**CST-247 Activity Guide**

Contents

[Activity 3 2](#_Toc60388405)

[Part 1: AJAX, Partial Views, and Partial Page Updates 2](#_Toc60388406)

[Part 2: Visual Studio Debugger 4](#_Toc60388407)

# Activity 3

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

## Part 1: AJAX, Partial Views, and Partial Page Updates

**Overview**

In this activity, students will learn how to build Razor Forms, AJAX enabled Razor Forms, Partial Views, and Partial Page Updates.

**Execution**

Execute this activity according to the following guidelines:

1. Create a Model named CustomerModel with the following properties. Also create a nondefault constructor that initializes all the class properties.

int ID

string Name

int Age

1. Create a default Customer Controller.
2. Create an empty Customer View with no layout and no model.
3. Create a Layout page \_Layout in the Shared directory that contains the following:
   1. @Renderpage("\_Header.cshtml"). Create this Page so it simply displays the following two lines:
      1. Title 'Welcome to Activity 3 Part 1' in H2 formatting.
      2. Current Time 'Current Time is @DateTime.Now in H2 formatting.
   2. @RenderBody()
4. Create a Partial page \_CustomerDetails in the Shared directory that contains the following:
   1. A Model declaration to use the CustomerModel.
   2. Title 'Customer Details' using a Razor Label.
   3. @Html.DisplayNameFor Razor Bind to the Model's Name property.
   4. @Html.DisplayNameFor Razor Bind to the Model's Age property.
5. Modify the Customer Controller:
   1. Create a class scoped member variable ‘customer’ that is a List of type CustomerModel.
   2. Create a default constructor method. Create a List of Customer Model object models within the constructor implementation.
   3. Update Index() to create a C# Tuple with a customer list and first customer from the list as data then pass the Tuple to the Customer View.
   4. Create OnSelectCustomer() that takes a single string method argument (the customer from the Radio Button value) and that creates a C# Tuple with a customer list and customer passed in customer ID from the list as data, then pass the Tuple to the Customer View. The method should conform to an HTTP Post method.
6. Create a non AJAX Form without Partial Page updates:
   1. Change the Model declaration to use the Tuple created in the previous step.
   2. Change the HTML page title to a value set in the ViewBag.Title.
   3. Change the Layout to the layout created in the previous step.
   4. Add a Label '“Please select a customer”' using a Razor Label.
   5. Create a Razor Form that calls the CustomerController.OnSelectCustomer1, that uses a POST, and whose form ID is ‘myform.’
   6. Iterate over the Tuple Item 1 (list of Customer Models) to add a Razor Radio Button for each customer setting the name of the Radio Button group to Customer, the Customer ID as its value, with an onchange handler that submits the ‘myform.’ Also use a Razor Label tag to display the customer name.
   7. Add Partial page using @Html.Partial("\_CustomerDetails", ModelItem2) specifying the Customer Details Partial View and Tuple Item 2 (CustomerModel). Place this Razor tag inside a div tag with an ID of ‘customerInfo.’
   8. Run the application:
      1. Select each of the Customers, and make sure Customer Details Partial Page gets updated.
      2. Note how the time changes, indicating a full-page refresh.
      3. Take a screenshot. Label this Screenshot 1.
7. Create an AJAX Form with Partial Page updates:
   1. Using NuGet, make sure support for Microsoft jQuery Unobtrusive Ajax is installed (search for unobtrusive).
   2. In the \_Layout, add two script references in the HTML header to jquery.js and jquery.unobtrusive-ajax.js.
   3. Create a Razor variable ‘ajaxOptions’ at the top of the Customer View that creates in instance of AjaxOptions that sets the following properties:

HttpMethod = “POST”

InsertionMode = InsertionMode.Replace

UpdateTargetId = “customerInfo”

* 1. Change the Html.BeginForm to Ajax.BeginForm and insert the ‘ajaxOptions’ into the third parameter of the AjaxBegin.Form().
  2. Update the Customer Controller by changing the implementation of the OnSelectCustomer() to returns a ‘\_CustomerDetails’ PartialView with data of a CustomerModel obtained from the list of Customers.
  3. Run the application:
     1. Select each of the Customers, and make sure Customer Details Partial Page gets updated.
     2. Note how the time DOES NOT change, indicating no full-page refresh.
     3. Take a screenshot. Label this Screenshot 2.

1. Create an AJAX JavaScript callback handler:
   1. Update the Customer Controller adding a new controller method GetMoreInfo() that takes a string customer ID as an argument, returns a string, and conforms to a HTTP Post method. The method can just return any desired string value.
   2. Update the Razor variable ‘ajaxOptions’ at the top of the Customer View with the following property:

OnSuccess = “ajaxSuccess”

* 1. Create an inline script in the View that makes a jquery post request to Customer Controller to the GetMoreInfo().

$.post('@Url.Action("GetMoreInfo", "Customer")', { customer: 2 }, function (data) { alert(data); });

* 1. Run the application:
     1. Select each of the Customers, and make sure Customer Details Partial Page gets updated, as well as the AJAX callback alert displayed in JavaScript with your desired message.
     2. Take a screenshot. Label this Screenshot 3.

**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. Add screenshots of the following to your project report:
   1. Screenshot 1 of the Non AJAX Form
   2. Screenshot 2 of the AJAX Form with Partial Page Updates
   3. Screenshot 3 of the AJAX JavaScript Alert callback
3. Upload your code to GIT and include the URL of your GIT repository in the Project Report.

## Part 2: Visual Studio Debugger

**Overview**

In this activity, students will learn how use the Visual Studio Debugger. Use the code from Activity 2 Part 3.

**Execution**

Execute this assignment according to the following guidelines:

1. Open the solution for Activity 2 Part 3.
2. Setting Breakpoints:
   1. Open the LoginController class.
   2. Set a breakpoint on the ModelState.IsValid() line in doLogin().
   3. Set a breakpoint on the SecurityService service = new SecurityService() line in doLogin().
   4. Run the application in debug mode with known good log in credentials.
   5. Take a screenshot of a breakpoint from Step 2a. Label this Screenshot 4.
   6. Click the Continue button.
   7. Take a screenshot of a breakpoint from Step 2b. Label this Screenshot 5.
   8. Click the Continue button.
3. Inspecting Variables:
   1. Set the same breakpoints from Step 2a and Step 2b.
   2. Run the application in debug mode with known good log in credentials.
   3. Inspect the UserModel argument passed to the doLogin() by hovering over the variable.
   4. Take a screenshot from Step 3c. Label this Screenshot 6.
   5. Inspect the variables in the Locals Window.
   6. Take a screenshot from Step 3e. Label this Screenshot 7.
   7. Remove all breakpoints.
4. Stepping into a Function, over a Function, and out of a Function:
   1. Open up the LoginController class.
   2. Set a breakpoint on the service=Authenticate(model) line in doLogin().
   3. Run the application in debug mode (with known good log in credentials).
   4. Verify that code stopped at the breakpoint set in Step 4b.
   5. Click the Step Into icon from the debugger toolbar or enter the F11 key.
   6. Inspect the UserModel argument passed to the Authenticate() by hovering over the variable.
   7. Click the Step Over icon from the debugger toolbar or enter the F10 key. Continue stepping over code statements until the line of code that creates an instance of the SecurityDAO().
   8. Take a screenshot from Step 4g. Label this Screenshot 8.
   9. Open up the SecurityDAO class.
   10. Set a breakpoint on the line of code in the FindByUser() that calls ExecuteReader().
   11. Click the Continue button. Verify the code stopped at the breakpoint set in Step 4j.
   12. Take a screenshot from Step 4k. Label this Screenshot 9.
   13. Step through the code using Step Over until the return statement is reached.
   14. Click the Step Over icon from the debugger toolbar or enter the Shift F11 key until the return value from the Authenticate() has been set.
   15. Click the Step Over icon from the debugger toolbar or enter the Shift F11 key.
   16. Inspect the Authenticate() return value by hovering over the variable.
   17. Take a screenshot from Step 4p. Label this Screenshot 10.
   18. Click the Continue button.
5. Setting Conditional Breakpoints:
   1. Remove all breakpoints (see option from the Debug menu).
   2. Open up the LoginController class.
   3. Set a breakpoint on the line after the service=Authenticate(model) line in doLogin() and where the return value has been set.
   4. Click the “cog” that will be displayed when you hover over breakpoint “dot.”
   5. Click the Conditions checkbox. Set a conditional statement when the return value from Authenticate() is true. Uncheck the Actions checkbox. Click the close button.
   6. Run the application in debug mode.
   7. Verify that code stopped at the breakpoint set in Step 5e.
   8. Take a screenshot from Step 5g. Label this Screenshot 11.
   9. Click the Continue button.
   10. Click the “cog” that will be displayed when you hover over breakpoint “dot.”
   11. Leave the Condition checkbox checked. Check the Actions checkbox. Enter a message “I am here!!” in the ‘Log a message to Output Window.’ Check the Continue execution checkbox. Click the close button.
   12. Run the application in debug mode.
   13. Verify the message “I am here!!” got logged in the Output Window.
   14. Take a screenshot from Step 5m. Label this Screenshot 12.
   15. Click the Continue button.
6. Inspecting the Call Stack:
   1. Open up the SecurityDAO class.
   2. Set a breakpoint on the line of code in return statement in the FindByUser().
   3. Run the application in debug mode.
   4. Verify that code stopped at the breakpoint set in Step 6b.
   5. Inspect the Call Stack Window.
   6. Take a screenshot from Step 6e. Label this Screenshot 13.
   7. Click the Continue button.

**Extra Practice**

1. From the Analyze menu, select the Run Code Analysis menu option. Inspect the warnings returned from the analysis. Why should you fix the warnings in your code?
2. From the Analyze menu, select the Calculate Code Metrics menu option. Inspect the results returned from the analysis. Why is this an important tool to run on your code?

**Submission**

Submit the following to the learning management system:

1. Project Report that includes:
   1. A GCU standard project header/cover page to include your name, course, assignment name, and date
   2. Screenshots 1-13
   3. URL of your GIT repository
2. Upload your code to the GIT repository.