Data from web services

Lecture 7

Case study: Book store

- Two data collections:
  - Data about books in inventory.json
  - Data about sales in sales.json

- Task: create Web application which allows to add new sales data and keep track of sales on any device with a modern Web browser

From last lecture: steps of implementing XMLHTTPRequest (GET)

- Create an XMLHTTPRequest object
- Issue a GET request to retrieve the data file from server
- Handle the request

Two types of requests: GET and POST

```
request.open ("GET", "blog.xml", true);
request.send (null);

request.open("POST", "addblogentry.php", true);
request.setRequestHeader ("Content-Type",
"application/x-www-form-urlencoded;
charset=UTF-8");
request.send(“date=09/26/2008&text=These
dreams just...");
```

GET and POST requests

The type of an Ajax request reflects the intent of the request:

- **GET** – to **retrieve** data from server without affecting anything on the server
- **POST** - to **send** data to the server, after which the state of the server usually changes in response to the data that was sent

GETting book list: 1/3

```
window.onload = function ()
{
    loadInventory();
}
```

GETting book list: 2/3

```
function loadInventory () {
    var url = "inventory.json";

    try    {
        var ajax = new XMLHttpRequest();
        ajax.open("GET", url);

        ajax.onreadystatechange = function () {
            ...
        };
        ajax.send(null);
          }
     catch (e)    {   alert (e.message);    }
}
```

GETting book list: 3/3

```
function loadInventory () {
    ...
    var ajax = new XMLHttpRequest();
    ajax.open("GET", url);

    ajax.onreadystatechange = function () {
        if( ajax.readyState === 4)
            loadSales (ajax.responseText);
    };
    ...
}
```

Ajax requests are asynchronous

- While the server processes a request, the browser page is fully functional
- When each request is complete, the callback function is called, which then processes or displays data returned from the server

GETting sales data
When? After the first request is complete

```
function loadSales () {
   var url = “sales.json";

   try    {
   var ajax = new XMLHttpRequest();
   ajax.open("GET", url);
   ajax.overrideMimeType("text/javascript");
   ajax.onreadystatechange = function ()  {
      if( ajax.readyState === 4)
         displaySales (inventoryText, ajax.responseText);
   };
   ajax.send(null);
          }
    catch (e)   {  alert (e.message);   }
}
```

Ajax ready state

readyState

The numeric state of the request:

- 0 (uninitialized)
- 1 (open)
- 2 (sent)
- 3 (receiving)
- 4 (loaded)

Example of different states in onreadystatechange event handler: (link)

Response handler

- The response handler function processes the data which arrives in Ajax response from the server

- Two methods to get the data:
  - **getResponseXML**(): if the response data is XML
  - **getResponseText**(): if the response data is plain text

By applying **JSON.parse**, we convert plain text into JavaScript objects

Populating sales data using DOM

```
function displaySales (inventoryText,
                                        salesText)
{
    var sales=JSON.parse(salesText);
    var books = JSON.parse(inventoryText);


    ...
}
```

Populating sales data using DOM

```
    ...
var salesDiv =document.getElementById("sales");

for (var i=0; i< sales.length; i++){
    var sale = sales [i];
    var book = books[sale.id];
    var message = ... + book.category + (book.price
            *sale.amount).toFixed(2);

    var div = document.createElement ("div");
    div.innerHTML = message;
    salesDiv.appendChild (div);
}
```

Full example

- Demo: link
- In order to change sales data: we need to open file sales.json in LectureFiles folder, edit its content, and save – on server.
- How can we add new sales through Web page user interface?

POSTing new sales

Steps of adding new sales

1. Create a page with input fields
2. Write button “Add” handler:
    2.1. Collect values from input fields
    2.2. Open Ajax request with POST method
    2.3. Send collected values as HTTP parameters of this request

1. Create a page with input fields: index.html

```
Date: <input type="text" id="date" value="" size="10" />
Book ID: <input ... id="bookid" value="" size="4"/>
Amount: <input ... id="amount" value="" size="4" />
<input type="button" id="add" value="Add "  />
```

2. Write *button Add* handler: addSale

```
window.onload = function ()
{
  document.getElementById("date").value =
                (new Date()).shortFormat();
  document.getElementById("bookid").focus();
  document.getElementById("add").addEventListener
              ("click", addSale, false);
}
```

2.1. Collect values from input fields

```
function addSale () {
    var saleDate =
    document.getElementById("date").value;
    var bookID =
    document.getElementById("bookid").value;
    var amount =
        document.getElementById("amount").value;
    ...
}
```

2.2. Open Ajax request with POST method

```
function addSale () {
...
    var ajax = new XMLHttpRequest();
    ajax.open("POST",url,true);
    ajax.setRequestHeader("Content-type",
            "application/x-www-form-urlencoded");
...
}
```

2.3. Send values as HTTP parameters

```
function addSale () {
...
    ajax.send("&date=" + saleDate +
            "&bookid=" + bookID
"&amount=" + amount);
}
```

What happens with data POSTed on the server?

- We need to write data to file
- Client-side JavaScript (Ajax) cannot write files on the server: JavaScript (as we know it) cannot access files on server
- We need to write a script for processing POSTed data on server

Simple solution: PHP

PHP code for writing new data into saleswithdates.json

(link)

To invoke PHP script, set url parameter of Ajax request to:

var **url** = "addsaleentry.php";

To run the rest of demos

1. Download the entire lecture folder local copy, unzip it
2. Install web server and php

Install PHP

- To enable server to run PHP scripts, install PHP:

  http://windows.php.net/download/

- For simplicity (on Windows), select windows installer (msi) file.
- Run the installer and install PHP into a directory of your choice (for example, C:\PHP)

Start web server

Now, we need a web server.

A simple way to install and run web server – download and run Mongoose web server (.exe for Windows is in folder demo) :

https://code.google.com/p/mongoose/

After starting the server, right-click on the Mongoose icon, and go to Advanced settings. Here, you need to tell the server the path to cgi_interpreter: C:\PHP\php-cgi.exe.

Save settings to the config file.

To start the server: double-click on mongoose.exe

To shut it down: right-click on the icon and select exit.

PHP script: load existing file

```
//Declaring a variable for file name
$filename = "saleswithdates.json";
if (file_exists($filename)) {
    // Load the text from the json file
    $textSales = file_get_contents($filename);
```

PHP script: decode JSON object into PHP array

```
...
$textSales = file_get_contents($filename);
$dataToAugment = json_decode($textSales);
```

PHP script: add new entry (PHP associative array) to the end of data array

```
$dataToAugment = json_decode($textSales);
$dataToAugment [count($dataToAugment)] =
array
    ("date"=>$_REQUEST["date"],
    "id"=>$_REQUEST["bookid"],
    "amount"=>$_REQUEST["amount"]);
```

PHP script: encode PHP array to JSON

```
$data = json_encode($dataToAugment);
```

PHP script: write new data to a file

```
$file = fopen($filename, 'w');
fwrite($file, $data);
fclose($file);
```

Adding new sales data: demo

Start the server now.

In the browser go to:

http://localhost:8080/addnewbooks/

Getting sales report: demo

- http://localhost:8080/addnewbooks/getsales.html

That is all we need to know about web applications with Ajax and PHP:
we can get data from a file on server in JSON format and
we can add new data through a web page

Getting data in another app

- Scenario: We are now working in another company, and we want to get access to sales data located on an original server.

- For an example, consider a server which stores a discography of all artists.
- We know that we can get a discography in json format using the following url:

  http://lyrics.wikia.com/api.php?func=getSong&artist=Linkin_Park&fmt=json

Now, suppose that we issue Ajax request to get the same JSON object into our web application.

Reminder: XMLHttpRequest security policy

Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://lyrics.wikia.com/api.php?func=getSong&artist=Linkin_Park&fmt=json? This can be fixed by moving the resource to the same domain or enabling CORS

- Ajax request cannot be run from a web page stored on your hard drive
- It can only be run from a web page stored on a web server and downloaded to your browser
- **It can only fetch files from the same site that the page is on**
  - http://www.foo.com/a/b/c.html can only connect to files on www.foo.com

Browser security policy: why

- Say you’re running the site for DaddyBucksBank.com
- Someone has hacked into your system and inserted a bit of JavaScript that takes the user’s personal information and transfers it to the server HackersNeedMoreMoney.com.
- To prevent this, browsers do not allow making XMLHttpRequests to domains other than the original domain the page was served from.

Normal web app

- If we develop sales application, we put all files on the same server.
- Thus we can use Ajax to exchange server data between different pages from the same domain.

Wanting data from a different domain

- What if somebody wants to use our book data to create an advertising site for our company?
- Or just get all new published books?

- Can we make our book data accessible for everybody on the web?

(Like Google maps, Twitter tweets, Facebook graph…)

Web services

We want to convert our data to a **Web service**

- A **Web service** is a software function provided at a network address over the web, it is a service that is “always on”.

- A RESTful web service (also called a RESTful web API) is a web service implemented using HTTP: it constrains the interface to a set of standard operations (like GET, POST..).

  REST = REpresentational State Transfer

W3C definition "We can identify two major classes of Web services,

- REST-compliant Web services, in which the primary purpose of the service is to manipulate Web resources (data);
- Arbitrary Web services, in which the service may expose an arbitrary set of operations

JSON with Padding ?

WEB SERVICES WITH JSONP

So, how can we get data from a different domain into our page?

Code in a script tag

- What does the following code do?

```
alert (“woof”);
```

- If we save this code in “dog1.js” and create an HTML page with the following script tag:

```
<html
<body>
   <script src="dog1.js"> </script>
</body>
</html>
```

- What happens when we load the page? (link)

Script from a different domain

- What if we put script file: dog1.js on the server, and reference it in our page as:

```
<script src="http://domain/animals/dog1.js">
```

- What will happen when we load this page? (link)

Conclusion: a JavaScript code from a different domain can be executed in your browser

Interaction between our script and script from another domain

```
animalSays("dog", "woof");
```

- This code is on the server in:

http://domain/animals/dog2.js

- We write our own function *animalSays* in our HTML page

```
function animalSays(type, sound) {
    alert(type + " says " + sound);
    }
```

Conclusion: We can bring arguments to our custom function from a different domain

We can bring different arguments (data) by using different script url

cat2.js

animal2.html

So not only can a JavaScript file that was served from another domain **call any function** it wants in your code, but it can also **pass to it any data** it wants?

Passing arguments as JS objects

- In our page we define general function:

```
function animalSays(animal) {
    alert(animal.type + " says " + animal.sound);
}
```

- And we reference the data file as:

```
<script src="http://domain/animals/cat3.js">
</script>
```

- Which contains data in form of arguments:

```
animalSays({"type": "cat", "sound": "meow"});
```

Getting sales data from a different domain

- What if we rename *animalSays* to *displaySales*, and on server store in **sales.js**:

```
var sales = [{"id":0,"amount":"2"},
   {"id":1,"amount":"5"},
   {"id":0,"amount":"1"}]
displaySales(sales);
```

- And include the following script tag in our page:

```
<script src="http://domain/bookstore/sales.js">
</script>
```

Passing data through JS file

- We are passing data through the JavaScript file we’re referencing, rather than using XMLHttpRequest to retrieve it ourselves.

- Are we not also getting it from another domain? Something that is forbidden by XMLHttpRequest.

Meet JSON-P

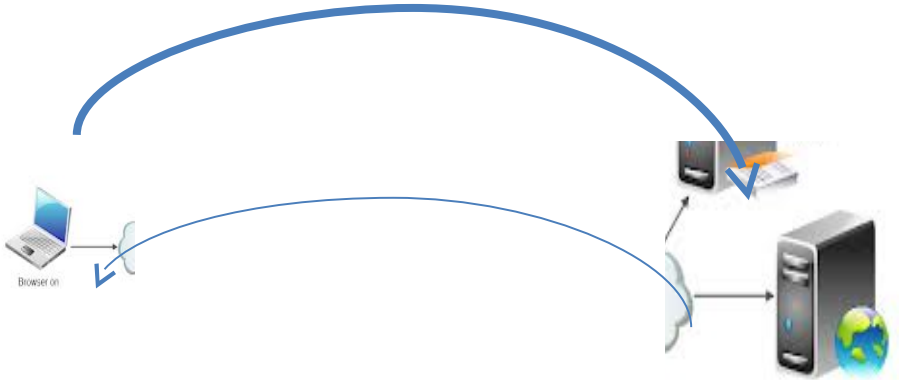
- JSONP is a way to retrieve JSON objects by using the <script> tag.
- It’s also a way of retrieving data (again, in the form of JSON objects) that avoids the same-origin security issues we saw with XMLHttpRequest.

How JSONP works

Sales.html

```
<body>
<h1>Book Sales</h1>
<div id="sales">
</div>
</script>
<script
src="http://domain/bookstore/sales.js"
>
</script>
</body>
```

HTTP request: get me sales.js file



HTTP response: here is your file

At this point file is just a **plain text**

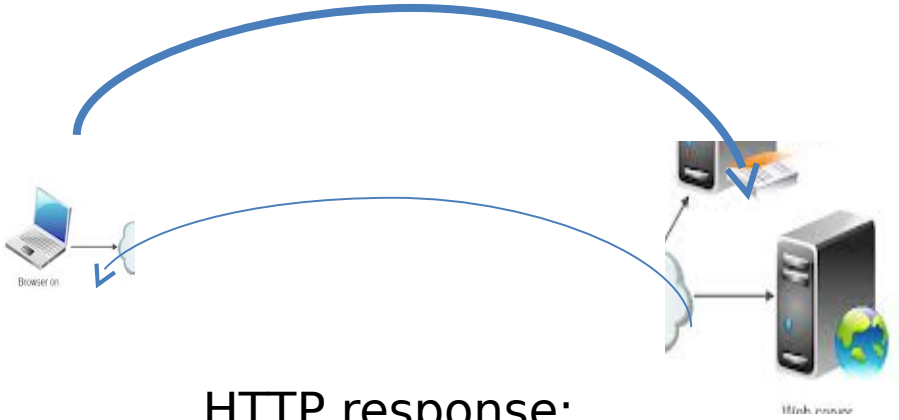
When the file reaches the browser, it contains **JS objects and function calls**, and it is parsed and interpreted by the browser, getting data from the server into your web page.

Why JSON with Padding

Sales.html

```
<body>
<h1>Book Sales</h1>
<div id="sales">
</div>
</script>
<script
src="http://domain/bookstore/sales.js"
>
</script>
</body>
```

HTTP request: get me sales.js file



HTTP response:
here is your data (JSON object)

displaySales (sales)

Data is padded with our custom function when it gets back

Wrapping JSON data in a function call

- In general, web services allow you to specify what you want the padding function to be named.
- Inventory and sales are our web services and we made them to support this functionality
- Here's how it works: when you specify your URL, add a parameter at the end, like this:

```
<script src="http://domain/inventory?callback=readInventory">
</script>
<script src="http://domain/sales?callback=displaySales">
</script>
```

Getting sales data from one server to another

- Suppose, our web service is located at one server:

http://**host1**/sales/

- If we type this in browser, it displays JSON sales array

http://localhost:8080/bookstoreservice/sales/

- The same data with callback:

http://localhost:8080/bookstoreservice/sales/?callback=displaySales

- If we type this, it returns the same object wrapped in a function call: dispaySales()

- Now we can get sales data into any page on any server:

http://localhost:8080/bookstore/getsalesJSONP.html

Simple way to create web services with PHP

- Create index.php in a separate folder
- This folder will be specified in the url for people who want to access your data:

http://localhost:8080/bookstoreservice/inventory/

http://localhost:8080/bookstoreservice/sales/

- In index.php
  - read JSON data into an object
  - get callback function from an HTTP GET parameter
  - wrap data with this function and return wrapped data

Web services with PHP (link)

```php
<?php
$filename = "saleswithdates.json";
$data = file_get_contents($filename); // json string

if(array_key_exists('callback', $_GET)){
    header('Content-Type: text/javascript; charset=utf8');
...
    $callback = $_GET['callback'];
    echo $callback.'('.$data.');';

}else{
    // normal JSON string
    header('Content-Type: application/json; charset=utf8');
    echo $data;
}
?>
```

Note: data comes as an object, not a string

- When we used XMLHttpRequest, the data came back in the form of a **string**.
- We needed to call **JSON.parse** to convert this string to JS object

```
function displaySales (inventoryText,
salesText)
{
    var sales=JSON.parse(salesText);
    var books=JSON.parse(inventoryText);
    for (var i=0; i< sales.length; i++) {
        var sale = sales [i];
        var book = books[sale.id];
...
}
```

Note: data comes as an object, not a string

- When we use the <script> element, we're telling the browser that it needs to retrieve JavaScript, - the browser retrieves it, parses it and evaluates it.
- By the time it gets to your displaySales function, the JSON data is no longer in string form, but is a first-class JavaScript **object**.

```
function displaySales (sales)
{
   for (var i=0; i< sales.length; i++)
  {

    var sale = sales [i];
    var book = books[sale.id];
   ...
}
```

JSONP problems?

- To retrieve data with JSONP it takes ZERO code. Just a simple HTML <script> tag.

- What if we need to do something dynamically, where you need to retrieve something over and over again?

Iterative data requests: solution

- Create a new <script> element any time we want the browser to do a JSONP-type operation for us.

- Example: We need to make the JSONP request every few seconds. We'll use JavaScript's *setInterval* method.

```
window.onload = function() {
    setInterval (handleRefresh, 3000);
}
```

Adding script tag dynamically

```
function handleRefresh() {
    var url = "http://localhost/bookstoreservices/sales/?
callback=updateSales" ;

    var newScriptElement =
            document.createElement("script");
    newScriptElement.setAttribute("src", url);
    newScriptElement.setAttribute("id", "jsonpSales");
```

Adding script tag dynamically

```
function handleRefresh() {
    ...
    var oldScriptElement =
            document.getElementById("jsonpSales");
    var head = document.getElementsByTagName("head")[0];
    if (oldScriptElement == null) {
        head.appendChild(newScriptElement);
    }
    else {
        head.replaceChild(newScriptElement,
                                        oldScriptElement);
    }
}
```

Refreshing data on timer events

Live demo on your local host (don't forget to start a server):

http://localhost:8080/bookstore/getsalesJSONPRefresh.html

Watch out for browser cache

- Most browsers have an interesting property in that if you retrieve the same URL over and over (like with our iterative JSONP request), the browser ends up caching it for efficiency, and so you just get the same cached file (or data) back over and over

- There is an easy and old-as-the-Web cure for this:

All we do is add a random number onto the end of the URL, and then the browser is tricked into thinking it's a new URL the browser's never seen before:

```
var url = "http://domain/sales?callback=updateSales"
+"&random=" + (new Date()).getTime();
```

What have we done

- We have learned JSONP – an alternative to XMLHttpRequest for accessing data made available by web services.
- Now we can dynamically retrieve data from any domain (which exposes its data) into our own web application.

What does it give us

We can embed, process, visualize a variety of data provided by REST APIs:

http://en.wikipedia.org/w/api.php

http://policeapi2.rkh.co.uk/api/docs/

http://thinkdiff.net/facebook/users-demographic-data-from-facebook/

And many-many more:

http://www.programmableweb.com/apis

Case study: Weather API:

http://www.worldweatheronline.com/free-weather-feed.aspx

Try the link below, but first receive a free developer key from http://www.worldweatheronline.com/register.aspx

http://api.worldweatheronline.com/free/v1/weather.ashx?key=xxxxxx&q=Toronto&num_of_days=3&format=json

App 1: display local weather

- Demo: link (might be blocked by browser security, as the page comes from google drive)

- From your local folder: demo/weather/myLoc.html

App 2. Wikipedia and word cloud

- Demo: link (might be blocked by browser security, as the page comes from google drive)

- In your local folder:

  demo/wiki_visualization/wordcloud.html