Superset ID:6362148

Name - Ishita Chatterjee

ENTITY FRAMEWORK CORE 8.0

Lab 1:

1. What is ORM?

ORM stands for Object Relational Mapping. It is a programming technique that allows you to interact with a relational database using the object-oriented paradigm of your programming language (such as C#), instead of writing raw SQL queries.

1. How Database Tables and C# Classes Are Mapped by ORM?

1.Every C# class, such as Product or Category, has a corresponding table in the database.

2.Every property in the class, such as ProductId and Name, corresponds to a table column.

3.Foreign key relationships in the database are mapped to relationships between classes (for example, a category has many products).

3.Benefits of Using ORM

* Productivity: Write less boilerplate code.
* Maintainability: Centralize logic in your models.
* Abstraction: Avoid complex SQL; use LINQ instead.
* Portability: Easily switch databases.

1. EF Core vs EF Framework

**Platform**

EF Core: Works on multiple platforms (Windows, Linux, macOS) using .NET Core and .NET 5/6/7+.

Because it requires the entire.NET Framework, EF6 can only run on Windows.

**Performance**

EF Core: Generally provides better performance and is lightweight.

Although heavier, EF6 is thought to be more stable for large, well-established applications.

**LINQ and Support for Async**

EF Core: Complete support for asynchronous processes and LINQ queries.

EF6: Async support is less effective and more restricted, but LINQ is supported.

**Combined Queries**

Compiled queries are supported by EF Core, which can enhance performance for frequently asked queries.

EF6: Compilation queries are not supported.

**Maturity**

EF Core: A more recent technology that is constantly changing with new releases.

EF6: More experienced, more widely used in business settings, and more mature.

4. EF Core 8.0 New Features

JSON Column Mapping: Store complex objects directly in a single column.

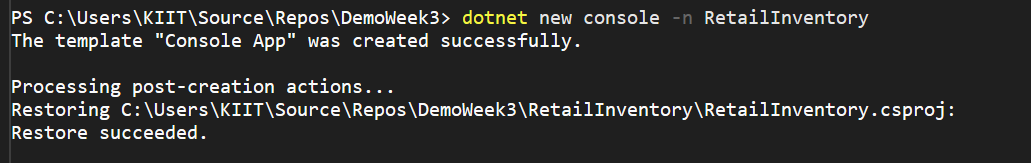
Compiled Models: Speeds up startup performance for large databases

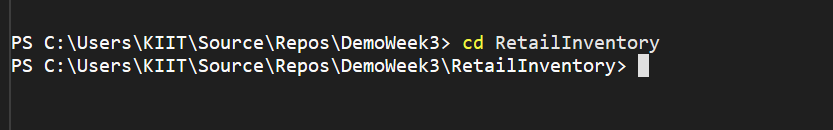
Interceptors: Hook into database calls for logging or validation.

Bulk Operations Improvements: More efficient insert/update/delete.

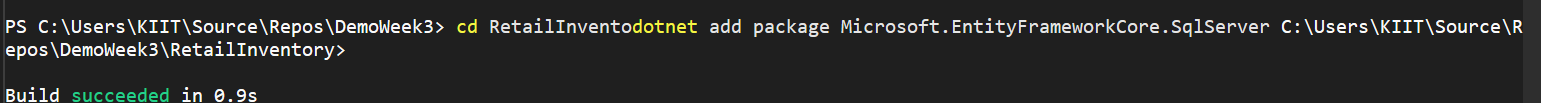
1. Project Setup

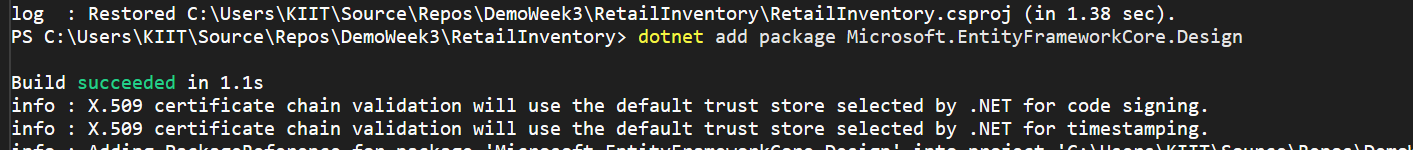
Create a new .NET Console App:



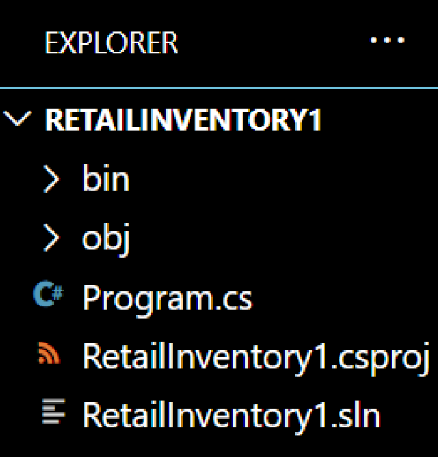


Install EF core packages:





After installation:



Lab 2:

Code:

Program.cs:

using System;

using System.Linq;

using RetailInventory.Models;

namespace RetailInventory

{

class Program

{

static void Main(string[] args)

{

using var context = new AppDbContext();

// Ensure database is created

context.Database.EnsureCreated();

// Add sample data if none exists

if (!context.Categories.Any())

{

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

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

context.Categories.AddRange(electronics, groceries);

context.Products.Add(new Product { Name = "Laptop", Price = 60000, Category = electronics });

context.Products.Add(new Product { Name = "Smartphone", Price = 30000, Category = electronics });

context.Products.Add(new Product { Name = "Rice", Price = 50, Category = groceries });

context.SaveChanges();

}

// Fetch and print all products with categories

var products = context.Products

.Select(p => new { p.Name, p.Price, CategoryName = p.Category.Name })

.ToList();

Console.WriteLine("Products in RetailInventory:");

foreach (var p in products)

{

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

}

}

}

}

Category.cs:

using System.Collections.Generic;

namespace RetailInventory.Models

{

public class Category

{

public int Id { get; set; }

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

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

}

}

Product.cs:

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!;

}

}

AppdbContextLab2.cs:

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

using System.Collections.Generic;

namespace RetailInventory

{

public class AppDbContext:DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

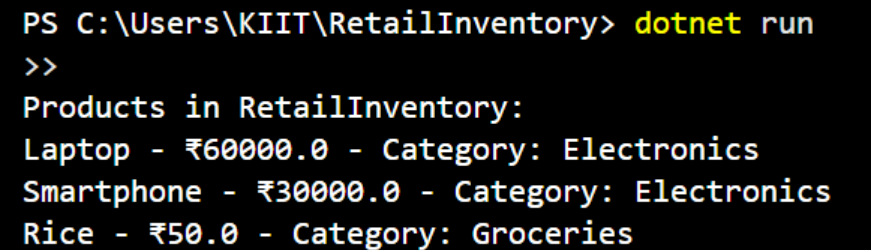
optionsBuilder.UseSqlite("Data Source=retail.db");

}

}

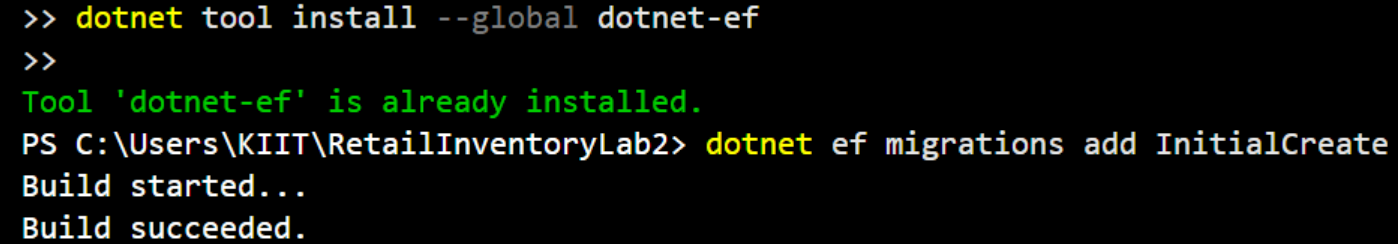
}

Output:

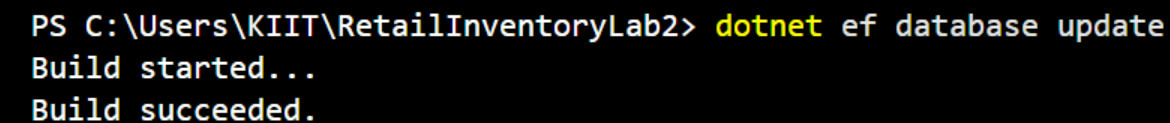


Lab 3:

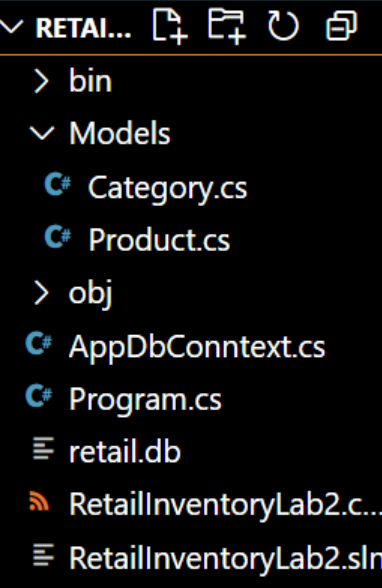
Command to create migration:



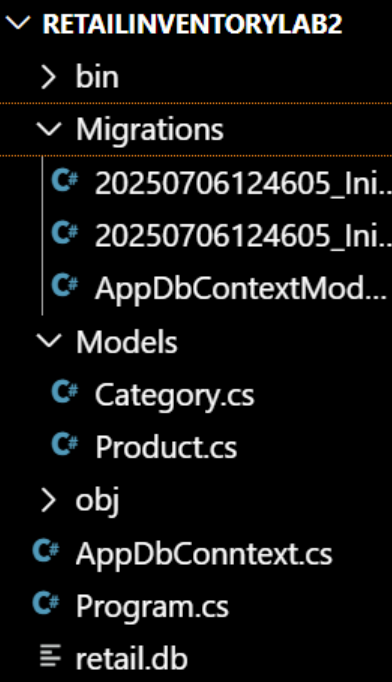
Command to create database:

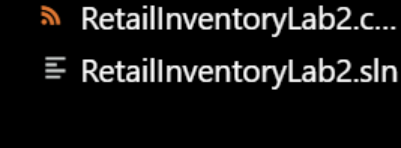


Before Migration and creating database:



After Migration and creating database:





Lab 4:

Code:

**AppDbContextlab4.cs**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory

{

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlite("Data Source=retail.db");

}

}

}

**Categorylab4.cs:**

using System.Collections.Generic;

namespace RetailInventory.Models

{

public class Category

{

public int Id { get; set; }

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

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

}

}

**Productslab4.cs:**

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!;

}

}

ProgramLab4.cs:

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory

{

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// Optional: Only insert data if empty

if (await context.Categories.CountAsync() == 0)

{

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.");

}

else

{

Console.WriteLine("Data already exists. Skipping insertion.");

}

// Optional: Show products

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

Console.WriteLine("\nProducts:");

await foreach (var p in products.AsAsyncEnumerable())

{

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

}

}

}

}

Lab 5:

Code:

**AppDbContextlab5.cs:**

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory

{

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlite("Data Source=retail.db");

}

}

}

**Productlab5.cs:**

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!;

}

}

Categorylab5.cs:

using System.Collections.Generic;

namespace RetailInventory.Models

{

public class Category

{

public int Id { get; set; }

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

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

}

}

**ProgramLab5.cs:**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory

{

class Program

{

static async Task Main(string[] args)

{

using var context = new AppDbContext();

// Seed data if none exists

if (!await context.Categories.AnyAsync())

{

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("Seeded initial data.");

}

// Now retrieve data

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

Console.WriteLine("All Products:");

foreach (var p in products)

{

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

}

Console.WriteLine();

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

Console.WriteLine($"Found product with ID=1: {product?.Name ?? "Not found"}");

Console.WriteLine();

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

Console.WriteLine($"First expensive product (>₹50000): {expensive?.Name ?? "None found"}");

}

}

}

AppDbContextLab5.cs:

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

namespace RetailInventory

{

public class AppDbContext : DbContext

{

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

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

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlite("Data Source=retail.db");

}

}

}

Output:

