

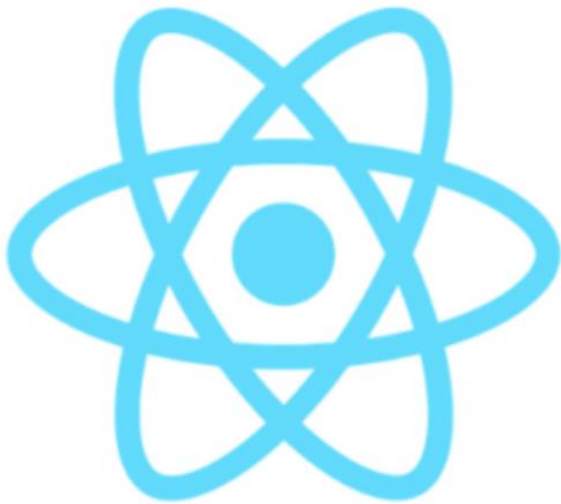


Full-stack Application Development

HTTP Client and Fetch APIs. REST Services

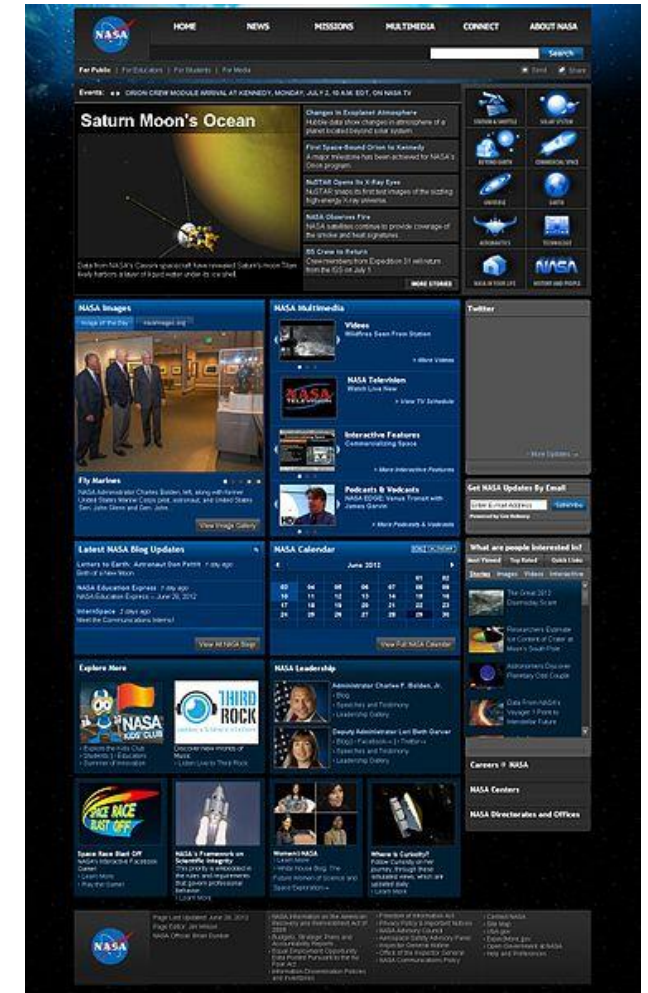
Where to Find The Code and Materials?

<https://github.com/iproduct/react-typescript-academy-2022>

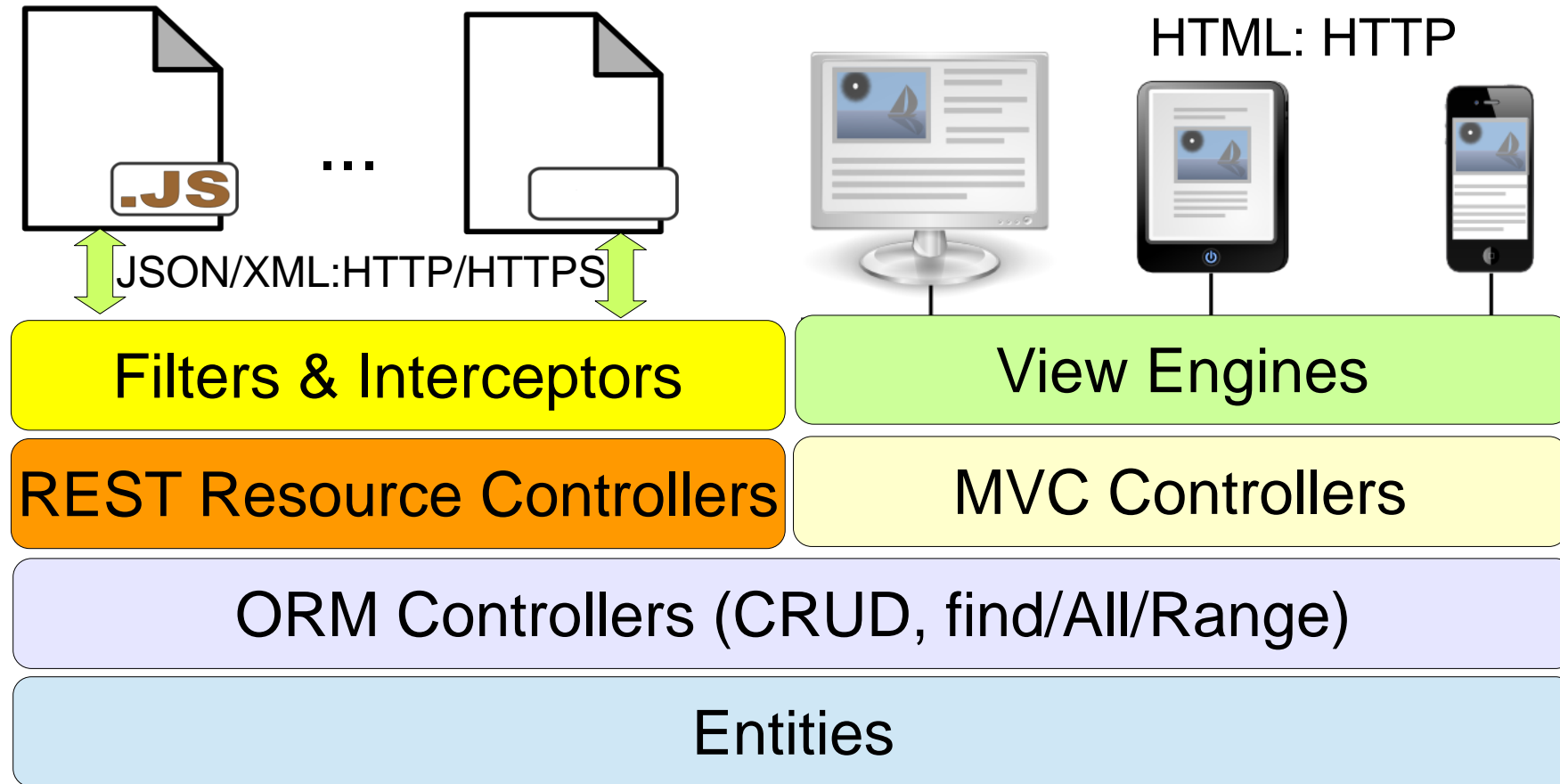


Types of Web Applications

- **Web Sites** – presenting interactive UI
 - **Static Web sites** – show the same information for all visitors – can include hypertext, images, videos, navigation menus, etc.
 - **Dynamic Web sites** – change and tune the content according to the specific visitor
 - **server-side** – use server technologies for dynamic web content (page) generation (data comes from DB)
 - **client-side** – use JavaScript and asynchronous data updates
- **Web Services** – managing (CRUD) data resources
 - **Classical** – SOAP + WSDL
 - **RESTful** – distributed hypermedia



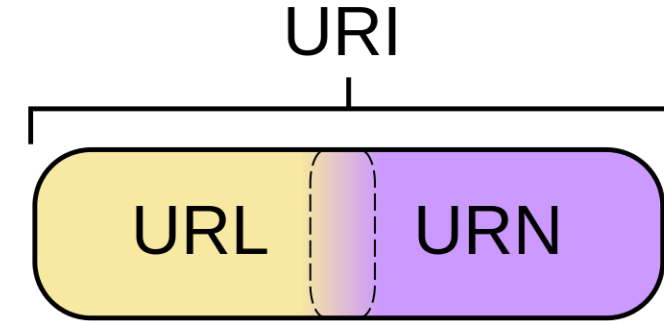
N-Tier Web Architectures



URLs, IP Addresses, and Ports

- Uniform Resource Identifier - URI:

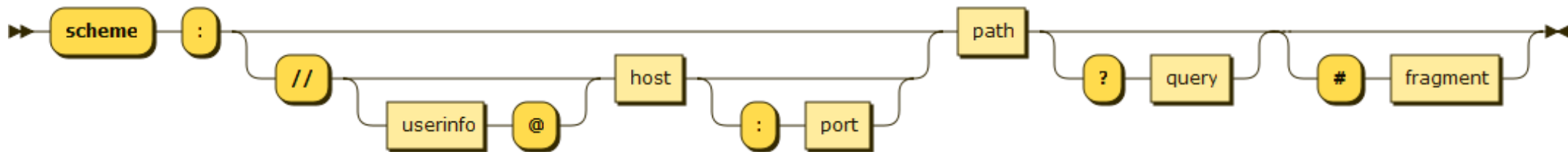
- Uniform Resource Locator – URL
- Uniform Resource Name – URN



- Format:

`scheme://domain:port/path?query_string#fragment_id`

- Schemas: `http://`, `https://`, `file://`, `ftp://`. `news://`, `mailto:`, `telnet://`

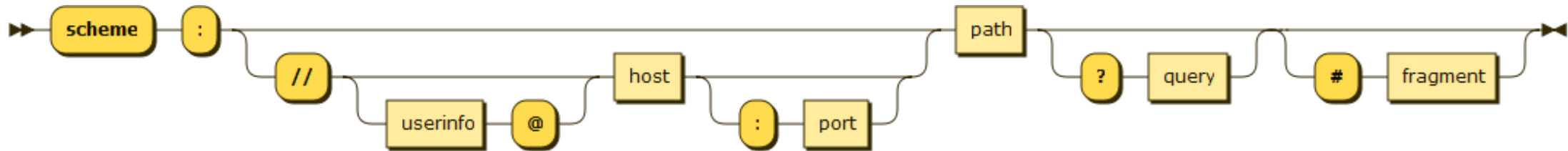


- Example:

`http://en.wikipedia.org/wiki/Uniform_resource_locator#History`

Query Parameters and Fragment Ids

- URLs pointing to **dynamic resources** often include:
- **/ {path_param}** – e.g.: `/users/12/posts/3`
- **?query_param=value** (query string) – e.g.:
`?role=student&mode=edit`
- **#fragment_identifier** – defines the fragment (part) of the resource we want to access, used by asynchronous javascript page loading (AJAX) applications to encode the local page state – e.g. `#view=fitb&nameddest=Chapter3`



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

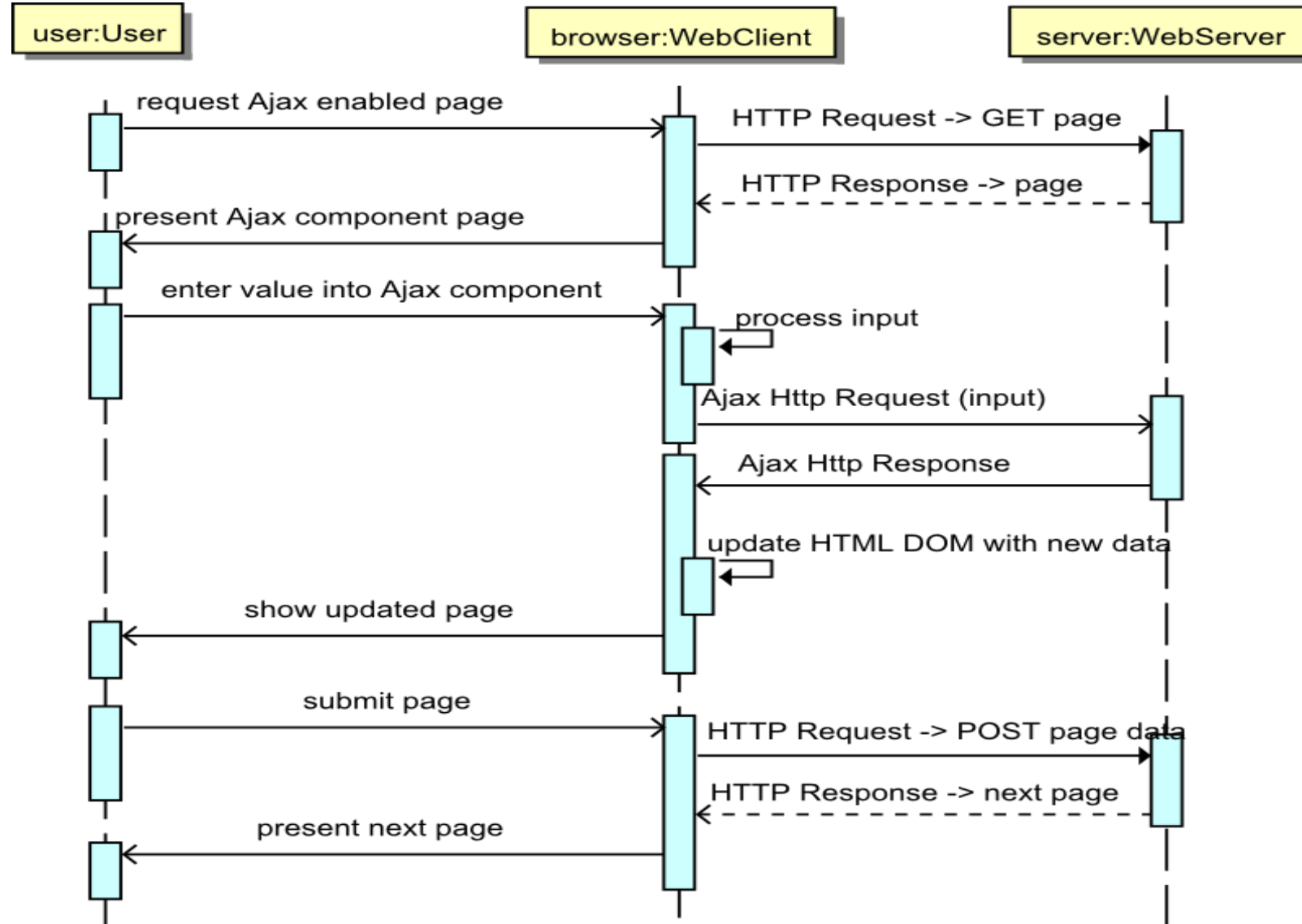
Main difference:

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

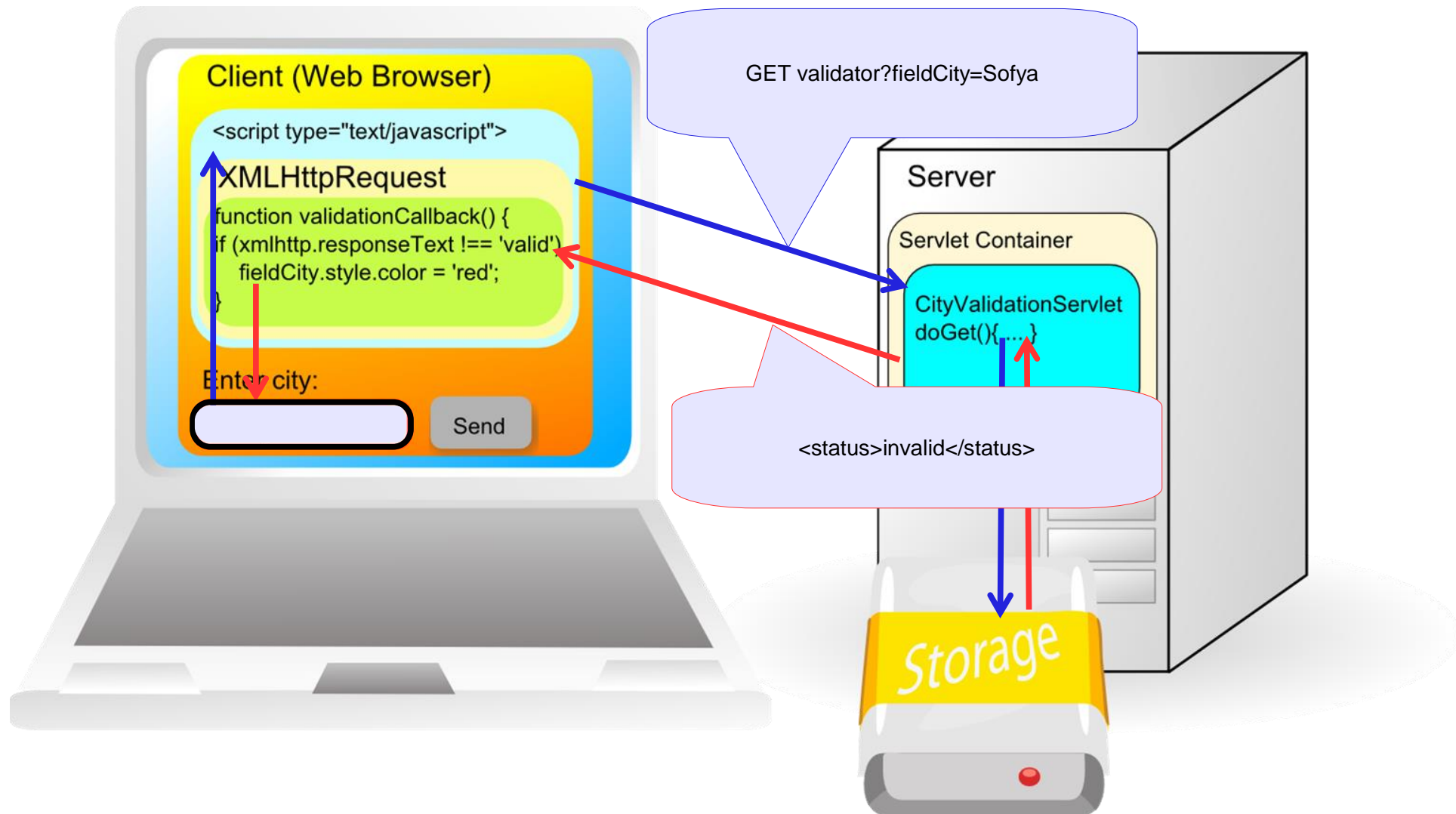
Problems connected with AJAX

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

isAsynchronous = false – NOT Recommended



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

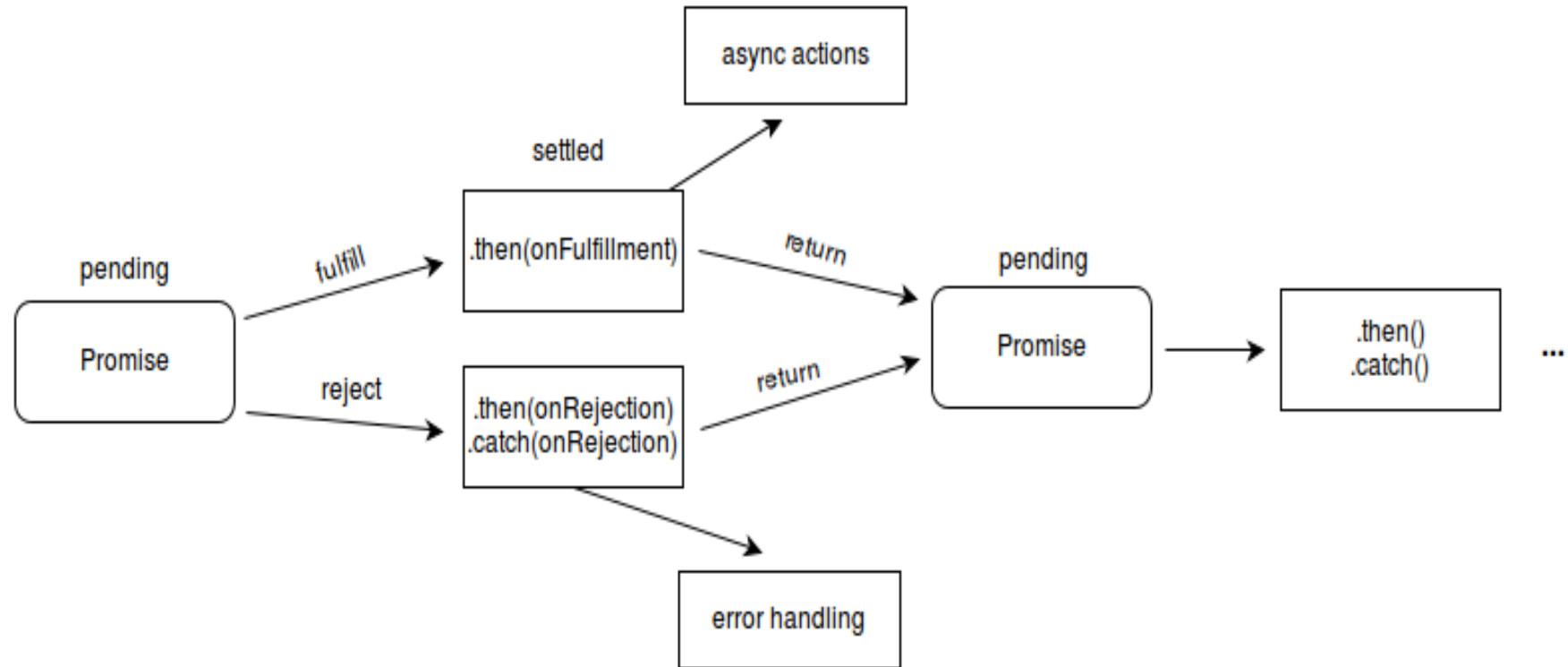
Code	Meaning
1	After the <code>XMLHttpRequest.open()</code> has been called successfully
2	HTTP response headers have been successfully received
3	HTTP response content loading started
4	HTTP response content has been loaded successfully

ES6 Promises [<http://es6-features.org/>]

```
function msgAfterTimeout (msg, who, timeout) {  
  return new Promise((resolve, reject) => {  
    setTimeout(() => resolve(`${msg} Hello ${who}!`), timeout)  
  })  
}
```

```
msgAfterTimeout("", "Foo", 1000).then((msg) => {  
  console.log(`done after 1000ms:${msg}`);  
  return msgAfterTimeout(msg, "Bar", 2000);  
}).then((msg) => {  
  console.log(`done after 3000ms:${msg}`)  
})
```

ES6 Promises



Combining ES6 Promises

```
function fetchAsync (url, timeout, onData, onError) { ... }  
fetchPromised = (url, timeout) => {  
  return new Promise((resolve, reject) => {  
    fetchAsync(url, timeout, resolve, reject)  
  })  
}
```

```
Promise.all([  
  fetchPromised("http://backend/foo.txt", 500),  
  fetchPromised("http://backend/bar.txt", 500)  
]).then( (data) => {  
  let [ foo, bar ] = data  
  console.log(`success: foo=${foo} bar=${bar}`)  
}).catch( (err) => {  
  console.log(`error: ${err}`)  
})
```

Combining ES6 Promises

```
function fetchAsync (url, timeout, onData, onError) { ... }  
fetchPromised = (url, timeout) => {  
  return new Promise((resolve, reject) => {  
    fetchAsync(url, timeout, resolve, reject)  
  })  
}
```

```
Promise.all([  
  fetchPromised("http://backend/foo.txt", 500),  
  fetchPromised("http://backend/bar.txt", 500)  
]).then( (data) => {  
  let [ foo, bar ] = data  
  console.log(`success: foo=${foo} bar=${bar}`)  
}, (err) => {  
  console.log(`error: ${err}`)  
})
```

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)

Fetch Example using Async – Await – Try – Catch

```
async function init() {  
  try {  
    const userResult = await fetch("user.json");  
    const user = await userResult.json();  
    const gitResp = await fetch(`http://api.github.com/users/${user.name}`);  
    const githubUser = await gitResp.json();  
    const img = document.createElement("img");  
    img.src = githubUser.avatar_url;  
    document.body.appendChild(img);  
    await new Promise((resolve, reject) => setTimeout(resolve, 6000));  
    img.remove();  
    console.log("Demo finished.");  
  } catch (err) { console.log(err); }  
}
```

Homework 1

Building a Google Books API search simple JavaScript application

<https://moodle.modis-ops.net/mod/assign/view.php?id=35>



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

Sample HTTP Request Headrs

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=1CkDb9sd22Cjzf4n1mGNHWuTY6WGxVvE; 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&usq=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					

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 & JSON

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

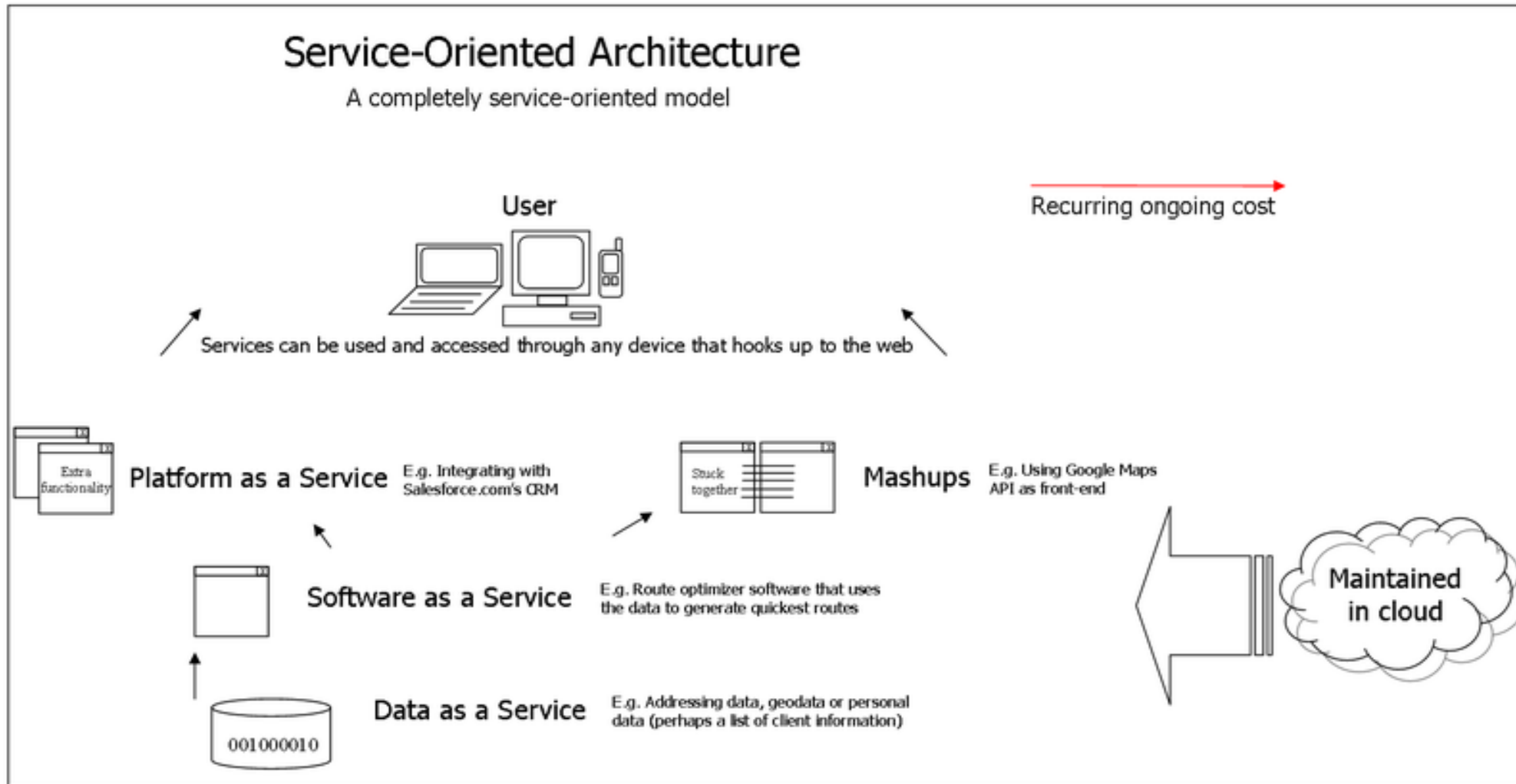
Response Status Codes

- 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)



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
 - Customizability
 - 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 URLs and are accessed and manipulated using an HTTP 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 relations (**rel** attribute - RFC5988, "Web Linking")

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.

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 tolerant, 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

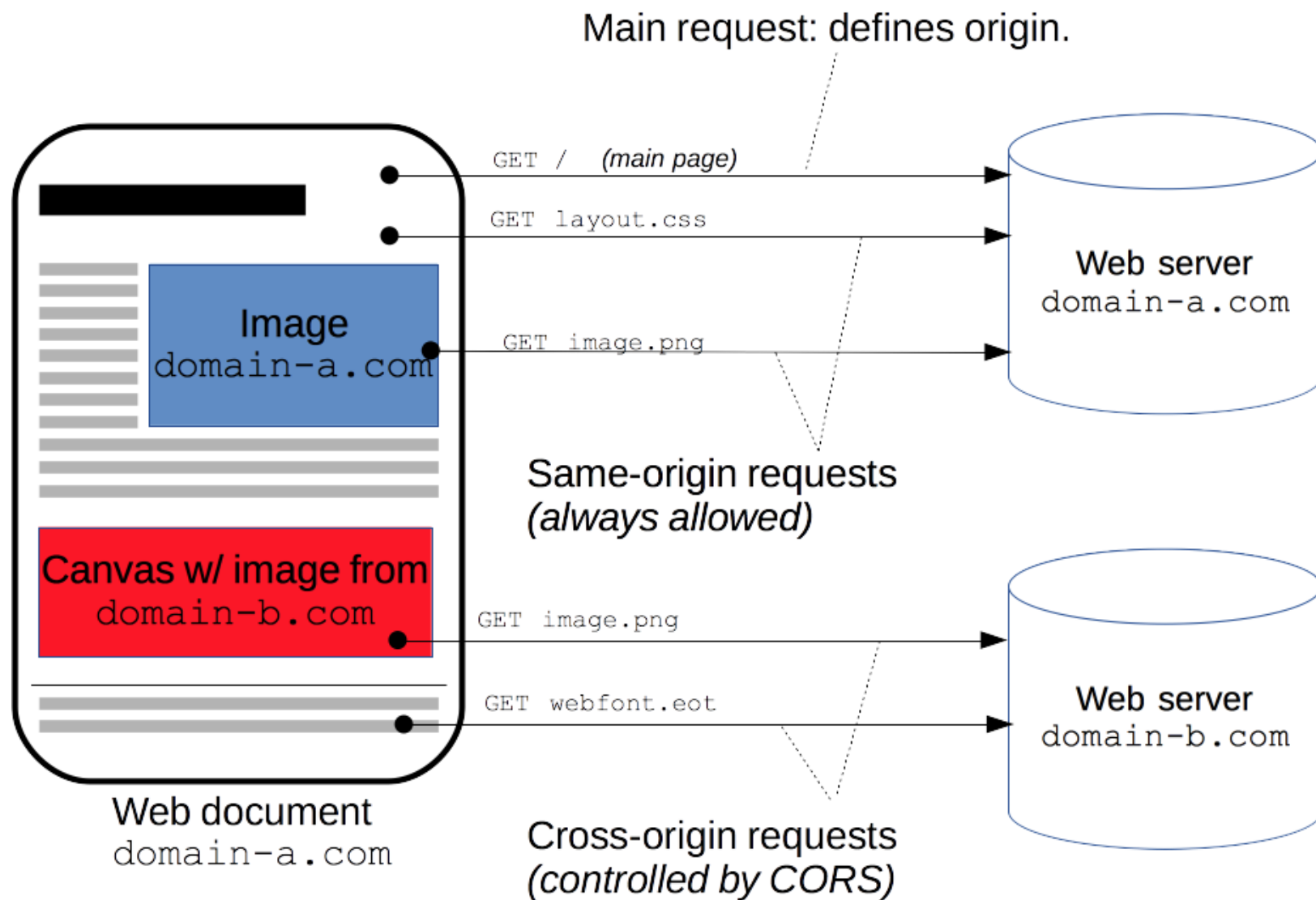
Content-Type →application/json

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/

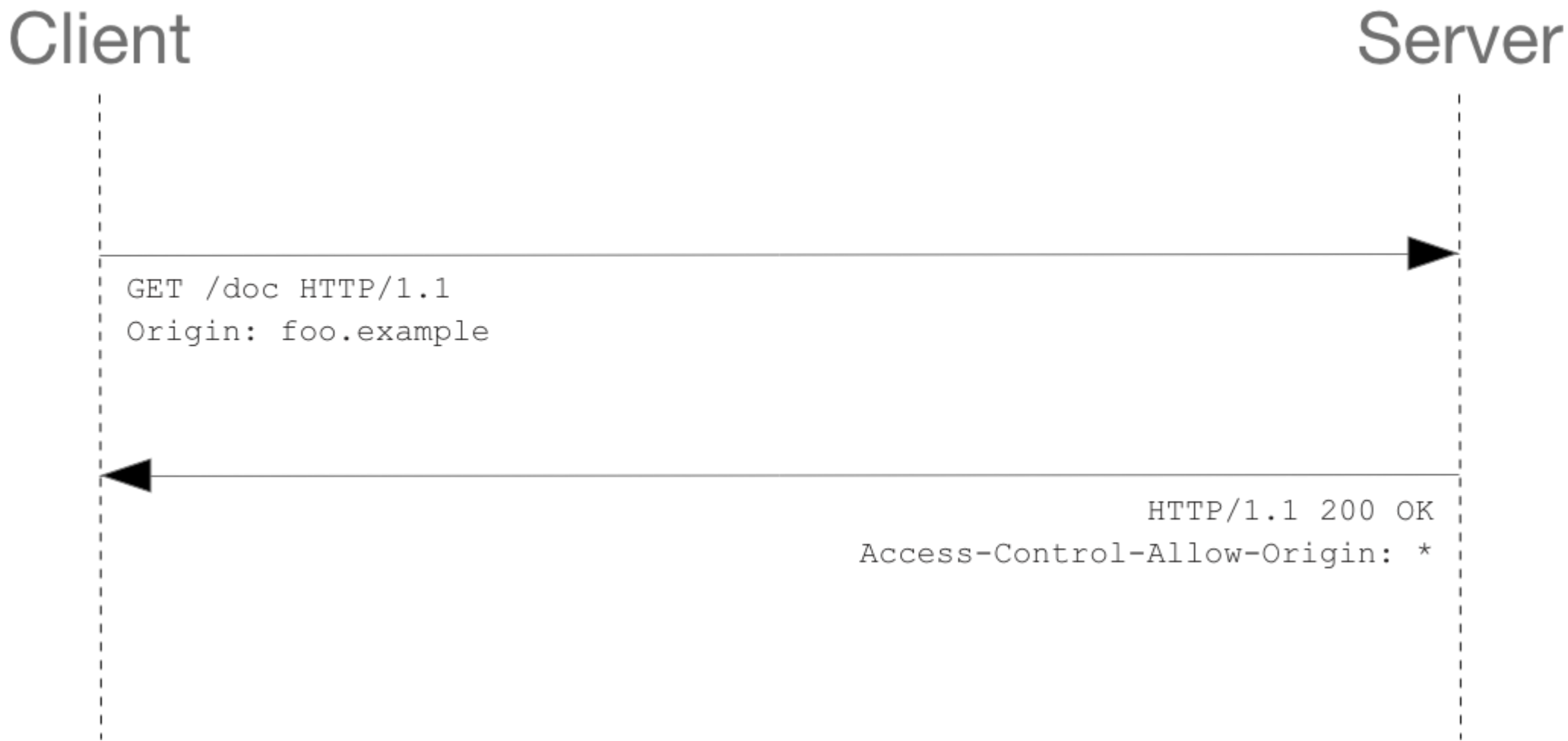
Origin: http://sample.com

- 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

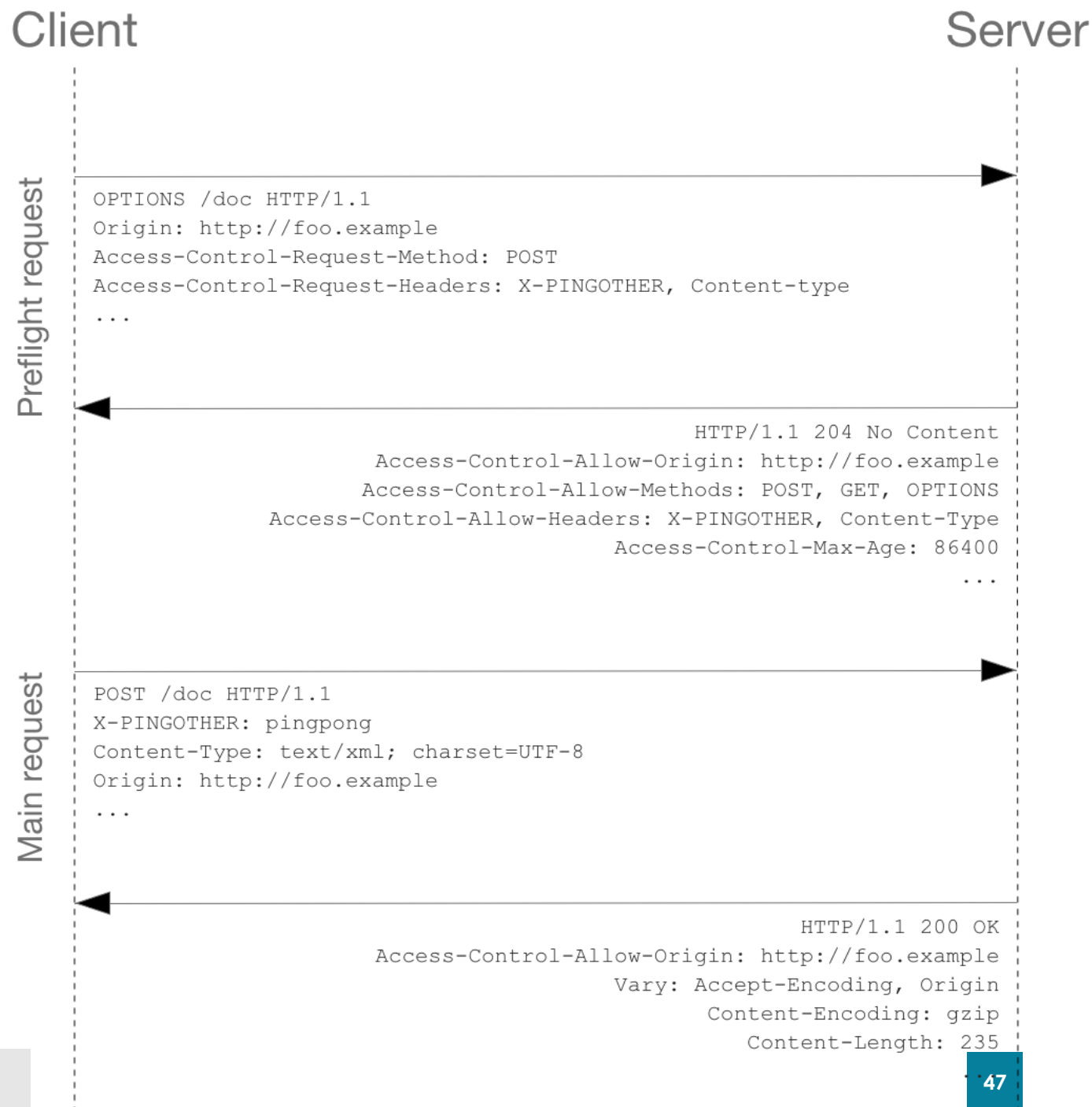
Access-Control-Allow-Origin: http://sample.com

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:



Server Sent Events (SSE)

[\[https://www.w3schools.com/html/html5_serversentevents.asp\]](https://www.w3schools.com/html/html5_serversentevents.asp)

- Traditionally, a web page has to send a request to the server to receive new data; that is, the page requests data from the server.
- With [server-sent events](#), it's possible for a server to send new data to a web page at any time, by [pushing messages](#) to the web page.
- These incoming messages can be treated as *Events* + *data* inside the web page.

https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events

Server Sent Events (SSE)

```
if (!!window.EventSource) {  
    var eventSource = new EventSource(api + "quotes");  
  
    eventSource.onmessage = function(e) {  
        var datapoint = JSON.parse(e.data)  
        console.log(datapoint);  
        chart.series.filter(function (serie) {  
            return serie.name == datapoint.symbol  
        }).forEach(function (serie) {  
            var shift = serie.data.length > 40;  
            serie.addPoint([datapoint.instant, datapoint.price], true, shift);  
        });  
  
        eventSource.addEventListener('open', function(e) {  
            console.log('Opened: ' + e);  
        }, false);  
  
        $('#content').bind('unload', function() {  
            eventSource.close();  
        });  
    }  
}
```

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>