Lesson 05 jQuery 2 and more EF Core

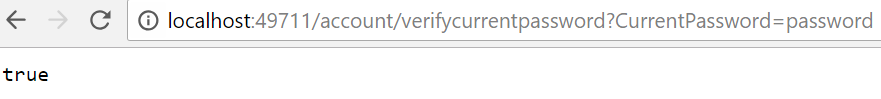
1. If you have not completed the project setup, please perform project setup following instructions in **C286 L05 Project Setup Guide.docx** first.
2. Open **Task List** window from the menu **VIEW**, **Task List**. Use the **Filter** function in the Project heading to display only the Task for Lesson05.
3. **Activity 1A:**
   * + 1. Perform **Tasks 1-1 to 1-2**.
       2. Refer to the **C286 L05 Student Notes** Section 1 slides to help you complete the task.
       3. Run the application and verify that the authentication functionality works as usual.
4. **Activity 1B:**
   * + 1. Perform **Tasks 1-3 to 1-4**.
       2. Refer to the **C286 L05 Student Notes** Section 1 slides to help you complete the task.
       3. Run the application and verify that the authentication functionality works as before.
5. **Activity 2:**
   * + 1. Perform **Tasks 2-1 to 2-3**.
       2. **Task 2-1** is in Views/MugOrder/Index.cshtml.
       3. Refer to the **C286 L05 Student Notes** slides to help you complete the task.
       4. Run the application, log in, and try to create and delete a mug order.
       5. Verify that you are prompted to confirm your deletion and you can cancel your delete by pressing the [Cancel] button in the confirmation dialog box.

# Solving the Problem

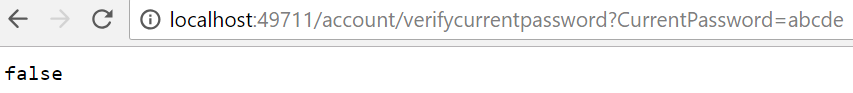
1. You are suggested to complete the solution in the following order:
   * + 1. Implement delete shirt order functionality.
       2. Complete the UI interactivity requirements in both Views/ShirtOrder/Create.cshtml and Update.cshtml (Hint: Refer to MugOrder):
          1. Create a jQuery function to calculate total.
          2. In document.ready function, ensure the jQuery function to calculate total is called whenever user changes price or quantity.
       3. Ensure update and add functionalities will go through the same confirmation procedure in delete shirt order functionality.
       4. Implement change password functionality:
          1. Implement **VerifyCurrentPassword** and **VerifyNewPassword** actions in **AccountController**. Follow the following steps to test out these 2 actions:

Login in as **deslee**

Test the **VerifyCurrentPassword** by setting the parameter **CurrentPassword=password**. You should receive **true** as the result as shown below:



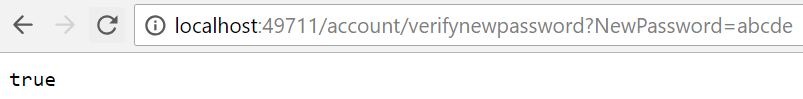
Test the **VerifyCurrentPassword** by setting the parameter **CurrentPassword=<Wrong password>**. You should receive **false** as the result as shown below:



Test the **VerifyNewPassword** by setting the parameter **NewPassword=password**. You should receive **false** as the result as shown below:



Test the **VerifyCurrentPassword** by setting the parameter **NewPassword=<Wrong password>**. You should receive **false** as the result as shown below:



* + - * 1. Create a model as follows:

Name the model class as **PasswordUpdate**

Create 3 string properties: **CurrentPassword**, **NewPassword** and **ConfirmPassword**

All 3 string properties has 2 common data annotation requirements:

DataType – must be password

Required – cannot be empty string

**CurrentPassword** property must be remotely validated through **Account** controller **VerifyCurrentPassword** action.

**NewPassword** property must be remotely validated through **Account** controller **VerifyNewPassword** action.

**ConfirmPassword** must be validated against the **NewPassword** property. *Hint: Use* ***Compare*** *data annotation attribute*

* + - * 1. Create **ChangePassword** HttpGet and HttpPost actions as follows:

**ChangePassword** HttpGet action:

Accept no parameters.

Return view, don’t need to pass in any model.

**ChangePassword** HttpPost action:

Accept a paramenter of type **PasswordUpdate**

Use **ExecuteSqlInterpolated** to update the password accordingly

If **ExecuteSqlInterpolated**  returns 1 then pass a successful message to view through **ViewData**.

If **ExecuteSqlInterpolated** returns other than 1 then pass a failure message to view through **ViewData**.

Return view, don’t need to pass in any model.

* + - * 1. Create **ChangePassword.cshtml** view as follows:

Strongly type the view to **PasswordUpdate** class.

Refer to problem statement appendix for the input and layout

Ensure all fields a validation messages displayed (Refer to P02 for validation).

Ensure messageds passed in from **ChangePassword** is displayed.

The form should post to **Account** controller **ChangePassword** http post action.

# Testing Your Solution

1. Test Cases:
   * + 1. Add a new shirt order and update the shirt order again. Ensure the confirm functionalities are working.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

* + - 1. Test the calculation of the subtotal.

Graphical user interface, application

Description automatically generated

* + - 1. Delete the shirt order you have added. . Ensure the confirm functionalities are working.

Graphical user interface, application

Description automatically generated

* + - 1. Try to change password and logout and login with the new password.

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Provide necessary screenshots here to prove you have successfully tested your web app.

*— End of Worksheet —*