jQuery Introduction

Estimated time for completion: **30 minutes**

Overview:

In this lab you will start to use jQuery. You'll handle the document ready event, query the DOM for elements, modify style properties, handle events and even perform some animations.

Goals:

- Get comfortable using basic selectors and functions from jQuery

Lab Notes:

The lab uses a movie review website as its base. Registered users can view movies and reviews. Reviewers can write reviews. Administrators can do most anything. The accounts (username and passwords are the same value) for the application are:

- **Registered Users:** user1, user2 and user3
- **Reviewers:** reviewer1, reviewer2 and reviewer3
- **Administrator:** admin

Getting started with the movie review application

You will run the movie review application to get familiar with the functionality and then add the scripts for jQuery.

Criteria:

- Run the movie review application
- Include jQuery in the application
- Handle the document ready event

Steps:

1. Open the solution from `~/jQueryIntro/before`.
    a. Notice it's an ASP.NET MVC 3 project. It should not be a problem if you are not experienced with MVC – you will mostly be working in JavaScript files and occasionally the markup (`.cshtml`/razor) files.

If you're curious feel free to explore the source code.

b. Get familiar with the application. From Visual Studio run the application (via Ctrl-F5) and browse the movies and reviews. As needed login to the application with the credentials above in the lab notes.

If you're curious feel free to view the rendered HTML source code.

2. The application currently does not load any JavaScript files. We're going to add jQuery as well as our own JavaScript file that will contain our code.

   a. In Visual Studio, open `~/Views/Shared/_Layout.cshtml`. This file is the layout template for the entire application and contains the common markup that will be on each page.

   b. Locate the bottom of the file where the comment indicates scripts should be added.

   c. Add a `<script>` to reference jQuery. It's located in `~/Scripts/jquery-1.8.2.js`.

   d. Add a new JavaScript file to `~/Scripts/MyScript.js` and include it after jQuery in the layout template.

```
   <!-- scripts go here -->
    @*
        the format for including scripts in a razor file is something
like this:

        <script src="~/Scripts/YourScript.js"></script>

        You would just need to fill in the relative path after the ~/
    *@

    <script src="~/Scripts/jQuery-1.8.2.js"></script>
    <script src="~/Scripts/MyScript.js"></script>
```

3. The most common place for any jQuery initialization code will be in the document ready event handler. Next you'll handle the document ready event in your script.

   a. Open `~/Scripts/MyScript.js`.

   b. Since we'll be using jQuery it will help to have intellisense inside of this file. Add a script line as the first line in this file to have Visual Studio load the intellisense information.

```
/// <reference path="jquery-1.8.2.js" />
```

   c. Next, use the `jQuery` function (or the `$` alias) to handle the document `ready` event.

d. Inside the callback use an `alert` (or `console.log`) so that you know it was invoked.

```
$(function () {
    alert("Hello jQuery");
});
```

4. Run the application and if everything's working you should see your `alert`. Once you know this is working, you can remove or comment out the call to `alert`.

Query elements and modify styles

Add zebra striping to the rows in the movie index page.

Criteria:

- Query elements by class name
- Use css() to apply styling

Steps:

1. In this part we'll use the jQuery function to select elements and modify their style sheet properties. Specifically, we'll zebra-stripe the rows in the table on the movie index page.
2. Run the application and browse to the "all movies" index page. Examine the HTML source code and notice that every other row in the table has a class name of `"evenRow"`. Use jQuery to select those rows and then apply a background color to zebra-stripe the rows.
    a. From Visual Studio open `~/Script/MyScript.js`.
    b. Inside of the `document.ready` event handler use the `jQuery` function to select all elements that match the `"evenRow"` CSS class name.
    c. On the matched items use the `css()` function to set the `"backgroundColor"` to a nice light shade of gray with the value `"#EEE"`.

```
$(function () {
    $(".evenRow").css("backgroundColor", "#EEE");
});
```

3. Run the application and browse to the movie index page. Don't forget to use Ctrl-F5 in the browser to force a refresh of the JavaScript. You should now see every other row with a light gray background color.

Set focus for input elements

Make the username field on the login page have focus.

Criteria:

- Use an id selector
- Use the focus() function

Steps:

1. If you run the application and browse to the login page and start typing a username you'll notice the username field is not being populated from the keyboard activity. The reason is that there is no focus in the input field. In this part you'll fix this.
2. Use jQuery to select the username field and set keyboard focus.
    a. Open `~/Scripts/MyScript.js`.
    b. Inside the `document.ready` event handler use jQuery to select the element with the `id` of `"Username"` (note this is case sensitive – the "U" is capitalized).
    c. On the matched element use `focus()` to set the focus.

```
$(function () {
    $(".evenRow").css("backgroundColor", "#EEE");

    $("#Username").focus();
});
```

3. Run the application and browse to the login page. Again, don't forget to use Ctrl-F5 in the browser to force a refresh of the JavaScript. Notice that the username field automatically has the keyboard focus so you can type your username without using the mouse to click the field.

Handle events, access input values and perform animations

On the login page perform some simple validation checks prior to submitting the form. Dynamically display an error message to the user and use an animation to close the error message.

Criteria:

- Handle a button click() event
- Access input values with val() and use $.trim() to strip whitespace
- Use show() to display hidden elements

Steps:

1. Run the movie review application and browse to the login page. Notice how if you submit with a blank username and/or password that it requires a round-trip to the server to validate the bad inputs.

   In this part you'll use jQuery to handle the client-side client event to validate the inputs. In doing so you'll get to dynamically display an element with an error message and use animation to hide the message when the user clicks a close button.

The error dialog we're going to use won't be as pretty as the server generated messages, but the point of this part is to use learn various jQuery features rather than make pretty error messages.

2. Run the application and browse to the login page. Open the HTML source. Notice the `<form>` element and inside is a submit button. You will register for the click event in order to perform validation on the username and password fields.

   a. Open `~/Scripts/MyScript.js`.

   b. Inside the `document.ready` event handler use jQuery to select the element with the `id` of `"loginButton"`.

   c. On the matched element use the `click()` method to register for the client event. Pass an anonymous function as the parameter to `click()`. This will be the event handler.

   d. Add an `alert` (or `console.log`) inside the client event handler to test that it's working.

```
$(function () {
    $(".evenRow").css("backgroundColor", "#EEE");

    $("#Username").focus();

    $("#loginButton")
        .click(function () {
            alert("click!");
        });
});
```

3. Run the application to test that your event handler is working and the `alert` is being displayed. Once you've confirmed that it is working then you can remove the call to `alert`.

4. Now check for empty values in the username or password fields and prevent the form from being submitted.

   a. Inside the login button's click handler, use jQuery to locate the element whose `id` is `"Username"` and store the result in a local variable. Do the same for the element whose id is `"Password"`.

   b. Use the `val()` function to check if either the username or password inputs values are blank. If so, return `false` from the click event handler – this will prevent the form from being submitted to the server.

```
$(function () {
    $(".evenRow").css("backgroundColor", "#EEE");

    $("#Username").focus();

    $("#loginButton")
        .click(function () {
            var uid = $("#Username");
            var pwd = $("#Password");

            if (uid.val() === "" || pwd.val() === "") {
                return false;
            }
        });
});
```

5. Run the application and try to submit the login page with an empty username and/or password. Confirm that the browser is not submitting the form to the server.
6. Now to let the user know the problem we need to give them a message. Open the HTML source on the login page and notice the `<div id="errorDialog">` inside of the `<form>`. This is our error dialog. It's not currently visible (due to a CSS rule) but once our validation fails we will display it.
    a. Inside of the `if` check for empty username and password, use jQuery to locate the element whose `id` is `"errorDialog"`.
    b. On the matched element use `show()` to display the error dialog.

```
$(function () {
    $(".evenRow").css("backgroundColor", "#EEE");

    $("#Username").focus();

    $("#loginButton")
        .click(function () {
            var uid = $("#Username");
            var pwd = $("#Password");

            if (uid.val() === "" || pwd.val() === "") {

                $("#errorDialog").show();

                return false;
            }
        });
});
```

7. Run the application and try to login with empty username again. Notice the dialog now appears. The next thing to do is allow the user to click "Close" to hide the error dialog.
    a. Back in `~/Scripts/MyScript.js`, inside the `document.ready` event handler use jQuery to select the element with the `id` of `"closeErrorDialog"`.

   b. On the matched element use the `click()` function and pass an anonymous function to handle the click event.

   c. Inside the click event handler use the jQuery function to locate the error dialog. Use `slideUp()` to hide it.

```
$(function () {
    $(".evenRow").css("backgroundColor", "#EEE");

    $("#Username").focus();

    $("#loginButton")
        .click(function () {
            var uid = $("#Username");
            var pwd = $("#Password");

            if (uid.val() === "" || pwd.val() === "") {

                $("#errorDialog").show();

                return false;
            }
        });

    $("#closeErrorDialog").click(function () {
        $("#errorDialog").slideUp();
    });
});
```

8. Run the application, try to submit an empty username/password and then click “Close” on the error dialog to ensure the animation is working properly.

9. Notice if you enter whitespace in the username or password fields that the form submit is not prevented. Fix this now.

   a. Use `$.trim()` on the username and password fields to trim the whitespace in the `if` check.

```
$(function () {
    $(".evenRow").css("backgroundColor", "#EEE");

    $("#Username").focus();

    $("#loginButton")
        .click(function () {
            var uid = $("#Username");
            var pwd = $("#Password");

            if ($.trim(uid.val()) === "" || $.trim(pwd.val()) === "") {

                $("#errorDialog").show();

                return false;
            }
        });

    $("#closeErrorDialog").click(function () {
        $("#errorDialog").slideUp();
    });
});
```

10. [OPTIONAL] If you're feeling adventurous, detect which of the input fields is blank (username or password) and update the `<p id="errorMessage">` inside of the error dialog dynamically with a message indicating which field is blank. Use `text()` to set the contents dynamically.

Solutions:

The final solution for this lab is available in the `~/after` directory.