LAB 3: Working with Ajax

Contents

Introduction 1
Resources 1
Solutions 1
Task 1: Using JSON 2
Task 2: Using jQuery 4
Task 3: Posting data 5
Task 4: Mocking Ajax 7
Task 5: Ajax data formats 9

Introduction

In this lab you will practice using JavaScript and jQuery to make asynchronous requests.

The lab consists of five tasks.

Resources

- Lecture notes and sample code from Module Website
- Visual Studio and Firefox/Firebug (or any combination of editor/browser)
- JavaScript reference, e.g. **https://developer.mozilla.org/en/JavaScript/Reference**

Solutions

Note that the VS2010 project you download contains sample solutions for lab tasks, in the *Views/LabSolutions* folder. Apologies for any inline event handlers which have been left in this code!

Task 1: Using JSON

Download *AjaxDestinations.zip* from your module website and extract the contents, which include a VS2010 ASP.NET MVC project called *AjaxDestinations*. The project includes a controller *LabController*, and a *Lab* folder inside *Views*. You will create web pages as views inside the *Lab* folder, and actions in the controller to render these views. There is also a controller *DatasourcesController* which you will use to provide data for your pages.

You will now create a page which requests data interactively when the user selects an item in a select box. The data retrieved depends on the selection.

1. First, test the data sources you will use. Run the project, and access the following URLs in your browser, where *xxxx* is the port number used by the web server.These should return JSON – examine the structure of the data which is returned by each.

   http://localhost:xxxx/Datasources/Packages
   http://localhost:xxxx/Datasources/PackageDetails/1

2. Create a new action called *PackageInfoJson* in *LabController* which simply returns a view without passing any data to the view.

3. Create a new view in the *Lab* folder called *PackageInfoJson.* Add the following references, which you should also add in **all views you create in this lab**.

```
<link rel="Stylesheet" type="text/css" href="/Content/gcutours.css"
media="screen" />
<script type="text/javascript" src="/Scripts/jquery-1.5.1.min.js">
</script>
```

4. Complete the view so that it meets the following specification.

   HTML should include
   - an HTML select element with *id="packagelist"*
   - a paragraph element with *id="info"*

   JavaScript should include
   - A function *getPackages* which sends an XMLHttpRequest to */Datasources/Packages*, with a callback function *getPackagesCallback*.
   - A function *getPackagesCallback* which checks the request ready state and status, and when the data is ready parses the response as JSON and uses the result to add option elements to the *packagelist* element (*text=packagename, value=packageid*)
   - A function *getPackageInfo* which sends an XMLHttpRequest to */Datasources/Packages/<id>*, where *<id>* is the value of the currently selected item in *packageList*, with a callback function *getPackageInfoCallback*.
   - A function *getPackageInfoCallback* which checks the request ready state and status, and when the data is ready parses the response as JSON and

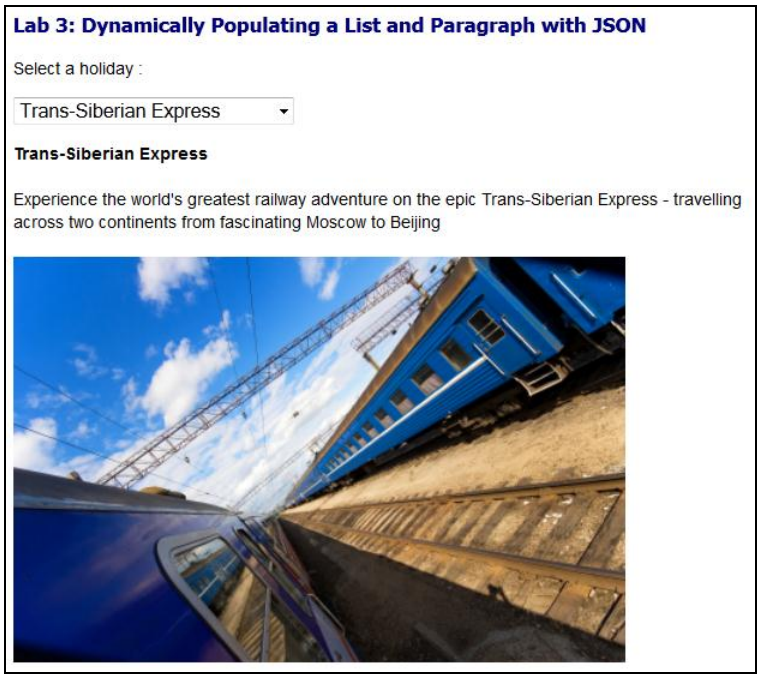
uses the result to set the *innerhtml* of the info element to display the *packagename* and *description* properties returned and the image indicated by the *tourpicurl* property.

- The window *onload* event should be handled by *getPackages*
- The *onchange* event of *packagelist* should be handled by *getPackageInfo*.
- *getPackagesCallback* should call *getPackageInfo* just before it finishes.

5. Run the project and access your page as

   http://localhost:xxxx/Lab/PackageInfoJson

   The page should look something like the figure below. Selecting a package in the dropdown box should cause the details of that package to be displayed without a page refresh.



6. Examine the requests in Firebug (or equivalent). Note that Firebug shows these as XHRs. Look at the content of the response to each request.

Task 2: Using jQuery

You will now create a version of the page from task 1, this time using jQuery to simplify your JavaScript code. The functionality of the page will be exactly the same as task 1, and the same data sources will be used.

1. Create a new action called *PackageInfoJsonJquery* in *LabController* which simply returns a view without passing any data to the view.

2. Create a new view in the *Lab* folder called *PackageInfoJsonJquery.* Complete the view so that it meets the following specification.

   HTML
   - Exactly the same as task 1

   JavaScript should include
   - An inline function which handles the $(document).ready event, and sends an XMLHttpRequest, with data type *json*, to */Datasources/Packages*, with an inline callback function. The callback function should use the result to add option elements to the *packagelist* element (*text=packagename, value=packageid*)
   - A function *getPackageInfo* which sends an XMLHttpRequest, again with data type *json*, to */Datasources/Packages/<id>*, where *<id>* is the value of the currently selected item in *packageList*, with an inline callback function.
   - The callback function should use the result to set the *innerhtml* of the info element to display the *packagename* and *description* properties returned and the image indicated by the *tourpicurl* property.
   - The *onchange* event of *packagelist* should be handled by *getPackageInfo*.
   - *getPackagesCallback* should call *getPackageInfo* just before it finishes.
   - Optionally, hide the info paragraph and fade it in when new data has been loaded into it.

3. Run the project and access your page as

   http://localhost:xxxx/Lab/PackageInfoJsonJquery

   Test as before and examine the requests using Firebug.

Task 3: Posting data

You will now create a page which contains a form which allows the user to enter data which will be used to add a record to a database. The data will be posted to the URL */Datasources/AddUser.*

1. Create a new action called *AddUserJquery* in *LabController* which simply returns a view without passing any data to the view.

2. Create a new view in the *Lab* folder called *AddUserJquery.* Complete the view so that it meets the following specification.

   HTML should include
   - A stylesheet link which refers to */Content/formstyles.css*
   - An HTML form with input elements with ids *firstname*, *lastname*, *address*, *username*, *password* and *datejoined,* and a button with id *savebutton.*
   - Wrap form elements and labels in CSS classes which will cause the form to be neatly styled, for example:

```
<div class="row">
    <span class="label">First name:</span>
    <span class="formw">
        <input type="text"
            id="firstname" />
    </span>
</div>
```

   - A span element with id *message.*

   JavaScript should include
   - An inline function which handles the $(document).ready event, and populates the datejoined element with a string representing today's date. Use the following code to get the date:

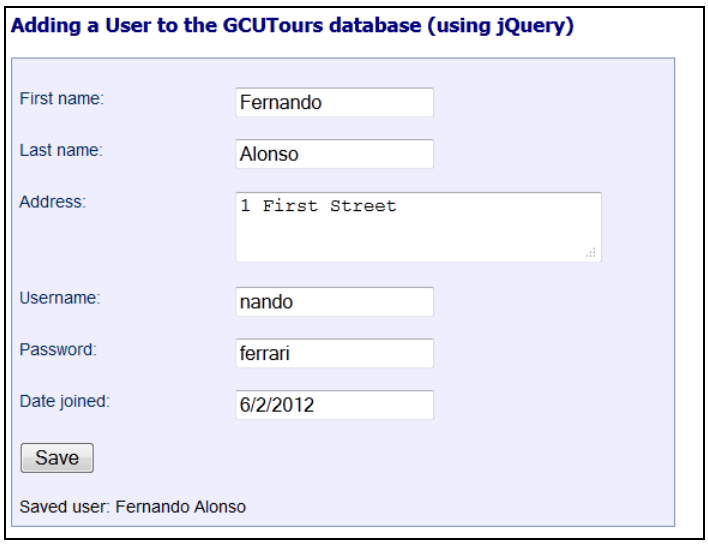
```
var today = new Date();
var todayStr = today.getDate() + "/" + (today.getMonth() + 1) + "/"
+ (today.getYear() + 1900);
```

   - A function *addUser which* creates a JSON object with property names matching the form element ids and property values obtained from the form data. The function should then post the JSON object using an XMLHttpRequest, with data type *json*, to */Datasources/AddUser*. The result of the request, which will contain a string, should be displayed in the *message* element.
   - Optionally, hide the info paragraph and fade it in when new data has been loaded into it.

3. Run the project and access your page as

   http://localhost:xxxx/Lab/AddUserJquery

4. Enter data in the form and click the button. Check that the message indicates that your data has been saved, as shown in the figure below. Examine the request using Firebug, noting the Post and Response data.



Task 4: Mocking Ajax

In this task you will use the **Mockjax** library to create a mock response to an Ajax request. The mocked request will return data which is used to display a list of users.

1. Create a new action called *UsersJquery* in *LabController* which simply returns a view without passing any data to the view.

2. Create a new view in the *Lab* folder called *UsersJquery.* Replace the code in the view with the following. Note that the Mockjax library is referenced.

```
<!DOCTYPE html>
<html>
<head>
    <title>Ajax Example</title>
    <link rel="Stylesheet" type="text/css" href="/Content/gcutours.css" />
    <script type="text/javascript" src="/Scripts/jquery-1.5.1.min.js"></script>
    <script type="text/javascript" src="/Scripts/jquery.mockjax.js"></script>
    <script language="JavaScript" type="text/JavaScript">
        $(document).ready(function () {
            $.ajax({
                type: "GET",
                url: "/Users",
                dataType: "json",
                success: function (json) {
                    var info = $('#userlist').empty().hide();
                    $.each(json.users, function (i, item) {
                        var firstname = item.firstname;
                        var lastname = item.lastname;
                        var address = item.address;
                        var datejoined = item.datejoined;
                        var content = "<p><b>" + firstname + " " + lastname +
                           "</b><br/>" + address + "<br/>Date joined: " +
                            datejoined + "</p>";
                        info.append(content);
                    });
                    info.fadeIn(1000);
                }
            });
        });
    </script>
</head>

<body>
       <h3>Lab 3: Displaying a list of users with JSON (using jQuery)</h3>
       <form id="userform" action="">
        <div id="userlist"></div>
       </form>
</body>
</html>
```

3. Add a call to *$.mockjax* so that the following data is displayed in the page when it loads:

Abu Abdullah Ibn Battuta
2 Silk Road
Date joined: 13 July 2007

Amerigo Vespucci
1499 America Avenue
Date joined: 09 June 2007

Bartolemeu Dias
1481 Gold Coast Road
Date joined: 03 April 2005

Jacques Cartier
156 Canada Crescent
Date joined: 15 March 2007

Christopher Columbus
1492 America Avenue
Date joined: 24 July 2006

4. Examine the content of the Net tab in Firebug – was the request recorder? Examine the Console tab in Firebug – what do you see there?

Task 5: Ajax data formats

You should research into the Ajax data formats which are currently in use in public websites. We can then gather this information to produce and analysis of current practice.

1. Find a web page which you think is using Ajax. Confirm by examining requests which are sent when pages load or when you interact with the page – look for XHRs displayed in the Net tab of Firebug.

2. Record the following information:

   - URL of the page
   - Data included in the query string (GET request)
   - Data posted with the request, and format of data
   - Format of response
   - Commentary on how you think the response data is used in the page

   You can repeat this for as many pages from different sites as you wish. Please email me any information you collect.