WEB05-jQuery, Ajax, jQuery validate, websocket

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**
- **könnyű kiterjeszthetőség, számos jQuery pluginnel**
  A jQuery egyik legfontosabb eseménye a `document ready` esemény.
- Akkor fut le, amikor a document **letöltődött**

⚠️ Az előtt mielőtt a képeket és külső hivatkozásokat elkezdené tölteni a böngésző.

Selectorok

`$("*")` - Minden elem
`$("#staff")` - A staf ID-jú elem. Amit először talál meg, ha több ugyanolyan ID-jű lenne.
`$(".meta")` - Adott osztállyal rendelkező elemek
`$("img")` - Adott tag

Ezután használhatóak összetett (akár hierarchikus) selectorok is, mint CSS-ben.

Validáció

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.

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

További lehetséges validate elemek:

`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.

$.ajax({...})

A szinkron kommunikáció előnytelen számunka, nem a legjobb a felhasználói élmlny sem, villan, ugrál, stb...

Megoldás: aszinkron kommunikáció

- 👍 Jobb felhasználói élmény
- 👍 Gyorsabb
- 👍 Nincs felhasználói felület bokkolás
- 👍 Nem frissül a teljes oldal
- 👍 Megmarad a munkaállapot
- 👍 Kevesebb adat a hálózaton
- 👍 Kevesebb szerveroldali feldolgozás

Vannak azért hátrányai is:

- 👎 A böngésző history funkciók támogatása nehézkes
- 👎 Forgalomszámlálás, méretezés, tesztelés nehéz
- 👎 Felhasználói szokások változnak, új dizájn elvárások.
- 👎 Komplex fejlesztői feladat

Négy technológia szükséges a működéséhez:

- XMLHttpRequest (XHR)
  - Lényegében egy mini-böngésző, aszinkron végrehajtással. A választ a callback függvényben lehet feldolgozni
- JavaScript
- DHTML + DOM
  - DOM + JS + CSS
  - A szervertől érkező válasz alapján a felhasználói felület frissítésére
- XML vagy JSON
  - Átküldött adatstruktúra sorosítására alkalmas
  - Az XML redundáns, nehézkes
  - A JSON viszont egyszerű adatcserére született, sorosítás jól támogatott.

jQuery függvények:
`$.ajax`: Ez mindent tud, csak sok a paramétere
`$.load, $.get, $.post, $.script, $.json`

```
// A kérés elküldése

var xhr = new XMLHttpRequest()
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 ); }
        }
}
```

Same-origin policy: Csak oda lehet visszaívni, ahonnan az oldal letöltődött!

CORS:

Cross-Origin Resource Sharing

Tartozik hozzá egy Preflight request, amivel elkéri a szervertől az Access-Controlhoz tartozó header-eket a kliens. A szerver megmondja, mit tehet a kliens. Ehhez süti alapból nem megy át.

A kliens Az *Origin* mezőben elküldi a kérő oldal címét.

Egyes HTML elemekre nem vonatkozik a same-origin policy, mint például az `img, script, lnk, iframe`

Nehézségek:

- Átirányítás a válaszban:
  - A kliensnek követnie kell.
- Lejár 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.
- Szerveroldali hiba
  - 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.
- A sok kérés együtt nagy forgalmat generálhat.

Adott intervallumonként pollozhatunk a szervertől ajax-al, de nem azonnal jelennek meg az adatok, és sok a felesleges kommunikáció. Megoldás lehet a **Long-polling**, ahol a kline sdirekt sokáig nyitva tartja a kapcsolatot. Ha van változás, akkor vissza tudja küldeni a szerver-

👍 Egyszerű megvalósítás
👍 Működik minden böngészőben
👍 Azonnal értesül a kliens
👎 Bonyolultabb szerveroldali implementáció
👎 Jobban terheli a szerveroldalt, mert foglalni kell a kapcsolathoz tartozó erőforrásokat.

Websocket

Full-duplex, kétirányú TCP kommunikáció egyetlen socketen keresztül.

- HTTP-től független TCP kommunikáció.
- ws:// és wss:// URI séma.
- A kliens egy `Connection:Upgrade` fejléccel kéri a protokoll váltást (handshake).
- Bármilyen alkalmazásban használható.
- 80 és 443 portokat használ, nincs tűzfal probléma.
- Nem bájt, hanem üzenetfolyam

👍 Nincsenek HTTP fejlécekkel járó overheadek, gyorsabb
👎 A szabvány sok változáson ment keresztül, az RFC-nek megfelelőt csak azújabb böngészők támogatják.
👎 Régi webszerverek nem támogatják

```
//Használat példa
var socket = new WebSocket('ws://localhost:8080/');
socket.onopen = function () {
        console.log('Connected!');
};
```

```
socket.onmessage = function (event) {
        console.log('Received data: ' + event.data);
        socket.close();
};

socket.onclose = function () {
        console.log('Lost connection!');
};

socket.onerror = function () {
        console.log('Error!');
};

socket.send('hello, world!');
```