Mobil- és webes szoftverek

jQuery, jQuery validate, AJAX

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Gincsai Gábor
gincsai@aut.bme.hu

Mi a jQuery?

- A jQuery egy gyors, kicsi és funkciógazdag JavaScript könyvtár, amivel egyszerű:
  - a HTML dokumentum manipulálása
  - a kliens oldali eseménykezelők készítése
  - az animáció
  - és az aszinkron kommunikáció a szerverrel (ajax kérések)
- Böngészőfüggetlen kód
- Könnyű kiterjeszthetőség
  - Számos jQuery plugin található

forrás: https://jquery.com

Dokumentum betöltősét jelző esemény

- A jQuery egyik legfontosabb eseménye a `document ready` esemény.
- A `document ready` egy jQuery specifikus esemény, ami akkor fut le, amikor a document letöltődött
  - nem az egész window,
  - az előtt mielőtt a képeket és külső hivatkozásokat elkezdené tölteni a böngésző.
  - A DOMContentLoaded eseménynek a megfelelője.
    - Emulálja a DOMContentLoaded viselkedését, ha azt a böngésző nem támogatja.

jQuery betöltése

- CDN:
  - `<script src="http://code.jquery.com/jquery-3.2.1.js"></script>`
  - fallback mechanizmust érdemes készíteni.
- NPM:
  - `npm intall jquery`
- bower:
  - `bower install jquery`
- Vagy egyszerűen letöltjük a jquery oldaláról.

document.ready

- Készítsük el a document ready eseménykezelőt, melyben a konzolra naplózzuk a „Document ready...” szöveget.

```
$(document).ready(function(){
    console.log("Document ready...");
});
```

- Ugyanezt rövidebben az alábbi módon tehetjük meg.

```
$( function(){
    console.log("Document ready...");
});
```

Egyszerű Selectorok

- Minden elem

```
$("*")
```

- A staff ID-jú elem

```
$("#staff")
```

  - Amit először talál meg, ha több azonos ID-jú lenne

- A meta CSS osztállyal rendelkező elemek:

```
$(".meta")
```

- Az összes img tag

```
$("img")
```

Összetett selectorok

- Tag és CSS
  - Minden meta CSS osztállyal rendelkező span.

```
$("span.meta")
```

- Tag és ID
  - A staff ID-val rendelkező div.

```
$("div#staff")
```

- ID vagy CSS
  - A staff ID-val vagy az összes meta CSS osztállyal rendelkező elem.

```
$("#staff, .meta")
```

Hierarchikus selectorok

- Leszármazottak ( ):

```
$("ul img")
```

- Gyerekek (>):

```
$("ul > li > div > img")
```

- Következő (+):

```
$("img + a")
```

- Testvérek (~):

```
$("#staff ~ h2")
```

DEMO

jQuery

- szelektorok,
- események kezelése,
- elemek dinamikus létrehozása

További linkek és információk

- http://learn.jquery.com/about-jquery/
- https://www.tutorialspoint.com/jquery/

jQuery Validate

HTML 5 validáció

- HTML5-ben is van validáció, de
  - minden böngészőben másképp néz ki a hiba,
  - nem lehet CSS-ből és sehogy sem testreszabni,
  - korlátozott a szabályok
  - nem lehet új validációs készíteni

jQuery validate

- Ha a beépített szabályok nem megfelelőek készíthetünk egyedi szabályt.
  - jQuery validate-et pont erre találták ki.
    - Szabályok JavaScriptben megadhatók.
    - Egyedi hibaüzenetet adhatunk meg.
    - Kliens oldalon validál, nem küldi el a szerverre ha hibás.
- Akár generálható is (pl: ASP.NET MVC).

jQuery validate Használata

```
$(document).ready(function() {
    $("#registrationForm").validate({
    rules: {
        firstName: "required",
    }
    messages: {
        firstName: "Kötelező megadni"
    }
});
```

További testreszabási lehetőségek

- **errorClass** és **validClass**
  - Milyen CSS osztály vonatkozzon a helyes és az érvénytelen adatokra.
- **Highlight()** és **unhighlight()** függvények
  - Függvény amivel testreszabhatjuk a validációs szabályok kiértékelése utáni megjelenítést.
- setDefault
  - Alapértelmezett működést tudjuk vele átállítani.

DEMO

jQuery validate

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

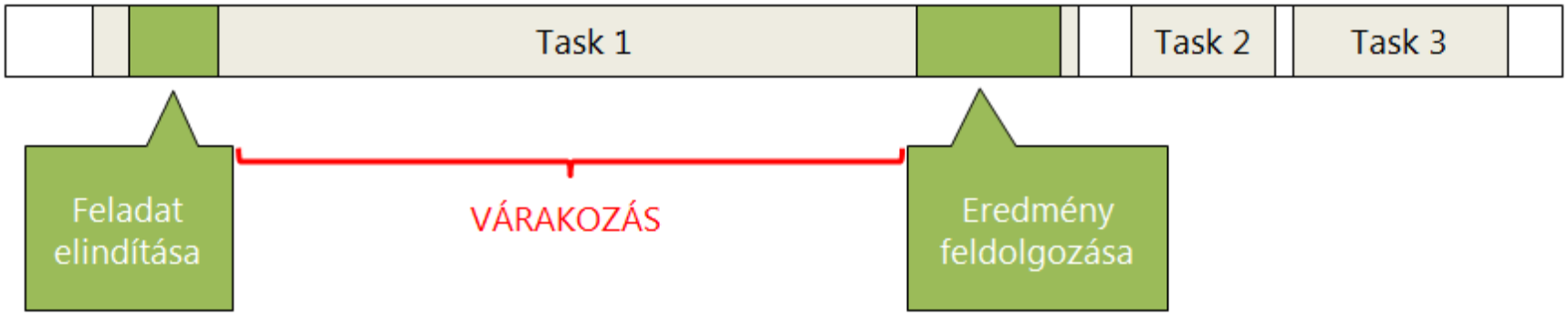
Aszinkronitás

AJAX, JSON, JSONP

Probléma

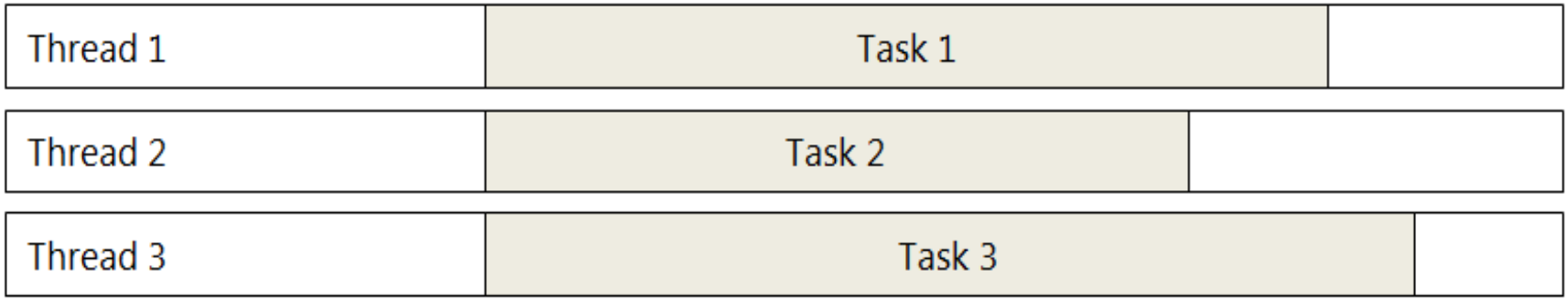
- Az egyre bonyolultabb kliens oldali (JavaScript) logikát egyetlen szálon futtatja a böngésző.
- Régebbi lehetőségek
  - aktív vezérlő (applet, Flash, Silverlight, ActiveX)
- A régebbi megoldások nehézségei
  - Nehéz telepíteni.
  - Nem fut minden böngészőben.
  - Más technológiával kell fejleszteni.
  - Biztonsági és stabilitási kockázatot jelent.
  - Vagy akár már nem is támogatják a böngészők

Végrehajtás egy szálon (szinkron)



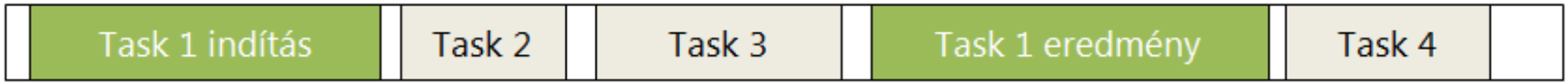
- Előny
  - ismert a sorrend,
  - Task 1 kimenete lehet Task 2 bemenete

Végrehajtás több szálon



- Egymástól független végrehajtás.
- A szálak közötti kommunikáció és koordináció nehéz.
- Több CPU mag esetén lehet konkurens, egy magos esetben időosztásos.
- JavaScript egy szálon fut.

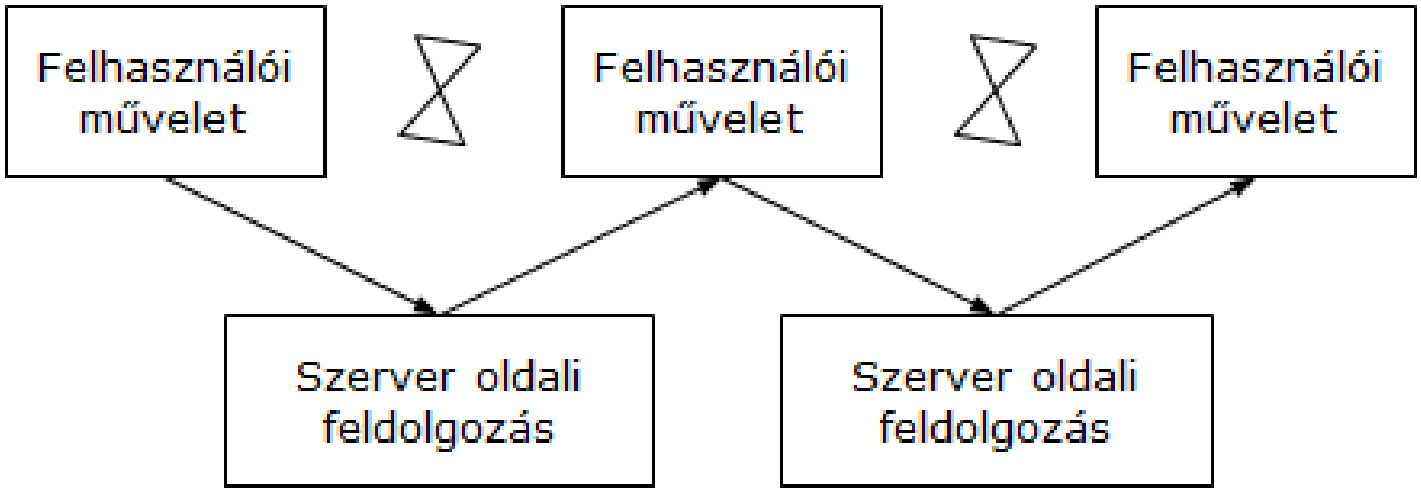
Azinkron végrehajtás egy szálon



- Átlapolódó végrehajtás
  - > többmagos rendszerben is.
- Egy időben biztosan csak egyetlen folyamat fut.
- Akkor célszerű, ha blokkoló műveletet hajtunk végre (pl. hálózati letöltés).
- Az eredmény feldolgozásának sem szabad blokkolónak lennie.

Szinkron kommunikáció

- a teljes oldal újratöltése lassú.
  - Rossz felhasználói élmény minden kattintás után.
  - Kizökken a felhasználó a folyamatból, mert villog az oldal
  - Folyamatos interaktivitás kellene.

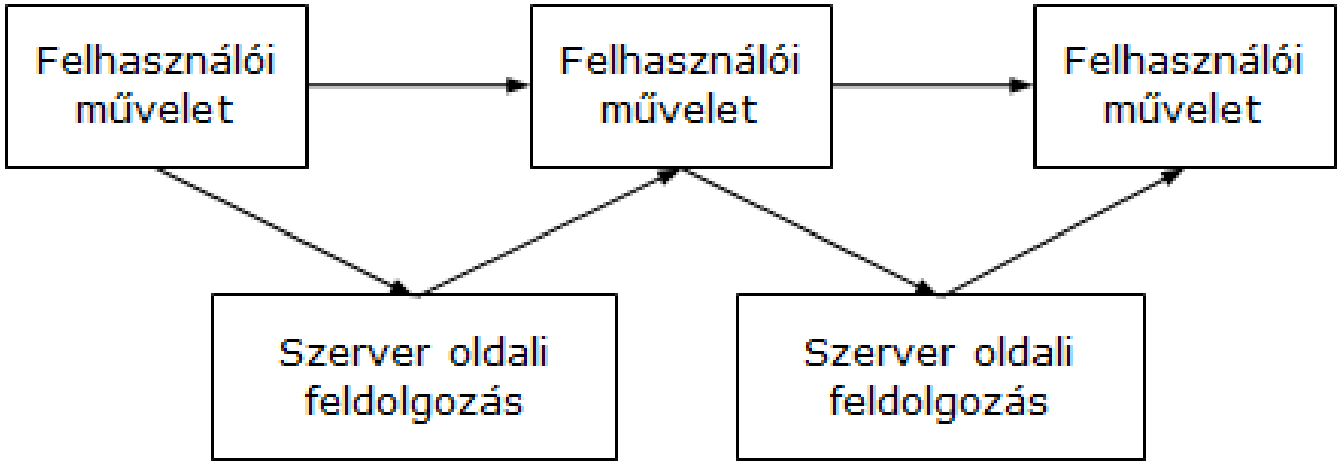


Szinkron kommunikáció

- A kliens és szerver oldali műveletek szinkronban vannak
- Sávszélesség (bandwidth)
  - Sok adat megy át a hálózaton, mindig az oldal teljes kódja.
- Szerver terhelés (server load)
  - Fel kell dolgozni a kérésben érkező minden adatot.
  - Mindig elő kell állítani az oldal teljes kódját, még ha csak kevés is változik.
- Felhasználói élmény (user experience, UX)
  - Blokkolt felhasználói felület.
  - Várakozás (click → wait → refresh).
  - Teljes oldal frissül → villanás, scroll context változás.

Aszinkron végrehajtás

- A kliens és a szerver tevékenysége nincs egymáshoz szinkronizálva.
  - Kattint a felhasználó → szervernek elmegy a kérés de a felhasználói felület használható marad.



Asynchronous JavaScript and XML

- A kliens először a teljes oldalt kéri le, később csak a változtatásokat.
- Az oldal ebben az esetben az URL nem tudja egyértelműen azonosítani.
  - Oldal = URL + DOM változások

AJAX előnyei

- Jobb felhasználói élmény (elsődleges cél)
  - > Gyorsabb.
  - > Nincs felhasználói felület blokkolás.
  - > Nem frissül a teljes oldal
    - a munkaállapot (scroll context) megmarad.
- Kevesebb adat megy át a hálózaton (csak változások)
  - > kisebb sávszélességet igényel.
- Kevesebb feldolgozás kell szerver oldalon
  - > csökkentett szerver terhelés.

AJAX hátrányai

- Kereső motorok támogatása korlátozott.
  - Tudják-e értelmezni és futtatni a JavaScriptet,
- Permalinkek támogatása nehéz.
  - Oldal = URL + változások
- Böngésző funkciók: history, back, forward.
  - A böngésző nem menti automatikusan a változásokat.
- Forgalomszámlálás, méretezés, tesztelés.
  - Mit számoljunk?
- Felhasználói szokások változnak, új dizájn elvárások.
  - Hogyan jelezzük, hogy mi történik? Értik a felhasználók?
- Komplex fejlesztői feladat.
  - Sok callback → hibakeresés nehezebb.

Technológiai megvalósítás

- Négy technológia szükséges a működéséhez:
  - XMLHttpRequest (XHR)
  - JavaScript
  - DHTML + DOM
  - Plain Old XML (POX) vagy JSON
- jQuery függvények
  - $.ajax: Ez minden tud, csak sok a paramétere
  - $.load, $.get, $.post, $.script, $.json

XMLHttpRequest (XHR) objektum

- Eredetileg a Microsoft találta ki az Outlook Web Accesshez és implementálta az IE5-ben ActiveX objektumként.
  - Később a többi böngésző natív JavaScript objektumként valósította meg (azonos metódusok, de más példányosítás).
  - IE7-től kezdve már natív objektumként is elérhető.
- Lényegében egy mini böngésző (HTTP kérést küld és HTTP választ vár).
  - Aszinkron végrehajtás (nem blokkolja a szálat; lehet szinkron is).
  - Callback függvényben lehet feldolgozni a választ.
  - Szerverplatform független.

XHR inicializálása

```
// Inicializálás
if( window.XMLHttpRequest )
{
  // IE7, Mozilla, Safari, Firefox: natív objektum.
  var xhr = new XMLHttpRequest()
}
else
{
  if( window.ActiveXObject )
  {
    // ActiveX IE5.x és IE6 esetén
    var xhr = new ActiveXObject( "Microsoft.XMLHTTP" );
  }
}
```

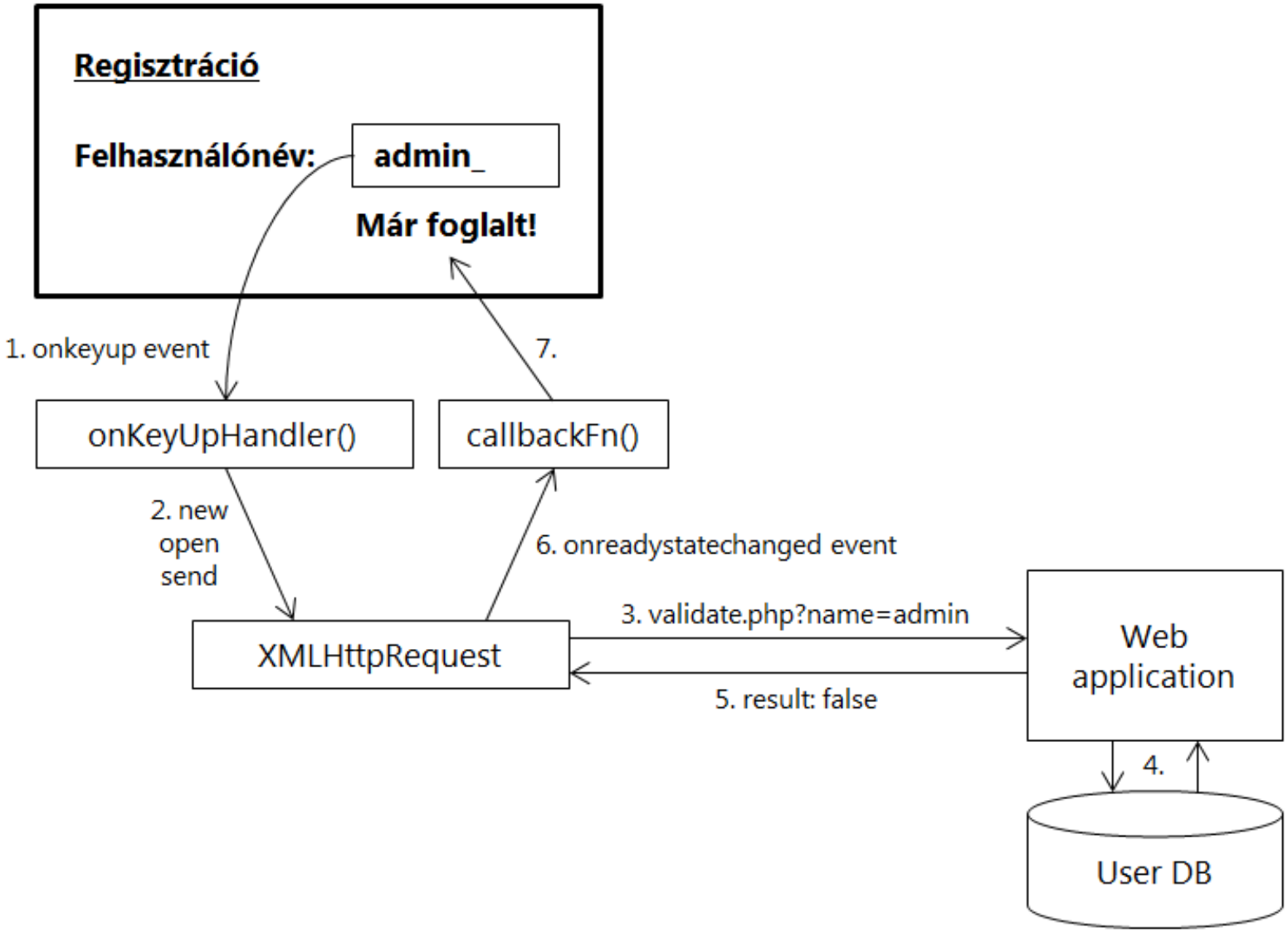
- Szerencsére a jQuery ezt elfedi előlünk és IE6-ot már senki sem támogat.

XHR kérés / válasz

```
// A kérés elküldése
xhr.open( "GET", strUrl, true );  // true = aszinkron
xhr.onreadystatechange = onStateChanged;
xhr.send( null );

// A válasz feldolgozása
function onStateChanged() {
   if( xhr.readyState == 4 ) {    // READYSTATE_COMPLETE
      if( xhr.status == 200 ) {    // HTTP_OK
         // Az xhr.responseText tulajdonság feldolgozása.
      }
      else {
         alert( "Hiba történt, a hibakód: " + xhr.status ); }
  }
}
```

Mire és hogyan használjuk



DHTML és a DOM

- A szervertől érkező válasz alapján a felhasználói felület frissítésére.
  - DHTML = DOM + JS + CSS
- Eltérések az egyes böngészőkben.
  - HTML elemek elkérése.
  - Feliratkozás eseményekre.
  - Eltérő tulajdonságok, metódusok.

XML / JSON

- Az átküldött adatstruktúrákat sorosítani kell.
- Az XML nehézkes
  - Redundáns, sok adat megy át (verbose).
- JSON
  - A szintaktika az objektumok leírásához használt szintaktikát követi.
  - Egyszerű adatcserére született, nem markup nyelv, a JavaScript object literal subsetje.
  - Beépített támogatás a sorosításra és a biztonságos parszolásra
    - **JSON.parse()** és **JSON.stringify()**
    - **eval()**-lal nem szabad feldolgozni!

JSON használata

```
var gj = {
  "nev": "Gipsz J",
  "kor": 35,
  "szul": {
    "hely": "Cegléd",
    "ev": 1978
  }
};
var json = JSON.stringify( gj );
alert( json );
// {"nev":"Gipsz J","kor":35,"szul":{"hely":"Cegléd","ev":1978}}
var obj = JSON.parse( json );
alert( obj.nev );
```

Implementációs nehézségek

- RFC 2616 (HTTP 1.1) Section 8.1.4 Practical Considerations (1999.)
- Max. 2 connection / hostname (proxy).
- A modern böngészők eltérnek ettől.
  - IE 8-9, Opera, Chrome : 6 db
  - IE 10: 8 db
  - Firefox: 6 db
    - network.http.max-persistent-connections-per-server
- Kérések és válaszok veszhetnek el
  - manuális queuing.
- **Same-origin policy.**
  - Csak oda lehet visszahívni, ahonnan az oldal letöltődött.
  - Scheme + host + port (RFC 6454 The Web Origin Concept)

Implementációs nehézségek

- Átirányítás a válaszban.
  - A kliensnek automatikusan követnie kell → nincs is rá flag vagy callback.
- Lejárt a cookie.
  - Lejár a session cookie → megszűnik a session a szerveren, de nem tudja értesíteni a klienst.
  - Lejárt az authentication cookie → átirányítás a bejelentkezés oldalra, ami HTML választ küld.
- Hiba a szerver oldalon.
  - pl. kezeletlen kivétel, 5xx szerver oldali hiba, túl nagy méretű kérés, timeout.
  - Szerver oldali általános hibakezelő átirányít egy HTML hibaoldalra.
  - Érvénytelen XML vagy JSON tartalom, a kliens nem tudja feldolgozni.
- Sok kis kérés összességében nagy forgalmat generálhat.

Progress Event

```
var progressBar = document.getElementById("p");
var client = new XMLHttpRequest();
client.open("GET", "magical-unicorns")

client.onprogress = function(pe) {
        if(pe.lengthComputable) {
                progressBar.max = pe.total;
                progressBar.value = pe.loaded;
        }
}
client.onloadend = function(pe) {
        progressBar.value = pe.loaded;
}
client.send();
```

Cross-domain hívások

- Same-origin policy
  - Csak oda hívhat vissza a kliens ahonnan az oldal letöltődött.
  - Biztonság miatt fontos
- Pedig szükség lenne cross-domain hívásokra
  - pl. webszolgáltatások hívása más oldalról

Cross-Origin Resource Sharing (CORS)

- Szabványos megoldás, beépül az XMLHttpRequest Level 2-be (XHR2).
  - A kliens **Origin** fejlécben elküldi a kérő oldal címét:

    ```
    Origin: http://example.com
    ```

  - A szerver ez alapján eldöntheti, hogy kiszolgálja-e a kérést. Ha igen, akkor a válaszban **Access-Control-Allow-Origin** fejléccel jelzi:

    ```
    Access-Control-Allow-Origin: http://example.com

    Access-Control-Max-Age: 2520

    Access-Control-Allow-Methods: PUT, DELETE
    ```

  - A szerver dönti el, hogy engedélyezi a kérést
    - másik elnevezés: **HTTP access control**.

Preflight request

- Előzetes OPTIONS kérés a szerverhez az adott verb támogatott-e
  - mielőtt a valódi kérés mellékhatást okozna
  - safe és idempotent metódusok

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

- A szerver a válaszban megmondja, hogy a kliens mit tehet:

```
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
```

- A kliens gyorsítótárazza (cache) a választ.

Credential kezelés

- Credential (cookie, HTTP authentication) alapból nem megy át
  - **XMLHttpRequest.withCredentials = true** beállítással engedélyezhető.
- Ha ez a tulajdonság létezik, akkor a kliens CORS-képes (capability detection).

JSON with Padding (JSONP)

- Régebbi böngészőkkel is működik.
- A JSONP azt használja ki, hogy a <script> tag segítségével lehetőségünk van más forrásból érkezett kódot is futtatni.
- A válasz (a betöltött JavaScript tartalom) egy, a hívó oldal által előre definiált függvénynév, paraméterekkel, amelyek a kért JSON adatokat tartalmazzák.
- Amikor a script lefut, a függvény meghívódik a JSON adattal, ilyen formán megengedve a hívó oldalnak az adatok feldolgozását.

JSONP

- Trükk: egyes HTML elemekre nem vonatkozik a same-origin policy
  - pl. **img**, **script**, **link**, **iframe**.
- JSON esetén a visszatérési érték egy adatstruktúra:

```
http://example.com/getData

{ "name": "Jack the Ripper", "year": 1888 }
```

- JSONP esetén a távoli szolgáltatás ezt az adatstruktúrát egy függvényhívásba ágyazva adja vissza.
  - Át kell adnunk a callback függvény nevét:

```
http://example.com/getData?callback=myCallbackFn

myCallbackFn( { "name": "Jack the Ripper", "year": 1888 } );
```

JSONP válasz

- JSONP esetén a válasz tartalmazza
  - adatot JSON formátumban,
  - amit becsomagol a szerver egy függvényhívásba,
  - mellyel a kliens által meghatározott callback függvénynek adja át a kért adatokat.

Lépések

1. Callback metódus, ami paraméterként megkapja a kért adatot.
2. JavaScriptből beszúrunk az oldalra egy **<script>** elemet.
3. A **<script>** elem **src** attribútumát beállítjuk a JSONP URL-re, paraméterként átadva a callback függvény nevét → a böngésző elindítja a letöltést.

- Előny: működik minden böngészőből.
- Hátrányok
  - A szervernek támogatnia kell.
  - Csak GET kérés, POST nem.
  - Nincs olyan gazdag hibakezelés, mint XHR-nél.
  - Biztonság kérdése: XSS.
    - **application/javascript** MIME type
    - tetszőleges JavaScript injektálható az oldalba

JSONP és a jQuery

- jQuery támogatja: **dataType="jsonp"** hatására automatikusan hozzáadásra kerül egy **"?callback=randomFn"** paraméter.
- **$.ajax** paraméterezése:
  - **jsonp**: a paraméter neve (ne **callback** legyen).
  - **jsonpCallback**: a callback függvény neve.

DEMO

jQuery AJAX JSONP-vel

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék