**CST-247 Activity Guide**

Contents

[Activity 2 2](#_Toc60388269)

[Part 1: Introduction to .NET Razor 2](#_Toc60388270)

[Part 2: Advanced .NET Razor 6](#_Toc60388271)

[Part 3: Data Validation 9](#_Toc60388272)

[Part 4: Practicing Agile Scrum 10](#_Toc60388273)

# Activity 2

This activity has multiple parts/assignments. All assignments must be completed prior to documentation submission.

## Part 1: Introduction to .NET Razor

**Overview**

In this activity, students will use .NET MVC Views to learn some of the Razor Syntax and Razor Helper methods, display a list of C# object models in a standard HTML table, and submit Image Buttons to a Controller.

**Execution**

Execute this activity according to the following guidelines:

*Section 1*

1. Create a new .NET MVC Application:
   1. Select File->New Project menu items. Under the Visual C# Windows templates select Web. From the listed templates select the ASP.NET Web Application template. Name your application Activity2Part1. Click OK.
   2. From the ASP.Net Template dialog, select the Empty template, check the MVC option under folders and core references, and uncheck the Host in the Cloud Azure option. Click OK.
2. Create a Test Controller and Route to test the Razor View:
   1. Right click on the Controllers Folder.
   2. Select the Add->Controller… menu items. Then select the ‘MVC 5 Controller – Empty’ scaffolding type. Click the Add button. Name your Controller ‘Test’ and click the Add button. Inspect the Test Controller (as TestController.cs class) within the Controllers folder.
   3. Add a Route for the Test Controller by updating the RouteConfig.cs class located with the App\_Start folder.

routes.MapRoute(

name: "Test",

url: "{Test}",

defaults: new { controller = "Test", action = "Index", id= UrlParameter.Optional }

);

1. Create a User Model class (to test our Razor View):
   1. Right click on the Models Folder.
   2. Create a Model C# class by selecting Add New Item menu. Then, under the Add New Item dialog, select Code under Visual C#, select Class, and enter UserModel for the name. Click Add.
   3. Add three properties to the UserModel class with both setter and getter methods: Name, Email Address, and Phone Number. Also add a nondefault constructor that will initialize the three properties.
      1. public UserModel(string Name, string Email, string Phone) { };
      2. public string Name { get; set; }
      3. public string EmailAddress { get; set; }
      4. public string PhoneNumber { get; set; }
2. Create a Test View:
   1. Right click on any part of the code body for the Test Controller. Select the Add View menu option. Change the View name to Test, select the Empty template, and from the Model class dropdown select the UserModel created in Step 3. Uncheck the Use Layout option. Click Add. Inspect the Test Page (as Test.cshtml) within the Views/Test folder.
   2. Update the Test Controller Index() to return an ActionResult of View(“Test”).
   3. Validate the Test Page by running the solution using localhost:[port]/Test as the URL.
3. Using the HTML Label Helper:
   1. Create two labels using the Html.Label():
      1. Label 1: Welcome to Razor, centered in bold, 24-point Arial font.
      2. Label 2: My Customers are:, left aligned in bold, 18-point Arial font.
   2. To test your View, refresh your browser when you update your View.
   3. Take a screenshot. Label this Screenshot 1.
4. Using the HTML Action Link Helper:
   1. Create a link labeled ‘Refresh Page’ to this page, using the Html.ActionLink().
   2. To test your View, refresh your browser when you update your View.
   3. Hover your mouse over the hyperlink to display the URL in your status bar.
   4. Take a screenshot. Label this Screenshot 2.
5. Using an iterator to display a list of Object Model objects:
   1. In the Index() of the Test Controller create and initialize a List of UserModels. Pass this list as model data to the Test View.
   2. At the top of the Test View change the model to:

@model List<Activity2Part1.Models.UserModel>

* 1. Create an HTML Table in the View.
  2. Create a Row of Headers in Table: Name, Email Address, Phone Number.
  3. Iterate over the Model and List of UserModel objects using the C# for each construct. For each iteration, render a table row and three columns of data from the Model:

<td>@Html.DisplayFor(modelItem => user.Name)</td>

<td>@Html.DisplayFor(modelItem => user.EmailAddress)</td>

<td>@Html.DisplayFor(modelItem => user.PhoneNumber)</td>

* 1. To test your View, refresh your browser when you update your View.
  2. Take a screenshot. Label this Screenshot 3.

*Section 2*

1. Open solution from Activity 2 Part 1, Section 1.
2. Create a Button Controller to test new Razor View:
   1. Right click on the Controllers Folder.
   2. Select the Add->Controller… menu items. Then select the ‘MVC 5 Controller – Empty’ scaffolding type. Click the Add button. Name your Controller ‘Button,’ and click the Add button. Inspect the Button Controller (as ButtonController.cs class) within the Controllers folder.
3. Create a Button Model class to test the Razor View:
   1. Right click on the Models Folder.
   2. Create a Model C# class by selecting Add New Item menu. Then under the Add New Item dialog select Code under Visual C#, select Class, and enter ButtonModel for the name. Click Add.
   3. Add one property named State ButtonModel class with both setter and getter methods. Also add a nondefault constructor that will initialize the three properties.
      1. public ButtonModel(bool State) { };
      2. public bool State { get; set; }
4. Create a Button View:
   1. Right click on any part of the code body for the Button Controller. Select the Add View menu option. Change the View name to Button, select the Empty template, and from the Model class dropdown, and select the ButtonModel created in Step 3. Uncheck the Use Layout option. Click Add. Inspect the Button Page (as Button.cshtml) within the Views/Button folder.
   2. Update the Button Controller Index() to return an ActionResult of View(“Button”).
   3. Validate the Button Page by running the solution using localhost:[port]/Button as the URL.
5. Add Images for Buttons:
   1. Right click on your Project, select the Add->New Folder menu options, and create a new Folder called Images.
   2. Go to Google Images and search for two Button Images. Using Microsoft Paint (or any installed image editor), resize the button images to 50x50 pixels.
   3. Copy and paste the image files from File Explorer into your Projects Images folder.
6. Using the HTML Label Helper:
   1. Create two labels using the Html.Label():
      1. Label 1: Welcome to Razor, centered in bold, 24-point Arial font.
      2. Label 2: Play with Buttons, left aligned in bold, 18-point Arial font.
   2. To test your View, refresh your browser when you update your View.
7. Display Image Buttons in the View:
   1. Create static class scoped member variable named buttons as a List of type ButtonModel.
   2. In the Index() of the Button Controller add two ButtonModels to the buttons list. Pass this list as model data to the Button View.
   3. At the top of the Button View, change the model to:

@model List<Activity2Part1.Models.ButtonModel>

* 1. Using the Html.BeginForm() Razor method, create a Form that posts to the OnButtonClick method in the Button Controller.
  2. Add two HTML buttons with dynamic Images to the Form:

<button type=”submit” name=”mine” value=”1”>

@if(Model[0].State) <img src=”~/Images/On.png” />

else <img src=”~/Images/Off.png” />

</button><br/>

<button type=”submit” name=”mine” value=”2”>

@if(Model[1].State) <img src=”~/Images/On.png” />

else <img src=”~/Images/Off.png” />

</button>><br/>

1. Update the Button Controller:
   1. Add a new public method OnButtonClick that returns an ActionResult and takes a single method argument named mine of type string.
   2. In the OnButtonClick method implementation, check to see whether the mine argument is a value of “1” by toggling the ButtonModel state in the first element of the buttons list. If the mine argument is a value of “2,” then toggle the ButtonModel state in the second element of the buttons list. Return the Button view, and pass the buttons list model to the view.
2. Test your View:
   1. Validate the Button Page by running the solution using localhost:[port]/Button as the URL.
   2. Toggle the buttons and take at least two screenshots showing different states of your button images. Label these Screenshot 4 and 5.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Create a project report using a GCU standard project header/cover page to include a header, your name, course, assignment name, and date.
2. For Section 1, add screenshots of the following to your project report:
   1. Screenshot 1 of the HTML Label Helper
   2. Screenshot 2 of the HTML Action Link Helper
   3. Screenshot 3 of the HTML Table rendering a list of Users
3. For Section 2, add screenshots of the following to your project report:
   1. Screenshots 4 and 5 of at least two different button states
4. Upload your code to GIT, and include the URL of your GIT repository in the project report.

## Part 2: Advanced .NET Razor

**Overview**

In this activity, students will use .NET MVC Views to learn Razor Layouts. They will also learn how to use the NuGet Packet Manager to add Bootstrap and jQuery to their existing activity.

**Execution**

Execute this activity according to the following guidelines:

1. We use the code from .NET MVC Application from the Activity 2 Part 1. Open the Activity 2 Part 1 Solution.
2. Create a Default Layout Page:
   1. Inspect the Views menu, and if a shared folder does not exist, complete the following: Right click on the Views folder. Select the Add->New Folder menu options. Create a folder called Shared. This folder will be used to hold your shared layout files.
   2. Right click on the Shared folder and select the Add->New Item menu options. In the Add New Item dialog, select the Web->Razor options. Then select the Layout Page option from the right pane. Then name your layout \_MyDefaultLayout.cshtml. Click the Add button. Validate that your new Layout Page was created in the Shared folder.
   3. Create a Layout Page with the following content:

<!DOCTYPE html>

<html>

<head>

<title>@ViewBag.Title</title>

</head>

<body>

@RenderPage("\_Header.cshtml")

@RenderBody()

<hr/>

@RenderPage("\_Footer.cshtml")

</body>

</html>

1. Create Header and Footer Pages:
   1. Right click on the Shared folder and select the Add->New Item menu options. In the Add New Item dialog, select the Web->Razor options. Then select the Web Page option from the right pane. Then name your layout to \_Header.cshtml. Click the Add button. Validate that your new Header Page was created in the Shared folder.
   2. Remove all the content from the Header Page, and replace it with the following content:

<div align="center">

<h2>Welcome to our Activity</h2>

</div>

* 1. Right click on the Shared folder and select the Add->New Item menu options. In the Add New Item dialog, select the Web->Razor options. Then select the Web Page option from the right pane. Then name your layout to \_Footer.cshtml. Click the Add button. Validate that your new Footer Page was created in the Shared folder.
  2. Remove all the content from the Footer Page and replace with the following content:

<style>

.footer {

text-align: center;

position: absolute;

bottom: 0;

width: 100%;

}

</style>

<div class="footer">

<h5>Copyright &copy; My Own Company Name</h5>

</div>

1. Update the Test View to use the new Layout Page:
   1. Remove all top-level html, header, and body tags.
   2. Update the Layout Specification to:

@{

ViewBag.Title = “My App”;

Layout = "~/Views/Shared/\_MyDefaultLayout.cshtml";

}

* 1. Validate the Test Page by running the solution using localhost:[port]/Test as the URL.
  2. Take a screenshot. Label this Screenshot 6.

1. Add Bootstrap and jQuery to your Application:
   1. Right click on your project and select the Manage NuGet Packages menu item.
   2. Select the Browse menu from the NuGet Package Manager dialog.
   3. In the search box, enter Bootstrap. Select the ‘bootstrap by Twitter’ package. Then click the Install button to add Bootstrap (and jQuery) into your Solution. Validate by inspecting the Content folder and Scripts folder in your Solution.
   4. NOTE: This same process can be used to update any packages in your Solution, but rather than selecting the Browse menu, you select the Updates menu from the NuGet Package Manager dialog.
   5. Update the Layout Page to add the Bootstrap and jQuery libraries:

<!DOCTYPE html>

<html>

<head>

<title>@ViewBag.Title</title>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link href="~/Content/Site.css" rel="stylesheet" type="text/css" />

<link href="~/Content/bootstrap.min.css" rel="stylesheet" type="text/css" />

<script src="~/Scripts/jquery-1.10.2.min.js"></script>

<script src="~/Scripts/bootstrap.min.js"></script>

</head>

<body>

@RenderPage("\_Header.cshtml")

<div class="container body-content">

@RenderBody()

</div>

<hr />

@RenderPage("\_Footer.cshtml")

</body>

</html>

* 1. Update the Header Page to add a Menu Bar—similar code can also be found from our first MVC app built from Activity #1 in the \_layout.cshtml file:

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink((string)ViewBag.Title, "", "Test", new { area = "" }, new { @class = "navbar-brand" })

<a href="<http://www.gcu.edu>" class="navbar-brand" target="\_blank">GCU</a>

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav navbar-right">

<a href="<http://www.google.com>" class="navbar-brand" target="\_blank">Google</a>

<a href="<http://www.yahoo.com>" class="navbar-brand" target="\_blank">Yahoo</a>

</ul>

</div>

</div>

</div>

* 1. Resize your browser. Notice how your application is now responsive.
  2. Validate the Test Page by running the solution using localhost:[port]/Test as the URL.
  3. Take a screenshot. Label this Screenshot 7.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Updates to the project report created in Part 1.
2. Add screenshots of the following to your project report:
   1. Screenshot 6 of the Test View rendered with a layout, header, and footer.
   2. Screenshot 7 of the Test View rendered with Menu Bar using Bootstrap.
3. Upload your code to GIT, and include the URL of your GIT repository in the project report.

## Part 3: Data Validation

**Overview**

In this activity, students will use .NET MVC Data Validation Framework to add validation logic to your Object Model.

**Execution**

Execute this activity according to the following guidelines:

1. Use the code from .NET MVC Application from Activity 1 Part 3. Open the Activity1Part3 Solution.
2. Add Data Validation Rules to the User Model:
   1. Add the following Data Annotations above the Username Property:

[Required]

[DisplayName("User Name")]

[StringLength(20, MinimumLength = 4)]

[DefaultValue("")]

* 1. Add the following Data Annotations above the Password Property:

[Required]

[DisplayName("Password")]

[StringLength(20, MinimumLength = 4)]

[DefaultValue("")]

1. Update the Login Controller to handle Data Validation:
   1. Add the following code to the Login Controller prior to calling the Business Service:

// Validate the Form POST

if (!ModelState.IsValid)

return View("Login");

1. Inspect the Login View for Data Validation Razor Tags:
   1. Note the use of the Html.ValidationMessagesFor() Razor method.
2. Test the Data Validation Rules:
   1. Validate the Login Page by running the solution using localhost:[port]/Login as the URL.
   2. Take a screenshot of validation errors on each field of the Login Form. Label this Screenshot 8.
   3. Take a screenshot of a successful login. Label this Screenshot 9.

**Documentation**

All documentation will be submitted at the end of the activity to the learning management system. Ensure documentation of the following:

1. Updates to the project report created in Part 1.
2. Add screenshots of the following to your project report:
   1. Screenshot 8 of the Login View rendered with data validation errors.
   2. Screenshot 9 of the Login response rendered with a successful login.
3. Upload your code to GIT, and include the URL of your GIT repository in the Project Report.

## Part 4: Practicing Agile Scrum

**Overview**

In this activity, students will practice Scrum and work on their team project. They will emulate a “compressed” 3-week Sprint over the course of this week as well as perform Scrum Daily Standups, Sprint Planning, a Burn Down Chart, and a Retrospective on their team project.

**Execution**

Execute this assignment according to the following guidelines:

1. This week will be used for working on your milestone project.
2. You are expected to complete Daily Standups, Sprint Planning, a Sprint Burn Down Chart, and Retrospective.
3. Post the following in a Word document for this activity to LoudCloud:
   1. Results of your Daily Standup (who did what over the course of the week)
   2. Sprint Plan for this week
   3. Sprint Burn Down Chart for this week
   4. Results of Retrospective for this week

**Submission**

Submit the following to the learning management system:

**Parts 1-3**

1. Project Report that includes:
   1. A GCU standard project header/cover page to include your name, course, assignment name, and date
   2. Screenshots 1-9
   3. URL of your GIT repository
2. Upload your code to the GIT repository.

**Part 4**

1. A Word document to include:
   1. Results of your Daily Standup (who did what over the course of the week)
   2. Sprint Plan for this week
   3. Sprint Burn Down Chart for this week
   4. Results of Retrospective for this week