INFO 2302 Web Technologies

JSON

Dr. Marini Othman
Department of Information Systems
Kulliyyah of Information and Communication Technology
International Islamic University Malaysia

الجامعة الإسلامية العالمية ماليزيا
INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA
يونيۡرسيتي إسلام انتارابڠسا مليسيا
Garden of Knowledge and Virtue

LEADING THE WAY
KHALĪFAH • AMĀNAH • IQRA' • RAḤMATAN LIL-ĀLAMĪN

RCE
Greater Gombak

RCE
REGIONAL CENTRE OF EXPERTISE ON EDUCATION FOR SUSTAINABLE DEVELOPMENT
ACKNOWLEDGED BY
UNITED NATIONS UNIVERSITY

What is JSON?

- JavaScript Object Notation (JSON) is a standard text-based format for representing structured data based on JavaScript object syntax.

- Commonly used for transmitting data in web applications (e.g., sending some data from the server to the client, so it can be displayed on a web page, or vice versa).

JSON Specifications

- JSON was originally created by Douglas Crockford originally in 2001, and initially standardized it in 2006 under RFC 4627 through the IETF.
- In 2013, Ecma International also standardized JSON under ECMA 404.

Serialization and deserialization

Serialization
- Converts a JavaScript object to a string (a JSON file).
- Useful when you want to transmit data across a network.
- JSON.stringify(obj);

- Deserialization
  - Converts a JSON string to a JavaScript object
  - Useful when you want to access the data
  - JSON.parse(string);

JSON file and MIME type

- A JSON string is just a text file with *.json* extension and a MIME type application/json.

JSON Example

```
//JSON Object

{
    "employee": {
        "id": 1,
        "name": "Admin",
        "location": "USA"
    }
}
```

```
//JSON Array

{
    "employees": [
        {
            "id": 1,
            "name": "Admin",
            "location": "India"
        },
        {
            "id": 2,
            "name": "Author",
            "location": "USA"
        },
        {
            "id": 3,
            "name": "Visitor",
            "location": "USA"
        }
    ]
}
```

Stringify a JSON Object

```
<!DOCTYPE html>
<html>
<body>

<h2>Create a JSON string from a JavaScript object.</h2>
<p id="demo"></p>

<script>
const obj = {name: “Ahmad", age: 25, city: “Kuala Lumpur"};
const myJSON = JSON.stringify(obj);
document.getElementById("demo").innerHTML = myJSON;
</script>

</body>
</html>
```

Parsing a JSON String

```
<!DOCTYPE html>
<html>
<body>

<h2>Creating an Object from a JSON String</h2>

<p id="demo"></p>

<script>
const txt = '{"name":“Ahmad", "age":25, "city":“Kuala Lumpur"}’;
const obj = JSON.parse(txt);
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
</script>

</body>
</html>
```

INFO 2302 Web Technologies

Asynchronous JavaScript

Dr. Marini Othman
Department of Information Systems
Kulliyyah of Information and Communication Technology
International Islamic University Malaysia

الجامعة الإسلامية العالمية ماليزيا
INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA
يونيۡرسِيتِي إِسۡلَام أَنتَارَابَڠسَا مَلِيسِيَا
Garden of Knowledge and Virtue

LEADING THE WAY
KHALĪFAH • AMĀNAH • IQRA' • RAḤMATAN LIL-ĀLAMĪN

RCE
Greater Gombak

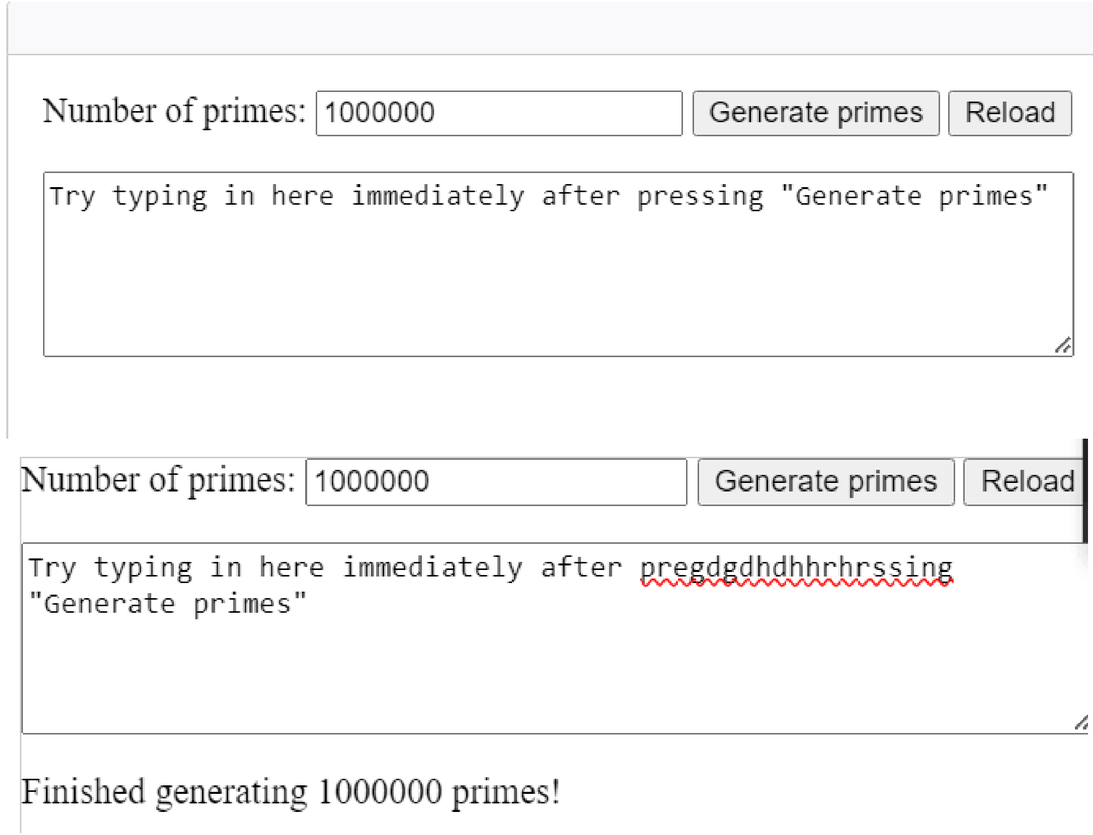
RCE
REGIONAL CENTRE OF EXPERTISE ON EDUCATION FOR SUSTAINABLE DEVELOPMENT
ACKNOWLEDGED BY
UNITED NATIONS UNIVERSITY

Synchronous programming

```
const name = "Miriam";
const greeting = `Hello, my name is ${name}!`;
console.log(greeting);
// "Hello, my name is Miriam!"
```

- The browser waits for the line to finish its work before going on to the next line.
- Each line depends on the work done in the preceding lines.
- That makes this a **synchronous program**

Long-running synchronous program



While generatePrimes() function is running, our program is completely unresponsive: you can't type anything, click anything, or do anything else.

Event handlers

- Event handlers are really a form of asynchronous programming: you provide a function (the event handler) that will be called, not right away, but whenever the event happens.

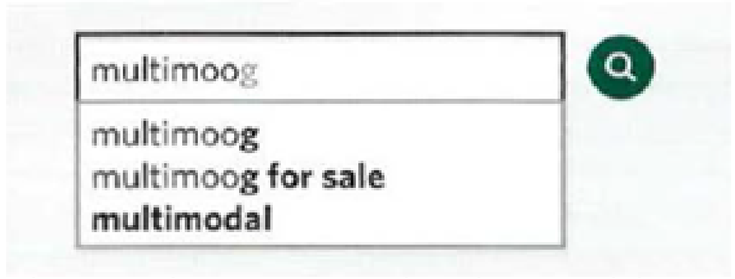
- If "the event" is "the asynchronous operation has completed", then that event could be used to notify the caller about the result of an asynchronous function call.

Asynchronous JavaScript

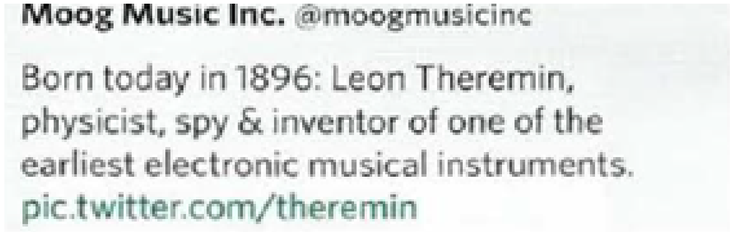
- A web development technique in which a web app fetches content from the server to update the relevant parts of the page without requiring a full-page load.

- This can make the page more responsive because only the parts that need to be updated are requested.

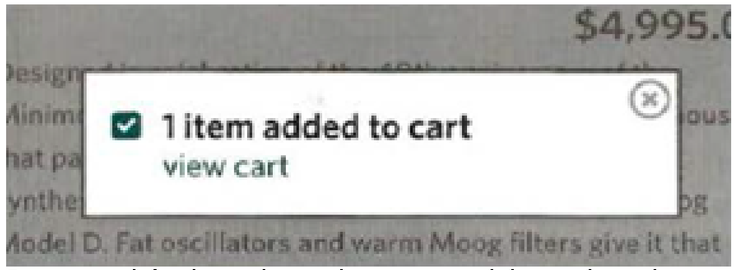
Common uses of asynchronous JavaScript



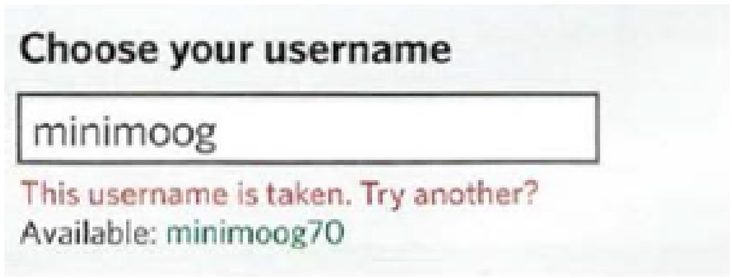
Live search or auto-complete (eg Google)



Websites with user-generated content may allow you to display your information (such as your latest tweets or photographs) on your own website

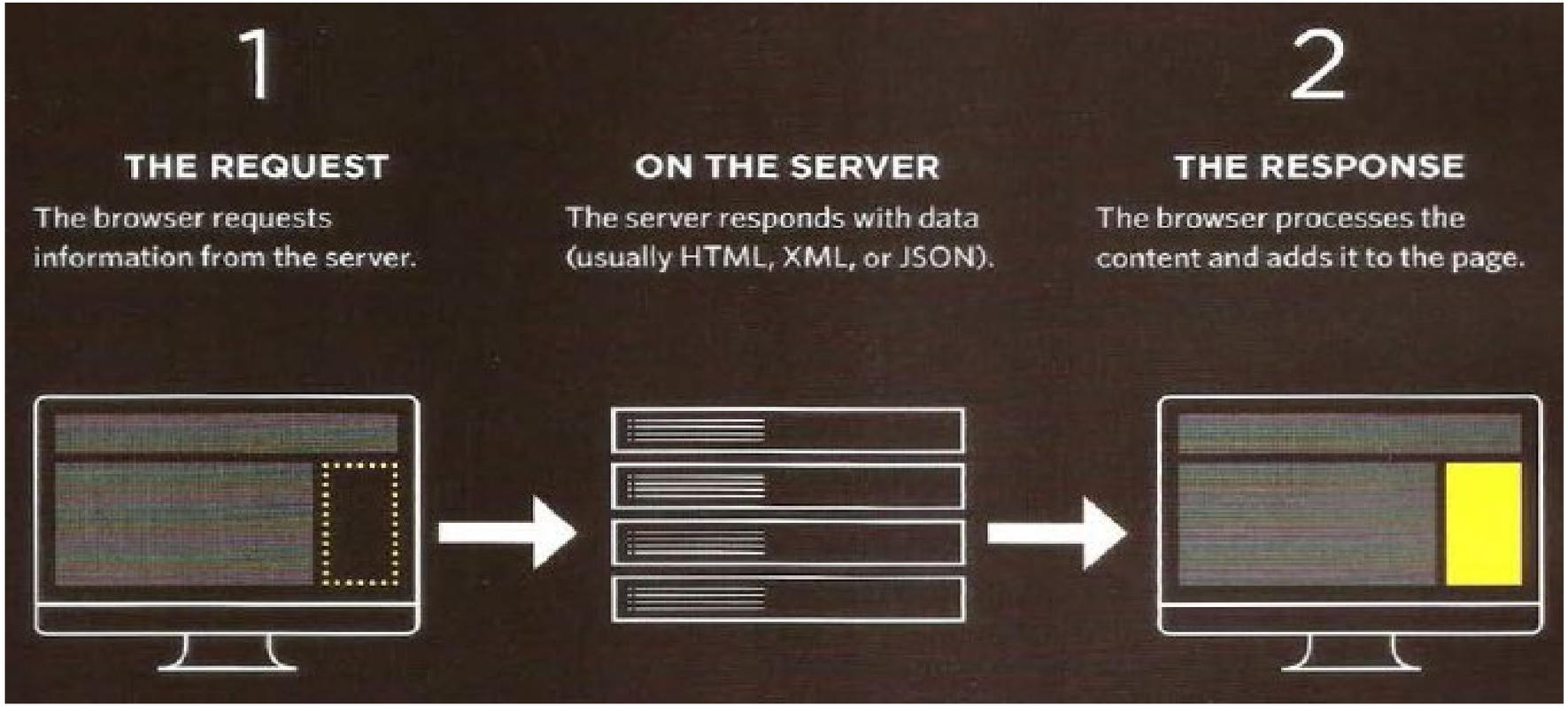


Items added to shopping cart without leaving the page



When registering for a website, the site checks whether your username is available without completing the form

How asynchronous JavaScript works?



JSON vs. XML

- Asynchronous JavaScript was initially used with XML (hence the name Ajax)
- Modern asynchronous JavaScript is increasingly using JSON as the data interchange format due to its simplicity

```xml
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

XML

```json
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

JSON

JSON vs. XML

JSON	XML
Stands for JavaScript Object Notation	Stands for eXtensible Markup Language
A data interchange format consisting a collection of name/value pairs	A document markup and a data interchange format
Smaller file size, easier to manipulate, better performance	Bigger file size, strict format but has support for namespaces and schemas

Approaches for asynchronous JavaScript

1) XMLHttpRequest() object
2) Promise based API

Using XMLHttpRequest object

```
<!—Example: using XMLHttpRequest. To run this program, follow the instructions in the exercise folder-->
<!DOCTYPE html>
	<html>
	<body>
		<div id="foo">
			<h2>The XMLHttpRequest Object</h2>
			<button type="button" onclick="changeContent()"> Change Content</button>
		</div>
		 <script>
			function changeContent() {
				var xhttp = new XMLHttpRequest();
				xhttp.onreadystatechange = function() {
					if (this.readyState == 4 && this.status == 200) {
						document.getElementById("foo").innerHTML = this.responseText;
				} };
				xhttp.open("GET", "content.txt", true);
				xhttp.send(); }
		</script>
	</body>
	</html>
```

Frequently used XMLHttpRequest methods

Methods	Descriptions
open (“method”, “URL”, [async, username, password])	Assigns destination URL, method, etc.
send (params)	Sends request including postable string or DOM object data

Frequently used XMLHttpRequest properties

Properties	Descriptions
onreadystatechange	Event handler (your code) that fires at each state change
readyState	0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	HTTP Status returned from server: 200-299 = OK
responseText	String version of data returned from server
responseXML	XML DOM document of data returned

Using Promise-based API

```
<!--Example: using fetch. To run this program, follow the instructions in the exercise folder-->
<!DOCTYPE html>
    <html>
    <body>
        <div id="foo">
            <h2>Using fetch API</h2>
            <button type="button" onclick="changeContent()"> Change Content</button>
            <div id="result"></div>
        </div>
         <script>
            async function changeContent() {
                let file = "content.txt";
                let myObject = await fetch(file);
                let myText = await myObject.text();
                document.getElementById("result").textContent = myText;
            }
        </script>
    </body>
 </html>
```

Frequently used Promise-based API methods and properties

Methods/property	Description
fetch(resource)	Starts the process of requesting a resource through a network. Returns a Promise that resolves with a Response object.
Response.text()	Returns a promise that resolves with a text representation of the response body.
Response.json()	Returns a promise that resolves with the result of parsing the response body text as JSON.
Response.status	The status code of the response. (This will be 200 for a success).

References

MDN Web Docs: Working with JSON retrieved from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

REST API Tutorial: Introduction to JSON retrieved from https://restfulapi.net/introduction-to-json/