**SUPERSET ID - 6365402**

**EF Core 8.0 Guided Hands-On Exercises**

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

Scenario:

You’re building an inventory management system for a retail store. The store wants to track products, categories, and stock levels in a SQL Server database.

Objective:

Understand what ORM is and how EF Core helps bridge the gap between C# objects and relational tables.

Steps:

1. What is ORM?

• Explain how ORM maps C# classes to database tables.

• Benefits: Productivity, maintainability, and abstraction from SQL.

Ans:

2. EF Core vs EF Framework:

• EF Core is cross-platform, lightweight, and supports modern features like LINQ, async queries, and compiled queries.

• EF Framework (EF6) is Windows-only and more mature but less flexible.

3. EF Core 8.0 Features:

• JSON column mapping.

• Improved performance with compiled models.

• Interceptors and better bulk operations.

4. Create a .NET Console App:

dotnet new console -n RetailInventory

cd RetailInventory

5. Install EF Core Packages:

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design

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

Scenario:

The retail store wants to store product and category data in SQL Server.

Objective:

Configure DbContext and connect to SQL Server.

Steps:

1. Create Models:

public class Category {

public int Id { get; set; }

public string Name { get; set; }

public List Products { get; set; }

}

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; }

}

2. Create AppDbContext:

public class AppDbContext : DbContext {

public DbSet Products { get; set; }

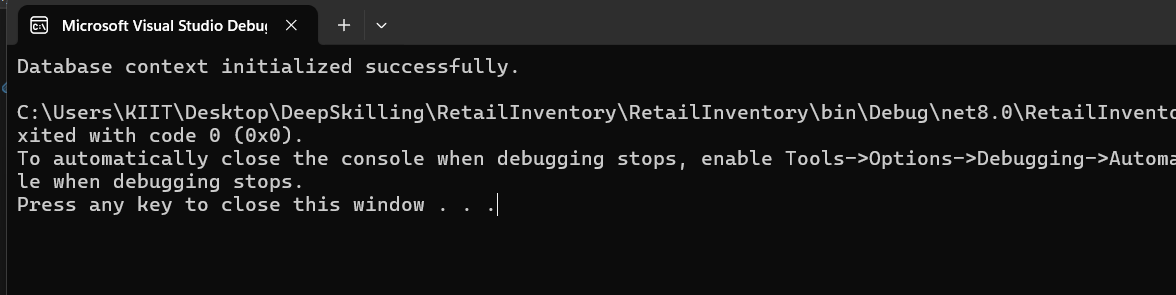
public DbSet Categories { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) {

optionsBuilder.UseSqlServer("Your\_Connection\_String\_Here");

}

}



1. Add Connection String in appsettings.json (optional for ASP.NET Core).

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

Scenario:

The retail store's database needs to be created based on the models you've defined.

You’ll use EF Core CLI to generate and apply migrations.

Objective:

Learn how to use EF Core CLI to manage database schema changes.

Steps:

1. Install EF Core CLI (if not already):

dotnet tool install --global dotnet-ef

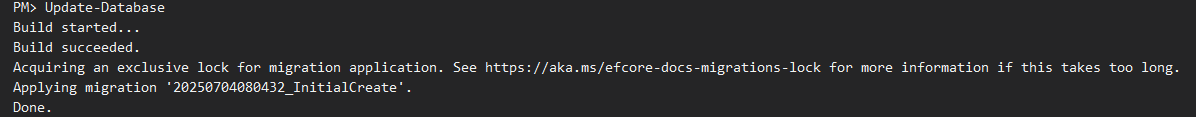
2. Create Initial Migration:

dotnet ef migrations add InitialCreate

This generates a Migrations folder with code that represents the schema.

3. Apply Migration to Create Database:

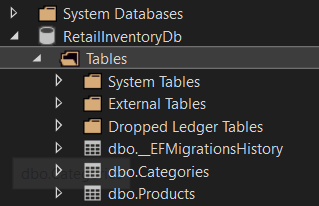
dotnet ef database update



4. Verify in SQL Server:

Open SQL Server Management Studio (SSMS) or Azure Data Studio and confirm

that tables Products and Categories are created.



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

Scenario:

The store manager wants to add initial product categories and products to the system.

Objective:

Use EF Core to insert records using AddAsync and SaveChangesAsync.

Steps:

1. Insert Data in Program.cs:

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 };

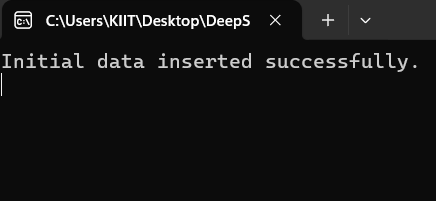
var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };

await context.Products.AddRangeAsync(product1, product2);

await context.SaveChangesAsync();

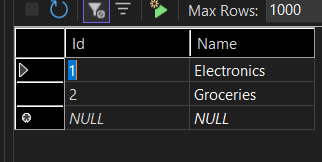
2. Run the App:

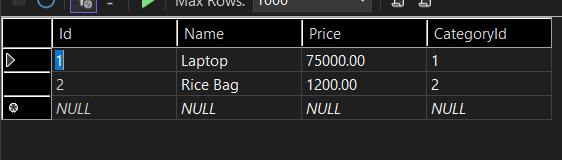
dotnet run



3. Verify in SQL Server:

Check that the data is inserted correctly.





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

Scenario:

The store wants to display product details on the dashboard.

Objective:

Use Find, FirstOrDefault, and ToListAsync to retrieve data.

Steps:

1. Retrieve All Products:

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

foreach (var p in products)

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

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}");

