In-class Lab 02

ASP.NET Core MVC

1 Beginning the lab

1. Create a new project. The target framework should be .NET Core 2.0. Select `File` ▶ `New Project` ▶ `Visual C#` ▶ `Web`. Select `ASP.NET Core Web Application`. Name the application `PartyInvites` and save it in your `/aspnetcore/projects` directory. See figure 1. Click `OK`.

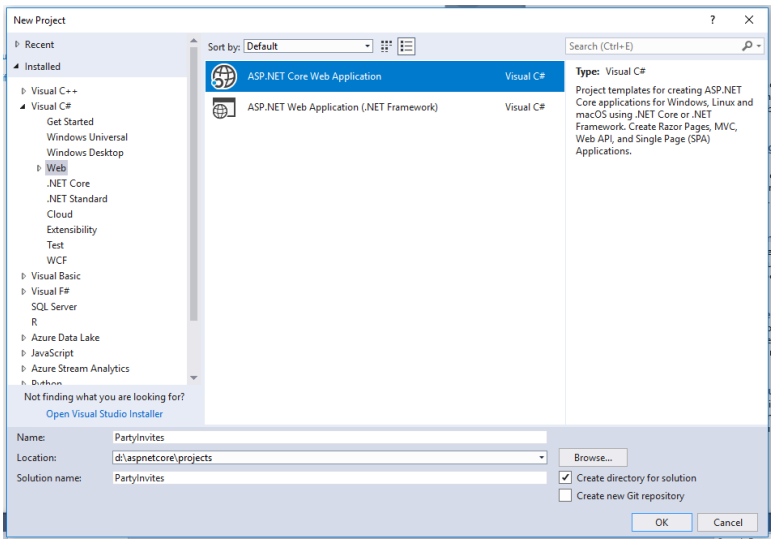

Figure 1: Create a Web Application

2. Select the `Web Application (Model-View-Controller` template. Make sure that **No Authentication** is selected. See figure 2. Click `OK`.

3. Review the directory structure that Visual Studio creates for you. Eventually, you will become familiar with all of these directories and files. See figure 3.

4. Select `Debug` ▶ `Start Debugging`. When the web page loads, click through the menu and explore your new web application. When you finish, close the tab and stop debugging.

5. Select the `Home Controller` to open it in the code window, and edit it to match listing 1.

Listing 1: Home controller edit

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
```

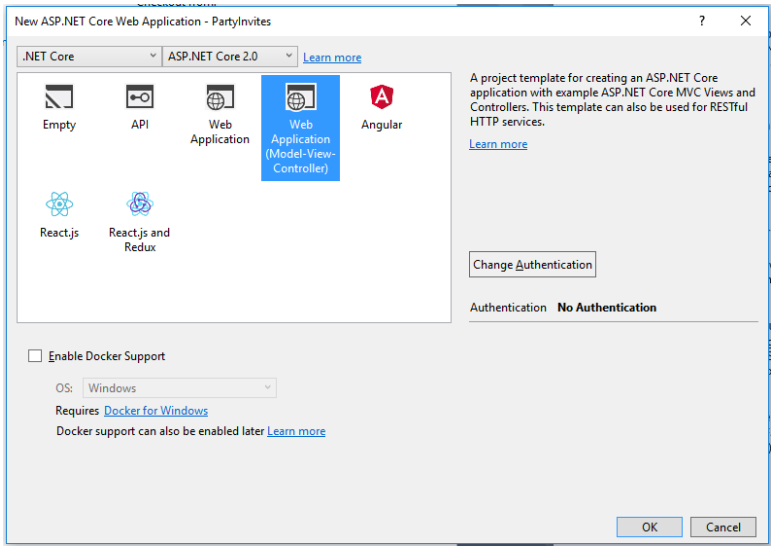

Figure 2: Use the MVC template

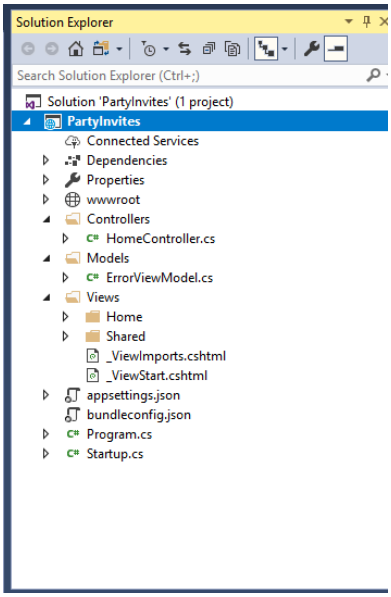

Figure 3: Examine the directory structure

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

namespace PartyInvites.Controllers
{
    public class HomeController : Controller
    {
        public string Index()
```

```
        {
            return "Hello, MSSA!";
        }
    }
}
```

6. Select `Debug ▶ Start Debugging`. When the web page loads, the address bar will display something similar to this: `http://localhost:50025/`. What do you see when you append `/home/` and `/home/index/` to the base URL?

2 Creating a view

7. Edit the home controller so that the `Index()` method matches listing 2. Start without debugging. What happens? Can you understand these error messages?

```
An unhandled exception occurred while processing the request.
InvalidOperationException: The view 'MyView' was not found.
            The following locations were searched:
            /Views/Home/MyView.cshtml
            /Views/Shared/MyView.cshtml
```

Listing 2: Home controller returning a view

```
        public ViewResult Index()
        {
            return View("MyView");
        }
```

8. The problem is that ASP.NET could not find any view named "MyView." To add a View, right click anywhere in the definition of the `Index` method. Click `Add View`. Name the view `MyView`. See figure 4.

9. Edit the view you just created to match listing 3. Start without debugging and examine the result. What happened? When you figure it out, close the browser page.

Listing 3: `MyView` edit

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Index</title>
</head>
<body>
    <div>
```

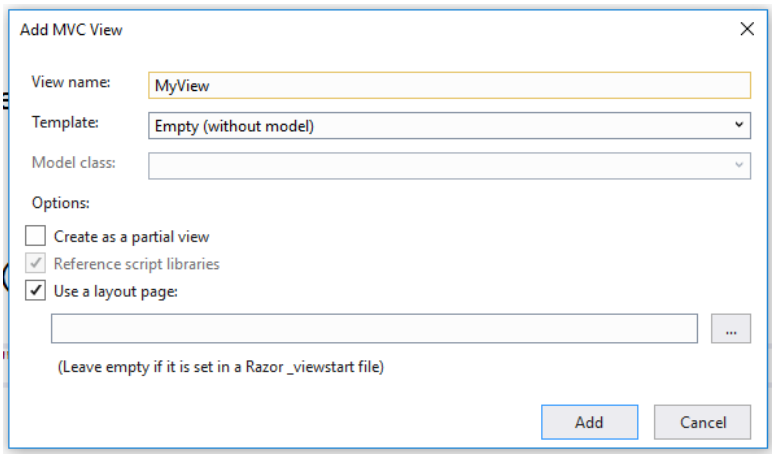

Figure 4: Adding a view

```
        Hello MSSA from the view.
    </div>
</body>
</html>
```

10. Now, edit the `HomeController` file by adding the code in listing 4 to the `Index()` method.

Listing 4: `Home Controller` edit

```
public ViewResult Index()
{
    int hour = DateTime.Now.Hour;
    string timeOfDay;
    if (hour < 12)
        timeOfDay = "Good Morning";
    else if (hour < 18)
        timeOfDay = "Good Afternoon";
    else
        timeOfDay = "Good Evening";
    ViewBag.Greeting = timeOfDay;
    return View("MyView");
}
```

11. Next, edit the `Index.cshtml` by changing the code within the `<div>` tag to the listing 5. Now, run without debugging and look at the result. What happened? When you have figured it out, you may close your browser window.

Listing 5: `MyView` edit

```
<body>
    <div>
        <p>@ViewBag.Greeting, Earth and World.</p>
```

```
        <p>Hello MSSA from the view.</p>
    </div>
</body>
```

3 Create a simple data entry application

12. Edit `MyView.cshtml` to match the listing in 6. Start without debuggng. What happened?

Listing 6: `MyView` edit

```
    @{
    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>
        <h2>@ViewBag.Greeting, Earth and World.</h2>
        <p>We're going to have an exciting party.<br />
            (TODO: sell it better. Add pictures or something.)
        </p>
    </div>
</body>
</html>
```

13. Right click `Models` in the `Solution Explorer` window and select `Add` ► `Class` from the pop-up menus. Set the file name to `GuestResponse.cs` and click `Add` to add the class. See figure 5. Edit the contents of the class to match listing 7. What is the type of the property `WillAttend`? Why?

Listing 7: `GuestResponse.cs`

```
namespace PartyInvites.Models
{
    public class GuestResponse
    {
        public string Name { get; set; }
        public string Email { get; set; }
        public string Phone { get; set; }
        public bool? WillAttend { get; set; }
```

```
    }
}
```

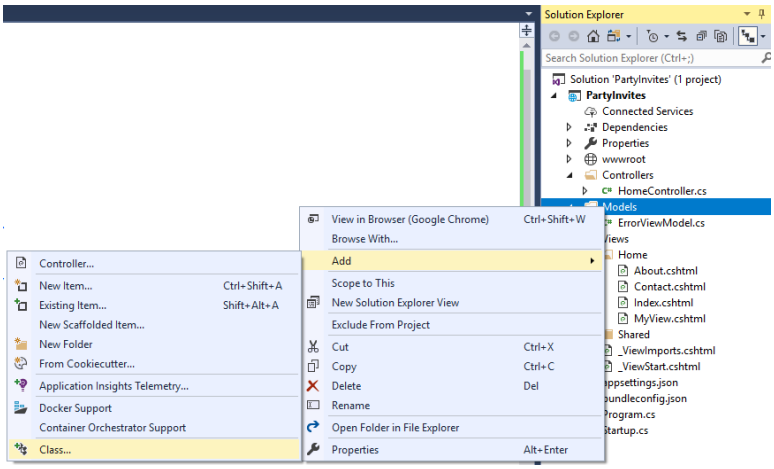

Figure 5: Adding a class

14. Edit `HomeController.cs` to add another method shown in listing 8.

Listing 8: `RsfpForm()`

```
public ViewResult RsvpForm()
{
    return View();
}
```

15. Build your project. If you have errors, correct all errors before proceeding. To add a view for the `RsvpForm()` method, right click the `Views/Home` folder, and select `Add` ► `View`. See figure 6. Name the view `RsvpForm.cs` and select the Empty template. Edit the page as shown in listing 9. Then, start without debugging, and navigate to `http://localhost:50025/home/rsvpform` (making sure that you have the right port). What happened?

Listing 9: `RsvpForm.cshtml`

```
@model PartyInvites.Models.GuestResponse

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
```

```
    <title>RsvpForm</title>
</head>
<body>
    <div>
        This is the RsvpForm.cshtml view.
    </div>
</body>
</html>
```

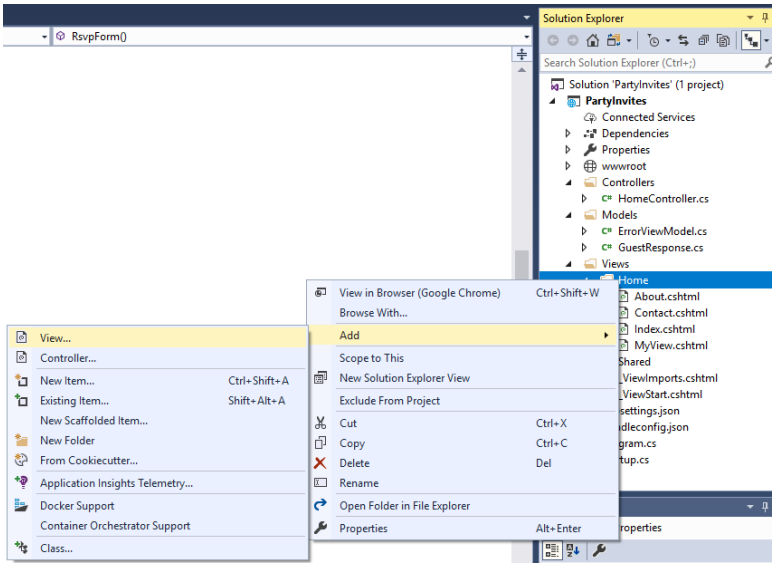

Figure 6: Adding a view

16. Edit the `MyView.cshtml` file by adding **<a asp-action="RsvpForm">RSVP Now**, as shown in listing 10. Start without debugging. You should see a hyperlink on the home page. Click the link. What happens? When you figure it out, close the page.

Listing 10: Adding a link to the MyView page

```
<div>
    <h2>@ViewBag.Greeting, Earth and World.</h2>
    <p>
        We're going to have an exciting party.<br />
        (TODO: sell it better. Add pictures or something.)
    </p>
    <a asp-action="RsvpForm">RSVP Now</a>
</div>
```

17. Now, edit the `RsvForm.cshtml` so that it conforms to listing 11. Start without debugging and click the link to the `RsvpForm`. You should see a form wooth labells and text entry boxes. Enter values in the text entry boxes, select an option from the drop down list, and click `Submit RSVP`. What happened? Close the page.

Listing 11: Adding a form to the RsvpForm

```
<body>
    <form asp-action="RsvpForm" method="post">
        <p>
            <label asp-for="Name">Your name:</label>
            <input class="form-control" asp-for="Name" />
        </p>
        <p>
            <label asp-for="Email">Your email:</label>
            <input class="form-control" asp-for="Email" />
        </p>
        <p>
            <label asp-for="Phone">Your phone:</label>
            <input class="form-control" asp-for="Phone" />
        </p>
        <p>
            <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>
        </p>
        <p>
        <button type="submit">Submit RSVP</button>
    </form>
</body>
```

18. In order to persist the form values, we need to make three changes to the `HomeControllers.cs` file.

    (a) First, add this line to the top of the file, following the other `using` statements:
    `using PartyInvites.Models;`

    (b) Second, alter the existing `RsvpForm()` method by adding the following attribute before the function definition:
    `[HttpGet]`

    (c) Finally, add a second `RsvpForm()` method, like this:

```
[HttpPost]
public ViewResult RsvpForm(GuestResponse guestResponse)
// TODO: Email response to the party organizer
return View("Thanks", guestResponse);
```

These two methods should look like listing 12. When you do this, navigate to the RSVP form, comnplete the form, and submit it. What happens? Close the page.

Listing 12: Adding attributes and a second method to `HomeControllers.cs`

```
[HttpGet]
```

```
public ViewResult RsvpForm()
{
    return View();
}

[HttpPost]
public ViewResult RsvpForm(GuestResponse guestResponse)
{
    // TODO: Email response to the party organizer
    return View();
}
```

19. In order to persist the RSVP resposes, we need a class to hold them. This will be saved in memory — we will use a real database a little later. Creaate a new class in the Models folder by right clicking Models, and selecting `Add` ► `Class`. See figure 7. Select `Class` and name the class `Repository.cs`. Click `Add`. See figure 8. Edit the class like listing 13.

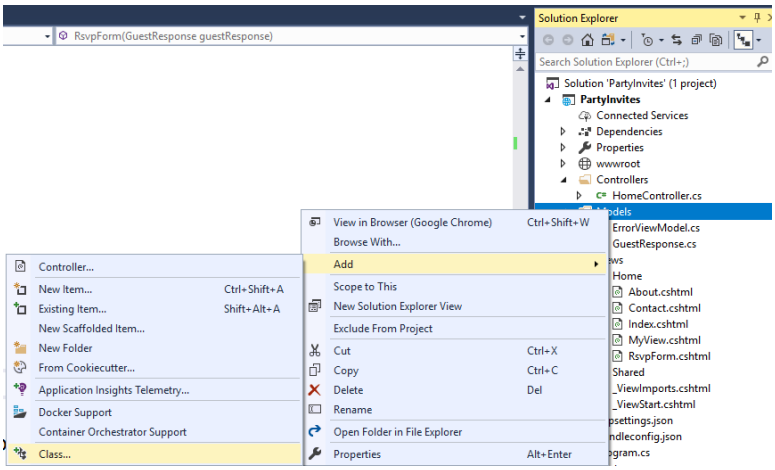

Figure 7: Adding a class to Models

Listing 13: Contents of `Repository.cs`

```
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;
            }
        }
```

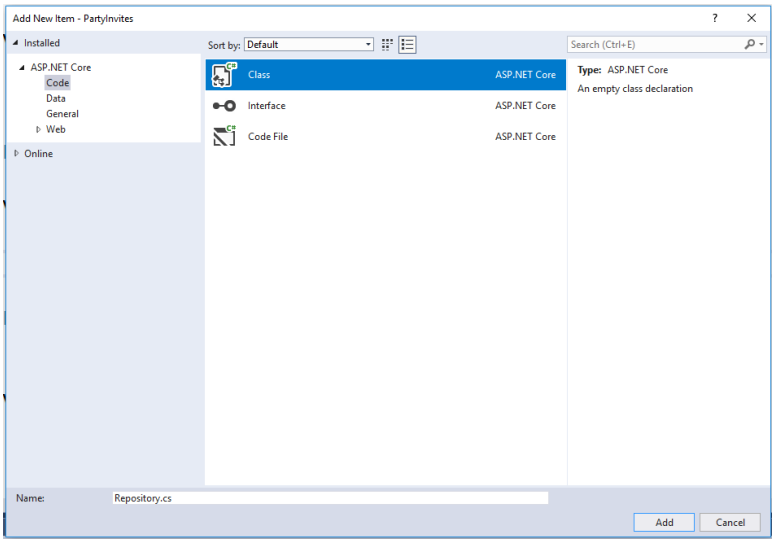

Figure 8: Creating class `Repository.cs`

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

20. Next, modify the `RsvpForm()` method in the `HomeController` class like listing 14. Note that there are two methods with the same name. Modify the method with the **[HttpPost]** attribute.

Listing 14: Editing the `RsvoForm()` method

```
[HttpPost]
public ViewResult RsvpForm(GuestResponse guestResponse)
{
    // TODO: Email response to the party organizer
    Repository.AddResponse(guestResponse);
    return View("Thanks", guestResponse);
}
```

21. In order to actually save the guest responses, you need a new view named `Thanks.cshtml`. Create a new view by right clicking the `Views/Home` folder and selecting `Add` ▶ `View`. See figure 9. Name the view `Thanks.cshtml` and click `Add`. See figure 10. Edit the view like listing 15. Start without debugging and navigate to the `RsvpForm`. Fill out the form and submit it. What happened? Close the page.

Listing 15: Editing `Thanks.cshtml`

```
@model PartyInvites.Models.GuestResponse
```

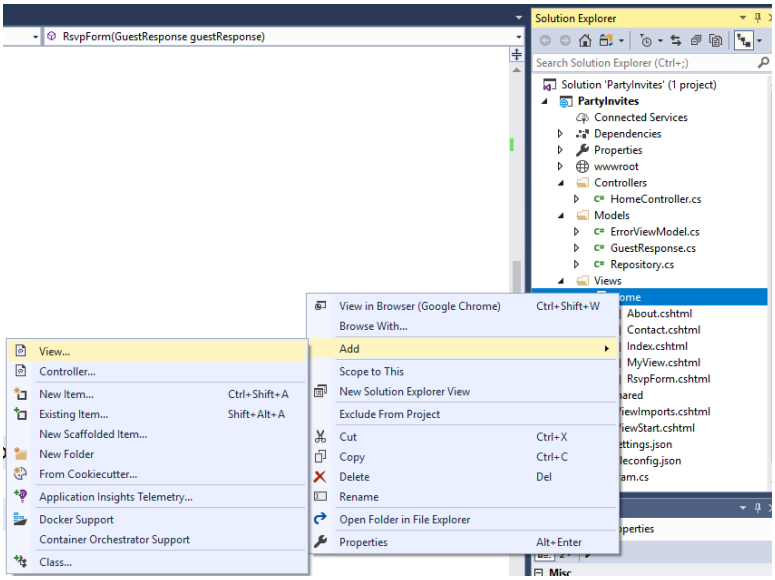

Figure 9: Adding a view to Views/Home

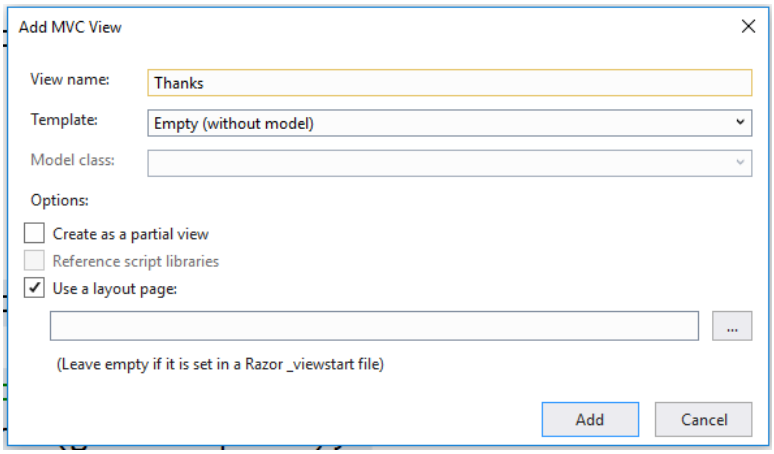

Figure 10: Adding view `Thanks.cshtml`

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Thanks</title>
```

```
</head>
<body>
    <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 asp-action="ListResponses">here</a> to see who is coming.
</body>
</html>
```

22. To display the list of those coming to the party, you will need to make two changes in `HomeController.cs` First, add **using System.Linq** to the `using` statements at the top of the file. Then, add a `ListResponses()` method shown in listing 16.

Listing 16: ListResponses() method

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

23. To create the view, right click the `Views/Home` folder, and add a view named `ListResponses.cshtml`. Then, edit `ListResponses.cshtml` to match the listing in 17. Start without debugging. RSVP to the party, and then click the link to see the responses. What happened? Close the window.

Listing 17: ListResponses.cshtml

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

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <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>
```

4 Adding Validation

24. Edit the `GuestResponse` class to add **using System.ComponentModel.DataAnnotations;** to the `using` statements at the top of the file. Then, add attributes to each of the four properties. Edit the file like listing 18.

Listing 18: Edits to `GuestResponse.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
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; }
    }
}
```

25. Edit `HomeController.cs` by changing the `RsvpForm()` method like 19. Start without debugging, navigate to the RSVP form, and make attempt to submit without completing all the form fields. What happens? Now submit the form completing all the fields. What happened? Close the browser window.

Listing 19: Edits to `RsvpMethod()`

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

26. Edit the `RsvpForm.cshtml` view by adding ** ** just below the opening form tag. See listing 20. Start without debugging, navigate to the RSVP form, and make attempt to submit without completing all the form fields. What happens? Now submit the form completing all the fields. What happened? Close the browser window.

Listing 20: Adding

```
<form asp-action="RsvpForm" method="post">
    <div asp-validation-summary="All"></div>
        <p>
            <label asp-for="Name">Your name:</label>
            <input class="form-control" asp-for="Name" />
        </p>
```

```
        <p>
            <label asp-for="Email">Your email:</label>
            <input class="form-control" asp-for="Email" />
        </p>
        <p>
            <label asp-for="Phone">Your phone:</label>
            <input class="form-control" asp-for="Phone" />
        </p>
        <p>
            <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>
        </p>
        <p>
        <button type="submit">Submit RSVP</button>
    </form>
```

Start without debugging, navigate to the RSVP form, and make attempt to submit without completing all the form fields. What happens? Now submit the form completing all the fields. What happened? Close the browser window.

27. Add a stylesheet by right clicking the `wwwroot/css` folder, and selecting `Add` ▶ `New Item`. See figure 11. Select `Style Sheet`, name the file `Styles.css`, and click `Add`. Edit the stylesheet to match listing 21. Then, edit the `RsvpForm` view by adding **<link rel="stylesheet" href="/css/styles.css" />** to the head, matching listing 22. Start without debugging, navigate to the RSVP form, and make attempt to submit without completing all the form fields. What happens? Now submit the form completing all the fields. What happened? Close the browser window.

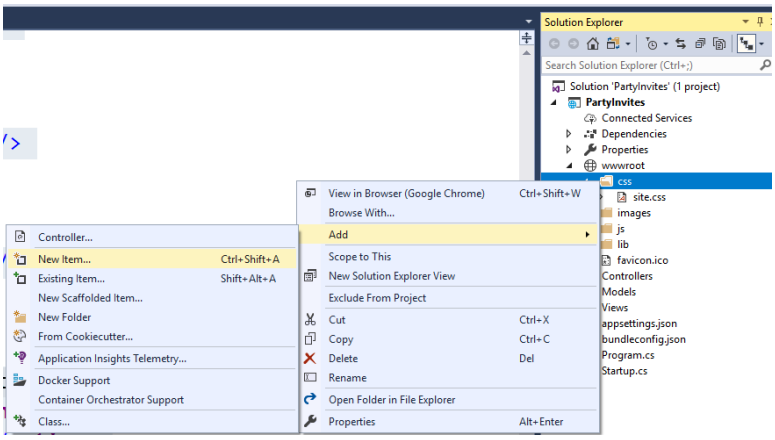

Figure 11: Adding a stylesheet, part 1

Listing 21: `Styles.css` style sheet

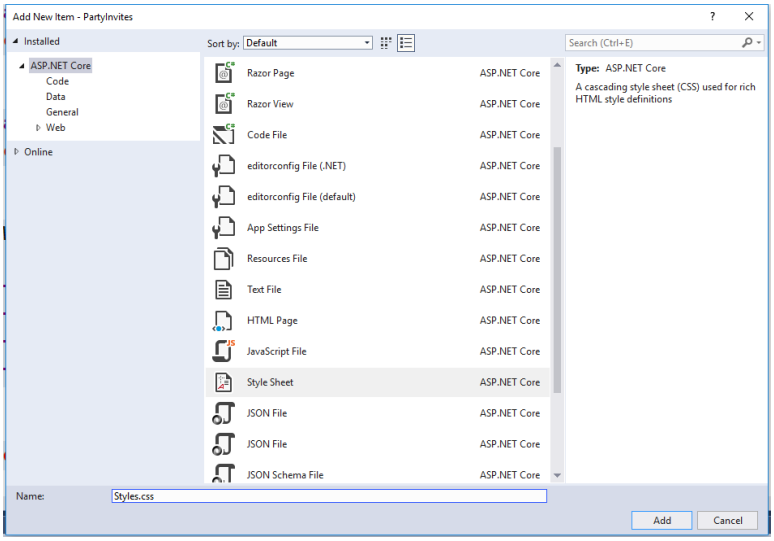

Figure 12: Adding a stylesheet, part 2

```
.field-validation-error    {color: #f00;}
.field-validation-valid    { display: none;}
.input-validation-error    { border: 1px solid #f00; background-color: #fee; }
.validation-summary-errors { font-weight: bold; color: #f00;}
.validation-summary-valid  { display: none;}
```

Listing 22: Adding the link to `RsvpForm`

```
<head>
    <meta name="viewport" content="width=device-width" />
    <title>RsvpForm</title>
    <link rel="stylesheet" href="/css/Styles.css" />
</head>
```

5 Adding style

28. Edit `MyView.cshtml` by adding a link element to the header, and class attributes to the body, shown in listing 23.

Listing 23: Final edit tyy `MyView.cshtml`

```
@{
    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>
```

29. Edit `RsvpForm.cshtml` by adding a link element to the header, and class attributes to the body, shown in listing 24.

Listing 24: Final edit to `RsvpForm.cshtml`

```
@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>
```

30. Edit `Thanks.cshtml` by adding a link element to the header, and class attributes to the body, shown in listing 25.

Listing 25: Final edit to `Thanks.cshtml`

```
@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>
```

31. Edit `ListResponses.cshtml` by adding a link element to the header, and class attributes to the body, shown in listing 26.

Listing 26: Findal edit to `ListResponses.cshtml`

```
@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>
```

32. Start without debugging, and put your PartyInvites application through its paces.