Table of Contents

Demo: Shop MVC

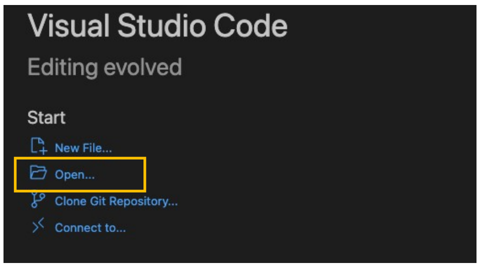
Author: Baifan Zhou
Detailed explanations are created with the help of ChatGPT.

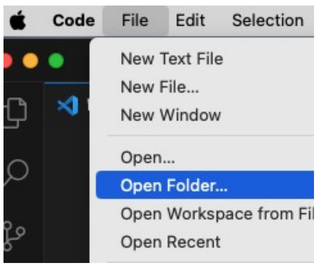
Introduction

In this demo, we are starting to create a new web project named as “MyShop” from scratch. MyShop will be a project for shop item management, including displaying shop items, and the CRUD (Create, Read, Update, Delete) operations on the shop items.
We will create the basic MVC, namely Models, Views, and Controllers for MyShop.

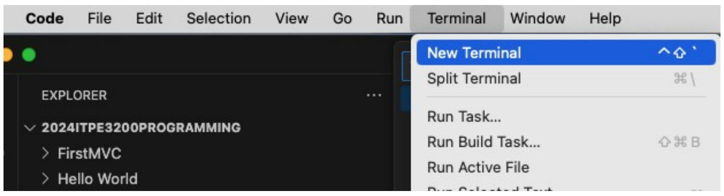
Creating a new empty web project

Start with creating a new empty web project.
Click `Open...` or `Open Folder...` in VS Code, select a folder where you would like to put the MyShop project, namely the parent folder of MyShop project.



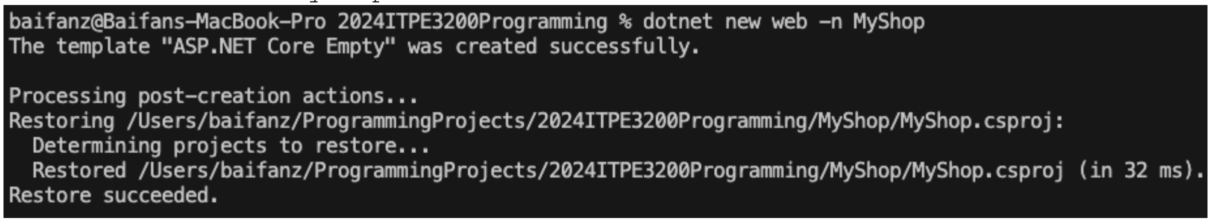


Open a terminal via menu bar or short cut.



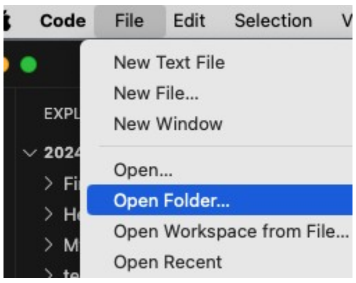
Run the following command to create a new web project (with a new folder) named as MyShop.

```
dotnet new web -n MyShop
```

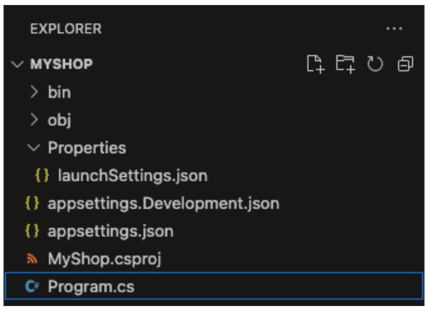


```
baifanz@Baifans-MacBook-Pro 2024ITPE3200Programming % dotnet new web -n MyShop
The template "ASP.NET Core Empty" was created successfully.

Processing post-creation actions...
Restoring /Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/MyShop.csproj:
  Determining projects to restore...
  Restored /Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/MyShop.csproj (in 32 ms).
Restore succeeded.
```

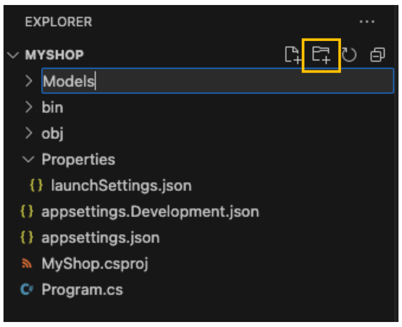
To select the folder MyShop as the top folder in VS Code Explorer.
Open a new window of VS Code by “Open Folder...”, and select the folder we just created with the command, MyShop



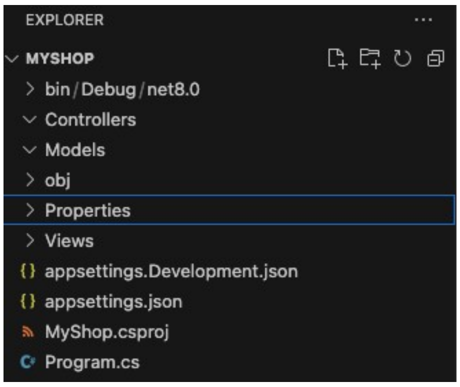
Now we have the new folder MyShop as the top folder, and the follow files in Explorer:



Add three folders (Models, Views, Controllers) to the project by clicking the marked icon. You need to expand MyShop by clicking the top MYSHOP, then you will see the icon.

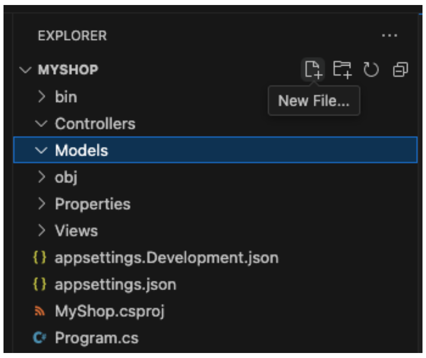


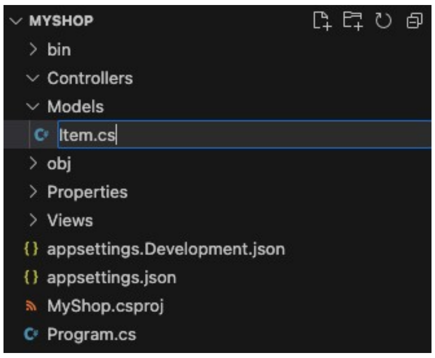
After adding the folders:



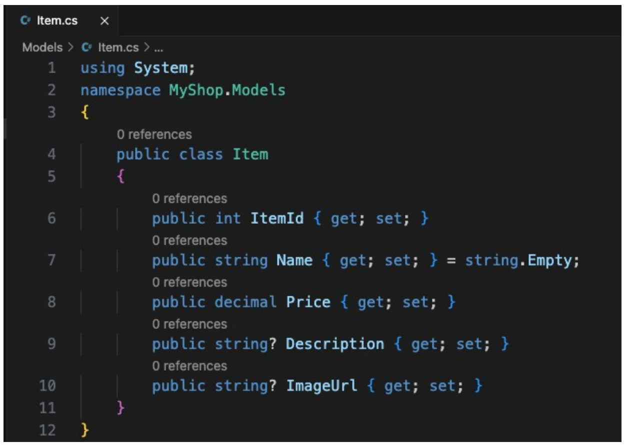
Model

Add a new class Item.cs to the folder Models





Change the class definition to this:



```
using System;
namespace MyShop.Models
{
    0 references
    public class Item
    {
        0 references
        public int ItemId { get; set; }
        0 references
        public string Name { get; set; } = string.Empty;
        0 references
        public decimal Price { get; set; }
        0 references
        public string? Description { get; set; }
        0 references
        public string? ImageUrl { get; set; }
    }
}
```

Note:
(Line6) The member variables must begin with upper case letters to follow C# convention.
(Line7) The strings and classes must be declared with default values (`string.Empty` or `default!` for class) to state the value is a mandatory value OR
(Line9) must be following with a question mark to state it is nullable.

Detailed explanation

1. using System;

- This line includes the System namespace, which provides basic classes and base classes that define commonly-used values and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.

2. namespace MyShop.Models

- This line defines a namespace called MyShop.Models.
- Namespaces are used to organize code into a hierarchical structure. This particular namespace suggests that the code belongs to a Models folder or component within the MyShop application.

3. public class Item

- This line declares a public class named Item.
- The public keyword makes this class accessible from other classes and components.
- A class is a blueprint for objects that contains fields, properties, methods, and other members.

4. public int ItemId { get; set; }

- This line declares a property named ItemId of type int (integer).
- public makes the property accessible from outside the class.
- { get; set; } defines an auto-implemented property with a public getter and setter, allowing the value to be read and written.

5. public string Name { get; set; } = string.Empty;

- This line declares a property named Name of type string.
- = string.Empty; initializes the Name property with an empty string by default. This ensures that Name is never null, preventing potential null reference issues.

6. public decimal Price { get; set; }

- This line declares a property named Price of type decimal.
- decimal is a data type suited for financial and monetary calculations due to its high precision.

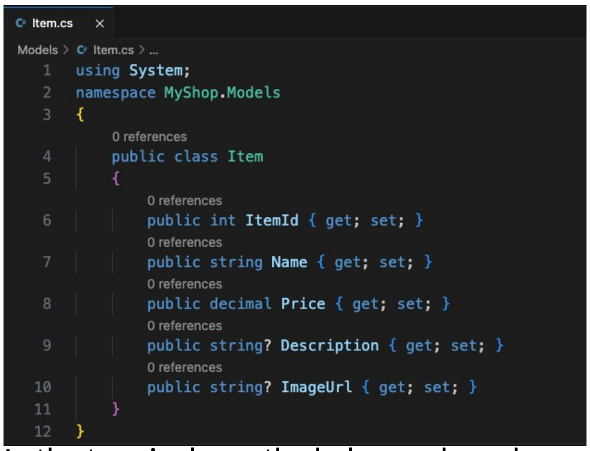
7. public string? Description { get; set; }

- This line declares a property named Description of type string?.
- The ? after string indicates that the Description property is nullable, meaning it can hold either a string value or null.

8. public string? ImageUrl { get; set; }

- This line declares a property named ImageUrl of type string?.
- Similar to Description, the ? indicates that ImageUrl is nullable.

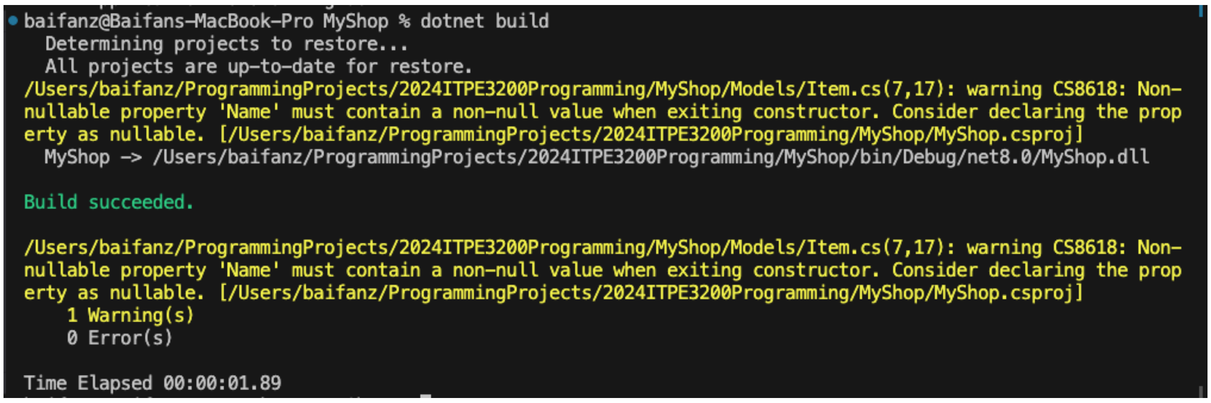
If you do not follow Line7, it will create a warning. We now remove the "= string.Empty;" and build the project.



```
using System;
namespace MyShop.Models
{
    public class Item
    {
        public int ItemId { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
        public string? Description { get; set; }
        public string? ImageUrl { get; set; }
    }
}
```

In the terminal, run the below code and you will see the errors and warnings.

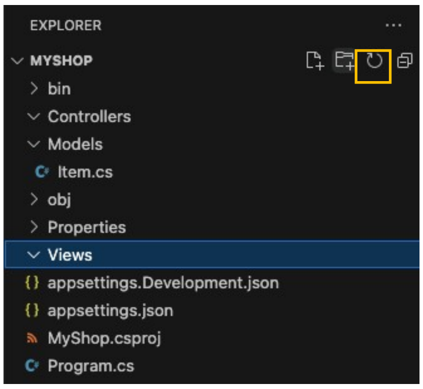
```
dotnet build
```

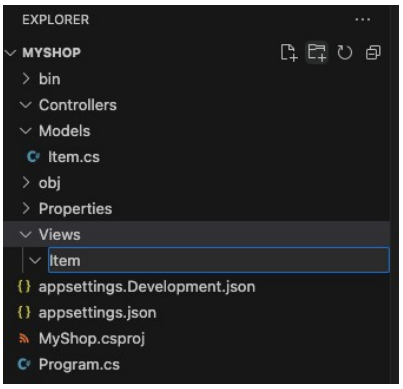


```
baifanz@Baifans-MacBook-Pro MyShop % dotnet build
  Determining projects to restore...
  All projects are up-to-date for restore.
/Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/Models/Item.cs(7,17): warning CS8618: Non-nullable property 'Name' must contain a non-null value when exiting constructor. Consider declaring the property as nullable. [/Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/MyShop.csproj]
  MyShop -> /Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/bin/Debug/net8.0/MyShop.dll

Build succeeded.

/Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/Models/Item.cs(7,17): warning CS8618: Non-nullable property 'Name' must contain a non-null value when exiting constructor. Consider declaring the property as nullable. [/Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/MyShop.csproj]
    1 Warning(s)
    0 Error(s)

Time Elapsed 00:00:01.89
```

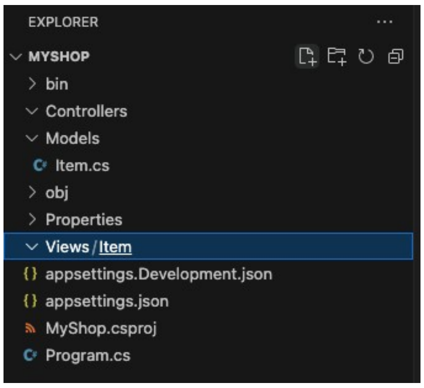
View

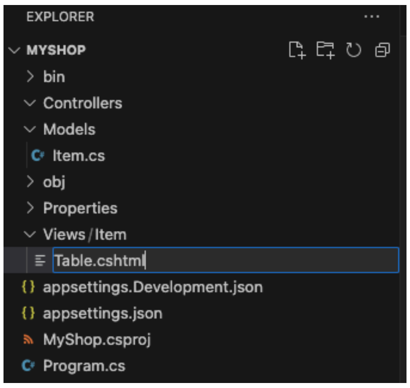
Add a subfolder “Item” in the folder Views. Select Views and click the icon.



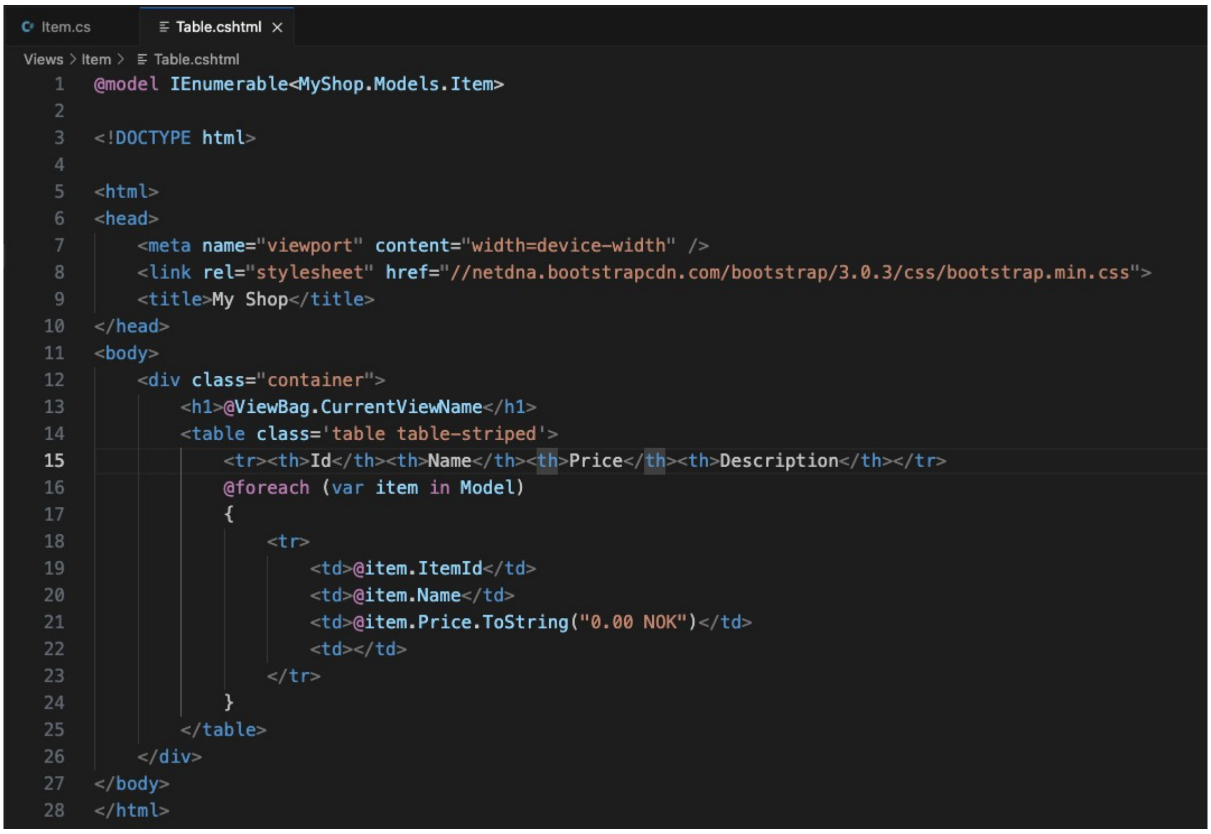


And add a new file “Table.cshtml in the Item subfolder





Change the Table.cshtml file like this:



```
@model IEnumerable<MyShop.Models.Item>

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css">
    <title>My Shop</title>
</head>
<body>
    <div class="container">
        <h1>@ViewBag.CurrentViewName</h1>
        <table class='table table-striped'>
            <tr><th>Id</th><th>Name</th><th>Price</th><th>Description</th></tr>
            @foreach (var item in Model)
            {
                <tr>
                    <td>@item.ItemId</td>
                    <td>@item.Name</td>
                    <td>@item.Price.ToString("0.00 NOK")</td>
                    <td></td>
                </tr>
            }
        </table>
    </div>
</body>
</html>
```

Detailed explanations

<h1>@ViewBag.CurrentViewName</h1>

- This line outputs a h1 header with the value of ViewBag.CurrentViewName.
- @ViewBag.CurrentViewName is a Razor syntax to embed server-side code in the view. ViewBag is a dynamic object used to pass data from the controller to the view.

<table class='table table-striped'>

- This line defines a table element with classes table and table-striped.
- These classes are typically from the Bootstrap CSS framework, where table applies basic table styling, and table-striped adds zebra-striping to table rows.

<tr><th>Id</th><th>Name</th><th>Price</th><th>Description</th></tr>

- This line creates a table row (tr) containing table headers (th) for Id, Name, Price, and Description.
- These headers label the columns of the table.

@foreach (var item in Model)

- This line starts a Razor foreach loop, which iterates over each item in the Model.
- Model is a dynamic object passed to the view from the controller, typically containing a collection of data (e.g., a list of items).

Iterating Over Items

- For each item in the Model, this block of code creates a new table row (tr).
- **<td>@item.ItemId</td>**: This creates a table cell (td) and displays the ItemId property of the current item.
- **<td>@item.Name</td>**: This creates a table cell and displays the Name property of the current item.
- **<td>@item.Price.ToString("0.00 NOK")</td>**: This creates a table cell and displays the Price property of the current item, formatted as a string with two decimal places followed by "NOK" (Norwegian Krone).
- **<td>@item.Description</td>**: This creates a table cell and displays the Description property of the current item.

What is a Razor View?

Definition: A Razor view is a file in an ASP.NET application that uses the Razor syntax to dynamically generate HTML content. It integrates C# code with HTML markup to create dynamic web pages.

File Extension: Razor views typically use the `.cshtml` file extension for C#-based views in ASP.NET Core MVC.

Syntax:

Razor Syntax: Uses `@` to switch between HTML and C# code. For example:

```
<h1>Hello, @ViewBag.CurrentViewName</h1>
```

Inline C# Code: Allows embedding C# code directly within HTML:

```
<td>@item.ItemId</d>
```

Dynamic Content:

Razor views enable dynamic content generation by integrating C# logic with HTML markup. This allows you to create views that can display data from models, perform conditional rendering, and more.

Model Binding:

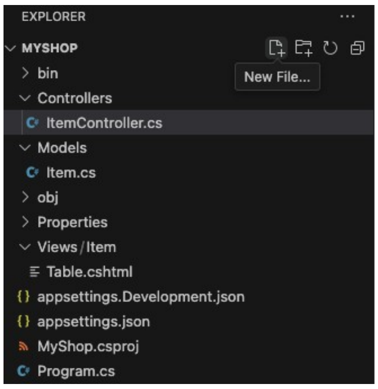
Razor views can receive data through models. Models represent the data and business logic that the view uses. The model is typically passed from a controller action to the view:

```
@model ItemsViewModel
```

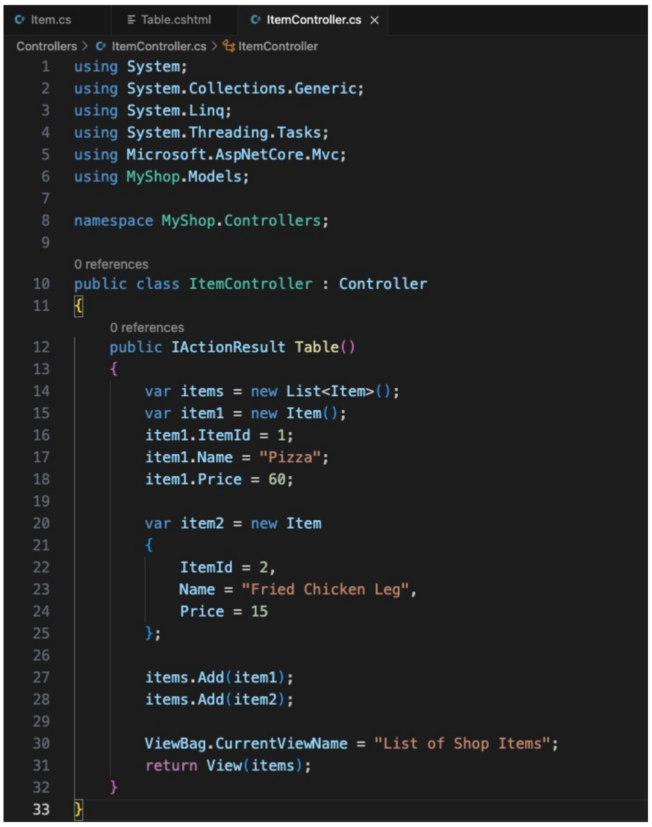
Tag Helpers:
Tag Helpers: Extend HTML elements with server-side functionality in a more declarative way. For example, you can use `<a asp-controller="Home" asp-action="Index">Home</a>` to create links that automatically generate URLs based on routing.
View Engine:
The Razor view engine processes `.cshtml` files and converts them into HTML that is sent to the client’s browser. It performs the combination of C# and HTML, producing the final output.

Controller

Add a new class "ItemController.cs" in the folder Controllers



Change the ItemController.cs code to this:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using MyShop.Models;

namespace MyShop.Controllers;

0 references
public class ItemController : Controller
{
    0 references
    public IActionResult Table()
    {
        var items = new List<Item>();
        var item1 = new Item();
        item1.ItemId = 1;
        item1.Name = "Pizza";
        item1.Price = 60;

        var item2 = new Item
        {
            ItemId = 2,
            Name = "Fried Chicken Leg",
            Price = 15
        };

        items.Add(item1);
        items.Add(item2);

        ViewBag.CurrentViewName = "List of Shop Items";
        return View(items);
    }
}
```

Actions

An **Action** is a method within a Controller that handles a specific request. Each Action method typically corresponds to a user interaction, such as viewing a list of items, displaying details of an item, or submitting a form.

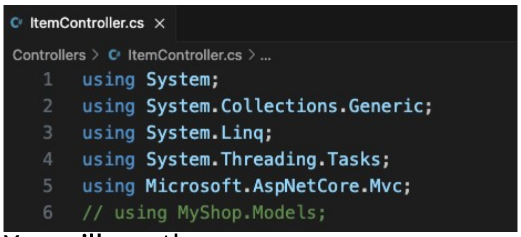
- **Return Types**: Action methods usually return an IActionResult, which could be a View, JSON, a Redirect, or other result types.

In this example, the Table action:

- Creates a list of items in a mock fashion.
- Sets a value in ViewBag to pass additional data to the View.
- Returns a View that will render the list of items.

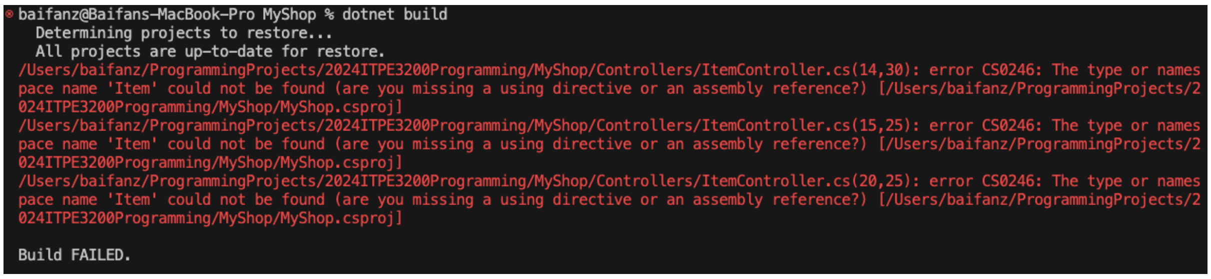
Note1:
If you did not have line 6, the Item Class will not be identified. Line 6 says we use data models from the folder “Models”.

We comment out Line 6 and build the project.



```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Mvc;
6  // using MyShop.Models;
```

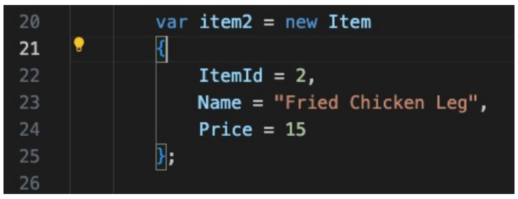
You will see the error message:



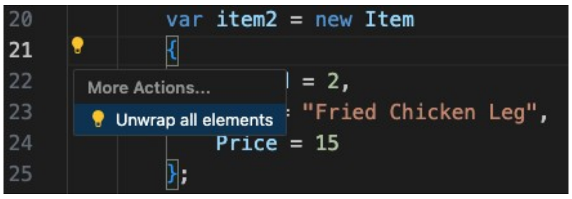
```
baifanz@Baifans-MacBook-Pro MyShop % dotnet build
  Determining projects to restore...
  All projects are up-to-date for restore.
/Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/Controllers/ItemController.cs(14,30): error CS0246: The type or names
pace name 'Item' could not be found (are you missing a using directive or an assembly reference?) [/Users/baifanz/ProgrammingProjects/2
024ITPE3200Programming/MyShop/MyShop.csproj]
/Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/Controllers/ItemController.cs(15,25): error CS0246: The type or names
pace name 'Item' could not be found (are you missing a using directive or an assembly reference?) [/Users/baifanz/ProgrammingProjects/2
024ITPE3200Programming/MyShop/MyShop.csproj]
/Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/MyShop/Controllers/ItemController.cs(20,25): error CS0246: The type or names
pace name 'Item' could not be found (are you missing a using directive or an assembly reference?) [/Users/baifanz/ProgrammingProjects/2
024ITPE3200Programming/MyShop/MyShop.csproj]

Build FAILED.
```

Note2:
Here we add two instances to the class Item in a mock fashion.
In the next lecture we will introduce ways to connect to the database.
We have two ways of adding instances to a class. Both are legal but the second way is preferred.

Sometimes, VS Code will have a small bulb. Click it will show actions for refactoring the code.



```
    var item2 = new Item
    {
        ItemId = 2,
        Name = "Fried Chicken Leg",
        Price = 15
    };
```

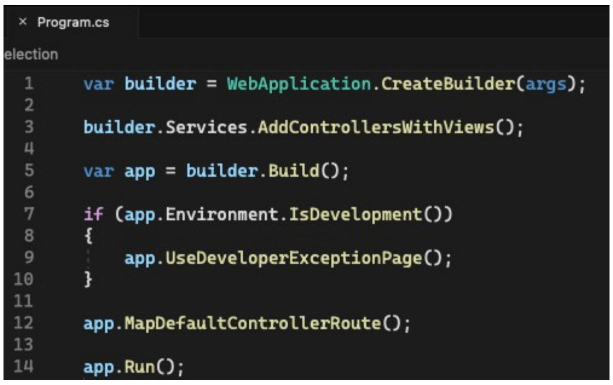


```
    var item2 = new Item
    {
            = 2,
            "Fried Chicken Leg",
        Price = 15
    };
```

Now we have all the basic files for Model-View-Controller.
We need to register the service of Controller with Views and add a ControllerRoute middle.

Program.cs

Change the Program.cs file like this:



```
1   var builder = WebApplication.CreateBuilder(args);
2
3   builder.Services.AddControllersWithViews();
4
5   var app = builder.Build();
6
7   if (app.Environment.IsDevelopment())
8   {
9       app.UseDeveloperExceptionPage();
10  }
11
12  app.MapDefaultControllerRoute();
13
14  app.Run();
```

(Line3)
This line adds services required for handling controllers and views to the dependency injection container. It sets up the application to use the MVC (Model-View-Controller) pattern for handling HTTP requests.

Services:
The term "services" refers to the components that are registered with the ASP.NET Core Dependency Injection (DI) container. Services are objects that provide specific functionality and can be used throughout the application by other components. Examples of services include database contexts, authentication services, logging, and more.

(Line7 – Line10)
This block checks whether the application is running in the development environment.
If yes, it adds the Developer Exception Page middleware to the application.
The Developer Exception Page is a helpful error page that is shown in the development environment when an unhandled exception occurs. It provides detailed information about the exception, which is useful for debugging.

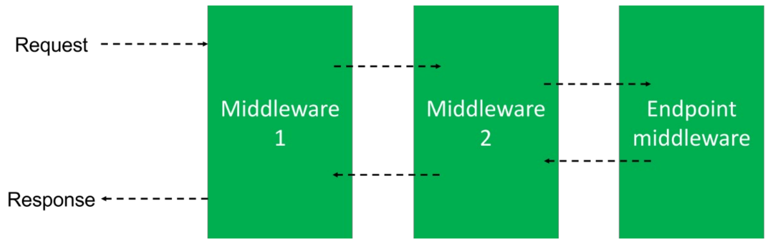
(Line12)
This line maps the default controller route for the application. It sets up a route that matches URLs to controller actions based on the default patterns. The default controller route patterns are "{controller=Home}/{action=Index}/{id?}. For example, a URL like "/Item/Table" would map to the List action in the ItemController. Line 12 is equal to the following code:

```
// app.MapControllerRoute(
//     name: "default",
//     pattern: "{controller=Home}/{action=Index}/{id?}");
```

Middlewares:
Middlewares in ASP.NET Core are components that sit in the request/response processing pipeline and can perform various operations on the incoming HTTP request or outgoing response. Each middleware has a specific role and can modify the request/response as it flows through the pipeline.

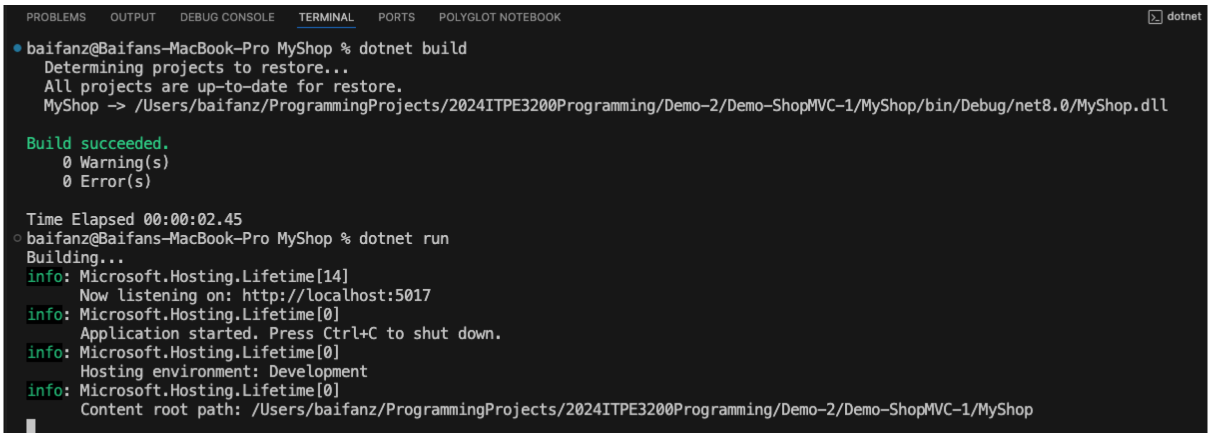
In the code, the app.Run(); method adds a middleware to the application. This middleware is the final piece in the pipeline and is responsible for handling the request and generating the response. It is executed when the pipeline reaches the end and no other middleware has handled the request.

The app.MapDefaultControllerRoute(); method also adds a middleware to the pipeline. This middleware handles the routing of incoming requests to the appropriate controller and action based on the defined routes.



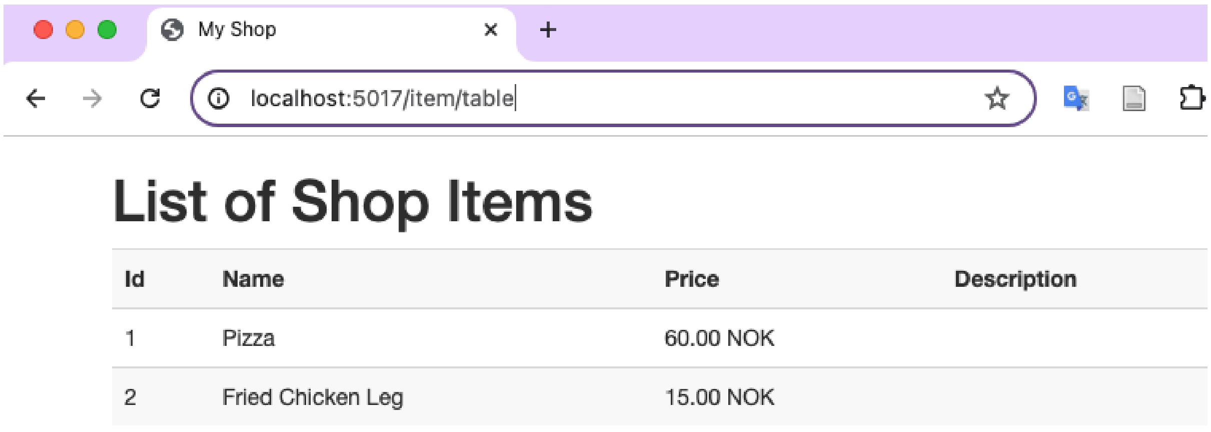
Build and Run

Now build and run the project and manually add “/item/table” to the end of the URL



```
baifanz@Baifans-MacBook-Pro MyShop % dotnet build
  Determining projects to restore...
  All projects are up-to-date for restore.
  MyShop -> /Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/Demo-2/Demo-ShopMVC-1/MyShop/bin/Debug/net8.0/MyShop.dll

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:02.45
baifanz@Baifans-MacBook-Pro MyShop % dotnet run
Building...
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5017
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /Users/baifanz/ProgrammingProjects/2024ITPE3200Programming/Demo-2/Demo-ShopMVC-1/MyShop
```

Note that the URL points to the default routing pattern: [controller]/[action]



List of Shop Items

Id	Name	Price	Description
1	Pizza	60.00 NOK	
2	Fried Chicken Leg	15.00 NOK	

Congratulations! ☺