5. Laboratóriumi gyakorlat

JavaScript a böngészőben gyakorlat 2. (űrlapok és DOM)

A gyakorlat célja: Űrlapok kezelése, űrlapok ellenőrzése, DOM módosítás. A szükséges segéd fájlok anyaga a `05_js_form_lab.zip` és `js_eloads.zip` állományokban tölthetőek le a Moodle honlapról. A hivatkozott példaprogramok, amennyiben nincsenek ebben a labor fájlban, az előadás példái közt találhatóak, a `form, formteszt,domtree` könyvtárban.

Tartalom

1. Tananyagok....1
2. Regisztrációs űrlap....1
3. Személyes adatok....2
4. Jelölőnégyzet....4
5. Szöveg....4
6. Színek....5
7. TAMOP feladatok....5
8. A beépített űrlap ellenőrző interfész (*Form data validation API*)....5
9. DOM fa gyakorló feladatok....5

1. Tananyagok

a) Ismételjük át a HTML űrlapok megadásának szintaxisát: ELTE tananyag, példák az előadás anyagában.

b) TAMOP tananyag, űrlapok kezelése JavaScript-ből .

c) Nézzük át az előadáson bemutatott példákat az űrlap kezelése és ellenőrzése vonatkozásában előadás bemutató pdf és példák (`js_eloadas`: `form` és `fteszt` könyvtárak).

d) Mozilla anyagok:
https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation ,
https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation#Validating_forms_using_JavaScript .

e) javascript.info anyagok: Forms, controls

f) Opcionális. A HTML5 nyelv több új űrlap típust vezetett be. Olvassuk át otthon Pilgrim könyvéből a „Form of Madness" c. fejezetet (http://diveintohtml5.info/forms.html).

g) A DOM műveletekhez szükséges tananyagok megtalálhatóak a `07_JS_DOM.pdf` előadás utolsó diáján. előadás

2. Regisztrációs űrlap

Készítsünk regisztrációt elindító HTML űrlapot, tartalma:

- e-mail cím,
- jelszó kétszer,
- elküldő gomb, törlő gomb.

Ennél a feladatnál ***mellőzzük*** a HTML5 új input elem tulajdonságokat, mint pl. a `pattern` .

Példa HTML fájlok az előadás példa könyvtáraiban: `form/form_input.html` (elavult eseménykezelő beállítás), `form/form_input1.html` (DOM3 szerinti beállítás: ezt használjuk!), egy email reguláris kifejezést a `formteszt/form_e.html` példában találunk. Ez utóbbi tartalmaz egy megoldást a hibák CSS kezelésére a `submit` eseménykezelőben. A `HTMLElement` típusú objektumok interfészében a `nextElementSibling` tulajdonság az elem következő testvérére mutat (lásd forráskód `setError()` függvénye).

1. A `submit` eseménykor ellenőrizzük JavaScript-el:
   - ki van-e töltve mind a három mező?
   - e-mail cím helyes?
   - a két jelszó megegyezik-e?

2. Írjunk egy olyan változatot is, ahol a mező kitöltésének befejezésekor (`blur` esemény) ellenőrizzük a mezőket egyenként. Az ellenőrzések eredményét tároljuk egy globális tömbben, és csak akkor küldjük el az űrlapot, ha minden ellenőrzés jól futott le előzőleg.
3. Írjunk egy olyan változatot is, ahol minden egyes billentyű felemelés (`keyup`) eseményre ellenőrzünk, és a mezők melletti kis szöveg CSS elemeit változtatjuk meg pirosra addig, amíg helytelen a mező: ha kitöltöttük a mezőt helyesen, a kis feliratok legyenek zöldek.
4. A `form_e.html/form_e.js` példát írjuk át úgy, hogy a `chekForm()` függvény *clojure* legyen, és a szkript elején található két globális változó (`form` és `error` legyen a *clojure*-be zárva).

3. Személyes adatok

Készítsünk egy HTML4 űrlapot, amelyik regisztráció után megoldja a személyes adatok bevitelét. Használjuk a `fieldset` , `legend` és `label` elemeket a megíráskor. A mezők tartalma:

a) vezetéknév, személynév
b) életkor
c) megye – `select` elemmel (3-4 megyét írjunk be az opció listába)
d) postai irányítószám
e) telefon
f) nem: férfi/nő – rádió gombokkal
g) Saját használatú gépkocsi adatai:
   a. jelölőnégyzet: „Van gépkocsim”,
   b. szöveges mező: Gépkocsi rendszáma.
h) elfogadom a honlap használatának feltételeit – jelölőnégyzet

1. Ellenőrizzük, hogy minden mező megérkezik-e a szerverhez, irányítsuk az űrlap `action` tulajdonságát a `https://ms.sapientia.ro/~lszabo/internet/action.php` címre.
2. Írjuk meg a mezőket ellenőrző, `submit` eseményre aktivált szkriptet, ami a következőket ellenőrzi:

- személynév-vezetéknév mező legalább 2 szó, de lehet 3 is (pl: `Próba Dezső` és `Próba Dezső Péter` alakokat fogad el), ha magyar szöveget is akarunk illeszteni, készítsünk olyan halmazt, amelyik lefed minden karaktert, a következő módszerrel:
  `/[a-zA-ZáÁéÉíÍóÓöÖőŐúÚüÜűŰ]/`
  az `a-z` implicit csak az angol ABC-t tartalmazza;
- életkor: szám 18-99 között,
- megye ki van választva, és nem az első opció van kiválasztva (Megye),
- postai irányítószám megegyezik a romániai formátummal (6 számjegy),
- telefonszám: romániai telefonszám,
- ha a gépkocsi be van jelölve, akkor román autórendszám tábla
- elfogadom a honlap használatának feltételeit: ki van választva

A hibák jelenjenek meg az űrlap fölött egy paragrafusban piros színnel (mint pl. a `form_text_e.html` példában, vagy hasonló megoldással).

3. Írjuk át az űrlapot HTML5 űrlapra (lásd előadás utolsó része, HTML5 input tulajdonságok). A kötelező mezőknél használjuk a `required` tulajdonságot, tegyünk `placeholder` tulajdonságot mindenhova, ahova lehet. Az első mezőre tegyünk `autofocus` tulajdonságot. Az ellenőrzéshez tegyünk `pattern` tulajdonságokat azokra a mezőkre ahol van értelme. Állítsuk be a `tabindex` tulajdonságot a mezőkön való ugrálás biztosításához. Az alábbi `input` mezőket használjuk:
   a) vezetéknév, személynév : `text` + `pattern` tulajdonság
   b) életkor : `number` típusú input elem (állítsuk be a `min` , `max`, `step` tulajdonságokat, lásd `form/html5_number.html` példa)
   c) megye – `select` elemmel (3-4 megyét írjunk be az opció listába)
   d) postai irányítószám `input` + `pattern`
   e) telefon `tel` + `pattern`
   f) nem: férfi/nő – rádió gombokkal
   g) Saját használatú gépkocsi adatai:
      a. jelölőnégyzet: „Van gépkocsim”,
      b. szöveges mező legyen írható csak akkor, ha a négyzet be van jelölve: Gépkocsi rendszáma `text` + `pattern`
   h) elfogadom a honlap használatának feltételeit – jelölőnégyzet

HTML5 input elemek: Mozilla: HTML input elem. A `submit` eseményre a mezők tesztelésénél használhatjuk az előző pontban megírt szkriptet, és amennyiben a böngésző ismeri a HTML5 HTML5 constraint validation API -t, akkor írhatunk egy külön szkript ágat ennek felhasználásával, példa az előadás anyagában és itt a Mozillánál.

4. Tanulmányozzuk a `select` és `options` elemeket (előadáson bemutatott példák, ELTE anyag). A `megyek.js` fájlt felhasználva oldjuk meg, hogy a megye mező `option` elemeit dinamikusan állítsuk elő az oldal betöltődés után. Tehát az eredeti HTML-ből vegyük ki a `select` elem `option` elemeit teljesen, és állítsuk őket elő a `window.onload` eseményre, felhasználva a `megyek` objektumot. A kulcsok legyenek az `option` elemek `value` tulajdonságai, míg az értékek a

megjelenő HTML opciók szövege.

Írjuk meg az opciókat előállító függvényt két módszerrel:

- HTML-t állítunk elő és beszúrjuk a `select` elem `innerHTML` tulajdonságába (`form/select_js_add.html` példa),
- a DOM `document.createElement()` és az elemek `appendChild()` függvényeit használva (`DOMtree/select_js_add_dom.html` példa). Ezt a feladatot a 9. , DOM fa gyakorló feladatokban oldjuk meg (3. pont).

A feladatot elvégezhetjük egy külön HTML-ben, amelyben egy űrlapnak csak egy `select` eleme van, kiinduló HTML: `megyek.html`.

5. Oldjuk meg, hogy a Gépkocsi rendszáma mező csak akkor jelenjen meg, ha bekattintottuk a „Van gépkocsim” négyzetet.

4. Jelölőnégyzet

Módosítsuk a `radio_js.html` példát a következőképpen: tegyünk az űrlapra egy jelölőnégyzet elemet „Házas” felirattal. Oldjuk meg:

A leánykori név mezőt csak akkor kelljen kitölteni, ha a Házas négyzet és a Nő rádió gomb van kiválasztva. Ha ez a feltétel nem teljesül, akkor a leánykori név mező egyáltalán ne jelenjen meg.

5. Szöveg

Írjunk egy `textarea` mezőbe egy hosszabb szöveget. Egy *Statisztika* nevű gomb kiválasztásakor írjuk ki a szöveg mellé jobb oldalra az alábbiakat:

- karakterek száma,
- sorok száma,
- szavak száma (a szövegből vegyük ki a punktuációs karaktereket, pl. a `/[-.,:;'"„”!()\[{}\]]/g` reguláris kifejezést használva, amit tovább bővíthetünk más karakterekkel ha szükséges),
- leggyakrabban használt szavak és minden szó után zárójelben a számossága: pl.: alma(3), piros(5), stb.
  Mivel tömböket tudunk szám szerint rendezni, egy megoldás, ha kialakítunk egy olyan tömböt amelynek elemei kis 2 elemű tömbök: [szó, számosság] és ezt rendezzük. Más megoldás is használható. A szavakra bontásnál használjuk a `String` osztály `split()` függvényét reguláris kifejezéssel, de előtte a szöveget alakítsuk kisbetűsre.
- tegyünk a lista végére egy jelölőnégyzetet, és azt kiválasztva módosítsuk az előző pontot úgy, hogy csak a nem stop szavak jelenjenek meg a listában (`english.stop` , `hungarian.stop`).
- ugyanitt egy szövegmezővel szabályozzuk, hogy hány szó jelenjen meg.

Példák az előadás `form` könyvtárában `textarea_browser.html` és `textarea_js.html`. Kiinduló állományok: `text_stat.html` , `stop_hu.js` . Mivel a kód hosszabb, írjuk egy külön állományba: `text_stat.js`. Működés közben egy hasonló képernyőt kell látnunk mint a `text_stat.png` és `text_stat_stop_szo_nelkul.png` képeken látható.

Ha működik a számítás a *Statisztika* gombra, akkor a számító függvényt áttehetjük a `textarea` elem `keyup` eseményére, így írás közben azonnal látjuk a statisztikát.

6. Színek

A `select2.html` fájlból kiindulva oldjuk meg a következőket:

Ha kiválasztunk egy színt az első `select` elemből (pl. Piros) akkor a második `select` elemben jelenjen meg a piros színnek megfelelő lista a változatok objektumból: Crimson, IndianRed, LightCoral.

Ha ez után kiválasztunk egy színt a második `select` elemből, a `szin` nevű `input` elem vegye fel ennek a színnek az értékét, pl. Crimson esetében: `"#DC143C"` .

7. TAMOP feladatok

További feladatok találhatóak a TAMOP tananyag 9. rész végén. Oldjuk meg őket.

8. A beépített űrlap ellenőrző interfész (Form data validation API)

Tanulmányozzuk a modern böngészők beépített űrlap ellenőrző interfészét (*Form data validation API*): https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation és használjuk fel az alábbi feladatban.

A `html5_tel_validate.html/.js` példából kiindulva oldjuk meg:

1. Oldjuk meg, hogy a telefonszámot 999-999-999 és 999 999 999 alakban is be lehessen vinni (*Validating against a regular expression* szakasz)
2. Vegyük le a `checkForm()` és `inputTel()` esemény kezelőket. Mi történik, ha levesszük a `novalidate` tulajdonságot az űrlapról?
3. Az `:invalid` és `:valid` pseudo szelektorokat használva tegyünk saját keretet és háttérszínt az input elemre helyes és hibás adat eseteire (*The required attribute* szakasz).
4. Tegyünk saját hibaüzenetet a beépített helyébe (*Customized error messages* szakasz), pl.: *„Telefonszámot írj be!”.*
5. Olvassuk át a *Constraint validation API properties fejezetet*, ahol megtalálhatóak az API elemei.
6. Olvassuk át a *Validating forms without a built-in API* fejezetet és a *How can I help the user to correct invalid data?* szakaszban ajánlott külső anyagokat.

9. DOM fa gyakorló feladatok

Tanulmányozzuk az előadáson bemutatott DOM szerkezetet kezelő interfészeket. Az előadás *Tartalom* vagy *Gyors linkek* diáján követni tudjuk a megfelelő függvény csoportokat, az utolsó diáján a tananyagokat.

A kiinduló fájlok az előadás **`domtree`** könyvtárában vannak.

1. A `scan.html/.js` fájlokból kiindulva, írjunk egy függvényt, amelyik a `body`-ból kiindulva megtalálja az olyan paragrafus elemeket, amelyben van `em` elem.

A feladatot két módon lehet megoldani:

   a) a `QuerySelectorAll()` függvénnyel
   b) a DOM fa végigjárásával **`p`** elemek keresésével, és azok gyerek elemei közt az **`em`**-ek keresésével.

2. A `tree.html/.js` fájlokból kiindulva, írjunk egy függvényt, amelyik megkeresi, hogy két HTML elem gyerek elemei közt vannak-e olyan elemek, amelyeknek a `textContent` tartalma a egyenlő (egyik elem legyen az első elem gyereke, a másik a másodiké, pl. a `#tarto` és `#tarto1` gyerek elemei közt van ilyen).

3. (3. feladat 4. pont más módszerrel)

A `megyek.html/.js` fájlokat felhasználva oldjuk meg, hogy a megye mező `option` elemeit dinamikusan állítsuk elő az oldal betöltődés után a DOM `document.createElement()` és az elemek `appendChild()` valamint `setAttribute()` függvényeit használva (`domtree/select_js_add_dom.html` példa).

A feladatot elvégezhetjük egy külön HTML-ben, amelyben egy űrlapnak csak egy `select` eleme van, kiinduló HTML: `megyek.html`.

4. Nézzük át az `elements.html/.js` hagyományos DOM módosító függvényeket használó példát. A függvények teszt hívásai a `window.onload` eseménykezelőben vannak.

Írjunk egy függvényt, amelyik DOM módosító függvényeket használ (pl. `replaceChild()`), és kicseréli egy elem N. és M. gyerek elemét. Ha nincs N. és M. gyerek, akkor jelzi hibával a konzolon és nem végez semmit. Egy DOM fa csomópontról másolatot (*clone*) a `cloneNode()` függvénnyel készítünk.

5. Tanulmányozzuk a `mlevel_li.html/.js` és `mlevel_li1.html/.js` példákat, amelyek listákat hoznak létre JavaScript objektumokból. Nézzük át az előadás diáiból az `Objekt` objektum felhasznált függvényeit.

Írjunk egy függvényt, amelyik a `mlevel_li.html/.js`- ben felépített listából DOM függvényekkel kiveszi a 3. szintben levő listákat (akkor is ha több van, ehhez készítsünk teszt esetet) és átteszi első szintű listává (a példában a *Harmadik ...* listát kell megtalálni, és áttenni az *Első, ..., Második, ...* listák szintjére).