EF Core 8.0 Guided Hands-On Exercises

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

**What is ORM?**

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

Each class represents a table.

Each property in the class maps to a column in the table.

Relationships (like one-to-many) are represented by navigation properties.

**Benefits:**

**Productivity:** Write C# code instead of SQL for most data operations.

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

**Abstraction:** Developers work with objects, not raw SQL, reducing errors and complexity.

**EF Core vs EF Framework:**

* **EF Core:**
  1. Cross-platform (.NET 6/7/8, Windows, Linux, macOS)
  2. Lightweight, modular, and supports modern features (LINQ, async, compiled queries)
  3. Actively developed and recommended for new projects
* **EF Framework (EF6):**
  1. Windows-only (.NET Framework)
  2. More mature, but less flexible and not cross-platform
  3. Lacks some modern features

**EF Core 8.0 Features:**

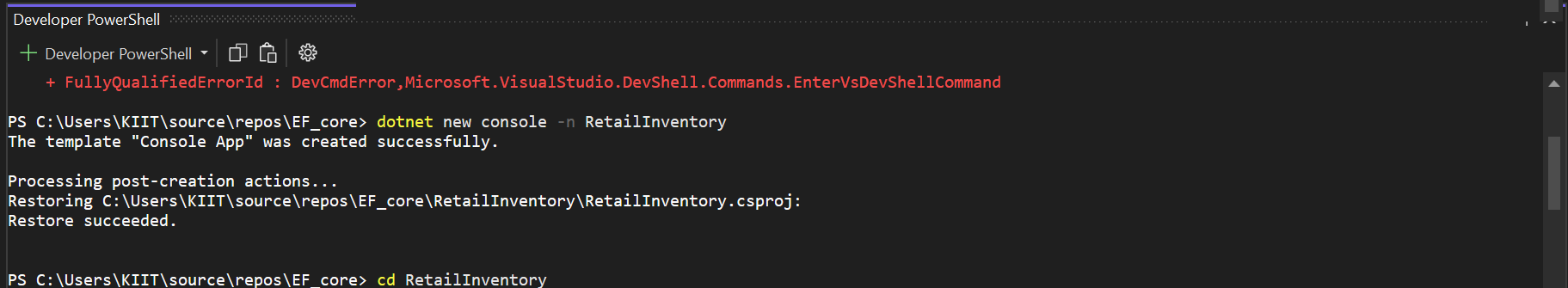
**JSON column mapping:** Store and query JSON data directly in SQL Server columns.

**Compiled models:** Faster startup and query execution.

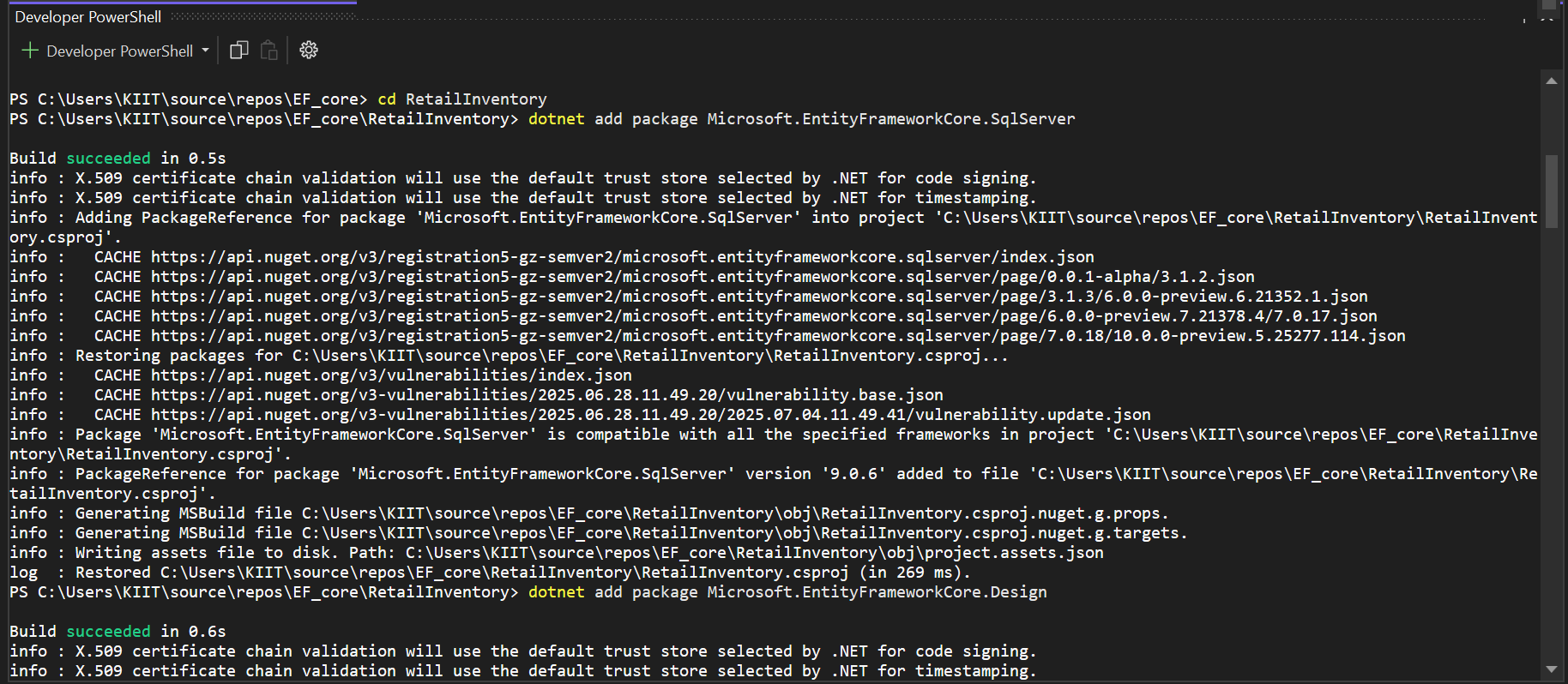
**Interceptors:** Hook into EF Core’s operations for logging, validation, etc.

**Bulk operations:** Improved support for batch inserts, updates, and deletes.

**Create a .NET Console App:**

****

**Install EF Core Packages:**

****

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

**Create Models:**

namespace Models;

public class Category

{

public int Id { get; set; }

public required string Name { get; set; }

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

}

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; }

}

**Create AppDbContext:**

using Microsoft.EntityFrameworkCore;

using Models;

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

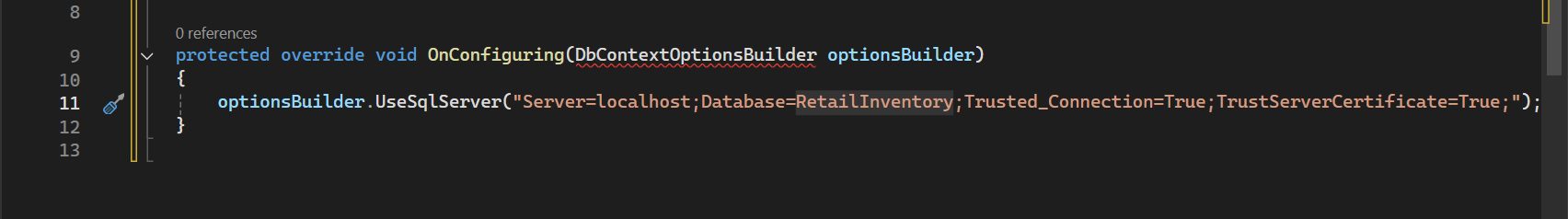
{

optionsBuilder.UseSqlServer("Server=localhost;Database=RetailInventory;Trusted\_Connection=True;TrustServerCertificate=True;");

}

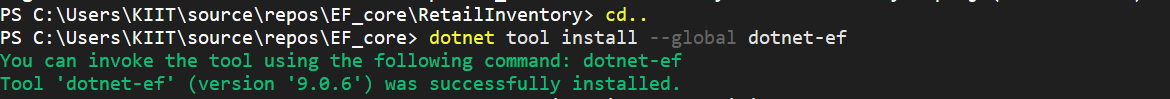
}

**Add Connection String in appsettings.json (optional for ASP.NET Core).**



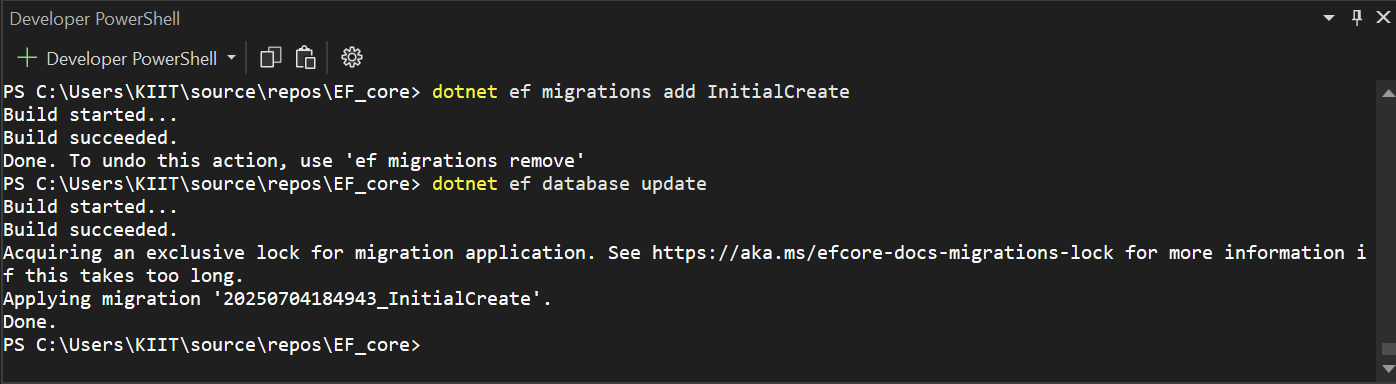
# Lab 3: Using EF Core CLI to Create and Apply Migrations

**Install EF Core CLI**

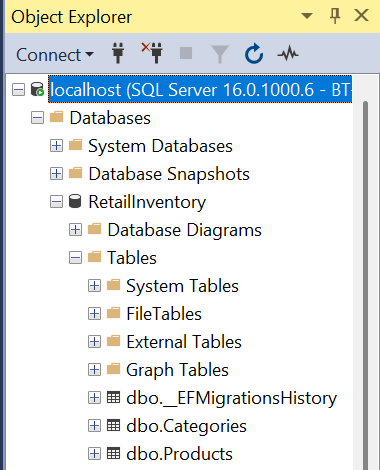
****

**Create Initial Migration**

**Apply Migration to Create Database:**



**Verify in SQL Server:**

****

# Lab 4: Inserting Initial Data into the Database

**Insert Data in Program.cs:**

using Microsoft.AspNetCore.Builder;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.EntityFrameworkCore;

using Models;

namespace RetailInventory;

public class Program

{

public static async Task Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<AppDbContext>();

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

// Seed initial data

using (var scope = app.Services.CreateScope())

{

var db = scope.ServiceProvider.GetRequiredService<AppDbContext>();

if (!db.Categories.Any())

{

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

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

await db.Categories.AddAsync(electronics);

await db.Categories.AddAsync(groceries);

await db.SaveChangesAsync();

var product1 = new Product { Name = "Laptop", Price = 1200.00m, CategoryId = electronics.Id, Category = electronics };

var product2 = new Product { Name = "Apple", Price = 1.50m, CategoryId = groceries.Id, Category = groceries };

await db.Products.AddAsync(product1);

await db.Products.AddAsync(product2);

await db.SaveChangesAsync();

}

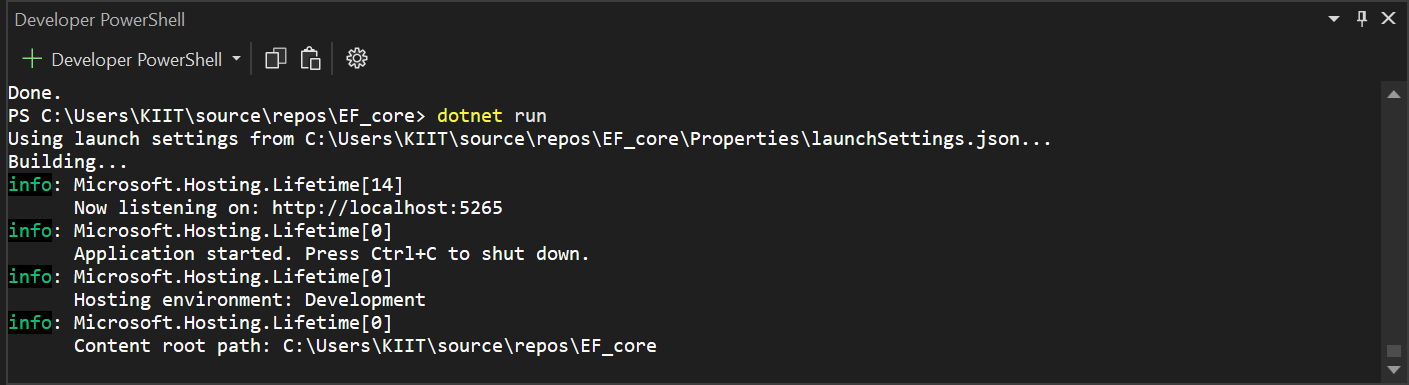
}

await app.RunAsync();

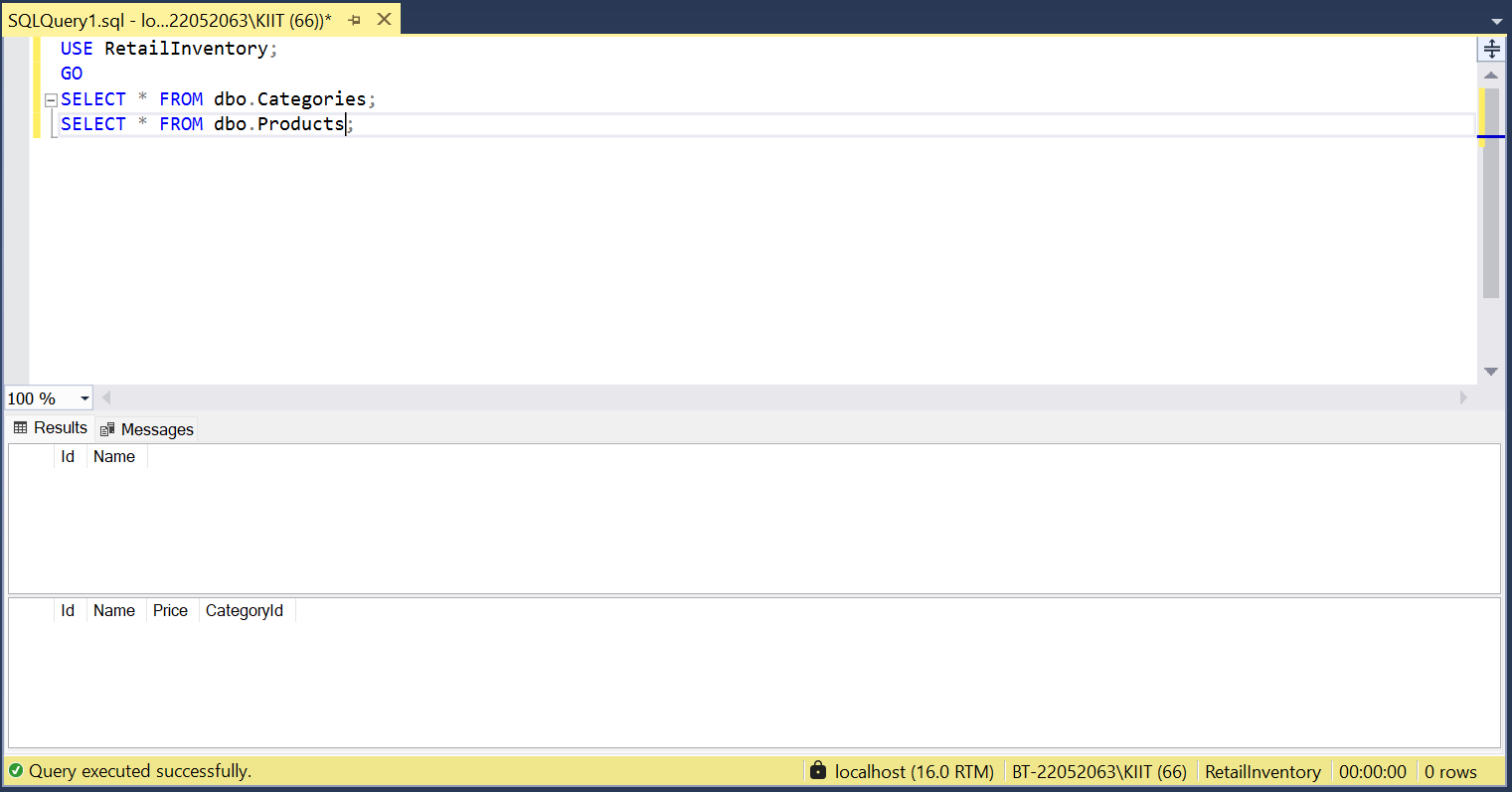
}

}

**Run the App:**

****

**Verify in SQL Server:**

****

# Lab 5: Retrieving Data from the Database

**Retrieve All Products:**

**Find by ID:**

**FirstOrDefault with Condition:**

using Microsoft.AspNetCore.Builder;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.EntityFrameworkCore;

using Models;

namespace RetailInventory;

public class Program

{

public static async Task Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<AppDbContext>();

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

// Seed initial data

using (var scope = app.Services.CreateScope())

{

var db = scope.ServiceProvider.GetRequiredService<AppDbContext>();

if (!db.Categories.Any())

{

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

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

await db.Categories.AddAsync(electronics);

await db.Categories.AddAsync(groceries);

await db.SaveChangesAsync();

var product1 = new Product { Name = "Laptop", Price = 1200.00m, CategoryId = electronics.Id, Category = electronics };

var product2 = new Product { Name = "Apple", Price = 1.50m, CategoryId = groceries.Id, Category = groceries };

await db.Products.AddAsync(product1);

await db.Products.AddAsync(product2);

await db.SaveChangesAsync();

}

}

// Retrieve and display data

using (var scope = app.Services.CreateScope())

{

var context = scope.ServiceProvider.GetRequiredService<AppDbContext>();

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

foreach (var p in products)

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

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

Console.WriteLine($"Found: {product?.Name}");

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

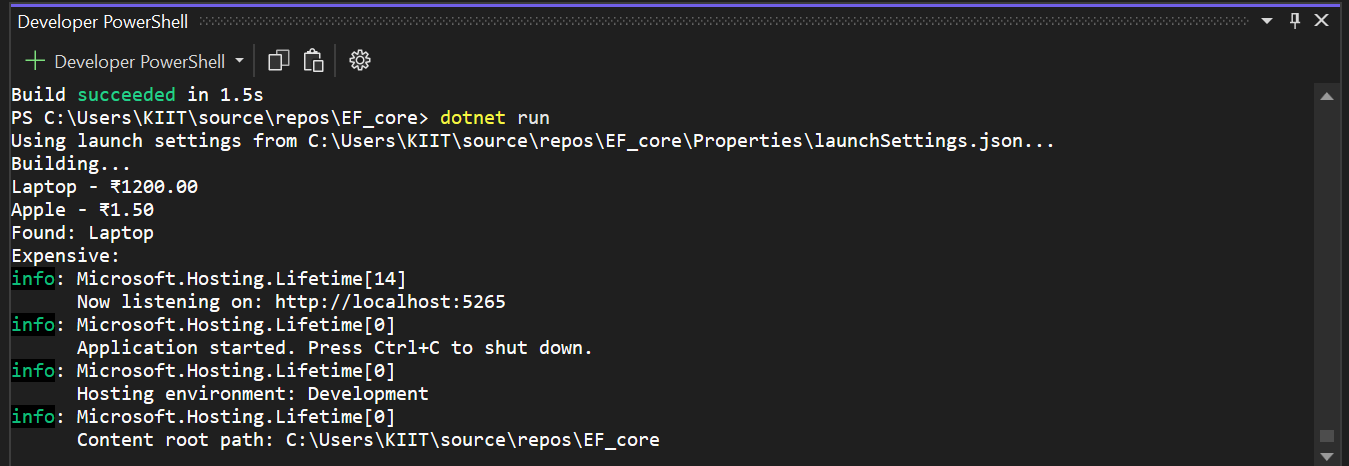
Console.WriteLine($"Expensive: {expensive?.Name}");

}

await app.RunAsync();

}

}

****