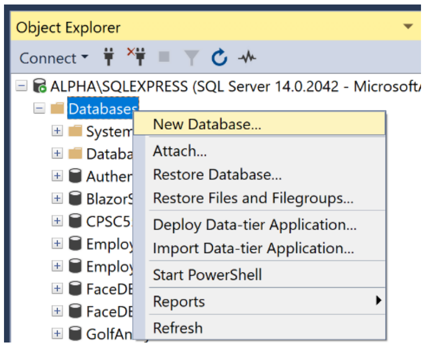
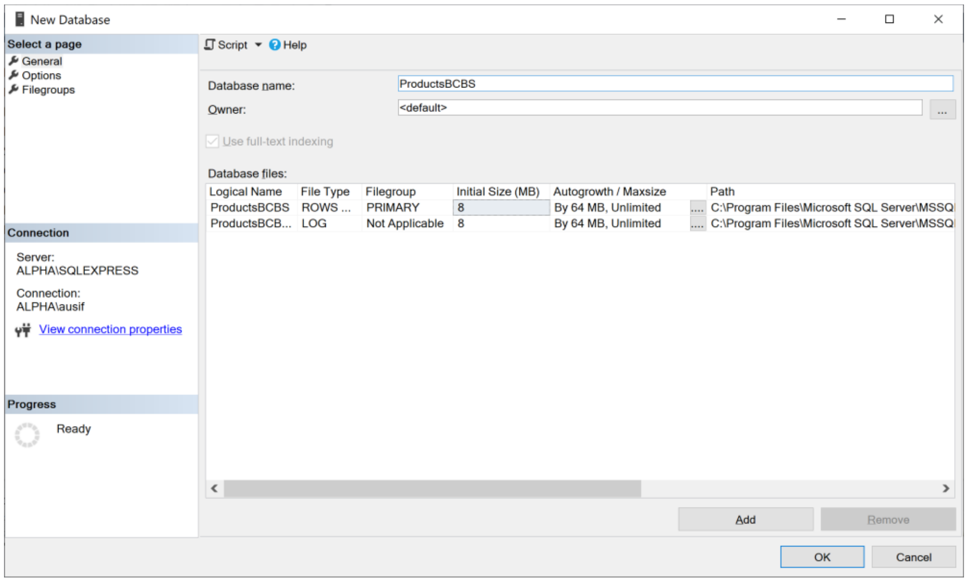
CPSC 555 – Assignment #4

ProductsApp Web Site Design

Database Design: Launch SQL server Management studio as an administrator, then right click on the dataases and choose new database as:



Give the database name as ProductsBCBS as shown below.



Database name: ProductsBCBS
Owner: <default>
Use full-text indexing

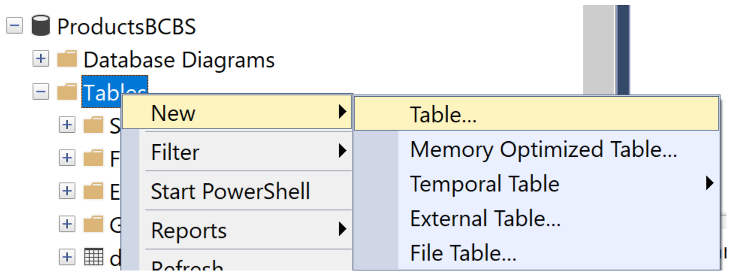
Server:
ALPHA\SQLEXPRESS

Connection:
ALPHA\ausif

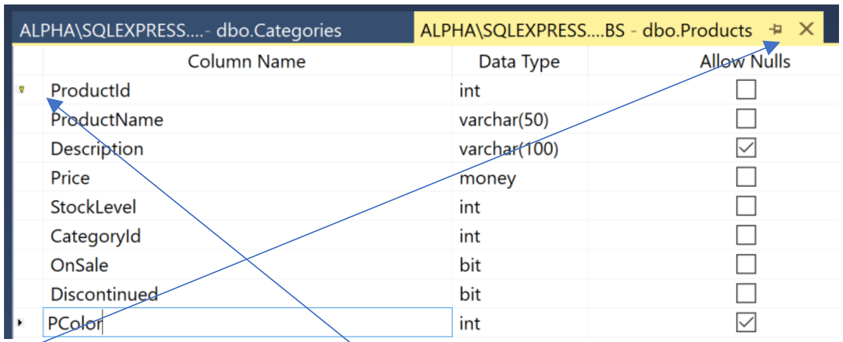
Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth / Maxsize	Path
ProductsBCBS	ROWS ...	PRIMARY	8	By 64 MB, Unlimited	C:\Program Files\Microsoft SQL Server\MSSQ
ProductsBCB...	LOG	Not Applicable	8	By 64 MB, Unlimited	C:\Program Files\Microsoft SQL Server\MSSQ

We will be adding three tables to the ProductsBCBS database. To add a table, right click on the tables folder under ProductsBCBS, and choose new table.



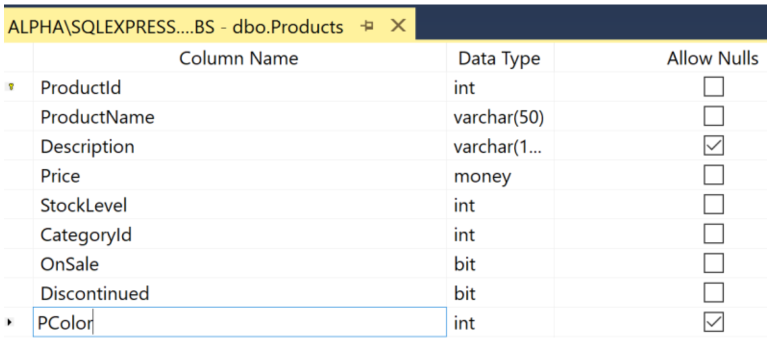
Enter the following design for the Products table.



ALPHA\SQLEXPRESS....- dbo.Categories | ALPHA\SQLEXPRESS....BS - dbo.Products

Column Name	Data Type	Allow Nulls
ProductId	int	☐
ProductName	varchar(50)	☐
Description	varchar(100)	✓
Price	money	☐
StockLevel	int	☐
CategoryId	int	☐
OnSale	bit	☐
Discontinued	bit	☐
PColor	int	✓

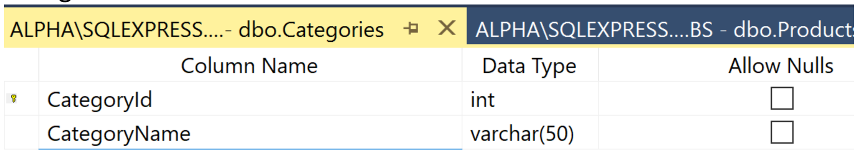
Mark the ProductId as the primary key by right clicking on it and choosing set primary key.
Once you x out the table designer, it will ask you to save changes. Give the table name Products.



ALPHA\SQLEXPRESS....BS - dbo.Products

Column Name	Data Type	Allow Nulls
ProductId	int	☐
ProductName	varchar(50)	☐
Description	varchar(1...	✓
Price	money	☐
StockLevel	int	☐
CategoryId	int	☐
OnSale	bit	☐
Discontinued	bit	☐
PColor	int	✓

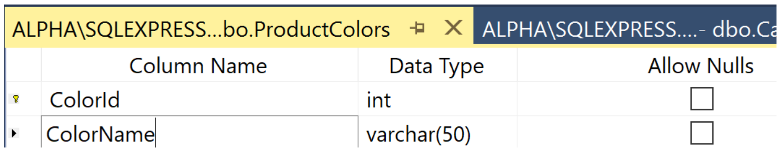
Similarly add Categories table. Make sure to set the CategoryId as the primary key.
Categories table:



ALPHA\SQLEXPRESS....- dbo.Categories | ALPHA\SQLEXPRESS....BS - dbo.Product

Column Name	Data Type	Allow Nulls
CategoryId	int	☐
CategoryName	varchar(50)	☐

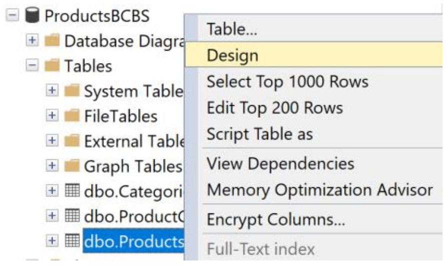
Similarly add another table called ProductColors with the following fields. ColorId should be set as the primary key.
ProductColors table:



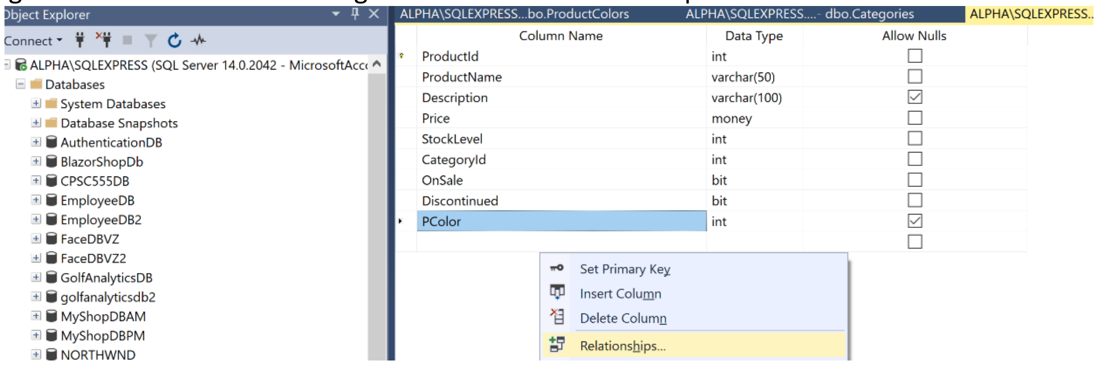
ALPHA\SQLEXPRESS...bo.ProductColors | ALPHA\SQLEXPRESS....- dbo.Ca

Column Name	Data Type	Allow Nulls
ColorId	int	☐
ColorName	varchar(50)	☐

We also need to set the primary-foreign key constraints for the Products table. Right click on the Products table, and choose design.



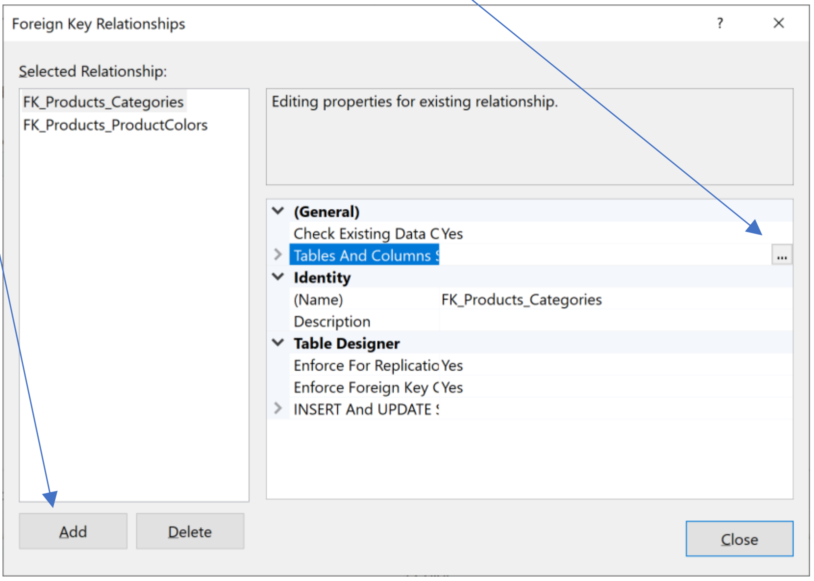
Then right click below the table designer and choose relationships.



- ALPHA\SQLEXPRESS (SQL Server 14.0.2042 - MicrosoftAcc
  - Databases
    - System Databases
    - Database Snapshots
    - AuthenticationDB
    - BlazorShopDb
    - CPSC555DB
    - EmployeeDB
    - EmployeeDB2
    - FaceDBVZ
    - FaceDBVZ2
    - GolfAnalyticsDB
    - golfanalyticsdb2
    - MyShopDBAM
    - MyShopDBPM
    - NORTHWND

Column Name	Data Type	Allow Nulls
ProductId	int	☐
ProductName	varchar(50)	☐
Description	varchar(100)	✓
Price	money	☐
StockLevel	int	☐
CategoryId	int	☐
OnSale	bit	☐
Discontinued	bit	☐
PColor	int	✓
		☐

Click on the Add button and then click on the three dotted button in the row where it indicates Tables and columns specifications.



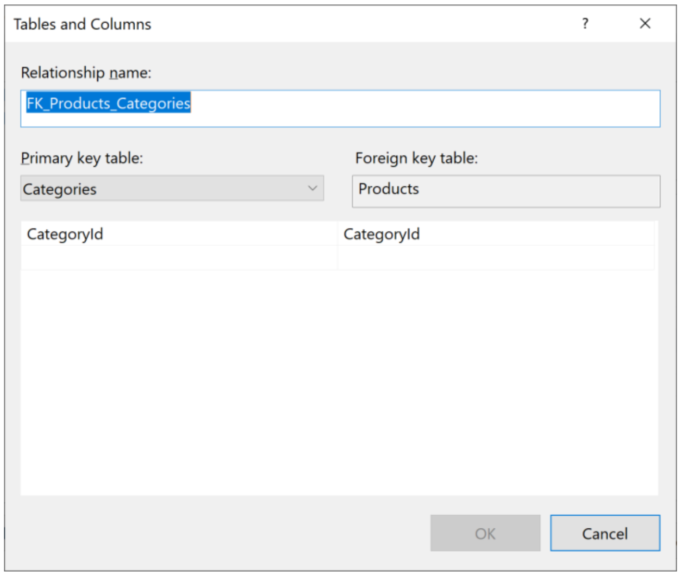
Selected Relationship:

FK_Products_Categories
FK_Products_ProductColors

Editing properties for existing relationship.

(General)	
Check Existing Data C	Yes
Tables And Columns S	
Identity	
(Name)	FK_Products_Categories
Description	
Table Designer	
Enforce For Replicatio	Yes
Enforce Foreign Key C	Yes
INSERT And UPDATE S	

Set the primary key table as Categories, and tie the CategoryId in the Categories table to the CategoryId in the Products table as shown below.



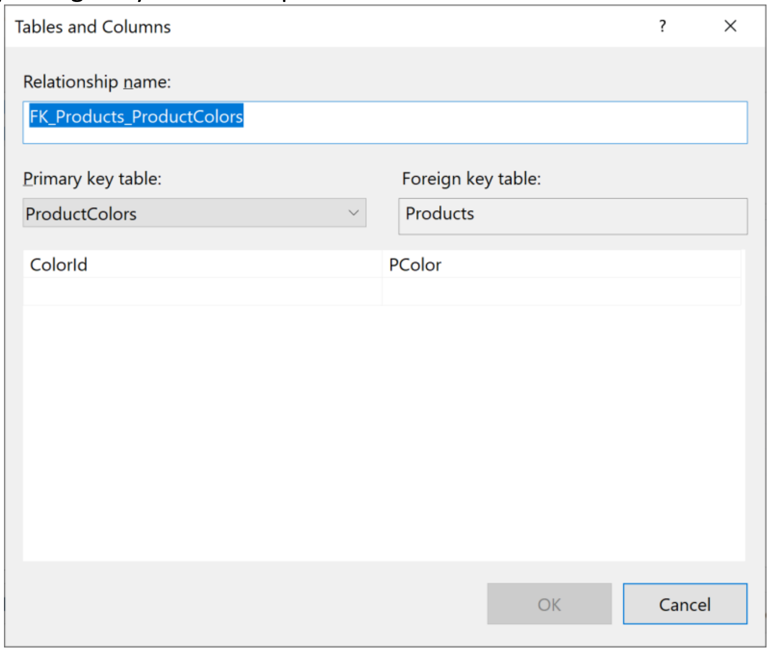
Relationship name: FK_Products_Categories

Primary key table: Categories

Foreign key table: Products

Categories	Products
CategoryId	CategoryId

Similarly click on the Add button again in the relationships and after clicking the three dotted button, add another primary foreign key relationship as shown below.



Tables and Columns

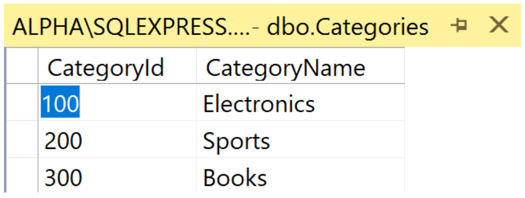
Relationship name: FK_Products_ProductColors

Primary key table: ProductColors

Foreign key table: Products

ColorId	PColor

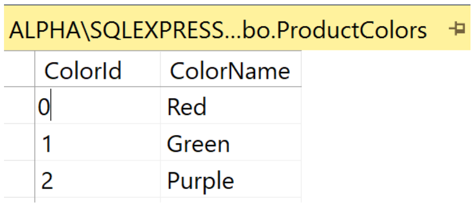
After you click OK, close the table designer, it will ask you save changes to the tables, click yes.
Now lets add some data to the three tables that we have designed. Right click on the Categories table and choose “Edit top 200 rows”, then enter the following data.



ALPHA\SQLEXPRESS....- dbo.Categories

CategoryId	CategoryName
100	Electronics
200	Sports
300	Books

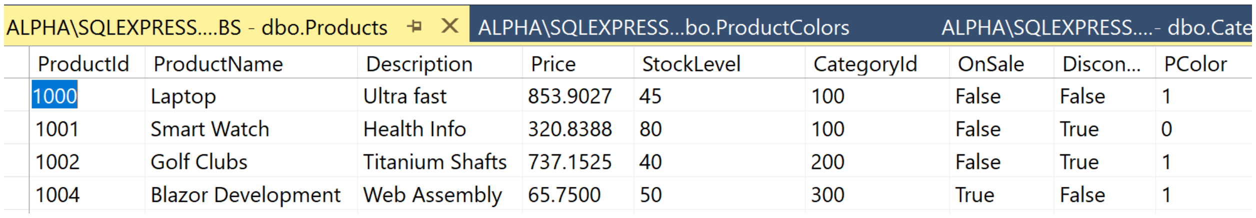
Close the table designer after the table data has been entered. Similarly, right click on the ProductColors table and choose “Edit top 200 rows”. Then put the following data in it.



ALPHA\SQLEXPRESS...bo.ProductColors

ColorId	ColorName
0	Red
1	Green
2	Purple

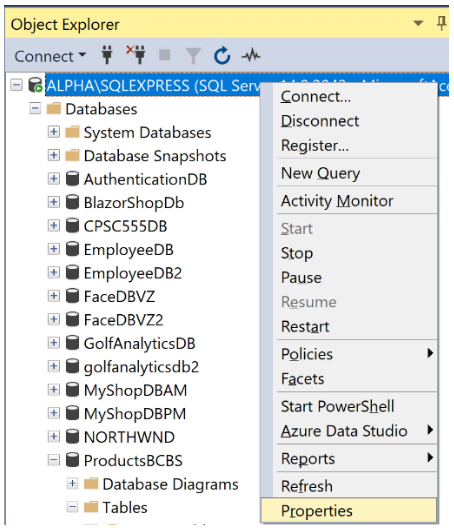
Similarly put the following data in the Products table.



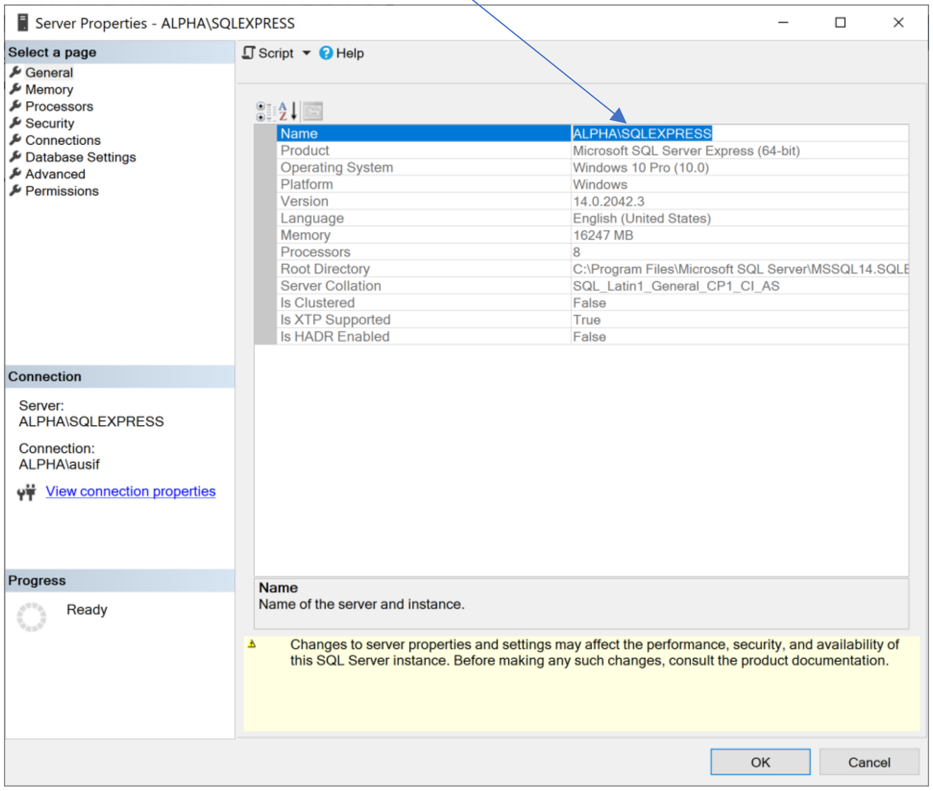
ALPHA\SQLEXPRESS....BS - dbo.Products | ALPHA\SQLEXPRESS...bo.ProductColors | ALPHA\SQLEXPRESS....- dbo.Cate

ProductId	ProductName	Description	Price	StockLevel	CategoryId	OnSale	Discon...	PColor
1000	Laptop	Ultra fast	853.9027	45	100	False	False	1
1001	Smart Watch	Health Info	320.8388	80	100	False	True	0
1002	Golf Clubs	Titanium Shafts	737.1525	40	200	False	True	1
1004	Blazor Development	Web Assembly	65.7500	50	300	True	False	1

Make sure to close the table entry otherwise it will not save the data in the table. Now our database is ready. In order to connect to it, we will need to know the name of the database server. Copy the name of the database server by right clicking on the database server and choosing properties.



Highlight and copy the name of your database server.



Name	ALPHA\SQLEXPRESS
Product	Microsoft SQL Server Express (64-bit)
Operating System	Windows 10 Pro (10.0)
Platform	Windows
Version	14.0.2042.3
Language	English (United States)
Memory	16247 MB
Processors	8
Root Directory	C:\Program Files\Microsoft SQL Server\MSSQL14.SQLE
Server Collation	SQL_Latin1_General_CP1_CI_AS
Is Clustered	False
Is XTP Supported	True
Is HADR Enabled	False

Server:
ALPHA\SQLEXPRESS

Connection:
ALPHA\ausif

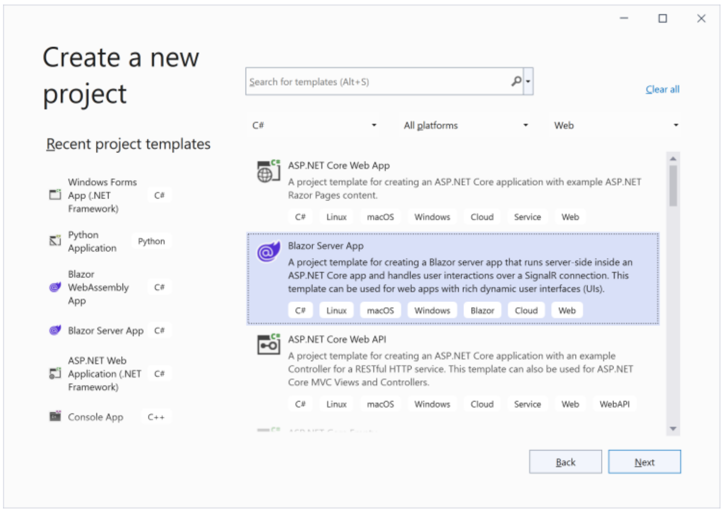
Name
Name of the server and instance.

Changes to server properties and settings may affect the performance, security, and availability of this SQL Server instance. Before making any such changes, consult the product documentation.

e.g., my database server name appears as:
ALPHA\SQLEXPRESS

You can copy this name in a text file. Close the database properties window.

Create a new Blazor Server type of project in Visual Studio as shown below.



Create a new project

Search for templates (Alt+S)

Clear all

C# | All platforms | Web

Recent project templates

Windows Forms App (.NET Framework)	C#
Python Application	Python
Blazor WebAssembly App	C#
Blazor Server App	C#
ASP.NET Web Application (.NET Framework)	C#
Console App	C++

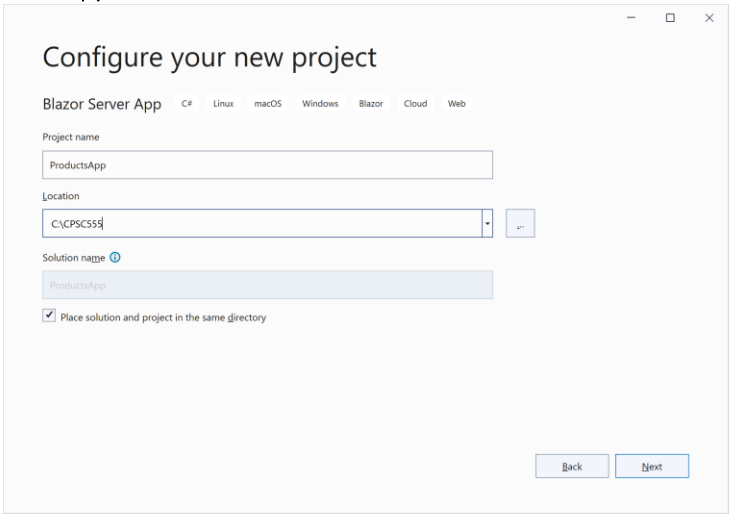
ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

Blazor Server App
A project template for creating a Blazor server app that runs server-side inside an ASP.NET Core app and handles user interactions over a SignalR connection. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web WebAPI

Back | Next

Name the project ProductsApp.



Configure your new project

Blazor Server App C# Linux macOS Windows Blazor Cloud Web

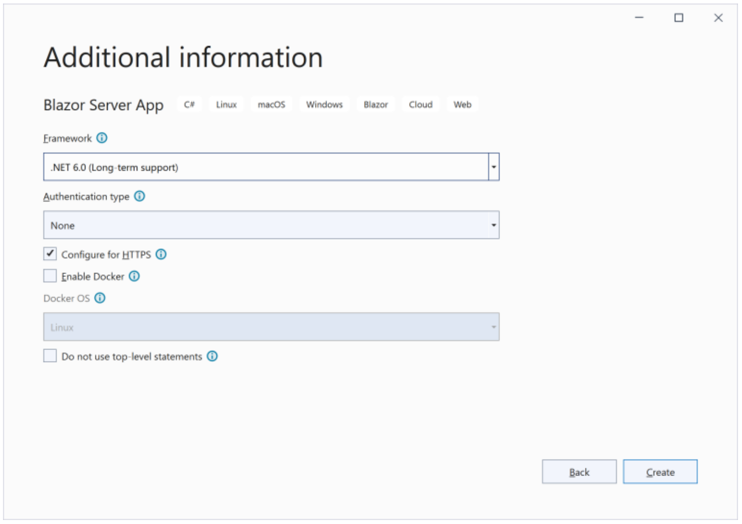
Project name: ProductsApp

Location: C:\CPSC555

Solution name: ProductsApp

☑ Place solution and project in the same directory

Back | Next



Additional information

Blazor Server App C# Linux macOS Windows Blazor Cloud Web

Framework: .NET 6.0 (Long-term support)

Authentication type: None

☑ Configure for HTTPS

☐ Enable Docker

Docker OS: Linux

☐ Do not use top-level statements

Back | Create

Add a folder called Utils to the ProductsApp project. Then right click on it and add a class called ConnectionStringHelper with the following code in it.

```
namespace ProductsApp.Utils
{
    public class ConnectionStringHelper
    {
        public static string CONNSTR { get; set; }
    }
}
```

This is a simple helper class that will store the connection string for the database in it.

Modify the appsettings.json file to contain the connection string information as shown below. Parts to be added are shown in bold. Replace the ALPHA\\SQLEXPESS with the name of your database server.

```
{
  "ConnectionStrings": {
    "ProductsDBConn":
"server=ALPHA\\SQLEXPRESS;Database=ProductsBCBS;Trusted_Connection=True;MultipleActi
veResultSets=true"
  },
  "Logging": {
```

Add the following lines (shown in bold) to the Program.cs file. These lines read the connection string info from the appsettings.json file and store it in the CONNSTR filed of the ConnectionStringHelper class.

```
var builder = WebApplication.CreateBuilder(args);
var connstr = builder.Configuration.GetConnectionString("ProductsDBConn");
ConnectionStringHelper.CONNSTR = connstr;
```

Add a folder to the ProductsApp project called DataLayer.

Add a folder called Models to the project. Then by right clicking on it, add a class called Product with the following code in it.

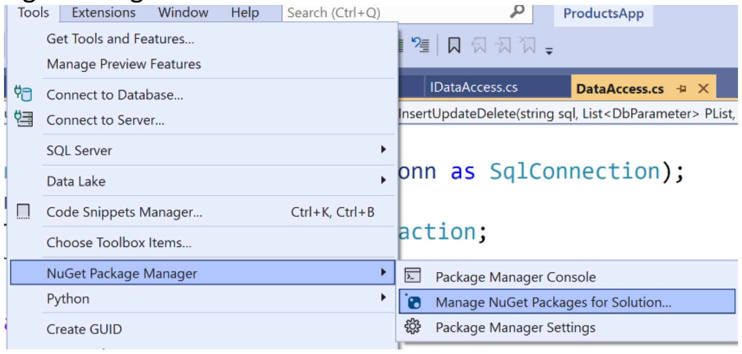
```
namespace ProductsApp.Models
{
    public enum ProductColor
    {
        RED,
        GREEN,
        PURPLE
    }
    public class Product
    {
        public int ProductId { get; set; }
        public string ProductName { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public int StockLevel { get; set; }
        public bool OnSale { get; set; }
        public bool Discontinued { get; set; }
        public int CategoryId { get; set; }
        public int? PColor { get; set; }
    }
}
```

Add a class called Category to the Models folder with the following code in it.

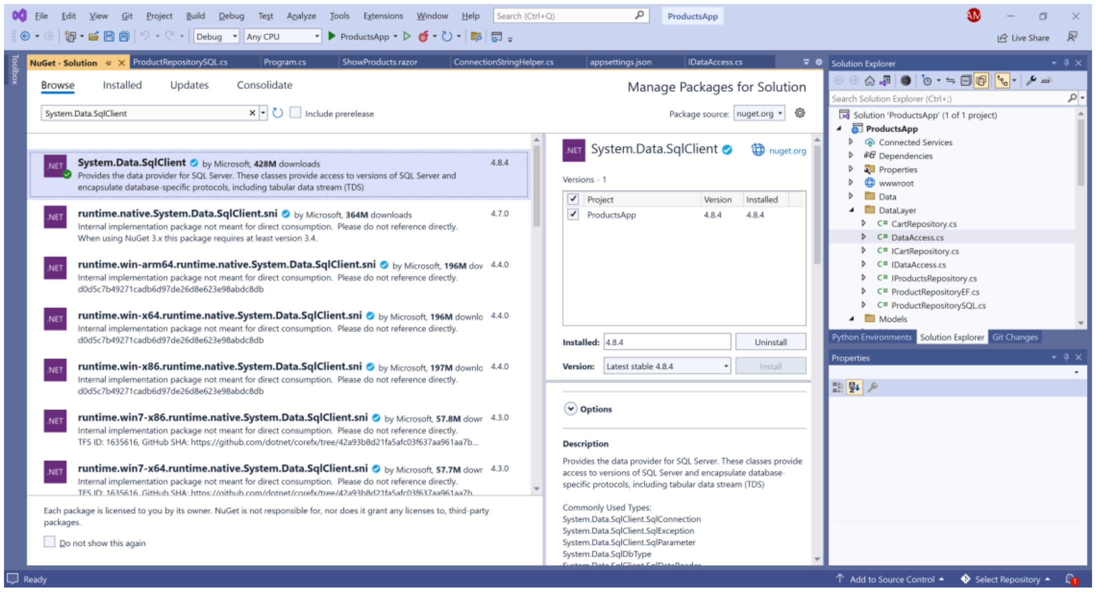
```
namespace ProductsApp.Models
{
    public class Category
    {
        public int CategoryId { get; set; }
        public string CategoryName { get; set; }
    }
}
```

<u>Design of the DataLayer</u>:

To be able to talk to the SQL server database via that SQL client, we will need to install the Nuget package System.Data.SqlClient in the ProductsApp project. From the Tools menu, choose NuGet Package Manager-> Manage NugetPackages for solution.



Then search for System.Data.SqlClient in the browse link as shown below. Select the System.Data.SqlClient, and then check the project checkbox and click on the Install button. Accept the licenses and finish the installation of the package.



System.Data.SqlClient

System.Data.SqlClient by Microsoft, **428M** downloads 4.8.4
Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream (TDS)

runtime.native.System.Data.SqlClient.sni by Microsoft, **364M** downloads 4.7.0
Internal implementation package not meant for direct consumption. Please do not reference directly.
When using NuGet 3.x this package requires at least version 3.4.

runtime.win-arm64.runtime.native.System.Data.SqlClient.sni by Microsoft, **196M** dov 4.4.0
Internal implementation package not meant for direct consumption. Please do not reference directly.
d0d5c7b49271cadb6d97de26d8e623e98abdc8db

runtime.win-x64.runtime.native.System.Data.SqlClient.sni by Microsoft, **196M** downlo 4.4.0
Internal implementation package not meant for direct consumption. Please do not reference directly.
d0d5c7b49271cadb6d97de26d8e623e98abdc8db

runtime.win-x86.runtime.native.System.Data.SqlClient.sni by Microsoft, **197M** downlo 4.4.0
Internal implementation package not meant for direct consumption. Please do not reference directly.
d0d5c7b49271cadb6d97de26d8e623e98abdc8db

runtime.win7-x86.runtime.native.System.Data.SqlClient.sni by Microsoft, **57.8M** dowr 4.3.0
Internal implementation package not meant for direct consumption. Please do not reference directly.
TFS ID: 1635616, GitHub SHA: https://github.com/dotnet/corefx/tree/42a93b8d21fa5afc03f637aa961aa7b...

runtime.win7-x64.runtime.native.System.Data.SqlClient.sni by Microsoft, **57.7M** dowr 4.3.0
Internal implementation package not meant for direct consumption. Please do not reference directly.

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

Manage Packages for Solution

Package source: nuget.org

System.Data.SqlClient nuget.org

Versions - 1

✓	Project	Version	Installed
✓	ProductsApp	4.8.4	4.8.4

Installed: 4.8.4 Uninstall

Version: Latest stable 4.8.4 Install

Options

Description

Provides the data provider for SQL Server. These classes provide access to versions of SQL Server and encapsulate database-specific protocols, including tabular data stream (TDS)

Commonly Used Types:
System.Data.SqlClient.SqlConnection
System.Data.SqlClient.SqlException
System.Data.SqlClient.SqlParameter
System.Data.SqlDbType

Add a folder called DataLayer to the project. Then add an interface to it by right clicking on the folder and choosing “add new item”. Name the interface IDataAccess and type the following code in it.

```
using System.Data;
using System.Data.Common;
namespace ProductsApp.DataLayer
{
    public interface IDataAccess
    {
        // last three parameters are optional and useed if a transaction is involved
        object GetSingleAnswer(string sql, List<DbParameter> PList, DbConnection 
conn = null,
            DbTransaction sqtr = null, bool bTransaction = false);
        DataTable GetManyRowsCols(string sql, List<DbParameter> PList, DbConnection 
conn = null,
            DbTransaction sqtr = null, bool bTransaction = false);
        int InsertUpdateDelete(string sql, List<DbParameter> PList, DbConnection 
conn = null,
            DbTransaction sqtr = null, bool bTransaction = false);
    }
}
```

Then add a class called DataAccess to the DataLayer folder that implements the three functions in the IDataAccess interface. The code for the DataAccess class appears as:

```
using ProductsApp.Utils;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient;

namespace ProductsApp.DataLayer
{
    public class DataAccess : IDataAccess
    {
        string CONNSTR = ConnectionStringHelper.CONNSTR;
        public DataAccess() { }
        public DataAccess(string connstr)  // to allow change of connectionstring
        {
            this.CONNSTR = connstr;
        }
        public DataTable GetManyRowsCols(string sql, List<DbParameter> PList, 
DbConnection conn = null, DbTransaction sqtr = null, bool bTransaction = false)
        {
            DataTable dt = new DataTable();
            if (bTransaction == false)
                conn = new SqlConnection(CONNSTR);
            try
            {
                conn.Open();
                SqlDataAdapter da = new SqlDataAdapter();
                SqlCommand cmd = new SqlCommand(sql, conn as SqlConnection);
                if (PList != null)
                {
                    foreach (DbParameter p in PList)
                        cmd.Parameters.Add(p);
                }
                if (bTransaction == true)
                    cmd.Transaction = sqtr as SqlTransaction;
                da.SelectCommand = cmd;
                da.Fill(dt);
            }
            catch (Exception)
            {
                throw;
            }
            finally
            {
                if (bTransaction == false)
                    conn.Close();
            }
            return dt;
        }

        public object GetSingleAnswer(string sql, List<DbParameter> PList,
DbConnection conn = null, DbTransaction sqtr = null, bool bTransaction = false)
        {
            object obj = null;
            if (bTransaction == false)
                conn = new SqlConnection(CONNSTR);
            try
            {
                if (bTransaction == false)
                    conn.Open();
                SqlCommand cmd = new SqlCommand(sql, conn as SqlConnection);
                if (bTransaction == true)
                    cmd.Transaction = sqtr as SqlTransaction;
                if (PList != null)
                {
                    foreach (DbParameter p in PList)
                        cmd.Parameters.Add(p);
                }
                obj = cmd.ExecuteScalar();
            }
            catch (Exception)
            {
                throw;
            }
            finally
            {
                if (bTransaction == false)
                    conn.Close();
            }
            return obj;
        }

        public int InsertUpdateDelete(string sql, List<DbParameter> PList,
DbConnection conn = null, DbTransaction sqtr = null, bool bTransaction = false)
        {
            int rows = 0;
            if (bTransaction == false)
                conn = new SqlConnection(CONNSTR);
            try
            {
                if (bTransaction == false)
                    conn.Open();
                SqlCommand cmd = new SqlCommand(sql, conn as SqlConnection);
                if (bTransaction == true)
                    cmd.Transaction = sqtr as SqlTransaction;
                if (PList != null)
                {
                    foreach (SqlParameter p in PList)
                        cmd.Parameters.Add(p);
                }
                rows = cmd.ExecuteNonQuery();
            }
            catch (Exception)
            {
                throw;
            }
            finally
            {
                if (bTransaction == false)
                    conn.Close();
            }
            return rows;
        }
    }
}
```

Add an interface to the DataLayer folder called IProductsRepository with the following code in it. This interface defines the different functions that we will need in obtaining or modifying data in the database.

```
using ProductsApp.Models;

namespace ProductsApp.DataLayer
{
    public interface IProductsRepository
    {
        bool AddProduct(Product prod);
        bool ApplyDiscount(int prodid, double percentDiscount);
        bool DeleteProduct(int prodid);
        List<Category> GetCategories();
        Product GetProductById(int prodid);
        List<Product> GetProductsByCatId(int catid);
        bool UpdateProduct(Product prod);
    }
}
```

Add a class called ProductRepositorySQL to the DataLayer folder with the following code in it.

```
using ProductsApp.Models;
using System.Data;
using System.Data.Common;
using System.Data.SqlClient;

namespace ProductsApp.DataLayer
{
    public class ProductRepositorySQL : IProductsRepository
    {
        IDataAccess _idac = new DataAccess();
        public bool AddProduct(Product prod)
        {
            string sql = "insert into
Products(ProductId,ProductName,Description,Price," +
                "OnSale,Discontinued,StockLevel,PColor,CategoryId) values(" +
                "@ProductId,@ProductName,@Description,@Price,@OnSale,@Discontinued,"
+
                "@StockLevel,@PColor,@CategoryId)";
            List<DbParameter> ParamList = new List<DbParameter>();
            SqlParameter p1 = new SqlParameter("@ProductId", prod.ProductId);
            ParamList.Add(p1);
            SqlParameter p2 = new SqlParameter("@ProductName", prod.ProductName);
            ParamList.Add(p2);
            SqlParameter p3 = new SqlParameter("@Description", prod.Description);
            ParamList.Add(p3);
            SqlParameter p4 = new SqlParameter("@Price", prod.Price);
            ParamList.Add(p4);
            SqlParameter p5 = new SqlParameter("@OnSale", prod.OnSale);
            ParamList.Add(p5);
            SqlParameter p6 = new SqlParameter("@Discontinued", prod.Discontinued);
            ParamList.Add(p6);
            SqlParameter p7 = new SqlParameter("@StockLevel", prod.StockLevel);
            ParamList.Add(p7);
            SqlParameter p8 = new SqlParameter("@PColor", prod.Pcolor);
            ParamList.Add(p8);
            SqlParameter p9 = new SqlParameter("@CategoryId", prod.CategoryId);
            ParamList.Add(p9);
            int rows = _idac.InsertUpdateDelete(sql, ParamList);
            if (rows > 0)
                return true;
            else
                return false;
        }

        public bool ApplyDiscount(int prodid, double percentDiscount)
        {
            double factor = (1 - percentDiscount / 100.0);
            string sql = "Update Products set Price=Price*" + factor.ToString() +
                " where ProductId=@ProductId";
            List<DbParameter> ParamList = new List<DbParameter>();
            SqlParameter p1 = new SqlParameter("@ProductId", prodid);
            ParamList.Add(p1);
            int rows = _idac.InsertUpdateDelete(sql, ParamList);
            if (rows > 0)
                return true;
            else
                return false;
        }

        public bool DeleteProduct(int prodid)
        {
            string sql = "Delete from Products where ProductId=@ProductId";
            List<DbParameter> ParamList = new List<DbParameter>();
            SqlParameter p1 = new SqlParameter("@ProductId", prodid);
            ParamList.Add(p1);
            int rows = _idac.InsertUpdateDelete(sql, ParamList);
            if (rows > 0)
                return true;
            else
                return false;
        }

        public List<Category> GetCategories()
        {
            string sql = "Select  * from Categories";
            DataTable dt = _idac.GetManyRowsCols(sql, null);
            List<Category> CList = new List<Category>();
            foreach (DataRow dr in dt.Rows)
            {
                Category c1 = new Category();
                c1.CategoryId = (int)dr["CategoryId"];
                c1.CategoryName = dr["CategoryName"].ToString();
                CList.Add(c1);
            }
            return CList;
        }

        public Product GetProductById(int prodid)
        {
            string sql = "Select * from Products where ProductId=@ProductId";
            List<DbParameter> ParamList = new List<DbParameter>();
            SqlParameter p1 = new SqlParameter("@ProductId", prodid);
            ParamList.Add(p1);
            DataTable dt = _idac.GetManyRowsCols(sql, ParamList);
            Product pr = new Product();
            if (dt.Rows.Count > 0)
            {
                DataRow dr = dt.Rows[0];
                pr.ProductId = (int)dr["ProductId"];
                pr.ProductName = dr["ProductName"].ToString();
                pr.Description = dr["Description"].ToString();
                pr.Price = (decimal)dr["Price"];
                pr.StockLevel = (int)dr["StockLevel"];
                pr.OnSale = (bool)dr["OnSale"];
                pr.Discontinued = (bool)dr["Discontinued"];
                pr.CategoryId = (int)dr["CategoryId"];
                pr.Pcolor = dr["PColor"] == null ? null : (int)dr["PColor"];
            }
            return pr;
        }

        public List<Product> GetProductsByCatId(int catid)
        {
            string sql = "Select * from Products where CategoryId=@CategoryId";
            List<DbParameter> ParamList = new List<DbParameter>();
            SqlParameter p1 = new SqlParameter("@CategoryId", catid);
            ParamList.Add(p1);
            DataTable dt = _idac.GetManyRowsCols(sql, ParamList);
            List<Product> PList = new List<Product>();
            foreach (DataRow dr in dt.Rows)
            {
                Product pr = new Product();
                pr.ProductId = (int)dr["ProductId"];
```

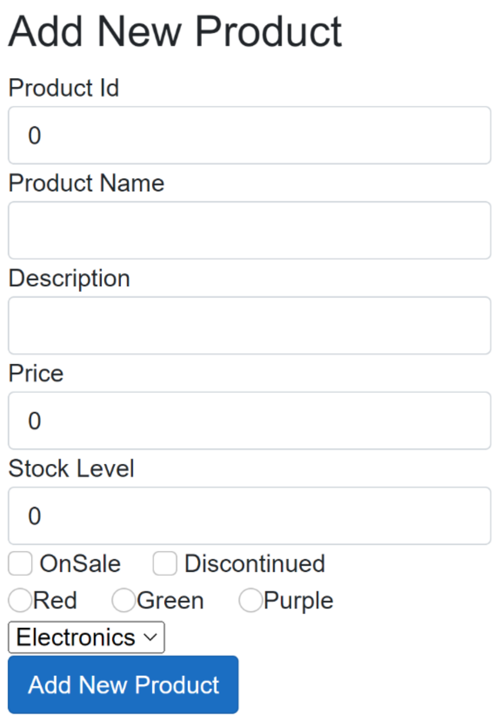
```
                pr.ProductName = dr["ProductName"].ToString();
                pr.Description = dr["Description"].ToString();
                pr.Price = (decimal)dr["Price"];
                pr.StockLevel = (int)dr["StockLevel"];
                pr.OnSale = (bool)dr["OnSale"];
                pr.Discontinued = (bool)dr["Discontinued"];
                pr.CategoryId = (int)dr["CategoryId"];
                var pcol = dr["PColor"].ToString();
                pr.Pcolor = pcol == "" ? null : (int)dr["PColor"];
                PList.Add(pr);
            }
            return PList;
        }

        public bool UpdateProduct(Product prod)
        {
            string sql = "Update Products set ProductName=@ProductName," +
                "Description=@Description,Price=@Price,StockLevel=@StockLevel," +
                "OnSale=@OnSale,Discontinued=@Discontinued,PColor=@PColor," +
                "CategoryId=@CategoryId where ProductId=@ProductId";
            List<DbParameter> ParamList = new List<DbParameter>();
            SqlParameter p1 = new SqlParameter("@ProductId", prod.ProductId);
            ParamList.Add(p1);
            SqlParameter p2 = new SqlParameter("@ProductName", prod.ProductName);
            ParamList.Add(p2);
            SqlParameter p3 = new SqlParameter("@Description", prod.Description);
            ParamList.Add(p3);
            SqlParameter p4 = new SqlParameter("@Price", prod.Price);
            ParamList.Add(p4);
            SqlParameter p5 = new SqlParameter("@OnSale", prod.OnSale);
            ParamList.Add(p5);
            SqlParameter p6 = new SqlParameter("@Discontinued", prod.Discontinued);
            ParamList.Add(p6);
            SqlParameter p7 = new SqlParameter("@StockLevel", prod.StockLevel);
            ParamList.Add(p7);
            SqlParameter p8 = new SqlParameter("@PColor", prod.Pcolor);
            ParamList.Add(p8);
            SqlParameter p9 = new SqlParameter("@CategoryId", prod.CategoryId);
            ParamList.Add(p9);
            int rows = _idac.InsertUpdateDelete(sql, ParamList);
            if (rows > 0)
                return true;
            else
                return false;
        }
    }
}
```

Add the following line to the Program.cs so that ProductRepository is available to the different pages in the application (add this line after the `ConnectionStringHelper.CONNSTR = connstr;`)

```
builder.Services.AddScoped<IProductsRepository, ProductRepositorySQL>();
```

The above line provides the dependency injection of ProductRepository class via the interface IProductsRepository. *AddSoped* means that the same object of this class will be used by web pages during the session of a user, but different users will get a different object of the ProductRepository class.

Now that our DataLayer is ready, we can move on to create the different pages. Since the user interface for AddNewProduct and EditProduct is similar, we will create a reusable component first and then use this component in the AddNewProduct and EditProduct pages. Here is how the AddNewProduct page and the EditProduct pages may look like. You will lot of similarity in the UI except that the button text and page headings are different.



Add New Product

Product Id: 0

Product Name:

Description:

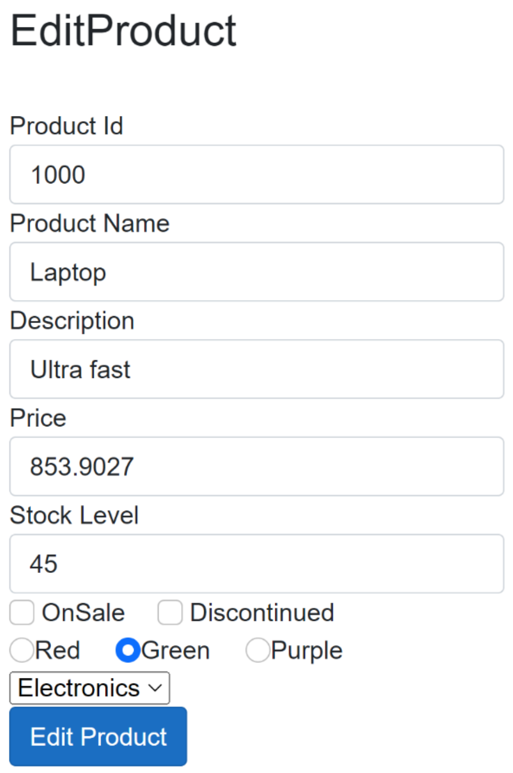
Price: 0

Stock Level: 0

☐ OnSale ☐ Discontinued

○ Red ○ Green ○ Purple

Electronics



EditProduct

Product Id: 1000

Product Name: Laptop

Description: Ultra fast

Price: 853.9027

Stock Level: 45

☐ OnSale ☐ Discontinued

○ Red ● Green ○ Purple

Electronics

Right click on the Pages folder and add a new folder underneath it called Components. Then right click on the Components folder and add a razor component to it called AddEditProductComp.razor. Type the following code in it. The html part of this code is representing the UI as shown above.

```
@using ProductsApp.Models
<div class="row">
    <div class="col-md-4">
        <EditForm Model="Prod" OnValidSubmit="@ValidSubmit">
            <DataAnnotationsValidator/>
            <div class="form-group">
                <label for="Prod.ProductId" class="control-label">Product Id</label>
                <input @bind="Prod.ProductId" class="form-control"/>
            </div>
            <div class="form-group">
                <label for="Prod.ProductName" class="control-label">Product 
Name</label>
                <input @bind="Prod.ProductName" class="form-control"/>
            </div>
            <div class="form-group">
                <label for="Prod.Description" class="control-
label">Description</label>
                <input @bind="Prod.Description" class="form-control"/>
            </div>
            <div class="form-group">
                <label for="Prod.Price" class="control-label">Price</label>
                <input @bind="Prod.Price" class="form-control"/>
            </div>
            <div class="form-group">
                <label for="Prod.StockLevel" class="control-label">Stock 
Level</label>
                <input @bind="Prod.StockLevel" class="form-control"/>
            </div>
            <div class="form-group form-check-inline">
                <label class="form-check-label">
                    <input type="checkbox" @bind="Prod.OnSale" class="form-check-
input"/>
                    OnSale
                </label>
            </div>
            <div class="form-group form-check-inline">
                <label class="form-check-label">
                    <input type="checkbox" @bind="Prod.Discontinued" class="form-
check-input"/>
                    Discontinued
                </label>
            </div>
            <div class="form-group">
                <InputRadioGroup Name="ProdColor" @bind-Value="Prod.Pcolor">
                    <InputRadio class="form-check-input" Value="0"/>Red 
&nbsp;&nbsp;&nbsp;
                    <InputRadio class="form-check-input" Value="1"/>Green 
&nbsp;&nbsp;&nbsp;
                    <InputRadio class="form-check-input" Value="2"/>Purple 
&nbsp;&nbsp;&nbsp;
                </InputRadioGroup>
            </div>
            <div class="form-group">
                <select @bind="Prod.CategoryId">
                    @foreach(var cat in CList)
                    {
                        <option value="@cat.CategoryId">@cat.CategoryName</option>
                    }
                </select>
            </div>
            <div class="form-group">
                <input type="submit" value="@ButtonText" class="btn btn-primary"/>
            </div>
        </EditForm>
    </div>
</div>
@code {
    [Parameter]
    public Product Prod { get; set; }
    [Parameter]
    public List<Category> CList{ get; set; }
    [Parameter]
    public string ButtonText { get; set; }
    [Parameter]
    public EventCallback ValidSubmit { get; set; }
}
```

To make the above component reusable, we have provided four parameters that the user of this component will provide i.e, the product fields to be initially displayed, the category list, the button text and a callback when valid data is submitted by the user from the above form.

Add a razor component to the Pages folder called AddNewProduct.razor with the following code in it.

```
@page "/addnewprod"
@using ProductsApp.DataLayer
@using ProductsApp.Models
@using ProductsApp.Pages.Components
@inject IProductsRepository irep
<h3>Add New Product</h3>

<AddEditProductComp ButtonText="Add New Product" Prod="Prodct" CList="CatList"
     ValidSubmit="@AddProduct"/>
<p>@Msg</p>
@code {
    List<Category> CatList;
    Product Prodct;
    string Msg = "";
    protected override void OnInitialized()
    {
        //IProductsRepository irep = new ProductRepositorySQL();
        CatList = irep.GetCategories();
        Prodct = new Product();
        Prodct.CategoryId = 100;
    }

    void AddProduct()
    {
        //IProductsRepository irep = new ProductRepositorySQL();
        bool ret = false;
        try
        {
            ret = irep.AddProduct(Prodct);
        }
        catch(Exception ex)
        {
            Msg = ex.Message;
        }
        if (ret)
            Msg = "Product added successfully..";
        else
            Msg = "Problem in adding product..";
    }
}
```

Notice how the above page uses the AddEditProductComp component and passes it a blank Product object to display. Also, the page injects the ProductRepository via the line:

```
@inject IProductsRepository irep
```

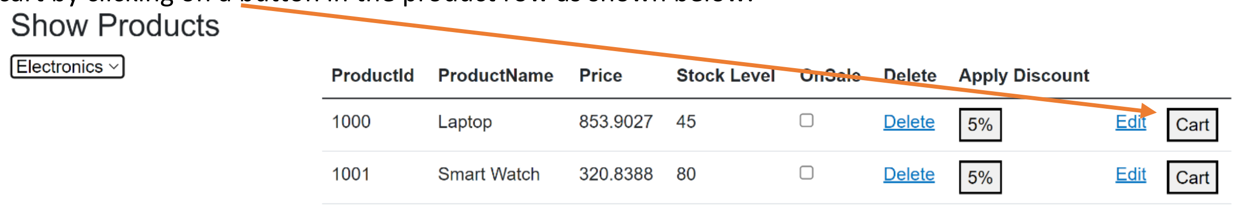
The UI for the AddNewProduct and the C# code is extremely small as we are reusing the AddEditProductComp component and the DataLayer repository to store the new product in the database.

Add a razor component to the Pages folder called EditProduct.razor with the following code in it.

```
@page "/editproduct/{prodid}"
@using ProductsApp.DataLayer
@using ProductsApp.Models
@using ProductsApp.Pages.Components
@inject IProductsRepository irep
<h3>EditProduct</h3>
<br/>
<AddEditProductComp ButtonText="Edit Product" CList="CatList" Prod="Prodct"
     ValidSubmit="@UpdateProduct"/>
<br/>
<p>@Msg</p>
@code {
    [Parameter]
    public string prodid { get; set; }
    string Msg = "";
    List<Category> CatList;
    Product Prodct;

    protected override void OnInitialized()
    {
        CatList = irep.GetCategories();
        Prodct = irep.GetProductById(int.Parse(prodid));
    }

    void UpdateProduct()
    {
        bool ret = false;
        try
        {
            ret = irep.UpdateProduct(Prodct);
        }
        catch(Exception ex)
        {
            Msg = ex.Message;
        }
        if (ret == true)
            Msg = "Product updated successfully..";
        else
            Msg = "Problem in updating product ";
    }
}
```

This page also uses the AddEditProductComp component. The overall code in the EditProduct component is very similar to the AddNewProduct page.

We will like to show the products by category and also allow the user to add a product to the shopping cart by clicking on a button in the product row as shown below.


Show Products

Electronics

ProductId	ProductName	Price	Stock Level	OnSale	Delete	Apply Discount		
1000	Laptop	853.9027	45	☐	Delete	5%	Edit	Cart
1001	Smart Watch	320.8388	80	☐	Delete	5%	Edit	Cart

Our shopping cart may be displayed in different pages so it makes sense to also create it as a component. Each row in shopping cart will display the product id, product name, its price, quantity, and the total. So for this purpose, lets create a class called CartItem. We will put this class in a new folder called ModelsVM (short for view models). Add a folder to the project called ModelsVM, then right click on it and add a class called CartItem with the following code in it.

```
namespace ProductsApp.ModelsVM
{
    public class CartItem
    {
        public int ProductId { get; set; }
        public string ProductName { get; set; }
        public int Quantity { get; set; }
        public double Price { get; set; }
    }
}
```

The shopping cart will be stored in the ProtectedLocalStore so that even if user closes the browser, the shopping cart data is maintained (until the user explicitly checks out or clears the cart). Since the List of CartItems will be json serialized and deserialized in storing and retrieving the shopping cart in the ProtectedLocalStorage, we need to add the Nuget package called NewtonSoft.json. From the tools menu, choose Nuget Package Manager-> Manage Packages for solution, then browse for NewtonSoft.json and install the package in the project.

To able to add an item, or view items in the cart, or store the cart, add an interface to the DataLayer folder called ICartRepository with the following code in it.

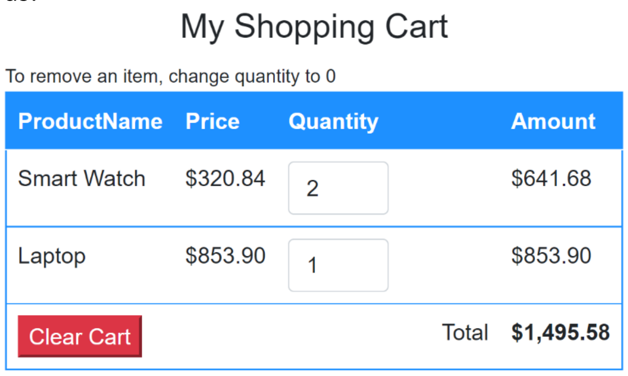
```
using ProductsApp.ModelsVM;
namespace ProductsApp.DataLayer
{
    public interface ICartRepository
    {
        void AddItemToCart(CartItem item);
        Task<List<CartItem>> GetCart();
        void StoreCart(List<CartItem> CList);
    }
}
```

Then add a class called CartRepository to the DataLayer folder with the following code in it.

```
using Microsoft.AspNetCore.Components.Server.ProtectedBrowserStorage;
using Newtonsoft.Json;
using ProductsApp.ModelsVM;
namespace ProductsApp.DataLayer
{
    public class CartRepository : ICartRepository
    {
        ProtectedLocalStorage _LocalStore; // storage, retrieval is key, value based
        public CartRepository(ProtectedLocalStorage ps)
        {
            _LocalStore = ps;
        }
        string CartKey = "CartKey1";
        public async Task<List<CartItem>> GetCart()
        {
            List<CartItem> CList = new List<CartItem>();
            var json = await _LocalStore.GetAsync<string>(CartKey);
            string jsonstr = json.Value;
            if (jsonstr != null)
                CList = JsonConvert.DeserializeObject<List<CartItem>>(jsonstr);
            return CList;
        }

        public void StoreCart(List<CartItem> CList)
        {
            string json = JsonConvert.SerializeObject(CList);
            _LocalStore.SetAsync(CartKey, json);
        }

        public async void AddItemToCart(CartItem item)
        {
            List<CartItem> CList = await GetCart();
            // if item exists, then increment its quantity, else add it to cart
            bool found = false;
            foreach (var row in CList)
            {
                if (row.ProductId == item.ProductId)
                {
                    row.Quantity = row.Quantity + 1;
                    found = true;
                    break;
                }
            }
            if (found == false)
                CList.Add(item);
            StoreCart(CList);
        }
    }
}
```

So that our web pages can access the CartRepository, do a dependency injection on it by adding the following line in the Program.cs file.
`builder.Services.AddScoped<ICartRepository, CartRepository>();`

Our shopping may appear as:



My Shopping Cart

To remove an item, change quantity to 0

ProductName	Price	Quantity	Amount
Smart Watch	$320.84	2	$641.68
Laptop	$853.90	1	$853.90
Clear Cart		Total	**$1,495.58**

To create the shopping cart as a component, right click on the Components folder (under the Pages folder), and add a razor component to it called ShoppingCart.razor with the following code in it.

```
@using ProductsApp.ModelsVM
@using ProductsApp.DataLayer
@inject ICartRepository cartRepository
<div class="row">
    <div class="col-md-5">
        <h4 class="text-center">My Shopping Cart</h4>
        <span style="font-size:smaller">To remove an item, change quantity to
0</span>
        <table style="border:1px solid dodgerblue;margin:0" class="table">
            <thead>
                <tr style="background-color:dodgerblue;color:white">
                    <th>ProductName</th>
                    <th>Price</th>
                    <th>Quantity</th>
                    <th>Amount</th>
                </tr>
            </thead>
            <tbody>
                @{
                    if (MyCart != null)
                    {
                        double total = 0;
                        foreach (var item in MyCart)
                        {
                            var amount = item.Price * item.Quantity;
                            <tr>
                                <td>@item.ProductName</td>
                                <td>@String.Format("{0:c}",item.Price)</td>
                                <td><input class="form-control w-50"
@onchange="(ce=>UpdateCart(item.ProductId,ce))" value="@item.Quantity" /></td>
                                <td>@String.Format("{0:c}",amount)</td>
                            </tr>
                            total = total + item.Price * item.Quantity;
                        }
                        <tr>
                            <td><button @onclick="ClearCart" class="btn-
danger">Clear Cart</button></td>
                            <td colspan="2" align="right">Total</td>
                            <td><b>@String.Format("{0:c}",total)</b></td>
                        </tr>
                    }
                }
            </tbody>
        </table>
    </div>
</div>

@code {
    [Parameter]
    public List<CartItem> MyCart { get; set; }

    void UpdateCart(int prodid, ChangeEventArgs ce) // triggered when quantity is
changed
    {
        if (MyCart == null)
```

```
                return;
            var quantity = int.Parse(ce.Value.ToString());
            CartItem itemToRemove = null;
            foreach (var item in MyCart)
            {
                if ((quantity <= 0) && (item.ProductId == prodid))
                {
                    itemToRemove = item;
                    break;
                }
                if ((quantity > 0) && (item.ProductId == prodid))
                    item.Quantity = quantity;
            }

            if (itemToRemove != null)
                MyCart.Remove(itemToRemove);

            cartRepository.StoreCart(MyCart);
        }

        void ClearCart()
        {
            MyCart.Clear();
            cartRepository.StoreCart(MyCart);
        }
    }
```

Add a rzor component to the Pages folder called ShowProducts.razor with the following code in it.

```
@page "/showproducts/{catid}"
@page "/showproducts"
@inject IJSRuntime js
@inject NavigationManager navManager
@using ProductsApp.DataLayer
@using ProductsApp.Models
@using ProductsApp.ModelsVM
@inject IProductsRepository irep
@inject ICartRepository icartRep

<h3>Show Products</h3>
<div class="row">
    <div class="col-md-3">
        <select value="@catselection" @onchange="CatSelected">
            <option value="0"></option>
            @foreach(var cat in CList)
            {
                <option value="@cat.CategoryId">@cat.CategoryName</option>
            }
        </select>
    </div>
    <div class="col-md-9">
        <table class="table">
            <thead>
                <tr>
                    <th>ProductId</th>
                    <th>ProductName</th>
                    <th>Price</th>
                    <th>Stock Level</th>
                    <th>OnSale</th>
                    <th>Delete</th>
                    <th>Apply Discount</th>
                    <th></th>
                    <th></th>
                 </tr>
            </thead>
            <tbody>
                @foreach(var prod in PList)
                {
                    <tr>
                        <td>@prod.ProductId</td>
                        <td>@prod.ProductName</td>
                        <td>@prod.Price</td>
                        <td>@prod.StockLevel</td>
                        <td><input type="checkbox" @bind="prod.OnSale"/></td>
                        <td><a @onclick:preventDefault
@onclick="@(()=>Delete(prod.ProductId,prod.CategoryId))" href="">Delete</a></td>
                        <td><button
@onclick="@(()=>ApplyDiscount(@prod.ProductId))">5%</button></td>
                        <td><a href="editproduct/@prod.ProductId">Edit</a></td>
                        <td><button
@onclick="@(()=>AddToCart(@prod.ProductId))">Cart</button></td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
</div>
<p>@Msg</p>
@code {
    List<Category> CList = new List<Category>();
    List<Product> PList = new List<Product>();
    string Msg = "";
    string catselection = "";
    [Parameter]
    public string catid { get; set; } = "";// route parameter

    void AddToCart(int prodid)
    {
        Product prod = irep.GetProductById(prodid);
        CartItem item = new CartItem
            {
                ProductId = prod.ProductId,
                ProductName = prod.ProductName,
                Price = (double)prod.Price,
                Quantity = 1
            };
        icartRep.AddItemToCart(item);
    }

    void ApplyDiscount(int prodid)
    {
        //IProductsRepository irep = new ProductRepositorySQL();
        irep.ApplyDiscount(prodid, 5);
        PList = irep.GetProductsByCatId(int.Parse(catid));
    }

    protected override void OnInitialized()
    {
        //IProductsRepository irep = new ProductRepositorySQL();
        Msg = "";
        CList = irep.GetCategories();
        if (catid != null)
        {
            if (catid != "")
            {
                int catSelected = int.Parse(catid);
                PList = irep.GetProductsByCatId(catSelected);
            }
        }
    }

    void CatSelected(ChangeEventArgs ce)
    {
        //IProductsRepository irep = new ProductRepositorySQL();
        int catSelected = int.Parse(ce.Value.ToString());
        PList = irep.GetProductsByCatId(catSelected);
        catid = catSelected.ToString();
    }

    async Task Delete(int prodid, int catid)
    {
        // javascript confirmation
        if (await js.InvokeAsync<bool>("confirm",$"Delete Product with
ProductId={prodid}?"))
        {
            //IProductsRepository irep = new ProductRepositorySQL();
            bool ret = irep.DeleteProduct(prodid);
            if (ret == true)
            {
                Msg = $"Product with ProductId={prodid} deleted..";
                catselection = catid.ToString();
                PList = irep.GetProductsByCatId(catid);
                //navManager.NavigateTo($"showproducts/{catid}", false);
            }
            else
                Msg = "Problem in deleting product";
        }
    }
}
```

Add a razor component called ViewCart.razor with the following code in it.

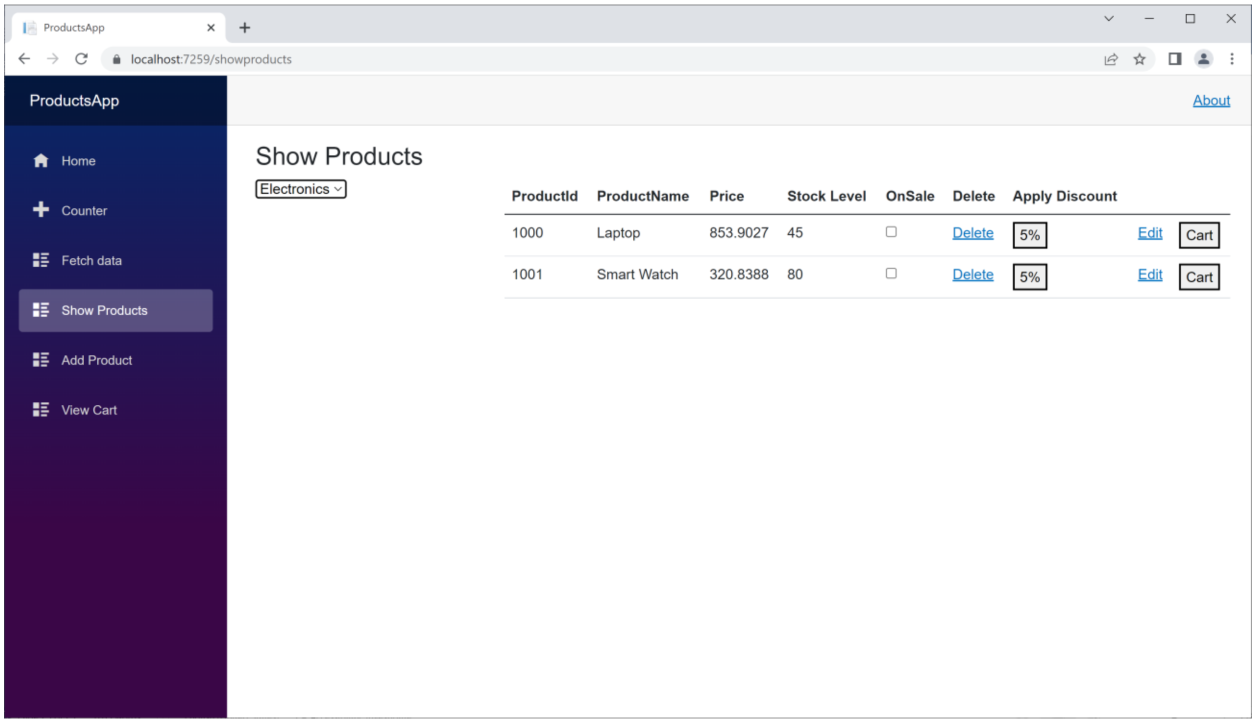
```
@page "/viewcart"
@using ProductsApp.ModelsVM
@using ProductsApp.DataLayer
@using ProductsApp.Pages.Components
@inject ICartRepository cartRepository
<br />
<ShoppingCart MyCart="CList" />
```

```
@code {
    List<CartItem> CList;
    protected async override void OnAfterRender(bool firstRender)
    {
        CList = await cartRepository.GetCart();
        await InvokeAsync(StateHasChanged);
    }
}
```

Add the following links to the NavMenu.razor.

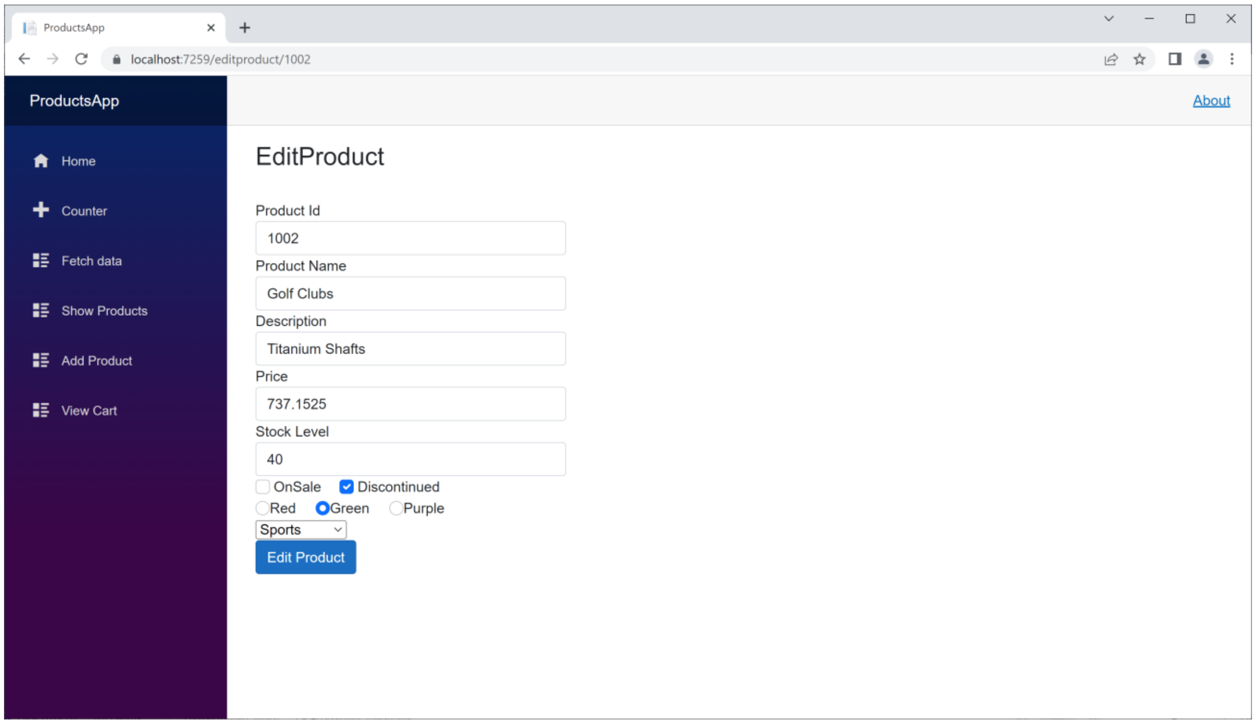
```
<div class="nav-item px-3">
            <NavLink class="nav-link" href="showproducts">
                <span class="oi oi-list-rich" aria-hidden="true"></span> Show
Products
            </NavLink>
        </div>
         <div class="nav-item px-3">
            <NavLink class="nav-link" href="addnewprod">
                <span class="oi oi-list-rich" aria-hidden="true"></span> Add Product
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="viewcart">
                <span class="oi oi-list-rich" aria-hidden="true"></span> View Cart
            </NavLink>
        </div>
```

Build and test the application. You will be able to see the products, add a product to the shopping cart, edit the shopping cart, edit a product, add a new product and delete a product.



Show Products

ProductId	ProductName	Price	Stock Level	OnSale	Delete	Apply Discount		
1000	Laptop	853.9027	45	☐	Delete	5%	Edit	Cart
1001	Smart Watch	320.8338	80	☐	Delete	5%	Edit	Cart



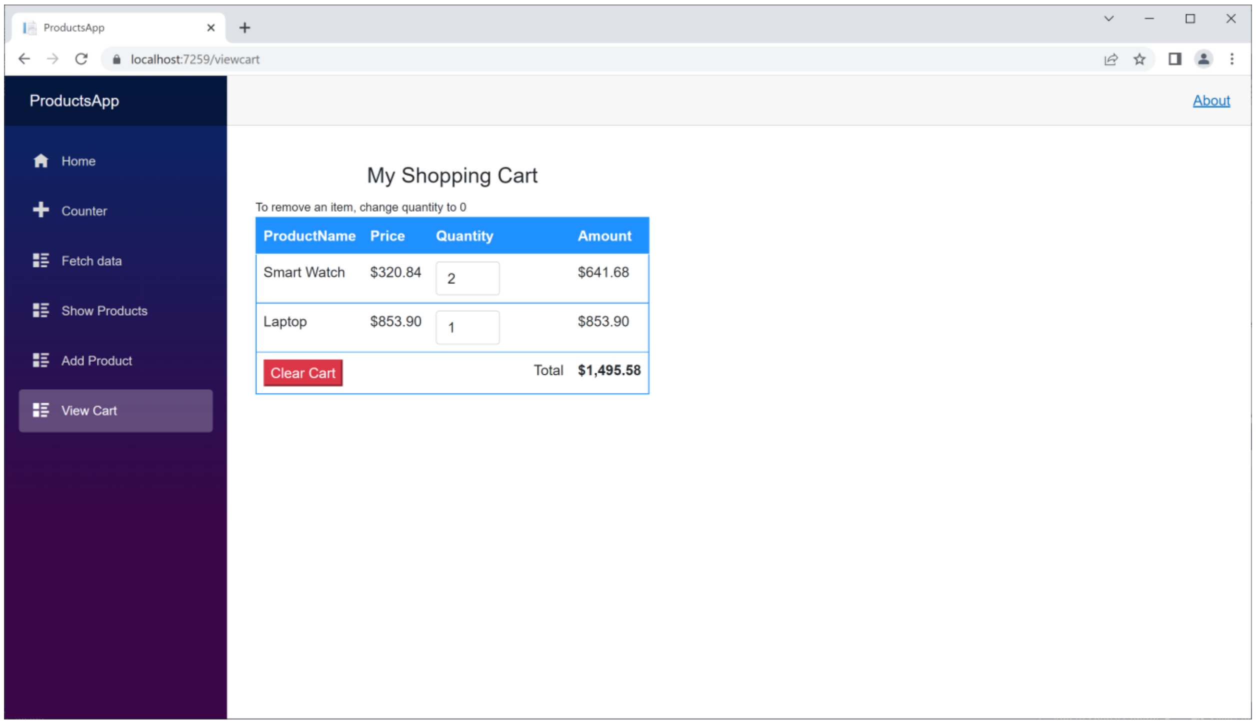
EditProduct

Product Id: 1002
Product Name: Golf Clubs
Description: Titanium Shafts
Price: 737.1525
Stock Level: 40

[ ] OnSale [✓] Discontinued

Red [✓ Green] Purple

Sports



My Shopping Cart

To remove an item, change quantity to 0

ProductName	Price	Quantity	Amount
Smart Watch	$320.84	2	$641.68
Laptop	$853.90	1	$853.90
Clear Cart		Total	**$1,495.58**