EF Core 8.0 Hands-On Labs (Lab 1 to Lab 7)

# Lab 1: Understanding ORM

ORM (Object-Relational Mapping) allows developers to map C# classes to relational database tables. EF Core provides a way to interact with the database using objects instead of writing SQL directly.

# Lab 2: Setting Up the Database Context

**Category.cs**

public class Category  
{  
 public int Id { get; set; }  
 public string Name { get; set; } = string.Empty;  
 public List<Product> Products { get; set; } = new();  
}  
**Product.cs**

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!;  
}  
**AppDbContext.cs**

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=BT-22051323\\SQLEXPRESS;Database=RetailInventoryDB;Trusted\_Connection=True;Encrypt=False;");  
 }  
}

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

Commands used:

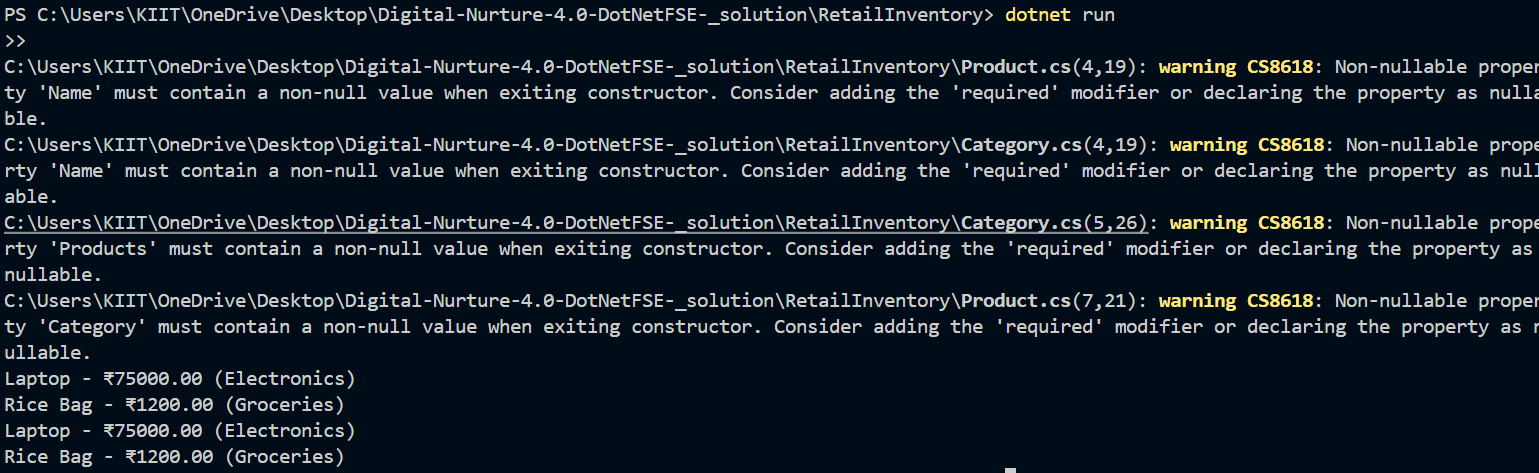
dotnet ef migrations add InitialCreate

dotnet ef database update

# Lab 4: Inserting Initial Data into the Database

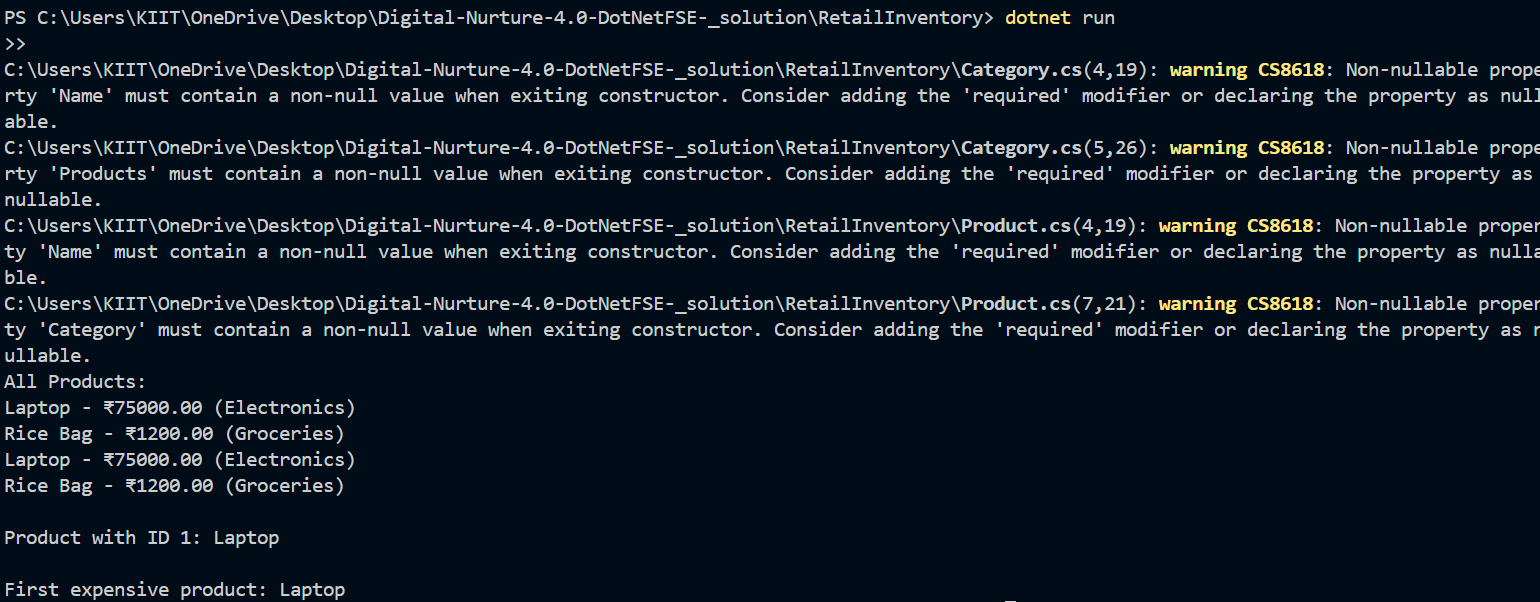
**Program.cs**

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();

**Output:**  


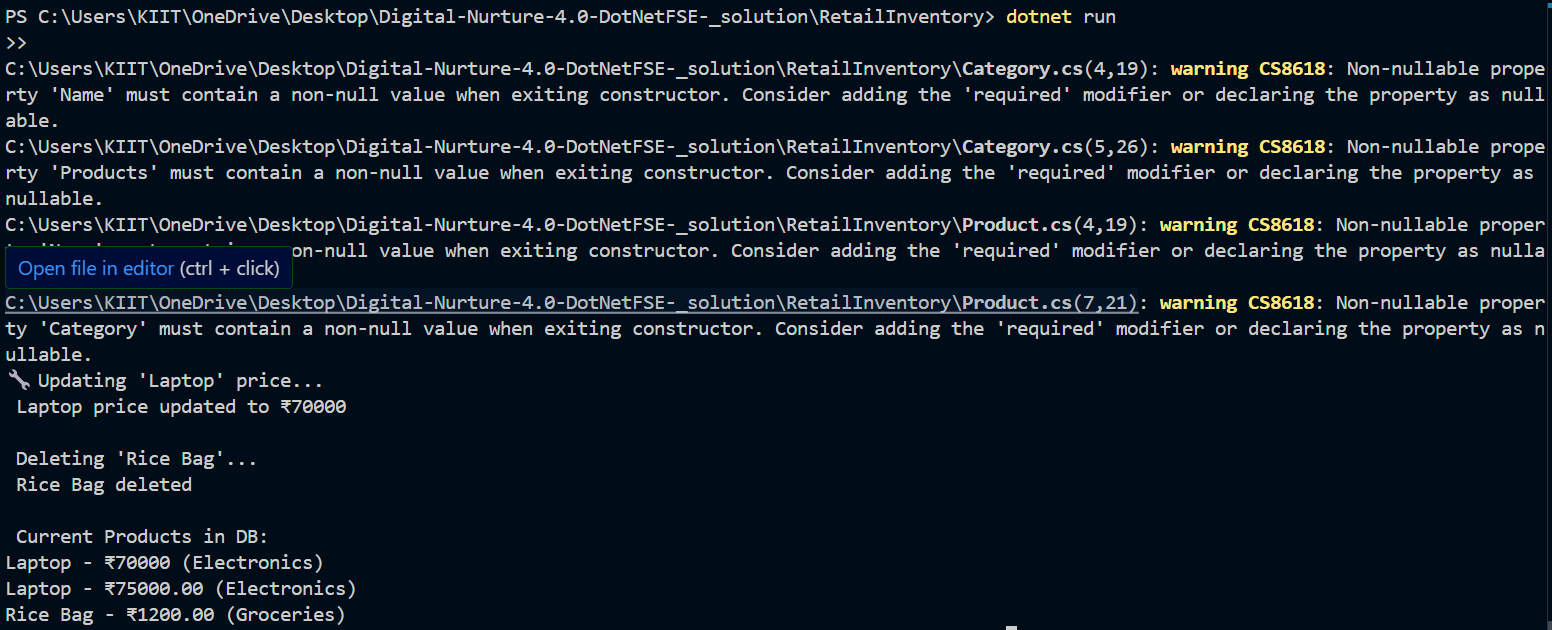
# Lab 5: Retrieving Data from the Database

var products = await context.Products.Include(p => p.Category).ToListAsync();  
foreach (var p in products)  
 Console.WriteLine($"{p.Name} - ₹{p.Price} ({p.Category.Name})");  
  
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}");

**Output:**  


# Lab 6: Updating and Deleting Records

// Update  
var laptop = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");  
if (laptop != null)  
{  
 laptop.Price = 70000;  
 await context.SaveChangesAsync();  
}  
  
// Delete  
var riceBag = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");  
if (riceBag != null)  
{  
 context.Products.Remove(riceBag);  
 await context.SaveChangesAsync();  
}

**Output:**  


# Lab 7: Writing Queries with LINQ

var filtered = await context.Products  
 .Where(p => p.Price > 1000)  
 .OrderByDescending(p => p.Price)  
 .ToListAsync();  
  
foreach (var p in filtered)  
 Console.WriteLine($"{p.Name} - ₹{p.Price}");  
  
var productDTOs = await context.Products  
 .Select(p => new { p.Name, p.Price })  
 .ToListAsync();  
  
foreach (var dto in productDTOs)  
 Console.WriteLine($"{dto.Name} - ₹{dto.Price}");

**Output:**  
