

BKSZC Pogány Frigyes Technikum

Bokszklub

szoftverfejlesztő és -tesztelő vizsgaremek

2025. május

Készítette: Németh Dániel, Vámosi Kevin

## A feladat leírása

Boksz edzéseknek készített weboldal a KeDa Boxing azért jött létre, hogy az emberek jelentkezhessenek online a boksz és önvédelmi tudásuk elsajátítására.

Célunk az, hogy tanoncainkkal megszerettessük a küzdősportot és tiszteletre, alázatra neveljük őket.

A feladat olyan webhely létrehozása, amelyen a látogatók tájékozódhatnak a boksz edzésekről az induló boksz csoportokról, és online jelentkezhetnek ezekre. Egy csoportba legfeljebb 10 tanonc jelentkezhet.

Az adatokat egy online adatbázisban kell tárolni, amelyet egy API-n keresztül lehet elérni.

Az elkészített weblapok az API-ról töltik le és jelenítik meg az adatokat.

## GitHub repository

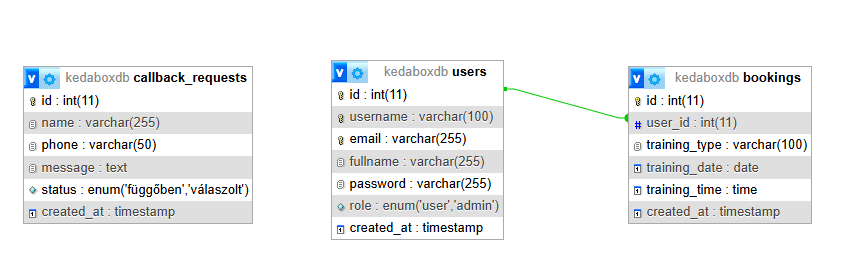
Az alkalmazások kódjai ebben a repository-ban érhetők el:

https://github.com/nemethdanielf/kedabox-vizsgaremek

## Az adatbázis elkészítése

Az adatbázist a MySQL adatbázis-kezelővel és a PHPMyAdmin programmal készítettük el.

Az adatokat három táblára bontottuk:



## A callback\_requests adatai

A name a kliensnek a neve azonosítója (automatikusan növekedik), ez a kulcs.

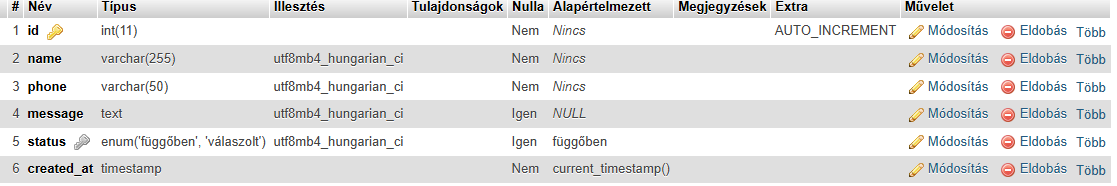
A phone a kliensnek a telefonszáma, nem lehet üres.

A message a kliensnek az üzenete amit nem kötelező írnia.

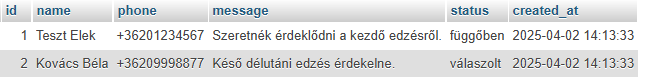
A status pedig azt mutatja, hogy a kliens visszahívását megválaszolták-e avagy függőben hagyták.

A created\_at pedig megmutatja nekünk, hogy a kliens mikor küldte a visszahívást.

Ennek megfelelően a callback\_requests tábla szerkezete:



Az alkalmazás teszteléséhez az alábbi tesztadatokat vittük be.:



## A users adatai

A user azonosítója (automatikusan növekedik), ez a kulcs.

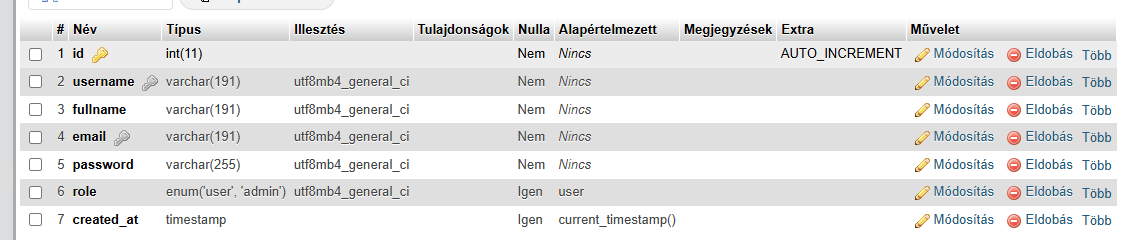
A user neve, nem lehet üres.

A user e-mail címe.

A role pedig kiválasztja, hogy a felhasználó admin-e vagy sem.

A created\_at pedig megmutatja nekünk, hogy a felhasználó mikor lett regisztrálva.

Ennek megfelelően a users tábla szerkezete:



Az alkalmazás teszteléséhez az alábbi tesztadatokat vittük be.



## A bookings adatai

Az id a foglalásnak az azonosítója (automatikusan növekedik), ez a kulcs.

A user id az a vendégnek az azonosítója, nem lehet üres, külső kulcs.

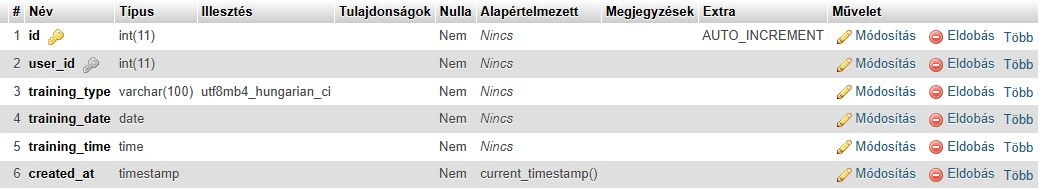
A training\_type megmutatja, hogy a vendég milyen típusú edzést kért nem lehet üres, külső kulcs.

A training\_date az edzésnek, a dátuma nem lehet üres.

A training\_time az edzésnek a pontos időpontját mutatja meg nem lehet üres.

A created\_at pedig azt mutatja, hogy a vendég mikor foglalta le az adott edzést.

Ennek megfelelően a bookings tábla szerkezete:



Az alkalmazás teszteléséhez az alábbi tesztadatokat vittük be.



# Fejlesztői dokumentáció-index.html

Ez a dokumentáció a KeDa Boxing weboldal kezdőlapját tartalmazó **index.html** fájlhoz készült. Az oldal felépítése és a különböző HTML, CSS, valamint JavaScript funkciók részletes magyarázatát találhatjátok alább.

## **HTML Dokumentum Felépítése**

### 1.1 **Fejléc (Head) szakasz**

A <head> szakaszban található metaadatok és külső erőforrások linkjei.

#### - <meta charset="utf-8"/>

Az oldal karakterkódolása, amely biztosítja, hogy az oldal a nemzetközi karakterek (pl. ékezetek) helyesen jelenjenek meg.

#### - <meta content="IE=edge" http-equiv="X-UA-Compatible"/>

Ez az utasítás arra kéri az Internet Explorer böngészőt, hogy az oldal legújabb renderelési módjában működjön, ezzel biztosítva a megfelelő kompatibilitást.

#### - <meta content="width=device-width, initial-scale=1, shrink-to-fit=no" name="viewport"/>

Ez a meta tag mobilbarát kialakítást biztosít, amely megfelelő méretben jeleníti meg az oldalt minden eszközön, figyelembe véve a képernyő szélességét.

#### - Külső stíluslapok:

* **Owl Carousel** (karusszel) stíluslapja.
* **Bootstrap**: A responsive dizájn és az alapvető UI komponensek stílusait tartalmazza.
* **Google Fonts**: A betűtípusok betöltéséhez szükséges linkek (pl. Baloo Chettan, Dosis, Poppins).
* **Egyéni stílusok** (css/style.css és css/responsive.css): Az oldal egyedi megjelenését és reszponzivitását biztosítják.

## 2. **Törzs (Body) szakasz**

### 2.1 **Fejléc (Header) szakasz**

A fejléc a weboldal tetején található navigációs és egyéb alapvető információkat tartalmazza.

* **Navigációs menü**: A navigációs sáv (navbar) biztosítja a látogatók számára, hogy könnyen navigálhassanak az oldalon. A **navbar-toggler** gomb lehetővé teszi a mobil eszközökön történő menü megjelenítést és elrejtést.
* **Kapcsolat információk**: Az oldal tetején elérhetőek az alapvető kapcsolati információk (telefon, email, cím). Ezek a szakaszok ikonokkal és szövegekkel vannak reprezentálva.
* **Dinamikus navigációs linkek**: A bejelentkezési állapot alapján kétféle navigációs menüpont jelenhet meg:
  + Bejelentkezés gomb, ha nincs bejelentkezve a felhasználó.
  + Profil link, amely a felhasználó nevével jelenik meg, ha be van jelentkezve.

### 2.2 **Slider szakasz**

A slider (karusszel) dinamikusan váltakozó képeket jelenít meg, melyek a különböző szolgáltatásokat népszerűsítik (pl. edzések, edzésterv, felszerelés). Az Owl Carousel segítségével valósul meg a görgethető képsorozat. A navigációs gombok és az indikátorok biztosítják, hogy a felhasználó könnyen váltogathasson a képek között.

#### - **Autoplay**: A képek automatikusan váltakoznak, de a felhasználó megállíthatja a forgást a kurzorral.

#### - **Mobilbarát dizájn**: A slider rugalmasan alkalmazkodik a különböző képernyőméretekhez.

### 2.3 **Rólunk szakasz**

Ebben a szakaszban a KeDa Boxing edzéseinek leírása található. Az információk a szövegen kívül egy gombot is tartalmaznak, amely kapcsolatfelvételt ajánl.

#### - **Elrendezés**: A rólunk szakasz egy kétszintes elrendezésben jelenik meg, amely biztosítja, hogy a tartalom áttekinthető legyen különböző eszközökön.

### 2.4 **Edzések szakasz**

Ez a szakasz az edzővideókat népszerűsíti, amelyek a felhasználók számára otthoni edzést és technikai gyakorlatokat biztosítanak.

#### - **Owl Carousel**: A videókat tartalmazó szakasz ismételten egy dinamikus, görgethető karusszel, amely lehetővé teszi a felhasználók számára, hogy gyorsan átlássák az edzéseket.

### 2.5 **Blog szakasz**

A blog szakasz bemutatja a legutóbbi bejegyzéseket, amelyek például a bokszolókról és azok motiváló történeteiről szólnak. Az egyes blogposztok egy-egy képpel és rövid leírással vannak ellátva, amelyben a felhasználók további információkat találhatnak.

### 2.6 **Vélemények szakasz**

A tanulók véleményei jelennek meg egy újabb Owl Carousel komponens segítségével. A szakasz a tanulók pozitív visszajelzéseit mutatja be, képekkel és szövegekkel, hogy bemutassa az edzések hatékonyságát és a közösségi élményt.

## 3. **Lábjegyzet (Footer) szakasz**

A lábjegyzet a weboldal alján található. Itt találhatóak a közösségi média ikonok, amelyek az oldal különböző platformjaira (Facebook, Twitter, Instagram stb.) mutatnak. A végén található egy háttérkép is.

## 4. **JavaScript és Dinamikus Funkciók**

### 4.1 **JQuery és Bootstrap**

A weboldal interaktív elemeinek működését a JQuery és Bootstrap biztosítja. Például a navigációs menü mobil eszközökre való alkalmazkodása, illetve a slider animációi.

### 4.2 **Owl Carousel**

Az oldal két fő Owl Carousel példát tartalmaz, amelyek a különböző szakaszok (edzések, vélemények) dinamikus, görgethető listáit biztosítják. Az Owl Carousel funkciók, mint az automatikus lejátszás és a különböző elemek számának dinamikus változtatása biztosítják a reszponzív felületet.

### 4.3 **Google Maps**

A kapcsolat szakaszban található Google Maps integráció, amely megjeleníti az üzlet helyét. A JavaScript funkciók segítségével a térkép dinamikusan tölti be a weboldalon.

### 4.4 **Visszahívás Kérés Formája**

A visszahívás kérése form segítségével történik, ahol a felhasználó megadja nevét, telefonszámát és opcionálisan üzenetét. Az űrlap küldése után az adatokat a szerver felé továbbíthatja.

Class.html

Ez a dokumentáció a **class.html** nevű oldal felépítését és működését ismerteti. Az oldal a KeDa Boxing edzéseit mutatja be, valamint lehetőséget ad az edzésfoglalásra és a visszahívás kérésére.

### 1. HTML Strukturális Felépítés

A HTML dokumentum a **KeDa Boxing** weboldal edzésekkel kapcsolatos aloldalát tartalmazza. A dokumentum három fő részből áll: a fejlécekből, az edzésről szóló szekcióból, és az információs szekcióból.

#### 1.1. <head> szakasz

A **<head>** rész tartalmazza a metaadatokat, a stíluslapokat és az oldalon használt külső könyvtárak hivatkozásait. Fontos, hogy a metaadatok megfelelőek legyenek a keresőoptimalizálás (SEO) és a mobil eszközök számára.



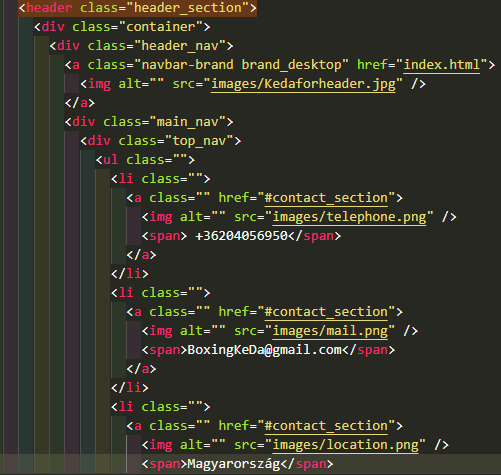
**Fontos szakszavak és magyarázatok:**

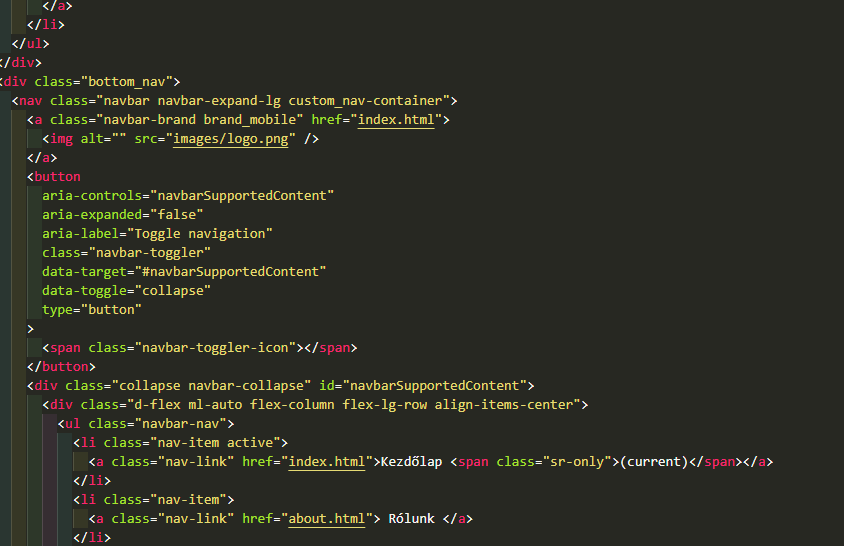
* **Meta charset="utf-8"**: Beállítja az oldal karakterkódolását UTF-8-ra, biztosítva a speciális karakterek megfelelő megjelenítését.
* **Meta viewport**: Segít optimalizálni az oldal megjelenítését mobil eszközökön a reszponzív tervezéshez.

#### 1.2. <body> szakasz

A **<body>** szakasz tartalmazza az oldal valódi tartalmát. A következő részeket találhatjuk meg itt:

* **Fejléc** (<header>): Az oldal navigációját és elérhetőségi adatokat tartalmazza.
* **Edzés szekció** (class\_section): Az edzésről szóló videókat és edzéstípusokat mutatja be.
* **Információs szekció** (info\_section): Kapcsolatfelvételi űrlapot és visszahívás kérését tartalmazza.





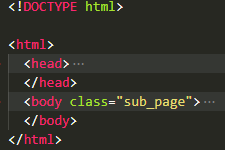


Blog.html

A blog.html fájl egy weboldal egyik aloldalát valósítja meg, amely bemutatja a "KeDa Boxing" blog legfrissebb bejegyzését. A weboldalon elhelyezett tartalmak között szerepel két fiatal bokszoló története: Szabó Marcell és Német Gábor. Az oldal több szempontból is testreszabott, beleértve a stílusokat, a navigációs menüt, valamint a közösségi média hivatkozásokat.

### 1. **HTML Alapfelépítés**

A fájl az alábbi szintaxist követi:

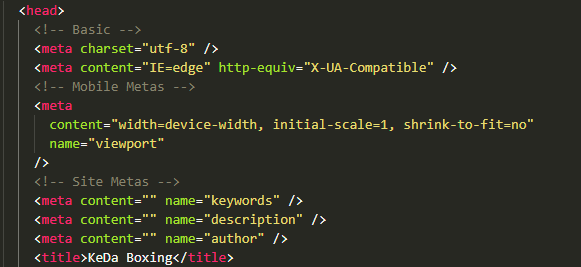


A dokumentum a HTML5 szabványt követi (<!DOCTYPE html>), és két fő részre oszlik:

* **<head>**: A fejrészt tartalmazza, ahol a metaadatok és stíluslapok találhatók.
* **<body>**: Az oldal tartalmát tartalmazó szakasz.

A <head> szakaszban találhatóak a következő elemek:

* **<meta charset="utf-8" />**: A karakterkódolás UTF-8-ra van állítva, hogy támogassa a nemzetközi karaktereket.
* **<meta content="IE=edge" http-equiv="X-UA-Compatible" />**: A böngészőnek jelzi, hogy a legújabb renderelési módot használja.
* **<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />**: A reszponzivitás érdekében a viewport beállításokat tartalmazza, amely biztosítja, hogy az oldal megfelelően jelenjen meg különböző képernyőméreteken.

Fejlécek: 

#### Külső források és stíluslapok:

A fejlécben külső CSS fájlokat és betűtípusokat is betöltünk:

* **Owl Carousel**: Egy népszerű JavaScript könyvtár, amely lehetővé teszi képes galériák és egyéb slider elemek megjelenítését.
* **Bootstrap**: A Bootstrap keretrendszer a reszponzív és mobilbarát dizájnhoz szükséges alapokat biztosítja.
* **Egyedi stíluslapok**: A projekt egyedi stílusait a css/style.css és css/responsive.css fájlok biztosítják.

### 2. **Navigáció és Menü**

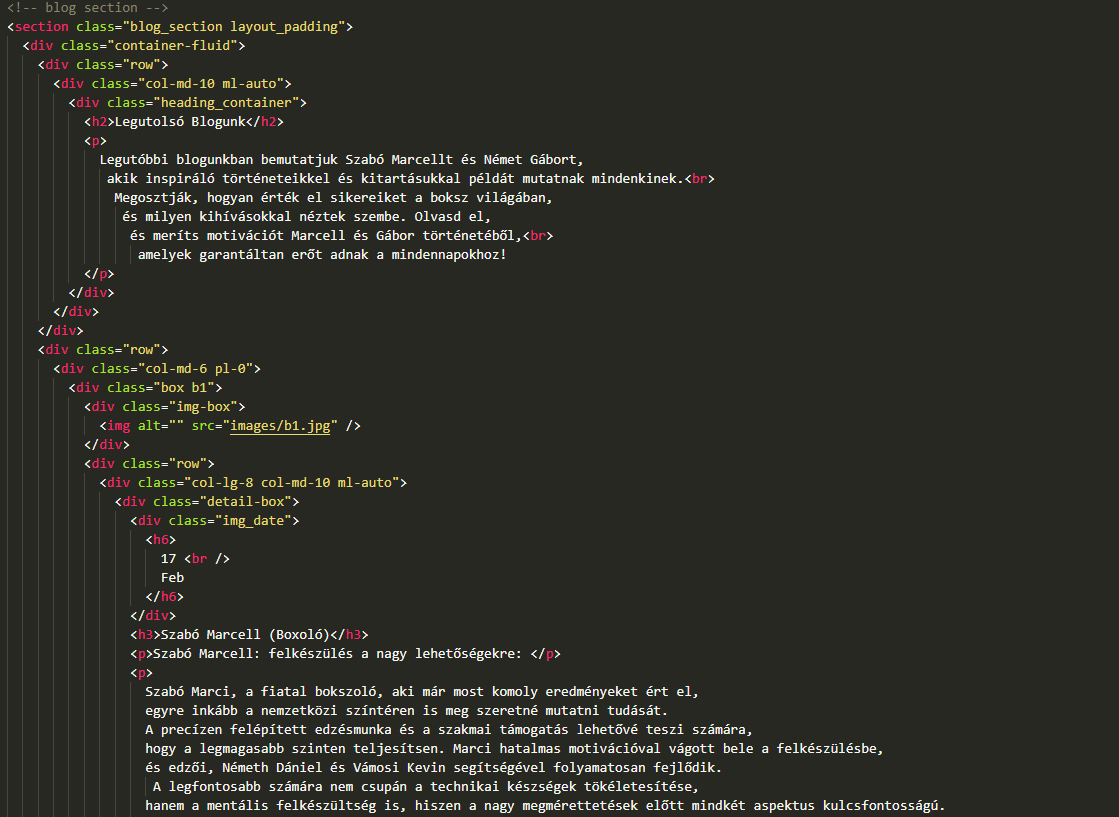
A navigációs menü az oldal tetején található, és reszponzív módon van kialakítva:

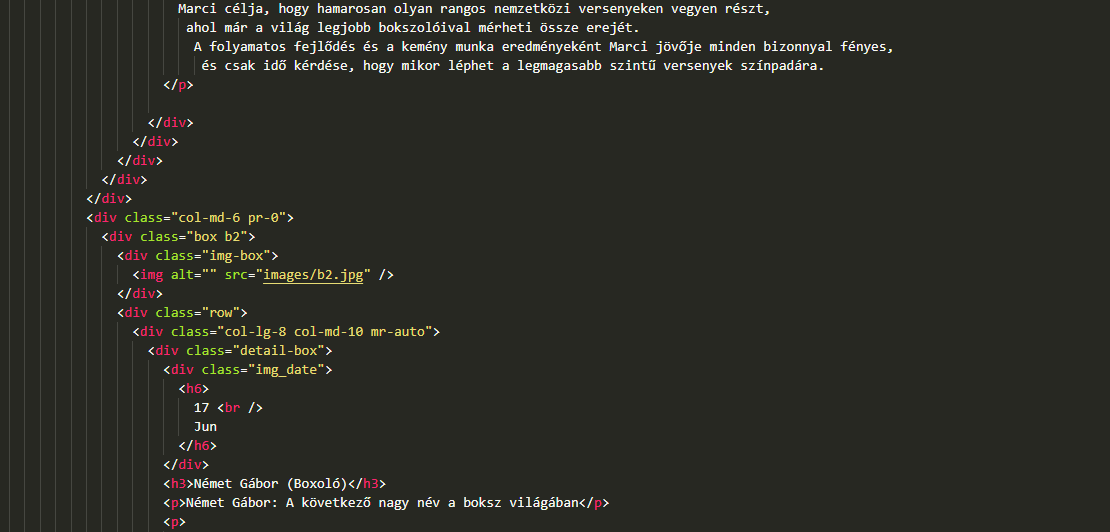


A navigációs struktúra Bootstrap komponensre épül, reszponzív, és tartalmaz egy hamburger menüt a mobil nézetekhez. A menüpontok közvetlenül az oldal más szakaszaiba (pl. kezdőlap, rólunk, blog) navigálnak. Az "Bejelentkezés" és "Profil" menüpontok dinamikusan változnak a felhasználói állapot alapján.

### 3. **Blog Bejegyzés**

A blog bejegyzéseket a következő szakasz tartalmazza:





Ez a szakasz két külön bejegyzést mutat be: Szabó Marcell és Német Gábor történetét. Mindkét bejegyzés tartalmaz egy képet (img-box), egy címkét (h3), egy rövid leírást és egy hosszabb szöveget. Az elrendezés a Bootstrap grid rendszerén alapul, két oszlopra bontva.

### 4. **Térkép és Kapcsolat**

A térkép a következőképpen van implementálva:



A Google Maps API használatával kerül betöltésre egy térkép, amely egy meghatározott koordinátára (Budapestre) van központosítva. Az API kulcsot a <script> tag tartalmazza, amely az adott térképet jeleníti meg.

### 5. **Közösségi Média és Lábléc**

A lábléc a közösségi média ikonjaival rendelkezik, amelyeket az alábbiakban találunk:



A közösségi média linkek képei mutatják az oldal számára az elérhető platformokat, például Facebookot, Twittert, LinkedIn-t, Instagramot és YouTube-ot.

### 6. **JavaScript és Egyéb Funkciók**

A JavaScript kódok különböző funkcionalitásokat biztosítanak:

* **Owl Carousel**: Képes galériák és slider elemek vezérlésére.
* **Google Maps API**: A térkép betöltése és interaktív kezelése.
* **Responsive Menü**: A reszponzív menükezelés a mobil nézetekhez.

Admin.html

#### 1. **Áttekintés**

Az adminisztrációs oldal célja, hogy a weboldal adminisztrátorai könnyedén kezelhessék a visszahívási kéréseket. Az oldal lehetőséget ad a kérések megtekintésére, státusz módosítására, és exportálására CSV fájlba. A felhasználóbarát felület egyszerű navigációt biztosít, amely lehetővé teszi az adminisztrátorok számára a kérések kezelését.

#### 2. **Oldal Felépítése**

Az oldal struktúrája három fő részből áll:

* **Fejléc (Header)**
* **Törzs (Body)**: A visszahívási kérések táblázatának és az exportálási funkcióknak ad helyet.
* **Lábléc (Footer)**: A weboldal készítőjének és jogi információinak feltüntetése.

### 3. **HTML Szerkezet**

A HTML kód az alapvető weboldal struktúrát és tartalmat biztosítja. Az oldal különböző CSS fájlokat és JavaScript fájlokat is betölt, amelyek segítenek a stílus és interakciók kezelésében.

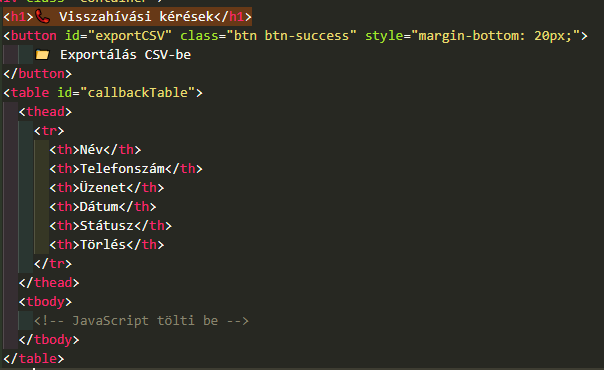
##### Fejléc (Header) szakasz

A fejléc tartalmazza a navigációs menüt és a kapcsolatfelvételi információkat. Az alábbi kód részlet például a navigációs menüt jeleníti meg:



##### Törzs (Body) szakasz

A törzsben található a visszahívási kérések táblázata és az exportálás gomb. A táblázatban az adatok dinamikusan jelennek meg a JavaScript segítségével. A gomb segítségével CSV fájlba exportálhatjuk a kéréseket.



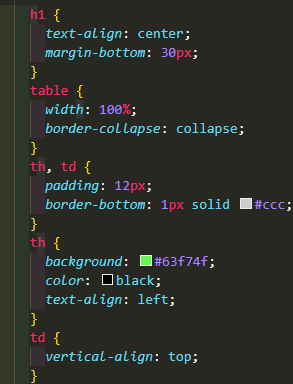
##### Lábléc (Footer)

A lábléc tartalmazza a weboldal készítőit és a jogi információkat:



### 4. **CSS Stílusok**

A CSS stílusok felelősek az oldal vizuális megjelenéséért. Az alábbi kód például a táblázat és az alapvető szövegformázás megjelenítéséért felelős:



### 5. **JavaScript Funkciók**

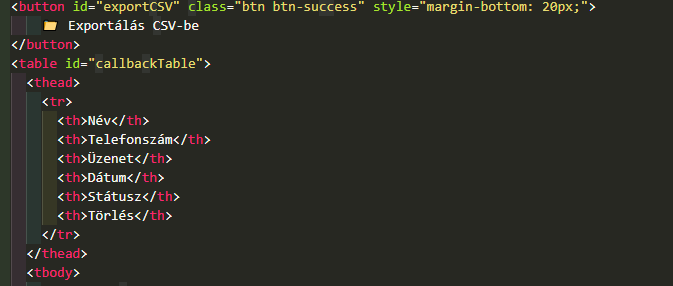
#### 5.1. **Visszahívási Kérések Táblázatának Dinamikus Betöltése**

A JavaScript (admin.js) felelős a táblázat dinamikus feltöltéséért a backend vagy a statikus adatforrásokból. A következő kódrészlet például a táblázat feltöltését végzi:



#### 5.2. **CSV Exportálás Funkció**

A "📁 Exportálás CSV-be" gomb működését a következő kódrészlet valósítja meg. Ez a funkció lehetővé teszi a visszahívási kérések CSV fájlba történő exportálását:



### 6. **Bővítési Lehetőségek**

* **Törlés Funkció**: A kód tartalmaz egy "Törlés" gombot, amely lehetővé teszi a visszahívási kérések törlését, de a törlés logikáját még implementálni kell.
* **Státusz Módosítás**: Jelenleg csak a kérés státusza van megjelenítve, de az adminisztrátorok számára hasznos lehetőség lenne a státusz módosításának lehetősége is.
* **Adatok Szűrése**: A táblázat szűrési lehetőséget is kaphat, hogy az adminisztrátorok gyorsabban megtalálják a keresett kéréseket.

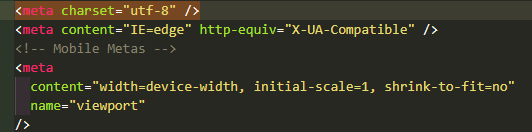
About.html

Ez a dokumentáció a **KeDa Boxing** weboldal **"Rólunk"** szekciójához kapcsolódik. Az oldal célja, hogy bemutassa a KeDa Boxing vezető edzőit, azok sportpályafutását és edzői filozófiáját. Továbbá tartalmazza az edzőterem elérhetőségeit és a közösségi média hivatkozásokat, valamint egy Google Map szegmenst, amely segíti a látogatókat a helyszín könnyű megtalálásában.

### 1. **HTML Szerkezet és Metaadatok**

#### **1.1 Alapvető Metaadatok**

A HTML dokumentum az alapvető metaadatokat tartalmazza a megfelelő karakterkódolás, a weboldal kompatibilitásának biztosítása és a mobil eszközökre való optimalizálás érdekében:



A metaadatok további részletezésével (például kulcsszavak és leírások) a SEO optimalizálásának biztosítása érdekében.

#### **1.2 Linkek és Stíluslapok**

Az oldal a következő külső és belső stíluslapokat alkalmazza:

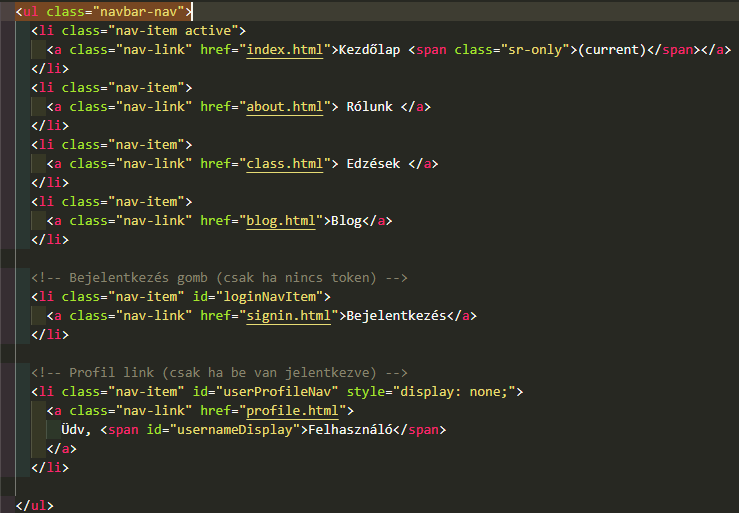
* **Owl Carousel** CSS a képgaléria funkcióhoz.
* **Bootstrap** CSS a reszponzív elrendezéshez.
* Egyedi stíluslapok (style.css, responsive.css), amelyek a weboldal kinézetét és érzését alakítják.

### 2. **Navigációs és Fejléc Szekció**

#### 2.1 **Fejléc**

A fejléc tartalmazza az oldal logóját, a kapcsolati információkat (telefonszám, email, cím) és a navigációs menüt. A navigációs menü az alábbi linkekből áll:

* Kezdőlap
* Rólunk
* Edzések
* Blog
* Bejelentkezés / Profil (dinamikusan jelenik meg a felhasználói állapot szerint)



#### 2.2 **Interaktív Navigáció**

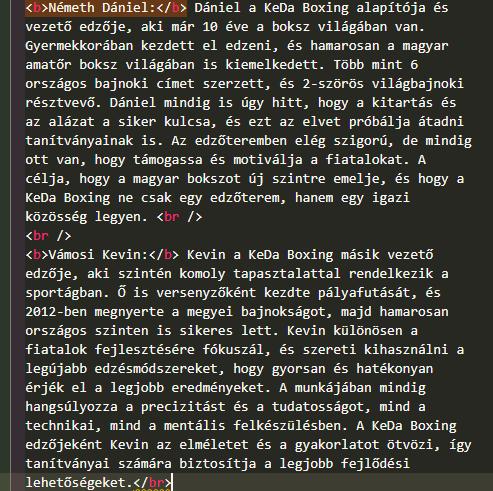
A weboldal reszponzív navigációt alkalmaz, amely mobilnézetekre optimalizált, és tartalmaz egy hamburger menüt a kisebb képernyőkön.

### 3. **A KeDa Boxing Edzői Bemutatása**

#### 3.1 **Edzők és Pályafutásuk**

A "Rólunk" szekció bemutatja a két vezető edzőt, Németh Dánielt és Vámosi Kevint, akik a KeDa Boxing alapítói. Minden edző részletes információkat kapott a pályafutásáról, eredményeiről és edzői filozófiájáról.

##### Példa:



A két edző filozófiája és közös munkája segít kialakítani a KeDa Boxing edzéseit, amelyeket nem csupán sportként, hanem közösségi élményként is kezelnek.

#### 3.2 **Link az Edzésekhez**

Az edzők bemutatása alatt található egy link, amely az edzésekről ad információt:



### 4. **Helyszín és Elérhetőségek**

#### 4.1 **Google Map Integráció**

A weboldal tartalmaz egy Google Map beágyazást, amely segíti a látogatókat az edzőterem helyszínének megtalálásában. A térkép közvetlenül a következő script segítségével töltődik be:



A térkép középpontja a Budapest (47.497912, 19.040235) koordinátákra van állítva.

#### 4.2 **Kapcsolati Információk**

A kapcsolati információk a fejlécben találhatóak, és elérhetők az oldalon. Tartalmazzák a telefonszámot, email címet, és a címet:



### 5. **Lábléc és Közösségi Média Hivatkozások**

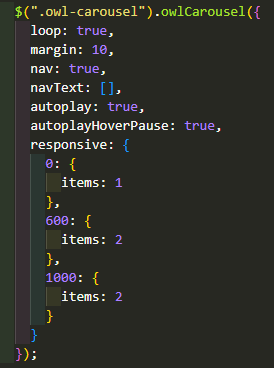
A lábléc tartalmazza a **KeDa Boxing** társadalmi kapcsolatait, valamint az oldalt készítő személyeket. A közösségi média ikonok (Facebook, Instagram, Twitter, LinkedIn, YouTube) segítenek a látogatók számára a közvetlen kapcsolatfelvételben.



### 6. **Kiegészítő JavaScript Funkciók**

A dokumentum végén található JavaScript kódok segítenek az interaktív funkciók működtetésében:

* **Owl Carousel** a képek és szövegek dinamikus megjelenítéséhez.
* **Google Maps API** a térkép betöltéséhez.
* **Dinamikus menü megjelenítés** bejelentkezett felhasználók számára.



Profile.html

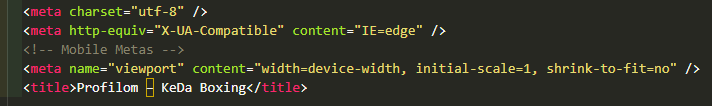
A profile.html fájl a KeDa Boxing weboldal profilkezelő oldalát tartalmazza. A felhasználók ezen az oldalon kezelhetik profiljukat, módosíthatják jelszavukat, megtekinthetik foglalásaikat, és kijelentkezhetnek az alkalmazásból. Az oldal tartalmazza a felhasználói fiók információit, jelszóváltoztatási lehetőséget és egy foglalásokat listázó táblázatot, valamint egy adminisztrátori panel elérhetőségét, ha a felhasználó admin jogokkal rendelkezik.

### Fájl felépítése

* **HTML szerkezet**: A fájl HTML5 alapú, és tartalmazza a szükséges metaadatokat, stílusokat, valamint az oldal funkcionális elemeit, mint például a profil beállításokat és a foglalások listáját.
* **CSS stílusok**: Az oldalhoz tartozó stílusokat a <style> blokkban találjuk, míg a külső CSS fájlok a vizuális megjelenést szabályozzák.
* **JavaScript**: A fájl tartalmaz JavaScript-et is a dinamikus interakciókhoz, mint például a jelszóváltoztatás formának kezelése és az admin panel megjelenítése.

## Részletes kódmagyarázat

### 1. Metaadatok

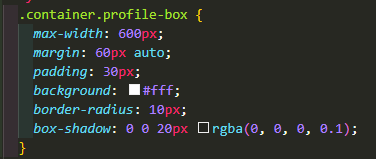


* **charset="utf-8"**: A karakterkódolás beállítása, biztosítva ezzel a nemzetközi karakterek (pl. magyar ékezetek) helyes megjelenését.
* **meta http-equiv="X-UA-Compatible" content="IE=edge"**: A Microsoft Internet Explorer böngésző számára biztosítja az oldalt modern módon történő renderelését.
* **meta name="viewport"**: A mobil eszközökön történő optimalizálás érdekében, hogy az oldal rugalmasan alkalmazkodjon a képernyő méretéhez.
* **title**: Az oldal címét adja meg, amely a böngésző fülén jelenik meg.

### 2. Stílusok

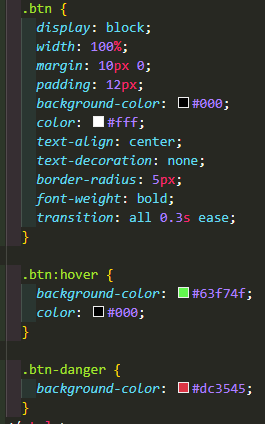
A stílusok a belső <style> tag-ben találhatóak, és a következő elemeket formázzák:

#### a) **Profil box**



Ez a szakasz formázza a profil beállítások panelt (profile-box). A max-width: 600px; beállítás limitálja a maximális szélességet, míg a margin: 60px auto; középre helyezi a panelt a képernyőn. A box-shadow és a border-radius modern, lekerekített kinézetet adnak.

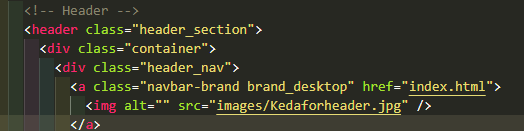
#### b) **Gombok stílusa**



A gombok a .btn osztállyal vannak stílusozva, amelyek szélességét 100%-ra állítják, így mindig kitöltik a rendelkezésre álló helyet. A transition tulajdonság biztosítja, hogy a gomb színe zökkenőmentesen változzon a felhasználói interakciók során. A .btn-danger osztály piros színt ad a kijelentkezés gombnak.

### 3. HTML Struktúra

#### a) **Fejléc**



A fejléc tartalmazza a navigációs sávot, amely hivatkozásokat biztosít a weboldal többi szekciójához. Az a tagek a navigációs linkek, míg a navbar-toggler egy mobilnézethez optimalizált gomb, amely lehetővé teszi a navigációs menü elrejtését és megjelenítését.

#### b) **Profil információk és jelszó módosítás**



* **Felhasználónév és jelszó módosítás**: Az oldal központi része, ahol a felhasználó megtekintheti és módosíthatja a profilját. A form elem tartalmazza a három jelszó mezőt (Régi jelszó, Új jelszó, Új jelszó megerősítése), és a gombok segítségével a felhasználó módosíthatja a beállításait.
* **Kijelentkezés gomb**: A .btn-danger osztály segítségével piros gombot kap a kijelentkezés link.

#### c) **Foglalások táblázat**



* A felhasználó foglalásait listázza ki egy táblázatban. A <tbody> elembe a JavaScript tölti be a foglalásokat dinamikusan. A táblázat segítségével a felhasználó megtekintheti a foglalt edzéseit és törölheti őket, ha szükséges.

### 4. JavaScript



A profil oldal JavaScript fájlja, amely kezeli az interaktív funkciókat, például:

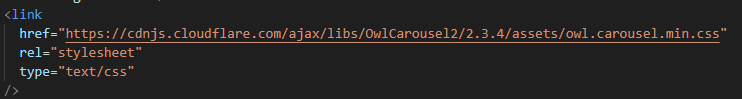
* **Jelszó módosítás**: A jelszó változtatási űrlap elküldésekor a JavaScript végzi el a szükséges érvényesítést és kérést a szerver felé.
* **Foglalások dinamikus betöltése**: Az oldal dinamikusan tölti be a felhasználó foglalásait az adatbázisból.

Register.html

A register.html fájl egy egyszerű HTML regisztrációs űrlapot tartalmaz, amely felhasználók számára lehetővé teszi a fiók létrehozását az oldalon. Az űrlap biztosítja a szükséges adatokat, mint a név, felhasználónév, e-mail cím, jelszó és annak megerősítése. Az oldal tartalmazza a navigációs sávot, a regisztrációs űrlapot, valamint egy láblécet.

### HTML Szerkezet

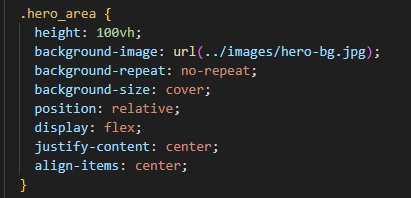
1. **HTML Fejléc** (<head>):
   * A <meta charset="UTF-8"> a karakterkódolást állítja be UTF-8-ra, biztosítva, hogy a speciális karakterek (pl. magyar ékezetek) helyesen jelenjenek meg.
   * A <meta name="viewport" content="width=device-width, initial-scale=1.0"> sor a mobil eszközökön történő helyes megjelenítést segíti.
   * A különböző stíluslapok (<link> elemek) külső CSS fájlok betöltésére szolgálnak, mint például a Bootstrap, a Google Fonts és a saját stíluslapok.



Ez a rész biztosítja, hogy a megfelelő külső CSS fájlok betöltődjenek az oldalra, biztosítva ezzel a dizájn reszponzív és esztétikus megjelenését.

1. **Stílusok a <style> blokkon belül**:
   * A belső CSS szabályok a body, .hero\_area, .signin-container, és más osztályok megjelenését és elrendezését határozzák meg.
   * A .hero\_area osztály például az oldal háttérképét és a központi elhelyezést szabályozza, míg a .signin-container az űrlapok számára biztosít egy tiszta és rendezett elrendezést.

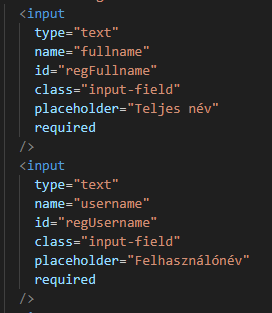
#### Példa:



Ez a szabály biztosítja, hogy a háttérkép kitöltse a teljes képernyőt, és a regisztrációs űrlap középen helyezkedjen el.

1. **A regisztrációs űrlap**:
   * A regisztrációs űrlap form elemet tartalmaz, amely a felhasználó adatainak begyűjtésére szolgál.
   * A input mezők mindegyike required attribútummal van ellátva, biztosítva, hogy a felhasználó minden szükséges adatot megadjon a regisztrációs folyamat során.
   * A regisztrációs gomb a type="submit" értékkel van ellátva, tehát a felhasználó kattintására az űrlap adatokat elküldi.

#### Példa:



Ezek az input mezők azokat az adatokat gyűjtik, amelyeket a felhasználó megad a regisztrációs folyamat során (például teljes név, felhasználónév).

### JavaScript

1. **Űrlap validálása**: A fájl tartalmaz egy hivatkozást egy külső JavaScript fájlra, a js/auth.js-t, amely valószínűleg a regisztrációs űrlap validálásáért felelős. Az űrlap mezőinek ellenőrzésére használható a JavaScript, például a jelszavak egyezőségének biztosítása.

#### Példa:

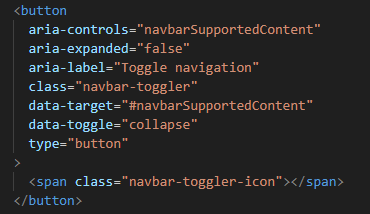


Ez a sor biztosítja, hogy a auth.js fájl betöltődjön, ami kezelheti a regisztrációs űrlap érvényesítését.

### Navigációs Sáv

A navigációs sáv lehetővé teszi a felhasználók számára, hogy navigáljanak az oldalon, például a főoldalra, a blogra, az edzésekhez vagy a bejelentkezéshez. A navbar-toggler biztosítja, hogy mobil eszközökön a menü legördülhessen, biztosítva a reszponzív dizájnt.

#### Példa:

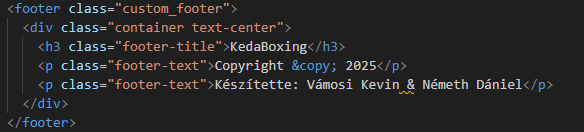


Ez a kód biztosítja a legördülő menüt mobil nézetben.

### Lábléc

A lábléc tartalmazza a cég nevét és a szerzői jogi információkat. Emellett a láblécben a weboldal készítői, Vámosi Kevin és Németh Dániel is feltüntetésre kerültek.

#### Példa:



Ez a rész tartalmazza a weboldal jogi információit, és biztosítja a megfelelő lábléc megjelenítését.

Signin.html

A login.html fájl egy bejelentkezési űrlapot tartalmaz, amely lehetővé teszi a felhasználók számára, hogy bejelentkezzenek az alkalmazásba. Az oldal HTML kódja a bejelentkezéshez szükséges input mezőket és gombot tartalmaz, és egyaránt van lehetőség regisztrálni, ha a felhasználónak nincs fiókja.

#### 1. **HTML struktúra és alapvető felépítés**

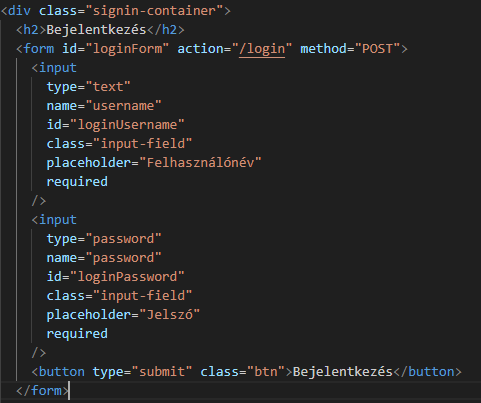
A fájl a következő HTML elemeket tartalmazza:

* **<head> szekció**:
  + A karakterkódolás és a viewport beállítások meghatározásával biztosítja a megfelelő működést különböző eszközökön.
  + Stíluslapok és külső CSS fájlok, például a Bootstrap és egyéni CSS fájlok linkelése.
  + A cím (<title>) meghatározása, amely a böngésző tabján jelenik meg.



**<body> szekció**:

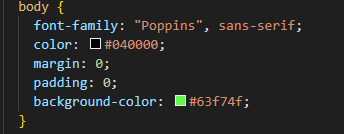
* A bejelentkezési űrlap és a navigációs menü található itt, amely a felhasználói interakciók kezelésére szolgál.
* A bejelentkezési felületet egy "signin-container" div foglalja magába, amely központosan helyezkedik el a képernyőn.



#### 2. **Formázás és stílusok**

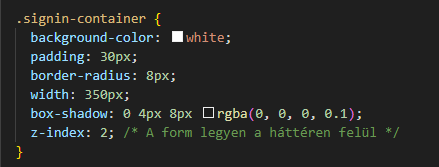
A weboldalon a következő CSS stílusok segítenek az oldal megjelenésének kialakításában:

* **body stílus**:
  + A betűtípus beállítása a "Poppins" fontra, a háttérszín zöld (#63f74f), amely a vizuális megjelenést javítja.



**signin-container stílus**:

* A formázásnak köszönhetően a bejelentkezési űrlap középen jelenik meg a képernyőn, és árnyékolt hátteret kap, hogy kiemelkedjen a háttérből.

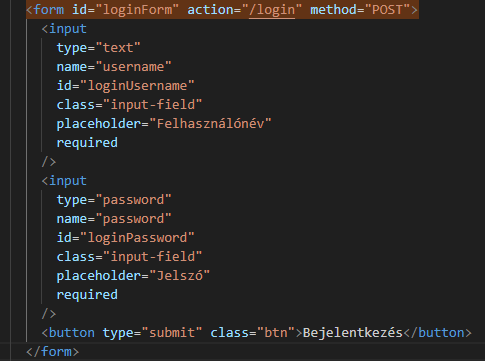


#### 3. **Bejelentkezési űrlap**

A bejelentkezési űrlap a loginForm form tagbe van ágyazva, amely az alábbi mezőket tartalmazza:

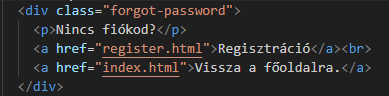
* **Felhasználónév (username)**: Egy kötelező szöveges input mező, amely a felhasználó felhasználónevét kéri.
* **Jelszó (password)**: Egy kötelező jelszó mező, amely lehetővé teszi a felhasználó számára, hogy biztonságos módon adja meg a jelszavát.

A bejelentkezési form elküldése a **POST** metódus segítségével történik az /login URL-re.



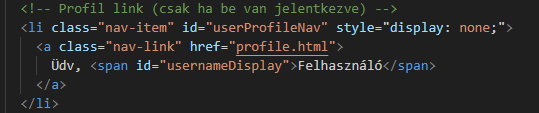
#### 4. **További interakciók**

* **Regisztrációs link**: Az oldal tartalmaz egy hivatkozást, amely lehetővé teszi a felhasználók számára, hogy regisztráljanak, ha még nincs fiókjuk.



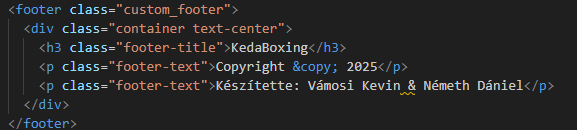
#### 5. **Fejléc és navigációs menü**

A navigációs menü a header\_section div-en belül helyezkedik el. A menü dinamikusan változik attól függően, hogy a felhasználó be van-e jelentkezve. Ha nincs bejelentkezve, akkor a "Bejelentkezés" link jelenik meg. Ha a felhasználó be van jelentkezve, akkor a profilja jelenik meg.



#### 6. **Lábjegyzet**

A lábjegyzet tartalmazza a weboldal készítőjét és a copyright információt, amely az alkalmazás alján helyezkedik el.



#### 7. **JavaScript**

Az oldal JavaScript fájlokat is tartalmaz, amelyek a bejelentkezéshez és a navigációs menühöz kapcsolódnak. A login.js fájl valószínűleg az űrlap érvényesítését, a bejelentkezési folyamatot vagy más logikai műveleteket tartalmaz.



Responsive.css

A **responsive.css** fájl célja, hogy az oldal stílusait reszponzív módon, különböző képernyőméretekhez alkalmazza. A CSS @media lekérdezések használatával biztosítja, hogy a felhasználói élmény optimális maradjon mobiltelefonokon, tableteken, valamint asztali eszközökön. A fájl az alábbi különböző képernyő szélességeket célozza meg: **1250px**, **1120px**, **992px**, **767px**, **576px**, **480px**, **400px**, és **360px**.

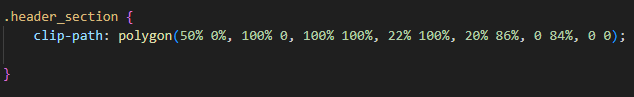
#### 1. **@media (max-width: 1250px)**

Ez a media query 1250px szélesség alatti eszközökre alkalmazza a következő stílusokat:

* **.header\_section::before**: A háttér elem szélességét 75%-ra csökkenti, hogy a fejléc részét jobban illeszkedjen a kisebb képernyőkhöz.



**.header\_section**: A fejléc alakját egy sokszögre (clip-path: polygon) formálja, így a fejléc egyedi, nem hagyományos éleket kap. Ez vizuálisan hangsúlyosabbá teszi a területet.



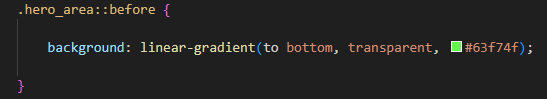
#### 2. **@media (max-width: 992px)**

Ez a lekérdezés a 992px vagy kisebb szélességű eszközökre vonatkozik, jellemzően táblagépekre és kisebb képernyőkre.

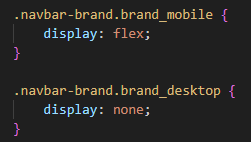
* **.hero\_area**: A hősáv (hero section) magasságát dinamikusra állítja, hogy a szöveges és képi tartalom a kisebb képernyőkön is megfelelően jelenjen meg.



**.hero\_area::before**: A háttér átlátszó színátmenetet kap, hogy jobban illeszkedjen az oldal zöld színéhez.



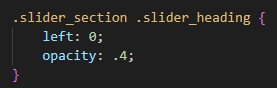
**.navbar-brand (Mobil verzió)**: A logó, amely korábban csak asztali nézetben volt látható, most már mobil nézetben is megjelenik (display: flex), míg az asztali verziót elrejtjük (display: none).



#### 3. **@media (max-width: 767px)**

Ez a query kifejezetten a kisebb mobil eszközökre van optimalizálva (tipikusan okostelefonok).

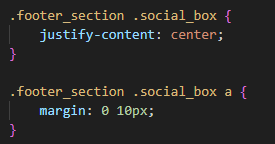
* **.slider\_section .slider\_heading**: A szöveges rész elhelyezkedését és átlátszóságát módosítja, hogy jobban illeszkedjen a kisebb képernyőkhöz. Itt a pozícióját balra állítja és csökkenti az átlátszóságot.



**.info\_section .info\_main-row**: Az információk elrendezését függőlegesen változtatja, hogy jobban illeszkedjen a kisebb eszközökhöz.



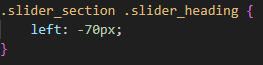
**.footer\_section .social\_box**: A közösségi ikonok középre igazítása és közötti margók beállítása segíti, hogy jobban nézzenek ki a kisebb képernyőkön.



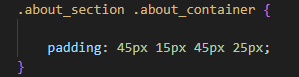
#### 4. **@media (max-width: 576px)**

Ez a lekérdezés a kis okostelefonokra van optimalizálva.

* **.slider\_section .slider\_heading**: A szöveges elem elhelyezkedését tovább finomítja, hogy jobban igazodjon a képernyőhöz.



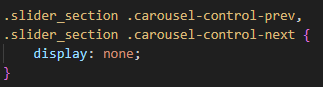
**.about\_section .about\_container**: A padding értékét módosítja, hogy a tartalom jobban illeszkedjen a kisebb kijelzőkhöz.



#### 5. **@media (max-width: 400px)**

Egy extrém esetet kezel, amikor a képernyő szélessége 400px vagy kisebb.

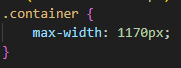
* **.slider\_section .slider\_heading**: A szöveges elem pozícióját és átlátszóságát drámaian módosítja, hogy a szöveg elférjen az ilyen kicsi képernyőkön.



#### 6. **@media (min-width: 1200px)**

Ez a query az asztali eszközökre vonatkozik, és biztosítja, hogy a tartalom ne legyen szélesebb, mint a képernyő szélessége.

* **.container**: A konténer maximális szélességét 1170px-re korlátozza, hogy az oldal jól nézzen ki nagyobb képernyőkön is.



Style.css

Ez a CSS fájl a weboldal vizuális megjelenését és elrendezését szabályozza, amely több különböző szekcióra van felosztva, mint például a fejléc, slider, about, osztályok, blogok és ügyfél visszajelzések. A fájl modern dizájn technikákat alkalmaz, mint például a flexbox és clip-path, hogy dinamikus és reszponzív felhasználói élményt biztosítson.

### 1. **Alap stílusok**

* **Betűtípus**: A weboldal alapértelmezett betűtípusa a Poppins, egy tiszta és modern sans-serif font.
* **Színpaletta**: A fő szövegszín #040000, míg a háttérszín zöld árnyalata #8BC34A, amely friss és természetes hangulatot ad az oldalnak.

### 2. **Szekciók elrendezése és pozicionálása**

* **Flexbox és Grid**: A fájl széleskörűen alkalmazza a flexbox elrendezést az elemek igazításához és elhelyezéséhez. Minden fő szekció (pl. .heading\_container, .slider\_section) flexbox alapú elrendezést használ a tartalom középre igazítására és a választható irányokban való elrendezésére.
* **Előre beállított padding és margók**: A .layout\_padding, .layout\_padding2, stb. osztályok biztosítják a megfelelő belső térközöket, amelyek elősegítik a vizuális tisztaságot és a megfelelő szellőzést a szekciók között.

### 3. **Fejléc szekció**

* **Clip-path**: A fejlécben található .header\_section egyedi formáját a clip-path tulajdonság adja, amely egy többszög alakú kivágást eredményez a háttér színén.
* **Navigációs menü**: A navigációs menü a flex elrendezést használja, hogy az elemek rugalmasan és középre igazodva jelenjenek meg. Az aktív linkek és gombok színei változnak a felhasználói interakciók hatására (hover).

### 4. **Slider szekció**

* **Carousel**: A .slider\_section reszponzív, teljes képernyős háttérképpel rendelkezik, amely a képek rotálására szolgál a carousel komponens segítségével. A navigációs nyilak (.carousel-control-prev, .carousel-control-next) és az indikátorok a felhasználói élményt segítik, és testreszabott képeket használnak a gombokhoz.
* **Szöveges animációk**: A szövegek, például a .slider\_heading h2, különleges rotációval és tükrözéssel (transform: rotate) jelennek meg a képernyőn, amely dinamikus megjelenést biztosít.

### 5. **Részletek és tartalom**

* **Szövegek és gombok**: A szövegek színét és stílusát folyamatosan testreszabják a háttérrel való kontraszt maximalizálása érdekében. A gombok (.btn-1, .btn-box) minden szekcióban kulcsfontosságúak a felhasználói interakciókhoz, és vizuálisan vonzóvá válnak a színek változásával a hover állapotokban.
* **Reszponzivitás**: A dizájn reszponzív, azaz a képek és szövegek megfelelően skálázódnak az eszközök szélességétől függően. Az elemeken belüli képek (img) 100%-os szélességre vannak állítva, hogy a tartalom minden képernyőn jól megjelenjen.

### 6. **Speciális stílusok**

* **Ügyfélszövegek és logók**: A .client\_section szekcióban az ügyféltámogatási képek kör alakúak, amelyeket border-radius: 100% biztosít, így a vizuális megjelenés dinamikus és esztétikus.
* **Blog bejegyzések**: A blogok képei (.blog\_section .box .img-box img) rugalmasan alkalmazkodnak a képernyőmérethez, és az oldalszélek között elegendő margóval rendelkeznek, hogy a szövegek ne torlódjanak.

Ez a CSS fájl egy weboldal különböző szekcióinak megjelenését és stílusát szabályozza. A fájl felöleli az oldal struktúrájának és elemeinek stílusát, beleértve a navigációs menüt, a fejlécet, a slider-t, a különböző szekciókat (pl. ügyfél szekció, információs szekció, blog szekció), valamint az oldalsó navigációt és a láblécet.

### Főbb szekciók és stílusok

#### 1. **Általános Stílusok**

* **Alapértelmezett betűtípus**: A weboldalon a "Poppins" betűtípus van beállítva, amely sans-serif stílust biztosít a szövegeknek.
* **Háttérszín**: A háttér színét a #8BC34A hex kóddal zöldre állították, ami egy friss, természetes hatást kelt.
* **Szöveg szín**: Az alap szövegszín #040000, ami szinte fekete árnyalatú.

#### 2. **Fejléc (Header)**

* A **fejléc** szekció háttérszíne fehér, de egyedi polygonális klipszel ellátott formával rendelkezik, amely vizuálisan kiemeli a szekciót.
* A **navigációs menü** rugalmasan van kialakítva a flexbox segítségével, az elemek középre rendeződnek, és a linkek színükben fehérre vannak állítva.
* **Mobil verzió**: A mobil verziónál a navigációs menü hamburger ikonnal válik láthatóvá, és a menü ikonok is a háttérszíntől függően változnak.

#### 3. **Slider szekció**

* A slider háttérként egy teljes képernyős kép van beállítva, amely a background-image és background-size: cover segítségével dinamikusan alkalmazkodik a képernyőhöz.
* A navigációs nyilak (.owl-prev, .owl-next) egyedi hátteret és alakot kaptak, és az interakciókhoz kifejezetten meg van adva egy transform és background-position.

#### 4. **Különböző Szekciók**

* **Ügyfélszekció (Client Section)**: Az ügyfélszekcióban a boxok közötti elrendezés flexbox alapú, és a képek kör alakúvá vannak formázva (border-radius: 100%).
* **Információs szekció (Info Section)**: A űrlapok színei fekete háttéren fehér szöveggel jelennek meg, a beviteli mezőkben a helykitöltő szövegek színe halvány szürke (#737272).
* **Blog szekció**: A blog bejegyzésekhez egyéni dobozok és képek vannak rendelve. A dátumot kör alakú háttérrel jelenítjük meg, kiemelve a szövegeket.
* **Lábléc (Footer)**: A lábléc háttérszíne sötét (fekete), és zöld színnel kiemelt szövegekkel rendelkezik. A közösségi média ikonjai flexbox-szal vannak elrendezve, és az ikonok mérete 25px.

#### 5. **Részletek és Interakciók**

* **Hover effektusok**: Az interaktív elemek, mint a gombok, a hover állapotnál változnak, például a gombok háttérszíne megváltozik és a szövegszín is átfordul a háttérhez illően.
* **Reszponzív viselkedés**: A legtöbb elem és szekció reszponzív, így alkalmazkodik különböző képernyőméretekhez. Az elemek, mint a slider, a navigáció és a gombok, mind a képernyő szélességétől függően átrendeződnek.

#### 6. **Elrendezés és Flexbox**

* A fájl nagy része a flexbox modellt használja a különböző szekciók elemeinek rugalmas és dinamikus elrendezésére. Az elrendezési irányokat (flex-direction, justify-content, align-items) a különböző szekciókhoz igazítják a kívánt vizuális hatás elérése érdekében.

#### 7. **Speciális Stílusok**

* **Gombok**: A gombok mindkét állapotában (normál és hover) világos és sötét színváltásokat használnak, hogy a felhasználói interakciók vizuálisan is kiemeltek legyenek.
* **Árnyékok és hatások**: Az árnyékok és egyéb vizuális hatások, mint például a box-shadow, finoman használva vannak az interaktív elemek körül, hogy fokozzák azok vizuális megjelenését.

Admin.js

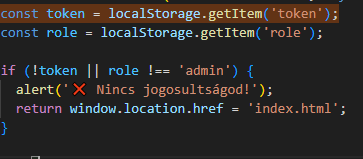
Az admin.js fájl egy adminisztrátori felület kezelésére szolgáló JavaScript kódot tartalmaz, amely lehetővé teszi az adminisztrátorok számára, hogy megtekintsék, módosítsák és töröljék a visszahívási adatokat, valamint exportálják őket CSV formátumban. A kód használata során fontos szerepe van a felhasználói jogosultságok ellenőrzésének, az adatok dinamikus betöltésének és kezelésének, valamint a felhasználói interakciók megfelelő kezelésének.

## Főbb Funkciók

1. **Jogosultság-ellenőrzés:** Az oldal betöltődésekor az adminisztrátori jogosultságot ellenőrzi a rendszer. Ha a felhasználó nincs bejelentkezve vagy nincs admin szerepe, átirányítja őt a kezdőlapra, és egy figyelmeztető üzenetet jelenít meg.
2. **Visszahívások listázása:** Az adminisztrátor számára elérhetővé válik a visszahívások listája, amely tartalmazza a visszahívásokat és azok állapotát.
3. **Státusz frissítése:** Az adminisztrátor módosíthatja a visszahívások státuszát (pending vagy done).
4. **Törlés funkció:** Az adminisztrátor törölheti a visszahívásokat az adatbázisból.
5. **CSV exportálás:** Lehetőség van a visszahívások adatait CSV fájl formátumban történő exportálásra.

## Kód Magyarázata

### Jogosultság-ellenőrzés

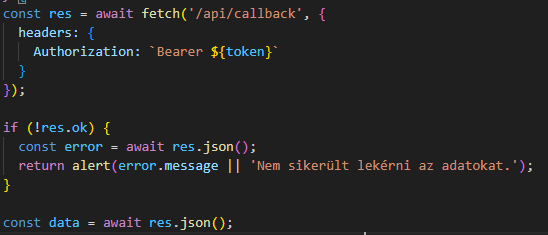


A kód első része ellenőrzi, hogy a felhasználó rendelkezik-e érvényes token-nel és adminisztrátori jogosultsággal (role === 'admin'). Ha bármelyik feltétel nem teljesül, az adminisztrátort átirányítja a kezdőlapra (index.html), és egy figyelmeztető üzenetet jelenít meg.

**Példa**:

* Ha egy felhasználó, aki nem adminisztrátor, megpróbálja elérni az adminisztrátori felületet, akkor a böngésző automatikusan visszairányítja őt a index.html oldalra.

### Visszahívások lekérése és megjelenítése



A kód fetch API-t használ az /api/callback végpontról való adatlekéréshez. Az adatokat JSON formátumban kérjük le, és a válasz státusza alapján kezeljük a hibaüzeneteket.

**Példa**:

* Ha a szerver nem válaszol megfelelően, egy hibaüzenet jelenik meg, amely informálja a felhasználót, hogy nem sikerült lekérni az adatokat.

A válaszból kapott adatokat a kód HTML elemekbe (a táblázatba) jeleníti meg:



**Példa**:

* A data.forEach ciklus minden visszahívás adatát egy táblázat sorba (<tr>) helyezi, és beállítja az adatokat a megfelelő táblázatos cellákba (<td>).

3. Státusz módosítás

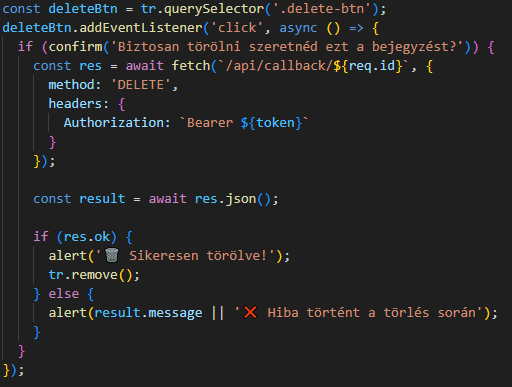


Ez a kód lehetővé teszi, hogy az adminisztrátor módosítsa a visszahívás státuszát. Ha a státusz megváltozik a legördülő menüben, a rendszer egy PUT kérést küld a szerverre a státusz frissítésére.

**Példa**:

* Ha a felhasználó a státuszt "Függőben"-ről "Megoldva"-ra állítja, a kód elküldi a PUT kérést, amely frissíti a visszahívás adatát az adatbázisban.

### 4. Törlés funkció



Ez a kód kezeli a törlés funkciót, amely egy törlés előtt egy megerősítést kér a felhasználótól. Ha a felhasználó megerősíti a törlést, a rendszer egy DELETE kérést küld, hogy eltávolítsa a visszahívást az adatbázisból.

**Példa**:

* Ha egy adminisztrátor törölni szeretné a visszahívást, és a törlés sikeres, akkor a sor eltávolításra kerül a táblázatból, és egy sikeres törlés üzenet jelenik meg.

1. CSV Exportálás



Ez a kód a visszahívások adatainak CSV fájlba történő exportálására szolgál. A gomb megnyomása után a rendszer lekéri az adatokat a szerverről, majd létrehozza a CSV fájlt és letöltésre kínálja azt a felhasználónak.

**Példa**:

* Ha az adminisztrátor az "Export CSV" gombra kattint, a rendszer generál egy CSV fájlt, amely tartalmazza a visszahívásokat, és automatikusan letölthetővé válik.

Auth.js

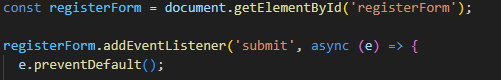
Az auth.js fájl egy regisztrációs űrlap kezelésére szolgáló JavaScript kódot tartalmaz. A felhasználók számára lehetőséget biztosít új fiók létrehozására, a megadott adatok alapján. Az alkalmazás szerver felé egy POST kérést küld a regisztrációs adatokat tartalmazó JSON formátumban, és megfelelő válasz esetén visszajelzést ad a felhasználónak. Ha a regisztráció sikeres, a felhasználó átirányítódik a bejelentkezési oldalra.

## Főbb Funkciók

1. **Jelszóellenőrzés:** A felhasználó által megadott két jelszó összehasonlítása történik, hogy biztosítva legyen, hogy a felhasználó nem tévesztette el a jelszót.
2. **API kérés regisztrációs adatokkal:** A felhasználó által megadott adatokat egy HTTP POST kéréssel továbbítja a szerver felé a regisztrációs végpontra (/api/auth/register).
3. **Hiba- és sikerüzenetek:** A kód kezel minden lehetséges hibát, például ha a jelszavak nem egyeznek, vagy ha a regisztrációs kérés sikertelen a szerver felől.

## Kód Magyarázata

### Form Submit esemény kezelése

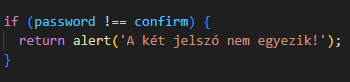


A kód a regisztrációs űrlap submit eseményére hallgat, és amikor a felhasználó elküldi az űrlapot, megakadályozza az alapértelmezett viselkedést (e.preventDefault()). Az alapértelmezett viselkedés, azaz az űrlap valódi beküldése, itt meg van akadályozva, hogy JavaScript-ben kezelhessük az adatokat.

**Példa**:

* Ha a felhasználó kitöltötte a regisztrációs űrlapot és rákattintott a "Regisztrálás" gombra, akkor a submit esemény bekövetkezik, és a kód végrehajtódik.

1. Jelszóellenőrzés

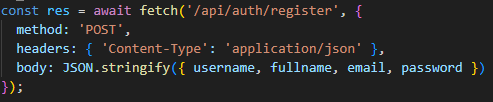


Ez a rész ellenőrzi, hogy a felhasználó által megadott password és confirm mezők értékei megegyeznek-e. Ha nem, akkor a rendszer figyelmezteti a felhasználót, hogy a két jelszó nem egyezik, és nem küldi el a regisztrációs adatokat a szerverre.

**Példa**:

* Ha a felhasználó az password mezőbe "mypassword" értéket ír, de a confirm mezőbe "mypassword123" értéket ír, akkor a rendszer figyelmeztetést jelenít meg, és a regisztráció nem történik meg.

1. API kérés regisztrációs adatokkal



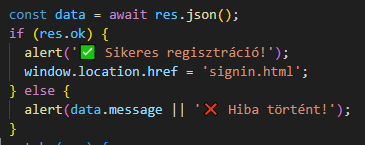
A kód egy fetch kérést küld a /api/auth/register végpontra POST módszerrel. Az adatokat JSON formátumban küldi a szerver felé, a username, fullname, email, és password mezőkkel.

* **HTTP Kérés:** A fetch API egy modern JavaScript API, amely HTTP kérések küldésére és válaszok fogadására használható. A kéréshez beállítjuk a Content-Type: application/json fejléccel, hogy a szerver tudja, hogy a kérés törzse JSON formátumban érkezik.

**Példa**:

* A felhasználó által megadott adatok, mint például a felhasználónév, teljes név, e-mail cím és jelszó, JSON objektummá lesznek alakítva és elküldve a szerver felé a regisztrációs végponton.

1. Sikeres vagy hibás válasz kezelése

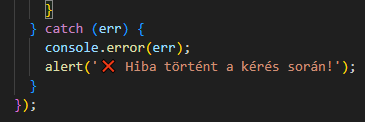


A válasz, amelyet a szerver visszaküld, JSON formátumban érkezik, és a data változó tárolja. Ha a válasz sikeres (res.ok), akkor a felhasználó sikeres regisztrációval értesül, és átirányításra kerül a bejelentkezési oldalra (signin.html). Ha nem sikeres, akkor egy hibaüzenet jelenik meg a felhasználó számára.

**Példa**:

* Ha a szerver válasza tartalmaz egy sikerüzenetet, akkor a felhasználó értesítést kap arról, hogy sikeres volt a regisztráció, és átirányítódik a bejelentkezési oldalra.

1. Hiba kezelése



Ha bármilyen hiba lép fel a kérés küldése vagy a válasz feldolgozása közben, a kód elkapja a hibát a catch blokkban. A hiba részletei ki lesznek írva a konzolra, és egy általános hibaüzenet jelenik meg a felhasználónak.

**Példa**:

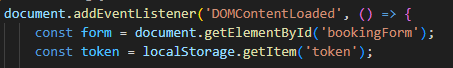
* Ha a hálózati kapcsolat nem elérhető, vagy a szerver nem válaszol, a felhasználó a "❌ Hiba történt a kérés során!" üzenetet kapja meg, és a fejlesztő a konzolban látja a hiba részleteit.

Booking.js

A booking.js fájl célja, hogy kezelje a felhasználók edzés foglalásait egy webalkalmazásban. A kód ellenőrzi, hogy a felhasználó be van-e jelentkezve (ezt a token érték alapján ellenőrzi a localStorage-ban). Ha a felhasználó nem jelentkezett be, egy üzenet figyelmezteti őt. Ha be van jelentkezve, a kód lehetőséget biztosít az edzés típusának, időpontjának és időpontjának megadására, majd az adatokat a szerverre küldi, hogy a foglalás létrejöhessen.

## Kód Magyarázata

### 1. DOMContentLoaded **Eseménykezelő**

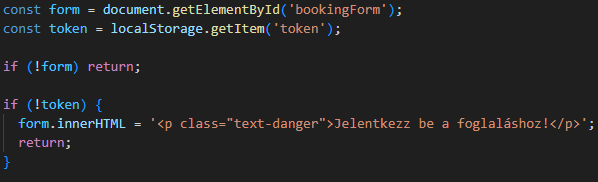


A DOMContentLoaded esemény akkor fut le, amikor az összes HTML elem betöltődött, de a külső fájlok, mint például a stíluslapok (CSS) vagy a képek, nem szükségesek ahhoz, hogy a kód működjön. Ezzel biztosítjuk, hogy a JavaScript kód csak akkor kezdődik el, amikor az oldal HTML tartalma már teljesen betöltődött.

**Példa**:

* Ha az oldal több külső JavaScript fájlt vagy CSS fájlt tartalmaz, a DOMContentLoaded esemény biztosítja, hogy a szkriptek csak azután hajtódjanak végre, hogy a DOM teljesen felépült.

2. **Foglalási Form Ellenőrzés**



Ebben a szakaszban a kód először ellenőrzi, hogy létezik-e a bookingForm elem az oldalon. Ha nem találja, a kód nem folytatódik tovább.

Ezután a localStorage.getItem('token') segítségével megvizsgálja, hogy a felhasználó be van-e jelentkezve (a token értéke egy érvényes hitelesítési token). Ha nincs token, a formot módosítja egy üzenettel, hogy a felhasználó be kell, hogy jelentkezzen a foglalás elvégzéséhez. Ezzel biztosítjuk, hogy csak bejelentkezett felhasználók tudják használni a foglalási funkciót.

**Példa**:

* Ha a felhasználó nem rendelkezik érvényes token értékkel, a form helyett egy figyelmeztető üzenet jelenik meg: "Jelentkezz be a foglaláshoz!" és nem történik további interakció.

3. **Form Beküldése és Adatok Küldése a Szerverre**



Ebben a szakaszban a kód a foglalási űrlap (form) beküldését kezeli.

* **e.preventDefault()**: Az alapértelmezett form küldést megakadályozza, így a böngésző nem frissíti az oldalt, és a JavaScript kód folytathatja az adatküldést.
* **A form adatainak lekérése**: A felhasználó által megadott értékek a form mezőiből (training\_type, training\_date, training\_time) a getElementById metódus segítségével lekérésre kerülnek.

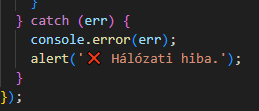
Ezután az adatokat egy POST kéréssel küldi el a szervernek a /api/bookings végpontra, ahol a kérést az Authorization header-rel is ellátja, hogy a bejelentkezett felhasználó tokenje biztosítja az autentikációt.

* **JSON Formátum**: Az adatokat JSON formátumban küldi el a szervernek, mivel a Content-Type: application/json header biztosítja, hogy a szerver felismeri a kérés típusát.
* **Kérés válaszának kezelése**: A válasz sikeres érkezése esetén a formot törli (form.reset()) és sikerüzenetet jelenít meg. Ha a válasz nem volt sikeres, a hibaüzenet jelenik meg.
* **Hibakezelés**: Ha a kérés során valamilyen hálózati hiba történik, a kód elkapja és kiírja a hibát a konzolra, valamint egy figyelmeztető üzenetet jelenít meg a felhasználónak.

**Példa**:

* Ha a felhasználó a következő adatokat adja meg: training\_type = "Spinning", training\_date = "2025-04-15", és training\_time = "10:00", akkor ezek az adatok JSON formátumban küldésre kerülnek a szerverre. Ha a válasz sikeres, a felhasználó értesítést kap: "✅ Edzés sikeresen lefoglalva!" és az űrlap kitöltött mezői kiürülnek.

1. **Hibaüzenet Kezelése**



Ha a fetch kérés során valamilyen hiba történik (például a hálózati kapcsolat megszakad), a kód elkapja a hibát és megjeleníti a felhasználónak a "Hálózati hiba." üzenetet. A hiba részleteit a fejlesztők számára a console.error() segítségével kiírja a konzolra.

**Példa**:

* Ha a felhasználó nem rendelkezik internetkapcsolattal, vagy a szerver nem válaszol, akkor a kód hibát jelez: "❌ Hálózati hiba."

CallbackForm.js

A callbackform.js egy JavaScript fájl, amely kezeli a felhasználói űrlapok adatainak elküldését egy aszinkron (AJAX) kéréssel. Az űrlap adatainak beküldésekor a kód megakadályozza az oldal frissítését, majd az adatokat egy POST kéréssel küldi el egy backend végpontra (/api/callback). Az adatokat JSON formátumban küldi el, és az érkező válaszból egy üzenetet jelenít meg a felhasználónak. A sikeres kérés után az űrlap törlésre kerül.

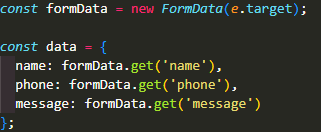
## Kód részletes magyarázata

### 1. **Eseménykezelő hozzáadása az űrlaphoz**



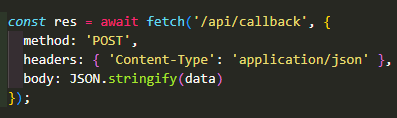
* **document.getElementById('callbackForm')**: Ez a kód egy HTML elemet keres a DOM-ban az id-jával (callbackForm). Az űrlapnak kell rendelkeznie egy id="callbackForm" attribútummal.
* **addEventListener('submit', async (e) => {...})**: A submit eseményre eseménykezelőt rendel hozzá. Ez azt jelenti, hogy amikor az űrlapot benyújtják (például a felhasználó rákattint a "Send" gombra), akkor a függvény végrehajtódik. A submit esemény kezelésekor egy aszinkron függvényt használunk, hogy az adatokat háttérben küldjük el anélkül, hogy újratöltenénk az oldalt.
* **e.preventDefault()**: Ez a hívás megakadályozza, hogy az alapértelmezett űrlapbeküldési művelet végbemenjen (ami az oldal újratöltését eredményezné). Az aszinkron kérések miatt az oldal frissítése helyett adatokat küldünk háttérben.

### 2. **Űrlapadatok összegyűjtése**



* **ew FormData(e.target)**: A FormData objektumot az űrlap elem (e.target) alapján hozunk létre, amely lehetővé teszi az űrlapmezők adatainak könnyű összegyűjtését. A FormData az összes űrlapadatot egy egyszerű kulcs-érték páros formában tárolja.
* **formData.get('name'), formData.get('phone'), formData.get('message')**: Az egyes űrlapmezők (mint a name, phone, és message) értékeit a get metódus segítségével kérdezzük le. A name, phone és message az űrlapmezők nevei, amelyek az HTML-ben az input vagy textarea elemek name attribútumaival vannak azonosítva.

### **Aszinkron** fetch **kérés a backendhez**



* **await fetch('/api/callback', {...})**: A fetch API-t használjuk HTTP kérés küldésére. Az await kulcsszó biztosítja, hogy a kód várjon a válaszra, mielőtt folytatódna. Ez az aszinkron művelet lehetővé teszi, hogy az oldal ne fagyjon le a kérés feldolgozása alatt.
* **method: 'POST'**: A kérés típusa POST, mivel adatokat küldünk a szerverre. A GET metódus csak adatokat kérne, míg a POST metódus lehetővé teszi, hogy adatokat küldjünk el a backendnek.
* **headers: { 'Content-Type': 'application/json' }**: A headers beállítás jelzi a szerver számára, hogy a küldött adatokat JSON formátumban küldjük. A Content-Type fejléc segíti a backend-et abban, hogy helyesen dolgozza fel a beérkező adatokat.
* **body: JSON.stringify(data)**: Az data objektumot JSON formátumban küldjük el. A JSON.stringify(data) az adatokat egy JSON stringgé alakítja, amelyet a POST kérés body részébe illesztünk.

### **Válasz feldolgozása és üzenet megjelenítése**



* **const result = await res.json()**: A válasz JSON formátumban érkezik, ezért a res.json() metódussal azt JavaScript objektummá alakítjuk. Az await biztosítja, hogy a válasz megérkezése előtt a kód ne folytatódjon.
* **alert(result.message)**: Az érkező válasz egy message mezőt tartalmaz, amelyet egy alert ablakban jelenítünk meg. Ez lehet például egy visszaigazolás vagy hibaüzenet, amely a felhasználó számára tájékoztatást ad a kérés állapotáról.

### 5. **Űrlap alaphelyzetbe állítása sikeres kérés esetén**



* **res.ok**: A res.ok egy boolean érték, amely igaz (true), ha a kérés sikeres volt (a válasz státusza 2xx, azaz sikeres válasz). Ha a kérés sikeres, akkor az if blokk végrehajtódik.
* **e.target.reset()**: A reset() metódus visszaállítja az űrlapot az alapértelmezett állapotába. Ezt akkor hívjuk meg, ha a szerver sikeresen feldolgozta a kérést, és az űrlap mezői törlődnek, hogy a felhasználó újabb adatokat adjon meg, ha szükséges.

## Példák

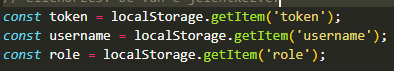
1. **Űrlap beküldése és válasz megjelenítése**
   * Ha a felhasználó kitölti az űrlapot, majd rákattint a "Send" gombra, a kód a következőket végzi:
     + Megakadályozza az oldal újratöltését.
     + Az űrlap adatait JSON formátumban elküldi a backendhez.
     + A szerver válaszát feldolgozza, és megjeleníti a válasz message mezőjét egy figyelmeztető ablakban.
2. **Sikeres kérés után az űrlap törlése**
   * Ha a szerver válasza sikeres (például "Üzenet sikeresen elküldve!"), az űrlap automatikusan törlődik, így a felhasználó számára egy tiszta űrlap jelenik meg.

Header.js

A header.js egy JavaScript fájl, amely a weboldal navigációs fejlécének dinamikus kezelésére szolgál. A kód célja annak ellenőrzése, hogy a felhasználó be van-e jelentkezve, majd ennek megfelelően módosítja a navigációs elemek láthatóságát. Ha a felhasználó be van jelentkezve (azaz rendelkezik token, username, és role adatokkal a localStorage-ban), akkor megjeleníti a felhasználói profilra mutató hivatkozást, valamint eltünteti a bejelentkezési gombot. Ha a felhasználó nincs bejelentkezve, akkor a profil link és a felhasználónév nem jelennek meg.

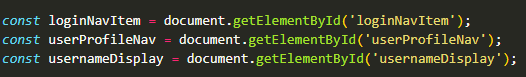
## Kód részletes magyarázata

### 1. **Adatok lekérése a** localStorage**-ból**



* **localStorage.getItem('key')**: A localStorage egy böngésző tároló mechanizmus, amely lehetővé teszi az adatok tartós tárolását az oldal újratöltései között. Ebben az esetben a token, username és role adatok kerülnek lekérésre a böngésző helyi tárolójából. Ezek az értékek azok a kulcsok, amelyek a felhasználó bejelentkezési állapotát és profil információit tárolják.
  + **token**: A felhasználó autentikációs tokenje, amely igazolja a bejelentkezését.
  + **username**: A felhasználó neve, amelyet a navigációs menüben jeleníthetünk meg.
  + **role**: A felhasználó szerepe (például adminisztrátor vagy normál felhasználó), amely alapvetően nem kerül közvetlenül felhasználásra ebben a kódban, de később esetleg szerepet kaphat.

### 2. **DOM elemek lekérése a navigációs menühöz**



* **document.getElementById('id')**: A document.getElementById() metódus a DOM (Document Object Model) segítségével lehetővé teszi, hogy lekérjük a HTML dokumentumban található elemeket az id attribútumuk alapján. Ebben az esetben három elemet keresünk:
  + **loginNavItem**: Az elem, amely a "Bejelentkezés" gombot képviseli a navigációs menüben. Ha a felhasználó be van jelentkezve, ezt el kell rejteni.
  + **userProfileNav**: Az elem, amely a felhasználói profilra mutató hivatkozást tartalmazza. Ha a felhasználó be van jelentkezve, ezt meg kell jeleníteni.
  + **usernameDisplay**: Az elem, ahol a felhasználó neve fog megjelenni, ha be van jelentkezve.

### 3. **Ellenőrzés és a navigációs elemek kezelése**



* **if (token && username && role)**: Ez a feltétel ellenőrzi, hogy mindhárom kulcs (token, username, és role) létezik-e a localStorage-ban. Ha bármelyik érték nem található, az azt jelenti, hogy a felhasználó nincs bejelentkezve, és a navigációs menü nem módosul.
  + **if (userProfileNav) userProfileNav.style.display = 'block';**: Ha a felhasználó be van jelentkezve, akkor a userProfileNav elem (a felhasználói profil link) láthatóvá válik a display: block stílus alkalmazásával. Ha az elem létezik, akkor ezt a beállítást végrehajtjuk.
  + **if (usernameDisplay) usernameDisplay.textContent = username;**: Ha a felhasználó be van jelentkezve, akkor a usernameDisplay elem szöveges tartalmát beállítjuk a felhasználó nevére, amelyet a localStorage-ból nyerünk ki. Ezáltal a navigációs menüben megjelenik a felhasználó neve.
  + **if (loginNavItem) loginNavItem.style.display = 'none';**: Ha a felhasználó be van jelentkezve, akkor elrejtjük a "Bejelentkezés" gombot, amelyet a loginNavItem elem reprezentál. Ezt úgy érjük el, hogy a display: none stílust alkalmazzuk, ami teljesen eltünteti az elemet a felületen.

### 4. **Példák**

#### Példa 1: Felhasználó be van jelentkezve

Tegyük fel, hogy a felhasználó sikeresen bejelentkezett, és a következő adatok vannak a localStorage-ban:

* **token = "abcd1234"**
* **username = "john\_doe"**
* **role = "user"**

Amikor a kód fut, a következő történik:

1. A token, username, és role változók értékei megtalálhatók a localStorage-ban, így a if (token && username && role) feltétel igaz lesz.
2. A userProfileNav elem (a profil link) láthatóvá válik, mivel a style.display = 'block' beállítás történik.
3. A usernameDisplay elem szövege "john\_doe"-ra változik, tehát a felhasználó neve megjelenik a navigációs menüben.
4. A loginNavItem elem eltűnik, mivel a style.display = 'none' beállítást alkalmazzuk.

#### Példa 2: Felhasználó nincs bejelentkezve

Tegyük fel, hogy a felhasználó nincs bejelentkezve, és a localStorage-ban nincsenek meg a szükséges adatok:

* **token = null**
* **username = null**
* **role = null**

Ebben az esetben:

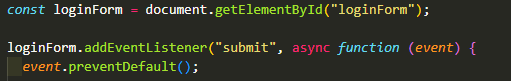
1. A if (token && username && role) feltétel hamis lesz, tehát a kód nem fut le, és a navigációs menü nem változik.
2. A "Bejelentkezés" gomb látható marad, mivel nem alkalmazzuk a display: none beállítást.
3. A profil link és a felhasználói név nem jelennek meg, mivel a userProfileNav és usernameDisplay elemek nincsenek módosítva.

Login.js

A login.js JavaScript fájl a felhasználói bejelentkezés folyamatát kezeli a webalkalmazásban. A felhasználó által megadott felhasználónevet és jelszót egy aszinkron HTTP POST kéréssel küldi el a backend szervernek. A válasz alapján a rendszer elvégzi a szükséges lépéseket, mint például a felhasználói adatokat (token, username, role) menti a localStorage-ba, valamint átirányítja a felhasználót a megfelelő oldalra, jogosultságának megfelelően. A kód kezeli a hibák megjelenítését is, például ha a felhasználó nem ad meg érvényes adatokat, vagy ha a szerverrel probléma adódik.

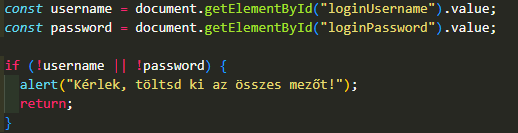
## Kód részletes magyarázata

### 1. **A bejelentkezési űrlap eseménykezelője**



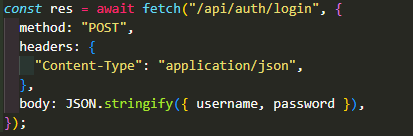
* **document.getElementById("loginForm")**: A kód első lépése, hogy lekéri az űrlapot a loginForm ID alapján a DOM-ból. Ez az elem képviseli azt az űrlapot, amelyet a felhasználó kitölt a bejelentkezéshez.
* **loginForm.addEventListener("submit", async function (event) {...})**: A submit eseményre eseménykezelőt rendelünk, amely akkor fut le, amikor a felhasználó a bejelentkezéshez szükséges űrlapot elküldi (például rákattint a "Bejelentkezés" gombra). A submit eseményt aszinkron függvényben kezeljük, így lehetőség van az aszinkron HTTP kérések kezelésére is.
* **event.preventDefault()**: A preventDefault() metódus megakadályozza az alapértelmezett submit művelet végrehajtását, amely az oldal frissítésével járna. Ez szükséges, mert az adatokat háttérben küldjük el, anélkül, hogy az oldal újratöltődne.

### 2. **Felhasználói adatok ellenőrzése**



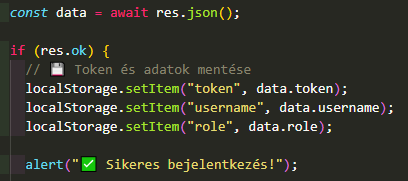
* **document.getElementById("loginUsername").value** és **document.getElementById("loginPassword").value**: Az űrlapon található input mezők értékeit lekérjük a loginUsername és loginPassword ID-k alapján. Ezek a felhasználó által megadott felhasználónevet és jelszót tartalmazzák.
* **if (!username || !password)**: A feltétel ellenőrzi, hogy a felhasználó kitöltötte-e mindkét mezőt. Ha bármelyik mező üres, akkor egy figyelmeztető üzenetet jelenít meg, amely arra kéri a felhasználót, hogy töltse ki az összes mezőt, és megakadályozza a további kód végrehajtását a return kulcsszóval.

### 3. **Aszinkron HTTP kérés a backendhez**



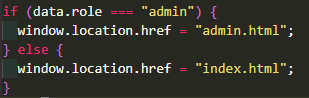
* **await fetch("/api/auth/login", {...})**: A fetch API segítségével indítunk egy HTTP POST kérést a /api/auth/login végpontra, amely a backend szerver számára küldi el a felhasználó bejelentkezési adatokat (username és password). Az await kulcsszó biztosítja, hogy a kód ne folytatódjon, amíg a válasz meg nem érkezik a szervertől.
* **method: "POST"**: A kérés típusa POST, mivel adatokat küldünk a szervernek a felhasználó hitelesítésére.
* **headers: { "Content-Type": "application/json" }**: A kéréshez egy Content-Type fejlécet adunk, amely jelzi, hogy a küldött adatokat JSON formátumban küldjük el. A backend számára ez segít az adat helyes értelmezésében.
* **body: JSON.stringify({ username, password })**: Az űrlap adatait egy JavaScript objektumból JSON stringgé alakítjuk a JSON.stringify segítségével, és a body részben küldjük el a szervernek.

### 4. **A válasz feldolgozása**



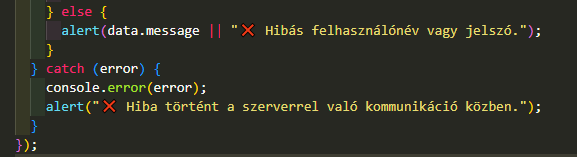
* **const data = await res.json()**: A válasz JSON formátumban érkezik, ezért a res.json() metódussal JavaScript objektummá alakítjuk a választ. Az await kulcsszó biztosítja, hogy a válasz megérkezése előtt a kód ne folytatódjon.
* **if (res.ok)**: A res.ok egy boolean érték, amely akkor igaz, ha a szerver válasza sikeres (2xx státusz kód). Ha a válasz sikeres, akkor a felhasználó be van jelentkezve, és a következő lépések végrehajtódnak.
* **localStorage.setItem("token", data.token)**: A szerver válaszából (amely a bejelentkezési adatokat tartalmazza) a token, username, és role értékeket a localStorage-ba mentjük, hogy azokat később felhasználhassuk a felhasználó autentikációjának kezelésére és az oldalon való navigációhoz.
* **alert("✅ Sikeres bejelentkezés!")**: A sikeres bejelentkezés után a felhasználó számára egy visszaigazolást jelenítünk meg egy üzenetablakban.

### 5. **Átirányítás jogosultság alapján**



* **if (data.role === "admin")**: A szerver válaszában található role mező alapján döntünk arról, hogy melyik oldalra irányítsuk a felhasználót. Ha a felhasználó admin jogosultsággal rendelkezik, akkor az adminisztrátori oldalra (admin.html) irányítjuk át, különben a normál felhasználói oldalra (index.html).

### 6. **Hibakezelés**



* **else { alert(data.message || "❌ Hibás felhasználónév vagy jelszó."); }**: Ha a válasz nem sikeres (nem res.ok), akkor hibát jelez a felhasználónak. Ha a szerver küldött egy hibaüzenetet (például data.message), azt jelenítjük meg, különben egy általános hibát ("Hibás felhasználónév vagy jelszó") jelenítünk meg.
* **catch (error)**: A catch blokk kezeli a potenciális hibákat, amelyek a fetch kéréssel kapcsolatban felmerülhetnek, például ha a szerver nem érhető el. Ilyen esetekben a felhasználó számára egy általános hibát jelenítünk meg, és a hiba részleteit a konzolon naplózzuk.

## Példák

#### Példa 1: Sikeres bejelentkezés

* A felhasználó helyesen adja meg a felhasználónevet (john\_doe) és a jelszót (password123).
* A backend válasza tartalmazza a következő adatokat:
  + **token: "abcd1234"**
  + **username: "john\_doe"**
  + **role: "user"**

A kód elmenti a localStorage-ba a szükséges adatokat, és átirányítja a felhasználót a index.html oldalra.

#### Példa 2: Hibás bejelentkezési adatok

* Ha a felhasználó helytelen jelszót ad meg, a backend hibát küld, például:
  + **message: "Hibás felhasználónév vagy jelszó."**

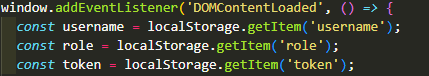
A kód megjeleníti ezt a hibaüzenetet az alert ablakban.

Profile.js

A profile.js JavaScript fájl a felhasználói profiloldal interakcióit kezeli, amely az alkalmazásban a felhasználó beállításainak és foglalásainak kezelésére szolgál. A kód biztosítja a felhasználói bejelentkezés állapotának ellenőrzését, a felhasználói adatok megjelenítését, a jelszócsere funkciót, valamint a foglalások megjelenítését és törlését. Ha a felhasználó nincs bejelentkezve, a rendszer visszairányítja őt a főoldalra.

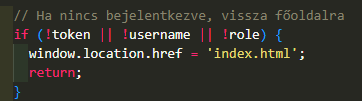
## Kód részletes magyarázata

### 1. **Eseménykezelő hozzáadása a DOMContentLoaded eseményhez**



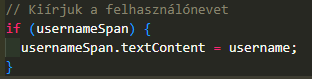
* **window.addEventListener('DOMContentLoaded', () => {...})**: Az eseménykezelő az oldal teljes DOM struktúrájának betöltődése után fut le. A DOMContentLoaded esemény biztosítja, hogy az összes HTML elem és a szkriptek betöltődtek, mielőtt bármilyen manipulációt végeznénk a DOM-on.
* **localStorage.getItem('username')**, **localStorage.getItem('role')**, **localStorage.getItem('token')**: Az aktuálisan bejelentkezett felhasználó adatai (username, role, token) a localStorage-ból kerülnek lekérésre, hogy megerősítsük a felhasználó hitelesítését, valamint a szükséges információkat a profiloldal megjelenítéséhez.

### 2. **Bejelentkezés ellenőrzés és átirányítás**



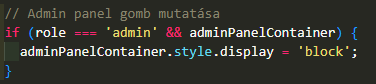
* **Ellenőrzés**: Ha a localStorage-ban nem találhatóak a felhasználói adatok (token, username, role), a felhasználó nincs bejelentkezve. Ekkor a rendszer automatikusan visszairányítja őt a főoldalra (index.html), megakadályozva, hogy hozzáférjen a profiloldalhoz jogosultságok nélkül.

### 3. **Felhasználói név megjelenítése**



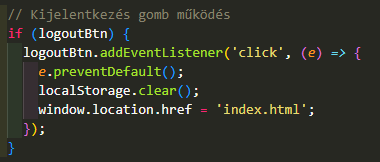
* **usernameSpan.textContent = username**: Ha a profileUsername ID-jú HTML elem létezik, a kód beállítja a felhasználó nevét a HTML elem szöveges tartalmának (textContent) a localStorage-ban tárolt username értékére. Ez a felhasználói profiloldalon való megjelenítést szolgálja.

### 4. **Admin panel gomb megjelenítése**



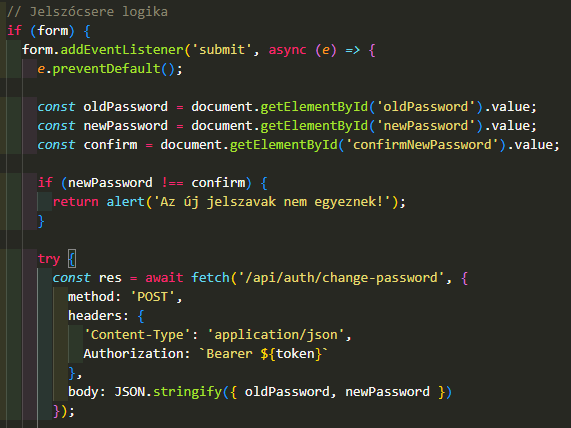
* **if (role === 'admin')**: Ha a felhasználó admin jogosultsággal rendelkezik (azaz a role értéke 'admin'), akkor megjelenítjük az admin panelhez tartozó HTML elemet (adminPanelContainer), így az adminisztrátor hozzáférhet az adminisztratív funkciókhoz. Ez a rész biztosítja, hogy csak a megfelelő jogosultságú felhasználók láthassák az admin panelt.

### 5. **Kijelentkezés funkció**



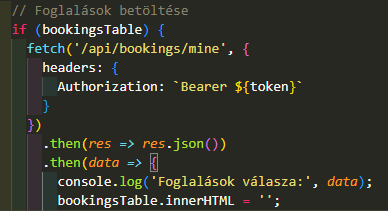
* **logoutBtn.addEventListener('click', (e) => {...})**: Az eseménykezelő a kijelentkezés gombra van hozzárendelve. Amikor a felhasználó rákattint, a click esemény aktiválódik, és a következő lépéseket hajtjuk végre:
  + **e.preventDefault()**: A gomb alapértelmezett viselkedését (pl. űrlap elküldését) meggátoljuk.
  + **localStorage.clear()**: A felhasználó adatai (token, username, role) törlődnek a localStorage-ból, így a felhasználó kijelentkezik.
  + **window.location.href = 'index.html'**: A felhasználó átirányításra kerül a főoldalra.

### 6. **Jelszócsere logika**



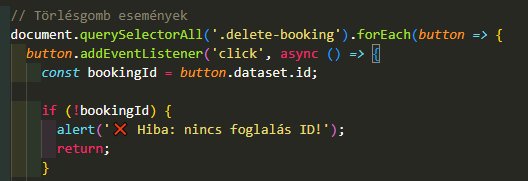
* **form.addEventListener('submit', async (e) => {...})**: A jelszócsere űrlap elküldésekor a kód az eseményt aszinkron módon kezeli. Az e.preventDefault() megakadályozza az alapértelmezett űrlap beküldést, mivel a jelszócsere adatokat háttérben kell küldeni.
* **Ellenőrzés**: Ha az új jelszavak nem egyeznek, a rendszer figyelmezteti a felhasználót, és nem küldi el a kérését.
* **await fetch('/api/auth/change-password', {...})**: A backend szervernek egy POST kérést küldünk a jelszó cseréjére, a felhasználó régi és új jelszavával. A Authorization fejléc tartalmazza a felhasználó érvényes tokenjét, így biztosítva, hogy csak bejelentkezett felhasználók módosíthassák a jelszavukat.
* **Sikeres válasz**: Ha a válasz sikeres, a felhasználó kijelentkezik, és újra be kell jelentkeznie az új jelszóval.

### 7. **Foglalások megjelenítése és törlése**



* **fetch('/api/bookings/mine', {...})**: A rendszer lekéri a felhasználó saját foglalásait a backend szerverről a /api/bookings/mine végponton keresztül. Az Authorization fejléc tartalmazza a felhasználó tokenjét, amely a hitelesítést biztosítja.
* **bookingsTable.innerHTML = ''**: Az előző foglalások törlésre kerülnek a táblázatból, hogy helyet adjanak az új adatoknak.
* **data.forEach(booking => {...})**: A válaszként kapott foglalásokat megjelenítjük a táblázatban. Minden foglalás egy új sorban (tr) jelenik meg, amely tartalmazza a típusát, dátumát, időpontját és létrehozásának dátumát.

### 8. **Foglalás törlése**



* **Törlésgomb eseménykezelő**: Minden foglalás sorában található törlés gombra (.delete-booking) egy eseménykezelőt rendelünk. Ha a felhasználó rákattint, a rendszer megerősíti a törlést, és ha a felhasználó megerősíti, a rendszer a megfelelő foglalást törli a backend rendszerből.

## Példák

#### Példa 1: Bejelentkezett adminisztrátor

* Ha egy adminisztrátor bejelentkezik, az admin panel gomb láthatóvá válik, és az adminisztrátori jogosultságokat biztosító funkciók elérhetőek.

#### Példa 2: Foglalás törlése

* A felhasználó rákattint a foglalás törlésére, majd megerősíti, hogy törölni kívánja azt. A törlés sikeres esetén a foglalás eltűnik a táblázatból.

AuthMiddleware.js

Az authmiddleware.js fájl két fontos middleware funkciót biztosít a Node.js és Express alapú alkalmazásokhoz: az autentikálás ellenőrzését és az adminisztrátori jogosultságok kezelését. A middleware-ek közvetlenül a HTTP kérés kezelésére szolgálnak, és biztosítják, hogy a felhasználók megfelelő jogosultságokkal rendelkezzenek a rendszer bizonyos funkcióinak eléréséhez.

A kód két fő funkciót tartalmaz:

1. **authenticateToken** - Ellenőrzi a beérkező HTTP kéréshez csatolt JSON Web Token (JWT) érvényességét.
2. **requireAdmin** - Ellenőrzi, hogy a felhasználó rendelkezik-e admin jogosultsággal.

## Kód részletes magyarázata

1. **authenticateToken Middleware**



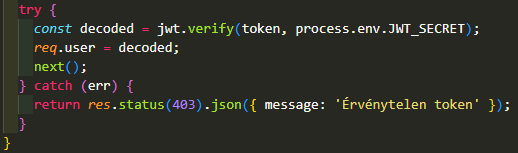
* **const authHeader = req.headers.authorization;**: A kód első lépése, hogy lekéri az HTTP kérés fejlécéből az Authorization mezőt. Az Authorization fejléc tartalmazza a felhasználó hitelesítésére szolgáló tokent. A fejlécre vonatkozó standard az, hogy a token típusa (pl. "Bearer") és maga a token egy szóköz választja el egymástól.
* **const token = authHeader && authHeader.split(' ')[1];**: Ha az Authorization fejléc létezik, a kód elválasztja a token típusát és magát a tokent. A split(' ')[1] rész a szóközön alapuló szétválasztást végzi el, és a második elemet (maga a token) választja ki. Ha az Authorization fejléc nem létezik, a token értéke undefined lesz.

### 2. **Token ellenőrzése**



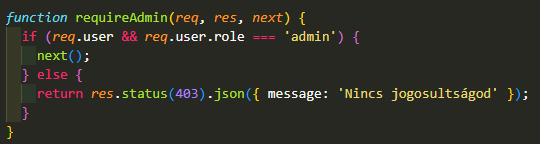
* **if (!token)**: Ha nincs token a kérésben (azaz a felhasználó nem adta meg vagy nem tartalmazza a fejléc a szükséges információt), akkor a rendszer egy **401 Unauthorized** választ küld, amely azt jelzi, hogy a kérés hitelesítése hiányzik. A válaszban egy hibaüzenet szerepel, hogy "Nincs token megadva".

### 3. **JWT token dekódolása**



* **const decoded = jwt.verify(token, process.env.JWT\_SECRET);**: Az jwt.verify() függvény használatával a kód megpróbálja dekódolni a tokent. Ehhez szükséges egy titkos kulcs (JWT\_SECRET), amelyet az alkalmazás környezeti változói között tárolunk (process.env.JWT\_SECRET). Ha a token érvényes, akkor a dekódolt adatokat (decoded) visszaadja, és ezek az adatok a kérés (req) objektumába kerülnek a req.user mezőbe.
* **req.user = decoded;**: Ha a token érvényes, az decoded objektum, amely tartalmazza a felhasználó adatait (például felhasználónév, jogosultságok stb.), hozzárendelődik a req.user mezőhöz. Ez lehetővé teszi, hogy az alkalmazás többi része hozzáférjen a felhasználói adatokhoz az adott kérésen belül.
* **next();**: Miután a token sikeresen érvényesítve lett, a kód meghívja a next() függvényt, amely továbbítja a kérést a következő middleware-hez vagy végponthoz.
* **catch (err)**: Ha a token érvénytelen, a jwt.verify() függvény hibát dob, amit a catch blokk kezel. Ilyenkor a rendszer egy **403 Forbidden** választ küld, jelezve, hogy a token érvénytelen. A válasz tartalmazza az "Érvénytelen token" hibaüzenetet.

### 4. requireAdmin **Middleware**



* **if (req.user && req.user.role === 'admin')**: Az requireAdmin middleware egy fontos jogosultsági ellenőrzést végez. Ellenőrzi, hogy a kéréshez tartozó felhasználói adatok (req.user) léteznek-e és hogy a felhasználó szerepköre (role) adminisztrátor-e. Ha igen, akkor a kérést továbbítja a következő middleware-hez vagy végponthoz a next() függvénnyel.
* **else { return res.status(403).json({ message: 'Nincs jogosultságod' }); }**: Ha a felhasználó nem admin, akkor a rendszer **403 Forbidden** választ küld, jelezve, hogy a felhasználónak nincs megfelelő jogosultsága az adott művelet végrehajtásához. A válasz tartalmazza a "Nincs jogosultságod" üzenetet.

### 5. **Exportálás**



* A modul exportálja a két middleware-t (authenticateToken és requireAdmin), hogy más fájlokban használhatóak legyenek. A module.exports segítségével a függvények elérhetők lesznek a rendszer többi részén, például a route handler-ekben.

Az authmiddleware.js két alapvető middleware-t biztosít, amelyek segítenek a felhasználók hitelesítésében és jogosultságok ellenőrzésében egy Node.js Express alapú alkalmazásban:

* **authenticateToken**: Ellenőrzi a beérkező kéréshez tartozó JWT érvényességét, és biztosítja, hogy csak érvényes tokennel rendelkező felhasználók férhessenek hozzá a védett végpontokhoz.
* **requireAdmin**: Ellenőrzi, hogy a felhasználó adminisztrátori jogosultságokkal rendelkezik-e, és csak az admin felhasználók számára teszi elérhetővé a védett admin felületeket.

A middleware-ek biztosítják a szükséges biztonsági ellenőrzéseket a rendszer különböző végpontjain, és segítenek megelőzni a jogosulatlan hozzáféréseket.

Backend

Admin.js

Az admin.js fájl egy Express routert tartalmaz, amely az adminisztrátorok számára védett végpontokat biztosít. A fájlban található végpontok hozzáférése kizárólag az admin felhasználók számára engedélyezett. Az autentikálás és a jogosultság ellenőrzése middleware-ek (mint például authenticateToken és requireAdmin) segítségével történik.

A router az adminisztrátori feladatokat kezelő API végpontokat biztosít. Ebben az esetben csak egy végpont található, amely a callback kéréseket jeleníti meg.

## Kód részletes magyarázata

### 1. **Express router létrehozása**



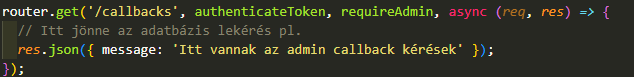
* **const express = require('express');**: Az Express keretrendszer betöltése, amely lehetővé teszi a gyors webes alkalmazások fejlesztését. Az Express segít az HTTP kérések és válaszok kezelésében.
* **const router = express.Router();**: Létrehozunk egy új router példányt, amely az Express router funkcióját biztosítja. A router segít az útvonalak (endpoints) és azokhoz tartozó logikák kezelésében.

### 2. **Middleware-ek importálása**



* **const { authenticateToken, requireAdmin } = require('../middleware/authMiddleware');**: A authenticateToken és requireAdmin middleware-ek importálása a authMiddleware fájlból. Ezek a middleware-ek biztosítják a megfelelő hitelesítést és jogosultsági ellenőrzést az adminisztrátori végpontokhoz.
  + **authenticateToken**: Ellenőrzi, hogy a beérkező kérés tartalmaz-e egy érvényes JSON Web Tokent (JWT), amely hitelesíti a felhasználót.
  + **requireAdmin**: Ellenőrzi, hogy a felhasználó rendelkezik-e adminisztrátori jogosultságokkal. Ha nem, hozzáférés megtagadva.

### 3. **Admin Callback Végpont**



* **router.get('/callbacks', authenticateToken, requireAdmin, async (req, res) => {**: Ez a sor egy új GET kérés kezelőjét hozza létre a /callbacks végponton. A kérés csak akkor lesz sikeres, ha két feltétel teljesül:
  1. **authenticateToken** middleware: Ellenőrzi, hogy a kérés tartalmaz-e érvényes JWT-t.
  2. **requireAdmin** middleware: Ellenőrzi, hogy a felhasználó admin jogosultságokkal rendelkezik.
  3. **authenticateToken** middleware biztosítja, hogy a kérés csak akkor folytatódjon, ha a felhasználó be van jelentkezve.
  4. **requireAdmin** middleware biztosítja, hogy csak admin felhasználók férhessenek hozzá a /callbacks végponthoz.
* **async (req, res) => {**: Az aszinkron függvény, amely az admin számára a callback kéréseket jeleníti meg. A függvény bemeneti paraméterei a req (request) és res (response), amelyek a kérés és válasz kezelésére szolgálnak.
  1. Az aszinkron kulcsszó (async) azt jelenti, hogy itt aszinkron műveletek végezhetők el, például adatbázis-lekérdezések. Jelen esetben a kommentár jelzi, hogy itt adatbázisból kellene adatokat lekérni, de jelenleg egy statikus válasz kerül visszaadásra.
* **res.json({ message: 'Itt vannak az admin callback kérések' });**: A válasz visszaküldése JSON formátumban. A válasz tartalmaz egy üzenetet, amely tájékoztatja az admin felhasználót arról, hogy itt találhatók az admin callback kérések. Az adatbázis lekérdezése ezen a ponton valósulhatna meg, hogy dinamikusan jelenítse meg a callback kéréseket.

### 4. **Exportálás**



**module.exports = router;**: Az Express router exportálása, hogy más fájlok is használhassák. Ezáltal a router a szerver beállításai között elérhetővé válik, és a különböző API végpontokat kezelni tudja.

* Ez a kódrészlet lehetővé teszi, hogy a routert importáljuk és használjuk az app.js vagy egyéb fájlokban, ahol az alkalmazás fő routerének részét képezi.

Az admin.js fájl egy Express routert biztosít, amely csak autentikált admin felhasználók számára elérhető végpontokat tartalmaz. Az autentikálás és jogosultságkezelés middleware-ek segítségével történik, így biztosítva, hogy a /callbacks végpontra csak azok a felhasználók férhessenek hozzá, akik megfelelő jogosultságokkal rendelkeznek.

* **authenticateToken**: Ellenőrzi, hogy a felhasználó érvényes JWT-t tartalmaz-e.
* **requireAdmin**: Ellenőrzi, hogy a felhasználó admin jogosultságokkal rendelkezik-e.

A jelenlegi kódban a /callbacks végpont egy üzenetet küld vissza, de itt lenne lehetőség adatbázis lekérdezések és dinamikus tartalom megjelenítésére.

Auth.js

Ez a kód egy Express.js alapú API végpontot tartalmaz, amely felhasználói regisztrációval, bejelentkezéssel és jelszó módosítással kapcsolatos műveleteket kezel. A kód az alábbi három fő funkciót valósítja meg:

1. **Regisztráció** – Új felhasználó létrehozása a rendszerben.
2. **Bejelentkezés** – A felhasználó hitelesítése és hozzáférési token generálása.
3. **Jelszó módosítása** – A felhasználó jelszavának megváltoztatása hitelesítést követően.

### Használt modulok

1. **express** – A web framework, amely segít API végpontok létrehozásában.
2. **bcrypt** – A jelszavak biztonságos titkosítására és ellenőrzésére szolgáló könyvtár.
3. **jsonwebtoken** – JSON Web Token (JWT) használata a felhasználók hitelesítésére és azonosítására.
4. **db** – Az adatbázis kapcsolatot biztosító modul, amely az SQL lekérdezések futtatásáért felelős.

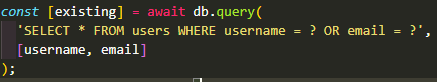
### API végpontok

#### 1. **Regisztráció (**POST /register**)**

Ez a végpont lehetővé teszi új felhasználók regisztrálását a rendszerbe. A kérésben meg kell adni a felhasználó nevét (username), teljes nevét (fullname), e-mail címét (email) és jelszavát (password).

##### Működés:

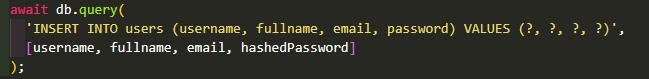
1. **Felhasználó keresése** – A rendszer először lekérdezi az adatbázist, hogy megtudja, van-e már olyan felhasználó, aki az adott username vagy email mezők valamelyikével egyezik. Ha igen, akkor 400-as hibakóddal válaszol a Felhasználónév vagy e-mail már létezik üzenettel.



**Jelszó titkosítása** – Ha nem találunk létező felhasználót, akkor a jelszót bcrypt segítségével titkosítjuk (bcrypt.hash).



**Felhasználó hozzáadása az adatbázishoz** – A jelszó titkosítása után az adatokat beszúrjuk az adatbázisba.



#### 2. **Bejelentkezés (**POST /login**)**

A bejelentkezés végpontja a felhasználó hitelesítésére szolgál. A felhasználó megadja a username és password mezőket, majd a rendszer ellenőrzi a megadott jelszót.

##### Működés:

1. **Felhasználó keresése** – A kód először lekérdezi az adatbázisból az adott username-hez tartozó adatokat.



**Jelszó ellenőrzése** – Ha a felhasználó létezik, akkor bcrypt segítségével összehasonlítja a megadott jelszót a tárolt, titkosított jelszóval (bcrypt.compare).



**JWT token generálása** – Ha a jelszó helyes, akkor a rendszer egy JWT tokent generál, amelyet a felhasználó a későbbi kérdésekhez használhat.



#### **Jelszó módosítása (**POST /change-password**)**

Ez a végpont lehetővé teszi a felhasználó számára, hogy megváltoztassa a jelszavát, ha rendelkezik érvényes hozzáférési tokennel.

##### Működés:

**Token ellenőrzése** – Az első lépés, hogy a kérés tartalmazzon egy Authorization fejlécet, amelyben egy JWT token található. A rendszer először ellenőrzi a token érvényességét.



### Hiba kezelés

A kódban számos hibaellenőrzés található, amelyek biztosítják, hogy a rendszer megfelelően reagáljon a nem várt vagy hibás bemeneti adatokra. A válaszok a következő hibákat kezelhetik:

* **400** – Hibás bemenet (pl. meglévő felhasználónév vagy e-mail, vagy hiányzó mezők).
* **401** – Jogosulatlan hozzáférés (pl. helytelen felhasználónév/jelszó vagy érvénytelen token).
* **500** – Szerverhiba.
* Ez az Express.js alkalmazás egy egyszerű, de biztonságos felhasználói hitelesítési rendszert valósít meg, amely bcrypt és JWT segítségével védi a felhasználói adatokat és lehetővé teszi a felhasználók számára, hogy biztonságosan regisztráljanak, bejelentkezzenek, és módosítsák jelszavukat.

Bookings.js

Ez a kód egy Express.js alapú API végpontot tartalmaz, amely a felhasználók edzésfoglalásait kezeli. A rendszer három fő műveletet valósít meg:

1. **Új foglalás létrehozása** – Lehetővé teszi a felhasználók számára új edzésfoglalás létrehozását.
2. **Saját foglalások lekérése** – A felhasználók megtekinthetik saját, már meglévő foglalásaikat.
3. **Foglalás törlése** – A felhasználók törölhetik saját foglalásaikat.

Mindezek a műveletek autentikált felhasználók számára elérhetők, az autentikációs middleware segítségével.

### Használt modulok

1. **express** – A web framework, amely az API végpontok kezelését segíti.
2. **db** – Az adatbázis kapcsolatot biztosító modul, amely SQL lekérdezések végrehajtásáért felelős.
3. **authMiddleware** – A middleware, amely a felhasználók hitelesítését végzi, és biztosítja, hogy csak jogosult felhasználók végezhetnek műveleteket.

### API végpontok

#### 1. **Új foglalás létrehozása (**POST /**)**

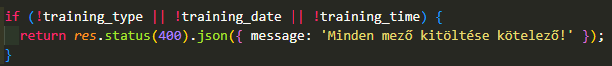
Ez a végpont lehetővé teszi a felhasználók számára, hogy új edzésfoglalást hozzanak létre a rendszerben. A kéréshez szükséges megadni az edzés típusát (training\_type), időpontját (training\_date) és időpontját (training\_time).

##### Működés:

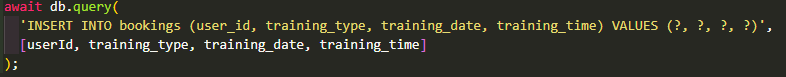
**Token validálása** – A kérésben a authenticateToken middleware ellenőrzi a felhasználó hitelesítő tokenjét. Ha a token érvénytelen vagy hiányzik, a kérés nem kerül feldolgozásra.



**Bemenet ellenőrzése** – A kód ellenőrzi, hogy az összes szükséges mezőt (training\_type, training\_date, training\_time) megadták-e. Ha bármelyik mező hiányzik, akkor 400-as hibával válaszol.



**Foglalás rögzítése** – Ha a bemenet érvényes, az adatokat beszúrjuk az adatbázisba a bookings táblába. A felhasználó azonosítóját (userId) a middleware-ben dekódolt JWT tokenből nyerjük ki.



#### **Saját foglalások lekérése (**GET /mine**)**

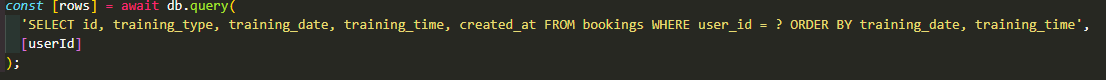
Ez a végpont lehetővé teszi a felhasználók számára, hogy lekérdezzék saját foglalásaikat. Csak azok a foglalások jelennek meg, amelyek az adott felhasználóhoz tartoznak.

##### Működés:

1. **Token validálása** – A kérésben található autentikációs token ellenőrzése a authenticateToken middleware segítségével.



**Foglalások lekérése** – Az adatbázisból lekérjük a felhasználó összes foglalását, amelyek a user\_id-val megegyeznek a jelenleg bejelentkezett felhasználó azonosítójával. A válaszban az összes foglalás részletei visszaküldésre kerülnek.

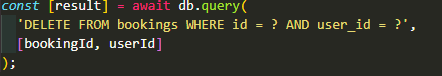


#### **Foglalás törlése (**DELETE /:id**)**

Ez a végpont lehetővé teszi a felhasználók számára, hogy töröljék saját foglalásaikat. A törlés csak akkor hajtható végre, ha a felhasználó a foglalás tulajdonosa.

##### Működés:

1. **Token validálása** – A kérésben található token érvényességét ellenőrzi a authenticateToken middleware.
2. **Foglalás törlése** – A kérésben szereplő foglalás ID-ját (:id paraméter) és a felhasználó ID-ját összevetjük, hogy csak a felhasználó saját foglalását töröljük. Ha a törölt foglalás nem található, vagy a felhasználó nem jogosult a törlésre, 404-es hibát adunk vissza.



**Törlés visszaigazolása** – Ha a törlés sikeres, a válaszban egy sikeres üzenet jelenik meg.

### Hiba kezelés

A kód megfelelő hibakezelést alkalmaz a következő esetekben:

* **400** – Hibás bemenet (pl. hiányzó mezők).
* **401** – Jogosulatlan hozzáférés (pl. érvénytelen token).
* **404** – Nem található foglalás, vagy a felhasználó nem jogosult a törlésre.
* **500** – Szerverhiba, ha a foglalás létrehozása, lekérése vagy törlése nem sikerül.

Ez az API lehetővé teszi a felhasználók számára az edzésfoglalások kezelését Express.js segítségével. Az autentikációs middleware biztosítja, hogy csak jogosult felhasználók végezhetnek műveleteket, így az edzésfoglalások biztonságosan kezelhetők. A kód az alapvető CRUD (Create, Read, Delete) műveleteket valósítja meg foglalásokra, és részletes hibaüzeneteket biztosít minden lépés során.

Callback.js

Ez a kód egy Express.js alapú API végpontot tartalmaz, amely a felhasználók visszahívási kéréseivel kapcsolatos műveleteket valósít meg. A rendszer három fő funkcióval rendelkezik:

1. **Új visszahívási kérés rögzítése** – A felhasználók nyilvánosan küldhetnek visszahívási kéréseket.
2. **Visszahívási kérések lekérése** – Az adminisztrátorok számára lehetővé teszi a visszahívási kérések lekérdezését.
3. **Visszahívási kérés törlése** – Az adminisztrátorok számára lehetővé teszi egy adott kérés törlését.

A rendszer az autentikációs middleware-t (authenticateToken) és jogosultsági ellenőrzést (requireAdmin) használja annak érdekében, hogy a visszahívási kérések biztonságos módon legyenek kezelve.

### Használt modulok

1. **express** – A web framework, amely az API végpontok kezelését segíti.
2. **db** – Az adatbázis kapcsolatot biztosító modul, amely SQL lekérdezések végrehajtásáért felelős.
3. **authMiddleware** – A middleware, amely a felhasználók hitelesítését és jogosultságainak kezelését végzi.

### API végpontok

#### 1. **Új visszahívási kérés rögzítése (**POST /**)**

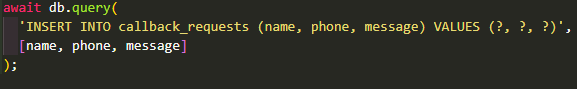
Ez a végpont lehetővé teszi, hogy a felhasználók visszahívási kéréseket küldjenek a rendszernek. A kéréshez szükséges adatokat a kérés törzsében (body) kell megadni: a felhasználó nevét (name), telefonszámát (phone) és opcionálisan üzenetet (message).

##### Működés:

1. **Bemenet validálás** – A kód először ellenőrzi, hogy a kötelező mezők (name és phone) megvannak-e. Ha bármelyik hiányzik, a válasz egy 400-as hibát ad vissza.



**Új kérés beszúrása az adatbázisba** – A megadott adatokat rögzíti a callback\_requests táblában, ha a bemenet érvényes.



#### **Visszahívási kérések lekérése (**GET /**)**

Ez a végpont lehetővé teszi az adminisztrátorok számára a visszahívási kérések lekérését. Az autentikációs middleware segítségével biztosított, hogy csak azok férjenek hozzá, akik adminisztrátori jogosultsággal rendelkeznek.

##### Működés:

1. **Autentikáció és jogosultság ellenőrzése** – A authenticateToken middleware ellenőrzi a felhasználó hitelesítő tokenjét, majd a requireAdmin middleware biztosítja, hogy a felhasználó adminisztrátor legyen.



**Visszahívási kérések lekérése** – Az adatbázisból lekérdezi az összes visszahívási kérést, és azokat JSON formátumban visszaküldi a felhasználónak. A válasz a kérések created\_at mezője szerint csökkenő sorrendben lesz rendezve.



#### **Visszahívási kérés törlése (**DELETE /:id**)**

Ez a végpont lehetővé teszi az adminisztrátorok számára, hogy töröljenek egy adott visszahívási kérést az adatbázisból.

##### Működés:

1. **Autentikáció és jogosultság ellenőrzése** – A authenticateToken és a requireAdmin middleware-ek biztosítják, hogy a kérésben szereplő adminisztrátor jogosult legyen a törlés végrehajtására.



**Kérés törlése** – Az adatbázisból töröljük azt a visszahívási kérést, amelynek ID-ja megegyezik a kérés URL-jében szereplő id paraméterrel. Ha a törlés nem található, 404-es hibát adunk vissza.

**Törlés visszaigazolása** – Ha a törlés sikeres, a válaszban egy sikeres üzenet jelenik meg.

### Hiba kezelés

A kód megfelelő hibakezelést alkalmaz az alábbi esetekben:

* **400** – Hibás bemenet (pl. hiányzó kötelező mezők).
* **401** – Jogosulatlan hozzáférés (pl. érvénytelen token).
* **404** – A törölni kívánt visszahívási kérés nem található.
* **500** – Szerverhiba (pl. adatbázis műveletek közben).

Ez az API biztosítja a visszahívási kérések kezelését. A felhasználók nyilvánosan küldhetnek kéréseket, amelyeket az adminisztrátorok az API-n keresztül lekérhetnek és törölhetnek. Az autentikációs és jogosultságkezelési mechanizmusok biztosítják, hogy a rendszer csak megfelelő jogosultságokkal rendelkező felhasználók számára legyen elérhető. A kód alapvető CRUD (Create, Read, Delete) műveleteket valósít meg a visszahívási kérésekre, és részletes hibaüzeneteket biztosít a műveletek során.

Db.js

Ez a modul felelős az adatbázis kapcsolat kezeléséért, amely egy MySQL adatbázist használ. A modul a **MySQL2** csomagot és a **dotenv** csomagot alkalmazza, hogy egy dinamikus, környezeti változókkal konfigurált adatbázis kapcsolatot hozzon létre, és ezt a kapcsolatot egy **connection pool**-on keresztül használja. A kapcsolat és a pool konfigurációja biztonságosan történik, mivel az adatbázis kapcsolatot konfiguráló paraméterek környezeti változókból (.env fájlból) kerülnek betöltésre.

### Használt modulok

1. **mysql2/promise** – A MySQL adatbázis kezelésére szolgáló csomag, amely támogatja a Promise alapú API-t. Ez segít az aszinkron adatbázis műveletek könnyebb kezelésében.

A createPool függvény lehetővé teszi a kapcsolatokat egy "pool"-ban kezelni, így javítva a teljesítményt, mivel a kapcsolatokat újrahasználhatjuk, nem kell minden egyes adatbázis művelethez új kapcsolatot létrehozni.

1. **dotenv** – A .env fájl kezelésére szolgáló csomag, amely segít a környezeti változók betöltésében és biztonságos tárolásában.

### Működés

A modul célja egy új MySQL adatbázis kapcsolat létrehozása és az adatbázis műveletekhez szükséges kapcsolatpool biztosítása. A konfigurációs beállításokat a **.env** fájlból olvassa be, így a környezetfüggő paraméterek (például adatbázis elérhetőség, felhasználói név, jelszó) biztonságosan tárolhatók.

### Kód részletes magyarázata

1. **Modulok betöltése:**

Először is betöltjük a szükséges modulokat:



A mysql2/promise a MySQL adatbázis kezelésére szolgál aszinkron Promise alapú API-val.

* A dotenv modul segítségével betöltjük a környezeti változókat, amelyek biztonságosan tárolják az adatbázis eléréséhez szükséges paramétereket.

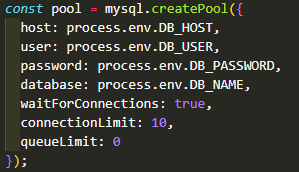
**A környezeti változók betöltése:**



A dotenv.config() függvény segítségével betöltjük a .env fájlt, amely tartalmazza az adatbázis elérhetőségi adatait. Ez lehetővé teszi, hogy az adatbázis kapcsolódási beállítások környezeti változók alapján dinamikusan változhassanak.

**MySQL kapcsolat létrehozása:**

A mysql.createPool segítségével létrehozzuk a kapcsolatot az adatbázissal, és beállítunk egy kapcsolatot kezelő "pool"-t:



**host**: Az adatbázis hosztja, például localhost vagy egy távoli szerver IP címe. Az értéke a .env fájlból érkezik a process.env.DB\_HOST változó segítségével.

* **user**: Az adatbázis felhasználója, amit a process.env.DB\_USER ad.
* **password**: Az adatbázis felhasználóhoz tartozó jelszó, amit a process.env.DB\_PASSWORD tárol.
* **database**: Az adatbázis neve, amellyel a kapcsolat létrejön. A process.env.DB\_NAME környezeti változóból érkezik.
* **waitForConnections**: Ha a kapcsolat pool tele van, akkor várakozni kell, amíg egy kapcsolat felszabadul. Az érték **true**, hogy engedélyezze a várakozást.
* **connectionLimit**: A maximális kapcsolatok száma, amelyeket a pool fenntart. Jelen esetben 10.
* **queueLimit**: A várakozó kapcsolatok maximális száma. Ha elérjük a limitet, további kapcsolatok nem kerülnek sorba. Az érték **0**, amely azt jelenti, hogy nincs limit.

**Modul exportálása:**



A kapcsolatot kezelő pool objektumot exportáljuk, így más modulok is használhatják a MySQL adatbázissal való kommunikációhoz.

A kapcsolatot kezelő pool objektumot exportáljuk, így más modulok is használhatják a MySQL adatbázissal való kommunikációhoz.

Server.js

Ez a fájl az alkalmazás fő szerverét hozza létre az **Express.js** keretrendszer segítségével. A szerver biztosítja az API-k elérhetőségét, a statikus fájlok kiszolgálását és az adatbázis kapcsolódást. A fájl tartalmazza a köztes szoftvereket (middleware-eket), az egyes útvonalakhoz rendelt route-okat és a szerver indításához szükséges kódot.

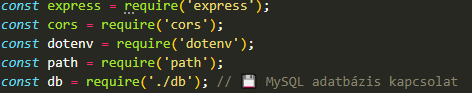
### Használt modulok

1. **express** – Az Express.js keretrendszert használjuk a szerver létrehozásához és az API-k kezeléséhez.
2. **cors** – A Cross-Origin Resource Sharing (CORS) engedélyezése, hogy az alkalmazás más domainekről is elérhető legyen.
3. **dotenv** – A környezeti változók betöltésére szolgál, lehetővé téve a konfigurációk dinamikus kezelését.
4. **path** – Az elérési utak kezelésére használt modul, amely segít a fájlok helyes kezelésében, különösen a statikus fájlok kiszolgálásakor.
5. **db** – Az adatbázis kapcsolatot kezelő modul (MySQL), amely az adatbázis műveletekhez szükséges kapcsolatot biztosít.

### Kód részletes magyarázata

1. **Modulok betöltése:**

Az első lépésben betöltjük a szükséges modulokat, és inicializáljuk a környezeti változók kezelését:



Az express a szerver indításához szükséges.

* A cors lehetővé teszi a CORS engedélyezését, hogy más domainekről is hozzáférhessenek az API-hoz.
* A dotenv segít betölteni a környezeti változókat a .env fájlból.
* A path modul az elérési utak kezelésére szolgál, amely szükséges a statikus fájlok kiszolgálásakor.
* A db a MySQL adatbázis kapcsolatot kezeli, amit az előző db.js fájl biztosít.

**Környezeti változók betöltése:**

A következő kódsorban betöltjük a környezeti változókat a .env fájlból:



Ezáltal elérhetők lesznek az olyan változók, mint a port, adatbázis kapcsolati adatok, titkos kulcsok, stb.

**Express alkalmazás létrehozása:**

Az Express alkalmazás inicializálása:



Az app az Express alkalmazás példánya, amely kezeli az API kéréseket. A PORT változóban tároljuk a port számot, amelyen a szerver futni fog. Ha a .env fájlban nincs beállítva a PORT környezeti változó, akkor az alapértelmezett érték 5000.

**Middleware-ek beállítása:**

Az alábbi kódrészletek beállítják a köztes szoftvereket:

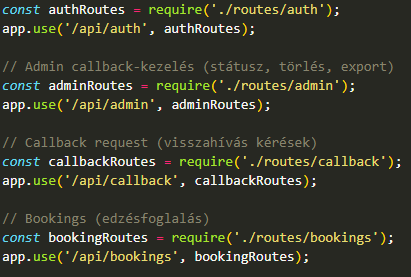


cors() engedélyezi, hogy más domainekről is hozzáférjenek az API-hoz, megoldva ezzel a **Cross-Origin Resource Sharing** problémát.

* Az express.json() middleware lehetővé teszi a JSON formátumú adatok fogadását a HTTP kérdésekben (pl. POST, PUT).

**API Route-ok bekötése:**

Az API útvonalakhoz rendeljük a megfelelő route-okat:



Az /api/auth útvonalhoz az authRoutes modul van rendelve, amely az autentikációs műveleteket (bejelentkezés, regisztráció, stb.) kezeli.

* Az /api/admin adminisztrációs műveletekhez (pl. callback kérések kezelése) rendeljük hozzá az adminRoutes modult.
* A /api/callback útvonalon kezelhetjük a visszahívási kéréseket.
* Az /api/bookings az edzésfoglalásokat kezeli.

**Statikus fájlok kiszolgálása:**

Az Express segítségével statikus fájlokat szolgálhatunk ki, például a frontend alkalmazást:



Ez azt jelenti, hogy a client mappában található fájlokat (pl. HTML, CSS, JS fájlok) a szerver statikusan szolgálja ki.

**Wildcard route (Minden más kérést a frontend kiszolgálása):**

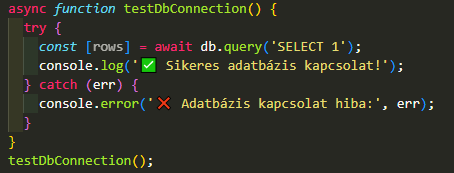
Az alábbi kódrészlet biztosítja, hogy minden olyan kérést, ami nem tartozik egy API végponthoz, a frontend index.html fájl kiszolgálása kövessen:



Ez különösen hasznos, ha az alkalmazás egy SPA (Single Page Application), például React vagy Vue.js frontend-tel rendelkezik.

**Adatbázis kapcsolat tesztelése:**

Az alábbi funkció teszteli, hogy az adatbázis kapcsolat sikeresen létrejött:



A testDbConnection függvény egy egyszerű SQL lekérdezést (SELECT 1) futtat, hogy megbizonyosodjon arról, hogy az adatbázis elérhető.

**Szerver indítása:**

Végül, az alkalmazás elindítása a kívánt porton:



Az alkalmazás most már képes fogadni a kéréseket és kiszolgálni az API-t, illetve a frontend fájlokat.

Ez a fájl felelős az alkalmazás szerverének létrehozásáért és az API-k kezeléséért, beleértve az autentikációt, adminisztrációt, visszahívási kéréseket és edzésfoglalásokat. A statikus fájlok kiszolgálása és az adatbázis kapcsolat tesztelése biztosítja, hogy a backend és frontend összhangban működjön. Az Express.js és a CORS lehetővé teszik, hogy a szerver biztonságosan és hatékonyan kezelje a különböző típusú kéréseket.