2Lab 4 – Razor and Request/Session

# Description

This lab is designed to give you a basic understanding of the Razor Rendering Engine and an overview of MVC’s routing feature.

# Estimated Time

This lab will take an estimated 4 hours to complete.

# Deliverable

* Push your code to GitHub and submit the link to Brightspace.
* Deploy your website to Windows Azure and submit the link to Brightspace.

See “Brightspace -> Course Content -> Extra Materials -> Microsoft Azure Web Application” for information about deploying Azure Web Apps.

# Notes

* Be sure Visual Studio is up to date.
* Follow along closely to the instructions!
* When using the <input> tag be sure to set the type to collect the proper values
* Use the example code ‘<https://github.com/afrasia/IntroToASP.NETMVC>’ for help and tips.
* Demo can be found at: <http://afrasialab4.azurewebsites.net/>

# Step 1: Create a new MVC Core project called ‘Lab3.’

1. Open Visual Studio 2022
2. Click: Create a new project
3. Select: C# -> Windows -> Web
4. Select ‘ASP.NET Core Empty’
5. Name the application ‘Lab4’, save the project in your desired location and click the button ‘Next.’
6. Select .NET 8.0 for Target Framework
7. Uncheck configure for HTTPS.
8. Check Do not use top-level statements.
9. Click the ‘Create’ button.

# Step 2: Configure your new Web Application

1. Replace the program class with: <https://gist.github.com/afrasia/dfaf0f134323495ee56656b1914b2ea5>
   1. Make sure you replace the /\* … \*/ with the proper values

# Step 3: Create the ‘Controllers’ and ‘Views’.

1. Create a folder in your project called ‘Controllers’.
   1. Right click on the project in solution explorer, then ‘Add’, then ‘New Folder’
2. Create a new Controller in this folder called ‘Home’ – Note that your class should be named ‘HomeController.cs’
   1. To do so:
      1. Right click on the ‘Controllers’ folder
      2. Click ‘Add’
      3. Click ‘Controller...’
      4. Select the ‘MVC Controller - Empty’ and click ‘Add.’
      5. Select ‘MVC Controller – Empty’, type the right name, and click ‘Add.’
   2. Add the following code to the body of your controller:

public IActionResult SongForm() => View();

[HttpPost]

public IActionResult Sing()

{

// you will complete this

}

public IActionResult CreateStudent() => View();

[HttpPost]

public IActionResult DisplayStudent(Student student)

{

// you will complete this

}

public IActionResult Error()

{

return View();

}

1. There will be two syntax errors, on Sing and DisplayStudent methods. Ignore them for now. You will fix them later.
2. Create a folder in your project called ‘Views’.
3. At the root of this new ‘Views’ folder, create a file called ‘\_ViewImports.cshtml’. You can use the following steps:
   1. Right click on the ‘Views’ folder
   2. Click ‘Add’
   3. Click ‘New Item...’
   4. Select ‘Razor View Imports’, and click ‘Add.’
4. Add the following lines of code to the file ‘\_ViewImports.cshtml’

@using Lab4

@addTagHelper "\*, Microsoft.AspNetCore.Mvc.TagHelpers"

1. Follow similar steps and add ‘\_ViewStart.cshtml’, and do not change the content of the created file.

# Step 4: You are on your own (Part 1)

1. Create wwwroot directory on the root of the project.
2. Create your own StyleSheet.css, or reuse what you find in my sample code.
   1. You can find it in my sample code under, wwwroot->css
3. Create a \_Layout.cshtml file.
   1. Check the lecture to decide where it must go.
   2. Then you can add the file that by doing Add->New Item…->Razor Layout
4. Inside the layout file:
5. Add a header and footer.
6. In header, put:
   1. Your name and your welcome message
7. In the footer, use HTML tags/style of your choice to explain steps you took to add CSS file and \_Layout. Should include the following:
   1. Short description of what layout does
   2. Why do you put the files in those directories. E.g., CSS in directory x, and why. Layout in directory Y and why. Etc.
   3. You should not include the content of CSS or how you defined your CSS.
   4. Open \_ViewStart.cshtml and specify the layout you created as default layout for all views (you are on your own)

# Step 5: You are on your own (Part 2)

1. Create a folder called Home under Views
2. Create a View in Views/Home called ‘Index.cshtml’
   1. Right click on Home, then Add->View…->Razor View – Empty
   2. Click Add
   3. Select Razor View – Empty and type the right name.
   4. Click Add
   5. Add some text or HTML welcome message to this view.
3. Create a View in Views/Home called ‘SongForm.cshtml’
4. Create a loosely typed view called ‘Sing.cshtml’
5. SongForm view should contain an MVC form that gets the number of monkeys from user and passes them down to the action called ‘Sing’ in HomeController
   1. Make sure you limit the user to the inclusive range of 50 to 100 monkeys.
6. Action Sing in HomeController, must store the number of monkeys in session or ViewData/ViewBag
7. Action Sing in HomeController, will call the view ‘Sing’ with the number of monkeys still stored in session or ViewData/ViewBag
8. In Sing view, use Razor to draw the lyrics to the children’s song ‘Five little monkeys…’ (see: <https://en.wikipedia.org/wiki/Five_Little_Monkeys> )
   1. Use the value user entered for the monkeys in SongForm view, and you already stored in session or ViewData/ViewBag
   2. Note that the plural/singular forms of ‘monkeys’ should be correct when you count down to 1, and when there are no monkeys left.
   3. Alternate between “he” and “she” for monkeys. For odd number monkeys we use he and for even ones we use she.
   4. When the last monkey falls off, we do not use “One fell off…”. See the sample.
   5. Pay attention to the part where there are no monkeys left.
   6. What the doctor says must be in quotes.
9. Add a MVC Form in Sing view, with a button to let the user play again.
   1. Play Again, should take the user back to the SongForm view.
10. Add a hyperlink in the layout header that points to the SongForm view.

@Html.ActionLink("Show the song!", "SongForm", "Home")

1. Add a hyperlink in the layout header that points to the Index view.

@Html.ActionLink("Home", "Index", "Home")

# Step 6: You are on your own (Part 3)

1. Create a folder in your project called ‘Models’.
2. Create a model for Student, called Student.cs, under the ‘Models’ folder.
   1. You can do that by right clicking on Models and adding a new ‘Class’.
3. Student has the following properties:
   1. First Name -- string
   2. Last Name -- string
   3. StudentId -- int
   4. Email Address -- string
   5. Password -- string
   6. Description of Student -- string
4. Create a view called ‘CreateStudent.cshtml’ under Views->Home.
5. Add a MVC Form in CreateStudent and collect the following information from the user.
   1. First Name
   2. Last Name
   3. StudentId
   4. Email Address
   5. Password (should be masked as you type, and it should not be shown to the user on the DisplayStudent.cshtml – see below)
   6. Description of Student
6. Once collected, post the data back to DisplayStudent action in HomeController
7. Create a view called ‘DisplayStudent.cshtml’
   1. Make this view a strongly typed view, using the student model.
8. Pass the Student data from DisplayStudent action to DisplayStudent view.
9. Display the collected results on DisplayStudent view.
10. Add a hyperlink in the layout header that points to the CreateStudent view.

@Html.ActionLink("Students", "CreateStudent", "Home")

# Step 7: You are on your own (Part 4)

Create a view called ‘Error’ under Views->Home and put a custom error message (some text or HTML) there.

# Marking Scheme

|  |  |  |
| --- | --- | --- |
| task | mark |  |
| Program.cs | 2 |  |
| folder structures | 1 |  |
| Error page | 2 |  |
| css | 1 |  |
| homecontroller | 6 |  |
| viewimports | 1 |  |
| viewstart | 1 |  |
| layout | 4 |  |
| SongForm View | 2 |  |
| Sing View | 5 |  |
| Create Student | 3 |  |
| Display Student | 4 |  |
| Student Model | 3 |  |
| gitignore | 2 |  |
| publishing | 3 |  |
| working end-to-end solution | 5 |  |
|  |  |  |
|  |  |  |
| Total | 45 |  |