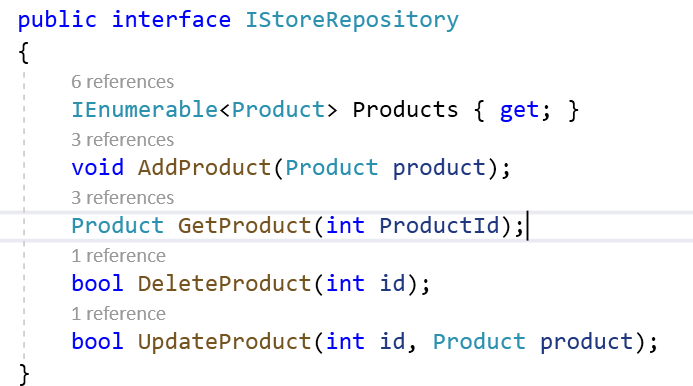
Complete Delete and Update Functionality

1. Change IStoreRepository Interface



1. Implement those functionality

public bool UpdateProduct(int id, Product product)

{

var pro = products.Single(x => x.ProductID == id);

pro.Name = product.Name;

pro.MfgDate = product.MfgDate;

pro.Price = product.Price;

pro.Category = product.Category;

pro.Description = product.Description;

return true;

}

public bool DeleteProduct(int id)

{

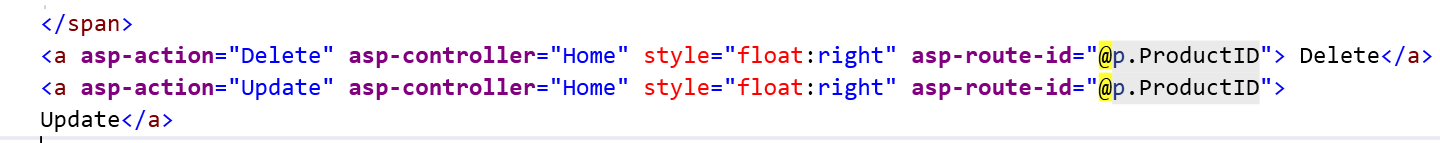
var product = products.Single(x => x.ProductID == id);

products.Remove(product);

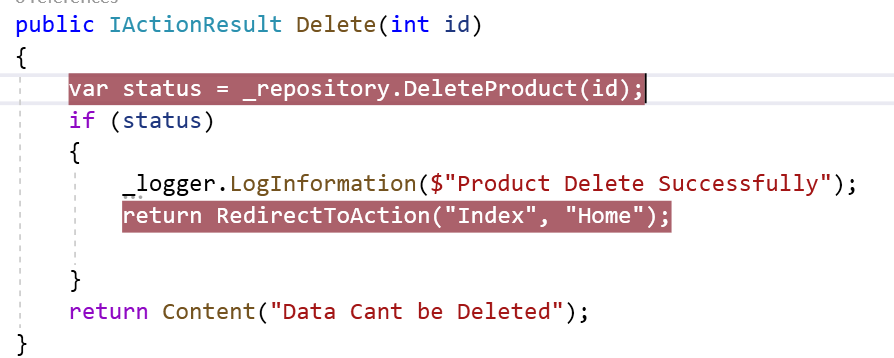
return true;

}

1. Add Buttons to UI and call these functionality in Index.cshtml



1. Add Action for the Same



1. Check the Output for Delete
2. Add a code for Update

public IActionResult Update(int id)

{

var product = \_repository.GetProduct(id);

ProductViewModel productViewModel = new ProductViewModel

{

ProductID = product.ProductID,

Name = product.Name,

Price = product.Price,

Category = product.Category,

Description=product.Category,

MfgDate=product.MfgDate

};

return View(productViewModel);

}

[HttpPost]

public IActionResult Update(Product product)

{

if (ModelState.IsValid)

{

var status = \_repository.UpdateProduct(Convert.ToInt32(product.ProductID), product);

if (status)

{

return RedirectToAction("Index", "Home");

}

else

{

return Content("Data Cant Be Updated");

}

}

else

{

return View();

}

}

1. Add a code for Update.cshtml

@model sample\_app.ViewModels.ProductViewModel

<h3>Update Product</h3>

<hr />

<form **asp-action**="Update" **asp-controller**="Home">

<div class="form-group">

<label **asp-for**="@Model.ProductID">ID</label>

<input **type**="text" **asp-for**="@Model.ProductID" class="form-control" placeholder="ID">

<span **asp-validation-for**="@Model.ProductID" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.Name">Name</label>

<input **type**="text" **asp-for**="@Model.Name" class="form-control" placeholder="Name">

<span **asp-validation-for**="@Model.Name" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.Price">Price</label>

<input **type**="text" **asp-for**="@Model.Price" class="form-control" placeholder="Price">

<span **asp-validation-for**="@Model.Price" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.Category">Category</label>

<input **type**="text" **asp-for**="@Model.Category" class="form-control" placeholder="Category">

<span **asp-validation-for**="@Model.Category" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.Description">Description</label>

<input **type**="text" **asp-for**="@Model.Description" class="form-control" placeholder="Description">

<span **asp-validation-for**="@Model.Description" class="text-danger"></span>

</div>

<div class="form-group">

<label **asp-for**="@Model.MfgDate">Mfg Date</label>

<input **type**="text" **asp-for**="@Model.MfgDate" class="form-control" placeholder="Mfg Date">

<span **asp-validation-for**="@Model.MfgDate" class="text-danger"></span>

</div>

<button type="submit" class="btn btn-default">Update</button>

</form>

1. Check the Output

Entity Framework : ORM

Object

Relation

StoreContext

(connString)

StoreDB

Application

tracks

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Price | Category |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Product

ID

Name

Price

Category

Entity Framework Core

* Lightweight , Extensible , Open Source and Cross Platform version of the Popular EF data access technology.
* EF Core is well integrated into ASP.NEt Core Platform
* Good Support for creating a database from C# classes and for creating C# classes to represent existing database.

Compare : EF core with EF

[Compare EF6 and EF Core | Microsoft Docs](https://docs.microsoft.com/en-us/ef/efcore-and-ef6/)

Advantages

* .Net Developers to work with Database using .Net Objects
* Eliminates the need for most of the data access code that typically needs to be written.

EF Supports

1. Generate a model from existing database.
2. Once a model is created, use EF migrations to create database from the model. Migrations allow evolving the database as the model changes.

Comparison between EF6 with EF Core : [Compare EF6 and EF Core | Microsoft Docs](https://docs.microsoft.com/en-us/ef/efcore-and-ef6/)

* EF Core Supports Azure CosmosDB,In Memory Database for testing that EF 6 doesn’t support.
* EF6 has Graphical Visualization of model Not Available with EF Core
* EF6 supports EDMX format Not Available with EF Core
* Create model from database from VS wizard Not Available with EF Core

EFCore 3. Runs on .Net Core and .Net Framework., the next version of EF 5 will not support .Net Framework.

Guidance for Existing EF6 applications

1. Keep using EF6 if data access code is stable and not likely to evolve or need new features
2. Use EF core if the data access code is evolving.

Install Libraries Required

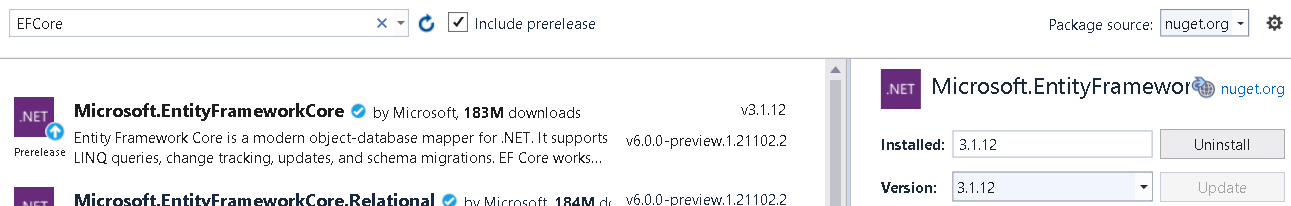
1. Microsoft.EntityFrameworkCore
2. Microsoft.EntityFrameworkCore.Design
3. Microsoft.EntityFrameworkCore.SqlServer

Steps To Connect to DB using EF Core

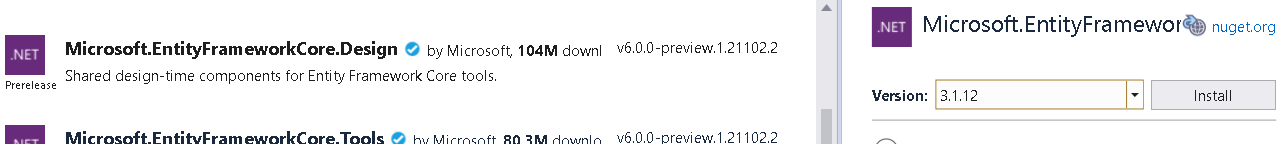
1. Create Context Class
2. Add ConnectionString in appSetting.json
3. Add Context Support for SQL Server in ConfigureServices
4. Dependency Injection to the Controller

Lab : EF

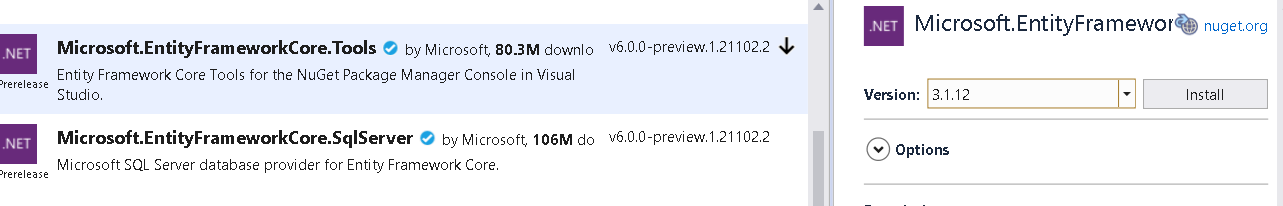
1. EF core requires packages 3 packages to be added to the project.
2. Microsoft.EntityFramework.Core



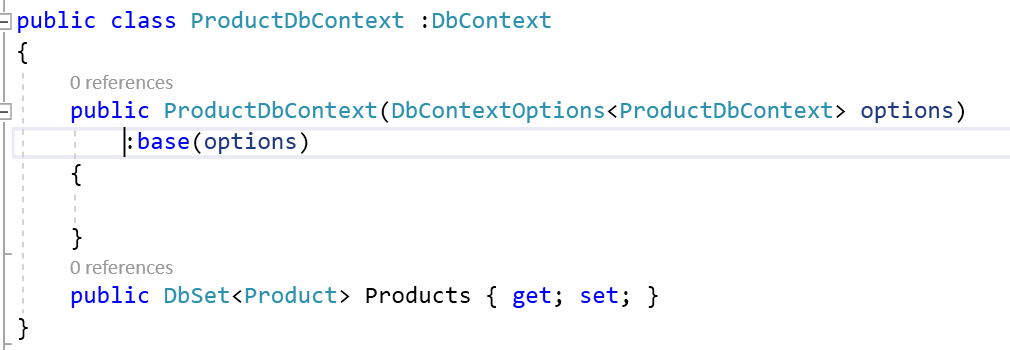
1. Microsoft.EntityFrameworkCore.Design



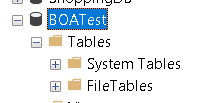
1. Microsoft.EntityFrameworkCore.SqlServer



1. EF Core Provides DatabaseContext , that gives us access to the database.



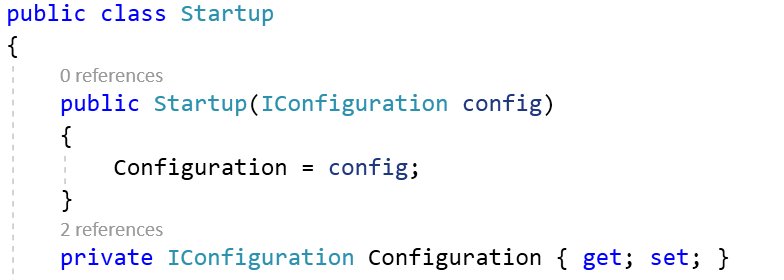
1. Create Database



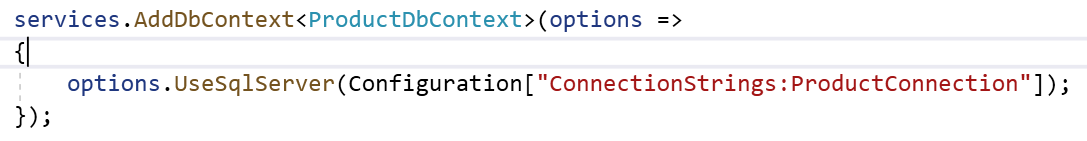
1. Add ConnectionString in appSetting.json



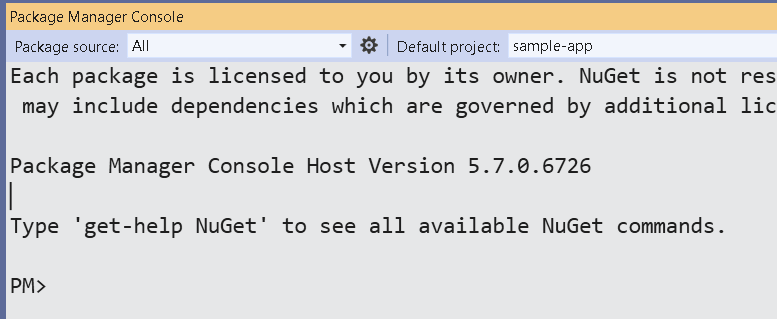
1. Add Configuration Property



1. Pass this ConnectionString to Context in ConfigureServices



1. Ef Core Manages the relationship between data model classes and the database using migrations. When changes are made to model classes , a new migration is created that modifies the database to match those changes.
2. Open Package Manager Console



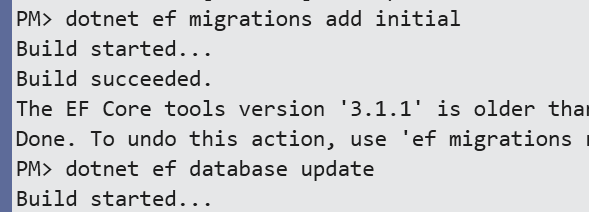
1. If its not working install Tools Library for EFCore



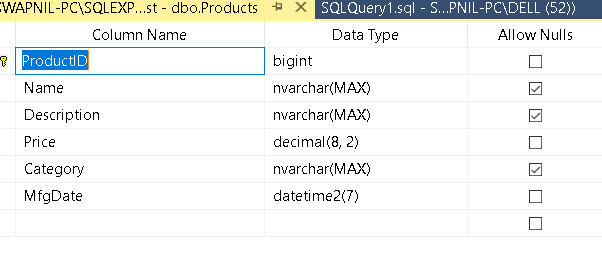
dotnet add package Microsoft.EntityFrameworkCore.Design --version 3.1.1

dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 3.1.1

1. Add migrations and Update database



1. It will create the needed tables



1. Lets add some mock data to it using Seeding : Most Application require some seed data , during development . database seeding provides this f feature

Create SeedData.cs File

public class SeedData

{

private readonly ProductDbContext \_context;

public SeedData(ProductDbContext context)

{

\_context = context;

}

public void SeedDatabase()

{

if(\_context.Products.Count() == 0)

{

\_context.Products.Add(new Product

{

Name="Bat",Category="Cricket",Description="Cricket Bat",MfgDate=DateTime.Now,Price=30.3m

});

\_context.Products.Add(new Product

{

Name = "Ball",

Category = "Soccer",

Description = "Soccer Ball",

MfgDate = DateTime.Now,

Price = 40.3m

});

\_context.Products.Add(new Product

{

Name = "Board",

Category = "Chess",

Description = "This is Chess Board",

MfgDate = DateTime.Now,

Price = 10.3m

});

\_context.Products.Add(new Product

{

Name = "Cricket Ball",

Category = "Cricket",

Description = "Cricket Ball",

MfgDate = DateTime.Now,

Price = 30.3m

});

\_context.Products.Add(new Product

{

Name = "Net",

Category = "Soccer",

Description = "Soccer Ball",

MfgDate = DateTime.Now,

Price = 40.3m

});

\_context.Products.Add(new Product

{

Name = "Pawns",

Category = "Chess",

Description = "This is Chess Pawns",

MfgDate = DateTime.Now,

Price = 10.3m

});

\_context.Products.Add(new Product

{

Name = "Stumps",

Category = "Cricket",

Description = "Cricket Stumps",

MfgDate = DateTime.Now,

Price = 30.3m

});

\_context.Products.Add(new Product

{

Name = "Whistle",

Category = "Soccer",

Description = "Soccer Whistle",

MfgDate = DateTime.Now,

Price = 40.3m

});

\_context.Products.Add(new Product

{

Name = "Timer",

Category = "Chess",

Description = "This is Chess Timer",

MfgDate = DateTime.Now,

Price = 10.3m

});

\_context.SaveChanges();

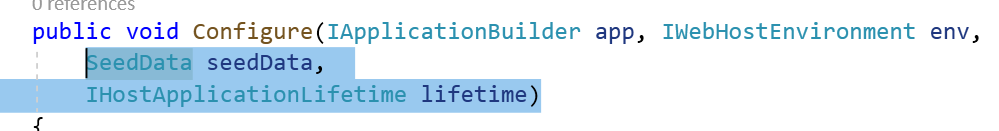
}

}

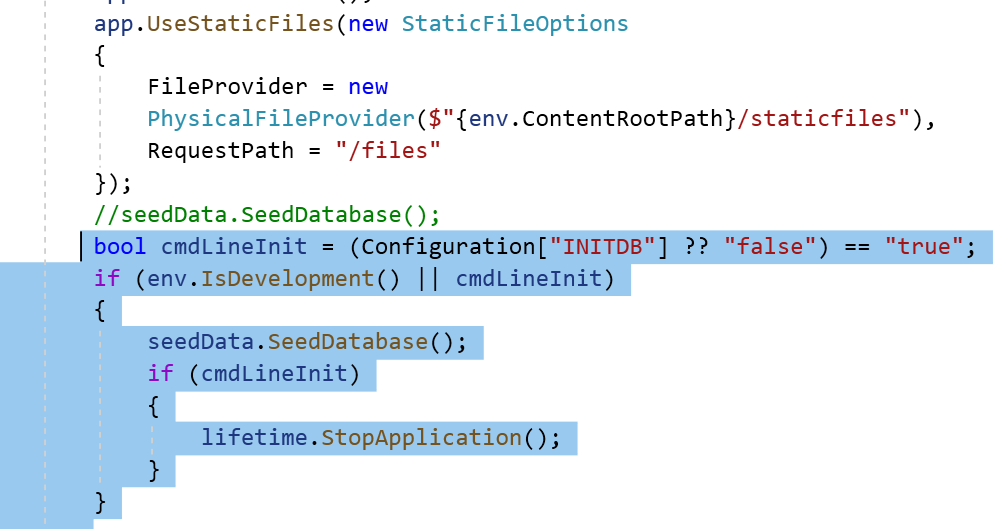
1. Inject Services Required to Run this mock data



1. Inject Various Other Services in Request Pipeline to Run the SeedData file



1. Add following code in Request Pipeline when InitDB=true is passed as a parameter



// Statement checks for the configuration settings named InitDB

bool cmdLineInit =Convert.ToBoolean( Configuration["INITDB"]);

if (env.IsDevelopment () || cmdLineInit)

{

seedData.SeedDatabase();

if (cmdLineInit)

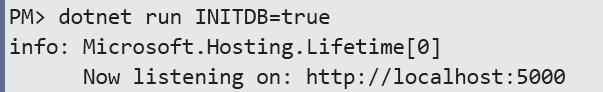
{

lifetime.StopApplication();

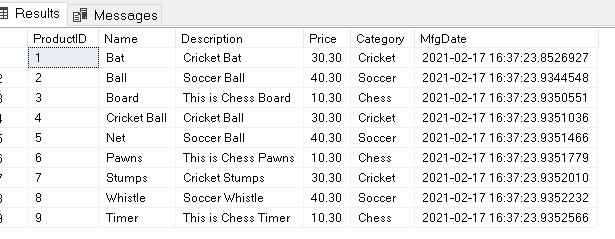
}

}

1. Run SeedData File by passing InitDB through Package manager console.



1. This will create mock data required for development



CRUD Operation

1. Add ProductSQLRepository Class and Implement it

public class ProductSQLRepository : IStoreRepository

{

private readonly ProductDbContext \_context;

public ProductSQLRepository(ProductDbContext context)

{

\_context = context;

}

public IEnumerable<Product> Products => \_context.Products.ToList();

public void AddProduct(Product product)

{

\_context.Products.Add(product);

\_context.SaveChanges();

}

public bool DeleteProduct(int id)

{

var product = \_context.Products.Single(p => p.ProductID == id);

\_context.Products.Remove(product);

\_context.SaveChanges();

return true;

}

public Product GetProduct(int ProductId)

{

return \_context.Products.Single(p => p.ProductID == ProductId);

}

public bool UpdateProduct(int id, Product product)

{

// Get the Product Information you want to update

var pro = \_context.Products.Single(x => x.ProductID == id);

pro.Name = product.Name;

pro.MfgDate = product.MfgDate;

pro.Price = product.Price;

pro.Category = product.Category;

pro.Description = product.Description;

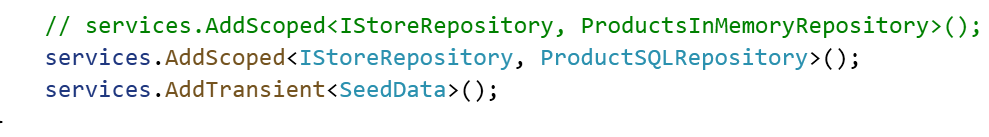
\_context.SaveChanges();

return true;

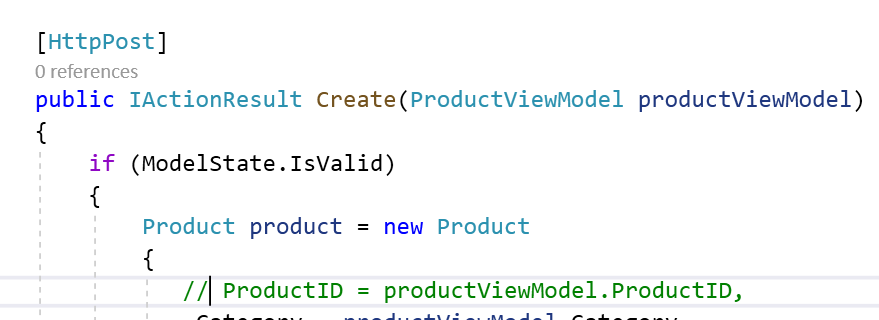
}

}

1. Point to this new Database



1. Remove ID in order to make Insertion to work Correctly



1. Check the Output

What is Problem

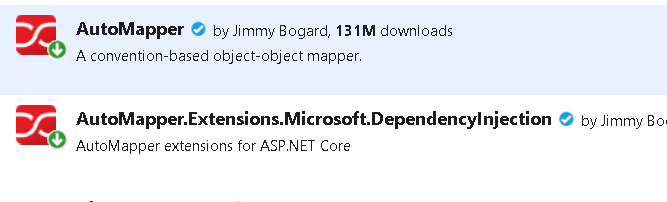
* Problem : n order to maintain a separation of concern between various of your layers in a ASP.NET Core Project, it is advisable to have different models / Data transfer objects that are associated with a specific layer. This is a very good practice.
* Lets say whenever somone add new Customer we want to stored his information to file system. While accepting customername we are taking Full Name but while transferring same data into file we want to split into FirstName and LastName

AutoMapper

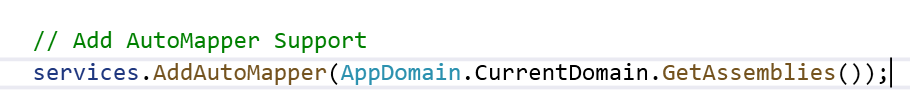
[AutoMapper](https://automapper.org/) in C# is a mapper between two objects. That is, AutoMapper is an object-object mapper. It maps the properties of two different objects by transforming the input object of one type to the output object of another type.

Demo : How to use AutoMapper

1. Add Required Libraries



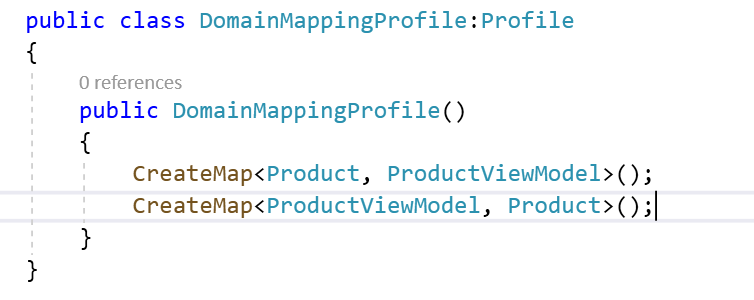
1. Activate Automapper Services



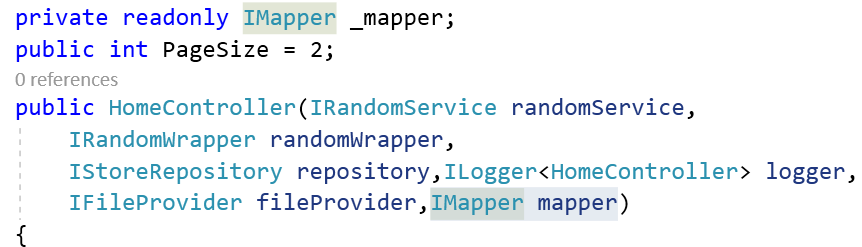
1. Add DomainMappingProfile Class



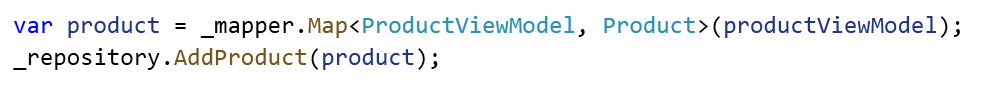
1. Add a code to Perform Conversion



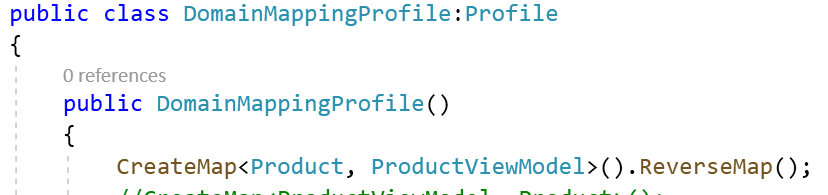
1. DI IMapper into the Class where you want to use it.



1. Use \_mapper.Map function wherever you are doing Conversion



1. Check the Output
2. Change DomainMapping class



1. Check the Output

How to CSRF

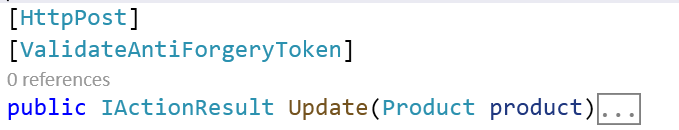
* Consider a scenario user is using web and in between he ha sgone somewhere without logging out from app
* Ur session is active
* Hacker might hack ur session id from cookie
* And created a page that will perform all the operation on ur behalf
* In order to avoid this we use validateAntiforgery token

Demo :

1. Add Anti-Forgery Token attribute to Form Element if Create.cshtml and Update.cshtml



1. Add Attribute to Create and Update Action



1. Go to Application and Change the Token
2. Check the Output

Transactions with EF Core : [Transactions - EF Core | Microsoft Docs](https://docs.microsoft.com/en-us/ef/core/saving/transactions)

* Allows Several Database Operations to be processed in an atomic manner.
* If the transaction is committed all the operations are successfully applied to the database. If the transaction rollback , none of the operations are applied to database.
* Need to use Begin, Commit and Rollback transactions method .

For Example : We have 2 Tables

1. Sales

* ProductId, ProductName

1. Inventory

* ProductId, ProductName, Quantity

As soon as the product is sold, Information should be modified in both the tables or none of them.

Use BoaTest

Create Table Sales

(

ProductID int primary key,

ProductName varchar(400)

)

Create Table Inventories

(

ProductID int primary key,

ProductName varchar(400),

Quantity int

)

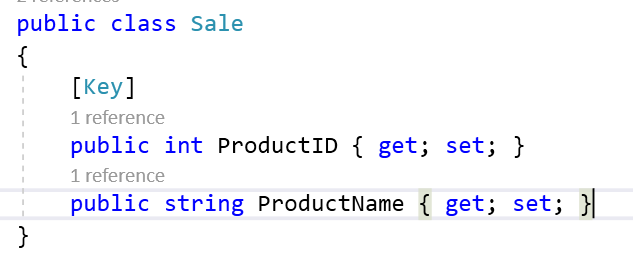
Insert into Inventories Values(1,'IPhone',3)

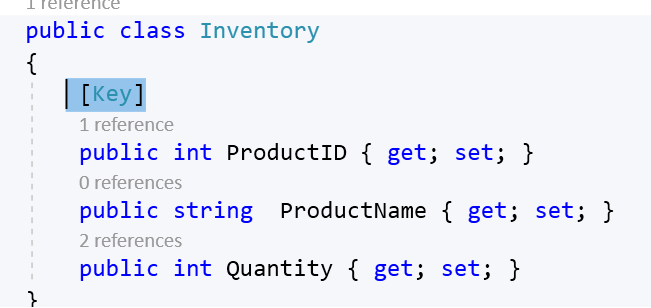
Insert into Inventories Values(2,'Samsung',4)

Select \* From Sales

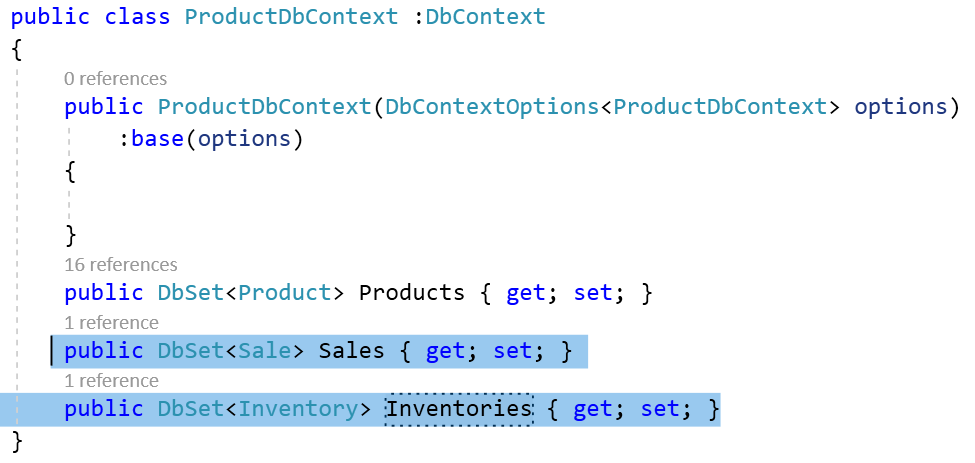
Select \* From Inventories

1. Create Above Tables
2. Add Model Class for Both of the above tables





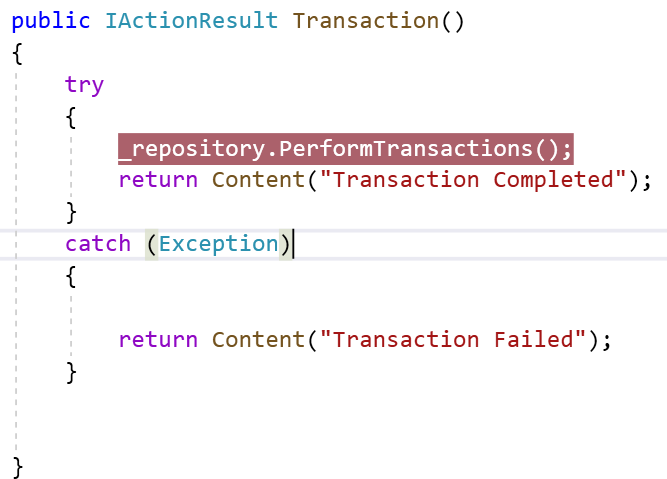
1. Modify Context class to accommodate above tables



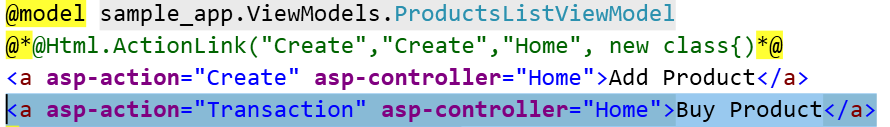
1. Add PerformTransaction Method in ProductSQLRepository



1. Call this Method on HomeController



1. Create link to call this method in Index.cshtml

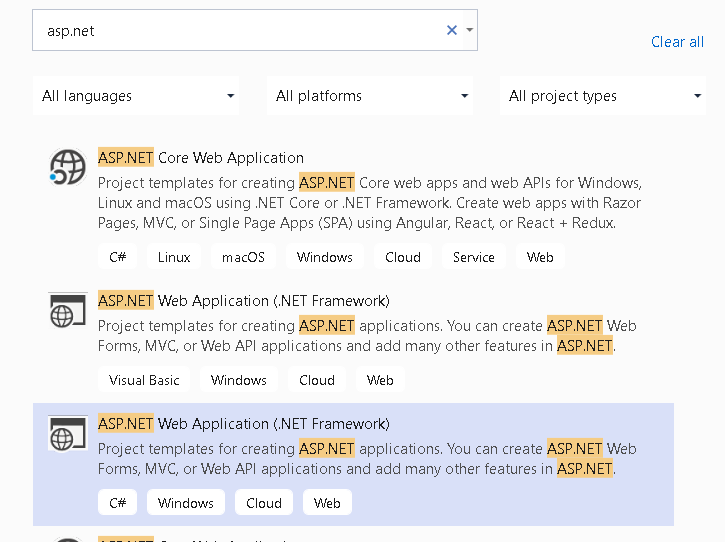


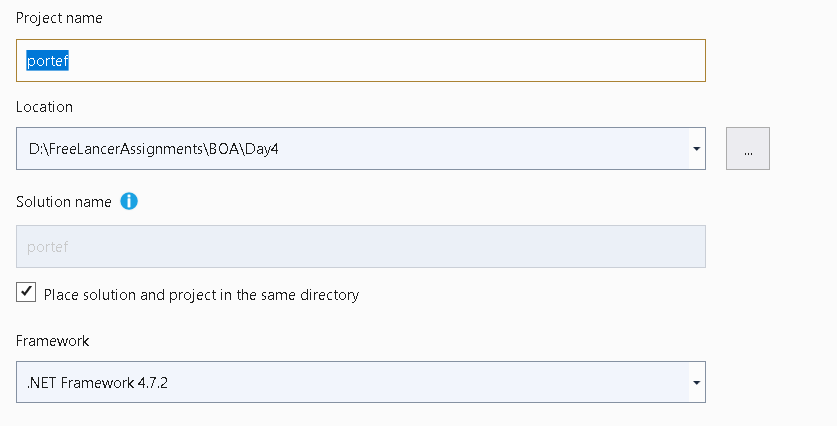
1. Check the Output

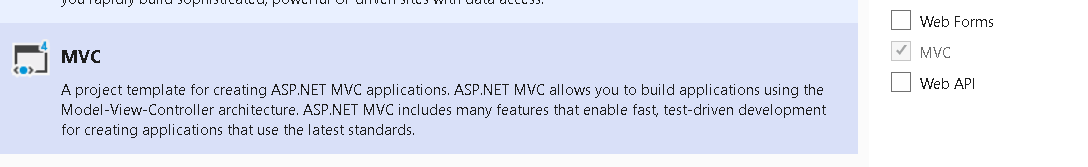
If the transaction is Successful then Completed Message will appear else failed.

Migrations of EF To EF Core

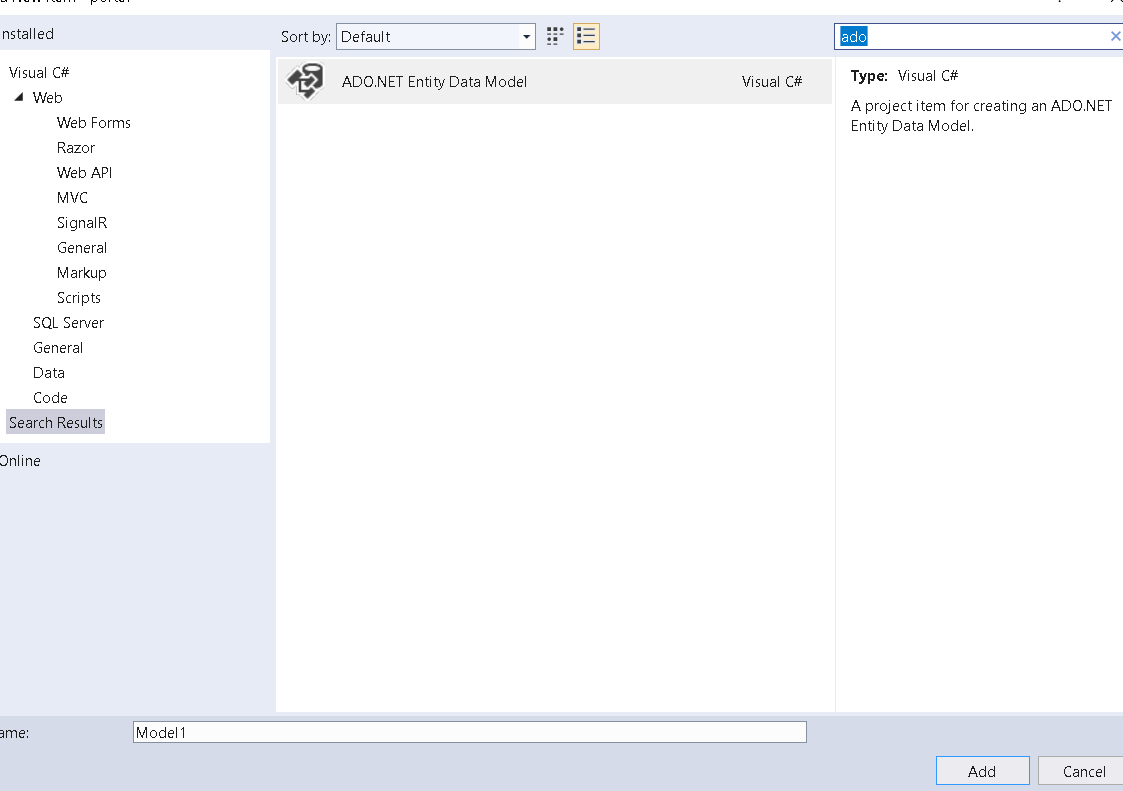
1. Create an application using EF 6 and .Net Framework

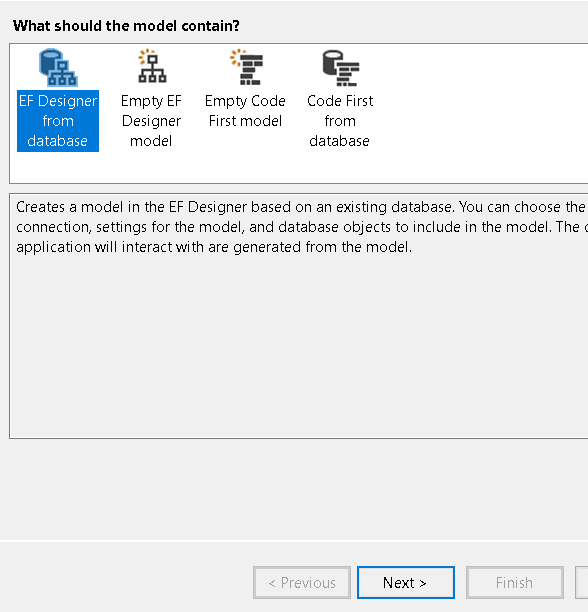


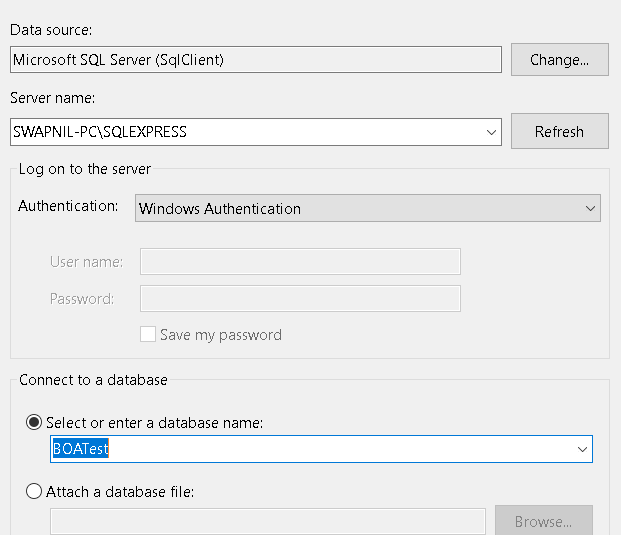


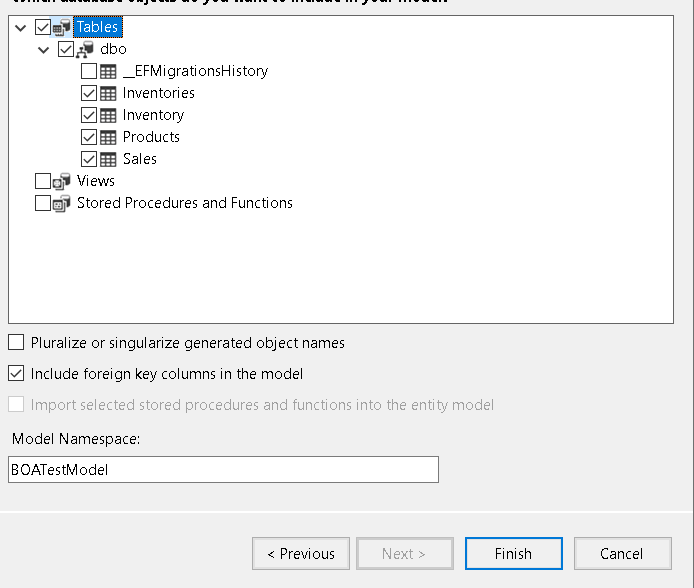


1. Add New Entity Data Model

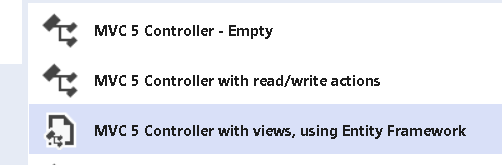


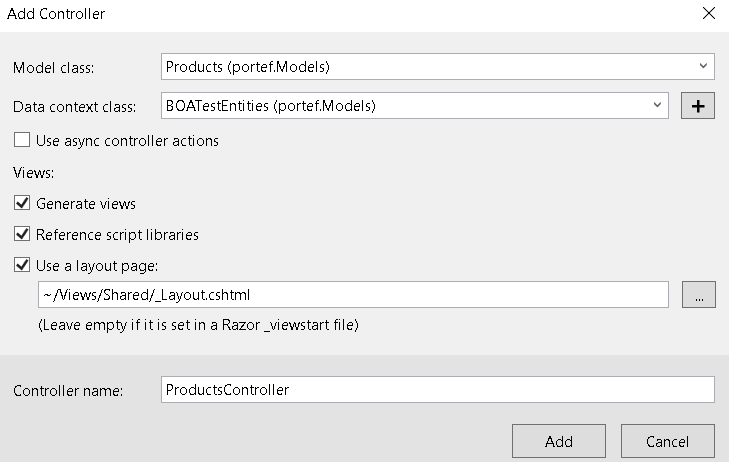




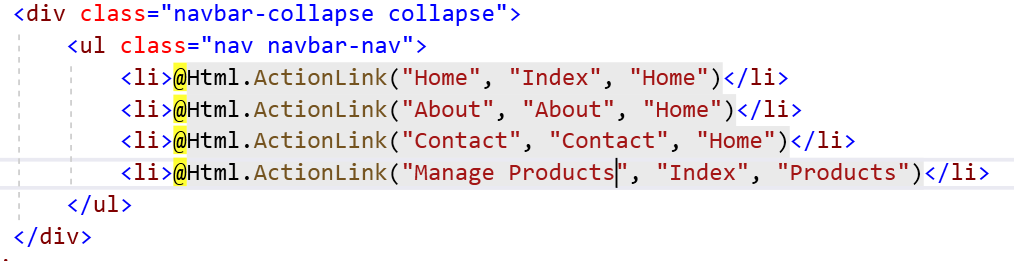


1. Add ProductController to Manage Products

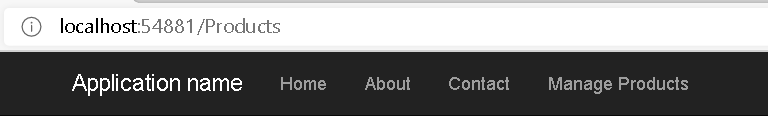




1. Add One more Link Manage Products



1. Check the Output



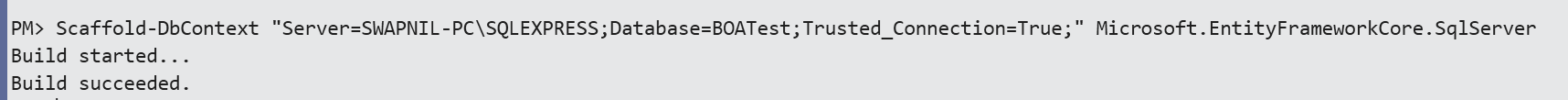
1. Now we want to Port application to EF Core
2. Add Following Libraries

Microsoft.EntityFrameworkCore.Tools

Microsoft.EntityFrameworkCore.SqlServer

1. Go to Package Manager Console and apply following commands

Scaffold-DbContext "Server=SWAPNIL-PC\SQLEXPRESS;Database=BOATest;Trusted\_Connection=True;" Microsoft.EntityFrameworkCore.SqlServer



1. This will generate BOAContext class



1. Resolve the errors
2. Delete Edmx File

Assignment : How to Perform EF Migrations for Code First Model : [Porting from EF6 to EF Core - Porting a Code-Based Model - EF | Microsoft Docs](https://docs.microsoft.com/en-us/ef/efcore-and-ef6/porting/port-code)

Filters

* Filters in ASP.Net core allow code to be run before or after specific stages in the request pipeline.

Real world scenario

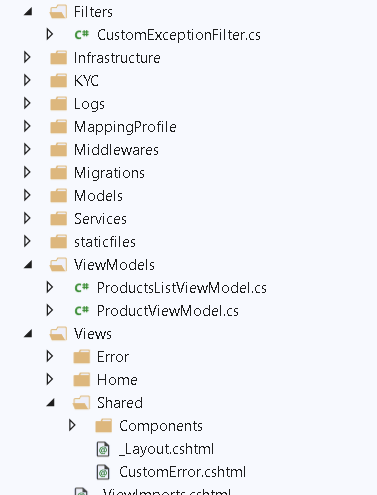
1. Error Handling
2. Caching
3. Configuration
4. Authorization
5. Logging

* Purpose of Filters is to avoid duplicating code.

For Example an Error handling exception filter could consolidate error handling

Exception Filter

* Use to handle any unhandled exceptions that occur in an application.
* They don’t have before or after methods.
* It Implements OnException Method
* Method will be called whenever any unhandled exception occurs.



1. Add Filters Folder and CustomExceptionfilter.cs class

public class CustomExceptionFilter : IExceptionFilter

{

private readonly IModelMetadataProvider \_modelMetadataProvider;

public CustomExceptionFilter(

IModelMetadataProvider modelMetadataProvider)

{

\_modelMetadataProvider = modelMetadataProvider;

}

public void OnException(ExceptionContext context)

{

var result = new ViewResult { ViewName = "CustomError" };

result.ViewData = new ViewDataDictionary(\_modelMetadataProvider, context.ModelState);

result.ViewData.Add("Exception", context.Exception);

// Here we can pass additional detailed data via ViewData

context.ExceptionHandled = true; // mark exception as handled

context.Result = result;

}

}

1. Activate CustomException Class



1. Create CustomError.cshtml

@{

ViewData["Title"] = "CustomError";

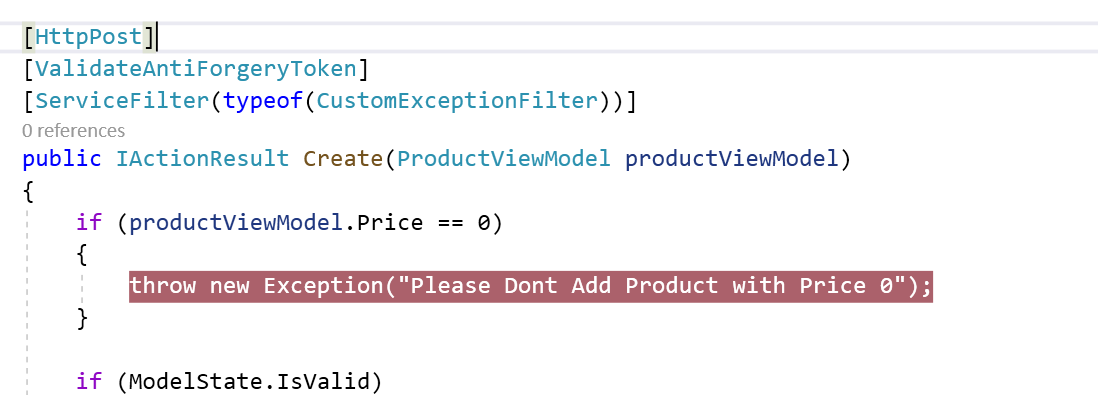
var exception = ViewData["Exception"] as Exception;

}

<h1>An Error has Occurred</h1>

<p>@exception.Message</p>

1. Add into Create Action Method



1. Check the Output

