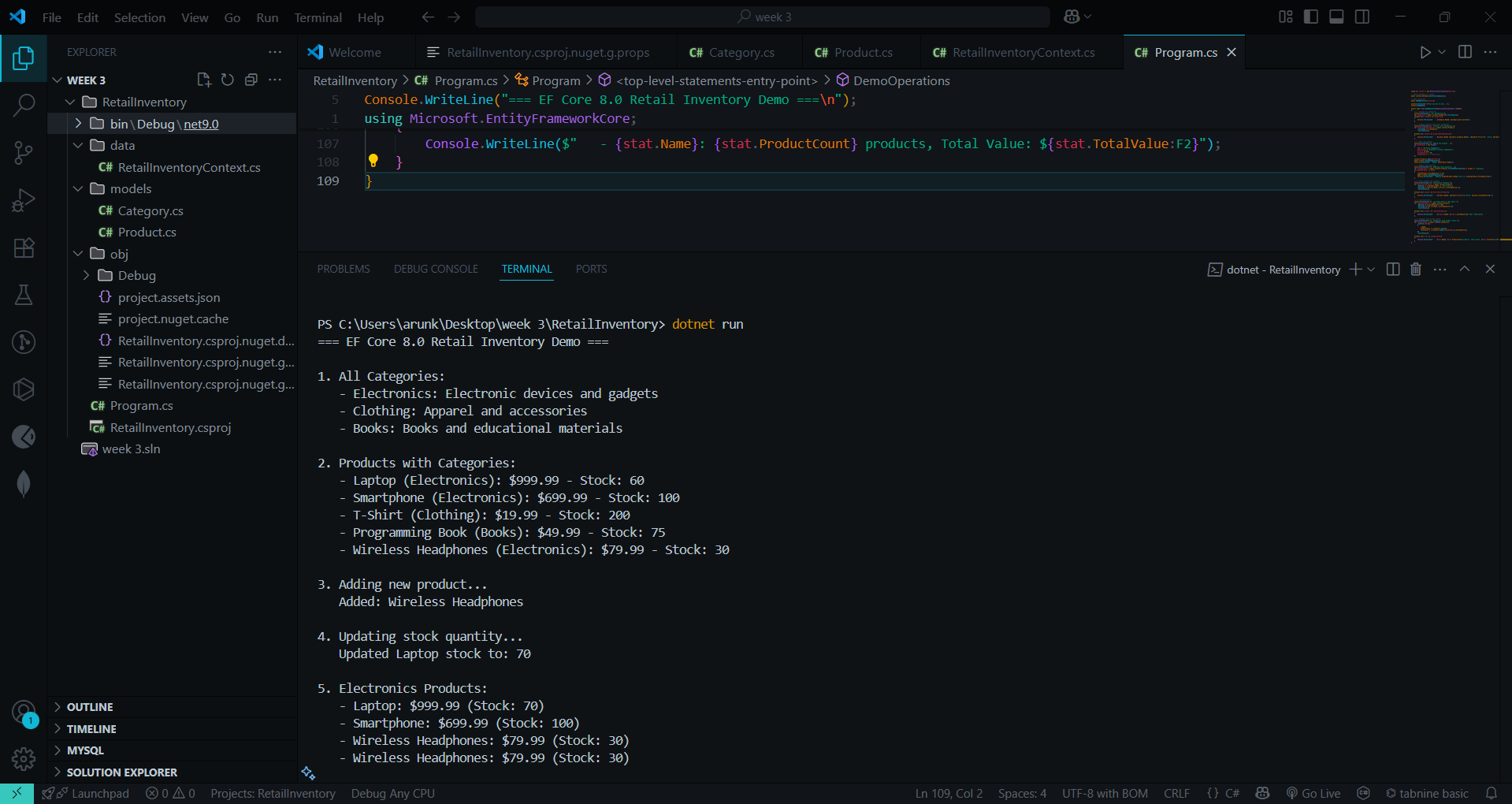
{

Console.WriteLine($" - {stat.Name}: {stat.ProductCount} products, Total Value: ${stat.TotalValue:F2}");

}

}

Output:



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

Program:

Models:

[Category.cs](http://category.cs)

using System.ComponentModel.DataAnnotations;

namespace RetailStore.Models

{

public class Category

{

public int Id { get; set; }

[Required]

[MaxLength(100)]

public string Name { get; set; } = string.Empty;

// Navigation property - One Category can have many Products

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

}

}

[Product.cs](http://product.cs)

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace RetailStore.Models

{

public class Product

{

public int Id { get; set; }

[Required]

[MaxLength(200)]

public string Name { get; set; } = string.Empty;

[Column(TypeName = "decimal(18,2)")]

public decimal Price { get; set; }

// Foreign key

public int CategoryId { get; set; }

// Navigation property - Each Product belongs to one Category

public Category Category { get; set; } = null!;

}

}

Data

[AppDbContext.cs](http://appdbcontext.cs)

using Microsoft.EntityFrameworkCore;

using RetailStore.Models;

namespace RetailStore.Data

{

public class AppDbContext : DbContext

{

// DbSet properties for each entity

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

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

// Constructor for dependency injection (recommended)

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

{

}

// Alternative: Configure connection string directly

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// Only configure if not already configured (for flexibility)

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=RetailStoreDB;Trusted\_Connection=true;");

}

}

// Configure entity relationships and constraints

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Configure Category entity

modelBuilder.Entity<Category>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.Name).IsRequired().HasMaxLength(100);

entity.HasIndex(e => e.Name).IsUnique(); // Unique category names

});

// Configure Product entity

modelBuilder.Entity<Product>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.Name).IsRequired().HasMaxLength(200);

entity.Property(e => e.Price).HasPrecision(18, 2);

// Configure foreign key relationship

entity.HasOne(d => d.Category)

.WithMany(p => p.Products)

.HasForeignKey(d => d.CategoryId)

.OnDelete(DeleteBehavior.Restrict); // Prevent cascade delete

});

// Seed initial data

SeedData(modelBuilder);

}

private void SeedData(ModelBuilder modelBuilder)

{

// Seed Categories

modelBuilder.Entity<Category>().HasData(

new Category { Id = 1, Name = "Electronics" },

new Category { Id = 2, Name = "Clothing" },

new Category { Id = 3, Name = "Books" },

new Category { Id = 4, Name = "Sports" }

);

// Seed Products

modelBuilder.Entity<Product>().HasData(

new Product { Id = 1, Name = "Laptop", Price = 999.99m, CategoryId = 1 },

new Product { Id = 2, Name = "Smartphone", Price = 699.99m, CategoryId = 1 },

new Product { Id = 3, Name = "T-Shirt", Price = 19.99m, CategoryId = 2 },

new Product { Id = 4, Name = "Jeans", Price = 49.99m, CategoryId = 2 },

new Product { Id = 5, Name = "Programming Book", Price = 39.99m, CategoryId = 3 },

new Product { Id = 6, Name = "Basketball", Price = 29.99m, CategoryId = 4 }

);

}

}

}

#### **Program.cs (Console Application)**

using Microsoft.EntityFrameworkCore;

using RetailStore.Data;

using RetailStore.Models;

Console.WriteLine("=== Retail Store Database Demo ===\n");

// Configure DbContext

var connectionString = "Server=(localdb)\\MSSQLLocalDB;Database=RetailStoreDB;Trusted\_Connection=true;";

var options = new DbContextOptionsBuilder<AppDbContext>()

.UseSqlServer(connectionString)

.EnableSensitiveDataLogging() // For development only

.Options;

using var context = new AppDbContext(options);

// Ensure database is created

await context.Database.EnsureCreatedAsync();

Console.WriteLine("Database created successfully!");

// Demo operations

await DemoOperations(context);

Console.WriteLine("\nPress any key to exit...");

Console.ReadKey();

static async Task DemoOperations(AppDbContext context)

{

// 1. Display all categories

Console.WriteLine("1. All Categories:");

var categories = await context.Categories.ToListAsync();

foreach (var category in categories)

{

Console.WriteLine($" - ID: {category.Id}, Name: {category.Name}");

}

// 2. Display products with their categories

Console.WriteLine("\n2. Products with Categories:");

var productsWithCategories = await context.Products

.Include(p => p.Category)

.ToListAsync();

foreach (var product in productsWithCategories)

{

Console.WriteLine($" - {product.Name} ({product.Category.Name}): ${product.Price:F2}");

}

// 3. Add a new product

Console.WriteLine("\n3. Adding new product...");

var newProduct = new Product

{

Name = "Wireless Mouse",

Price = 25.99m,

CategoryId = 1 // Electronics

};

context.Products.Add(newProduct);

await context.SaveChangesAsync();

Console.WriteLine($" Added: {newProduct.Name}");

// 4. Query products by category

Console.WriteLine("\n4. Electronics Products:");

var electronicsProducts = await context.Products

.Where(p => p.Category.Name == "Electronics")

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

.ToListAsync();

foreach (var product in electronicsProducts)

{

Console.WriteLine($" - {product.Name}: ${product.Price:F2}");

}

// 5. Update product price

Console.WriteLine("\n5. Updating product price...");

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

if (laptop != null)

{

laptop.Price = 899.99m;

await context.SaveChangesAsync();

Console.WriteLine($" Updated {laptop.Name} price to: ${laptop.Price:F2}");

}

// 6. Count products per category

Console.WriteLine("\n6. Products per Category:");

var categoryStats = await context.Categories

.Select(c => new

{

c.Name,

ProductCount = c.Products.Count(),

TotalValue = c.Products.Sum(p => p.Price)

})

.ToListAsync();

foreach (var stat in categoryStats)

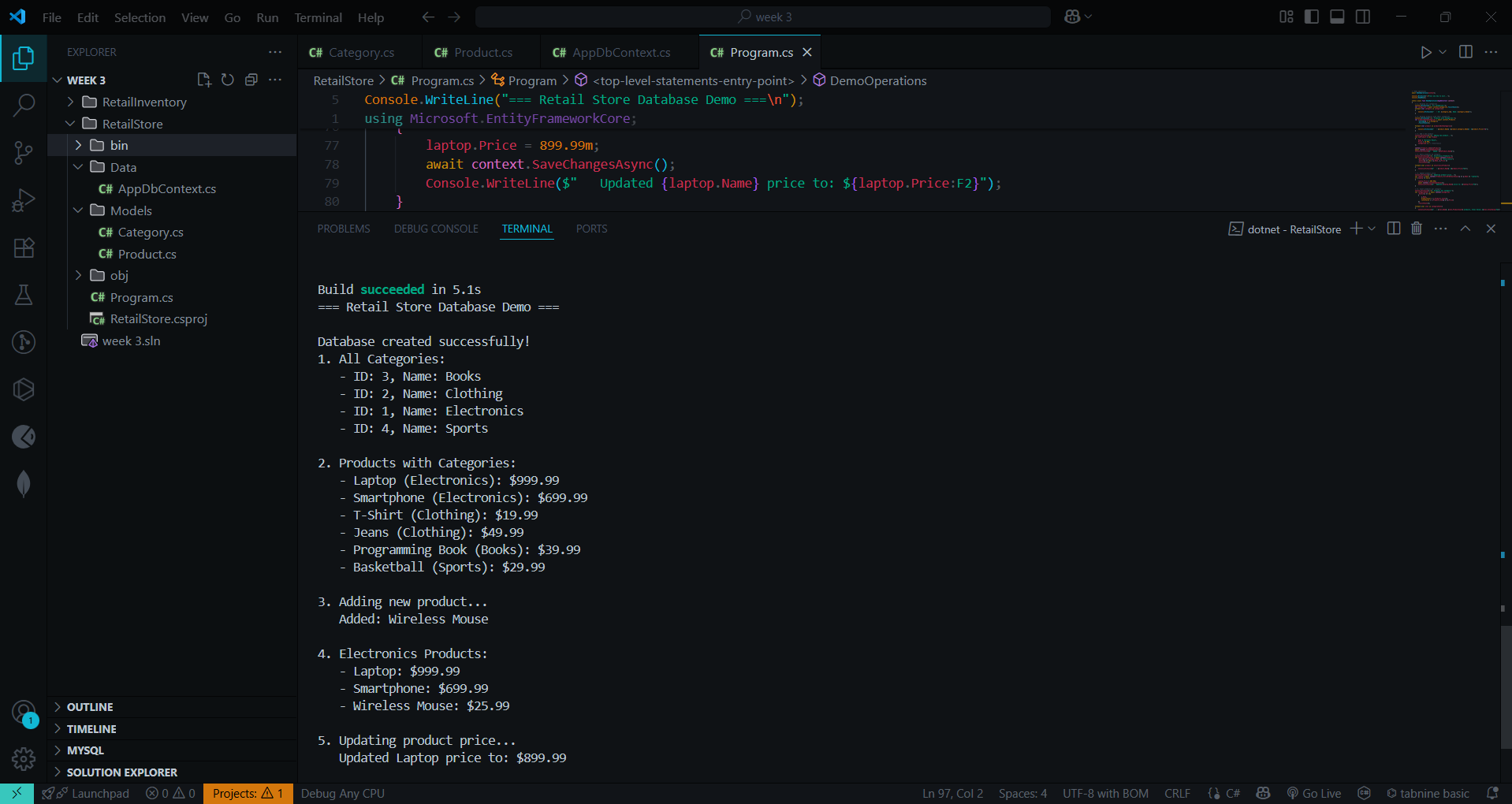
{

Console.WriteLine($" - {stat.Name}: {stat.ProductCount} products, Total Value: ${stat.TotalValue:F2}");

}

}

Output:



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

Programs

Models:

[Category.cs](http://category.cs)

using System.ComponentModel.DataAnnotations;

namespace RetailStore.Models

{

public class Category

{

public int Id { get; set; }

[Required]

[MaxLength(100)]

public string Name { get; set; } = string.Empty;

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

}

}

[Products.cs](http://products.cs)

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace RetailStore.Models

{

public class Product

{

public int Id { get; set; }

[Required]

[MaxLength(200)]

public string Name { get; set; } = string.Empty;

[Column(TypeName = "decimal(18,2)")]

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; } = null!;

}

}

Data:

[Appcontext.cs](http://appcontext.cs)

using Microsoft.EntityFrameworkCore;

using RetailStore.Models;

namespace RetailStore.Data

{

public class AppDbContext : DbContext

{

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

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

// Constructor for runtime use (with dependency injection)

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

{

}

// Parameterless constructor for EF Core CLI design-time operations

public AppDbContext() : base()

{

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// This will be used by EF Core CLI when no options are provided

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=RetailStoreDB;Trusted\_Connection=true;");

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Configure Category

modelBuilder.Entity<Category>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.Name).IsRequired().HasMaxLength(100);

entity.HasIndex(e => e.Name).IsUnique();

});

// Configure Product

modelBuilder.Entity<Product>(entity =>

{

entity.HasKey(e => e.Id);

entity.Property(e => e.Name).IsRequired().HasMaxLength(200);

entity.Property(e => e.Price).HasPrecision(18, 2);

entity.HasOne(d => d.Category)

.WithMany(p => p.Products)

.HasForeignKey(d => d.CategoryId)

.OnDelete(DeleteBehavior.Restrict);

});

// Seed data

modelBuilder.Entity<Category>().HasData(

new Category { Id = 1, Name = "Electronics" },

new Category { Id = 2, Name = "Clothing" },

new Category { Id = 3, Name = "Books" }

);

modelBuilder.Entity<Product>().HasData(

new Product { Id = 1, Name = "Laptop", Price = 999.99m, CategoryId = 1 },

new Product { Id = 2, Name = "Smartphone", Price = 699.99m, CategoryId = 1 },

new Product { Id = 3, Name = "T-Shirt", Price = 19.99m, CategoryId = 2 },

new Product { Id = 4, Name = "Programming Book", Price = 39.99m, CategoryId = 3 }

);

}

}

}

[Product.cs](http://product.cs)

public class AppDbContext : DbContext

{

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

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

public AppDbContext() { }

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

if (!optionsBuilder.IsConfigured)

{

optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=RetailStoreDB;Trusted\_Connection=true;");

}

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

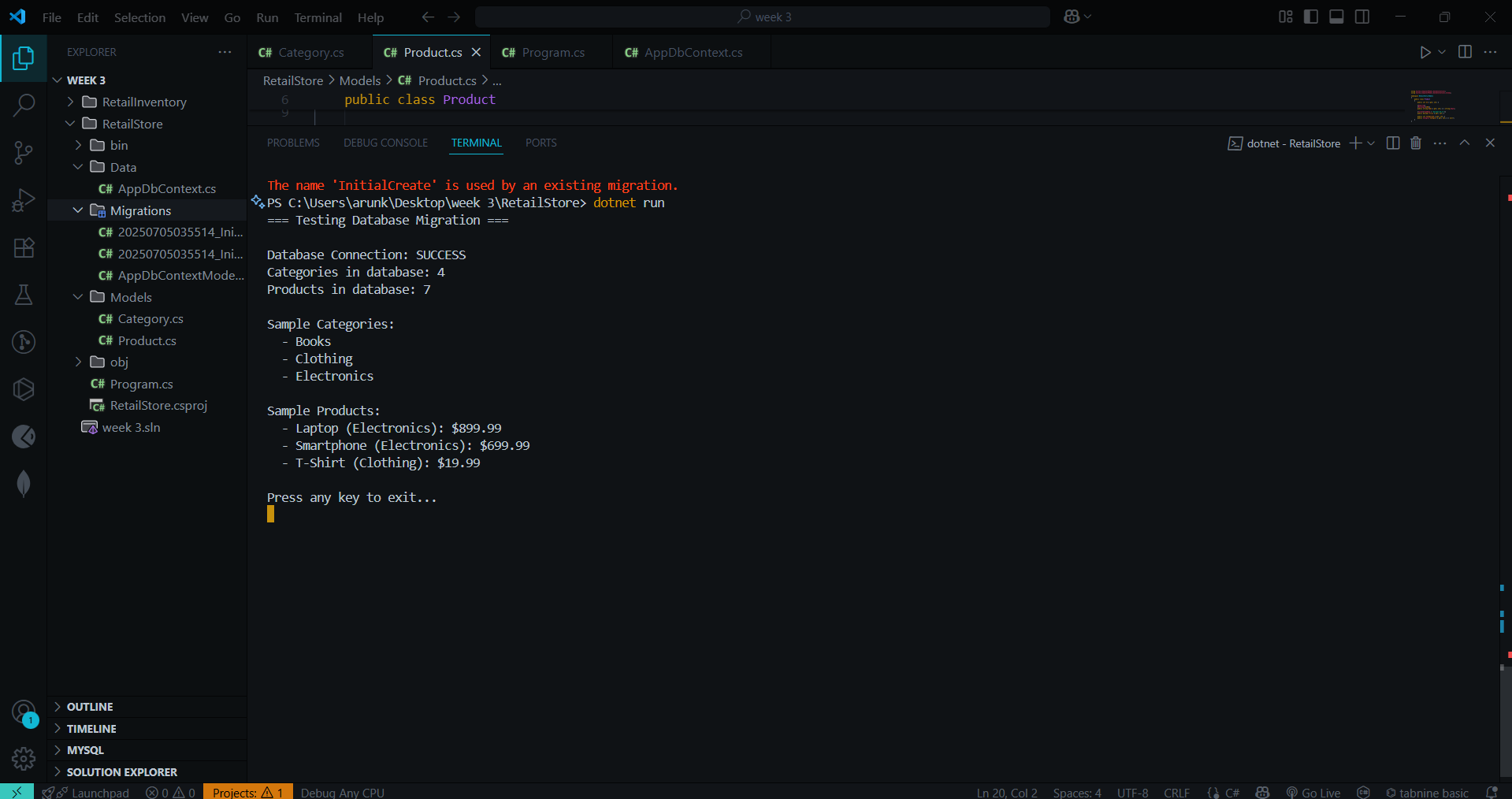
{

// your model configs and seed data

}

}

Output:



Lab 4: Inserting Initial Data into the Database   
  
Proograms:

Models:

[Category.cs](http://category.cs)

namespace Lab4Project.Models

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

// Navigation property

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

}

}

[Product.cs](http://product.cs)  
namespace Lab4Project.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

public decimal Price { get; set; }

public int CategoryId { get; set; }

// Navigation property

public Category Category { get; set; } = null!;

}

}

Data

[AppDbContext.cs](http://appdbcontext.cs)

using Microsoft.EntityFrameworkCore;

using Lab4Project.Models;

namespace Lab4Project.Data

{

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

// Try multiple connection string options

optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=StoreDB;Trusted\_Connection=true;MultipleActiveResultSets=true;");

}

}

}

Create appsettings.json  
{

"ConnectionStrings": {

"DefaultConnection": "Server=.;Database=StoreDB;Trusted\_Connection=true;TrustServerCertificate=true;"

}

}  
[Program.cs](http://program.cs)  
using Lab4Project.Data;

using Lab4Project.Models;

Console.WriteLine("Starting data insertion...");

using var context = new AppDbContext();

// Ensure database is created

await context.Database.EnsureCreatedAsync();

// Create categories

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

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

// Add categories to context

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

// Create products

var product1 = new Product

{

Name = "Laptop",

Price = 75000,

Category = electronics

};

var product2 = new Product

{

Name = "Rice Bag",

Price = 1200,

Category = groceries

};

// Add products to context

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

// Save all changes to database

await context.SaveChangesAsync();

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

Console.WriteLine($"Electronics Category ID: {electronics.Id}");

Console.WriteLine($"Groceries Category ID: {groceries.Id}");

Console.WriteLine($"Laptop Product ID: {product1.Id}");

Console.WriteLine($"Rice Bag Product ID: {product2.Id}")  
**Lab4Project.csproj**<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net8.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.0" />

<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.0" />

<PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="8.0.0" />

</ItemGroup>

<ItemGroup>

<None Update="appsettings.json">

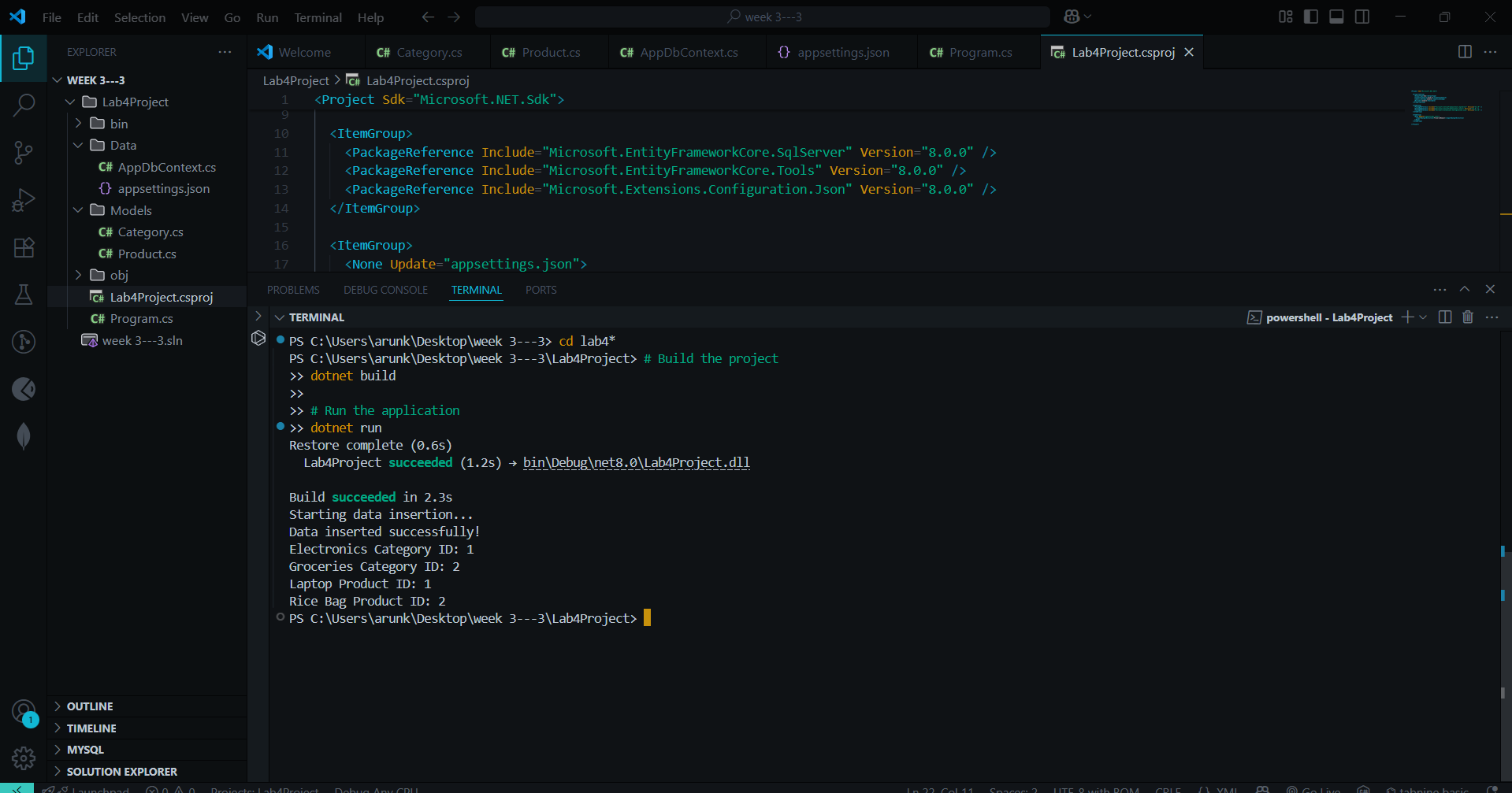
<CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>

</None>

</ItemGroup>

</Project>

Output:



LAB 5: Retrieving Data from the Database  
  
Programs

Models

[Category.cs](http://category.cs)

namespace Lab5Project.Models

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

// Navigation property

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

}

}

[Product.cs](http://product.cs)  
namespace Lab5Project.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

public decimal Price { get; set; }

public int CategoryId { get; set; }

// Navigation property

public Category Category { get; set; } = null!;

}

}

Data

[AppDbContext.cs](http://appdbcontext.cs)

using Microsoft.EntityFrameworkCore;

using Lab5Project.Models;

namespace Lab5Project.Data

{

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=StoreDB;Trusted\_Connection=true;MultipleActiveResultSets=true;");

}

}

}

[Program.cs](http://program.cs)

using Lab5Project.Data;

using Lab5Project.Models;

using Microsoft.EntityFrameworkCore;

Console.WriteLine("=== Lab 5: Retrieving Data from Database ===\n");

using var context = new AppDbContext();

// Ensure database is created and add sample data if empty

await EnsureDataExists(context);

Console.WriteLine("1. Retrieving All Products:");

Console.WriteLine("==========================");

// 1. Retrieve All Products using ToListAsync

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

foreach (var p in products)

{

Console.WriteLine($"{p.Name} - ₹{p.Price:N2}");

}

Console.WriteLine("\n2. Find Product by ID:");

Console.WriteLine("=====================");

// 2. Find by ID using FindAsync

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

if (product != null)

{

Console.WriteLine($"Found: {product.Name} - ₹{product.Price:N2}");

}

else

{

Console.WriteLine("Product with ID 1 not found");

}

Console.WriteLine("\n3. Find Expensive Product:");

Console.WriteLine("=========================");

// 3. FirstOrDefault with Condition (Fixed: removed extra 0 from 50000)

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

if (expensive != null)

{

Console.WriteLine($"Expensive: {expensive.Name} - ₹{expensive.Price:N2}");

}

else

{

Console.WriteLine("No expensive product found (> ₹50,000)");

}

Console.WriteLine("\n4. Additional Query Examples:");

Console.WriteLine("============================");

// 4. Get products with category information

var productsWithCategory = await context.Products

.Include(p => p.Category)

.ToListAsync();

Console.WriteLine("\nProducts with Categories:");

foreach (var p in productsWithCategory)

{

Console.WriteLine($"{p.Name} - ₹{p.Price:N2} (Category: {p.Category.Name})");

}

// 5. Count products

var productCount = await context.Products.CountAsync();

Console.WriteLine($"\nTotal Products: {productCount}");

// 6. Get products ordered by price

var orderedProducts = await context.Products

.OrderByDescending(p => p.Price)

.ToListAsync();

Console.WriteLine("\nProducts ordered by Price (High to Low):");

foreach (var p in orderedProducts)

{

Console.WriteLine($"{p.Name} - ₹{p.Price:N2}");

}

// 7. Get products by category

var electronicsProducts = await context.Products

.Where(p => p.Category.Name == "Electronics")

.ToListAsync();

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

foreach (var p in electronicsProducts)

{

Console.WriteLine($"{p.Name} - ₹{p.Price:N2}");

}

Console.WriteLine("\n=== End of Lab 5 ===");

// Helper method to ensure data exists

static async Task EnsureDataExists(AppDbContext context)

{

// Ensure database is created

await context.Database.EnsureCreatedAsync();

// Check if data already exists

if (await context.Products.AnyAsync())

{

Console.WriteLine("Data already exists in database.\n");

return;

}

Console.WriteLine("Adding sample data to database...\n");

// Create categories

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

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

var books = new Category { Name = "Books" };

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

// Create products

var products = new List<Product>

{

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

new Product { Name = "Smartphone", Price = 25000, Category = electronics },

new Product { Name = "Tablet", Price = 30000, Category = electronics },

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

new Product { Name = "Cooking Oil", Price = 350, Category = groceries },

new Product { Name = "Programming Book", Price = 800, Category = books },

new Product { Name = "Novel", Price = 450, Category = books }

};

await context.Products.AddRangeAsync(products);

await context.SaveChangesAsync();

Console.WriteLine("Sample data added successfully!\n");

}

Lab5Project.csproj  
<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net8.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="8.0.0" />

<PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="8.0.0" />

<PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="8.0.0" />

</ItemGroup>

<ItemGroup>

<None Update="appsettings.json">

<CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>

</None>

</ItemGroup>

</Project>

Output:

