CHAPTER 1

Using Visual Studio 2017

In this chapter, I explain the process for installing Visual Studio 2017 and recreate the Party Invites project from Chapter 2 of Pro ASP.NET Core MVC. As you will see, there are few differences from earlier versions of Visual Studio when working on simple projects. Working with Visual Studio 2017 does require changes for more complex projects, which you can see in later chapters in this update.

Installing Visual Studio 2017

Visual Studio 2017 is available in Community, Professional and Enterprise editions. I use the Community edition for the examples in this update, which is available for free and which has all of the features required for day-to-day development.

Download and run the installer from `www.visualstudio.com`. During setup make sure that you select the `.NET Core Cross-Platform Development` workload, as shown in Figure 1. Click the `Install` button to begin the installation process.

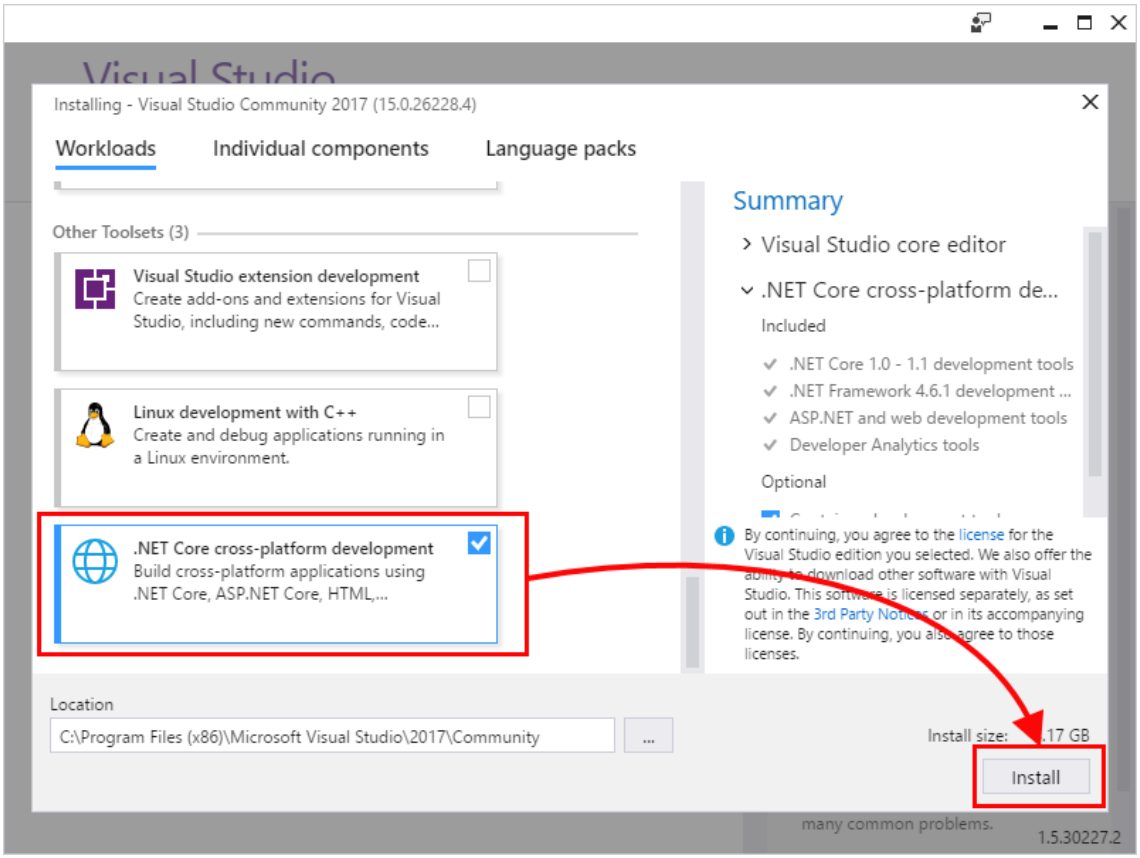

Figure 1. Installing Visual Studio 2017

Adding the Visual Studio Extensions

There are two Visual Studio extensions that are essential for working on ASP.NET Core MVC projects. The first is called Razor Language Service and it provides IntelliSense support for tag helpers when editing Razor views. The seconds is called Project File Tools and it provides automatic completion for editing `csproj` files, which have replaced `project.json` as the file that describes a project.

Select `Extensions and Updates` from the Visual Studio `Tools` menu, select the `Online` section and use the search box to locate the extensions. Click the `Download` button, as shown in Figure 2, to download the extension files.

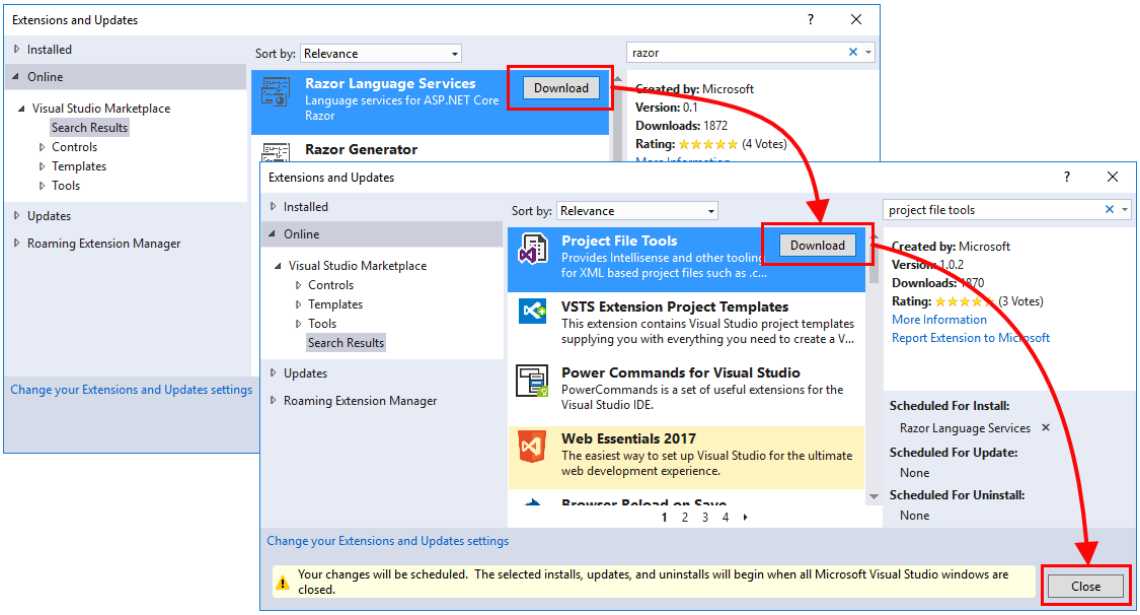

Figure 2. Downloading Visual Studio extensions

Click the **Close** button to dismiss the list of extensions and then close Visual Studio, which will trigger the installation process for the extensions you downloaded. You will be prompted to accept the changes that will be made and the license terms, as shown in Figure 3. Click the **Modify** button to install the extensions. Once the process has completed, you can start Visual Studio and begin development.

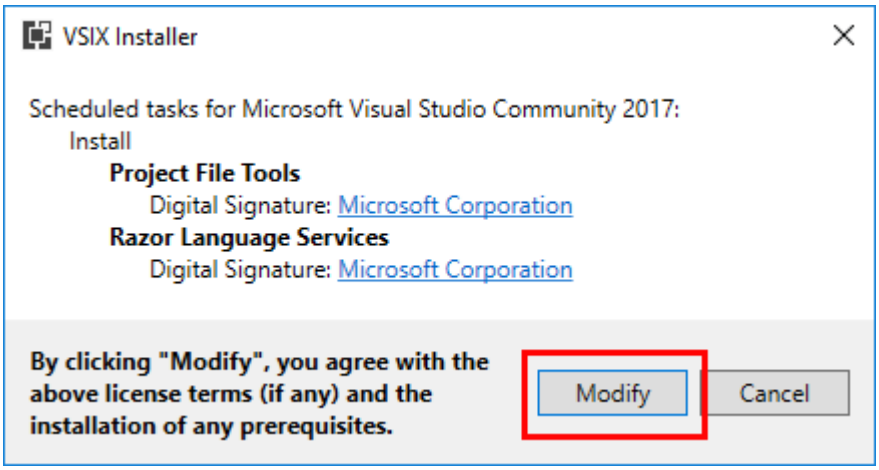

Figure 3. Installing Visual Studio extensions

Recreating the Party Invites Application

Once you have Visual Studio 2017 installed, the development process is largely unchanged from earlier releases. There are some important changes for more complex projects, which I explain in the next chapter of this update, but working on simple projects requires no substantial changes. As a demonstration, I am going recreate the Party Invites application from Chapter 2 of Pro ASP.NET Core MVC. I recreate the project without explaining the individual steps; see the original book chapter if you want the step-by-step instructions.

Creating the Project

To create the project, select `New` > `Project` from the Visual Studio `File` menu and select the `Templates` > `Visual C#` > `.NET Core` section of the `New Project` dialog window. Select the ASP.NET Core Web Application (.NET Core) item, as shown in Figure 4, and enter `PartyInvites` into the `Name` field.

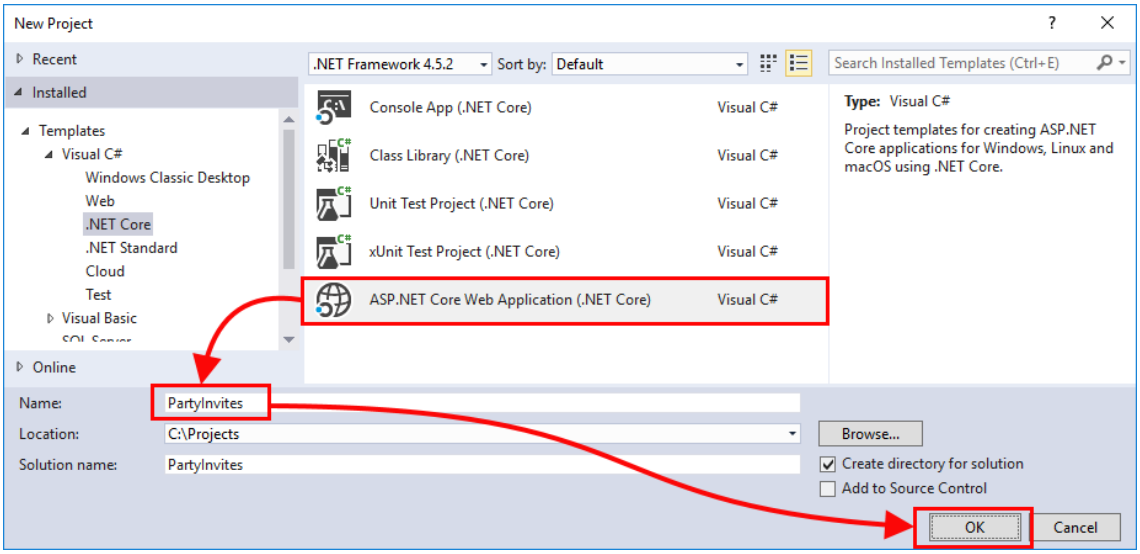

Figure 4. Creating the new project

Click the OK button. The next step is to configure the project and select its initial content. Ensure that **`ASP.NET Core 1.1`** is selected from the drop-down list, as shown in Figure 5, click the **`Web Application`** item and ensure that the **`No Authentication`** option is selected.

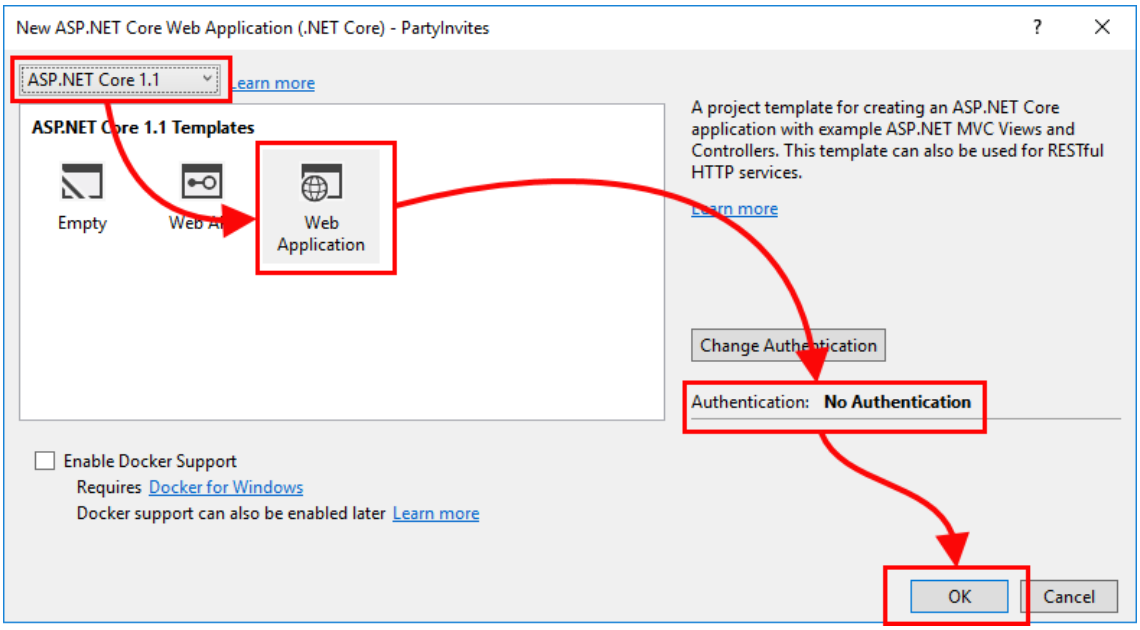

Figure 5. Configuring the ASP.NET Core MVC project

Click the OK button and Visual Studio will create the project, add the initial content and install the NuGet packages that the project requires.

Creating the Model

Right-click on the **PartyInvites** project item in the Solution Explorer window and select **Add** > **New Folder** from the popup list and set the name of the folder to **Models**. Right click on the **Models** folder, select **Add** > **Class** and create a new class file called **GuestResponse.cs**. Replace the contents of the class file with the code shown in Listing 1.

Listing 1. The Contents of the GuestResponse.cs File in the Models Folder

```
using System.ComponentModel.DataAnnotations;

namespace PartyInvites.Models {

    public class GuestResponse {

        [Required(ErrorMessage = "Please enter your name")]
```

```
        public string Name { get; set; }

        [Required(ErrorMessage = "Please enter your email address")]
        [RegularExpression(".+\\@.+\\..+",
            ErrorMessage = "Please enter a valid email address")]
        public string Email { get; set; }

        [Required(ErrorMessage = "Please enter your phone number")]
        public string Phone { get; set; }

        [Required(ErrorMessage = "Please specify whether you'll attend")]
        public bool? WillAttend { get; set; }
    }
}
```

This project includes a simple in-memory repository to store the responses from users. Add a new class file called **Repository.cs** in the **Models** folder and replace its contents with the code shown in Listing 2.

Listing 2. The Contents of the Repository.cs File in the Models Folder

```
using System.Collections.Generic;

namespace PartyInvites.Models {
    public static class Repository {
        private static List<GuestResponse> responses = new List<GuestResponse>();

        public static IEnumerable<GuestResponse> Responses {
            get {
                return responses;
            }
        }

        public static void AddResponse(GuestResponse response) {
            responses.Add(response);
        }
    }
}
```

Creating the Controller and Views

This project uses a single controller to select the views displayed to users and to receive form data. Edit the **HomeController.cs** file in the **Controllers** folder to replace the placeholder code provided by Visual Studio with the statements shown in Listing 3.

Listing 3. The Contents of the HomeController.cs File in the Controllers Folder

```
using System;
using Microsoft.AspNetCore.Mvc;
using PartyInvites.Models;
using System.Linq;

namespace PartyInvites.Controllers {

    public class HomeController : Controller {

        public ViewResult Index() {
            int hour = DateTime.Now.Hour;
            ViewBag.Greeting = hour < 12 ? "Good Morning" : "Good Afternoon";
            return View("MyView");
        }

        [HttpGet]
        public ViewResult RsvpForm() {
            return View();
        }

        [HttpPost]
        public ViewResult RsvpForm(GuestResponse guestResponse) {
            if (ModelState.IsValid) {
                Repository.AddResponse(guestResponse);
                return View("Thanks", guestResponse);
            } else {
                // there is a validation error
                return View();
            }
        }

        public ViewResult ListResponses() {
            return View(Repository.Responses.Where(r => r.WillAttend == true));
        }
    }
}
```

To create the view that is presented to users when they start using the application, add a Razor file called `MyView.cshtml` to the `Views/Home` folder and add the markup shown in Listing 4.

Listing 4. The Contents of the MyView.cshtml File in the Views/Home Folder

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

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.css" />
</head>
<body>
    <div class="text-center">
        <h3>We're going to have an exciting party!</h3>
        <h4>And you are invited</h4>
        <a class="btn btn-primary" asp-action="RsvpForm">RSVP Now</a>
    </div>
</body>
</html>
```

Next, add a Razor file called `RsvpForm.cshtml` to the `Views/Home` folder and add the content shown in Listing 5. This is the form that is used to gather a response from the user.

Listing 5. The Contents of the RsvpForm.cshtml File in the Views/Home Folder

```
@model PartyInvites.Models.GuestResponse

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
    <link rel="stylesheet" href="/css/styles.css" />
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.css" />
</head>
<body>
    <div class="panel panel-success">
        <div class="panel-heading text-center"><h4>RSVP</h4></div>
        <div class="panel-body">
            <form class="p-a-1" asp-action="RsvpForm" method="post">
                <div asp-validation-summary="All"></div>
                <div class="form-group">
                    <label asp-for="Name">Your name:</label>
                    <input class="form-control" asp-for="Name" />
                </div>
                <div class="form-group">
                    <label asp-for="Email">Your email:</label>
```

```
                    <input class="form-control" asp-for="Email" />
                </div>
                <div class="form-group">
                    <label asp-for="Phone">Your phone:</label>
                    <input class="form-control" asp-for="Phone" />
                </div>
                <div class="form-group">
                    <label>Will you attend?</label>
                    <select class="form-control" asp-for="WillAttend">
                        <option value="">Choose an option</option>
                        <option value="true">Yes, I'll be there</option>
                        <option value="false">No, I can't come</option>
                    </select>
                </div>
                <div class="text-center">
                    <button class="btn btn-primary" type="submit">
                        Submit RSVP
                    </button>
                </div>
            </form>
        </div>
    </div>
</body>
</html>
```

To create the view that is presented to the user at the end of the response process, add a Razor file called `Thanks.cshtml` to the `Views/Home` folder and add the content shown in Listing 6.

Listing 6. The Contents of the Thanks.cshtml File in the Views/Home Folder

```
@model PartyInvites.Models.GuestResponse

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Thanks</title>
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.css" />
</head>
<body class="text-center">
    <p>
        <h1>Thank you, @Model.Name!</h1>
        @if (Model.WillAttend == true) {
```

```
            @:It's great that you're coming. The drinks are already in the fridge!
        } else {
            @:Sorry to hear that you can't make it, but thanks for letting us know.
        }
    </p>
    Click <a class="nav-link" asp-action="ListResponses">here</a>
    to see who is coming.
</body>
</html>
```

The final view provides the list of responses that have been made. Add a Razor file called **ListResponses.cshtml** in the **Views/Home** folder and add the content shown in Listing 7.

Listing 7. The Contents of the ListResponses.cshtml File in the Views/Home Folder

```
@model IEnumerable<PartyInvites.Models.GuestResponse>

@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.css" />
    <title>Responses</title>
</head>
<body>
    <div class="panel-body">
        <h2>Here is the list of people attending the party</h2>
        <table class="table table-sm table-striped table-bordered">
            <thead>
                <tr><th>Name</th><th>Email</th><th>Phone</th></tr>
            </thead>
            <tbody>
                @foreach (PartyInvites.Models.GuestResponse r in Model) {
                    <tr><td>@r.Name</td><td>@r.Email</td><td>@r.Phone</td></tr>
                }
            </tbody>
        </table>
    </div>
</body>
</html>
```

Running the Example Project

To test the application, select **Start without Debugging** from the Visual Studio **Debug** menu. Visual Studio will build and start the project and open a new browser tab that displays the application, as shown in Figure 6.

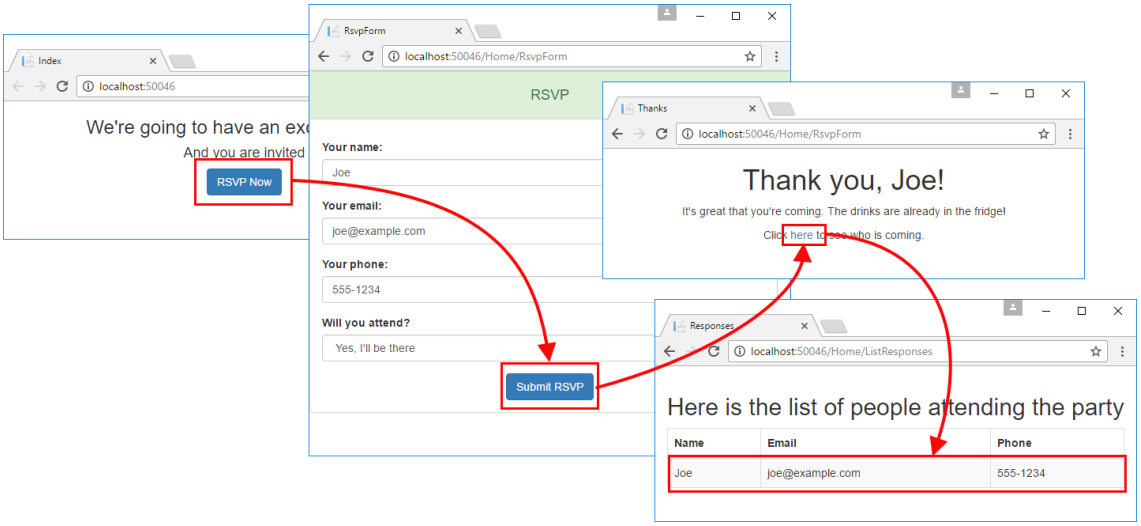

Figure 6. Running the example project

Summary

In this chapter, I explained how to install Visual Studio 2017 and demonstrated that the development process is largely unchanged for simple projects. In the next chapter, I use the SportsStore example to highlight some important differences for more complex projects.