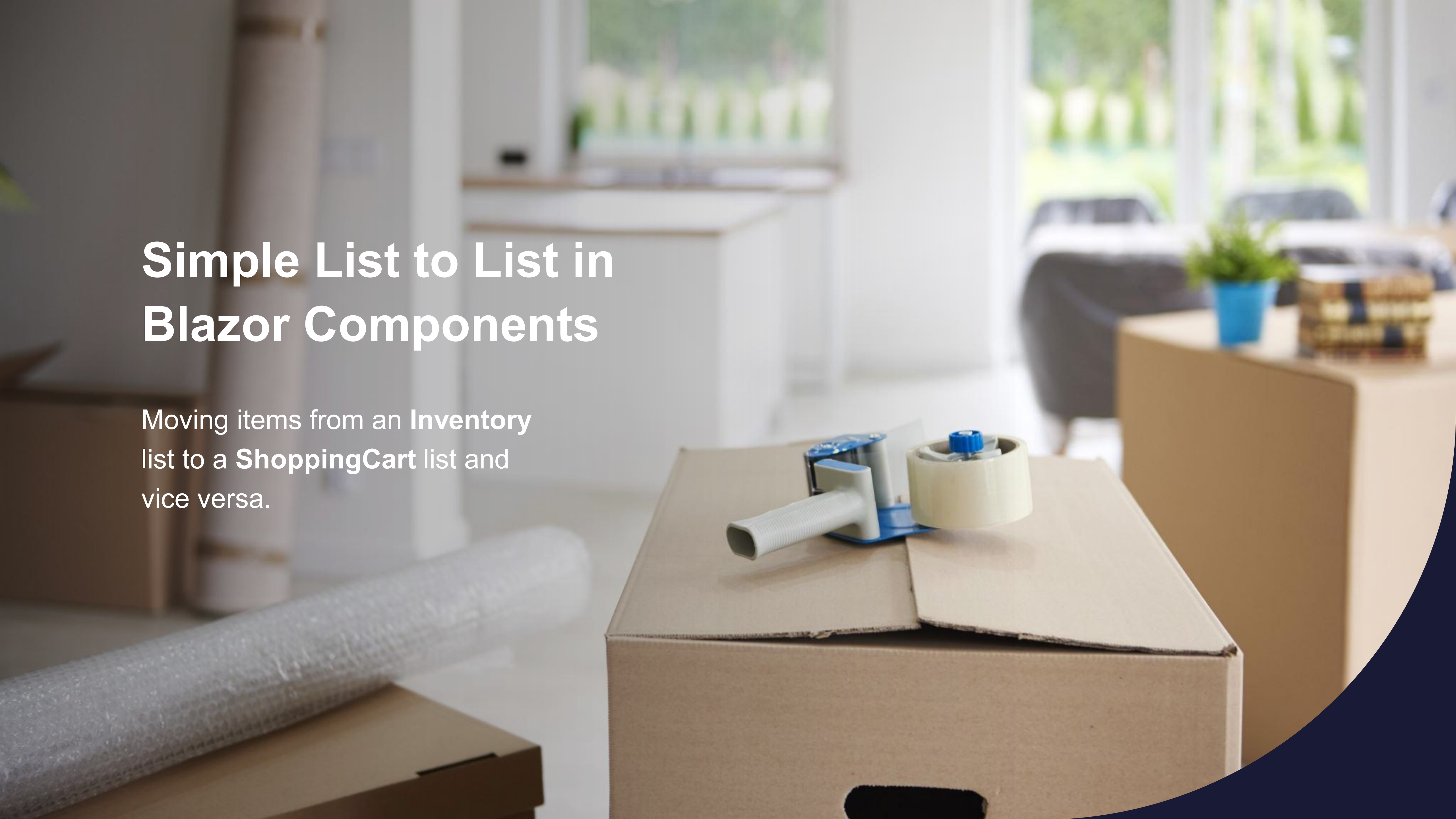
Session 9 – Simple List to List and Simple Non-Index List



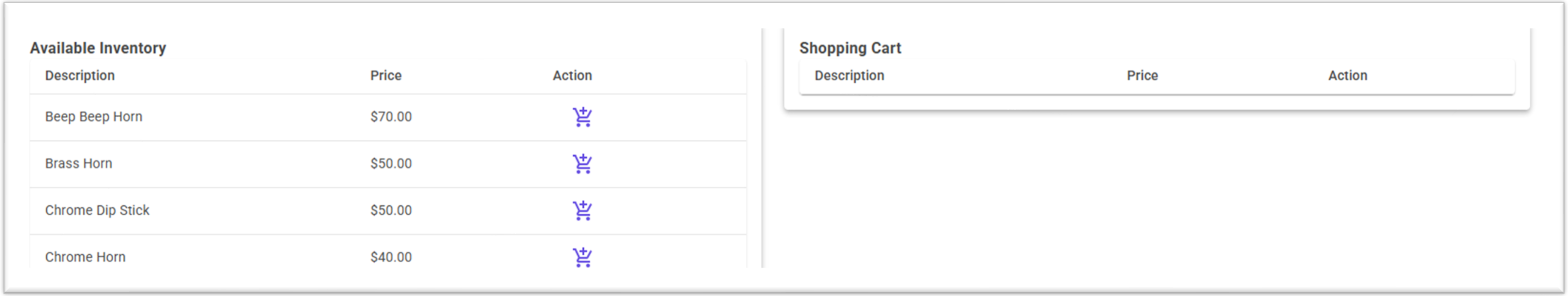
Simple List to List in Blazor Components

Moving items from an **Inventory** list to a **ShoppingCart** list and vice versa.

Simple List-to-List Operations

- **Scenario**: Imagine an online store where users can browse a catalog of products and add items to their shopping cart. As they add items to the cart, those items disappear from the catalog and appear in their cart. Similarly, if they remove an item from the cart, it reappears in the catalog.
- **Usage**: This is an example of a simple list-to-list operation. The application keeps track of two lists: the inventory (catalog) and the shopping cart. Items move between these two lists based on user actions.

Blazor Page Output

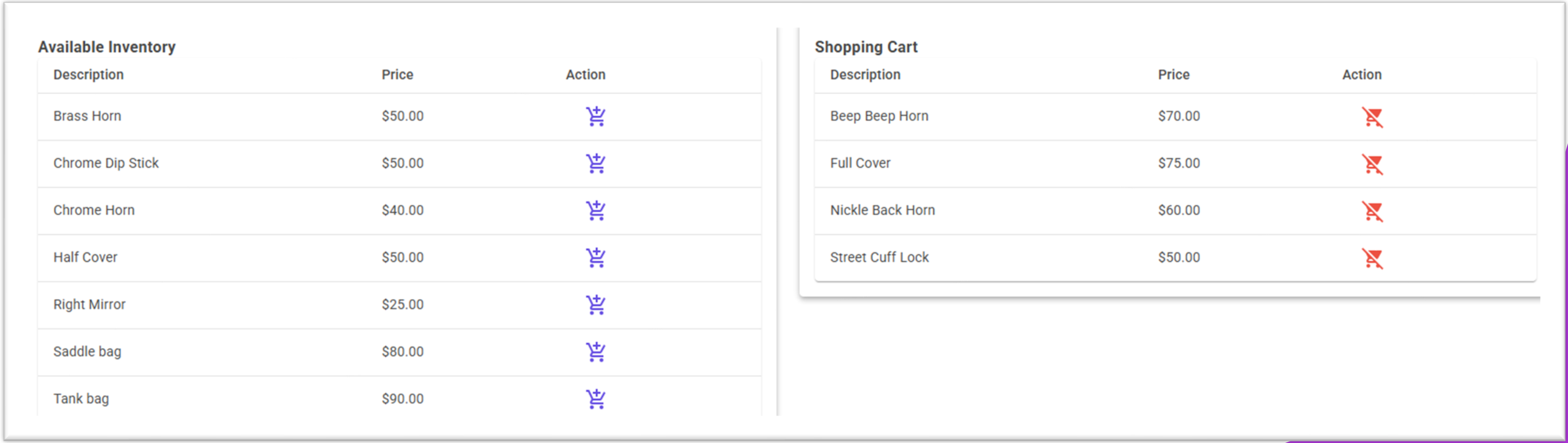


Available Inventory

Description	Price	Action
Beep Beep Horn	$70.00	🛒
Brass Horn	$50.00	🛒
Chrome Dip Stick	$50.00	🛒
Chrome Horn	$40.00	🛒

Shopping Cart

Description	Price	Action



Available Inventory

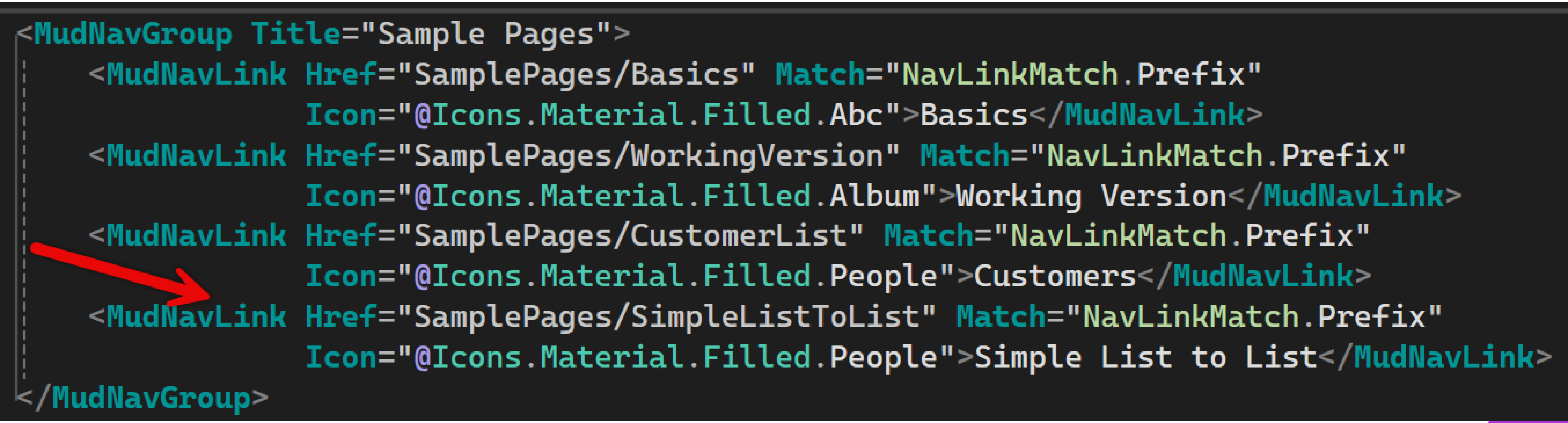
Description	Price	Action
Brass Horn	$50.00	🛒
Chrome Dip Stick	$50.00	🛒
Chrome Horn	$40.00	🛒
Half Cover	$50.00	🛒
Right Mirror	$25.00	🛒
Saddle bag	$80.00	🛒
Tank bag	$90.00	🛒

Shopping Cart

Description	Price	Action
Beep Beep Horn	$70.00	🛒
Full Cover	$75.00	🛒
Nickle Back Horn	$60.00	🛒
Street Cuff Lock	$50.00	🛒

Navigation Manager

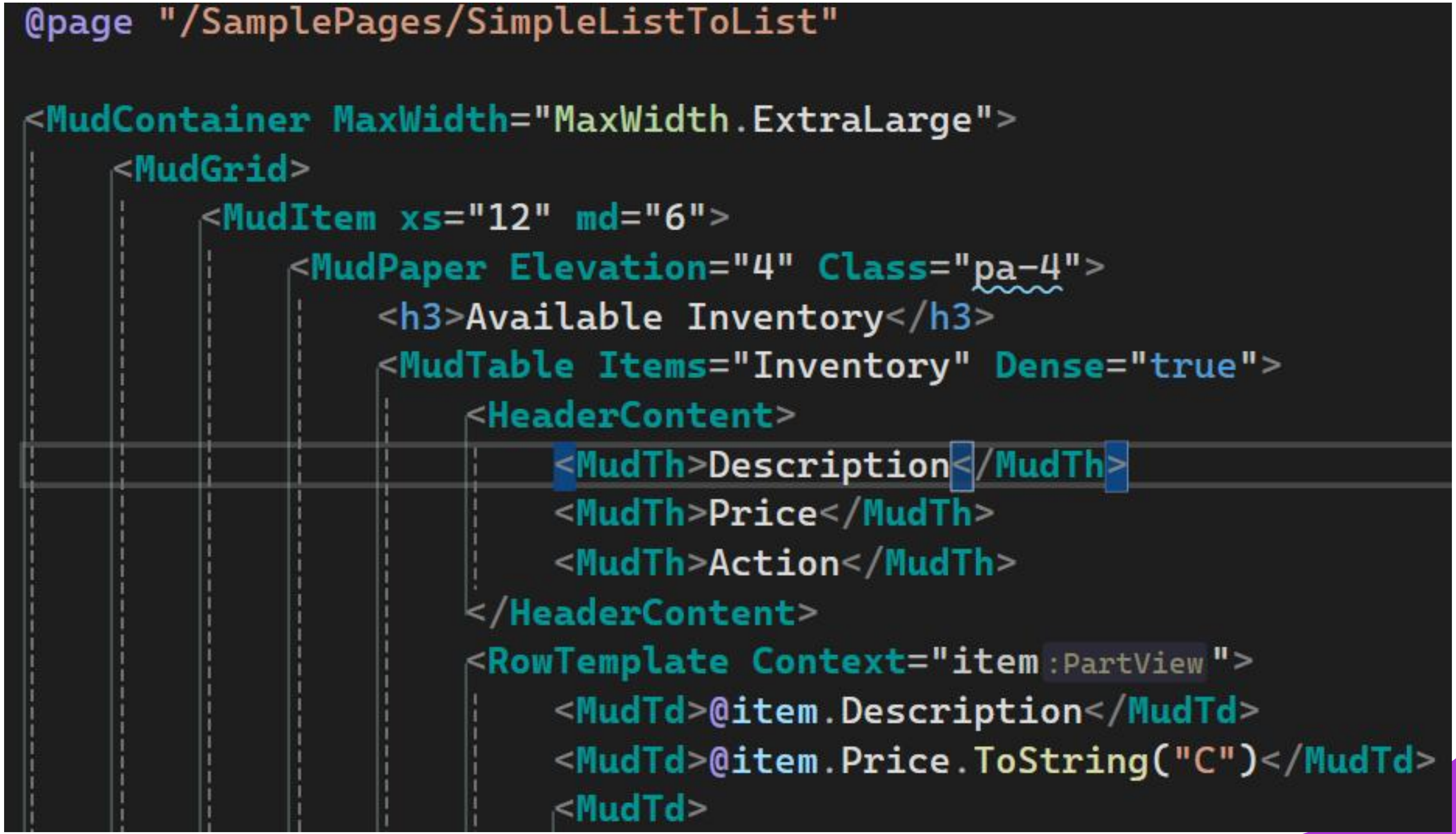
NavigationManager.razor



```
<MudNavGroup Title="Sample Pages">
    <MudNavLink Href="SamplePages/Basics" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.Abc">Basics</MudNavLink>
    <MudNavLink Href="SamplePages/WorkingVersion" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.Album">Working Version</MudNavLink>
    <MudNavLink Href="SamplePages/CustomerList" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.People">Customers</MudNavLink>
    <MudNavLink Href="SamplePages/SimpleListToList" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.People">Simple List to List</MudNavLink>
</MudNavGroup>
```

Blazor Page

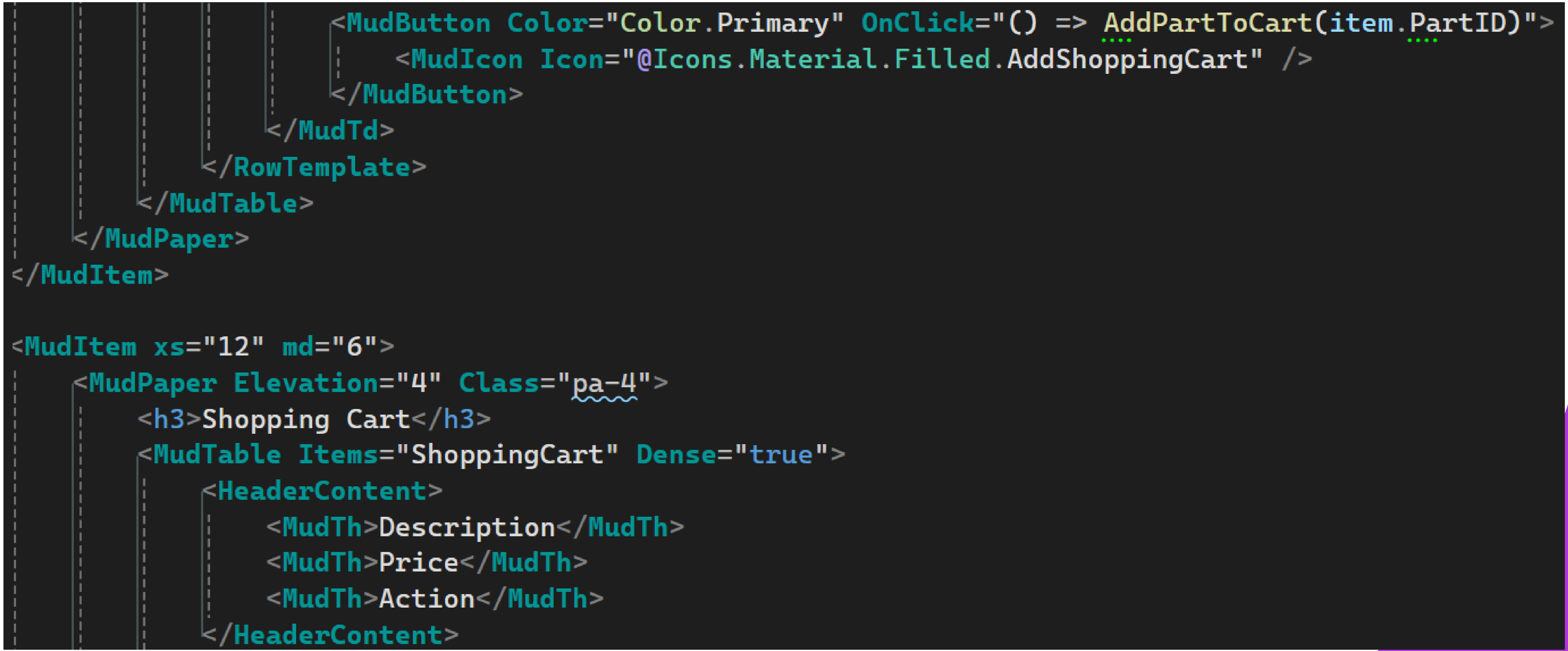
SimpleListToList.razor



```
@page "/SamplePages/SimpleListToList"

<MudContainer MaxWidth="MaxWidth.ExtraLarge">
    <MudGrid>
        <MudItem xs="12" md="6">
            <MudPaper Elevation="4" Class="pa-4">
                <h3>Available Inventory</h3>
                <MudTable Items="Inventory" Dense="true">
                    <HeaderContent>
                        <MudTh>Description</MudTh>
                        <MudTh>Price</MudTh>
                        <MudTh>Action</MudTh>
                    </HeaderContent>
                    <RowTemplate Context="item :PartView ">
                        <MudTd>@item.Description</MudTd>
                        <MudTd>@item.Price.ToString("C")</MudTd>
                        <MudTd>
```

Blazor Page

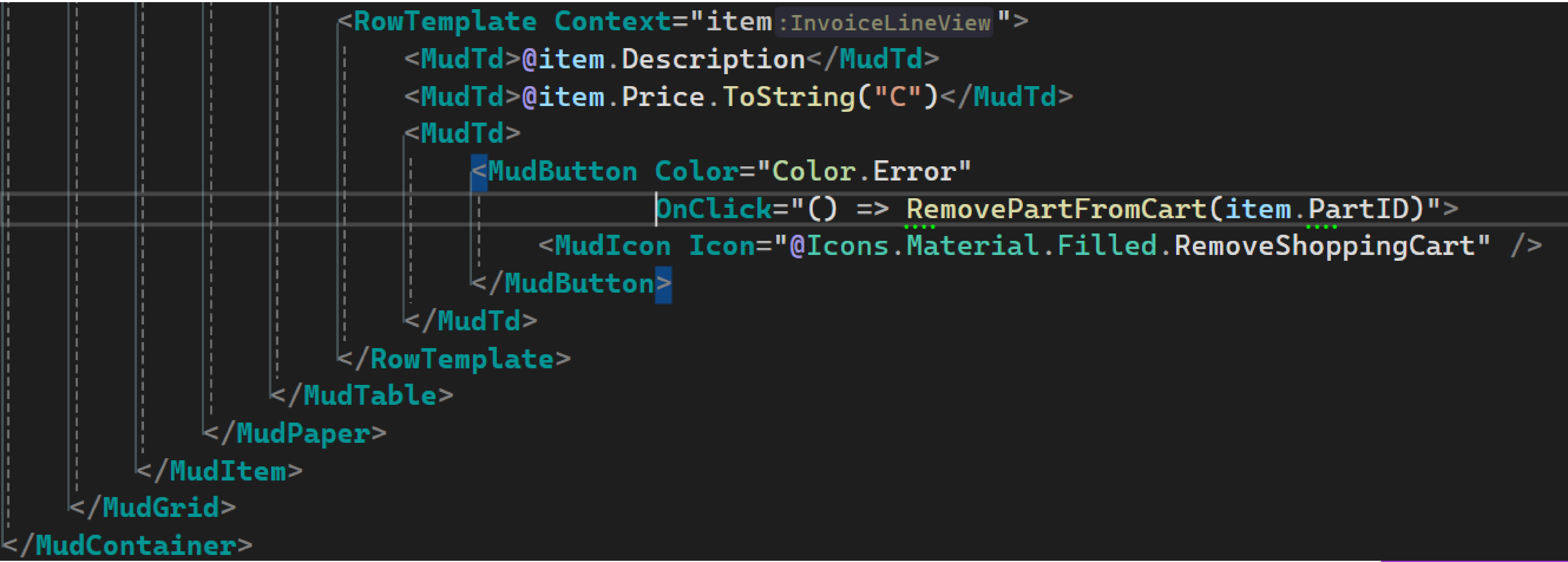
SimpleListToList.razor



```
                    <MudButton Color="Color.Primary" OnClick="() => AddPartToCart(item.PartID)">
                        <MudIcon Icon="@Icons.Material.Filled.AddShoppingCart" />
                    </MudButton>
                </MudTd>
            </RowTemplate>
        </MudTable>
    </MudPaper>
</MudItem>

<MudItem xs="12" md="6">
    <MudPaper Elevation="4" Class="pa-4">
        <h3>Shopping Cart</h3>
        <MudTable Items="ShoppingCart" Dense="true">
            <HeaderContent>
                <MudTh>Description</MudTh>
                <MudTh>Price</MudTh>
                <MudTh>Action</MudTh>
            </HeaderContent>
```

Blazor Page

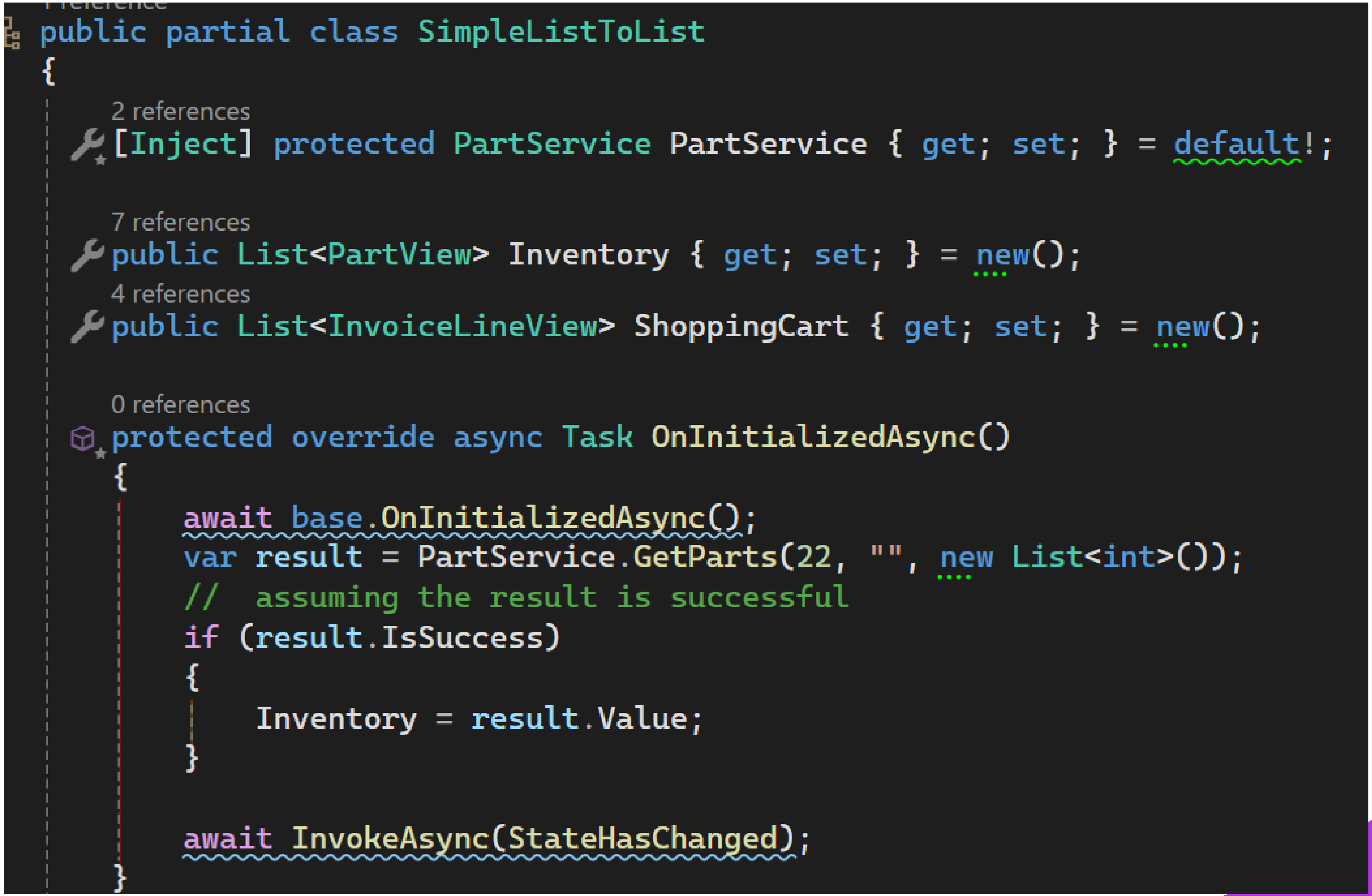
SimpleListToList.razor



```
                    <RowTemplate Context="item :InvoiceLineView">
                        <MudTd>@item.Description</MudTd>
                        <MudTd>@item.Price.ToString("C")</MudTd>
                        <MudTd>
                            <MudButton Color="Color.Error"
                                       OnClick="() => RemovePartFromCart(item.PartID)">
                                <MudIcon Icon="@Icons.Material.Filled.RemoveShoppingCart" />
                            </MudButton>
                        </MudTd>
                    </RowTemplate>
                </MudTable>
            </MudPaper>
        </MudItem>
    </MudGrid>
</MudContainer>
```

Setup & Properties

SimpleListToList.razor.cs



```
public partial class SimpleListToList
{
    2 references
    [Inject] protected PartService PartService { get; set; } = default!;

    7 references
    public List<PartView> Inventory { get; set; } = new();
    4 references
    public List<InvoiceLineView> ShoppingCart { get; set; } = new();

    0 references
    protected override async Task OnInitializedAsync()
    {
        await base.OnInitializedAsync();
        var result = PartService.GetParts(22, "", new List<int>());
        //  assuming the result is successful
        if (result.IsSuccess)
        {
            Inventory = result.Value;
        }

        await InvokeAsync(StateHasChanged);
    }
```

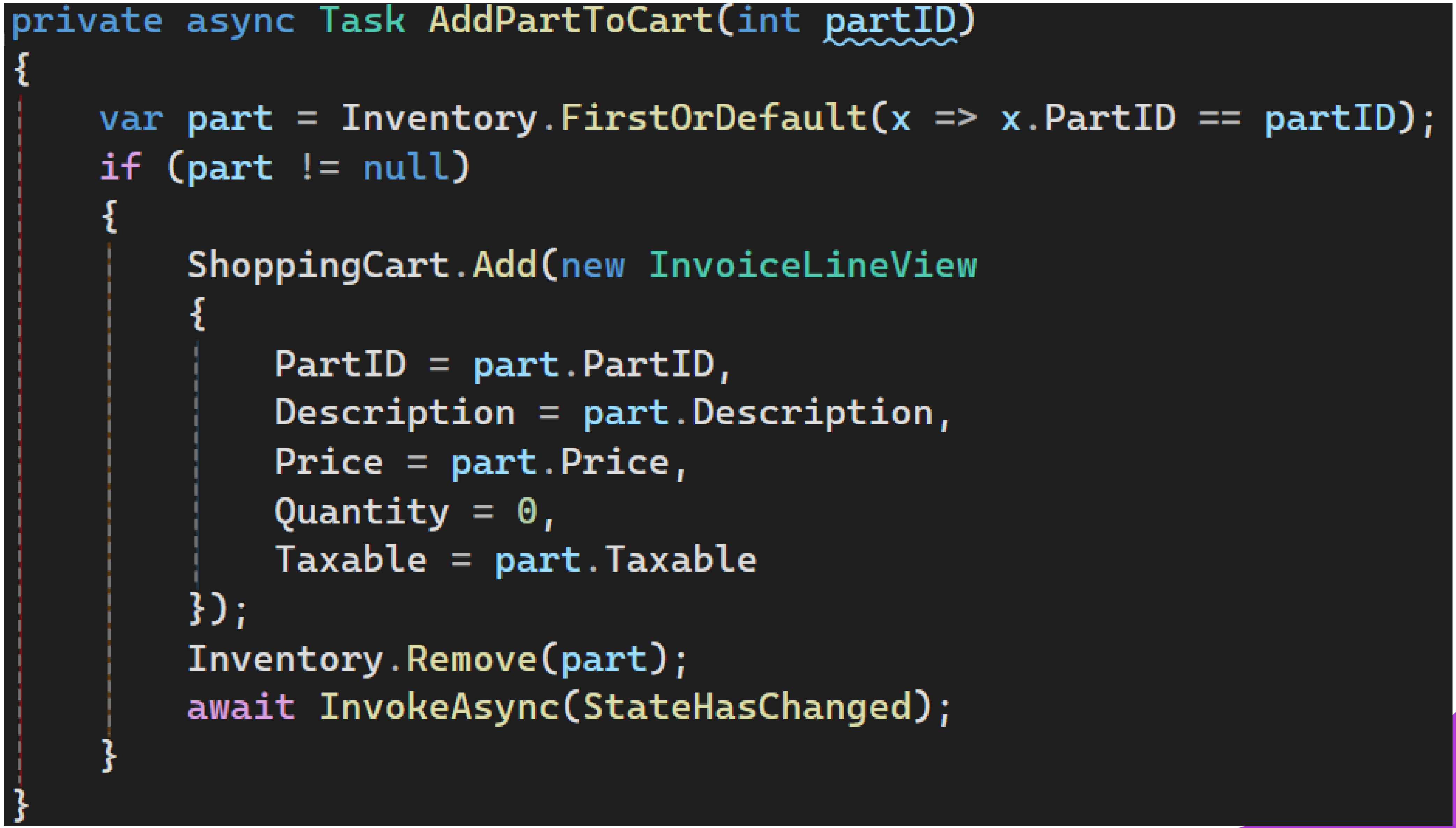
Add Inventory To Shopping Cart

- Purpose:
  - Move a selected part from the Inventory to the shopping cart.
- Steps:
  1. Identify the part in the Inventory based on its ID.
  2. Create a new InvoiceLineView object from the selected part.
  3. Add the InvoiceLineView object to ShoppingCart.
  4. Remove the part from Inventory.
  5. Update the UI.

apart

Add Inventory To Shopping Cart

SimpleListToList.razor.cs



```
private async Task AddPartToCart(int partID)
{
    var part = Inventory.FirstOrDefault(x => x.PartID == partID);
    if (part != null)
    {
        ShoppingCart.Add(new InvoiceLineView
        {
            PartID = part.PartID,
            Description = part.Description,
            Price = part.Price,
            Quantity = 0,
            Taxable = part.Taxable
        });
        Inventory.Remove(part);
        await InvokeAsync(StateHasChanged);
    }
}
```

Remove Part From Shopping Cart

Purpose:

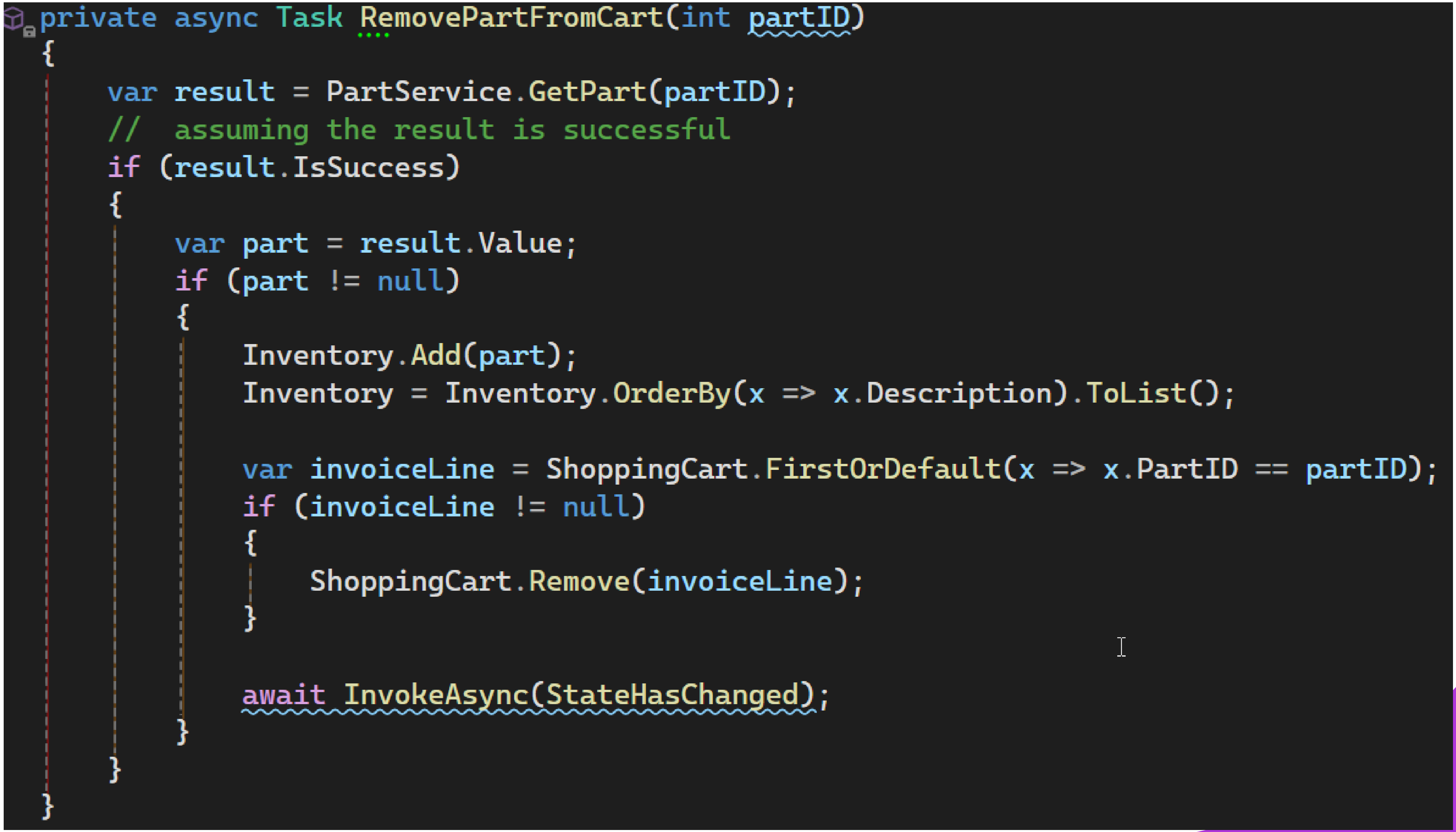
- Move a part from the shopping cart back to the Inventory.

Bullet 2: Steps:

1. Add the part back to Inventory using PartService GetPart method.
2. Sort the Inventory based on the description.
3. Identify the corresponding InvoiceLineView object in the ShoppingCart based on its ID.
4. Remove the InvoiceLineView object from ShoppingCart.
5. Update the UI.

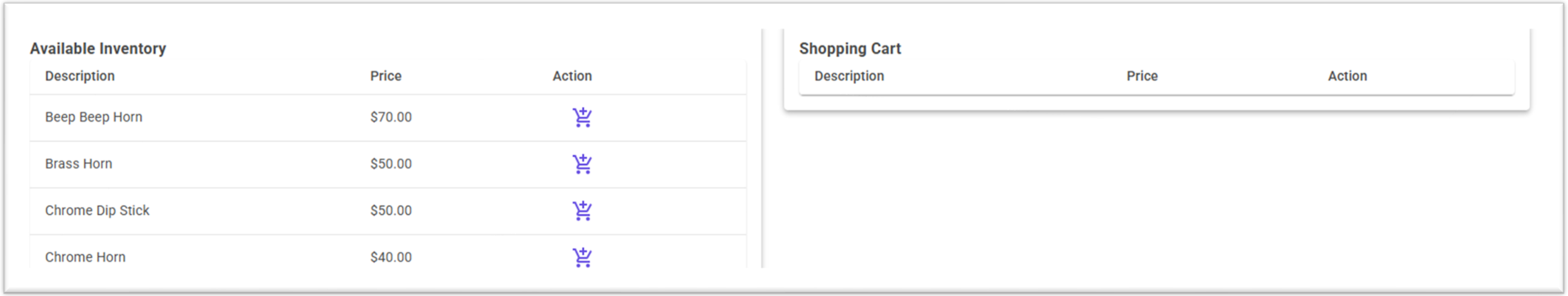
SimpleListToList.razor.cs

Remove Part From Shopping Cart



```
private async Task RemovePartFromCart(int partID)
{
    var result = PartService.GetPart(partID);
    //  assuming the result is successful
    if (result.IsSuccess)
    {
        var part = result.Value;
        if (part != null)
        {
            Inventory.Add(part);
            Inventory = Inventory.OrderBy(x => x.Description).ToList();

            var invoiceLine = ShoppingCart.FirstOrDefault(x => x.PartID == partID);
            if (invoiceLine != null)
            {
                ShoppingCart.Remove(invoiceLine);
            }

            await InvokeAsync(StateHasChanged);
        }
    }
}
```

Blazor Page Output

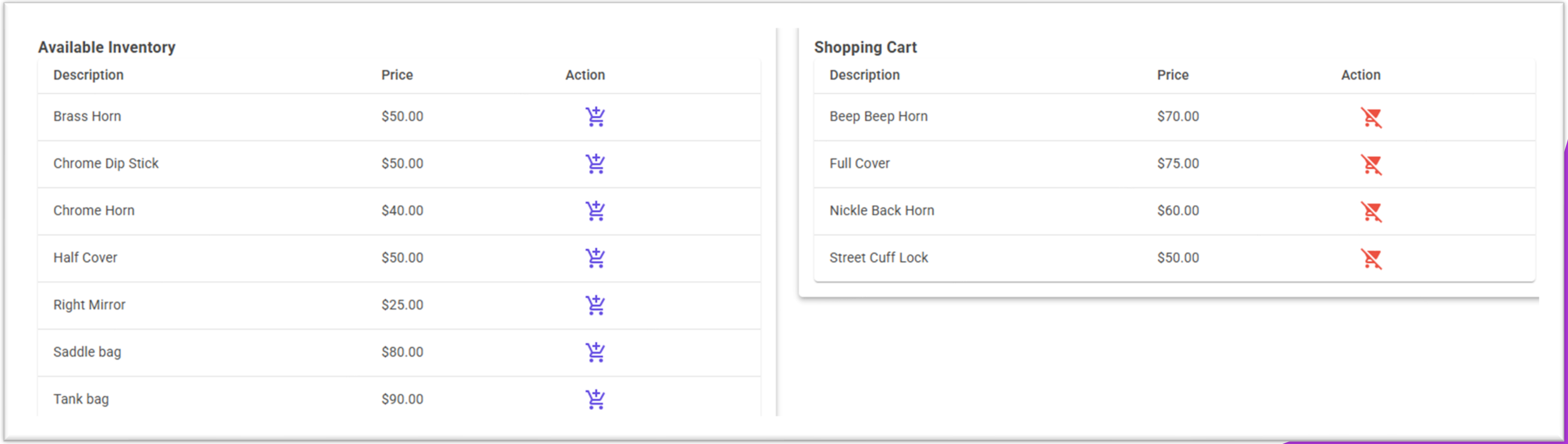


Available Inventory

Description	Price	Action
Beep Beep Horn	$70.00	🛒
Brass Horn	$50.00	🛒
Chrome Dip Stick	$50.00	🛒
Chrome Horn	$40.00	🛒

Shopping Cart

Description	Price	Action



Available Inventory

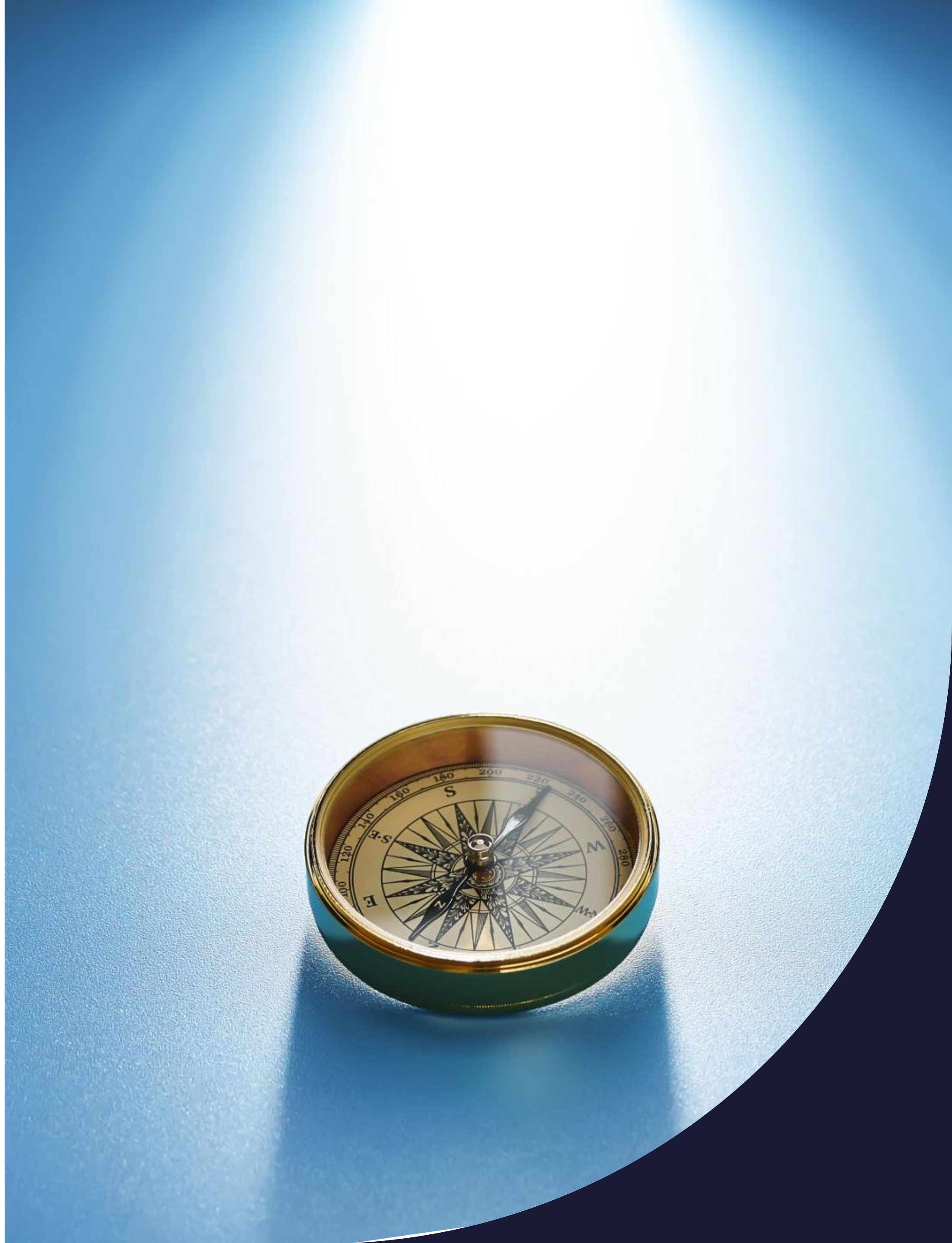
Description	Price	Action
Brass Horn	$50.00	🛒
Chrome Dip Stick	$50.00	🛒
Chrome Horn	$40.00	🛒
Half Cover	$50.00	🛒
Right Mirror	$25.00	🛒
Saddle bag	$80.00	🛒
Tank bag	$90.00	🛒

Shopping Cart

Description	Price	Action
Beep Beep Horn	$70.00	❌
Full Cover	$75.00	❌
Nickle Back Horn	$60.00	❌
Street Cuff Lock	$50.00	❌

Simple Non-Index List in Blazor Components

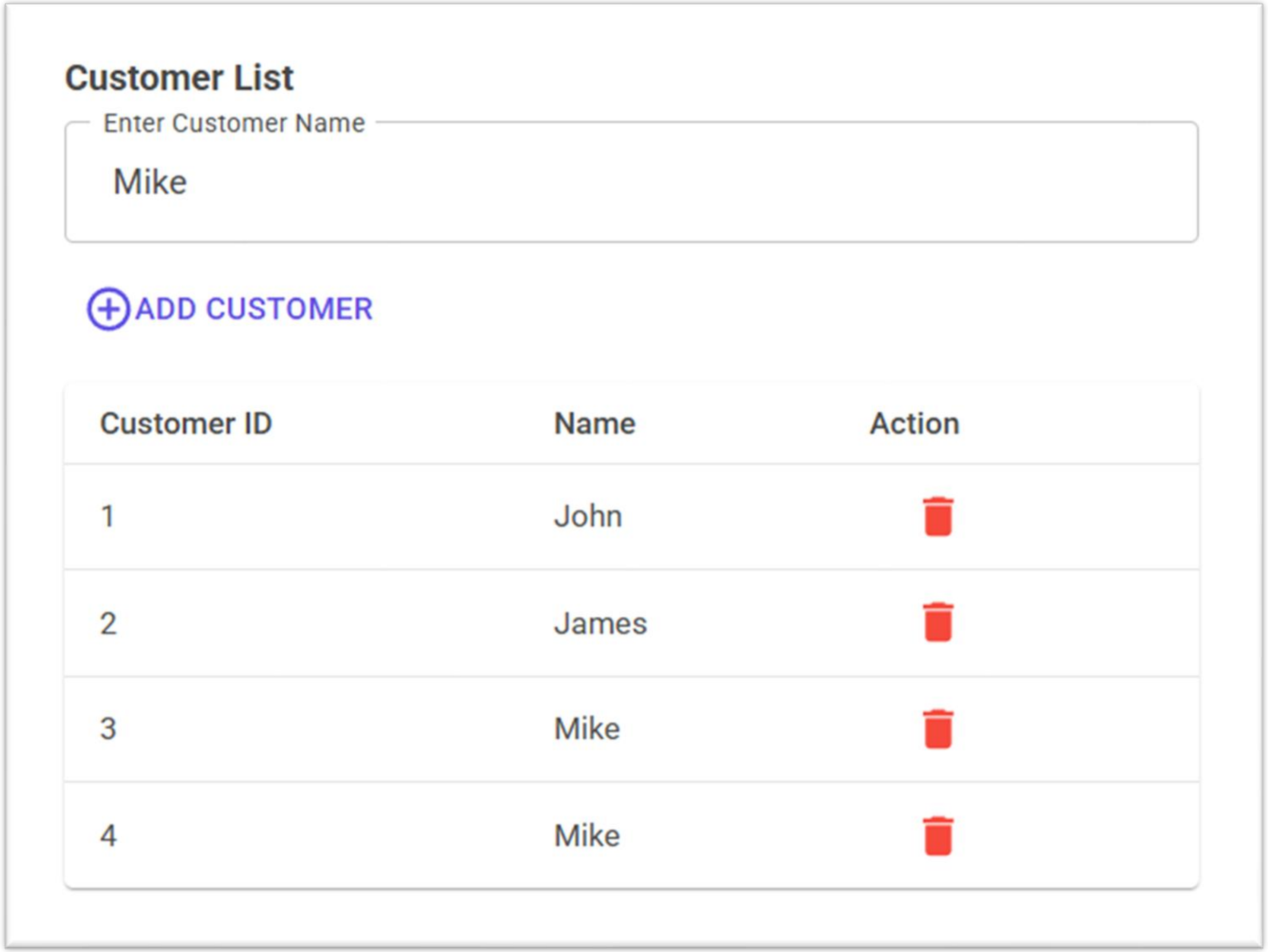
- Adding random strings to a list.



Simple Non-Indexed List Operations

- **Scenario**: Consider a registration system for a workshop. Participants can sign up and their names get added to a list. If a participant decides to unregister, their name gets removed. Each participant has a unique ID that gets assigned upon registration.
- **Usage**: Here, the focus isn't on moving items between lists but on adding or removing them from a single list. The challenge is ensuring data integrity, especially when assigning unique identifiers or handling other attributes.

Simple Non-Index List



Customer List

Enter Customer Name

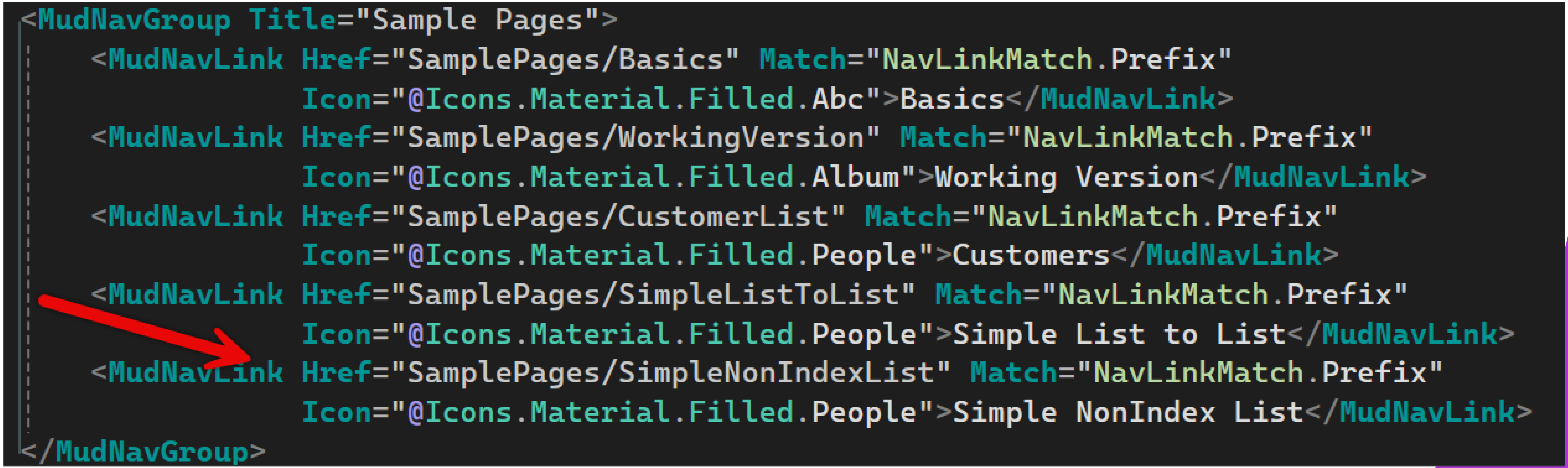
Mike

ADD CUSTOMER

Customer ID	Name	Action
1	John	🗑
2	James	🗑
3	Mike	🗑
4	Mike	🗑

Navigation Manager

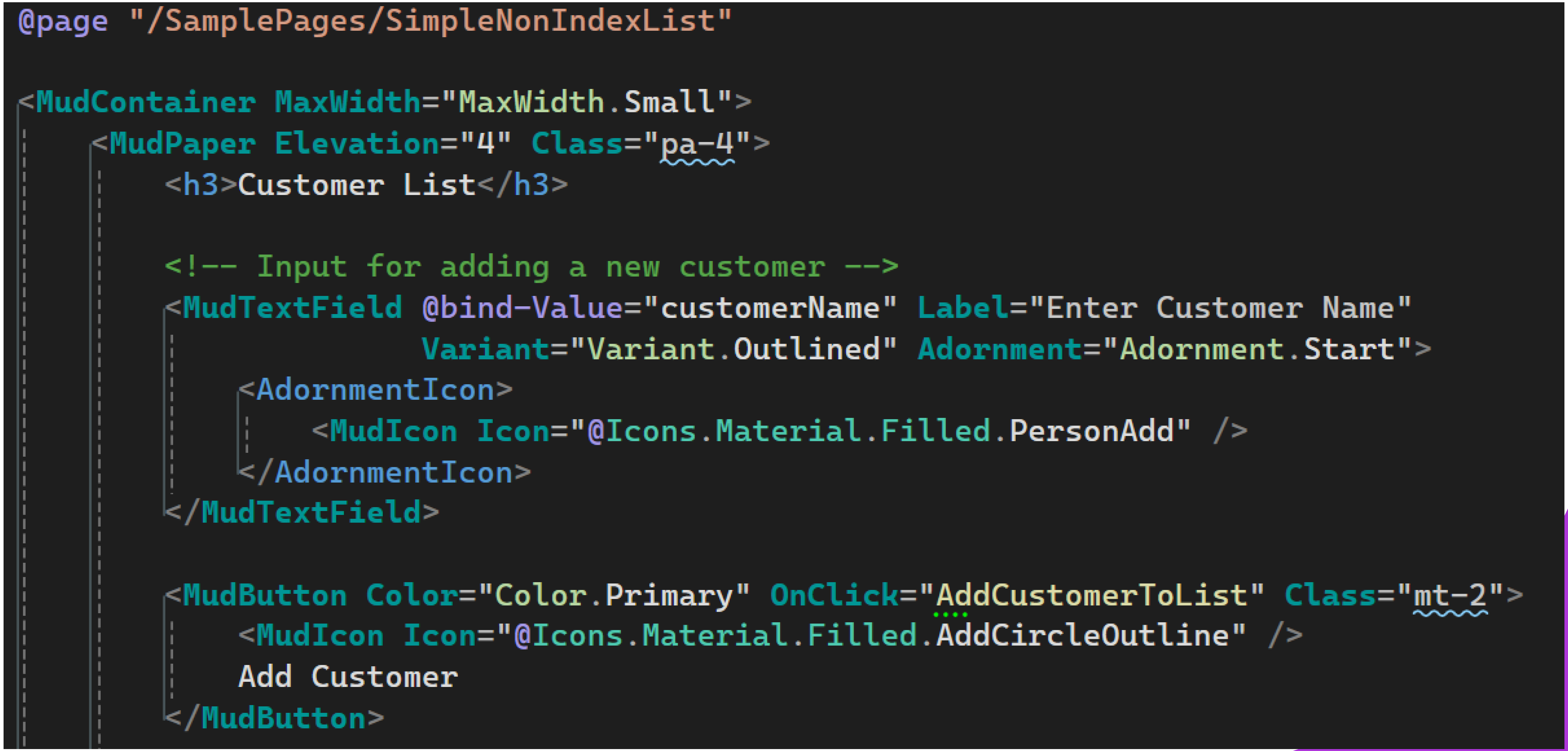
NavigationManager.razor



```
<MudNavGroup Title="Sample Pages">
    <MudNavLink Href="SamplePages/Basics" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.Abc">Basics</MudNavLink>
    <MudNavLink Href="SamplePages/WorkingVersion" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.Album">Working Version</MudNavLink>
    <MudNavLink Href="SamplePages/CustomerList" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.People">Customers</MudNavLink>
    <MudNavLink Href="SamplePages/SimpleListToList" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.People">Simple List to List</MudNavLink>
    <MudNavLink Href="SamplePages/SimpleNonIndexList" Match="NavLinkMatch.Prefix"
                Icon="@Icons.Material.Filled.People">Simple NonIndex List</MudNavLink>
</MudNavGroup>
```

Blazor Page

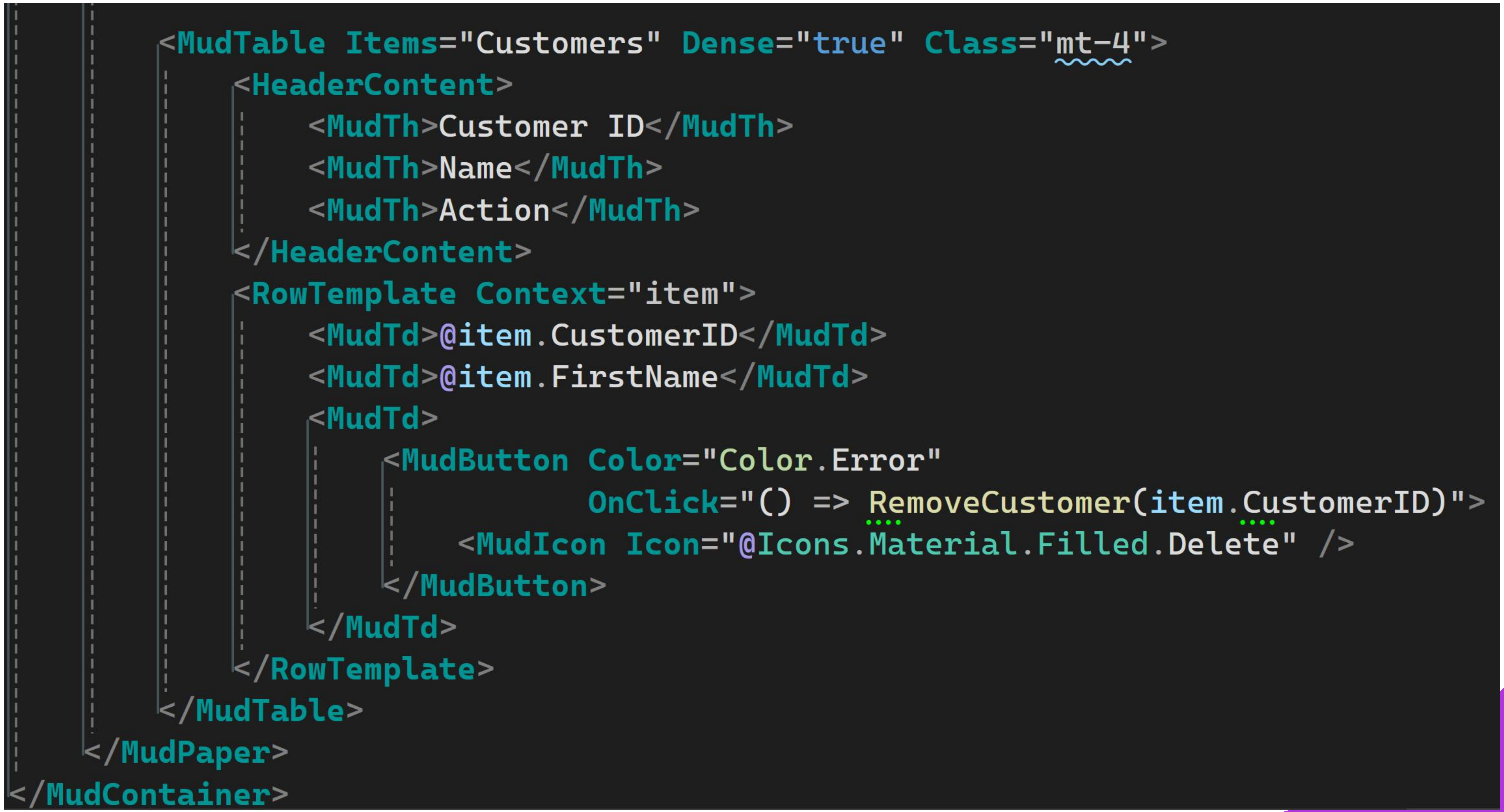
SimpleNonIndexList.razor



```
@page "/SamplePages/SimpleNonIndexList"

<MudContainer MaxWidth="MaxWidth.Small">
    <MudPaper Elevation="4" Class="pa-4">
        <h3>Customer List</h3>

        <!-- Input for adding a new customer -->
        <MudTextField @bind-Value="customerName" Label="Enter Customer Name"
                      Variant="Variant.Outlined" Adornment="Adornment.Start">
            <AdornmentIcon>
                <MudIcon Icon="@Icons.Material.Filled.PersonAdd" />
            </AdornmentIcon>
        </MudTextField>

        <MudButton Color="Color.Primary" OnClick="AddCustomerToList" Class="mt-2">
            <MudIcon Icon="@Icons.Material.Filled.AddCircleOutline" />
            Add Customer
        </MudButton>
```

Blazor Page

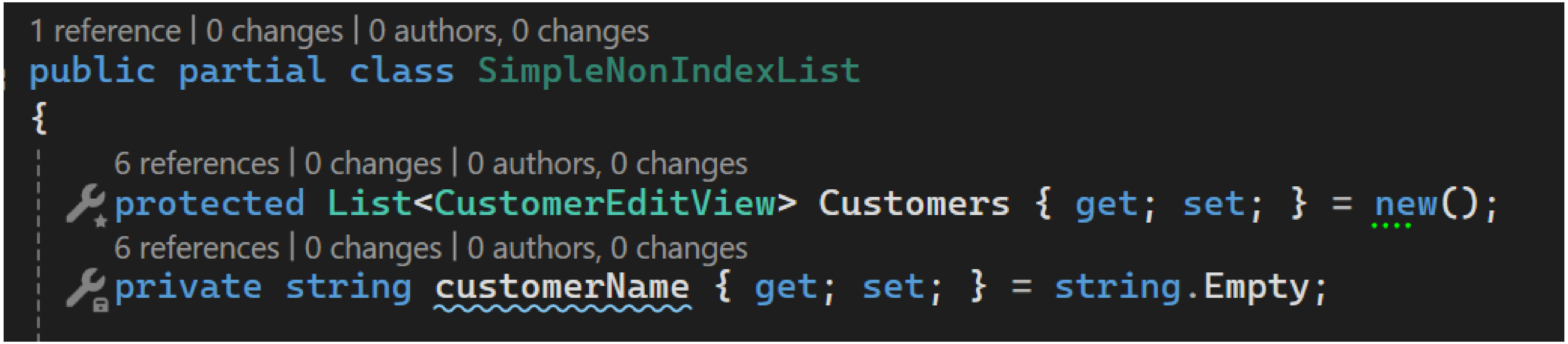
SimpleNonIndexList.razor



```
        <MudTable Items="Customers" Dense="true" Class="mt-4">
            <HeaderContent>
                <MudTh>Customer ID</MudTh>
                <MudTh>Name</MudTh>
                <MudTh>Action</MudTh>
            </HeaderContent>
            <RowTemplate Context="item">
                <MudTd>@item.CustomerID</MudTd>
                <MudTd>@item.FirstName</MudTd>
                <MudTd>
                    <MudButton Color="Color.Error"
                               OnClick="() => RemoveCustomer(item.CustomerID)">
                        <MudIcon Icon="@Icons.Material.Filled.Delete" />
                    </MudButton>
                </MudTd>
            </RowTemplate>
        </MudTable>
    </MudPaper>
</MudContainer>
```

Setup & Properties

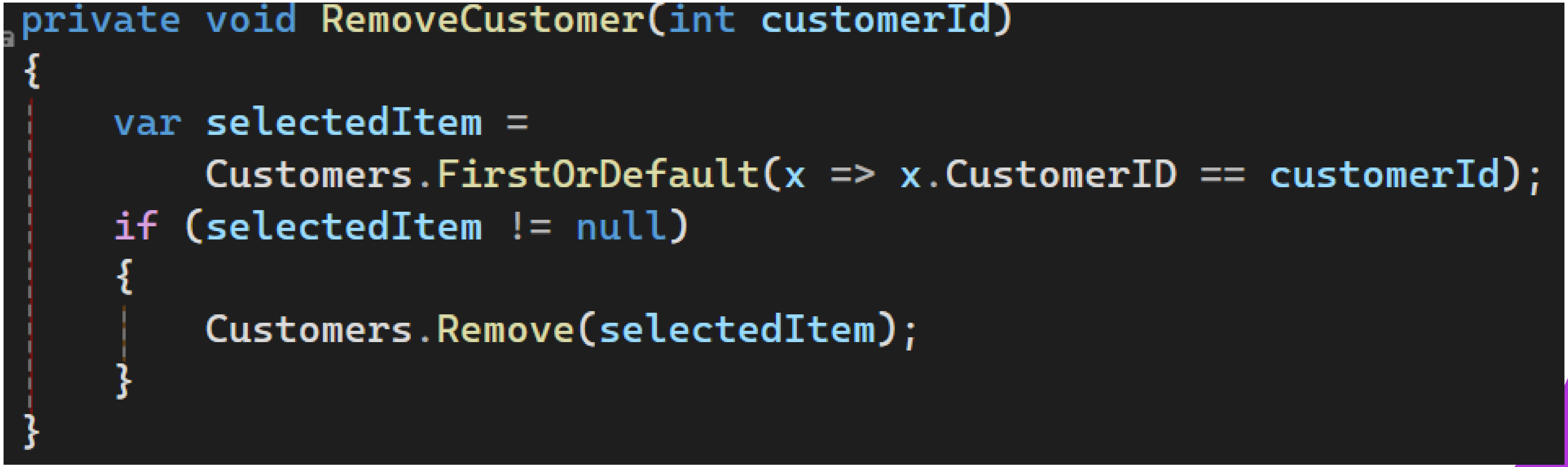
SimpleNonIndexList.razor.cs



```
1 reference | 0 changes | 0 authors, 0 changes
public partial class SimpleNonIndexList
{
    6 references | 0 changes | 0 authors, 0 changes
    protected List<CustomerEditView> Customers { get; set; } = new();
    6 references | 0 changes | 0 authors, 0 changes
    private string customerName { get; set; } = string.Empty;
```

Remove Customer

SimpleNonIndexList.razor.cs



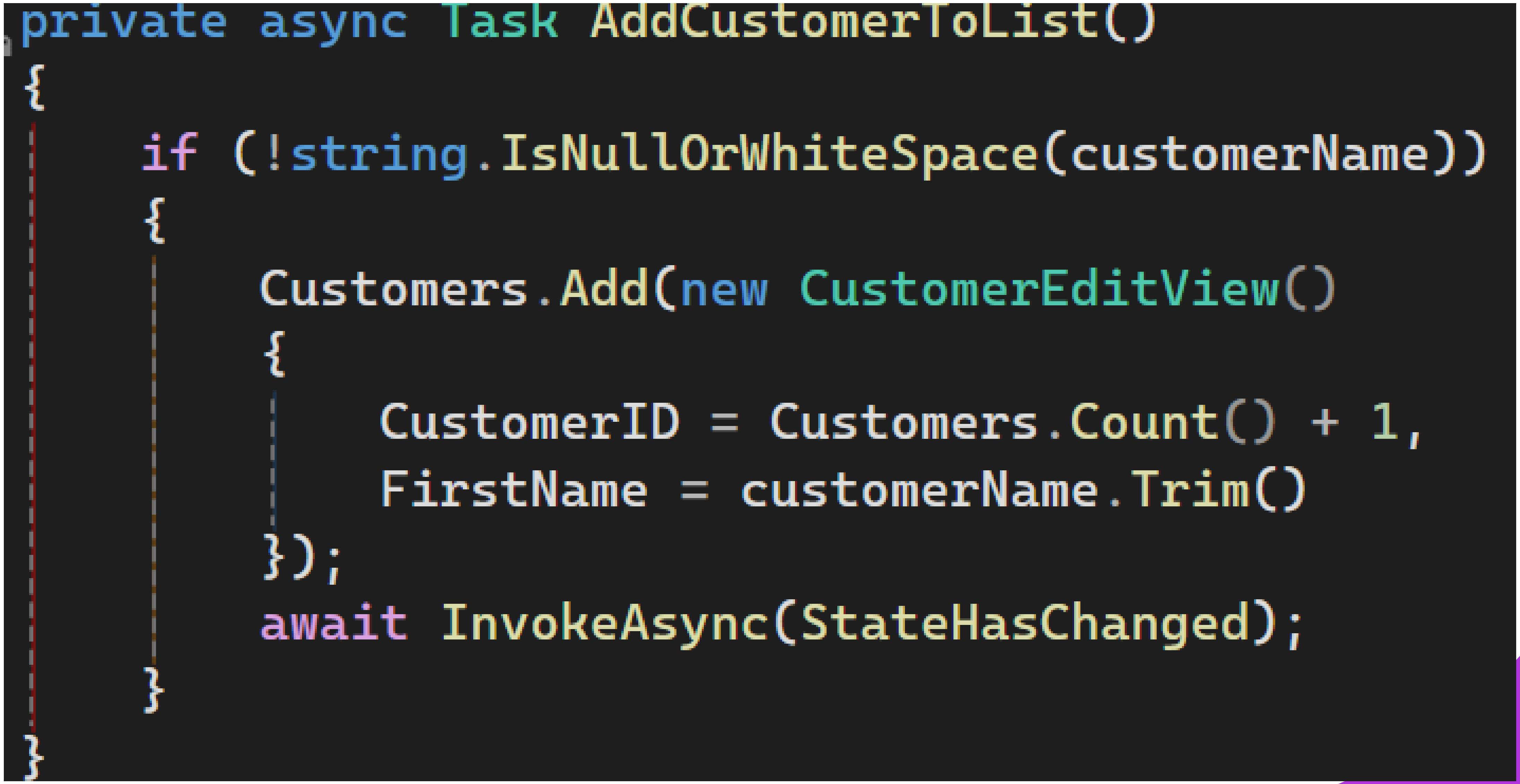
```
private void RemoveCustomer(int customerId)
{
    var selectedItem =
        Customers.FirstOrDefault(x => x.CustomerID == customerId);
    if (selectedItem != null)
    {
        Customers.Remove(selectedItem);
    }
}
```

Add Customer Name (BAD)

- The CustomerID is calculated by taking the count of existing customers and adding 1.
- In this method, if a customer were to be removed from the middle of the list (or from anywhere but the end) and then a new customer added, there could be two customers with the same ID. This can lead to all sorts of data integrity issues.

Add Customer Name (BAD)

SimpleNonIndexList.razor.cs



```
private async Task AddCustomerToList()
{
    if (!string.IsNullOrWhiteSpace(customerName))
    {
        Customers.Add(new CustomerEditView()
        {
            CustomerID = Customers.Count() + 1,
            FirstName = customerName.Trim()
        });
        await InvokeAsync(StateHasChanged);
    }
}
```

he code first checks if the Customers list is empty. If it is, it assigns a value of **1** as the ID. If it's not empty, it finds the highest current ID in the list and adds 1 to it.

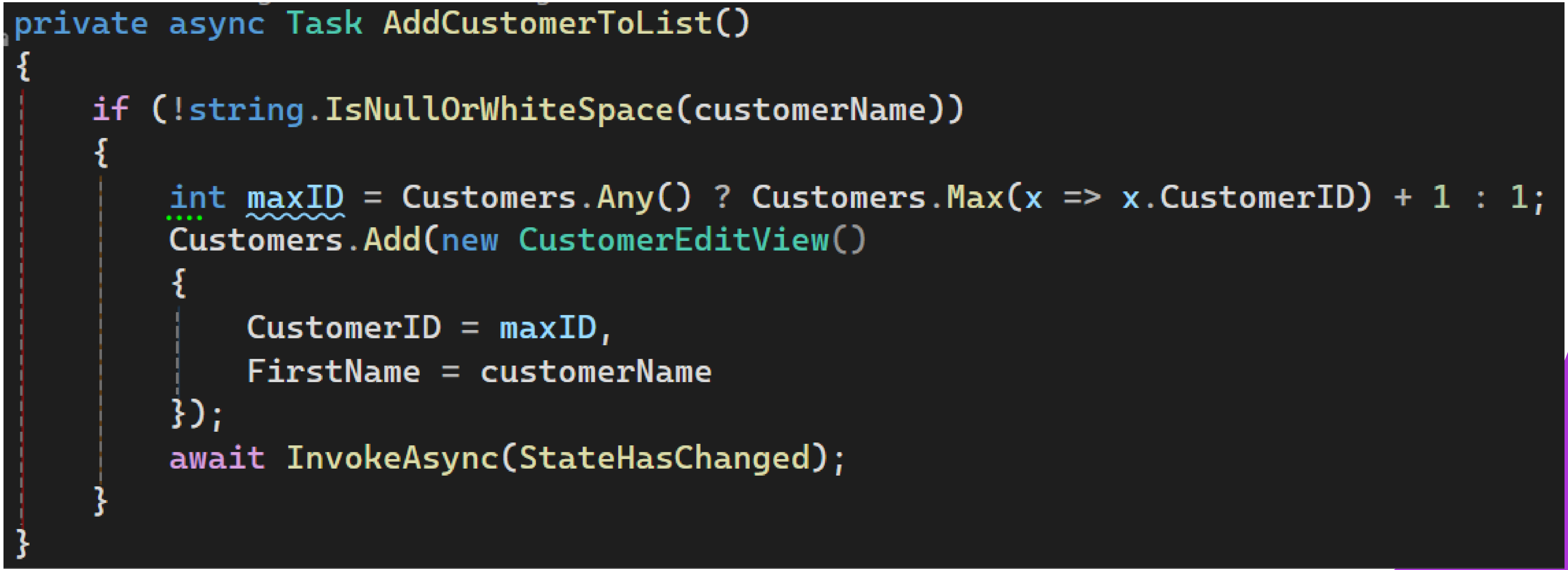
Add Customer Name (Good)

The code first checks if the Customers list is empty. If it is, it assigns a value of **1** as the ID. If it's not empty, it finds the highest current ID in the list and adds 1 to it.

This method, on the other hand, always ensures the new customer will have a unique ID that's one greater than the current highest ID, even if there are deletions in the list.

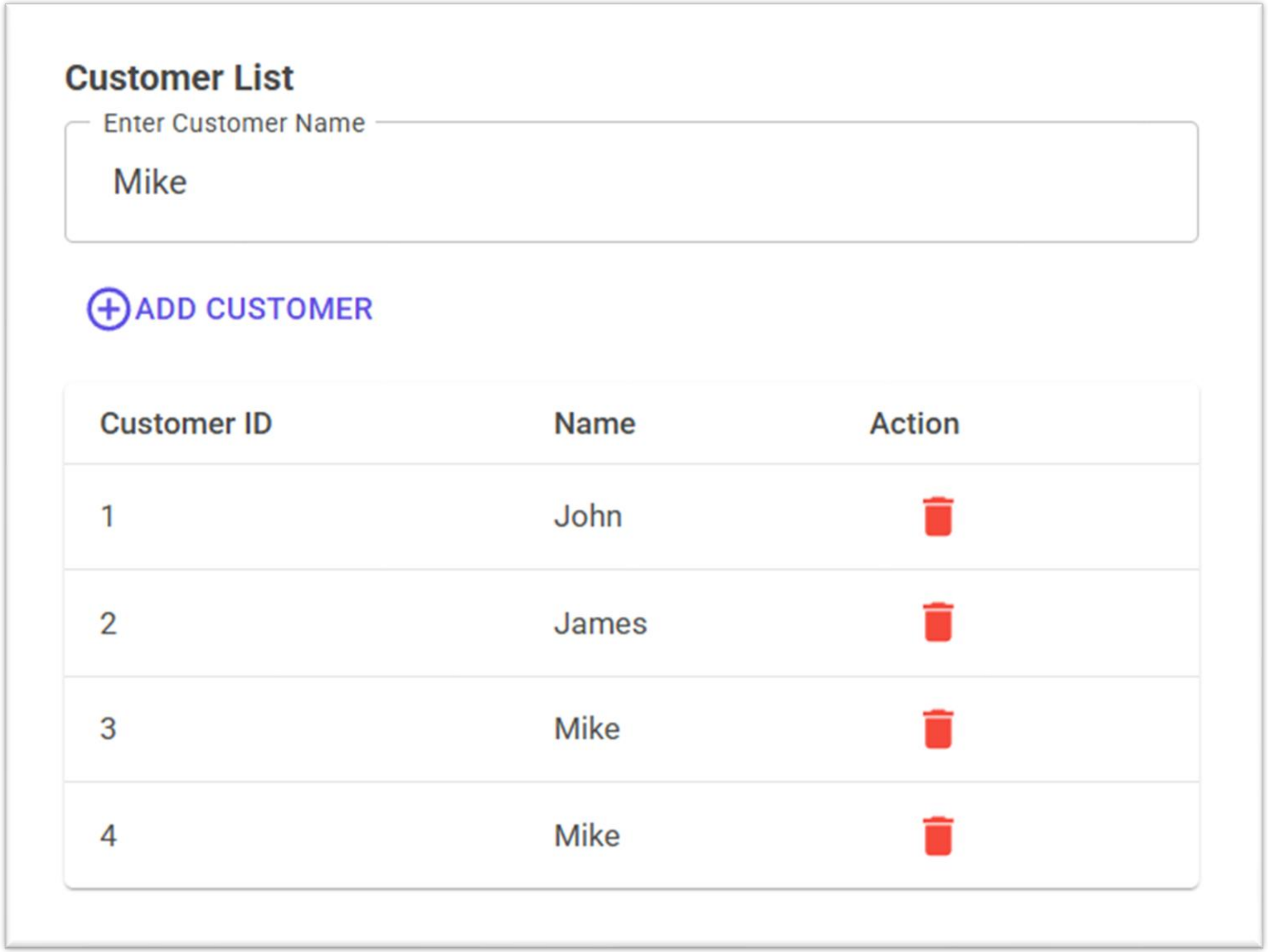
Add Customer Name (Good)

SimpleNonIndexList.razor.cs



```
private async Task AddCustomerToList()
{
    if (!string.IsNullOrWhiteSpace(customerName))
    {
        int maxID = Customers.Any() ? Customers.Max(x => x.CustomerID) + 1 : 1;
        Customers.Add(new CustomerEditView()
        {
            CustomerID = maxID,
            FirstName = customerName
        });
        await InvokeAsync(StateHasChanged);
    }
}
```

Simple Non-Index List



Customer List

Enter Customer Name

Mike

Customer ID	Name	Action
1	John	🗑
2	James	🗑
3	Mike	🗑
4	Mike	🗑

Statement Regarding Slide Accuracy and Potential Revisions

- Please note that the content of these PowerPoint slides is accurate to the best of my knowledge at the time of presentation. However, as new research and developments emerge, or to enhance the learning experience, these slides may be subject to updates or revisions in the future. I encourage you to stay engaged with the course materials and any announcements for the most current information