AJAX and JSON

Day 8
Overview

- HTTP as a data exchange protocol
- Components of AJAX
- JSON and XML
- XMLHttpRequest Object
- Updating the HTML document

References

- Duckett, chapter 8
- http://www.w3schools.com/ajax/default.ASP
Traditional Web Application

- In the beginning there were web applications
- Easy to build … pages are just static views
Problems..

- Requests to the server to update the data means a whole page refresh
- And the browser is effectively locked to the user while it fetches the page using HTTP
  - Think about Word locking up for seconds every time you scrolled to a new page..

- Problem is network latency and the need to refresh the complete page when comparatively little information changes
Train times & tickets

Leicester [LEI] to Hereford [HFD]

Outward Wed 02 Apr

Earlier trains

<table>
<thead>
<tr>
<th>Dep.</th>
<th>From</th>
<th>To</th>
<th>Arr.</th>
<th>Dur.</th>
<th>Chg.</th>
<th>Status</th>
<th>Buy now</th>
<th>Single from</th>
<th>Cheapest fare</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:46</td>
<td>Leicester [LEI] Platform 3</td>
<td>Hereford [HFD] Platform 2</td>
<td>18:51</td>
<td>3h 03m</td>
<td>1</td>
<td>Details</td>
<td>£34.60</td>
<td>Buy now</td>
<td>£34.60</td>
</tr>
<tr>
<td>16:18</td>
<td>Leicester [LEI] Platform 1</td>
<td>Hereford [HFD] Platform 1</td>
<td>19:24</td>
<td>3h 06m</td>
<td>1</td>
<td>Details</td>
<td>£34.60</td>
<td>Buy now</td>
<td>£34.60</td>
</tr>
<tr>
<td>16:48</td>
<td>Leicester [LEI] Platform 3</td>
<td>Hereford [HFD] Platform 1</td>
<td>20:29</td>
<td>3h 41m</td>
<td>2</td>
<td>Details</td>
<td>£34.60</td>
<td>Buy now</td>
<td>£34.60</td>
</tr>
<tr>
<td>17:48</td>
<td>Leicester [LEI] Platform 1</td>
<td>Hereford [HFD] Platform 2</td>
<td>21:03</td>
<td>3h 15m</td>
<td>1</td>
<td>Details</td>
<td>£34.60</td>
<td>Buy now</td>
<td>£34.60</td>
</tr>
<tr>
<td>18:48</td>
<td>Leicester [LEI] Platform 1</td>
<td>Hereford [HFD] Platform 1</td>
<td>22:29</td>
<td>2h 41m</td>
<td>1</td>
<td>Details</td>
<td>£34.60</td>
<td>Buy now</td>
<td>£34.60</td>
</tr>
</tbody>
</table>

Long journey? Why not upgrade to First Class from £54.40
HTTP - data exchange protocol

- Hypertext Transfer Protocol
- agreed set of requests between computers connected by means of TCP/IP
- the protocols are built on top of each other
- TCP/IP provides reliable communication ensuring no data is lost
- HTTP negotiates about what data will be transferred
HTTP - methods

- **Methods**
  - **GET**
    - retrieve a URL from the server
      - simple page request
      - run a CGI program
      - run a CGI with arguments attached to the URL
  - **POST**
    - preferred method for forms processing
    - run a CGI program
    - more secure and private
Use 'Network' option to view traffic
PUT
- Used to transfer a file from the client to the server

HEAD
- requests URLs status header only
- used for conditional URL handling for performance enhancement schemes
  - retrieve URL only if not in local cache or date is more recent than cached copy
Solutions..

- I need to be able to start something off, do something else and come to the job when it's ready.
- Think about making a cup of tea, putting the tea bag in the cup, fill the kettle and switch it on..
  - … then wait doing nothing until it’s boiled..
  - …then fill the cup

- I want to start the kettle boiling and do something else meanwhile..

... I want to be asynchronous
AJAX: Asynchronous JavaScript and XML

- Browser calls for data from the server (asynchronously) and updates only those elements displaying the data.
- Steps are:
  - ask for the file from the server
  - wait until it arrives
  - unpack the file into a collection of data objects
  - update a collection of html objects on document page
- The transfer may be
  - in response to a user command (‘Show earlier trains’)
  - in case the user might want the data (Google Maps uses this extensively)
AJAX Architecture

- **HTML** for the data presentation of the view layer
- **DOM** (Document Object Model) to dynamically manipulate the presentation
- **XML or JSON** for data exchange
- **XMLHttpRequest** to create a direct connection between the browser and the Web server

- All tied together by **Javascript**....
JSON

- JavaScript Object Notation
- data is transferred from server to client as a string
- converted into a JavaScript object in the browser
- simple example 'names1.json'

```json
{
    "name": "Bob",
    "age": 31
}
```
JSON - still just a string

```json
[{  
    "name": "Bob",
    "age": 31
},
{
    "name": "Brian",
    "age": 35
},
{
    "name": "Harry",
    "age": 27
},
{
    "name": "Bill",
    "age": 35
}]
```
JSON - still just a string

- name:value pairs
- some values are objects
- some values are arrays of objects
- here there is one pair
  
  
- value is an object

```json
{
  "timedata": {
    "day": "Fri",
    "date": "28 Feb 2014",
    "departure": {
      "placename": "Leicester",
      "placecode": "LEI"
    },
    "destination": {
      "placename": "Hereford",
      "placecode": "HFD"
    },
    "journey": [
      {
        "depart": "09:53",
        "arrive": "12:54",
        "duration": "3.01",
        "changes": "2"
      },
      {
        "depart": "11:00",
        "arrive": "13.28",
        "duration": "2.44",
        "changes": "1"
      }
    ]
  }
}
```
XML (eXtensible Markup Language)

- Simplified version of SGML, enabling users to define their own language
- Tags not concerned with how to render data, but instead define content
  - (XML) `<employee>` indicates that data is about employees
  - HTML5 has moved towards emphasis on content - not presentation
- Definition is either implicit (deduced from document structure) or is explicit (defined in Document Type Definition or DTD)
<timedata>
  <day>Fri</day>  
  <date>28 Feb 2014</date>

  <departure>
    <placename>Leicester</placename>
    <placecode>LEI</placecode>
  </departure>

  <destination>
    <placename>Hereford</placename>
    <placecode>HFD</placecode>
  </destination>

  <journey>
    <depart>09:53</depart>
    <arrive>12:54</arrive>
    <duration>3.01</duration>
    <changes>2</changes>
  </journey>

  <journey>
    <depart>11:00</depart>
    <arrive>13.28</arrive>
    <duration>2.44</duration>
    <changes>1</changes>
  </journey>
</timedata>
XMLHttpRequest Object

- Enables web clients to retrieve and submit XML data in the background (asynchronously)
- First implemented by Microsoft as an ActiveX object in IE5
- Modern browsers implement this as a native object

```javascript
var xhr = new XMLHttpRequest();
```
Sending an HTTP request using GET

```javascript
var xhr = new XMLHttpRequest();
xhr.overrideMimeType("application/json");

function getJsonData(){
    xhr.open('GET','data/names1.json',true);
    xhr.send(null);
}

xhr.onload = function(){
    if (xhr.status === 200){
        responseObject = JSON.parse(xhr.responseText);
    }
}
```
Responding to the request

```javascript
var xhr = new XMLHttpRequest();
xhr.overrideMimeType("application/json");

function getJsonData(){
    xhr.open('GET','data/names1.json',true);
    xhr.send(null);
}

xhr.onload = function(){
    if (xhr.status === 200){
        responseObject = JSON.parse(xhr.responseText);
        myfunctionToProcessData();
    }
}
```
# XMLHttpRequest Properties and Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new XMLHttpRequest()</code></td>
<td>Creates a new XMLHttpRequest object</td>
</tr>
</tbody>
</table>
| `open(method, url, async)` | Specifies the type of request  
  *method*: the type of request: GET or POST  
  *url*: the file location  
  *async*: true (asynchronous) or false (synchronous) |
| `send()`          | Sends a request to the server (used for GET)     |
| `send(string)`    | Sends a request string to the server (used for POST) |
| `onreadystatechange` | A function to be called when the `readyState` property changes |
| `readyState`      | The status of the XMLHttpRequest  
  0: request not initialized  
  1: server connection established  
  2: request received  
  3: processing request  
  4: request finished and response is ready |
| `status`          | 200: OK                                           
  404: Page not found |
| `responseText`    | The response data as a string                     |
| `responseXML`     | The response data as XML data                     |
Updating the page

- create new elements containing the data and append these to a container element (usually a <div>)
- assign single values
  - build a collection of HTML elements
  - build an equivalent collection of XML elements
  - loop though the HTML collection and assign the new values from the XML collection
Updating via new elements

- Create the element
- Build a string containing the HTML inside the element
- Use innerHTML to add the HTML to the element
- Add the new element to the container element

```javascript
function createDOMObject(item) {

  var newItem = document.createElement('div');
  newItem.setAttribute('class', 'newsitem');
  var content = '<div class='textinfo'>'
  content += 'name: ' + item.name + ', age: ' + item.age + '<br>'';
  content += '</div>';
  newItem.innerHTML = content;
  containerElem.appendChild(newItem);
}
```
Generating the JavaScript

- Produce the element with HTML and CSS classes in the HTML document
- Copy and paste the HTML markup as a string in the JavaScript function
- Test that it works
- Delete the original HTML element
Update via collections

- we build a collection of similar data elements from the XML document represented by the request object

```javascript
req = new XMLHttpRequest();
...
var depart_data = req.responseXML.getElementsByTagName("depart");
```

- we can build a collection of elements from the html document and assign the nodeValue from the data element to the page element

```javascript
departtd[i +1].firstChild.nodeValue = depart_data[i].firstChild.nodeValue;
```
data elements in xml document

```xml
<timedata>
  <journey>
    <depart>09:53</depart>
    <arrive>12:54</arrive>
    <duration>3.01</duration>
    <changes>2</changes>
  </journey>

  <journey>
    <depart>11:00</depart>
    <arrive>13.28</arrive>
    <duration>2.44</duration>
    <changes>1</changes>
  </journey>

</timedata>
```

- the `<depart>` element has a text node as its firstChild
References

- (override MIME type)