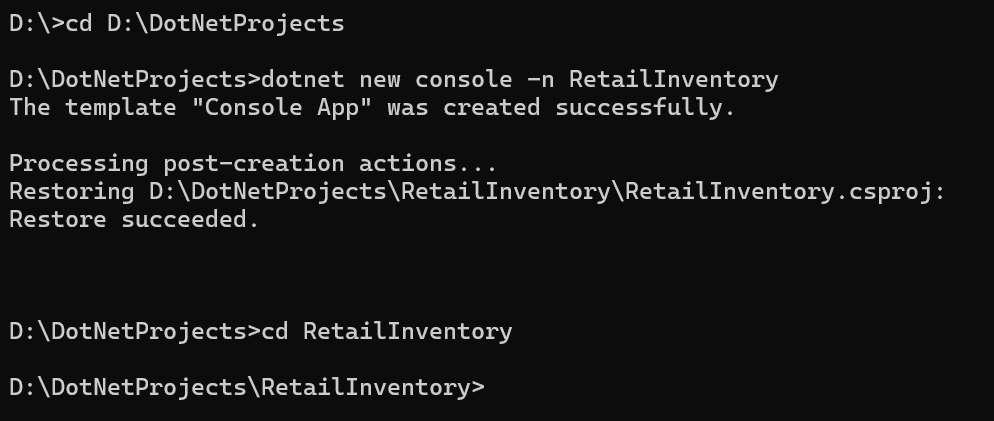
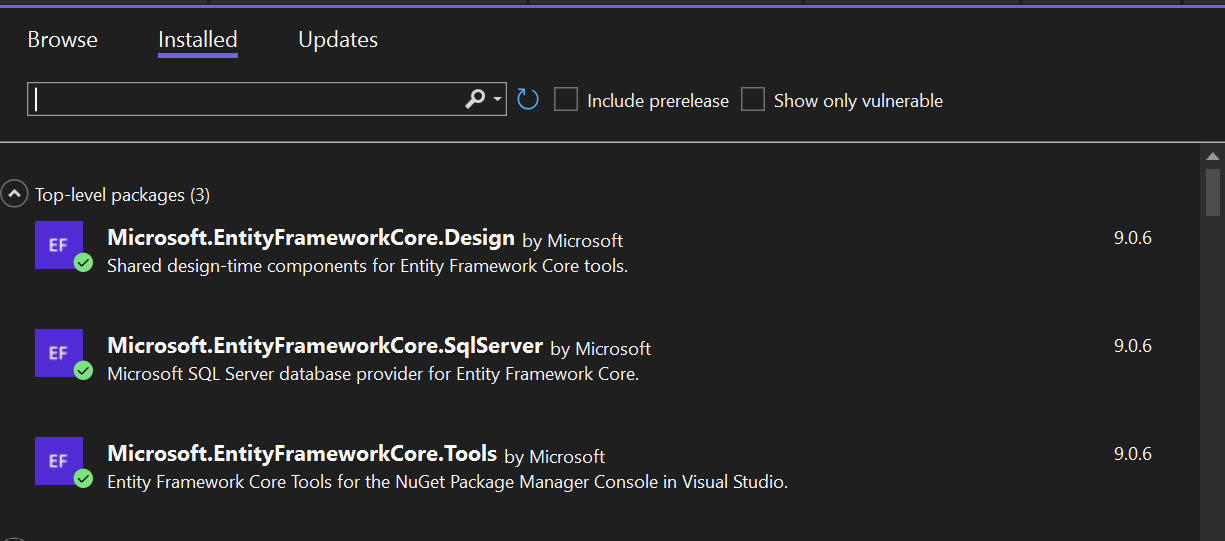
Lab 1: Understanding ORM with a Retail Inventory System

ORM is a method that uses less SQL to link an application's rich objects to tables in a relational database.

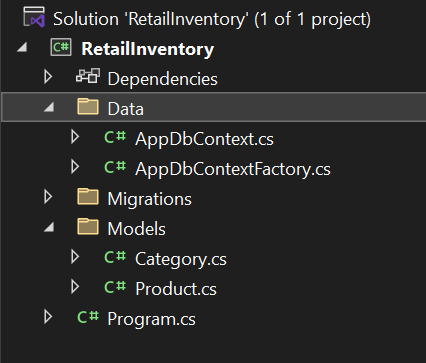
Create a .NET Console App





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

Create Models Folder:



Code For Category.cs in Models Folder:

using System.Collections.Generic;

namespace RetailInventory.Models

{

public class Category

{

public int Id { get; set; }

public string Name { get; set; } = null!;

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

}

}

Code For Product.cs in ModelsFolder:

namespace RetailInventory.Models

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = string.Empty;

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; } = null!;

}

}

Code for AppDbContext in Data Folder:

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory.Data

{

public class AppDbContext : DbContext

{

public DbSet<Product> Products { get; set; } = null!;

public DbSet<Category> Categories { get; set; } = null!;

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Server=(localdb)\\MSSQLLocalDB;Database=RetailDb;Trusted\_Connection=True;");

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Category>()

.HasMany(c => c.Products)

.WithOne(p => p.Category)

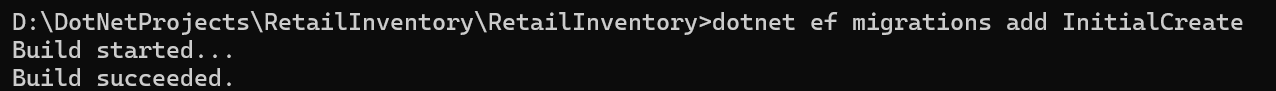
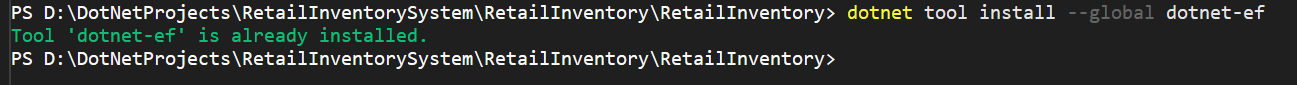
.HasForeignKey(p => p.CategoryId);

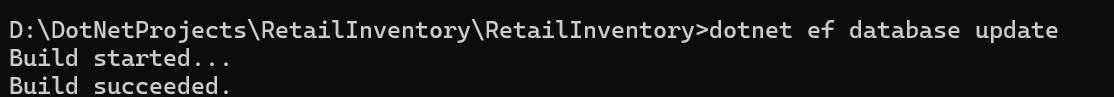
}

}

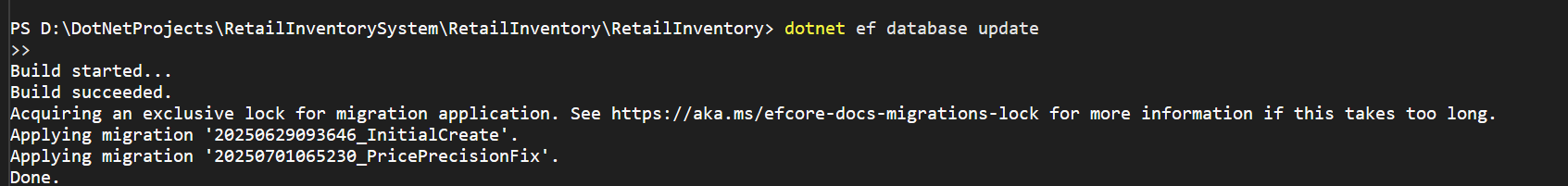
}

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

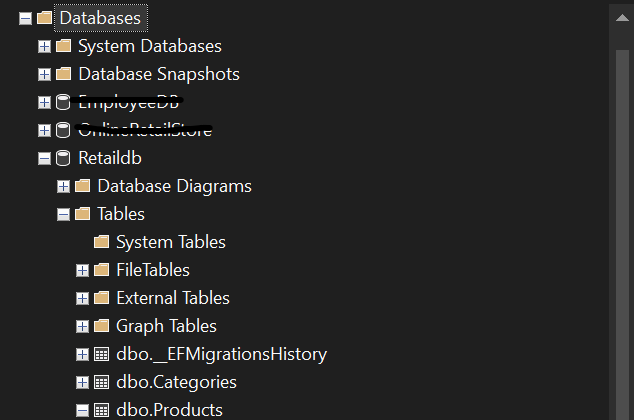




Updating the Database:



Verify SQL Server confirm that tables Products and Categories are created



Lab 4: Inserting Initial Data into the Database

Code for Program.cs in RetailInventory

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using RetailInventory.Models;

Console.WriteLine("Inserting Initial Data...");

using var context = new AppDbContextFactory().CreateDbContext(args);

if (context.Categories.Any() || context.Products.Any())

{

Console.WriteLine("⚠️ Data already exists. Skipping insert.");

}

else

{

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

var clothing = new Category { Name = "Clothing" };

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

var laptop = new Product { Name = "Laptop", Price = 75000m, Stock = 10, Category = electronics };

var tshirt = new Product { Name = "T-Shirt", Price = 999.99m, Stock = 50, Category = clothing };

var headphones = new Product { Name = "Headphones", Price = 2999.99m, Stock = 25, Category = electronics };

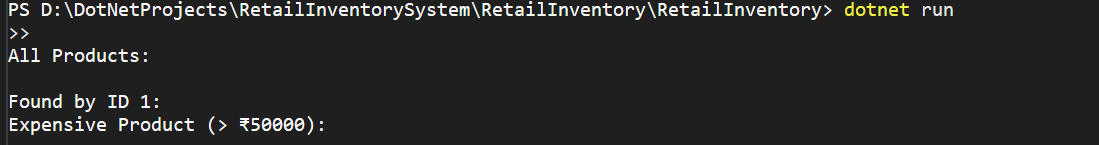
await context.Products.AddRangeAsync(laptop, tshirt, headphones);

await context.SaveChangesAsync();

Console.WriteLine("✅ Initial data inserted.");

}

Run the App and Verify the SQL Server



Lab 5: Retrieving Data from the Database

For Retrieve All Products , Find by ID and First Or Default with Condition

using Microsoft.EntityFrameworkCore;

using RetailInventory.Data;

using RetailInventory.Models;

var context = new AppDbContextFactory().CreateDbContext(args);

if (!context.Categories.Any())

{

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.\n");

}

else

{

Console.WriteLine("Data already exists.\n");

}

Console.WriteLine("All Products:");

var products = await context.Products.Include(p => p.Category).ToListAsync();

foreach (var p in products)

{

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

}

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

Console.WriteLine($"\nFound by ID 1: {productById?.Name ?? "Not Found"}");

var expensiveProduct = await context.Products

.FirstOrDefaultAsync(p => p.Price > 50000);

if (expensiveProduct != null)

{

Console.WriteLine($"Expensive Product (> ₹50000): {expensiveProduct.Name}");

}

else

{

Console.WriteLine("Expensive Product (> ₹50000): None found");

}

Verifying the data

