Aman Raj

6358186

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

**What is ORM?**

Object-Relational Mapping (ORM) is a technique that allows you to interact with a database using your programming language's objects (in this case, C#).

EF Core is an ORM tool that maps C# classes to SQL tables, and properties to columns.

**Benefits of ORM (EF Core):**

Productivity – No need to write raw SQL for most operations.

Maintainability – Cleaner, easier-to-read code with LINQ.

Abstraction – Handles database logic while you focus on domain logic.

**EF Core 8.0 Highlights**

JSON column mapping – Store JSON in SQL Server 2022 columns.

Compiled Models – Reduces startup time and improves performance.

Interceptors – Hook into EF events (e.g., logging SQL, auditing).

Improved bulk operations – Better performance in mass updates/inserts.

using RetailInventorySystem.Models;

using System;

using System.Linq;

class Program

{

static void Main()

{

using var context = new RetailContext();

// Ensure database is created

context.Database.EnsureCreated();

// Seed data only if Products table is empty

if (!context.Products.Any())

{

var products = new[]

{

new Product { Name = "Laptop", Quantity = 10, Price = 75000 },

new Product { Name = "Mouse", Quantity = 50, Price = 700 },

new Product { Name = "Keyboard", Quantity = 30, Price = 1500 },

new Product { Name = "Monitor", Quantity = 15, Price = 12000 }

};

context.Products.AddRange(products);

context.SaveChanges();

Console.WriteLine("✅ Initial products added to database.\n");

}

// Retrieve and display all products

Console.WriteLine("📦 Product Inventory:\n");

var productList = context.Products.ToList();

foreach (var product in productList)

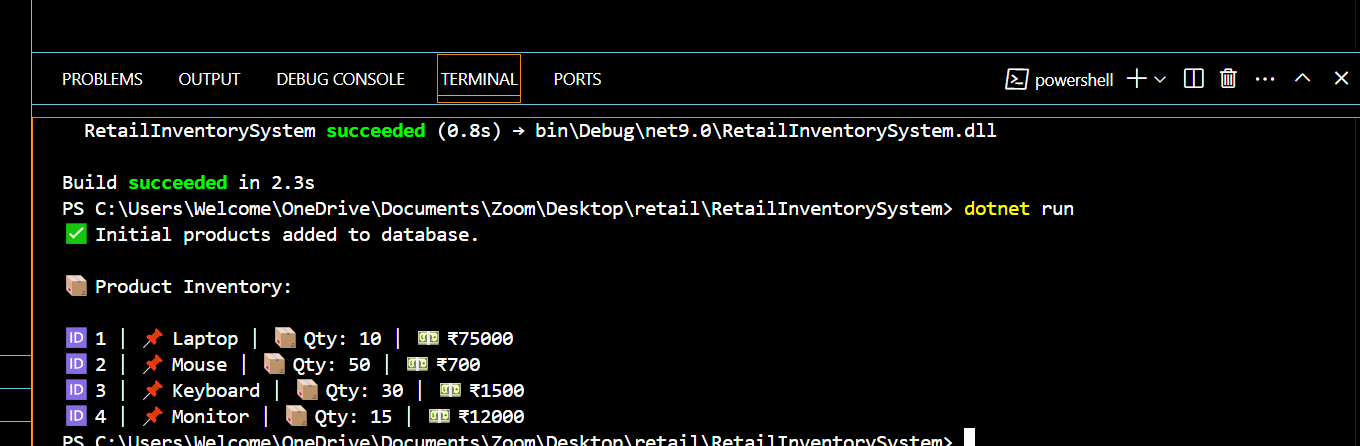
{

Console.WriteLine($"🆔 {product.ProductId} | 📌 {product.Name} | 📦 Qty: {product.Quantity} | 💵 ₹{product.Price}");

}

}

}

****

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

using RetailStoreApp;

using RetailStoreApp.Models;

class Program

{

static void Main()

{

using var context = new RetailContext();

// Add product

var product = new Product

{

Name = "Notebook",

Quantity = 50,

Price = 89.99m

};

context.Products.Add(product);

context.SaveChanges();

// Read and display

var products = context.Products.ToList();

foreach (var p in products)

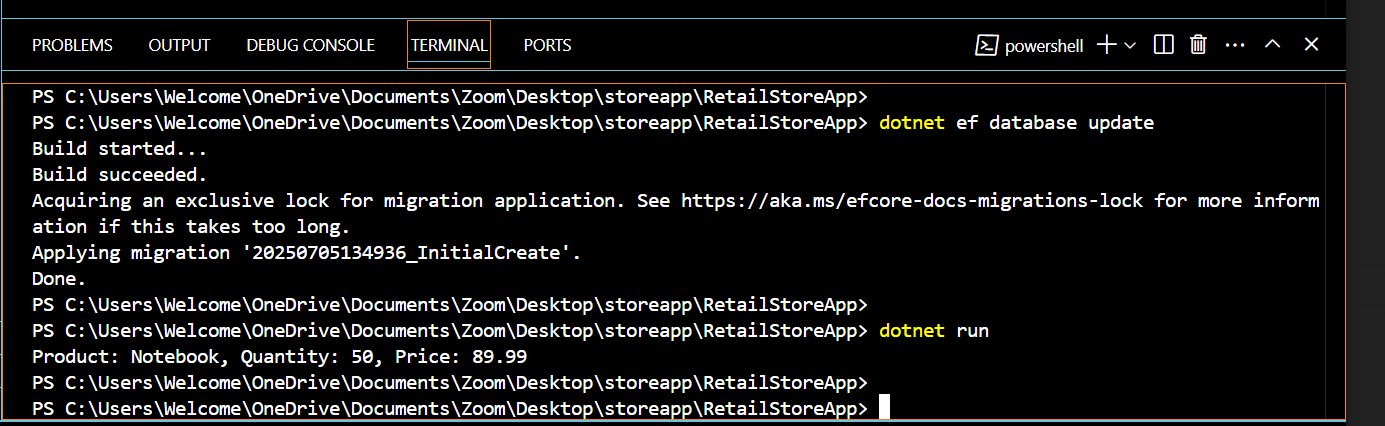
{

Console.WriteLine($"Product: {p.Name}, Quantity: {p.Quantity}, Price: {p.Price}");

}

}

}

****

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

using var context = new RetailContext();

if (!context.Products.Any())

{

context.Products.Add(new Product { Name = "Keyboard", Quantity = 10, Price = 1499 });

context.Products.Add(new Product { Name = "Mouse", Quantity = 25, Price = 599 });

context.SaveChanges();

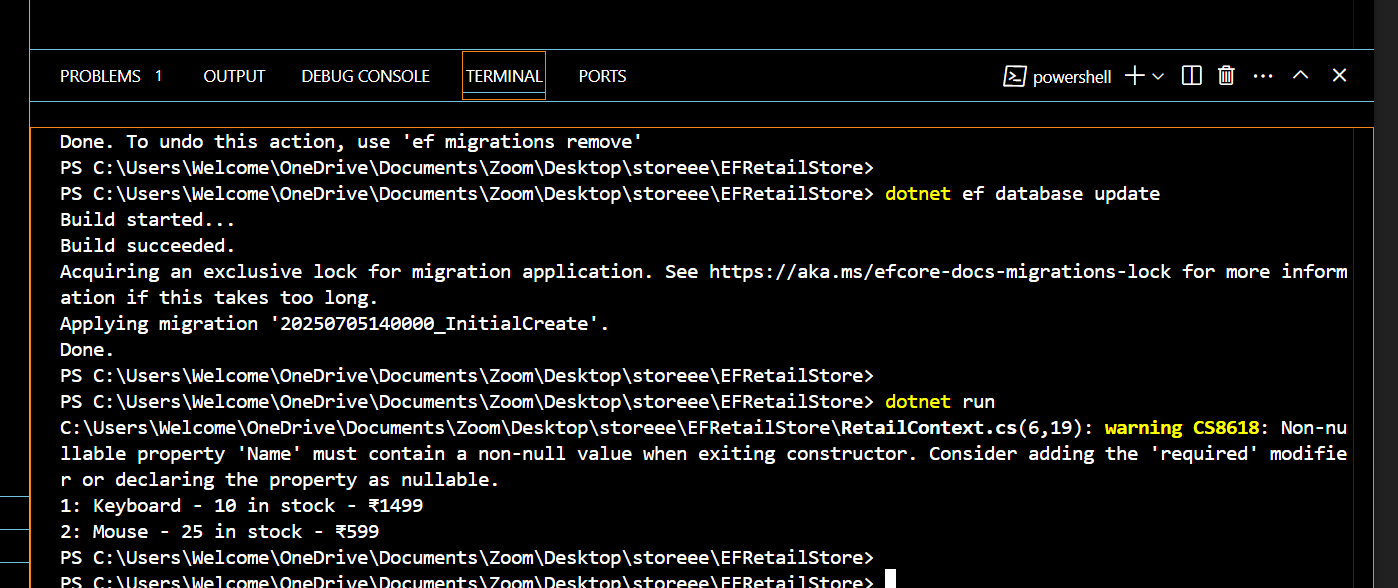
}

foreach (var product in context.Products)

{

Console.WriteLine($"{product.Id}: {product.Name} - {product.Quantity} in stock - ₹{product.Price}");

}

****

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

using EFCoreSeedDemo.Data;

using (var context = new AppDbContext())

{

var products = context.Products.ToList();

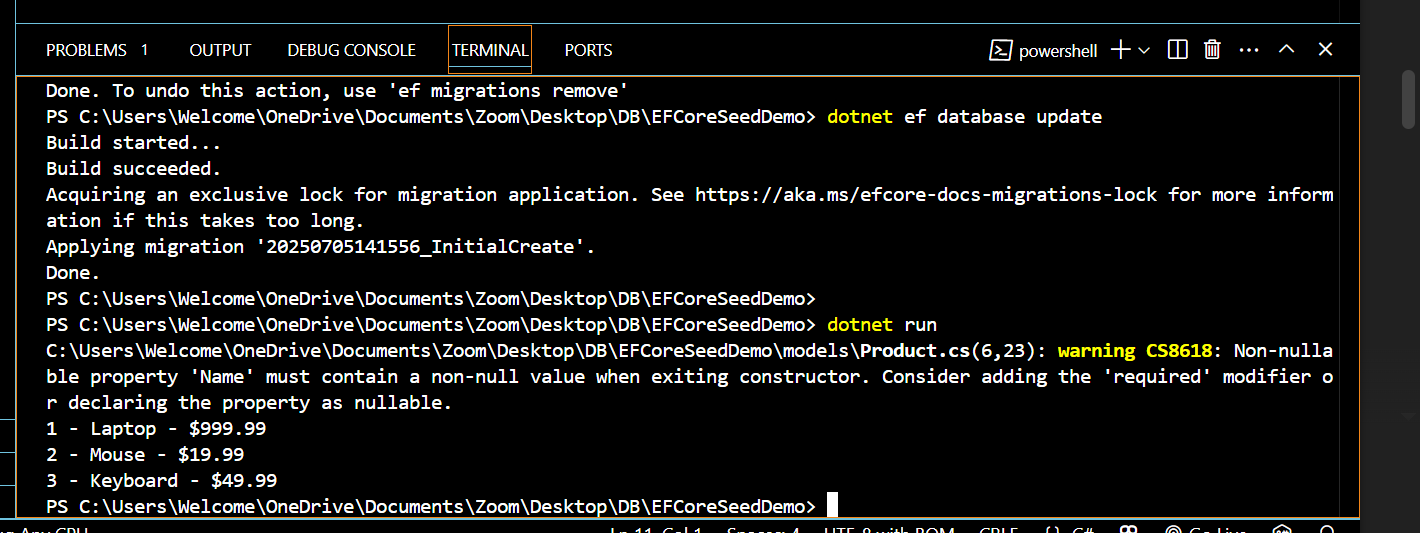
foreach (var product in products)

{

Console.WriteLine($"{product.Id} - {product.Name} - {product.Price:C}");

}

}

****

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

using RetailStoreApp.Data;

using RetailStoreApp.Models;

class Program

{

static void Main()

{

using var context = new RetailDbContext();

Console.WriteLine("📦 Product List:");

var products = context.Products.ToList();

foreach (var product in products)

{

Console.WriteLine($"🆔 {product.ProductId} | 📛 {product.Name} | 💰 ₹{product.Price} | 📦 Qty: {product.Quantity}");

}

}

}

****