**Week 3 - Entity Framework Core 8.0**

***EF Core 8.0 Guided Hands-On Exercises***

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

Scenario: You’re building an inventory management system for a retail store. The store wants to track products, categories, and stock levels in a SQL Server database.

Objective: Understand what ORM is and how EF Core helps bridge the gap between C# objects and relational tables.

**Step 1: What is ORM?**

ORM (Object-Relational Mapping) is a technique that connects C# classes to database tables. It maps class properties to table columns.  
It allows CRUD operations using LINQ instead of SQL.  
It tracks changes to objects and syncs with the database.

**Step 2: EF Core vs EF Framework:**

| Feature | EF Core 8.0 | EF Framework (EF6) |
| --- | --- | --- |
| Platform | Cross-platform | Windows only |
| Performance | Faster (compiled models) | Slower |
| Modern C# Support | Yes (LINQ, async) | Limited |
| JSON Column Mapping | Supported | Not Supported |

### **Step 3: EF Core 8.0 New Features**

* JSON Column Mapping – Store complex objects as JSON.
* Compiled Models – Drastically improves startup performance.
* Interceptors & Bulk Ops – Add hooks and efficient operations.

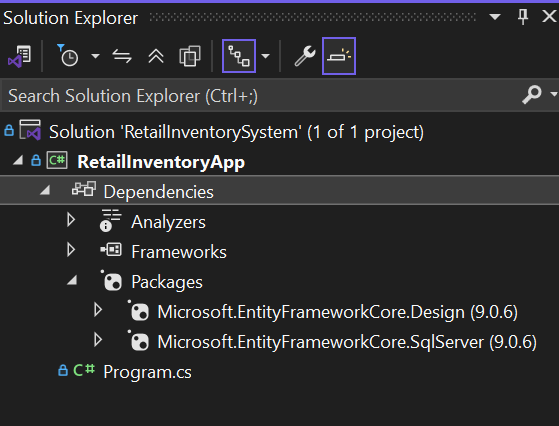
**Step 4: Create a .NET Console App:**



**Step 5: Install EF Core Packages:**



**Solution:**



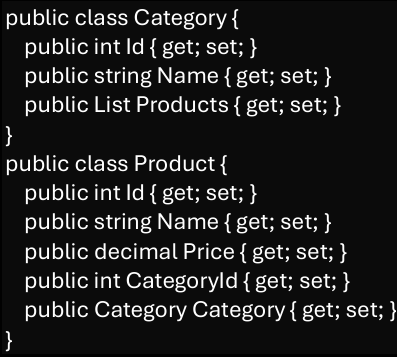
***Lab 2****: Setting Up the Database Context for a Retail Store*

Scenario: The retail store wants to store product and category data in SQL Server.

Objective: Configure DbContext and connect to SQL Server.

**Steps:**

1. Create Models:



*Category.cs*

using System.Collections.Generic;

namespace RetailInventoryApp.Models

{

public class Category

{

public int Id { get; set; }

public required string Name { get; set; }

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

}

}

*Product.cs*

using System;

using System.Collections.Generic;

namespace RetailInventoryApp.Models

{

public class Product

{

public int Id { get; set; }

public required string Name { get; set; }

public decimal Price { get; set; }

public int CategoryId { get; set; }

public required Category Category { get; set; }

}

}

**2. Create AppDbContext:**

*RetailDbContext.cs*

using Microsoft.EntityFrameworkCore;

using RetailInventoryApp.Models;

namespace RetailInventoryApp

{

public class RetailDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

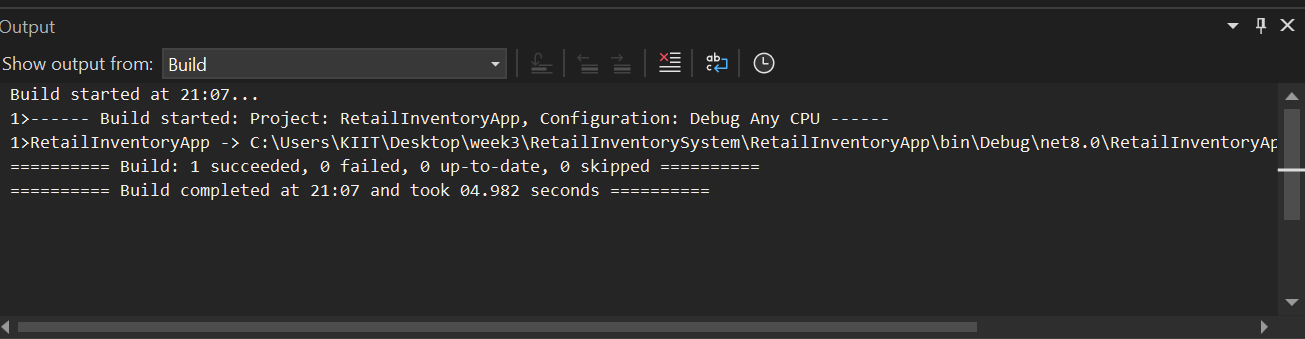
{

optionsBuilder.UseSqlServer("Server=BT-22051667;Database=RetailDB;Trusted\_Connection=True;Encrypt=False;");

}

}

}



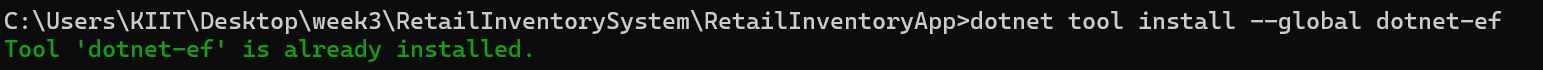
***Lab 3:*** *Using EF Core CLI to Create and Apply Migrations*

Scenario: The retail store's database needs to be created based on the models you've defined. You’ll use EF Core CLI to generate and apply migrations.

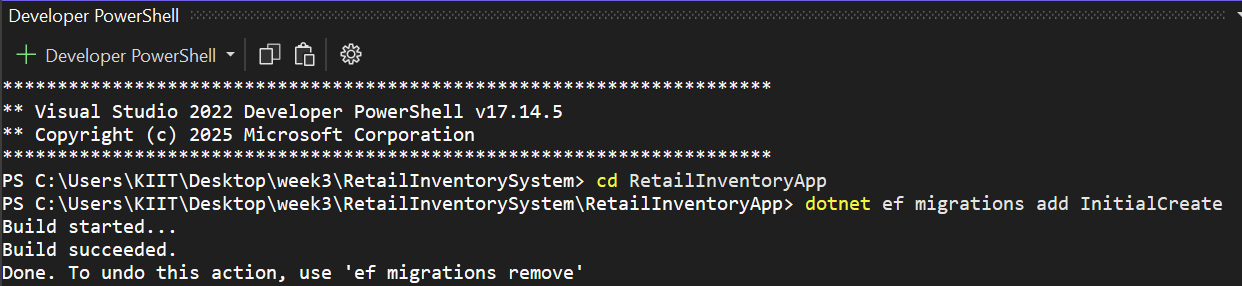
Objective: Learn how to use EF Core CLI to manage database schema changes.

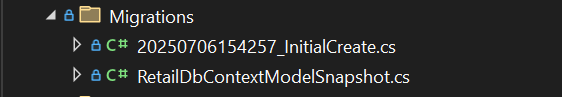
**Steps:**

**1. Install EF Core CLI :**

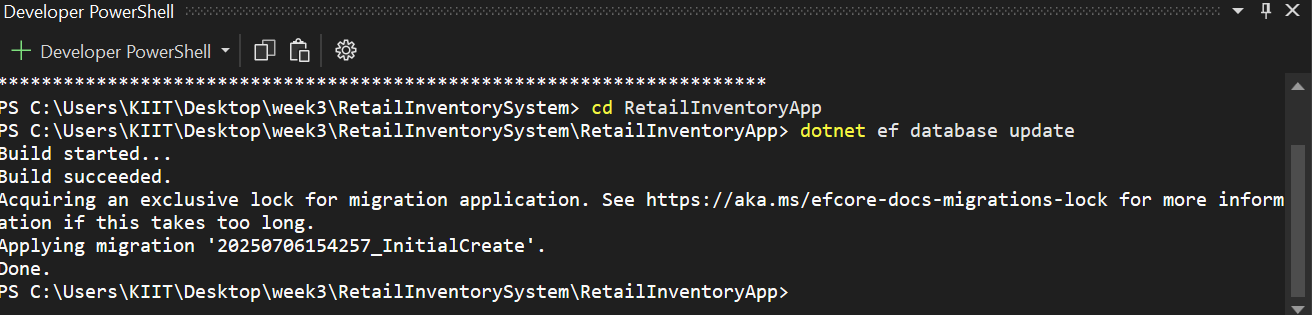


**2. Create Initial Migration:**

****

****

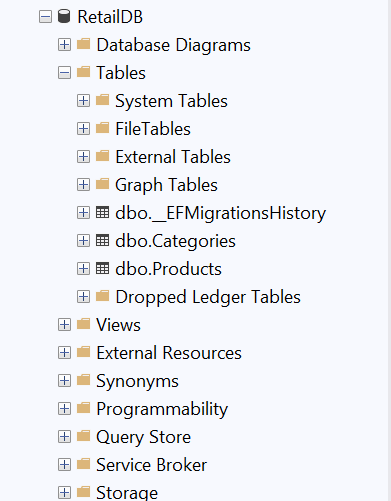
**3. Apply Migration to Create Database:**

****

**4. Verify in SQL Server:**

Open SQL Server Management Studio (SSMS) or Azure Data Studio and confirm

that tables Products and Categories are created.



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

Scenario: The store manager wants to add initial product categories and products to the system.

Objective: Use EF Core to insert records using AddAsync and SaveChangesAsync.

**Steps:**

**1. Insert Data in *Program.cs*:**

using RetailInventoryApp.Models;

using System;

namespace RetailInventoryApp

{

internal class Program

{

static void Main(string[] args)

{

using var context = new RetailDbContext();

if (!context.Categories.Any())

{

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

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

context.Categories.AddRange(electronics, groceries);

context.SaveChanges();

var product1 = new Product

{

Name = "Laptop",

Price = 75000,

Category = electronics

};

var product2 = new Product

{

Name = "Rice Bag",

Price = 1200,

Category = groceries

};

context.Products.AddRange(product1, product2);

context.SaveChanges();

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

}

else

{

Console.WriteLine(" Database already contains data. Skipping insertion.");

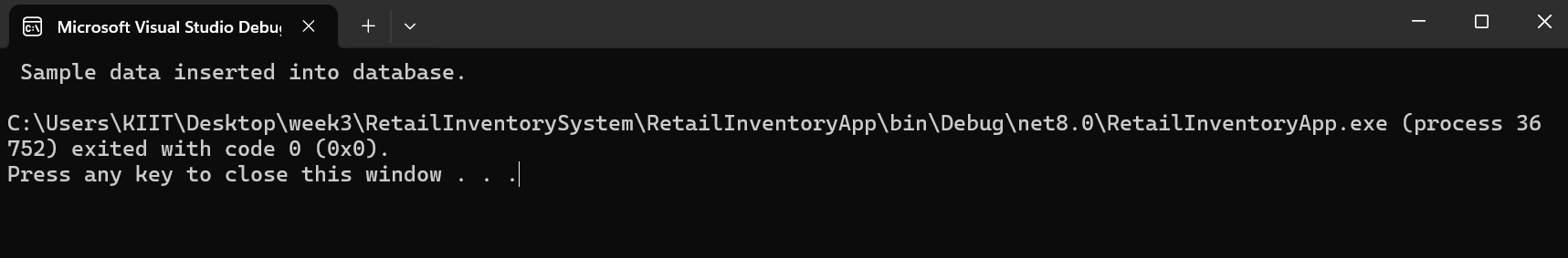
}

}

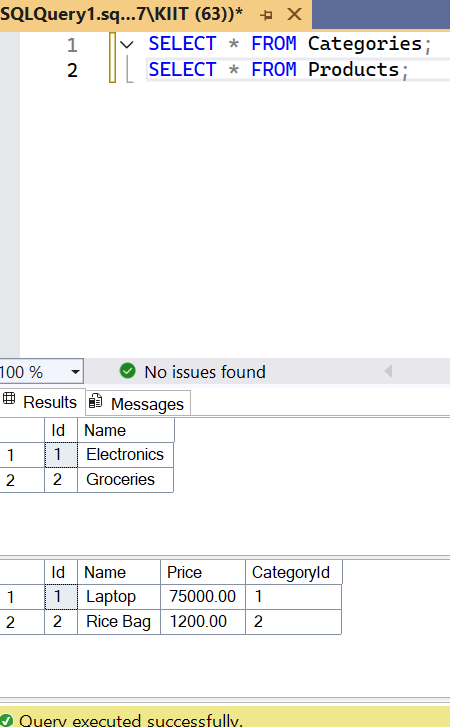
}

}

**2. Run the App:**

****

**3. Verify in SQL Server:** Check that the data is inserted correctly.



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

Scenario: The store wants to display product details on the dashboard.

Objective: Use Find, FirstOrDefault, and ToListAsync to retrieve data.

**Steps:**

**1. Retrieve All Products:**

**2. Find by ID:**

**3. FirstOrDefault with Condition:**

*Program.cs*

using System;

using System.Linq;

using Microsoft.EntityFrameworkCore;

using RetailInventoryApp.Models;

namespace RetailInventoryApp

{

internal class Program

{

static void Main(string[] args)

{

using var context = new RetailDbContext();

if (!context.Products.Any())

{

Console.WriteLine(" No product data found. Please insert data using Lab 4.");

return;

}

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

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

foreach (var product in products)

{

Console.WriteLine($" Product: {product.Name,-15} | Rs:{product.Price,-8} | Category: {product.Category?.Name}");

}

var productById = context.Products.Find(1);

Console.WriteLine("\n Product with ID = 1:");

Console.WriteLine(productById != null ? $"> {productById.Name}" : "Not found.");

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

Console.WriteLine("\n First Expensive Product (Price > ₹50,000):");

Console.WriteLine(expensive != null ? $"> {expensive.Name}" : "No expensive product found.");

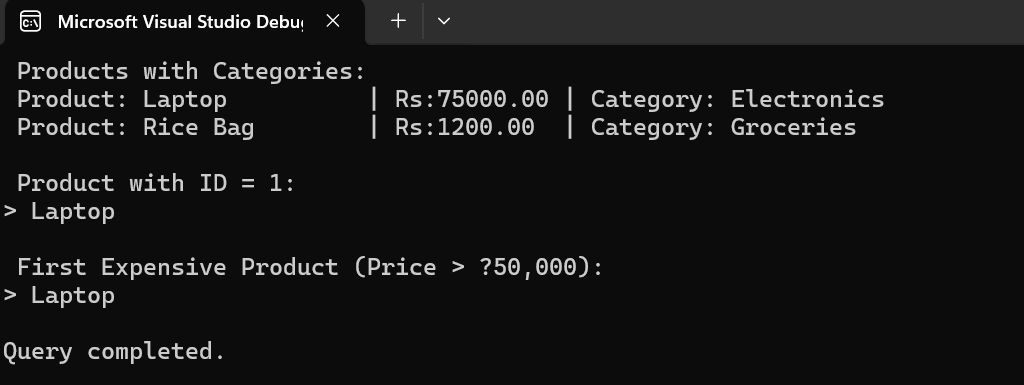
Console.WriteLine("\nQuery completed.");

}

}

}

***Output:***



**Ankita Pande**

**ID: 6362381**