# Webtechnológia beadandó dokumentáció


## Kovács Dániel

## F9Y7TW

# 0.Bevezetés

Ez a weboldal, amely egy egyetemi beadandó a webtechnológiák 1 tantárgyhoz, egy gyorsokkal kapcsolatos tartalommal rendelkezik. Megtudhatjuk, a történetüket, híres gyorsokat a történelemből, valamint megtudhatjuk azt is, hogy milyen gyrosok is vagyunk valójában. Az ötlet abból fakad, hogy mikor gondolkodtam, hogy miből is kéne a beadandót írnom, nagyon éhes voltam, és csak a gyros jártak a fejemben.

# 1.Használt technológiák:

- Programok:
    - Visual studio code (VSC): Egyszerű szövegszerkesztő program, amit különböző extensionokkal (bővítményekkel) lehet a kedvünk szerint formázni. A legfontosabb számomra a HTML, css, JavaScript támogatásáról szólók, valamint egy live server, aminek a segítségével amint mentettem a html-t, frissült a weboldal.
- Programozási nyelvek:
    - HTML: Weboldalak frontendjében nagy szerepet játszó script-nyelv, amivel a weboldal elmeit tudjuk létrehozni
    - CSS: A weboldal elemeinek a stilizálására használt nyelv, elhelyezkedésüket, kinézetüket lehet velük befolyásolni
    - JavaScript: A frontendekhez (a weboldalaknál) használt tényleges programozási nyelv.

*Megjegyzés: a program mérete miatt nem használtam semmiféle keretrendszert*

2. A program elemei:

A program 4 html-ből, hozzájuk tartozó css-ekből, valamint egy globálisból áll ami álltalános stílusokat tartalmaz. Valamint két Javascript file-ból áll. Az egyik az animációkért, a másik a kérdőív kiértékelésért, valamit egyéb funkciójáéért felel. Található még egy assets mappa, ami a képeket tartalmazza, amelyeket a weboldalon használtam.

## *HTML-ek (a tartalmuk különbözik, de például a fejléc mindegyiknél ugyanaz):*

index.html: A weboldal belépési része, van egy fejléce (ez mindegyikre igaz lesz), ami tartalmazza a weboldal nevét, valamit az egyéb lehetőségeket: kezdőlap (az index), történelme, ami a gyros történelmén vezet végig minket, híres ételek, itt a történelem folyamán élt és megevett gyrosokról tanulhatunk, valamint a gyros-kérdőív, ahol megtudhatjuk, hogy milyen eledelek vagyunk is valójában. Maga a html egy rövid, a bobster classú divben található, (AI-által generált) bemutatót tartalmaz egy cím objektummal, egy paragrafussal, valamint egy felsorolással.

history.html: Itt is van ugyanolyan fejléc, és a bobster div is megtalálható, mint az összes többinél, ami az indexben is található annyi különbséggel, hogy a történelem gom szürke, ez mindig is igaz lesz, ha az adott html-en vagyunk, ahova a gomb hivatkozik. Maga a sub-weboldal tartalma semmi különös leírások, illetve képek, amelyeket flex-boksz-okkal oldottam meg, hogy egymás mellett jelenjenek meg. Az ő kinézetéért a history.css, valamint a style_all.css felel.

famous.html: A híres gyrosokat tartalmaz, melyeket egymás mellett helyezkednek el. Úgy van megoldva, hogy van egy div conatiner (ami felx-re van állítva) és abban van egy kép, aminek a border radius-a hatalmas, hogy kör alakú legyen. Az animációért a saját, faous.css felel, ugye mikor hover-el rámegyünk a képre, akkor növelem a box-shadow-át, illetve annak színét

```
.profile:hover{
    box-shadow: 0 0 5px 10px rgb(146, 160, 232);
}
.profile {
    display: flex;
    justify-content: center;
    width: 195px;
    height: 195px;
    margin: 20%;
    margin: 2px;
    box-shadow: 0 0 0 0px rgb(255, 255, 255);
    border-radius: 100px;
    animation: testAnim 1s forwards;
    margin: 20px;
    cursor: url("assets/cursor_point.png"), pointer;
    transition: box-shadow 0.5s ease, border-color 0.5s ease
}
```

Illetve van még egy hidden popup div, is, ami a képekre történő kattintás után ugrik fel, ő egy css-animációval jelenik meg, de erről majd az animations.js során fogok jobban belemenni.

me_gyros.html: A legkomplexebb, leghosszabb html az összes közül ugyanis ez tartalmazza a kérdőívet. A fejléc, mint a többinél, itt is ugyanaz, valamint megtalálható a bobster is. Ezután jön a leghosszabb része, a form. Mindegyik kérdésnek van egy class-a a kerdes, ami majd a válaszok könnyebb szűréséért felel majd, aztán jön a választások része, amit id-k valamint név alapján köt össze, mint bármelyik html-ben szereplő form-nál. Fontos hogy mindegyik kérdéshez van egy value rendelve, ami majd a kiértékelésnél játszik majd nagy szerepet. A végén szerepel még egy gomb, ami a kiértékelést hajtja végre (lefuttatja a kerdo() függvényt a kerdo.js-ből), valamint egy hidden kiértékelés div, ami egy képet és a hozzá tartozó szövegeket tartalmazza.

```
<div id="kiertekeles" hidden>
    <img class="kepcske">
    <h3></h3>
    <p></p>
</div>
<div class="submit_button">
    <button id="submit" onclick="kerdo()">Küldés</button>
</div>
```

## *CSS-ek:*

styles_all.css: Egy univerzális css ami az összes html-ben találhatüó elemek stílusát határozza meg. Ilyen a fejléc, a gombok stílusa, elhelyezkedése, igazodása a weboldal méretéhez (@media-val), ő állítja be a gyros cursort az assets mappából, valamint az animációt is ő tartalmazza, ha új html-tölt be.

history.css: ő csupán a history-nál a képek szöveg melletti beállításáért, felel.

famous.css: ő állítja be a #box-class-t a mi a profilokat tárolja, maga a profilok stilizálását is ő végzi, hogy mindig középről töltsék fel a teret, ha meg kicsi a képernyő akkor egymás alatt jelenjenek meg, valamint a hover-ért is ő felel, ami egy egyszerű box-shadow növelés.

```
.profile:hover{
    box-shadow: 0 0 5px 10px rgb(146, 160, 232);
}
```

Végezetül pedig ő felelős a popup stílusának is a kezeléséért, ami akkor történik, ha valaki rámegy a profilra. Kezdetben az hidden, amint kiválasztás után az animations.js segítségével jeleníti meg, azzal, hogy rákerül a .visible class.

kerdo.css: Akárcsak a html, ez a leghosszabb css, ami csupán egy weboldal-szakaszhoz kell. Mert hát neki kell beállítani a kérdések kinézetét, amit .kerdes osztálynak a segítségével végez, valamint a submit, illetve a kiértékelésnek, azon belül is a képnek, meg a szövegnek a stílusát is ő állítja be.

## *JavaScriptek:*

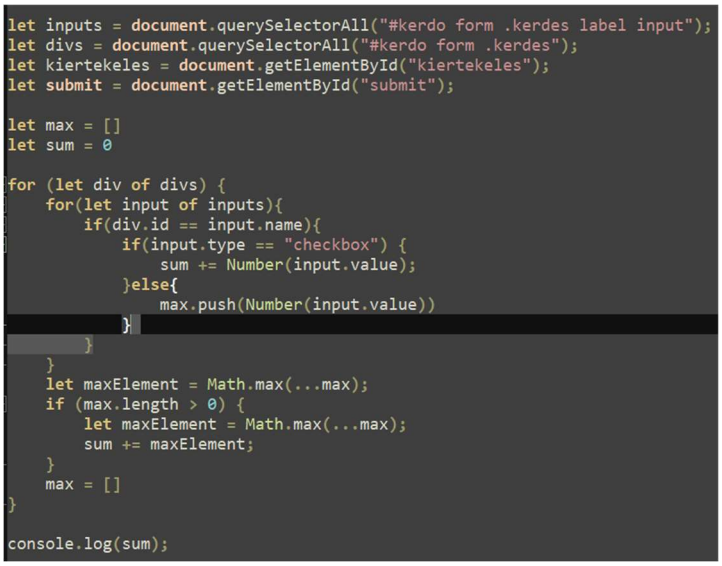
Ők a legfontosabb elemei az egész weboldalnak, attól függetlenül, hogy csupán ketten vannak. Haladjunk az egyszerűbbtől a bonyolultabb felé.

animations.js: A legelső része az egésznek a html váltásoknál látható kis animáció, hogy a body-ja alulról úszik fel. Ezt a documentLoaded event listenerrel oldottam meg. Kezdetben a bobster (ami igazából nálam a body) láthatatlan, de amit betölt ez a JavaScript kód ad hozzá egy visible-classt.

```
document.addEventListener('DOMContentLoaded', function() {
    setTimeout(() => {
        const content = document.querySelector('.bobster');
        content.classList.add('visible');}, 30);
});
```

Következő része az egyszerűen a history.html szövegeit, letároló, majd megjelenítő rész. Ennek a legfontosabb függvénye a speekingBox(text, color), ami az egyik profile-ra történő kattintásra jelenik meg. Bekapja a text-azonosítóját, amit a fenti paragraphs-ból kiszedve belerakja az popup inner html-jébe, de először annak hátterét beállítja a color-ként kapott változóra. Majd végül elveszi töle a hidden classt, és átírja appear-ra. A speekingBoxClose() ugyanezt csinálja csak fordítva.

kerdo.js: A kérdőiv működéséért felelős komplexebb javascript file. Található egy, html betöltésénél automatikusan lefutó script, ami nem más, mint a max pontszámot kiszámító kódrészlet. Ez a script alján helyezkedik el, és így néz ki:



```
let inputs = document.querySelectorAll("#kerdo form .kerdes label input");
let divs = document.querySelectorAll("#kerdo form .kerdes");
let kiertekeles = document.getElementById("kiertekeles");
let submit = document.getElementById("submit");

let max = []
let sum = 0

for (let div of divs) {
    for(let input of inputs){
        if(div.id == input.name){
            if(input.type == "checkbox") {
                sum += Number(input.value);
            }else{
                max.push(Number(input.value))
            }
        }
    }
    let maxElement = Math.max(...max);
    if (max.length > 0) {
        let maxElement = Math.max(...max);
        sum += maxElement;
    }
    max = []
}

console.log(sum);
```

kiszedi az inputok közül a legnagyobb value-kat (checkbox-oknál az összeset), majd összeadja őket. Ez lesz a kérdőív maximum pontszáma. Ezt úgy éri el, hogy a kerdes osztály alapján kigyűjti az inputokat és az azonos nevűeket egy tömbbe rakja, majd kiválasztja ki a legnagyobbat és hozzáadja a sum változóhoz.

*kerdo():* A kérdőív működtetéséért felelős függvény. Legelőször kiszedi a kerdes diveket, ők tartalmazzák a kérdéseket, és a rájuk felelhető válaszokat. Aztán egy for ciklus következik benne, ami az összes kérésre megadott pontokat adja össze (ezt a kiert(name) fügvény meghívásával teszi meg). Ezáltal fogja eldönteni, hogy milyen gyrosok is vagyunk. Átállítja a beküldés gombján a szöveget, illetve az onclick effectet, hogy az oldalt frissítse a kerdo() meghívása helyett, majd módosítja a kiértékelés-nek a képét, illetve a szövegét, valamint eltünteti a kérdőívet, valamint láthatóvá teszi a kiértékelés div-jét. A szöveget és a képet a választások folyamán megadott válaszok pontjaiból értékeli ki, mint egy dolgozat. Minél nagyobb értelmetlenséget jelölt be a delikvens annál több pontot kap.

*kiert(name):* Ez a fügvény az ellenörző, illetve a pontkereső függvényé a javascriptnek. Legelőször is leveszi az összes error classt a kérdésekről, majd utána megkeresi mindegyik elemnél a bejelöltet (checkbox-nál nem áll megf az elsőnél), és visszaadja az értékét, ha nem talált semmit -1 a visszatérés, ebből tudja a program, hogy valami gubanc van, valami nincs bejelölve, valamint hozzáadja az adott kerdes-hez az error class-t, ami csupán pirosra színezi a hátterét az adott kérdésnek, valamint nem engedi a beküldést sem, ha van ilyen.