**Week 3: Entity Framework Core 8.0**

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

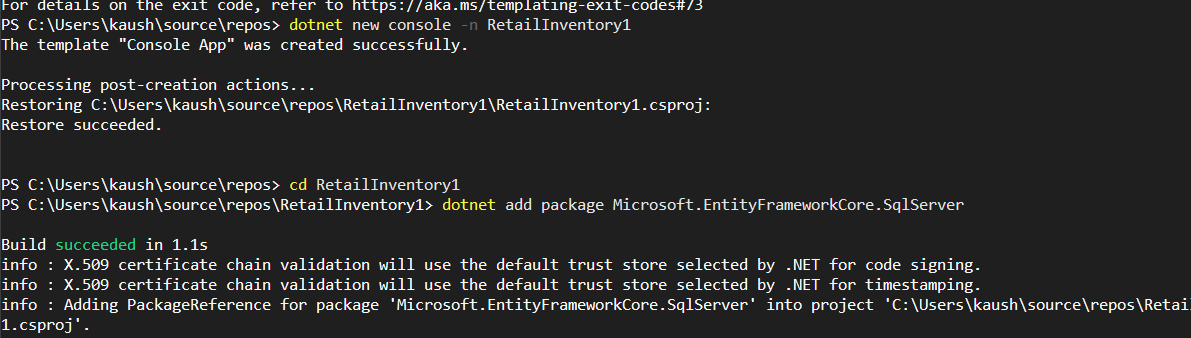
**1.What is ORM?**

* **ORM** means Object Relational Mapping is a way to connect your programming code to a database using objects instead of writing SQL queries.

**Benefits:**

* Working with C# objects for interacting with database
* Easy for maintenance and testing purposes.

**2. Create a .NET Console App**

****

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

**1.Create Models.cs**

using System.Collections.Generic;

public class Category

{

public int Id { get; set; }

public string Name { get; set; }

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

}

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; }

}

**3. Create 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)

{

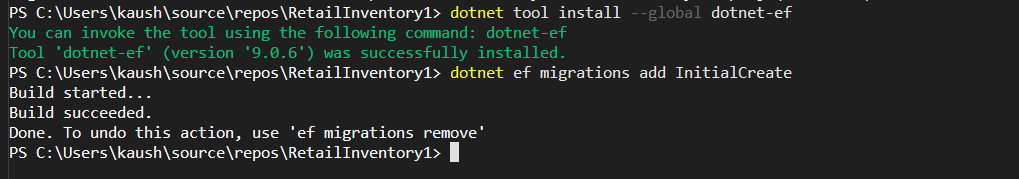
optionsBuilder.UseSqlServer("Server=Kavirajan\\SQLEXPRESS;Database=RetailInventoryDb;Trusted\_Connection=True;TrustServerCertificate=True;");

}

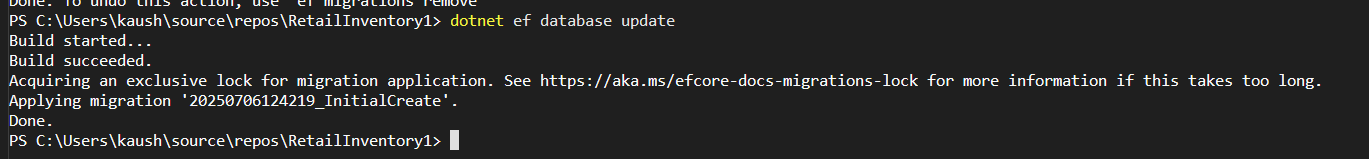
}

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

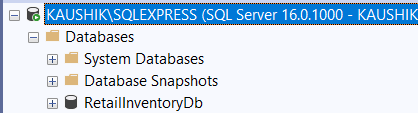
**1.Create Initial Migration:**

****

**2.Apply Migration to Create Database:**

****

**3. Verify in SQL Server:**

****

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

**1.Inserting Data in Program.cs**

using RetailInventory1;

using Microsoft.EntityFrameworkCore;

Console.WriteLine("Inserting initial data...");

using var context = new AppDbContext();

await context.Database.MigrateAsync();

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 };

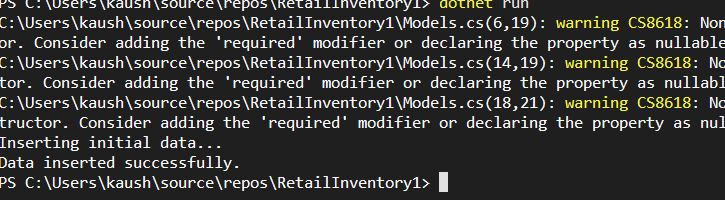
var product2 = new Product { Name = "Rice Bag", Price = 1200, Category = groceries };

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

await context.SaveChangesAsync();

Console.WriteLine("Data inserted successfully.");

**Output:**

****

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

**1.Retrieve All Products:**

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

Console.WriteLine("All Products:");

foreach (var p in products)

{

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

}

**2. Find by ID:**

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

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

**3. FirstOrDefault with Condition:**

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

Console.WriteLine($"First Expensive Product (> ₹50,000): {expensive?.Name ?? "None"}");

**Full Program:**

using RetailInventory1;

using Microsoft.EntityFrameworkCore;

Console.WriteLine("Retrieving data from the database...\n");

using var context = new AppDbContext();

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

Console.WriteLine("All Products:");

foreach (var p in products)

{

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

}

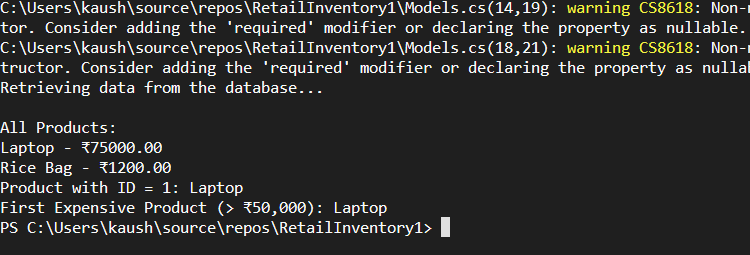
var product = await context.Products.FindAsync(1);

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

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

Console.WriteLine($"First Expensive Product (> ₹50,000): {expensive?.Name ?? "None"}");

**Output:**

****

**Lab 6: Updating and Deleting Records**

**1.Update a Product:**

var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");

if (product != null){

product.Price = 70000;

await context.SaveChangesAsync();

Console.WriteLine($" Updated: {product.Name} price to ₹{product.Price}");

}

else{

Console.WriteLine(" Product 'Laptop' not found for update.");

}

**2. Delete a Product:**

var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");

if (toDelete != null){

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

Console.WriteLine($" Deleted: {toDelete.Name}");

}

else{

Console.WriteLine("Product 'Rice Bag' not found for deletion.");

}

**Full Program:**

using RetailInventory1;

using Microsoft.EntityFrameworkCore;

Console.WriteLine(" Updating and Deleting Products...");

using var context = new AppDbContext();

var product = await context.Products.FirstOrDefaultAsync(p => p.Name == "Laptop");

if (product != null){

product.Price = 70000;

await context.SaveChangesAsync();

Console.WriteLine($" Updated: {product.Name} price to ₹{product.Price}");

}

else{

Console.WriteLine(" Product 'Laptop' not found for update.");

}

var toDelete = await context.Products.FirstOrDefaultAsync(p => p.Name == "Rice Bag");

if (toDelete != null){

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

Console.WriteLine($"Deleted: {toDelete.Name}");

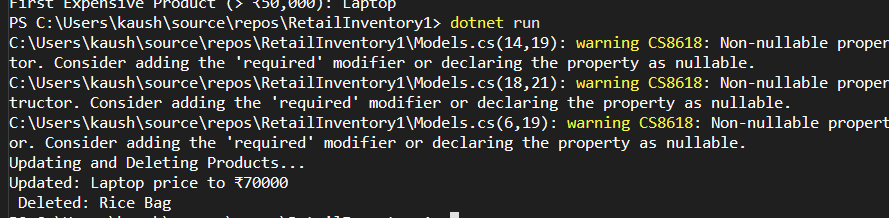
}

else{

Console.WriteLine(" Product 'Rice Bag' not found for deletion.");

}

**Output:**

****

**Lab 7: Writing Queries with LINQ**

**1.Filter and Sort:**

var filtered = await context.Products

.Where(p => p.Price > 1000)

.OrderByDescending(p => p.Price)

.ToListAsync();

Console.WriteLine("\n Filtered & Sorted Products (Price > 1000):");

foreach (var p in filtered)

{

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

}

**2. Product into DTO:**

var productDTOs = await context.Products

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

.ToListAsync();

Console.WriteLine("\n Product DTOs:");

foreach (var dto in productDTOs)

{

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

}

**Full Program:**

using RetailInventory1;

using Microsoft.EntityFrameworkCore;

Console.WriteLine(" Querying Products with LINQ...");

using var context = new AppDbContext();

var filtered = await context.Products

.Where(p => p.Price > 1000)

.OrderByDescending(p => p.Price)

.ToListAsync();

Console.WriteLine("\n Filtered & Sorted Products (Price > 1000):");

foreach (var p in filtered)

{

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

}

var productDTOs = await context.Products

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

.ToListAsync();

Console.WriteLine("\n Product DTOs:");

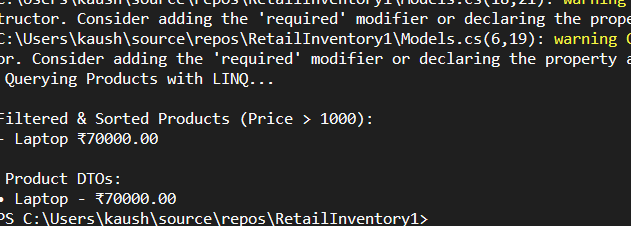
foreach (var dto in productDTOs)

{

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

}

**Output:**

****