**Week-3: EF Core 8.0 Guided Hands-On Exercises**

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

[Models.cs](http://models.cs)

using System.Collections.Generic;

public class Category

{

public int CategoryId { get; set; }

public string Name { get; set; }

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

}

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

public int Stock { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

[AppDbContext.cs](http://appdbcontext.cs)

using Microsoft.EntityFrameworkCore;

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=(localdb)\MSSQLLocalDB;Database=RetailStoreDb;Trusted\_Connection=True;");

}

}

[Program.cs](http://program.cs)

using System;

using System.Linq;

class Program

{

static void Main()

{

using (var context = new AppDbContext())

{

if (!context.Categories.Any())

{

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

var product = new Product { Name = "Laptop", Stock = 10, Category = category };

context.Add(product);

context.SaveChanges();

}

var products = context.Products

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

.ToList();

Console.WriteLine("Product List:");

foreach (var p in products)

{

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

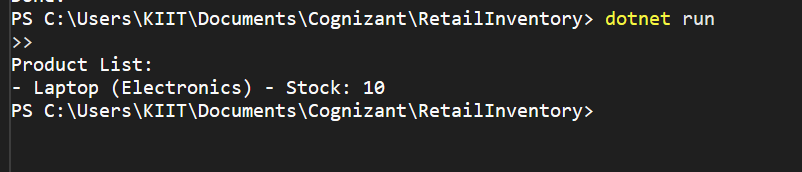
}

}

}

}

**OUTPUT:**



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

[Category.cs](http://category.cs)

using System.Collections.Generic;

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

[Product.cs](http://product.cs)

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

[AppDbContext.cs](http://appdbcontext.cs)

using Microsoft.EntityFrameworkCore;

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=(localdb)\MSSQLLocalDB;Database=RetailStoreDb;Trusted\_Connection=True;");

}

}

**OUTPUT**

Product: Smartphone, Category: Electronics, Price: ₹25000

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

[Category.cs](http://category.cs)

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

[Product.cs](http://product.cs)

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; }

}

[AppDbContext.cs](http://appdbcontext.cs)

public class AppDbContext : DbContext

{

public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

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

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

}

**OUTPUT**

C:\RetailStoreApp> dotnet tool install --global dotnet-ef

You can invoke the tool using the following command: dotnet-ef

Tool 'dotnet-ef' was successfully installed.

C:\RetailStoreApp> dotnet ef migrations add InitialCreate

Build started...

Build succeeded.

Done. To undo this action, use 'dotnet ef migrations remove'.

C:\RetailStoreApp> dotnet ef database update

Build started...

Build succeeded.

Applying migration '20250706121345\_InitialCreate'.

Done.

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

[Program.cs](http://program.cs)

using Microsoft.EntityFrameworkCore;

var options = new DbContextOptionsBuilder()

.UseSqlServer

.Options;

using var context = new AppDbContext(options);

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.");

**OUTPUT**

C:\RetailStoreApp> dotnet run

Initial data inserted successfully.

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

[Program.cs](http://program.cs)

var expensiveProduct = await context.Products

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

if (expensiveProduct != null)

Console.WriteLine($"First expensive product: {expensiveProduct.Name} - ₹{expensiveProduct.Price}");

else

Console.WriteLine("No expensive product found.");

**OUTPUT**

Laptop - ₹75000 - Category: Electronics

Rice Bag - ₹1200 - Category: Groceries

Found: Laptop - ₹75000

First expensive product: Laptop - ₹75000