

[Lesson 8-9]

Roi Yehoshua 2018

[Our targets for today]

- JSON format
- AJAX
- XHR
- jQuery.ajax()

[JSON]

- JSON stands for JavaScript Object Notation
- JSON is a human-readable text-based format for transmitting data
- It is language-independent format derived from JavaScript but is widely used outside it as well
- It is used to transfer serialized data between different systems like web client and server

- The string "application/json" is the official internet media type for JSON
- Extension ".json" is used for JSON files

- It consists of attribute-value pairs and array data types
- Values generally follow JavaScript conventions

[JSON values]

→ JSON values can be:

- **ordered list** - javascript array-like data
- **unordered list** - javascript object-like data
- **string** - must be taken in double quotes. Stored in UTF-8 or in UTF-16 when using backslash escapement
- **number** - a signed decimal number that may contain a fractional part and may use exponential E notation
- **boolean** - either of the values true or false
- **null** - an empty value. null must be used instead of NaN, undefined and Infinity

[Unordered list]

- Unordered object-like data must begin with { symbol and ends with } symbol
- Attribute name must be in double quotes
- Attribute and value must be separated by : symbol (colon)
- Attribute/value pairs must be separated by , symbol (comma)

```
{  
  "id": 12345,  
  "fullName": "Dan Jobs",  
  "role": "admin"  
}
```

[Ordered list and nesting]

- Ordered array-like data must begin with [symbol and ends with] symbol
- Attribute/value pairs must be separated by , symbol (comma)

```
[ "Dan", "Joe", "Steve", "Bob" ]
```

- Ordered and Unordered types can be nested in each other

```
{  
  "name": {  
    "simple": "Zebra",  
    "scientific": "Hippotigris"  
  },  
  "tags": [ "horse-like", "striped", "vegetarian" ]  
}
```

[JSON Object]

→ In JavaScript, there is a special object called JSON

→ It has two methods

- **JSON.parse()** - parses a JSON string, constructing the JavaScript value or object described by the string
- **JSON.stringify()** - converts a JavaScript object or value to a JSON string

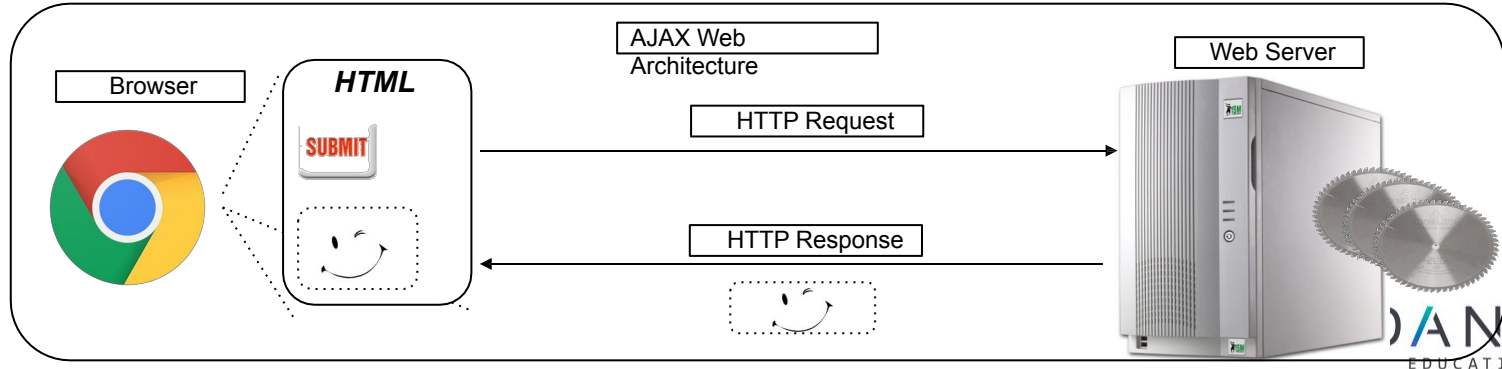
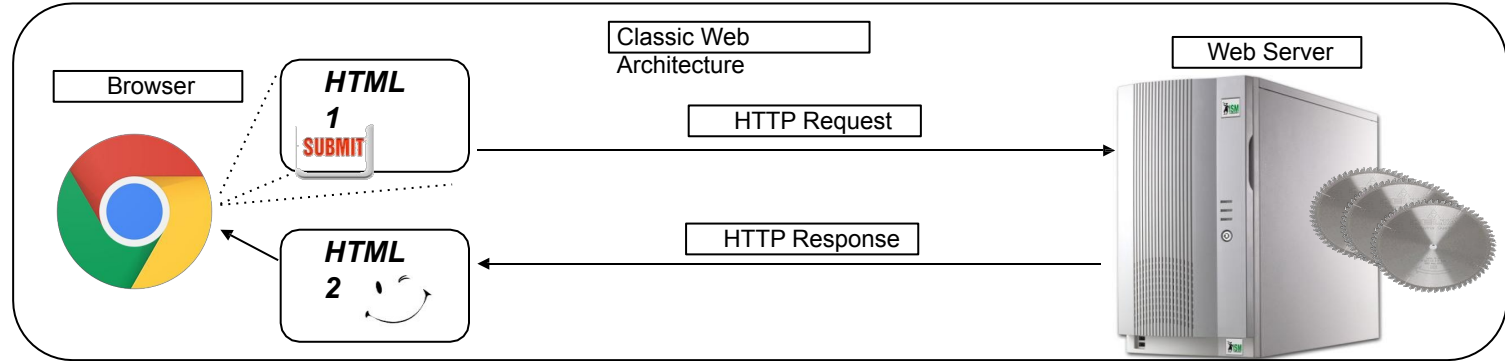
```
console.log(JSON.stringify({ x: 5, y: 6 }));  
// expected output: '{"x":5,"y":6}'  
  
console.log(JSON.stringify({ x: [10, undefined, function(){}], Symbol('')] }));  
// expected output: '{"x":[10,null,null,null]}'  
  
console.log(JSON.stringify(new Date(2006, 0, 2, 15, 4, 5)));  
// expected output: '"2006-01-02T15:04:05.000Z'"
```

```
var json = '{"result":true, "count":42}';  
obj = JSON.parse(json);  
  
console.log(obj.count);  
// expected output: 42  
  
console.log(obj.result);  
// expected output: true
```

[AJAX]

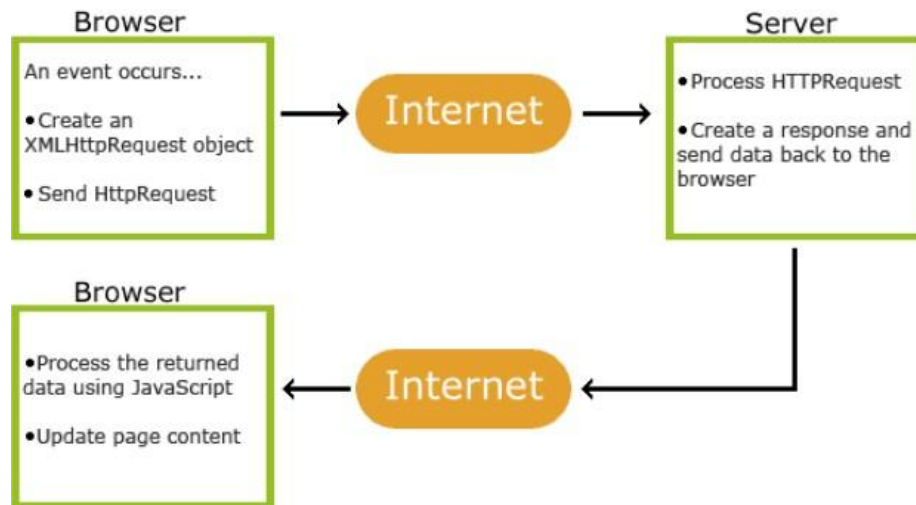
- Asynchronous Javascript And XML
- A combination of technologies
 - XML – data exchange (it also can be JSON, HTML & plain text)
 - JavaScript – XMLHttpRequest object (**XHR**)
- Send & receive data on the background
 - Parts of the page are "refreshed", without reloading the whole page
 - Smoother user experience – no need to wait for a new page to load
- AJAX is a misleading name, since AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text

[Classic vs. AJAX Web Architectures]



[How AJAX Works]

- The built-in **XMLHttpRequest** object can be used in JavaScript to exchange data with a server behind the scenes



[Generating AJAX Requests]

url – the address of this ajax call. May send parameters just like any HTTP request

When a response is received stateChange() function will be asynchronously called

open() - Setting request data format:

- method (GET/POST)
- url
- asynchronous call – true enables it (default = true)

Send method takes a DOM Object.
It may take an object that represents the request body (in case of an HTTP POST request)

stateChange() method will be called asynchronously. Will be explained later.

```
function generateRequest()
{
  const xhr = new XMLHttpRequest();
  let url="http://localhost:8080/service";
  url+="?command=doIt";
  xhr.onreadystatechange=stateChange;
  xhr.open("GET", url, true);
  xhr.send();
}

function stateChange()
{
  if (xhr.readyState==4)
    // ...some code here...
  else
    alert("Problem retrieving data");
}
...
```

[Handling Server Response]

→ XMLHttpRequest response related methods & fields:

Method / Field	Description
onreadystatechange	Event handler for an event that fires at every state change
readyState	Object status: 0 = uninitialized 3 = processing request 1 = connection established 4 = response is ready 2 = request received
responseText	Get the server's data as a string
responseXML	Get the server's data as an XML DOM object
status, statusText	Numeric code & description of the HTTP status returned by server such as 404 for "Not Found" or 200 for "OK"

[First Example – Server Side]

- First, create a simple calculator service in ExpressJS
- Create a new NodeJS package named ajax-demo
 - Run "npm install express"
- Create app.js and copy the following code into it

```
const express =  
require('express');  
const path = require('path');  
const app = express();  
app.use(express.static(path.join(__dirname, 'public')));  
  
app.get('/add', (req, res) => {  
  if (!req.query.num1 ||  
      !req.query.num2) return  
    res.sendStatus(400);  
  
  let num1 = Number(req.query.num1);  
  let num2 = Number(req.query.num2);  
  let result = num1 + num2;  
  res.send(result.toString());  
});  
app.listen(3000);
```

[First Example – Client Side]

- Add calculator.html to the /public folder of your app
- Copy the following code into it:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Calculator</title>
</head>
<body>
  Num1: <input type="number" id="num1" /><br />
  Num2: <input type="number" id="num2" /><br />
  <button id="btnAdd">+</button><br />
  <span id="sResult"></span>

  <script>
  </script>
</body>
</html>
```

Num1:

Num2:

Note that when working with AJAX, we neither need a <form> tag, nor a submit button (we just use a “regular” button)

[First Example – Client Side]

→ Add the following script to the bottom of the page:

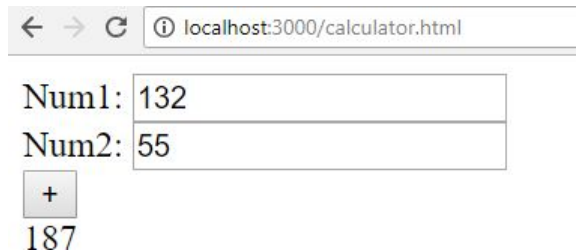
```
<script>
  const btnAdd = document.getElementById("btnAdd");

  btnAdd.onclick = () => {
    const xhr = new XMLHttpRequest();
    const num1 = document.getElementById("num1").value;
    const num2 = document.getElementById("num2").value;

    let url = `/add?num1=${num1}&num2=${num2}`;
    xhr.onreadystatechange = () => {
      if (xhr.readyState == 4 && xhr.status == 200) {
        const result = Number(xhr.responseText);
        document.getElementById("sResult").innerHTML = result;
      }
    };
    xhr.open("GET", url, true);
    xhr.send();
  };
</script>
```

[First Example – Client Side]

- Run your server
 - nodemon app.js
- Go to <http://localhost:3000/calculator.html>

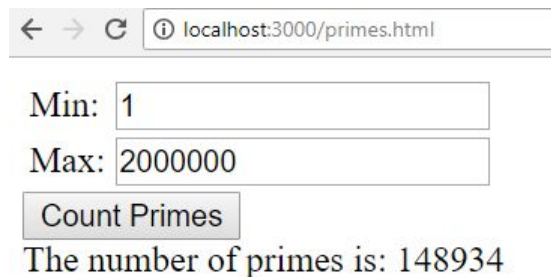


A screenshot of a web browser window. The address bar shows 'localhost:3000/calculator.html'. The page content includes two input fields: 'Num1: 132' and 'Num2: 55'. Below these is a button with a '+' sign. The result '187' is displayed below the button.

- Note how the page does not refresh when clicking the button!

[Exercise (1)]

- Create a NodeJS server that receives two numbers (min, max) and returns how many prime numbers exist between min and max
- Create an HTML page that enables the user to enter (min, max) and displays the number of prime numbers in the specified range, by using an AJAX call to the server



A screenshot of a web browser window. The address bar shows 'localhost:3000/primes.html'. Below the address bar, there are two input fields. The first is labeled 'Min:' and contains the value '1'. The second is labeled 'Max:' and contains the value '2000000'. Below these fields is a button labeled 'Count Primes'. Below the button, the text 'The number of primes is: 148934' is displayed.

[AJAX & jQuery]

- **\$.ajax()** is the main AJAX method
 - Has many shortcut methods
- Provides precise control over your Ajax call
- Supports all HTTP methods and various formats of data
- Gets a set of key/value pairs that configure the Ajax request

```
$.ajax({  
  method: "POST",  
  url: "/server/path",  
  data: { name: "John", location: "Boston" }  
})  
  
.done(function (msg) {  
  alert("Data Saved: " + msg);  
});
```

[AJAX & jQuery]

→ Useful AJAX settings:

Setting	Description
url	A string containing the URL to which the request is sent (default: the current page)
method	The HTTP method to use for the request (default: "GET")
data	The data to be sent to the server. Optional. This can either be an object or a query string, such as foo=bar&baz=bim
contentType	The type of data sent to the server (default: "application/x-www-form-urlencoded")
dataType	The type of data that you're expecting back from the server, e.g. "xml", "json", "html"
timeout	The time in milliseconds to wait before considering the request a failure
async	Set to false if the request should be sent synchronously. Defaults to true. Note that if you set this option to false, your request will block execution of other code until the response is received
cache	Whether to use a cached response if available. Defaults to true for all dataTypes except "script" and "jsonp". When set to false, the URL will simply have a cache busting parameter appended to it

→ For the full list of settings visit <http://api.jquery.com/jquery.ajax/>

[AJAX & jQuery]

- The `jqXHR` object returned by `$.ajax()` implements the Promise interface
 - This allows you to assign multiple callbacks on a single request, and even to assign callbacks after the request may have completed
- Available Promise methods of the `jqXHR` object include:
 - **`jqXHR.done`**(function(data, textStatus, jqXHR) {}) - called in response to a successful request
 - **`jqXHR.fail`**(function(jqXHR, textStatus, errorThrown) {}) - called in response to a failed request
 - **`jqXHR.always`**(function(data|jqXHR, textStatus, jqXHR|errorThrown) {}) - called in response to both successful and failed requests
 - **`jqXHR.then`**(function(data, textStatus, jqXHR) {}, function(jqXHR, textStatus, errorThrown) {}) - incorporates the functionality of the `.done()` and `.fail()` methods, allowing the underlying Promise to be manipulated

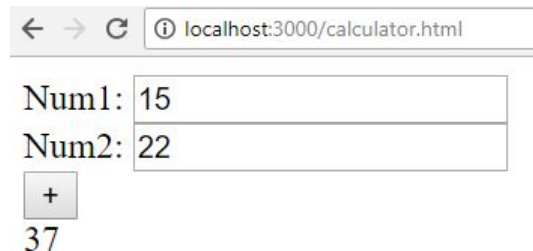
[Example – Client Side]

- Add the jquery library to your /public/scripts folder
- Change the script code in calculator.html to the following:

```
<script src="scripts/jquery-3.3.1.js"></script>

<script>
    $("#btnAdd").click(() => {
        const num1 =
            ($("#num1").val());
        const num2 = ($("#num2").val());

        $.ajax({
            url: "/add",
            data: { num1, num2 },
        })
        .done(result => {
            $("#sResult").html(result);
        });
    });
</script>
```



← → ↻ ⓘ localhost:3000/calculator.html

Num1: 15

Num2: 22

+

37

[AJAX Convenience Methods]

- If you don't need the extensive configurability of `$.ajax()`, and you don't care about handling errors, the Ajax convenience functions provided by jQuery can be useful, providing easier ways to accomplish Ajax requests
- These methods are just "wrappers" around the core `$.ajax()` method, and simply pre-set some of the options on the `$.ajax()` method

Method	Description
<code>\$.get()</code>	Perform a GET request to the provided URL
<code>\$.post()</code>	Perform a POST request to the provided URL
<code>\$.getScript()</code>	Add a script to the page
<code>\$.getJSON()</code>	Perform a GET request, and expect JSON to be returned

[AJAX Convenience Methods]

→ These convenience methods take the following arguments, in order:

Argument	Description
url	The URL for the request. Required.
data	The data to be sent to the server. Optional. This can either be an object or a query string, such as <code>foo=bar&baz=bim</code>
success callback	A callback function to run if the request succeeds. Optional. The function receives the response data (converted to a JavaScript object if the data type was JSON), as well as the text status of the request and the raw request object.
data type	The type of data you expect back from the server. Optional. This option is only applicable for methods that don't already specify the data type in their name.

[Example for AJAX Convenience Method]

→ For example, we can use \$.get() instead of \$.ajax() in our previous example:

```
<script>
    $("#btnAdd").click(() => {
        const num1 = $("#num1").val();
        const num2 = $("#num2").val();

        $.get("/add", {
            url: "/add",
            data: { num1, num2 },
        }, function (result) {
            $("#sResult").html(result);
        })
    });
</script>
```


[AJAX Load]

- The .load() method is unique among jQuery's Ajax methods in that it is called on a selection
- The .load() method fetches HTML from a URL, and uses the returned HTML to populate the selected element(s)
- In addition to providing a URL to the method, you can optionally provide a selector: jQuery will fetch only the matching content from the returned HTML

```
// Using .load() to populate an element
$("#newContent").load("/foo.html");

// Using .load() to populate an element based on a selector
$("#newContent").load("/foo.html #myDiv", function (html)
    { alert("Content updated!");
});
```

[Getting JSON from the Server]

- **\$.getJSON**(url [, data] [, success]) - loads JSON-encoded data from the server using a GET HTTP request
- For example:

```
$.getJSON("/server/path", function (data) {  
    for (let item of data) {  
        $("- " + item + "</li>").appendTo("#mylist");  
    }  
});

```

[Exercise (2)]

- Create a server that stores a list of countries
 - A list can be downloaded from [here](#)
- Define a function that returns a list of countries, whose name starts with a given prefix
- Create an HTML page with an autocomplete box
 - Use jQuery UI autocomplete() function <https://jqueryui.com/autocomplete/>
- When the user starts to type a country name, display a list of the countries whose name starts with the input entered

Country:

- Chad
- Chile
- China
- Christmas Island

[AJAX Limitations]

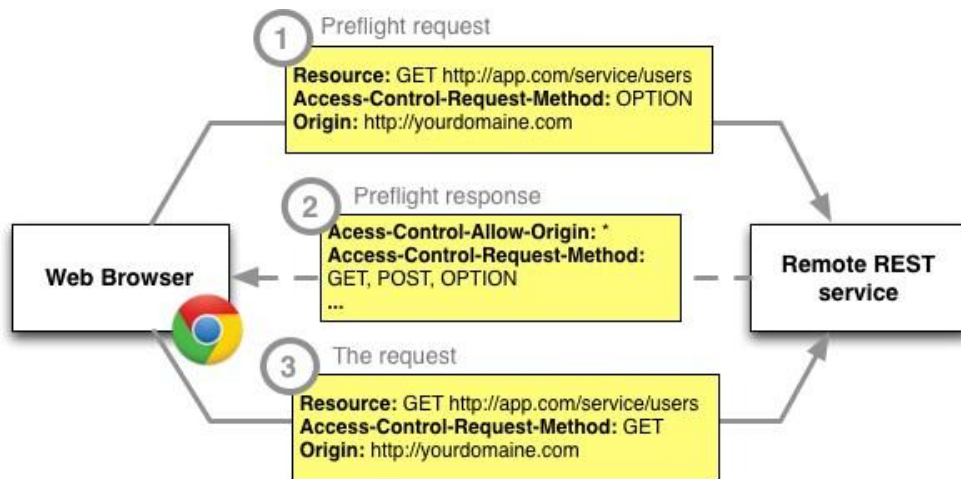
- The same-origin policy
- Prevents loading of resources from one site to another
- It prevents attacks like hijacking user's session
 - You can read more about it [here](#)

<http://www.company.com/page.html>

URL	Same origin?	Reason
http://www.company.com/-page2.html	YES	
http://www.company.com/dir/-page.html	YES	
http://blog.company.com/page.html	NO	Different host
https://www.company.com/-page2.html	NO	Different protocol
http://www.company.com:8080/-page2.html	NO	Different port

[HTML5 CORS]

- Cross-Origin Resource Sharing (CORS) is a W3C spec that allows cross-domain communication from the browser
- JQuery's \$.ajax() method can be used to make both regular XHR and CORS requests



[Control questions]

1. What is JSON, what difference does it have from JavaScript notation?
2. What is AJAX and how does it work?
3. Why do we need AJAX?
4. Name jQuery method for working with AJAX and how you call it from the code?
5. Explain CORS