**WEEK 3-HANDSON**

**Module 4-EF core 8.0**

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.

**1.What is ORM?**

Ans: ORM is a programming technique that enables developers to interact with a relational database using object-oriented code. Instead of writing raw SQL, you use classes in your application, and the ORM handles the translation between these classes and the database tables.

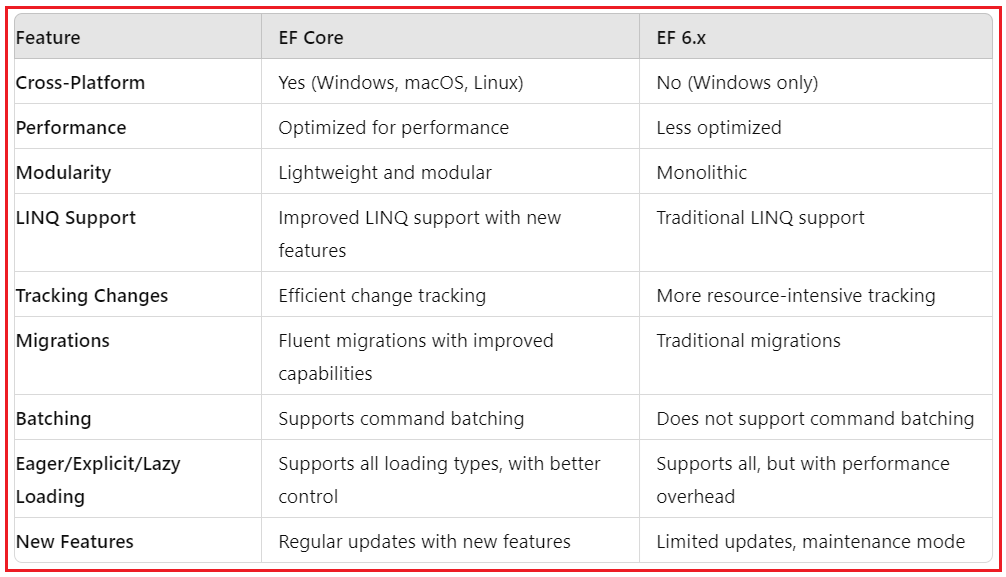
**Mapping Example:**

* + A Product class in C# maps to a Products table.
  + Properties like Id, Name, and Price map to columns in that table.
  + A list of OrderItems within a Product class can represent a one-to-many relationship with another table.

**Benefits:**

* + **Productivity**: Write less boilerplate code.
  + **Maintainability**: Centralized models make code easier to update.
  + **Abstraction from SQL**: Focus on business logic without worrying about SQL syntax or DB-specific quirks

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

**3.EF Core 8.0 Features:**

EF Core 8 brings powerful new capabilities:

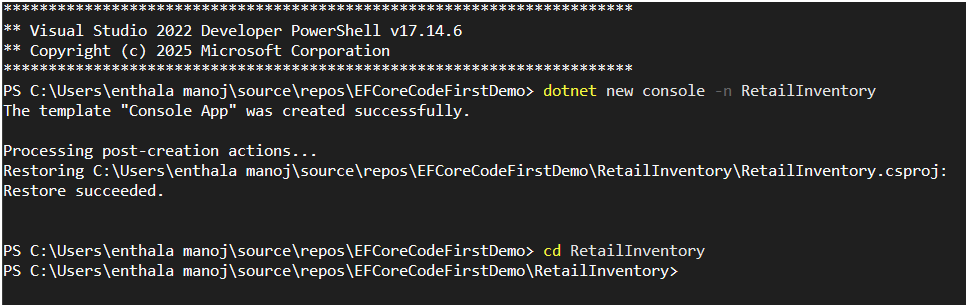
* **JSON Column Mapping:** Map complex nested objects directly to JSON columns in databases like PostgreSQL and SQL Server 2022.
* **Compiled Models:** Drastically improve startup time and query execution speed by pre-compiling models at build time.
* **Interceptors & Logging Enhancements:** Inject custom logic into DB operations or audit changes.
* **Improved Bulk Operations:** Better support for ExecuteUpdate, ExecuteDelete, and batch saving.

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

Execute the below lines:

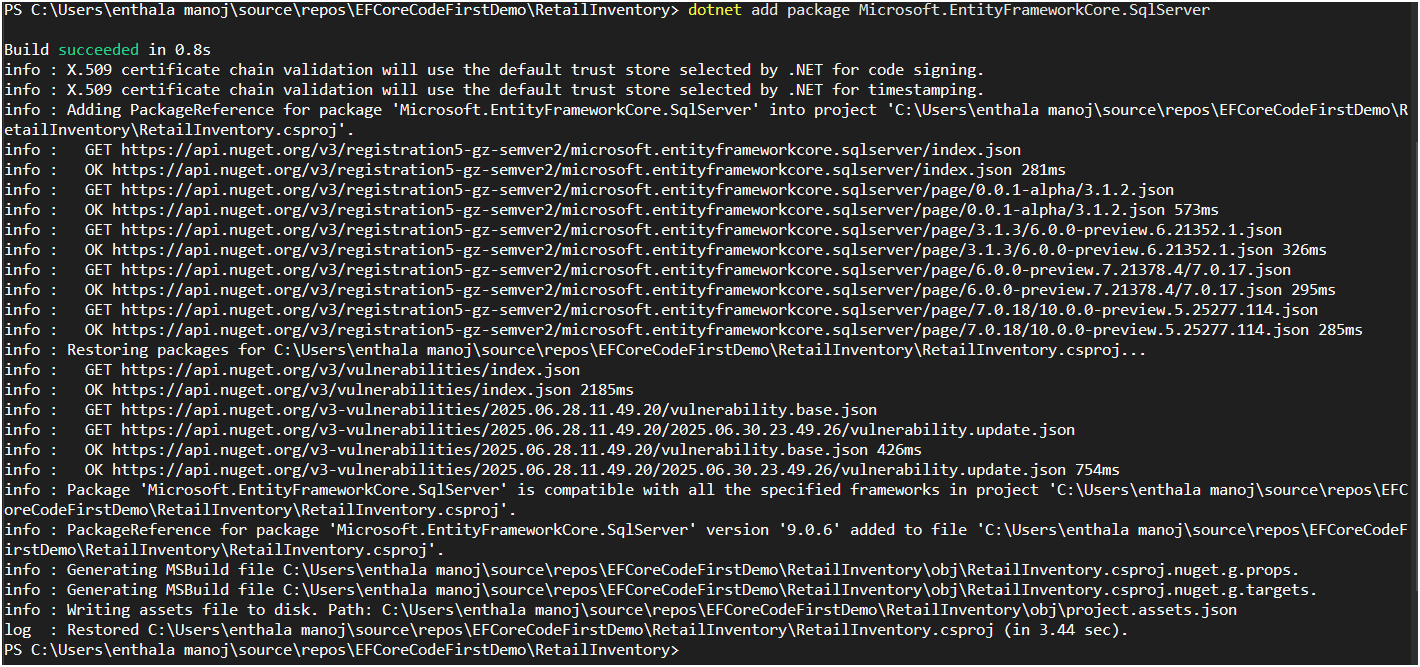
dotnet new console -n RetailInventory

cd RetailInventory

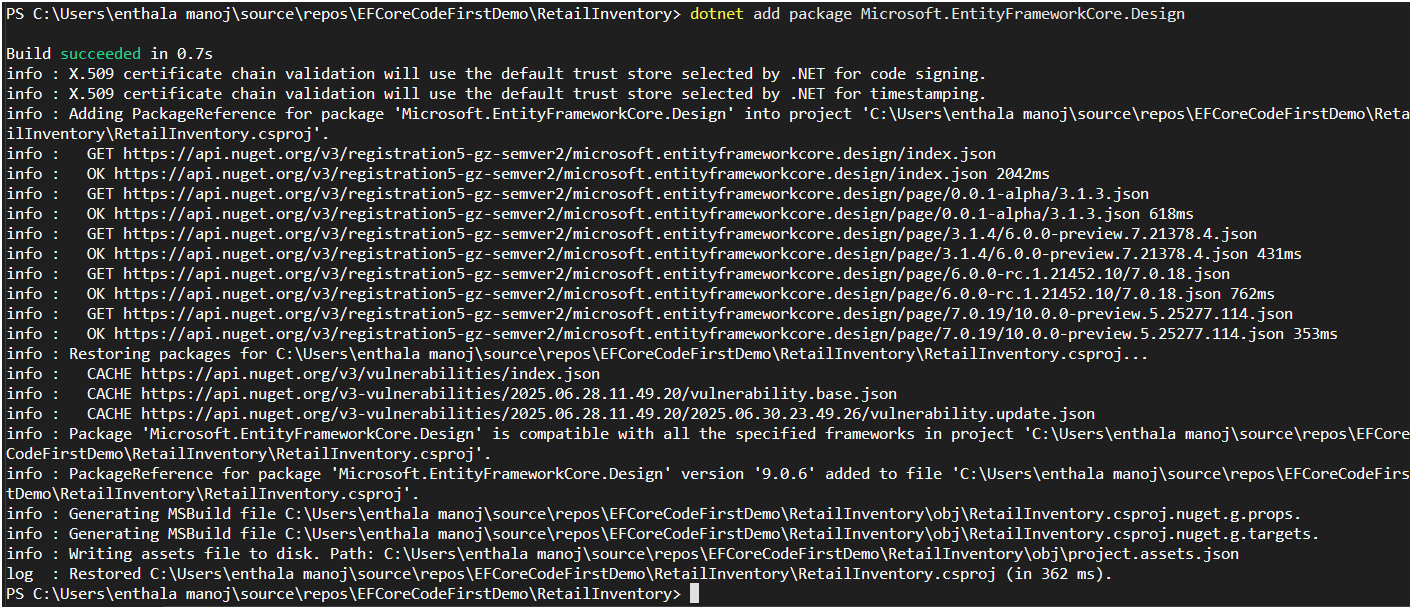
****

**5.Install EF Core Packages:**

dotnetadd package Microsoft.EntityFrameworkCore.SqlServer

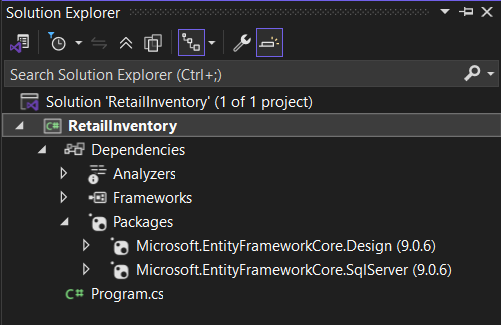


dotnetadd package Microsoft.EntityFrameworkCore.Design



**Output:**

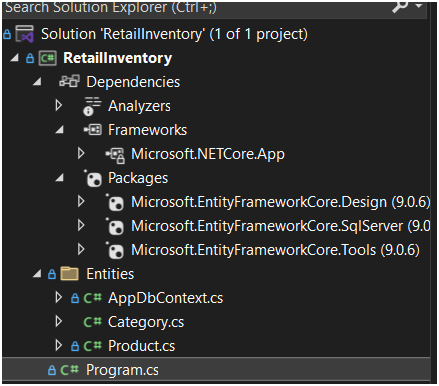
After successfully installing the packages, you can verify them from Solution Explorer under the Dependencies => Packages folder, as shown in the image below.

****

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

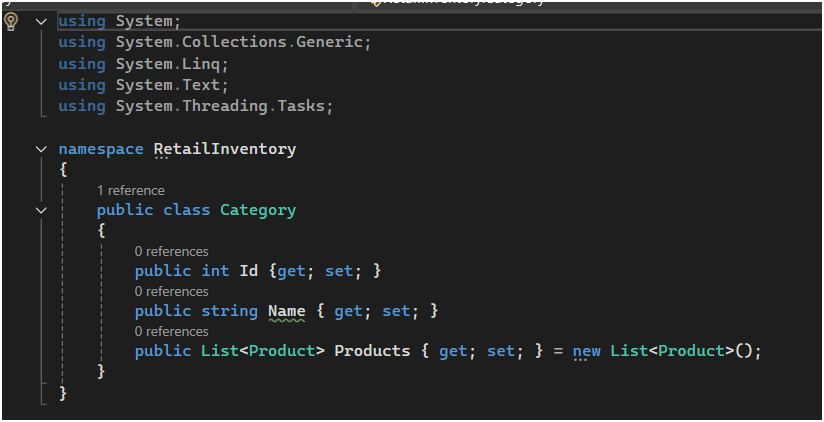
**Step1:**

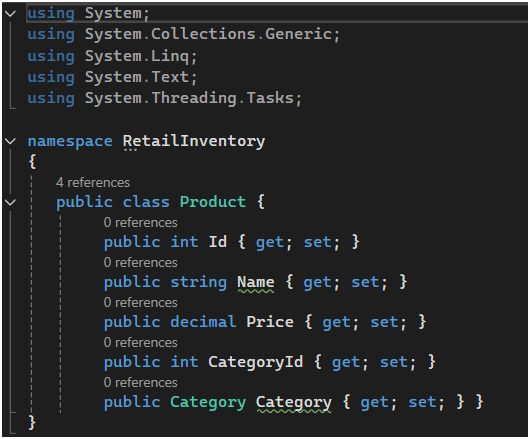
Create Folder Entities inside Project and create Category.cs, Product.cs & AppDbContext.cs:



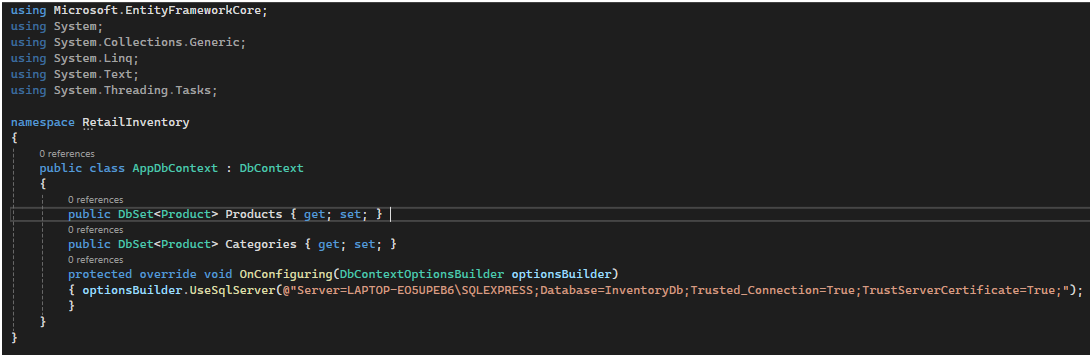
**Step2:**

Type the below code inside Category.cs

 Type the below code inside Product.cs



Type the below code inside AppDbContext.cs

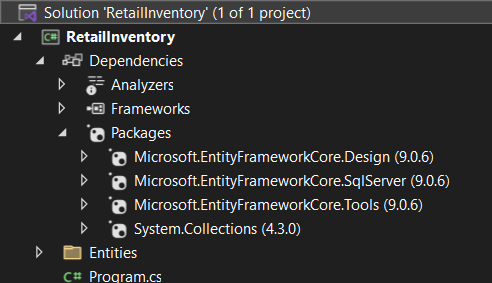


**Step3:**

**Install** EF Core tools package

Tools-> NuGet Package Manager->Package Manager Console

**Install-Package Microsoft.EntityFrameworkCore.Tools**

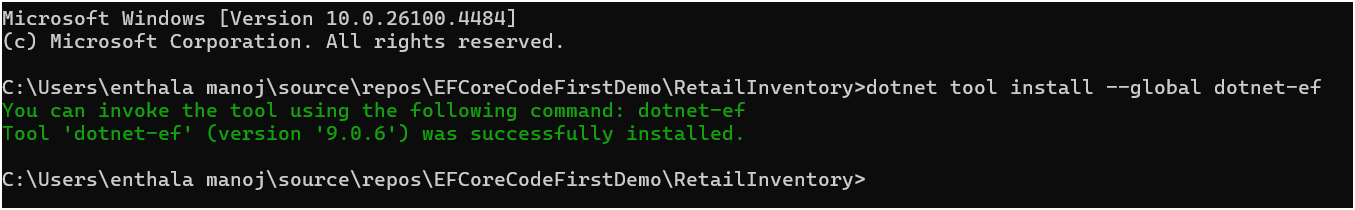
****

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

**Step1:** Install EF Core CLI (if not already):

Open terminal and type

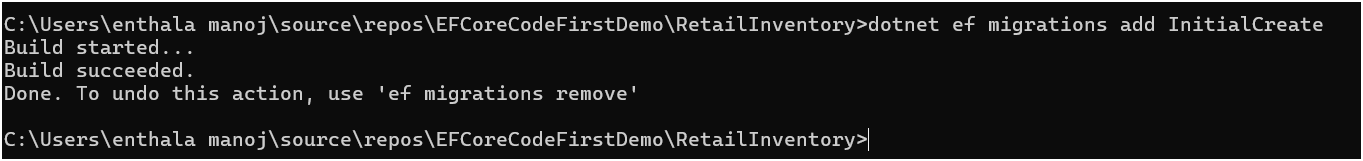
dotnet tool install --global dotnet-ef



**Step2:** **Create Initial Migration:**

type

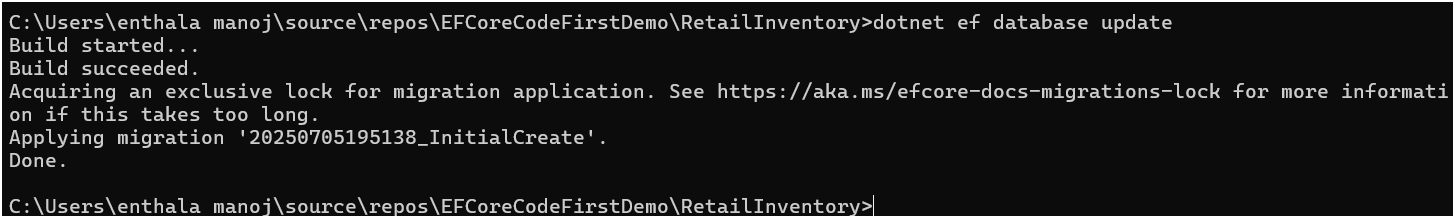
dotnet ef migrations add InitialCreate



**Step3:** **Apply Migration to Create Database:**

type

dotnet ef database update

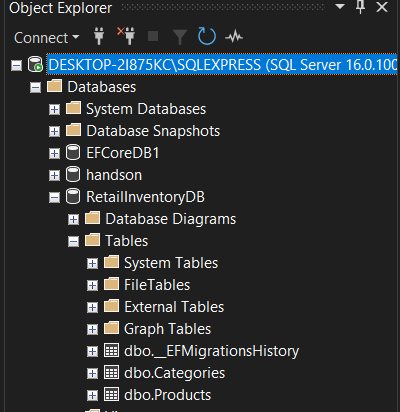


**Step4:** **Verify in SQL Server:**

Open SQL Server Management Studio (SSMS) or Azure Data Studio and confirm that tables Products and Categories are created.

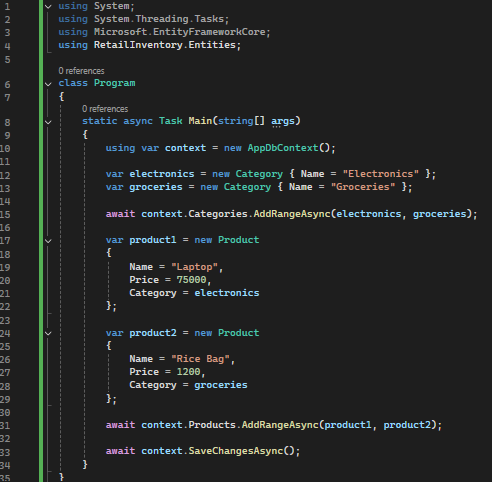
**Step4:** **Verify in SQL Server:**

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**

**Step1: Type the below code inside Program.cs to insert data inside table**

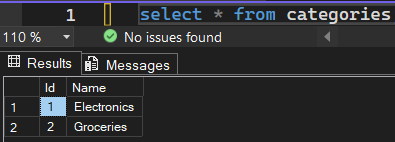
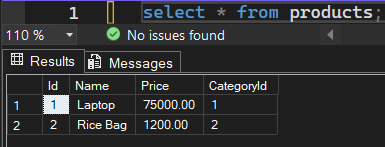
****

**Step2:** Run the App:

Open Terminal and type

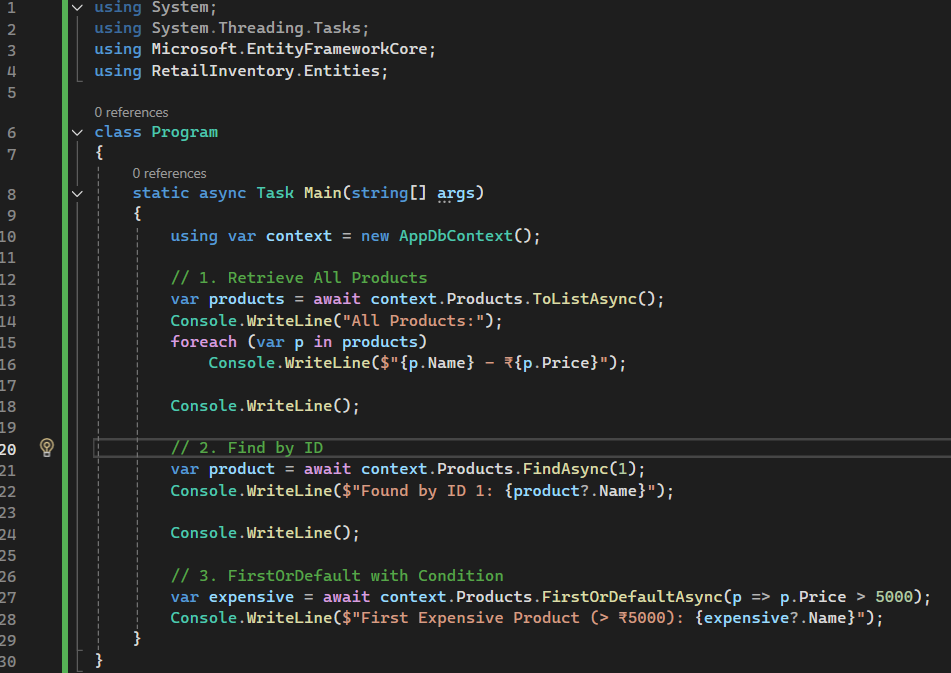
**dotnet run**

then check at SQL SERVER



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

Step1: write the below code inside Program.cs



Step2:run “dotnet” in terminal

