**Horváth Gergő DHWIDH**

**WEB-programozás I. előadás beadandó magyarázat**

Ezt a beadandót egyedül készítettem, a következő github profilokat használva:

hor-ger

ger-hor123

Link a weboldalhoz:  
**http://majasmiujsag.nhely.hu**

Link a github repository-hoz:

**https://github.com/hor-ger/WEB-EA/tree/main**

**Kezdő oldal:** *index.html*

**Alapvető elemek:**

* <!DOCTYPE html>:  
   Ez az első sor deklarálja, hogy a dokumentum egy HTML5 típusú fájl, ami biztosítja, hogy a böngésző a modern szabványok szerint értelmezze a kódunkat.
* <html lang="hu">:  
  Ez a html elem határozza meg az oldal gyökérstruktúráját, és a lang="hu" jelzi, hogy az oldal magyar nyelvű.
* <head>:   
  Itt találhatóak az oldal metaadatai és hivatkozásai. Ez a rész felelős a látogatók által nem látható, de fontos információkért, mint a karakterkódolás, stíluslapok és JavaScript fájlok hozzácsatolása.
* CSS és JavaScript csatolás:

<link rel="stylesheet" href="index.css">:  
A stíluslapot (CSS fájlt) csatolja az oldalhoz. Ezzel testre szabhatjuk az oldal kinézete, például a színek, betűtípusok és elrendezések és sok más egyéb dolgot.   
<script type="text/javascript" src="index.js">:   
Ez a JavaScript fájl kapcsolódik az oldalhoz, amely az oldalak interaktivitást biztosítja.

**Az oldal felépítése(body):**

* Navigációs sáv:   
  <div class="navbar">:   
  Ez a rész tartalmazza az oldal menüjét, amely különböző aloldalakra navigálja a felhasználót. Minden menüpont egy <button> elembe ágyazott linkként működik (<a> tag). Például a "CRUD" gomb a crud.html fájlra irányít. Az "aktív" gombot a class="active" attribútum kiemeli.
* Fejléc (Header):  
  <header>: Ez egy, az oldal tetején található szekció, amely a főcímet tartalmazza: **"Web-programozás-1 Előadás Házi feladat"**. Ez egy <h1> elemben van elhelyezve, amely az oldal legnagyobb és legfigyelemfelkeltőbb címsora.
* Lábléc (Footer):   
  <footer>: Ez az oldal alján található szekció, amely információs célú szöveget jelenít meg. Például a készítőt azonosítja: **"Horváth Gergő DHWIDH"**, és egyszerű <p> (paragrafus) elemben van formázva.

**index.css:**

Ez a legfontosabb css file mert ez be van ágyazva az oldal minden lapjára. Az oldal ezen kívül rendelkezik még más css fileokkal is, hogy ne keveredjenek össze a különböző lapok formázásai. Nem minden oldalhoz használtam külön css file-t, mivel a vége fele észrevettem, hogy több minta kódban is használt a html-ben style tag-eket (<style> </style>).

**Általános stílusok:**

* body:

A teljes oldal hátterének színe rgb(170, 132, 57), ami egy barna tónusú árnyalat, és az oldal hangulatát határozza meg.

Ez az alapréteg az oldal minden elemére kiterjed.

**Fejléc (**header**):**

* A fejléc középre igazított szövege text-align: center, így minden tartalom pontosan a képernyő közepére kerül.
* A margin-top: 40px hozzáad egy kis helyet a fejléc teteje és az oldal széle között, ami szellősebbé teszi a megjelenést.
* Az align-self: center biztosítja, hogy a fejléc minden eszközön középen maradjon.

**Navigációs sáv (**.navbar**):**

* display: flex: A rugalmas dobozmodellel rendezi el a navigációs gombokat egy sorban.
* justify-content: center: Középre igazítja az összes navigációs elemet.
* background: rgb(170, 156, 127): A sáv háttérszíne egy világosbarna árnyalat, ami jól harmonizál az oldal többi részével.
* position: fixed és top: 0: A navigáció fixen marad az oldal tetején, görgetés közben is látható.
* border-radius: 5px: Az élek enyhén lekerekítettek, ami modern megjelenést ad.

**Gombok (**.navbar button**):**

* **Stílusos alapok**:

A gombok háttérszíne rgb(255, 200, 91) egy élénk sárga árnyalat, amely figyelemfelkeltő és barátságos.

A lekerekített élek (border-radius: 5px) és a vastag betűstílus (font-weight: bold) modern és professzionális megjelenést biztosítanak.

A nagyobb, kényelmes klikkelési területet a padding: 5px 25px adja.

* **Hover effektus**:

Amikor az egér a gomb fölé kerül, a háttérszín rgb(225, 175, 85)-re változik, ami vizuális visszajelzést ad a felhasználónak.

**Linkek (**a**):**

* A text-decoration: none eltávolítja az aláhúzást a linkekről, hogy tisztább legyen a megjelenés.
* A a:visited szabály biztosítja, hogy a linkek színe ne változzon meg a meglátogatott oldalak után, így az egységes stílus megmarad.

**Táblázatok és szekciók:**

* #tableview:

A táblázat szélessége width: 100%, így az oldal teljes szélességét kitölti.

A border-radius: 5px itt is szerepel a lekerekített szegélyek érdekében.

* section:

Ez a szekciók kinézetét szabályozza. A világos sárga háttérszín (rgb(255, 200, 91)) jól kiemeli a tartalmat.

A szekciók mérete fix: width: 50% és height: 300px, amelyek egyenletes megjelenést biztosítanak.

A fekete keret (border: 3px solid black) határozottan keretezi az elemeket, ami letisztult és rendezett kinézetet ad.

**Canvas és táblák:**

* #div1:

Egy kisebb, keretezett doboz, amely valószínűleg interaktív vagy statikus tartalom (pl. szöveg vagy ikon) megjelenítésére szolgálnak.

* #canvasf:

Egy vászonszerű elem, amelyen valószínűleg grafikák vagy diagramok jelennek meg, fekete kerettel hangsúlyozva (border: 1px solid black).

* .charttabla:

Ez a táblázatos megjelenítéshez tartozó stílus. Az display: flex a dobozokat egymás mellé helyezi, miközben a tartalom középre igazítva jelenik meg.

**CRUD html:**

**Keresés mező**

***<input type="text" id="searchInput" placeholder="Keresés...":***

Egy szövegmezőt jelenít meg, amibe beírt keresőkifejezés alapján lehet szűrni a táblázat tartalmát.

Az onkeyup="searchTable()" attribútum azt jelenti, hogy minden billentyűleütés után meghívja a searchTable() JavaScript függvényt.

Interaktív táblázat

***<table id="dataTable">:***

Egy táblázatot hoz létre, amiben megjeleníthetők adatok (pl. név, Neptun kód, osztályzat).

***Fejléc (<thead>):***

Tartalmazza az oszlopok neveit: "Név", "Neptun kód", "Osztályzat", és "Műveletek".

Az oszlopok fejlécei kattinthatók (onclick="sortTable(n)"), így a táblázat sorai rendezhetők az adott oszlop alapján.

***Törzs (<tbody>):***

Ide kerülnek dinamikusan az adatok sorai, amelyeket JavaScript segítségével töltenek be vagy módosítanak.

Adatbevitel mezők

***<input> mezők:***

Három mező, amelyekbe új adatok (Név, Neptun kód, Osztályzat) írhatók be.

***id="nameInput":*** Név bevitele.

***id="subjectInput":*** Neptun kód megadása.

***id="gradeInput":*** Osztályzat számmal történő megadása.

***Hozzáadás gomb (<button id="addButton">):***

Ez a gomb a felvitt adatokat hozzáadja a táblázathoz, amikor megnyomják.

**CRUD JavaScript:**

**1. Események kezelése:**

* DOMContentLoaded: Amikor az oldal teljesen betöltődik, hozzákapcsolja az addRow funkciót a "Hozzáadás" gombhoz, így a gomb megnyomásakor új sorokat lehet hozzáadni a táblázathoz.

**2. Sorok hozzáadása (**addRow**):**

* Bekéri az űrlapmezők értékeit (Név, Neptun kód, Osztályzat), ellenőrzi azok helyességét (például hossz vagy értéktartomány alapján), majd hozzáad egy új sort a táblázat törzséhez.
* A sorhoz automatikusan "Szerkesztés" és "Törlés" gombokat rendel, amelyekkel interaktívan módosítható vagy törölhető a sor.

**3. Bemenetek ellenőrzése (**validateInput**):**

* Ellenőrzi, hogy a mezők értékei megfelelnek-e az előírt szabályoknak (például karakterek száma, osztályzat értéke), és hibás bemenet esetén figyelmeztetést jelenít meg.

**4. Sorok szerkesztése (**editRow**):**

* Az "Szerkesztés" gomb megnyomásakor egy felugró ablakban lehet módosítani a sor adatait. A változtatások csak akkor menthetők, ha az új értékek is megfelelnek az ellenőrzési szabályoknak.

**5. Sorok törlése (**deleteRow**):**

* A "Törlés" gomb eltávolítja az adott sort a táblázatból.

**6. Táblázat keresése (**searchTable**):**

* A keresőmezőbe beírt szöveg alapján szűri a táblázat sorait, és csak azokat jeleníti meg, amelyek tartalmazzák a keresett kifejezést.

**7. Táblázat rendezése (**sortTable**):**

* Az oszlopok fejlécére kattintva a táblázat sorait rendezni lehet az adott oszlop értékei alapján (például név vagy osztályzat szerint), növekvő vagy csökkenő sorrendben.

**HTML5 html:**

**Általános felépítés**

1. **Konténer (**<div class="container">**)**:
   * Az oldal fő kerete, amely a táblázatot és annak különálló interaktív elemeit tartalmazza.
   * A konténer célja, hogy logikailag és vizuálisan egységes tartalmi blokkot hozzon létre.
2. **Táblázat (**<table id="tableview">**)**:
   * Egy táblázatba rendezett elrendezés, amely soronként (<tr>) és cellánként (<td class="section">) különböző funkciókat és technológiákat tartalmaz.

**Funkciók részletes bemutatása**

**Web Storage**

* **HTML elemek**:
  + <h3> **cím**: A funkció neve, "Web storage", amely egyértelműen jelzi a felhasználónak, hogy az adatkezeléssel kapcsolatos.
  + **Gomb (**<button onclick="clickCounter()">Kattints rám!**)**: Minden gombnyomás frissíti a tárolt kattintásszámot a sessionStorage API segítségével.
  + **Eredmény megjelenítése (**<p id="wsresult">**)**: Ebben az elemben jelenik meg, hogy a gombot hány alkalommal nyomta meg a felhasználó.
* **Működés**:
  + A gombnyomások számát a böngésző sessionStorage tárhelye tárolja, így a kattintásszám akkor is megmarad, ha az oldalt újratöltik.
  + Az eredményt dinamikusan frissíti a clickCounter() függvény.

**Web Worker**

* **HTML elemek**:
  + <p id="counter">: Egy szám, amelyet a Web Worker szál folyamatosan növel, ha aktív.
  + **Indítás és leállítás gombok**:
    - Start: A Web Worker elindításáért felelős.
    - Stop: Leállítja az aktuálisan futó szálat.
* **Működés**:
  + A JavaScript Web Worker API külön szálon futtat egy scriptet (általában aszinkron feladatokra használják, mint például számlálás, nagy számítások).
  + Az startWorker() függvény indítja a munkát, míg a stopWorker() leállítja.

**Helyzetmeghatározás**

* **HTML elemek**:
  + **Gomb (**Mutasd a helyzetem!**)**: A böngésző helyzetmeghatározási API-ját hívja meg.
  + **Eredmény megjelenítése (**<p id="location">**)**: Itt jelennek meg a felhasználó szélességi és hosszúsági koordinátái.
* **Működés**:
  + A getLocation() függvény megpróbálja lekérni a felhasználó aktuális földrajzi helyzetét a navigator.geolocation API segítségével.
  + Ha a böngésző nem támogatja a helymeghatározást, hibaüzenet jelenik meg.

**Drag and Drop**

* **HTML elemek**:
  + **Drop zóna (**<div id="div1"**)**: Olyan terület, ahová elemeket lehet húzni és "ejteni".
  + **Húzható kép (**<img id="img1"**)**: A felhasználó egérrel mozgathatja a képet, majd elhelyezheti a célterületen.
* **Működés**:
  + A dragstartHandler(event) függvény elindítja a húzást.
  + A dragoverHandler(event) engedélyezi, hogy a célterületre lehessen húzni az elemet.
  + A dropHandler(event) az elem elhelyezését kezeli a célterületen.

**Canvas (Grafikus vászon)**

* **HTML elemek**:
  + **Canvas (**<canvas id="canvasf">**)**: Egy 100x100 pixeles elem, amelyre rajzolni lehet JavaScript segítségével.
* **Működés**:
  + A Canvas API lehetőséget biztosít különböző grafikai elemek (például vonalak, formák) kirajzolására.

**HTML5 JavaScript:**

**Session Storage és Kattintásszámláló**

* DOMContentLoaded: Az oldal betöltésekor ellenőrzi, hogy van-e elmentett kattintásszám a sessionStorage-ban. Ha nincs, 0-t állít be alapértelmezettként, és megjeleníti.
* clickCounter: Minden gombnyomás után növeli a sessionStorage-ban tárolt kattintások számát, majd frissíti a megjelenített értéket. Ha a böngésző nem támogatja a Web Storage-t, figyelmeztetést jelenít meg.

**Web Worker**

* startWorker: Indít egy külön szálon futó "munkást" (worker.js), amely adatokat küld vissza (pl. számláló értékek) az oldalon történő megjelenítéshez.
* stopWorker: Leállítja a Web Worker működését, ha már fut.

**Helyzetmeghatározás**

* getLocation: A böngésző helyzetmeghatározási API-ját használja a felhasználó földrajzi koordinátáinak lekérdezésére. Ha a böngésző nem támogatja ezt, hibaüzenetet jelenít meg.
* showPosition: Megjeleníti a földrajzi hely adatait (szélesség, hosszúság).

**Drag and Drop**

* dragstartHandler: A húzott elem adatait elmenti a drag esemény során.
* dragoverHandler: Engedélyezi, hogy az elem fölé lehessen húzni más elemeket.
* dropHandler: Az elemet elhelyezi a célterületen a húzás végén.

**ChartJS:**

**HTML szerkezete**

1. <h2>5x5-ös Táblázat:
   * Ez az alcím a táblázat fölött jelenik meg, egyértelműen jelölve a tartalmat.
2. **Táblázat (**<table id="dataTable">**)**:
   * Az 5x5-ös táblázat celláiban numerikus adatok találhatók. Ezek például különböző statisztikai vagy példaszámokat képviselhetnek.
   * A táblázat tartalma statikus (fixen megadott az HTML-ben), de interaktív JavaScript segítségével kapcsolódik a vonaldiagramhoz.
3. <canvas id="lineChart">:
   * A **Canvas API** segítségével egy grafikus vonaldiagram jelenik meg. Ez az elem 600x300 pixeles méretet kapott, és interaktív módon frissül a táblázat adatai alapján.

**Vonaldiagram létrehozása és frissítése**

1. **Chart.js konfiguráció**:
   * A JavaScript kód a Chart.js könyvtárat használja, amely egy népszerű adatvizualizációs eszköz.
   * A diagram típusa: **line (vonaldiagram)**.
   * Az X-tengely **labels** értékei: ['1', '2', '3', '4', '5'], amelyek az adatpontok sorozatát jelölik.
   * Az Y-tengely skálája automatikusan a 0-tól kezdődik (beginAtZero).
2. **Adatok frissítése**:
   * Az eseményfigyelő (table.addEventListener) biztosítja, hogy ha egy táblázatsorra kattintasz, annak értékei (a cellák tartalma) bekerülnek a vonaldiagram adatmezőjébe.
   * A cellák értékeit a Array.from(row.children) metódus segítségével gyűjti össze és átalakítja számmá (parseFloat).
   * A frissített adatokat a chart.update() metódus jeleníti meg a diagramon.

**Interaktivitás és működés**

1. **Táblázatsorokra kattintás**:
   * Amikor a táblázat egyik sorára kattintasz, a JavaScript kód az adott sorban lévő cellák tartalmát begyűjti, majd ezeket az értékeket megjeleníti a vonaldiagramon.
   * Ha nem egy sorra kattintasz (például az üres területre), a kód egyszerűen visszatér (return).
2. **Vonaldiagram frissítése**:
   * Az adatokat a datasets[0].data mezőbe helyezi, amely a vonaldiagram jelenlegi állapotát reprezentálja.
   * A diagram az új adatokkal azonnal frissül, vizuális visszajelzést adva a felhasználónak.

**Miért hasznos ez a kód?**

* **Vizualizáció és interakció**:
  + Ez a megoldás nagyszerű példája annak, hogy a statikus HTML-t hogyan egészíthetjük ki interaktív JavaScript és adatvizualizációs eszközökkel.
  + Könnyen bővíthető például valós idejű adatokkal vagy adatbázis-kapcsolattal.
* **Felhasználóbarát funkciók**:
  + A táblázat sorai kattinthatók, és azonnal frissítik a diagramot, ami egyszerűvé és intuitívvá teszi az adatfeldolgozást.

**Ajax html:**

**Adatbeviteli mezők**

* A <label> és <input> mezők segítségével a felhasználó megadhat adatokat:
  + **ID**: Az adatok módosítására vagy törlésére használt azonosító.
  + **Név**, **Magasság**, **Súly**: Az új adatok megadása vagy meglévők frissítése ezekben a mezőkben történik.

**Funkciógombok**

* **Lekérdezés (**readData()**)**: Betölti és megjeleníti a meglévő adatokat.
* **Új adat (**createData()**)**: Hozzáad egy új adatot a megadott mezők alapján.
* **Adatok betöltése (**getDataForId()**)**: Az ID alapján egy adott adatot tölt be, amit módosítani lehet.
* **Adatok módosítása (**updateData()**)**: A megadott ID alapján frissíti az adatokat.
* **Törlés (**deleteData()**)**: Az ID alapján töröl egy adott adatot.

**Visszajelzés**

* <div id="feedback">: Az itt megjelenő üzenetek tájékoztatják a felhasználót az egyes műveletek sikerességéről vagy hibájáról.

**Eredmények megjelenítése**

* <div id="output">: A lekérdezett vagy frissített adatok itt jelennek meg a felhasználó számára.

**OOJS:**

**HTML elem**

* <h1>Vállalati dolgozók:
  + Ez a főcím megjeleníti az oldal tartalmának témáját, amely a vállalati dolgozókról szól.

**Dolgozó osztály**

1. **Osztály definiálása (**class Dolgozo**)**:
   * Az osztály egy sablon, amely lehetővé teszi, hogy dolgozók adatait struktúráltan tároljuk (név, beosztás, fizetés).
   * **Konstruktor (**constructor**)**: Az osztály példányosításakor inicializálja a dolgozó nevét, beosztását és fizetését.
   * info() **metódus**:
     + Létrehoz egy <p> elemet, amelyben megjeleníti a dolgozó adatait (név, beosztás, fizetés).
     + Az elem a weboldalra kerül (document.body.appendChild(p)).

**Vezető osztály**

1. **Osztály definiálása (**class Vezeto**)**:
   * A Vezeto osztály egy speciális típusú dolgozó, amely a Dolgozo osztályból örököl.
   * **Konstruktor**:
     + Az osztaly attribútum az adott vezető részlegét tárolja.
     + A super() hívás a szülő osztály konstruktorát hívja meg, ahol beállítja a dolgozó nevét, beosztását ("Vezető"), és fizetését.
   * bemutatkozas() **metódus**:
     + Létrehoz egy <p> elemet, amelyben megjeleníti a vezető nevét és osztályát.
     + Az elem a weboldalra kerül.

**Példányosítás**

1. **Dolgozó objektumok létrehozása**:
   * Példa dolgozók:
     + alkalmazott1: Kovács Anna, Fejlesztő, 600000 Ft/hó.
     + alkalmazott2: Nagy Péter, HR szakértő, 500000 Ft/hó.
   * A info() metódust hívva megjelennek ezek az adatok a weboldalon.
2. **Vezető objektum létrehozása**:
   * vezeto1: Horváth Gergő, IT részleg vezetője, 800000 Ft/hó.
   * A bemutatkozas() metódus megjeleníti a vezető nevét és osztályát, majd az info() metódus kiírja az általános dolgozói információkat.

**Mi történik futáskor?**

1. Az oldal betöltése után a dolgozók és vezető adatai dinamikusan jelennek meg.
2. A dolgozók adatai (alkalmazott1 és alkalmazott2) egyszerű <p> elemekben jelennek meg.
3. A vezető bemutatkozása és általános információja külön <p> elemben jelenik meg, demonstrálva a szülő osztály és az öröklés működését.

**Használati előnyök**

* **Modularitás**: Az osztályokkal könnyen kezelhető a dolgozói adatok szerkezete.
* **Öröklés**: A Vezeto osztály új funkciókat ad a Dolgozo osztály alapjaihoz anélkül, hogy ismétlődő kódot írnánk.
* **Interaktivitás**: Az adatok közvetlenül a weboldalon jelennek meg JavaScript segítségével.