

## Table of Contents

Introduction.....	2
Add Grid View.....	3
Modify ItemController.....	5
ViewModel.....	7
Details View.....	10
Partial View: _ItemCard, _Carousel, _ItemTable.....	13

# Demo: Shop MVC 4, dynamic views

Author: Baifan Zhou

Some explanations are created with the help of ChatGPT.

## Introduction

This demo will walk you through the process of creating and refining your web application's views, controllers, and data management practices to build a more dynamic and maintainable project. You will improve your ASP.NET Core MVC application with advanced features like grid views, view models, and partial views. You will learn how to:

### 1. Add a Grid View:

- Create a new Razor View named `Grid.cshtml` and modify `Table.cshtml` to include a link to the grid view.
- Enhance the visual presentation of your items by adding item descriptions and improving the layout.

### 2. Modify ItemController:

- Update the `ItemController` to include actions for the table and grid views.
- Separate data retrieval into a `GetItems` method and enrich the item data.

### 3. ViewModel:

- Implement a view model to encapsulate all data needed for a view, following the Model-View-ViewModel (MVVM) pattern.
- Create an `ItemListViewModel` to streamline data passing between the controller and views.

### 4. Details View:

- Add a detailed view for individual items to provide more information to the users.
- Update existing views to include links to the details view, allowing users to navigate seamlessly.

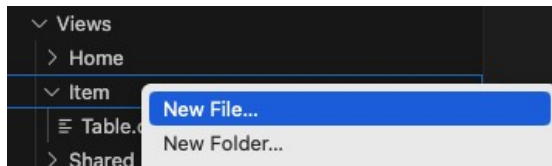
### 5. Partial View: `_ItemCard`, `_Carousel`, `_ItemTable`:

- Refactor repetitive code into partial views for better reusability and maintainability.
- Create partial views for item cards, a carousel, and item tables, and integrate them into your main views.

Happy coding!

## Add Grid View

Add a new Razor View with the name "Grid.cshtml" under the folder Item



Change the Grid.cshtml to this:

```
Grid.cshtml
Views > Item > Grid.cshtml
1 @model IEnumerable<Item>
2
3 <h1>List of Shop Items (@ViewBag.CurrentViewName View)</h1>
4 <a href="/Item/Table">Table View</a>
5
6 <div class="row row-cols-1 row-cols-md-3 g-4">
7     @foreach (var item in Model)
8     {
9         <div class="col">
10             <div>
11                 
12                 <div>
13                     <div class="d-flex justify-content-between mt-2">
14                         <h5 class="text-start">
15                             @item.Name
16                         </h5>
17                         <h5 class="text-nowrap">
18                             @item.Price.ToString("0.00 NOK")
19                         </h5>
20                     </div>
21                 </div>
22             </div>
23         </div>
24     }
25 </div>
```

### Detailed explanations

- Model Declaration:

@model IEnumerable<Item>: Specifies that the view expects a collection of Item objects.

- Header:

<h1>List of Shop Items (@ViewBag.CurrentViewName View)</h1>: Displays the title of the view, including a dynamic part from ViewBag.CurrentViewName.

- Navigation Link:

<a href="/Item/Table">Table View</a>: Provides a link to navigate to the table view of the items.

- Grid Layout:

<div class="row row-cols-1 row-cols-md-3 g-4">: Sets up a responsive grid with one column per row on small screens and three columns per row on medium and larger screens.

- Item Loop:

@foreach (var item in Model): Iterates over each Item in the model collection.

- Grid Columns:

<div class="col">: Represents a column in the grid for each item.

Inside each column:

Image:

: Displays the item's image.

- Item Details:

- `<div class="d-flex justify-content-between mt-2">`: Creates a flex container to align item name and price.
- `<h5 class="text-start">@item.Name</h5>`: Displays the item's name, aligned to the start (left).
- `<h5 class="text-nowrap">@item.Price.ToString("0.00 NOK")</h5>`: Displays the item's price, formatted as a currency with "NOK".

Note that we have created a link to the table view in Line4.

Accordingly, change the Table.cshtml to this:

```

1  @model IEnumerable<Item>
2
3  <h1>List of Shop Items (@ViewBag.CurrentViewName View)</h1>
4  <a href="/Item/Grid">Grid View</a>
5
6  <div class="container">
7      <table class='table table-striped'>
8          <tr><th>Id</th><th>Name</th><th>Price</th><th>Description</th></tr>
9          @foreach (var item in Model)
10             {
11                 <tr>
12                     <td>@item.ItemId</td>
13                     <td>@item.Name</td>
14                     <td>@item.Price.ToString("0.00 NOK")</td>
15                     <td>@item.Description</td>
16                 </tr>
17             }
18         </table>
19     </div>
20

```

For the grid to look better and to add Item Description, we need to add some data (next section).

## Modify ItemController

Modify the IActionResult for Table

Add the IActionResult for Grid

Separate the GetItems() from the IActionResult, and add more data into ItemController.cs:

(Complete code in the github repo)

```
1  using System.Linq;
2  using System.Threading.Tasks;
3  using Microsoft.AspNetCore.Mvc;
4  using MyShop.Models;
5
6  namespace MyShop.Controllers;
7
8  public class ItemController : Controller
9  {
10     public IActionResult Table()
11     {
12         var items = GetItems();
13         ViewBag.CurrentViewName = "Table";
14         return View(items);
15     }
16
17     public IActionResult Grid()
18     {
19         var items = GetItems();
20         ViewBag.CurrentViewName = "Grid";
21         return View(items);
22     }
23
24     public List<Item> GetItems()
25     {
26         var items = new List<Item>();
27         var item1 = new Item
28         {
29             ItemId = 1,
30             Name = "Pizza",
31             Price = 150,
32             Description = "Delicious Italian dish with a thin crust topped with tomato sauce, cheese,
33                 and various toppings.",
34             ImageUrl = "/images/pizza.jpg"
35         };
36
37         var item2 = new Item();
38
39         var item3 = new Item();
40
41         var item4 = new Item();
42
43         var item5 = new Item();
44
45         var item6 = new Item();
46
47         var item7 = new Item();
48
49         var item8 = new Item();
50
51         items.Add(item1);
52         items.Add(item2);
53         items.Add(item3);
54         items.Add(item4);
55         items.Add(item5);
56         items.Add(item6);
57         items.Add(item7);
58         items.Add(item8);
59         return items;
60     }
61 }
```

Run the project and you will see the modified Table View and the new Grid View

My Shop

Home

Items

List of Shop Items (Table View)

[Grid View](#)

Id	Name	Price	Description
1	Pizza	150.00 NOK	Delicious Italian dish with a thin crust topped with tomato sauce, cheese, and various toppings.
2	Fried Chicken Leg	20.00 NOK	Crispy and succulent chicken leg that is deep-fried to perfection, often served as a popular fast food item.
3	French Fries	50.00 NOK	Crispy, golden-brown potato slices seasoned with salt and often served as a popular side dish or snack.
4	Grilled Ribs	250.00 NOK	Tender and flavorful ribs grilled to perfection, usually served with barbecue sauce.
5	Tacos	150.00 NOK	Tortillas filled with various ingredients such as seasoned meat, vegetables, and salsa, folded into a delicious handheld meal.
6	Fish and Chips	180.00 NOK	Classic British dish featuring battered and deep-fried fish served with thick-cut fried potatoes.
7	Cider	50.00 NOK	Refreshing alcoholic beverage made from fermented apple juice, available in various flavors.
8	Coke	30.00 NOK	Popular carbonated soft drink known for its sweet and refreshing taste.


My Shop

Home

Items


List of Shop Items (Grid View)

[Table View](#)




Pizza

150.00 NOK




Fried Chicken Leg

20.00 NOK




French Fries

50.00 NOK




Grilled Ribs

250.00 NOK




Tacos

150.00 NOK




Fish and Chips

180.00 NOK



Cider

50.00 NOK



Coke

30.00 NOK

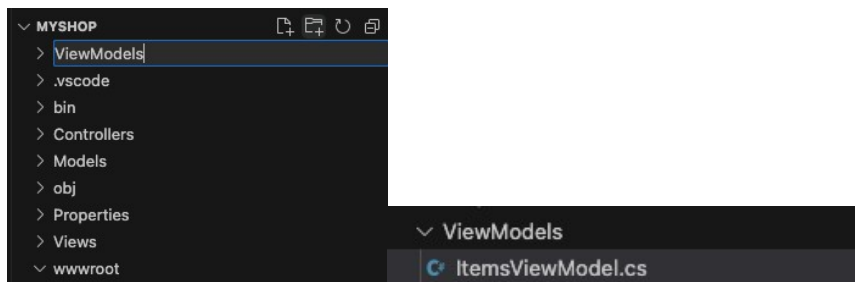
## ViewModel

To always have the items and CurrentViewName is not practical. As you can image, we may want to pass many data into the View.

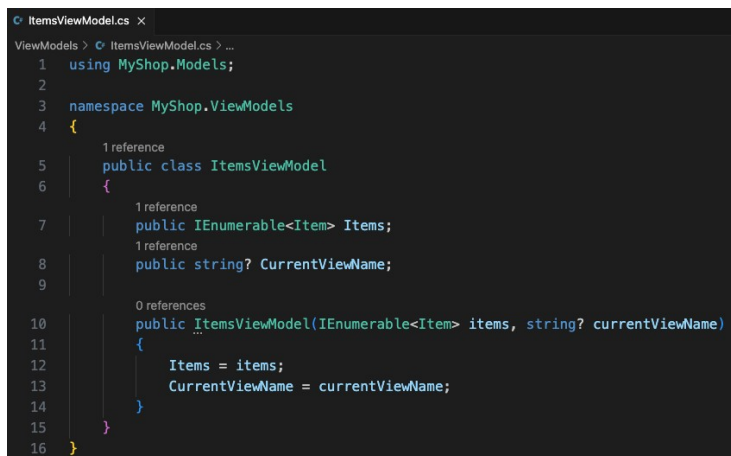
A good practice is to wrap all data needed in a View to a ViewModel

Note: this is sometimes called the Model-View-ViewModel pattern (MVVM)

Add new folder ViewModels under the upper folder



Add a new class ItemsViewModel.cs under the folder ViewModels and change the code to this:



Now we can wrap up the items and CurrentViewName into the ItemsViewModel. Go to ItemController.cs and change the code:

```

ItemController.cs x
Controllers > ItemController.cs > ItemController
1  using Microsoft.AspNetCore.Mvc;
2  using MyShop.Models;
3  using MyShop.ViewModels;
4
5  namespace MyShop.Controllers;
6
7  0 references
8  public class ItemController : Controller
9  {
10     0 references
11     public IActionResult Table()
12     {
13         var items = GetItems();
14         var itemsViewModel = new ItemsViewModel(items, "Table");
15         return View(itemsViewModel);
16     }
17
18     0 references
19     public IActionResult Grid()
20     {
21         var items = GetItems();
22         var itemsViewModel = new ItemsViewModel(items, "Grid");
23         return View(itemsViewModel);
24     }
25
26     //public IActionResult Table()
27     //{
28         //    var items = GetItems();
29         //    ViewBag.CurrentViewName = "Table";
30         //    return View(items);
31     //}
32
33     //public IActionResult Grid()
34     //{
35         //    var items = GetItems();
36         //    ViewBag.CurrentViewName = "Grid";
37         //    return View(items);
38     //}

```

Note that we need to add “using MyShop.ViewModels” (Line3) in the import part and comment out the previous Actions of Table and Grid.

Since we changed the model passed to the view, we need to change files under the folder Views.

First change the global import Views /\_ViewImports.cshtml by adding Line3

```

1  @using MyShop
2  @using MyShop.Models
3  @using MyShop.ViewModels
4  @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

```

Then change the Views/Item/Table.cshtml file:



```
Table.cshtml x Search MyShop - Table.cshtml - MyShop
Views > Item > Table.cshtml
1 @model ItemsViewModel
2
3 <h1>List of Shop Items (@Model.CurrentViewName View)</h1>
4 <a href="/Item/Grid">Grid View</a>
5
6 <div class="container">
7     <table class="table table-striped">
8         <tr><th>Id</th><th>Name</th><th>Price</th><th>Description</th></tr>
9         @foreach (var item in Model.Items)
10         {
11             <tr>
12                 <td>@item.ItemId</td>
13                 <td>@item.Name</td>
14                 <td>@item.Price.ToString("0.00 NOK")</td>
15                 <td>@item.Description</td>
16             </tr>
17         }
18     </table>
19
20 </div>
```

Similarly, change the Views/Item/Grid.cshtml file:

```
Grid.cshtml x
Views > Item > Grid.cshtml
1 @model ItemsViewModel
2
3 <h1>List of Shop Items (@Model.CurrentViewName View)</h1>
4 <a href="/Item/Table">Table View</a>
5
6 <div class="row row-cols-1 row-cols-md-3 g-4">
7     @foreach (var item in Model.Items)
8     {
9         <div class="col">
10             <div>
11                 
12                 <div>
13                     <div class="d-flex justify-content-between mt-2">
14                         <h5 class="text-start">
15                             @item.Name
16                         </h5>
17                         <h5 class="text-nowrap">
18                             @item.Price.ToString("0.00 NOK")
19                         </h5>
20                     </div>
21                 </div>
22             </div>
23         </div>
24     }
25 </div>
```

Now run the project, you should get the exactly same results.

## Details View

We now have the list and grid view of the items, but sometimes we also want to have a detailed look at the items.

Add a Razor View with the name "Details.cshtml" under the Views/Item folder

Change the Details.cshtml file to this:

```
Details.cshtml x
Views > Item > Details.cshtml
1  @model Item
2
3  <h3 class="my-5">
4      @Model.Name
5  </h3>
6
7  <div class="row gx-5">
8      
9      <div class="col-7">
10         <h5>@Model.Description</h5>
11         <h3 class="pull-right">@Model.Price.ToString("0.00 NOK")</h3>
12         <p><a href="/Item/Table">Back to Table View</a></p>
13         <p><a href="/Item/Grid">Back to Grid View</a></p>
14     </div>
15 </div>
```

Note that line 12 and line 13 create two links that point to the Table View and Grid View.

Each view requires an Action in the Controller.

Add the code (Line23 – Line30) into the Controllers/ItemController.cs:

```
9      public IActionResult Table()
10     {
11         var items = GetItems();
12         var itemsViewModel = new ItemsViewModel(items, "Table");
13         return View(itemsViewModel);
14     }
15
16     0 references
17     public IActionResult Grid()
18     {
19         var items = GetItems();
20         var itemsViewModel = new ItemsViewModel(items, "Grid");
21         return View(itemsViewModel);
22     }
23
24     0 references
25     public IActionResult Details(int id)
26     {
27         var items = GetItems();
28         var item = items.FirstOrDefault(i => i.ItemId == id);
29         if (item == null)
30             return NotFound();
31         return View(item);
32     }
```

Now we need to create links to the Detailed View. We can change the item names in the Grid.cshtml and Table.cshtml to links.

For this, we need to use the tag helpers (already added in the \_ViewImports.cshhtml Line4)

```
_ViewImports.cshhtml x
Views > _ViewImports.cshhtml
1  @using MyShop
2  @using MyShop.Models
3  @using MyShop.ViewModels
4  @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

For the Grid.cshtml, we change the @item.Name to a link: Line15 – Line17

```

Grid.cshtml x
Views > Item > Grid.cshtml
1  @model ItemsViewModel
2
3  <h1>List of Shop Items (@Model.CurrentViewName View)</h1>
4  <a href="/Item/Table">Table View</a>
5
6  <div class="row row-cols-1 row-cols-md-3 g-4">
7      @foreach (var item in Model.Items)
8      {
9          <div class="col">
10             <div>
11                 
12                 <div>
13                     <div class="d-flex justify-content-between mt-2">
14                         <h5 class="text-start">
15                             <a asp-controller="Item"
16                                 asp-action="Details"
17                                 asp-route-id="@item.ItemId">@item.Name</a>
18                         </h5>
19                         <h5 class="text-nowrap">
20                             @item.Price.ToString("0.00 NOK")
21                         </h5>
22                     </div>
23                 </div>
24             </div>
25         </div>
26     }
27 </div>

```

This code snippet generates a link (<a> tag) that will navigate to a specific action method within a controller in an ASP.NET Core MVC application. Here's what each attribute does:

- **<a> Tag:** Represents a hyperlink in HTML. The content inside the tag (@item.Name) will be the clickable text of the link.
- **asp-controller="Item":** Specifies the name of the controller that the link should target. In this case, it's the `Item` controller.
- **asp-action="Details":** Specifies the name of the action method within the specified controller that the link should target. Here, it's the `Details` action method.
- **asp-route-id="@item.ItemId":** Specifies a route parameter named `id` that should be passed to the action method. The value `@item.ItemId` is dynamically inserted, meaning it will use the `ItemId` property of the current `item` object in the loop.
- **@item.Name:** This is the content inside the <a> tag, which will be displayed as the clickable text of the link. It dynamically inserts the `Name` property of the current `item` object.

Remember that in our `Program.cs`, we have `app.MapDefaultControllerRoute()`; which is equal to `app.MapControllerRoute`

```

name: "default",
pattern: "{controller=Home}/{action=Index}/{id?}";

```

Here the `asp-controller`, `asp-action`, and `asp-route-id` together specify a link that points to an `Item` entry in the list `ItemsViewModel.Items`

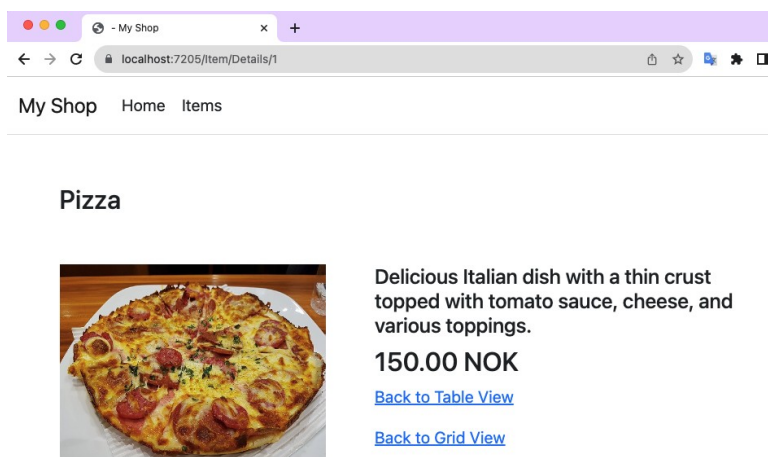
### How It Works Together

- **Controller:** When the link is clicked, the request is routed to the `ItemController`.
- **Action Method:** The `Details` action method of the `ItemController` will be invoked.
- **Route Parameter:** The `ItemId` of the current `item` object will be passed as a route parameter to the action method.

Similarly, we change the @item.Name in Table.cshtml to a link: Line14 – Line17

```
Table.cshtml x
Views > Item > Table.cshtml
1 @model ItemsViewModel
2
3 <h1>List of Shop Items (@Model.CurrentViewName View)</h1>
4 <a href="/Item/Grid">Grid View</a>
5
6 <div class="container">
7     <table class="table table-striped">
8         <tr><th>Id</th><th>Name</th><th>Price</th><th>Description</th></tr>
9         @foreach (var item in Model.Items)
10         {
11             <tr>
12                 <td>@item.ItemId</td>
13                 <td>
14                     <a asp-controller="Item"
15                       asp-action="Details"
16                       asp-route-id="@item.ItemId">@item.Name</a>
17                 </td>
18                 <td>@item.Price.ToString("0.00 NOK")</td>
19                 <td>@item.Description</td>
20                 <td></td>
21             </tr>
22         }
23     </table>
24 </div>
```

Run the project, you should be able to go to the Details View of each Item and go back to the Grid View or Table View.



## Partial View: \_ItemCard, \_Carousel, \_ItemTable

### Partial View in ASP.NET Core MVC

A partial view in ASP.NET Core MVC is a reusable component that encapsulates a portion of HTML markup and can be included in multiple views. Partial views help you avoid code duplication and promote a clean, maintainable codebase.

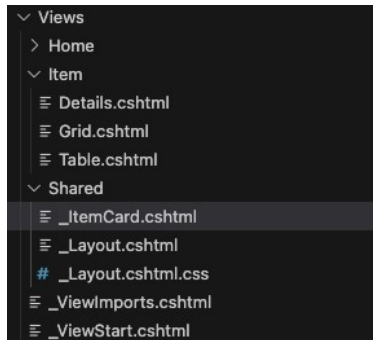
#### Why We Need Partial Views?

- **Reusability:** Partial views allow you to reuse common UI components across multiple views. For example, headers, footers, navigation menus, and item listings can be created as partial views and included wherever needed.
- **Maintainability:** By breaking down a complex view into smaller, manageable pieces, partial views make your code more organized and easier to maintain. Changes in the partial view will automatically reflect in all the views that include it.
- **Separation of Concerns:** Partial views promote the separation of concerns by allowing you to isolate and manage different parts of your UI separately. This improves code readability and makes it easier to track down issues.
- **Performance:** Partial views can be rendered independently and reused without reloading the entire page, which can improve the performance of your application, especially when dealing with dynamic or frequently updated content.

We can create a Partial View from Line9 -Line25 for the Grid view

```
Grid.cshtml x
Views > Item > Grid.cshtml
1  @model ItemsViewModel
2
3  <h1>List of Shop Items (@Model.CurrentViewName View)</h1>
4  <a href="/Item/Table">Table View</a>
5
6  <div class="row row-cols-1 row-cols-md-3 g-4">
7      @foreach (var item in Model.Items)
8      {
9          <div class="col">
10             <div>
11                 
12                 <div>
13                     <div class="d-flex justify-content-between mt-2">
14                         <h5 class="text-start">
15                             <a asp-controller="Item"
16                                 asp-action="Details"
17                                 asp-route-id="@item.ItemId">@item.Name</a>
18                         </h5>
19                         <h5 class="text-nowrap">
20                             @item.Price.ToString("0.00 NOK")
21                         </h5>
22                     </div>
23                 </div>
24             </div>
25         </div>
26     }
27 </div>
```

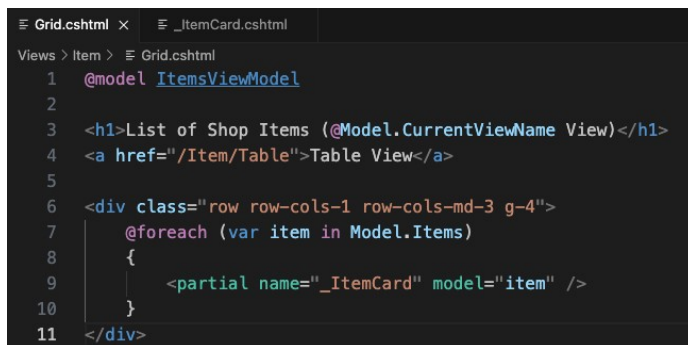
Add a new Razor View named “\_ItemCard” under Views/Shared



Cut the Line9 – Line25 into the \_ItemCard.cshtml, add the @model Item (Line1), replace all “@item” To “@Model”:



In Grid.cshtml, replace the original Line9 – Line25 with the Partial View code (Line9)



Similarly, we can move the Carousel (Line9 – Line26) in Index.cshtml under Views/Home/ into a partial view

```
Index.cshtml x
Views > Home > Index.cshtml
1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <div class="text-center">
6      <h1 class="display-4">Welcome to My Shop</h1>
7  </div>
8
9  <div id="carouselImages" class="carousel slide" data-bs-ride="true">
10     <div class="carousel-indicators">
11         <button type="button" data-bs-target="#carouselImages" data-bs-slide-to="0" class="active" aria-current="true"
12             aria-label="Slide 1"></button>
13         <button type="button" data-bs-target="#carouselImages" data-bs-slide-to="1" aria-label="Slide 2"></button>
14         <button type="button" data-bs-target="#carouselImages" data-bs-slide-to="2" aria-label="Slide 3"></button>
15     </div>
16     <div class="carousel-inner">
17         <div class="carousel-item">
18             
19         </div>
20         <div class="carousel-item active">
21             
22         </div>
23         <div class="carousel-item">
24             
25         </div>
26     </div>
27 </div>
```

Create a new Razor View named “\_Carousel.cshtml” under Views/Shared, and cut the Line9 – Line26 into the file \_Carousel.cshtml

Change the Index.cshtml by replacing with original Line9 – Line26 with the Line9

```
Index.cshtml x
Views > Home > Index.cshtml
1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <div class="text-center">
6      <h1 class="display-4">Welcome to My Shop</h1>
7  </div>
8
9  <partial name="_Carousel" />
```

Similarly, we move the Line6 – Line 24 in Table.cshtml into a partial view



```

Table.cshtml x
Views > Item > Table.cshtml
1 @model ItemsViewModel
2
3 <h1>List of Shop Items (@Model.CurrentViewName View)</h1>
4 <a href="/Item/Grid">Grid View</a>
5
6 <div class="container">
7     <table class='table table-striped'>
8         <tr><th>Id</th><th>Name</th><th>Price</th><th>Description</th></tr>
9         @foreach (var item in Model.Items)
10         {
11             <tr>
12                 <td>@item.ItemId</td>
13                 <td>
14                     <a asp-controller="Item"
15                        asp-action="Details"
16                        asp-route-id="@item.ItemId">@item.Name</a>
17                 </td>
18                 <td>@item.Price.ToString("0.00 NOK")</td>
19                 <td>@item.Description</td>
20             </tr>
21         }
22     </table>
23 </div>
24

```

Create a Razor View with the name “\_ItemTable.cshtml” under Views/Shared, and cut the Line6 – Line24 from Table.cshtml into the file \_ItemTable.cshtml

```

Shared
  _Carousel.cshtml
  _ItemCard.cshtml
  _ItemTable.cshtml
  _Layout.cshtml
  # _Layout.cshtml.css

```

Change the add the @model IEnumerable<Item> (Line1), replace “Model.Items” To “Model” (Line6):

```

_ItemTable.cshtml x
Views > Shared > _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 </div>
21

```

Change the Table.cshtml under Views/Item by replacing the original Line6 – Line24 with Line6:



```
Table.cshtml x
Views > Item > Table.cshtml
1 @model ItemsViewModel
2
3 <h1>List of Shop Items (@Model.CurrentViewName View)</h1>
4 <a href="/Item/Grid">Grid View</a>
5
6 <partial name="_ItemTable" model="Model.Items" />
```

Run the project, you should see the exact same results as before.

Congratulations! You have completed the MVC module of the course! 😊