Web 2.0 and RIA

Alt:
https://docs.google.com/document/d/1AiBgsP0K1O1gluRmx6U25ZApAVbh4NenvbVbupWSA2M/edit?usp=sharing

List of Topics

REVERSE AJAX:
Need for Reverse AJAX - typical use cases.
Http Streaming vs Long Poll, timing diagrams.
COMET - using Hidden Frames - the need for heartbeat() function here.
- using XHR (the use of readyState == 3)
  - using server-sent events (SSE).
  - The advantages of using SSEs.
  - The problem with simultaneous requests (http 1.1 restriction)due to COMET calls.

AJAX - PERFORMANCE:
The "2 simultaneous requests only" limitation with http 1.1.
The reason for the limitation.
Impact on performance on client and server sides - possible workarounds
- Maintaining priority Queues,
- increasing connections at browser end
- using sub-domains.
- using separate servers with Access-control headers.
- packing requests into one.

Status code 200 does not mean all is ok. why??
Race conditions on the server.
Use of the "HEAD" call.
Use of Compression techniques. TTFB, TTLB. Minimising TTFB.
Caching: The use of Cache-control and Expires headers. Difference between the two.
GET vs POST requests to save performance.
Techniques to reduce size of markup/javascript being transferred.
Minifying code.
Using external Javascripts.

Misuse of the "src" attribute in <img> tags

WEB SERVICES:
Web Services vs Web Sites.
Properties of a Web Service - discoverable, self-describing etc.
Types of Web Services - RESTful and SOAP-based.
How SOAP-based services work. UDDI, SOAP and WSDL.
REST - meaning and architecture.
REST constraints.
RESTful APIs.
Standard guidelines for designing RESTful services.
JSON vs XML as data choice.
RESTful vs SOAP based services - Advantages and disadvantages.

SECURITY:
Restricting data access using HTTP Authentication.
The configuration directives in Apache.
The http status 401.
Types of Web attacks-
- Phishing
- Man-in-the-middle, session-hijacking.
- SQL Injection.
- Cross Site Scripting - XSS. Variations (Permanent, reflection)
- CSRF - Cross Site Request Forgery.
- Difference between XSS and CSRF.
- General countermeasures/precautions for protection.
- The "Heartbleed" vulnerability
- (SSL/https excluded for now)

SEMANTIC WEB:
Differences between Syntactic Web and Semantic Web.
Basic Working of the Semantic Web.

Reverse AJAX

Need for Reverse AJAX

The main idea behind Reverse AJAX is to allow servers to push data to clients whenever an event occurs. It bypasses the limitation of a traditional AJAX request which is stateless and must be initiated by the client.

Much more suitable for real time applications where clients must constantly fetch data from the server

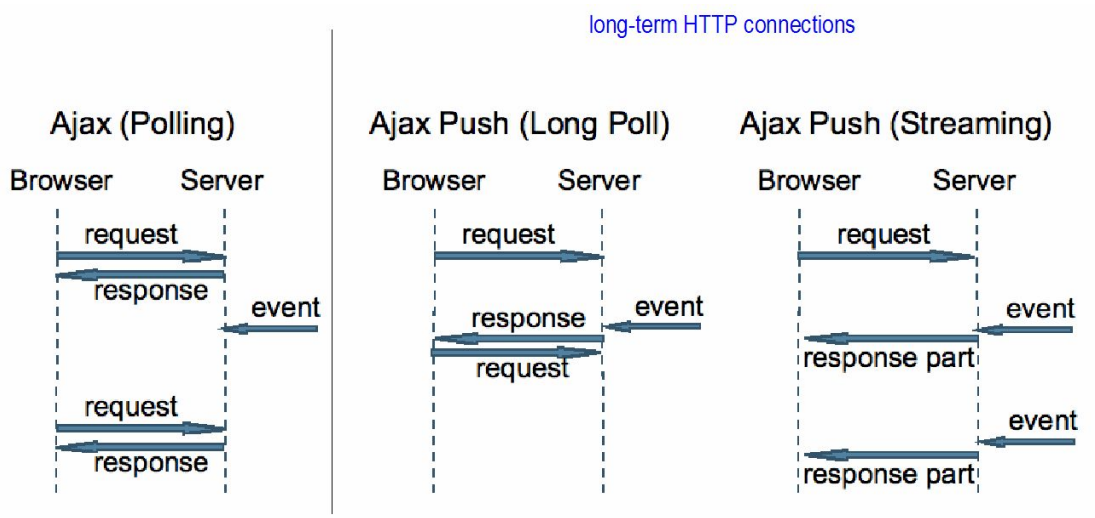
Reverse AJAX techniques

1. **Long polling**: polling involves periodically issuing a request to the server to ask for some data. If the interval b/w successive requests is low, more requests will be made which is wasteful.

2. **Piggybacking**: a modification to polling – when the client makes a request, the server can respond in two ways: send a normal response, or, send a mixed response.
    a. normal response is sent when the server has no event-related data to send
    b. when an event occurs, then the next request made by the client will see a response from the server containing the normal response as well as the event related response

    The advantage of piggybacking over long polling is it eliminates the need to make unnecessary requests, but the latency is much higher.

3. **Http streams / COMET**: COMET is a web application model where a request is sent to a server and then kept alive for a long time until a time-out or a server event occurs. With COMET, web servers can send data to client without the client having to explicitly request it.



COMET

- Allows a web server to push data to a browser with a long held HTTP request i.e., the browser doesn't explicitly request the server each time, through a single, previously opened connection

- Frameworks (more here)
    - Java - asynchronous servlets
    - DWR
    - ICEfaces
    - Pushlets
- Techniques
    - XHR
    - iFrames
        - Problem: error handling
        - Solution: heartbeat
    - SSE
        - Server-sent events

XHR

- We're doing long polling here. So, a request is opened by the client and a response is received whenever some event occurs on the server side.

- What the server does is keep sending outputs whenever some event occurs and the client receives it as a **partial output**.

```
if(this.readyState == 3 && this.status == 200)
```

- And finally when we reconnect:

```
if(this.readyState == 4 && this.status == 200)
{
        xhr.abort();
        monitor();     // create a new XHR
}
```

- Problem – in php.ini, max_execution_time = 30, so if no event occurs for 30 seconds, it gives a "Fatal error". Add this to deal with it.

```
if(this.responseText.indexOf("Fatal") == -1)
```

iFrames

- HTTP streaming, **not** long polling
- The server monitors the text file perpetually
- If the file has been modified, the data is pushed to the client by firing an update event

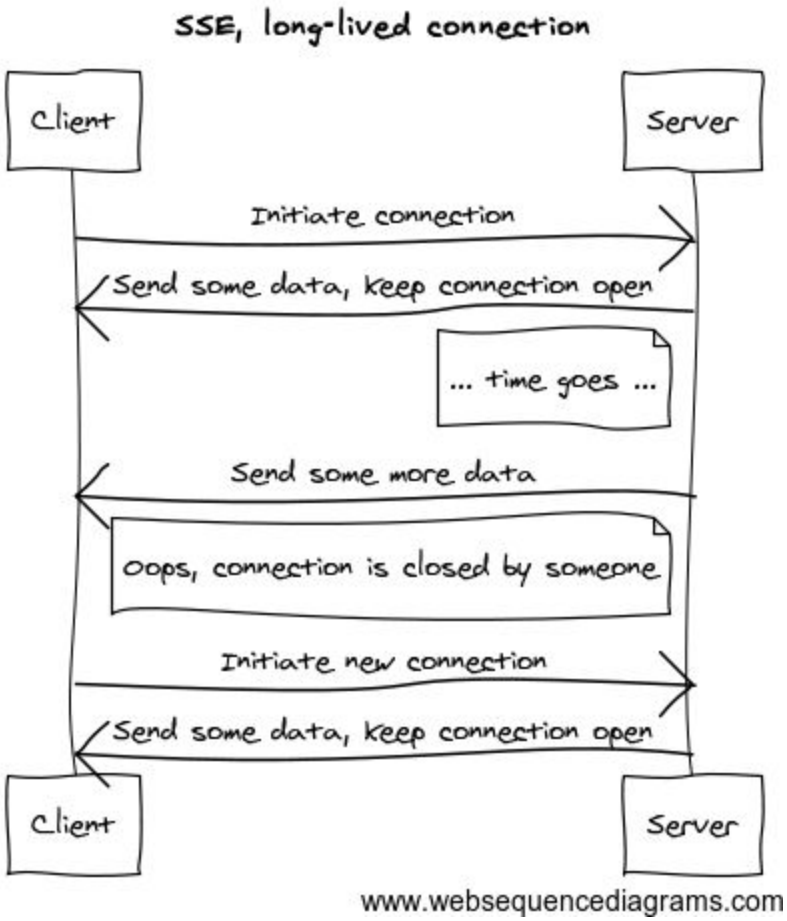
```
echo "<script>parent.obj.updateMain(\"$str\");</script>";
```

- If the file hasn't been modified for a long time, the server sends a heartbeat to the client telling the client that it's alive

```
echo "<script>parent.obj.heartbeat();</script>";
```

- What would happen if there was no heartbeat? Failure if the connection breaks on either side.

SSE



- Unidirectional – the server automatically sends data to the browser
- Pure HTTP
- No polling
- Automatic reconnection
- On the client side:

```
e = new EventSource('server.php');
e.addEventListener('myevent', handler);
```

- On the server side, behind the scene

```
echo "event:myevent\n"       // each event has data associated with it and can have multiple data lines
echo "data:hello\n"
echo "data:world\n\n"        // '\n\n' to indicate end of data
echo "retry:10000\n"         // retry after 10 seconds if connection closes
```

Coding - remember the following

```
date_default_timezone_set("Asia/Calcutta");
ob_start();
$time = filemtime("data.txt");
clearstatcache();            // while comparing modification times
date('H:i:s', $newtime);
ob_flush();
flush();
```

Chat

- Code review
  ```
  $msgarr = file("bchat.txt");
  $latestmsg = array_pop($msgarr);
  $.post("savetodb.php", {'msg': msg.value}, obj.updateWin);
  ```
- monitorchat.php - EventSource server
- savetodb.php - save chats to a file
- chat.js
  - monitor() - create EventSource() to monitor chat.txt
  - updateChat() - on a reply from ^
  - reconnect() - on an error from ^^
  - checkSend() - if “send on enter” is enabled and event.keyCode == 13
  - sendToServer() - send whatever has been typed to the server use jQuery to send a POST to savetodb.php
  - updateWin() - clear the textbox

Web Services

- Website - usually serves static pages
- Webapp - more intelligent than a website

Website and webapp are used by individual

Websites	Web service
Double role:	No concept of presentation

• provide information • present information	
Returns markup (HTML)	Returns XML or JSON or both
Used by humans	Used by other programs

- Web service
    - It’s a functionality/software built over the web and is used by a machine (server) for some processing.
    - Doesn’t return markup
    - Returns XML or JSON or both
    - Its interface is described in a machine processable format like WSDL
    - Properties
        - Self-contained and self-describring
        - Modular
        - Language independent
        - Open and standards based
        - Loosely coupled
- Why do we need web services?
    - We don’t wanna reinvent the wheel
    - Faster product development
    - Cross platform access possible

```
CLIENT ---------> SERVER ----------> WEB SERVICE
       <---------        <---------
```

ReST-ful API (Representational State Transfer)

- Makes only HTTP requests
- No markup (try to provide both JSON and XML, JSON if both not possible)
- Discoverable
- Set of URLs to access the service
- Self describing
- CRUD operations
    - CREATE - POST

    - READ - GET
    - UPDATE - PUT
    - DELETE - DELETE
- Components of the architectural model:
    - Resources (eg: image)
    - Representation (eg: jpeg/png)
    - Connectors
- Client-server based
- All operations must be stateless
- All responses must be cacheable
- Server architecture must be layered
    - Clients hide what servers are giving
- All components in the system must provide a uniform interface
- The communication between the client and server is independent of the communication between the server and web service

- On the client side, the representation of the resource is rendered
    - representation is the only handle the client has
    - Given permissions, the client can manipulate the resource through the representation
    - Eg: Resource -> image, Representation -> jpeg
- Connector
    - Software connector: Request, Response
    - Hardware connector: Cable

- How will the server know the URL of the web service?
    - Through API - API is a URL that's readable and makes a call of the type `https://helpline.com/tax/salary/10k`

- URN + URL = URI
    - http://localhost/usn.html#three
        - URI: http://localhost/usn.html#three
        - URL: http://localhost/usn.html
        - URN: localhost/usn.html#three
        - Resource is three

- http://localhost/update.php?usn=9
    - Not a good API cuz it gives the hacker an idea of what to process
    - http://localhost/update/get/usn/9 -> Better

- Mapping should be done between the above two

- More examples:

  GET a user details with ID=1.

  URI: "http://localhost/ws.php?method=Fetch&ID=1

  REST-API: http://localhost/MISC/get/user/1.json

  http://localhost/MISC/get/user/1.xml

  DELETE a user details with ID=3.

  URI: "http://localhost/ws.php?method=delete&ID=3

  REST-API: http://localhost/MISC/delete/user/3

- Disadvantages of REST:
  - Assumes point-to-point communication
  - Lacks standard for security policy

SOAP (Simple Object Access Protocol)

- XML based protocol
- Need not necessarily happen over HTTP
- Platform independent
- **UDDI** (Universal Description, Discovery and Integration):

  It's an XML based standard for describing, finding, and publishing web services
- **WSDL (**Web service Description Language)

  It's an is an XML-based language for describing web services, how to access them, what functionalities they're providing, etc

- Server can be written in any language (JAVA, C, C++, etc)
- Since it's possible that web server and client server are written in different languages, the web server serializes the object it needs to send and then sends it to the server.
- The server then deserializes it and uses it
- For serializing and deserializing, there are **stubs** that are attached to the server and web server.

- Naming service: Locating an object over the web using a name
- The client server can directly invoke the objects on the web server

- How it works:
    - Using UDDI, the client server gets the name of the object it requires
    - Using WSDL, it locates the object by its name and obtains a handle on the object
    - Using the handle, it invokes the object on the web server

---

Performance of AJAX

Potential issues with AJAX - Client side

1. Request is never sent
    - the browser drops requests

2. Request never returns
    - happens due to timeouts. One solution is to wait for a fixed time interval before aborting and resending the request. With XHR, this can be done using the timeout and ontimeout properties

3. Request returns, but takes too long
    - the server is overloaded with multiple requests

4. Response from the server but an error has occurred
    - permission issues
    - error during processing
    - overloaded
    - malformed data received

If multiple requests are made and the responses are out of order, then a lot of bandwidth is wasted if the ordering was important.

In cases 1 through 3, the reason could be due to the client sending an avalanche of requests in which case either

i) server chokes requests (meaning there's either a timeout or a huge latency)

ii) browser drops requests due to the 2 connection limitation imposed by HTTP 1.1.

The 2 connection limitation

According to HTTP 1.1; at any point of time there can only be a maximum of 2 simultaneous connections from a single client to a particular domain.

Reason for imposition of the limit

This is done so that clients do not overload servers with too many request (or else the server might classify it as a DDoS attack and block the client altogether).

Solutions to the 2-connection limitation

1. Use a priority queue on the client side so that requests are not dropped by the browser

2. Use sub-domains, eg; mail.google.com and drive.google.com

3. Split up resources across multiple servers and use Access-Control-* headers

4. Increase the max no. of connections from the browser's side; this is just a hack and not recommended (change param in about:config for Firefox)

5. Send requests in batches; this requires code to pack and unpack requests, not to mention the speed of the slowest request determines the speed of the entire batch

Potential issues with AJAX - Server side

1. **Racing**: If multiple AJAX requests are made which involve manipulating the superglobal $_SESSION, then this could lead to racing. Luckily, PHP has built in locking mechanisms

2. **Server availability; and use of the HEAD call**:

If a request is to be made with a lot of data, it is better to ensure the server is offline before making such an expensive request. This is accomplished by sending a simple HEAD call to the server. Server responds with headers, not data (a simple heartbeat).

3. **Client availability**: similar to point 2; this can also be solved in the same request.

Content Optimization

Compression Techniques

```
header("Content-Encoding:deflate");
header("Content-Encoding:gzip");
$output = gzdeflate($datastring, 9);
$output = gzencode($datastring, 9);
// 9 = max compression
```

Caching

- AJAX calls are not cached by the browser
- We need to use header("Expires: $expr");
- $expr format => date("D, d M Y H:i:s", time() + 10);
- Problem with Expires is the browser may keep the data cached even after the expiry date.
- Solution: header("Cache-control: max-age=10");

Miscellaneous

1. GET vs POST: In all browsers except Firefox, POST is a two step process. First, the headers are sent, followed by the data. GET, on the other hand, uses only one TCP packet. Since POST is more expensive, GET

should be used wherever possible.

2. When using images, so not set the src attribute unless required. Even setting src="" (empty attribute) wastes a call.

3. Code minification:

- remove white spaces, comments, etc
- replace long winded variables names with one/two letter names
- change colour names to hex-code
- minimize DOM access; store them in variables and use

Web Security

Possible risks:

- Phishing
- Spam
- DDoS
- DNS poisoning
- Packet sniffing (case study: heartbleed)
- Database breach (SQL injection)
- Expired certificates
- XSS (Cross site scripting)
- CSRF

HTTP Authorization

- In case, some files need to be protected on the server
- If some file needs authorization, server sends 401 as the status and the browser gives a popup to enter username and password
- This can be sent to the server with BASIC (base64) encoding or DIGEST encryption
- BASIC + HTTPS is normally used, DIGEST is complicated
- The username and password combination is cached and there is no popup until server sends another 401

```
string = btoa(user + ':' + pass);
xhr.setRequestHeader('Authorization', 'BASIC ' + string);
```

CSRF

Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a malicious website, email, blog, instant message, or program causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated.

Example:

A user is logged into Gmail and checks his mails. After she stays logged in for a while – that's a regular behavior – the user opens a new tab and navigates to another site. This site contains code that fires a regular HTTP request to the Gmail servers – an image tag is enough to do so. Since the user is still logged in, the request is processed with her privileges – may be changes some settings or deletes some mails, thanks to the fact that HTTP is stateless – that's it. The attack has been performed successfully and neither the user nor Gmail have even noticed.

SQL Injection

ref: studsearch.php

XSS

- Nice looking hacker's website
- Will have one link that's malicious which points to a popular website
- Prevent the default action onclick
- The link might be part of a form which can execute a malicious script on the user's browser
- Can use an image to send cookies over cross domain
- Solution:
    - Sanitise the inputs
    - Use HTTP-only cookies (setcookies("key", "value", time()+10, NULL, NULL, NULL, **TRUE**))
    - Can still be retrieved using TRACE method
    - Optimal option: HTTPS + HTTP-only cookies

Solutions

- Sanitize the inputs, don't accept something you're not expecting
- Give vague, generic errors
- Never use eval()
- HTTP-only cookies
- Don't run multiple queries in one stroke
- Use SSL + Basic authentication
- Use condoms - not 100% safe though

Codes

RESTful API

.htaccess

```
RewriteRule ^(get)/usn/([0-9]+)\.(json|xml) usn.php?usn=$2&rettype=$3
RewriteRule ^updt usn.php
```

usn_client.php

```
extract($_GET);
$data_arr = array("usn" => $usn, "grd" => $grd);

$data = http_build_query($data_arr);

$header = array("Content-type:application/x-www-form-urlencoded");

$req = array
(
	"http" => array
	(
		"method"=>"PUT",
		"header"=>$header,
		"content"=>$data,
	)
);


$context = stream_context_create($req);

$retval = file_get_contents("http://localhost/class/T2/mohit/10-05/priv/updt", false,
$context);
```

```
echo $retval;
```

usn.php

```
if ($_SERVER["REQUEST_METHOD"] == 'PUT')
{
        $data = file_get_contents("php://input");
        $data_arr = explode("&", $data);

        $usn = explode("=", $data_arr[0])[1];
        $grd = explode("=", $data_arr[1])[1];
        $str = $usn . ":" . $grd;

        $file = fopen("usn.txt", "w");
        fwrite($file, $str);

        $retarr = array();
        $retarr["status"] = 200;
        $retarr["message"] = "Successfully saved";
        $retarr["data"] = null;
        echo json_encode($retarr);
}
```