

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.

Steps: 1

1. What is ORM ?

=> ORM (Object-Relational Mapping) is a technique that maps C# classes to database tables.

* **Benefits: Productivity, maintainability, and abstraction**

from SQL. Benefits: -

- **Productivity:** No need to write repetitive SQL queries.

- **Maintainability:** Changes to the model are reflected in the database via migrations.

- **Abstraction:** Developers can focus on business logic instead of database syntax.

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

=> ORM (Object-Relational Mapping) is a programming technique that helps map C# classes to relational database tables. It allows developers to interact with the database using objects in C# instead of writing raw SQL queries. Entity Framework Core (EF Core) is an ORM for .NET that automates this process.

Example of Mapping :-

```
public class Product
{
    public int Id { get; set; } public string Name { get; set; }
    public decimal Price { get; set; }
}
```

When using EF Core, this class will automatically be mapped to a SQL table like this:-

SQL Table: Products

Id (int)	Name (nvarchar)	Price (decimal)
1	Laptop	75000.00

2. EF Core vs EF Framework

Feature	EF Core	EF Framework (EF6)
Platform Support	Cross-platform	Windows only
Performance	High	Moderate
Async Query Support	Yes	Limited
LINQ Support	Advanced	Basic
Compiled Queries	Yes	No
JSON Column Support	Yes (EF Core 8)	No
Best Use Case	New, modern .NET applications	Legacy .NET Framework apps

3. EF Core 8.0 Features

I. JSON Column Mapping.

EF Core 8.0 provides native support for mapping JSON columns in SQL Server. This feature enables developers to store structured data in JSON format within a single column, while still being able to query and manipulate the data using EF Core LINQ queries.

Use Case: Ideal for storing dynamic or hierarchical data such as product specifications, user preferences, or metadata without requiring schema changes.

II. Compiled Models for Enhanced Performance.

Compiled models reduce the runtime cost of model building by allowing the application to precompile the EF Core model during build time. This significantly improves application startup time and reduces CPU usage in high-load scenarios.

Benefit: Faster startup performance and better scalability for large-scale applications.

III. Interceptors for Advanced Query and Command Handling

EF Core 8.0 introduces improved support for interceptors, which allow developers to intercept and customize EF Core operations such as query execution, data saving, or database connection management.

Use Case: Logging database operations, enforcing security rules, or injecting tenant-based filters in multi-tenant applications.

IV. Enhanced Bulk Operations.

With EF Core 8.0, bulk operations such as insert, update, and delete are optimized to handle large volumes of data more efficiently. This reduces the overhead of looping through records and executing individual operations.

Benefit: High-performance data manipulation, particularly useful for batch processing, data synchronization, and ETL scenarios.

4 . Create a .NET Console App:-

```
dotnet new console -n RetailInventory
cd RetailInventory
```

```

Developer PowerShell
+ Developer PowerShell | X | 
Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
Applying migration '20250701174554_InitialCreate'.
Done.
PS C:\Users\KIIT\source\repos\RetailInventory> dotnet new console -n RetailInventory
>> cd RetailInventory
>>
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Users\KIIT\source\repos\RetailInventory\RetailInventory\RetailInventory.csproj
Restore succeeded.

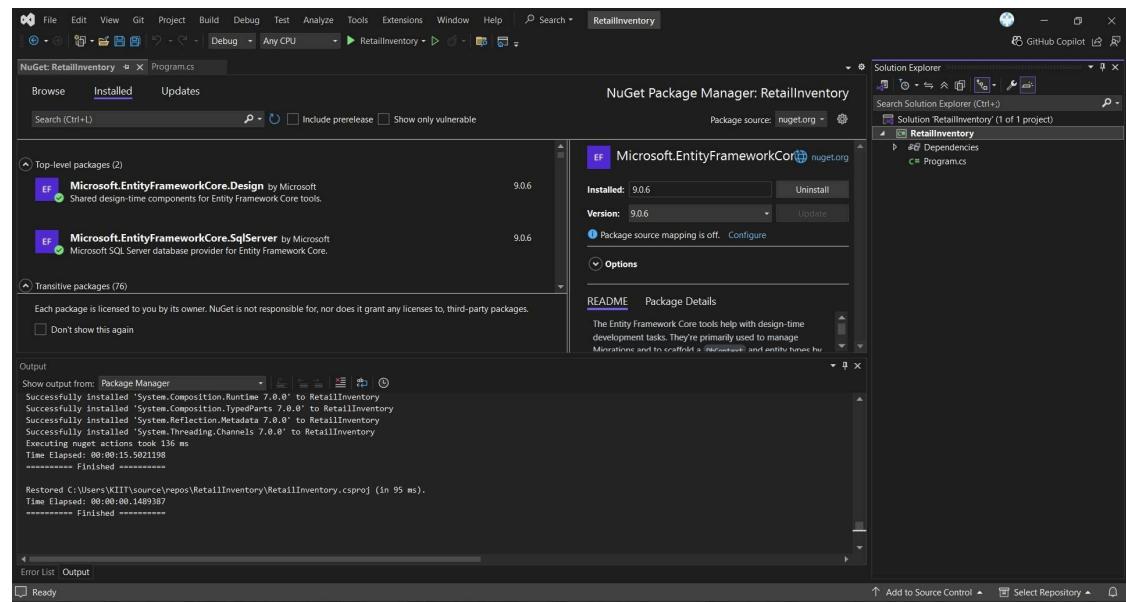
PS C:\Users\KIIT\source\repos\RetailInventory\RetailInventory> 

```

Developer PowerShell | Error List | Output

5. Install EF Core Packages.

**dotnet add package Microsoft.EntityFrameworkCore.SqlServer
dotnet add package Microsoft.EntityFrameworkCore.Design**



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:

I. Create

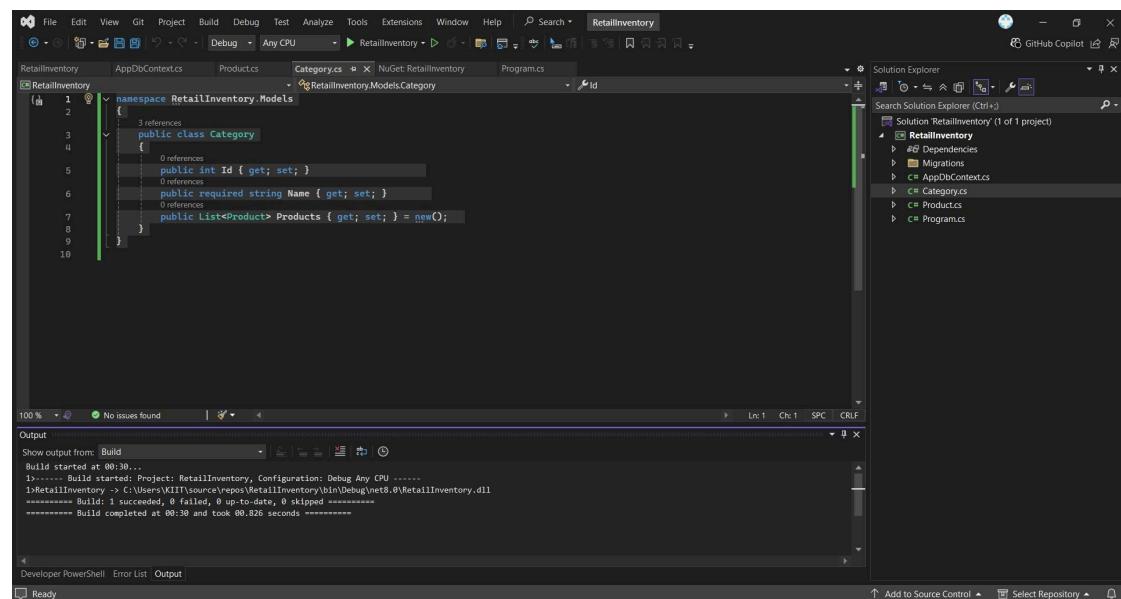
Models

Category.cs

Code :-

```
namespace RetailInventory.Models
{
    public class Category
    {
        public int Id { get; set; }
        public required string Name { get; set; }
        public List<Product> Products { get; set; } = new();
    }
}
```

OUTPUT:-



Product.cs

CODE:-

```
namespace RetailInventory.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; }
    }
}
```

OUTPUT:-

The screenshot shows the Visual Studio IDE interface. The code editor displays the 'Category.cs' file within the 'RetailInventory.Models' namespace. The file contains the definition of a 'Category' class with properties for Id and Name, and a collection of 'Products'. The Solution Explorer on the right shows the project structure with files like 'AppDbContext.cs', 'Category.cs', 'Product.cs', and 'Program.cs'. The Output window at the bottom shows a successful build process.

```
File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search RetainInventory
```

```
namespace RetailInventory.Models
{
    public class Category
    {
        public int Id { get; set; }
        public required string Name { get; set; }
        public List<Product> Products { get; set; } = new();
    }
}
```

```
Search Solution Explorer (Ctrl+F)
Solution RetailInventory (1 of 1 project)
  Assets
    Dependencies
      Migrations
      AppDbContext.cs
      Category.cs
      Product.cs
      Program.cs
```

```
Output
Show output from: Build
Build started at 00:30...
Build started at 00:30...
1>----- Build started: Project: RetailInventory, Configuration: Debug Any CPU -----
1>Metadata.cs(1,1): warning CS8008: The assembly 'C:\Users\KITT\source\repos\RetaillInventory\bin\debug\net8.0\RetaillInventory.dll' has an invalid assembly name 'RetaillInventory'. It must be a valid .NET identifier.
1>----- Build completed at 00:30 and took 00.826 seconds -----
----- Build completed at 00:30 and took 00.826 seconds -----
```

```
Developer PowerShell Error List Output
Ready
```

II. Create AppDbContext:

CODE :-

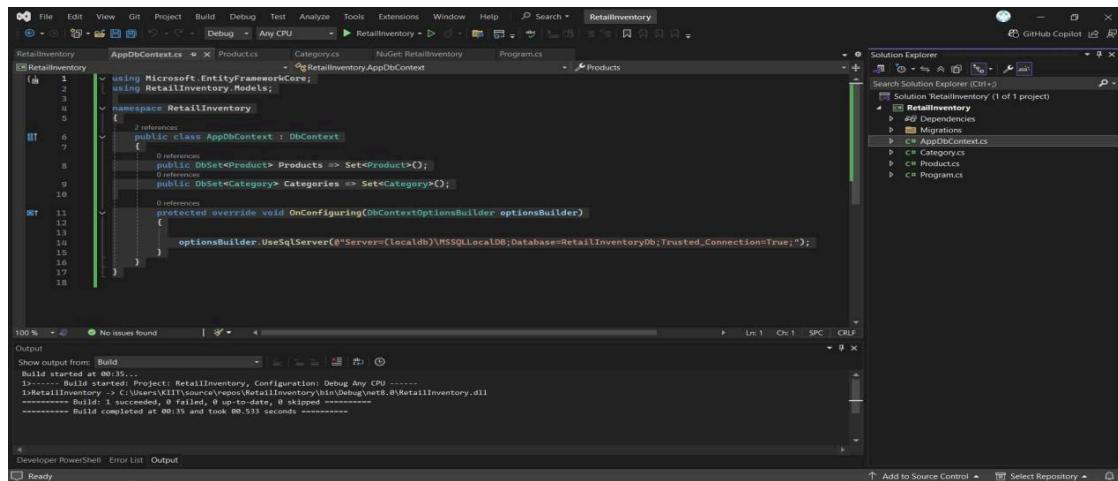
```
using Microsoft.EntityFrameworkCore; using
RetailInventory.Models;

namespace RetaillInventory
{
    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)\MSSQL\localDB;Database=Retai
lInventoryDb;Trusted_Connection=True;");
        }
    }
}
```

OUTPUT:-



III. add Connection String in appsettings.json (optional for ASP.NET Core).

To make the database connection configurable, we added a JSON file named `appsettings.json` to the project. This file stores the SQL Server connection string in a structured format.

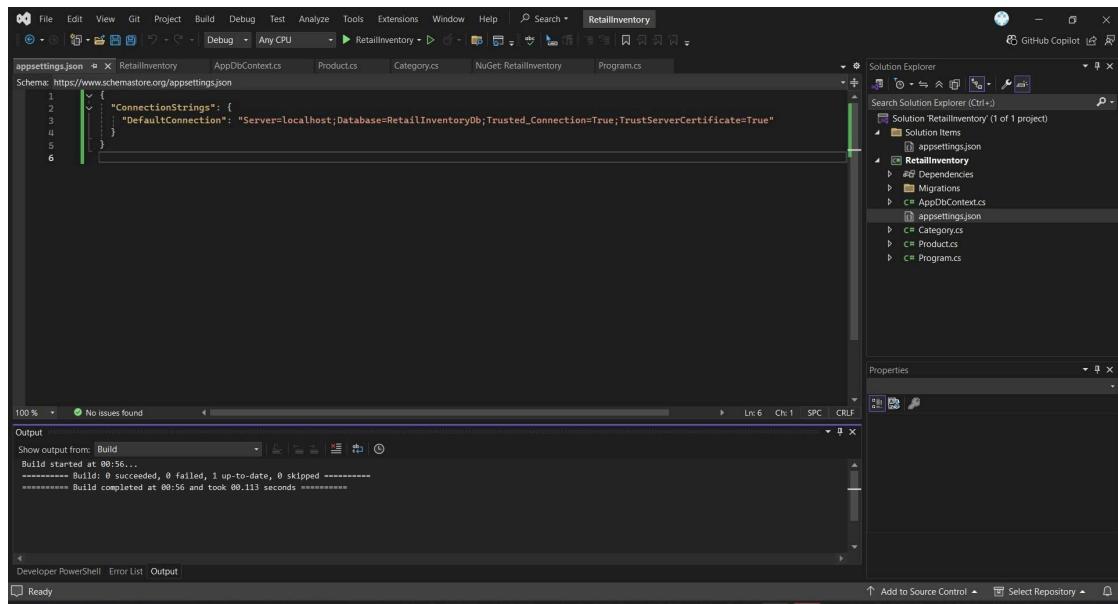
JSON Content:

appsettings.json

CODE :-

```
{
  "ConnectionStrings": {
    "DefaultConnection": {
      "ConnectionString": "Server=localhost;Database=RetailInventoryDb;Trusted_Connection=True;TrustServerCertificate=True"
    }
}
```

OUTPUT:-



Configuration in AppDbContext.cs:

In the `OnConfiguring()` method of `AppDbContext.cs`, the following code reads the connection string from `appsettings.json`:

CODE:-

```
using Microsoft.EntityFrameworkCore; using
Microsoft.Extensions.Configuration; using
RetailInventory.Models;
using System.IO;

public class AppDbContext : DbContext
{
    public DbSet<Product> Products { get; set; } public
    DbSet<Category> Categories { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        var config = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("appsettings.json")
            .Build();

        var connectionString = config.GetConnectionString("DefaultConnection");
        optionsBuilder.UseSqlServer(connectionString);
    }
}
```

OUTPUT:-

The screenshot shows the Microsoft Visual Studio IDE interface. The code editor displays the `AppDbContext.cs` file within the `RetailInventory` project. The file contains C# code for a DbContext class that inherits from `Microsoft.EntityFrameworkCore.DbContext`. It includes properties for `Products` and `Categories`, and overrides the `OnConfiguring` method to read the connection string from the `appsettings.json` file. The Solution Explorer window on the right shows the project structure with files like `appsettings.json`, `Program.cs`, and `Category.cs`. The Properties window is also visible.

```
1  using Microsoft.EntityFrameworkCore;
2  using Microsoft.Extensions.Configuration;
3  using RetailInventory.Models;
4  using System.IO;
5
6  public class AppDbContext : DbContext
7  {
8      public DbSet<Product> Products { get; set; }
9      public DbSet<Category> Categories { get; set; }
10
11     protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
12     {
13         var config = new ConfigurationBuilder()
14             .SetBasePath(Directory.GetCurrentDirectory())
15             .AddJsonFile("appsettings.json")
16             .Build();
17
18         var connectionString = config.GetConnectionString("DefaultConnection");
19         optionsBuilder.UseSqlServer(connectionString);
20     }
21 }
22
```

Packages Installed:-

`dotnet add package`

Microsoft.Extensions.Configuration.Json OUTPUT:-

```

** Visual Studio 2022 Developer PowerShell v17.4.7
** Copyright (c) 2025 Microsoft Corporation
*****dotnet add package Microsoft.Extensions.Configuration.Json
>> C:\Users\KITT\source\repos\Retaillnventory

Build succeeded in 0:00:00
Info : X.509 certificate chain validation will use the default trust store selected by .NET for code signing.
Info : X.509 certificate chain validation will use the default trust store selected by .NET for timestamping.
Info : Adding PackageReference for package "Microsoft.Extensions.Configuration.Json" into project "C:\Users\KITT\source\repos\Retaillnventory\Retaillnventory.csproj".
Info : GET https://api.nuget.org/v3/registration-v2/microsoft.extensions.configuration.json/index.json
Info : OK https://api.nuget.org/v3/registration-v2/microsoft.extensions.configuration.json/index.json 804ms
Info : GET https://api.nuget.org/v3/registration-v2/microsoft.extensions.configuration.json/index/v1/1.23.json
Info : OK https://api.nuget.org/v3/registration-v2/microsoft.extensions.configuration.json/index/v1/1.23.json 789ms
Info : GET https://api.nuget.org/v3/registration-v2/microsoft.extensions.configuration.json/index/v3/1.24/9.0.0.json
Info : OK https://api.nuget.org/v3/registration-v2/microsoft.extensions.configuration.json/index/v3/1.24/9.0.0.json 762ms
Info : GET https://api.nuget.org/v3/registration-v2/microsoft.extensions.configuration.json/index/v9/0.1/10.0.0-preview.5.25277.114.json
Info : OK https://api.nuget.org/v3/registration-v2/microsoft.extensions.configuration.json/index/v9/0.1/10.0.0-preview.5.25277.114.json 1044ms
Info : Reading package manifest for 'Microsoft.Extensions.Configuration.Json'...
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration.json/index.json
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration.json/index.json 706ms
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration.json/index/9.0.6/microsoft.extensions.configuration.json.9.0.6.nupkg
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration.json/index/9.0.6/microsoft.extensions.configuration.json.9.0.6.nupkg 452ms
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration.json/index/9.0.6/microsoft.extensions.configuration.json.9.0.6.nupkg/index.json
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration.json/index/9.0.6/microsoft.extensions.configuration.json.9.0.6.nupkg/index.json
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration/index.json
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration/index.json 445ms
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration_fileextensions/index.json
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration_fileextensions/9.0.6/microsoft.extensions.configuration_fileextensions.9.0.6.nupkg
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration_fileextensions/9.0.6/microsoft.extensions.configuration_fileextensions.9.0.6.nupkg 163ms
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration/index.json 1128ms
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration/9.0.6/microsoft.extensions.configuration.9.0.6.nupkg
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.fileproviders.abstractions/index.json 121ms
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.fileproviders.abstractions/index.json 120ms
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.fileproviders.abstractions.9.0.6.nupkg
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.configuration/9.0.6/microsoft.extensions.configuration.9.0.6.nupkg 263ms
Developer PowerShell Error List Output

```



```

info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.fileproviders.abstractions/9.0.6/microsoft.extensions.fileproviders.abstractions.9.0.6.nupkg 22ms
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.fileproviders.physical/index.json
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.fileproviders.physical/index.json 358ms
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.fileproviders.physical/9.0.6/microsoft.extensions.fileproviders.physical.9.0.6.nupkg 207ms
Info : GET https://api.nuget.org/v3/flatcontainer/microsoft.extensions.filesystemglobbing/index.json 804ms
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.filesystemglobbing/9.0.6/microsoft.extensions.filesystemglobbing.9.0.6.nupkg
Info : OK https://api.nuget.org/v3/flatcontainer/microsoft.extensions.filesystemglobbing/9.0.6/microsoft.extensions.filesystemglobbing.9.0.6.nupkg 388ms
Info : Installed Microsoft.Extensions.FileSystemGlobbing.Configuration 9.0.6 from https://api.nuget.org/v3/index.json to C:\Users\KITT\source\repos\Retaillnventory\NuGetLocal\Microsoft.Extensions.FileSystemGlobbing.Configuration\9.0.6\content\hash\W85jdxkRiuTmQ032RA0w5fF\AuclR3rZMwdb1XXJjn1mRfV+3\WAmnQg7shsyv+5PA=.
Info : Installed Microsoft.Extensions.FileProviders.Abstractions 9.0.6 with content hash qMFPkSOA0ip259pPBAVMw5tZjyip\MSw+4dFwFmZahsFwipoxlt5\vtM0tHs2m\HmfUxlg=.
Info : Installed Microsoft.Extensions.FileProviders.Abstractions 9.0.6 from https://api.nuget.org/v3/index.json to C:\Users\KITT\source\repos\Retaillnventory\NuGetLocal\Microsoft.Extensions.FileProviders.Abstractions\9.0.6\content\hash\NddgCV9tizXoutx0zgphchZp8aYoyVt8s5qYwD8S2z1lthAzm+qpr35\jboog\#2cda\0g5d4=.
Info : GET https://api.nuget.org/v3/vulnerabilities/index.json 926ms
Info : OK https://api.nuget.org/v3/vulnerabilities/index.json 926ms
Info : GET https://api.nuget.org/v3/vulnerabilities/2025_06_28_11_49_20/vulnerability.base.json
Info : OK https://api.nuget.org/v3/vulnerabilities/2025_06_28_11_49_20/vulnerability.base.json 955ms
Info : OK https://api.nuget.org/v3/vulnerabilities/2025_06_28_11_49_20/vulnerability.update.json
Info : OK https://api.nuget.org/v3/vulnerabilities/2025_06_28_11_49_20/vulnerability.update.json 952ms
Info : Package 'Microsoft.Extensions.Configuration.Json' is compatible with all the specified frameworks in project 'C:\Users\KITT\source\repos\Retaillnventory\Retaillnventory.csproj'.
Info : PackageReference for package "Microsoft.Extensions.Configuration.Json" version "9.0.6" added to file 'C:\Users\KITT\source\repos\Retaillnventory\Retaillnventory.csproj'.
Info : Writing assets file to disk. Path: C:\Users\KITT\source\repos\Retaillnventory\obj\project.assets.json
log : Restored C:\Users\KITT\source\repos\Retaillnventory\Retaillnventory\Retaillnventory.csproj (in 9.9 sec).
PS C:\Users\KITT\source\repos\Retaillnventory>

```

Lab 3: Using EF Core CLI to Create and Apply Migration

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:-

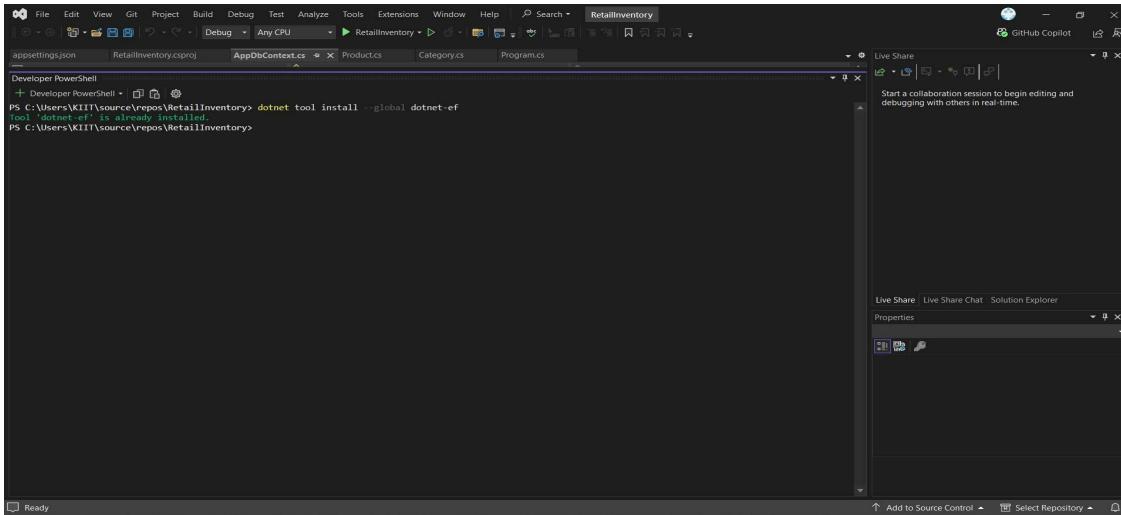
Learn how to use EF Core CLI to manage database schema changes.

Steps:-

I. Install EF Core CLI (if not already):-

```
dotnet tool install --global dotnet-e
```

OUTPUT:-

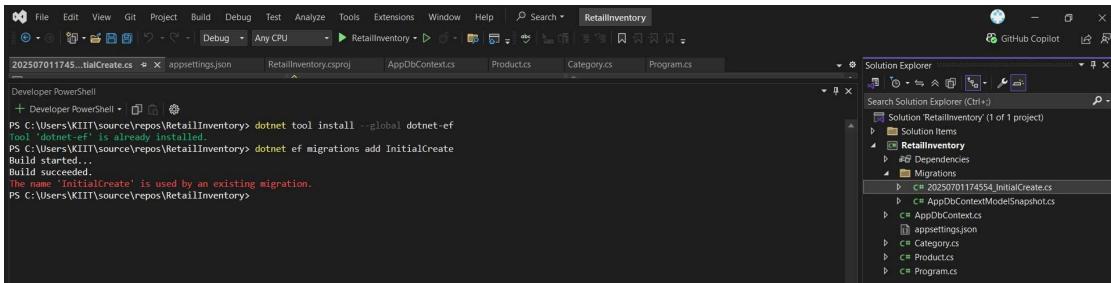


```
dotnet tool install --global dotnet-e
PS C:\Users\KLIT\source\repos\RetailInventory> dotnet tool install --global dotnet-e
Tool 'dotnet-e' is already installed.
PS C:\Users\KLIT\source\repos\RetailInventory>
```

II. Create Initial

Migration:- dotnet ef migrations

add InitialCreate OUTPUT:-



```
dotnet tool install --global dotnet-e
PS C:\Users\KLIT\source\repos\RetailInventory> dotnet tool install --global dotnet-e
Tool 'dotnet-e' is already installed.
PS C:\Users\KLIT\source\repos\RetailInventory> dotnet ef migrations add InitialCreate
Build started.
Build succeeded.
The name 'InitialCreate' is used by an existing migration.
PS C:\Users\KLIT\source\repos\RetailInventory>
```

Solution Explorer:

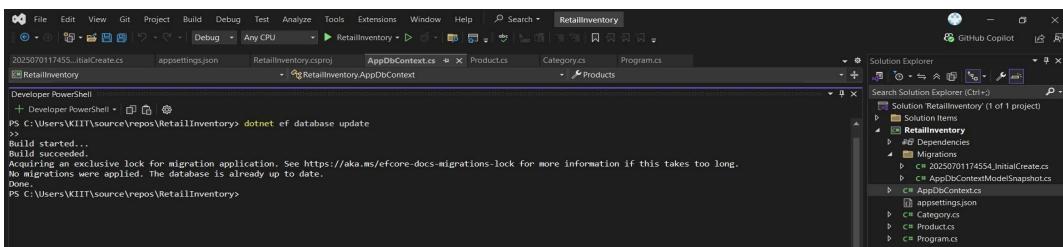
- RetailInventory (1 of 1 project)
 - Solution Items
 - RetailInventory
 - Dependencies
 - Migrations
 - C# 20250701174554_InitialCreate.cs
 - C# AppDbContextModelSnapshot.cs
 - appsettings.json
 - C# Category.cs
 - C# Product.cs
 - C# Program.cs

This generates a Migrations folder with code that represents the schema based on the models I previously created in Category.cs, Product.cs, and AppDbContext.cs.

III. Apply Migration to Create Database:-

dotnet ef database update

OUTPUT:-



```
dotnet tool install --global dotnet-e
PS C:\Users\KLIT\source\repos\RetailInventory> dotnet tool install --global dotnet-e
Tool 'dotnet-e' is already installed.
PS C:\Users\KLIT\source\repos\RetailInventory> dotnet ef database update
Build started...
Build succeeded.
Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
No migrations were applied. The database is already up to date.
Done.
PS C:\Users\KLIT\source\repos\RetailInventory>
```

Solution Explorer:

- RetailInventory (1 of 1 project)
 - Solution Items
 - RetailInventory
 - Dependencies
 - Migrations
 - C# 20250701174554_InitialCreate.cs
 - C# AppDbContextModelSnapshot.cs
 - appsettings.json
 - C# Category.cs
 - C# Product.cs
 - C# Program.cs

IV. Verify in SQL Server:-

The image consists of three vertically stacked screenshots of the SQL Server Management Studio (SSMS) interface. Each screenshot shows a different query being run against the 'RetailInventoryDB' database.

Screenshot 1: The first screenshot shows a query to select top 1000 products from the 'Products' table. The results show columns: Id, Name, Price, and CategoryId. The output panel at the bottom indicates "Query executed successfully".

```

SELECT TOP (1000) [Id]
      ,[Name]
      ,[Price]
      ,[CategoryId]
  FROM [RetailInventoryDB].[dbo].[Products]
  
```

Screenshot 2: The second screenshot shows a query to select top 100 categories from the 'Categories' table. The results show the column 'Name'. The output panel at the bottom indicates "Query executed successfully".

```

SELECT TOP (1000) [Id]
      ,[Name]
  FROM [RetailInventoryDB].[dbo].[Categories]
  
```

Screenshot 3: The third screenshot shows a query to select top 100 employees from the 'Employees' table. The results show columns: Id, Name, and BirthDate. The output panel at the bottom indicates "Query executed successfully".

```

SELECT TOP (1000) [Id]
      ,[Name]
      ,[BirthDate]
  FROM [RetailInventoryDB].[dbo].[Employees]
  
```

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:-

I. Insert Data in Program.cs:-

CODE:-

```
using RetailInventory;
using RetailInventory.Models; using
System;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {
        using var context = new AppDbContext();
        var electronics = new Category { Name = "Electronics" }; var groceries =
        new Category { Name = "Groceries" };

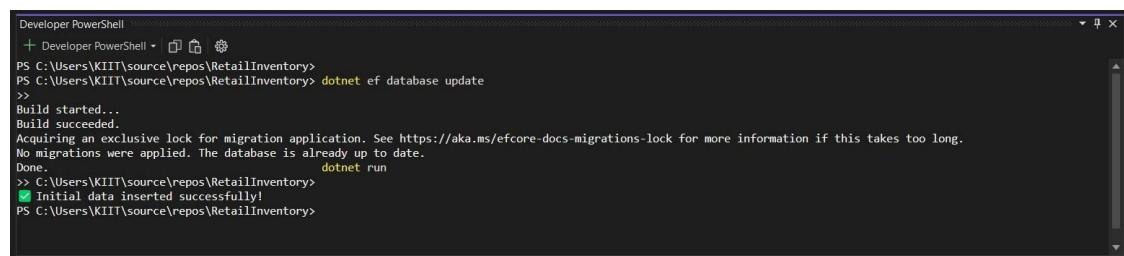
        await context.Categories.AddRangeAsync(electronics, groceries);

        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 successfully!");
    }
}
```

OUTPUT:-



The screenshot shows a terminal window titled 'Developer PowerShell' with the following command history:

```
PS C:\Users\KIIT\source\repos\RetailInventory> dotnet ef database update
>>
Build started...
Build succeeded.
Acquiring an exclusive lock for migration application. See https://aka.ms/efcore-docs-migrations-lock for more information if this takes too long.
No migrations were applied. The database is already up to date.
Done.          dotnet run
>> C:\Users\KIIT\source\repos\RetailInventory>
  Initial data inserted successfully!
PS C:\Users\KIIT\source\repos\RetailInventory>
```

The Below Screenshot showing successful insertion of products into RetailInventoryDB's Products table.

The screenshot shows the SSMS interface with the Object Explorer on the left and a query results window on the right. The query executed was:

```

1  SELECT TOP (1000) [Id]
2      ,[Name]
3      ,[Price]
4      ,[CategoryId]
5  FROM [RetailInventoryDB].[dbo].[Products]
6

```

The results grid shows the following data:

	Name	Price	CategoryId
1	Laptop	7500.00	2
2	Rice Bag	1200.00	2
3	Laptop	7500.00	3
4	Rice Bag	1200.00	4

Query executed successfully.

The Below Screenshot showing successful insertion of products into RetailInventoryDB's Categories table.

The screenshot shows the SSMS interface with the Object Explorer on the left and a query results window on the right. The query executed was:

```

1  SELECT TOP (1000) [Id]
2      ,[Name]
3  FROM [RetailInventoryDB].[dbo].[Categories]
4

```

The results grid shows the following data:

	Name
1	Electronics
2	Groceries
3	Electronics
4	Groceries

Query executed successfully.

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:-

I. Retrieve All

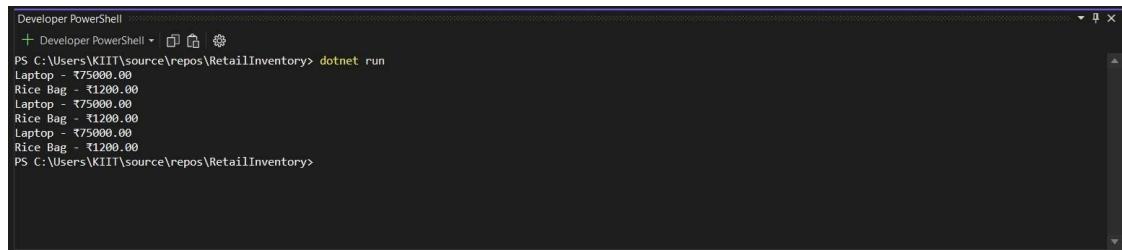
Products:- CODE:-

```
using Microsoft.EntityFrameworkCore; using
RetailInventory;
using RetailInventory.Models; using
System;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {

        using var context = new AppDbContext();
        var products = await context.Products.ToListAsync(); foreach (var p in
products)
            Console.WriteLine($"{p.Name} - ₹{p.Price}");
    }
}
```

OUTPUT:-



The screenshot shows a Developer PowerShell window. The command run was `dotnet run`. The output displayed is:

```
Laptop - ₹75000.00
Laptop - ₹12000.00
Laptop - ₹75000.00
Rice Bag - ₹1200.00
Laptop - ₹75000.00
Rice Bag - ₹1200.00
Laptop - ₹75000.00
Rice Bag - ₹1200.00
```

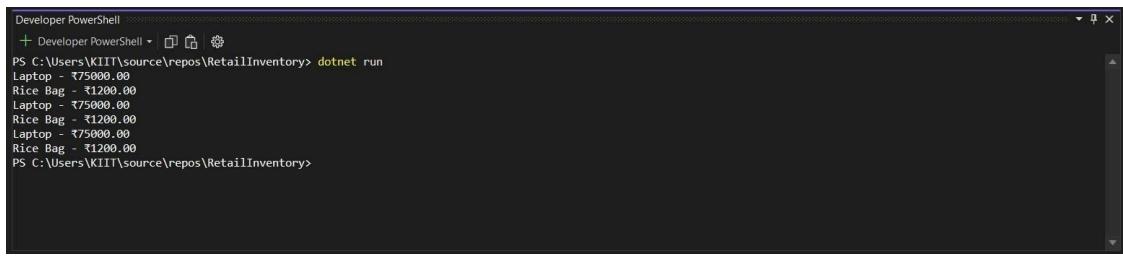
II. Find by

ID:- CODE:-

```
using Microsoft.EntityFrameworkCore; using
RetailInventory;
using RetailInventory.Models; using
System;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {
        using var context = new AppDbContext();
        var product = await context.Products.FindAsync(1); Console.WriteLine($"Found:
{product?.Name}");
    }
}
```

OUTPUT:-



```
Developer PowerShell
+ Developer PowerShell • 🌐 🌐 🌐
PS C:\Users\KIIT\source\repos\RetailInventory> dotnet run
Laptop - ₹75000.00
Rice Bag - ₹1200.00
Laptop - ₹75000.00
Rice Bag - ₹1200.00
Laptop - ₹75000.00
Rice Bag - ₹1200.00
PS C:\Users\KIIT\source\repos\RetailInventory>
```

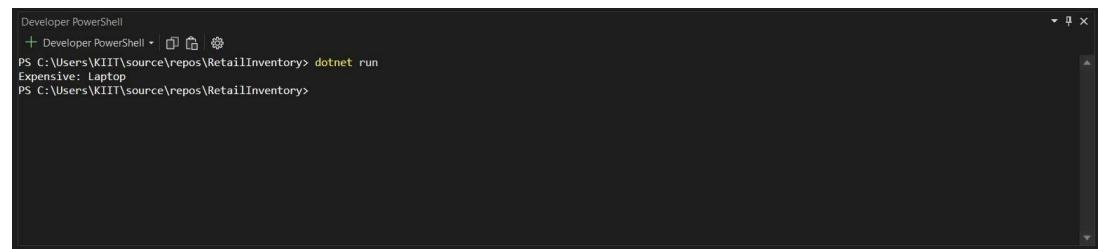
III. FirstOrDefault with

Condition:- CODE:-

```
using Microsoft.EntityFrameworkCore; using
RetailInventory;
using RetailInventory.Models; using
System;
using System.Threading.Tasks;

class Program
{
    static async Task Main(string[] args)
    {
        using var context = new AppDbContext();
        var expensive = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);
        Console.WriteLine($"Expensive: {expensive?.Name}");
    }
}
```

OUTPUT:-



```
Developer PowerShell
+ Developer PowerShell • 🌐 🌐 🌐
PS C:\Users\KIIT\source\repos\RetailInventory> dotnet run
Expensive: Laptop
PS C:\Users\KIIT\source\repos\RetailInventory>
```