Lab 1:

**What is ORM?**

* ORM (Object-Relational Mapping) maps **C# classes** to **database tables**.
* Benefits:  
  + Less SQL, more C# (Productivity)
  + Easy maintenance
  + Cross-platform abstraction

**EF Core vs EF Framework**

* EF Core is:  
  + Lightweight
  + Cross-platform (.NET Core)
  + Supports modern features like LINQ, async, compiled queries
* EF Framework (EF6):  
  + Mature, but Windows-only and less flexible

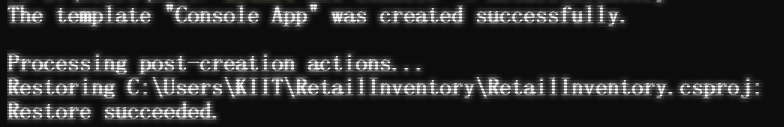
**EF Core 8.0 Features**

* JSON column mapping
* Compiled models for better performance
* Interceptors, improved bulk ops

Code: dotnet new console -n RetailInventory

cd RetailInventory

Output:

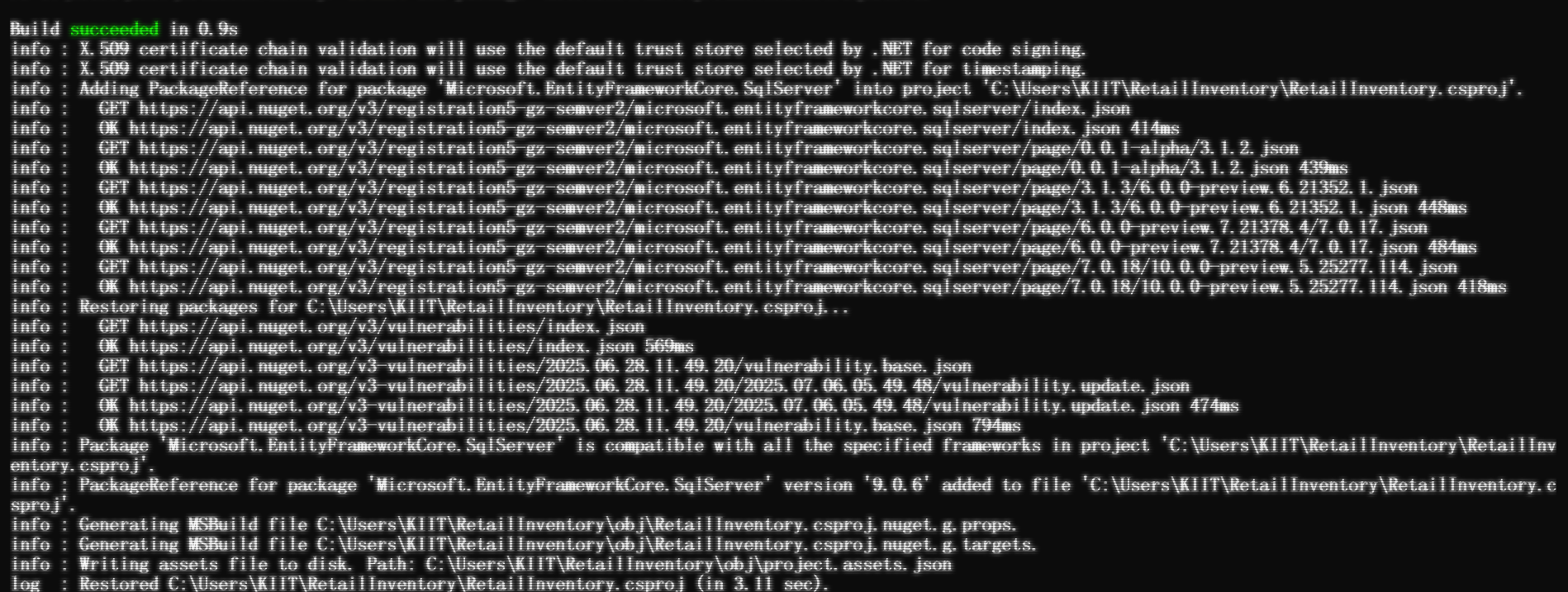


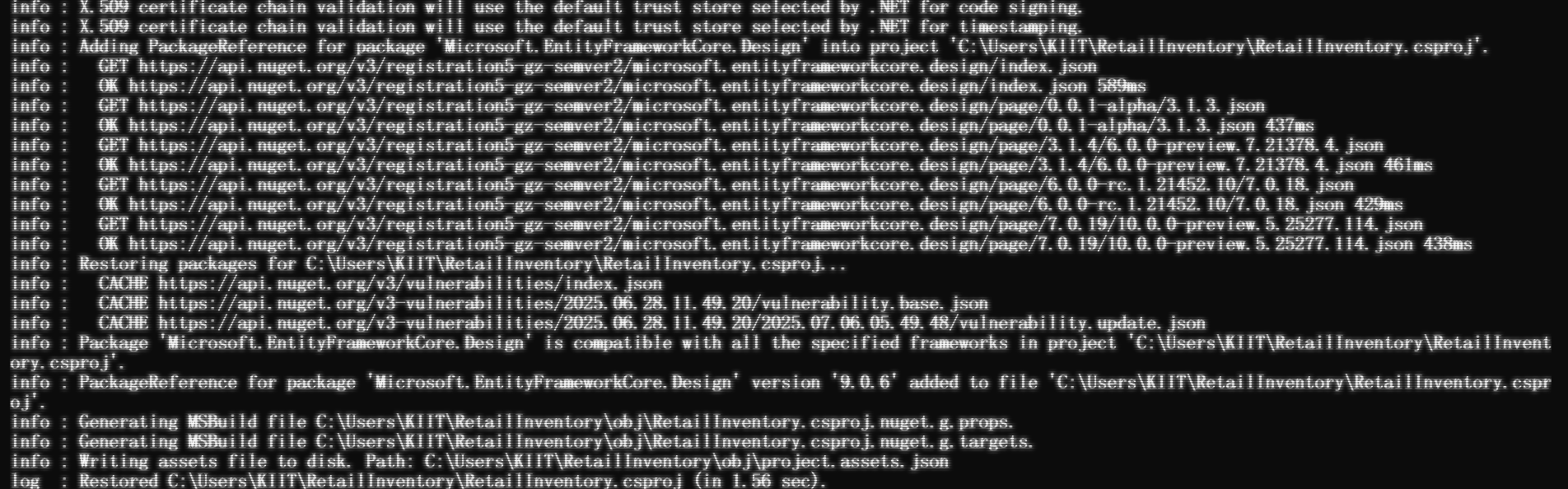
Install EF Core Packages:

Code: dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design

Ouptut:





Lab 2:  
Create Models:  
Code:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using RetailInventory;

namespace RetailInventory

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

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; }

}

}

Create AppDbContext:

Code:

using Microsoft.EntityFrameworkCore;

using RetailInventory;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace RetailInventory

{

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=abcd\\SQLEXPRESS;Database=RetailInventoryDb;User Id=sa;Password=abcd;TrustServerCertificate=True;");

}

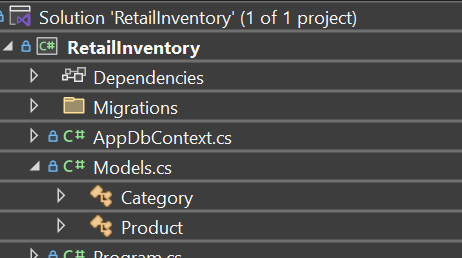
}

}

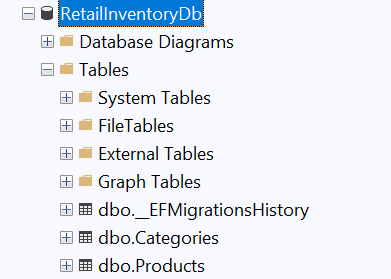
Lab 3:

1. Install EF Core CLI (if not already): dotnet tool install --global dotnet-ef

2. Create Initial Migration: dotnet ef migrations add InitialCreate



3. Apply Migration to Create Database: dotnet ef database update

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:  
  
1. Insert Data in [Program.cs](http://program.cs):

using RetailInventory;

using System;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// Create Categories

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

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

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 };

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

// Save all changes to the database

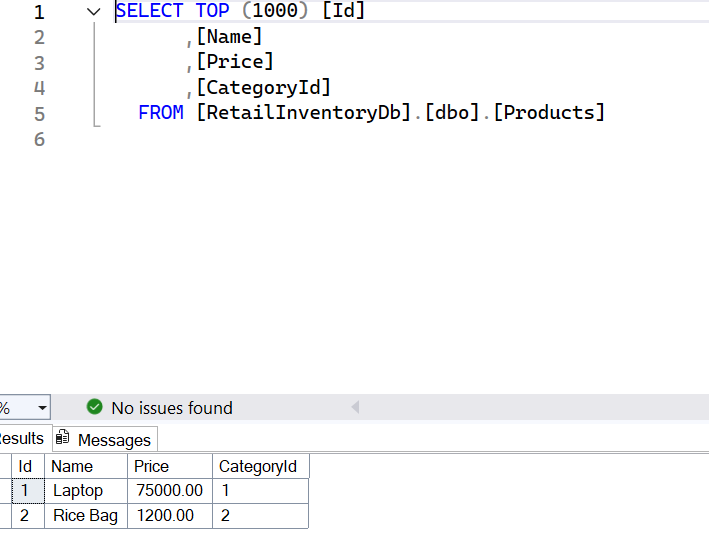
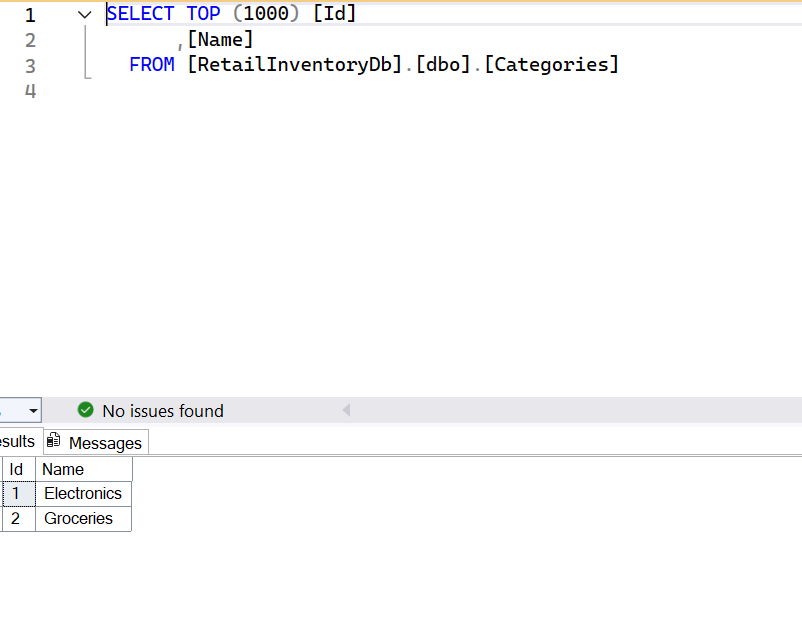
await context.SaveChangesAsync();

Console.WriteLine(" Sample categories and products inserted successfully.");

}

}

2. Run the App: dotnet run

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

Lab 5:

Code:

using Microsoft.EntityFrameworkCore;

using RetailInventory;

using System;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

Console.WriteLine("All Products:");

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

foreach (var p in products)

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

Console.WriteLine("\nFind product by ID (1):");

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

Console.WriteLine(productById != null

? $"Found: {productById.Name}"

: " Product not found.");

Console.WriteLine("\n First expensive product (Price > 50,000):");

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

Console.WriteLine(expensiveProduct != null

? $" Expensive: {expensiveProduct.Name}"

: " No expensive product found.");

}

}

Output:

