

go deploy



Screenshots ▼

Module 4: Developing Controllers

Lab: Developing Controllers

? Scenario

You have been asked to add controllers to a new application. The controllers should include actions that return a view, also add an action that returns the photo as a .jpg file to show on a webpage, and an action that redirects to another action in another controller. Additionally, you have been asked to configure routes in a variety of ways.

The members of your development team are new to ASP.NET Core MVC and they find the use of controller actions confusing. Therefore, you need to help them by adding a component that displays action parameters in an external file whenever an action runs. To achieve this, you will add an action filter.

Exercise 1: Adding Controllers and Actions to an MVC Application

? Scenario

In this exercise, you will create the MVC controller that handles user operations. You will also add the following actions:

- Index. This action displays the Index view.
- Display. This action takes an ID to find a single City object. It passes the City object to the Display view.
- GetImage. This action returns the photo image from the service as a .jpg file.

The main tasks for this exercise are as follows:




- Add controllers to an MVC application
- Add actions to a controller
- Change actions to get a parameter
- Change an action to redirect to another action in another controller
- Use a service
- Store the result in a ViewBag property
- Run the application

Task 1: Add controllers to an MVC application

go deploy



⚠ Note: If a **Security Warning for WorldJourney** dialog box appears, verify that the **Ask me for every project in this solution** check box is cleared, and then click OK.

- ☐ 2. In Solution Explorer, right-click **WorldJourney**, point to **Add**, and then select **New Folder**.
- ☐ 3. In the **NewFolder** box, type  Controllers, and then press Enter.
- ☐ 4. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, right-click the **Controllers** folder, point to **Add**, and then select **Controller...**
- ☐ 5. In the **Add New Scaffolded Item** dialog box, click **MVC Controller - Empty**, and then click **Add**.
- ☐ 6. In the **Add New Item** dialog box, in the **Name:** textbox, type  HomeController, and then click **Add** (if this is the first controller, it will already be named for you).
- ☐ 7. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, right-click the **Controllers** folder, point to **Add**, and then select **Controller...**
- ☐ 8. In the **Add New Scaffolded Item** dialog box, click **MVC Controller - Empty**, and then click **Add**.
- ☐ 9. In the **Add New Item** dialog box, in the **Name:** textbox, type  CityController, and then click **Add**.

Task 2: Add actions to a controller


- ☐ 1. In the **CityController** class code block, in the **Index** action code block, locate the following code:

```
 return View();
```

- ☐ 2. Place the cursor before the located code, and type the following code:

```
 ViewData["Page"] = "Search city";
```

- ☐ 3. In the **CityController** code window, ensure that the cursor is at the end of the **Index** action code block, press Enter, and then type the following code:

```
 public IActionResult Details()  
{  
}
```

- ☐ 4. In the **Details** action code block, type the following code:

go deploy



```
if (city == null)
{
    return NotFound();
}

return View(city);
```

- ☐ 5. Verify that Visual Studio has automatically added a using statement for the **WorldJourney.Models** namespace as a result of you referencing the **City** type:

```
using WorldJourney.Models;
```

- ☐ 6. In the **CityController** code window, ensure that the cursor is at the end of the **Details** action code block, press Enter, and then type the following code:

```
public IActionResult GetImage()
{
}
```

- ☐ 7. In the **GetImage** action code block, type the following code:

```
ViewData["Message"] = "display Image";
City requestedCity = null;
if (requestedCity != null)
{
    string fullPath = "";
    FileStream fileOnDisk = new FileStream(fullPath, FileMode.Open);
    byte[] fileBytes;
    using (BinaryReader br = new BinaryReader(fileOnDisk))
    {
        fileBytes = br.ReadBytes((int)fileOnDisk.Length);
    }
    return File(fileBytes, requestedCity.ImageMimeType);
}
else
{
    return NotFound();
}
```

Task 3: Change actions to get a parameter

- ☐ 1. In the **CityController** class code block, select the following code:

```
public IActionResult Details()
```

- ☐ 2. Add a parameter by replacing the selected code with the following code:

go deploy



- ☐ 3. In the **CityController** class code block, select the following code:

```
public IActionResult GetImage()
```

- ☐ 4. Add a parameter by replacing the selected code with the following code.:

```
public IActionResult GetImage(int? cityId)
```

Task 4: Change an action to redirect to another action in another controller

- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, expand **Controllers**, and then click **HomeController.cs**.
- ☐ 2. In the **HomeController** code window, in the **Index** action code block, select the following code:

```
return View();
```

- ☐ 3. Replace the selected code with the following code:

```
return RedirectToAction("Index", "City");
```

Task 5: Use a service

- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, under **Controllers**, click **CityController.cs**.
- ☐ 2. In the **CityController** class code block, locate the following code:

```
public IActionResult Index()
```

- ☐ 3. Place the mouse cursor before the located code, type the following code, and then press Enter.


```
private IData _data;  
private IWebHostEnvironment _environment;  
  
public CityController(IData data, IWebHostEnvironment environment)  
{  
    _data = data;  
    _environment = environment;  
    _data.CityInitializeData();  
}
```

- ☐ 4. In the **Details** action code block, select the following code:


```
City city = null;
```

go deploy




 `City city = _data.GetCityById(id);`


- ☐ 6. In the **GetImage** action code block, select the following code:

 `City requestedCity = null;`


- ☐ 7. Replace the selected code with the following code:

 `City requestedCity = _data.GetCityById(cityId);`

- ☐ 8. In the **GetImage** action code block, select the following code:

 `string fullPath = "";`

- ☐ 9. Replace the selected code with the following code:


 `string webRootpath = _environment.WebRootPath;
string folderPath = "\\images\\";
string fullPath = webRootpath + folderPath + requestedCity.ImageName;`

Task 6: Store the result in a ViewBag property

- ☐ 1. In the **CityController** class code block, in the **Details** action code block, locate the following code:

 `return View(city);`

- ☐ 2. Place the mouse cursor before the located code, and type the following code:

 `ViewBag.Title = city.CityName;`

Task 7: Run the application

- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, on the **File** menu, click **Save All**.
- ☐ 2. In the **WorldJourney - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Without Debugging**.

 **Note:** The browser displays the **Index** action result inside the **City** controller.

- ☐ 3. In Microsoft Edge, on the **Earth** image, click the **London** area. Note the red arrow at the center of the **Earth** image (the map's hot area is not very accurate - you may find you need to click over Scotland or Ireland!).

go deploy



- ☐ 4. In Microsoft Edge, click **Close**.

✓ **Results:** After completing this exercise, you have created MVC controllers that implement common actions for the **City** model class in the application.

Exercise 2: Configuring Routes by Using the Routing Table


? Scenario


An important design priority for the application is that the visitors should be able to easily and logically locate cities. To implement these priorities, you have been asked to configure routes by using the routing table that enables the entry of user-friendly URLs to access cities.

The main tasks for this exercise are as follows:

- Add a controller with an action
- Run the application
- Register new routes in the routing table
- Run the application and verify that the new route works

Task 1: Add a controller with an action

- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, right-click the **Controllers** folder, point to **Add**, and then select **Controller....**
- ☐ 2. In the **Add New Scaffolded Item** dialog box, click **MVC controller - Empty**, and then click **Add**.
- ☐ 3. In the **Add New Item** dialog box, in the **Name:** textbox, type  TravelerController, and then click **Add**.
- ☐ 4. In the **TravelerController** class code block, select the following code:

```
 public IActionResult Index()  
{  
    return View();  
}
```


- ☐ 5. Replace the selected code with the following code:

go deploy



```
        ViewBag.VisitorName = name;
        return View();
    }
```

Task 2: Run the application


- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, on the **File** menu, click **Save All**.
- ☐ 2. In the **WorldJourney - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Without Debugging**.
- ☐ 3. In Microsoft Edge, in the address bar, type  [http://localhost:\[port\]/Traveler/Index](http://localhost:[port]/Traveler/Index), and then press Enter.

 **Note:** In the next task you will register a new route with the routing table. Then, you will not need to manually enter the **Traveler/Index** relative URL in the address bar.


- ☐ 4. In Microsoft Edge, click **Close**.

Task 3: Register new routes in the routing table

- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, click **Program.cs**.
- ☐ 2. Inside the **Program.cs** file, select the following code:

```
 app.MapDefaultControllerRoute();
```

- ☐ 3. Replace the selected code with the following code:

```
 app.MapControllerRoute(
    "TravelerRoute",
    "{controller}/{action}/{name}",
    new { controller = "Traveler", action = "Index", name = "Katie Bruce" },
    new { name = "[A-Za-z ]+" }
);
app.MapControllerRoute(
    "defaultRoute",
    "{controller}/{action}/{id?}",
    new { controller = "Home", action = "Index" },
    new { id = "[0-9]+" }
);
```

 **Note:** You can replace the default name **Katie Bruce** with your own name.

go deploy



- ☐ 2. In the **WorldJourney - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Without Debugging**.

Note: The name "**Katie Bruce**" shown in the title comes from the new "**TravelerRoute**" route, registered in the routing table.

- ☐ 3. In Microsoft Edge, click **Close**.

✓ **Results:** After completing this exercise, you will be able to register new custom routes in the request pipeline for controllers in the application.

Exercise 3: Configuring Routes by Using Attributes

? Scenario

In addition to configuring routes by using the routing table, you have been asked to configure routes by using attributes as well to enable the entry of user-friendly URLs. In this exercise you will use custom routes by means of attributes.

Task 1: Apply custom routes to a controller by using attributes

- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, under **Controllers**, click **CityController.cs**.
- ☐ 2. In the **Index** action code block, locate the following code:

```
public IActionResult Index()
```

- ☐ 3. Place the cursor before the located code, press Enter, and then type the following code:

```
[Route("WorldJourney")]
```

- ☐ 4. In the **Details** action code block, locate the following code:

```
public IActionResult Details(int? id)
```

- ☐ 5. Place the cursor before the located code, press Enter, and then type the following code:

```
[Route("CityDetails/{id?}")]
```


go deploy



- ☐ 2. In the **WorldJourney - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Without Debugging**.
- ☐ 3. In Microsoft Edge, right-click the page, and then select **View page source**.
- ☐ 4. Locate the line of HTML (near the end of the page):

```
<div class="button-center">
```

- ☐ 5. Find the anchor tag on the next line, and verify that the **href** attribute is **/WorldJourney**:

```
<a href="/WorldJourney">Go Next</a>
```

- ☐ 6. Close the page source tab.
- ☐ 7. In the normal Microsoft Edge view, click on the **Go Next** button, and verify that it takes you to the **WorldJourney** view.

Note: As a result of applying a custom route to **CityController** in the **Index** action by using attributes, **http://localhost:[port]/WorldJourney** should appear in the address bar.

- ☐ 8. In Microsoft Edge, right-click the page anywhere outside the large graphic, and then select **View page source**.
- ☐ 9. Find the **map** element on the page, which defines the hot areas on the large graphic.
- ☐ 10. Verify that the value of **href** attribute for the **London** hot area is **/CityDetails/2**.
- ☐ 11. Close the page source tab.
- ☐ 12. In Microsoft Edge, on the **Earth** image, click the **London** area (next to the red arrow at the center of the image), and verify that it takes you to the London details page.

Note: As a result of applying a custom route to a **CityController** in the **Details** action using attributes, **http://localhost:[port]/CityDetails/2** should appear in the address bar.

- ☐ 13. In Microsoft Edge, click **Close**.

✓ **Results:** After completing this exercise, you have added custom routes to the **City** controller by using the **Route** attribute.

go deploy





? Scenario

Your development team is new to ASP.NET Core MVC and is having difficulty in passing the right parameters to controllers and actions. You need to implement a component that displays the controller names and action names in an external file to help with this problem. In this exercise, you will create an action filter for this purpose.

The main tasks for this exercise are as follows:


- Add an action filter class
- Add a handler for the OnActionExecuting event
- Add a handler for the OnActionExecuted event
- Add a handler for the OnResultExecuted event
- Apply the action filter to the controller action
- Run the application and verify that the new filter works

Task 1: Add an action filter class

- ☐ 1. In Solution Explorer, right-click **WorldJourney**, point to **Add**, and then select **New Folder**.
- ☐ 2. In the **NewFolder** box, type  **Filters**, and then press Enter.
- ☐ 3. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, right-click **Filters**, point to **Add**, and then select **Class...**
- ☐ 4. In the **Add New Item -- WorldJourney** dialog box, in the **Name** box, type  **LogActionFilterAttribute**, and then click **Add**.
- ☐ 5. In **LogActionFilterAttribute** locate the following code:

```
 using System.Threading.Tasks;
```

- ☐ 6. Ensure that the cursor is at the end of the **using System.Threading.Tasks** namespace, press Enter, and then type the following code:

```
 using System.IO;  
using Microsoft.AspNetCore.Hosting;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.AspNetCore.Mvc.Filters;
```

- ☐ 7. In the **LogActionFilterAttribute** class code window, locate the following code:

```
 public class LogActionFilterAttribute
```

go deploy



```
public class LogActionFilterAttribute : ActionFilterAttribute
```

- ☐ 9. Verify that Visual Studio has automatically added a using statement for the **Microsoft.AspNetCore.Mvc.Filters** namespace.
- ☐ 10. In the **LogActionFilterAttribute** class code block, press Enter, and then type the following code:

```
private IWebHostEnvironment _environment;
private string _contentRootPath;
private string _logPath;
private string _fileName;
private string _fullPath;

public LogActionFilterAttribute(IWebHostEnvironment environment)
{
    _environment = environment;
    _contentRootPath = _environment.ContentRootPath;
    _logPath = _contentRootPath + "\\LogFile\\";
    _fileName = $"log {DateTime.Now.ToString("MM-dd-yyyy-H-mm")}.txt";
    _fullPath = _logPath + _fileName;
}
```

Task 2: Add a handler for the OnActionExecuting event

- ☐ 1. In the **LogActionFilterAttribute** class code block, ensure that the cursor is at the end of the **LogActionFilterAttribute** constructor code block, press Enter, and then type the following code:

```
public override void OnActionExecuting(ActionExecutingContext filterContext)
{
}
```

- ☐ 2. In the **OnActionExecuting** method code block, press Enter, insert the following code:

```
Directory.CreateDirectory(_logPath);
string actionName = filterContext.ActionDescriptor.RouteValues["action"];
string controllerName = filterContext.ActionDescriptor.RouteValues["controller"];
using (FileStream fs = new FileStream(_fullPath, FileMode.Create))
{
    using (StreamWriter sw = new StreamWriter(fs))
    {
        sw.WriteLine($"The action {actionName} in {controllerName} controller sta
    }
}
```

Task 3: Add a handler for the OnActionExecuted event

- ☐ 1. In the **LogActionFilterAttribute** class code block, ensure that the cursor is at the end of the **OnActionExecuting** method code block, press Enter, and then type the following code:

go deploy



}

- ☐ 2. In the **OnActionExecuted** method code block, press Enter, type the following code, and then press Enter.

```

string actionName = filterContext.ActionDescriptor.RouteValues["action"];
string controllerName = filterContext.ActionDescriptor.RouteValues["controller"];
using (FileStream fs = new FileStream(_fullPath, FileMode.Append))
{
    using (StreamWriter sw = new StreamWriter(fs))
    {
        sw.WriteLine($"The action {actionName} in {controllerName} controller fin
    }
}

```

Task 4: Add a handler for the OnResultExecuted event

- ☐ 1. In the **LogActionFilterAttribute** class code block, ensure that the cursor is at the end of the **OnActionExecuted** method code block, press Enter, and then type the following code:

```

public override void OnResultExecuted(ResultExecutedContext filterContext)
{
}

```

- ☐ 2. In the **OnResultExecuted** method code block, press Enter, type the following code, and then press Enter.

```

string actionName = filterContext.ActionDescriptor.RouteValues["action"];
string controllerName = filterContext.ActionDescriptor.RouteValues["controller"];
ViewResult result = (ViewResult)filterContext.Result;
using (FileStream fs = new FileStream(_fullPath, FileMode.Append))
{
    using (StreamWriter sw = new StreamWriter(fs))
    {
        sw.WriteLine($"The action {actionName} in {controllerName} controller has
    }
}

```

Task 5: Apply the action filter to the controller action

- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, click **Program.cs**.
- ☐ 2. In the **Program.cs** code window, locate the following code:

```

builder.Services.AddSingleton<IData, Data>();

```

- ☐ 3. Place the mouse cursor after the located code, and add the following:


go deploy



- ☐ 4. Verify that Visual Studio has automatically added a using statement for the **WorldJourney.Filters** namespace.
- ☐ 5. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, expand **Controllers**, and then click **CityController.cs**.
- ☐ 6. In the **CityController** class code block, locate the following code:

```
 [Route("WorldJourney")]
```

- ☐ 7. Place the mouse cursor before the located code, press Enter, and then type the following code:

```
 [ServiceFilter(typeof(LogActionFilterAttribute))]
```

Task 6: Run the application and verify the new filter works

- ☐ 1. In the **WorldJourney - Microsoft Visual Studio** window, on the **File** menu, click **Save All**.
- ☐ 2. In the **WorldJourney - Microsoft Visual Studio** window, on the **Debug** menu, click **Start Without Debugging**.
- ☐ 3. In Microsoft Edge, click on **Go Next**.
- ☐ 4. In Microsoft Edge, on the Earth image, click the **London** area (next to the red arrow).
- ☐ 5. Click **Go Back**.
- ☐ 6. In Microsoft Edge, click **Close**.
- ☐ 7. In the **WorldJourney - Microsoft Visual Studio** window, on the **File** menu, click **Exit**.
- ☐ 8. Navigate to **D:\Allfiles\Mod04\Labfiles\01_WorldJourney_begin\LogFile**. There should be one or more log files with names of the form **log [date] .txt**.
- ☐ 9. Open one of the log files to verify that controller action events have been logged.

 **Note:** The log files contain the output of the action filter.

✓ **Results:** After completing this exercise, you have created an action filter class that logs the details of actions, controllers, and parameters to an external log file whenever an action is called.