**Week-3 Entity Framework Core 8.0**

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

**1: What is ORM?**

**Definition:**

**ORM (Object-Relational Mapping)** allows you to map **C# classes** to **SQL database tables**.

|  |  |  |
| --- | --- | --- |
| **Concept** | **C# Class** | **SQL Table** |
| Object | Product | Products |
| Property | Product.Name | Name column |
| Query | db.Products.ToList() | SELECT \* FROM Products |

**Benefits:**

* **Productivity**: You write **C# code**, not raw SQL
* **Maintainability**: Code-first approach is easier to update
* **Abstraction**: EF handles SQL behind the scenes

**2: EF Core vs EF Framework**

|  |  |  |
| --- | --- | --- |
| **Feature** | **EF Core** | **EF Framework (EF6)** |
| Cross-platform | Yes | No (Windows only) |
| Lightweight | Yes | No |
| Async Support | Yes (natively) | Limited |
| JSON Mapping | EF Core 8+ | Not Supported |
| Maturity | Still evolving | Very stable |

**3: EF Core 8.0 Cool Features**

1. **JSON Column Mapping**  
   → You can map a nested C# object to a single JSON column in SQL
2. **Compiled Models = Faster Startup**  
   → Great for large schemas
3. **Bulk Operations & Interceptors**  
   → Custom logic when saving changes, and better performance in bulk inserts/updates

**4: Create Console App**

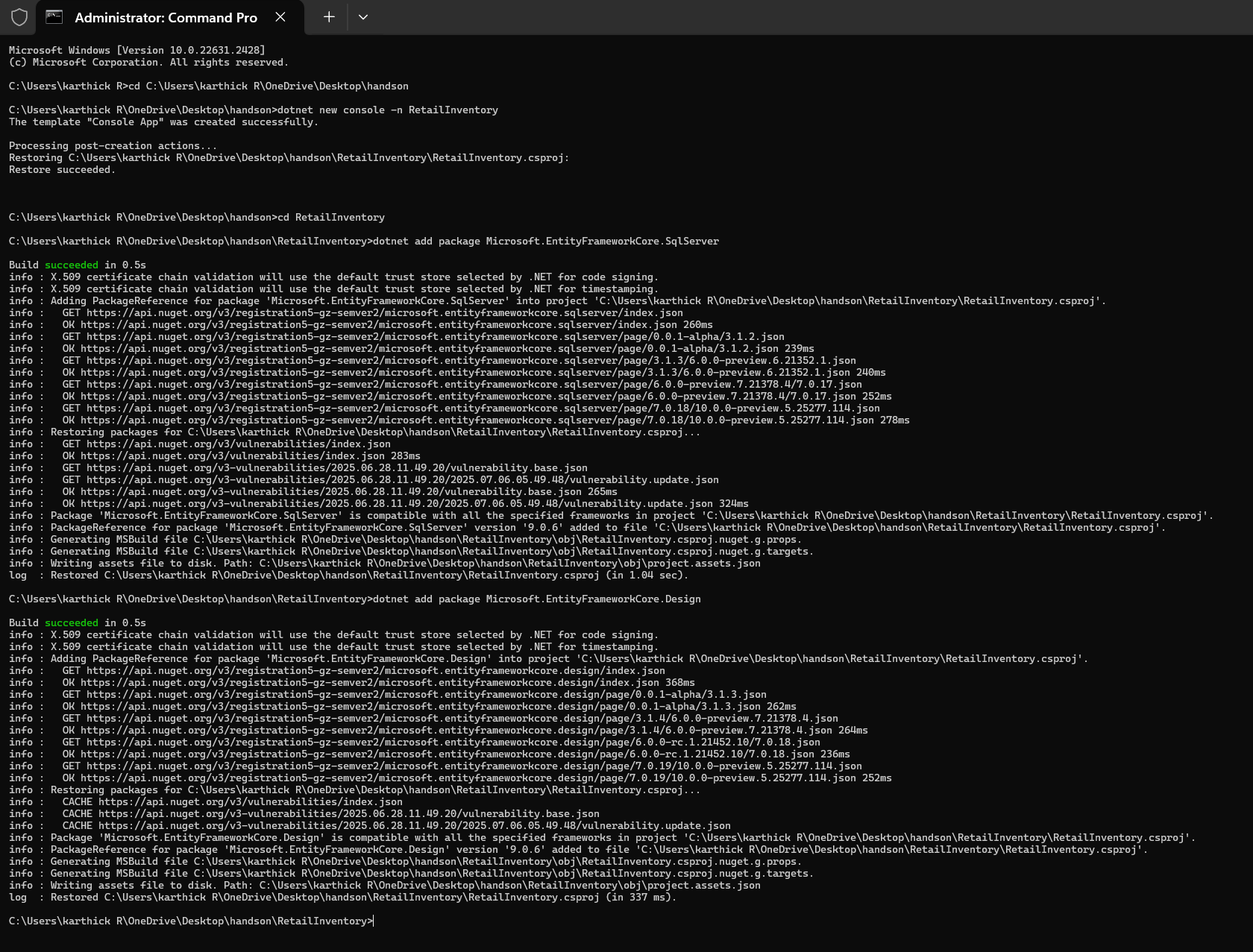
dotnet new console -n RetailInventory

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**

**1: Create Models**

**Models/Category.cs**

namespace Models;

public class Category

{

public int Id { get; set; }

public string Name { get; set; } = default!;

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

}

**Models/Product.cs**

namespace Models;

public class Product

{

public int Id { get; set; }

public string Name { get; set; } = default!;

public decimal Price { get; set; }

public int CategoryId { get; set; }

public Category Category { get; set; } = default!;

}

**2: Create AppDbContext**

**Data/AppDbContext.cs**

using Microsoft.EntityFrameworkCore;

using Models;

namespace Data;

public class AppDbContext : DbContext

{

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

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

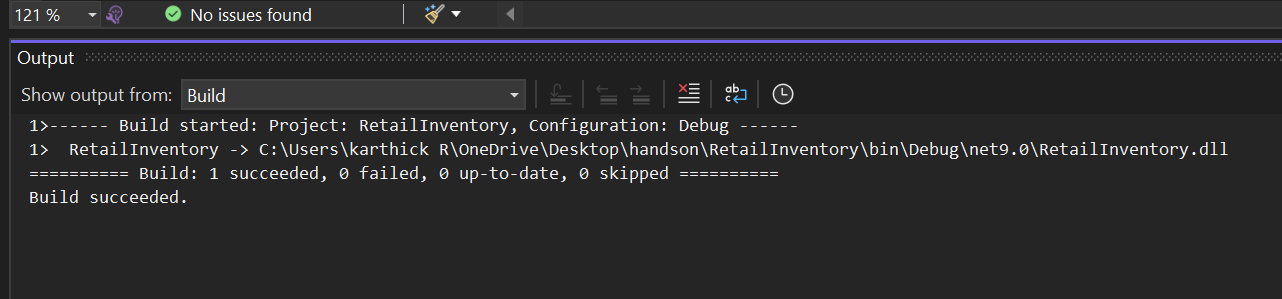
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Server=YOUR\_SERVER\_NAME;Database=RetailInventoryDB;Trusted\_Connection=True;TrustServerCertificate=True;");

}

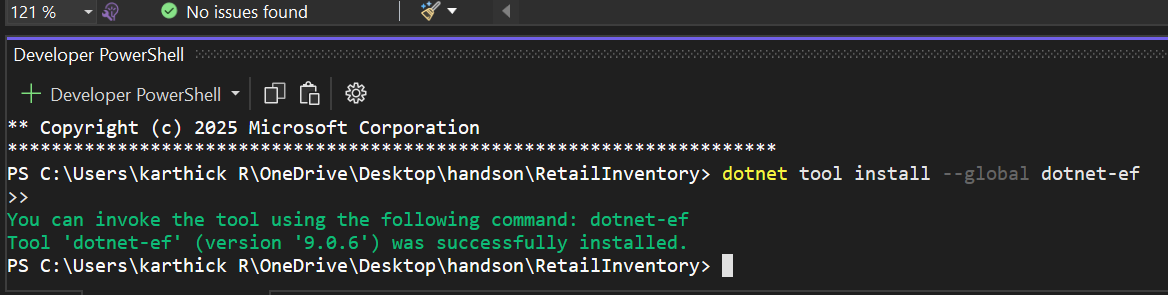
}



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

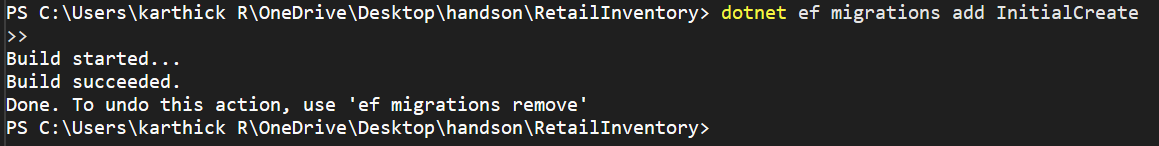
**1: Install EF Core CLI**

dotnet tool install --global dotnet-ef



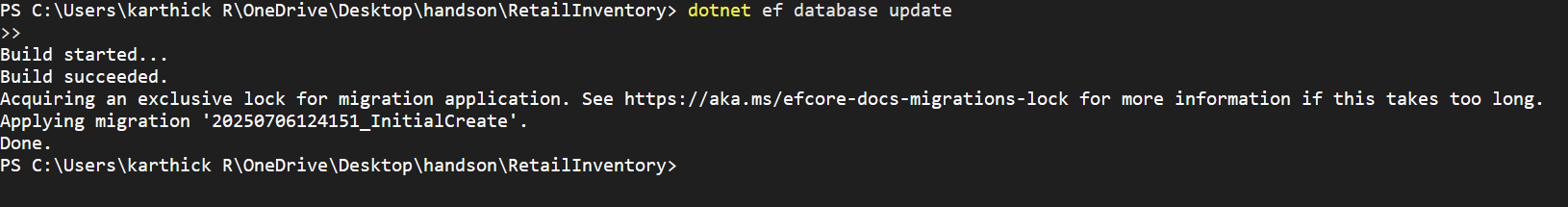
**2. Create Initial Migration**

dotnet ef migrations add InitialCreate

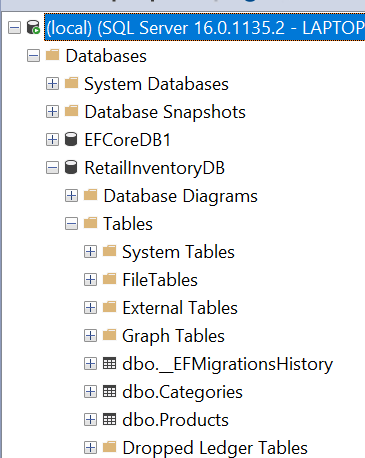


**3. Apply Migration to Create Database**

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**

**1. Insert Data in Program.cs:**

using Data;

using Models;

class Program

{

static async Task Main()

{

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();

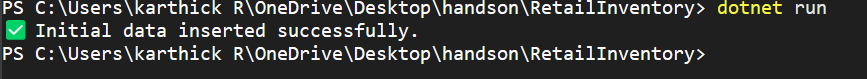
Console.WriteLine("Initial data inserted successfully.");

}

}

**2. Run the App:**

dotnet run

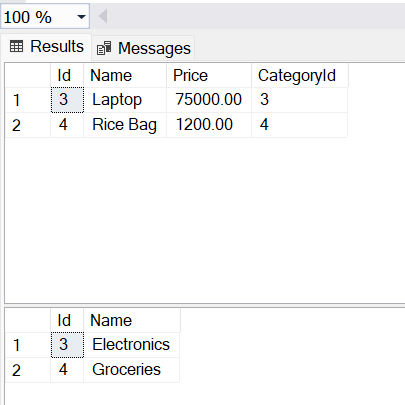


**3. Verify in SQL Server:**

Check that the data is inserted correctly

SELECT \* FROM Products;

SELECT \* FROM Categories;



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

using Data;

using Microsoft.EntityFrameworkCore;

using Models;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Retrieve All Products

Console.WriteLine("All Products:");

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

foreach (var p in products)

{

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

}

// Find by ID

Console.WriteLine("\nFind by ID:");

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

Console.WriteLine($"Found: {product?.Name}");

// FirstOrDefault with Condition

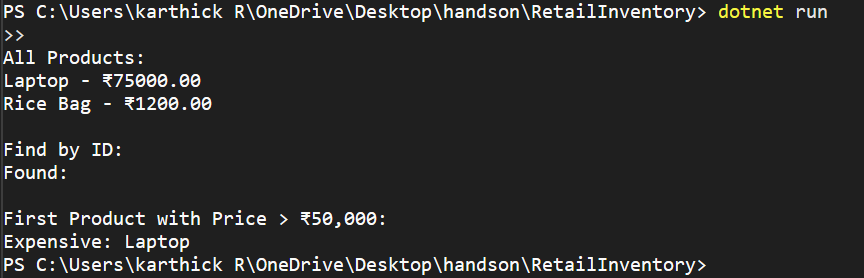
Console.WriteLine("\nFirst Product with Price > ₹50,000:");

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

Console.WriteLine($"Expensive: {expensive?.Name ?? "None found"}");

}

}



**Lab 6: Updating and Deleting Records**

using Data;

using Microsoft.EntityFrameworkCore;

using Models;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Update the price of "Laptop"

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

if (product != null)

{

Console.WriteLine($"Updating price of {product.Name} from ₹{product.Price} to ₹70000...");

product.Price = 70000;

await context.SaveChangesAsync();

Console.WriteLine("Price updated successfully.");

}

else

{

Console.WriteLine("'Laptop' not found.");

}

// Delete the product "Rice Bag"

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

if (toDelete != null)

{

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

context.Products.Remove(toDelete);

await context.SaveChangesAsync();

Console.WriteLine("Product deleted successfully.");

}

else

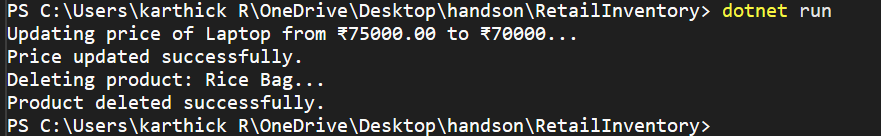
{

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

}

}

}

****

**Lab 7: Writing Queries with LINQ**

using Data;

using Microsoft.EntityFrameworkCore;

using Models;

class Program

{

static async Task Main()

{

using var context = new AppDbContext();

// Filter and Sort: Price > 1000, Descending

Console.WriteLine("Filtered Products (Price > ₹1000, Sorted Desc):");

var filtered = await context.Products

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

.OrderByDescending(p => p.Price)

.ToListAsync();

foreach (var product in filtered)

{

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

}

Console.WriteLine("\nProjected DTO List:");

// Project into DTO (Anonymous Type)

var productDTOs = await context.Products

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

.ToListAsync();

foreach (var dto in productDTOs)

{

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

}

}

}

