Awarding Great British Qualifications

Dynamic Websites

Topic 10:

Web services

Scope and Coverage

This topic will cover:

- An overview of web services such as:
  - SOAP
  - REST
  - RSS
  - Web API

Learning Outcomes

By the end of this topic students will be able to:

- Make use of SOAP-based web services;
- Make use of REST-based web services;
- Make use of RSS and Web API;
- Integrate data from two web services into a mash-up.

Introduction

- In this lecture, we are going to look at the topic of web services.
  - Specifically, how we can consume the services that others have written.
- We are ready to create simple web services.
  - A PHP script that interprets a GET request makes an effective “first draft” of a web service.
- There are many pre-written services on the Internet that we can profitably integrate into our own code.

Web API

- Web services are based on SOAP protocol.
- Web API is framework used to help build REST interfaces.
- Web SOAP based service returns data as XML.
- Web API is a HTTP based service which returns JSON or XML data.

REST and SOAP

- Most web services are based on one or two systems.
  - ***Simple Object Access Protocol*** (SOAP)
  - ***Representational State Transfer*** (REST)
- The former is a more mature system for implementing web service.
  - Heavily standards based
- The latter is more modern and lightweight.
  - Simple communication via XML

SOAP

- SOAP is an XML based protocol.
  - However, the structure of the documents that are sent via SOAP are very strictly controlled.
- The root element of the XML document is called the ***envelope***.
  - Inside this envelope are the ***header*** and the ***body***.
- We construct an envelope that contains the information we want.
- We get back an envelope that contains the response.

Example SOAP Envelop

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Using SOAP

- Most SOAP implementation engines will let you ignore the details of constructing the building SOAP envelopes.
- SOAP documents will often be more complex than simple example.
  - They can include other things, such as error handling.
- SOAP envelopes can be sent using a number of protocols.
  - SMTP and HTTP being two valid examples.

REST

- REST is a more lightweight protocol associated with the Web 2.0 movement.
  - This focus is on using HTTP to transport simple XML documents.
- REST is a style of interaction summed up as four actions:
  - GET, POST, PUT, DELETE
- Each of these verbs map onto particular states.
  - You use these operations to do different things.

GET & PUT

- We have already seen GET and PUT
  - But we didn't talk about what they were ***supposed*** to be used for.
- GET is meant as a mechanism for querying data, provided that query has no side-effects.
- PUT is meant to permit you to add or modify existing data.
- They serve as opposites of each other.

POST & DELETE

- DELETE is self explanatory.
  - It deletes something on the server.
- POST, which we have used before, is a little different.
  - It allows ***anything*** to happen.
  - It provides “a black of data” as a response to a “data handling request”
- The distinction between PUT and POST is a subject of much debate!

REST Web Services

- A REST web service is one in which three elemtns are defined.
  - The location of the service (a URL)
  - The type of data supported by the service (we use XML, but other protocols are possible)
  - The set of operations that are supported.
- When we access a REST web service, the type of action we perform will influence the outcome.
- There are no set standards for REST web services.

SOAP versus REST - 1

- These are complementary systems.
  - They both have their strengths and weaknesses.
- REST has a much lower overhead.
  - Much less data to be transported.
  - Makes use of existing HTTP protocols.
- SOAP is standardised.
  - There is a set of protocol in a way that does not exist for REST services.
- SOAP makes no assumptions of the transport layer.

SOAP versus REST - 2

- REST requires the implementation of an HTTP stack on both systems.
  - This can make it problematic for some specialist devices.
- SOAP allows for the incorporation of further document standards, such as WS Security.
- For day-to-day purposes, REST is a more flexible and popular system.
- For enterprise use, SOAP has the edge.

WSDL

- The details of a web service are encoded in an XML document following the format defined as the Web Services Description Language (WDSL).
  - Only WSDL 2.0 supports the REST architecture.
- This defines all of the information we require to make use of a service.
  - URIs, parameters, and how to interpret output.
- It can be quite difficult to read, and so we usually rely on web-tools that provide the data in a more comprehensible form.

Consuming Web Services

- Let us look at an example of consuming a web-service defined in SOAP.
  - We will use the web service defined at:
    - http://www.webservicex.net/WS/WSDetails.aspx?WSID=9&CATID=2 for this.
- This web service permits us to provide the symbol for a corporation and receive stock information back.
  - We will use this as the basis for a stock query system.

Consuming a SOAP Service - 1

- To consume the SOAP service, we need to know the location of its WSDL.
  - This tells us what methods it exposes, what parameters it provides, and what the response will be.
- We will use SoapClient object to create the connection to the webservice.
  - We call methods on the service using -> notation.
- Parameters must be provided as an associative array.

Consuming a SOAP Service - 2

```
<?
      header('Content-Type: text/xml; charset=utf-8');

  $soap = new
SoapClient("http://www.webservicex.net/stockquote.asmx?WSDL");
  $result = $soap->GetQuote(array('symbol' => "goog"));

  $doc = new DOMDocument();

  $doc->loadXML ( $result->GetQuoteResult );

  echo $doc->saveXML();

?>
```

Response for GOOG

```
<StockQuotes>
        <Stock>
                <Symbol>GOOG</Symbol>
                <Last>493.65</Last>
                <Date>6/28/2011</Date>
                <Time>4:00pm</Time>
                <Change>0.00</Change>
                <Open>N/A</Open>
                <High>N/A</High>
                <Low>N/A</Low>
                <Volume>1929</Volume>
                <MktCap>159.1B</MktCap>
                <PreviousClose>493.65</PreviousClose>
                <PercentageChange>0.00%</PercentageChange>
                <AnnRange>433.63 - 642.96</AnnRange>
                <Earns>25.75</Earns>
                <P-E>19.17</P-E>
                <Name>Google Inc.</Name>
        </Stock>
</StockQuotes>
```

The Ajax Frontend - 1

- This PHP script creates the connection to the web service and parses it out as an XML document.
  - We can then navigate this in the ways we have already explored during the module.
- The nature of the XML document we get is not defined in the WSDL file.
  - We need to manually appraise it before working out how to display it.
- Luckily, it does not matter at all to DOM.

The Ajax Frontend - 2

```
function createTable (XML) {
     var table;
     var elements;
     var stock, last;
     var root;

     elements = XML.documentElement.getElementsByTagName("Stock");
     table = "<table border = \"1\">";

     table += "<tr>";
     table += "<th>Stock</th>";
     table += "<th>Last Value</th>";
     table += "</tr>";
     for (i = 0; i < elements.length; i++) {
       stock = elements[i].getElementsByTagName ("Symbol");
       last = elements[i].getElementsByTagName ("Last");

       table +=  "<tr>";
       table +=  "<td>" + stock[0].firstChild.nodeValue + "</td>";
       table +=  "<td>" + last[0].firstChild.nodeValue + "</td>";
       table += "</tr>";
     }
     table += "</table>";
     return table;
   }
```

Consuming a REST Service

- Consuming a REST service follows the same general structure.
  - Create the appropriate object
  - Configure it
  - Return an XML tree of the results.
- However, PHP has no built-in objects for handling REST communications.
  - This is one of the downsides of a ***lightweight*** architecture.
  - We have to do it manually.

CURL

- PHP comes equipped with a range of functions designed to simplify communication with remote resources.
  - It is called CURL.
- We use CURL to negotiate our REST requests.
  - We can use it for many other things also.
- To Setup a CURL request, we need to pass a URL to the curl_init function.
  - Let us use a different web service for this.

Google Directions

- The REST service we are going to consume for our example if the Google Directions web service.
  - http://code.google.com/apis/maps/documentation/directions/
- This needs us to provide three pieces of information:
  - The starting point
  - The destination
  - Whether our information comes from a device with a location sensor.

Consuming a REST Service

```
<?
  header('Content-Type: text/xml; charset=utf-8');

  $c = curl_init
      ("http://maps.googleapis.com/maps/api/directions/xml?se
       nsor=false&origin=edinburgh&destination=london");
  curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
  $responseXML = curl_exec($c);
  curl_close($c);

  $doc = new DOMDocument();
  $doc->loadXML ($responseXML);
  echo $doc->saveXML();
?>
```

Google Directions

- The document that we get from this is quite complex.
  - And full of things we do not need.
- However, it has the following nodes:
  - html_instructions
  - Distance
    - Value/Text
  - Duration
    - Value/Text
  - We already know how to parse this.

Mash-Ups - 1

- Many of the most vibrant new websites are ***mash-ups***.
  - Taking the processing of one service and integrating it with another.
- We can perform a simple mash-up here.
  - What if we knew where the user was located, and automatically gave them directions to a location?
- There is a web-service that tells us where an IP address is located:
  - https://www.easycounter.com/report/freegeoip.appspot.com

Mash-Up PHP - 1

```
<?
  header('Content-Type: text/xml; charset=utf-8');

  $IP = $_SERVER['REMOTE_ADDR'];
```

```
$c = curl_init ("https://www.easycounter.com/report/freegeoip.appspot.com");
Curl_setopt($c, CURLOPT_RETURNTRANSFER, true);
$responseXML = curl_exec($c) ;
Curl_close($c);
```

```
  $ipadd = new DOMDocument();
  $ipadd->loadXML ($responseXML);

  $response= $ipadd->getElementsByTagName ('Response')->item(0);
  $lat = $response->getElementsByTagName('Latitude')->item(0)-
>nodeValue;
  $long = $response->getElementsByTagName('Longitude')->item(0)-
>nodeValue;
```

Mash-Up PHP - 2

```
  $c2 = curl_init
("http://maps.googleapis.com/maps/api/directions/xml
?sensor=false&origin=$lat,$long&destination=london")
;
  curl_setopt($c2, CURLOPT_RETURNTRANSFER, true);
  $responseXML = curl_exec($c2);
  curl_close($c2);

  $doc= new DOMDocument();
  $doc->loadXML ($responseXML);
  echo $doc->saveXML();
?>
```

Ajax Frontend Table

```
function createTable (XML) {
     // Variables as normal
     elements = XML.documentElement.getElementsByTagName("step");

     // Set up usual headers for the table

     for (i = 0; i < elements.length; i++) {
       step = elements[i].getElementsByTagName ("html_instructions");
       len = elements[i].getElementsByTagName ("distance");
       dur = elements[i].getElementsByTagName ("duration");

       table +=  "<tr>";
       table +=  "<td>" + step[0].firstChild.nodeValue + "</td>";
       table +=  "<td>" +
         len[0].getElementsByTagName ("text")[0].firstChild.nodeValue +
"</td>";
       table +=  "<td>" +
         dur[0].getElementsByTagName ("text")[0].firstChild.nodeValue +
"</td>";
       table += "</tr>";
     }
     table += "</table>";
     return table;
   }
```

Mash-Ups 2

- The power of web-services is based on the easy deployment and consumption of distributed functionality.
- That in itself is based on the relative ubiquity of XML as a data format.
  - Every web service gives us a data structure that we know how to parse.
  - All we need to know is what the elements are and what they mean.

Mash-Ups - 3

- Modern web applications are becoming tremendously powerful.
  - And often leverage vast data sets out of the feasible grasp of independent developers.
- Sometimes gaining access to these applications is costly.
  - Sometimes you just need to register as a developer.
  - Sometimes you need to pay a fee.
- There is much value though to be had in innovative integration of these services.

Mash-Ups - 4

- Sometimes we only use a web-service to gain access to otherwise difficult functionality.
  - Validating credit cards
  - Address lookups
- It is risky to build an application around an external web service.
  - They are not always available.
  - The terms and conditions are not always stable.
- Sometimes though, you have no real choice.

Web Services - 1

- The web is full of services, both free and commercial.
  - http://www.programmableweb.com/ is a useful website that permits you to query what is out there.
- Most “big name” internet companies will have API that will allow you to leverage their power.
  - Many sites for example permit you to login via a facebook/twitter user account rather than through a bespoke login system.
- You need to weigh up the pros and cons ….

RSS

- RSS feeds are useful as they display up to date content which is delivered regularly to the subscriber.
- RSS feedback can be added to a website. The orange icon below identifies RSS feedback. This is linked to the BBC website and can be added to any website.

Conclusion

- Web services come in two main flavours – SOAP and REST.
- SOAP is a well-defined, enterprise level protocol for web services.
- REST is a more lightweight, ad hoc architecture.
- Consuming web services makes “free” complex functionality available to your web applications.
- But you need to be mindful of the potential consequences of utilising them.

Terminology

- *SOAP -* A well defined protocol for remove web application access
- *REST -* A lightweight architecture for accessing remote web functionality
- *Web service -* A remote web application that that permits for external applications to make use of its functionality
- *Mash-up -* A web application that integrates data from multiple web services.

References

- BBC, 2017. [online] Available at www.bbc.co.uk
- Googlemaps, 2017. [online] Available at : http://code.google.com/apis/maps/documentation/directions/
- Freegeoip.appspot.com, 2017. [online] Available at:http://freegeoip.appspot.com/
- Programmableweb.com, 2017. [online] Available at: http://www.programmableweb.com

Awarding Great British Qualifications

Topic 10 – Web Services

Any Questions?