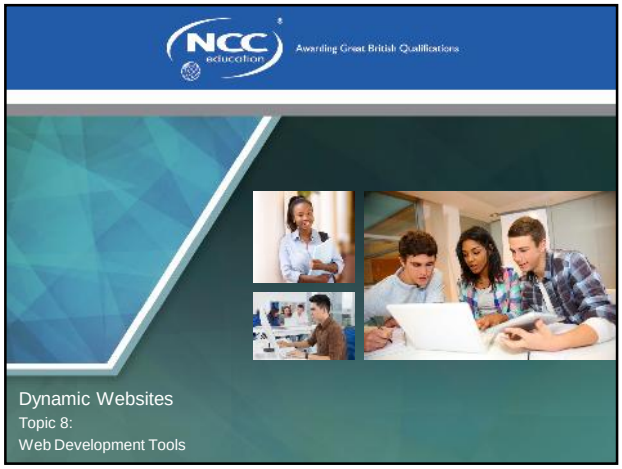


Scope and Coverage

This topic will cover:

- Using cookies to provide persistent data for PHP applications;
- Use sessions to provide persistent data for PHP applications;
- Use Ajax to build a database.

Learning Outcomes

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

- Understand cookies and sessions and how they can be used in a website;
- Use AJAX to create a database.

Introduction

- In this lecture, we will look at how cookies and sessions can be integrated into a website.
- We are going to expand our Ajax understanding so that we can profitably use it to create front-ends to our databases.
  - We will create an Ajax front-end that allows us to both query and manipulate a database.
- By the end of this lecture, you will be well placed to script compelling user interfaces for your users.

Cookies and Sessions - 1

- Cookies are files that are stored on a user's computer that contains certain pieces of information.
- Sessions fulfil the same role, but most of the information does not get stored on a user's computer.
- Cookies are declared before any HTML in a script and are available on the ***next page load*** by using the **setcookie function.**

Cookie Example

```
<?
        $thetext = $_POST["mytext"];
        setcookie ("textkeep", $thetext, time() + 10000);
?>

<html>
  <head>
    <title>Cookie Page</title>
  </head>
  <body>
  <?
        echo "<p>The post text  " . $_POST["mytext"] .
          ", we won't be able to pass that on.</p>";
  ?>

  <a href = "next_page.php">Onto the next page</a>

  </body>
</html>
```

The Next Page

```
<html>
  <head>
    <title>Passed it on</title>
  </head>
  <body>

  <?
      echo "<p>The post text is " . $_POST["mytext"] .
         ", we didn't get that passed on.</p>";
      echo "<p>The text is still " .
$_COOKIE["texttokeep"] .
         ", as we know from cookies.</p>";
  ?>

  </body>
</html>
```

Manipulating Cookies

- We can change the value of a cookie by altering it directly in the $_COOKIE variable:
  - $_COOKIE[“texttokeep”] = “Hello World”;
- Cookies can be deleted by setting an expiry date:
  - Setcookie (“texttokeep”, “”, time() – (60*60));

Sessions - 1

- Sessions fill the same role as cookies.
  - They are managed by a pair of cookies – one on the server and one on the client
- The Client cookie contains a reference to a session stored on the server.
  - The server manages the data for that session.
- To setup a session, we use the session_start function of PHP.
  - As with a cookie, this must come ***before*** any HTML is sent to the browser.

```
<?
Session_start();
?>
```

Sessions - 2

- Once you have a session open, you can register something as being a session variable, like so:
  - $_SESSION[“mytext”]=$mytext;
- This makes sure that the mytext variable is available on any other pages making use of the session.
- The variables are stored in the $_SESSION variables in the same way that cookies are.

Sessions Example

```
<?
      session_start();
?>

<html>
  <head>
    <title>Cookie Page</title>
  </head>
  <body>
  <?
        $mytext = $_POST["mytext"];
        echo "<p>The post text is $mytext and we'll register that "
              "in a session.</p>";
        $_SESSION["mytext"] = $mytext;
  ?>

  <a href = "session_next_page.php">Onto the next page</a>

  </body>
</html>
```

Session_next_page.php

```
<?
      session_start();
?>

<html>
  <head>
    <title>Passed it on</title>
  </head>
  <body>
  <?
      echo "<p>The session variable mytext is  " .
          $_SESSION["mytext"] . ".</p>";
  ?>

  </body>
</html>
```

Manipulation of Sessions

- Once a session has been created it is easy to manipulate through $_SESSION variable.
- Session data can be deleted through unset function:
  - Unset($_SESSION["something_sensitive"]);
- You can destroy a session using session_destroy.

A Simple Database

- Now we will create an Ajax front-end to a simple database.
  - It has two tables, ID And Description.
- We need to create this database on our server, which we ill do with a dedicated 'setup.php' file.
  - This creates the table and populates it with some basic test data.
- With Ajax, we must create pages that can handle our queries.
  - This is done using PHP.

Setup.php (Abridged)

```
<?
    $host = "localhost";
    $user = "mysql3_mmmmm";
    $pass  "ASdl";
    $database = "mysql3_mmm";
    $connection    mysql_connect($host, $user, $pass)
      or die ("Unable to connect to database");
    mysql_select_db ($database);

    $query   "DROP TABLE Things";

    $ret   mysql_query ($query, $connection);

    $query   "CREATE TABLE Things ( ID varchar (15), Description varchar (15) )";

    $ret   mysql_query ($query, $connection);

    $query   "INSERT INTO Things ( ID, Description) values ( \"1\", \"Blue Meanies\")";
    $ret = mysql_query ($query, $connection);
    $query   "INSERT INTO Things ( ID, Description) values ( \"2\", \"Yellow Submarines\")";
    $ret = mysql_query ($query, $connection);
    $query   "INSERT INTO Things ( ID, Description) values ( \"3\", \"Red Letter days\")";
    $ret = mysql_query ($query, $connection);
    $query   "INSERT INTO Things ( ID, Description) values ( \"4\", \"White christmases\")";
    $ret   mysql_query ($query, $connection);

    ?>
```

An Ajax Frontend

- We can already create an Ajax-front end to this.
  - It is just a little limited.
- In an ideal web application, we separate presentation from content.
  - We have not really been doing this so far.
- If it were the case that our PHP scripts were to be responsible for presentation, then it would be quite simple to create the front end.
  - We just change the URL for our Ajax requests.

Querying Content - 1

```
$host = 'localhost';
$user = 'monkeil3_nccuser';
$pass = 'ucul';
$database = 'monkeil3_ncc';

$thing = $_GET['thing'];

$connection  = mysql_connect($host, $user, $pass)
  or die ("Couldn't connect to database");
mysql_select_db ($database);

$query = 'SELECT * from Things where ID='$thing'';

$rsl = mysql_query ($query, $connection);

if (!$rsl) {
  echo "<p>Something went wrong: ' . mysql_error() + '</p>';
}

$num_results = mysql_num_rows ($rsl);
```

Querying Content - 2

```
if ($num_results == 0) {
  echo "<p>No such entry</p>";
}
else {
    echo "<table border  1>";
    echo "<tr>";
    echo "<th>ID</th>";
    echo "<th>Description</th>";
    echo "</tr>";

    for ($i = 0; $i < sizeof ($num_results); $i++) {
      $row = mysql_fetch_array ($rsl);
      echo "<tr>";
      echo "<td>" . $row['ID'] . "</td>";
      echo "<td>" . $row['Description'] . "</td>";
      echo "</tr>";
    }

    echo "</table>";

    mysql_close($connection);
}
?>
```

Ajax Frontend - 1

```
function makeAjax(term) {
    var text = form.myText.value;
    var url = 'crazy_content.php?thing=' + text;

    if (window.XMLHttpRequest) {
      // Code for modern browsers
      request=new XMLHttpRequest();
    }
    else {
      // code for older versions of Internet Explorer
      request = new ActiveXObject("Microsoft.XMLHTTP");
    }

    request.onreadystatechange=function() {
      if (request.readyState==4 && request.status==200) {
        document.getElementById("results").innerHTML= request.responseText;
      }
    }

    request.open ("GET", url, true);
    request.send();

  }
```

XML Output

- The XML discussion we had in a previous lecture is the foundation for this.
  - We want to output our data as an XML file and have Ajax format it for us.
- To do this, we need to discuss some new PHP syntax.
  - The creation and manipulation of a DOM file.
- This is done through the DOMDocument class.

Creating a DOM Tree

- We are going to manually construct this.
  - Luckily, the process is not complicated.
- At each step, we create a ***node***.
- We configure that node.
- We append it to a parent node (unless it is the root note).
- We then output it as the content of our PHP page.
- The important thing is not to lose track of what is being appended to what.

Creating a DOM Node

- We need a root note
  - This is the one to which all our records in the database will be appended.
- The syntax for this in PHP is as follows:
  - $doc = new DPMDocument();
  - $doc->formatOutput = true;
- Then within the loop over our results, we append the contents of results in turn to our root.

Iterating Over Results

```
for ($i = 0; $i < $num_results; $i++) {
    $row = mysql_fetch_array ($ret);

    $node = $doc->createElement( "Listing" );

    $name = $doc->createElement( "ID" );

    $name->appendChild($doc->createTextNode($row["ID"]));

    $node->appendChild( $name );

    $description = $doc->createElement( "Description" );

    $description->appendChild($doc
->createTextNode($row["Description"]));

    $node->appendChild( $description );

    $root->appendChild ($node);
}
```

Finally

- At the end, we use the saveXML method to output the contents of our DOM tree.
  - This gives us the document out as a simple string which we can echo in the normal way.
  - Echo $doc->saveXML();
- At the end of this, we get an XML document from our PHP script which we can then interpret and parse in our Ajax front-end.
  - Properly separating presentation from processing.

Serving an XML Document

- Unless we tell PHP otherwise, it will attempt to serve this as a standard HRML page.
  - We can overrise this by issuing a header directive:
  - header('Content-Type: text/xml; charset=utf-8')
- This ***must*** come before all other output (including whitespace).
  - When Ajax receives a document with this header information, the results go into ***responseXML*** rather than ***responseText***.
    - And we can then parse it as a DOM document.

The XML Document We Get

```
<?xml version="1.0"?>
<all_things>
  <thing>
    <ID>1</ID>
    <description>Blue Mean Max</description>
  </thing>

  <thing>
    <ID>2</ID>
    <description>Yellow Submarine</description>
  </thing>

  <thing>
    <ID>3</ID>
    <description>Red Letter Days</description>
  </thing>

  <thing>
    <ID>4</ID>
    <description>White Christmas</description>
  </thing>
</all_things>
```

Back to Ajax

- Our next step is to interpret this XML in Ajax.
  - This too involves some XML parsing of the document we obtain via our Ajax request.
- We use the responseXML property of our XMLHttpRequest objective for this, rather than responseText.
- To begin with, we will convert the XML we get into a table representation within our HTML pages.
  - and then look at other ways to spruce up our application.

Interpreting the DOM Tree

- We do not need to do anything extra to get a DOM tree.
  - That is handled for us by Ajax.
- Getting an array that contains all of our things is easy:
  - Elements = XML.documentElement.getElementsbyTagName("thing");
- We can iterate over this array to construct our table in Ajax.
  - To do that, we need to understand what is in a node.

Ajax Create Table Function

```
function createTable (XML) {
    var table;
    var elements;
    var id, description;

    elements = XML.documentElement.getElementsByTagName("thing");

    table = "<table border = \'1\'>";

    table += '<tr>';
    table += "<th>ID</th>";
    table += "<th>Description</th>";
    table += '</tr>';
    for (i = 0; i < elements.length; i++) {
      id = elements[i].getElementsByTagName ("ID");
      description = elements[i].getElementsByTagName ("description");

      table +=  "<tr>";
      table += "<td>" + id[0].firstChild.nodeValue + "</td>";
      table +=  "<td>" + description[0].firstChild.nodeValue + "</td>";
      table += '</tr>';
    }
    table += '</table>';
    return table;
}
```

Outputting the Table

- The responseXML property contains the formatted DOM tree.
  - We just pass that to our create table function to create our output.

```
request.onreadystatechange=function() {
    if (request.readyState==4 && request.status==200) {
      document.getElementById("results").innerHTML=
createTable
            (request.responseXML);
    }
}
```

Browsing the Database

- We are going to populate a combo box that contains all the valid user IDs in our database.
  - There are other techniques we can use, but this is the one for us.
- To do this, we need to adjust our PHP page so that we can query a full table if no parameters are provided:

```
if (isset($thing)) {
  $query = "SELECT * from Things where ID= $thing";
}
else {
  $query = "SELECT * from Things";
}
```

Populating the Combo Box - 1

- We populate the combo box in the same way we built the table.
  - Construct the HTML.
  - Place it somewhere on the form.
- Assume that we have a ***select*** form element called data.
  - We want to put the options between the opening and closing tags for that element.
- This is something we can do.

Populating the Combo Box - 2

```
function updateComboBox() {
    var url;

    url = 'query_content_xml.php';

    // Usual code for creating an XMLHttpRequest object goes here.

    request.onreadystatechange=function() {
      if (request.readyState==4 && request.status==200) {
        var text = "";
        var elements;
        var id;
        elements = request.responseXML.documentElement.getElementsByTagName('thing');

        for (i = 0; i < elements.length; i++) {
          id = elements[i].getElementsByTagName ('id');
          text += '<option>' + id[0].firstChild.nodeValue + '</option>';
        }
        document.getElementById ('data').innerHTML = text;
      }
    }
    request.open ('GET', url, true);
    request.send();
  }
```

Populating the Combo Box - 3

- We bind this into the load event of out HTML page.
  - That goes into onLoad event handler of the <body> tag.
- Next, we need to create a function that lets us query the database for the description associated with an ID.
  - We will notify our setup Ajax function to do this, to improve the modularity of our code.
- We bind this function into the onChange event handler of our Select element.

Navigating the Database - 1

```
function setupAjax(xml,id) {
    var url;

    // Usual code for creating an XMLHttpRequest object goes here.
    request.onreadystatechange=function() {
      if (request.readyState==4 && request.status==200) {
        if (request.responseXML) {
          updateFrontend (request.responseXML);
        }
      }
    };

    request.open ("GET", url, true);
    request.send();
  }

  function navigateDatabase (form) {
    var url;
    var id;

    id = form.data.value;
    url = "query.content.xml.php?thing=" + id;
    setupAjax (url).
  }
```

Navigating the Database - 2

```
function updateFrontend (XML) {
    var form = document.getElementById("mainForm")
    var elements =
XML.documentElement.getElementsByTagName("thing");
    var id, description;

    if (elements.length == 0) {
      document.getElementById ("id").innerHTML = "";
      form.description.value = "";
    }
    else {
      description = elements[0].getElementsByTagName
("description");
      form.description.value = description[0].firstChild.nodeValue;
    }
  }
```

Updating the Database

- Updating the database requires both a new function in our front-end, and a PHP script on the server.

```
function updateDatabase (form) {
    var url
    var desc;
    var id;

    id = form.data.value;
    desc = form.description.value;

    if (desc.length == 0) {
      return;
    }

    url = 'update_content_ajax.php?id=' + id + "&description=" + desc;
    setup4jax (url);
  }
```

Updating the Database - PHP

```
<?
  $host = "localhost";
  $user = "nonkaiii nonkaw";
  $pass = "pass";
  $database = "nonkaiii_nonc";

  $id = $_GET["id"]
  $description = $_GET["description"];

  $connection = mysql_connect($host, $user, $pass)
    or die ("Problem: Connect to database");
  mysql_select_db ($database);

    $id = mysql_real_escape_string ($id);
  $description = mysql_real_escape_string ($description);

  $query = "UPDATE Things SET Description = '$description' WHERE ID='$id'";

  $out = mysql_query ($query, $connection);

  mysql_close($connection);

?>
```

The HTML

- The HTML that defines our static code is very simple, setting up only the containers and the event handlers:

```
<body onload="updateCombobox ()">
    <script language="JavaScript">
    // Look good here
    </script>

    <form name = "mainForm" id = "mainForm">
      <p>id</p>
      <select id = "data" onChange="navigateDatabase(this.form)">
      </select>

      <p>description</p>
      <input type = 'text' name = 'description' onBlur="updateDatabase (this.form)">
    </form>
  </body>
</html>
```

The Result

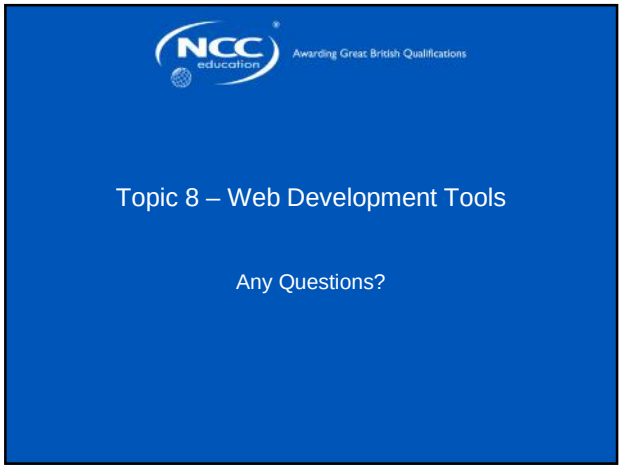
- The result is a simple dynamic application that uses Ajax to create a seamless user experience.
- An important element of the design here is that we have progressed from using PHP to handle our presentation.
  - It is now a job for JavaScript and Ajax.
- The main reason for this is to ensure ***modularity***.
  - We can easily swap out back-end and front-end elements if their roles are well defined.

Conclusion

- At this point, you are capable of creating very rich and interactive dynamic websites for data driven applications.
- In the next topic you will look at integrating more mobile technologies with website design and how web services can be used to enhance the website.

References

- W3.schools.com, 2017. [online] Available at www.w3schools.com


NCC education
Awarding Great British Qualifications
Topic 8 – Web Development Tools
Any Questions?