# Web API-k összefoglalás és gyakorlás

A Web API-k olyan, a böngészők által nyújtott, szolgáltatásokat jelentenek amelyeket JavaScriptből érhetünk el, segítségükkel jelentősen funkciógazdagabbá tehetjük a webalkalmazásunkat. Pl.: A Fetch API segítségével HTTP kéréseket indíthatunk. Az egyes API-k érettsége és böngészőtámogatottsága jelentősen eltérhet, használat előtt érdemes ellenőrizni, hogy mely böngészők milyen verziótól támogatják az adott funkcionalitást (ez leginkább újonnan bevezetett funkciók esetén fontos).

Elérhető Web API-k listája: https://developer.mozilla.org/en-US/docs/Web/API

Az alapozó képzésen két Web API-val ismerkedtünk meg részletesebben, ezek a LocalStorage és a DOM API, ez a dokumentum ezekhez tartalmaz egy összefoglalót illetve gyakorló feladatokat.

Megjegyzés: a feladatokat lokális fejlesztői környezetben, vagy online editorban (pl.: StackBlitz (Static projectet hozzunk létre)) is elvégezhetjük.

# LocalStorage API

A LocalStorage segítségével kulcs-érték párokat tárolhatunk a böngészőben, a kulcs és az érték is mindig string típusú. Ezek az értékek megmaradnak akkor is, ha a felhasználó bezárja az oldalt, vagy a böngészőablakot, vagy akár újraindítja a gépét (az itt tárolt adatok alapvetően egészen addig nem törlődnek, amíg a felhasználó nem törli a böngészési előzményeit).

Egy lehetséges használati eset lehet, hogyha a felhasználóinknak azzal akarunk kedveskedni, hogy a nevükön köszöntjük őket látogatáskor. Ekkor egy input segítségével bekérhetjük a nevüket, amit aztán a LocalStorage-ba tárolunk, és újboli látogatás esetén felolvassuk az eltárolt nevet.

Fontos tudni, hogy az egyes weboldalak LocalStorage tárolói egymástól szeparáltak, tehát, ha A weboldal elmenti a felhasználó nevét, akkor ahhoz a B oldal nem fog hozzáférni.

Legfontosabb metódusok (https://developer.mozilla.org/en-US/docs/Web/API/Storage#methods):

- `getItem` - egy adott kulcshoz tárolt értéket adja vissza
- `setItem` - egy adott kulcshoz tárolhatunk el értéket
- `removeItem` - egy adott kulcsot, és a hozzá tárolt értéket törölhetjük
- `clear` - törli a LocalStorage teljes tartalmát

**Feladat:** Készítsünk egy weboldalt, amin egy input szerepel, ide írhatja a felhasználó a nevét, ezt egy gombbal elmentheti a LocalStorage-ba, egy másik gombbal pedig kitörölheti onnan. Oldal betöltéskor olvassuk ki az eltárolt nevet a LocalStorage-ból, és jelenítsük meg az értéket az inputban (ha volt eltárolt érték).

**Megoldás:** https://github.com/Webuni-JavaScript-courses/localstorage-example

# DOM API

A DOM API segítségével a DOM fához férhetünk hozzá JavaScriptből, ez az egyik legalapvetőbb Web API, ennek segítségével tudjuk dinamikussá tenni az oldalunkat, és pl.: egy gombhoz esemény kezelőt regisztrálni, amely kattintásra fut le.

Az alábbiakban átnézünk néhány alapvető DOM API használati esetet:

## DOMContentLoaded esemény

A `document` nevű változón keresztül tudjuk a DOM fát elérni. Gyakran különféle eseménykezelőket akarunk a DOM fa egyes elemeihez kötni (pl.: gombokhoz, inputokhoz, stb.), ezt viszont csak azután tehetjük meg, ha a böngésző már kirajzolta ezeket az elemeket (ellenkező esetben nem tudjuk mihez kötni ezeket az eseménykezelőket). A DOM fában levő elemek kirajzolásáról a `DOMContentLoaded` esemény értesít minket.

**Feladat:** Készítsünk egy HTML oldalt, amin egy gomb van, regisztráljunk be egy eseménykezelőt a gomb klikk eseményéhez. Először ne várjuk meg a DOMConentLoaded eseményt, nézzük meg mi történik így. Javítsuk a működést: iratkozzunk fel a DOMContentLoaded eseményre, és csak ennek megtörténte után regisztráljuk be a gombhoz tartozó eseménykezelőt.

**Megoldás:**

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <button id="button">Click me</button>
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document
   .getElementById('button')
   .addEventListener('click', () => console.log('clicked'));
});
```

Megjegyzés: Ha StackBlitzen dolgozunk, akkor nem kapunk hibát, ha a DOMContentLoaded esemény előtt regisztráljuk be a gombhoz tartozó eseménykezelőt, de ettől még az nem fog működni, csupán a StackBlitz elnyeli a hibaüzenetet.

# HTML elem elérése ID alapján

HTML elemet ID alapján többféle módon is el lehet érni, ezek azonban a leggyakrabban használatos módok:

- `getElementById`
- `querySelector` - ami egy CSS selectort vár, id alapú selector: #<id>

Az ID alapú elérést akkor szoktuk használni, ha egyetlen elemhez akarunk eseménykezelőt regisztrálni, az ID alapú elérés előnye, hogy az ID attribútum várhatóan nem fog változni, de az elem neve (pl.: div helyett p elemet használunk) és CSS osztálya gyakrabban változhat.

**Feladat:** Helyezzünk el egy HTML oldalon egy inputot, adjunk neki egy ID-t, majd a fenti módszerek egyikével regisztráljunk egy eseménykezelőt az `input` eseményhez.

**Megoldás:**

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <input id="my-input" />
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document
   .querySelector('#my-input') // vagy getElementById('my-input')
   .addEventListener('input', (event) => console.log(event));
});
```

Bármilyen HTML elemehez regisztrálunk bármilyen eseménykezelőt általánosan igaz: az eseménykezelő függvény paraméterül fog kapni egy objektumot (a példában ezt “event” néven vesszük át), amely az eseményről tartalmaz információkat. Az objektum szerkezete HTML elemenként és esemény típusonként eltérhet.

Az esemény leíró egy fontos paramétere a `target` attribútum, amely azt a HTML elemet tartalmazza, amelyhez az esemény tartozik.

# Inputok

A HTML input elemeken kiemelt jelentősége van, mert ezeken keresztül kérhetünk be adatokat a felhasználóinktól. Az input elemeknek több altípusa van aszerint, hogy milyen

adatokat tudunk bekérni használatukkal (pl.: szám vagy dátum), a teljes lista itt található: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input. Ugyan nem az input elemek típusai, de a select és textarea elemeket is használhatjuk adatbekérésre.

A fenti elemeknél általános szabály, hogy az adott elemben tárolt adatot a `value` attribútum segítségével olvashatjuk/írhatjuk. A `value` attribútum értéke mindig string típusú lesz input típustól függetlenül (ha a value property-n keresztül írjuk az elem értékét, akkor az átadott érték string-re lesz alakítva).

Az input elemekben tárolt adatok módosítását leggyakrabban az `input` és `change` eseményeken keresztül figyelhetjük. Az input esemény minden módosításnál elsül (pl.: minden billentyű leütésnél, amikor az inputba gépelünk), a `change` esemény pedig csak akkor, ha a felhasználó befejezte a bevitelt (pl.: kikattint az inputból, vagy az entert leüti)

**Feladat:** Helyezzünk el egy szöveges inputot egy HTML oldalon, és iratkozzunk fel az `input` és `change` eseményekre. Vizsgáljuk meg, hogy mikor hívod meg egyik ill. másik esemény. Az eseménykezelőben írjuk ki a console-ra az input aktuális tartalmát. Az inputnak JavaScriptből állítsuk be az alábbi kezdőértéket: “Webuni”

**Megoldás:**

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <input id="my-input" />
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 const input = document.querySelector('#my-input');

 input.value = 'Webuni';

 input.addEventListener('input', (e) =>
   console.log('input esemény', e.target.value)
 ); // Input aktuális értékének elérése 1.: e.target.value
 input.addEventListener('change', (e) =>
   console.log('change esemény', input.value)
 ); // Input aktuális értékének elérése 2.: input.value
});
```

# HTML elem tartalmának módosítása

Gyakori feladat, hogy a DOM fánkat JavaScriptből akarjuk módosítani. Pl.: dinamikus tartalmat akarunk megjeleníteni a felhasználónak (pl.: webszervertől lekért adat), ez az adat nem áll rendelkezésre a weboldal HTML tartalmának megírásakor, oldalbetöltés után kell lekérdeznünk a szervertől, emiatt nem tudjuk betenni ezt a tartalmat a HTML fájlba, JavaScriptből kell azt utólag beillesztenünk. Ilyen esetben megtehetjük azt, hogy a HTML fájlba egy üres elemet helyezünk el, ahova majd a tartalmat illesztjük, egy HTML elem tartalmát az alábbi két attribútum segítségével módosíthatjuk:

- `innerHTML`, ha HTML tartalmat akarunk megjeleníteni
- `innerText`, ha szöveges tartalmat akarunk megjeleníteni

**Feladat:** Helyezzünk el egy szám típusú inputot (`type=number`) egy HTML dokumentumban, ide tudja megadni a felhasználó, hogy hány gombot rajzoljunk ki. Helyezzünk el egy üres div elemet is az oldalon. Az inputba beírt számnak megfelelő számú gombot helyezzünk el az üres div-ben.

**Megoldás**:

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <input id="my-input" type="number" />
   <div id="buttons"></div>
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document.querySelector('#my-input').addEventListener('change', (e) => {
   const value = e.target.value;

   const div = document.getElementById('buttons');
   div.innerHTML = ''; // Visszaállítjuk a tartalmat, hogy az esetleges korábban
kirajzolt gombokat eltüntessük
   for (i = 0; i < value; i++) { // For ciklust nem csak tömbökkel használhatjuk, akkor
is jól jön, ha egy feladatot x-szer kell elvégezni
     div.innerHTML = div.innerHTML + `<button>${i}</button>`; // Minden iterációban
hozzáfűzünk a tartalomhoz
   }
 });
});
```

# HTML elemek elérése tag név alapján

Előfordul, hogy egyszerre nem egy HTML elemhez, hanem többhöz is szeretnénk eseménykezelőt regisztrálni (vagy más műveletet végezni rajtuk), ilyenkor nehézkes lenne ID alapján elérni az elmeket, mert akkor mindhez külön ID-t kell megadnunk, és egyesével elérni őket. A DOM API azonban lehetőséget ad egyszerre több elem lekérése is, ezt többféleképp is megtehetjük, de az egyik megoldás, ha a HTML elem neve alapján (pl.: input vagy div) kérjük le őket. Ilyen esetekben az alábbi függvényeket tudjuk használni:

- `getElementsByTagName`
- `querySelectorAll` - tag name alapú selector pl.: div

Mindkét függvény visszaadja az összes olyan HTML elemet, amelyre igaz volt a megadott keresési feltétel. A `querySelectorAll` használatának előnye, hogy a kapott tömbön aztán a megszokott módszerekkel végigiterálhatunk (pl.: `forEach` metódus).

**Feladat:** Helyezzünk le három gombot egy HTML dokumentumban, és JavaScriptből regisztráljunk minden gombhoz egy eseménykezelőt a klikk eseményre.

**Megoldás:**

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <button>1</button>
   <button>2</button>
   <button>3</button>
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document.querySelectorAll('button').forEach((button, index) => {
   button.addEventListener('click', () =>
     console.log(`Az ${index + 1}. gombot nyomtad meg`)
   );
 });
});
```

# HTML elemek elérése CSS osztály alapján

A tag name alapú kiválasztás sok eseteben nem elfogadható, mert az oldalon több példány lehet egy adott elemből, mint ahányat együttesen akarunk kezelni. Ilyen esetekben jó megoldás lehet, hogyha a megadott elemekre közös CSS osztályt helyezünk, és ez alapján kérjük le őket. Ehhez az alábbi metódusok használhatók:

- `getElementsByCssClassName`
- `querySelectorAll` - CSS osztály alapú selector pl.: .green

Mindkét függvény visszaadja az összes olyan HTML elemet, amelyre igaz volt a megadott keresési feltétel. A `querySelectorAll` használatának előnye, hogy a kapott tömbön aztán a megszokott módszerekkel végigiterálhatunk (pl.: `forEach` metódus).

**Feladat:** Helyezzünk el négy gombot egy HTML dokumentumban, és JavaScriptből regisztráljunk az első három gombhoz egy eseménykezelőt a klikk eseményre.

**Megoldás:**

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <button class="selected">1</button>
   <button class="selected">2</button>
   <button class="selected">3</button>
   <button>4</button>
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document.querySelectorAll('.selected').forEach((button, index) => {
   button.addEventListener('click', () =>
     console.log(`Az ${index + 1}. gombot nyomtad meg`)
   );
 });
});
```