Lab: Developing Controllers

Lab Setup

Estimated Time: **60 minutes**

Preparation Steps

1. Navigate to **[Repository Root]\Allfiles\Mod04\LabFiles\01_WorldJourney_begin**, and then open the **WorldJourney.sln**.

   > **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 the **WorldJourney - Microsoft Visual Studio** window, on the **DEBUG** menu, click **Start Without Debugging**.

3. In Microsoft Edge, in the address bar, note the port number that appears at the end of the URL **http://localhost:[port]**. You will use the port number during this lab.

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

Exercise 1: Adding Controllers and Actions to an MVC Application

Task 1: Add controllers to an MVC application

1. Navigate to **[Repository Root]\Allfiles\Mod04\Labfiles\01_WorldJourney_begin** and double-click **WorldJourney.sln**.

   > **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 Scaffold** dialog box, click **MVC Controller - Empty**, and then click **Add**.

6. In the **Add Empty MVC Controller** dialog box, in the **Controller name** box, type **HomeController**, and then click **Add**.

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 Scaffold** dialog box, click **MVC Controller - Empty**, and then click **Add**.

9. In the **Add Empty MVC Controller** dialog box, in the **Controller name** box, type **CityController**, and then click **Add**.

Task 2: Add actions to a controller

1. In the **CityController.cs** code window, locate the following code:

```
using Microsoft.AspNetCore.Mvc;
```

2. Ensure that the cursor is at the end of the **Microsoft.AspNetCore.Mvc** namespace, press Enter, and then type the following code:

```
using System.IO;
using Microsoft.AspNetCore.Hosting;
using WorldJourney.Models;
```

3. In the **CityController** class code block, in the **Index** action code block, locate the following code:

```
return View();
```

4. Place the cursor before the located code, and type the following code:

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

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

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

6. In the **Details** action code block, type the following code:

```
ViewData["Page"] = "Selected city";
City city = null;
if (city == null)
{
     return NotFound();
}

return View(city);
```

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

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

8. 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. Replace the selected code with the following code:

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

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

```
public IActionResult GetImage()
```

4. Replace 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 IHostingEnvironment _environment;

public CityController(IData data, IHostingEnvironment environment)
{
     _data = data;
```

```
        _environment = environment;
        _data.CityInitializeData();
    }
```

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

```
City city = null;
```

5. Replace the selected code with the following code:

```
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.

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

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

Results: After completing this exercise, you will be able to create MVC controllers that implement common actions for the **City** model class in the application.

Exercise 2: Configuring Routes by Using the Routing Table

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 Scaffold** dialog box, click **MVC controller - Empty**, and then click **Add**.

3. In the **Add Empty MVC Controller** dialog box, in the **Controller name** box, 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:

```
public IActionResult Index(string name)
{
   ViewBag.VisiterName = 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**, 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 **Startup.cs**.
2. Inside the **Configure** method code block, in the **Startup** class, select the following code:

```
     app.UseMvcWithDefaultRoute();
```

3. Replace the selected code with the following code:

```
     app.UseMvc(routes =>
          {
              routes.MapRoute(
                   name: "TravelerRoute",
                   template: "{controller}/{action}/{name}",
                   constraints: new { name = "[A-Za-z ]+" },
                   defaults: new { controller = "Traveler", action = "Index", name =
"Katie Bruce" });

              routes.MapRoute(
                  name: "defaultRoute",
                  template: "{controller}/{action}/{id?}",
                  defaults: new { controller = "Home", action = "Index" },
                  constraints: new { id = "[0-9]+" });
          });
```

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

Task 4: Run the application and verify the new route 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**.

   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

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?}")]
```

Task 2: Run the application and verify the new routes work

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, right-click the page, and then select **View source**.

4. In **Developer Tools**, click **Elements**.

5. Press Ctrl + B.

6. Place the cursor over the **Go Next** button, and then click.

   Note: In **Developer Tools**, in the **a** tag, verify that the **href** attribute is **/WorldJourney**.

7. In **Developer Tools**, click **Close**.

8. Click **Go Next**.

   Note: Verify that the new route works. 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.

9. In Microsoft Edge, right-click the page, and then select **View source**.

10. In **Developer Tools**, click **Elements**.

11. Press Ctrl + B.

12. Put the cursor over the **Earth** image and then press Enter.

    Note: In the **Developer Tools**, under the **map** tag, inside the **area** tag, verify that the value of **href** for the **London** attribute is **/CityDetails/2**.

13. In Microsoft Edge, on the **Earth** image, click the **London** area. Note the red arrow at the center of the **Earth** image.

    Note: Verify that the new route works. 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.

14. In Microsoft Edge, click **Close**.

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

Exercise 4: Adding an Action Filter

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
```

8. Append the following code to the existing line of code:

```
 : ActionFilterAttribute
```

9. In the **LogActionFilterAttribute** class code block, press Enter, and then type the following code:

```
private IHostingEnvironment _environment;
private string _contentRootPath;
private string _logPath;
```

```
     private string _fileName;
     private string _fullPath;

     public LogActionFilterAttribute(IHostingEnvironment 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** method code block, press Enter twice, and then type the following code:

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

2. In the **OnActionExecuting** method code block, press Enter, type the following code, and then press Enter.

```
     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
started, event fired: OnActionExecuting");
          }
     }
```

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 twice, and then type the following code:

```
     public override void OnActionExecuted(ActionExecutedContext filterContext)
     {
     }
```

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
finished, event fired: OnActionExecuted");
          }
     }
```

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 twice, 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 the following viewData : {result.ViewData.Values.FirstOrDefault()}, event fired:
OnResultExecuted");
          }
     }
```

Task 5: Apply the action filter to the controller action

1. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, click **Startup.cs**.

2. Place the cursor at the end of the **using WorldJourney.Models** namespace code, press Enter, and then type the following code:

```
using WorldJourney.Filters;
```

3. In the **Startup.cs** code window, locate the following code:

```
services.AddSingleton<IData, Data>();
```

4. Place the mouse cursor after the located code, type the following code, and then press Enter.

```
services.AddScoped<LogActionFilterAttribute>();
```

5. In the **WorldJourney - Microsoft Visual Studio** window, in Solution Explorer, expand **Controllers**, and then click **CityController.cs**.

6. In the **CityController.cs** code block, locate the following code:

```
using WorldJourney.Models;
```

7. Ensure that the cursor is at the end of the **using WorldJourney.Models** namespace, press Enter, and then type the following code:

```
using WorldJourney.Filters;
```

8. In the **CityController** class code block, locate the following code:

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

9. 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. Click **Go Next**.

4. In Microsoft Edge, on the **Earth** image, click the **London** area. Note the red arrow in the center of the **Earth** image.

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 **[Repository Root]\Allfiles\Mod04\Labfiles\01_WorldJourney_begin\WorldJourney\LogFile** and open **Text file**.

   Note: **Text file** displays the new filter result.

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