**DN 4.0 Dotnet FSE**

**Name: Aditya Mohanty**

**Superset ID:6361043**

**Week 3-**

**Entity Framework Core 8.0**

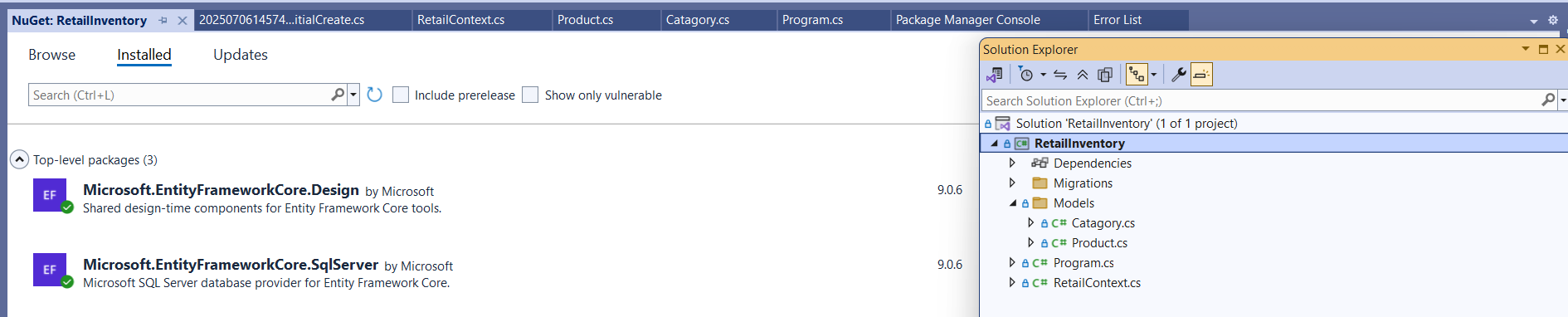
**Lab 1: Understanding ORM with a Retail Inventory System**

ORM(Object-Relational Mapping) and EF(Entity Framework)------

ORM allows developers to connect their C# code to a database without writing SQL manually. Instead of dealing directly with database tables, the developer creates C# classes like Product, with properties such as Name and Price. The ORM automatically maps these classes and properties to database tables and columns. This helps developers work with objects in code while the ORM handles all the data storage and retrieval in the background. As a result, development becomes faster, the code is easier to manage, and changes to the database can be handled with fewer updates in the code.

EF Core is a lightweight, modern version of Microsoft’s ORM tool for .NET applications. It allows developers to work with a database using C# objects, without having to write SQL queries manually. EF Core maps C# classes to database tables and handles tasks like inserting, updating, and retrieving data.

Screenshot of Console And Packages after installation:



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

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

Code for models:

**Products.cs**

namespace RetailInventory.Models;

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = null!;

public int StockLevel { get; set; }

public int CategoryId { get; set; }

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

public decimal Price { get; set; }

}

**Catagory.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; } = null!;

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

}

Code for Database Context:

**AppDbContext.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

public class AppDbContext : DbContext

{

public DbSet<Product> Products => Set<Product>();

public DbSet<Category> Categories => Set<Category>();

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

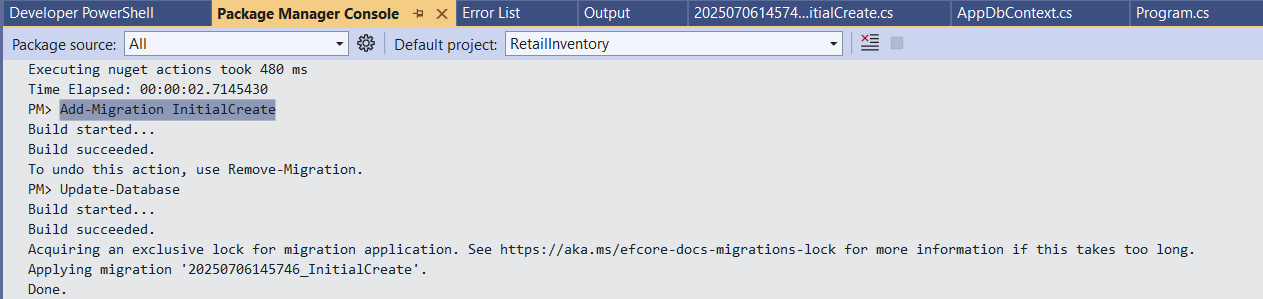
optionsBuilder.UseSqlServer(@"Server=(localDb)\MSSQLLocalDB;Database=RetailDb;Trusted\_Connection=True;");

}

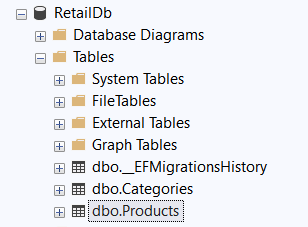
}

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

Screenshot of Package Manager Console creating migrations(Add Migration and Update Database)



SSMS Screenshot of Database after migration:



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

**Code for Program.cs**

using RetailInventory.Models;

using System.Threading.Tasks;

await InsertInitialData();

static async Task InsertInitialData()

{

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 };

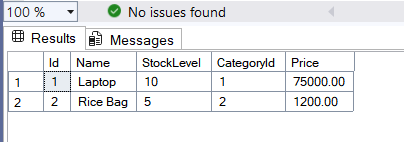
await context.Products.AddRangeAsync(product1, product2);

await context.SaveChangesAsync();

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

}

Screenshot of Product table after running the app:



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

**Changing Program.cs for Retrieval:**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

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}");

}

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

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

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

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

Output:

