**Week 3:**

**EntityFrameworkCore8.0:**

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

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

**1.what is ORM?**

**• Explain how ORM maps C# classes to database tables.**

**• Benefits: Productivity, maintainability, and abstraction from SQL.**

**Ans:** ORM (Object-Relational Mapping) is a programming technique that allows developers to interact with a relational database using object-oriented programming languages like C#.

* Mapping: ORM maps C# classes to database tables, properties to columns, and objects to rows. This enables developers to work with data as strongly-typed objects instead of writing raw SQL queries.
* Benefits:
  + Productivity: Less boilerplate SQL, focus on business logic.
  + Maintainability: Database schema changes can be managed through code, making maintenance easier.
  + Abstraction: Developers are shielded from complex SQL, reducing the need to write and debug SQL manually.

**2.EF Core vs EF Framework:**

**Ans:**

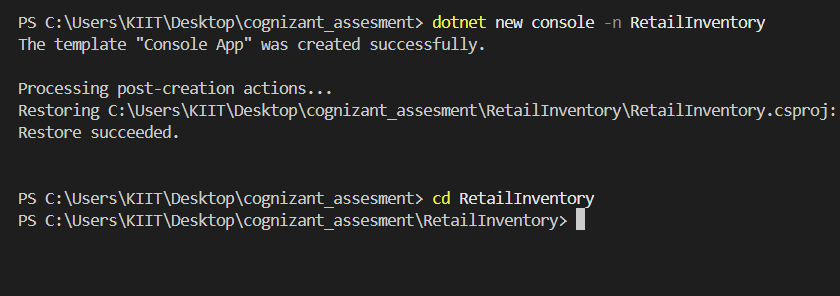
| **Feature** | **EF Core** | **EF Framework (EF6)** |
| --- | --- | --- |
| **Platform Support** | **Cross-platform (Windows, Linux, macOS)** | **Windows-only** |
| **Modularity** | **Lightweight and modular** | **Monolithic** |
| **Features** | **LINQ, async queries, compiled queries** | **Mature, less flexible** |
| **Development** | **Actively developed, new features** | **Stable, slower feature adoption** |
| **Use Case** | **Modern, cross-platform apps** | **Legacy, full .NET Framework** |

**3.EF Core 8.0 Features:**

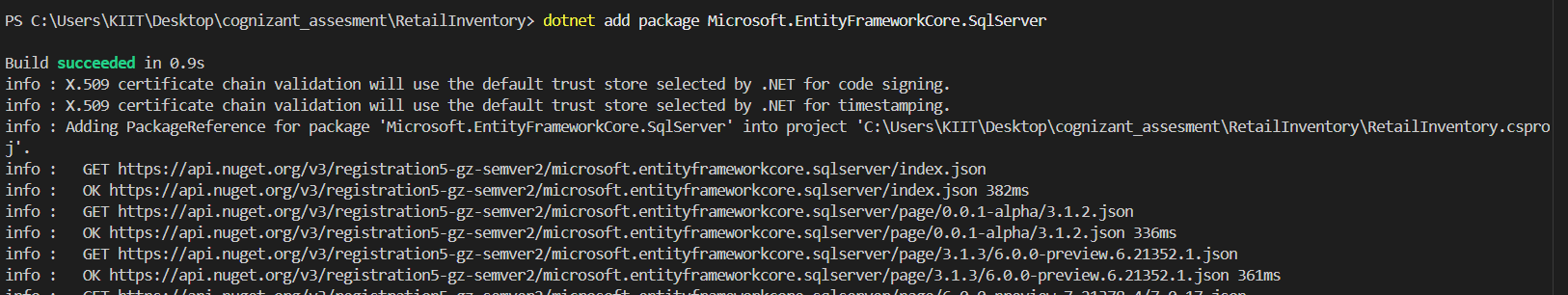
**Ans:**

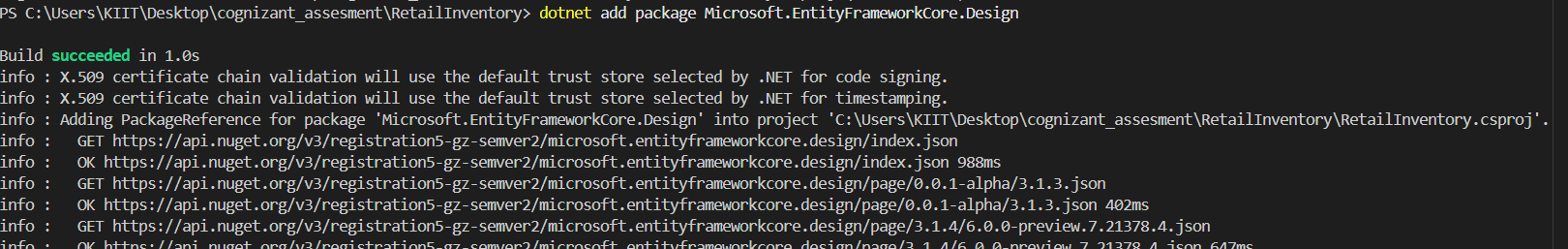
* JSON column mapping: Store and query JSON data directly in database columns.
* Improved performance with compiled models: Faster startup and query execution by pre-compiling models.
* Interceptors: Customize database operations (commands, transactions).
* Better bulk operations: Enhanced support for bulk insert, update, and delete

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

****

**5.Install EF Core Packages:**

****

****

**Lab2: Setting Up the Database Context for a Retail Store:**

**Step1:**

**Code:**

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

**using System.Collections.Generic**

**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)**:**

**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;**

**namespace RetailInventory.Data**

**{**

**public class AppDbContext : DbContext**

**{**

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

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

**protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)**

**{**

**optionsBuilder.UseSqlServer("Your\_Connection\_String\_Here");**

**}**

**}**

**}**

**Appsetting.json:**

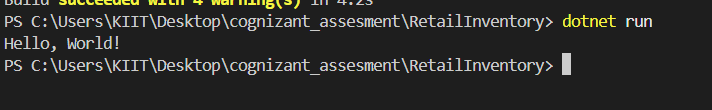
**{**

**"ConnectionStrings": {**

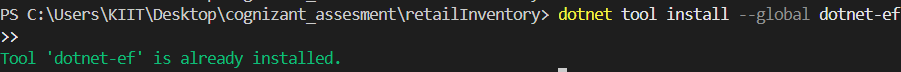
**"DefaultConnection": "Server=your\_server\_name;Database=RetailInventoryDb;Trusted\_Connection=True;MultipleActiveResultSets=true"**

**}**

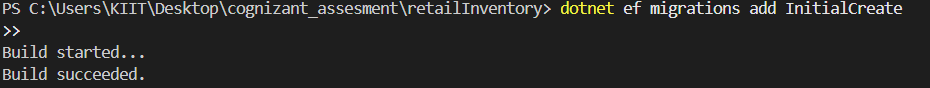
**}**

**Output:**

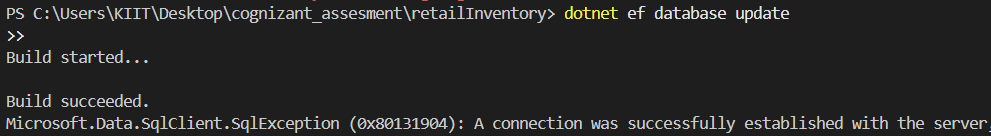
**Lab3:Using EF Core CLI to Create and Apply Migrations:**

**Step1: Install EF Core:**

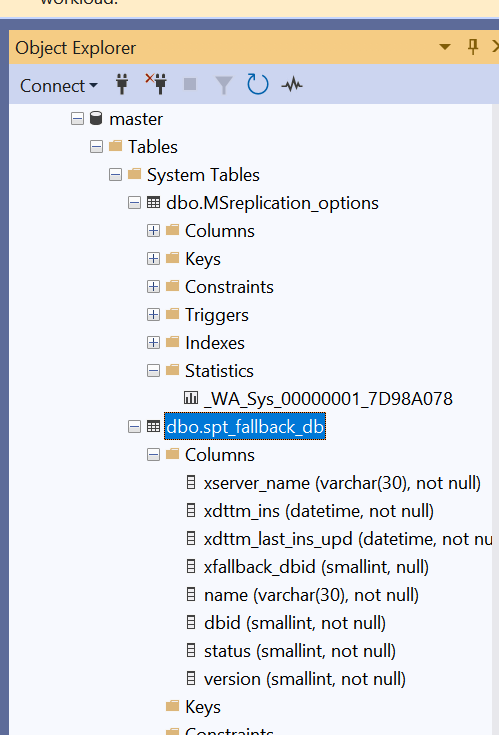
**Step 2: Create Initial Migration:**

****

**Step3: Update the Database:**

****

**Step4: Verify in SQL Server:**

****

**Lab4: Inserting Initial Data Into Database:**

[**Program.cs**](http://program.cs)**:**

**using RetailInventory.Models;**

**using RetailInventory.Data;**

**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();**

**Lab5: Retrieving Data From The Database:**

[**Program.cs**](http://program.cs)**:**

**using Microsoft.EntityFrameworkCore;**

**using RetailInventory.Data;**

**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($"\nProduct with ID 1: {product?.Name ?? "Not found"}");**

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

**Console.WriteLine($"\nFirst product with price > ₹50000: {expensive?.Name ?? "Not found"}");**

**Output:**

All Products:

Laptop - ₹75000

Rice Bag - ₹1200

Product with ID 1: Laptop

First product with price > ₹50000: Laptop