Module 7: Validation

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2020 Microsoft Corporation. All rights reserved.

Copyright and Trademarks

© 2020 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at
http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx

Internet Explorer, Microsoft, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Lab 07: Validation

Introduction

The aim of this exercise is to explore MVC Validation, by adding some new features to our MyShuttle web application.

Objectives

After completing this lab, you will be able to:

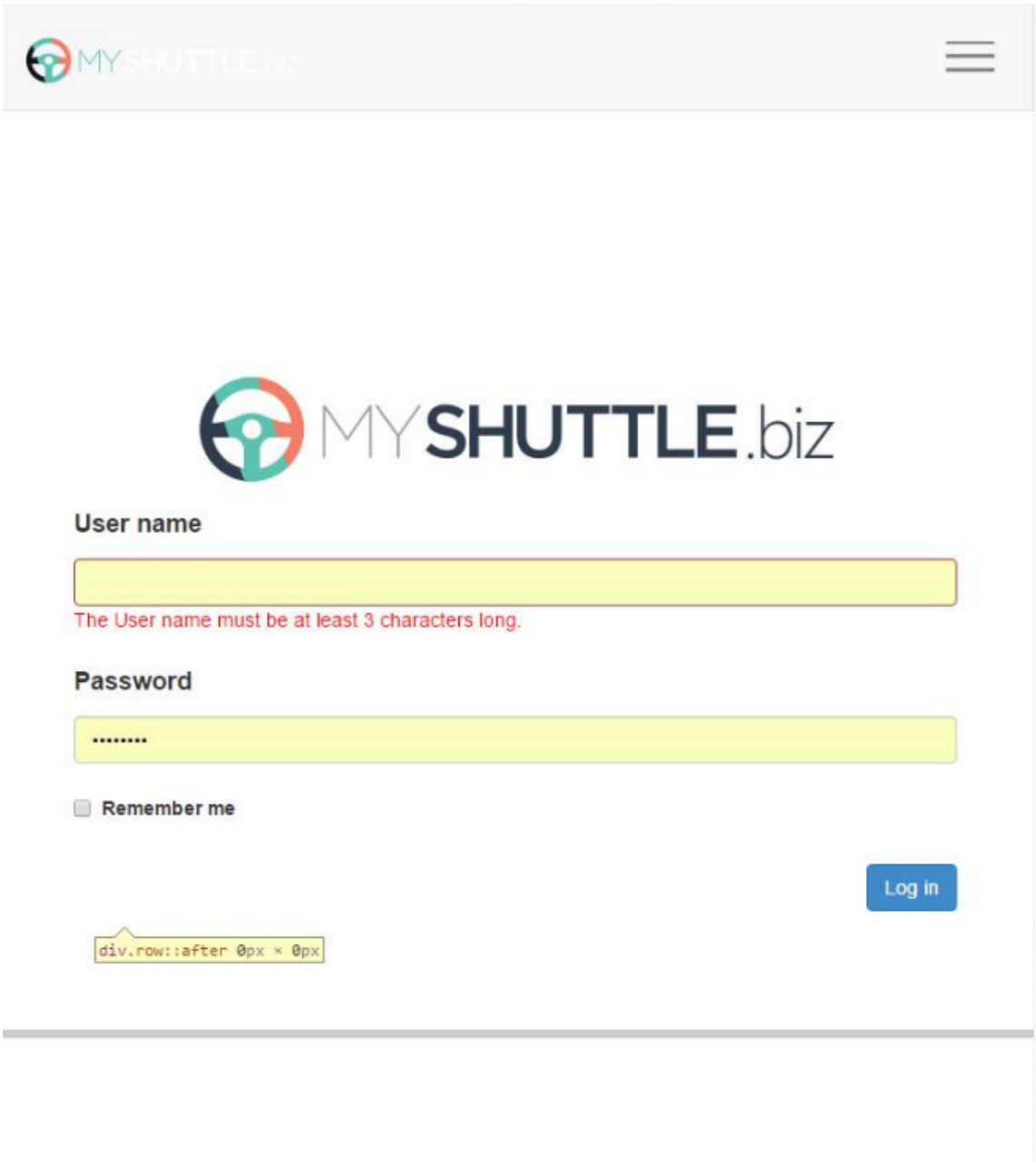
- Leverage client-side validation.
- Understand Data Annotation attributes.

Prerequisites (if applicable)

This lab will continue building on the previous labs, adding features to the MyShuttle web application. You can choose either to continue with the solution you ended up with following the previous lab, or start with the solution included in the folder **Labs\Module 07 – Validation\Begin**. No additional SDKs or tools are required for this lab.

Scenario

There is a requirement to add a login page to the application, which will look like the screenshot below. Part of the functionality required is to ensure that the user can only enter data that is valid according to the specification, and display an appropriate error message to the client while the input is not valid. This scenario does not include login Authorization login that is part of lab 8.



System Requirements

To complete this lab, you need:

- Microsoft Visual Studio 2019 or higher
- Microsoft SQL Server (any edition)

Estimated time to complete this lab

20 minutes

Exercise 1: Add Login and Register Pages

Objectives

In this exercise, you will:

- Add new model classes for use within the Login and Register Views.
- Implement the CarrierController and its Login methods (Get and Post).
- Create the Login and Register views.

Scenario

Our specification says that our UI must display a **User name**, **Password**, and the **Remember me** checkbox. When registering, the user must also enter a confirmation password that has to match the other password field on the form.

The Password can be up to 100 characters long, but must be at least six. The Username must be at least three characters long, but cannot exceed 30.

Task 1: Create Models, Views, and Controller

1. Add a new model class to the MyShuttle.Web project:
    - Locate the **Models** folder in the **MyShuttle.Web** project
    - Right-click the Models folder and select **Add > New Item**
    - Select **"Class"** from the list, and give it the name **"AccountViewModels.cs"**
    - Click **Add**
2. Replace the boilerplate code in **AccountViewModels.cs** with the code below:

```
using System.ComponentModel.DataAnnotations;

namespace MyShuttle.Web.Models
{
    public abstract class BasePasswordModel
    {
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [Display(Name = "User name")]
        public string UserName { get; set; }
    }

    public class LoginViewModel : BasePasswordModel
    {
        [Display(Name = "Remember me?")]
        public bool RememberMe { get; set; }
    }

    public class RegisterViewModel : BasePasswordModel
    {
        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        public string ConfirmPassword { get; set; }
    }

}
```

```
The class file now contains an abstract class (**BasePasswordModel**) and two concrete
classes: **LoginViewModel** and **RegisterViewModel**.

We have used data annotations to modify the data type (which will affect how the HTML
is rendered) and the display names for the labels, but no validation support has been
added yet. We will add those later.
```

3. Now add the **CarrierController**:
    - Right-click the **Controllers** folder, select **Add > New Controller**
    - Select **MVC Controller - Empty** and name it **"CarrierController"**
    - Click **Add**
4. In the new **CarrierController.cs** file, add the following two *using* statements, and **delete** the Index() method that's in the class:

```
using Microsoft.AspNetCore.Authorization;
using MyShuttle.Web.Models;
```

5. Add the following new methods to the **CarrierController.cs**

    The following code will implement handlers for displaying the two views, and processing form data posted back to the controller from the views:

```
[HttpGet]
[AllowAnonymous]
public IActionResult Login(string returnUrl = null)
{
    ViewBag.ReturnUrl = returnUrl;
    return View();
}

[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl
= null)
{
    try
    {
        if (ModelState.IsValid)
        {
            return RedirectToAction("Index", "Home");
        }

        else
        {
            return RedirectToAction("Register", "Carrier");
        }
    }
    catch (System.Exception ex)
```

```
        {
        }

        // If we got this far, something failed, redisplay form

        return View(model);
    }

    [HttpGet]
    public async Task<IActionResult> LogOff()
    {
        return RedirectToAction("Index", "Home");
    }

    private IActionResult RedirectToLocal(string returnUrl)
    {
        if (Url.IsLocalUrl(returnUrl))
        {
            return Redirect(returnUrl);
        }
        else
        {
            return RedirectToAction("Index", "Home");
        }
    }

    [HttpGet]
    [AllowAnonymous]
    public IActionResult Register()
    {
        return View();
    }

    [HttpPost]
    [AllowAnonymous]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Register(RegisterViewModel model)
    {
        try
        {
            if (ModelState.IsValid)
            {
                return RedirectToAction("Login", "Carrier");
            }
            else
            {
                return RedirectToAction("Register", "Carrier");
            }
        }
        catch (System.Exception ex)
        {
        }
```

```
    return RedirectToAction("Register", "Carrier");
}
```

6. The above code uses **ValidateAntiForgeryToken** to ensure that the model being posted back to the controller comes from the same client to which the view was posted, preventing cross-site request attacks. The **ModelState.IsValid** check is used to see if any errors have been added to the model state during serialization from the client. Otherwise, the code is straightforward and simple to follow.

7. Now to add the Carrier Views to the MyShuttle.Web project:

    - Find the **Views** folder right-click it and select **Add > New Folder**,
    - Name the folder **Carrier** (to match the controller name).
    - Copy two view files from the **Assets** folder to Views/Carrier folder of this Lab's files:
        - **Login.cshtml**
        - **Register.cshtml**

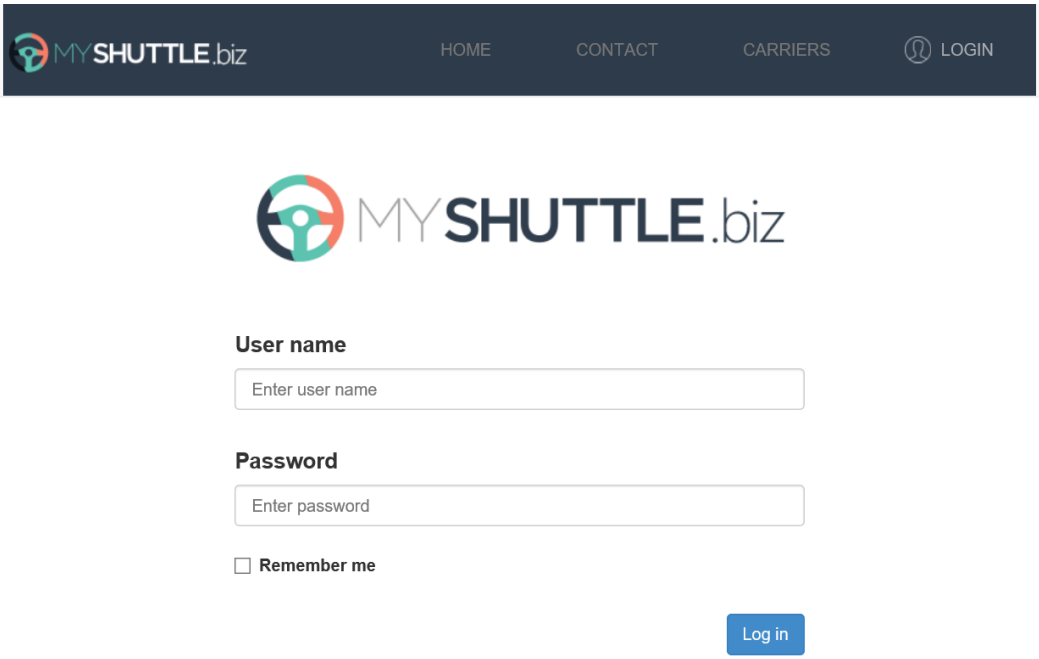
8. Change the shared **Views / Shared / _Layout.cshtml** view to point the Login command to the correct controller endpoint (This should be roughly *around* line 65. Note the change of page from **Index** to **Login**.).

```
@Html.ActionLink("Login", "Login", "Carrier", null, new { @Class = "login-item" })
```

9. At this point, **build** and **run** the app.

    Click the **Login** button and you will see the login form appear, but there is no validation – you can enter any text, click **OK** and it will return you to the homepage.

    Stop the debugger, and we will now take the steps to add in the validation.



Task 2: Adding validation support to the model

1. The password and username fields are mandatory. Therefore, we need to add the annotation **[Required]** above those properties. If the user omits the field, we want to display an error message telling them that it is a required field.

   Open the the **AccountViewModels.cs** file. In the **BasePasswordModel** class, add the following attribute to the **Password** property:

   ```
   [Required(ErrorMessage = "Password is required")]
   ```

2. And to the **UserName** property:

   ```
   [Required(ErrorMessage = "User name is required")]
   ```

3. To limit the length and set minimum length of the fields, we will add the **StringLength** attribute to the **Password** property:

   ```
   [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters
   long and no more than 100", MinimumLength = 6)]
   ```

4. And the **UserName** property:

   ```
   [StringLength(30, ErrorMessage = "The {0} must be at least {2} characters long
   and no more than 30", MinimumLength = 3)]
   ```

5. Finally, the **ConfirmPassword** field needs to be validated to make sure that it matches the original password. In the **RegisterViewModel** class in AccountViewModel.cs file, add the following attribute to the **ConfirmPassword** field,:

   ```
   [Compare("Password", ErrorMessage = "The password and confirmation password do
   not match.")]
   ```

   When done the *AccountViewModels.cs* file should look like:

Task 3: Adding validation support to the view

1. Open the **Login.cshtml** file from the **Views/Carrier** folder.

2. Locate the form-group containing the **UserName** property. We see a validation message after the input control – when the user edits this input control, then leaves it (by entering another field or submitting the form), the client side validator will execute, and if the input is invalid, the message will then be displayed.

   ```
   @Html.ValidationMessageFor(x => x.UserName)
   ```

3. Look for the **Password** property, so you end up with the two input groups looking like this:

4. We will now add in a validation summary, which will show a list of errors found in the **ViewData's ModelState**. Below the **@Html.AntiForgeryToken** line, add the following markup:

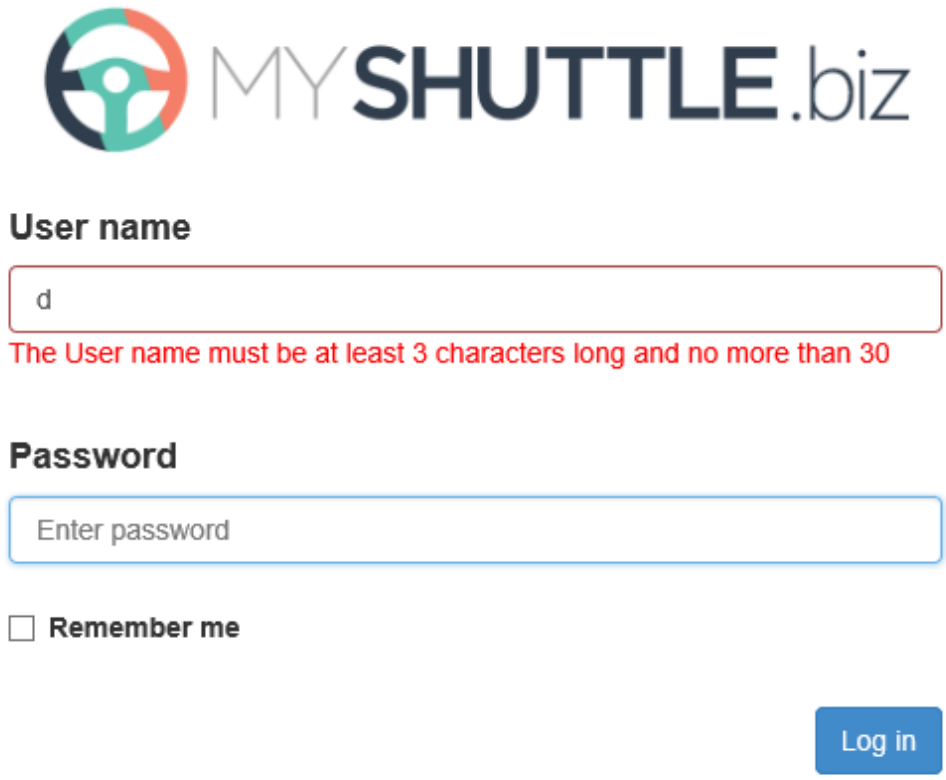
   ```
   <div class="form-group login_message_summary">
   ```

```
     @if (ViewData.ModelState[""] != null &&
ViewData.ModelState[""].Errors.Count() > 0)
     {
         <div class="alert alert-dismissable alert-danger">
             <button class="close" type="button" data-
dismiss="alert">×</button>
             @Html.ValidationSummary(true)
         </div>
     }

 </div>
```

5. Now run the application, go to the login view. Click in the **User name** field, type one character, then press **Tab** key to move the cursor to the **Password** field. You should see the error message appear immediately, and similarly for the Password field.

   Note that the error message will be clear as you type after the field is valid, since the validation is all executed on the client using JavaScript.



Note that although we have created the forms for login and register, there is no functionality behind

the controller yet to actually log in or register a user.