WEB Beadandó Dokumentáció

Készítette: Wéber Marcell BW2OK8, Veres István

Weboldal linkje: http://gamf12.nhely.hu/WEB1-Beadando-main/index.html

A főoldal az index.html, ez egy egyszerű HTML5 szerkezetű kezdőlap. A dokumentum tartalmaz egy fejlécet (header) a címmel, egy navigációs menüt (nav) több oldalra mutató linkkel (kezdőlap, táblázat, HTML5, stb.), egy oldalsávot (aside), egy fő tartalmi részt (content div) üdvözlő szöveggel, és egy láblécet (footer) a készítők nevével. A HTML struktúra reszponzív designra készült (viewport meta tag), és külső CSS fájllal (style.css) van alakítva.

## Table.html



Ez a kód egy interaktív adatkezelő felületet valósít meg, amely lehetővé teszi felhasználói adatok hozzáadását, megjelenítését és kezelését táblázatos formában. A felület HTML5, CSS és JavaScript technológiákat használ

**Funkcionalitás:**

* **Név mező**: Szöveges bevitel, kötelező mező (2-30 karakter)
* **Kor mező**: Numerikus bevitel, életkorhoz (0-120 értékhatárokkal)
* **Város mező**: Szöveges bevitel (2-30 karakter)
* **Email mező**: Email cím validációval
* **Hibaüzenet**: Validációs hibák megjelenítésére szolgáló terület

Kereső funkció

**Funkcionalitás:**

* Dinamikus keresés a táblázat adatai között
* Valószínűleg JavaScript eseménykezelő figyeli a bevitelt és szűri az eredményeket

## Adattáblázat



**Funkcionalitás:**

* **Oszlop rendezés**: Kattintásra rendezhető oszlopok (név, kor, város, email)
* **Dinamikus tartalom**: A <tbody> elemet JavaScript feltölti a bevitt adatokkal
* **Műveletek oszlop**: Valószínűleg szerkesztés/törlés gombokat tartalmaz

Működési Elv

1. A felhasználó kitölti az űrlapot és elküldi
2. Az adatok validálódnak (kliens oldali validáció)
3. Ha minden rendben, az adat hozzáadódik a táblázathoz
4. A táblázat adatai rendezhetők és kereshetők
5. Hibák esetén hibaüzenet jelenik meg

Ez a kód egy tipikus CRUD (Create, Read, Update, Delete) felület vázlatos megvalósítását mutatja be, ahol a Create és Read műveletek láthatóak, az Update és Delete valószínűleg a "Műveletek" oszlopban lesz megvalósítva.



Ez a JavaScript kód egy interaktív adatkezelő rendszert valósít meg, amely a következő fő funkciókat tartalmazza:

* Űrlap adatok validálása és feldolgozása
* Dinamikus táblázatkezelés
* Adatok rendezése
* Hibakezelés

Változók:

* **form**: Az adatbeküldő űrlap referenciája
* **tableBody**: A táblázat törzsének referenciája, ahova az új sorok kerülnek
* **searchInput**: A keresőmező referenciája
* **errorMsg**: Hibaüzenetek megjelenítésére szolgáló elem
* **editingRow**: Jelenleg szerkesztett sor referenciája (ha van)
* **sortDirection**: Tömb, amely tárolja az egyes oszlopok rendezési irányát (növekvő/csökkenő)

Az űrlap a submit eseményére reagál, megakadályozza az alapértelmezett űrlapküldést (e.preventDefault()), és helyette saját logikát hajt végre.

Minden validációs hiba esetén a megfelelő hibaüzenet megjelenik az errorMsg elemben, és a függvény visszatér (return) a további feldolgozás megszakításával.Sikeres validáció esetén a hibaüzenet törlődik.



Ez a JavaScript kód egy adatkezelő rendszer része, ami két fő műveletet kezel: a táblázat sorainak szerkesztését és új sorok hozzáadását. Amikor a felhasználó kitölti az űrlapot, a rendszer először ellenőrzi, hogy épp szerkesztés van-e folyamatban. Ha igen, frissíti a kiválasztott sor adatait az űrlapból érkező új értékekkel, majd befejezi a szerkesztési módot. Ha nem szerkesztés, hanem új adatről van szó, létrehoz egy teljesen új sort a táblázatban a megadott név, kor, város és email címmel, valamint hozzáad egy "Szerkeszt" és "Töröl" gombot is. Mindkét esetben végül alaphelyzetbe állítja az űrlapot, hogy készen álljon a következő műveletre. A kód feltételezi, hogy a bevitt adatok már átmentek a szükséges ellenőrzéseken, és a szerkesztéshez és törléshez külön függvények (editRow, deleteRow) tartoznak, amik itt nincsenek részletezve.



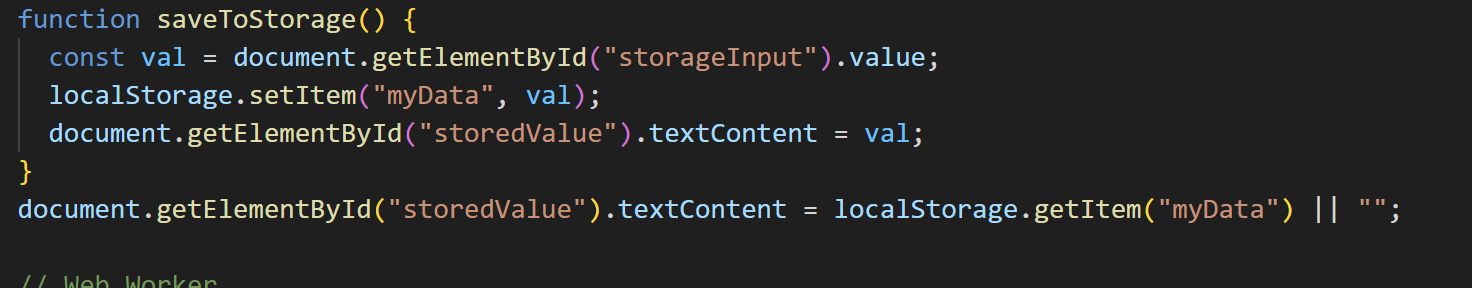
Ez a kód három fő funkciót valósít meg:

1. **Szerkesztési funkció**: Az editRow() függvény lehetővé teszi a táblázat sorainak módosítását. Amikor a "Szerkeszt" gombra kattintanak, a függvény kiválasztja a sort, kitölti az űrlapot a sor adataival, és eltárolja a szerkesztés alatt álló sort.
2. **Rendezési funkció**: A sortTable() függvény lehetővé teszi a táblázat oszlopainak rendezését. A függvény figyelembe veszi az oszlop típusát (szám vagy szöveg), és váltogatja a növekvő/csökkenő sorrendet. Külön kezeli a számértékeket (kor oszlop) és a szöveges értékeket (magyar nyelvi rendezés).
3. **Keresési funkció**: A keresőmező változásaira figyel (input esemény), és szűri a táblázat sorait a megadott érték alapján. A sorok csak akkor maradnak láthatóak, ha bármelyik cellájuk tartalmazza a keresett kifejezést (kis-nagybetű érzéketlenül).

Mindhárom funkció együttműködik a táblázat kezeléséhez, lehetővé téve az adatok interaktív szerkesztését, rendezését és szűrését a felhasználó számára. A kód figyelembe veszi a magyar nyelv sajátosságait a rendezés során, és reszponzívan reagál a felhasználói beavatkozásokra.

## Web Storage



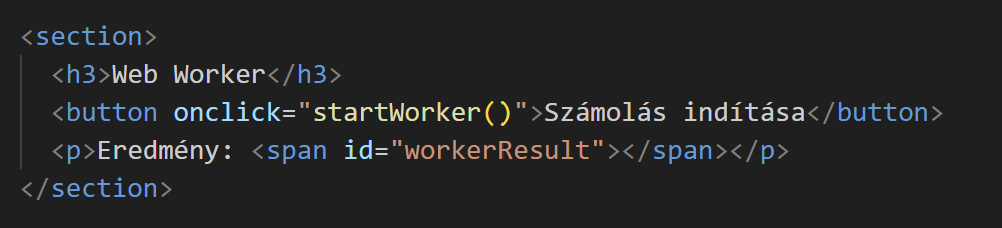


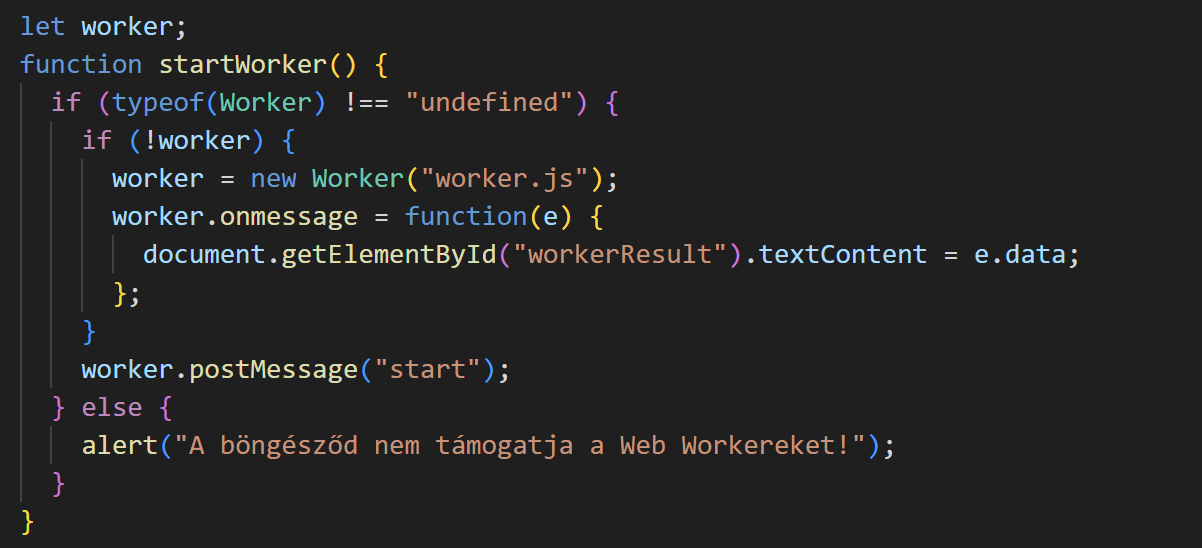
A rendszer egy beviteli mezőből, egy mentés gombból és egy megjelenítő szövegből áll, amely mindig mutatja az elmentett értéket. Amikor a felhasználó beír valamit a mezőbe és rákattint a mentés gombra, a saveToStorage() függvény elmenti ezt az értéket a localStorage-ba "myData" néven, majd azonnal frissíti a képernyőn látható értéket. Az oldal betöltésekor automatikusan beolvassa és megjeleníti a korábban elmentett adatot, ha az létezik, így a felhasználó mindig láthatja az utoljára mentett információt, még akkor is, ha bezárta és újra megnyitotta az oldalt. Ez a megoldás szemlélteti a localStorage alapvető működését, amely lehetővé teszi egyszerű adatok perzisztens tárolását a felhasználó böngészőjében.

**Funkcionalitás**:

* A felhasználó beírhat egy értéket a mezőbe
* A Mentés gombra kattintva az érték elmentődik a böngésző lokális tárolójába
* Az oldal újratöltése után is megmarad az érték
* A mentett adat mindig látható a felületen

## Web Worker



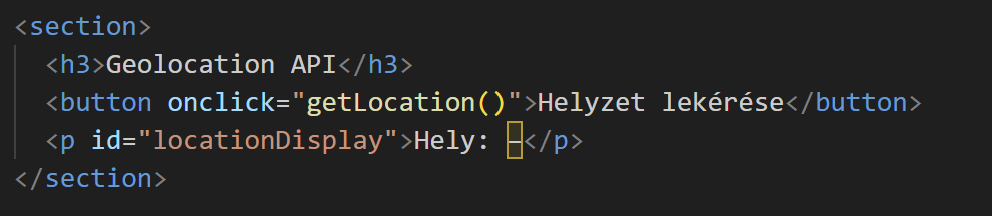


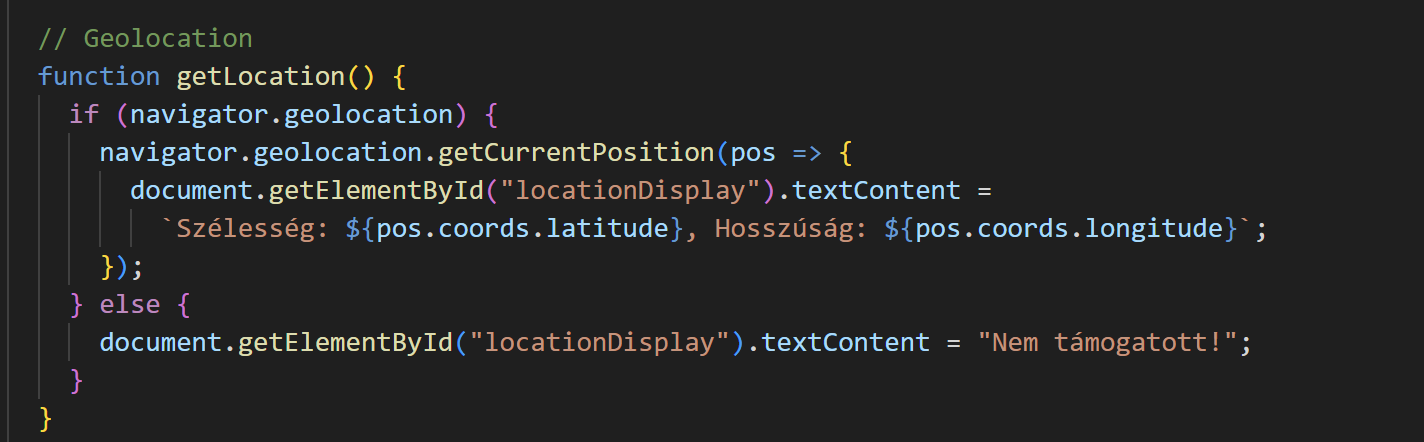
Ez a kód egy Web Worker-t használó megoldást mutat be, amely lehetővé teszi a processzor-igényes számítások háttérben történő elvégzését anélkül, hogy blokkolná a felhasználói felületet, ahol a "Számolás indítása" gombra kattintás elindítja a worker.js fájlban definiált háttérfolyamatot, melynek eredménye a "workerResult" elemben jelenik meg, miközben a rendszer figyelmezteti a felhasználót, ha a böngészője nem támogatja a Web Worker technológiát.

**Funkcionalitás**:

* A felhasználó a gombra kattintva indíthat háttérszámítást
* A számítás eredménye automatikusan megjelenik a felületen
* A megoldás megakadályozza, hogy a nehéz számítások befagyják a böngészőt
* Kompatibilitási ellenőrzéssel biztosítja, hogy csak támogatott böngészőkben fusson

Geolokáció





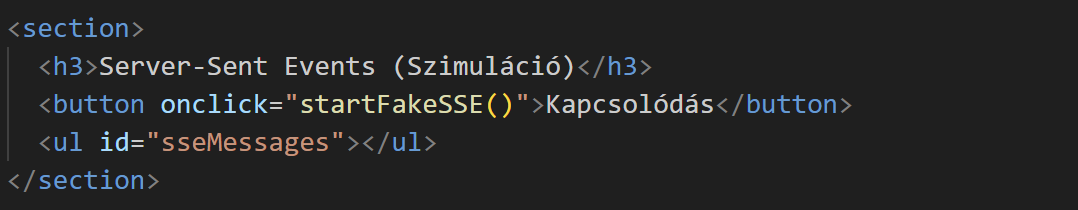
A HTML rész egy <section> elemet tartalmaz, amelyben egy címsor (<h3>), egy gomb (<button>), és egy bekezdés (<p>) található. A gombra kattintva meghívódik a getLocation() függvény, amely a locationDisplay azonosítójú bekezdésben jeleníti meg az eredményt.

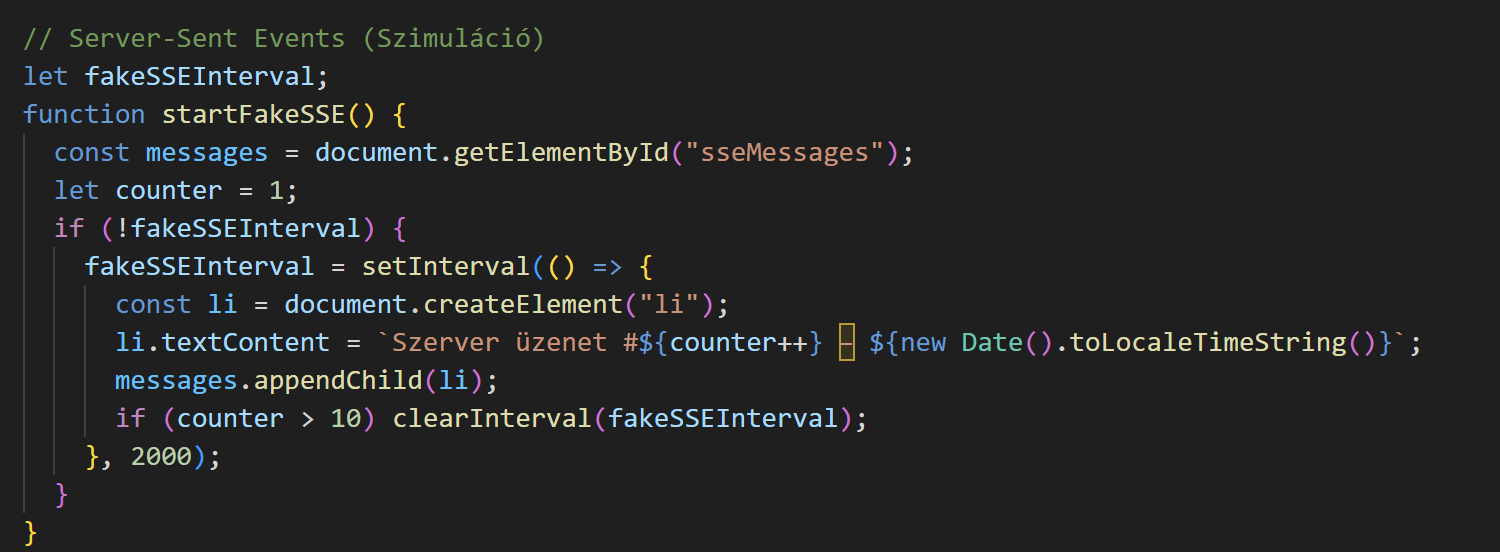
A JavaScript részben a getLocation() függvény először ellenőrzi, hogy a böngésző támogatja-e a Geolocation API-t (navigator.geolocation). Ha igen, akkor meghívja a getCurrentPosition() metódust, amely aszinkron módon lekéri a felhasználó pozícióját. A pozíció (pos) objektum tartalmazza a szélességi (latitude) és hosszúsági (longitude) koordinátákat, amelyeket a függvény beilleszt a locationDisplay elem szövegébe. Ha a böngésző nem támogatja a Geolocation API-t, a függvény a "Nem támogatott!" üzenetet jeleníti meg.

Példa kimenet:  
Szélesség: 47.4979, Hosszúság: 19.0402 (ez Budapest koordinátái).

A kód tehát egy felhasználóbarát módon kéri le és jeleníti meg a felhasználó aktuális helyzetét a böngésző képességeit felhasználva.

## Server-Sent Events (Szimuláció)





Ez a kód egy **Server-Sent Events (SSE)** szimulációját mutatja be, amelyet JavaScript segítségével valósít meg. Az SSE egy olyan technológia, amely lehetővé teszi a szerver számára, hogy valós időben küldjön üzeneteket a kliensnek (pl. böngészőnek) egy egyirányú kapcsolaton keresztül. Ebben a példában azonban nincs valódi szerver, helyette egy időzítő (setInterval) szimulálja az üzenetek érkezését.

**HTML rész**

* Egy <section> elemen belül található:
  + Címsor (<h3>), amely leírja a szimuláció célját.
  + Egy **"Kapcsolódás"** gomb (<button>), amely a startFakeSSE() függvényt hívja meg, amikor rákattintanak.
  + Egy üres <ul> lista (sseMessages azonosítóval), ahova a szimulált üzenetek kerülnek.

**JavaScript rész**

A startFakeSSE() függvény a következőket végzi:

1. **Megkeresi az üzenetek listáját** (sseMessages).
2. Létrehoz egy számlálót (counter), amely nyomon követi a küldött üzenetek számát.
3. **2 másodpercenként** (2000 ms) generál egy új üzenetet a setInterval segítségével:
   * Létrehoz egy <li> elemet.
   * Beállítja a szövegét: "szerver üzenet #X [aktuális dátum-idő]" (pl. szerver üzenet #1 [2025.04.26. 14:30:00]).
   * Hozzáadja az üzenetet a listához (appendChild).
4. **10 üzenet után** leállítja az időzítőt (clearInterval), ezzel szimulálva a kapcsolat lezárását.

## Drag and Drop





A kód egy **Drag and Drop (húzd és ejtsd)** funkciót valósít meg, amely lehetővé teszi, hogy egy elemet áthúzzunk egy másik területre, majd ott elengedjük. A megvalósítás során HTML, CSS és JavaScript együttes használatára van szükség.

**HTML rész**

A HTML kódban egy section elem található, amelyben a következők szerepelnek:

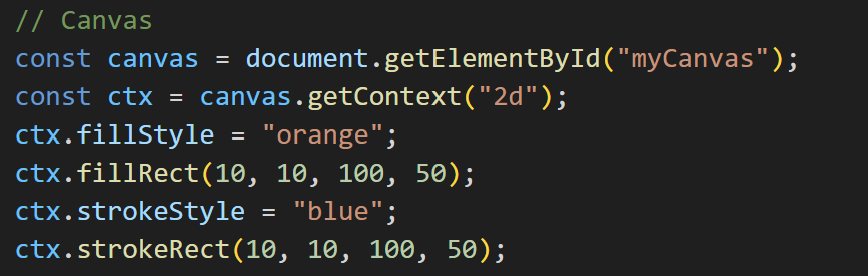
* **Címsor (**h3**):** "Drag and Drop" szöveg jelenik meg, ami bemutatja a funkció célját.
* **Húzható elem (**div**):**
  + A draggable="true" attribútum teszi lehetővé a húzást.
  + Az id="dragMe" segítségével a JavaScript kódban kiválasztható.
  + Stílusa (CSS) meghatározott: kék háttér (#3498db), fehér szöveg, rögzített szélesség és magasság.
  + Tartalma: "Huzz el!" szöveg.
* **Célzóna (**div**):**
  + Az id="dropZone" azonosítja a területet, ahova az elemet el lehet helyezni.
  + Tartalma: "Dobj ide" szöveg.

**JavaScript rész**

A JavaScript kód három eseményfigyelőt (event listener) állít be:

1. dragstart**(húzás kezdete):**
   * A dragMe elemen figyeli, amikor a felhasználó elkezdi húzni az elemet.
   * Az e.dataTransfer.setData() metódus segítségével elmenti a "Húzott elem" szöveget, ami később felhasználható.
2. dragover**(elem áthalad a célzóna felett):**
   * A dropZone elemen figyeli, amikor a húzott elem a célterület felett van.
   * Az e.preventDefault() metódus meggátolja a böngésző alapértelmezett viselkedését, ami egyébként blokkolja az ejtést.
3. drop**(elem elengedése a célzónában):**
   * A dropZone elemen figyeli, amikor a felhasználó elengedi a húzott elemet.
   * Az e.preventDefault() biztosítja, hogy a dobás működjön.
   * A célzóna szövegét módosítja "Sikeres dobás!"-ra.

## Canvas



A kód elején a canvas változóba eltároljuk a HTML dokumentumból azt az elemet, amelynek az azonosítója "myCanvas".  
Ezután a ctx változóban lekérjük ennek a canvas elemnek a kétdimenziós ("2d") rajzoló kontextusát, amelyen rajzolni tudunk.  
A ctx.fillStyle = "orange"; sor beállítja, hogy a kitöltés színe narancssárga legyen.  
A ctx.fillRect(10, 10, 100, 50); sor egy narancssárga téglalapot rajzol a canvasra, amelynek a bal felső sarka a (10, 10) koordinátán van, szélessége 100 pixel, magassága pedig 50 pixel.  
Ezután a ctx.strokeStyle = "blue"; sor beállítja, hogy a vonal (kontúr) színe kék legyen.  
Végül a ctx.strokeRect(10, 10, 100, 50); sor egy kék színű keretet (kontúrt) rajzol pontosan ugyanoda, ahol a kitöltött téglalap is található.

## Chart.html

A táblázat öt különböző sorból áll, mindegyikhez öt érték tartozik. Amikor a felhasználó rákattint egy sorra, az adott sor értékei megjelennek a vonaldiagramon. A kiválasztott sor vizuálisan is kiemelésre kerül. Az oldal tartalmaz továbbá egy egyszerű navigációs menüt, oldalsávot, illetve egy láblécet a készítők nevével.

## Ajax.html

A képen szöveg, képernyőkép, Betűtípus látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

egy **navigációs menü** (<nav>), amelyben több menüpont is található különböző oldalakra mutató hivatkozásokkal (index.html, tabla.html, html5.html, stb.), ahol az aktuális oldal (ajax.html) menüpontja ki van emelve az active osztály segítségével.  
Az oldalsávban (<aside>) egy egyszerű szöveg jelenik meg: *"Oldalsáv tartalom"*.  
A fő tartalom (<div class="content">) négy nagy részre tagolódik:

* Az első szekcióban (**"Adatok megjelenítése"**) található egy gomb, amely a readData() függvényt hívja meg kattintásra; ez a gomb valószínűleg lekéri az adatokat egy szerverről és megjeleníti azokat a #data-list azonosítójú <div>-ben, statisztikai összegzésekkel a #stats nevű <div>-ben.
* A második szekció (**"Új adat hozzáadása"**) három bemeneti mezőt tartalmaz (név, magasság, súly megadására), és egy gombot, amely a createData() függvényt hívja meg, amivel az új adatot lehet létrehozni; visszajelzések a #create-msg divbe kerülnek.
* A harmadik szekció (**"Adat módosítása"**) lehetőséget ad egy meglévő rekord ID alapján történő betöltésére (getDataForId() függvény), a módosított adatok beírására, majd a updateData() függvény segítségével azok frissítésére; van egy "Visszavonás" gomb is (deleteUpdateId()), amely valószínűleg törli vagy kiüríti az aktuálisan módosításra kiválasztott adatot.
* A negyedik szekció (**"Adat törlése"**) egy bemeneti mezőt tartalmaz a törlendő adat ID-jának megadására, és egy gombot, amely a deleteData() függvényt indítja el, a sikerességről vagy hibáról a #delete-msg div ad visszajelzést.

Chart.js

A képen szöveg, képernyőkép, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

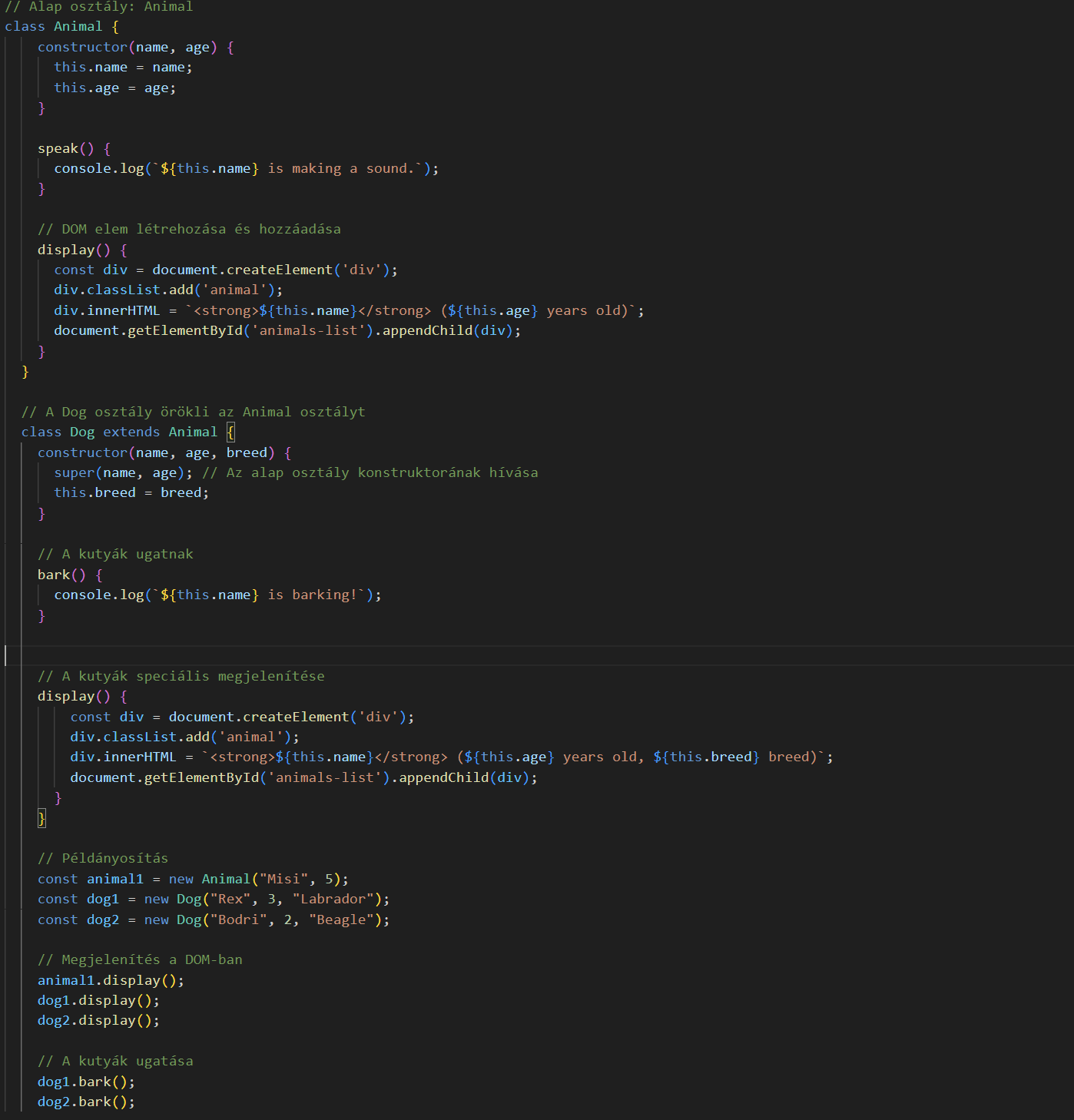
Ez a kód létrehoz egy **vonaldiagramot** a **Chart.js** segítségével, amely az oldal <canvas id="lineChart"> elemébe rajzolódik ki. A diagram kezdetben üres adattal indul (data: []), az X-tengelyen az "Érték 1"–"Érték 5" címkék szerepelnek. A grafikon megjelenési beállításai között szerepel például a vonal színe, az adatpontok mérete, az animált vonalhajlítás (tension), valamint hogy a diagram kitöltse a vonal alatti területet halvány színnel. A responsive: true miatt a grafikon automatikusan méreteződik a képernyőhöz. A title plugin megjelenít egy címet a diagram felett, míg a tooltip beállítások segítik a lebegő információk (tooltipek) helyes megjelenítését az adatok fölé mozgatva az egeret. A skálák (scales) beállításai biztosítják, hogy a tengelyek nulláról induljanak és halvány rácsvonalakkal legyenek ellátva.

A képen szöveg, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Ez a kódrészlet azt a működést valósítja meg, hogy amikor a felhasználó rákattint egy sorra a #dataTable azonosítójú HTML táblázatban, akkor először eltávolítja az összes korábban kiválasztott sorból a selected CSS-osztályt, majd az éppen kattintott sorhoz hozzáadja azt, ezáltal vizuálisan kiemelve a kijelölt sort, majd az adott sor celláiból (az első cellát kihagyva, mivel az általában egy cím vagy azonosító) egy tömböt készít, amely minden cella szövegét számmá alakítja (Number), ezekből az értékekből pedig frissíti a korábban létrehozott vonaldiagram adatpontjait (chart.data.datasets[0].data = values), illetve a diagram címkéjét is dinamikusan módosítja úgy, hogy az első cella (pl. név vagy sorazonosító) értékét megjeleníti a címben (${row.cells[0].textContent} értékei), végül pedig a chart.update() hívással újrarajzolja a grafikont, hogy a változások azonnal láthatóak legyenek, továbbá az utolsó sorban a document.querySelector("#dataTable tbody tr").click(); parancs gondoskodik arról, hogy az oldal betöltődésekor automatikusan az első sor legyen kijelölve, és így rögtön látható legyen a hozzátartozó adatsor a grafikonon.

## ooj.js



Ez a JavaScript kód két **osztályt** (class) definiál: egy általános **Animal** (állat) osztályt és egy **Dog** (kutya) osztályt, ami az Animal-ből öröklődik.

* **Animal osztály:**
  + Minden állatnak van **neve** és **kora**.
  + Tud egy általános hangot kiadni (speak() függvény – ami most csak kiírja a konzolra).
  + A display() függvény segítségével létrehoz egy új HTML <div> elemet, amiben az állat nevét és korát megjeleníti, majd hozzáadja azt az oldal #animals-list azonosítójú eleméhez.
* **Dog osztály:**
  + Az Animal osztálytól örökli a **nevet** és **kort**, de emellett van egy saját új tulajdonsága is: **fajta** (breed).
  + Tud ugatni a bark() függvénnyel (konsolra írja ki az ugatást).
  + A saját display() függvényét felülírja, hogy a kutya **nevét**, **korát** *és* **fajtáját** is megjelenítse a DOM-ban.
* **Példányosítás:**
  + Készült egy animal1 nevű sima állat (Misi, 5 éves).
  + Készült két kutya (dog1 és dog2) különböző nevekkel, korokkal, fajtákkal.
  + Ezek az objektumok megjelennek a weboldalon az #animals-list részben.
  + A kutyák külön meg is ugatnak (bark() hívásra a konzolban).

## React

A képen szöveg, képernyőkép, Betűtípus, szoftver látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A reactek itt találhatóak meg. Az app.js-be találhatóak a játékok. src2-ben a kő-papír olló, src1-ben a számkitalálós játék találhatóak.

A képen szöveg, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Az alkalmazás egy egyszerű számkitalálós játék, amelyben a getRandomNumber függvény segítségével egy véletlenszerű 1 és 100 közötti szám kerül generálásra, amikor az oldal betöltődik. A useState hook-ok segítségével külön tárolja a kitalálandó számot, a felhasználó aktuális tippjét, az üzenetet, az eddigi próbálkozásokat, valamint azt az állapotot, hogy véget ért-e már a játék. A felhasználó beír egy számot az input mezőbe, majd a "Tippelés" gomb megnyomására lefut a handleGuess függvény, amely ellenőrzi, hogy a bevitt érték valóban szám-e, illetve 1 és 100 között van-e. Ha a tipp helyes, akkor a program kiírja, hogy hányadik próbálkozásra sikerült eltalálni a számot, és lezárja a játékot, ha pedig a tipp kisebb vagy nagyobb a célszámnál, akkor a felhasználót figyelmezteti, hogy próbálkozzon magasabb vagy alacsonyabb számmal. A felhasználó minden tippje eltárolásra kerül a guesses tömbben, amelynek elemei megjelennek a képernyőn. A "Új játék" gomb megnyomásával a restartGame függvény új véletlenszámot generál, kiüríti a próbálkozásokat és üzeneteket, így a játék újraindul.

A képen szöveg, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

A képen szöveg, képernyőkép látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

Ez a React alapú alkalmazás egy **Kő-Papír-Olló** játékot valósít meg, amelyben a felhasználó a három lehetőség közül (Kő, Papír, Olló) választhat egy gomb megnyomásával, miközben a számítógép véletlenszerűen választ a három opció közül. A useState hook segítségével az alkalmazás külön-külön tárolja a felhasználó választását, a számítógép választását, az eredményt, valamint a győzelmek, vereségek és döntetlenek számát egy score nevű objektumban. Amikor a felhasználó rákattint valamelyik gombra, a handleClick függvény hívódik meg, amely generál egy véletlenszerű számítógépes választ, majd meghívja a getResult függvényt, amely kiértékeli a játék kimenetelét a két választás alapján, és visszaadja az eredményt ("Nyertél!", "Vesztettél!" vagy "Döntetlen"). Az eredményt megjeleníti a képernyőn, illetve frissíti a pontszámokat attól függően, hogy a felhasználó nyert, vesztett vagy döntetlent játszott. A "Pontszám nullázása" gomb segítségével a resetGame függvény nullázza a játék állapotát és az összes pontszámot, így a játékos tiszta lappal kezdheti újra a játékot.