**Entity Framework Core 8.0**

**EF Core 8.0 HOL**

**Lab 1: Understanding ORM with a Retail Inventory System**

ORM (Object-Relational Mapping) allows developers to work with a database using object-oriented programming concepts instead of writing raw SQL.  
Entity Framework Core is a .NET ORM that maps C# classes to database tables. It enables CRUD operations through LINQ queries and generates SQL behind the scenes

**Lab 2 – Setting Up the Database Context**

**Code:**

using Microsoft.EntityFrameworkCore;

public class RetailContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder options)

{

options.UseSqlServer("Server=(localdb)\\mssqllocaldb;Database=RetailDb;Trusted\_Connection=True;");

}

}

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; }

}

public class Category

{

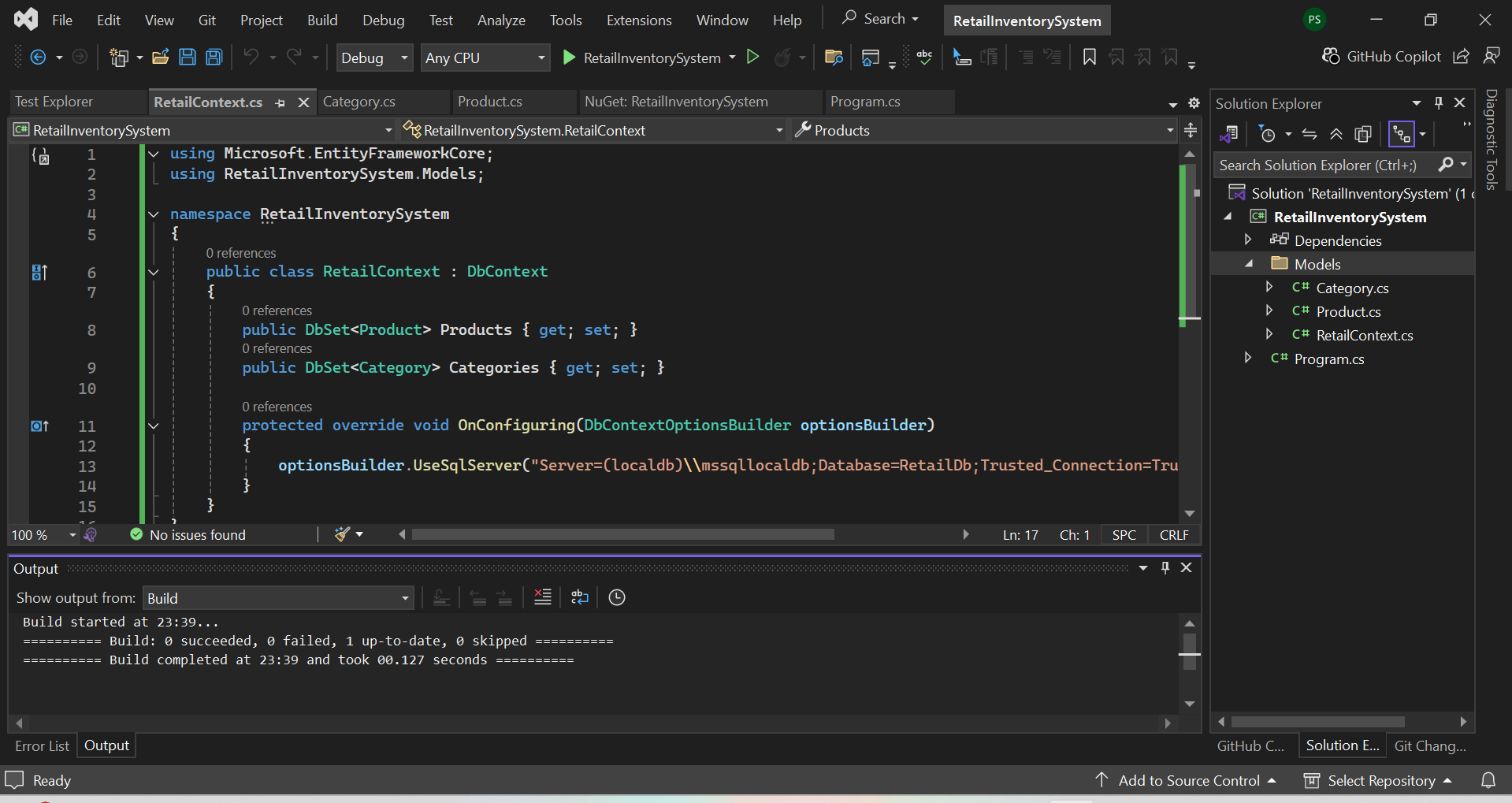
public int Id { get; set; }

public string Name { get; set; }

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

}

**OUTPUT:**

****

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

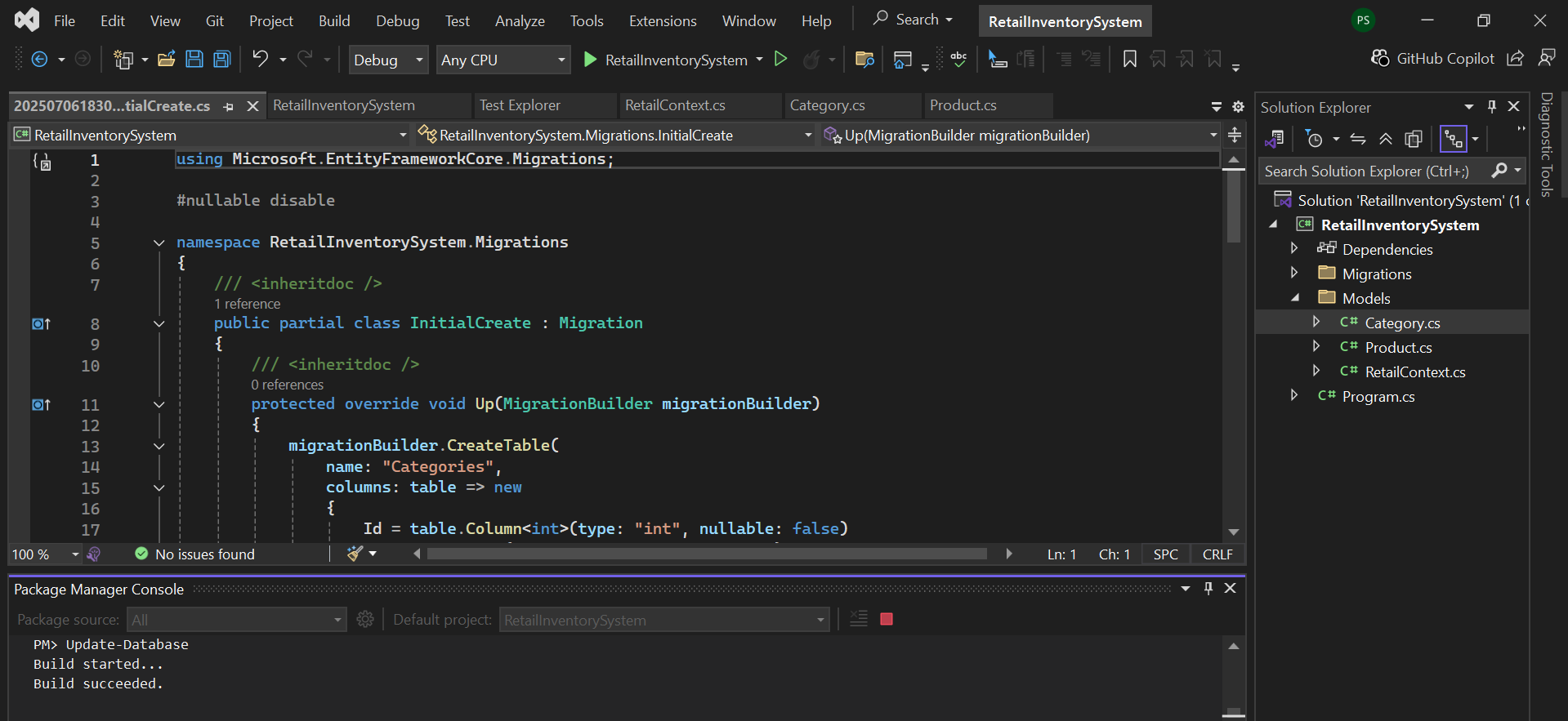
**Commands Used:**

Add-Migration InitialCreate

Update-Database

**Ans -** I used the EF Core CLI to create the database schema from the model classes. The first command generated migration files, and the second one applied the migration to the SQL Server LocalDB to create the tables.

**OUTPUT:**

****

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

**Code (Program.cs):**

using RetailInventorySystem.Models;

internal class Program

{

static void Main(string[] args)

{

using (var context = new RetailContext())

{

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

var smartphone = new Product

{

Name = "Smartphone",

Price = 19999.99M,

Category = electronics

};

context.Categories.Add(electronics);

context.Products.Add(smartphone);

context.SaveChanges();

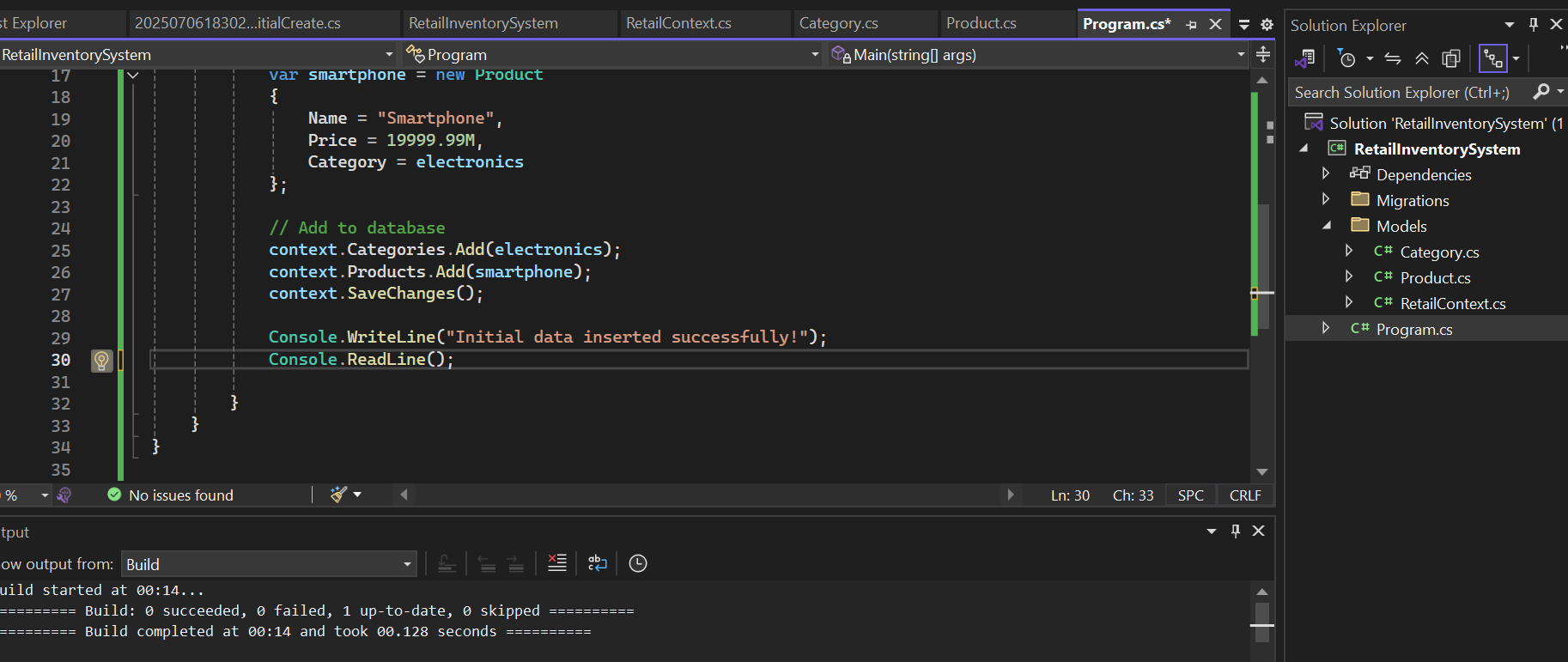
Console.WriteLine("Initial data inserted successfully!");

}

}

}

**OUTPUT:**



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

**Code (Program.cs):**

using Microsoft.EntityFrameworkCore;

using RetailInventorySystem.Models;

internal class Program

{

static void Main(string[] args)

{

using (var context = new RetailContext())

{

var products = context.Products

.Include(p => p.Category)

.ToList();

foreach (var product in products)

{

Console.WriteLine($"Product: {product.Name}, Price: {product.Price}, Category: {product.Category.Name}");

}

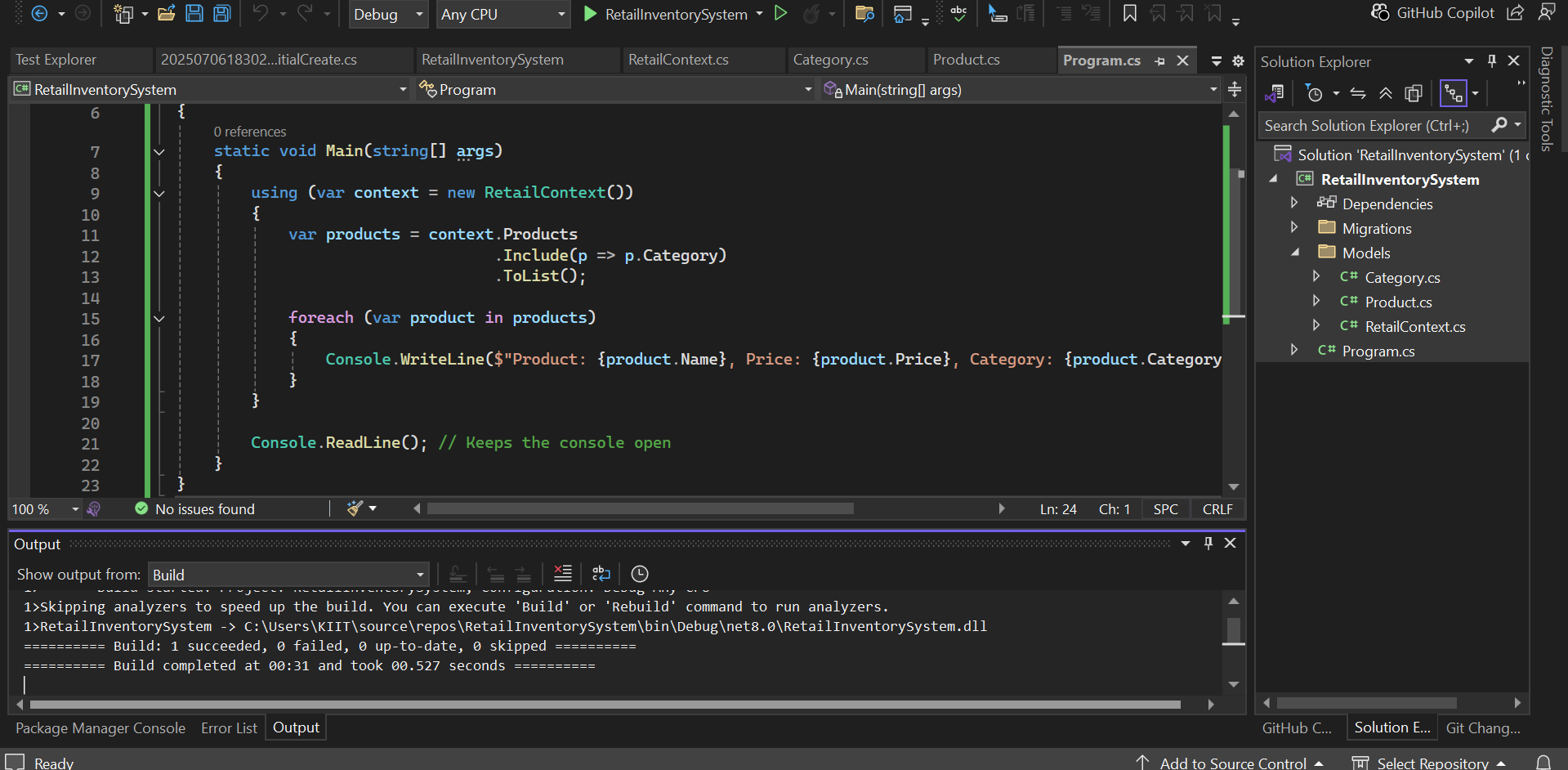
}

Console.ReadLine();

}

}

**OUTPUT:**



**Lab 6: Updating and Deleting Records**

**Code (Program.cs):**

using RetailInventorySystem.Models;

using Microsoft.EntityFrameworkCore;

internal class Program

{

static void Main(string[] args)

{

using (var context = new RetailContext())

{

var productToUpdate = context.Products.FirstOrDefault(p => p.Name == "Smartphone");

if (productToUpdate != null)

{

productToUpdate.Price = 17999.99M;

context.SaveChanges();

Console.WriteLine($"Updated Product Price: {productToUpdate.Name} => ₹{productToUpdate.Price}");

}

else

{

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

}

var productToDelete = context.Products.FirstOrDefault(p => p.Name == "Smartphone");

if (productToDelete != null)

{

context.Products.Remove(productToDelete);

context.SaveChanges();

Console.WriteLine($"Deleted Product: {productToDelete.Name}");

}

else

{

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

}

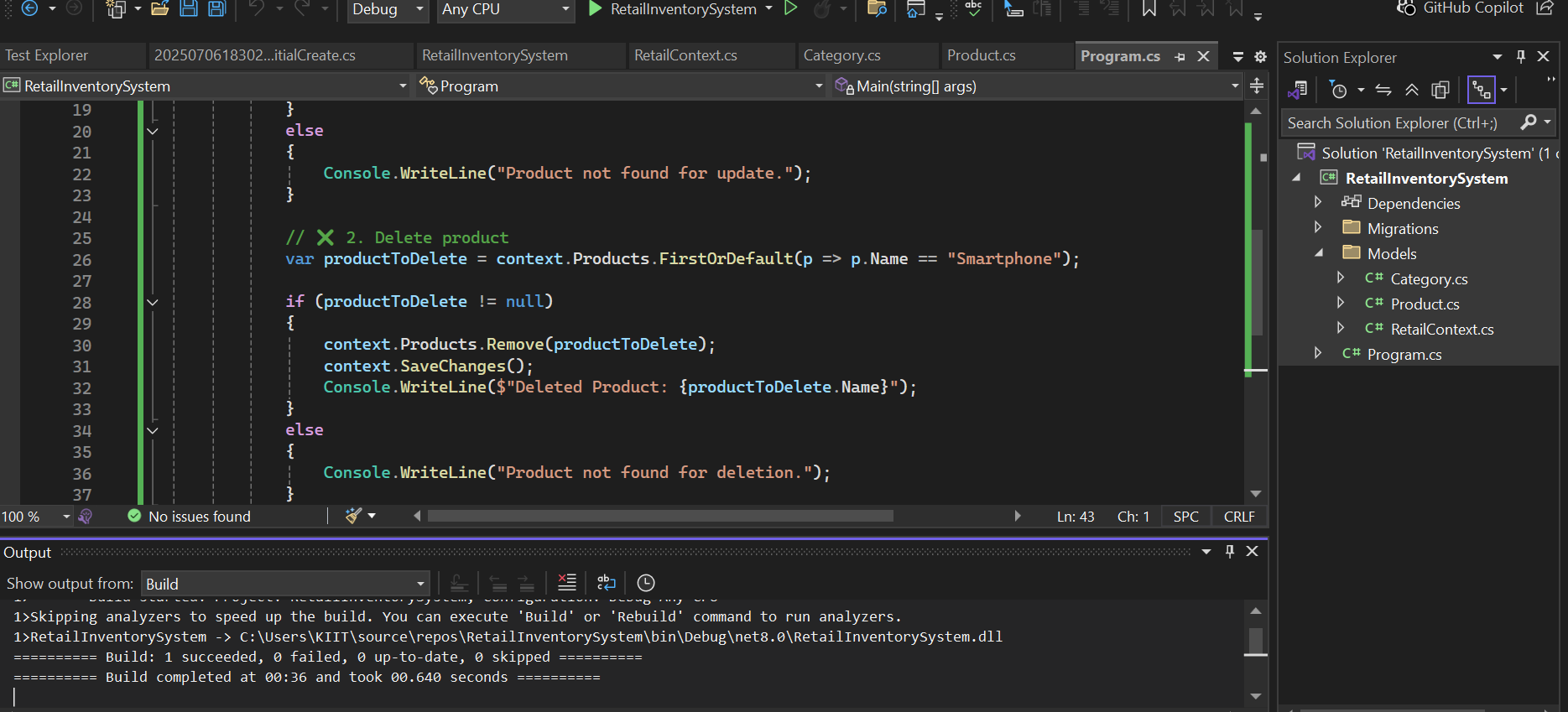
}

Console.ReadLine();

}

}

**OUTPUT:**

****

**Lab 7: Writing Queries with LINQ**

**Code (Program.cs):**

using RetailInventorySystem.Models;

using Microsoft.EntityFrameworkCore;

using System.Linq;

internal class Program

{

static void Main(string[] args)

{

using (var context = new RetailContext())

{

var expensiveProducts = context.Products

.Include(p => p.Category)

.Where(p => p.Price > 10000)

.ToList();

Console.WriteLine("Products with price > ₹10,000:\n");

foreach (var product in expensiveProducts)

{

Console.WriteLine($"Product: {product.Name}, Price: {product.Price}, Category: {product.Category.Name}");

}

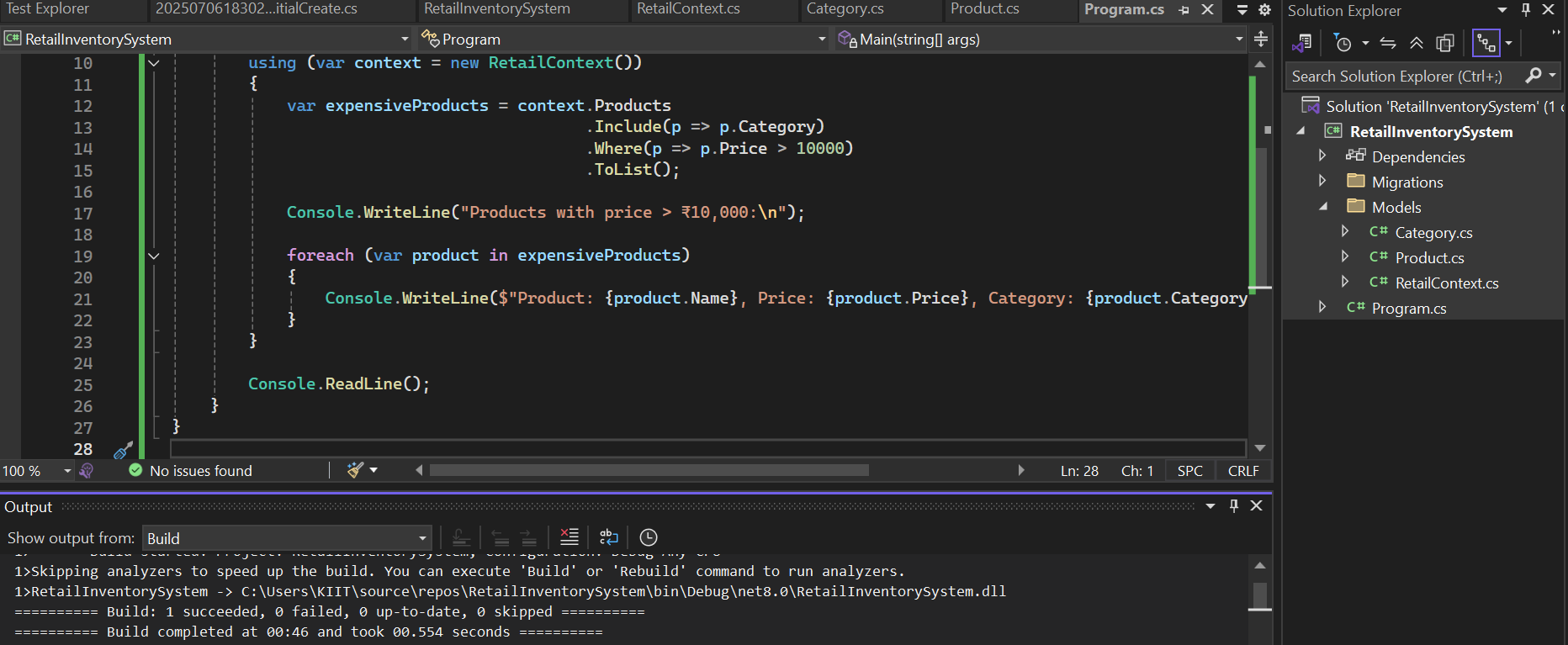
}

Console.ReadLine();

}

}

**OUTPUT:**

****