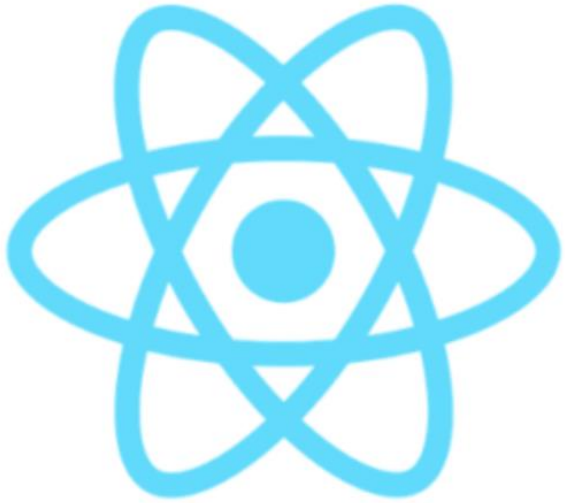
ReactJS Application Development

HTTP Client and Fetch APIs. REST Services

Where to Find The Code and Materials?

https://github.com/iproduct/fullstack-typescript-react



Asynchronous JavaScript & XML - AJAX

- **Ajax** – A New Approach to Web Applications, J. Garrett February, 2005 http://www.adaptivepath.com/publications/essays/archives/000385.php
- Presentation based on standards **HTML 5 / XHTML, CSS**
- Dynamic visualisation and interaction using **Document Object Model (DOM)**
- Exchange and manipulation of data using XML and XSLT or **JavaScript Object Notation (JSON)**
- Asynchronous data fetch using **XMLHttpRequest**
- And **JavaScript** who wraps everything above in one application

AJAX and Traditional Web Applications

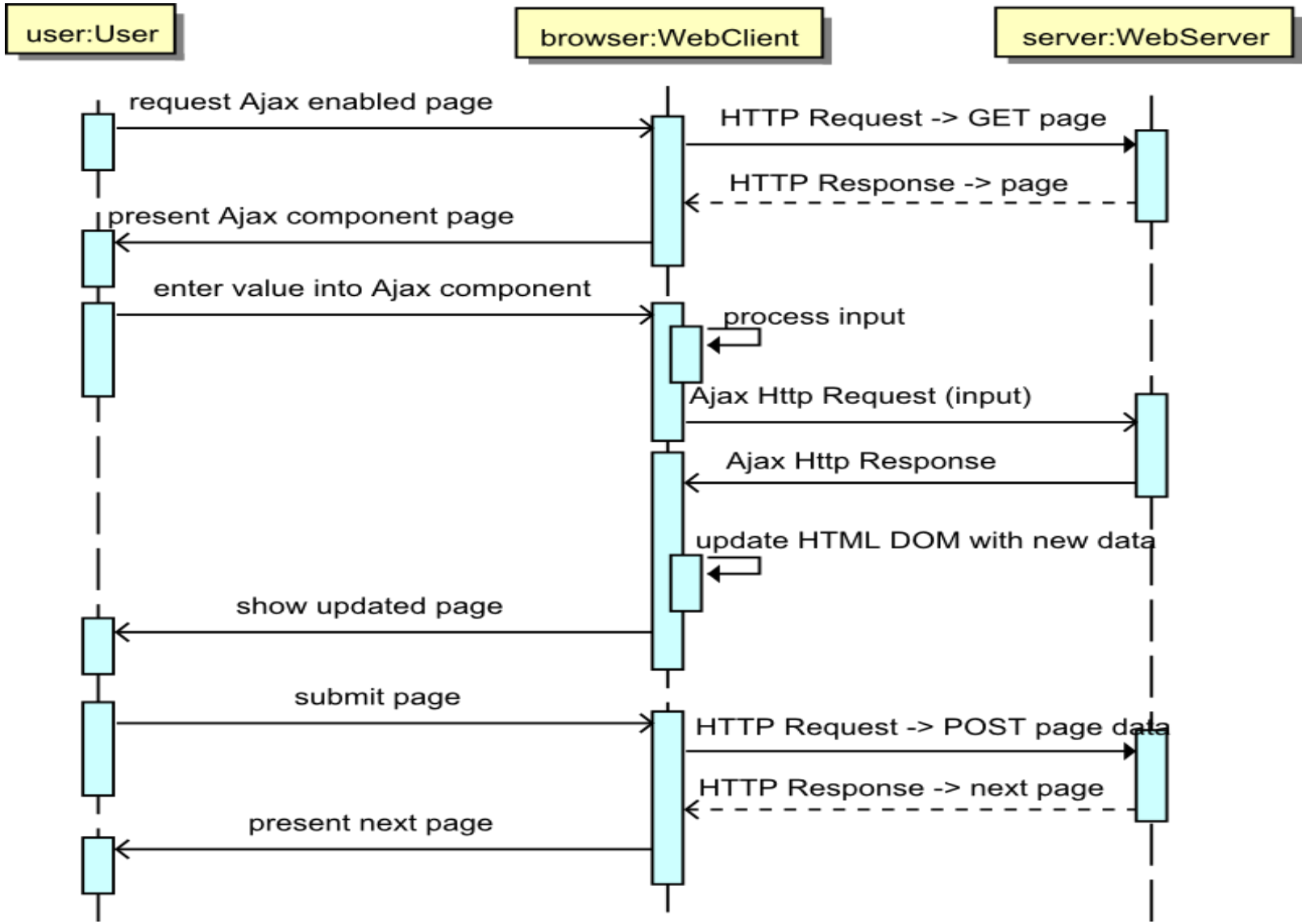
Main difference:

- **Ajax apps** are based on processing of **events** and **data**
- **Traditional web applications** are based on presenting **pages** and **hyperlink transitions** between them

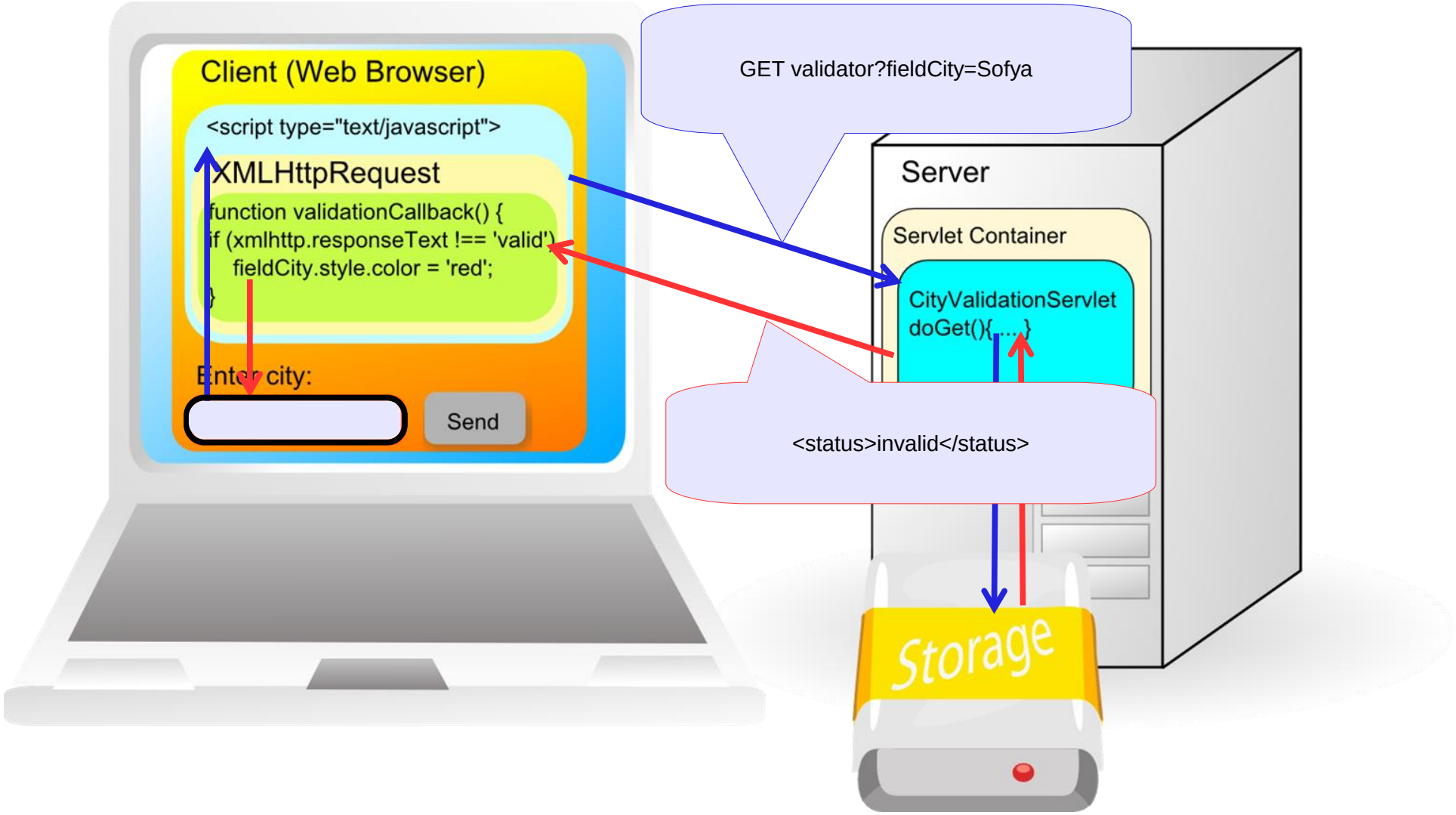
Problems connected with AJAX

- Sandboxing
- Scripting switched off
- Speed of client processing
- Time for script download
- Search engine indexing
- Accessibility
- More complex development
- More complex profiling – 2 cycles
- Cross Domain AJAX

AJAX Interactions Flowchart



AJAX Interactions



Basic Structure of Synchronous AJAX Request

```
var method = "GET";
var url = "resources/ajax_info.html";

if (window.XMLHttpRequest)   {// IE7+, Firefox, Safari, Chrome, Opera,
        xmlhttp=new XMLHttpRequest();
    }  else {// IE5, IE6
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
}
```

isAsynchronous = false – NOT Recommended

```
xmlhttp.open(method, url, false);
xmlhttp.send();
document.getElementById("results").innerHTML =  xmlhttp.responseText;
```

AJAX Request with XML Processing and Authentication

```
if (window.XMLHttpRequest)   {// IE7+, Firefox, Safari, Chrome, Opera,
        xmlhttp=new XMLHttpRequest();
    }  else {// IE5, IE6
        xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
}
xmlhttp.open("GET", "protected/products.xml", false, "trayan", "mypass");
xmlhttp.send();
if (xmlhttp.status == 200 &&
          xmlhttp.getResponseHeader("Content-Type") == "text/xml") {
        var xmlDoc = xmlhttp.responseXML;
        showBookCatalog(xmlDoc);  // Do something with xml document
    }
}
```

AJAX Request with XML Processing (2)

```
function showBookCatalog(xmlDoc){
        txt="<table><tr><th>Title</th><th>Artist</th></tr>";
    var x=xmlDoc.getElementsByTagName("TITLE");
    var y=xmlDoc.getElementsByTagName("AUTHOR");
    for (i=0;i<x.length;i++) {
        txt=txt +"<tr><td>"
            + x[i].firstChild.nodeValue
            + "</td><td>"+ y[i].firstChild.nodeValue
            + "</td></tr>";
    }
    txt += "</table>"
    document.getElementById("book_results").innerHTML=txt;
}
```

Basic Structure of Asynchronous AJAX Request

```
if (window.XMLHttpRequest)   {// IE7+, Firefox, Safari, Chrome, Opera,
      xmlhttp=new XMLHttpRequest();
}  else {// IE5, IE6
      xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.onreadystatechange = function(){
      if (xmlhttp.readyState==4 && xmlhttp.status==200){
         callback(xmlhttp);
      }
}
xmlhttp.open(method, url, true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xmlhttp.send(paramStr);
```

Callback function

isAsynchronous = true

XMLHttpRequest.readyState

Code	Meaning
1	After the **XMLHttpRequest.open()** has been called successfully
2	HTTP response headers have been successfully received
3	HTTP response content loading started
4	HTTP response content has been loaded succesfully

HTTP Request Headers

- In **HTTP 1.0** all request headers are optional
- In **HTTP 1.1** all request headers but **Host** are optional
- It is necessary to always check if given header is present

HTTP Requests Status Codes - RFC2616

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Connection
- Content-Length
- Cookie
- Host
- If-Modified-Since
- If-Unmodified-Since
- Referer
- User-Agent

HTTP Request Structure

GET */context/Servlet* **HTTP/1.1**

Host: *Client_Host_Name*

Header2: Header2_Data

…

HeaderN: HeaderN_Data

<empty line>

POST */context/Servlet* **HTTP/1.1**

Host: *Client_Host_Name*

Header2: Header2_Data

…

HeaderN: HeaderN_Data

<empty line>

POST_Data

HTTP Response Structure

HTTP/1.1 200 OK

Content-Type: application/json

Header2: Header2_Data

...

HeaderN: HeaderN_Data

<empty line>


```
[{ "id":1,
   "name":"Novelties in Java EE 7 ...",
   "description":"The presentation is ...",
   "created":"2014-05-10T12:37:59",
   "modified":"2014-05-10T13:50:02",
 },
 { "id":2,
   "name":"Mobile Apps with HTML5 ...",
   "description":"Building Mobile ...",
   "created":"2014-05-10T12:40:01",
   "modified":"2014-05-10T12:40:01",
 }]
```

Response Status Codes

- 100 Continue
- 101 Switching Protocols
- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content
- 205 Reset Content
- 301 Moved Permanently
- 302 Found
- 303 See Other
- 304 Not Modified
- 307 Temporary Redirect
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found

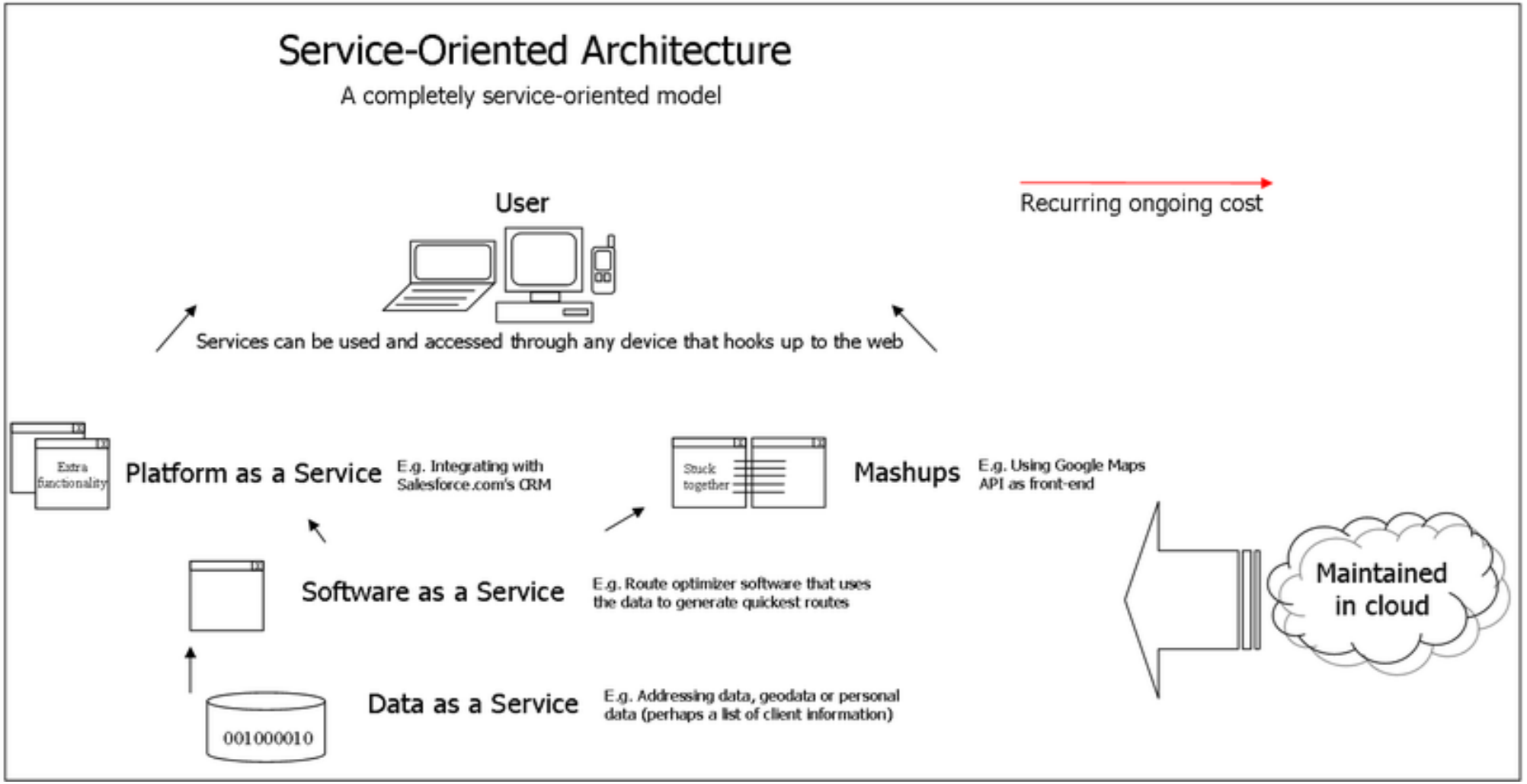
Response Status Codes

- 405 Method Not Allowed
- 415 Unsupported Media Type
- 417 Expectation Failed
- 500 Internal Server Error
- 501 Not Implemented
- 503 Service Unavailable
- 505 HTTP Version Not Supported

HTTP Response Headers

- Allow
- Cache-Control
- Pragma
- Connection
- Content-Disposition
- Content-Encoding
- Content-Language
- Content-Length
- Content-Type
- Expires
- Last-Modified
- Location
- Refresh
- Retry-After
- Set-Cookie
- WWW-Authenticate

Service Oriented Architecture (SOA)



Source: http://en.wikipedia.org/wiki/File:SOA_Detailed_Diagram.png,
Author: JamesLWilliams2010, License: Creative Commons Attribution 3.0 Unported

REST Service Architecture

- According to **Roy Fielding** [Architectural Styles and the Design of Network-based Software Architectures, 2000]:
  - Performance
  - Scalability
  - Reliability
  - Simplicity
  - Extensibility
  - Dynamic evolvability
  - Cusomizability
  - Configurability
  - Visibility
- All of them should be present in a desired Web Architecture and REST architectural style tries to preserve them by consistently applying several **architectural constraints**

REST Architectural Constraints

According to **Roy Fielding** [Architectural Styles and the Design of Network-based Software Architectures, 2000]:

- Client-Server
- Stateless
- Uniform Interface:
  - **Identification** of resources
  - Manipulation of resources through **representations**
  - **Self-descriptive** messages
  - **Hypermedia as the engine of application state (HATEOAS)**
- Layered System
- Code on Demand (optional)

Representational State Transfer (REST) [1]

- **REpresentational State Transfer (REST)** is an architecture for accessing distributed hypermedia web-services
- The resources are identified by URIs and are accessed and manipulated using an HHTP interface base methods (**GET, POST, PUT, DELETE, OPTIONS, HEAD**, **PATCH**)
- Information is exchanged using **representations** of these resources
- Lightweight alternative to SOAP+WSDL -> HTTP + Any representation format (e.g. **JavaScript Object Notation – JSON**)

Representational State Transfer (REST) [2]

- Identification of resources – **URIs** – URL, URN, IRI
- Representation of resources – e.g. HTML, XML, JSON, etc.
- Manipulation of resources through these representations
- Self-descriptive messages - Internet media type (**MIME type**) provides enough information to describe how to process the message. Responses also explicitly indicate their cacheability.
- Hypermedia as the engine of application state (aka **HATEOAS**)
- Application contracts are expressed as media types and [semantic] link realtions (**rel** attribute - RFC5988, "Web Linking")

Source: https://en.wikipedia.org/wiki/Representational_state_transfer

Simple Example: URLs + HTTP Methods

Uniform Resource Locator (URL)	GET	PUT	POST	DELETE
Collection, such as http://api.example.com/comments/	**List the URIs and perhaps other details of the collection's members.**	**Replace the entire collection with another collection.**	**Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.**	**Delete the entire collection.**
Element, such as http://api.example.com/comments/11	**Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.**	**Replace the addressed member of the collection, or if it does not exist, create it.**	Not generally used. Treat the addressed member as a collection in its own right and create a new entry in it.	**Delete the addressed member of the collection.**

Source: https://en.wikipedia.org/wiki/Representational_state_transfer

Advantages of REST

- Scalability of component interactions – through layering the client server-communication and enabling load-balancing, shared caching, security policy enforcement;
- Generality of interfaces – allowing simplicity, reliability, security and improved visibility by intermediaries, easy configuration, robustness, and greater efficiency by fully utilizing the capabilities of HTTP protocol;
- Independent development and evolution of components, dynamic evolvability of services, without breaking existing clients.
- Fault tolerat, Recoverable, Secure, Loosely coupled

Richardson's Maturity Model of Web Services

According to **Leonard Richardson** [Talk at QCon, 2008 - http://www.crummy.com/writing/speaking/2008-QCon/act3.html]:

- **Level 0 – POX**: Single URI (XML-RPC, SOAP)
- **Level 1 – Resources**: Many URIs, Single Verb (URI Tunneling)
- **Level 2 – HTTP Verbs**: Many URIs, Many Verbs (CRUD – e.g Amazon S3)
- **Level 3 – Hypermedia**: Links Control the Application State
  = HATEOAS (Hypertext As The Engine Of Application State)
  === **truely** RESTful Services

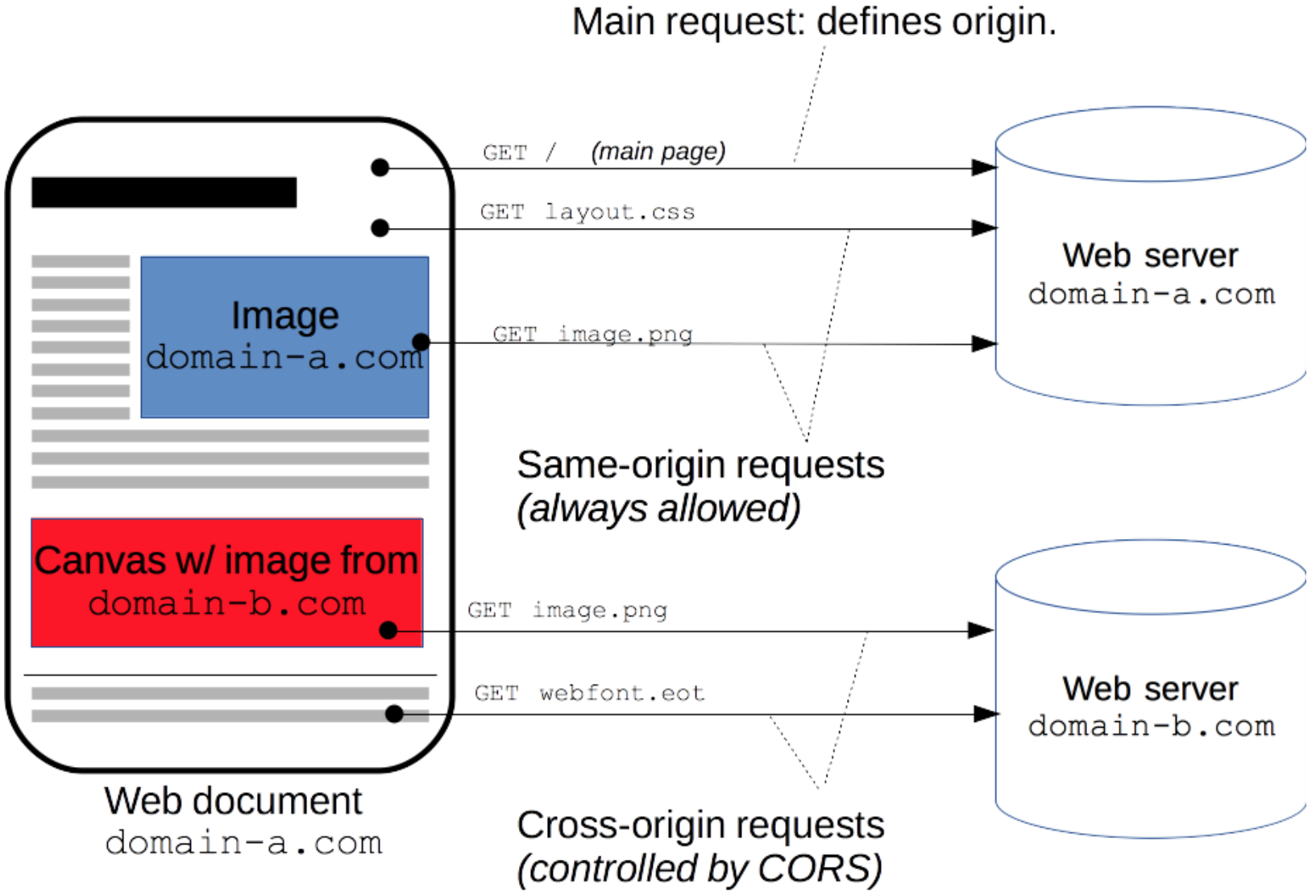
Hypermedia As The Engine Of Application State (HATEOAS) – New Link Header (RFC 5988) Example

```
Content-Length →1656

Content-Type →application/json

Link →<http://localhost:8080/polling/resources/polls/629>; rel="prev";
  type="application/json"; title="Previous poll",
  <http://localhost:8080/polling/resources/polls/632>; rel="next";
  type="application/json"; title="Next poll",
  <http://localhost:8080/polling/resources/polls>; rel="collection";
  type="application/json"; title="Polls collection",
  <http://localhost:8080/polling/resources/polls>; rel="collection up";
  type="application/json"; title="Self link",
  <http://localhost:8080/polling/resources/polls/630>; rel="self"
```

Web Application Description Language (WADL)

- XML-based file format providing machine-readable description of HTTP-based web application resources – typically RESTful web services
- WADL is a W3C Member Submission
  - Multiple resources
  - Inter-connections between resources
  - HTTP methods that can be applied accessing each resource
  - Expected inputs, outputs and their data-type formats
  - XML Schema data-type formats for representing the RESTful resources
- But WADL resource description is **static**

Cross-Origin Resource Sharing (CORS)



Cross-Origin Resource Sharing (CORS)

- Allows serving requests to domains that are different from the domain of the script is loaded from.
- The server decides which requests to serve, based on the **Origin** of the script issuing the request, the method (**GET, POST**, etc.), **credentials/cookies**, and the **custom headers** to be sent.
- When the method is different from **simple GET, HEAD** or **POST**, or the **Content-Type** is different from **application/x-www-form-urlencoded**, **multipart/form-data**, or **text/plain**, a **preflight OPTIONS request** is mandated by the specification.
- In response to **preflight OPTIONS request**, the server should return which **origins**, **methods**, **headers**, **credentials/cookies** are allowed in cross-domain requests to that server, and for **how long**.

New HTTP CORS Headers – Simple Request

- HTTP GET request:

GET /crossDomainResource/ HTTP/1.1

Referer: http://sample.com/crossDomainMashup/

Origin: http://sample.com

- HTTP GET response:

Access-Control-Allow-Origin: http://sample.com

Content-Type: application/xml

CORS - Simple Request:

New HTTP CORS Headers – PUT / DELETE/ Custom Content/Headers

- HTTP OPTIONS preflight request:

OPTIONS /crossDomainPOSTResource/ HTTP/1.1

Origin: http://sample.com

Access-Control-Request-Method: POST

Access-Control-Request-Headers: MYHEADER, Content-Type

- HTTP response:

HTTP/1.1 200 OK

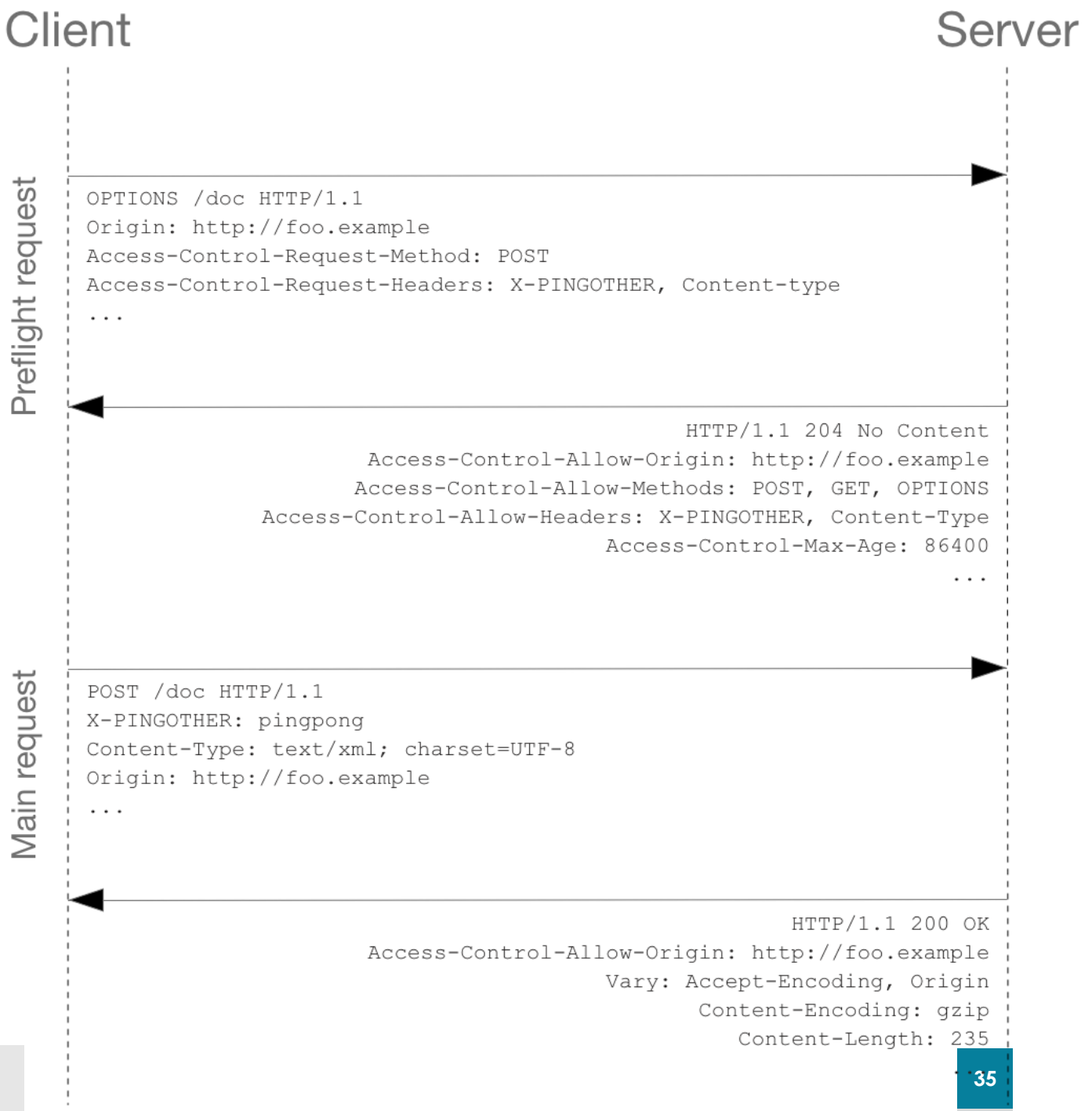
Access-Control-Allow-Origin: http://sample.com

Access-Control-Allow-Methods: POST, GET, OPTIONS

Access-Control-Allow-Headers: MYHEADER, Content-Type

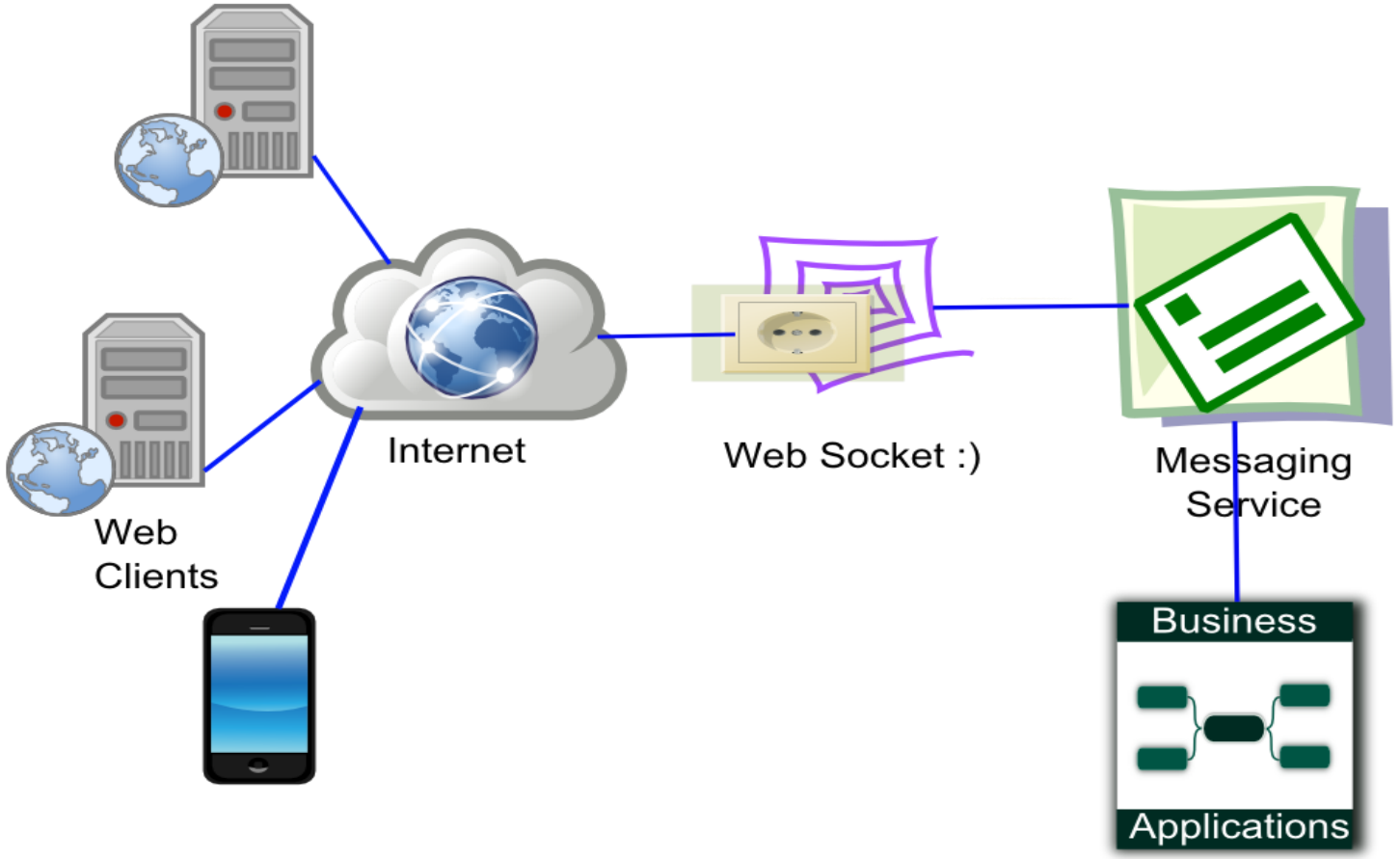
Access-Control-Max-Age: 864000

CORS – POST / PUT / DELETE/ Custom Content/Headers:



Web Socket Based Communication Architecture

- Proxies: HTTP CONNECT – Tunnelling
- HTTP/S can be used WebSockets over SSL (wss:// scheme)



WebSocket Main Applicatin Areas

- Massively multiplayer online role-playing game (MMORPG)
- Online trading – large scale auctions, stock tickers
- Interactive synchronous communication – chat, audio- & video-conferencing
- Collaborative authoring, groupware & social applications - including modelling and art
- Dynamic data monitoring and control – e.g. management dashboards presenting SLA, KPI and BI data in real time
- Remote observation and control of devices and services – e.g. remote monitoring of home security, energy consumption, data center services and devices performance visualizations

Other Alternatives for Bidirectional Communication over HTTP

- Ajax polling
- Long polling - Comet, HTTP server push, Reverse Ajax
- HTTP Streaming - Comet, chunked HTTP responses
- Bayeux protocol by Dojo Foundation – channels, publish/subscribe model
- Extensible Messaging and Presence Protocol (XMPP) over Bidirectional-streams Over Synchronous HTTP (BOSH)
- Drawbacks – buffering the streamed response, HTTP request and response headers on each message, two connections – coordination overhead and increased complexity that does not scale. HTTP wasn't designed for real-time, full-duplex communication

Sample HTTP Request Headrs – Much Overhead

GET JSON 304 Not Modified en.wikipedia.org 28.3 KB 91.198.174.192:80 90ms

Headers Response HTML Cache Cookies

Response Headers view source

Accept-Ranges	bytes
Age	33614
Cache-Control	private, s-maxage=0, max-age=0, must-revalidate
Connection	keep-alive
Content-Encoding	gzip
Content-Language	en
Content-Type	text/html; charset=UTF-8
Date	Fri, 13 Jun 2014 08:14:24 GMT
Last-Modified	Thu, 12 Jun 2014 22:54:07 GMT
Server	Apache
Vary	Accept-Encoding,Cookie
Via	1.1 varnish, 1.1 varnish, 1.1 varnish
X-Cache	cp1053 hit (10), amssq52 hit (95), amssq56 frontend miss (0)
X-Content-Type-Options	nosniff
X-Varnish	2755970074 2722575545, 1855856392 1851900024, 1030168116
x-ua-compatible	IE=Edge

Request Headers view source

Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding	gzip, deflate
Accept-Language	en,bg;q=0.7,en-us;q=0.3
Cache-Control	max-age=0
Connection	keep-alive
Cookie	centralnotice_bannercount_fr12=3; centralnotice_bannercount_fr12-wait=3%7C1401359244420%7C0; enwiki-gettingStartedUserId =1CkDb9sdZ2Cjzf4n1mGNHWuTY6WGxVvE; GeoIP=BG:Sofia:42.6833:23.3167:v4; uls-previous-languages=%5B%22en %22%5D; mediaWiki.user.sessionId=nQViPsuMNGYnKGD3BcfCMtUP7i28jlm3; mw_hidetoc=1; centralnotice_bucket =1-4.2
Host	en.wikipedia.org
If-Modified-Since	Thu, 12 Jun 2014 22:54:07 GMT
Referer	http://www.google.bg/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0CCcQFjAB&url=http%3A%2F%2Fen.wikipedia .org%2Fwiki%2FJSON&ei=4ayaU6rmK8fB0QWD4YGoAg&usg=AFQjCNHfk8CeJn25-S_gvF4dnY6ZaKxg4g&sig2=z2kN3sEMPh_SeBz8cKK-gw &bvm=bv.68911936,d.d2k
User-Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:29.0) Gecko/20100101 Firefox/29.0

Response Headers From Cache

Accept-Ranges bytes

IETF WebSocket protocol (RFC 6455) (1)

- Official IETF standard - **RFC 6455**
- TCP-based **full-duplex** protocol
- Starts as standard HTTP /HTTPS connection to web server port **80/443** (**handshake phase**) – easy firewall and proxy traversal without the overhead connected with polling
- Uses HTTP protocol upgrade mechanism (Upgrade: websocket + Connection: Upgrade) – communication is immediately upgraded to more efficient WebSocket protocol (**data transfer phase**)
- Allows **bidirectional streaming** of data (partial messages)

IETF WebSocket protocol (RFC 6455) (2)

- Designed with security and extensibility in mind: Origin validation, sub-protocols & extensions negotiation (through standardized HTTP headers exchanged in handshake phase)
- **WebSocket API in Web IDL** is being standardized by **W3C**
- Supported by latest versions of all major web browsers –

Implementation status

Protocol	Draft date	Internet Explorer	Firefox[17] (PC)	Firefox (Android)	Chrome (PC, Mobile)	Safari (Mac, iOS)	Opera (PC, Mobile)	Android Browser
hixie-75	February 4, 2010				4	5.0.0		
hixie-76 hybi-00	May 6, 2010 May 23, 2010		4.0 (disabled)		6	5.0.1	11.00 (disabled)	
7 hybi-07	April 22, 2011		6[18]1					
8 hybi-10	July 11, 2011		7[19]1	7	14[20]			
13 RFC 6455	December, 2011	10[21]	11	11	16[22]	6	12.10[23]	4.4

[1] Gecko-based browsers versions 6–10 implement the WebSocket object as "MozWebSocket",[24] requiring extra code to integrate with existing WebSocket-enabled code.

Source: Wikipedia, [http://en.wikipedia.org/wiki/WebSockets]

WebSocket Request Example

[Request URL: **ws://localhost:8080/ipt-present/ws**]:

GET /ipt-present/ws HTTP/1.1

Host: localhost:8080

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: 3RhAwlJCs7wbj3xUdeDTXA==

Sec-WebSocket-Protocol: epresentation, ipt_present

Sec-WebSocket-Version: 13

Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits, x-webkit-deflate-frame

Origin: http://localhost:8080

WebSocket Response Example

[Request URL: **ws://localhost:8080/ipt-present/ws**]:

HTTP/1.1 **101 Switching Protocols**

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Accept: QIMZj0lbIv1TM+JMx/JsoSKwYb8=

Sec-WebSocket-Protocol: epresentation

Server: GlassFish Server Open Source Edition 4.0

X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition 4.0 Java/Oracle Corporation/1.7)

W3C JavaScript WebSocket API [Web IDL]

```
WebSocket WebSocket(
  in DOMString url,
  in optional DOMString[] protocols
);
```

WebSocket JS API Example (1)

- Example 1:

```
connection = new WebSocket('ws://h2j.org/echo', ['soap', 'xmpp']);
```

- Example 2:

```
const rootWsUri = "ws://" + (document.location.hostname.length >0 ?
  document.location.hostname : "localhost") + ":" +
    (document.location.port.length > 0 ?
        document.location.port : "8080") + "/ipt-present/ws";

const websocket = new WebSocket( rootWsUri );
```

WebSocket JS API Example (2)

```
websocket.onopen = function (event) {
    onOpen(event);
};
websocket.onmessage = function (event) {
    onMessage(event)
};
websocket.onerror = function (event) {
    onError(event)
};
```

WebSocket JS API Example (3)

```
function onMessage(evt) {
        var jso = JSON.parse(evt.data);
        switch(jso.type){
          case "login-resp":
            conversationId = jso.cid;
            $(".logged-name").html(" - " + userName);
            $("#button-login").hide();
            $("#button-logout").show();
            showToster(jso.data.message, "info");
            break;
          ( - continues in next slide - )
```

WebSocket JS API Example (4)

```
        case "logout-resp":
          conversationId = ""; userName = "";
          $(".logged-name").html("");
          $("#button-logout").hide();
          $("#button-login").show();
          showToster(jso.data.message, "info");
          break;
      case "online-resp":
      case "offline-resp":
          showToster(jso.data.message, "info");
          break;  ( - continues in next slide - )
```

WebSocket JS API Example (5)

```
            case "error-resp":
                showToster(jso.data.message, "error");
                break;
        }
        console.log(jso.data);
}
```

Fetch API

[https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API , https://blog.logrocket.com/axios-or-fetch-api/]

- The **Fetch API** provides an interface for fetching resources like XMLHttpRequest, but more powerful and flexible feature set.
- Promise<Response> WorkerOrGlobalScope.fetch(input[, init])
  - **input** - resource that you wish to fetch – url string or Request
  - **init** - custom settings that you want to apply to the request:
  - method: (e.g., GET, POST),
  - headers, body (Blob, BufferSource, FormData, URLSearchParams, or USVString),
  - mode: (cors, no-cors, or same-origin),
  - credentials: (omit, same-origin, or include. to automatically send cookies this option must be provided),
  - cache: (default, no-store, reload, no-cache, force-cache, or only-if-cached),
  - redirect (follow, error or manual),
  - referrer (default is client),
  - referrerPolicy:(no-referrer, no-referrer-when-downgrade, origin, origin-when-cross-origin, unsafe-url),
  - integrity (subresource integrity value of request)

References

- R. Fielding, Architectural Styles and the Design of Networkbased Software Architectures, PhD Thesis, University of California, Irvine, 2000
- Fielding's blog discussing REST – http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
- Representational state transfer (REST) in Wikipedia – http://en.wikipedia.org/wiki/Representational_state_transfer
- Hypermedia as the Engine of Application State (HATEOAS) in Wikipedia – http://en.wikipedia.org/wiki/HATEOAS
- JavaScript Object Notation (JSON) – http://www.json.org/

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

http://robolearn.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD