**Understanding Endpoints**

In an ASP.NET Core application, incoming requests are handled by endpoints. The endpoint that produced the response is an action, which is a method that is written in C#. An action is defined in a controller, which is a C# class that is derived from the *Microsoft.AspNetCore.Mvc.Controller* class, the built-in controller base class.

**Controllers**

Each public method defined by a controller is an action, which means you can invoke the action method to handle an HTTP request. The convention in ASP.NET Core projects is to put controller classes in a folder named Controllers.

**Controller Conventions**

Controller classes contain a name followed by the word Controller, which means that when you see a file called HomeController.cs, you know that it contains a controller called Home, which is the default controller that is used in ASP.NET Core applications.

**Index() Method**

The browser will make an HTTP request to the ASP.NET Core server. The configuration of the project created by the template means the HTTP request will be processed by the *Index* action defined by the *Home controller*. Put another way, the request will be processed by the *Index* method defined by the *HomeController* class. The string produced by the Index method is used as the response to the browser's HTTP request.

```
public class HomeController : Controller{
   public string Index() => "Hello, World!";
}
```

**Understanding Routes**

The ASP.NET Core *routing system* is responsible for selecting the endpoint that will handle an HTTP request. A route is a rule that is used to decide how a request is handled. When the project was created, a default rule was created to get started. You can request any of the following URLs, and they will be dispatched to the Index action defined by the Home controller:

- /
- /Home
- /Home/Index

So, when a browser requests http://yoursite/ or http://yoursite/Home or http://yoursite/Home/Index, it gets back the output from HomeController's Index method.

**Understanding HTML Rendering**

The output from the previous example wasn't HTML—it was just the string Hello World. To produce an HTML response to a browser request, I need a view, which tells ASP.NET Core how to process the result produced by the Index method into an HTML response that can be sent to the browser.

**Creating and Rendering a View**

The first thing I need to do is modify my Index action method.

*public ViewResult Index() => View("MyView");*

When I return a ViewResult object from an action method, I am instructing ASP.NET Core to render a view. I create the ViewResult by calling the View method, specifying the name of the view that I want to use, which is MyView.

Add a new file to Views/Home named *MyView.cshtml.* The file should contain mostly html code and the following lines of code (*razor expression*):

```
@{
   Layout = null;
}
```

**What is Razor?**

The above lines of code is an expression that will be interpreted by Razor, which is the component that processes the contents of views and generates HTML that is sent to the browser. Razor is a view engine, and the expressions in views are known as Razor expressions.

**Views**

Views are stored in the Views folder, organized into subfolders. Views that are associated with the Home controller, for example, are stored in a folder called Views/Home. Views that are not specific to a single controller are stored in a folder called Views/Shared.

**How is rendered a view, actually?**

When I first edited the Index action method, it returned a string value. This meant that ASP.NET Core did nothing except pass the string value as is to the browser. Now that the Index method returns a ViewResult, Razor is used to process a view and render an HTML response. Razor was able to locate the view because I followed the standard naming convention, which is to put view files in a folder whose name matched the controller that contains the action method. In this case, this meant putting the view file in the Views/Home folder, since the action method is defined by the Home controller.

**What can you return from Index method except a string and a ViewResult?**

I can return other results from action methods besides *strings* and *ViewResult* objects. For example, if I return a *RedirectResult*, the browser will be redirected to another URL. If I return an *HttpUnauthorizedResult*, I can prompt the user to log in. These objects are collectively known as action results. The action result system lets you encapsulate and reuse common responses in actions.

**Adding Dynamic Output**

The whole point of a web application is to construct and display dynamic output. The job of the action method is to construct data and pass it to the view so it can be used to create HTML content based on the data values. Action methods provide data to views by passing arguments to the View method. The data provided to the view is known as the *view model*.

```
public ViewResult Index()
{
   int hour = System.DateTime.Now.Hour;
   string viewModel = hour < 12 ?
      "Good Morning!" :
      "Good Afternoon!";
   return View("MyView", viewModel);
}
```

The view model in this example is a string, and it is provided to the view as the second argument to the View method. Now you must add the following lines of code to the *MyView.cshtml* file so that the view will receive and use the view model.

```
…
@model string
…
<body>
   <div>
      @Model World (from the view)
   </div>
</body>
```

**@model and @Model**

The type of the view model is specified using the @model expression, with a lowercase m. The view model value is included in the HTML output using the @Model expression, with an uppercase M.

*@model* – Represents the type of the view model
*@Model* – Represents the value of the view model

When the view is rendered, the view model data provided by the action method is inserted into the HTML response.

**Overall Workflow**

The ASP.NET Core platform receives an HTTP request and uses the routing system to match the request URL to an endpoint. The endpoint, in this case, is the Index action method defined by the Home controller. The method is invoked and produces a ViewResult object that contains the name of a view and a view model object. The Razor view engine locates and processes the view, evaluating the @Model

expression to insert the data provided by the action method into the response, which is returned to the browser and displayed to the user.