LAB 4: Ajax and services

Contents

Introduction

In this lab you will practice using public web services.

The lab consists of two 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 a web service

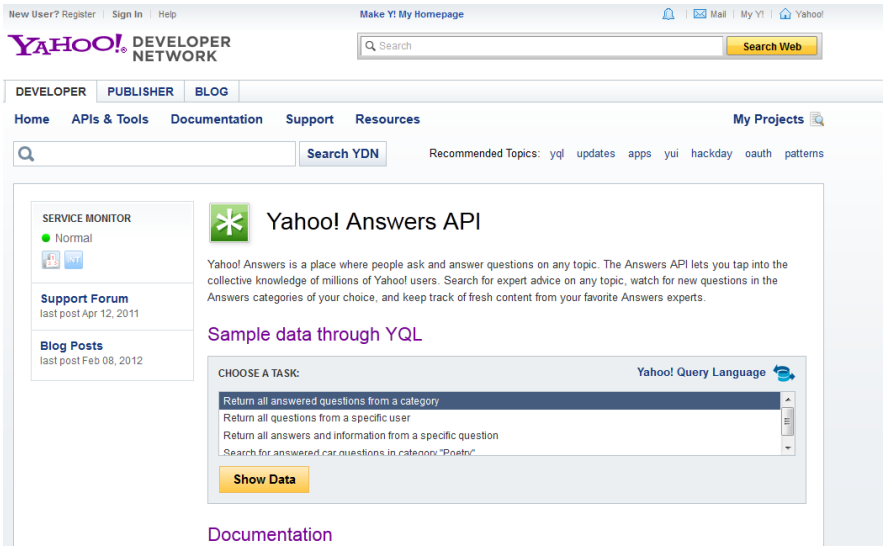
In this task you will look an example of a public Web Service.

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.

1. Open the following URL in a browser:

   http://developer.yahoo.com/answers/

   This page describes the Yahoo Answers web services, one of the many categories of service available on the Yahoo Developer Network.



2. Scroll down to the Documentation section of the page, click on the questionSearch link. This takes you to a page which documents the API for accessing the service via a GET request. Read over this documentation.

3. Compare the following URL with the documentation, and then open the URL in your browser. This URL contains a developer key. Make sure the key is not changed and that there are no spaces in the URL. Before you do so, compare the URL with the documentation.

   http://answers.yahooapis.com/AnswersService/V1/questionSearch?appid=X717TwPV34Fqnw_4m9jVeH73a650dszmHQ8N1NsXYUgLJLNMXIrapq_.A4hh7MZDoWr9IA&query=jQuery

   The result should be an XML document containing a set of questions about the topic “jQuery”, with answers and other associated information for each question.

4. With the aid of the documentation, modify this request to return data in JSON format. The result should be JSON text containing the same information as the XML version. Compare the results from these two requests. Look carefully at the structure of the JSON data – you will need to understand the format for step 7 below.

5. Append the following to the JSON URL and examine the result:

   &callback=myCallbackFunction

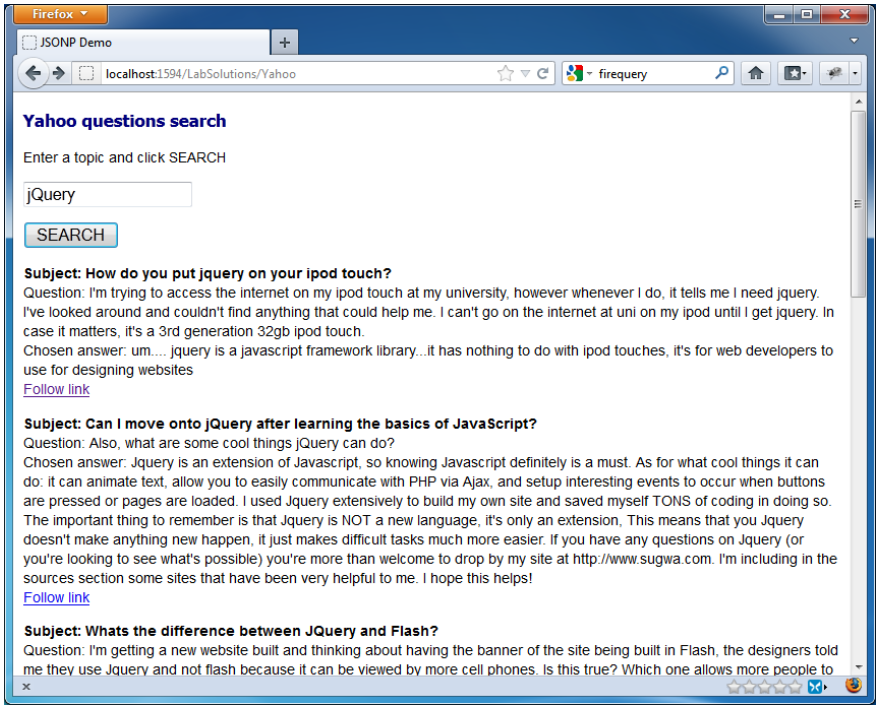
   What has been added to the output?

   This option is provided to support JSONP.

   Now you will use this service in a web page.

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

7. Create a new view in the *Lab* folder called *Yahoo.* Complete the view so that it displays a text box and a button. Write code so that when the button is clicked, a list of questions related to the topic entered in the text box is displayed, using JSON data retrieved from the Yahoo Questions web service. You may use any technique which your test browser supports to send a request to the web service. You may use jQuery in your code if you wish. The page content should look similar to the following after data has been retrieved.



8. Once you have the page working, you can explore the API documentation and add further filter(s) to your page, for example to filter by region.

Task 2: Using a JavaScript API

In this task you will make use of a JavaScript API, which is actually in the form of a jQuery plugin, to display content from Twitter in a page.

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

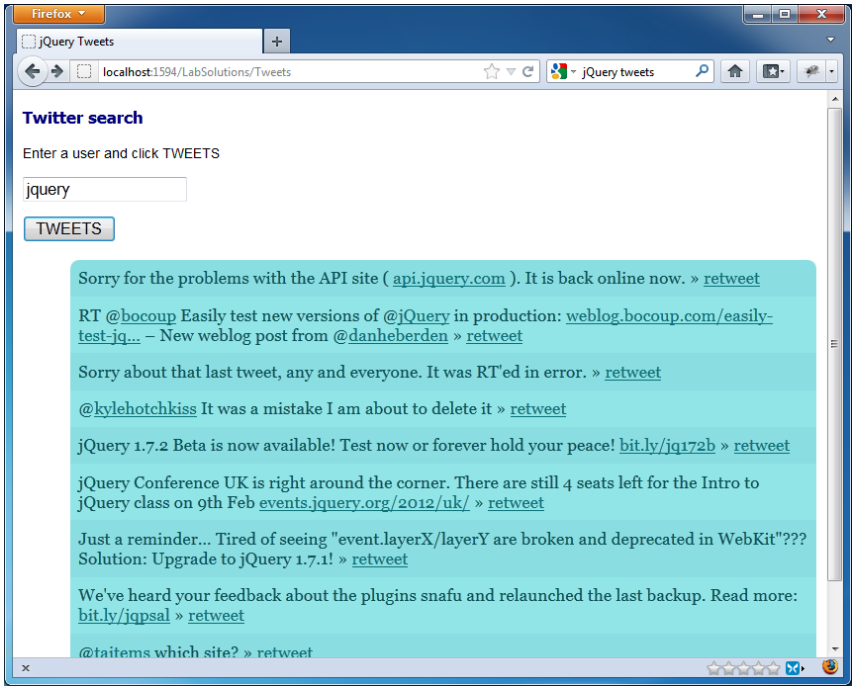
2. Create a new view in the *Lab* folder called *Tweets.*

3. You will be using the Tweet! Plug-in. Go to **http://tweet.seaofclouds.com/** to see the documentation for this. The script file and style sheets have been downloaded and are included in the *AjaxDestinations* project.

4. Add script and stylesheet references to your page.

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

   With the help of the documentation, add a search box and write code which displays a list of tweets for the username entered in the search box. The result should look similar to the following:



5. Use Firebug to examine the requests and responses. Note that this jQuery plug-in sends requests to the Twitter REST API.

   What technique is used to send the requests (XHR, JSONP, etc)?

Task 3: Using a server-side proxy

In this task you will use a Geonames REST web service to display selected details of a specified country. You will access the service using a server-side proxy so that only the parts of the data which are to be displayed need to pass from your server to the browser.

1. Find the documentation for the *countryInfo* web service at **http://www.geonames.org/export/web-services.html**. Open the example URL in your browser to see the format of the data returned. Note that this is in XML format. There is a JSON version of this service, but we will use XML here as it is convenient to work with XML data in .NET.

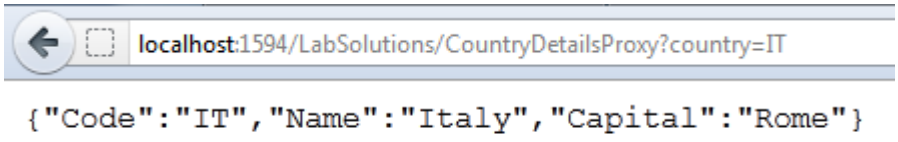
2. Create a new action called *CountryDetailsProxy* in *LabController.* The action should take a String parameter called *country*, which will be the search term. Complete the code in this so that:

   - It uses the *Load* method of the *System.Xml.Linq.XElement* class to access the service URL, with the value of *country* as a parameter, to construct an instance of *XElement*. An *XElement* object represents an XML element, and can be used to hold the XML returned by the service. You may need to consult the documentation of this class on MSDN

   - It uses a LINQ query to select the values in the *countryCode*, *countryName* and *capital* elements from the *XElement* instance into appropriately named properties of an anonymous typed object. Use the *FirstOrDefault* method to ensure that only one result is returned. See this example if you are not sure how to do this: **http://msdn.microsoft.com/en-us/library/bb387028.aspx**

   - It returns the anonymous typed object in a *JsonResult*.

3. Access this action method in a browser to test your proxy. You should see a simple JSON object like this:



4. Now create a basic action called *CountryDetails* which simply returns a view, and create a matching view. Complete the client-side code in the view so that it:

   - Displays a text box with *id=country*, and a button.

   - Calls a function when the button is clicked which sends an XHR to your proxy's URL with the value entered in the text box by the user appended as a parameter. You can use jQuery if you wish.

- Uses the result of the XHR to display data in the page, like this

5. Start Fiddler if you have it installed. Submit the country code from your page again, and look in Fiddler at the call to the remote service and the call from the browser to your proxy. Note the formats, and note that the volume of data in the latter case is much reduced.

Notes:

This example asks you to use a standard MVC controller as your proxy. If you have installed the ASP.NET MVC 4 beta, you could modify this example to use the new ASP.NET Web API to create your proxy service. See **http://www.asp.net/web-api** for more information.

This example uses LINQ to query the response from an XML service. If you want to use a JSON service instead then you can use JSON.NET, a .NET library which supports JSON serialisation and LINQ to JSON. See **http://json.codeplex.com**.