**WEEK 3**

**EF Core 8.0 HOL**

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

Models/Category.cs

namespace RetailInventory.Models;

public class Category

{

    public int CategoryId { get; set; }

    // Ensure it's never null by using 'required'

    public required string Name { get; set; }

    // Initialize the list to avoid null reference issues

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

}

Models/Product.CS

namespace RetailInventory.Models;

public class Product

{

    public int ProductId { get; set; }

    public required string Name { get; set; }

    public decimal Price { get; set; }

    public int CategoryId { get; set; }

    public required Category Category { get; set; }

    public required Stock Stock { get; set; }

}

RetailDBContext.CS

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

public class RetailDbContext : DbContext

{

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

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

    public DbSet<Stock> Stocks { get; set; }

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

    {

        // Replace (localdb)\MSSQLLocalDB with your server if different

        optionsBuilder.UseSqlServer(@"Server=DESKTOP-NTP7OEI\HP;Database=RetailInventoryDb;Trusted\_Connection=True;");

    }

}

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

Category.CS

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();

    }

}

Product.CS

namespace RetailInventory.Models

{

    public class Product

    {

        public int Id { get; set; }

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

        public decimal Price { get; set; }

        public int CategoryId { get; set; }

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

    }

}

AppDbContext.cs

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

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=DESKTOP-NTP7OEI\SQLEXPRESS;Database=RetailInventoryDb;Trusted\_Connection=True;");

    }

}

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

AppDbContext.CS

using Microsoft.EntityFrameworkCore;

using RetailInventory.Models;

public class AppDbContext : DbContext

{

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

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

    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

    {

        optionsBuilder.UseSqlServer("Server=DESKTOP-ABC123\\SQLEXPRESS;Database=RetailDB;Trusted\_Connection=True;TrustServerCertificate=True;");

    }

}

**Lab 4: Inserting Initial Data into the Database**

Program.CS

using RetailInventory.Models;

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!");

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

Program.CS

using RetailInventory.Models;

using Microsoft.EntityFrameworkCore;

using var context = new AppDbContext();

// 1. Retrieve all products

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

foreach (var p in products)

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

// 2. Find by ID

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

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

// 3. FirstOrDefault with condition

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

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