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

1. **What is ORM?**
2. It is a programming technique allows developers to interact with a database using object oriented code rather than writing sql queries.

**2. How does it work?**

1. ORM maps C# classes to database tables and class properties to table columns.

public class Product

{

public int ProductId { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

}

By using **get** and **set** we can easily retrieve and insert the data into the database using c#.

**Benefits:**

**Productivity:** Developers write less code and avoid manual SQL.  
 **Maintainability:** Changes in models are reflected across the application easily.  
 **Abstraction:** Simplifies data access by removing the need for complex SQL.  
 **Security:** Reduces SQL injection risks by using parameterized queries.

**2. EF Core vs EF Framework:**

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

**EF Framework (EF6):**  This is a 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 RetailInventorySystem

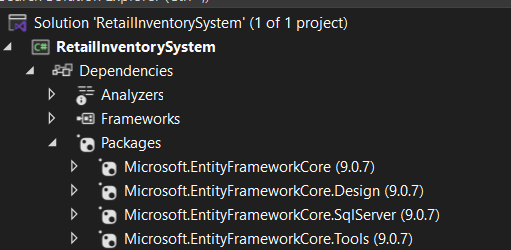
cd RetailInventorySystem

**5. Install EF Core Packages:**

dotnet add package Microsoft.EntityFrameworkCore.SqlServer

dotnet add package Microsoft.EntityFrameworkCore.Design

**Lab 1 REsult:**

****

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

**Procedure:**

* In this i created the models folder and i added the model files as [category.cs](http://category.cs) and [product.cs](http://product.cs) under the models folder and saved it.

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

using RetailInventoryApp.Models;

namespace RetailInventorySystem.Models

{

public class Category

{

public int CategoryId { get; set; }

public string? CategoryName { get; set; }

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

}

}

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

using RetailInventorySystem.Models;

namespace RetailInventoryApp.Models

{

public class Product

{

public int ProductId { get; set; }

public string? Name { get; set; }

public decimal Price { get; set; }

public int CategoryId { get; set; }

Public Category? Category { get; set; }

}

}

And under the RetailInventorySystem project I added the [AppDbContext.cs](http://appdbcontext.cs) then I installed the sqlexpress and ssm for visual studios and I added the connecting string inside the code to connect the database with our project.

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

using Microsoft.EntityFrameworkCore;

using RetailInventoryApp.Models;

using RetailInventorySystem.Models;

namespace RetailInventorySystem

{

public class AppDbContext : DbContext

{

public DbSet<Product> Products => Set<Product>();

public DbSet<Category> Categories => Set<Category>();

protected override void OnConfiguring(DbContextOptionsBuilder options)

{

options.UseSqlServer("Server=localhost\\SQLEXPRESS;Database=RetailDB;Trusted\_Connection=True;TrustServerCertificate=True;");

}

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

modelBuilder.Entity<Product>()

.Property(p => p.Price)

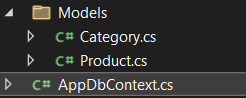
.HasPrecision(18, 2); // prevent precision warning

}

}

}

**Lab 2 result:**

****

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

After creating these files i opened the package manager console and and execute the commands like:

Add-Migration InitialCreate

Add-Migration AddPricePrecision

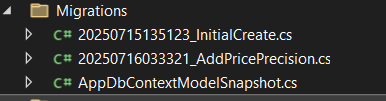
Update-Database

After the migration I can update the database using this command.

database update

And after verify the data base is connected then after i modify the code in the [program.cs](http://program.cs)

Result:



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

I created a [program.cs](http://program.cs) code to insert the data into the database.

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

using Microsoft.EntityFrameworkCore;

using RetailInventoryApp.Models;

using RetailInventorySystem;

using RetailInventorySystem.Models;

Console.WriteLine("Adding Categories and Products");

using var db = new AppDbContext();

var categoryElectronics = new Category { CategoryName = "Electronics" };

var categoryGroceries = new Category { CategoryName = "Groceries" };

await db.Categories.AddRangeAsync(categoryElectronics, categoryGroceries);

var laptop = new Product { Name = "Laptop", Price = 75000, Category = categoryElectronics };

var riceBag = new Product { Name = "Rice Bag", Price = 1200, Category = categoryGroceries };

await db.Products.AddRangeAsync(laptop, riceBag);

await db.SaveChangesAsync();

Console.WriteLine("\nData insert successfully.\n");

var products = await db.Products.Include(p => p.Category).ToListAsync();

Console.WriteLine("Prod Invent:");

foreach (var product in products)

{

Console.WriteLine($"• {product.Name} - ₹{product.Price} ({product.Category?.CategoryName})");

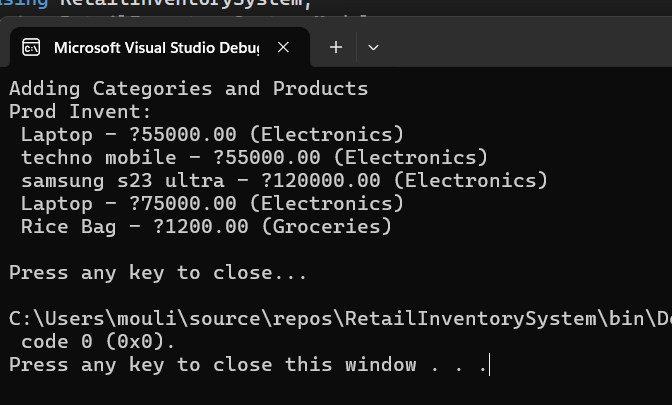
}

Console.WriteLine("\nPress any key to close...");

Console.ReadKey();

**Result:**

After retrieving the data after inserting.



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

**Solution:**

Now after the data inserted inside the database I can use the methods Find, FirstOrDefault, and ToListAsync to retrieve data.

And I updated the [program.cs](http://program.cs) class file with the above methods.

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

using Microsoft.EntityFrameworkCore;

using RetailInventorySystem;

using RetailInventorySystem.Models;

Console.WriteLine("Prod Data Ret");

using var context = new AppDbContext();

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

Console.WriteLine("\nAll Available Prods:");

foreach (var item in allProducts)

{

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

}

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

Console.WriteLine($"\nProd with ID 1: {(productById != null ? productById.Name : "Not Found")}");

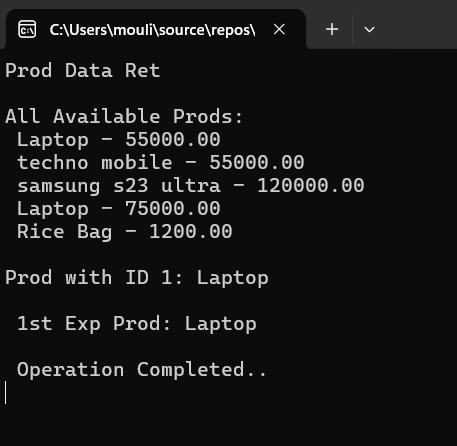
var highPricedItem = await context.Products.FirstOrDefaultAsync(p => p.Price > 50000);

Console.WriteLine($"\n 1st Exp Prod: {(highPricedItem != null ? highPricedItem.Name : "No Prod Found")}");

Console.WriteLine("\n Operation Completed..");

Console.ReadKey();

**Result:**

****