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

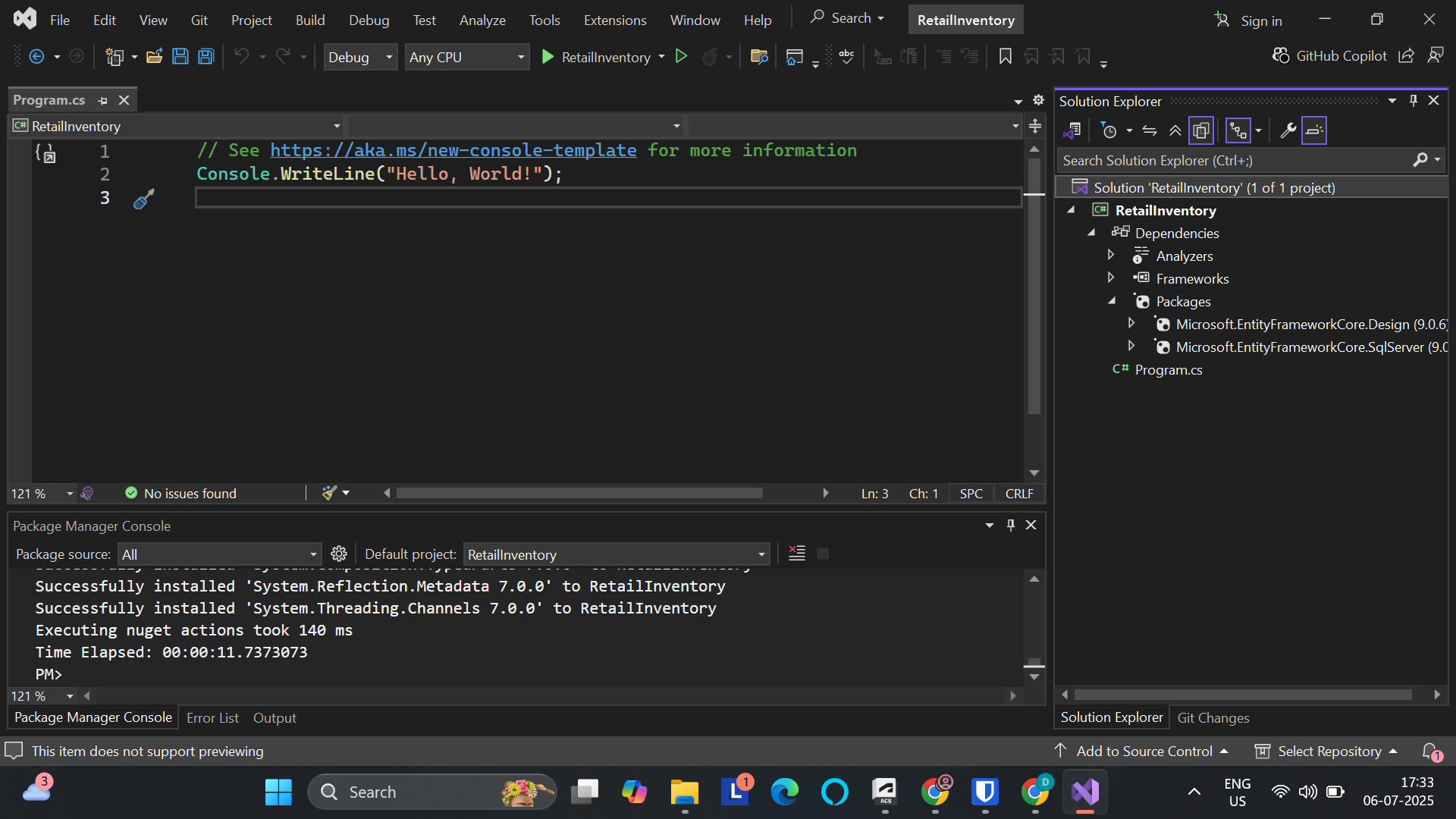
**Code:**

//To install packages via nuget manager:

Install-Package Microsoft.EntityFrameworkCore.SqlServer

Install-Package Microsoft.EntityFrameworkCore.Design

**Output:**

****

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

**Code:**

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

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RetailInventory.Models

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

}

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

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

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; }

}

}

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

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

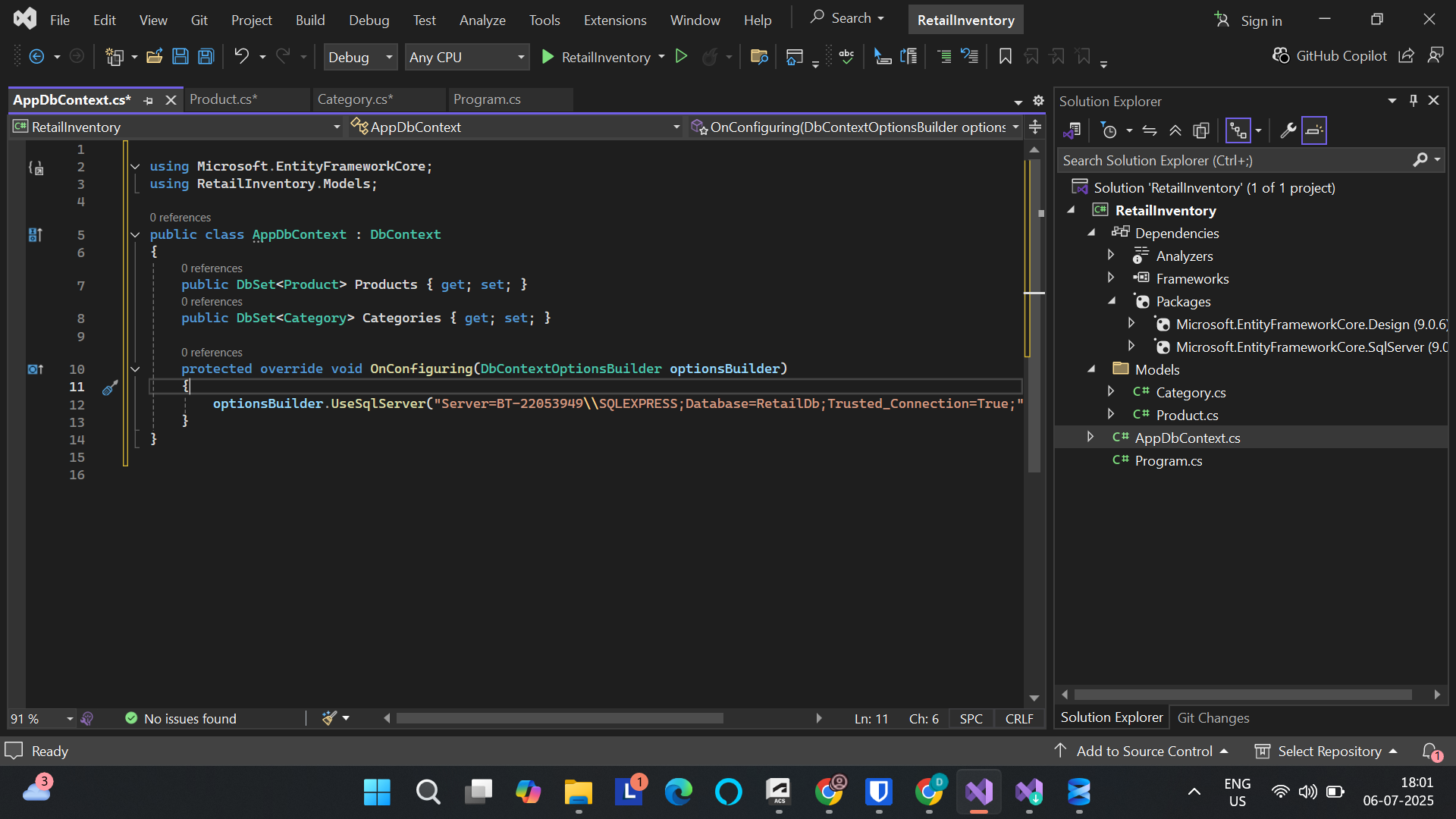
{

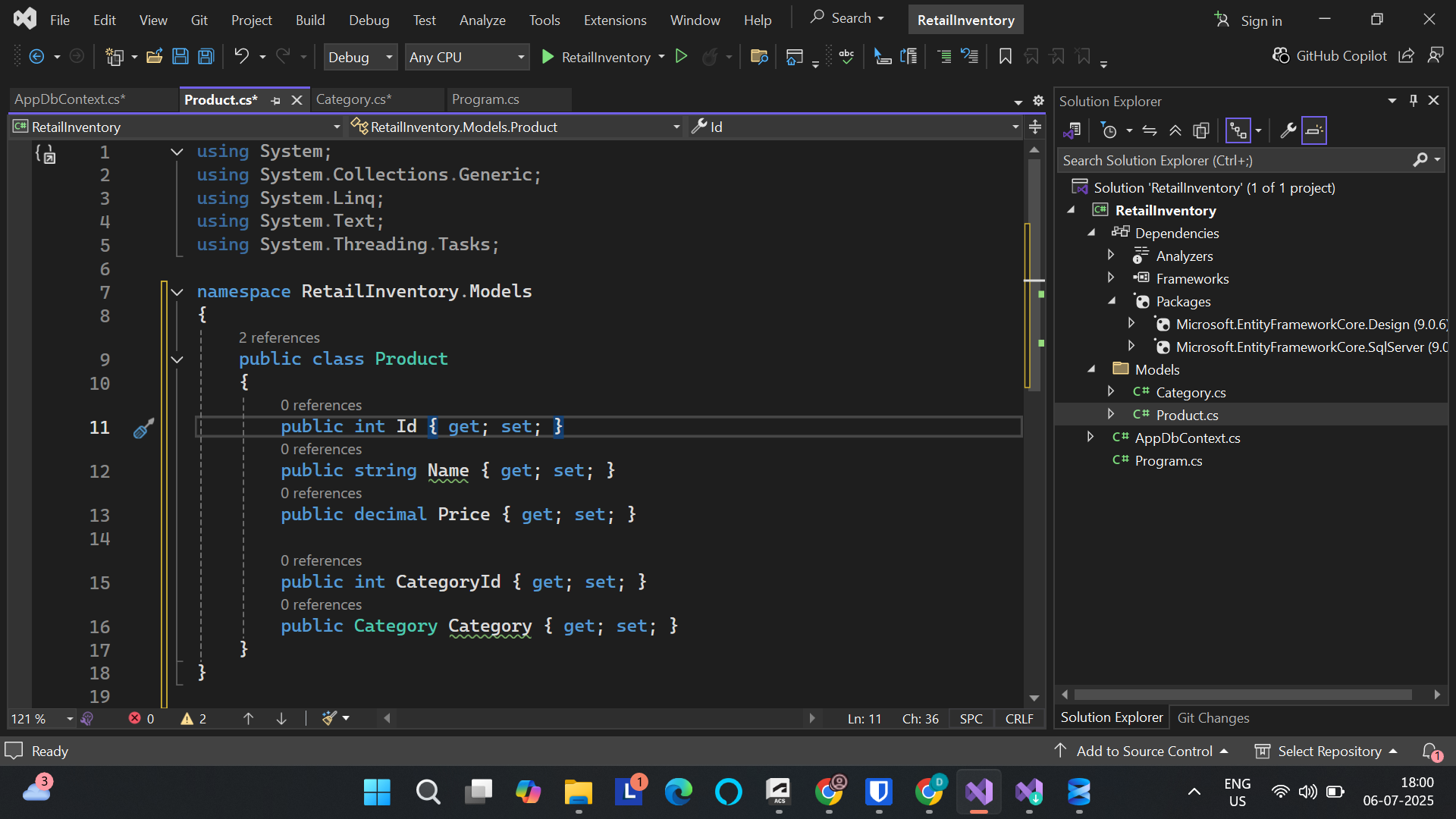
optionsBuilder.UseSqlServer("Server=BT-22053949\\SQLEXPRESS;Database=RetailDb;Trusted\_Connection=True;");

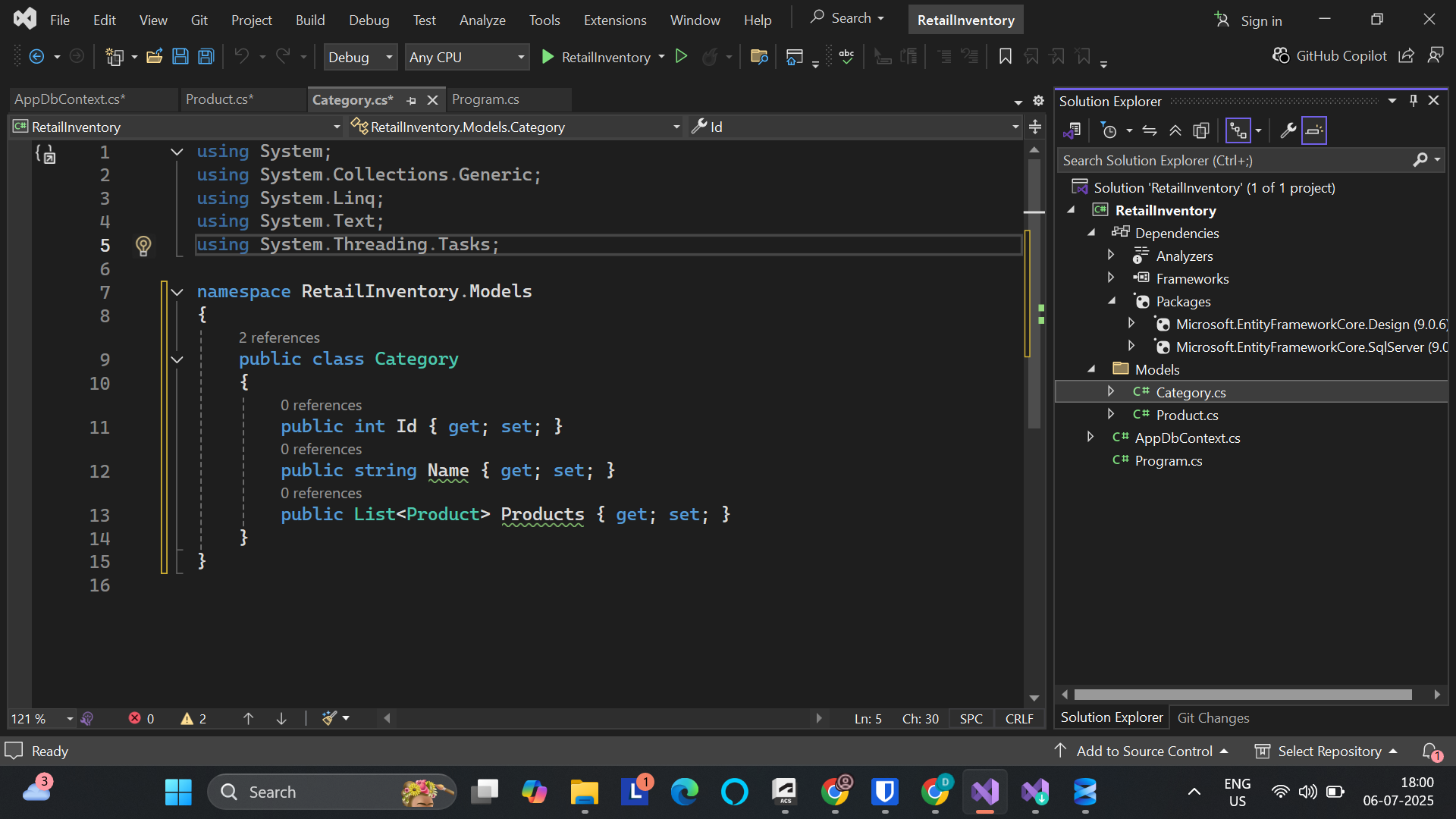
}

}

**Output:**

****

****

****

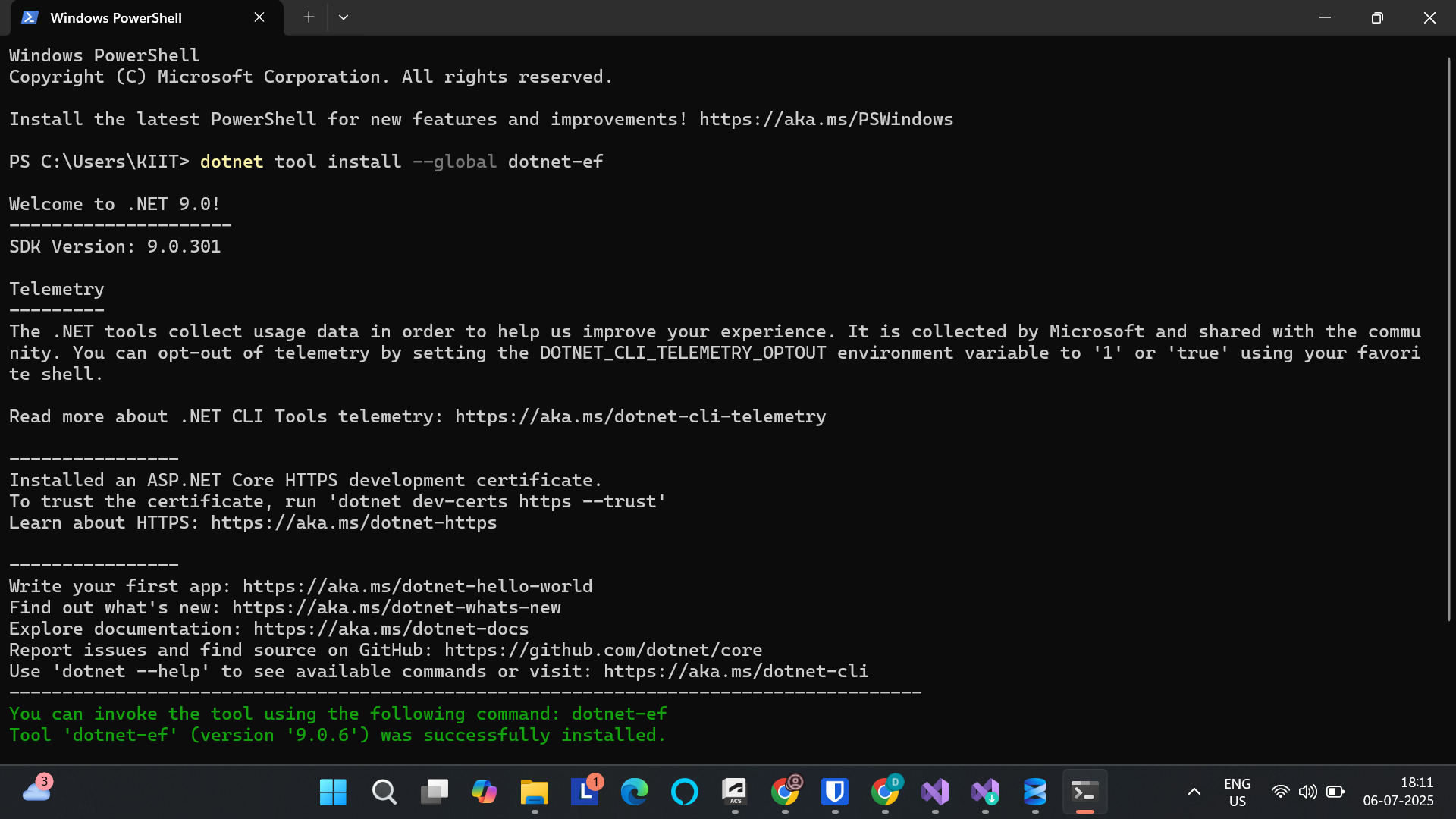
**Lab 3: Using EF Core CLI to Create and Apply Migrations**

**Code:**

1.Installing EF core cli:

dotnet tool install --global dotnet-ef

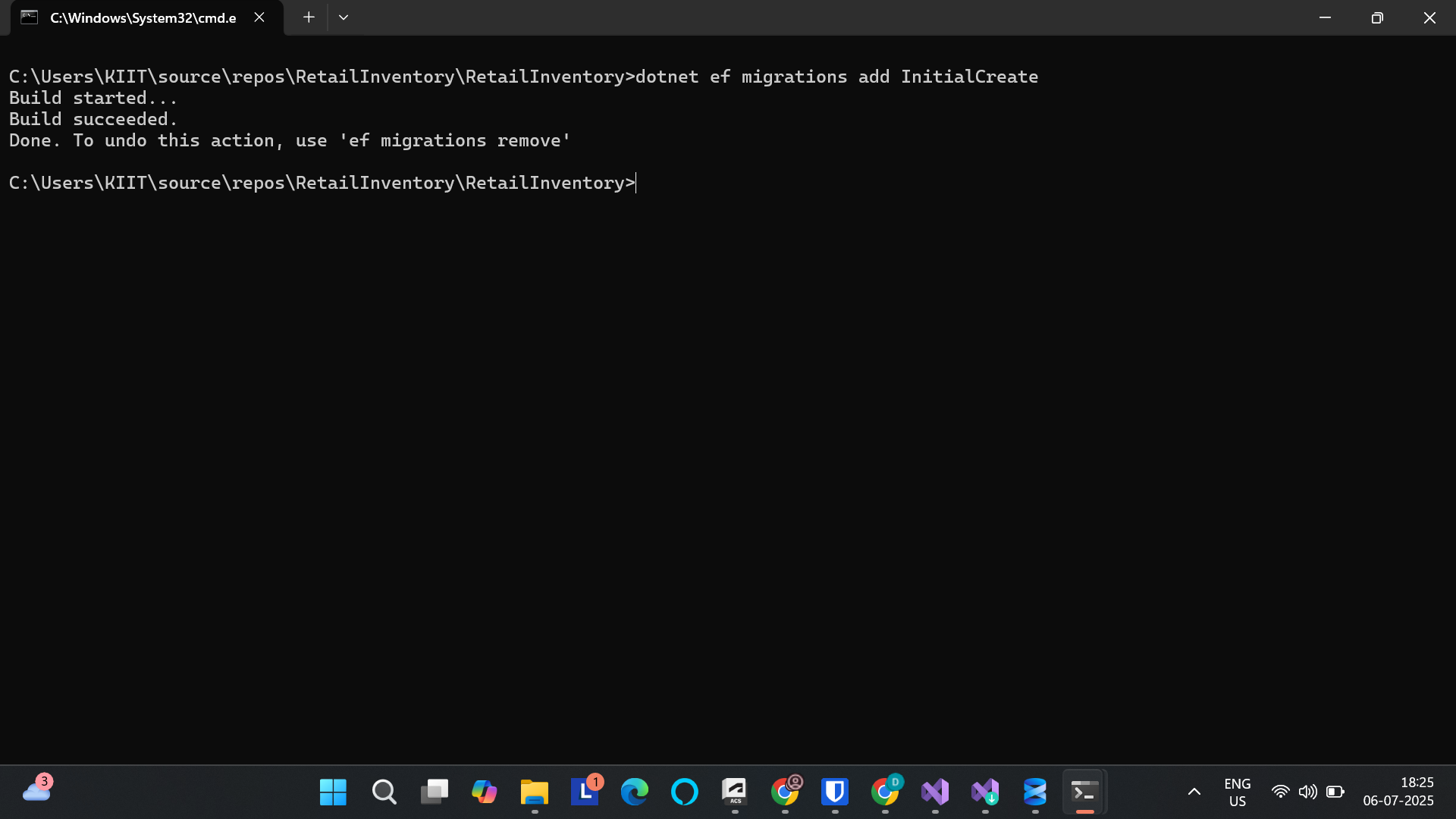
**Output:**

****

2.Create the First Migration:

dotnet ef migrations add InitialCreate

**Output:**

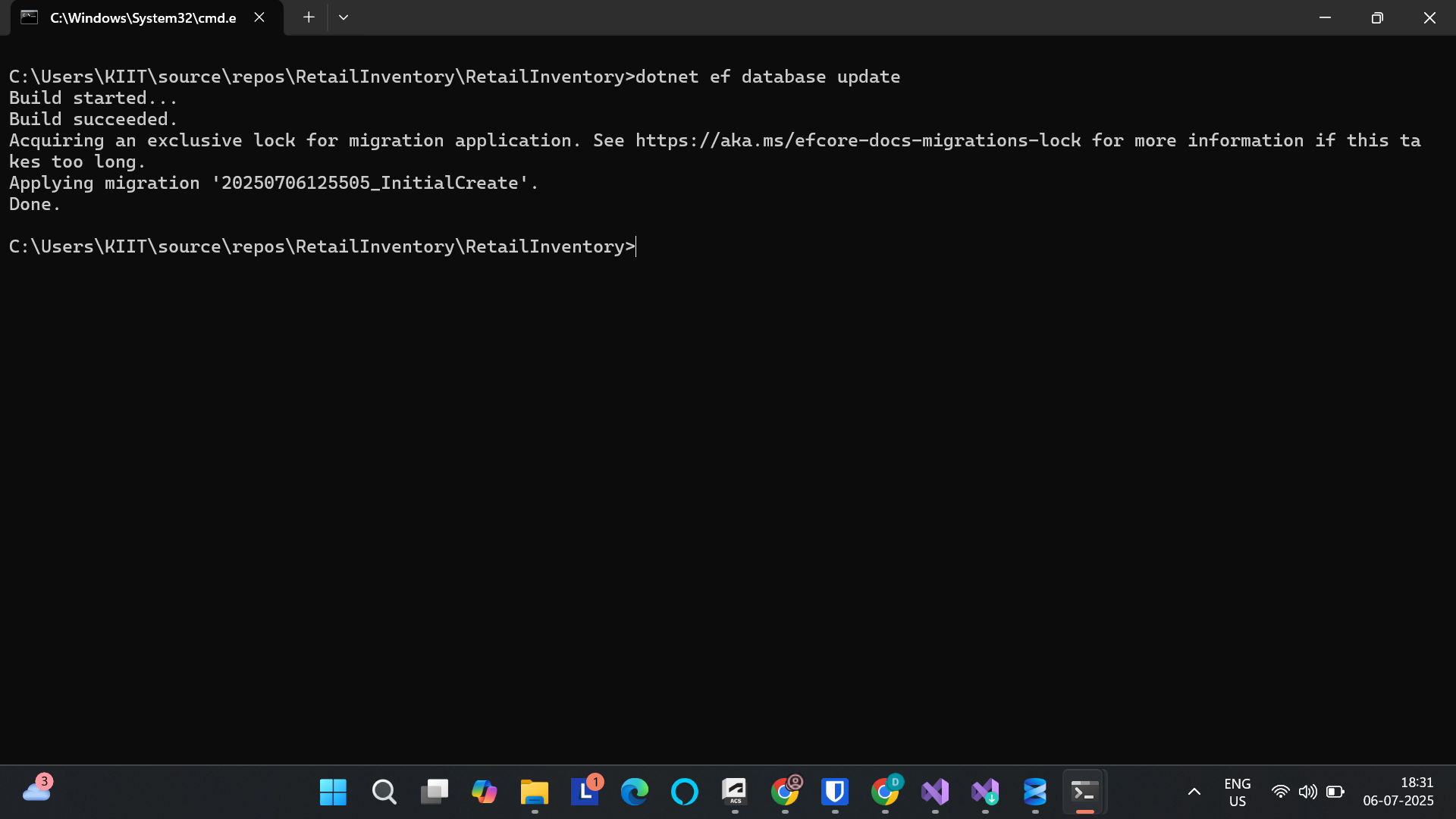
****

3: Apply the Migration to Create the Database

**Code:**

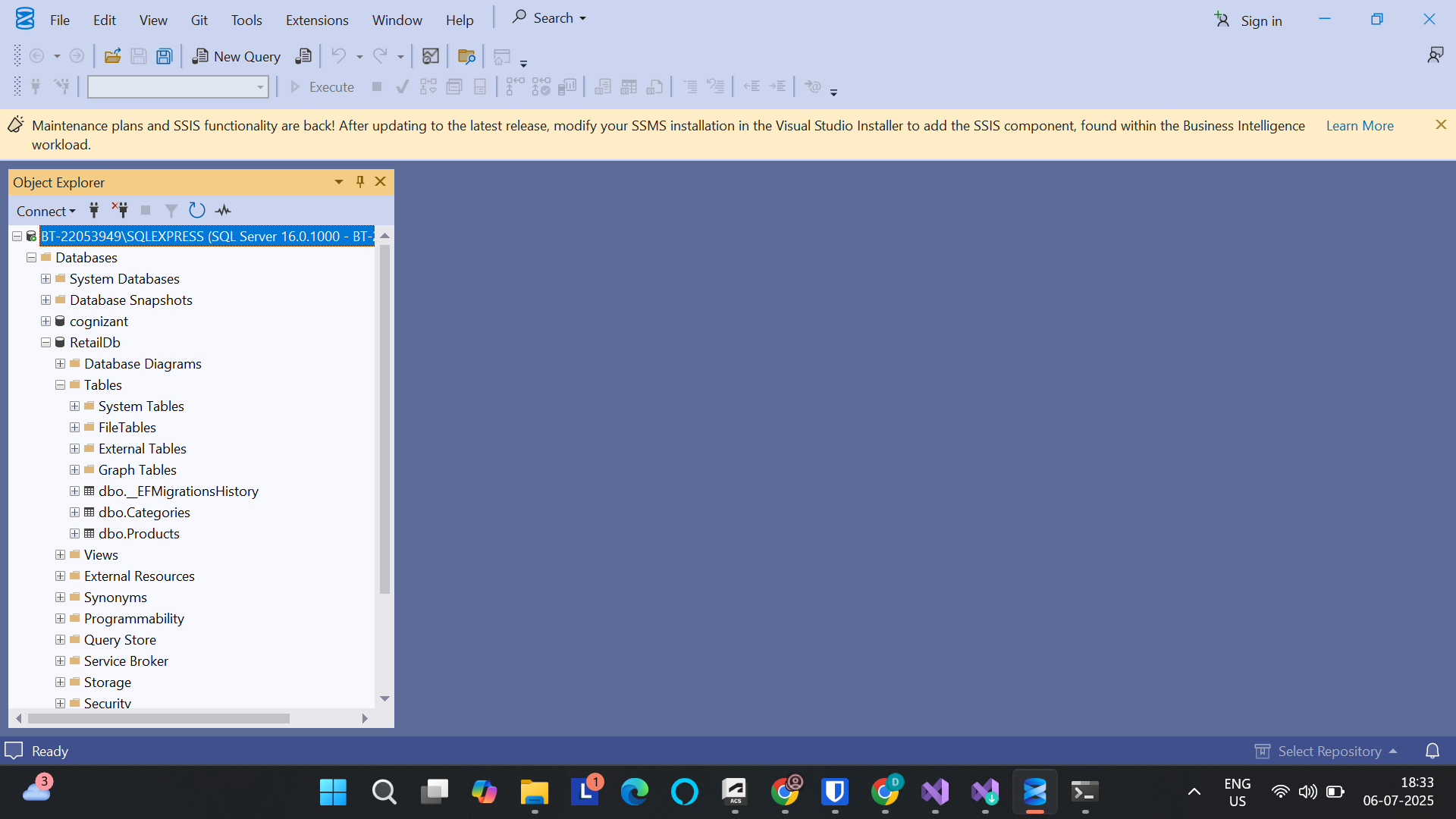
dotnet ef database update

**Output:**

****

4. Verify in SQL Server:

**Output:**

****

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

**Code:**

using RetailInventory.Models;

using var context = new AppDbContext();

// Create categories

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

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

// Add categories to database

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

// Create products linked to categories

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

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

// Add products to database

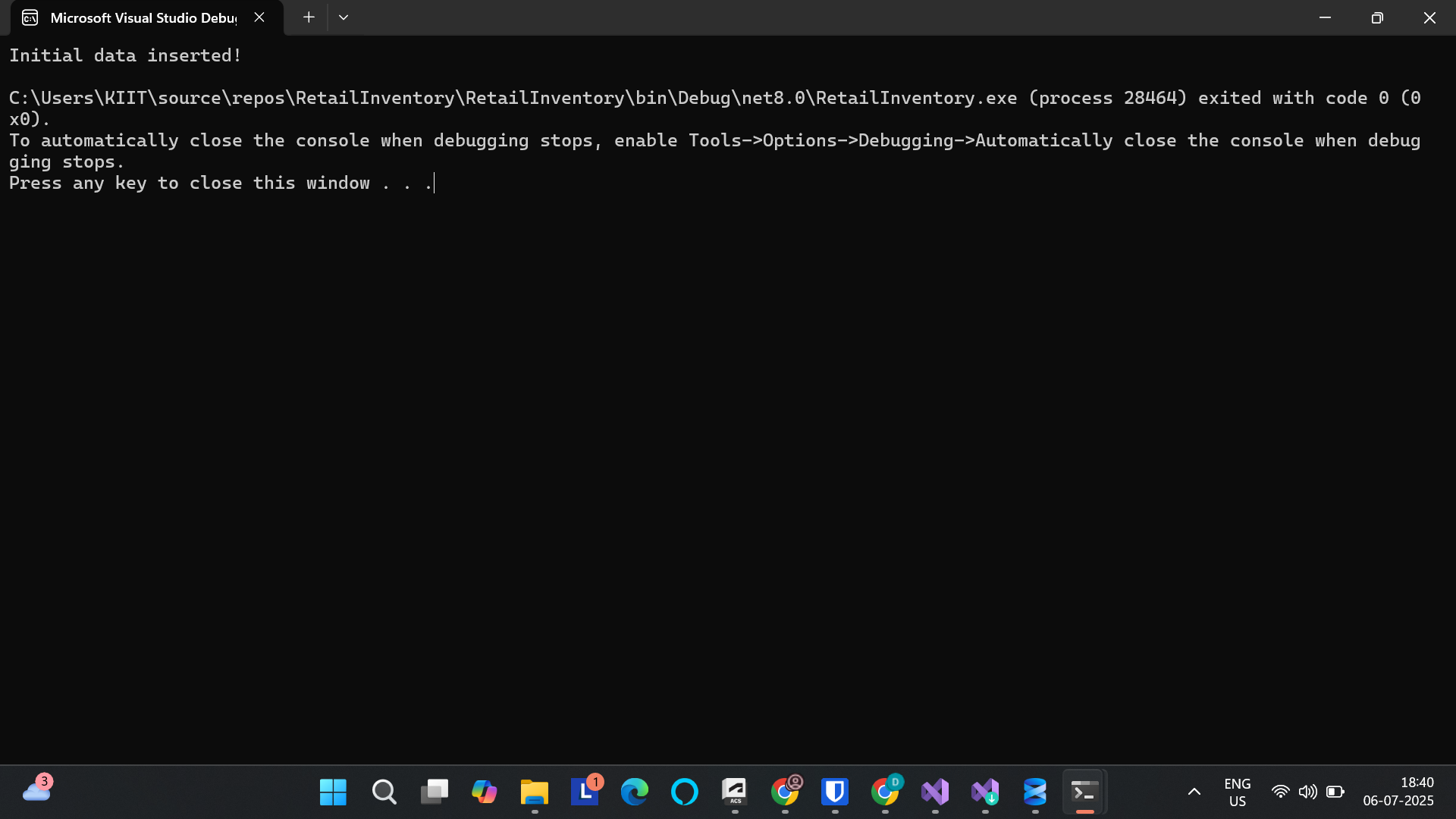
await context.Products.AddRangeAsync(product1, product2);

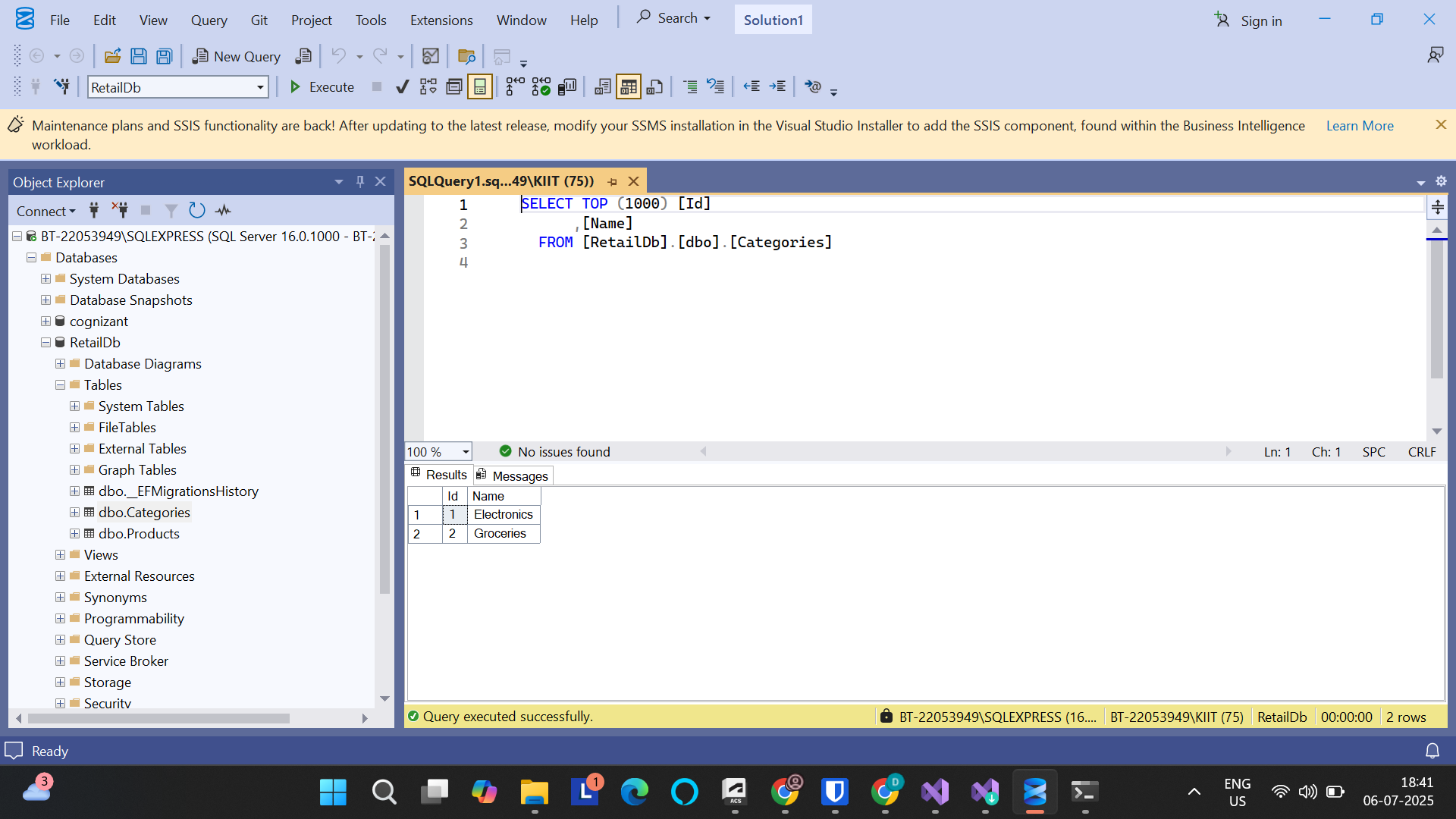
// Save everything

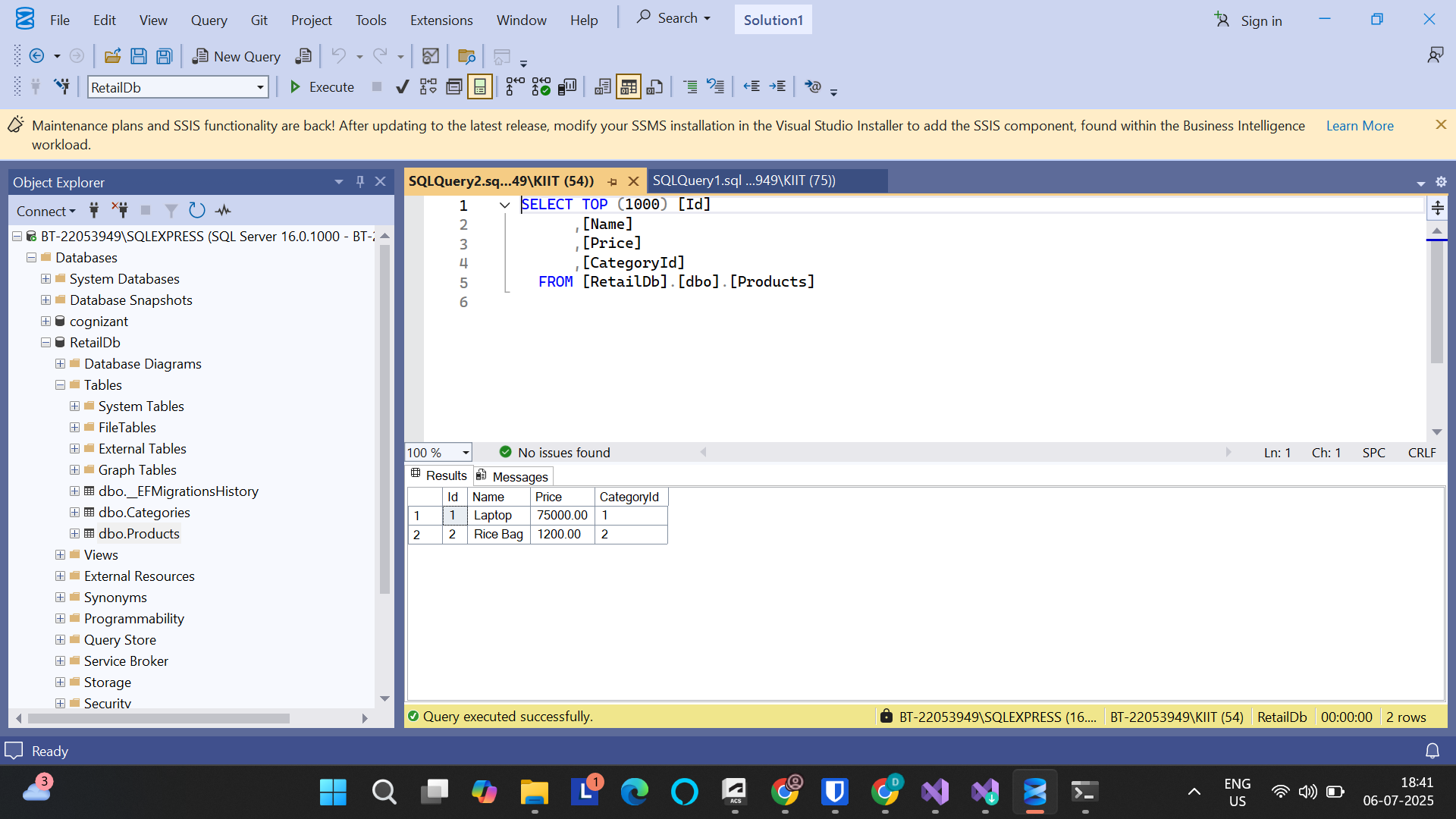
await context.SaveChangesAsync();

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

**Output:**







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

1: Retrieve All Products

**Code:**

using RetailInventory.Models;

using Microsoft.EntityFrameworkCore;

using var context = new AppDbContext();

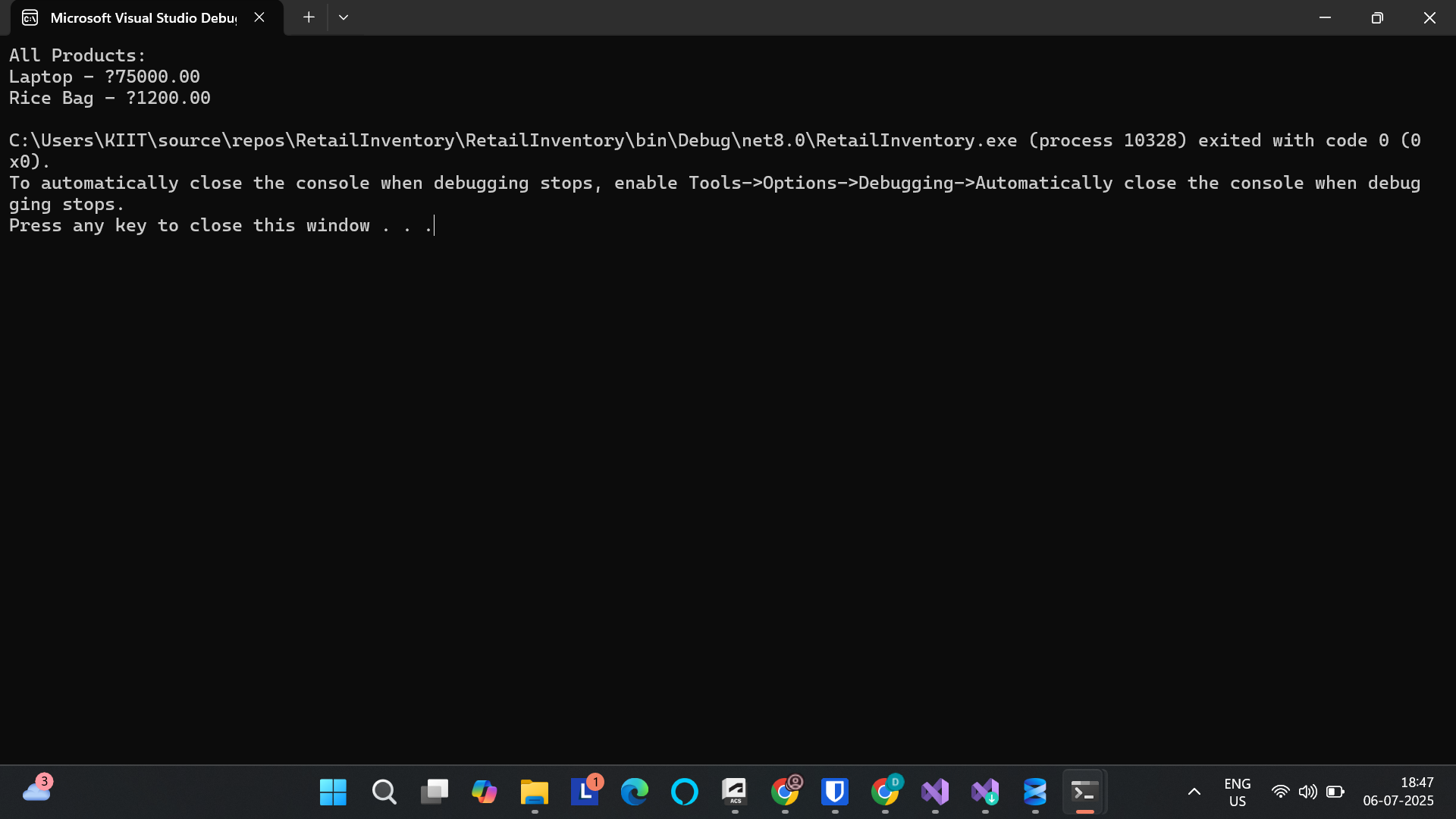
var products = await context.Products.ToListAsync();

Console.WriteLine("All Products:");

foreach (var p in products)

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

**Output:**

****

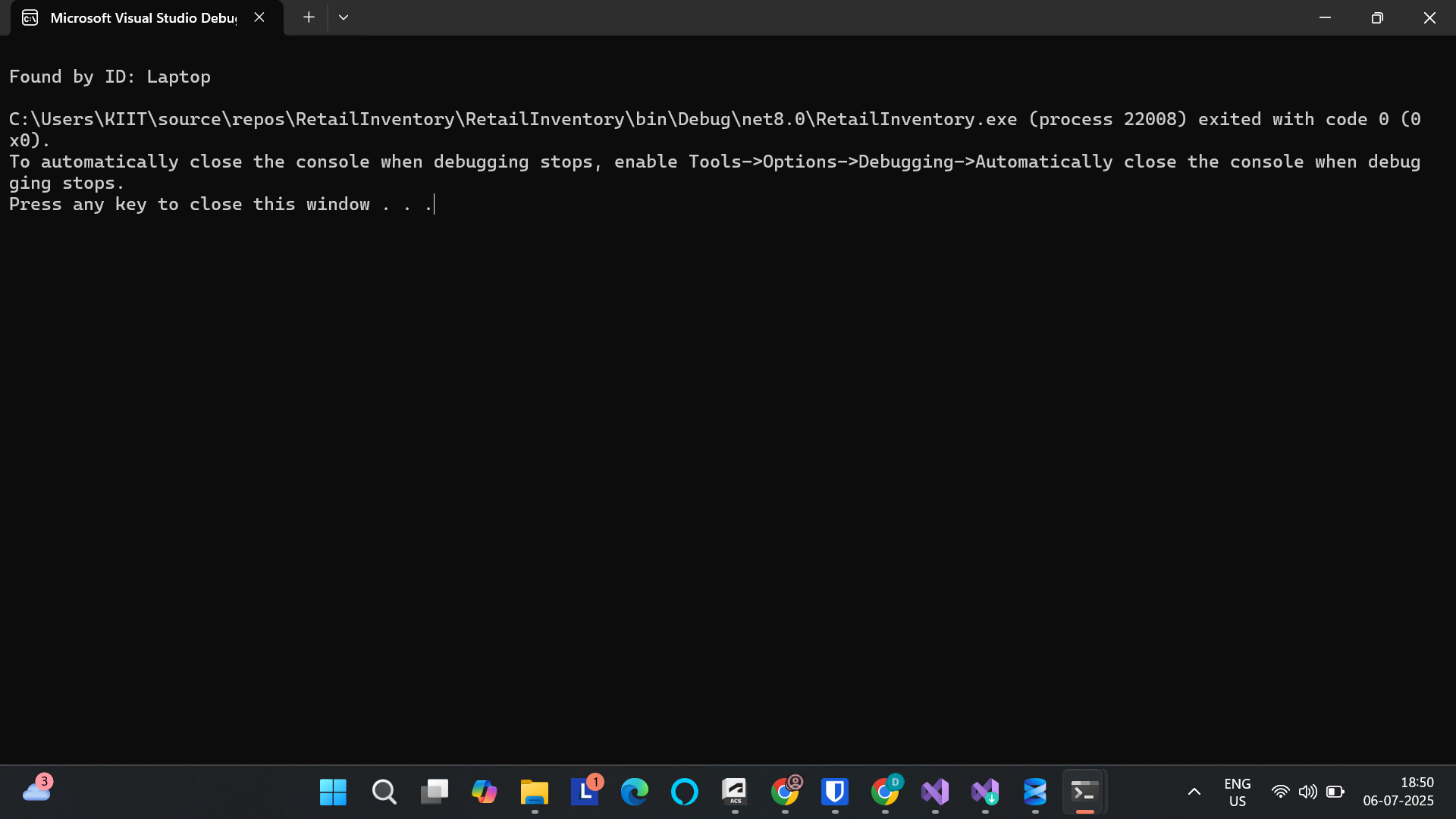
2: Find a Product by ID

**Code:**

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

Console.WriteLine($"\nFound by ID: {product?.Name ?? "Not found"}");

**Output:**

****

3: Find First Expensive Product (> ₹50,000)

**Code:**

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

Console.WriteLine($"\nExpensive product: {expensive?.Name ?? "None"}");

**Output:**

