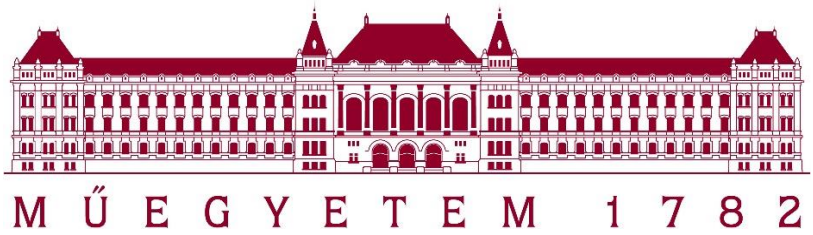


Mobile and Web Development

Department of Automation and Applied Informatics

SCSS, AJAX

Created: 1/2024/2025
Dr. Mohammad Saleem
msaleem@aut.bme.hu

SCSS

Introduction

AJAX

- The client-side JavaScript code gets more and more complicated, and it is executed in a single threaded environment
- Legacy solutions
  - > Active controls (applet, Flash, Silverlight, ActiveX)
- The problems with these solutions
  - > Problematic to install
  - > Not supported by all the browsers
  - > A different technology (not JS)
  - > Security and reliability risk

Sync vs Async

- The main difference when talking about operation synchronicity is in their thread usage. While synchronous processing is done sequentially and in a specific order, asynchronous processing is done in parallel. Tasks that are not dependent on others can be offloaded and executed at the same time as the main operation and then report back the result when they are done.
- When talking about synchronous vs asynchronous communication programming, synchronous communication is often called tightly coupled because all executions depend on each other.
- In contrast, asynchronous communication is called loosely coupled because asynchronous operations are independent.

Is Asynchronous Better Than Synchronous?

Comparison Table

Comparison	Synchronous	Asynchronous
Execution	Sequential, tasks depend on each other	Independent, tasks run concurrently
Blocking	Blocking – each task waits for the previous	Non-blocking – tasks don't wait
Ease of Implementation	Easier to implement and debug	More complex and harder to debug
Performance	Slower, not ideal for long tasks	Faster, better for multitasking
Ideal Use Case	Short, simple tasks	Long-running tasks (e.g., network operations)

Async and Sync Use Cases

You'll want synchronous programming when your project focuses on a single task (especially if it's computationally heavy) or tasks that are highly dependent and cannot run in parallel:

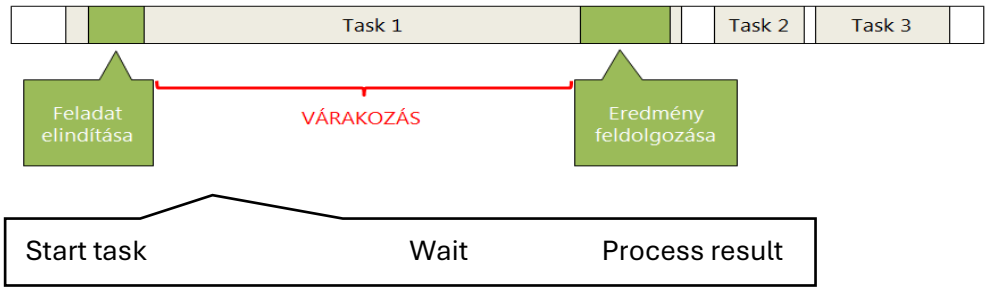
- **Web Pages**: By loading a web page synchronously, you make it easier for Search Engine Optimization (SEO) engines to find and categorize your page's content.

- **Video Rendering**: This is a task that takes a lot from your CPU (if not most of it), and thus running other tasks in parallel would oversaturate it.

You'll want asynchronous programming when you have a lot of tasks that do not depend on each other, or tasks that take a long time and can be run in the background:

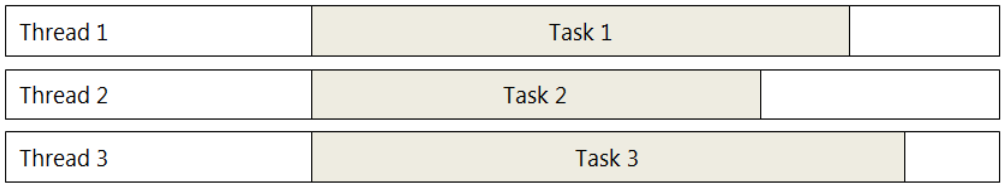
- **Database Manipulation**: Databases, especially large ones, are notorious for the long times queries take when trying to retrieve information from them. By using a different thread for that job, you can let the rest of your application function while you wait for the results.
- **Dashboards**: If you have a complex dashboard with a lot of information to present, you can use different threads to keep each part updated in real-time or as the system can handle.

Execution in one thread (synchronous)



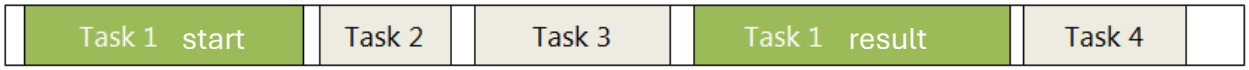
- Advantage
  - > The order of the execution is deterministic
  - > The output of Task 1 can be the input of Task 2

Execution in multiple threads



- Independent execution
- Synchronization and communication between threads are difficult
- In case of multiple CPU core it means real parallelism; otherwise it uses time-sharing in one CPU core.

Asynchronous execution in one thread



- Overlapping execution
  - > Also in systems with multiple CPU cores
- Only one task at a time
- A good choice when executing a blocking operation (e.g. downloading content through the network).
- Processing the result should never be blocking

Synchronous communication

- Client and server-side operations are synchronized
- Bandwidth
  - > Lot of data, the whole page travels through the network
- Server load
  - > Have to process all the input of the request
  - > Have to render the whole page, even when only a minor piece is changed
- User experience, UX
  - > Blocked user interface
  - > User is waiting (click → wait → refresh).
  - > The whole page is refreshed → blink, scroll context is lost

AJAX - Asynchronous JavaScript and XML

AJAX is a developer's dream, because you can:

- Read data from a web server - after a web page has loaded
- Update a web page without reloading the page
- Send data to a web server - in the background
- The client first downloads the whole page but later it only requests the changes
- Page = URL + DOM changes

Advantages of AJAX

- Less data through the network
  - Less network bandwidth
- Decreased server load
- Better user experience (this is the primary goal)
  - Faster
  - No blocking of the UI
  - The whole page is not refreshed
    - Scroll context is preserved

Disadvantages of AJAX

- Open-source. View source is allowed, and anyone can view the code source written for Ajax, which makes it less secure compared to other technologies
- Search Engines cannot index Ajax pages can not be indexed by Google as well as other search engines
- The usage of Ajax can cause difficulties for your web pages to debug as well as make them prone to possible security issues in the future
- Most importantly, Ajax has a considerable dependency on JavaScript, so only browsers that support Javascripts or XMLHttpRequest can use pages with Ajax techniques
- Users will find it challenging to bookmark a specific state of the application due to the dynamic web page

- From the users' perspective, when you click the back button on the browser, you may not return to the previous state of the page but the entire page. This happens because the pages with successive Ajax requests are unable to register with the browser's history

Technologies required

- It requires a lot of technologies
  - > XMLHttpRequest (XHR)
  - > JavaScript
  - > DHTML + DOM
  - > Plain Old XML (POX) or JSON
- jQuery functions
  - > $.ajax, $.load, $.get, $.port, $.script, $.json

XMLHttpRequest (XHR) object

- Originally it was introduced by Microsoft in Outlook Web Access and implemented in IE5 as an ActiveX object.
  - > Other browser later also implemented it as a native JavaScript object (methods are the same, instantiation is different).
  - > In IE7 it is available as a native JS object
- Technically it is a tiny browser (sends HTTP request and receives HTTP response)
  - > Async execution (doesn't block the calling thread, it can also be called synchronously)
  - > The result can be handled in a callback function
  - > Server platform independent

XHR inicialialization

```
// Initialization
if( window.XMLHttpRequest )
{
  // IE7, Mozilla, Safari, Firefox: native objects.
  var xhr = new XMLHttpRequest()
}
else
{
  if( window.ActiveXObject )
  {
    // ActiveX in case of IE5.x and IE6
    var xhr = new ActiveXObject( "Microsoft.XMLHTTP" );
  }
}
```

XHR request/response

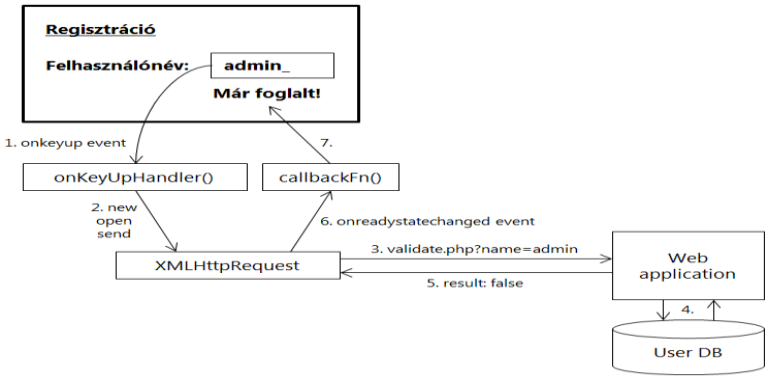
```
// Sending the request
xhr.open( "GET", strUrl, true );  // true = async
xhr.onreadystatechange = onStateChanged;
xhr.send( null );

// Processing the response
function onStateChanged() {
    if( xhr.readyState == 4 ) {    // READYSTATE_COMPLETE
```

```
        if( xhr.status == 200 ) {    // HTTP_OK
            // Processing the xhr.responseText property }
        else { alert( "Hiba történt, a hibakód: " + xhr.status ); }
  }
}
```

Value	State	Description
`0`	`UNSENT`	Client has been created. `open()` not called yet.
`1`	`OPENED`	`open()` has been called.
`2`	`HEADERS_RECEIVED`	`send()` has been called, and headers and status are available.
`3`	`LOADING`	Downloading; `responseText` holds partial data.
`4`	`DONE`	The operation is complete.

When and how to use it?



DHTML and the DOM

DHTML, or Dynamic HTML, is a technology that differs from traditional HTML. DHTML combines HTML, CSS, JavaScript, and the Document Object Model (DOM) to create dynamic content.

- Refresh the user interface based on the server response
  - > DHTML = DOM + JS + CSS
- Variations in different browsers
  - > Requesting HTML elements
  - > Subscribing to events
  - > Different methods and properties

XML / JSON

- Data sent through the network has to be serialized
- XML is verbose
  - > Redundant, too much data
- JSON
  - > Simple, not a markup language, JavaScript object literal subset.

- Built-in support for serialization: IE8+
  - **JSON.parse()** and **JSON.stringify()**

Using JSON

- **Format**: JSON is a text format that represents data objects consisting of key-value pairs.
- **Structure**:
  - **Objects**: Enclosed in curly braces {}, with keys and values.
  - **Arrays**: Enclosed in square brackets [], used for lists of values.
- **Data Types**: Supports strings, numbers, booleans, null, arrays, and objects

```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "skills": ["JavaScript", "Python", "HTML"],
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "zip": "12345"
  }
}
```

Implementation difficulties

- RFC 2616 (HTTP 1.1) Section 8.1.4 Practical Considerations (1999.)
- The maximum number of concurrent AJAX connections a browser can handle is determined by browser-specific limits. These limits vary and can impact the performance of web applications that rely on multiple simultaneous AJAX calls.
- Max. 2 connection / hostname (proxy).
- Modern browsers
  - IE 8-9, Opera, Chrome : 6 connections
  - IE 10: 8 connections
  - Firefox: 6 connections
    - network.http.max-persistent-connections-per-server
- Requests and responses may get lost
  - manual queuing.
- **Same-origin policy.**
- Ajax calls are only allowed to domains where the page was downloaded from
- Scheme + host + port (RFC 6454 The Web Origin Concept)
  - An "origin" is defined as a combination of the protocol (e.g., HTTP/HTTPS), domain (e.g., example.com), and port (e.g., 80, 443). For example:
  - https://example.com:443/page1 and https://example.com:443/page2 share the same origin.
  - https://example.com:443 and http://example.com:80 do not share the same origin (different protocols).
- Redirections
  - Clients have to follow → no dedicated flag or callback

> AJAX redirection issues occur when an AJAX request results in a response that attempts to redirect the browser. Unlike standard browser navigation, AJAX requests do not automatically follow redirects in the same way, and this can lead to unexpected behavior or failure to handle the redirect properly.

- Cookie expires
  > Session cookie expires → the server session expires, but cannot notify the client about it
  > Authentication cookie expires → redirects to the login page, but that is a HTML page (the response is HTML not JSON)
  > Async pinging of the server may solve the problem
- Handling server errors
  > E.g. unhandles exceptions, 5xx server errors, request size exceeded, timeout.
  > The general server side event handler logic redirect to the error page (that it HTML)
  > Invalid XML or JSON content, the client cannot process it
  > Information disclosure.

Cross-domain calls

Cross-domain AJAX calls occur when your web page (client-side) makes a request to a server that resides on a different domain, protocol, or port than the one serving the web page.

- **Same-origin policy**: prevents potentially malicious behavior by restricting how scripts loaded from one origin can interact with resources from another origin.

XDomainRequest (XDR)

is a browser-specific JavaScript object that was introduced by **Microsoft Internet Explorer** (IE) to enable **cross-domain requests** in a way similar to **AJAX**. It was introduced in **Internet Explorer 8** as a way to bypass the **Same-Origin Policy (SOP)** limitations that prevent JavaScript from making requests to a domain other than the one from which the page was served.

- Limitations:
  > Only from IE 8+
  > Only GET or POST (e.g no OPTIONS).
  > No custom header
  > The **Content-Type** can only be **text/plain**
  > No authentication and cookie support
  > Same schema as the page

Cross-Origin Resource Sharing (CORS)

- CORS is a mechanism that allows servers to specify who can access their resources from different origins.
- **How it works**: The server must include specific HTTP headers, such as Access-Control-Allow-Origin, in its response to indicate which domains are allowed to access its resources.
- Standardized solution, became the part of XMLHttpRequest in Level 2 (XHR2).
- W3C Candidate Recommendation (2013. január 29.): http://www.w3.org/TR/cors/
- How does it work?

> The client send the URL of the requesting page in the **Origin** header:
Origin: http://example.com
> Based on the client's origin the server decides if the request can be served. The server sends a response with the **Access-Control-Allow-Origin** header if it can:

```
Access-Control-Allow-Origin: http://example.com
Access-Control-Max-Age: 2520
Access-Control-Allow-Methods: PUT, DELETE
```

> This is the decision of the server
– aka: **HTTP access control**.

Preflight request

A **preflight request** is an initial request sent by the browser to the server to check if the **CORS** (Cross-Origin Resource Sharing) request is safe to send.

- OPTIONS request to the server to find out if a given verb is supported
  > Before the real request (that may have side effects)
  > safe and idempotent methods

```
Origin: http://foo.example
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER
```

- The server responds with the allowed verbs the client may use

```
Access-Control-Allow-Origin: http://foo.example
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER
Access-Control-Max-Age: 1728000
```

- The client caches the resposne.

JSON with Padding (JSONP)

- Also supported by legacy browsers
- JSONP exploits the fact that source code in the <script> tag can be executed even if it comes from an other domain
- The response is a function call with the requested real JSON data as parameter. The name of the function is defined by the client.
- When the script is executed the function gets called with the requested JSON data. The client side can process the data.
- Trick: the same-origin policy doesn't apply to some HTML elements
  - E.g. **img**, **script**, **link**, **iframe**.
- In case of JSON the response is a JSON object

```
http://example.com/getData
{ "name": "Jack the Ripper", "year": 1888 }
```

- In case of JSONP the real response data is wrapped into a function call
  - The client also has to pass the name of the callback function:

```
http://example.com/getData?callback=myCallbackFn
myCallbackFn( { "name": "Jack the Ripper", "year": 1888 } );
```

Steps

1. Create a callback method that gets the requested data as parameter
2. Insert a **<script>** element using JS.
3. Set the **src** attribute of the **<script>** tag to the JSONP URL and pass the name of the callback function as parameter → the browser downloads the content.

- Advantage
  - > Works in all browsers
- Disadvantage
  - > Server has to handle it
  - > Works only with GET, POST is not supported
  - > Error handling is not so rich as with XHR
  - > Security: XSS.
    - **application/javascript** MIME type
    - JavaScript code can be injected into the page

```
$.ajax({
    dataType: "jsonp",  // Specifies that the response will be in JSONP format.
    url: "https://diplomaterv.vik.bme.hu/OData/V1/Theses?$filter=Supervisor/DisplayName eq '" +
$("#supervisor").val() + "'&$expand=Supervisor, Student",
// The URL of the API endpoint with query parameters.
    jsonp: '$callback',  // Specifies the query parameter that the server will use to return JSONP.
The value '$callback' is the default callback function.
    success: function (response, status, xhr) {
// Success callback when the data is fetched successfully.
        $("#result").empty();  // Clears any existing content in the '#result' div.
        const list = response.d.results;  // Extracts the list of results from the response object.

        // Printing the received list to the console for debugging purposes.
        console.log(list);

        // Iterating over each item in the list and appending it to the DOM.
        for (let i = 0; i < list.length; i++) {
            const item = $("<li/>");  // Creates a new <li> element dynamically.
            const link = $("<a/>", {href: list[i].PortalUrl});  // Creates a new <a> element and
sets the href attribute to the student's portal URL.
            link.text(list[i].Student.DisplayName + ": " + list[i].TitleHu);  // Sets the text of
the link to show the student's name and title.
            item.append(link).appendTo("#result");  // Appends the link to the <li> and then appends
the <li> to the #result div.
        }
    }
});
```

- jsonp: '$callback' tells jQuery that the server will respond with a JSONP response where the data is wrapped inside a callback function named $callback.
- The callback=? part of the URL is used to specify the callback function that the server will use to wrap the data.
- In JSONP, the server's response is a call to the callback function, with the actual data passed as an argument to that function. The browser then executes that function and processes the data.

DEMO: https://diplomaterv.vik.bme.hu/en/OData

USEFUL LINKS AND REFERENCES:

https://distantjob.com/blog/synchronous-vs-asynchronous-programming/#:~:text=While%20synchronous%20processing%20is%20done,result%20when%20they%20are%20done.
https://www.mageplaza.com/blog/advantages-and-disadvantages-of-ajax.html
https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readyState

Mobile and Web Development

https://www.geeksforgeeks.org/dhtml-introduction/