CSE 106

Lecture 11 – AJAX and REST

Acknowledgement: w3schools.com, developer.mozilla.org, mulesoft.com/resources/api/what-is-rest-api-design, docs.microsoft.com/en-us/azure/architecture/best-practices/api-design

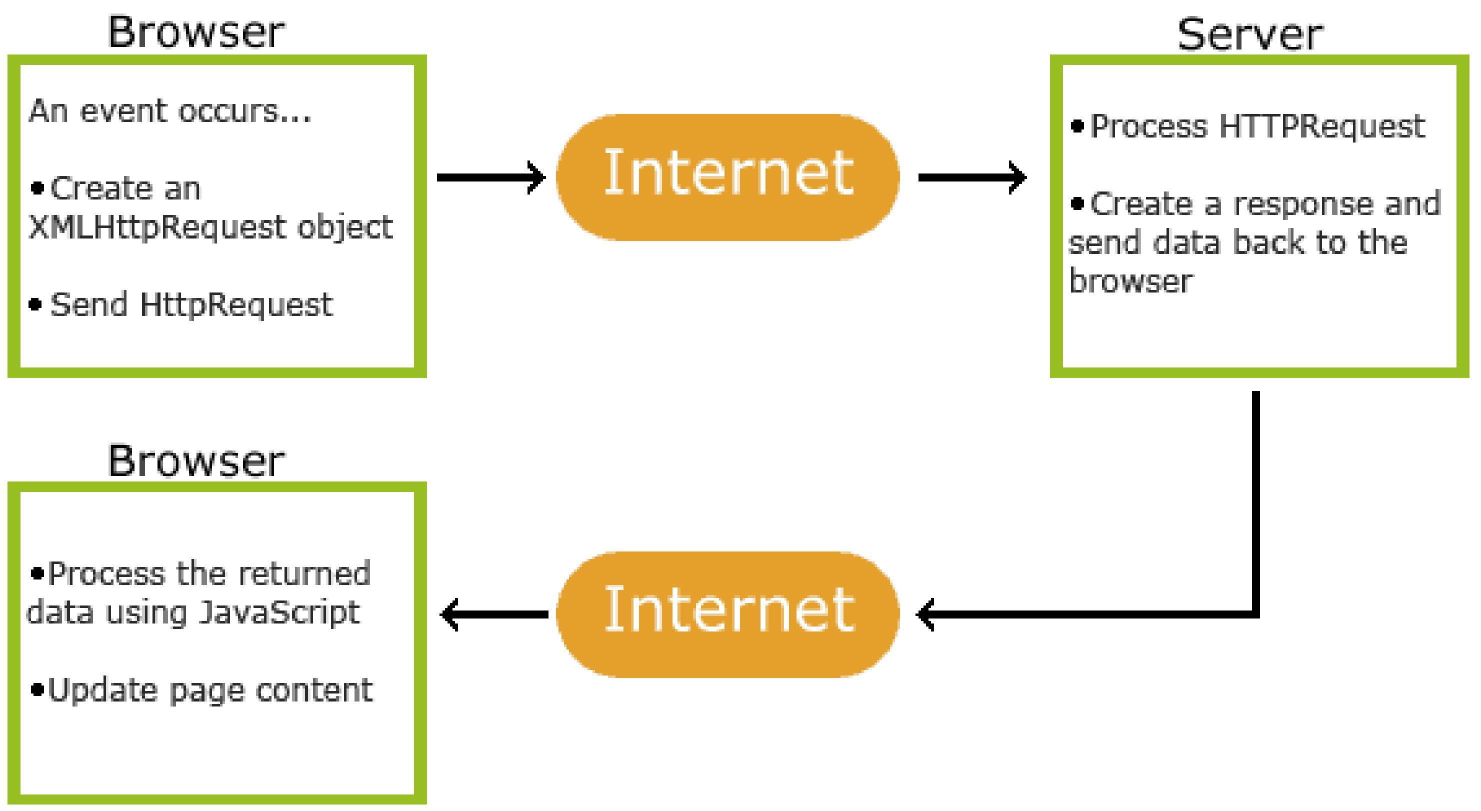
AJAX

- Stands for Asynchronous JavaScript And XML (not just for XML)
  - Send/receive plain text or JSON too
- With AJAX, you can:
  - Update a web page without reloading the page
  - Request data from a server after the page has loaded
  - Receive data from a server after the page has loaded
  - Send data to a server in the background
- Uses **XMLHttpRequest** object in JS

AJAX - XMLHttpRequest

```
const xhttp = new XMLHttpRequest();
const method = "GET";  // Could be GET, POST, PUT, DELETE, etc.
const url = "https:://sample.com";
const async = true;   // asynchronous (true) or synchronous (false) – don’t use synchronous
xhttp.open(method, url, async);
xhttp.send();
xhttp.onload = function() {
  document.getElementById("demo").innerHTML = this.responseText;
};
```

How AJAX works


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Internet
Browser
•Process the returned data using JavaScript
•Update page content

AJAX - XMLHttpRequest

```
<div id="demo">
  <button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
  function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
    xhttp.onload = function() {
      document.getElementById("demo").innerHTML = this.responseText;
    };
}
</script>
```

POST

```
var xhttp = new XMLHttpRequest();
xhttp.open("POST", url);
xhttp.setRequestHeader("Content-Type", "application/json");
const body = {"id": 235625, "jail cell": "1141A"};
xhttp.send(JSON.stringify(body));

xhttp.onload = function() {
  document.getElementById("demo").innerHTML = this.responseText;
};
```

XMLHttpRequest Callbacks

- Onload - called when an XMLHttpRequest transaction completes successfully

```
const xmlhttp = new XMLHttpRequest();
const method = 'GET';
const url = 'https://developer.mozilla.org/';
xmlhttp.open(method, url, true);
xmlhttp.send();
xmlhttp.onload = function () {
 // Do something with the retrieved data ( found in xmlhttp.response )
};
```

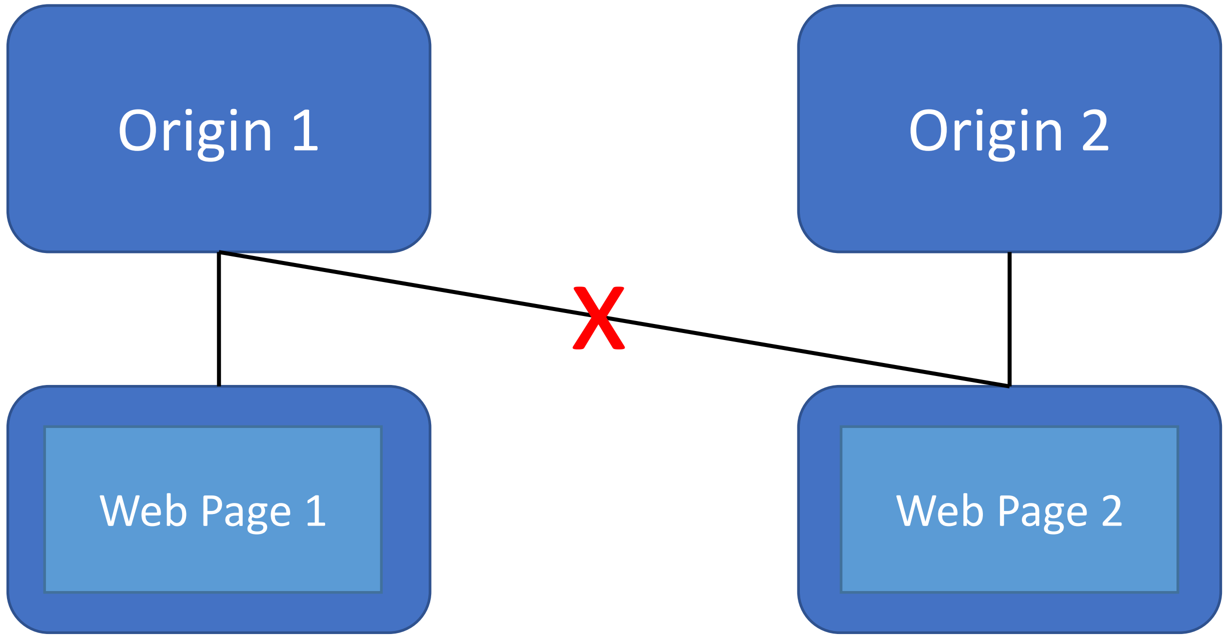
XMLHttpRequest Callbacks

- Onerror - called when an XMLHttpRequest transaction fails due to an error

```
const xmlhttp = new XMLHttpRequest();
const method = 'GET';
const url = 'https://developer.mozilla.org/';
xmlhttp.open(method, url, true);
xmlhttp.send();
xmlhttp.onerror = function () {
 console.log("** An error occurred during the transaction");
};
```

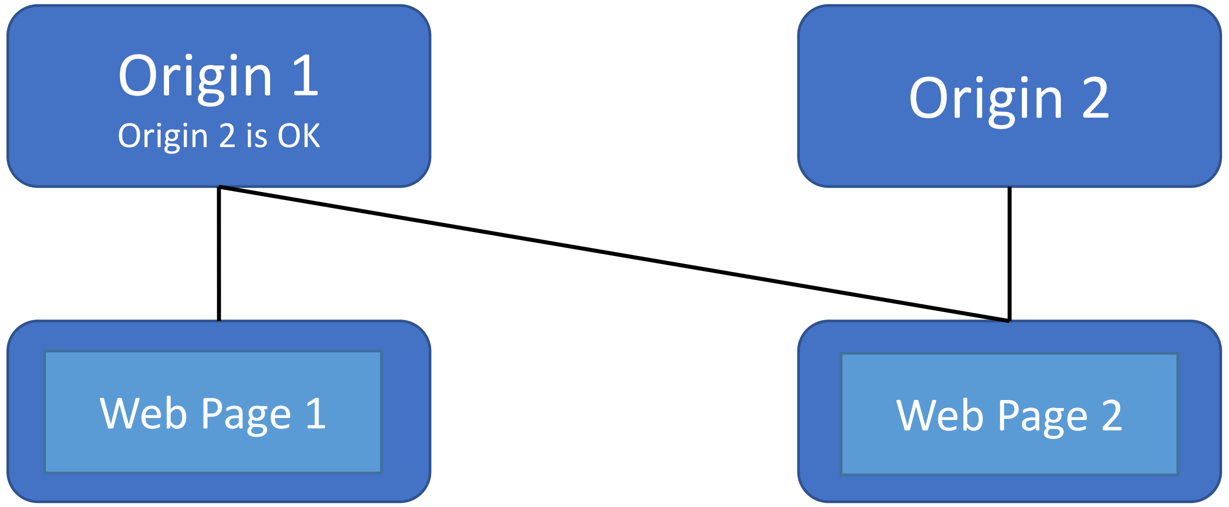
Same-origin policy

- For security, browsers do not allow access across domains by default
- Generally, both the web page and the resources it requests must be hosted by the same server



Cross Origin Resource Sharing

- Cross-Origin Resource Sharing (CORS) allows a server to indicate any domain other than its own, for a browser to permit loading resources
  - Server includes a specific header in responses to include a domain
  - Can set it up to accept all methods from any origin
  - Used for APIs (you'll do this in your lab this week)



Wakeup!

- https://youtu.be/vM7IDgCuL8s

REST API

- REST is an acronym for **Re**presentational **S**tate **T**ransfer
- REST defined by Dr. Roy Fielding in his 2000 doctorate dissertation
- An architecture style that is client-server and stateless
- Noted for its level of flexibility
- Protocol independent, but uses the HTTP protocol for Web APIs
- Can return XML, JSON, YAML or any other format

Designing a RESTful API using HTTP

- Designed around resources
  - Any kind of object, data, or service that can be accessed by the client
- A resource has an identifier, which is a URI that uniquely identifies it
- Clients interact with a service by exchanging representations of resources
  - Many web APIs use JSON as the exchange format
- For example, a GET request to the below URI for a particular customer order might return this JSON:

```
GET https://adventure-works.com/orders/1
```

```
{"orderId":1,"orderValue":99.90,"productId":1,"quantity":1}
```

Designing a RESTful API using HTTP

- REST APIs use a uniform interface, to decouple the client and server
- For REST APIs on HTTP this includes using standard HTTP verbs
    - GET, POST, PUT, DELETE, etc
- REST APIs use a stateless request model
    - HTTP requests should be independent and may occur in any order
    - No need to retain any affinity between clients and specific servers
    - Enables any server can handle any request from any client

Tips to Design a RESTful API - URI

- Focus on the business entities that the web API exposes
  - For example, in an e-commerce system, the primary entities might be customers and orders
- Resource URIs should be based on nouns (the resource) and not verbs (the operations on the resource)

```
https://adventure-works.com/orders // Good
https://adventure-works.com/create-order // Avoid
```

Tips to Design a RESTful API – HTTP Methods

- Define API operations in terms of HTTP methods
  - GET
    - **Retrieves** a representation of the resource at the specified URI
    - The body of the response message contains the details of the requested resource
  - POST
    - **Creates** a new resource at the specified URI
    - The request body provides the details of the new resource (often in JSON)
  - PUT
    - **Updates** the resource at the specified URI
    - The request body specifies the details of the update (often in JSON)
  - DELETE
    - **Removes** the resource at the specified URI

Tips to Design a RESTful API – HTTP Methods

- Use identifier when accessing, editing or deleting a specific resource

```
GET /books
GET /books/<id>
POST /books
      Body: {"id":21352, "title":"The Hobbit", Author:"JRR Tolkien"}
PUT /books/<id>
      Body: {"title":"The Hobbit", Author:"Peter Jackson"}
DELETE /books/<id>
```

Tips to Design a RESTful API – HTTP Status

- Return standard HTTP response codes to indicate the error
- Common codes:
  - 400 Bad Request – client-side input fails validation
  - 401 Unauthorized – The user isn’t not authorized to access a resource
  - 403 Forbidden – The user is authenticated, but is not allowed
  - 404 Not Found – A resource is not found
  - 503 Service Unavailable – Something unexpected happened on server side
- Include error messages with error codes

REST API Implementation

- REST API will often be built using a web framework (like Flask)
- Web frameworks have methods to make implementing a REST API very easy
- Hosted by a server and given a domain name
- Add CORS if you want other origins to access the API