

Table of Contents

Create.....	2
Update and Delete.....	8

Demo: Shop Database, CRUD

Author: Baifan Zhou

Create

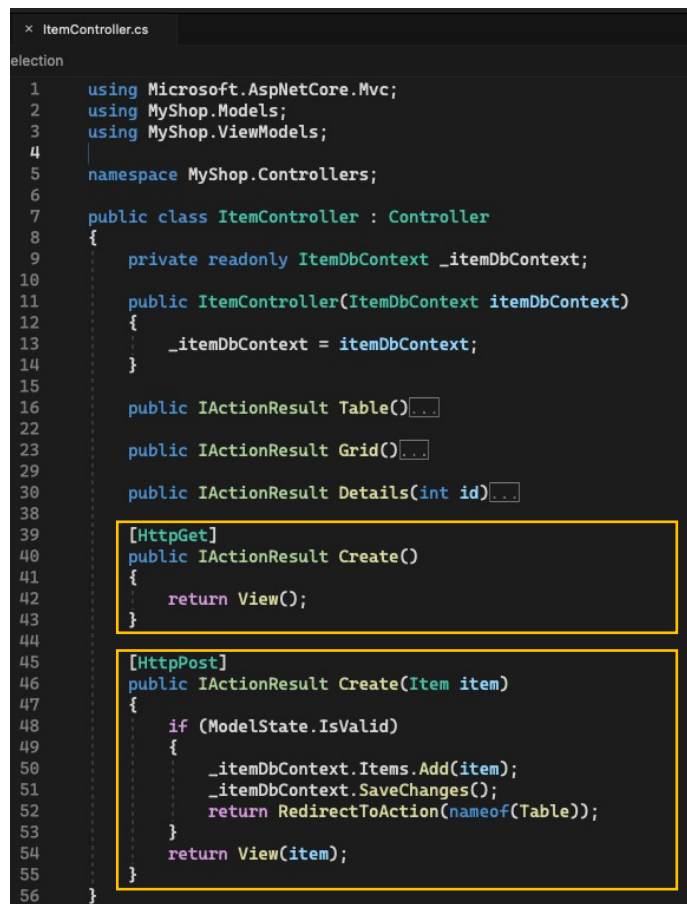
In this Demo, we will add mechanism of CRUD (Create, Read, Update, Delete) in our web application. Note that Read has been already finished before in the Action methods of Table(), Grid(), and Details():

```
ItemController.cs
1 using Microsoft.AspNetCore.Mvc;
2 using MyShop.Models;
3 using MyShop.ViewModels;
4
5 namespace MyShop.Controllers;
6
7 public class ItemController : Controller
8 {
9     private readonly ItemDbContext _itemDbContext;
10
11     public ItemController(ItemDbContext itemDbContext)
12     {
13         _itemDbContext = itemDbContext;
14     }
15
16     public IActionResult Table()
17     {
18         List<Item> items = _itemDbContext.Items.ToList();
19         var itemListViewModel = new ItemListViewModel(items, "Table");
20         return View(itemListViewModel);
21     }
22
23     public IActionResult Grid()
24     {
25         List<Item> items = _itemDbContext.Items.ToList();
26         var itemListViewModel = new ItemListViewModel(items, "Grid");
27         return View(itemListViewModel);
28     }
29
30     public IActionResult Details(int id)
31     {
32         List<Item> items = _itemDbContext.Items.ToList();
33         var item = items.FirstOrDefault(i => i.ItemId == id);
34         if (item == null)
35             return NotFound();
36         return View(item);
37     }
38 }
```

We first add the mechanism to create new data entry. We start by adding a “Create” button under the table (in Views/Shared/_ItemTable.cshtml): Line22 – Line25

```
_ItemTable.cshtml
1 @model IEnumerable<Item>
2
3 <div class="container">
4     <table class="table table-striped">
5         <tr><th>Id</th><th>Name</th><th>Price</th><th>Description</th></tr>
6         @foreach (var item in Model)
7         {
8             <tr>
9                 <td>@item.ItemId</td>
10                <td>
11                    <a asp-controller="Item"
12                      asp-action="Details"
13                      asp-route-id="@item.ItemId">@item.Name</a>
14                </td>
15                <td>@item.Price.ToString("0.00 NOK")</td>
16                <td>@item.Description</td>
17            </tr>
18        }
19    </table>
20    <p>
21        <a class="btn btn-secondary"
22          asp-controller="Item"
23          asp-action="Create">Create New Item</a>
24    </p>
25 </div>
```

To make the button usable, we add two Action methods to Controllers/ItemController.cs: (Line39 – Line54).



```
1 using Microsoft.AspNetCore.Mvc;
2 using MyShop.Models;
3 using MyShop.ViewModels;
4
5 namespace MyShop.Controllers;
6
7 public class ItemController : Controller
8 {
9     private readonly ItemDbContext _itemDbContext;
10
11     public ItemController(ItemDbContext itemDbContext)
12     {
13         _itemDbContext = itemDbContext;
14     }
15
16     public IActionResult Table()...
17
18     public IActionResult Grid()...
19
20     public IActionResult Details(int id)...
```

```
39 [HttpGet]
40 public IActionResult Create()
41 {
42     return View();
43 }
44
45 [HttpPost]
46 public IActionResult Create(Item item)
47 {
48     if (ModelState.IsValid)
49     {
50         _itemDbContext.Items.Add(item);
51         _itemDbContext.SaveChanges();
52         return RedirectToAction(nameof(Table));
53     }
54     return View(item);
55 }
56 }
```

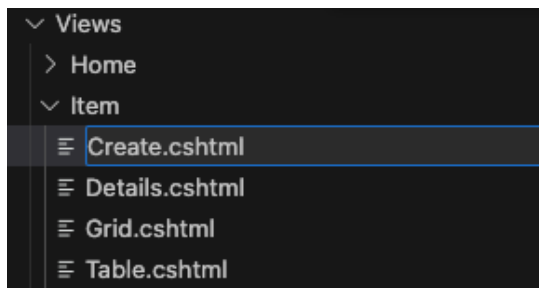
Note: Line39 – Line43 is a GET method that is used to display the initial form for creating a new item.

- When a user navigates to the "Create" page, this method is invoked.
- It returns a view (the "Create" view) that contains a form where the user can enter details for creating the new item.
- The [HttpGet] attribute is used to decorate a controller action (method) to handle HTTP GET requests. GET requests are used to retrieve data from the server.

Line 45 – Line54 is a POST method that is used to handle the submission of the form when the user clicks the "Create" button.

- It takes an Item object as a parameter, which is automatically bound to the form data submitted by the user in the View Create (later).
- Inside this method, it checks if the model state is valid (i.e., if the form data passed validation rules).
- If the model state is valid, it means the user has entered valid data, so the item is added to the database using the _itemDbContext and the changes are saved using _itemDbContext.SaveChanges().
- After successful item creation, it redirects the user to the "Table" view to show all items in the table.
- The [HttpPost] attribute is used to decorate a controller action (method) to handle HTTP POST requests. POST requests are used to submit data to the server for processing.

Then we create a View that allows the user to create new items. Create a file under Views/Item named as “Create.cshhtml”



We want to create a Create View with a form like this:

A screenshot of a web browser window. The address bar shows 'https://localhost:7205/Item/Cr...'. The page has a navigation bar with 'My Shop', 'Home', and 'Items'. The main content area is titled 'Create New Item' and contains a form with four input fields: 'Name*' (with a red asterisk), 'Price*' (with a red asterisk), 'Description', and 'ImageUrl'. Below the fields are two buttons: a blue 'Create' button and a grey 'Back to Table View' button.

Modify the Create.cshhtml file to this:

```
• Create.cshhtml
1  @model Item
2
3  <h2>Create New Item</h2>
4
5  <form asp-action="Create">                                Allow input of
6      <div class="form-group">                                Name
7          <label asp-for="Name"></label><span class="text-danger">*</span>
8          <input asp-for="Name" class="form-control" />
9          <span asp-validation-for="Name" class="text-danger"></span>
10     </div>
11     <div class="form-group">
12         <label asp-for="Price"></label><span class="text-danger">*</span>
13         <input asp-for="Price" class="form-control" />          The red asterisk
14         <span asp-validation-for="Price" class="text-danger"></span>
15     </div>
16     <div class="form-group">
17         <label asp-for="Description"></label>
18         <input asp-for="Description" class="form-control" />
19         <span asp-validation-for="Description" class="text-danger"></span>
20     </div>
21     <div class="form-group">
22         <label asp-for="ImageUrl"></label>
23         <input asp-for="ImageUrl" class="form-control" />
24         <span asp-validation-for="ImageUrl" class="text-danger"></span>
25     </div>
26     <button type="submit" class="btn btn-primary">Create</button>    The Create button
27     <a asp-action="Table" class="btn btn-secondary">Back to Table View</a>
28 </form>
```

Note: The asp-for attribute (e.g., Line7) is a tag helper in ASP.NET Core that helps with model binding and validation. It is used to associate a form input element with a specific property of the model class.

Line9 and Line14 verify that users' inputs against the Item Model, defined by Models/Item.cs.

The Name and Price are not nullable, therefore they are being validated for empty values. The form does not have a form field for ItemId, because it is an auto-generated property with a unique value generated by the database. In most cases, the ItemId is managed by the database itself and does not need to be entered by the user when creating a new item.

Recall from Item.cs:

```
× Item.cs
election
1  using System;
2  namespace MyShop.Models
3  {
4      public class Item
5      {
6          public int ItemId { get; set; }
7          public string Name { get; set; } = string.Empty;
8          public decimal Price { get; set; }
9          public string? Description { get; set; }
10         public string? ImageUrl { get; set; }
11     }
12 }
```

Other properties like Description and ImageUrl are allowed to be empty since they are nullable (specified by string?).

In case you want to validate Description or ImageUrl as required fields, you can add the [Required] attribute (Line1 is needed) to them in the Item class definition. For example:

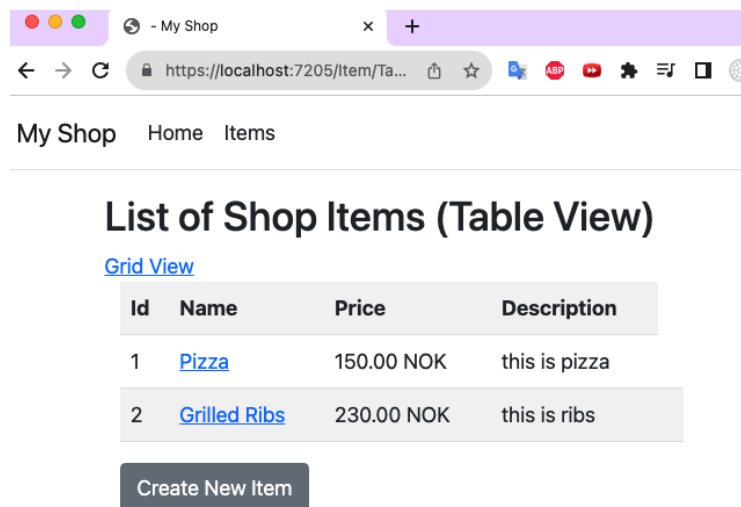
```

1 using System.ComponentModel.DataAnnotations;
2
3 namespace MyShop.Models
4 {
5     public class Item
6     {
7         public int ItemId { get; set; }
8         public string Name { get; set; } = string.Empty;
9         public decimal Price { get; set; }
10        [Required]
11        public string? Description { get; set; }
12        [Required]
13        public string? ImageUrl { get; set; }
14    }
15 }

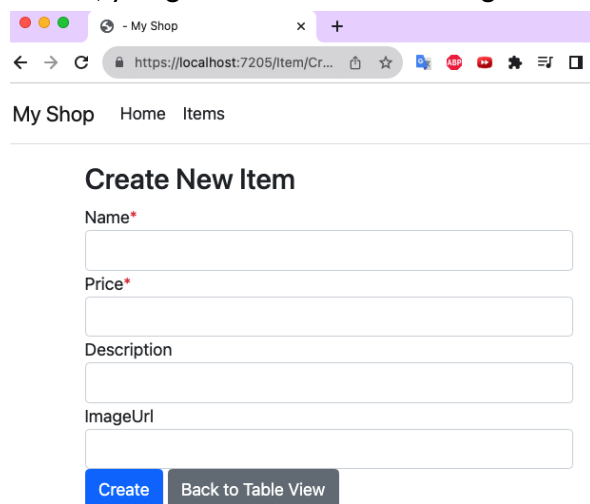
```

In the following we will not require Description and ImageUrl.

Now run the project in debugging mode (F5), you will see a “Create New Item” button appear under the table.



Click it, you go to the view of creating a new item



The Name and Price are validated against the Models/Item.cs

My Shop

Home

Items

Create New Item

Name*

Coke

Price*

i don't known

The value 'i don't known' is not valid for Price.

Description

ImageUrl

/images/coke.jpg

Create

Back to Table View

Now we input some data for a new item “Coke” and click “Create”

My Shop

Home

Items

Create New Item

Name*

Coke

Price*

30

View site information

Description

ImageUrl

/images/coke.jpg

Create

Back to Table View

The new Item appears:

My Shop

Home

Items

List of Shop Items (Table View)

[Grid View](#)

Id	Name	Price	Description
1	Pizza	150.00 NOK	this is pizza
2	Grilled Ribs	230.00 NOK	this is ribs
3	Coke	30.00 NOK	

Create New Item

Update and Delete

Similarly, we add the mechanism of Update and Delete.

In Views/Shared/_ItemTable.cshtml, we add two extra columns "Image" and "Actions" (Line10, Line11).

We set the image to be a small size (Line24 – Line26) in the item table.

We also add two small buttons (with only text and links) in the item table (Line27 – Line30): Views/Shared/_ItemTable.cshtml




```
1 @model IEnumerable<Item>
2
3 <div class="container">
4     <table class="table table-striped">
5         <tr>
6             <th>Id</th>
7             <th>Name</th>
8             <th>Price</th>
9             <th>Description</th>
10            <th>Image</th>
11            <th>Actions</th>
12        </tr>
13        @foreach (var item in Model)
14        {
15            <tr>
16                <td>@item.ItemId</td>
17                <td>
18                    <a asp-controller="Item"
19                      asp-action="Details"
20                      asp-route-id="@item.ItemId">@item.Name</a>
21                </td>
22                <td>@item.Price.ToString("0.00 NOK")</td>
23                <td>@item.Description</td>
24                <td>
25                    
26                </td>
27                <td>
28                    <a asp-action="Update" asp-route-id="@item.ItemId">Update</a>
29                    <a asp-action="Delete" asp-route-id="@item.ItemId">Delete</a>
30                </td>
31            </tr>
32        }
33    </table>
34    <p>
35        <a class="btn btn-secondary"
36          asp-controller="Item"
37          asp-action="Create">Create New Item</a>
38    </p>
39 </div>
```

Which should look like this:

My Shop Home Items

List of Shop Items (Table View)

[Grid View](#)

Id	Name	Price	Description	Image	Actions
1	Pizza	150.00 NOK	this is pizza		Update Delete
2	Grilled Ribs	230.00 NOK	this is ribs		Update Delete
3	Coke	30.00 NOK			Update Delete

[Create New Item](#)

To make this work, we need some Action methods in the Controllers/ItemController.cs

```
39 [HttpGet]
40 public IActionResult Create()...
44
45 [HttpPost]
46 public IActionResult Create(Item item)...
56
57 [HttpGet]
58 public IActionResult Update(int id)
59 {
60     var item = _itemDbContext.Items.Find(id);
61     if (item == null)
62     {
63         return NotFound();
64     }
65     return View(item);
66 }
67
68 [HttpPost]
69 public IActionResult Update(Item item)
70 {
71     if (ModelState.IsValid)
72     {
73         _itemDbContext.Items.Update(item);
74         _itemDbContext.SaveChanges();
75         return RedirectToAction(nameof(Table));
76     }
77     return View(item);
78 }
79
80 [HttpGet]
81 public IActionResult Delete(int id)
82 {
83     var item = _itemDbContext.Items.Find(id);
84     if (item == null)
85     {
86         return NotFound();
87     }
88     return View(item);
89 }
90
91 [HttpPost]
92 public IActionResult DeleteConfirmed(int id)
93 {
94     var item = _itemDbContext.Items.Find(id);
95     if (item == null)
96     {
97         return NotFound();
98     }
99     _itemDbContext.Items.Remove(item);
100     _itemDbContext.SaveChanges();
101     return RedirectToAction(nameof(Table));
102 }
```

Note: Line57 – Line66 is a GET Action method that is used to display the form for updating an existing item. It has the [HttpGet] attribute that indicates that it handles HTTP GET requests.

- When a user navigates to the "Update" page, this method is invoked.
- It takes an id parameter to identify the item to be updated.
- It retrieves the item with the given id from the database using `_itemDbContext.Items.Find(id)`.
- If the item is found, it returns the view with the item data so that the user can edit it.
- If the item is not found (i.e., null), it returns a 404 Not Found response.

Line68 – Line78 is a POST Action method that handles HTTP POST requests to process the form submission for updating an item.

- It takes an item parameter representing the updated item data submitted through a form in Update View (not created yet).
- It checks if the model state is valid (i.e., if the submitted data passes the validation rules defined in the Item model class).
- If the model state is valid, it updates the item in the database using `_itemDbContext.Items.Update(item)` and saves the changes using `_itemDbContext.SaveChanges()`.

- It then redirects the user to the Table action (view) to display the updated item list.
- If the model state is not valid (i.e., the submitted data is invalid), it returns the same view with the validation error messages.

Line80 – Line89 is a Get Action method to display the confirmation page for deleting an item.

- It takes an id parameter to identify the item to be deleted.
- It retrieves the item with the given id from the database using `_itemDbContext.Items.Find(id)`.
- If the item is found, it returns the view with the item data for confirmation.
- If the item is not found (i.e., null), it returns a 404 Not Found response.

Line91 – Line102 is a Post Action method to perform the actual deletion of the item.

- It takes an id parameter representing the ID of the item to be deleted.
- It retrieves the item with the given id from the database using `_itemDbContext.Items.Find(id)`.
- If the item is found, it removes the item from the database using `_itemDbContext.Items.Remove(item)` and saves the changes using `_itemDbContext.SaveChanges()`.
- It then redirects the user to the Table action (view) to display the updated item list after the deletion.
- If the item is not found (i.e., null), it returns a 404 Not Found response.

Then we need to create an Update View and Delete View. Create a file under Views/Item named as "Update.csthml". Modify the Update.csthml file to this:
 Note: Line6 is important for binding the right item, to ensure that the update changes the correct item. Otherwise,

```

1  @model Item
2
3  <h2>Update Item</h2>
4
5  <form asp-action="Update">
6      <input type="hidden" asp-for="ItemId" />
7      <div class="form-group">
8          <label asp-for="Name"></label>
9          <input asp-for="Name" class="form-control" />
10         <span asp-validation-for="Name" class="text-danger"></span>
11     </div>
12     <div class="form-group">
13         <label asp-for="Price"></label>
14         <input asp-for="Price" class="form-control" />
15         <span asp-validation-for="Price" class="text-danger"></span>
16     </div>
17     <div class="form-group">
18         <label asp-for="Description"></label>
19         <input asp-for="Description" class="form-control" />
20         <span asp-validation-for="Description" class="text-danger"></span>
21     </div>
22     <div class="form-group">
23         <label asp-for="ImageUrl"></label>
24         <input asp-for="ImageUrl" class="form-control" />
25         <span asp-validation-for="ImageUrl" class="text-danger"></span>
26     </div>
27     <button type="submit" class="btn btn-primary">Save Changes</button>
28     <a asp-action="Table" class="btn btn-secondary">Cancel</a>
29 </form>
  
```

Annotations in the image:

- Line 6: `<input type="hidden" asp-for="ItemId" />` is annotated with "Hidden form for" and "id important for binding the right item".
- Line 9: `<input asp-for="Name" class="form-control" />` is annotated with "Allow input of Name".
- Line 27: `<button type="submit" class="btn btn-primary">Save Changes</button>` is annotated with "The Update".

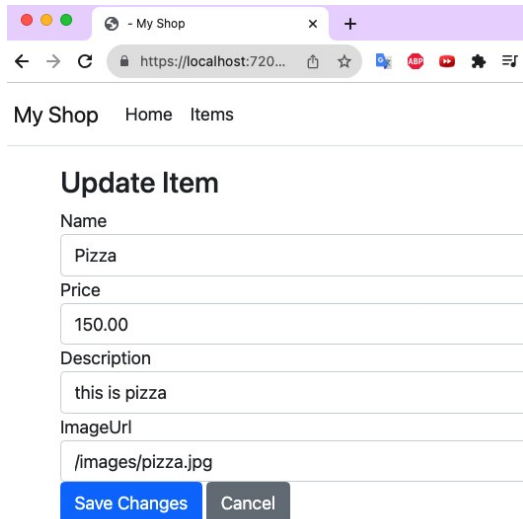
Note: Similarly, Line10 and Line15 verify that users' inputs against the Item Model, defined by Models/Item.cs.

Similarly, create a file under Views/Item named as "Delete.csthml. This is a confirmation page and does not have any forms. Modify it like this:

```

1  @model Item
2
3  <h2>Delete Item</h2>
4
5  <h3>Are you sure you want to delete this item?</h3>
6  <div>
7      <h4>Item</h4>
8      <hr />
9      <dl class="row">
10         <dt class="col-sm-2">Item Id</dt>
11         <dd class="col-sm-10">@Model.ItemId</dd>
12
13         <dt class="col-sm-2">Name</dt>
14         <dd class="col-sm-10">@Model.Name</dd>
15
16         <dt class="col-sm-2">Price</dt>
17         <dd class="col-sm-10">@Model.Price</dd>
18
19         <dt class="col-sm-2">Description</dt>
20         <dd class="col-sm-10">@Model.Description</dd>
21
22         <dt class="col-sm-2">Image URL</dt>
23         <dd class="col-sm-10">@Model.ImageUrl</dd>
24     </dl>
25
26     <form asp-action="DeleteConfirmed" asp-route-id="@Model.ItemId" method="post">
27         <button type="submit" class="btn btn-danger">Delete</button>
28         <a asp-action="Table" class="btn btn-secondary">Cancel</a>
29     </form>
30 </div>
  
```

Now run the project, click “Update” after any item, you will see the update form:



My Shop Home Items

Update Item

Name
Pizza

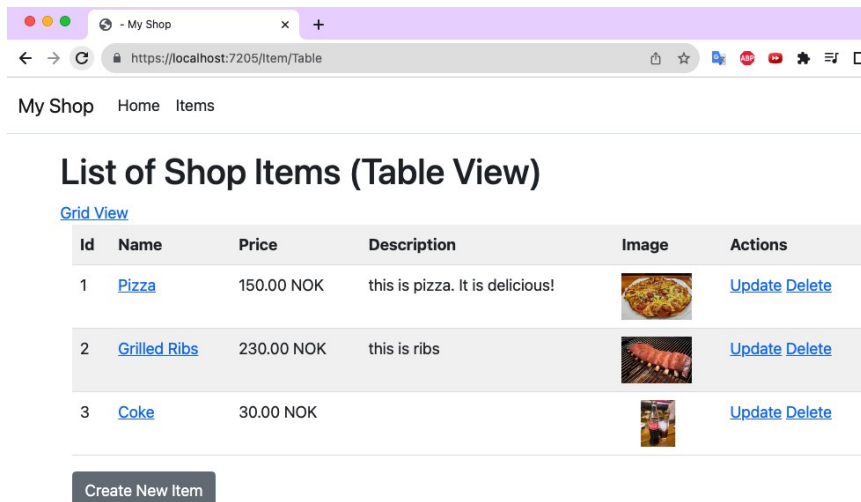
Price
150.00

Description
this is pizza

ImageUrl
/images/pizza.jpg

Save Changes Cancel




If you click “Cancel”, you will return to the Table View. If you make any change and click Save Changes, you will see the updated Table View.



My Shop Home Items

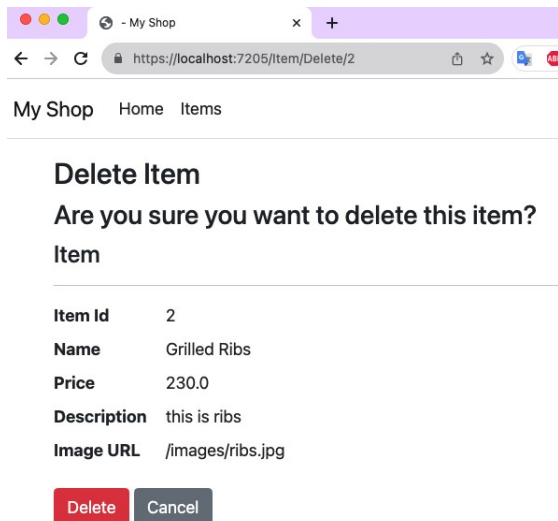
List of Shop Items (Table View)

[Grid View](#)

Id	Name	Price	Description	Image	Actions
1	Pizza	150.00 NOK	this is pizza. It is delicious!		Update Delete
2	Grilled Ribs	230.00 NOK	this is ribs		Update Delete
3	Coke	30.00 NOK			Update Delete

Create New Item

Click Delete of any item, you will go to the confirmation page:
The Name and Price are validated against the Models/Item.cs



My Shop Home Items

Delete Item

Are you sure you want to delete this item?

Item

Item Id 2

Name Grilled Ribs

Price 230.0

Description this is ribs

Image URL /images/ribs.jpg

Delete Cancel



If you click “Cancel”, you will return to the Table View. If you click Delete, you will see the updated Table View where the item has been deleted:

My Shop

HomeItems

List of Shop Items (Table View)

[Grid View](#)

Id	Name	Price	Description	Image	Actions
1	Pizza	150.00 NOK	this is pizza. It is delicious!		Update Delete
3	Coke	30.00 NOK			Update Delete

Create New Item

Congratulations! 😊 You have introduced the CRUD operations into your web application!