CHAPTER 13

Creating the Example Project

In Part 1, I focused on explaining how to use Identity. In this part of the book, I explain how Identity works, revisiting the major features and describing what happens behind the scenes. I create custom user role stores, use custom user and role classes, and implement many of the interfaces that ASP.NET Core Identity uses.

In this chapter, I create a simple example project that is used in the chapters that follow. I use this project to explain how ASP.NET Core approaches authentication and authorization and how Identity builds on those features.

Creating the Project

Open a new PowerShell command prompt from the Windows Start menu and run the commands shown in Listing 13-1.

Tip You can download the example project for this chapter—and for all the other chapters in this book—from https://github.com/Apress/pro-asp.net-core-identity. See Chapter 1 for how to get help if you have problems running the examples.

Listing 13-1. Creating the Project

```
dotnet new globaljson --sdk-version 5.0.100 --output ExampleApp
dotnet new web --no-https --output ExampleApp --framework net5.0
dotnet new sln -o ExampleApp

dotnet sln ExampleApp add ExampleApp
```

Open the project for editing and make the changes shown in Listing 13-2 to the `launchSettings.json` file in the `Properties` folder to set the port that will be used to handle HTTP and requests.

Listing 13-2. Configuring HTTP Ports in the launchSettings.json File in the Properties Folder

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:5000",
      "sslPort": 0
    }
```


A. Freeman, *Pro ASP.NET Core Identity*, https://doi.org/10.1007/978-1-4842-6858-2_13

```
  },
  "profiles": {
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IdentityApp": {
      "commandName": "Project",
      "dotnetRunMessages": "true",
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Installing the Bootstrap CSS Framework

Use a command prompt to run the commands shown in Listing 13-3 in the ExampleApp folder to initialize the Library Manager tool and install the Bootstrap CSS package, which I use to style HTML content.

Listing 13-3. Installing the Client-Side CSS Package

```
dotnet tool uninstall --global Microsoft.Web.LibraryManager.Cli
dotnet tool install --global Microsoft.Web.LibraryManager.Cli --version 2.1.113
libman init -p cdnjs
libman install twitter-bootstrap@4.5.0 -d wwwroot/lib/twitter-bootstrap
```

Configuring Razor Pages

Create the ExampleApp/Pages folder and add to it a Razor View Imports file named _ViewImports.cshtml with the contents shown in Listing 13-4.

Listing 13-4. The Contents of the _ViewImports.cshtml File in the Pages Folder

```
@namespace ExampleApp.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@using Microsoft.AspNetCore.Mvc.RazorPages
```

Add a Razor View Start file named _ViewStart.cshtml to the Pages folder with the content shown in Listing 13-5.

Listing 13-5. The Contents of the _ViewStart.cshtml File in the ExampleApp/Pages Folder

```
@{
    Layout = "_Layout";
}
```

Create the Pages/Shared folder and add to it a Razor Layout named _Layout.cshtml with the contents shown in Listing 13-6.

Listing 13-6. The Contents of the _Layout.cshtml File in the Pages/Shared Folder

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>ExampleApp</title>
    <link href="/lib/twitter-bootstrap/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

Add a Razor Page named Test.cshtml to the Pages folder with the content shown in Listing 13-7.

Listing 13-7. The Contents of the Test.cshtml File in the Pages Folder

```
@page

<h4 class="bg-primary m-2 p-2 text-white text-center">
    Example App Razor Page
</h4>
```

This Razor Page will be used to ensure that the project is correctly configured and that the HTML content is styled using the Bootstrap CSS framework.

Configuring the MVC Framework

I use Razor Pages when I need something simple and self-contained. For more complex features, I prefer to use the MVC Framework. Create the ExampleApp/Controllers folder and add to it a class file named HomeController.cs with the code shown in Listing 13-8.

Listing 13-8. The Contents of the HomeController.cs File in the Controllers Folder

```
using Microsoft.AspNetCore.Mvc;

namespace ExampleApp.Controllers {

    public class HomeController: Controller {
```

```
        public IActionResult Test() => View();
    }
}
```

The Home controller defines an action named Test that renders its default view. Create the ExampleApp/Views folder and add to it a Razor View Imports file named _ViewImports.cshtml with the contents shown in Listing 13-9.

Listing 13-9. The Contents of the _ViewImports.cshtml File in the Views Folder

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Add a Razor View Start file named _ViewStart.cshtml to the Views folder with the content shown in Listing 13-10.

Listing 13-10. The Contents of the _ViewStart.cshtml File in the Views Folder

```
@{
    Layout = "_Layout";
}
```

Create the Views/Home folder and add to it a Razor View (using the Razor View – Empty template in Visual Studio) named Test.cshtml with the content shown in Listing 13-11.

Listing 13-11. The Contents of the Test.cshtml File in the Views/Home Folder

```
@model string

<h4 class="bg-primary m-2 p-2 text-white text-center">
    @(Model ?? "Example App Controller")
</h4>
```

Configuring the Application

The final step is to configure ASP.NET Core to enable Razor Pages, the MVC Framework, and the features that support them. Replace the contents of the Startup.cs file with the code shown in Listing 13-12.

Listing 13-12. Configuring the Application in the Startup.cs File in the ExampleApp Folder

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;

namespace ExampleApp {
    public class Startup {

        public void ConfigureServices(IServiceCollection services) {
            services.AddRazorPages();
            services.AddControllersWithViews();
        }
```

```
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env) {
            app.UseDeveloperExceptionPage();
            app.UseStaticFiles();
            app.UseRouting();

            app.UseEndpoints(endpoints => {
                endpoints.MapGet("/", async context => {
                    await context.Response.WriteAsync("Hello World!");
                });
                endpoints.MapRazorPages();
                endpoints.MapDefaultControllerRoute();
            });
        }
    }
}
```

This configuration enables Razor Pages and the MVC Framework and adds support for serving static files, which is required for the CSS stylesheet added to the project in Listing 13-12 and used in the layouts for HTML content.

Testing the Application

Run the command shown in Listing 13-13 in the ExampleApp folder to start ASP.NET Core and wait for HTTP requests.

Listing 13-13. Starting the Application

```
dotnet run
```

Open a new browser window and request `http://localhost:5000`; you will see the response shown in Figure 13-1, which uses the placeholder code added to projects when they are created using the command in Listing 13-1.

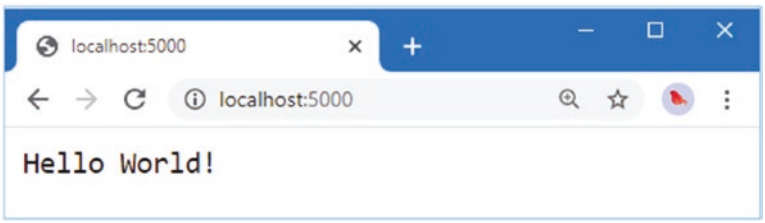

Figure 13-1. *The default application response*

Next, request `http://localhost:5000/test` and `http://localhost:5000/home/test`, which will produce the responses shown in Figure 13-2, confirming that Razor Pages and the MVC Framework are working.

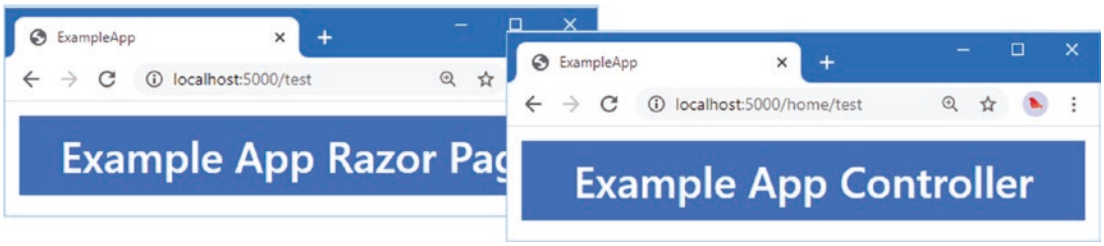

Figure 13-2. *Responses from Razor Pages and the MVC Framework*

Summary

In this chapter, I created the example project that I use throughout this part of the book. In the next chapter, I explain how the ASP.NET Core platform approaches request authentication.