Using jQuery

Overview

This lab leads you step-by-step through the process of using jQuery in a Web application. You'll start with an empty-ish Web page, link in the jQuery script file, and explore numerous jQuery techniques as you progress through the exercises.

Roadmap

There are 7 exercises in this lab, of which the last three exercises are "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Including the jQuery JavaScript file in the Web page
2. Using core jQuery syntax
3. Using jQuery selectors
4. Using jQuery events
5. Going further with jQuery events (if time permits)
6. Using effects (if time permits)
7. Additional suggestions (if time permits)

Server

HTML files can be opened from the file system, but for security reasons browsers will limit the possibilities. It is better to open a server in the project folders. Here are two ways to do that.

Visual Studio Code

1. Open Visual Studio Code
2. Choose menu "File | Open…" and open the root folder of the course material. After opening you should see a folder called ".vscode" at the top of the tree in the Explorer on the left.
3. Choose menu "Tasks | Run Task…" and choose task "npm: install".
4. Select/Open a file in the folder that you want to be the root of your server. This will generally be the homepage of your app.
5. Choose menu "Tasks | Run Build Task…" or choose the shortcut.
6. To close the server again, choose menu "Tasks | Terminate Task…".

Command line

1. Globally install “live-server” with command “npm install -g live-server” (see: https://www.npmjs.com/package/live-server)
2. Open the command line (Windows) or terminal (macOS) in the folder that you want to be the root of your server. This will generally be the homepage of your app.
3. Run “live-server” to open the server. Choose Ctrl+c to close the server again.

Getting started with the Student project

In Visual Studio, open the *Student* project for this lab as follows:

- *Folder:*
  `\start`

The project contains an HTML page named `index.html`. Take a look at this page now.

- The `<head>` links in a style sheet, and also has an empty `<script>` tag where you'll add your JavaScript code in this lab.
- The `<body>` is empty at the moment – you'll add your HTML content here.

There's a simple style sheet named `Stylesheet.css`. Take a look if you like, it's very minimalistic.

Exercise 1: Including the jQuery JavaScript file in the Web page

If you want to use jQuery in a Web page, you must include the jQuery JavaScript file in your Web page. You can either define an external link to jQuery on a CDN website on the Internet, or download the jQuery JavaScript file and reference it locally.

We'll take the latter approach here, just in case there are Internet connectivity difficulties in the classroom environment. We've already downloaded the jQuery JavaScript file, in the root you'll find a `\jQuery` folder. This folder actually contains two jQuery files:

- `jquery-1.10.2.js` is the human-readable version, for easier use during development
- `jquery-1.10.2.min.js` is the minified version, for faster downloads in production

Add the human-readable version into your project's folder.

The last step in this exercise is to add a `<script>` tag to `index.html`, to link in the jQuery JavaScript file. Add the following statement to your page:

```
<script src="jquery-1.10.2.js"></script>
```

Exercise 2: Using core jQuery syntax

Almost everything you do when using jQuery will access the DOM tree. It's common to start adding events as soon as the DOM is ready. To do this, register a `ready` event for the `document` as follows (put this code in the empty `<script>` tag in the `<head>` section of the Web page):

```
$(document).ready(function() {
   // Do stuff when DOM is ready.
});
```

You could put an `alert()` into this function, but that doesn't make much sense because `alert()` doesn't need the DOM to be loaded. So let's try something a little more sophisticated, such as showing an alert when the user clicks a link. Add the following HTML to the `<body>`:

```
<a href="">Link</a>
```

Now update the `$(document).ready` handler as follows:

```
$(document).ready(function() {
   $("a").click(function() {
     alert("Hello world!");
   });
});
```

Here's what's going on:

- `$` is an alias for the jQuery class. Therefore, `$()` constructs a new jQuery object.
- `$("a")` is a jQuery selector, in this case, it selects all <a> elements.
- The `click()` function is a method of the jQuery object. It binds a `click` event to all selected elements (in this case, a single <a> element) and executes the provided function when the event occurs.

This is similar to the following code:

```
<a href="" onclick="alert('Hello world')">Link</a>
```

The difference is quite obvious: You don't need to write an `onclick` for every single element. You have a clean separation of structure (HTML) and behavior (JS), just as you can separate structure and presentation by using CSS.

Exercise 3: Using jQuery selectors

jQuery allows you to use a combination of CSS and XPath selectors to locate elements. To try some of these selectors, first add a couple of lists to the <body> section, something like this:

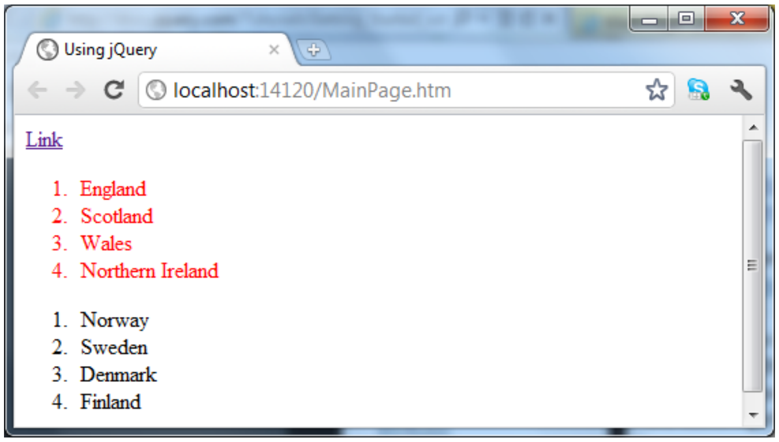
```
<ol id="ukCountriesList">
      <li>England</li>
      <li>Scotland</li>
      <li>Wales</li>
      <li>Northern Ireland</li>
</ol>

<ol id="nordicCountriesList">
      <li>Norway</li>
      <li>Sweden</li>
      <li>Denmark</li>
      <li>Finland</li>
</ol>
```

In this example, you can access a whole list via its `id`, e.g. `"ukCountriesList"`. In classic JavaScript, you'd select it via `document.getElementById("ukCountriesList")`. With jQuery, you do it like this (add this code into the `<script>` element in the head of your document):

```
$(document).ready(function() {
   $("#ukCountriesList").addClass("red");
});
```

This code locates the element that has the specified `id`, and adds the "`red`" class to it (this CSS class is defined in `Stylesheet.cs`). If you reload the page in your browser, you should see that the first ordered list has red text. The second list is not modified.



Link

1. England
2. Scotland
3. Wales
4. Northern Ireland

1. Norway
2. Sweden
3. Denmark
4. Finland

It's also possible to access children or descendant elements. To illustrate this, modify your first list as follows:

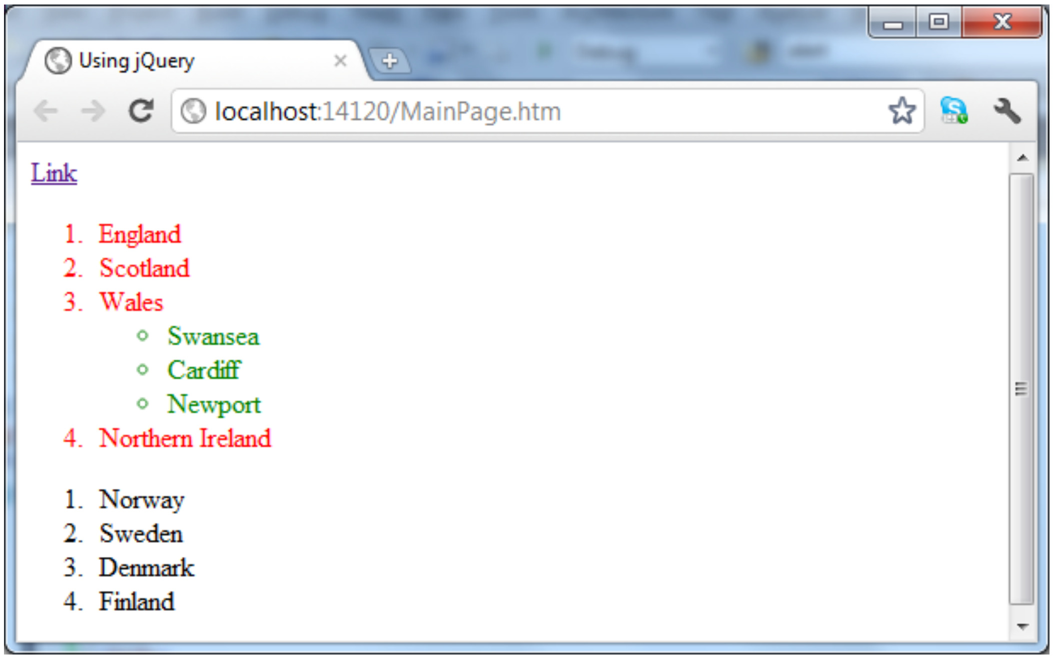
```
<ol id="ukCountriesList">
      <li>England</li>
      <li>Scotland</li>
      <li>Wales
            <ul>
                  <li>Swansea</li>
                  <li>Cardiff</li>
                  <li>Newport</li>
            </ul>
      </li>
      <li>Northern Ireland</li>
</ol>
```

Now add the following script:

```
$(document).ready(function () {
    $("#ukCountriesList > li li").addClass("green");
});
```

The selector syntax first selects the element with id "`ukCountriesList`", then drills down into child `<li>` elements (the > means *first-level child elements*), and then selects descendent `<li>` elements (the absence of a > means *descendant elements at any depth*).

Reload the Web page. It should appear as follows:



Link

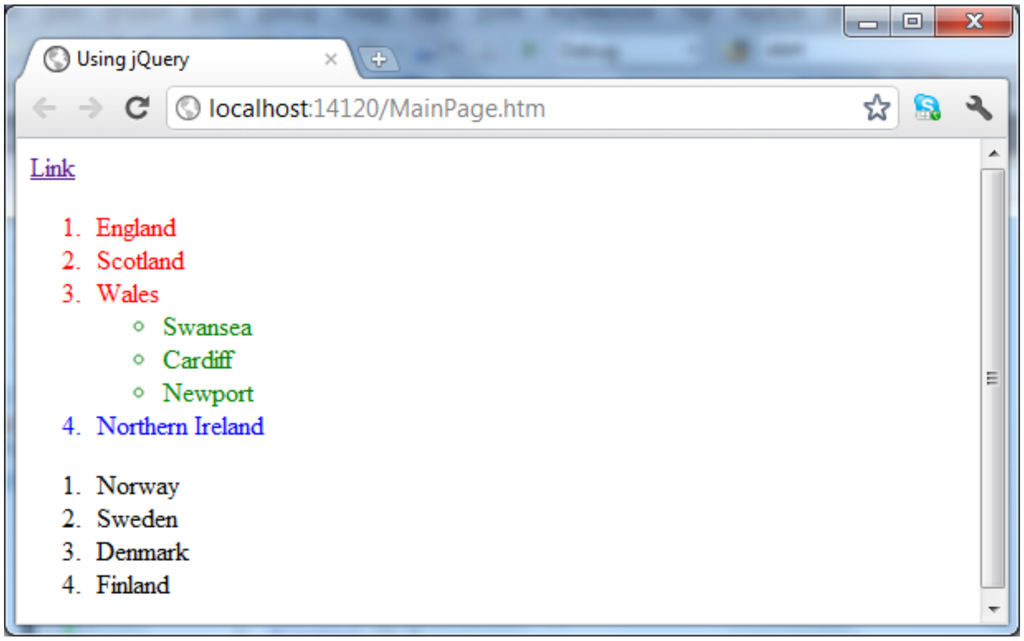
1. England
2. Scotland
3. Wales
   - Swansea
   - Cardiff
   - Newport
4. Northern Ireland

1. Norway
2. Sweden
3. Denmark
4. Finland

Now for something a little more sophisticated… Imagine you want to add and remove a CSS class when the user hovers over the `<li>` element, but only on the last element in the list. This is the script to achieve this effect

```
$(document).ready(function() {
   $("#ukCountriesList li:last").hover(function() {
     $(this).addClass("blue");
   },function(){
     $(this).removeClass("blue");
   });
});
```

Reload the Web page. It should appear as follows when the user hovers over the last list item in the UK countries list:



Link

1. England
2. Scotland
3. Wales
   - Swansea
   - Cardiff
   - Newport
4. Northern Ireland

1. Norway
2. Sweden
3. Denmark
4. Finland

There are many other selectors, similar to CSS and XPath syntax. You can find a list of all available expressions and some examples at http://api.jquery.com/category/selectors/.

Exercise 4: Using jQuery events

For every `onxxx` event available, such as `onclick`, `onchange`, and `onsubmit`, there's a jQuery equivalent. Some other events, such as `ready` and `hover`, are provided as convenient methods for certain tasks. You can find a complete list of all events in the jQuery Events Documentation at http://api.jquery.com/category/events/.

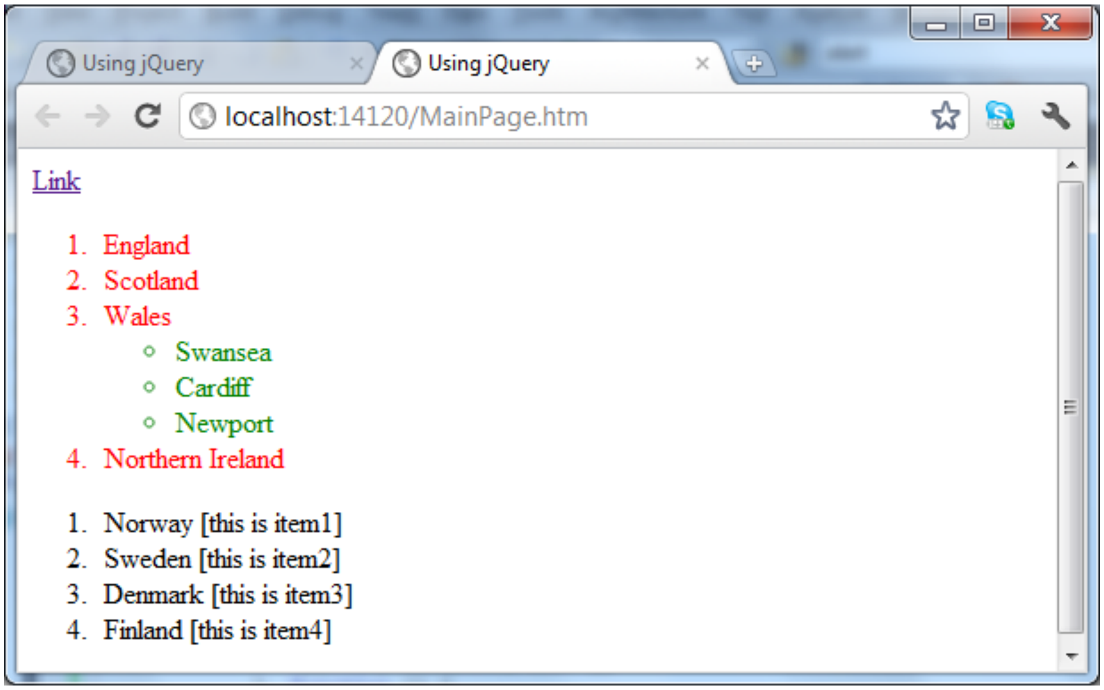
With those selectors and events you can already do a lot of things, but there is more. Add the following script to your page:

```
$(document).ready(function () {
    $("#nordicCountriesList").find("li").each(function (i) {
        $(this).append(" [this is item" + (i + 1) + "]");
    });
});
```

Let's analyze this example:

- `$("#nordicCountriesList")` locates the Nordic countries list.
- `find()` allows you to further search the descendants of the already selected elements. Therefore, `$("#nordicCountriesList").find("li")` is mostly the same as `$("#nordicCountriesList li")`.
- `each()` iterates over every selected element and allows further processing. Most methods, such as `addClass()`, use `each()` internally.
- Inside the function that's applied on each `<li>`, the `append()` function appends some text to the end of the current `<li>` element.

Reload the Web page. It should appear as follows:



Link

1. England
2. Scotland
3. Wales
    - Swansea
    - Cardiff
    - Newport
4. Northern Ireland

1. Norway [this is item1]
2. Sweden [this is item2]
3. Denmark [this is item3]
4. Finland [this is item4]

Another task you often face is to call methods on DOM elements that are not covered by jQuery. For example, if you had a form and you wanted to reset its contents (e.g. to clear the fields) when the user clicked a *Reset* button, you could use the following code:

```
$(document).ready(function() {
      // Use this to reset a single form.
      $("#reset").click(function() {
            $("form")[0].reset();
      });
});
```

This code selects the first `<form>` element and calls `reset()` on it. In case you had more than one form, you could also do this:

```
$(document).ready(function() {
      // Use this to reset several forms at once.
      $("#reset").click(function() {
            $("form").each(function() {
                  this.reset();
            });
      });
});
```

This would select all forms within your document, iterate over them and call `reset()` for each one. Note that in an `.each()` function, `this` refers to the actual element. Also note that, since the `reset()` function belongs to the `<form>` element (and not to the jQuery object), you cannot simply call `$("form").reset()` to reset all the forms on the page.

Exercise 5 (If time permits): Going further with jQuery events

A common challenge is to select only certain elements from a group of similar or identical ones. jQuery provides `filter()` and `not()` for this.

- `filter()` reduces the selection to the elements that fit the filter expression.
- `not()` does the contrary and removes all elements that fit the expression.

To illustrate how this works, recall the UK countries list you created earlier:

```
<ol id="ukCountriesList">
      <li>England</li>
      <li>Scotland</li>
      <li>Wales
            <ul>
                  <li>Swansea</li>
                  <li>Cardiff</li>
                  <li>Newport</li>
            </ul>
      </li>
      <li>Northern Ireland</li>
</ol>
```

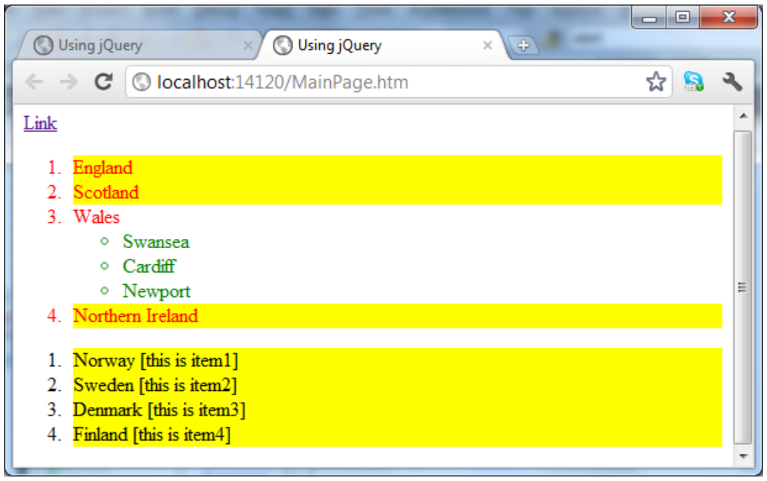
Imagine you wanted to select all `<li>` elements that have no `<ul>` children (i.e. imagine you wanted to select England, Scotland, and Northern Ireland in the example above). You can achieve this effect with the following script, which you should add to your Web page:

```
$(document).ready(function() {
      $("ol > li").not(":has(ul)").css("background-color", "yellow");
});
```

Let’s analyze this example:

- `$("ol > li")` selects `<li>` elements that are direct children of an `<ol>` element.
- `.not(":has(ul)")` removes all elements that have a `<ul>` child element.
- `.css("background-color", "yellow")` sets selected elements to yellow.

Reload the Web page. It should appear as follows:



Link

1. England
2. Scotland
3. Wales
   - Swansea
   - Cardiff
   - Newport
4. Northern Ireland

1. Norway [this is item1]
2. Sweden [this is item2]
3. Denmark [this is item3]
4. Finland [this is item4]

jQuery also supports the [expression] syntax, which is taken from XPath. You can use this to filter selections by attributes. Maybe you want to select all <a> elements that have a name attribute:

```
$(document).ready(function() {
     $("a[name]").css("background", "#eee");
});
```

More often than selecting <a> elements by name, you might need to select <a> elements by their "href" attribute. To match only a part of the value, you can use the *= operator rather than the = operator. Consider the following example:

```
$(document).ready(function() {
     $("a[href*='/gallery']").click(function() {
          // Do something with all links to somewhere on /gallery
  });
});
```

All the selectors we've seen so far have selected child or descendent elements, or filtered the current selection. There are situations where you need to select the previous or next elements, known as siblings. Think of a FAQ page, where all answers are hidden at first, and then shown when the question is clicked. The jQuery code to do this is as follows:

```
$(document).ready(function() {
     $('#faq').find('dd').hide().end().find('dt').click(function() {
          $(this).next().slideToggle();
     });
});
```

In this example:

- This example uses chaining to reduce the code size and gain better performance, as '#faq' is only selected once. By using end(), the first find() is undone, so you can start search with the nextfind() at your #faq element, instead of the <dd> children.
- Within the click handler function, we use $(this).next() to find the next sibling starting from the current <dt>. This allows us to quickly select the answer following the clicked question.

In addition to siblings, you can also select parent and ancestor elements. Imagine you want to highlight the paragraph that is the parent of the link the user hovers over. Try this:

```
$(document).ready(function(){
      $("a").hover(function(){
            $(this).parents("p").addClass("highlight");
      },
      function(){
            $(this).parents("p").removeClass("highlight");
      });
});
```

For all hovered <a> elements, the parent paragraph is searched and a class `"highlight"` added and removed.

One final observation here… jQuery is a lot about making code shorter and therefore easier to read and maintain. The following is a shortcut for the `$(document).ready(callback)` notation:

```
$(function() {
      // code to execute when the DOM is ready
});
```

Applied to the first example we saw at the start of the lab, we could use the following simplified syntax:

```
$(function() {
      $("a").click(function() {
            alert("Hello world!");
      });
});
```

Exercise 6 (If time permits): Using jQuery effects

Simple animations with jQuery can be achieved via the `show()` and `hide()` functions. Consider the following example:

```
$(document).ready(function(){
   $("a").toggle(function(){
         $(".stuff").hide('slow');
   },function(){
         $(".stuff").show('fast');
   });
});
```

Experiment with these capabilities in your Web page.

You can also create any combination of animations with `animate()`, such as a slide with a fade:

```
$(document).ready(function(){
   $("a").toggle(function(){
      $(".stuff").animate({ height: 'hide', opacity: 'hide' }, 'slow');
   },function(){
      $(".stuff").animate({ height: 'show', opacity: 'show' }, 'slow');
   });
});
```

Experiment with these capabilities in your Web page as well.

Exercise 7 (If time permits): Additional suggestions

Consider how you might retrofit jQuery into the Product Manager Web application that you've been working on during this course.